

ԿՐԹՄԱՆ ՆԱԹԵՆԱԿԱԺՎ

Visual

C#.NET

საქართველოს ტექნიკური უნივერსიტეტი

რომან სამხარაძე

Visual C#.NET



დამტკიცებულია
სახელმძღვანელოდ სტუ-ს
სარედაქციო-საგამომცემლო
საბჭოს მიერ

თბილისი
2014

სახელმძღვანელო წარმოადგენს მეოთხე, გადამუშავებულ გამოცემას. მასში გადმოცემულია Microsoft Visual Studio .NET გარემოში პროგრამების შემუშავების საკითხები. დაწვრილებითაა განხილული C# ენის საფუძვლები, ობიექტზე ორიენტირებული პროგრამირების პრინციპები, დელეგატები, მოვლენები, ინტერფეისები, მონაცემებისა და შესრულების ნაკადებთან, აგრეთვე, სერიულ პორტებთან და მონაცემთა ბაზებთან მუშაობის ხერხები. სახელმძღვანელოში დამატებულია LINQ ტექნოლოგიასთან მუშაობის პრინციპები. წიგნში უხვადაა მაგალითები და სავარჯიშოები თავიანთი ამოხსნებით.

განკუთვნილია კომპიუტერული ინჟინერიის დეპარტამენტის ბაკალავრების, მაგისტრებისა და დოქტორანტებისათვის, აგრეთვე პროგრამირების შესწავლის მსურველთათვის.

რეცენზენტი: ტექნიკის მეცნიერებათა კანდიდატი, სრული პროფესორი მ. ანდლულაძე
ტექნიკის მეცნიერებათა კანდიდატი, ასოცირებული პროფესორი გ. ჭიკაძე

© საგამომცემლო სახლი "ტექნიკური უნივერსიტეტი", 2014

ISBN 978-9941-20-443-2

<http://www.gtu.ge/publishinghouse/>

ყველა უფლება დაცულია. ამ წიგნის არც ერთი ნაწილი (იქნება ეს ტექსტი, ფოტო, ილუსტრაცია და სხვა) არანაირი ფორმით და საშუალებით (იქნება ეს ელექტრონული თუ მექანიკური) არ შეიძლება გამოყენებულ იქნას გამომცემლის წერილობითი ნებართვის გარეშე.

საავტორო უფლებების დარღვევა კანონით ისჯება.

წიგნს ვუძღვნი

ჩემს შვილებს ანას და საბას

სარჩევი	
წინასიტყვაობა.....	11
I ნაწილი. C# ენა.....	12
თავი 1. შესავალი.....	12
პროგრამირების ისტორიის მოკლე მიმოხილვა.....	12
C# ენის კავშირი .NET Framework გარემოსთან.....	13
Visual Studio 2012.....	14
ობიექტზე ორიენტირებული დაპროგრამების პრინციპები	15
ინკაფსულაცია	18
პოლიმორფიზმი	18
მემკვიდრეობითობა	18
თავი 2. C# ენის საფუძვლები.....	19
პირველი პროგრამა.....	19
პროექტის გადატანა	24
კომენტარები	25
C# ენის საბაზო ტიპები	25
მონაცემთა ჩვეულებრივი ტიპები	25
მთელრიცხვა ტიპები.....	25
მცურავწერტილიანი ტიპები.....	26
decimal ტიპი	26
char ტიპი (სიმბოლო)	27
bool ტიპი.....	29
ლიტერალები	30
ცვლადები და მათი ინიციალიზება	30
ცვლადის ინიციალიზება.....	31
ცვლადების დინამიური ინიციალიზება	31
ოპერატორები.....	31
არიტმეტიკის ოპერატორები.....	31
ინკრემენტისა და დეკრემენტის ოპერატორები	33
შედარებისა და ლოგიკის ოპერატორები	34
მინიჭების ოპერატორი.....	36
ოპერატორების პრიორიტეტები.....	37
გამოსახულებები. მათემატიკის მეთოდები	37
ტიპების გარდაქმნა.....	40
ტიპების დაყვანის ოპერატორი (cast operator)	41
ტიპების გარდაქმნა გამოსახულებებში.....	42
თავი 3. მმართველი ოპერატორები.....	45
ამორჩევის ოპერატორები.....	45
if ოპერატორი	45
ჩადგმული if ოპერატორი	45
პროგრამული კოდის ბლოკები. ხილვადობის უბანი	46
switch ოპერატორი	47
იტერაციის ოპერატორები.....	51
for ოპერატორი	51
მაგალითები	54
while ციკლი.....	55
do-while ციკლი	56
გადასვლის ოპერატორები	56
break ოპერატორი.....	56

continue ოპერატორი	57
goto ოპერატორი.....	58
ტერნარული ოპერატორი.....	58
თავი 4. მასივები, ბიტობრივი ოპერაციები და ჩამოთვლები	60
მასივები	60
ერთგანზომილებიანი მასივები	60
ორგანზომილებიანი მასივები.....	62
გაუსწორებელი ("დაკბილული") მასივები	64
foreach ციკლი.....	66
ბიტობრივი ოპერატორები.....	67
&, , ^ და ~ ბიტობრივი ოპერატორები	68
პერის ოპერატორები	71
ჩამოთვლები	72
თავი 5. კლასები, ინკაფსულაცია, მეთოდები	74
კლასის გამოცხადება	74
ინკაფსულაცია.....	75
ობიექტების შექმნა.....	77
მიმართვითი ტიპის მნიშვნელობის მინიჭება	78
მეთოდები	79
მეთოდიდან მართვის დაბრუნება	81
პარამეტრების გამოყენება	82
კონსტრუქტორები	84
დესტრუქტორები და "ნაგვის შეგროვება"	86
ობიექტების მასივის შექმნა	86
this საკვანძო სიტყვა	87
რთული კლასები	89
ჩადგმული კლასები.....	90
სტრუქტურები	91
შემთხვევითი რიცხვების გენერატორი	92
თავი 6. მეთოდები უფრო დაწვრილებით. პოლიმორფიზმი.....	94
მეთოდისათვის ობიექტის გადაცემა	94
მეთოდისათვის არგუმენტების გადაცემის ხერხები	95
ref და out მოდიფიკატორები	97
ref მოდიფიკატორი.....	97
out მოდიფიკატორი.....	98
params მოდიფიკატორი	99
მეთოდის მიერ ობიექტის დაბრუნება.....	101
მეთოდის გადატვირთვა. პოლიმორფიზმი	102
კონსტრუქტორების გადატვირთვა.....	105
გადატვირთვადი კონსტრუქტორის გამოძახება this სიტყვის გამოყენებით	107
რეკურსია	108
სტატიკური კომპონენტები.....	109
static მოდიფიკატორი.....	109
მუდმივები.....	111
მხოლოდწაკითხვადი ველები.....	111
თავი 7. მემკვიდრეობითობა. ვირტუალური მეთოდები	113
მემკვიდრეობითობის საფუძვლები.....	113
protected მოდიფიკატორი	115
კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა	116

წევრების დამალვა მემკვიდრეობითობის დროს	118
კლასების მრავალდონიანი იერარქიის შექმნა	120
მიმართვები წინაპარი და მემკვიდრე კლასების ობიექტებზე	123
ვირტუალური მეთოდები	124
აბსტრაქტული კლასები	130
მემკვიდრეობითობის აკრძალვა	134
პოლიმორფიზმის აკრძალვა	134
ობიექტების ტიპების დაყვანა	135
object კლასი	136
თავი 8. თვისებები, ინდექსატორები და ოპერატორები	137
გადატვირთვა	137
თვისებები	137
ინდექსატორები	139
ერთგანზომილებიანი ინდექსატორები	139
მრავალგანზომილებიანი ინდექსატორები	142
ოპერატორების გადატვირთვა	143
ოპერატორის მეთოდი	143
ბინარული ოპერატორების გადატვირთვა	144
უნარული ოპერატორების გადატვირთვა	145
შედარების ოპერატორების გადატვირთვა	147
ოპერატორების გადატვირთვისას არსებული შეზღუდვები	148
თავი 9. დელეგატები, მოვლენები, ინტერფეისები და სახელების სივრცე	149
დელეგატები	149
დელეგატების მრავალმისამართიანობა	152
ანონიმური მეთოდები	153
მოვლენები	155
ფართოსამაუწყებლო მოვლენა	157
ინტერფეისები	160
ინტერფეისების რეალიზება	160
კლასების მემკვიდრეობითობა და ინტერფეისის რეალიზება	162
წარმოებული ინტერფეისები	164
ინტერფეისული მიმართვები	166
ინტერფეისის თვისებები	168
ცხადი რეალიზება	169
ინტერფეისის წევრების დამალვა	170
სახელების სივრცე	171
სახელების სივრცის გამოცხადება	172
using დირექტივა	173
ჩადგმული სახელების სივრცე	174
ავტომატურად განსაზღვრული სახელების სივრცე	175
თავი 10. ინფორმაციის შეტანა-გამოტანა	176
შეტანა-გამოტანის ნაკადების კლასები	176
ბაიტების ნაკადები	176
სიმბოლოების ნაკადები	179
ორობითი ნაკადები	179
FileStream კლასი	179
ფაილის გახსნა და დახურვა	179
ფაილიდან ბაიტების წაკითხვა	180
ფაილში ბაიტების ჩაწერა	181

ფაილში სიმბოლოების შეტანა-გამოტანა	183
StreamWriter კლასი	183
StreamReader კლასი	184
ორობითი მონაცემების წაკითხვა და ჩაწერა	184
BinaryWriter კლასი.....	184
BinaryReader კლასი.....	185
ფაილებთან პირდაპირი მიმართვა	187
ფაილის შესახებ ინფორმაციის მიღება.....	188
კატალოგებთან მუშაობა	191
ფაილების არჩევა	194
კატალოგების ხეზე ძებნა	195
NetworkStream კლასი	196
MemoryStream და BufferedStream კლასები	198
მონაცემების ასინქრონული შეტანა-გამოტანა	200
თავი 11. განსაკუთრებული სიტუაციები.....	202
განსაკუთრებული სიტუაციების დამუშავების საფუძვლები	202
try და catch ბლოკები.....	203
try და catch ბლოკების გამოყენების მაგალითები	206
რამდენიმე catch ბლოკის გამოყენება	210
ყველა განსაკუთრებული სიტუაციის დაჭერა	210
ჩადგმული try ბლოკები.....	212
განსაკუთრებული სიტუაციის იძულებით გენერირება	213
finally ბლოკი	214
საკუთარი განსაკუთრებული სიტუაციის შექმნა.....	215
checked და unchecked ოპერატორები.....	216
გამართვა	218
წყვეტის წერტილების შექმნა.....	219
ცვლადების მნიშვნელობების ნახვა.....	219
პროგრამის კოდის შესრულება ბიჯობრივ რეჟიმში.....	222
თავი12. სტრიქონები.....	224
სტრიქონები	224
სტრიქონებთან სამუშაო მეთოდები	224
სტრიქონების მასივები.....	233
სტრიქონების უცვლელობა	233
დინამიური სტრიქონები	234
დინამიური სტრიქონის შექმნა	234
StringBuilder კლასის თვისებები და მეთოდები	235
II ნაწილი. C# ენის სხვა შესაძლებლობები.....	238
თავი 13. თარიღი, დრო და დროის ინტერვალები	238
თარიღი და დრო	238
დროის ინტერვალები.....	246
თავი 14. კოლექციები	251
დინამიური მასივები.....	251
ჰემ-ცხრილები	260
დახარისხებული სიები	263
რიგები	268
სტეკები	269
ბიტების მასივები.....	271
თავი 15. შესრულების ნაკადები.....	274

შესრულების ნაკადის განსაზღვრა	274
შესრულების ნაკადის შექმნა	275
შესრულების ნაკადის პრიორიტეტები	278
შესრულების ნაკადის მდგომარეობები.....	279
შესრულების ნაკადის ლოკალური მონაცემები	281
შესრულების ნაკადების მართვა	283
Timer კლასი.....	283
Join() მეთოდი.....	284
lock საკვანძო სიტყვა	285
Interlocked კლასი.....	288
Monitor კლასი	290
Mutex კლასი	291
თავი 16. საბაზო ბიბლიოთეკის სხვა კლასები	294
გლობალიზაცია.....	294
გრაფიკა	298
გეომეტრიული ფიგურების ხატვა.....	302
ფიგურების შევსება ფერით	305
სტრიქონების ხატვა.....	306
გამოსახულებასთან მუშაობა.....	307
მონაცემების დაშიფვრა და გაშიფვრა.....	310
სიმეტრიული კრიპტოგრაფია.....	310
ასიმეტრიული კრიპტოგრაფია	313
სერიულ პორტებთან მუშაობა.....	313
III ნაწილი. მონაცემთა ბაზები	317
თავი 17. ADO.NET კლასები	317
მონაცემთა ბაზასთან მიმართვა Visual Studio .NET-ის საშუალებებით	317
მონაცემთა ბაზასთან მუშაობა ვიზუალური და არავიზუალური კომპონენტებით.....	321
ცხრილებს შორის ბმების შექმნა	332
მონაცემების გაფილტვრა.....	333
მონაცემების დახარისხება	333
მონაცემების ძებნა	334
ნავიგაცია	335
გამოთვლები.....	335
ცხრილებს შორის არსებული ბმებით მანიპულირება	337
ADO.NET კლასების მომიხილვა	338
ADO.NET კლასები და ობიექტები	338
ობიექტი-მომხმარებლები და შესაბამისი კლასები	338
ობიექტი-მიმწოდებლები და შესაბამისი კლასები	340
System.Data სახელების სივრცე	340
მოთხოვნების შესრულება ADO.NET კლასების გამოყენებით	341
მონაცემების წაკითხვა DataReader ობიექტის საშუალებით	341
მონაცემთა ბაზასთან მუშაობა DataSet ობიექტის გამოყენებით.....	343
მონაცემების კითხვა	343
მონაცემების გაფილტვრა.....	347
მონაცემების დახარისხება	348
სტრიქონების ძებნა.....	349
მონაცემთა ბაზაში მონაცემების შეცვლა.....	350
SELECT, UPDATE, INSERT და DELETE ბრძანებების ნახვა	353

ცხრილში სტრიქონების დამატება.....	354
სტრიქონების წაშლა	355
ორ ცხრილს შორის ბმის შექმნა.....	357
SQL ბრძანებების უშუალო შესრულება.....	361
ერთი მნიშვნელობის მიღება	361
UPDATE მოთხოვნის უშუალოდ შესრულება	361
INSERT მოთხოვნის უშუალოდ შესრულება	367
DELETE მოთხოვნის უშუალოდ შესრულება	368
ტრანზაქციებთან მუშაობა ADO.NET საშუალებებით	370
შენახული პროცედურის გამოძახება	371
ფუნქციის გამოძახება	373
თავი 18. ინტეგრირებული მოთხოვნების ენა LINQ.....	375
var საკვანძო სიტყვა	375
LINQ მოთხოვნის სინტაქსი	376
LINQ მეთოდის სინტაქსი და ლამბდა გამოსახულებები	377
მოთხოვნის შედეგების დახარისხება	378
მონაცემთა დიდი ზომის ნაკრებთან მუშაობა.....	379
აგრეგირების ოპერატორები	380
რთულ ობიექტებთან მუშაობა	381
მოთხოვნაში ახალი ობიექტის შექმნა.....	382
LINQ მეთოდები.....	384
Distinct() მეთოდი	384
Any() და All() მეთოდები	384
დახარისხება რამდენიმე ველის მიხედვით	385
ჯგუფური მოთხოვნა.....	385
Take() და Skip() მეთოდები	386
First() და FirstOrDefault() მეთოდები	386
Intersect(), Except() და Union() მეთოდები.....	386
Joins() მეთოდი	387
LINQ to SQL.....	388
ობიექტურ-რელაციური ასახვა	388
ნავიგაცია LINQ to SQL ბმების მიმართ	389
Group და Orderby გაფართოებული მოთხოვნები	394
მონაცემების მიბმა LINQ to SQL-თან	396
ADO.NET და LINQ.....	399
დანართი 1. სავარჯიშოები თავების მიხედვით.....	402
დანართი 2. სავარჯიშოების ამოხსნები.....	433
ლიტერატურა	506

წინასიტყვაობა

Visual Studio .NET წარმოადგენს დაპროგრამების თანამედროვე ტექნოლოგიას. მისი საშუალებით შესაძლებელია სხვადასხვა ტიპის პროგრამა-დანართების შემუშავება, როგორცაა, Windows და ვებ-პროგრამები, ქსელური პროგრამები, მონაცემთა ბაზები და ა.შ. ის განკუთვნილია ქსელში ჩართულ კომპიუტერებთან და მოწყობილობებთან სამუშაოდ.

წიგნის მიზანია მკითხველს შეასწავლოს C# ენა, ობიექტზე ორიენტირებული პროგრამირების საფუძვლები და განკუთვნილია დამწყები პროგრამისტებისათვის. თუმცა, გადმოცემულმა მასალამ შეიძლება პროფესიონალებიც დაინტერესოს.

სახელმძღვანელო წარმოადგენს მეოთხე, გადამუშავებულ გამოცემას. წინა გამოცემისაგან განსხვავებით წიგნში გაახლებულია მონაცემთა ბაზებთან მუშაობის საკითხები და დამატებულია ინტეგრირებული მოთხოვნების ენა - LINQ. დაწვრილებითაა აღწერილი მისი გამოყენების ხერხები და მეთოდები.

წიგნში აღწერილია Microsoft Visual Studio .NET 2012 სისტემა.

აუცილებელი პროგრამული უზრუნველყოფაა - Windows 7, Windows 8, Visual Studio.NET.

წიგნი არის ზემოთ აღნიშნული საკითხების ქართულ ენაზე გადმოცემის ერთ-ერთი პირველი მცდელობა, ამიტომ ის არ არის დაზღვეული ხარვეზებისაგან. ავტორი მაღლიერებით მიიღებს წიგნში გადმოცემული მასალის დახვეწისა და სრულყოფის მიზნით გამოთქმულ შენიშვნებსა და მოსაზრებებს. ისინი შეგიძლიათ მომაწოდოთ მისამართებზე:

rsamkharadze@mail.ru, roman.samkharadze@yahoo.com, samkharadze.roman@gmail.com .

საქართველოს ტექნიკური უნივერსიტეტის
ინფორმატიკისა და მართვის სისტემების ფაკულტეტის
კომპიუტერული ინჟინერიის დეპარტამენტის
სრული პროფესორი, ტექნიკის მეცნიერებათა დოქტორი

რომან სამხარაძე

I ნაწილი. C# ენა

თავი 1. შესავალი

პროგრამირების ისტორიის მოკლე მიმოხილვა

დაპროგრამების ენები გამოიყენება ისეთი მრავალფეროვანი ამოცანების გადასაწყვეტად, როგორცაა მონაცემთა საინფორმაციო სისტემების მართვა, რთული მათემატიკური და ეკონომიკური ამოცანების გადაწყვეტა, მედიცინა და ა.შ. C# ენა, რომელიც შეიმუშავა Microsoft კომპანიამ, მთლიანად პასუხობს პროგრამირების თანამედროვე სტანდარტებს და განკუთვნილია .NET Framework ტექნოლოგიის განვითარების უზრუნველყოფისათვის. ის არის დაპროგრამების მძლავრი ენა განკუთვნილი Windows გარემოში მომუშავე თანამედროვე კომპიუტერული სისტემებისთვის, რომლებიც იყენებენ ინტერნეტ-ტექნოლოგიებს.

დაპროგრამების ენებს შორის არსებობს კავშირი. ყოველი ახალი ენა ადრე შექმნილი ენებისაგან მემკვიდრეობით იღებს გარკვეულ თვისებებს. კერძოდ, C# ენამ ბევრი სასარგებლო თვისება C, C++ და Java ენებისაგან მემკვიდრეობით მიიღო.

C ენა შეიქმნა ნიუ-ჯერსის შტატის ქალაქ მიურეი-ჰილის Bell Laboratories კომპანიის სისტემური პროგრამისტის დენის რიჩის მიერ 1972 წელს. თითქმის მთლიანად ამ ენაზე დაიწერა Unix ოპერაციული სისტემის ბირთვი. შემდგომში პროგრამების ზომისა და სირთულის ზრდამ საკმაოდ გააძნელა მათთან მუშაობა. გამოსავალი იყო სტრუქტურულ პროგრამირებაზე გადასვლა. სწორედ C გახდა 1980 წლებიდან ყველაზე ხშირად გამოყენებადი სტრუქტურული პროგრამირების ენა.

პროგრამირების განვითარებასთან ერთად კვლავ დადგა დიდი ზომის პროგრამებთან მუშაობის პრობლემა. აუცილებელი გახდა ახალი მიდგომების შემუშავება. ერთ-ერთი მათგანია ობიექტზე ორიენტირებული პროგრამირება (ოპ). ის იძლევა დიდი ზომის პროგრამებთან ეფექტური მუშაობის შესაძლებლობას. შესაბამისად, შეიქმნა C ენის ობიექტზე ორიენტირებული ვერსია, რომელსაც 1983 წლიდან C++ დაერქვა. ის შემუშავებულ იქნა იმავე Bell Laboratories კომპანიაში ბიარნ სტრაუსტრაპის მიერ. C++ მთლიანად მოიცავს C ენას და შეიცავს ობიექტზე ორიენტირებული პროგრამირების შესაძლებლობებს. 1990 წლიდან იწყება C++ ენის მასობრივი გამოყენება და ის ხდება ყველაზე პოპულარული დაპროგრამების ენებს შორის.

დაპროგრამების ენების განვითარების საქმეში მნიშვნელოვანი მიღწევა იყო Java ენის შემუშავება Sun Microsystem კომპანიაში. მისი ავტორები იყვნენ ჯეიმს გოსლინგი, პატრიკ ნოტონი, კრის ვორტი, ედ ფრანკი და მაიკ შერიდანი. მასზე მუშაობა დაიწყო 1993 წლიდან. Java არის სტრუქტურული ობიექტზე ორიენტირებული ენა, რომელმაც C++ ენიდან აიღო სინტაქსი და სტრატეგია. ინტერნეტის ფართო გავრცელებამდე პროგრამების უმრავლესობის კომპილირება ხდებოდა კონკრეტული პროცესორისა და ოპერაციული სისტემისათვის. ინტერნეტის განვითარებასთან ერთად გაჩნდა სხვადასხვა პროცესორებისა და ოპერაციული სისტემის მქონე კომპიუტერების დაკავშირების შესაძლებლობა. ამან წინა პლანზე წამოსწია ერთი პლატფორმიდან მეორეზე პროგრამების ადვილად გადატანის პრობლემა. მის გადასაწყვეტად საჭირო გახდა ახალი ენის შემუშავება. სწორედ ასეთი ენა გახდა Java.

Java თავიდანვე შეიქმნა როგორც პლატფორმაზე დამოუკიდებელი ენა. მას შეეძლო პლატფორმათაშორის გადატანადი კოდის შექმნა, რაც გახდა მისი სწრაფი გავრცელების მიზეზი. გადატანადობა მიიღწევა პროგრამის საწყისი კოდის შუალედურ ენაში ტრანსლირების გზით, რომელსაც ბაიტ-კოდი ეწოდება. შემდეგ ეს შუალედური ენა სრულდება Java ვირტუალური მანქანის მიერ (Java Virtual Machine, JVM). შედეგად, Java-პროგრამა შეიძლება შესრულდეს ნებისმიერ პლატფორმაზე, რომელსაც Java ვირტუალური მანქანა აქვს.

Java ენისაგან განსხვავებით C და C++ ენებში პროგრამის საწყისი კოდის კომპილაცია ხორციელდება შესრულებად კოდში, რომელიც დაკავშირებულია კონკრეტულ პროცესორთან და ოპერაციულ სისტემასთან. ასეთი პროგრამის გასაშვებად სხვადასხვა პლატფორმაზე აუცილებელია პროგრამის საწყისი კოდის კომპილირება თითოეული პლატფორმისათვის. ეს კი შრომატევადი და ძვირადღირებული პროცესია. ამიტომაც, C# ენაში გადმოტანილი იქნა Java ენაში გამოყენებული მიდგომა - შუალედური ენის გამოყენება.

მართალია, Java ენამ გადაჭრა ერთი პლატფორმიდან მეორეზე პროგრამების გადატანასთან დაკავშირებული ბევრი პრობლემა, მაგრამ, მას აქვს რამდენიმე ნაკლი. მაგალითად, ის ვერ უზრუნველყოფს რამდენიმე დაპროგრამების ენის ურთიერთქმედებას, ე.ი. არ უზრუნველყოფს მრავალენობრივ პროგრამირებას. მრავალენობრივი პროგრამირების ქვეშ იგულისხმება დაპროგრამების სხვადასხვა ენაზე დაწერილი კოდის ერთად მუშაობის უნარი. ეს შესაძლებლობა მეტად მნიშვნელოვანია დიდი პროგრამების შემუშავებისას, აგრეთვე, ცალკეული კომპონენტების შექმნისას, რომელთა გამოყენება შესაძლებელი იქნება მრავალ დაპროგრამების ენასა და სხვადასხვა ოპერაციულ სისტემაში.

Java ენის ერთ-ერთი სერიოზული ნაკლია, აგრეთვე, Windows პლატფორმის პირდაპირი უზრუნველყოფის არქონა, რომელიც ამჟამად მსოფლიოში ყველაზე გავრცელებული ოპერაციული სისტემაა. Java-პროგრამების შესრულება Windows გარემოში შესაძლებელია იმ შემთხვევაში თუ დაყენებულია (დაინსტალირებულია) Java ვირტუალური მანქანა.

ამ პრობლემის გადასაწყვეტად Microsoft კომპანიამ 1990 წლების ბოლოს შეიმუშავა C# ენა. მისი ავტორია ანდერს ჰელსბერგი. C# ენა მჭიდროდაა დაკავშირებული C, C++ და Java ენებთან. ის აგებულია C++ ენაში განსაზღვრულ ობიექტურ მოდელზე, C ენიდან აღებულია სინტაქსი, ოპერატორები და საკვანძო სიტყვები, Java ენიდან კი - შუალედური ენის გამოყენება. C# ენის ერთ-ერთი მნიშვნელოვანი სიახლეა პროგრამული უზრუნველყოფის კომპონენტების ჩადგმული უზრუნველყოფა. ფაქტობრივად C# ენა შექმნილია, როგორც კომპონენტებზე ორიენტირებული ენა, რომელიც მოიცავს ისეთ ელემენტებს, როგორიცაა თვისებები, მეთოდები და მოვლენები. მაგრამ, ყველაზე მნიშვნელოვანი სიახლე C# ენაში არის მრავალენობრივ გარემოში მისი მუშაობის უნარი.

C# ენაზე პროგრამა-დანართების შემუშავება უფრო ადვილია, ვიდრე C++ ენაზე, რადგან მისი სინტაქსი უფრო მარტივია. არის საკითხების ძალიან მცირე წრე, რომელთა გადაწყვეტა შეიძლება მხოლოდ C++ ენაზე, მაგალითად მის კოდში ასენბლერ ენაზე შედგენილი კოდის ჩართვა და ა.შ. მიუხედავად ამისა, C# არის მძლავრი ენა, რომლის საშუალებითაც შესაძლებელია Windows-დანართების, Web-დანართების, Web-სამსახურებისა და პრაქტიკულად ნებისმიერი ტიპის პროგრამა-დანართის შემუშავება. C++ ენის ზოგიერთი ფუნქცია, მაგალითად სისტემურ მეხსიერებასთან პირდაპირი მიმართვა, C# ენაში შეიძლება რეალიზებული იყოს როგორც unsafe კოდი. ზოგჯერ, C# ენაზე შედგენილი კოდი უფრო დიდია, ვიდრე C++ ენაზე შედგენილი. ეს აიხსნება იმით, რომ C# ენა C++ ენისგან განსხვავებით არის უფრო უსაფრთხო ტიპების მიმართ (typesafe). ეს იმას ნიშნავს, რომ თუ ტიპს მივანიჭებთ რაიმე მნიშვნელობას, ის შემდეგ ვეღარ გარდაიქმნება სხვა ტიპად. C# ენის ერთ-ერთი უპირატესობაა ის, რომ იგი თავიდანვე შეიქმნა სპეციალურად .NET Framework-თვის, ამიტომ ის დიდი წარმატებით გამოიყენება .NET დანართების შესაქმნელად, ვიდრე სხვა ენები [21].

C# ენის კავშირი .NET Framework გარემოსთან

.NET Framework არის გარემო, რომელიც უზრუნველყოფს პლატფორმაზე დამოუკიდებელი პროგრამა-დანართების (Application) შემუშავებასა და შესრულებას. ის, აგრეთვე, უზრუნველყოფს პროგრამების გადატანადობას. შედეგად, Windows-პროგრამები

შეგვიძლია სხვა პლატფორმებზე ვამუშავოთ.

C# ენა იყენებს .NET Framework გარემოს ორ მთავარ ნაწილს. პირველია ენაზე დამოუკიდებელი შესრულების გარემო (Common Language Runtime, CLR). ის მართავს ჩვენი პროგრამის შესრულებას და წარმოადგენს .NET Framework ტექნოლოგიის ნაწილს, რომელიც უზრუნველყოფს პროგრამების გადატანადობასა და პროგრამირებას რამდენიმე ენის გამოყენებით. მეორეა - კლასების ბიბლიოთეკა .NET Framework, რომელიც პროგრამას საშუალებას აძლევს მიმართოს შესრულების გარემოს, მაგალითად, მონაცემების შეტანა-გამოტანისათვის.

მოკლედ განვიხილოთ CLR სისტემის მუშაობა. ფაილს, რომელიც შეიცავს C#-პროგრამის საწყის კოდს .cs გაფართოება აქვს. ამ ფაილის კომპილირებას მანქანურ კოდებში ასრულებს პროგრამა, რომელსაც *კომპილატორი* ეწოდება. C#-პროგრამების კომპილირების შედეგად მიიღება ფაილი, რომელიც შეიცავს შუალედურ ენას - MSIL (Microsoft Intermediate Language, MSIL). ის შედგება გადასატანი ინსტრუქციების ნაკრებისაგან, რომელიც არაა დამოკიდებული კონკრეტული პროცესორის ინსტრუქციების ნაკრებზე. CLR სისტემა ახდენს შუალედური კოდის ტრანსლირებას შესრულებად კოდში პროგრამის გაშვების დროს. MSIL ენაში კომპილირებული პროგრამა შეიძლება ნებისმიერ ოპერაციულ სისტემაში შესრულდეს.

MSIL ენა შესრულებად კოდში გარდაიქმნება JIT კომპილატორის მიერ (just in time - საჭირო მომენტში). C#-პროგრამის გაშვებისას CLR სისტემა ააქტიურებს JIT კომპილატორს, რომელიც MSIL ენას გარდაქმნის მოცემული პროცესორის შიდა კოდად. ამასთან, პროგრამის ნაწილების გარდაქმნა საჭიროებისდა მიხედვით სრულდება.

.NET Framework გარემოში მუშაობის დროს ჩვენ ვქმნით *მართვად კოდს* (managed code), რომელიც სრულდება CLR სისტემის მართვის ქვეშ. მართვად კოდს შემდეგი უპირატესობები აქვს: მეხსიერების მართვა, სხვადასხვა ენების შეთავსების შესაძლებლობა, მონაცემების გადაცემის უსაფრთხოების მაღალი დონე, ვერსიის კონტროლის უზრუნველყოფა და პროგრამული უზრუნველყოფის კომპონენტების ადვილი ურთიერთქმედების საშუალება.

არსებობს, აგრეთვე, *არამართვადი კოდი* (unmanaged code), რომელსაც CLR სისტემა არ ასრულებს. .NET Framework გარემოს შექმნამდე ყველა კოდი იყო არამართვადი. ამჟამად, ორივე სახის კოდს შეუძლია ერთად მუშაობა. მაგალითად, C++ ქმნის მართვად კოდს, რომელსაც შეუძლია არამართვად კოდთან ურთიერთქმედება.

თუ ჩვენს მიერ შექმნილ კოდს გამოიყენებენ სხვა ენებზე დაწერილი პროგრამები, მაშინ მათი მაქსიმალური თავსებადობისათვის უნდა დავიცვათ საერთოენობრივი სპეციფიკაცია (Common Language Specification, CLS). ის აღწერს სხვადასხვა ენებისათვის საერთო მახასიათებლებს.

Visual Studio 2012

Visual Studio-ის (VS) ინსტალირების შემდეგ, მისი პირველი გაშვებისას, ეკრანზე გაიხსნება ფანჯარა, რომელშიც უნდა მოვნიშნოთ Visual C# Development Settings ოპცია (რეჟიმი). თუ შეცდომით სხვა ოპცია მოვნიშნეთ, მაშინ Visual C# Development Settings პარამეტრების ასარჩევად უნდა შევასრულოთ მათი იმპორტი. ამისათვის, ვასრულებთ Tools→Import and Export Settings ... ბრძანებას. გახსნილ ფანჯარაში (ნახ. 1.1) ჩავრთოთ Reset all settings გადამრთველი და დავაჭიროთ Next კლავიშს. მომდევნო ფანჯარაში (ნახ. 1.2) არსებული პარამეტრების შესანახად უნდა ჩავრთოთ Yes, save my current settings გადამრთველი და დავაჭიროთ Next კლავიშს. უკანასკნელ ფანჯარაში (ნახ. 1.3) მოვნიშნოთ Visual C# Development Settings ოპცია და დავაჭიროთ Finish კლავიშს.

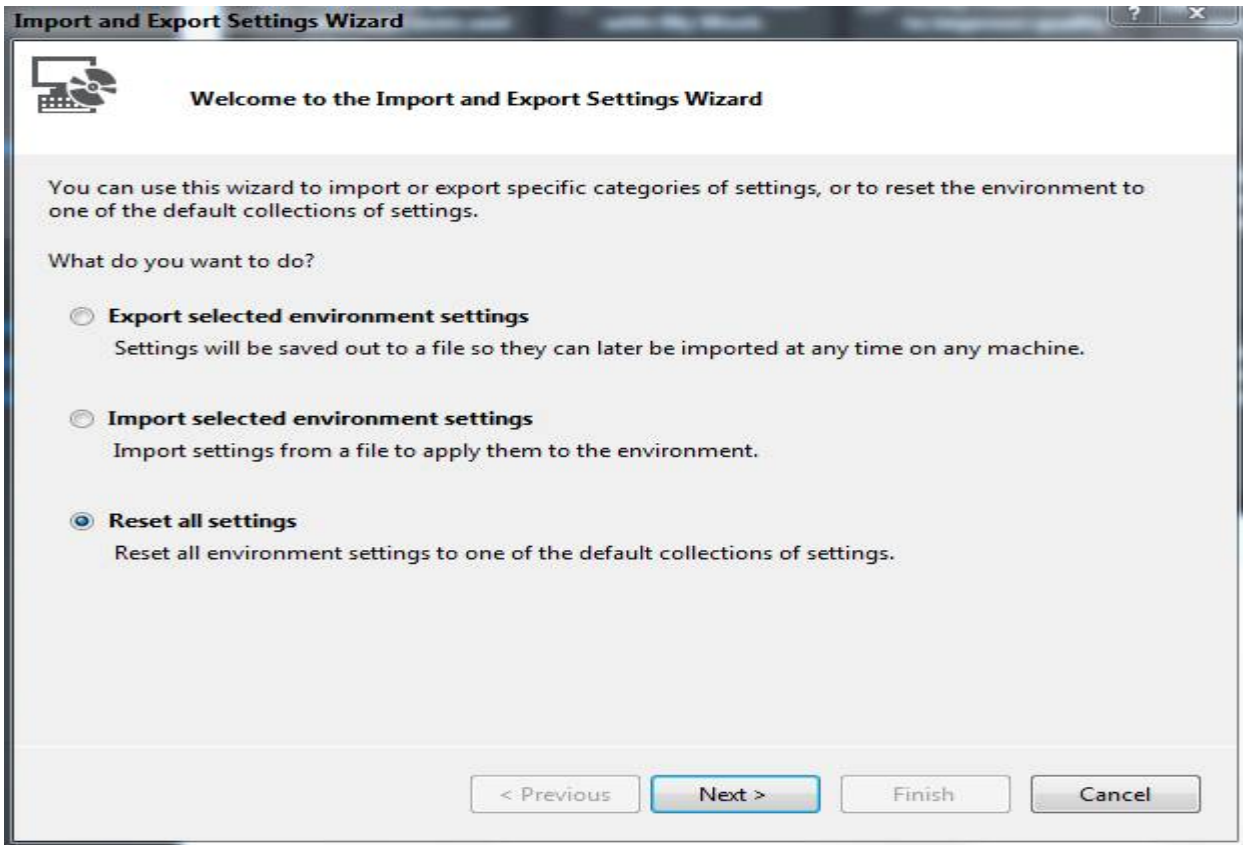
Visual Studio-ის გაშვების დროს გაიხსნება Start Page ფანჯარა (ნახ. 1.4), რომელშიც შემდეგი ფანჯრები აისახება:

1. Toolbox (ინსტრუმენტების პანელი) ფანჯარაში მოთავსებულია ვიზუალური და არავიზუალური კომპონენტები, რომლებიც გამოიყენება Windows-დანართებისთვის მომხმარებლის ინტერფეისის შესაქმნელად.
2. Server Explorer (სერვერის გზამკვლევი) ფანჯარა გამოიყენება მონაცემების წყაროსთან კავშირის დასამყარებლად, მაგალითად, როგორცაა SQL სერვერი და ა.შ.
3. Solution Explorer (გადაწყვეტების გზამკვლევი) ფანჯარაში აისახება ინფორმაცია მოცემულ მომენტში ჩატვირთული გადაწყვეტის (solution, решение) შესახებ. გადაწყვეტა არის ერთი ან მეტი პროექტი თავიანთ საკონფიგურაციო პარამეტრებთან ერთად. ამ ფანჯარაში აისახება ინფორმაცია გადაწყვეტის შემადგენლობაში არსებული პროექტების შესახებ, კერძოდ თუ რომელ ფაილებს მოიცავენ ისინი და ამ ფაილებში რა ინფორმაციაა მოთავსებული.
4. Properties (თვისებები) ფანჯარა იძლევა უფრო დაწვრილებით ინფორმაციას პროექტის ელემენტების შესახებ და საშუალებას გვაძლევს განვახორციელოთ მათი დამატებითი გაწყობები.
5. Error List (შეცდომების სია) ფანჯარაში აისახება შეცდომები, გაფრთხილებები და შეტყობინებები. მასში შეცდომები აისახება კოდის შეტანისა და კომპილირების დროს. მაგალითად, თუ ჩვენს მიერ შეტანილი კოდის რომელიმე სტრიქონში წავშლით ';' სიმბოლოს, თითქმის მაშინვე Error ფანჯარაში გამოჩნდება შესაბამისი შეტყობინება და შეცდომის შემცველი სტრიქონის ნომერი. თუ ამ შეტყობინებაზე ორჯერ სწრაფად დავაწკაპუნებთ, მაშინ შეცდომის შემცველ სტრიქონში მოინიშნება შესაბამისი ადგილი. თუმცა, ეს ადგილი, ორჯერ დაწკაპუნების გარეშეც მოინიშნება წითელი კლაკნილით. კოდის სტრიქონების დასანიმრად შეგვიძლია ჩავრთოთ Tools→Options→Text Editor→C#→General→Line numbers გადამრთველი.

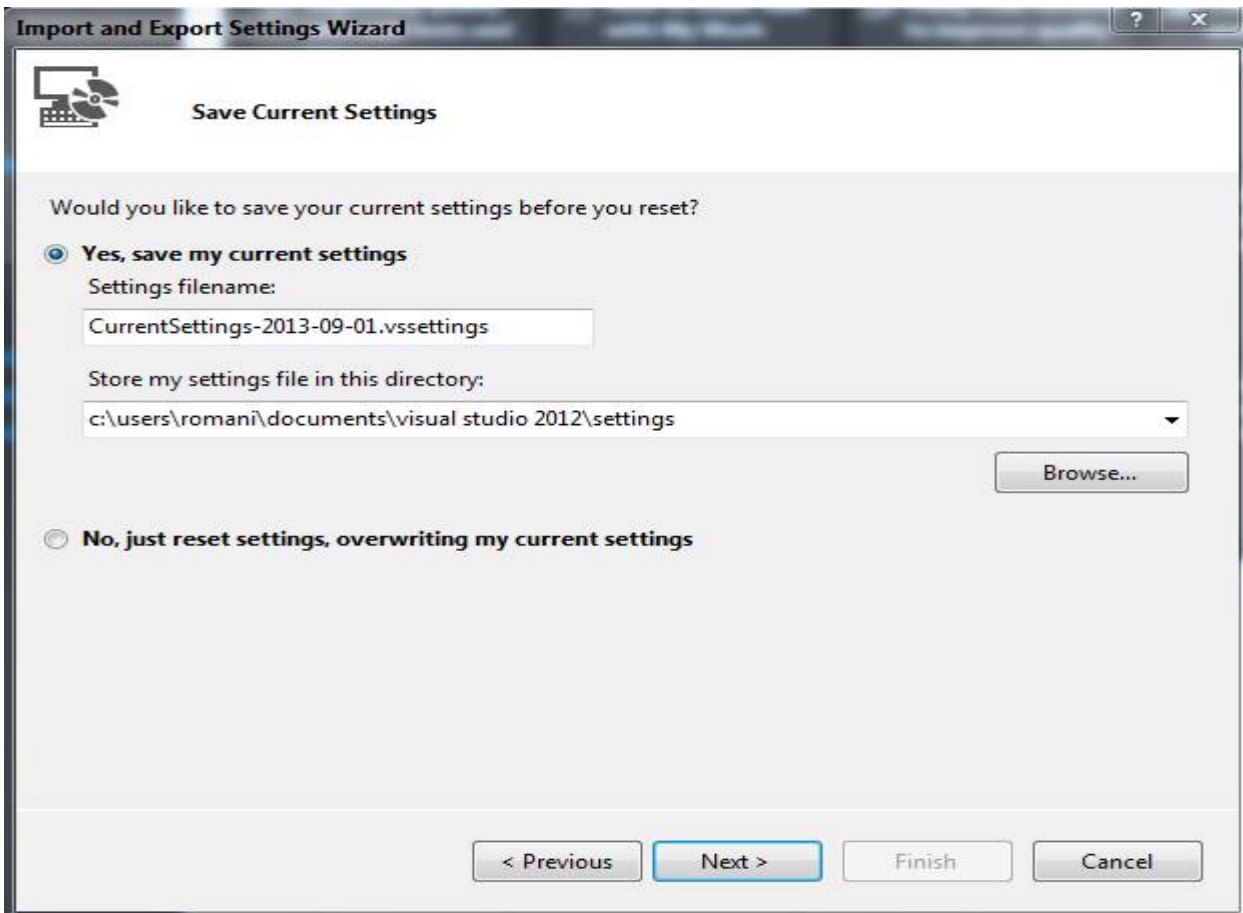
ობიექტზე ორიენტირებული დაპროგრამების პრინციპები

ჩვენს გარშემო ბევრი ობიექტია. მაგალითად, ავტომობილი, თვითმფრინავი, მატარებელი, მალაზია, ავადმყოფი, ექიმი, სტუდენტი, გეომეტრიული ფიგურა და ა.შ. მსგავსი ობიექტები შეგვიძლია დავაჯგუფოთ კლასებში. მაგალითად, სხვადასხვა ავტომობილს ბევრი საერთო თვისება აქვს, ამიტომ ისინი შეგვიძლია „ავტომობილი“ საერთო კლასში გავაერთიანოთ. ასევე სხვადასხვა ტიპის თვითმფრინავს ბევრი საერთო თვისება აქვს, ამიტომ ისინი შეგვიძლია „თვითმფრინავის“ საერთო კლასში გავაერთიანოთ და ა.შ.

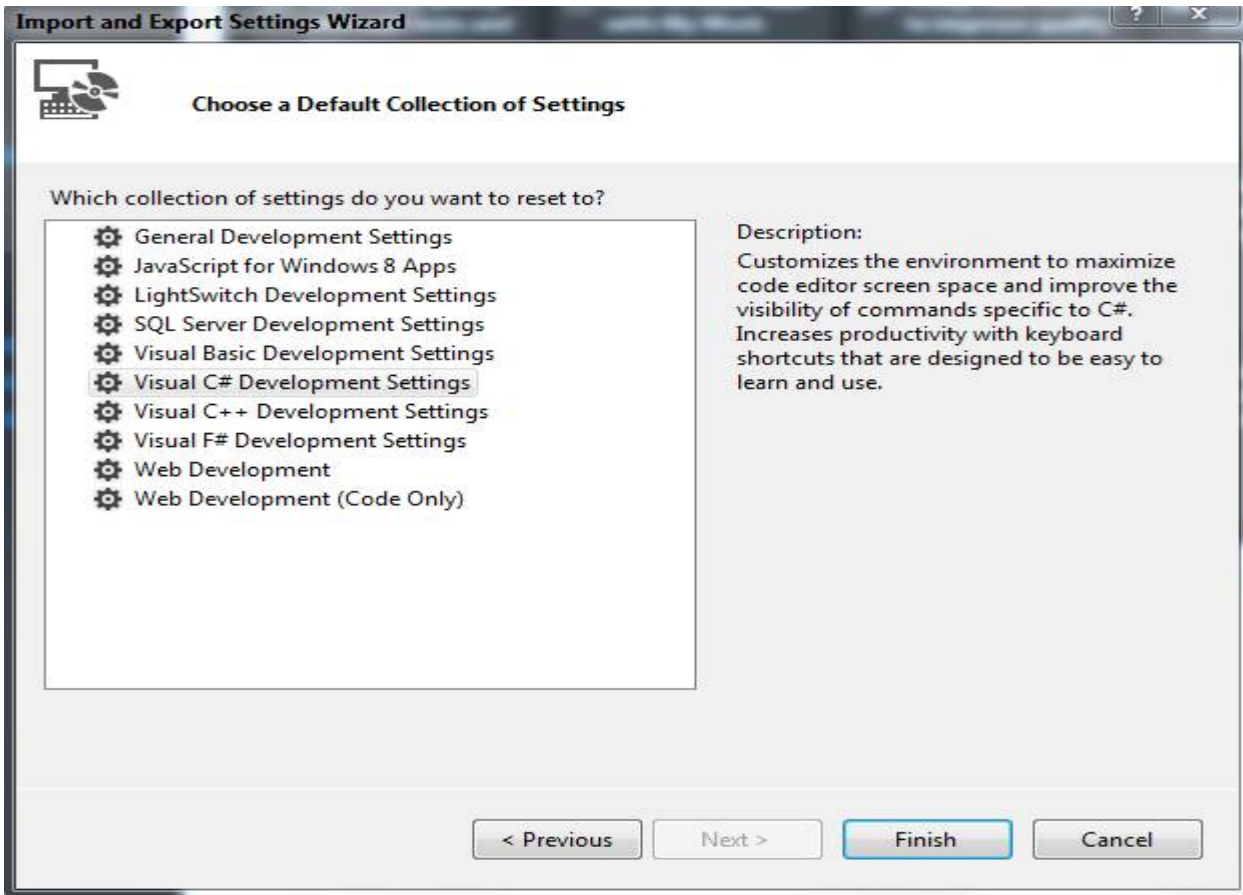
ობიექტზე ორიენტირებულ ენებში პროგრამები ორგანიზებულია მონაცემების გარშემო, ანუ ჩვენ განვსაზღვრავთ მონაცემებს და იმ პროგრამებს, რომლებიც ამ მონაცემებთან მუშაობენ. C# ენა ეფუძნება ობიექტზე ორიენტირებული დაპროგრამების (ოოდ) სამ პრინციპს პრინციპს: ინკაფსულაცია, პოლიმორფიზმი და მემკვიდრეობითობა. მოკლედ განვიხილოთ თითოეული მათგანი.



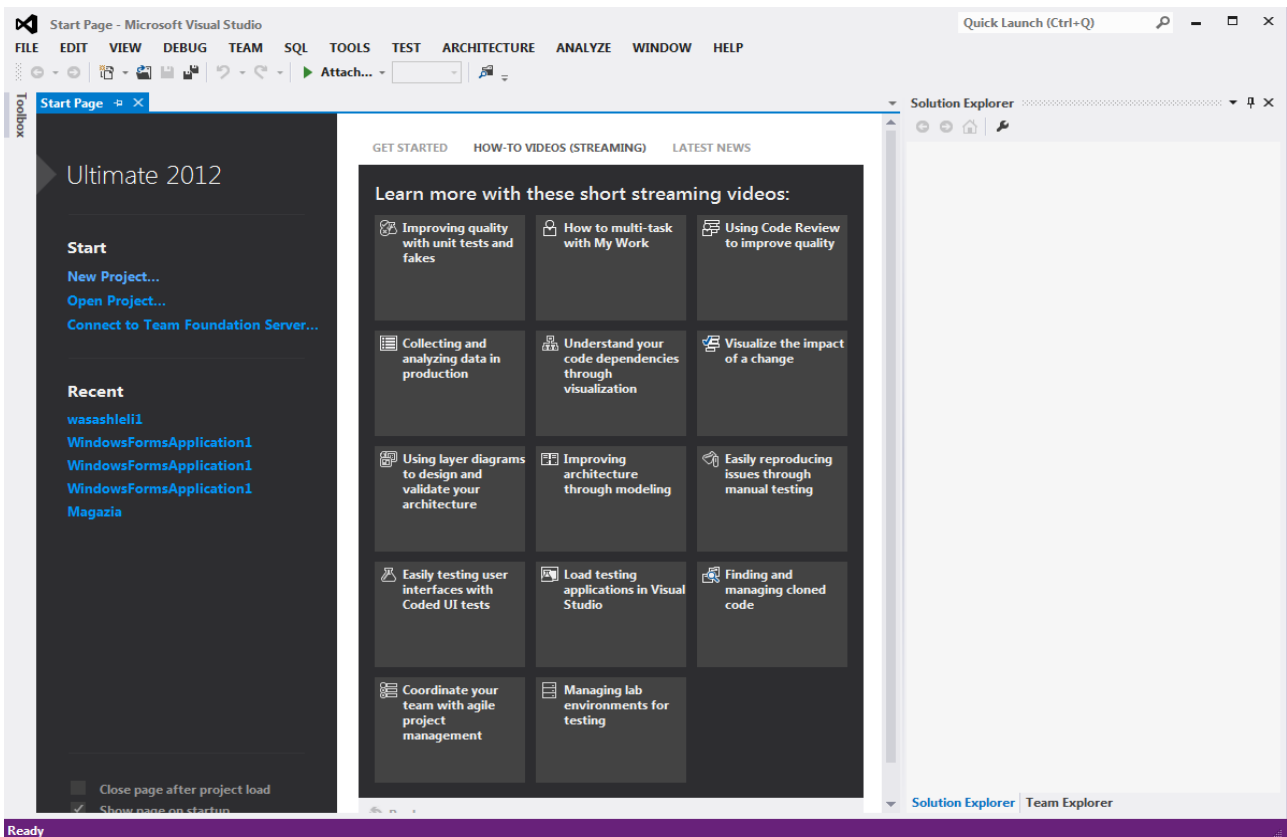
58b. 1.1.



58b. 1.2.



Біб. 1.3.



Біб. 1.4.

ინკაფსულაცია

ინკაფსულაცია (encapsulation) არის დაპროგრამების მექანიზმი, რომელიც აერთიანებს პროგრამის კოდსა და იმ მონაცემებს, რომლებთანაც ეს კოდი მუშაობს, აგრეთვე, გამიჯნავს მათ (კოდსა და მონაცემებს) სხვა პროგრამების მხრიდან მიმართვებისაგან. ამით ხდება მათი დაცვა არასწორი გამოყენებისაგან. ობიექტზე ორიენტირებულ ენაში პროგრამის კოდი და მონაცემები ერთმანეთს ისე უკავშირდება, რომ ქმნიან ერთ ავტონომიურ სტრუქტურას, რომელსაც ობიექტი ეწოდება.

ობიექტის შიგნით პროგრამის კოდი და მონაცემები სხვა ობიექტებისათვის შეიძლება იყოს დახურული (private) ან ღია (public). დახურულ (პრივატულ) კოდთან და მონაცემებთან მიმართვა შეუძლიათ მხოლოდ ამავე ობიექტში აღწერილ კოდებს. ეს იმას ნიშნავს, რომ დახურულ კოდსა და მონაცემებს ვერ მივმართავთ პროგრამის სხვა ნაწილიდან, რომელიც ობიექტის გარეთაა მოთავსებული. ღია (საერთოწვდომის) კოდთან და მონაცემებთან მიმართვა შესაძლებელია პროგრამის ნებისმიერი ნაწილიდან.

კლასი არის სტრუქტურა, რომელიც იყენებს რა ინკაფსულაციის პრინციპს, განსაზღვრავს კოდს და იმ მონაცემებს, რომლებთანაც ის მუშაობს. კლასი გამოიყენება ობიექტების აღსაწერად და შესაქმნელად. ობიექტები კლასის ეგზემპლარებს წარმოადგენენ. კლასის შემადგენელ კოდსა და მონაცემებს კლასის წევრები ეწოდებათ, კლასის მიერ განსაზღვრულ მონაცემებს კი - ეგზემპლარის ცვლადები. კლასში განსაზღვრულ პროგრამის კოდებს მეთოდები ეწოდებათ.

პოლიმორფიზმი

პოლიმორფიზმი (polymorphism) არის დაპროგრამების მექანიზმი, რომელიც მსგავს ობიექტებს საშუალებას აძლევს ერთი ინტერფეისის გამოყენებით მიმართონ სხვადასხვა მეთოდებს. სხვა სიტყვებით, რომ ვთქვათ, ერთი და იგივე სახელი შეიძლება რამდენიმე მეთოდს ჰქონდეს, რომლებიც ერთსა და იმავე მოქმედებებს სხვადასხვა ტიპის მონაცემებზე ასრულებენ. ამ შემთხვევაში არგუმენტის ტიპის მიხედვით სრულდება შესაბამისი მეთოდის გამოძახება. კონკრეტულ მეთოდს არგუმენტის ტიპის მიხედვით ირჩევს კომპილატორი. ამრიგად, პოლიმორფიზმი საშუალებას გვაძლევს რამდენიმე სახელის ნაცვლად გამოვიყენოთ ერთი. პოლიმორფიზმის უზრუნველყოფა ხდება მეთოდების გადატვირთვის საშუალებით.

მემკვიდრეობითობა

მემკვიდრეობითობა (inheritance) არის დაპროგრამების მექანიზმი, რომლის საშუალებითაც ერთ ობიექტს მემკვიდრეობით გადაეცემა მეორე ობიექტის ელემენტები. ამავე დროს დაცულია იერარქიული სტრუქტურა მიმართული ზევიდან ქვევით. მაგალითად, კლასი "ლომი" არის კატისებრთა კლასის ნაწილი, რომელიც თავის მხრივ ეკუთვნის „მტაცებლები“ კლასს. ეს უკანასკნელი კი არის „ცხოველები“ კლასის ნაწილი. „ცხოველები“ კლასს აქვს ისეთი მახასიათებლები, როგორცაა ტყეში ცხოვრება და ა.შ. იგივე მახასიათებლები შეგვიძლია გამოვიყენოთ „მტაცებლები“ კლასის მიმართ, რომელსაც დამატებით აქვს თავისი სპეციფიკური მახასიათებლები, როგორცაა ნადირობის უნარი და ა.შ., რომელიც მას სხვა ცხოველებისაგან განასხვავებს. აღნიშნული მახასიათებლების მატარებელია „კატისებრთა“ კლასი. ის დამატებით შეიცავს მისთვის დამახასიათებელ მახასიათებლებს, რომლებიც მას სხვა მტაცებლებისაგან განასხვავებს. დაბოლოს, „ლომი“ კლასი ყველა ზემოთ აღნიშნული მახასიათებლების მატარებელია და დამატებით შეიცავს მხოლოდ მისთვის დამახასიათებელ სპეციფიკურ მახასიათებლებს.

მემკვიდრეობითობის გამოყენება ობიექტს საშუალებას აძლევს განსაზღვროს ის ელემენტები, რომლებიც მხოლოდ მისთვისაა დამახასიათებელი. ობიექტი მშობელი (წინაპარი) კლასისაგან მემკვიდრეობით იღებს საერთო ელემენტებს. ამიტომ, მემკვიდრე კლასის გამოცხადებისას აღარ ხდება საჭირო მისი ყველა ელემენტის აღწერა.

თავი 2. C# ენის საფუძვლები

პირველი პროგრამა

C# ენაზე შევადგინოთ უმარტივესი პროგრამა, რომელიც შეასრულებს ორი მთელი რიცხვის შეკრებას.

```
{  
//   პროგრამა 2.1  
//   ეს არის ჩვენი პირველი პროგრამა  
int ricxvi1, ricxvi2, jami;  
  
ricxvi1 = 2;  
ricxvi2 = 3;  
jami   = ricxvi1 + ricxvi2;  
}
```

როგორც ვხედავთ C# ენაზე შედგენილი პროგრამა იწყება გამხსნელი ფიგურული ფრჩხილით { და მთავრდება დამხურავი ფიგურული ფრჩხილით }. პროგრამის მეორე და მესამე სტრიქონებში მოთავსებულია კომენტარები, რომლებიც // სიმბოლოებით იწყება (კომენტარებს მოგვიანებით განვიხილავთ). მომდევნო სტრიქონში ხდება ცვლადების გამოცხადება. int სიტყვა (integer - მთელი) იწყებს ცვლადების გამოცხადებას და მიუთითებს, რომ ისინი მთელ რიცხვებს წარმოადგენენ. მას მოსდევს ცვლადების სახელები ანუ იდენტიფიკატორები, რომლებიც ერთმანეთისაგან მძიმეებით გამოიყოფა. წერტილ-მძიმე ამთავრებს ცვლადების გამოცხადებას, აგრეთვე, ერთმანეთისაგან გამოყოფს ოპერატორებსა და მეთოდებს (მეთოდი არის მცირე ზომის პროგრამა, რომელიც გარკვეულ მოქმედებებს ასრულებს). ცვლადების სახელები აუცილებლად უნდა იწყებოდეს სიმბოლოთი და უმჯობესია შევარჩიოთ შინაარსიდან გამომდინარე. C# ენაში ნებისმიერი ცვლადი გამოცხადებული უნდა იყოს მის გამოყენებამდე.

ცვლადების გამოცხადების შემდეგ ricxvi1 ცვლადს ენიჭება მთელი რიცხვი - 2. "=" სიმბოლო წარმოადგენს მინიჭების ოპერატორს. ის ახდენს მის მარჯვნივ მოთავსებული მნიშვნელობის გადაწერას მის მარცხნივ მოთავსებულ ცვლადში. იგივე ეხება პროგრამის მომდევნო სტრიქონს. უკანასკნელ სტრიქონში jami ცვლადს ენიჭება ricxvi1 და ricxvi2 ცვლადების მნიშვნელობების ჯამი.

თვალსაჩინოების მიზნით პროგრამის თითოეულ სტრიქონში მოთავსებულია თითო ოპერატორი. თუმცა ერთ სტრიქონში შეიძლება რამდენიმე ოპერატორის მოთავსება. მთავარია, რომ ისინი ერთმანეთისაგან წერტილ-მძიმეებით იყოს გამოყოფილი.

ერთ-ერთი ნაკლი, რომელიც ამ პროგრამას აქვს ის არის, რომ სხვადასხვა რიცხვების შესაკრებად მოგვიწევს პროგრამის საწყისი კოდის შეცვლა, რაც არასწორია. გარდა ამისა, პროგრამას შედეგი ეკრანზე არ გამოაქვს. ამ ნაკლის აღმოსაფხვრელად გამოვიყენებთ ვიზუალურ კომპონენტებს, კერძოდ, მონაცემების შესატანად გამოვიყენებთ textBox კომპონენტს, გამოსატანად - label კომპონენტს, პროგრამის (მეთოდის) შესასრულებლად კი - button კომპონენტს. მათი გამოყენებით პროგრამა 2.1. შემდეგ სახეს მიიღებს:

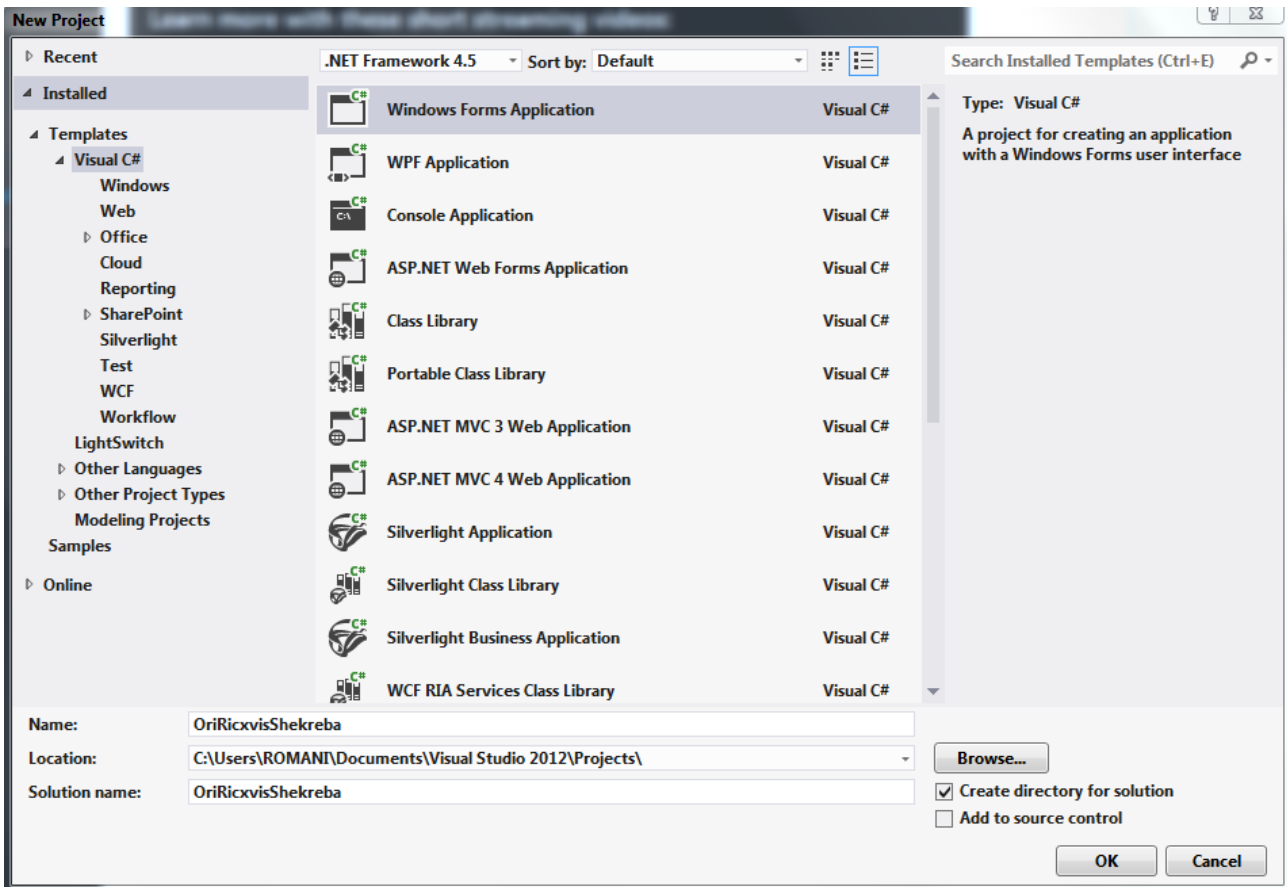
```
{  
//   პროგრამა 2.2  
//   ორი რიცხვის შეკრების პროგრამა  
int ricxvi1, ricxvi2, jami;  
  
ricxvi1 = Convert.ToInt32(textBox1.Text);
```

```

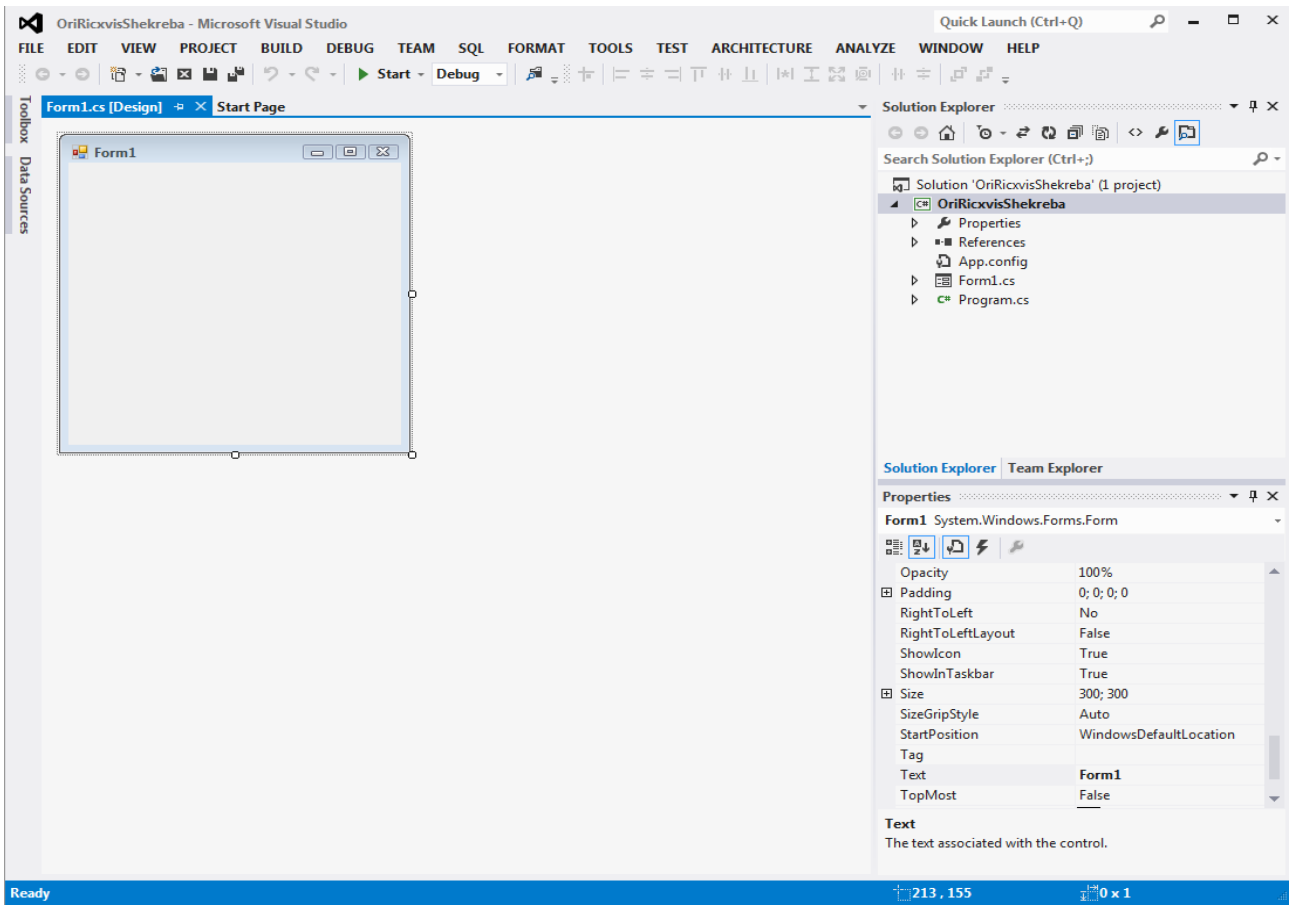
ricxvi2 = Convert.ToInt32(textBox2.Text);
jami = ricxvi1 + ricxvi2;
label5.Text = jami.ToString();
}

```

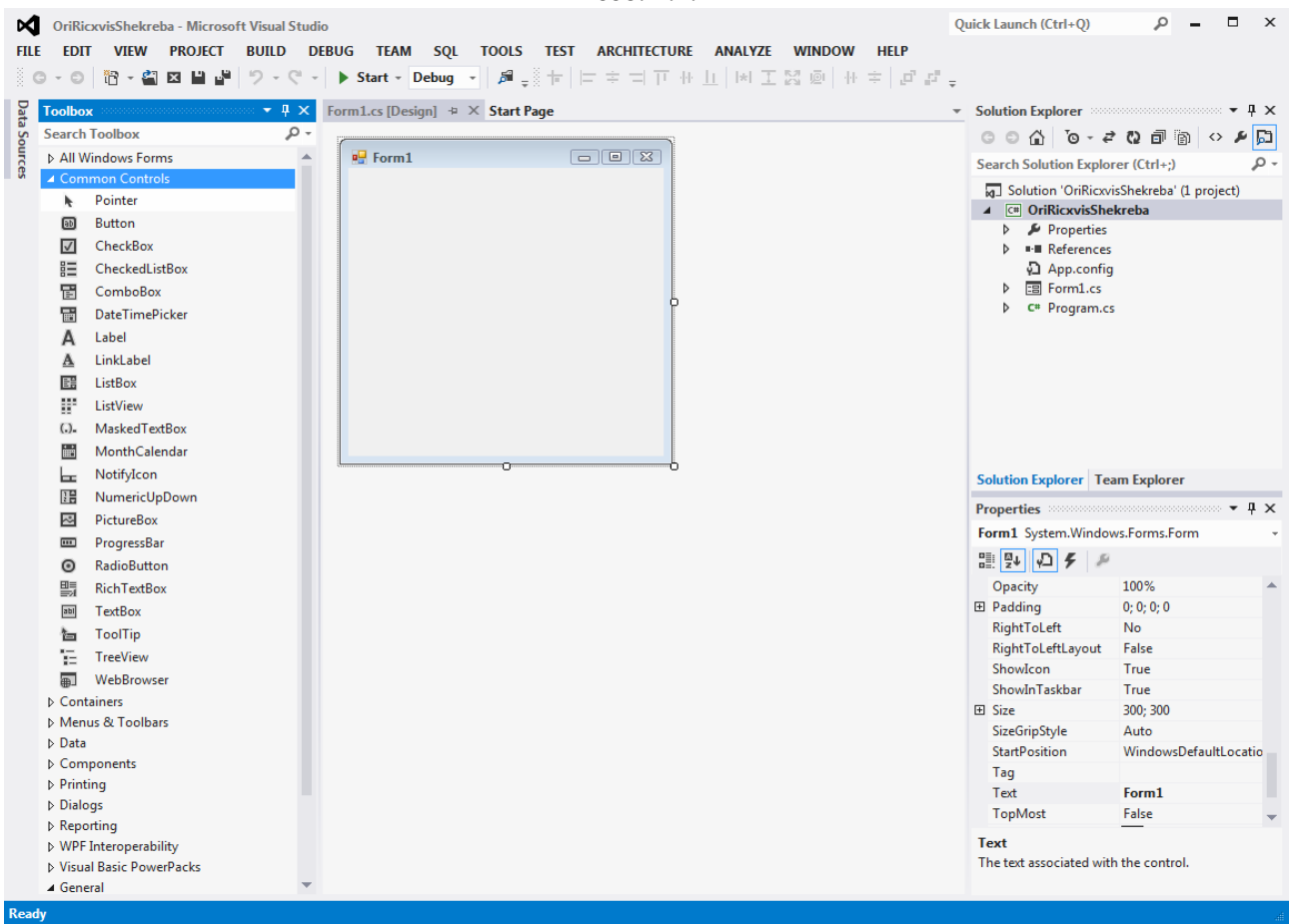
როგორც პროგრამიდან ჩანს ricxvi1 ცვლადს ენიჭება textBox1 კომპონენტში შეტანილი მნიშვნელობა (მაგალითად, 5), რომელიც Convert.ToInt32 მეთოდის საშუალებით მთელ რიცხვად გარდაიქმნება. საქმე ის არის, რომ textBox1 კომპონენტში შეტანილი მონაცემი მიენიჭება ამავე კომპონენტის Text თვისებას, რომელსაც სტრიქონული ტიპი აქვს. ამიტომ, ამ კომპონენტში შეტანილ ნებისმიერ მონაცემს სტრიქონული ტიპი ექნება. Convert.ToInt32 მეთოდი (პროგრამა) თავის არგუმენტს, ჩვენს შემთხვევაში ესაა textBox1.Text, გარდაქმნის რიცხვად. შედეგად, textBox1 კომპონენტში შეტანილი მონაცემი, რომელიც სინამდვილეში სტრიქონია, მთელ რიცხვად გარდაიქმნება. ასეთი გარდაქმნა საჭიროა იმიტომ, რომ ricxvi1 არის მთელი ტიპის მქონე ცვლადი, ამიტომ მას უნდა მიენიჭოს მხოლოდ მთელი რიცხვი. იგივე ეხება მომდევნო სტრიქონებს. ზოგად შემთხვევაში კი - მინიჭების ოპერატორის მარჯვნივ მოთავსებული გამოსახულების ტიპი დაყვანილი უნდა იყოს მინიჭების ოპერატორის მარცხნივ მოთავსებული ცვლადის ტიპზე.



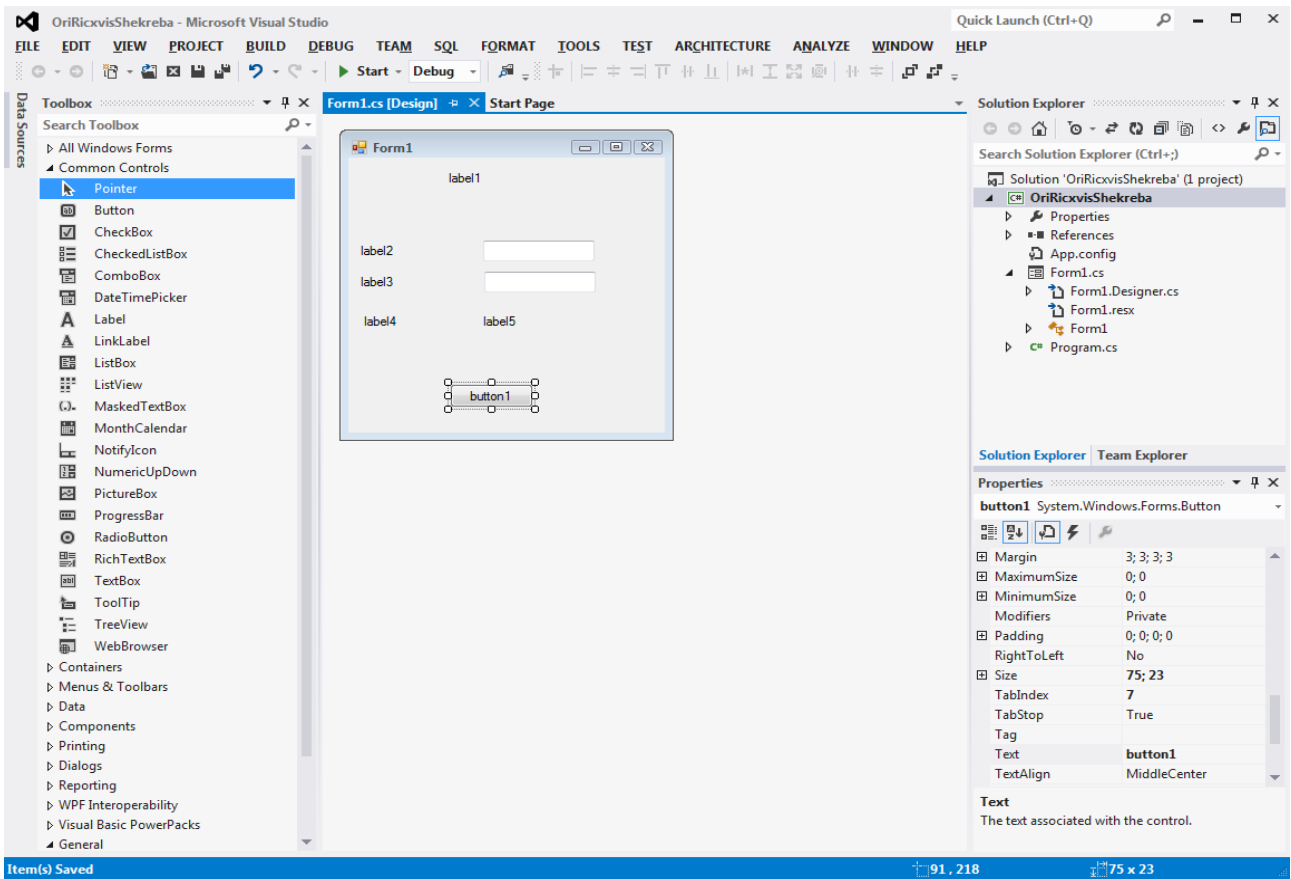
ნახ. 2.1.



б.2.2.



б.2.3.



ნახ. 2.4.

რაც შეეხება უკანასკნელ სტრიქონს, აქ მინიჭების ოპერატორის მარცხნივ მოთავსებულია label5 კომპონენტი. მის Text თვისებაში ვწერთ შედეგს, კერძოდ, jami ცვლადის მნიშვნელობას. რადგან Text თვისებას აქვს სტრიქონული ტიპი, ამიტომ jami ცვლადის ტიპიც უნდა გარდავქმნათ მთელი რიცხვიდან სტრიქონად. სწორედ ამას აკეთებს ToString() მეთოდი.

ახლა შევასრულოთ ეს პროგრამა. ამისათვის, ჯერ უნდა შევქმნათ პროექტი, რისთვისაც, ვუშვებთ Microsoft Visual Studio .NET 2012 სისტემას. გაიხსნება Start Page - Microsoft Visual Studio ფანჯარა (ნახ. 1.4). Start Page განყოფილებაში ვაჭერთ New Project მიმართვას. გაიხსნება New Project ფანჯარა (ნახ. 2.1). Templates ზონაში მოვნიშნით Visual C# ელემენტი. შემდეგ, მოვნიშნით Windows Forms Application ელემენტი. Name ველში უნდა შევიტანოთ პროექტის სახელი, მაგალითად, OriRicxvisShekreba. შემდეგ ვაჭერთ Browse კლავიშს და მოვნიშნავთ იმ კატალოგს, რომელშიც უნდა შეიქმნას OriRicxvisShekreba ქვეკატალოგი (შეგვიძლია წინასწარ შევქმნათ ის კატალოგი, რომელშიც მოვათავსებთ აღნიშნულ ქვეკატალოგს). ვაჭერთ Select Folder კლავიშს. ბოლოს ვაჭერთ Ok კლავიშს. თუ კატალოგს არ ავირჩევთ, მაშინ OriRicxvisShekreba ქვეკატალოგი Visual Studio 2012\Projects კატალოგში შეიქმნება. ეკრანზე გამოჩნდება ფორმა, რომლის სახელია Form1 (ნახ. 2.2). ფორმის მარცხნივ მოთავსებულია ვიზუალური კომპონენტების პანელი (ToolBox). თუ ეს პანელი არ ჩანს, მაშინ ვასრულებთ View→Toolbox ბრძანებას. ToolBox პანელზე მოვათავსოთ ისარი (კურსორი, მიმთითებელი). გამოჩნდება ინსტრუმენტების პანელი (ნახ. 2.3). გავხსნათ Common Controls პანელი. ავირჩიოთ textBox კომპონენტი. ამისათვის, მასზე მოვათავსოთ მიმთითებელი, დავაჭიროთ თავის მარცხენა კლავიშს, შემდეგ, მიმთითებელი მოვათავსოთ ფორმაზე საჭირო ადგილას და ისევ დავაჭიროთ თავის მარცხენა კლავიშს. შედეგად, კომპონენტი ფორმაზე გამოჩნდება მითითებულ ადგილზე. მისი სახელი იქნება textBox1. ასეთი გზით ფორმაზე მოვათავსოთ კიდევ ერთი textBox კომპონენტი - textBox2, ერთი Button კომპონენტი და ხუთი label კომპონენტი. ფორმა მიიღებს ნახ. 2.4-ზე ნაჩვენებ სახეს.

ვიზუალური კომპონენტები ფორმაზე შეგვიძლია განვათავსოთ ჩვენი მოსაზრებებიდან და გემოვნებიდან გამომდინარე. აქ შემოდის დიზაინის საკითხებიც, კერძოდ, ფორმაზე კომპონენტები ისე უნდა იყოს განლაგებული, რომ მათთან მუშაობა მოხერხებული იყოს.

მოვნიშნოთ label1 კომპონენტი და მის Text თვისებაში, რომელსაც ვიპოვით ფანჯრის მარჯვენა მხარეს მოთავსებული თვისებების (Properties) ფანჯარაში, შევიტანოთ ტექსტი: "ორი მთელი რიცხვის შეკრების პროგრამა" და დავაჭიროთ კლავიატურის Enter კლავიშს. შემდეგ, ამავე ფანჯრის Font თვისებაში ვირჩევთ ქართულ შრიფტს, ზომას და სტილს. ანალოგიურად ვიქცევით დანარჩენი კომპონენტებისთვისაც. მივიღებთ ნახ. 2.5-ზე ნაჩვენებ ფორმას.

ფორმაზე კომპონენტების განლაგების შემდეგ ორჯერ სწრაფად ვაწკაპუნებთ button1 კლავიშზე. გაიხსნება პროგრამის კოდის ზონა, რომელშიც შეგვაქვს პროგრამა 2.2. პროგრამის შეტანის შემდეგ მის შესანახად უნდა დავაჭიროთ ინსტრუმენტების პანელის Save All კლავიშს. ფორმაზე გადასასვლელად ვაჭერთ Shift+F7 კლავიშებს (+ ნიშნავს კლავიშებზე ერთდროულ დაჭერას), ფორმიდან პროგრამის კოდზე გადასასვლელად კი - F7 კლავიშს. პროგრამის შესასრულებლად გამოიყენება F5 კლავიში. თუ პროგრამა უშეცდომოდ შევიტანეთ, მაშინ ეკრანზე გამოჩნდება ფორმა (ნახ. 2.6). textBox1 და textBox2 კომპონენტებში შევიტანოთ მთელი რიცხვები და დავაჭიროთ button1 კლავიშს, რომლის სათაურში ჩანს სიტყვა "შეკრება". label5 კომპონენტში გამოჩნდება შეტანილი რიცხვების ჯამი. ამის შემდეგ, კვლავ შეგვიძლია შევიტანოთ სხვა რიცხვები და დავაჭიროთ "შეკრება" კლავიშს და ა.შ. ბოლოს, პროგრამის დასამთავრებლად ვაჭერთ ფორმის ზედა მარჯვენა კუთხეში მოთავსებულ კლავიშს.

პროგრამა 2.2 შეგვიძლია ასეც ჩავწეროთ:

```
{
//   პროგრამა 2.2.1
//   ორი რიცხვის შეკრების პროგრამა
int ricxvi1, ricxvi2, jami;

ricxvi1 = int.Parse(textBox1.Text);
ricxvi2 = int.Parse(textBox2.Text);
jami   = ricxvi1 + ricxvi2;
label5.Text = jami.ToString();
}
```

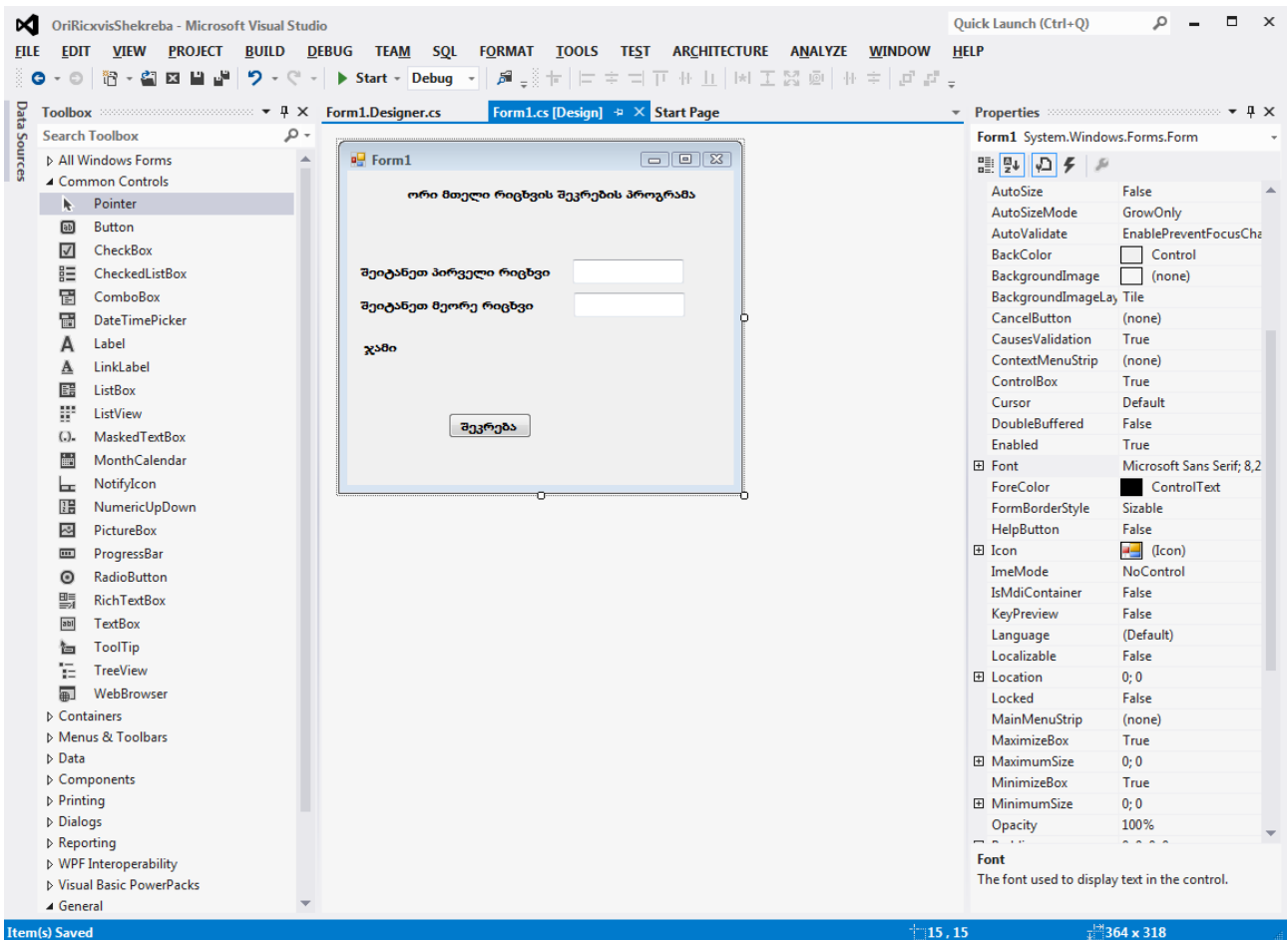
აქ Parse() მეთოდი textBox1 კომპონენტში მოთავსებულ მნიშვნელობას მთელ რიცხვად გარდაქმნის. შემდგომში ტიპის გარდაქმნის ხან ერთ ფორმას გამოვიყენებთ, ხან - მეორეს.

C# ენაში განსხვავებულია ზედა და ქვედა რეგისტრის სიმბოლოები. მაგალითად, განსხვავებულია სახელები Computer და computer. განვიხილოთ პროგრამა:

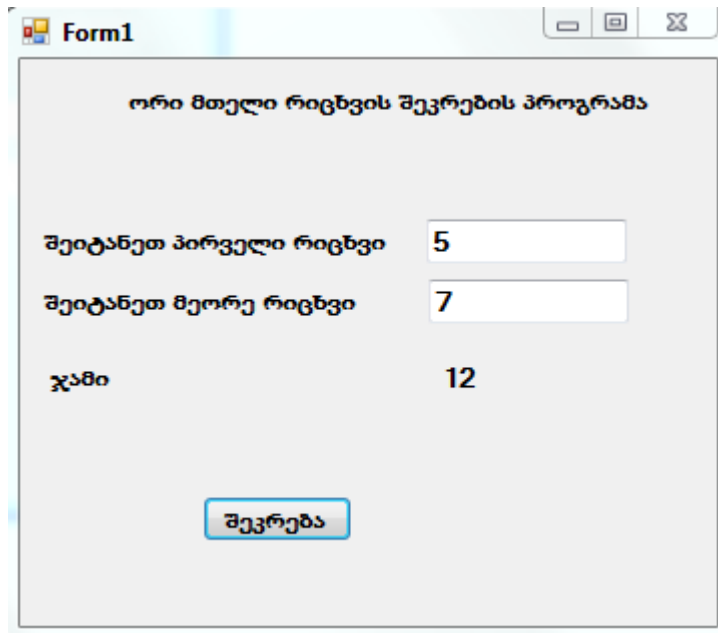
```
{
//   პროგრამა 2.3
//   C# ენაში განსხვავებულია ზედა და ქვედა რეგისტრის სიმბოლოები
int ricxvi1;
Ricxvi1 = 5; //   შეცდომა! Ricxvi1 ცვლადი არ არის გამოცხადებული
}
```

როგორც მაგალითიდან ჩანს, გამოცხადებული იყო ცვლადი სახელით - ricxvi1, ხოლო გამოიყენება სახელი - Ricxvi1. C# ენისთვის ეს ორი სხვადასხვა სახელია.

C# პროგრამის ფაილებს აქვთ .cs გაფართოება (ტიპი). პროგრამის კოდის ნაწილი ავტომატურად იქმნება Visual C#.NET სისტემის მიერ.



ნახ. 2.5.



ნახ. 2.6.

პროექტის გადატანა

როცა პროექტი გადაგვაქვს ერთი კომპიუტერიდან მეორეზე, მისი გახსნისას რომ გამოჩნდეს Designer ფანჯარა, ორჯერ სწრაფად უნდა დავაწკაპუნოთ Solution Explorer ფანჯრის Form1.cs ელემენტზე.

კომენტარები

C# ენაში არსებობს ერთსტრიქონიანი და მრავალსტრიქონიანი კომენტარი. ერთსტრიქონიანი კომენტარი // სიმბოლოებით იწყება. მრავალსტრიქონიანი კომენტარი /* სიმბოლოებით იწყება და */ სიმბოლოებით მთავრდება. როგორც წესი, კომენტარები შეიცავს პროგრამის სხვადასხვა ნაწილების განმარტებას, ცვლადებისა და მეთოდების დანიშნულებას და ა.შ. კომენტარების გამოყენება აადვილებს პროგრამის გარჩევას. პროგრამის შესრულების დროს ხდება კომენტარების გამოტოვება. ქვემოთ მოყვანილია პროგრამა, რომელიც ორივე სახის კომენტარს შეიცავს.

```
{
//      პროგრამა 2.4           ეს არის ერთსტრიქონიანი კომენტარი
/*
ეს არის ჩვენი               ეს არის მრავალსტრიქონიანი კომენტარი
პირველი პროგრამა
*/
int ricxvi1, ricxvi2, jami;

ricxvi1 = 2;
ricxvi2 = 3;
jami = ricxvi1 + ricxvi2;
}
```

C# ენის საბაზო ტიპები

მონაცემთა ჩვეულებრივი ტიპები

C# ენაში არსებობს ტიპების ორი ძირითადი კატეგორია: ჩვეულებრივი ტიპები (მარტივი ტიპები) და მიმართვითი ტიპები. ჩვეულებრივი ტიპის ცვლადები შეიცავენ კონკრეტულ მნიშვნელობებს, მიმართვითი ტიპის ცვლადები კი - ობიექტების მისამართებს. მიმართვით ტიპებს კლასების შესწავლისას განვიხილავთ. ჩვეულებრივი ტიპები მოყვანილია ცხრილში 2.1.

მთელრიცხვა ტიპები

მთელრიცხვა ტიპებია: char, byte, sbyte, short, ushort, int, uint, long და ulong. აქედან sbyte, short, int და long ტიპები გამოიყენება ნიშნის მთელი რიცხვების აღსაწერად, ხოლო byte, uint, ulong და ushort კი უნიშნო მთელი რიცხვების წარმოსადგენად. განსხვავება ნიშნის და უნიშნო მთელ რიცხვებს შორის შემდეგში მდგომარეობს. ნიშნის რიცხვებში უფროსი ბიტი (ორობითი თანრიგი) გამოიყენება ნიშნის მისათითებლად. თუ ეს ბიტი ნულის ტოლია, მაშინ რიცხვი დადებითია, თუ ერთის ტოლია, მაშინ - უარყოფითი. უნიშნო რიცხვებში უფროსი ბიტი არ გამოიყენება ნიშნის წარმოსადგენად და შედის რიცხვის შემადგენლობაში. ამიტომ, უნიშნო რიცხვების აბსოლუტური მნიშვნელობა ორჯერ აღემატება ნიშნის რიცხვის მნიშვნელობას. ამის დემონსტრირება ხდება ქვემოთ მოყვანილ პროგრამაში.

```
{
//      პროგრამა 2.5
//      პროგრამაში ხდება ნიშნის და უნიშნო რიცხვებთან მუშაობის დემონსტრირება
sbyte ricxvi1;           //      ricxvi1 იცვლება დიაპაზონში -128 ÷ 127
byte ricxvi2;           //      ricxvi2 იცვლება დიაპაზონში 0 ÷ 255
```

```
ricxvi1 = Convert.ToSByte(textBox1.Text);
```

```

ricxvi2 = 255;
label1.Text = ricxvi1.ToString();
label2.Text = ricxvi2.ToString();
}

```

ამ პროგრამაში `ricxvi1 = Convert.ToSByte(textBox1.Text);` სტრიქონი შეგვიძლია ასეც ჩავწეროთ: `ricxvi1 = SByte.Parse(textBox1.Text);`

მცურავწერტილიანი ტიპები

ორი ასეთი ტიპი არსებობს: `float` და `double`. ისინი გამოიყენება წილადების წარმოსადგენად. ამასთან, `double` ტიპი უფრო ხშირად გამოიყენება. ამის ერთ-ერთი მიზეზი ის არის, რომ მათემატიკური მეთოდების უმრავლესობა ამ ტიპს იყენებს. მაგალითად, ისეთ მეთოდებს, როგორცაა `Sqrt()`, `Sin()`, `Cos()` და ა.შ. აქვთ `double` ტიპის არგუმენტები და გასცემენ ამავე ტიპის შედეგს. ქვემოთ მოყვანილ პროგრამაში ხდება წილად რიცხვებთან მუშაობის დემონსტრირება.

```

{
//   პროგრამა 2.6
//   პროგრამაში ხდება წილადებთან მუშაობის დემონსტრირება
double wiladi1, wiladi2, shedegi;

```

```

wiladi1 = 5.5;
wiladi2 = double.Parse(textBox1.Text);
shedegi = wiladi1 + wiladi2;
label1.Text = shedegi.ToString();
}

```

ამ პროგრამაში `wiladi2 = double.Parse(textBox1.Text);` სტრიქონი შეგვიძლია ასეც ჩავწეროთ: `wiladi2 = Convert.ToDouble(textBox1.Text);`.

decimal ტიპი

`C++` და `Java` ენებისაგან განსხვავებით `C#` ენაში შემოტანილია `decimal` ტიპი. ამ ტიპის ცვლადები გამოიყენება ფინანსური გამოთვლების შესრულების დროს. ქვემოთ მოყვანილია პროგრამა, რომელიც ახდენს მოგებიდან პროცენტის გამოთვლას.

```

{
//   პროგრამა 2.7
//   მოგებიდან პროცენტის გამოთვლა
decimal mogeba, procenti, shedegi;

```

```

mogeba = Convert.ToDecimal(textBox1.Text);
procenti = 10.0m;
shedegi = mogeba * procenti / 100.0m;
label1.Text = shedegi.ToString();
}

```

ამ პროგრამაში `mogeba = Convert.ToDecimal(textBox1.Text);` სტრიქონი შეგვიძლია ასეც ჩავწეროთ: `mogeba = decimal.Parse(textBox1.Text);`

ცხრილი 2.1. მონაცემთა ჩვეულებრივი ტიპები

ტიპი	აღწერა	ბიტების რაოდენობა	მნიშვნელობების დიაპაზონი
bool	მნიშვნელობა true/false (ჭეშმარიტი/მცდარი)		
byte	8 ბიტიანი უნიშნო მთელი რიცხვი	8	0 ÷ 255
char	სიმბოლო	16	0 ÷ 65535
decimal	ციფრული ტიპი საფინანსო გამოთვლებისათვის	128	1E-28 ÷ 7.9E+28
double	ორმაგი სიზუსტის რიცხვი მცურავი წერტილით	64	5E-324 ÷ 1.7E+308
float	ერთმაგი სიზუსტის რიცხვი მცურავი წერტილით	32	1.5E-45 ÷ 3.4E+38
int	მთელი რიცხვი	32	-2147483648 ÷ 2147483647
long	გრძელი მთელი რიცხვი	64	-9223372036854775808 ÷ 9223372036854775807
sbyte	8 ბიტიანი ნიშნის მთელი რიცხვი	8	-128 ÷ 127
short	ნიშნის მოკლე მთელი რიცხვი	16	-32768 ÷ 32767
uint	უნიშნო მთელი რიცხვი	32	0 ÷ 4294967295
ulong	უნიშნო გრძელი მთელი რიცხვი	64	0 ÷ 18446774073709551615
ushort	უნიშნო მოკლე მთელი რიცხვი	16	0 ÷ 65535

char ტიპი (სიმბოლო)

როგორც ცნობილია დაპროგრამების ენებში სიმბოლოების წარმოსადგენად გამოიყენება 8 ბიტი. C# ენაში სიმბოლოს წარმოსადგენად გამოიყენება Unicode სიმბოლოების ნაკრები, რომლის თითოეული სიმბოლო 16 ბიტს იკავებს. ქვემოთ მოყვანილია პროგრამა, რომელშიც ხდება სიმბოლოებთან მუშაობის დემონსტრირება.

```
{
//   პროგრამა 2.8
//   პროგრამაში ხდება სიმბოლოებთან მუშაობის დემონსტრირება
char simbolo1, simbolo2, simbolo3;
```

```
simbolo1 = 'რ';
simbolo2 = textBox1.Text[0];
//   textBox1 კომპონენტში უნდა შევიტანოთ ერთი სიმბოლო
simbolo3 = Convert.ToChar(textBox2.Text);
label1.Text = simbolo1.ToString();
label2.Text = simbolo2.ToString();
label3.Text = simbolo3.ToString();
}
```

ამ პროგრამაში simbolo3 = Convert.ToChar(textBox2.Text); სტრიქონი შეგვიძლია ასეც ჩავწეროთ: simbolo3 = char.Parse(textBox2.Text);.

char ტიპის ცვლადებს შეგვიძლია, აგრეთვე, მივანიჭოთ მმართველი სიმბოლოები.

მმართველი სიმბოლო ეს არის სიმბოლო, რომელიც გამოიყენება გარკვეული მოქმედებების შესასრულებლად. ეს მოქმედებებია: ახალ სტრიქონზე გადასვლა, ჰორიზონტალური ტაბულაცია, სტრიქონის დასაწყისში გადასვლა და ა.შ. ცხრილში 2.2 მოყვანილია მმართველი სიმბოლოები.

ცხრილი 2.2. მმართველი სიმბოლოები.

მმართველი სიმბოლო	აღწერა
\'	ერთმაგი ბრჭყალი
\"	ორმაგი ბრჭყალი
\\	ირიბი დახრილი ხაზი
\0	Null (სიმბოლო, რომლის ნომერია 0)
\a	ზარი
\b	უკან დაბრუნება (BackSpace)
\f	გვერდის გადაფურცვლა
\n	ახალ სტრიქონზე გადასვლა
\r	სტრიქონის დასაწყისში გადასვლა
\t	ჰორიზონტალური ტაბულირება
\v	ვერტიკალური ტაბულირება

მაგალითი.

```
label1.Text = "საბა \n ანა";
```

ამ სტრიქონის შესრულების შედეგად label1 კომპონენტის პირველ სტრიქონში გამოჩნდება "საბა", მეორე სტრიქონში კი - "ანა".

მმართველი სიმბოლოების გამოყენების შედეგები უფრო მკაფიოდ ჩანს კონსოლურ პროგრამა-დანართებთან მუშაობის დროს. კონსოლური პროგრამის შესაქმნელად ნახ. 2.2-ზე ნაჩვენები ფანჯრის Templates ზონაში უნდა მოვნიშნოთ Console Application ელემენტი, Name: ველში შევიტანოთ პროგრამის სახელი, კატალოგი ავირჩიოთ Browse კლავიშის საშუალებით და დავაჭიროთ OK კლავიშს. გახსნილ ფანჯარაში შეგვაქვს ჩვენი პროგრამის კოდი ისე, როგორც ეს ქვემოთაა ნაჩვენები.

```
// პროგრამა 2.9
// პროგრამაში ხდება მმართველ სიმბოლოებთან მუშაობის დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
namespace Char_Console_2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("რომანი\tლიკა\tბექა");
            Console.WriteLine("ანა\nსაბა\ა");
            Console.WriteLine("რომანი\bლიკა");
            Console.WriteLine("რომანი ლიკა\rანა");
            Thread.Sleep(5000);
        }
    }
}
```

```

    }
}
}

```

პროგრამის შესრულების შედეგი:

```

romani lika   beqa
ana
saba
roman lika
anaani lika

```

პროგრამაში Thread.Sleep(5000); მეთოდი ახდენს ეკრანზე შედეგების დაყოვნებას 5 წამის განმავლობაში. როგორც პროგრამის შესრულების შედეგებიდან ჩანს, Console.WriteLine("რომანი\ლიკა\ბეკა"); მეთოდის შესრულების შედეგად შესრულდება გამოსატანი სტრიქონის ჰორიზონტალური ტაბულირება. "რომანი" სტრიქონი გამოჩნდება პირველი პოზიციიდან. შემდეგ სრულდება ჰორიზონტალური ტაბულირება. შედეგად მოხდება პირველი რვა პოზიციის გამოტოვება და მომდევნო "ლიკა" სტრიქონი გამოჩნდება მე-9 პოზიციიდან. შემდეგ ისევ შესრულდება ჰორიზონტალური ტაბულირება, ანუ გამოიტოვება მომდევნო რვა პოზიცია და "ბეკა" სტრიქონი გამოჩნდება მე-17 პოზიციიდან. ბოლოს WriteLine() მეთოდი ასრულებს მომდევნო, მეორე სტრიქონზე გადასვლას. არსებობს, აგრეთვე, Write() მეთოდი, რომელიც სტრიქონის გამოტანის შემდეგ არ ახდენს მომდევნო სტრიქონზე გადასვლას.

Console.WriteLine("ანა\nსაბა\ა"); მეთოდის შესრულების შედეგად ეკრანის მეორე სტრიქონში გამოჩნდება "ანა" სტრიქონი. შემდეგ მოხდება მომდევნო, მესამე სტრიქონზე გადასვლა, რადგან ამ სტრიქონს მოსდევს მომდევნო სტრიქონზე გადასვლის სიმბოლო. მესამე სტრიქონში გამოჩნდება "საბა" სტრიქონი და გავიგებთ კომპიუტერის დინამიკის ხმას.

Console.WriteLine("რომანი\ხლიკა"); მეთოდის შესრულების შედეგად "რომანი" სტრიქონის უკანასკნელი სიმბოლო წაიშლება და შემდეგ გამოჩნდება "ლიკა" სტრიქონი.

Console.WriteLine("რომანი ლიკა\rანა"); მეთოდის შესრულების შედეგად ჯერ გამოჩნდება "რომანი ლიკა" სტრიქონი, შემდეგ მოხდება ამავე სტრიქონის დასაწყისში გადასვლა და "ანა" სტრიქონის გამოტანა. შედეგად, მიიღება "ანაანი ლიკა" სტრიქონი.

ეკრანზე შედეგების დაყოვნება შესაძლებელია, აგრეთვე Console.ReadLine() მეთოდის გამოყენებით.

bool ტიპი

ბულის ტიპის ცვლადები იღებენ ორ მნიშვნელობას true (ჭეშმარიტი) და false (მცდარი). ქვემოთ მოყვანილია პროგრამა, რომელშიც ცვლადებს ენიჭებათ true და false მნიშვნელობები.

```

{
//   პროგრამა 2.10
//   პროგრამაში ხდება ლოგიკურ ცვლადებთან მუშაობის დემონსტრირება
bool b1, b2;

b1 = true;
b2 = Convert.ToBoolean(textBox1.Text);
label1.Text = b1.ToString();
label2.Text = b2.ToString();
}

```

ამ პროგრამაში b2 = Convert.ToBoolean(textBox1.Text); სტრიქონი შეგვიძლია ასეც ჩავწეროთ: b2 = bool.Parse(textBox1.Text);.

ToBoolean მეთოდი textBox1 კომპონენტში შეტანილ სტრიქონს გარდაქმნის ლოგიკურ

მონაცემად. ამ შემთხვევაში textBox1 კომპონენტში უნდა შევიტანოთ true ან false.

ლიტერალები

ლიტერალი (მუდმივა) არის ფიქსირებული მნიშვნელობა. მაგალითად, რიცხვი 50 არის ლიტერალი. ლიტერალი შეიძლება იყოს ნებისმიერი ტიპის მნიშვნელობა, მაგალითად, მთელი რიცხვები 5, -10, მცურავწერტილიანი რიცხვები 23.54, 1.09, სიმბოლოები 'Q', 'r', სტრიქონი "C# თანამედროვე ენაა" და ა.შ.

ლიტერალების გამოყენებისას უნდა დავიცვათ შემდეგი წესები. მთელრიცხვა ლიტერალის ტიპი შეიძლება იყოს int, uint, long ან ulong. ეს დამოკიდებულია რიცხვის სიდიდეზე. შეგვიძლია ტიპის ცხადად მითითებაც ლიტერალისთვის სუფიქსის დამატების გზით. მაგალითად, 10L (ან 10l) ნიშნავს, რომ ამ ლიტერალის ტიპია long. უნიშნო მთელრიცხვა ლიტერალისთვის (ტიპი uint) გამოიყენება u ან U სუფიქსი, მაგალითად, 20U. გრძელი უნიშნო მთელრიცხვა ლიტერალის განსაზღვრისათვის ვიყენებთ UL ან ul სუფიქსებს, მაგალითად, 40UL. float ტიპის ლიტერალს ერთვის f ან F სუფიქსი, მაგალითად, 35.41F. decimal ტიპის ლიტერალს ერთვის m ან M სუფიქსი, მაგალითად, 8.75M.

ქვემოთ მოყვანილია პროგრამა, რომელშიც ხდება ლიტერალებთან მუშაობის დემონსტრირება.

```
{
//   პროგრამა 2.11
//   პროგრამაში ხდება ლიტერალებთან მუშაობის დემონსტრირება
uint u1;
long l1;
float f1;
decimal d1;
char c1;
string s1;

u1 = 150U; label2.Text = u1.ToString();
l1 = 100L; label3.Text = l1.ToString();
f1 = 150F; label4.Text = f1.ToString();
d1 = 250M; label5.Text = d1.ToString();
c1 = 'ბ'; label6.Text = c1.ToString();
s1 = "C# თანამედროვე ენაა"; label7.Text = s1;
}
```

ცვლადები და მათი ინიციალიზება

ცვლადი არის მეხსიერების სახელდებული უბანი, რომელსაც მნიშვნელობა ენიჭება. ეს მნიშვნელობა შეიძლება შეიცვალოს პროგრამის მუშაობის პროცესში. ოპერატორს, რომლის საშუალებითაც ხდება ცვლადების გამოცხადება, შემდეგი სინტაქსი აქვს:

ტიპი ცვლადის_სახელი;

სადაც, ტიპი ცვლადის ტიპია, ცვლადის_სახელი - კი მისი სახელი. ნებისმიერი ცვლადი გამოცხადებული უნდა იყოს მის გამოყენებამდე, წინააღმდეგ შემთხვევაში ადგილი ექნება შეცდომას. ამასთან, ცვლადს უნდა მიენიჭოს მხოლოდ შესაბამისი ტიპის მნიშვნელობები. მაგალითად, bool ტიპის ცვლადს უნდა მიენიჭოს true ან false მნიშვნელობა და არა წილადი ან სხვა.

ცვლადის ინიციალიზება

ცვლადისთვის მნიშვნელობის მინიჭებას მისი გამოცხადების დროს ცვლადის ინიციალიზება ეწოდება. ცვლადების ინიციალიზების ოპერატორის სინტაქსია:

ტიპი ცვლადის_სახელი = მნიშვნელობა;

მნიშვნელობის ტიპი უნდა ემთხვეოდეს ცვლადის ტიპს. ცვლადს მნიშვნელობა უნდა მივანიჭოთ მის გამოყენებამდე. ეს შეიძლება გავაკეთოთ ან ცვლადის გამოცხადებისას ან გამოცხადების შემდეგ მინიჭების ოპერატორის გამოყენებით. მოყვანილი პროგრამა ახდენს ცვლადების ინიციალიზების დემონსტრირებას.

```
{
//   პროგრამა 2.12
//   პროგრამაში ხდება ცვლადების ინიციალიზების დემონსტრირება
int ricxvi1 = 25, ricxvi2 = 30;
char simbolo = 'რ';
double წილადი = 150.75;
label1.Text = ricxvi1.ToString() + " " + წილადი.ToString() + " " + simbolo;
}
```

ცვლადების დინამიური ინიციალიზება

ცვლადების ინიციალიზება შესაძლებელია, აგრეთვე, დინამიურად, ე.ი. პროგრამის შესრულების დროს. ამ შემთხვევაში, ცვლადის ინიციალიზება სრულდება არა პროგრამის დასაწყისში, არამედ მის ნებისმიერ ადგილში. მოყვანილი პროგრამა ახდენს ამის დემონსტრირებას.

```
{
//   პროგრამა 2.13
//   პროგრამაში ხდება ცვლადის დინამიური ინიციალიზების დემონსტრირება
int simagle = 5, sigane, sigrdze;

sigane = int.Parse(textBox1.Text);
sigrdze = int.Parse(textBox2.Text);
//   moculoba ცვლადის ინიციალიზება ხდება დინამიურად
int moculoba = simagle * sigane * sigrdze;
label1.Text = moculoba.ToString();
}
```

ოპერატორები

ოპერატორი არის სიმბოლო, რომელიც კომპილატორს ატყობინებს თუ რომელი სპეციფიკური ოპერაცია უნდა შესრულდეს. C# ენაში არსებობს ოპერატორების ოთხი ძირითადი კლასი: არითმეტიკის, ლოგიკის, შედარებისა და ბიტობრივი.

არითმეტიკის ოპერატორები

არითმეტიკის ოპერატორები შეგვიძლია გამოვიყენოთ როგორც მთელი, ისე წილადი რიცხვების მიმართ. ისინი მოყვანილია ცხრილში 2.3.

მოვიყვანოთ ზოგიერთი განმარტება. როცა გაყოფის ოპერატორს ვიყენებთ მთელი რიცხვების მიმართ, მაშინ შედეგი იქნება მთელი რიცხვი, ნაშთი კი უარიყოფა. მაგალითად, 20/6 გაყოფის შედეგი იქნება მთელი რიცხვი 3. ნაშთის მისაღებად უნდა გამოვიყენოთ % ოპერატორი. მაგალითად, 20%6 ოპერაციის შედეგი იქნება 2. % ოპერატორის გამოყენებით შეგვიძლია დავადგინოთ რიცხვი კენტია თუ ლუწი. მაგალითად, R%2 გამოსახულების გამოთვლისას თუ ნაშთი 1-ის ტოლია, მაშინ R რიცხვი კენტია, თუ ნაშთი 0-ის ტოლია, მაშინ R რიცხვი ლუწია. ანალოგიურად შეგვიძლია დავადგინოთ ერთი რიცხვი მეორის ჯერადია თუ არა.

ცხრილი 2.3. არითმეტიკის ოპერატორები

ოპერატორი	მნიშვნელობა
+	შეკრება
-	გამოკლება (და უნარული მინუსი)
*	გამრავლება
/	გაყოფა
%	მოდულით გაყოფა (ნაშთის გამოყოფა)
++	ინკრემენტი
--	დეკრემენტი

მოყვანილ პროგრამებში ხდება არითმეტიკის ოპერატორების გამოყენების დემონსტრირება მთელი რიცხვებისათვის.

```
{
//   პროგრამა 2.14
//   პროგრამაში ხდება არითმეტიკის ოპერაციების მუშაობის დემონსტრირება
//   მთელი რიცხვებისათვის
int ricxvi1, ricxvi2, jami, sxvaoba, namravli, ganayofi, nashti;
```

```
ricxvi1 = int.Parse(textBox1.Text);
ricxvi2 = int.Parse(textBox2.Text);
jami    = ricxvi1 + ricxvi2;
sxvaoba = ricxvi1 - ricxvi2;
namravli = ricxvi1 * ricxvi2;
ganayofi = ricxvi1 / ricxvi2;
nashti   = ricxvi1 % ricxvi2;
label1.Text = jami.ToString();
label2.Text = sxvaoba.ToString();
label3.Text = namravli.ToString();
label4.Text = ganayofi.ToString();
label5.Text = nashti.ToString();
}
```

ქვემოთ მოყვანილ პროგრამაში ხდება არითმეტიკის ოპერატორების მუშაობის დემონსტრირება წილადი რიცხვებისათვის.

```
{
//   პროგრამა 2.15
//   პროგრამაში ხდება არითმეტიკის ოპერატორების მუშაობის დემონსტრირება
//   წილადი რიცხვებისათვის
double wiladi1, wiladi2, jami, sxvaoba, namravli, ganayofi, nashti;
```



```

wiladi1 = double.Parse(textBox1.Text);
wiladi2 = double.Parse(textBox2.Text);
jami     = wiladi1 + wiladi2;
sxvaoba  = wiladi1 - wiladi2;
namravli = wiladi1 * wiladi2;
ganayofi = wiladi1 / wiladi2;
nashti   = wiladi1 % wiladi2;
label1.Text = jami.ToString();
label2.Text = sxvaoba.ToString();
label3.Text = namravli.ToString();
label4.Text = ganayofi.ToString();
label5.Text = nashti.ToString();
}

```

ინკრემენტისა და დეკრემენტის ოპერატორები

ინკრემენტის ოპერატორი (++) ერთით ზრდის თავისი ოპერანდის (არგუმენტის) მნიშვნელობას. ოპერანდი არის ცვლადი. დეკრემენტის ოპერატორი (--) ერთით ამცირებს თავისი ოპერანდის მნიშვნელობას. მაგალითად, ოპერატორი

```
ricxvi = ricxvi + 1;
```

ასრულებს ისეთივე მოქმედებებს, როგორსაც ოპერატორი

```
ricxvi++; ან ++ricxvi;
```

ანალოგიურად, ოპერატორი

```
ricxvi = ricxvi - 1;
```

ასრულებს იგივე მოქმედებებს, რასაც ოპერატორი

```
ricxvi--; ან --ricxvi;
```

როგორც ვხედავთ ეს ოპერატორები შეიძლება მიეთითოს როგორც ოპერანდამდე (პრეფიქსი), ისე ოპერანდის შემდეგ (პოსტფიქსი). ახლა ვნახოთ რაში მდგომარეობს მათ შორის განსხვავება. თუ ინკრემენტის ან დეკრემენტის ოპერატორი მითითებულია ოპერანდის წინ, მაშინ ჯერ მოხდება ოპერანდის მნიშვნელობის გაზრდა ან შემცირება, ხოლო შემდეგ მისი გამოყენება გამოსახულებაში. თუ ინკრემენტის ან დეკრემენტის ოპერატორი მითითებულია ოპერანდის შემდეგ, მაშინ ჯერ მოხდება ამ ოპერანდის გამოყენება გამოსახულებაში და შემდეგ მისი მნიშვნელობის გაზრდა ან შემცირება.

ქვემოთ მოყვანილ პროგრამაში ხდება ინკრემენტის ოპერატორის გამოყენების დემონსტრირება.

```
{
```

```
// პროგრამა 2.16
```

```
// პროგრამაში ხდება ++ ოპერატორის მუშაობის დემონსტრირება
```

```
int ricxvi, shedegi;
```

```
ricxvi = 5;
```

```
shedegi = ++ricxvi;
```

```
// ricxvi = 6, shedegi = 6
```

```
label1.Text = ricxvi.ToString();
```

```
label2.Text = shedegi.ToString();
```

```
ricxvi = 5;
```

```
shedegi = ricxvi++;
```

```
// shedegi = 5, ricxvi = 6
```

```
label3.Text = ricxvi.ToString();
```

```
label4.Text = shedegi.ToString();
```

```
}
```

თავდაპირველად ricxvi ცვლადს ენიჭება მნიშვნელობა 5. მომდევნო სტრიქონში ამ ცვლადის მარცხნივ მოთავსებულია ++, ამიტომ მისი მნიშვნელობა ჯერ გაიზრდება ერთით და გახდება 6-ის ტოლი, შემდეგ კი მიენიჭება shedegi ცვლადს. შემდეგ, ricxvi ცვლადს კვლავ ენიჭება მნიშვნელობა 5. მომდევნო სტრიქონში ++ მოთავსებულია ricxvi ცვლადის მარჯვნივ. ამიტომ, მისი მნიშვნელობა ჯერ მიენიჭება shedegi ცვლადს, შემდეგ კი გაიზრდება ერთით და გახდება 6-ის ტოლი.

ქვემოთ მოყვანილ პროგრამაში ხდება დეკრემენტის ოპერატორის გამოყენების დემონსტრირება.

```
{
//   პროგრამა 2.17
//   პროგრამაში ხდება -- ოპერატორის მუშაობის დემონსტრირება
int ricxvi, shedegi;

ricxvi = 5;
shedegi = --ricxvi;           //   ricxvi  = 4, shedegi = 4
label1.Text = ricxvi.ToString();
label2.Text = shedegi.ToString();
ricxvi = 5;
shedegi = ricxvi--;         //   shedegi = 5, ricxvi  = 4
label3.Text = ricxvi.ToString();
label4.Text = shedegi.ToString();
}
```

შედარებისა და ლოგიკის ოპერატორები

შედარების ოპერატორები ადარებენ ორ მნიშვნელობას და გასცემენ bool ტიპის მნიშვნელობას true ან false. ისინი მოყვანილია ცხრილში 2.4. ლოგიკის ოპერატორები ასრულებენ ლოგიკურ ოპერაციებს bool ტიპის მნიშვნელობებზე. ისინი მოყვანილია ცხრილში 2.5.

C# ენაში == და != ოპერატორები შეიძლება გამოვიყენოთ ყველა ობიექტის მიმართ მათი შედარებისათვის ტოლობაზე ან უტოლობაზე. შედარების დანარჩენი ოპერატორები შეგვიძლია გამოვიყენოთ მხოლოდ მონაცემების ჩამოთვლადი ტიპების მიმართ, რომლებიც მოწესრიგებულია თავიანთ სტრუქტურაში. ასეთია მაგალითად, რიცხვების მოწესრიგებული სტრუქტურა 1, 2, 3 და ა.შ., ან ანბანის მიხედვით დალაგებული სიმბოლოები. შედეგად, შედარების ყველა ოპერატორი შეგვიძლია გამოვიყენოთ მონაცემთა ყველა რიცხვითი ტიპის მიმართ. bool ტიპის მნიშვნელობების შედარება შეიძლება მხოლოდ ტოლობაზე ან უტოლობაზე.

ცხრილში 2.6 მოყვანილია ლოგიკის ოპერატორების ჭეშმარიტების ცხრილი. როგორც ცხრილიდან ჩანს & ოპერატორი გასცემს true მნიშვნელობას მხოლოდ მაშინ, როცა მისი ორივე ოპერანდის მნიშვნელობაა true. წინააღმდეგ შემთხვევაში, ის გასცემს false მნიშვნელობას. | ოპერატორი გასცემს true მნიშვნელობას, თუ მისი ერთ-ერთი ოპერანდის მნიშვნელობაა true. წინააღმდეგ შემთხვევაში, ის გასცემს false მნიშვნელობას. ! ოპერატორი გასცემს თავისი ოპერანდის ლოგიკურად საწინააღმდეგო მნიშვნელობას. ^ ოპერატორი გასცემს false მნიშვნელობას თუ მის ორივე ოპერანდს ერთნაირი მნიშვნელობა აქვს, წინააღმდეგ შემთხვევაში გასცემს true მნიშვნელობას.

ცხრილი 2.4. შედარების ოპერატორები

ოპერატორი	მნიშვნელობა
==	ტოლია

!=	არ არის ტოლი
>	მეტია
<	ნაკლებია
>=	მეტია ან ტოლი
<=	ნაკლებია ან ტოლი

ცხრილი 2.5. ლოგიკის ოპერატორები

ოპერატორი	მნიშვნელობა
&	AND (და)
	OR (ან)
^	XOR (გამომრიცხავი ან)
&&	Short-circuit AND (სწრაფი და ოპერატორი)
	Short-circuit OR (სწრაფი ან ოპერატორი)
!	NOT (არა)

ცხრილი 2.6. ლოგიკის ოპერატორების ჭეშმარიტების ცხრილი

B1	B2	B1 && B2	B1 B2	B1 ^ B2	!B1
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

მოყვანილ პროგრამაში ხდება ლოგიკის ოპერატორების მუშაობის დემონსტრირება.

```

{
//   პროგრამა 2.18
//   პროგრამაში ხდება ლოგიკის ოპერატორების მუშაობის დემონსტრირება
bool b1, b2, b3, b4, b5, b6;

b1 = true; b2 = false;
b3 = b1 && b2;           //   b3 = false
b4 = b1 || b2;          //   b4 = true
b5 = b1 ^ b2;           //   b5 = true
b6 = !b1;               //   b6 = false
label1.Text = b3.ToString();
label2.Text = b4.ToString();
label3.Text = b5.ToString();
label4.Text = b6.ToString();
}

```

მოყვანილი პროგრამით ხდება შედარების ოპერატორების მუშაობის დემონსტრირება.

```

{
//   პროგრამა 2.19
//   პროგრამაში ხდება შედარების ოპერატორების მუშაობის დემონსტრირება
int ricxvi1 = 5, ricxvi2 = 10;
bool shedegi;

shedegi = ricxvi1 > 0;           //   shedegi = true

```

```

label1.Text = shedegi.ToString();
shedegi = ( ricxvi1 < 2 ) && ( ricxvi2 > 7 );           //      shedegi = false
label2.Text = shedegi.ToString();
shedegi = ( ricxvi1 == 3 ) || ( ricxvi2 != 5 );        //      shedegi = true
label3.Text = shedegi.ToString();
}

```

არსებობს, აგრეთვე, ლოგიკის სწრაფი ოპერატორები && და ||. მათი გამოყენებისას პროგრამა უფრო სწრაფად სრულდება. თუ && ოპერატორის შესრულებისას პირველმა ოპერანდმა მიიღო false მნიშვნელობა, მაშინ შედეგი იქნება false მიუხედავად მეორე ოპერანდის მნიშვნელობისა. თუ || ოპერატორის შესრულებისას პირველმა ოპერანდმა მიიღო true მნიშვნელობა, მაშინ შედეგი იქნება true მიუხედავად მეორე ოპერატორის მნიშვნელობისა. ასეთ შემთხვევებში აღარ არის საჭირო მეორე ოპერანდის შეფასება, რადგან მისი მნიშვნელობა საბოლოო შედეგს ვერ შეცვლის. პროგრამა უფრო სწრაფად სრულდება.

მინიჭების ოპერატორი

მისი სინტაქსია:

ცვლადი = გამოსახულება;

ცვლადს და გამოსახულებას ერთნაირი ტიპი უნდა ჰქონდეს. მინიჭების ოპერატორი ცვლადს ანიჭებს გამოსახულების მნიშვნელობას.

მინიჭების ოპერატორი საშუალებას იძლევა, აგრეთვე, შევქმნათ მინიჭებების მიმდევრობა, მაგალითად,

```

int ricxvi1, ricxvi2, ricxvi3;
ricxvi1 = ricxvi2 = ricxvi3 = 50;

```

აქ სამივე ცვლადს ენიჭება რიცხვი 50. ასეთი მინიჭება საშუალებას გვაძლევს რამდენიმე ცვლადს ერთდროულად მივანიჭოთ ერთი მნიშვნელობა.

მინიჭების შედგენილი ოპერატორი

მინიჭების შედგენილ (შემოკლებულ) ოპერატორში არითმეტიკის ოპერატორები მოთავსებულია მინიჭების ოპერატორის მარცხნივ. მისი სინტაქსია:

ცვლადი ოპერატორი = გამოსახულება;

სადაც, **ოპერატორი** არის არითმეტიკის ან ლოგიკის ოპერატორი. მაგალითად, გამოსახულება

```
jami = jami + 25;
```

შედგენილი ოპერატორის გამოყენებით შეგვიძლია ასე ჩავწეროთ

```
jami += 25;
```

+= ოპერატორების წყვილი მიუთითებს, რომ jami ცვლადს უნდა მიენიჭოს მნიშვნელობა jami + 25.

შედგენილი (შემოკლებული) ოპერატორებია:

```
+=      -=      *=      /=      %=      &=      |=      ^=
```

მინიჭების შედგენილი ოპერატორები მინიჭების ჩვეულებრივ ოპერატორთან შედარებით უფრო კომპაქტურია. ეს განსაკუთრებით იგრძნობა მაშინ, როცა გრძელ სახელებს ვიყენებთ. ამ შემთხვევაში, სახელის შეტანა ერთხელ გვიწევს. მეორეც, მათი გამოყენება აჩქარებს კოდის კომპილირებას, რადგან ოპერანდის შეფასება მხოლოდ ერთხელ ხდება.

ოპერატორების პრიორიტეტები

ცხრილში 2.7 ნაჩვენებია C# ენის ყველა ოპერატორის პრიორიტეტი. ოპერატორები დალაგებულია პრიორიტეტების კლების მიხედვით. ცხრილის ერთ სტრიქონში ოპერატორებს თანაბარი პრიორიტეტები აქვთ.

როგორც ცხრილიდან ჩანს, *, / და % ოპერატორებს უფრო მაღალი პრიორიტეტი აქვთ, ვიდრე + და - . განვიხილოთ გამოსახულება:

$$y = x1 + x2 * x3 - x4 / x5 ;$$

ის შემდეგნაირად გამოითვლება. გამოსახულებაში პირველია + ოპერატორი. სანამ ეს ოპერატორი შესრულდება კომპილატორი ამოწმებს მის მარჯვნივ რომელი ოპერატორია მოთავსებული. ესაა *, რომელსაც უფრო მაღალი პრიორიტეტი აქვს, ვიდრე + ოპერატორს. შემდეგ, კომპილატორი ამოწმებს * ოპერატორის მარჯვნივ რომელი ოპერატორია მოთავსებული. ესაა -, რომელსაც უფრო დაბალი პრიორიტეტი აქვს, ვიდრე * ოპერატორს. ამიტომ პირველ რიგში შესრულდება * ოპერატორი. რადგან + და - ოპერატორებს თანაბარი პრიორიტეტი აქვთ და + რიგით პირველია გამოსახულებაში, ამიტომ შემდეგ შესრულდება + ოპერატორი. სანამ კომპილატორი შეასრულებს - ოპერატორს, ის ამოწმებს მის მარჯვნივ რომელი ოპერატორია მოთავსებული. ესაა / ოპერატორი, რომელსაც უფრო მაღალი პრიორიტეტი აქვს ვიდრე - ოპერატორს. ამიტომ, ჯერ შესრულდება / ოპერატორი, შემდეგ კი - ოპერატორი.

ახლა განვიხილოთ მეორე გამოსახულება:

$$y = (x1 + x2) * x3 - (x4 + x5) / x6 ;$$

აქ ჯერ შესრულდება პირველ მრგვალ ფრჩხილებში მოთავსებული გამოსახულება. შემდეგ, * ოპერატორი. შემდეგ, მეორე მრგვალ ფრჩხილებში მოთავსებული გამოსახულება. შემდეგ, / ოპერატორი და ბოლოს, - ოპერატორი.

როგორც ვხედავთ, პრიორიტეტების ცოდნა საჭიროა იმისათვის, რომ სწორად ჩავწეროთ გამოსახულება და შესაბამისად, მივიღოთ სწორი შედეგი.

გამოსახულებები. მათემატიკის მეთოდები

გამოსახულება არის ოპერატორების, ცვლადებისა და მეთოდების კომბინაცია.

მაგალითად,

$$y = x + z - 5;$$

$$y = x - \text{Math.Sin}(x) + 10.97;$$

პროგრამების კითხვადობის გაუმჯობესების მიზნით გამოსახულებებში ტაბულირების სიმბოლოები და ინტერვალები გამოიყენება. მაგალითად, მოყვანილი ორი გამოსახულებიდან მეორე მათგანის წაკითხვა უფრო ადვილია ვიდრე პირველის:

$$y = x / 5 + z * (w - 2);$$

$$y = x / 5 + z * (w - 2);$$

მრგვალი ფრჩხილები ზრდიან მასში მოთავსებული ოპერანდების პრიორიტეტს. ისინი ისევე გამოიყენება, როგორც ალგებრაში. მრგვალი ფრჩხილები გამოიყენება, აგრეთვე, პროგრამის კოტხვადობის გასაუმჯობესებლად. მაგალითად, მოყვანილი ორი სტრიქონიდან მეორე უკეთესად იკითხება:

$$y = x * 5 + z / 7.2 - w;$$

$$y = (x * 5) + (z / 7.2) - w;$$

მათემატიკის მეთოდები აღწერილია System.Math სტანდარტულ კლასში (ცხრილი 2.8). მათი უმრავლესობა double ტიპს იყენებს. მათ გამოსაძახებლად ჯერ ეთითება კლასის სახელი Math, შემდეგ წერტილი და მეთოდის სახელი.

მოკლედ განვიხილოთ ზოგიერთი მეთოდი. Pow(x, y) მეთოდს x რიცხვი აჰყავს y ხარისხში. მაგალითად, Pow(5, 2) მეთოდის შესრულების შედეგად გაიცემა 25. Exp(x) მეთოდს e რიცხვი აჰყავს x ხარისხში. მაგალითად, Exp(1) მეთოდის შესრულების შედეგად გაიცემა 2.71 . Max(x, y) მეთოდი გასცემს x და y რიცხვებს შორის უდიდესს, Min(x, y) მეთოდი კი - უმცირესს. Floor(x) მეთოდი x რიცხვს ამრგვალებს ნაკლებობით. მაგალითად, Floor(5.998) მეთოდის შესრულების შედეგად გაიცემა 5. Ceiling(x) მეთოდი x რიცხვს ამრგვალებს მეტობით. მაგალითად, Ceiling(5.123) მეთოდის შესრულების შედეგად გაიცემა 6. Round(x, y) მეთოდი გამოიყენება x რიცხვის ფორმატირებული გამოტანისათვის. y მიუთითებს ათწილადში თანრიგების რაოდენობას. მაგალითად, Round(5.1234, 2) მეთოდის შესრულების შედეგად გაიცემა ათწილადი 5.12 . Sign(x) მეთოდი გასცემს x რიცხვის ნიშანს. მაგალითად, Sign(-5) მეთოდის შესრულების შედეგად გაიცემა -1, ხოლო Sign(5) მეთოდის შესრულების შედეგად კი გაიცემა 1.

შევადგინოთ პროგრამა, რომელიც ახდენს კვადრატული ფესვის ამოღებას რიცხვიდან, რომელიც textBox1 კომპონენტში შეგვაქვს.

```
{
//   პროგრამა 2.20
//   კვადრატული ფესვის ამოღება
double ricxvi, shedegi;

ricxvi = double.Parse(textBox1.Text);
shedegi = Math.Sqrt(ricxvi);
label1.Text = shedegi.ToString();
label2.Text = Math.Round(shedegi,2).ToString();
}
```

შევადგინოთ კიდევ ერთი პროგრამა, რომელიც გამოთვლის მოცემული გამოსახულების მნიშვნელობას:

$$y = \frac{\sqrt{x^5 + \sin x}}{1 - e^x}$$

```
{
//   პროგრამა 2.21
//   ფორმულის გამოთვლა
double ricxvi, shedegi;

ricxvi = double.Parse(textBox1.Text);
shedegi = Math.Sqrt( Math.Pow(ricxvi,5) + Math.Sin(ricxvi) ) /
              ( 1 - Math.Exp(ricxvi) );
label1.Text = Math.Round(shedegi,2).ToString();
}
```

დანართში 1 მოყვანილია ამოცანები თავების მიხედვით.

ცხრილი 2.7. ოპერატორების პრიორიტეტები

კატეგორია	ოპერატორები
პირველადი	დაჯგუფება: (x)
	ველთან მიმართვა: x.y
	მეთოდის გამოძახება: f(x)
	ინდექსირებული მიმართვა: a[x]
	პოსტფიქსური ინკრემენტი: x++
	პოსტფიქსური დეკრემენტი: x--
	ახალი ობიექტის შექმნა: new
	ტიპის მიღება: typeof
	ზომის მიღება: sizeof
	გადავსების შემოწმება: checked
გადავსების არშემოწმება: unchecked	
უნარული	დადებითი მნიშვნელობა: +
	უარყოფითი მნიშვნელობა: -
	ლოგიკური "არა": !
	ბიტობრივი "არა": ~
	პრეფიქსური ინკრემენტი: ++x
	პრეფიქსური დეკრემენტი: --x
	ტიპის დაყვანა: (type)
	გამრავლება: *
მულტიპლიკატიური	გაყოფა: /
	ნაშთის გამოყოფა: %
	აღიტიური
აღიტიური	შეკრება: +
	გამოკლება: -
ბიტობრივი ძვრა	ძვრა მარცხნივ: <<
	ძვრა მარჯვნივ: >>
მიმართებები	ნაკლებია: <
	მეტია: >
	ნაკლებია ან ტოლი: <=
	მეტია ან ტოლი: >=
	ტიპების თავსებადობა: is
	ტიპების თავსებადობა გარდაქმნით: as
ტოლობა	ტოლობა: =
	უტოლობა: !=
ბიტობრივი "და"	&
ბიტობრივი გამომრიცხავი "ან"	^
ბიტობრივი "ან"	
ლოგიკური "და"	&&
ლოგიკური "ან"	
ტერნარული ოპერატორი	?:
მინიჭებები	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=

ცხრილი 2.8. მათემატიკის მეთოდები

მეთოდი	დანიშნულება
Abs(x)	აბსოლუტური მნიშვნელობა
Sin(x)	სინუსი
Cos(x)	კოსინუსი
Tan(x)	ტანგენსი
Sqrt(x)	კვადრატული ფესვი
Exp(x)	ექსპონენცია
Log(x)	ლოგარითმი
Log10(x)	ათობითი ლოგარითმი
Max(x, y)	ორ რიცხვს შორის უდიდესი
Min(x, y)	ორ რიცხვს შორის უმცირესი
Pow(x, y)	ახარისხება
Floor(x)	დამრგვალება ნაკლებობით
Ceiling(x)	დამრგვალება მეტობით
Round(x, y)	ფორმატირებული გამოტანა
E	2,71
PI	3,14
Sign(x)	არგუმენტის ნიშანი

ტიპების გარდაქმნა

დაპროგრამების დროს, როგორც წესი, ერთი ტიპის მქონე ცვლადს შეგვიძლია მივანიჭოთ მხოლოდ ამავე ტიპის მნიშვნელობა, ე.ი. მინიჭების ოპერატორის მარცხნივ მოთავსებულ ცვლადსა და მარჯვნივ მოთავსებულ გამოსახულებას ერთნაირი ტიპები უნდა ჰქონდეს. მაგრამ, რიგ შემთხვევებში, საჭირო ხდება ერთი ტიპის ცვლადისათვის სხვა ტიპის მნიშვნელობის მინიჭება. მაგალითად, double ტიპის ცვლადს შეგვიძლია მივანიჭოთ int ტიპის ცვლადის მნიშვნელობა:

```
int mteli;
double wiladi;
mteli = 25;
wiladi = mteli;
```

თუ მინიჭების ოპერატორში გამოიყენება მონაცემთა თავსებადი ტიპები, მაშინ მინიჭების ოპერატორის მარჯვნივ მოთავსებული გამოსახულების ტიპი ავტომატურად გარდაიქმნება მარცხნივ მოთავსებული ცვლადის ტიპად. აქედან გამომდინარე, ჩვენს მაგალითში mteli ცვლადის ტიპი ჯერ გარდაიქმნება double ტიპად და შემდგომ მისი მნიშვნელობა მიენიჭება wiladi ცვლადს. რადგან C# ენაში ტიპების კონტროლი მკაცრად ხორციელდება, ამიტომ ყველა ტიპის გარდაქმნა არ არის დაშვებული. მაგალითად, არათავსებადია int და bool ტიპები.

ერთი ტიპის მქონე ცვლადისთვის სხვა ტიპის მქონე ცვლადის მნიშვნელობის მინიჭებისას ტიპის ავტომატურად გარდაქმნა შესრულდება იმ შემთხვევაში თუ:

- ორივე ტიპი თავსებადია;
- საბოლოო ტიპი (მარცხენა) მეტია საწყის ტიპზე (მარჯვენა). ეს იმას ნიშნავს, რომ საბოლოო ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობა მეტია საწყისი ტიპის მქონე ცვლადის მიერ დაკავებული თანრიგების რაოდენობაზე.

ამ პირობების შესრულებისას ხორციელდება გაფართოებადი გარდაქმნა. მაგალითად, int

ტიპისთვის გამოყოფილი ბიტების (თანრიგების) რაოდენობა, ყოველთვის საკმარისია byte ტიპის მქონე მნიშვნელობების შესანახად, და რადგან ორივე მთელრიცხვა ტიპია, ამიტომ მათ მიმართ შესრულდება ტიპების ავტომატური გარდაქმნა.

გაფართოებადი გარდაქმნის შესასრულებლად ყველა რიცხვითი ტიპი მთელრიცხვა და მცურავწერტილიანი ტიპების ჩათვლით თავსებადი უნდა იყოს. მაგალითად, ქვემოთ მოყვანილ პროგრამაში სრულდება long ტიპის ცვლადის გარდაქმნა double ტიპის ცვლადად.

```
{
//   პროგრამა 2.22
//   პროგრამაში ხდება long ტიპის გარდაქმნა double ტიპად
long L1;
double D1;

L1 = 100321410L;
D1 = L1;
label1.Text = D1.ToString();
}
```

გარდაქმნა double ტიპიდან long ტიპად დაუშვებელია, რადგან ის არ არის გაფართოებადი. გარდა ამისა, ტიპების ავტომატური გარდაქმნა არ სრულდება decimal და float ცვლადებს შორის, აგრეთვე, decimal და double, char და bool, მთელი ტიპის ცვლადებსა და char, მთელი ტიპის ცვლადებსა და bool ტიპის ცვლადებს შორის.

ტიპების დაყვანის ოპერატორი (cast operator)

ტიპის დაყვანა - ესაა ერთი ტიპის მეორე ტიპად გარდაქმნის ოპერაცია. ის გამოიყენება მონაცემების შეუთავსები ტიპების მიმართ. ტიპების ასეთ გარდაქმნას ცხადი (აშკარა) ეწოდება. ტიპების დაყვანის ოპერატორის სინტაქსია:

(საბოლოო_ტიპი) გამოსახულება;

აქ საბოლოო_ტიპი მიუთითებს თუ რომელ ტიპად უნდა გარდაიქმნას მითითებული გამოსახულების შედეგი. მოყვანილი პროგრამით სრულდება ტიპების გარდაქმნის დემონსტრირება.

```
{
//   პროგრამა 2.23
//   პროგრამაში ხდება ტიპების გარდაქმნა შეუთავსებ ტიპებს შორის
double wiladi1, wiladi2;
byte baiti;
int mteli;
char simbolo;

wiladi1 = double.Parse(textBox1.Text);
wiladi2 = double.Parse(textBox2.Text);
//   double ტიპის გამოსახულება გარდაიქმნება int ტიპად
mteli = ( int ) ( wiladi1 / wiladi2 );           //   აქ ხდება ინფორმაციის დაკარგვა
label1.Text = mteli.ToString();
//   აქ ინფორმაცია არ იკარგება, რადგან 50 იმყოფება byte ტიპის დასაშვებ დიაპაზონში
mteli = 50;
baiti = ( byte ) mteli;
label2.Text = baiti.ToString();
//   აქ ინფორმაცია იკარგება, რადგან 300 სცილდება byte ტიპის დასაშვებ დიაპაზონს
```

```

mteli = 300;
baiti = ( byte ) mteli;
label3.Text = baiti.ToString();

baiti = byte.Parse(textBox3.Text);
simbolo = ( char ) baiti;           // ტიპების დაყვანის ოპერაცია შეუთავსებ ტიპებს შორის
label4.Text = baiti.ToString();
}

```

mteli = (int) (wiladi1 / wiladi2); სტრიქონში double ტიპის გამოსახულების შედეგი დაიყვანება int ტიპზე. მრგვალი ფრჩხილების მითითება, რომლებშიც მოთავსებულია wiladi1/wiladi2, აუცილებელია. წინააღმდეგ შემთხვევაში, int ტიპზე დაიყვანება მხოლოდ wiladi1 ცვლადი. ამ შემთხვევაში სრულდება ტიპების დაყვანის ოპერაცია, რადგან double ტიპის მნიშვნელობა int ტიპის მნიშვნელობად ავტომატურად არ გარდაიქმნება.

როცა ტიპების გარდაქმნის დროს სრულდება კუმშვადი გარდაქმნა, მაშინ ინფორმაცია შეიძლება დაიკარგოს. მაგალითად, თუ long ტიპის დაყვანისას int ტიპზე, long ტიპის მქონე ცვლადის მნიშვნელობა არ ეტევა int ტიპის დიაპაზონის ფარგლებში, მაშინ უფროსი თანრიგის ბიტები დაიკარგება, შედეგად, იკარგება ინფორმაციაც. როცა ხდება მცურავწერტილიანი მნიშვნელობის დაყვანა მთელ მნიშვნელობაზე, მაშინ წილადი ნაწილი იკარგება. მაგალითად, თუ მნიშვნელობა 7.85 გარდაიქმნება მთელ მნიშვნელობად, მაშინ შედეგი იქნება 7, ხოლო წილადი 0.85 იკარგება.

ზემოთ მოყვანილ სტრიქონში (wiladi1 / wiladi2) გამოსახულების შედეგი დაიყვანება int ტიპზე, რაც იწვევს წილადი ნაწილის დაკარგვას. შემდეგ, როცა baiti ცვლადს ენიჭება მნიშვნელობა 50, ინფორმაცია არ იკარგება, რადგან ის მოთავსებულია byte ტიპის დასაშვები მნიშვნელობების დიაპაზონში. მაგრამ, თუ baiti ცვლადს მივანიჭებთ მნიშვნელობას 300 (მისი ორობითი წარმოდგენაა 00000001 00101100), მაშინ ადგილი ექნება ინფორმაციის დაკარგვას, რადგან 300 აღემატება byte ტიპის მაქსიმალურ მნიშვნელობას. ამ შემთხვევაში baiti ცვლადს მიენიჭება 44 (მისი ორობითი წარმოდგენაა 00101100), რადგან დარჩება 300-ის უმცროსი ბაიტი, უფროსი კი ჩამოიჭრება. დაბოლოს, byte ტიპის გარდაქმნისას char ტიპად ინფორმაცია არ იკარგება.

ტიპების გარდაქმნა გამოსახულებებში

გამოსახულების გამოთვლის დროს შესაძლებელია ტიპების გარდაქმნა იმ პირობით, რომ ისინი თავსებადია. თუ გამოსახულებაში ხდება არათავსებადი ტიპის მონაცემების გამოყენება, მაშინ ისინი გარდაიქმნება ერთსა და იმავე ტიპად ბიჯობრივი გარდაქმნის ალგორითმის გამოყენებით.

გარდაქმნა სრულდება გამოსახულებებში ტიპების ავტომატური გარდაქმნის წესების მიხედვით. განვიხილოთ ეს ალგორითმი არითმეტიკის ოპერაციებისთვის:

- თუ ერთ ოპერანდს აქვს decimal ტიპი, მაშინ მეორე ოპერანდი ავტომატურად გარდაიქმნება ამ ტიპად. გამონაკლისია შემთხვევა, როცა მეორე ოპერანდს აქვს float ან double ტიპი. ამ შემთხვევაში ადგილი ექნება შეცდომას.
- თუ ერთ ოპერანდს აქვს double ტიპი, მეორე ოპერანდიც ავტომატურად გარდაიქმნება ამ ტიპად.
- თუ ერთ ოპერანდს აქვს float ტიპი, მაშინ მეორე ოპერანდიც ავტომატურად გარდაიქმნება ამ ტიპად.
- თუ ერთ ოპერანდს აქვს ulong ტიპი, მაშინ მეორე ოპერანდიც ავტომატურად გარდაიქმნება ამ ტიპად. გამონაკლისია შემთხვევა, როცა ოპერანდს აქვს sbyte, short, int ან long ტიპი. ამ

შემთხვევაში ადგილი ექნება შეცდომას.

– თუ ერთ ოპერანდს აქვს long ტიპი, მაშინ მეორე ოპერანდი ავტომატურად გარდაიქმნება ამ ტიპად.

– თუ ერთ ოპერანდს აქვს uint ტიპი, ხოლო მეორეს sbyte, short ან int, მაშინ ორივე ოპერანდი ავტომატურად გარდაიქმნება long ტიპად.

– თუ არც ერთი ზემოთ მოყვანილი წესი არ იყო გამოყენებული, მაშინ ორივე ოპერანდი int ტიპად გარდაიქმნება.

უკანასკნელი წესის თანახმად გამოსახულებაში char, sbyte, byte, ushort და short ტიპები გარდაიქმნება int ტიპად. გარდაქმნის ასეთ პროცედურას მთელრიცხვა ტიპად ავტომატური გარდაქმნა ეწოდება. ეს, აგრეთვე, იმას ნიშნავს, რომ ყველა მათემატიკური ოპერაციის შედეგს ექნება ტიპი, რომელსაც მნიშვნელობის შესანახად გამოყოფილი აქვს თანრიგების არანაკლები რაოდენობა, ვიდრე int ტიპს.

გამოსახულებაში შეუძლებელია ყველა ტიპის ავტომატურად შეთავსება. კერძოდ, შეუძლებელია float ან double ტიპის ავტომატურად გარდაქმნა decimal ტიპად. ასევე შეუძლებელია ulong ტიპის შეთავსება რომელიმე სხვა ნიშნიან მთელრიცხვა ტიპთან. მათი გარდაქმნისათვის საჭიროა ტიპის აშკარა გარდაქმნის ოპერაციის გამოყენება.

ოპერანდების მიმართ ტიპების ავტომატური გარდაქმნა სრულდება მხოლოდ მაშინ, როცა ხდება მათი გამოყენება გამოსახულების გამოთვლისას. მაგალითად, თუ გამოსახულებაში byte ტიპის მქონე ცვლადის მნიშვნელობა გარდაიქმნება int ტიპად, მაშინ გამოსახულების გარეთ ეს ცვლადი ინარჩუნებს byte ტიპს.

ტიპის გარდაქმნამ შეიძლება მოგვცეს არასწორი შედეგი. მაგალითად, როცა არითმეტიკულ ოპერაციას ვასრულებთ byte ტიპის ორ ცვლადზე, მაშინ ჯერ სრულდება ამ ცვლადების გარდაქმნა int ტიპის ცვლადებად, შემდეგ კი გამოითვლება გამოსახულება, რომლის შედეგს ექნება int ტიპი. ამიტომ, ჩვენ უნდა ვაკონტროლოთ იმ ცვლადების ტიპები, რომლებსაც ენიჭებათ გამოთვლების შედეგები.

განვიხილოთ პროგრამა.

```
{
//   პროგრამა 2.24
//   პროგრამაში ხდება ტიპების გარდაქმნის დემონსტრირება
int mteli;
byte bait1, bait2;

bait1 = Convert.ToByte(textBox1.Text);
bait2 = Convert.ToByte(textBox2.Text);
mteli = bait1 * bait2;           //   აქ არ არის საჭირო ტიპის აშკარა გარდაქმნა
label1.Text = mteli.ToString();

bait1 = Convert.ToByte(textBox3.Text);
//   შედეგის ტიპი დაიყვანება bait1 ცვლადის ტიპზე
bait1 = ( byte ) ( bait1 * bait2 );
label2.Text = bait1.ToString();
}
```

ტიპების გარდაქმნა საჭირო არ არის როცა bait1 * bait2 გამოსახულების შედეგი ენიჭება mteli ცვლადს, რადგან ამ გამოსახულების შედეგი ავტომატურად გარდაიქმნება int ტიპად. თუ bait1 ცვლადს მივანიჭებთ bait1 * bait2 გამოსახულების შედეგს, მაშინ აუცილებელი იქნება byte ტიპზე დაყვანის ოპერაციის შესრულება bait1 ცვლადის მიმართ.

ასეთივე სიტუაციას ადგილი აქვს char ტიპის მონაცემებზე ოპერაციების ჩატარებისას.

მაგალითად, პროგრამის კოდის ფრაგმენტში:

```
char simbolo1 = 'r', simbolo2 = 's';
```

```
simbolo1 = ( char ) ( simbolo1 + simbolo2 );
```

შედეგი უნდა გარდაიქმნას char ტიპად, რადგან გამოსახულებაში simbolo1 და simbolo2 ცვლადები გარდაიქმნება int ტიპად და შედეგსაც int ტიპი ექნება.

თავი 3. მმართველი ოპერატორები

არსებობს სამი კატეგორიის მმართველი ოპერატორი - ამორჩევის (if და switch), იტერაციისა (for, while, do-while და foreach) და გადასვლის (break, continue, goto და return).

ამორჩევის ოპერატორები

if ოპერატორი

if ოპერატორი ამოწმებს ლოგიკურ პირობას და თუ ის ჭეშმარიტია, მაშინ ასრულებს კოდის მითითებულ ფრაგმენტს. მისი სინტაქსია:

```
if ( პირობა ) { ოპერატორები1; }  
[ else { ოპერატორები2; } ]
```

აქ პირობა არის ლოგიკური გამოსახულება, რომელიც იღებს true ან false მნიშვნელობას. თუ პირობა ჭეშმარიტია, მაშინ შესრულდება ოპერატორები1, წინააღმდეგ შემთხვევაში - ოპერატორები2. ოპერატორები1 და ოპერატორები2 შეიძლება იყოს ერთი ან მეტი ოპერატორი. კვადრატული ფრჩხილები მიუთითებენ იმაზე, რომ else სიტყვის მითითება აუცილებელი არ არის. ასეთ შემთხვევაში, if ოპერატორის სინტაქსი იქნება:

```
if ( პირობა ) { ოპერატორები1; }
```

თუ პირობა იღებს true მნიშვნელობას, მაშინ შესრულდება ოპერატორები1, წინააღმდეგ შემთხვევაში კი - პროგრამის მომდევნო ოპერატორი.

if ოპერატორის მუშაობის საჩვენებლად შევადგინოთ პროგრამა, რომელიც განსაზღვრავს რიცხვი კენტია თუ ლუწი.

```
{  
// პროგრამა 3.1  
// პროგრამა განსაზღვრავს რიცხვი კენტია თუ ლუწი  
int ricxvi;
```

```
ricxvi = int.Parse(textBox1.Text);  
if ( ricxvi % 2 == 1 ) label1.Text = "რიცხვი კენტია";  
else label1.Text = "რიცხვი ლუწია";  
}
```

ჩადგმული if ოპერატორი

ჩადგმულია if ოპერატორი, რომელიც მოთავსებულია სხვა if ოპერატორის შიგნით. ჩადგმული if ოპერატორი იმ შემთხვევაში გამოიყენება, როცა მოქმედების შესასრულებლად საჭიროა რამდენიმე პირობის შემოწმება, რომელთა მითითება ერთ if ოპერატორში შეუძლებელია. ჩადგმულ if ოპერატორში else ნაწილი ყოველთვის ეკუთვნის უახლოეს if ოპერატორს. შევადგინოთ პროგრამა, რომელიც დაადგენს ორ რიცხვს შორის რომელია დადებითი:

```
{  
// პროგრამა 3.2  
// პროგრამა დაადგენს ორ რიცხვს შორის არის თუ არა დადებითი  
int ricxvi1, ricxvi2;
```

```
ricxvi1 = int.Parse(textBox1.Text);
```

```

ricxvi2 = int.Parse(textBox2.Text);
if ( ricxvi1 >= 0 ) label1.Text = "დადებითია პირველი რიცხვი";
    else    if ( ricxvi2 >= 0 ) label1.Text = "დადებითია მეორე რიცხვი";
        else label1.Text = "არც ერთი რიცხვი არ არის დადებითი";
}

```

თუ პირველი if ოპერატორის პირობა ჭეშმარიტია, მაშინ label1 კომპონენტში გამოიცივება შესაბამისი შეტყობინება და პროგრამა მუშაობას დაამთავრებს. წინააღმდეგ შემთხვევაში შემოწმდება მეორე if ოპერატორის პირობა. თუ ის ჭეშმარიტია მაშინ label1 კომპონენტში გამოიცივება შესაბამისი შეტყობინება და პროგრამა მუშაობას დაამთავრებს. თუ არც ერთი პირობა არ არის ჭეშმარიტი, მაშინ label1 კომპონენტში გამოიცივება შესაბამისი შეტყობინება და პროგრამა მუშაობას დაამთავრებს.

პროგრამული კოდის ბლოკები. ხილვადობის უბანი

აქამდე, ცვლადების გამოცხადებას ვახდენდით პროგრამის (მეთოდის) დასაწყისში. ისინი წარმოადგენენ *ლოკალურ ცვლადებს*, რადგან გამოცხადებული არიან მეთოდის *ხილვადობის უბანში* (მეთოდის შიგნით). ლოკალური ცვლადების გამოცხადება შეიძლება, აგრეთვე, მეთოდში *ბლოკის* შიგნით. ბლოკი არის ოპერატორების ერთობლიობა, რომელიც მოთავსებულია { და } ფრჩხილებს შორის მეთოდის შიგნით. ბლოკი განსაზღვრავს ხილვადობის უბანს. ხილვადობის ახალი უბანი იქმნება ბლოკის შექმნისას. ხილვადობის უბანი განსაზღვრავს გამოცხადებული ცვლადების *სიცოცხლის (არსებობის) დროს*.

ხილვადობის უბნები შეიძლება განისაზღვროს კლასით და მეთოდით. ამ ეტაპზე განვიხილავთ მეთოდის მიერ განსაზღვრულ ხილვადობის უბნებს. თუ მეთოდს აქვს პარამეტრები, მაშინ ისინიც ჩართული იქნებიან მეთოდის ხილვადობის უბანში. ხილვადობის უბანში გამოცხადებული ცვლადი უხილავია (მიუწვდომელი) ამ უბნის გარეთ განსაზღვრული კოდისათვის. შედეგად, ხდება ცვლადის დაცვა არასანქცირებული მიმართვისაგან.

მოყვანილ პროგრამაში ხდება ბლოკისა და ხილვადობის უბნის დემონსტრირება.

```

{
//    პროგრამა 3.3
//    პროგრამაში ხდება ბლოკისა და ხილვადობის უბნის დემონსტრირება
int ricxvi1 = int.Parse(textBox1.Text);

if ( ricxvi1 > 25 )
    {
        //        ბლოკის დასაწყისი
        int ricxvi2 = 30;
        ricxvi1 = ricxvi2 * ricxvi2;
    }
    //        ბლოკის დასასრული
label1.Text = ricxvi1.ToString();
//    ricxvi2 = 50;
//        ამ უბანში ricxvi2 რიცხვი არ არსებობს
}

```

ricxvi1 ცვლადი ხილულია მთელი კოდისათვის, რადგან გამოცხადებულია პროგრამის დასაწყისში. ricxvi2 ცვლადი გამოცხადებულია if ბლოკის შიგნით. რადგან ბლოკი განსაზღვრავს ხილვადობის უბანს, ამიტომ ricxvi2 ცვლადი ხილულია მხოლოდ ამ ბლოკის შიგნით. სწორედ ამიტომ არის შეცდომა ricxvi2 ცვლადის გამოყენება ამ ბლოკის გარეთ სტრიქონში ricxvi2 = 50; თუ წავშლით კომენტარის სიმბოლოებს, მაშინ პროგრამის კოდის კომპილირებისას გაიცივება შესაბამისი შეტყობინება შეცდომის შესახებ. if ბლოკის შიგნით შეიძლება ricxvi1 ცვლადის გამოყენება, რადგან ამ ბლოკის კოდს შეუძლია მიმართოს გარე ბლოკში გამოცხადებულ ცვლადს.

ცვლადთან მიმართვა შესაძლებელია მხოლოდ მისი გამოცხადების შემდეგ. ბლოკის შიგნით ცვლადის გამოცხადება ნებისმიერ ადგილში შეიძლება, რის შემდეგ ის ხდება მოქმედი. შედეგად, თუ ცვლადს გამოვაცხადებთ მეთოდის დასაწყისში, მაშინ მასთან მიმართვა შესაძლებელი იქნება მთელი კოდის შიგნით. თუ ცვლადს გამოვაცხადებთ ბლოკის სხვა ადგილში, მაშინ მასთან მიმართვას შევძლებთ მხოლოდ მისი გამოცხადების შემდეგ.

ცვლადები ხილვადობის რომელიმე უბანში იქმნება მათი გამოცხადებისას (მოცემულ უბანში შესვლისას) და იშლება ამ უბნიდან გამოსვლისას. ეს იმას ნიშნავს, რომ ცვლადი კარგავს თავის მნიშვნელობას ხილვადობის უბნიდან გამოსვლისას, ე.ი. მეთოდის შიგნით გამოცხადებული ცვლადი არ შეინახავს თავის მნიშვნელობას ამ მეთოდის გამოძახებებს შორის. ანალოგიურად, ბლოკის შიგნით გამოცხადებული ცვლადიც კარგავს თავის მნიშვნელობას ბლოკიდან გამოსვლისას. შედეგად, ცვლადის სიცოცხლის დრო შეზღუდულია მისი ხილვადობის უბნით.

თუ ცვლადის გამოცხადებისას ხდება მისი ინიციალიზება, მაშინ მისი ინიციალიზება შესრულდება ყოველთვის ამ ბლოკში შესვლისას.

ხილვადობის უბანი შეიძლება იყოს ჩადგმული. მაგალითად, როცა ვქმნით პროგრამული კოდის ბლოკს, ჩვენ ვქმნით ახალ ჩადგმულ ხილვადობის უბანს. ამასთან, გარე უბანი ხდება ხილული ჩადგმული უბნისათვის. ეს იმას ნიშნავს, რომ გარე ხილვადობის უბანში გამოცხადებული ობიექტები და ცვლადები ხილული იქნება შიდა ხილვადობის უბნის კოდისთვის. შიდა ხილვადობის უბანში გამოცხადებული ობიექტები და ცვლადები არ იქნება ხილული გარე ხილვადობის უბნის კოდისთვის. ბოლოს, შევნიშნოთ რომ გარე და ჩადგმულ ბლოკებში გამოცხადებულ ცვლადებს არ შეიძლება ერთნაირი სახელები ჰქონდეს.

გლობალურია ცვლადი, რომელიც გამოცხადებულია ყველა მეთოდის გარეთ. მისი გამოცხადება შეიძლება Main() მეთოდის შემდეგ. გლობალური ცვლადები მოქმედებენ ყველა მეთოდის შიგნით და არ კარგავენ თავიანთ მნიშვნელობებს მეთოდიდან გამოსვლის დროს. მოყვანილი პროგრამით ხდება გლობალურ ცვლადთან მუშაობის დემონსტრირება.

```
{
//   პროგრამა 3.4
//   პროგრამაში ხდება გლობალურ ცვლადთან მუშაობის დემონსტრირება
//   გლობალური ცვლადის გამოცხადება
int globaluri_cvლადი;
private void button1_Click(object sender, System.EventArgs e)
{
//   ლოკალური ცვლადის გამოცხადება
int lokaluri_cvლადი = 10;

globaluri_cvლადი = int.Parse(textBox1.Text);
label1.Text = globaluri_cvლადი.ToString();
if ( globaluri_cvლადი > 3 )
{
    globaluri_cvლადი = 50;
    lokaluri_cvლადი = 20;
}
label2.Text = globaluri_cvლადი.ToString();
}
```

switch ოპერატორი

ის საშუალებას გვაძლევს ვარიანტების სიიდან ამოვარჩიოთ საჭირო მოქმედება. ის შემდეგნაირად მუშაობს: გამოსახულების მნიშვნელობა მიმდევრობით შედარდება სიაში მითითებულ თითოეულ მუდმივას (კონსტანტას). ერთ-ერთ მუდმივასთან მნიშვნელობის დამთხვევისას შესრულდება მასთან ასოცირებული (დაკავშირებული) ოპერატორების მიმდევრობა. switch ოპერატორის სინტაქსია:

```
switch ( გამოსახულება )
{
    case მუდმივა1 :      ოპერატორების მიმდევრობა;
                        break;
    case მუდმივა2 :      ოპერატორების მიმდევრობა;
                        break;
    ...
    [ default :          ოპერატორების მიმდევრობა;
                        break; ]
}
```

აქ გამოსახულების მნიშვნელობას უნდა ჰქონდეს მთელი რიცხვა ტიპი - char, byte, short ან int, ან სტრიქონული ტიპი - string (მას მოგვიანებით განვიხილავთ). გამოსახულების მნიშვნელობას არ უნდა ჰქონდეს მცურავწერტილიანი ტიპი (double, float). switch ოპერატორის case განშტოებების მუდმივები უნდა იყოს ლიტერალები და ჰქონდეთ გამოსახულებების ტიპთან თავსებადი ტიპი. ამასთან, switch ოპერატორის ერთ ბლოკში მათ არ შეიძლება ერთნაირი მნიშვნელობები ჰქონდეთ.

ერთ case განშტოებასთან დაკავშირებულმა ოპერატორების მიმდევრობამ მართვა არ უნდა გადასცეს მომდევნო case განშტოების ოპერატორებს. ამის თავიდან ასაცილებლად თითოეული case განშტოების ოპერატორების მიმდევრობა break ოპერატორით უნდა დავამთავროთ.

default განშტოების შესაბამისი ოპერატორების მიმდევრობა სრულდება მაშინ, როცა case განშტოებების არც ერთი მუდმივა არ შეესაბამება გამოსახულების მნიშვნელობას. default განშტოების მითითება არ არის აუცილებელი. თუ ის არ არის მითითებული და switch ოპერატორში არც ერთი მუდმივა არ შეესაბამება გამოსახულების მნიშვნელობას, მაშინ არავითარი მოქმედება არ შესრულდება და მართვა გადაეცემა switch ოპერატორის შემდეგ მოთავსებულ ოპერატორს. თუ გამოსახულების მნიშვნელობა დაემთხვა მუდმივას მნიშვნელობას, მაშინ შესრულდება ამ case განშტოებასთან ასოცირებული ოპერატორების მიმდევრობა break ოპერატორამდე. break ოპერატორი მართვას გადასცემს switch ოპერატორის შემდეგ მოთავსებულ ოპერატორს. მოვიყვანოთ პროგრამა, რომელიც ახდენს ციფრების ფიქსირებას დიაპაზონში 0 - 2.

```
{
//   პროგრამა 3.5
//   პროგრამა აფიქსირებს ციფრებს დიაპაზონში 0 - 2
int ricxvi;

ricxvi = int.Parse(textBox1.Text);
switch ( ricxvi )
{
    case 0 : label1.Text = "თქვენ შეიტანეთ ციფრი - " + ricxvi.ToString();
            break;
    case 1 : label1.Text = "თქვენ შეიტანეთ ციფრი - " + ricxvi.ToString();
            break;
    case 2 : label1.Text = "თქვენ შეიტანეთ ციფრი - " + ricxvi.ToString();
```



```

        break;
    default : label1.Text = "შეტანილი რიცხვი არ არის დიაპაზონში 0 - 2";
        break;
    }
}

```

ამ პროგრამაში switch ოპერატორის მართვა ხდება int ტიპის ricxvi ცვლადის მეშვეობით.

ახლა შევადგინოთ პროგრამა, სადაც switch ოპერატორის მართვა ხორციელდება char ტიპის მქონე გამოსახულებით. ამ შემთხვევაში case განშტოების მუდმივებსაც უნდა ჰქონდეთ char ტიპი. პროგრამა გამოთვლებს ასრულებს იმის მიხედვით თუ არითმეტიკის რომელ ოპერატორს შევიტანთ.

```

{
//    პროგრამა 3.6
//    კალკულატორი
int shedegi, ricxvi1, ricxvi2;
char operacia;

ricxvi1 = int.Parse(textBox1.Text);
ricxvi2 = int.Parse(textBox3.Text);
operacia = char.Parse(textBox2.Text);
switch ( operacia )
{
case '+':      shedegi = ricxvi1 + ricxvi2;
                label1.Text = shedegi.ToString();
                break;
case '-':      shedegi = ricxvi1 - ricxvi2;
                label1.Text = shedegi.ToString();
                break;
case '*':      shedegi = ricxvi1 * ricxvi2;
                label1.Text = shedegi.ToString();
                break;
case '/':      shedegi = ricxvi1 / ricxvi2;
                label1.Text = shedegi.ToString();
                break;
case '%':      shedegi = ricxvi1 % ricxvi2;
                label1.Text = shedegi.ToString();
                break;
}
}

```

switch ოპერატორის ორი ან მეტი განშტოება შეიძლება დაკავშირებული იყოს ოპერატორების ერთსა და იმავე მიმდევრობასთან. ამას სხვანაირად ჩავარდნას უწოდებენ. ქვემოთ მოყვანილი პროგრამა ახდენს ჩავარდნის დემონსტრირებას. პროგრამა განასხვავებს ანბანის ხმოვან და თანხმოვან ასოებს.

```

{
//    პროგრამა 3.7
//    პროგრამაში ხდება ჩავარდნის დემონსტრირება
char simbolo;
label1.Text = "";

```

```

simbolo = textBox1.Text[0];
switch ( simbolo )
{
    case 'ა' :
    case 'ე' :
    case 'ო' :
    case 'ლ' :
    case 'უ' : label1.Text = "ეს არის ხმოვანი ასო";
                break;
    default : label1.Text = "ეს არ არის ხმოვანი ასო";
                break;
}
}

```

თუ simbolo ცვლადი იღებს მნიშვნელობას 'ა', 'ე', 'ო', 'ლ' ან 'უ', მაშინ შესრულდება label1.Text = "ეს არის ხმოვანი ასო"; ოპერატორი, წინააღმდეგ შემთხვევაში - label1.Text = "ეს არ არის ხმოვანი ასო"; ოპერატორი.

ჩადგმული switch ოპერატორი

ერთი switch ოპერატორი შეიძლება მოთავსებული იყოს გარე switch ოპერატორში. შიდა switch ოპერატორს ეწოდება ჩადგმული. შიდა და გარე switch ოპერატორების case განშტოებების მუდმივებს შეიძლება ერთნაირი მნიშვნელობები ჰქონდეთ. ეს არ გამოიწვევს კონფლიქტს პროგრამის შესრულების დროს. განვიხილოთ მაგალითი.

```

{
//      პროგრამა 3.8
//      პროგრამაში ხდება ჩადგმული switch ოპერატორის მუშობის დემონსტრირება
char simbolo1, simbolo2;

```

```

simbolo1 = textBox1.Text[0];
simbolo2 = textBox2.Text[0];
switch ( simbolo1 )
{
    case 'ა' :
        label1.Text = "ა მუდმივა ეკუთვნის გარე switch ოპერატორს";
        switch ( simbolo2 )
        {
            case 'ა' :
                label2.Text = "ა მუდმივა ეკუთვნის შიდა switch ოპერატორს";
                break;

            case 'ბ' :
                label2.Text = "ბ მუდმივა ეკუთვნის შიდა switch ოპერატორს";
                break;
        }
        break;

    case 'ბ' :

```

```

label1.Text = "ხ მუდმივა ეკუთვნის გარე switch ოპერატორს";
break;
}
}

```

იტერაციის ოპერატორები

for ოპერატორი

ის გამოიყენება ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად. მისი სინტაქსია:

```
for ( [ ინიციალიზატორი ]; [ პირობა ]; [ იტერატორი ] ) [ ციკლის კოდი ];
```

ინიციალიზატორი არის გამოსახულება, რომელიც ციკლის დაწყების წინ გამოითვლება. აქ ხდება ციკლის მმართველი ცვლადის ინიციალიზება. *პირობა* არის ლოგიკური გამოსახულება, რომელიც იღებს true ან false მნიშვნელობას. ის გამოითვლება ციკლის ყოველი იტერაციის წინ. *for* ციკლის გამეორება ხდება მანამ, სანამ პირობა იღებს true მნიშვნელობას. თუ მან მიიღო false მნიშვნელობა, მაშინ ციკლი მთავრდება და მართვა გადაეცემა *for* ოპერატორის შემდეგ მოთავსებულ ოპერატორს. *იტერატორი* არის გამოსახულება, რომელიც ზრდის ან ამცირებს (ციკლის) მმართველი ცვლადის მნიშვნელობას. ის გამოითვლება ციკლის ყოველი იტერაციის შემდეგ. *ციკლის კოდი* (ციკლის ტანი) შეიძლება შეიცავდეს ერთ ან მეტ ოპერატორს.

for ციკლი შემდეგნაირად მუშაობს. ჯერ შესრულდება ციკლის მმართველი ცვლადის ინიციალიზება. შემდეგ მოწმდება პირობა. თუ ის ჭეშმარიტია, მაშინ შესრულდება ციკლის კოდი. შემდეგ იცვლება ციკლის ცვლადის მნიშვნელობა. კვლავ მოწმდება პირობა და ა.შ.

შევადგინოთ პროგრამა, რომელიც ანგარიშობს 1-დან 10-მდე რიცხვების კვადრატს. პროგრამაში ციკლის მმართველი ცვლადი ერთით იზრდება.

```

{
//   პროგრამა 3.9
//   პროგრამა ანგარიშობს 1-დან 10-მდე რიცხვების კვადრატს
label1.Text = "";
double ricxvi, kvadrati;

for ( ricxvi = 1; ricxvi <= 10; ricxvi++ )
{
    kvadrati = Math.Pow(ricxvi, 2);
    label1.Text += ricxvi.ToString() + " - " + kvadrati.ToString() + '\n';
}
}

```

label1.Text = ""; ოპერატორის შესრულების შედეგად *label1* კომპონენტში გამოტანილი "label1" სტრიქონი წაიშლება. საქმე ის არის, რომ როცა *label1* კომპონენტს ფორმაზე ვათავსებთ, მის *Text* თვისებაში ავტომატურად გამოჩნდება "label1" სტრიქონი. მისი წაშლა შეგვიძლია *Properties* ფანჯარაში ან პროგრამულად ამ კომპონენტის *Text* თვისებისათვის ცარიელი სტრიქონის მინიჭების გზით: *label1.Text = ""*; . თუ ამ ოპერატორს არ შევასრულებთ, მაშინ *label1* კომპონენტში გამოტანილ სტრიქონს დაემატება პროგრამის მიერ გასაცემი მონაცემები. '\n' სიმბოლო ახდენს *label* კომპონენტის შიგნით მომდევნო სტრიქონზე გადასვლას..

ციკლის ცვლადი შეიძლება ზრდიდეს ან ამცირებდეს თავის მნიშვნელობას ნებისმიერი სიდიდით. ქვემოთ მოყვანილ პროგრამაში ციკლის ცვლადი მცირდება 10 ერთეულით.

```
{
```

```
//   პროგრამა 3.10
//   ციკლის მმართველი ცვლადი იცვლება უარყოფითი ბიჯით
label1.Text = "";
double ricxvi, kvadrati;

for ( ricxvi = 100; ricxvi >= 1; ricxvi -= 10 )
{
    kvadrati = Math.Pow(ricxvi, 2);
    label1.Text += ricxvi.ToString() + " - " + kvadrati.ToString() + '\n';
}
}
```

როგორც ვიცით, for ციკლის პირობა ციკლის დასაწყისში მოწმდება. ეს იმას ნიშნავს, რომ თუ პირობა თავიდანვე იღებს false მნიშვნელობას, მაშინ ციკლის ტანი არასოდეს არ შესრულდება, მაგალითად,

```
for ( ricxvi1 = 5; ricxvi1 < 3; ricxvi1++ )
    ricxvi2 = ricxvi1 * ricxvi1;
ამ ციკლის ტანი არასოდეს არ შესრულდება, რადგან პირობის გამოსახულება თავიდანვე იღებს false მნიშვნელობას.
for ციკლში ერთის ნაცვლად შეგვიძლია ორი მმართველი ცვლადის გამოყენება, მაგალითად,
```

```
{
//   პროგრამა 3.11
//   პროგრამაში for ციკლისათვის გამოიყენება ორი მმართველი ცვლადი
label1.Text = "";
int ricxvi1, ricxvi2;

for ( ricxvi1 = 0, ricxvi2 = 10; ricxvi1 < ricxvi2; ricxvi1++, ricxvi2-- )
    label1.Text = label1.Text + " " + ricxvi1.ToString() + " " + ricxvi2.ToString() + '\n';
}
```

აქ ინიციალიზაციის ორი ოპერატორი გამოყოფილია მძიმით. ასევე, ორი იტერაციული გამოსახულება გამოყოფილია მძიმით. როცა ციკლი იწყებს მუშაობას ორივე ცვლადს ენიჭება საწყისი მნიშვნელობები. ყოველი იტერაციის შემდეგ ricxvi1 ცვლადი მნიშვნელობა ერთით იზრდება, ricxvi2 ცვლადის მნიშვნელობა კი - ერთით მცირდება. for ციკლის მართვა რამდენიმე მმართველი ცვლადის გამოყენებით ზოგჯერ საკმაოდ მოხერხებულია და ამარტივებს ალგორითმებს. ციკლში დასაშვებია ნებისმიერი რაოდენობის ინიციალიზაციისა და იტერაციული ოპერატორების გამოყენება, მაგრამ პრაქტიკულად, ორ ცვლადზე მეტი იშვიათად გამოიყენება.

for ციკლის ჩაწერისას შეგვიძლია არ მივუთითოთ ინიციალიზაციის, პირობის ან იტერაციის ნაწილი, მაგალითად,

```
{
//   პროგრამა 3.12
//   პროგრამაში გამოყენებულია for ოპერატორი იტერაციის ნაწილის გარეშე
label1.Text = "";
int ricxvi1;

//   ამ ციკლში მითითებული არ არის იტერაციის ნაწილი
for ( ricxvi1 = 0; ricxvi1 < 10; )
```

```

{
    label1.Text += ricxvil.ToString() + '\n';
    ricxvil++;
}
}

```

აქ არ არის მითითებული იტერაციის ნაწილი, სამაგიეროდ, მმართველი ცვლადი ერთით იზრდება ციკლის ტანში.

ქვემოთ მოყვანილ პროგრამაში გამოყენებულია for ციკლი, რომელშიც არ არის მითითებული ინიციალიზაციისა და იტერაციის ნაწილები.

```

{
//    პროგრამა 3.13
//    პროგრამაში გამოყენებულია for ოპერატორი ინიციალიზებისა და იტერაციის
//    ნაწილების გარეშე
label1.Text = "";
int ricxvil = 0;

```

```

//    ციკლი ინიციალიზატორისა და იტერატორის გარეშე
for ( ; ricxvil < 10; )
{
    label1.Text += ricxvil.ToString() + '\n';
    ricxvil++;
}
}

```

აქ ricxvil ცვლადის ინიციალიზება ხდება ციკლამდე. ასე ძირითადად ვიქცევით მაშინ, როცა მმართველი ცვლადი საწყის მნიშვნელობას იღებს რაიმე რთული გამოსახულების გამოთვლის შედეგად.

თუ for ციკლის არც ერთი ნაწილი არ არის მითითებული, მაშინ მივიღებთ უსასრულო ციკლს, მაგალითად

```

for ( ; ; )
{
//    ციკლის კოდი
}

```

უსასრულო ციკლის შესაწყვეტად შეგვიძლია break ოპერატორი გამოვიყენოთ.

for ციკლს შეიძლება არ ჰქონდეს, აგრეთვე, კოდი, ანუ ის იყოს ცარიელი. ეს შესაძლებელია, რადგან ნულოვანი ოპერატორი სინტაქსურად დასაშვებია. ქვემოთ მოყვანილია პროგრამა, რომელშიც ასეთი ციკლი გამოიყენება 1-დან 5-მდე რიცხვების შესაკრებად.

```

{
//    პროგრამა 3.14
//    პროგრამაში გამოყენებულია for ოპერატორი კოდის გარეშე
label1.Text = "";
int ricxvi, jami = 0;

```

```

for ( ricxvi = 1; ricxvi <= 5; jami += ricxvi++ );           //    ამ ციკლში კოდი არ არის
label1.Text = jami.ToString();
}

```

პროგრამიდან ჩანს, რომ ჯამის გამოთვლა ხდება for ოპერატორის შიგნით. შედეგად, ციკლის ტანი საჭირო არ არის. რაც შეეხება გამოსახულებას jami += ricxvi++; ის შემდეგნაირად

მუშაობს. jami ცვლადს ენიჭება მისსავე მნიშვნელობას დამატებული ricxvi ცვლადის მნიშვნელობა. შემდეგ, ricxvi მნიშვნელობა ერთით იზრდება. ეს ოპერატორი შემდეგი ორი ოპერატორის ანალოგიურია:

```
jami = jami + ricxvi;
ricxvi++;
```

ხშირად მმართველი ცვლადი გამოიყენება მხოლოდ for ციკლის შიგნით. ასეთ შემთხვევაში მისი გამოცხადება ხდება ციკლის საინიციალიზაციო ნაწილში. განვიხილოთ პროგრამა, რომელიც ახდენს პირველი ხუთი რიცხვის შეკრებას.

```
{
//   პროგრამა 3.15
//   პროგრამაში გამოყენებულია for ოპერატორი, რომელშიც გამოცხადებულია
//   მმართველი ცვლადი
label1.Text = "";
int jami = 0;

for ( int ricxvi = 1; ricxvi <= 5; ricxvi++ )
    jami += ricxvi;
label1.Text = jami.ToString();
}
```

ricxvi მმართველი ცვლადი გამოცხადებულია for ციკლის საინიციალიზაციო ნაწილში, ამიტომ, ის ხილულია მხოლოდ ამ ციკლის ფარგლებში. მის გარეთ ის უხილავია. თუ ვაპირებთ მმართველი ცვლადის გამოყენებას პროგრამის სხვა ნაწილში, მაშინ ის უნდა გამოვაცხადოთ for ციკლის გარეთ.

მაგალითები

შევადგინოთ პროგრამა, რომელიც შეკრებს რიცხვებს მითითებულ დიაპაზონში.

```
{
//   პროგრამა 3.16
//   მითითებულ დიაპაზონში რიცხვების შეკრების პროგრამა
int index, min, max, jami = 0;
```

```
min = int.Parse(textBox1.Text);           //   საწყისი რიცხვი
max = int.Parse(textBox2.Text);           //   საბოლოო რიცხვი
for ( index = min; index <= max; index++ )
    jami += index;
label1.Text = jami.ToString();
}
```

შევადგინოთ ფაქტორიალის გამოთვლის პროგრამა.

```
{
//   პროგრამა 3.17
//   ფაქტორიალის გამოთვლის პროგრამა
int cvladi, ricxvi, shedegi = 1;

ricxvi = int.Parse(textBox1.Text);
for ( cvladi = 1; cvladi <= ricxvi; cvladi++ )
    shedegi *= cvladi;
label1.Text = shedegi.ToString();
}
```

```
}  
შევადგინოთ პროგრამა, რომელიც გამოთვლის მოცემული მწკრივის მნიშვნელობას
```

$$= \quad - \quad - \quad + \quad - \quad - \quad - \quad + \quad - \quad -$$

```
{  
// პროგრამა 3.18  
// მწკრივის გამოთვლის პროგრამა  
double shedegi = 0, x;  
int minusi = -1;  
  
x = Convert.ToDouble(textBox1.Text);  
for ( int ricxvi = 1; ricxvi <= 9; ricxvi += 2 )  
    {  
        minusi *= -1;  
        shedegi += minusi * Math.Pow(x, ricxvi) / ricxvi;  
    }  
label1.Text = shedegi.ToString();  
}
```

while ციკლი

ის გამოიყენება ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად. მისი სინტაქსია:

while (პირობა)

```
{  
ციკლის კოდი  
}
```

აქ პირობა არის ლოგიკური გამოსახულება, რომელიც გასცემს true ან false მნიშვნელობას. თუ ის იღებს false მნიშვნელობას, მაშინ ხდება ციკლიდან გამოსვლა და სრულდება მომდევნო ოპერატორი. თუ ის იღებს true მნიშვნელობას, მაშინ შესრულდება ციკლის კოდი.

განვიხილოთ პროგრამა, რომელსაც ეკრანზე გამოაქვს ინგლისური ანბანის ასოები.

```
{  
// პროგრამა 3.19  
// პროგრამას label1 კომპონენტში გამოაქვს ქართული ანბანის ასოები  
label1.Text = "";  
char simbolo = 'ა';  
  
while ( simbolo <= 'ჰ' )  
    {  
        label1.Text += simbolo.ToString() + '\n';  
        simbolo++;  
    }  
}
```

simbolo ცვლადს ენიჭება საწყისი მნიშვნელობა - 'ა'. ციკლის ყოველი შესრულებისას label1 კომპონენტში გამოიცემა simbolo ცვლადის მნიშვნელობა, შემდეგ კი მისი მნიშვნელობა ერთით იზრდება.

do-while ციკლი

ის გამოიყენება ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად. for და while ციკლებისაგან განსხვავებით, რომლებშიც ჯერ მოწმდება პირობა და შემდეგ სრულდება ციკლის კოდი, do-while ციკლში ჯერ შესრულდება ციკლის კოდი, შემდეგ კი მოწმდება პირობა. მისი სინტაქსია:

```
do
{
ციკლის კოდი
}
```

while (პირობა)

თუ ციკლში ერთი ოპერატორი სრულდება, მაშინ ფიგურული ფრჩხილების გამოყენება აუცილებელი არ არის, მაგრამ მათ ხშირად იყენებენ do-while ციკლის კითხვადობის გაუმჯობესების მიზნით, რადგან ის ადვილად შეიძლება აგვერიოს while ციკლში.

ქვემოთ მოყვანილია პროგრამა, რომელსაც label კომპონენტში გამოაქვს ქართული ანბანის ასოები 'ა'-დან 'კ'-მდე.

```
{
// პროგრამა 3.20
// პროგრამას label1 კომპონენტში გამოაქვს ქართული ანბანის ასოები
label1.Text = "";
char simbolo = 'ა';

do
{
    label1.Text += simbolo.ToString() + '\n';
    simbolo++;
}
while ( simbolo != 'კ' );
}
```

გადასვლის ოპერატორები

break ოპერატორი

ის გვაძლევს ციკლის შესრულების იძულებით შეწყვეტისა და ციკლიდან გამოსვლის საშუალებას. ამ დროს, ციკლის დანარჩენი ოპერატორები არ შესრულდება. მართვა გადაეცემა ციკლის შემდეგ მოთავსებულ ოპერატორს. ქვემოთ მოყვანილი პროგრამა ახდენს რიცხვების გამრავლებას მანამ, სანამ ნამრავლი არ გახდება 30-ზე მეტი.

```
{
// პროგრამა 3.21
// პროგრამა რიცხვებს ამრავლებს მანამ, სანამ ნამრავლი არ გახდება 30-ზე მეტი
label1.Text = "";
int ricxvi, namravli = 1;

for ( ricxvi = 1; ricxvi < 50; ricxvi++ )
{
    namravli *= ricxvi;
}
```



```

        if ( namravli > 30 ) break;
        label1.Text += '\n' + namravli.ToString();
    }
    ricxvi--;
    label2.Text = "ციკლი შესრულდა " + ricxvi.ToString() + "-ჯერ";
}

```

ციკლი უნდა შესრულდეს 50-ჯერ, მაგრამ break ოპერატორის შესრულება იწვევს მის ვადამდე შეწყვეტას. ციკლი შესრულდება მხოლოდ ოთხჯერ.

break ოპერატორის გამოყენება შეიძლება ნებისმიერი ციკლის შიგნით. თუ break ოპერატორი გამოიყენება ჩადგმული ციკლის შიგნით, მაშინ ის შეწყვეტს მხოლოდ შიდა ციკლის მუშაობას. მაგალითად,

```

{
//    პროგრამა 3.22
//    პროგრამა თვლის თუ რამდენჯერ შესრულდა თითოეული ციკლი
    label1.Text = ""; label2.Text = "";
    int gare = 0, shida = 1;

    for ( int i1 = 0; i1 < 5; i1++ )                //    გარე ციკლი
    {
        gare++;
        label1.Text = "გარე ციკლში იტერაციების რაოდენობა = " + gare.ToString();

        for ( ; shida < 30; )                        //    შიდა ციკლი
        {
            //    break ოპერატორი შეწყვეტს შიდა ციკლის შესრულებას
            //    თუ shida ცვლადი იღებს მნიშვნელობას 5
            if ( shida == 5 ) break;
            label2.Text += shida.ToString() + ' ';
            shida++;
        }
        label1.Text += '\n';
    }
}

```

continue ოპერატორი

ის საშუალებას გვაძლევს იძულებით გადავიდეთ მომდევნო იტერაციაზე. continue ოპერატორის შემდეგ მოთავსებული ოპერატორები არ შესრულდება. განვიხილოთ პროგრამა, რომელსაც label კომპონენტში გამოაქვს კენტი რიცხვები 1-დან 21-მდე.

```

{
//    პროგრამა 3.23
//    პროგრამას label1 კომპონენტში გამოაქვს კენტი რიცხვები 1-დან 21-მდე
    label1.Text = " ";
    int ricxvi;

    for ( ricxvi = 1; ricxvi <= 21; ricxvi++ )
    {
        if ( ( ricxvi % 2 ) != 1 ) continue;        //    მომდევნო იტერაციაზე გადასვლა
    }
}

```

```

    label1.Text += ricxvi.ToString() + '\n';
}
}

```

while და do-while ციკლებში continue ოპერატორის გამოყენება იწვევს მართვის გადაცემას უშუალოდ იტერატორისათვის, რის შემდეგ ციკლის შესრულება გრძელდება.

goto ოპერატორი

goto არის უპირობო გადასვლის ოპერატორი. მისი სინტაქსია:

goto ჭდე;

სადაც, ჭდე არის იდენტიფიკატორი, რომელსაც ორი წერტილი (:) მოსდევს. ის მიუთითებს იმ ადგილს, საიდანაც უნდა გაგრძელდეს პროგრამის შესრულება. თანამედროვე პროგრამირებაში ის ნაკლებად გამოიყენება, რადგან მისი გამოყენების დროს პროგრამა ხდება ნაკლებად მოდულური და სტრუქტურირებული. თუმცა მისი გამოყენება ზოგჯერ საკმაოდ მოხერხებულია. ჭდე მოქმედებს მხოლოდ მოცემული მეთოდის შეგნით. მოყვანილ პროგრამაში ნაჩვენებია თუ როგორ შეიძლება ციკლის ორგანიზება goto ოპერატორის გამოყენებით:

```

{
//    პროგრამა 3.24
//    პროგრამაში ხდება goto ოპერატორის მუშაობის დემონსტრირება
int ricxvi = 1;
cikli1:
    ricxvi++;
    if ( ricxvi < 8 ) goto cikli1;
label1.Text = ricxvi.ToString();
}

```

ტერნარული ოპერატორი

? ოპერატორს ეწოდება ტერნარული, რადგან მას სამი ოპერანდი აქვს. მისი სინტაქსია:

გამოსახულება1? გამოსახულება2: გამოსახულება3;

აქ გამოსახულება1 არის ლოგიკური გამოსახულება, ხოლო გამოსახულება2 და გამოსახულება3 გამოსახულებებია, რომლებსაც ერთნაირი ტიპები უნდა ჰქონდეთ. თავდაპირველად შეფასდება გამოსახულება1. თუ მისი მნიშვნელობაა true, მაშინ გაიცემა გამოსახულება2-ის მნიშვნელობა, წინააღმდეგ შემთხვევაში კი - გამოსახულება3-ის მნიშვნელობა. მაგალითად,

```
max = ricxvi1 > ricxvi2 ? ricxvi1 : ricxvi2;
```

```
min = ricxvi1 < ricxvi2 ? ricxvi1 : ricxvi2;
```

პირველი გამოსახულება ricxvi1 და ricxvi2 რიცხვებიდან ამოირჩევს მაქსიმალურს, მეორე კი - მინიმალურს.

ქვემოთ მოყვანილ პროგრამაში ხდება ორი რიცხვის გაყოფა ისე, რომ თავიდან ავიცილოთ ნულზე გაყოფა.

```

{
//    პროგრამა 3.25
//    50 იყოფა -3-დან 5-მდე მოთავსებულ რიცხვებზე ისე, რომ ხდება 0-ზე გაყოფის
//    თავიდან აცილება
label1.Text = "";
int shedegi;

```

```
for ( int ricxvi = -3; ricxvi < 5; ricxvi++)
{
    shedegi = ricxvi != 0 ? 50 / ricxvi : 0;
    if ( ricxvi != 0 ) label1.Text += ricxvi.ToString() + " " + shedegi.ToString() + '\n';
}
}
```

თავი 4. მასივები, ბიტობრივი ოპერაციები და ჩამოთვლები

მასივები

მასივი არის ერთი ტიპის მქონე ობიექტების კოლექცია (ერთობლიობა). ის გამოიყენება ერთნაირი ტიპის მქონე რამდენიმე ცვლადის ან ობიექტის შესანახად. მასივის ელემენტთან მიმართვისათვის უნდა მივუთითოთ მასივის სახელი და ელემენტის ინდექსი. მასივი შეიძლება იყოს როგორც ერთ, ისე მრავალგანზომილებიანი. ისინი მრავალი ამოცანის გადასაწყვეტად გამოიყენებიან, რადგან წარმოადგენენ ერთი ტიპის მქონე მონაცემების დაჯგუფების მოხერხებულ საშუალებას. მასივში შეგვიძლია შევინახოთ თანამშრომლების გვარები, მათი ხელფასები, ასაკი და ა.შ.

მასივები წარმოადგენენ მიმართვითი ტიპის მქონე ობიექტებს. მასივის ელემენტები დანომრილია, რაც აადვილებს მათთან მუშაობას. სწორედ ამაშია მასივის მთავარი დანიშნულება.

ერთგანზომილებიანი მასივები

ერთგანზომილებიანი მასივში კონკრეტული ელემენტის მდებარეობა განისაზღვრება ერთი ინდექსით. ერთგანზომილებიანი მასივის სინტაქსია:

ტიპი [] მასივის_სახელი = new ტიპი[ზომა];

აქ ტიპი განსაზღვრავს მასივის ტიპს, ანუ მისი თითოეული ელემენტის ტიპს. ტიპის მარჯვნივ კვადრატული ფრჩხილები მიუთითებენ ერთგანზომილებიან მასივზე. ზომა არის მასივში ელემენტების რაოდენობა. რადგან მასივები რეალიზებულია ობიექტების სახით, ამიტომ მასივის შექმნის პროცესი ორი ეტაპისაგან შედგება. პირველ ეტაპზე ხდება მასივის სახელის გამოცხადება, რომელიც წარმოადგენს მიმართვითი ტიპის ცვლადის სახელს. მეორე ეტაპზე მასივისთვის მეხსიერება გამოიყოფა და მასივის ცვლადს მიენიჭება მეხსიერების ამ უბანზე მიმართვა (ანუ ამ უბნის მისამართი). C# ენაში new ოპერატორის გამოყენებით მასივს მეხსიერება დინამიურად (პროგრამის შესრულების დროს) გამოეყოფა. ქვემოთ მოყვანილ სტრიქონში იქმნება int ტიპის მასივი, რომელიც 10 ელემენტისაგან შედგება და მიმართვითი ტიპის masivi ცვლადს ენიჭება მასზე მიმართვა:

```
int [ ] masivi = new int[10];
```

masivi ცვლადი ინახავს მიმართვას მეხსიერების უბანზე (ანუ ამ უბნის მისამართს), რომელიც new ოპერატორის მიერ გამოიყო მისთვის. გამოყოფილი მეხსიერების ზომა საკმარისია მასში 10 ცალი int ტიპის ელემენტის მოსათავსებლად. რადგან int ტიპის ცვლადი 4 ბაიტს (32 ბიტს) იკავებს, ამიტომ masivi მასივისათვის მეხსიერებაში სულ $4 \times 10 = 40$ ბაიტი გამოიყოფა.

მასივის ცალკეულ ელემენტთან მიმართვა ხორციელდება ინდექსის გამოყენებით. მისი საშუალებით ეთითება ელემენტის პოზიცია მასივის ფარგლებში. ნებისმიერი მასივის პირველ ელემენტს ნულოვანი ინდექსი აქვს. რადგან masivi მასივი 10 ელემენტისაგან შედგება, ამიტომ ამ მასივის ინდექსის მნიშვნელობები მოთავსებული იქნება დიაპაზონში 0÷9. შედეგად, masivi მასივის პირველი ელემენტი აღინიშნება ასე - masivi[0], უკანასკნელი კი - masivi[9].

C# ენაში მასივები რეალიზებულია როგორც ობიექტები, რაც იძლევა გარკვეულ უპირატესობებს. ერთ-ერთია Length თვისების არსებობა. ყოველ მასივს აქვს Length თვისება. მასში ავტომატურად იწერება მასივის ელემენტების მაქსიმალური რაოდენობა. ის არ ინახავს ფაქტიურად არსებული ელემენტების რაოდენობას. კერძოდ, თუ გვაქვს 10 ელემენტიანი მასივი და შევსებულია მისი პირველი 4 ელემენტი, მაშინ Length თვისების მნიშვნელობა იქნება 10 და არა 4. Length თვისება წარმატებით გამოიყენება მასივის ინდექსების მნიშვნელობების საზღვრებს გარეთ გასვლის თავიდან ასაცილებლად.

ქვემოთ მოყვანილ პროგრამაში სრულდება masivi მასივის შევსება ციფრებით 0-დან 9-მდე

და შემდეგ მათი გამოტანა label კომპონენტში.

```
{
//   პროგრამა 4.1
//   პროგრამაში ხდება მასივის შევსება რიცხვებით და მათი label კომპონენტში გამოტანა
label1.Text = "";
int[] masivi = new int[10];           //   masivi მასივის გამოცხადება
int indexi;

//   masivi მასივის შევსება რიცხვებით 0-დან 9-მდე
for ( indexi = 0; indexi < masivi.Length; indexi++ )
    masivi[indexi] = indexi;
//   masivi მასივის ელემენტების გამოტანა label1 კომპონენტში
for ( indexi = 0; indexi < masivi.Length; indexi++ )
    label1.Text += masivi[indexi].ToString() + " ";
}
```

მასივის ინიციალიზება

მასივის ინიციალიზება შესაძლებელია მისი შექმნისთანავე. ერთგანზომილებიანი მასივის ინიციალიზაციის სინტაქსია:

ტიპი [] მასივის_სახელი = new ტიპი [] { სიდიდე1, სიდიდე2, ..., სიდიდეN};

აქ საწყისი მნიშვნელობები მოიცემა *სიდიდე1*-დან *სიდიდეN*-მდე სიდიდეებით. მასივის ელემენტებს მნიშვნელობები ენიჭებათ რიგრიგობით მარცხნიდან მარჯვნივ დაწყებული ნულოვანი ინდექსის მქონე ელემენტიდან. ქვემოთ მოყვანილია პროგრამა, რომელშიც სრულდება მასივის ინიციალიზება. პროგრამა პოულობს masivi მასივის მაქსიმალურ და მინიმალურ ელემენტებს.

```
{
//   პროგრამა 4.2
//   პროგრამაში ხდება მასივის ინიციალიზება და
//   მისი მინიმალური და მაქსიმალური ელემენტების პოვნა
//   masivi მასივის ინიციალიზება
int[] masivi = new int[] { 2, -5, 50, 15, -20, 25, 12, -1, 120, 10 };
int indexi, max, min;

max = min = masivi[0];
for ( indexi = 1; indexi < masivi.Length; indexi++ )
{
    if ( masivi[indexi] > max ) max = masivi[indexi];
    if ( masivi[indexi] < min ) min = masivi[indexi];
}
label1.Text = " მაქსიმალური მნიშვნელობა = " + max.ToString() +
    "\n მინიმალური მნიშვნელობა = " + min.ToString();
}
```

ჩვენ განვიხილეთ მთელი რიცხვა მასივის მაგალითი. ბუნებრივია, შეგვიძლია შევქმნათ წილადი, ლოგიკური და ა.შ. ტიპის მასივიც. მაგალითად,

```
double[] wiladi_masivi = new double[5] { 1.2, 3.9, 4.7, 6.0, 8.6 };
```

```
bool[] logikuri_masivi = new bool[3] { true, false, true };
```

მასივის ინდექსის მნიშვნელობა არ უნდა გასცდეს ამ მასივის ინდექსის ზღვრულ

მნიშვნელობებს, წინააღმდეგ შემთხვევაში, აღიძვრება შესაბამისი შეცდომა.

მასივზე მიმართვის მინიჭება.

როცა მიმართვითი ტიპის ერთი ცვლადის მნიშვნელობა, რომელიც მასივს მიმართავს, ენიჭება ასეთივე ტიპის მეორე ცვლადს, მაშინ ამ მეორე ცვლადს ენიჭება მიმართვა იმავე მასივზე, რომელსაც პირველი ცვლადი მიმართავს. ამ დროს არ იქმნება მასივის ასლი და არ მოხდება ერთი მასივის გადაწერა მეორეში. მოყვანილ პროგრამაში ხდება მასივზე მიმართვის მინიჭების დემონსტრირება.

```
{
//   პროგრამა 4.3
//   პროგრამაში ხდება მასივზე მიმართვის მინიჭების დემონსტრირება
//   მასივების ინიციალიზება
label1.Text = ""; label2.Text = ""; label3.Text = ""; label4.Text = "";
int[] masivi1 = new int[] {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};
int[] masivi2 = new int[] {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
int indexi;

for ( indexi = 0; indexi < masivi1.Length; indexi++ )
    label1.Text += masivi1[indexi].ToString() + " ";
for ( indexi = 0; indexi < masivi2.Length; indexi++ )
    label2.Text += masivi2[indexi].ToString() + " ";
//   masivi2 ცვლადს ენიჭება masivi1 მასივზე მიმართვა, რის შედეგად ორივე ცვლადი
//   ერთსა და იმავე მასივს მიმართავს
masivi2 = masivi1;
//   ეკრანზე გამოგვაქვს masivi2
for ( indexi = 0; indexi < masivi2.Length; indexi++ )
    label3.Text += masivi2[indexi].ToString() + " ";
//   ამ მინიჭების შედეგად masivi1[5] ელემენტშიც 55 ჩაიწერება
masivi2[5] = 55;
//   ეკრანზე გამოგვაქვს masivi1
for ( indexi = 0; indexi < masivi1.Length; indexi++ )
    label4.Text += masivi1[indexi].ToString() + " ";
}
```

ორგანზომილებიანი მასივები

ორგანზომილებიანი მასივებში კონკრეტული ელემენტის განლაგება განისაზღვრება ორი ინდექსით. მასივის გამოცხადების დროს პირველი ინდექსი მიუთითებს სტრიქონების რაოდენობას, მეორე კი - სვეტების რაოდენობას. მაგალითად, ორგანზომილებიანი მთელი რიცხვების მასივი `cxrili` ზომით 5×4 შეგვიძლია შემდეგნაირად გამოვაცხადოთ:

```
int[,] cxrili = new int[5,4];
```

მასივის გამოცხადებისას ორივე განზომილება ერთმანეთისაგან მძიმით გამოიყოფა. ოპერატორის პირველ ნაწილში ფრჩხილებში მითითებული მძიმე მიუთითებს, რომ იქმნება მიმართვითი ტიპის ცვლადი ორგანზომილებიანი მასივისათვის. `new` ოპერატორი დინამიურად გამოყოფს მეხსიერებას ორგანზომილებიანი მასივისათვის. ოპერატორის მეორე ნაწილში რიცხვი 5 განსაზღვრავს სტრიქონების რაოდენობას, რიცხვი 4 - კი სვეტების რაოდენობას. მეხსიერება გამოიყოფა $5 \times 4 = 20$ რიცხვისათვის. რადგან მთელი რიცხვი იკავებს 4 ბაიტს, ამიტომ `cxrili` მასივისთვის მეხსიერებაში გამოიყოფა $4 \times 5 \times 4 = 80$ ბაიტი.

ორგანზომილებიანი მასივის კონკრეტულ ელემენტთან მიმართვისათვის აუცილებელია ორი ინდექსის მითითება, მაგალითად,

```
cxrili[3,2] = 15;
```

აქ რიცხვი 3 არის სტრიქონის ნომერი, რიცხვი 2 - კი სვეტის ნომერი.

ქვემოთ მოყვანილია პროგრამა, რომელიც ჯერ ავსებს ორგანზომილებიან მასივს მთელი რიცხვებით, შემდეგ კი ისინი label კომპონენტში გამოაქვს:

```
{  
//   პროგრამა 4.4  
//   პროგრამაში ხდება ორგანზომილებიანი მასივის შევსება რიცხვებით და  
//   მათი label კომპონენტში გამოტანა  
label1.Text = "";  
int[,] cxrili = new int[5,4];  
int striqoni, sveti;  
  
for ( striqoni = 0; striqoni < 5; striqoni++ )  
    for ( sveti = 0; sveti < 4; sveti++ )  
        cxrili[striqoni,sveti] = ( striqoni * 4 ) + ( sveti + 1 );  
for ( striqoni = 0; striqoni < 5; striqoni++ )  
    {  
        for ( sveti = 0; sveti < 4; sveti++ )  
            label1.Text += cxrili[striqoni,sveti].ToString() + " ";  
        label1.Text += '\n';  
    }  
}
```

ორგანზომილებიანი მასივების ინიციალიზება

ორგანზომილებიანი მასივების ინიციალიზაციისათვის თითოეული განზომილების მნიშვნელობების სია უნდა მოვათავსოთ ცალკე ბლოკში, რომელიც, თავის მხრივ, ფიგურულ ფრჩხილებში უნდა იყოს მოთავსებული. მაგალითად,

```
int[,] cxrili = new int[,] {  
    {1,1},  
    {2,4},  
    {3,9},  
    {4,16},  
    {5,25}  
};
```

აქ ყოველი შიდა ბლოკი მნიშვნელობას ანიჭებს შესაბამისი სტრიქონის ელემენტებს. სტრიქონის ფარგლებში პირველი მნიშვნელობა მოთავსებული იქნება სტრიქონის ნულოვან ელემენტში, მეორე - პირველ ელემენტში და ა.შ. ბლოკები ერთმანეთისაგან მძიმეებით გამოიყოფა, ხოლო დამხურავი ფიგურული ფრჩხილის შემდეგ მოთავსებულია წერტილ-მძიმე. ქვემოთ მოყვანილ პროგრამაში ხდება ორგანზომილებიანი მასივის ინიციალიზება და მისი ელემენტების ჯამის გამოთვლა.

```
{  
//   პროგრამა 4.5  
//   პროგრამა გამოთვლის ორგანზომილებიანი მასივის ელემენტების ჯამს  
label1.Text = "";  
int striqoni, sveti, jami = 0;
```

```

//      masivi მასივის ინიციალიზება
int[,] masivi = new int[,] {
                {1,1},
                {2,4},
                {3,9},
                {4,16},
                {5,25}
            };
//      მასივის ელემენტებისა და მათი ჯამის ეკრანზე გამოტანა
for ( striqoni = 0; striqoni < 5; striqoni++ )
{
    for ( sveti = 0; sveti < 2; sveti++ )
        {
            jami += masivi[striqoni,sveti];
            label1.Text += masivi[striqoni,sveti].ToString() + " ";
        }
    label1.Text += '\n';
}
label2.Text = jami.ToString();
}

```

გაუსწორებელი ("დაკბილული") მასივები

აქამდე ჩვენ განვიხილავდით ორგანზომილებიან მასივებს, რომლებსაც სხვანაირად სწორკუთხა მასივები ეწოდება. სწორკუთხა მასივი არის ორგანზომილებიანი მასივი, რომელშიც სტრიქონების სიგრძე ერთნაირია. C# ენაში შეგვიძლია შევექმნათ ორგანზომილებიანი მასივის სპეციალური ტიპი, რომელსაც გაუსწორებელი მასივი (არასწორკუთხა ან "დაკბილული" მასივი) ეწოდება. ესაა გარე მასივი, რომელიც შედგება სხვადასხვა სიგრძის შიდა მასივებისაგან. შედეგად, გაუსწორებელი მასივი შეგვიძლია გამოვიყენოთ ისეთი ცხრილის შესაქმნელად, რომელიც სხვადასხვა სიგრძის სტრიქონებს შეიცავს.

ორგანზომილებიანი გაუსწორებელი მასივის გამოცხადების სინტაქსია:

ტიპი [[] მასივის_სახელი = new ტიპი[ზომა] [];

აქ ზომა არის მასივში სტრიქონების რაოდენობა. სვეტების რაოდენობა არ ეთითება და მეხსიერება მათთვის არ გამოიყოფა. თითოეული სტრიქონის შესაქმნელად აუცილებელია ცალკე new ოპერატორის შესრულება. ასეთი ტექნოლოგია იძლევა სხვადასხვა სიგრძის სტრიქონების განსაზღვრის საშუალებას. მაგალითად, პროგრამის ქვემოთ მოყვანილ ფრაგმენტში masivi_g მასივის გამოცხადებისას მეხსიერება მხოლოდ გარე მასივისათვის გამოიყოფა, შემდეგ კი მომდევნო სამი ოპერატორის შესრულების შედეგად მეხსიერება ცალცალკე გამოეყოფა თითოეულ შიდა მასივს:

```

int[ ][ ] masivi_g = new int[3][ ];
masivi_g[0] = new int[2];
masivi_g[1] = new int[3];
masivi_g[2] = new int[4];

```

შედეგად, masivi_g შემდეგ სახეს მიიღებს:

```

masivi_g[0][0]          masivi_g[0][1]

```



```

masivi_g[1][0]          masivi_g[1][1]          masivi_g[1][2]
masivi_g[2][0]          masivi_g[2][1]          masivi_g[2][2]          masivi_g[2][3]

```

გაუსწორებელი მასივის შექმნის შემდეგ მის ელემენტებთან მიმართვა შესაძლებელია თითოეული ინდექსის ცალკე კვადრატულ ფრჩხილებში მითითებით. მაგალითად, masivi_g[1][2] = 15;

ქვემოთ მოყვანილ პროგრამაში ხდება გაუსწორებელი ორგანზომილებიანი მასივის შევსება და label2 კომპონენტში გამოტანა.

```

{
//   პროგრამა 4.6
//   პროგრამაში ხდება გაუსწორებელ ორგანზომილებიან მასივთან
//   მუშაობის დემონსტრირება
label1.Text = "";
int [][] g_masivi = new int[5][];
int striqoni, sveti;

label1.Text = "";
g_masivi[0] = new int[5];
g_masivi[1] = new int[5];
g_masivi[2] = new int[5];
g_masivi[3] = new int[2];
g_masivi[4] = new int[2];
//   g_masivi მასივის შევსება
for ( striqoni = 0; striqoni < 3; striqoni++ )
    for ( sveti = 0; sveti < 5; sveti++ )
        g_masivi[striqoni][sveti] = striqoni + sveti;
for ( striqoni = 3; striqoni < 5; striqoni++ )
    for ( sveti = 0; sveti < 2; sveti++ )
        g_masivi[striqoni][sveti] = striqoni + sveti;
//   g_masivi მასივის label კომპონენტში გამოტანა
for ( striqoni = 0; striqoni < 3; striqoni++ )
{
    for ( sveti = 0; sveti < 5; sveti++ )
        label1.Text += g_masivi[striqoni][sveti].ToString() + " ";
    label1.Text += '\n';
}
for ( striqoni = 3; striqoni < 5; striqoni++ )
{
    for ( sveti = 0; sveti < 2; sveti++ )
        label1.Text += g_masivi[striqoni][sveti].ToString() + " ";
    label1.Text += '\n';
}
}
}

```

გაუსწორებელი მასივები იშვიათად გამოიყენება. მაგრამ, რიგ შემთხვევებში მათი გამოყენება საკმაოდ ეფექტურია. მაგალითად, როცა გვაქვს დიდი ზომის ორგანზომილებიანი მასივი, რომელიც ნაწილობრივ იქნება შევსებული.

foreach ციკლი

foreach ციკლი გამოიყენება კოლექციის ელემენტების დასამუშავებლად. კოლექცია არის ობიექტების ჯგუფი. C# ენაში განსაზღვრულია კოლექციების რამდენიმე ტიპი, ერთ-ერთი მათგანია მასივი. foreach ციკლის სინტაქსია:

```
foreach ( ტიპი ცვლადის_სახელი in კოლექცია )  
{  
    ციკლის ტანი  
}
```

აქ ტიპი არის იტერაციული ცვლადის ტიპი და უნდა ემთხვეოდეს კოლექციის ტიპს. ცვლადის_სახელი არის იტერაციული ცვლადის სახელი. იტერაციული ცვლადი რიგ-რიგობით იღებს კოლექციის ელემენტების მნიშვნელობებს. კოლექცია შეიძლება იყოს მასივი, სტრიქონი და ა.შ. იტერაციული ცვლადი მისაწვდომია მხოლოდ წაკითხვის რეჟიმში, ამიტომ, ვერ შევძლებთ მასივის შეცვლას იტერაციული ცვლადისთვის ახალი მნიშვნელობის მინიჭების გზით.

ქვემოთ მოყვანილია foreach ციკლის მაგალითი. თავიდან იქმნება მთელი რიცხვა მასივი და სრულდება მისი ინიციალიზება. შემდეგ, ეს მნიშვნელობები გამოიყენება label კომპონენტში და გამოითვლება მათი ჯამი.

```
{  
    // პროგრამა 4.9  
    // პროგრამაში გამოითვლება მასივის ელემენტების ჯამი foreach ციკლის გამოყენებით  
    label1.Text = ""; label2.Text = "";  
    int[] masivi = new int[] { 9, -1, 101, 8, -9, 56, 3, -2, 87, 81 };  
    int jami = 0;  
  
    foreach ( int cvladi in masivi )  
        jami += cvladi;  
    label1.Text = jami.ToString();  
  
    foreach ( int cvladi in masivi )  
        label2.Text += cvladi.ToString() + " ";  
}
```

როგორც ვხედავთ მასივის ელემენტების დამუშავება ხდება მიმდევრობით, დაწყებული პირველი ელემენტიდან, ინდექსის გამოყენების გარეშე.

foreach ციკლის საშუალებით მასივი შეიძლება დავამუშავოთ მხოლოდ დასაწყისიდან ბოლოსკენ. ქვემოთ მოყვანილია მასივის მაქსიმალური და მინიმალური ელემენტების პოვნის პროგრამა foreach ციკლის გამოყენებით.

```
{  
    // პროგრამა 4.10  
    // პროგრამაში ხდება მასივის მაქსიმალური და მინიმალური ელემენტების პოვნა  
    label1.Text = "";  
    int max, min;  
    int[] masivi = new int[] { 9, -1, 101, 8, -9, 56, 3, -2, 87, 81 };  
  
    max = min = masivi[0];  
    // მასივის ელემენტების ეკრანზე გამოტანა  
    foreach ( int ricxvi in masivi )  
        label1.Text += ricxvi.ToString() + " ";  
}
```

```
foreach ( int ricxvi in masivi )
{
    if ( ricxvi > max ) max = ricxvi;
    if ( ricxvi < min ) min = ricxvi;
}
label2.Text = max.ToString();
label3.Text = min.ToString();
}
```

ქვემოთ მოყვანილ პროგრამაში foreach ციკლი გამოიყენება ორგანზომილებიანი მასივის ელემენტების ჯამის გამოსათვლელად.

```
{
// პროგრამა 4.11
// პროგრამაში ხდება ორგანზომილებიანი მასივის ელემენტების ჯამის გამოთვლა
label1.Text = "";
int jami = 0;
int[,] cxrili = new int[,] {
    { 1, 2, -3 },
    { -4, 5, 6 },
    { 7, -8, 9 },
};
foreach ( int ricxvi in cxrili )
{
    label1.Text += ricxvi.ToString() + " ";
    jami += ricxvi;
}
label2.Text = jami.ToString();
}
```

ბიტობრივი ოპერატორები

ბიტობრივი (თანრიგობრივი) ოპერატორების ოპერანდები მთელი რიცხვებია. ეს ოპერატორები არ შეიძლება გამოვიყენოთ bool, float ან double ტიპის მნიშვნელობების, აგრეთვე, კლასის ობიექტების მიმართ. ასეთ ოპერატორებს ბიტობრივი ეწოდება იმიტომ, რომ გამოიყენებიან მთელი რიცხვების ბიტების შესამოწმებლად, დასაყენებლად (1 მდგომარეობაში გადაყვანა) და ძვრისათვის. ბიტობრივი ოპერაციები ხშირად გამოიყენება სისტემურ დონეზე პროგრამირებისათვის, მაგალითად, როცა საჭიროა ინფორმაციის მიღება მოწყობილობის მდგომარეობის შესახებ და ა.შ. ბიტობრივი ოპერაციები მოყვანილია ცხრილში 4.1.

ცხრილი 4.1. ბიტობრივი ოპერაციები

ოპერატორი	აღწერა
&	ბიტობრივი ოპერატორი AND
	ბიტობრივი ოპერატორი OR

^	ბიტობრივი ოპერატორი XOR
>>	მარჯვნივ ძვრის ოპერატორი
<<	მარცხნივ ძვრის ოპერატორი
~	უნარული ოპერატორი NOT

&, |, ^ და ~ ბიტობრივი ოპერატორები

&, |, ^ და ~ ბიტობრივი ოპერატორები ისეთივე ოპერაციებს ასრულებენ, როგორსაც მათი ბულის ეკვივალენტები. განსხვავება იმაშია, რომ ბიტობრივი ოპერატორები ოპერანდებზე ზემოქმედებენ ბიტების დონეზე. ცხრილში 4.2 მოყვანილია თითოეული ოპერაციის შესრულების შედეგები.

ცხრილი 4.2. ბიტობრივი ოპერაციები.

b1	b2	b1&b2	b1 b2	b1 ^ b2	~b1
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

ბიტობრივი & ოპერაცია შეგვიძლია გამოვიყენოთ როგორც ბიტების განულების (ჩამოგდების) საშუალება. ეს იმას ნიშნავს, რომ თუ რომელიმე ოპერანდის ერთ-ერთ ბიტს აქვს მნიშვნელობა 0, მაშინ შედეგის შესაბამის ბიტსაც ექნება მნიშვნელობა 0. მაგალითად,

მოყვანილ პროგრამაში ნაჩვენებია & ოპერატორის გამოყენების მაგალითი. პროგრამა ქვედა რეგისტრის სიმბოლოებს გარდაქმნის ზედა რეგისტრის სიმბოლოებად მეექვსე ბიტის განულების გზით. Unicode/ASCII სიმბოლოების სტანდარტულ ნაკრებში ქვედა რეგისტრის სიმბოლოების მნიშვნელობა მეტია ზედა რეგისტრის სიმბოლოების შესაბამის მნიშვნელობაზე 32 ერთეულით ($32_{10}=0010000_2$). შედეგად, ქვედა რეგისტრის სიმბოლოების გადასაყვანად ზედა რეგისტრში საკმარისია მეექვსე ბიტის განულება.

```

{
//   პროგრამა 4.12
//   პროგრამაში ხდება ქვედა რეგისტრის სიმბოლოების გარდაქმნა ზედა
//   რეგისტრის სიმბოლოებად
label1.Text = "";
char simbolo;

for ( int indexi = 0; indexi < 10; indexi++ )
{
    simbolo = (char) ( 'a' + indexi );
    label1.Text += simbolo + " - ";
    simbolo = (char) ( simbolo & 65503 );
    label1.Text += simbolo + "\n";
}

```

```
}  
}
```

რიცხვს 65503 აქვს შემდეგი ორბაიტანი ორობითი წარმოდგენა 1111 1111 1101 1111. შედეგად, & ოპერაცია უცვლელად ტოვებს simbolo ცვლადის ყველა ბიტს მეექვსე ბიტის გარდა, რომელიც განულდება. & ოპერაცია შეგვიძლია გამოვიყენოთ, როცა საჭიროა განისაზღვროს ბიტი დაყენებულია (1) თუ ჩამოგდებული (0). მაგალითად, ქვემოთ მოყვანილ ოპერატორში მოწმდება status ცვლადის მეოთხე ბიტის მნიშვნელობა:

```
if ( status & 8 == true ) label2.Text = "მეოთხე ბიტი დაყენებულია";
```

რიცხვი 8 გამოიყენება იმიტომ, რომ მის ორობით წარმოდგენაში დაყენებულია მხოლოდ მეოთხე ბიტი (0000 1000). შედეგად, if ოპერატორი true შედეგს მხოლოდ მაშინ გასცემს, როცა status ცვლადის მნიშვნელობაში მეოთხე ბიტი იქნება დაყენებული. ქვემოთ მოყვანილ პროგრამაში ასეთი მიდგომა გამოიყენება ეკრანზე byte ტიპის მნიშვნელობის ორობითი წარმოდგენის გამოსატანად.

```
{  
// პროგრამა 4.13  
// ათობითი რიცხვის გარდაქმნა ორობით რიცხვად  
label1.Text = "";  
int indeqsi;  
byte ricxvi = Convert.ToByte(textBox1.Text);  
  
for ( indeqsi = 128; indeqsi > 0; indeqsi /= 2 )  
{  
    if ( ( ricxvi & indeqsi ) != 0 ) label1.Text += "1";  
    else label1.Text += "0";  
}  
}
```

for ციკლში & ოპერატორის გამოყენებით მოწმდება ricxvi ცვლადის თითოეული ბიტი. თუ ის დაყენებულია, მაშინ label2 კომპონენტში გამოჩნდება 1, თუ ჩამოგდებულია - 0.

& ოპერატორის საწინააღმდეგოდ | ოპერატორი შეგვიძლია გამოვიყენოთ ბიტების დასაყენებლად. თუ ერთ-ერთ ოპერანდში ბიტი დაყენებულია, მაშინ შედეგში შესაბამისი ბიტი იქნება დაყენებული. მაგალითად,

ზედა რეგისტრის სიმბოლოების გარდასაქმნელად ქვედა რეგისტრის სიმბოლოებად გამოვიყენოთ | ოპერატორი, როგორც ეს ქვემოთაა ნაჩვენები.

```
{  
// პროგრამა 4.14  
// პროგრამაში ხდება ზედა რეგისტრის სიმბოლოების გარდაქმნა  
// ქვედა რეგისტრის სიმბოლოებად  
label1.Text = "";  
char simbolo;  
  
for ( int indeqsi = 0; indeqsi < 10; indeqsi++ )  
{  
    simbolo = (char) ( 'A' + indeqsi );
```

```

    label1.Text += simbolo + " - ";
// ამ ოპერატორის შესრულება აყენებს მეექვსე ბიტს
// შედეგად simbolo ცვლადში ჩაიწერება ქვედა რეგისტრის სიმბოლო
    simbolo = (char) ( simbolo | 32 );
    label1.Text += simbolo + "\n";
}
}

```

| ოპერაცია გამოიყენება თითოეული სიმბოლოს მნიშვნელობის მიმართ და 32 მნიშვნელობის მიმართ, რომლის ორობითი წარმოდგენაა 0000 0000 0010 0000. ამ ორობით წარმოდგენაში დაყენებულია მხოლოდ მეექვსე ბიტი. თუ | ოპერაციის ერთი ოპერანდია რიცხვი 32 და მეორე ოპერანდი - ნებისმიერი რიცხვი, მაშინ შედეგად მივიღებთ მნიშვნელობას, რომელშიც დაყენებული იქნება მეექვსე ბიტი. დანარჩენი ბიტები დარჩება უცვლელი.

^ ოპერაციის შესრულების შედეგად ბიტის დაყენება მოხდება მხოლოდ მაშინ, როცა ორივე ბიტი განსხვავებულია. მაგალითად,

^ ოპერატორს საკმაოდ საინტერესო თვისება აქვს, რომელიც საშუალებას გვაძლევს ის გამოვიყენოთ შეტყობინების ან რაიმე ინფორმაციის დასაშიფრად. თუ ამ ოპერატორს გამოვიყენებთ x და y ცვლადების მიმართ და შემდეგ მიღებული შედეგისა და y ცვლადის მიმართ, მაშინ მივიღებთ x მნიშვნელობას, ე.ი.

$R1 = x \wedge y;$

$R2 = R1 \wedge y;$

ოპერატორების შესრულების შედეგად R2 ცვლადს მიენიჭება x ცვლადის მნიშვნელობა. ქვემოთ მოყვანილ პროგრამაში ეს პრინციპია გამოყენებული შეტყობინების შიფრაციისა და დეშიფრაციისათვის. პირველად ^ ოპერატორი გამოიყენება შიფრაციისათვის, მეორედ კი - დეშიფრაციისათვის. შედეგად, მივიღებთ საწყის ტექსტს. შიფრაციისას რაიმე მთელი რიცხვი შეგვიძლია გამოვიყენოთ როგორც გასაღები. ქვემოთ მოყვანილ პროგრამაში ხდება შიფრაციისა და დეშიფრაციის დემონსტრირება.

```

{
// პროგრამა 4.15
// პროგრამაში ხდება შიფრაციისა და დეშიფრაციის დემონსტრირება
string msg = "შეტყობინების შიფრაცია";
string encmsg = "";
string decmsg = "";
int key = 55;

label1.Text = "დასაშიფრი შეტყობინება: " + msg;
// შეტყობინების შიფრაცია
for ( int indeqsi = 0; indeqsi < msg.Length; indeqsi++ )
    encmsg += (char) ( msg[indeqsi] ^ key );
label2.Text = "დაშიფრული შეტყობინება: " + encmsg;
// შეტყობინების დეშიფრაცია
for ( int indeqsi = 0; indeqsi < msg.Length; indeqsi++ )
    decmsg += (char) ( encmsg[indeqsi] ^ key );
label3.Text = "გაშიფრული შეტყობინება: " + decmsg;

```

```
}
```

უნარული ~ ოპერაცია ცვლის ოპერანდის ყველა ბიტის მნიშვნელობას საწინააღმდეგოთი. მაგალითად, თუ A = 1001 0011, მაშინ ~A ოპერაციის შედეგი იქნება 0110 1100.

ქვემოთ მოყვანილ პროგრამაში ხდება ~ ოპერატორის გამოყენების დემონსტრირება. ეკრანზე ჯერ ხდება რიცხვი 35-ის (0010 0011) ორობითი წარმოდგენის გამოტანა, შემდეგ კი ამ რიცხვის მიმართ ~ ოპერაციის გამოყენების შედეგის გამოტანა (1101 1100).

```
{
```

```
// პროგრამა 4.16
```

```
// პროგრამაში ხდება ~ ოპერატორის მუშაობის დემონსტრირება
```

```
label1.Text = ""; label2.Text = "";
```

```
sbyte ricxvi = 35;
```

```
// ricxvi ცვლადის ორობითი წარმოდგენის label კომპონენტში გამოტანა
```

```
for ( int indeqsi = 128; indeqsi > 0; indeqsi /= 2 )
```

```
    if ( ( ricxvi & indeqsi ) != 0 ) label1.Text += " 1";
```

```
        else label1.Text += " 0";
```

```
// ricxvi ცვლადის ორობითი წარმოდგენის ინვერსია და label კომპონენტში გამოტანა
```

```
ricxvi = (sbyte) ~ricxvi;
```

```
for ( int indeqsi = 128; indeqsi > 0; indeqsi /= 2 )
```

```
    if ( ( ricxvi & indeqsi ) != 0 ) label2.Text += " 1";
```

```
        else label2.Text += " 0";
```

```
}
```

ყველა ორობითი ბიტობრივი ოპერატორი შეგვიძლია ჩავწეროთ შედგენილი ბიტობრივი ოპერატორების სახით. მაგალითად,

```
x ^= 127;
```

```
y &= 63;
```

```
z |= 31;
```

ძვრის ოპერატორები

არსებობს ოპერანდების მარცხნივ და მარჯვნივ ძვრის ოპერატორები. მათი სინტაქსია:

ცვლადის_სახელი << ბიტების_რაოდენობა

ცვლადის_სახელი >> ბიტების_რაოდენობა

მარცხნივ ძვრა (<<) იწვევს ყველა ბიტის გადაადგილებას ერთი ბიტით მარცხნივ, ამასთან, თავისუფლდება უმცროსი ბიტები, რომლებიც ნულებით ივსება, ხოლო უფროსი ბიტების შესაბამისი რაოდენობა იკარგება. მარჯვნივ ძვრა (>>) იწვევს ყველა ბიტის ერთი ბიტით მარჯვნივ გადაადგილებას, ამასთან, თუ მარჯვნივ იძვრის უნიშნო მნიშვნელობის ბიტები, მაშინ უფროსი თანრიგები ნულებით შეივსება და უმცროსი თანრიგების შესაბამისი რაოდენობა იკარგება. ნიშნის მნიშვნელობის მარჯვნივ ძვრისას ხდება ნიშნის ბიტის შენახვა. ნიშნის დადებითი რიცხვებისთვის უფროსი ბიტი არის 0, უარყოფითი რიცხვებისათვის - 1.

ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია ძვრის ოპერაციების გამოყენების მაგალითი.

```
{
```

```
// პროგრამა 4.17
```

```
// პროგრამაში ხდება ძვრის ოპერატორებთან მუშაობის დემონსტრირება
```

```
label1.Text = ""; label2.Text = "";
```

```
int ricxvi = 1;
```

```
label1.Text = ""; label2.Text = "";
```

```
// ricxvi ცვლადის ორობითი წარმოდგენის label კომპონენტში გამოტანა
for ( int i = 0; i < 8; i++ )
{
    for ( int j = 128; j > 0; j /= 2 )
        if ( ( ricxvi & j ) != 0 ) label1.Text += "1";
        else label1.Text += "0";
    label1.Text += '\n';
    ricxvi = ricxvi << 1; // ძვრა მარცხნივ ერთი ბიტით
}
ricxvi = 128;
// ricxvi ცვლადის ორობითი წარმოდგენის label კომპონენტში გამოტანა
for ( int i = 0; i < 8; i++ )
{
    for ( int j = 128; j > 0; j /= 2 )
        if ( ( ricxvi & j ) != 0 ) label2.Text += "1";
        else label2.Text += "0";
    label2.Text += '\n';
    ricxvi = ricxvi >> 1; // ძვრა მარჯვნივ ერთი ბიტით
}
}
```

ჩამოთვლები

ჩამოთვლები წარმოადგენენ სახელდებულ მთელრიცხვას მუდმივების ნაკრებს, რომლებიც განსაზღვრავენ ცვლადის ყველა დასაშვებ მნიშვნელობას მოცემული ტიპისთვის.

ჩამოთვლადი ტიპის განსაზღვრისათვის გამოიყენება enum საკვანძო სიტყვა. მისი სინტაქსია:

```
enum ჩამოთვლის_სახელი { მუდმივების_სია };
```

ქვემოთ მოყვანილია xili ჩამოთვლის განსაზღვრის მაგალითი:

```
enum xili { vaSli, msxali, leRvi, nesvi, yurZeni };
```

ჩამოთვლასთან მუშაობისას მთავარია ის ფაქტი, რომ თითოეულ მუდმივას შეესაბამება მთელრიცხვა მნიშვნელობა. შედეგად, მუდმივების გამოყენების სფერო ემთხვევა მთელრიცხვა მნიშვნელობების (int ტიპი) გამოყენების სფეროს. მთელრიცხვა მნიშვნელობა თითოეული მუდმივასთვის მეტია წინა მუდმივას მნიშვნელობაზე. პირველი მუდმივას მნიშვნელობა ნულის ტოლია.

ჩამოთვლის ელემენტთან მიმართვისათვის ჯერ უნდა მივუთითოთ ჩამოთვლის სახელი, შემდეგ წერტილი (.) და ელემენტის სახელი.

ქვემოთ მოყვანილია პროგრამის მაგალითი, რომელიც ახდენს xili ჩამოთვლასთან მუშაობის დემონსტრირებას.

```
{
// პროგრამა 4.18
// პროგრამაში ხდება ჩამოთვლასთან მუშაობის დემონსტრირება
// xili ჩამოთვლის გამოცხადება
enum xili { vaSli, msxali, leRvi, nesvi, yurZeni };
private void button1_Click(object sender, System.EventArgs e)
label1.Text = "";
```



```
// saxelebi სტრიქონების მასივის გამოცხადება
string[ ] saxelebi = { "ვაშლი", "მსხალი", "ლეღვი", "ნესვი", "ყურძენი" };
xili elementi; // ცვლადის გამოცხადება ჩამოთვლის დამუშავებისათვის

// elementi ცვლადის გამოყენება ჩამოთვლის დამუშავების ციკლისათვის
for ( elementi = xili.vaSli; elementi <= xili.yurZeni; elementi++ )
    label1.Text += saxelebi[(int)elementi] + "-ის მნიშვნელობაა - " + (int)elementi + '\n';
}
```

პროგრამაში for ციკლის მართვა ხორციელდება elementi ცვლადის საშუალებით, რომელსაც აქვს xili ტიპი. რადგან ჩამოთვლას აქვს მთელრიცხვა ტიპი, ამიტომ, ჩამოთვლის მნიშვნელობების მინიჭება შეიძლება იქ, სადაც შეიძლება მთელრიცხვა მნიშვნელობების გამოყენება. მაგრამ, ჩამოთვლის მნიშვნელობების გამოყენებისას saxelebi მასივის ინდექსაციის მიზნით საჭირო ხდება ტიპის გარდაქმნა.

ჩამოთვლების ინიციალიზება

ინიციალიზატორის საშუალებით შეგვიძლია მივუთითოთ ერთი ან მეტი მუდმივას მნიშვნელობა. ამისათვის, თითოეული მუდმივას შემდეგ უნდა მოვათავსოთ მინიჭების ოპერატორი და მთელრიცხვა მნიშვნელობა. ინიციალიზატორის მიერ განსაზღვრული მნიშვნელობები უნდა იყოს მეტი ინიციალიზაციის წინა მნიშვნელობაზე. მაგალითად, მოყვანილ გამოცხადებაში nesvi მუდმივას ენიჭება მნიშვნელობა 55:

```
enum xili { vaSli, msxali, leRvi, nesvi = 55, yurZeni } ;
```

ამის შემდეგ მუდმივებს ექნებათ შემდეგი მნიშვნელობები:

vaSli	0
msxali	1
leRvi	2
nesvi	55
yurZeni	56

თავი 5. კლასები, ინკაფსულაცია, მეთოდები

კლასის გამოცხადება

კლასი არის ობიექტზე ორიენტირებული პროგრამირების ძირითადი სტრუქტურა. მის საფუძველზე იქმნება ობიექტები. ობიექტი შეიძლება იყოს ავტომობილი, თვითმფრინავი, მაღაზია, გეომეტრიული ფიგურა და ა.შ. ობიექტები წარმოადგენენ კლასის ეგზემპლარებს. განსხვავება კლასსა და ობიექტს შორის იმაში მდგომარეობს, რომ კლასი არის ლოგიკური აბსტრაქცია, ობიექტი კი - ამ კლასის კონკრეტული რეალიზება კომპიუტერის მეხსიერებაში.

კლასში შემავალ მეთოდებსა და ცვლადებს (ველებს) კლასის წევრები (ელემენტები) ეწოდებათ. C# ენაში არსებობს კლასის წევრების რამდენიმე სახესხვაობა: ეგზემპლარის ცვლადები, სტატიკური ცვლადები, მუდმივები, მეთოდები, კონსტრუქტორები, დესტრუქტორები, ინდექსატორები, მოვლენები, ოპერატორები და თვისებები. ამ ეტაპზე განვიხილავთ კლასის ძირითად ელემენტებს - ცვლადებს და მეთოდებს.

კლასის სინტაქსი, რომელიც შეიცავს მხოლოდ ცვლადებსა და მეთოდებს ასეთია:

```
მიმართვის_მოდულიკატორი class კლასის_სახელი
{
// ეგზემპლარის ცვლადების გამოცხადება
მიმართვის_მოდულიკატორი ტიპი ცვლადის_სახელი1;
// ...
მიმართვის_მოდულიკატორი ტიპი ცვლადის_სახელიN;
// მეთოდების გამოცხადება
მიმართვის_მოდულიკატორი შედეგის_ტიპი მეთოდის_სახელი1(პარამეტრები)
{
მეთოდის კოდი
}
// ...
მიმართვის_მოდულიკატორი შედეგის_ტიპი მეთოდის_სახელიN(პარამეტრები)
{
მეთოდის კოდი
}
}
```

class საკვანძო სიტყვა იწყებს კლასის გამოცხადებას. მას კლასის სახელი მოსდევს. *მიმართვის_მოდულიკატორი* იღებს private (პრივატული) ან public (ლია) მნიშვნელობებს. თითოეული ცვლადისა და მეთოდის გამოცხადება იწყება ამ სიტყვებიდან ერთ-ერთით. ისინი განსაზღვრავენ თუ როგორ უნდა მოხდეს ცვლადთან ან მეთოდთან მიმართვა. private მოდიფიკატორი მიუთითებს, რომ კლასის წევრთან მიმართვა შეუძლიათ მხოლოდ ამ კლასის წევრებს. public მოდიფიკატორი მიუთითებს, რომ კლასის წევრთან მიმართვა შესაძლებელია კლასის გარეთ აღწერილი კოდიდან. თუ მოდიფიკატორი მითითებული არ არის, მაშინ იგულისხმება private. მოდიფიკატორის შემდეგ ეთითება ცვლადის ტიპი და სახელი. *შედეგის_ტიპი* არის მეთოდის მიერ გაცემული შედეგის ტიპი. მას მოსდევს მეთოდის სახელი და პარამეტრების სია.

კორექტულად დაწერილ პროგრამაში კლასი განსაზღვრავს მხოლოდ ერთ ლოგიკურ ერთეულს. მაგალითად, კლასი, რომელიც შეიცავს ინფორმაციას ავტომანქანების შესახებ, არ უნდა შეიცავდეს სხვა ინფორმაციას, მაგალითად, შენობების ან ცხოველების შესახებ.

ინკაფსულაცია

კლასი ორ ძირითად ფუნქციას ასრულებს, რომლებიც უზრუნველყოფენ მონაცემების ინკაფსულაციას. ჯერ ერთი, ის მონაცემებს აკავშირებს კოდთან, რომელიც მათზე მანიპულირებს. მეორეც, კლასი უზრუნველყოფს კლასის წევრებთან მიმართვის საშუალებებს. ინკაფსულაცია არის კლასის წევრებთან მიმართვის უფლებამოსილების გამიჯვნა.

როგორც აღვნიშნეთ, არსებობს კლასის წევრების ორი ძირითადი ტიპი - ღია (public) და დახურული (private). public ტიპის წევრებთან მიმართვა შეიძლება შესრულდეს ამ კლასის გარეთ განსაზღვრული კოდიდან. private ტიპის მქონე წევრებთან მიმართვა შეუძლიათ მხოლოდ ამავე კლასის მეთოდებს. ამ კლასის გარეთ განსაზღვრულ კოდს მხოლოდ ამავე კლასის ღია მეთოდების გამოყენებით შეუძლია მიმართოს ამ კლასის დახურულ წევრებს.

კლასის წევრებთან მიმართვის შეზღუდვა არის ობიექტზე ორიენტირებული პროგრამირების ძირითადი პრინციპი. ის ობიექტებს იცავს არასწორი გამოყენებისაგან. ვაძლევთ რა კლასის დახურულ წევრებთან მიმართვის უფლებას მხოლოდ კლასის შიგნით განსაზღვრულ მეთოდებს, ამით თავიდან ვიცილებთ მონაცემებისათვის არაკორექტული მნიშვნელობების მინიჭებას კლასის გარეთ განსაზღვრული კოდის მხრიდან.

კლასის წევრებთან მიმართვა ხორციელდება ოთხი მოდიფიკატორის გამოყენებით: public, private, protected და internal. თუ მოდიფიკატორი არ არის მითითებული, მაშინ იგულისხმება private.

შევქმნათ Samkutxedi კლასი და მასში მოვათავსოთ სამკუთხედის გვერდების ზომები. ქვემოთ მოყვანილია Samkutxedi კლასის პირველი ვერსია, რომელშიც განსაზღვრულია მხოლოდ სამი ცვლადი gverdi1, gverdi2 და gverdi3. კლასი არ შეიცავს არც ერთ მეთოდს.

```
class Samkutxedi
{
public int gverdi1;           // სამკუთხედის პირველი გვერდი
public int gverdi2;         // სამკუთხედის მეორე გვერდი
public int gverdi3;         // სამკუთხედის მესამე გვერდი
}
```

class სიტყვის გამოყენებით იქმნება მონაცემების ახალი ტიპი, რომლის სახელია Samkutxedi. მას გამოვიყენებთ Samkutxedi ტიპის მქონე ობიექტის გამოსაცხადებლად. უნდა გვახსოვდეს, რომ class სიტყვის გამოყენებით კლასის გამოცხადებისას ხდება მხოლოდ მისი აღწერა, მაგრამ ფიზიკურად ობიექტი ამ დროს არ იქმნება. Samkutxedi ტიპის ობიექტის შესაქმნელად უნდა გამოვიყენოთ new ოპერატორი:

```
Samkutxedi Sam1 = new Samkutxedi();
```

მისი შესრულების შედეგად მეხსიერებაში შეიქმნება Samkutxedi კლასის ეგზემპლარი - Sam1 ობიექტი.

ობიექტი შეიცავს კლასის თითოეული ცვლადის ასლს. ობიექტის ცვლადებთან მიმართვისათვის ვიყენებთ წერტილი (.) ოპერატორს. ის ობიექტის სახელს კლასის წევრის სახელს უკავშირებს. მისი სინტაქსია:

ობიექტის_სახელი.წევრის_სახელი

იმისათვის, რომ Sam1 ობიექტის gverdi2 ცვლადს მივანიჭოთ მნიშვნელობა 20, მინიჭების ოპერატორი უნდა ჩავწეროთ შემდეგნაირად:

```
Sam1.gverdi2 = 20;
```

წერტილი (.) ოპერატორი შეგვიძლია, აგრეთვე, გამოვიყენოთ მეთოდებთან მიმართვისთვისაც.

ქვემოთ მოყვანილია პროგრამა, რომელშიც ნაჩვენებია Samkutxedi კლასის გამოყენება.

```

//      პროგრამა 5.1
//      პროგრამაში ხდება ობიექტის ცვლადებთან მუშაობის დემონსტრირება
class Samkutxedi
{
    public int gverdi1;          //      სამკუთხედის პირველი გვერდი
    public int gverdi2;          //      სამკუთხედის მეორე გვერდი
    public int gverdi3;          //      სამკუთხედის მესამე გვერდი
}
private void button1_Click(object sender, System.EventArgs e)
{
    Samkutxedi Sam1 = new Samkutxedi();
    int perimetri;

    //      Sam1 ობიექტის ცვლადებს მნიშვნელობები ენიჭებათ
    Sam1.gverdi1 = int.Parse(textBox1.Text);
    Sam1.gverdi2 = int.Parse(textBox2.Text);
    Sam1.gverdi3 = int.Parse(textBox3.Text);
    //      სამკუთხედის პერიმეტრის გამოთვლა
    perimetri = Sam1.gverdi1 + Sam1.gverdi2 + Sam1.gverdi3;
    label1.Text = "სამკუთხედის პერიმეტრია " + perimetri.ToString();
}

```

კლასის აღწერა შეგვიძლია მოვათავსოთ Form1() მეთოდის შემდეგ. ამისათვის, ჯერ button კომპონენტი მოვათავსოთ ფორმაზე, შემდეგ კი ორჯერ სწრაფად დავაწკაპუნოთ მასზე. შემდეგ კურსორი მოვათავსოთ

```
private void button1_Click(object sender, System.EventArgs e)
```

სტრიქონის ზედა ცარიელ სტრიქონში და დავიწყოთ კლასის შეტანა. თუმცა, კლასი უმჯობესია შევიტანოთ ცალკე ფაილში. ამისათვის, ჯერ შევასრულოთ Project მენიუს Add Class (ან Add New Item) ბრძანება, შემდეგ მოვნიშნოთ Class ელემენტი, Name ველში შევიტანოთ კლასის სახელი, მაგალითად ChemiKlasi, დავაჭიროთ Add კლავიშს და დავიწყოთ კლასის შეტანა.

კლასის შეტანის კიდევ ერთი გზა ასეთია. ჯერ შევასრულოთ Project მენიუს Add New Item ბრძანება. შემდეგ, მოვნიშნოთ Visual C# Items განყოფილების Code File ელემენტი, Name ველში შევიტანოთ ამ ფაილის სახელი, მაგალითად Klasebi.cs, დავაჭიროთ Add კლავიშს და დავიწყოთ კლასის შეტანა. თუმცა, კლასის შესატანად უმჯობესია Class ელემენტი გამოვიყენოთ.

როგორც აღვნიშნეთ ობიექტების შექმნის ძირითადი პრინციპია ის, რომ თითოეულ ობიექტს აქვს კლასის ცვლადების საკუთარი ასლი. შედეგად, ერთი ობიექტის ცვლადები დამოუკიდებელია მეორე ობიექტის ცვლადებისაგან, ამიტომ მათი მნიშვნელობები შეიძლება ერთმანეთისაგან განსხვავდებოდეს. მაგალითად, თუ არსებობს Samkutxedi კლასის ორი ობიექტი, მაშინ თითოეულ მათგანს ექნება gverdi1, gverdi2 და gverdi3 ცვლადების საკუთარი ასლები, რომელთა მნიშვნელობები შეიძლება ერთმანეთისაგან განსხვავდებოდეს. ქვემოთ მოყვანილი პროგრამა ახდენს ამ პრინციპის დემონსტრირებას.

```

//      პროგრამა 5.2
//      პროგრამაში ხდება ორი ობიექტის ცვლადებთან მუშაობის დემონსტრირება
class Samkutxedi
{
    public int gverdi1;          //      სამკუთხედის პირველი გვერდი
    public int gverdi2;          //      სამკუთხედის მეორე გვერდი
    public int gverdi3;          //      სამკუთხედის მესამე გვერდი
}

```

```

}
private void button1_Click(object sender, System.EventArgs e)
{
    Samkutxedi Sam1 = new Samkutxedi();           // იქმნება Sam1 ობიექტი
    Samkutxedi Sam2 = new Samkutxedi();           // იქმნება Sam2 ობიექტი
    int Sam1_perimetri, Sam2_perimetri;

    // Sam1 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
    Sam1.gverdi1 = int.Parse(textBox1.Text);
    Sam1.gverdi2 = int.Parse(textBox2.Text);
    Sam1.gverdi3 = int.Parse(textBox3.Text);

    // Sam2 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
    Sam2.gverdi1 = int.Parse(textBox4.Text);
    Sam2.gverdi2 = int.Parse(textBox5.Text);
    Sam2.gverdi3 = int.Parse(textBox6.Text);

    // თითოეული სამკუთხედის პერიმეტრის გამოთვლა
    Sam1_perimetri = Sam1.gverdi1 + Sam1.gverdi2 + Sam1.gverdi3;
    Sam2_perimetri = Sam2.gverdi1 + Sam2.gverdi2 + Sam2.gverdi3;
    label1.Text = Sam1_perimetri.ToString();
    label2.Text = Sam2_perimetri.ToString();
}

```

ობიექტების შექმნა

წინა პროგრამებში Samkutxedi ტიპის ობიექტის გამოსაცხადებლად გამოიყენებოდა შემდეგი ოპერატორი

```
Samkutxedi Sam1 = new Samkutxedi();
```

მასში ორი მოქმედება სრულდება:

- ხდება Samkutxedi ტიპის ცვლადის გამოცხადება, რომლის სახელია Sam1. ის ობიექტს არ განსაზღვრავს (ე.ი. არ არის უშუალოდ ობიექტი), არამედ, მხოლოდ მიმართავს მას.

- new ოპერატორის შესრულებისას იქმნება ფიზიკური ობიექტი, ხოლო Sam1 ცვლადს ენიჭება მასზე მიმართვა (ანუ ამ ობიექტის მისამართი). შედეგად, Sam1 ცვლადი შეიცავს Samkutxedi ტიპის ობიექტზე მიმართვას.

new ოპერატორი დინამიურად (პროგრამის შესრულების დროს) გამოყოფს მეხსიერებას ობიექტისათვის და გასცემს მიმართვას მეხსიერების ამ უბანზე. მიმართვა არის მეხსიერების იმ უბნის მისამართი, რომელიც ობიექტს გამოეყო new ოპერატორის მიერ. შემდგომში ეს მიმართვა ინახება ცვლადში. ზემოთ მოყვანილი ოპერატორი უფრო მეტი თვალსაჩინოებისათვის შეგვიძლია ჩავწეროთ ორი ოპერატორის სახით:

```
Samkutxedi Sam1;
```

```
Sam1 = new Samkutxedi();
```

პირველ სტრუქტურში Sam1 ცვლადი გამოცხადებულია როგორც მიმართვა Samkutxedi ტიპის ობიექტზე. Sam1 ცვლადი მიმართავს ობიექტს, მაგრამ არ არის თვით ობიექტი. ამ ეტაპზე ამ ცვლადის მნიშვნელობაა null, რაც იმას ნიშნავს, რომ კავშირი ამ ცვლადსა და ობიექტს შორის არ არსებობს. მომდევნო სტრუქტურში იქმნება Samkutxedi ტიპის ახალი ობიექტი, ხოლო Sam1

ცვლადს ენიჭება მიმართვა ამ ობიექტზე. ამის შემდეგ, Sam1 ცვლადი დაკავშირებულია ობიექტთან.

რადგან კლასის ობიექტებთან მუშაობა ხორციელდება მიმართვის საშუალებით, ამიტომ კლასებს სხვანაირად მიმართვით ტიპებს უწოდებენ. როგორც ვიცით, მთავარი განსხვავება ჩვეულებრივ და მიმართვით ტიპებს შორის მდგომარეობს იმ მნიშვნელობებში, რომლებსაც ისინი ინახავენ. ჩვეულებრივი ტიპის ცვლადი ინახავს მნიშვნელობას, მიმართვითი ტიპის ცვლადი კი - ობიექტის მისამართს. განვიხილოთ მაგალითები.

```
int ricxvi;  
ricxvi = 20;
```

ამ ოპერატორების შესრულების შემდეგ ricxvi ცვლადში მოთავსებული იქნება მნიშვნელობა 20.

```
Samkutexedi Sam1 = new Samkutexedi();
```

ამ ოპერატორის შესრულების შემდეგ Sam1 ცვლადში მოთავსებული იქნება ობიექტზე მიმართვა ანუ ამ ობიექტის მისამართი.

მიმართვითი ტიპის მნიშვნელობის მინიჭება

მინიჭების ოპერატორის შესრულებისას მიმართვითი ტიპის ცვლადები ჩვეულებრივი ტიპის ცვლადებისაგან განსხვავებით სხვანაირად მოქმედებენ. როცა ჩვეულებრივი ტიპის ერთი ცვლადის მნიშვნელობას ვანიჭებთ ჩვეულებრივი ტიპის მეორე ცვლადის მნიშვნელობას, მაშინ მინიჭების ოპერატორის მარცხნივ მყოფ ცვლადს ენიჭება მარჯვნივ მყოფი ცვლადის მნიშვნელობის ასლი. როცა ერთი მიმართვითი ტიპის ცვლადის მნიშვნელობას ვანიჭებთ მიმართვითი ტიპის მეორე ცვლადის მნიშვნელობას, მაშინ მინიჭების ოპერატორის მარცხნივ მყოფ ცვლადს ენიჭება შესაბამის ობიექტზე მიმართვა. განვიხილოთ კოდის ფრაგმენტი:

```
Samkutexedi Sam1 = new Samkutexedi();  
Samkutexedi Sam2 = Sam1;
```

Sam2 ცვლადისათვის Sam1 ცვლადის მნიშვნელობის მინიჭებისას, Sam2 ცვლადს ენიჭება მიმართვა იმავე ობიექტზე, რომელსაც Sam1 ცვლადი მიმართავს. შედეგად, ერთსა და იმავე ობიექტთან მიმართვა შეიძლება შესრულდეს როგორც Sam1 ცვლადის, ისე Sam2 ცვლადის საშუალებით. სხვა სიტყვებით რომ ვთქვათ, Sam1 და Sam2 ცვლადები ერთსა და იმავე ობიექტს მიმართავენ. მაგალითად,

```
Sam1.gverdi1 = 30;
```

მინიჭების ოპერატორის შესრულების შედეგად Sam1.gverdi1 და Sam2.gverdi1 ცვლადებში აღმოჩნდება ერთნაირი მნიშვნელობა - 30. ეს ორი ცვლადი ერთმანეთთან არ არის დაკავშირებული, თუმცა ერთსა და იმავე ობიექტს მიმართავენ.

ქვემოთ მოყვანილი კოდის ფრაგმენტის შესრულების შედეგად Sam2 ცვლადი მიიღებს იმ ობიექტზე მიმართვას, რომელსაც Sam3 ცვლადი მიმართავს.

```
{  
Samkutexedi Sam1 = new Samkutexedi();  
Samkutexedi Sam2 = Sam1;  
Samkutexedi Sam3 = new Samkutexedi();  
Sam2 = Sam3;  
}
```

მეთოდები

მეთოდები - ესაა პროგრამები, რომლებიც ასრულებენ გარკვეულ ფუნქციას, მუშაობენ კლასში განსაზღვრულ მონაცემებთან და უზრუნველყოფენ ამ მონაცემებთან მიმართვას. მეთოდის გამოძახება ბევრჯერ შეიძლება.

მეთოდი შეიძლება შეიცავდეს ერთ ან მეტ ოპერატორს და უნდა წყვეტდეს მხოლოდ ერთ კონკრეტულ ამოცანას. მეთოდის სახელად შეგვიძლია გამოვიყენოთ ნებისმიერი იდენტიფიკატორი. საკვანძო სიტყვების გამოყენება მეთოდების სახელებად არ შეიძლება. Main() სახელი დარეზერვებულია იმ მეთოდისათვის, საიდანაც იწყება პროგრამის შესრულება.

მეთოდის სინტაქსია:

მიმართვის_მოდულიკატორი ტიპი მეთოდის_სახელი(პარამეტრების_სია)

{

მეთოდის ტანი

}

აქ მიმართვის_მოდულიკატორი იღებს private ან public მნიშვნელობებს. თუ ის არ არის მითითებული, მაშინ იგულისხმება private, ე.ი. მეთოდი მისაწვდომი იქნება მხოლოდ იმ კლასის შიგნით, რომელშიც ის არის აღწერილი. ტიპი არის იმ მნიშვნელობის ტიპი, რომელსაც მეთოდი გასცემს. თუ მეთოდი არ გასცემს მნიშვნელობას, მაშინ უნდა მივუთითოთ void ტიპი. მეთოდის_სახელი არის ნებისმიერი იდენტიფიკატორი, რომელიც არ ემთხვევა კლასის სხვა წევრების სახელებს ხილვადობის იმავე უბანში. პარამეტრების_სია შედგება პარამეტრების სახელებისაგან, რომლებიც ერთმანეთისაგან მძიმეებით გამოიყოფა. თითოეული პარამეტრის სახელის წინ ეთითება შესაბამისი ტიპი. პარამეტრები ესაა ცვლადები, რომლებიც იღებენ არგუმენტების მნიშვნელობებს, რომლებიც მეთოდს გადაეცემა მისი გამოძახებისას. თუ მეთოდს პარამეტრები არ აქვს, მაშინ პარამეტრების_სია ცარიელი იქნება.

ზემოთ მოყვანილ მაგალითებში სამკუთხედის პერიმეტრის გამოთვლა სრულდებოდა ძირითად პროგრამაში. პერიმეტრი უმჯობესია გამოითვალოს უშუალოდ კლასის შიგნით, რადგან მისი მნიშვნელობა დამოკიდებულია სამკუთხედის გვერდების ზომებზე. სამივე ეს სიდიდე კი ინკაფსულირებულია Samkutxedi კლასში. გარდა ამისა, Samkutxedi კლასისთვის იმ მეთოდის დამატება, რომელშიც სრულდება პერიმეტრის გამოთვლა აუმჯობესებს ამ კლასის ობიექტზე ორიენტირებულ სტრუქტურას.

ქვემოთ მოყვანილია Samkutxedi კლასის მომდევნო ვერსია. მის Perimetri() მეთოდს ფანჯარაში გამოაქვს სამკუთხედის პერიმეტრის მნიშვნელობა.

// პროგრამა 5.3

// პროგრამაში ხდება სამკუთხედის პერიმეტრის გამოთვლა კლასის შიგნით

// გამოცხადებულ მეთოდში

```
class Samkutxedi
```

```
{
```

```
public int gverdi1;
```

```
public int gverdi2;
```

```
public int gverdi3;
```

```
private int perim;
```

```
// Perimetri() მეთოდის განსაზღვრა
```

```
public void Perimetri()
```

```
{
```

```
perim = gverdi1 + gverdi2 + gverdi3;
```

```
MessageBox.Show("სამკუთხედის პერიმეტრია - " + perim.ToString());
```

```
}
```

```
}
```

```

private void button1_Click(object sender, System.EventArgs e)
{
    Samkutxedi Sam1 = new Samkutxedi();
    Samkutxedi Sam2 = new Samkutxedi();

    // Sam1 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
    Sam1.gverdi1 = int.Parse(textBox1.Text);
    Sam1.gverdi2 = int.Parse(textBox2.Text);
    Sam1.gverdi3 = int.Parse(textBox3.Text);
    // პერიმეტრის გამოთვლა პირველი სამკუთხედისათვის
    Sam1.Perimetri();
    // Sam2 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
    Sam2.gverdi1 = int.Parse(textBox4.Text);
    Sam2.gverdi2 = int.Parse(textBox5.Text);
    Sam2.gverdi3 = int.Parse(textBox6.Text);
    // პერიმეტრის გამოთვლა მეორე სამკუთხედისათვის
    Sam2.Perimetri();
}

```

პროგრამაში perim ცვლადი იმიტომ არის პრივატული, რომ დაცული იყოს ობიექტის მთლიანობა. თუ მას გამოვაცხადებთ როგორც public, მაშინ იარსებებს იმის საფრთხე, რომ ამ ცვლადს არასწორი მნიშვნელობა მიენიჭოს და შედეგად, სამკუთხედის გვერდების ჯამი შეიძლება არ დაემთხვეს პერიმეტრის მნიშვნელობას.

```
public void Perimetri()
```

სტრიქონში ხდება Perimetri() მეთოდის გამოცხადება, რომელსაც პარამეტრები არ აქვს. ის განსაზღვრულია როგორც public, რაც იძლევა მისი გამოძახების შესაძლებლობას პროგრამის ნებისმიერ ადგილიდან. void სიტყვა მიუთითებს იმას, რომ Perimetri() მეთოდი არ აბრუნებს (არ გასცემს) მნიშვნელობას. შემდეგ, ფიგურულ ფრჩხილებში მოთავსებულია მეთოდის ტანი, რომელშიც გამოითვლება სამკუთხედის პერიმეტრი. მისი მნიშვნელობა ეკრანზე გამოიციმა MessageBox.Show() მეთოდის გამოყენებით. რადგან Samkutxedi ტიპის თითოეულ ობიექტს აქვს gverdi1, gverdi2 და gverdi3 ცვლადების საკუთარი ასლები, ამიტომ Perimetri() მეთოდის გამოძახებისას პერიმეტრის გამოსათვლელად გამოყენებული იქნება გამომძახებელი ობიექტის ცვლადები.

```
Sam1.Perimetri();
```

სტრიქონში გამომძახებელია Sam1 ობიექტი. აქ ხდება Sam1 ობიექტის Perimetri() მეთოდის გამოძახება, რის შემდეგ მართვა ამ მეთოდს გადაეცემა. ის Sam1 ობიექტის ცვლადებთან იმუშავებს. როცა მეთოდი მუშაობას დაამთავრებს, მართვა გადაეცემა პროგრამის იმ ადგილს, საიდანაც მისი გამოძახება მოხდა. პროგრამის შესრულება გაგრძელდება კოდის იმ სტრიქონიდან, რომელიც მოსდევს მეთოდის გამომძახებელ სტრიქონს. ჩვენს შემთხვევაში Sam1.Perimetri() მეთოდის გამოძახება იწვევს პერიმეტრის მნიშვნელობის გამოტანას Sam1 სამკუთხედისათვის, ხოლო Sam2.Perimetri() მეთოდის გამოძახება კი იწვევს პერიმეტრის მნიშვნელობის გამოტანას Sam2 სამკუთხედისათვის.

პრივატული ცვლადის გამოტანა შეგვიძლია Perimetri() მეთოდისთვის პარამეტრად label1 კომპონენტის გადაცემის გზით. პროგრამას ექნება სახე:

```

class Samkutxedi
{
    public int gverdi1;
    public int gverdi2;

```



```

public int gverdi3;
private int perim;
// Perimetri() მეთოდის განსაზღვრა
public void Perimetri(Label lab1)
{
perim = gverdi1 + gverdi2 + gverdi3;
lab1.Text = "სამკუთხედის პერიმეტრია - " + perim.ToString();
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Samkutxedi Sam1 = new Samkutxedi();

// Sam1 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
Sam1.gverdi1 = int.Parse(textBox1.Text);
Sam1.gverdi2 = int.Parse(textBox2.Text);
Sam1.gverdi3 = int.Parse(textBox3.Text);
// პერიმეტრის გამოთვლა პირველი სამკუთხედისათვის
Sam1.Perimetri (label1);
}

```

როგორც ვხედავთ, Perimetri() მეთოდის გამოძახებისას, მას პარამეტრად label1 კომპონენტი გადაეცემა. მისი მისამართი მიენიჭება lab1 პარამეტრს, რომლის ტიპია Label. ამის შემდეგ, lab1 პარამეტრი მუშაობს ზუსტად ისე, როგორც label1 კომპონენტი.

მეთოდიდან მართვის დაბრუნება

მეთოდიდან მართვის დაბრუნება, ანუ მეთოდის შესრულების შეწყვეტა და გამომძახებელი პროგრამისათვის მართვის დაბრუნება, ორ შემთხვევაში ხდება. პირველი, როცა გვხვდება მეთოდის დამხურავი ფიგურული ფრჩხილი და მეორე, როცა სრულდება return ოპერატორი. არსებობს return ოპერატორის ორი ფორმა. პირველი ფორმა გამოიყენება მეთოდებში, რომლებსაც void ტიპი აქვთ, ანუ მნიშვნელობას არ აბრუნებენ, მეორე კი - მეთოდებში, რომლებიც მნიშვნელობას აბრუნებენ.

მეთოდებში, რომლებიც მნიშვნელობას არ აბრუნებენ, გამოიყენება შემდეგი სინტაქსის მქონე return ოპერატორი:

return;

ასეთ შემთხვევაში return ოპერატორი შეგვიძლია არც მივუთითოთ.

return ოპერატორის შესრულების დროს მართვა გადაეცემა გამომძახებელი პროგრამის იმ ადგილს, საიდანაც მეთოდი იყო გამოძახებული, ამასთან მეთოდის კოდის დარჩენილი ნაწილი არ შესრულდება.

მეთოდის ერთ-ერთი მნიშვნელოვანი თვისებაა - მის მიერ მნიშვნელობის დაბრუნება (გაცემა). მეთოდები მნიშვნელობას უბრუნებენ გამომძახებელ პროგრამას შემდეგი სინტაქსის მქონე return ოპერატორის გამოყენებით:

return მნიშვნელობა;

აქ *მნიშვნელობა* არის მეთოდის მიერ დაბრუნებული მნიშვნელობა.

ახლა Perimetri() მეთოდი გადავაკეთოთ ისე, რომ მან გამოთვალოს პერიმეტრი და დაუბრუნოს ის პროგრამას. ასეთი მიდგომის ერთ-ერთი უპირატესობაა ის, რომ დაბრუნებული

მნიშვნელობა შეგვიძლია გამოვიყენოთ სხვა გამოთვლებში. ქვემოთ მოყვანილ პროგრამაში Perimetri() მეთოდს ეკრანზე აღარ გამოაქვს პერიმეტრის მნიშვნელობა, არამედ ახდენს მის გამოთვლასა და შედეგის დაბრუნებას.

// **პროგრამა 5.4**

// **პროგრამაში ხდება return ოპერატორის გამოყენების დემონსტრირება**

```
class Samkutxedi
{
    public int gverdi1;
    public int gverdi2;
    public int gverdi3;
    public int Perimetri()                // მეთოდის განსაზღვრა
    {
        return gverdi1 + gverdi2 + gverdi3;    // მნიშვნელობის დაბრუნება
    }
}
private void button1_Click(object sender, System.EventArgs e)
{
    // tolgverda და tolfesda ობიექტების შექმნა
    Samkutxedi tolgverda = new Samkutxedi();
    Samkutxedi tolfesda = new Samkutxedi();
    int tolgverda_perimetri, tolfesda_perimetri;

    // tolgverda ობიექტის ცვლადებს მნიშვნელობები ენიჭებათ
    tolgverda.gverdi1 = int.Parse(textBox1.Text);
    tolgverda.gverdi2 = int.Parse(textBox2.Text);
    tolgverda.gverdi3 = int.Parse(textBox3.Text);

    // tolfesda ობიექტის ცვლადებს მნიშვნელობები ენიჭებათ
    tolfesda.gverdi1 = int.Parse(textBox4.Text);
    tolfesda.gverdi2 = int.Parse(textBox5.Text);
    tolfesda.gverdi3 = int.Parse(textBox6.Text);

    // tolgverda და tolfesda სამკუთხედებისათვის პერიმეტრი გამოითვლება
    tolgverda_perimetri = tolgverda.Perimetri();
    tolfesda_perimetri = tolfesda.Perimetri();
    label1.Text = tolgverda_perimetri.ToString();
    label2.Text = tolfesda_perimetri.ToString();
}
```

პროგრამაში Perimetri() მეთოდის მიერ გაცემული მნიშვნელობები შესაბამისად მიენიჭება tolgverda_perimetri და tolfesda_perimetri ცვლადებს.

პარამეტრების გამოყენება

გამოძახებისას მეთოდს შეგვიძლია გადავცეთ ერთი ან მეტი პარამეტრი. მეთოდისთვის გადასაცემ მნიშვნელობას არგუმენტი ეწოდება. ცვლადს მეთოდის შიგნით, რომელიც იღებს არგუმენტის მნიშვნელობას, პარამეტრი ეწოდება. პარამეტრების გამოცხადება ხდება მრგვალი

ფრჩხილების შიგნით, რომლებიც მეთოდის სახელს მოსდევს. პარამეტრის გამოცხადების სინტაქსი ცვლადების გამოცხადების სინტაქსის ანალოგიურია. პარამეტრი იმყოფება თავისი მეთოდის ხილვადობის უბანში, ანუ ხილულია მხოლოდ ამ მეთოდის შიგნით, და მოქმედებს როგორც ლოკალური ცვლადი.

ქვემოთ მოყვანილია პროგრამა, რომელშიც მეთოდისათვის მნიშვნელობის გადასაცემად გამოიყენება პარამეტრი. ArisLuwi კლასის შიგნით განსაზღვრული luwi_kenti(int x) მეთოდი აბრუნებს bool ტიპის შედეგს. თუ მისთვის გადაცემული მნიშვნელობა არის ლუწი მაშინ ის გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას.

// **პროგრამა 5.5**

// **პროგრამა განსაზღვრავს რიცხვი კენტია თუ ლუწი**

```
class ArisLuwi
{
// მეთოდის განსაზღვრა, რომელიც ამოწმებს რიცხვი კენტია თუ ლუწი
public bool luwi_kenti( int cvladi )
{
if ( ( cvladi % 2 ) == 0 ) return true;
else return false;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//
int ricxvi;
ArisLuwi obieqti = new ArisLuwi();

ricxvi = int.Parse(textBox1.Text);
if ( obieqti.luwi_kenti(ricxvi) )
label1.Text = "რიცხვი " + ricxvi.ToString() + " ლუწია";
else label1.Text = "რიცხვი " + ricxvi.ToString() + " კენტია";
}
```

მეთოდს შეიძლება რამდენიმე პარამეტრი ჰქონდეს. ისინი ერთმანეთისაგან მძიმეებით გამოიყოფა. მაგალითად, Gamyofi კლასი შეიცავს ArisGamyofi() მეთოდს, რომელიც განსაზღვრავს არის თუ არა პირველი პარამეტრი მეორე პარამეტრის გამყოფი.

// **პროგრამა 5.6**

// **პროგრამა განსაზღვრავს ერთი რიცხვი არის თუ არა მეორის გამყოფი**

```
class Gamyofi
{
// მეთოდი ამოწმებს მეორე პარამეტრი უნაშთოდ იყოფა თუ არა პირველ პარამეტრზე
public bool ArisGamyofi(int par1, int par2)
{
if ( ( par1 % par2 ) == 0 ) return true;
else return false;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int ricxvi1, ricxvi2;
```

```
Gamyofi obieqti = new Gamyofi();

ricxvi1 = int.Parse(textBox1.Text);
ricxvi2 = int.Parse(textBox2.Text);
if ( obieqti.ArisGamyofi(ricxvi1, ricxvi2) )
    label2.Text = ricxvi1.ToString() + " არის " + ricxvi2.ToString() + "-ის გამყოფი";
    else label2.Text = ricxvi1.ToString() + " არ არის " + ricxvi2.ToString() + "-ის გამყოფი";
}
```

ახლა განვიხილოთ მეთოდისათვის მასივის გადაცემის მაგალითი. ქვემოთ მოყვანილ პროგრამაში Minimumi() მეთოდს გადაეცემა მასივი. მეთოდი პოულობს და აბრუნებს მასივის მინიმალური ელემენტის მნიშვნელობას.

// **პროგრამა 5.7**

// **პროგრამაში ხდება მეთოდისათვის მასივის გადაცემის დემონსტრირება**

```
class MinSidide
{
// მეთოდი გასცემს მასივის მინიმალური ელემენტის მნიშვნელობას
public int Minimumi(int[] mas)
{
int min_ricxvi = mas[0];
for ( int indexi =1; indexi < mas.Length; indexi++ )
    if ( min_ricxvi > mas[indexi] ) min_ricxvi = mas[indexi];
return min_ricxvi;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int[] masivi = new int[] { 5, -7, 4, -6, 9 };
MinSidide obieqti = new MinSidide();
int shedegi;

shedegi = obieqti.Minimumi(masivi); // Minimumi მეთოდს masivi მასივი გადაეცემა
label1.Text = shedegi.ToString();
}
```

კონსტრუქტორები

კონსტრუქტორი არის მეთოდი, რომლის დანიშნულებაცაა ობიექტის ცვლადების ინიციალიზება. ობიექტის ინიციალიზება სრულდება მისი შექმნისას. კონსტრუქტორს ისეთივე სახელი აქვს, როგორც კლასს და მისი სინტაქსი მეთოდის სინტაქსის მსგავსია. მეთოდისგან განსხვავებით კონსტრუქტორში დასაბრუნებელი მნიშვნელობის ტიპი არ ეთითება. მისი სინტაქსია:

კლასის_სახელი()

```
{
კონსტრუქტორის კოდი
}
```

როგორც წესი კონსტრუქტორები გამოიყენება ობიექტის ცვლადებისათვის საწყისი

მნიშვნელობების მისანიჭებლად, ან ინიციალიზაციის ნებისმიერი სხვა პროცედურის შესასრულებლად, რომლებიც აუცილებელია მთლიანად ფორმირებული ობიექტის შესაქმნელად.

ყველა კლასს აქვს კონსტრუქტორი მიუხედავად იმისა ის განსაზღვრულია თუ არა. C# ენაში გათვალისწინებულია კონსტრუქტორის არსებობა, რომელიც ობიექტის ყველა ცვლადს ანიჭებს ნულოვან (ეს ეხება ჩვეულებრივი ტიპის ცვლადებს) ან null მნიშვნელობას (ეს ეხება მიმართვითი ტიპის ცვლადებს). ასეთ კონსტრუქტორს ავტომატური კონსტრუქტორი (კონსტრუქტორი გაჩუმებით) ეწოდება. ის იმ შემთხვევაში გამოიძახება, როცა კლასში კონსტრუქტორი განსაზღვრული არ არის. თუ კონსტრუქტორი ამკარადაა განსაზღვრული კლასში, მაშინ სწორედ ის იქნება გამოძახებული.

პროფესიულ დონეზე შედგენილ პროგრამაში ობიექტის ცვლადების ინიციალიზება ყოველთვის კონსტრუქტორის მიერ სრულდება. მაგალითად, 5.1, 5.2, 5.3, 5.5 პროგრამებში ობიექტის ცვლადებს მნიშვნელობებს ჩვენ ვანიჭებდით. უმჯობესია, ეს კონსტრუქტორმა გააკეთოს. განვიხილოთ 5.4 პროგრამის ახალი ვერსია, რომელშიც ობიექტის ცვლადებს მნიშვნელობებს კონსტრუქტორი ანიჭებს.

// **პროგრამა 5.8**

// **პროგრამაში ხდება კონსტრუქტორთან მუშაობის დემონსტრირება**

```
class Samkutxedi
{
public int gverdi1;
public int gverdi2;
public int gverdi3;
private int perimetri;
// კონსტრუქტორი პარამეტრებით
public Samkutxedi(int par1, int par2, int par3)
{
gverdi1 = par1;
gverdi2 = par2;
gverdi3 = par3;
}
public int Perim()
{
perimetri = gverdi1 + gverdi2 + gverdi3;
return perimetri;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int g1 = int.Parse(textBox1.Text);
int g2 = int.Parse(textBox2.Text);
int g3 = int.Parse(textBox3.Text);
int g4 = int.Parse(textBox4.Text);
int g5 = int.Parse(textBox5.Text);
int g6 = int.Parse(textBox6.Text);
Samkutxedi Sam1 = new Samkutxedi(g1, g2, g3);
Samkutxedi Sam2 = new Samkutxedi(g4, g5, g6);

int SamkutxedisPerimetri1 = Sam1.Perim();
```

```

int SamkutxedisPerimetri2 = Sam2.Perim();
label1.Text = "პირველი სამკუთხედის პერიმეტრი = " + SamkutxedisPerimetri1.ToString();
label2.Text = "მეორე სამკუთხედის პერიმეტრი = " + SamkutxedisPerimetri2.ToString();
}

```

პროგრამაში new ოპერატორის შესრულებისას გამოიძახება Samkutxedi() კონსტრუქტორი, რადგან ის განსაზღვრულია კლასში. მას სამი პარამეტრი აქვს. ისინი გამოიყენება სამკუთხედის გვერდებისათვის მნიშვნელობების მისანიჭებლად. პირველი new ოპერატორის შესრულების დროს Samkutxedi() კონსტრუქტორს g1, g2 და g3 არგუმენტები გადაეცემა. ისინი შესაბამისად par1, par2 და par3 პარამეტრებს მიენიჭება. ისინი თავის მხრივ gverdi1, gverdi2 და gverdi3 ცვლადებს ენიჭება.

დესტრუქტორები და "ნაგვის შეგროვება"

როცა C# ენაში არ ხდება ობიექტთან მიმართვა, მაშინ ის განიხილება როგორც შემდგომში არაგამოყენებადი და სისტემა ავტომატურად, პროგრამისტის მონაწილეობის გარეშე, ათავისუფლებს ობიექტის მიერ დაკავებულ მეხსიერებას. ამ ოპერაციას "ნაგვის შეგროვება" ეწოდება. გამოთავისუფლებული მეხსიერება შემდგომში შეიძლება გამოყენებული იყოს სხვა ობიექტებისათვის. თუ ეს ოპერაცია არ შესრულდა, მაშინ მეხსიერების შეზღუდული ზომის გამო new ოპერატორმა შეიძლება ვერ გამოყოს მეხსიერება სხვა ობიექტებისათვის.

"ნაგვის შეგროვება" პერიოდულად სრულდება პროგრამის მუშაობის განმავლობაში. ამ ოპერაციის დასაწყებად ორი პირობა უნდა შესრულდეს. ჯერ ერთი, უნდა არსებობდეს ობიექტები, რომლებიც შეიძლება წავშალოთ მეხსიერებიდან, და მეორე, ასეთი ოპერაციის აუცილებლობა. რადგან "ნაგვის შეგროვება" პროცესორის დროს იკავებს, ამიტომ ის მხოლოდ აუცილებლობის შემთხვევაში შესრულდება, ამასთან პროგრამისტი ზუსტად ვერ განსაზღვრავს ამ მომენტს.

დესტრუქტორი არის მეთოდი, რომელიც გამოიყენება მეხსიერებიდან ობიექტის კორექტულად წასაშლელად. მისი სინტაქსია:

```

~კლასის_სახელი()
{
დესტრუქტორის კოდი
}

```

როგორც ვხედავთ დესტრუქტორის სახელის წინ მითითებულია სიმბოლო ტილდა (~). დესტრუქტორის განსაზღვრისას დასაბრუნებელი მნიშვნელობის ტიპი არ ეთითება.

დესტრუქტორის გამოძახება ხდება უშუალოდ "ნაგვის შეგროვების" ოპერაციის წინ მაშინ, როცა სრულდება მისი კლასის ობიექტის წაშლა მეხსიერებიდან. რადგან, წინასწარ არ ვიცით, თუ როდის დაიწყება "ნაგვის შეგროვების" პროცესი, ამიტომ არ გვეცოდინება თუ როდის შესრულდება დესტრუქტორის გამოძახება. თუ პროგრამამ მუშაობა დაამთავრა "ნაგვის შეგროვების" ოპერაციის დაწყებამდე, მაშინ დესტრუქტორი არასოდეს არ გამოიძახება.

ობიექტების მასივის შექმნა

ობიექტებისაგან შეგვიძლია შევქმნათ როგორც ერთ, ისე რამდენიმე განზომილებიანი მასივი. მოყვანილი პროგრამით იქმნება Martkutxedi ტიპის ერთგანზომილებიანი მასივი, რომელიც 5 ობიექტისაგან შედგება. მასივის გამოცხადების შემდეგ, მისი თითოეული ელემენტი (ობიექტი) ცალ-ცალკე უნდა შეიქმნას.

```

class Martkutxedi
{
public int simagle;
public int sigane;
int perimetri;
public Martkutxedi(int par1, int par2)
{
simagle = par1;
sigane = par2;
perimetri = ( simagle + sigane ) * 2;
}
public int Gamotana()
{
return perimetri;
}
}
private void button1_Click(object sender, EventArgs e)
{
//      Martkutxedi ტიპის ერთგანზომილებიანი მასივის შექმნა,
//      რომელიც 5 ობიექტისაგან შედგება
Martkutxedi[] masivi = new Martkutxedi[5];
//      თითოეული ობიექტის შექმნა და მათი ცვლადების ინიციალიზება
for ( int ind = 0; ind < masivi.Length; ind++)
    masivi[ind] = new Martkutxedi(ind + 10, ind + 20);
//      თითოეული ობიექტის ცვლადების ეკრანზე გამოტანა
for ( int ind = 0; ind < masivi.Length; ind++ )
    label1.Text += masivi[ind].sigane.ToString() + " " + masivi[ind].simagle.ToString() + " " +
masivi[ind].Gamotana().ToString() + '\n';
}

```

this საკვანძო სიტყვა

მეთოდს გამოძახებისას ავტომატურად გადაეცემა არაცხადი არგუმენტი, რომელიც წარმოადგენს გამომძახებელ ობიექტზე მიმართვას. ამ მიმართვას ეწოდება this. იმის გასაგებად თუ როგორ მუშაობს this მიმართვა, განვიხილოთ პროგრამა, რომელშიც იქმნება Axarisxeba კლასი. ის განკუთვნილია რიცხვის ხარისხის გამოსათვლელად.

// პროგრამა 5.9

// პროგრამაში ხარისხის გამოთვლა სრულდება კონსტრუქტორის კოდში

```

class Axarisxeba
{
public double wiladi;
public int mteli;
public double shedegi;
public Axarisxeba(double par_ricxvi, int par_xarisxi)
{
wiladi = par_ricxvi;
mteli = par_xarisxi;
shedegi = 1;
}
}

```

```

if ( par_xarisxi == 0 ) return;
for ( ; par_xarisxi > 0; par_xarisxi-- ) shedegi *= wiladi;
}
public double xarisxis_dabruneba()
{
return shedegi;
}
}
private void button1_Click(object sender, EventArgs e)
{
double ricxvi = Convert.ToDouble(textBox1.Text);
int xarisxi = Convert.ToInt32(textBox2.Text);
Axarisxeba obieqti = new Axarisxeba(ricxvi, xarisxi);

label1.Text = obieqti.wiladi.ToString() + " ხარისხად " + obieqti.mteli.ToString() +
" არის " + obieqti.xarisxis_dabruneba().ToString();
}

```

როგორც ვიცით მეთოდის ფარგლებში კლასის სხვა წევრებთან მიმართვა შეძლება განხორციელდეს ობიექტის ან კლასის სახელის მითითების გარეშე, ე.ი. უშუალოდ. შედეგად, xarisxis_dabruneba() მეთოდის შიგნით return shedegi; ოპერატორის შესრულების შედეგად გაიცემა shedegi ცვლადის მნიშვნელობა, რომელიც ასოცირებულია გამომძახებელ ობიექტთან. მაგრამ ეს ოპერატორი შეგვიძლია ასეც ჩავწეროთ:

```
return this.shedegi;
```

აქ this მიმართვა მიუთითებს იმ ობიექტზე, რომელსაც ეკუთვნის xarisxis_dabruneba() მეთოდი. this.shedegi ცვლადი არის მოცემული ობიექტის shedegi ცვლადის ასლი. მაგალითად, თუ xarisxis_dabruneba() მეთოდი გამოძახებულია x ობიექტისთვის, მაშინ this მიმართვა მიუთითებს x ობიექტზე. ოპერატორის ჩაწერა this სიტყვის გარეშე არის ჩაწერის შემოკლებული ფორმა.

C# ენის სინტაქსი იძლევა ერთნაირი სახელების გამოყენების საშუალებას პარამეტრებისა და ლოკალური ცვლადებისათვის. ასეთ შემთხვევაში, პარამეტრი მაღავს ლოკალურ ცვლადს და მასთან მიმართვა შესაძლებელია მხოლოდ this მიმართვის გამოყენებით. ქვემოთ მოყვანილ პროგრამაში ხდება დამალულ ცვლადთან მიმართვა this სიტყვის გამოყენებით.

// პროგრამა 5.10

// პროგრამაში ხდება დამალულ ცვლადთან მიმართვის დემონსტრირება

```

class Axarisxeba
{
public double ricxvi;
public int xarisxi;
public double shedegi;
public Axarisxeba(double ricxvi, int xarisxi)
{
this.ricxvi = ricxvi;           // ობიექტის this.ricxvi ცვლადს ენიჭება ricxvi პარამეტრი
this.xarisxi = xarisxi;       // ობიექტის this.xarisxi ცვლადს ენიჭება
                                // xarisxi პარამეტრი

shedegi = 1;

if (this.xarisxi == 0 ) return;
for ( ; this.xarisxi > 0; this.xarisxi-- ) shedegi *= this.ricxvi;

```



```

}
}
private void button1_Click(object sender, System.EventArgs e)
{
double ricxvi1 = double.Parse(textBox1.Text);
int xarisxi1 = int.Parse(textBox2.Text);
Axarisxeba obieqti = new Axarisxeba(ricxvi1, xarisxi1);
label1.Text = obieqti.shedegi.ToString();
}

```

Axarisxeba() კონსტრუქტორის ამ ვერსიაში პარამეტრების სახელები და ობიექტის ცვლადების სახელები ერთნაირია, ამიტომ ობიექტის ცვლადები იქნება დამალული. this მიმართვის გამოყენებით შესაძლებელი ხდება მათთან მიმართვა.

რთული კლასები

ერთი ობიექტი შეიძლება მეორე ობიექტს შეიცავდეს. მაგალითად, ობიექტი „უნივერსიტეტი“ შეიძლება შეიცავდეს „სტუდენტი“ ობიექტს. ამ დროს კლასის ცვლადის გამოცხადება ხდება სხვა კლასის გამოყენებით. შედეგად, შეგვიძლია რთული კლასების შექმნა, რომლის წევრებს ექნება სხვა კლასების ტიპები. განვიხილოთ მაგალითი.

// პროგრამა 5.11

// პროგრამაში ხდება რთულ კლასთან მუშაობის დემონსტრირება

```

class Studenti // მარტივი კლასი
{
public string gvari;
public string fakulteti;
public int kursi;
public void Metodi_Studenti()
{
MessageBox.Show("მარტივი კლასი\n" + gvari + "\n" + fakulteti + "\n" + kursi.ToString());
}
}
class Universiteti // რთული კლასი
{
public Studenti studenti; // studenti ცვლადს აქვს Studenti კლასის ტიპი
public string misamarti;
public void Metodi_Universiteti()
{
MessageBox.Show("რთული კლასი\n" + misamarti);
}
}
private void button1_Click(object sender, EventArgs e)
{
Universiteti obieqti_universiteti = new Universiteti();
obieqti_universiteti.studenti = new Studenti();
obieqti_universiteti.studenti.gvari = textBox1.Text;
obieqti_universiteti.studenti.fakulteti = textBox2.Text;
obieqti_universiteti.studenti.kursi = int.Parse(textBox3.Text);
}

```

```

obieqti_universiteti.studenti.Metodi_Studenti();
obieqti_universiteti.misamarti = textBox4.Text;
obieqti_universiteti.Metodi_Universiteti();
}

```

ჩადგმული კლასები

ერთი კლასის შიგნით დასაშვებია მეორე კლასის გამოცხადება, ანუ ერთი კლასი შეიძლება ჩადგმული იყოს მეორის შიგნით. ჩადგმულ კლასს *შიდა კლასი* ეწოდება, მის შემცველ კლასს კი - *გარე*. შიდა კლასი შეიძლება იყოს როგორც პრივატული, ისე ღია. მოყვანილ პროგრამაში ხდება ჩადგმულ კლასებთან მუშაობის დემონსტრირება.

// პროგრამა 5.12

// პროგრამაში ხდება რთულ კლასთან მუშაობის დემონსტრირება

```

class Universiteti // გარე კლასი
{
    public class Studenti // შიდა კლასი
    {
        public string gvari;
        public string fakulteti;
        public int kursi;
        public void Metodi_Universiteti()
        {
            MessageBox.Show("შიდა კლასი\n" + gvari + "\n" + fakulteti + "\n" + kursi.ToString());
        }
    }
    public Studenti studenti;
    public string misamarti;
    public void Metodi_Studenti()
    {
        MessageBox.Show("გარე კლასი\n" + misamarti);
    }
}
private void button1_Click(object sender, EventArgs e)
{
    Universiteti obieqti = new Universiteti();
    obieqti.Metodi_Studenti();
    obieqti.misamarti = textBox4.Text;
    obieqti.studenti = new Universiteti.Studenti();
    obieqti.studenti.gvari = textBox1.Text;
    obieqti.studenti.fakulteti = textBox2.Text;
    obieqti.studenti.kursi = int.Parse(textBox3.Text);
    obieqti.studenti.Metodi_Universiteti();
}

```

სტრუქტურები

სტრუქტურა წარმოადგენს კლასის გამარტივებულ ვარიანტს. სტრუქტურებთან მუშაობა სრულდება ისე, როგორც მნიშვნელობის მქონე ტიპებთან. მათი ძირითადი განსხვავება კლასებისაგან შემდეგში მდგომარეობს. კლასის ობიექტი ინახება გროვაში და მესხერებიდან იშლება მხოლოდ "ნაგვის შეგროვების" დროს. სტრუქტურის ეგზემპლარი ინახება სტეკში და მესხერებიდან იშლება მაშინ, როცა ეს ეგზემპლარი გამოდის ხილვადობის უბნიდან. კლასების მსგავსად სტრუქტურები შეიძლება შეიცავდეს ცვლადებს, მეთოდებს, ინდექსატორებს, თვისებებს, მოვლენებსა და კონსტრუქტორებს. სტრუქტურები უნდა იყოს მცირე ზომის და მათი გამოყენება უმჯობესია მაშინ, როცა გვაქვს მცირე რაოდენობის ცვლადები და მეთოდები. კლასებისგან განსხვავებით მათ აქვთ შემდეგი შეზღუდვები:

- სტრუქტურები არ უზრუნველყოფენ მემკვიდრეობითობას.
- სტრუქტურისთვის არ შეიძლება ავტომატური კონსტრუქტორის განსაზღვრა.
- სტრუქტურისთვის არ შეიძლება დესტრუქტორის განსაზღვრა.
- არ შეიძლება ინიციალიზატორების გამოყენება ცვლადებისათვის მნიშვნელობების მისანიჭებლად.

სტრუქტურის გამოცხადებისათვის გამოიყენება `struct` საკვანძო სიტყვა. მოყვანილ პროგრამაში ხდება სტრუქტურასთან მუშაობის დემონსტრირება.

```
{
//   პროგრამა 5.13
//   პროგრამაში ხდება სტრუქტურასთან მუშაობის დემონსტრირება
public struct Samkutxedi
{
//   ცვლადების გამოცხადება
public int gverdi1;
public int gverdi2;
public int gverdi3;
//   კონსტრუქტორის განსაზღვრა
public Samkutxedi(int par1, int par2, int par3)
{
gverdi1 = par1;
gverdi2 = par2;
gverdi3 = par3;
}
//   მეთოდის განსაზღვრა
public int Perimetri()
{
return gverdi1 + gverdi2 + gverdi3;
}
}
private void button1_Click(object sender, EventArgs e)
{
int gverdi1 = int.Parse(textBox1.Text);
int gverdi2 = int.Parse(textBox2.Text);
int gverdi3 = int.Parse(textBox3.Text);
Samkutxedi struqtura = new Samkutxedi(gverdi1, gverdi2, gverdi3);
int Shedegi = struqtura.Perimetri();
label1.Text = Shedegi.ToString();
}
```

```
}
}
```

შეიძლება ერთი სტრუქტურის მეორეში გადაწერა. მაგალითად,
 Samkutexedi structura1 = new Samkutexedi(gverdi1, gverdi2, gverdi3);
 Samkutexedi structura2 = structura1;

შემთხვევითი რიცხვების გენერატორი

C# ენაში განსაზღვრულია System.Random კლასი შემთხვევით რიცხვებთან სამუშაოდ. ცხრილში 5.1 მოყვანილია ამ კლასის მეთოდები, რომლებიც ახდენენ შემთხვევითი რიცხვების გენერირებას.

ცხრილი 5.1. Random კლასის მეთოდები.

მეთოდი	აღწერა
int Random.Next(int ცვლადი)	გასცემს არაუარყოფით შემთხვევით მთელ რიცხვს, რომელიც ნაკლებია მითითებული ცვლადის მნიშვნელობაზე
double Random.NextDouble()	გასცემს შემთხვევით წილად რიცხვს, რომლის მნიშვნელობა მოთავსებულია 0.0-სა და 1.0-ს შორის
void Random.NextBytes(byte[] მასივი)	მითითებულ მასივს შეავსებს შემთხვევითი რიცხვებით, რომელთა ტიპია byte

მოვიყვანოთ ორი პროგრამა, რომლებშიც შემთხვევითი რიცხვების გენერატორი გამოიყენება მასივების შესავსებად. ქვემოთ მოყვანილ პროგრამაში ორგანზომილებიანი მასივის შესავსებად გამოიყენება Next() მეთოდი.

```
{
//   პროგრამა 5.14
//   პროგრამაში შემთხვევითი რიცხვების გენერატორი გამოიყენება ორგანზომილებიანი
//   მასივის შესავსებად
label1.Text="";
int max = int.Parse(textBox1.Text);
int striqoni = int.Parse(textBox2.Text);
int sveti = int.Parse(textBox3.Text);
int[,] mas = new int[striqoni, sveti];
System.Random obieqti = new System.Random();

for ( int i = 0; i < striqoni; i++ )
    for ( int j = 0; j < sveti; j++ )
        mas[i, j] = obieqti.Next(max);
for ( int i = 0; i < striqoni; i++ )
{
    for ( int j = 0; j < sveti; j++ )
        label1.Text += mas[i, j].ToString() + " ";
    label1.Text += "\n";
}
}
```

ამ პროგრამაში ერთგანზომილებიანი masivi მასივის შესავსებად გამოიყენება NextBytes() მეთოდი.

```
{  
//   პროგრამა 5.15  
//   პროგრამაში ხდება შემთხვევითი რიცხვების გენერატორის გამოყენება  
//   ერთგანზომილებიანი მასივის შესავსებად  
label1.Text="";  
byte[] masivi = new byte[5];  
System.Random obieqti = new System.Random();  
  
obieqti.NextBytes(masivi);  
for ( int i = 0; i < masivi.Length; i++ )  
    label1.Text += masivi[i].ToString() + " ";  
}
```

თავი 6. მეთოდები უფრო დაწვრილებით. პოლიმორფიზმი

მეთოდისათვის ობიექტის გადაცემა

მეთოდს პარამეტრებად ჩვეულებრივი ტიპების მნიშვნელობების გარდა შეგვიძლია გადავცეთ ობიექტებიც. განვიხილოთ პროგრამა, რომელშიც ხდება სამკუთხედების შედარება. მათი ზომები ინახება Samkutxedi კლასის ობიექტებში.

```
// პროგრამა 6.1
// პროგრამაში ხდება მეთოდისათვის ობიექტის გადაცემის დემონსტრირება
class Samkutxedi
{
int gverdi1, gverdi2, gverdi3;
int perimetri;
public Samkutxedi(int par1, int par2, int par3)
{
    gverdi1 = par1;
    gverdi2 = par2;
    gverdi3 = par3;
    perimetri = gverdi1 + gverdi2 + gverdi3;
}
/*
    IgiveSamkutxedi() მეთოდი გასცემს true მნიშვნელობას, თუ Sam1 ობიექტში არის
    სამკუთხედის ზომების იგივე მნიშვნელობები, რაც გამომდახებელ ობიექტში
*/
public bool IgiveSamkutxedi(Samkutxedi Sam1)
{
if ( ( Sam1.gverdi1 == gverdi1 ) && ( Sam1.gverdi2 == gverdi2 ) &&
    ( Sam1.gverdi3 == gverdi3 ) ) return true;
    else return false;
}
/*
    IgivePerimetri() მეთოდი გასცემს true მნიშვნელობას, თუ Sam1 ობიექტში არის
    სამკუთხედის პერიმეტრის იგივე მნიშვნელობა, რაც გამომდახებელ ობიექტში
*/
public bool IgivePerimetri(Samkutxedi Sam1)
{
if ( Sam1.perimetri == perimetri ) return true;
    else return false;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int arg1, arg2, arg3;

arg1 = int.Parse(textBox1.Text);
arg2 = int.Parse(textBox2.Text);
arg3 = int.Parse(textBox3.Text);
```

```

Samkutxedi obieqti1 = new Samkutxedi(arg1, arg2, arg3);
Samkutxedi obieqti2 = new Samkutxedi(10, 2, 5);
Samkutxedi obieqti3 = new Samkutxedi(int.Parse(textBox1.Text),
    int.Parse(textBox2.Text), int.Parse(textBox3.Text));

label1.Text = "პირველი და მეორე სამკუთხედების ზომები ერთნაირია --- " +
    obieqti1.IgiveSamkutxedi(obieqti2).ToString();
label2.Text = "პირველი და მესამე სამკუთხედების ზომები ერთნაირია --- " +
    obieqti1.IgiveSamkutxedi(obieqti3).ToString();
label3.Text = "მეორე და მესამე სამკუთხედების პერიმეტრები ერთნაირია --- " +
    obieqti2.IgivePerimetri(obieqti3).ToString();
}

```

დაწვრილებით განვიხილოთ ამ პროგრამის Samkutxedi კლასში გამოცხადებული IgiveSamkutxedi() მეთოდი. მას აქვს Sam1 პარამეტრი, რომლის ტიპია Samkutxedi, ე.ი. Sam1 არის Samkutxedi ტიპის ობიექტზე მიმართვა. შედეგად, ამ მეთოდის გამოძახებისას Sam1 ცვლადს შეგვიძლია გადავცეთ კონკრეტულ ობიექტზე მიმართვა. მეთოდის (Sam1.gverdi1 == gverdi1) გამოსახულებაში ხდება Sam1 ობიექტის gverdi1 ცვლადის შედარება ტოლობაზე გამოძახებული ობიექტის gverdi1 ცვლადთან. ძირითად პროგრამაში ამ მეთოდის გამოძახება ხდება obieqti1.IgiveSamkutxedi(obieqti2) სტრიქონში. obieqti1 არის გამოძახებული ობიექტი. obieqti2-ის მიმართვა მიენიჭება Sam1 პარამეტრს, ამიტომ, Sam1.gverdi1 ჩანაწერში gverdi1 ეკუთვნის obieqti2-ს, ხოლო შედარების (==) მარჯვნივ მითითებული gverdi1 ცვლადი კი - obieqti1-ს, ანუ გამოძახებულ ობიექტს.

მეთოდისათვის არგუმენტების გადაცემის ხერხები

მეთოდს არგუმენტები შეგვიძლია ორი გზით გადავცეთ. პირველია არგუმენტის გადაცემა მნიშვნელობის მიხედვით. ამ შემთხვევაში არგუმენტის მნიშვნელობის ასლი ენიჭება მეთოდის პარამეტრს. ამ შემთხვევაში, პარამეტრისა და არგუმენტის მნიშვნელობები მეხსიერების სხვადასხვა უბანშია მოთავსებული. შედეგად, პარამეტრის მნიშვნელობის ცვლილება არ შეცვლის არგუმენტის მნიშვნელობას. მეორეა არგუმენტის გადაცემა მიმართვის მიხედვით. ამ შემთხვევაში მეთოდის პარამეტრს გადაეცემა არგუმენტზე მიმართვა (არგუმენტის მისამართი). შედეგად, პარამეტრის მნიშვნელობის შეცვლისას, შესაბამისად შეიცვლება არგუმენტის მნიშვნელობაც. ეს იმიტომ ხდება, რომ პარამეტრიც და არგუმენტიც მეხსიერების ერთსა და იმავე უბანს მიმართავს. განვიხილოთ არგუმენტის გადაცემის თითოეული გზის რეალიზების მაგალითები.

მეთოდისთვის ჩვეულებრივი ტიპის მქონე მნიშვნელობების გადაცემისას, როგორცაა int ან double, გამოიყენება გადაცემა მნიშვნელობის მიხედვით. შედეგად, პარამეტრის ცვლილება არ გამოიწვევს არგუმენტის ცვლილებას. განვიხილოთ პროგრამა.

```

// პროგრამა 6.2
// პროგრამაში ხდება მეთოდისათვის არგუმენტის გადაცემა მნიშვნელობის მიხედვით
class Chveulebrivi_Tipi
{
// ამ მეთოდის შესრულება არ გამოიწვევს არგუმენტის შეცვლას
public void Metodi(int par1, int par2)
{

```

```

        par1 = par1 + par2;
        par2 = - par2;
    }
}
private void button1_Click(object sender, System.EventArgs e)
{
    Chveulebrivi_Tipi obieqti = new Chveulebrivi_Tipi();
    int ricxvi1, ricxvi2;

    ricxvi1 = int.Parse(textBox1.Text);
    ricxvi2 = int.Parse(textBox2.Text);
    label1.Text = "არგუმენტების მნიშვნელობები მეთოდის გამოძახებამდე: " +
        ricxvi1.ToString() + " " + ricxvi2.ToString();
    obieqti.Metodi(ricxvi1, ricxvi2); // metodi მეთოდის გამოძახება
    label2.Text = "არგუმენტების მნიშვნელობები მეთოდის გამოძახების შემდეგ: " +
        ricxvi1.ToString() + " " + ricxvi2.ToString();
}

```

როგორც ვიცით, კლასის ტიპის (მიმართვითი ტიპის) ცვლადის შექმნის დროს ჩვენ სინამდვილეში ვქმნით ობიექტზე მიმართვას და არა თვით ობიექტს. ობიექტი მესსიერებაში იქმნება new ოპერატორით, ხოლო მესსიერების ამ უბანზე მიმართვა კი ენიჭება მიმართვითი ტიპის ცვლადს. როცა მიმართვითი ტიპის ცვლადს ვიყენებთ როგორც არგუმენტს, მაშინ მეთოდის პარამეტრი იღებს მიმართვას იმავე ობიექტზე, რომელსაც არგუმენტი მიმართავს. შედეგად, არგუმენტიც და პარამეტრიც ერთსა და იმავე ობიექტს მიმართავენ. მოყვანილ პროგრამაში პარამეტრის მნიშვნელობის ცვლილება იწვევს არგუმენტის მნიშვნელობის ცვლილებას.

// **პროგრამა 6.3**

// **პროგრამაში ხდება მეთოდისათვის არგუმენტის გადაცემა მიმართვის მიხედვით**

```

class Mimartviti_tipi
{
    public int cvladi1, cvladi2;
    public Mimartviti_tipi(int par1, int par2)
    {
        cvladi1 = par1;
        cvladi2 = par2;
    }
    //
    public void Shecvla(Mimartviti_tipi obieqti1)
    {
        obieqti1.cvladi1 = obieqti1.cvladi1 + obieqti1.cvladi2;
        obieqti1.cvladi2 = -obieqti1.cvladi2;
    }
}
private void button2_Click(object sender, System.EventArgs e)
{
    int ricxvi1, ricxvi2;

    ricxvi1 = int.Parse(textBox1.Text);

```



```

ricxvi2 = int.Parse(textBox2.Text);

Mimartviti_tipi obieqti2 = new Mimartviti_tipi(ricxvi1, ricxvi2);

label1.Text = "არგუმენტების მნიშვნელობები მეთოდის გამოძახებამდე: " +
    obieqti2.cvladi1.ToString() + " " + obieqti2.cvladi2.ToString();
obieqti2.Shecvla(obieqti2);
label2.Text = "არგუმენტების მნიშვნელობები მეთოდის გამოძახების შემდეგ: " +
    obieqti2.cvladi1.ToString() + " " + obieqti2.cvladi2.ToString();
}

```

პროგრამის `obieqti2.Shecvla(obieqti2)` სტრიქონში ხდება `obieqti2` ობიექტის `Shecvla()` მეთოდის გამოძახება. მისი არგუმენტია `obieqti2`. ამიტომ, `obieqti2`-ის მიმართვა ენიჭება `obieqti1` ცვლადს. შედეგად, `obieqti1` და `obieqti2` მიმართვითი ცვლადები მეხსიერების ერთსა და იმავე უბანს მიმართავენ. ამიტომ, `obieqti1` ობიექტის ცვლადების ცვლილება ავტომატურად გამოიწვევს `obieqti2` ობიექტის ცვლადების შეცვლას.

ref და out მოდიფიკატორები

როგორც ვიცით, ჩვეულებრივი ტიპის მნიშვნელობა (მაგალითად, `int` ან `char`) მეთოდს მნიშვნელობის მიხედვით გადაეცემა. ამ შემთხვევაში პარამეტრის მნიშვნელობის ცვლილება არ შეცვლის არგუმენტის მნიშვნელობას. მაგრამ, თუ გამოვიყენებთ `ref` და `out` საკვანძო სიტყვებს, მაშინ შეგვეძლება ჩვეულებრივი ტიპის ნებისმიერი მნიშვნელობა გადავცეთ მიმართვის მიხედვით, რაც მეთოდს მისცემს არგუმენტის შეცვლის შესაძლებლობას.

ჩვეულებრივი ტიპის მქონე მნიშვნელობების გადაცემა მიმართვის მიხედვით საჭიროა მაშინ, როცა მეთოდმა უნდა შეცვალოს თავისი არგუმენტის მნიშვნელობა და როცა მეთოდმა უნდა დააბრუნოს ერთზე მეტი მნიშვნელობა.

ref მოდიფიკატორი

`ref` მოდიფიკატორის გამოყენება იწვევს პარამეტრისათვის არგუმენტის გადაცემას მიმართვის მიხედვით. ამ დროს, პარამეტრს გადაეცემა არგუმენტის მისამართი. შედეგად, ფუნქციის მუშაობის დროს, მონაცემის დამუშავება ხდება მეხსიერების იმ უბანში, რომელიც გამოყოფილი იყო არგუმენტისთვის გამომძახებელ პროგრამაში. ამიტომ, პარამეტრის მნიშვნელობის ცვლილება გამოიწვევს შესაბამისი არგუმენტის მნიშვნელობის ცვლილებას. `ref` მოდიფიკატორი ეთითება მეთოდის გამოცხადებისას პარამეტრის წინ და მისი გამოძახებისას არგუმენტის წინ.

`ref` მოდიფიკატორის გამოყენებისას არსებობს ორი შეზღუდვა: ა) რადგან ხდება არგუმენტის მნიშვნელობის შეცვლა, ამიტომ არ შეიძლება ის გამოვაცხადოთ როგორც მუდმივა (კონსტანტა); ბ) ფუნქციისთვის გადასაცემი არგუმენტი უნდა იყოს ინიციალიზებული, ანუ უნდა ჰქონდეს მინიჭებული საწყისი მნიშვნელობა;

ქვემოთ მოყვანილ პროგრამაში მოყვანილია `Gacvla()` მეთოდი, რომელიც თავისი პარამეტრების მნიშვნელობებს ადგილს უცვლის. შედეგად, იცვლება არგუმენტების მნიშვნელობებიც. ყურადღება მივაქციოთ იმას, რომ მეთოდის გამოცხადებისას `ref` მოდიფიკატორი ეთითება პარამეტრის ტიპის წინ, ხოლო მეთოდის გამოძახებისას კი არგუმენტის წინ.

```
// პროგრამა 6.4
```

```
// პროგრამაში ხდება ref მოდიფიკატორის გამოყენების დემონსტრირება
```

```

class RicxvebisGacvla
{
//      მეთოდი ადგილებს უცვლის ორ პარამეტრს
public void Gacvla(ref int par1, ref int par2)
{
int temp;

temp = par1;
par1  = par2;
par2  = temp;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//
RicxvebisGacvla obieqti = new RicxvebisGacvla();
int ricxvi1 = int.Parse(textBox1.Text);
int ricxvi2 = int.Parse(textBox2.Text);

//      არგუმენტების მნიშვნელობები გაცვლამდე
label1.Text = ricxvi1.ToString() + " " + ricxvi2.ToString();
obieqti.Gacvla(ref ricxvi1, ref ricxvi2);
//      არგუმენტების მნიშვნელობები გაცვლის შემდეგ
label2.Text = ricxvi1.ToString() + " " + ricxvi2.ToString();
}

```

პროგრამაში Gacvla() მეთოდს გამოძახების დროს გადაეცემა ricxvi1 და ricxvi2 არგუმენტების მისამართები, რომლებიც შესაბამისად მიენიჭება par1 და par2 პარამეტრებს. შედეგად, ricxvi1 და par1 მიმართავენ მეხსიერების ერთსა და იმავე უბანს. იგივე ეხება ricxvi2 და par2 ცვლადებს. ამიტომ, par1 და par2 პარამეტრების ცვლილება ავტომატურად გამოიწვევს ricxvi1 და ricxvi2 არგუმენტების მნიშვნელობების შეცვლას.

out მოდიფიკატორი

მეთოდს შეუძლია გამომძახებელ პროგრამას დაუბრუნოს მხოლოდ ერთი მნიშვნელობა. იმისათვის, რომ მეთოდმა დაგვიბრუნოს რამდენიმე მნიშვნელობა, უნდა გამოვიყენოთ out მოდიფიკატორი. მეთოდის გამოცხადებისას out მოდიფიკატორი უნდა მივუთითოთ იმ პარამეტრის წინ, რომლის დაბრუნებაც გვინდა. out მოდიფიკატორიანი პარამეტრი გამოიყენება მეთოდიდან მხოლოდ ინფორმაციის გასაცემად. მეთოდმა ასეთ პარამეტრს აუცილებლად უნდა მიანიჭოს მნიშვნელობა მანამ, სანამ მართვას დაუბრუნებს გამომძახებელ პროგრამას. მოყვანილ პროგრამაში ნაჩვენებია out მოდიფიკატორის გამოყენების მაგალითი.

```

//      პროგრამა 6.5
//      პროგრამაში ხდება out მოდიფიკატორის გამოყენების დემონსტრირება
class Samkutxedi
{
int gverdi1;
int gverdi2;
int gverdi3;
public Samkutxedi(int par1, int par2, int par3)

```

```

{
gverdi1 = par1;
gverdi2 = par2;
gverdi3 = par3;
}
public int SamkuxediInfo(out bool TolGverda)
{
if ( ( gverdi1 == gverdi2 ) && ( gverdi1 == gverdi3 ) && ( gverdi2 == gverdi3 ) ) TolGverda = true;
    else TolGverda = false;
return gverdi1 + gverdi2 + gverdi3;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
int gv1 = int.Parse(textBox1.Text);
int gv2 = int.Parse(textBox2.Text);
int gv3 = int.Parse(textBox3.Text);
Samkuxedi obieqti = new Samkuxedi(gv1, gv2, gv3);
int perimetri;
bool ArisTolGverda;

perimetri = obieqti.SamkuxediInfo(out ArisTolGverda);
if ( ArisTolGverda ) label2.Text = "სამკუთხედი არის ტოლგვერდი";
    else label2.Text = "სამკუთხედი არ არის ტოლგვერდი";
label1.Text = " სამკუთხედის პერიმეტრი = " + perimetri.ToString();
}

```

პროგრამაში SamkuxediInfo() მეთოდი ორ მნიშვნელობას აბრუნებს. ერთი მნიშვნელობაა სამკუთხედის პერიმეტრი, მეორე კი - ArisTolGverda ლოგიკური ტიპის ცვლადი. მას ენიჭება true მნიშვნელობა იმ შემთხვევაში, თუ სამკუთხედს ტოლი გვერდები აქვს, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობა.

params მოდიფიკატორი

მეთოდის შექმნისას წინასწარ ვაფიქსირებთ მისთვის გადასაცემი არგუმენტების რაოდენობას. მაგრამ, რიგ შემთხვევებში საჭირო ხდება მეთოდისათვის ცვლადი რაოდენობის არგუმენტების გადაცემა. ასეთ შემთხვევაში, უნდა გამოვიყენოთ params მოდიფიკატორი. ის საშუალებას გვაძლევს მეთოდს გადავცეთ არგუმენტების ნებისმიერი რაოდენობა. params მოდიფიკატორი მასივს გამოაცხადებს როგორც პარამეტრს, რომელსაც შეუძლია მიიღოს 0, 1 და ა.შ. რაოდენობის არგუმენტი. მოყვანილ პროგრამაში maxSidide მეთოდს ცვლადი რაოდენობის არგუმენტები გადაეცემა. ის გასცემს არგუმენტებს შორის მაქსიმალურს.

```

//      პროგრამა 6.6
//      პროგრამაში ხდება params მოდიფიკატორის გამოყენების დემონსტრირება
class Maximumi
{
public int maxSidide(params int[ ] ricxvebi)
{

```

```

int max;
if ( ricxvebi.Length == 0 )
{
//      რადგან არგუმენტი არ არის მითითებული, ამიტომ მეთოდი აბრუნებს ნულს
return 0;
}
max = ricxvebi[0];
for ( int indexi = 1; indexi < ricxvebi.Length; indexi++ )
    if ( ricxvebi[indexi] > max ) max = ricxvebi[indexi];
return max;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
    Maximumi obieqti = new Maximumi();
    int maximaluri;
    int ricxvi1 = 15, ricxvi2 = 25;
    int ricxvi3 = Convert.ToInt32(textBox1.Text);
    int[] masivi = new int[] { 11, 22, -33, -44, 5 };

    //      მეთოდისთვის ორი არგუმენტის გადაცემა
    maximaluri = obieqti.maxSidide(ricxvi1, ricxvi2);
    label1.Text = maximaluri.ToString();
    //      მეთოდისთვის ოთხი არგუმენტის გადაცემა
    maximaluri = obieqti.maxSidide(ricxvi1, ricxvi2, -30, ricxvi3);
    label2.Text = maximaluri.ToString();
    //      მეთოდისთვის მასივის გადაცემა
    maximaluri = obieqti.maxSidide(masivi);
    label3.Text = maximaluri.ToString();
}

```

maxSidide() მეთოდს ყოველი გამოძახებისას გადაეცემა სხვადასხვა რაოდენობის არგუმენტები ricxvebi მასივის საშუალებით. არგუმენტების ტიპები უნდა იყოს თავსებადი ricxvebi მასივის ტიპთან. მაგალითად, ორივე უნდა იყოს მთელირიცხვა, ან წილადი და ა.შ.

თუ maxSidide() მეთოდს არ გადავცემთ არგუმენტებს, მაშინ ის ჩაიწერება არგუმენტების გარეშე:

```
maximumi = obieqti.maxSidide();
```

ამ შემთხვევაში, საჭიროა ვაკონტროლოთ params მოდიფიკატორით გამოცხადებული პარამეტრის მნიშვნელობა. ამიტომ, maxSidide() მეთოდში მასივთან მიმართვამდე უნდა შევამოწმოთ შეიცავს თუ არა ის ერთ ელემენტს მაინც. ამისათვის, ვიყენებთ

```
if ( ricxvebi.Length == 0 )
```

ოპერატორს.

მეთოდში შეგვიძლია, აგრეთვე, მივუთითოთ როგორც ჩვეულებრივი, ისე ცვლადი სიგრძის პარამეტრები. როცა მეთოდში მითითებულია ჩვეულებრივი და ცვლადი სიგრძის პარამეტრები, მაშინ ცვლადი სიგრძის პარამეტრი პარამეტრების სიაში უნდა იყოს უკანასკნელი. ამასთან, მეთოდს უნდა ჰქონდეს მხოლოდ ერთი ცვლადი სიგრძის პარამეტრი. ქვემოთ მოყვანილ პროგრამაში ShowArgs() მეთოდს ორი პარამეტრი აქვს - ერთი string ტიპის, მეორე კი - int ტიპის ცვლადი სიგრძის.

```
//      პროგრამა 6.7
```

```

//      პროგრამაში ხდება ჩვეულებრივი და ცვლადი სიგრძის პარამეტრების
//      ერთობლივად გამოყენების დემონსტრირება
class ChemiKlasi
{
string str1;
public string Metodi(string striqoni, params int[ ] ricxvebi)
{
int jami = 0;

foreach ( int ricxvi in ricxvebi )
    jami += ricxvi;
str1 = striqoni + " ";
str1 += jami.ToString();
return str1;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti = new ChemiKlasi();

label1.Text=obieqti.Metodi("მეთოდს 5 არგუმენტი გადაეცა. მათი ჯამია: ", 1, 2, 3, 4, 5);
label2.Text = obieqti.Metodi("მეთოდს 2 არგუმენტი გადაეცა. მათი ჯამია: ", 6, 7);
}

```

მეთოდის მიერ ობიექტის დაბრუნება

მეთოდს შეუძლია დააბრუნოს როგორც ნებისმიერი ტიპის მონაცემი, ისე კლასის ობიექტიც. მაგალითისათვის განვიხილოთ ორი პროგრამა. პირველ პროგრამაში მეთოდი გასცემს სტრიქონების მასივის ელემენტს, მეორე პროგრამაში კი - ჩვენს მიერ შექმნილი კლასის ობიექტს.

```

//      პროგრამა 6.8
//      პროგრამაში ხდება მეთოდის მიერ სტრიქონის დაბრუნების დემონსტრირება
class ChemiKlasi
{
string[ ] xili = { "ვაშლი", "მსხალი", "ლევვი", "ყურბენი" };
//
public string StriqonisDabruneba(int indexi)
{
if ( ( indexi >= 0 ) & ( indexi < xili.Length ) ) return xili[indexi];
    else return "შეტანილია არასწორი ინდექსი";
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება ობიექტის დაბრუნების დემონსტრირება
ChemiKlasi obieqti = new ChemiKlasi();

```

```

label1.Text = obieqti.StriqonisDabruneba(3);
label2.Text = obieqti.StriqonisDabruneba(15);
}

//      პროგრამა 6.9
//      პროგრამაში ხდება მეთოდის მიერ ობიექტის დაბრუნების დემონსტრირება
class ChemiKlasi_1
{
public string xilis_saxeli;
public int kodi;
public ChemiKlasi_1(string par1, int par2)
{
xilis_saxeli = par1;
kodi = par2;
}
}
class ChemiKlasi_2
{
string[] xili = { "ვაშლი ", "მსხალი ", "ლევვი ", "ყურძენი " };
int[] kodebi = { 1, 0, 3, 7};
//
public ChemiKlasi_1 ObieqtisDabruneba(int indexi)
{
if ( ( indexi >= 0 ) && ( indexi < xili.Length ) )
//      ChemiKlasi_1 ობიექტის დაბრუნება
return new ChemiKlasi_1(xili[indexi], kodebi[indexi]);
else return new ChemiKlasi_1("მეტანილია არასწორი კოდი ", indexi);
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi_2 obieqti1 = new ChemiKlasi_2();
ChemiKlasi_1 obieqti2;

obieqti2 = obieqti1.ObieqtisDabruneba(3);
label1.Text = obieqti2.xilis_saxeli + obieqti2.kodi.ToString();
obieqti2 = obieqti1.ObieqtisDabruneba(20);
label2.Text = obieqti2.xilis_saxeli + obieqti2.kodi.ToString();
}

```

მეთოდის გადატვირთვა. პოლიმორფიზმი

C# ენაში ერთი კლასის შიგნით ორ ან მეტ მეთოდს შეიძლება ერთნაირი სახელი ჰქონდეს იმ პირობით, რომ ან პარამეტრების რაოდენობა უნდა იყოს სხვადასხვა ან პარამეტრებს სხვადასხვა ტიპი უნდა ჰქონდეთ. ასეთ მეთოდებს გადატვირთვადი ეწოდებათ, ხოლო ერთნაირი

სახელის მქონე მეთოდების განსაზღვრის პროცესს კი მეთოდის გადატვირთვა. მეთოდების გადატვირთვა არის პოლიმორფიზმის რეალიზების ერთ-ერთი საშუალება. გადატვირთვადი მეთოდის გამოძახებისას სრულდება ის მეთოდი, რომლის პარამეტრებიც ემთხვევა არგუმენტებს. გადატვირთვადი მეთოდები შეიძლება აბრუნებდნენ, სხვადასხვა ტიპის მონაცემებს. განვიხილოთ მაგალითი.

```
// პროგრამა 6.10
```

```
// პროგრამაში ხდება მეთოდის გადატვირთვის დემონსტრირება
```

```
class Gadatvirtva
{
// მეთოდს პარამეტრები არ აქვს და აბრუნებს სტრიქონს
public string Metodi()
{
return "გამოცხადებული იქნა უპარამეტრო მეთოდი";
}
// მეთოდს double ტიპის ერთი პარამეტრი აქვს და აბრუნებს მის კვადრატს
public double Metodi(double par1)
{
return Math.Pow(par1, 2);
}
// მეთოდს int ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს
public int Metodi(int par1, int par2)
{
return par1 + par2;
}
// მეთოდს double ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს
public double Metodi(double par1, double par2)
{
return par1 + par2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Gadatvirtva obieqti = new Gadatvirtva();
int ricxvi1;
double wiladi1, wiladi2;
string striqoni;

// Metodi მეთოდის ყველა ვერსიის გამოძახება
striqoni = obieqti.Metodi();
label1.Text = striqoni;
wiladi2 = obieqti.Metodi(5.7);
label2.Text = "გამოძახებული იქნა ერთპარამეტრიანი მეთოდი. შედეგი = " + wiladi2.ToString();
ricxvi1 = obieqti.Metodi(5, 10);
label3.Text = "გამოძახებული იქნა ორპარამეტრიანი მეთოდი. შედეგი = " + ricxvi1.ToString();
wiladi1 = obieqti.Metodi(5.5, 10.10);
label4.Text = "გამოძახებული იქნა ორპარამეტრიანი მეთოდი. შედეგი = " + wiladi1.ToString();
}
```

პროგრამაში ოთხჯერ ხდება Metodi() მეთოდის გადატვირთვა. პირველ ვერსიას არ აქვს პარამეტრი. მეორე ვერსიას აქვს ერთ მთელი ტიპის პარამეტრი, მესამეს - ორი მთელი ტიპის პარამეტრი და მეოთხეს კი - ორი წილადი პარამეტრი.

თუ პარამეტრების რაოდენობა თანაბარია და არგუმენტებსა და პარამეტრებს განსხვავებული ტიპები აქვს, მაშინ სრულდება ტიპების ავტომატური გარდაქმნა. განვიხილოთ მაგალითი.

```
// პროგრამა 6.11
// პროგრამაში სრულდება ტიპების ავტომატური გარდაქმნა მეთოდის
// გადატვირთვის დროს
class Gadatvirtva
{
public int Metodi( int par1 )
{
    return par1;
}
public double Metodi( double par1 )
{
    return par1;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Gadatvirtva obieqti = new Gadatvirtva();
int i1 = 15;
double d1 = 15.15;
byte b1 = 33;
short s1 = 20;
float f1 = 5.5F;
// აქ სრულდება Metodi(int par1) მეთოდი
label1.Text = obieqti.Metodi(i1).ToString();
// აქ სრულდება Metodi(double par1) მეთოდი
label2.Text = obieqti.Metodi(d1).ToString();
// აქ სრულდება Metodi(int par1) მეთოდი
label3.Text = obieqti.Metodi(b1).ToString();
// აქ სრულდება Metodi(int par1) მეთოდი
label4.Text = obieqti.Metodi(s1).ToString();
// აქ სრულდება Metodi(double par1) მეთოდი
label5.Text = obieqti.Metodi(f1).ToString();
}
```

პროგრამაში Metodi() მეთოდის ერთ ვერსიას აქვს int ტიპის პარამეტრი, მეორეს კი - double ტიპის. მათი გამოძახება ხდება შესაბამისად int, double, byte, short და float ტიპის მქონე არგუმენტებისათვის. byte და short ტიპის მნიშვნელობების გადაცემისას ისინი ავტომატურად გარდაიქმნება int ტიპად. მეთოდისათვის float ტიპის მნიშვნელობის გადაცემისას ის გარდაიქმნება double ტიპად.

მეთოდების გადატვირთვისას შეგვიძლია ref და out მოდიფიკატორების გამოყენება. ამის დემონსტრირება ქვემოთ მოყვანილ პროგრამაში ხდება.

```
// პროგრამა 6.12
// პროგრამაში ხდება ref და out მოდიფიკატორების გამოყენება მეთოდის
```



```

// გადატვირთვის დროს
class Gadatvirtva
{
public int Metodi(int par1, out int par2)
{
    par2 = par1 * par1;
    return par1;
}
public int Metodi(ref int par3)
{
    return par3;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Gadatvirtva obieqti = new Gadatvirtva();
int ricxvi1, ricxvi2, ricxvi3, ricxvi4, ricxvi5;

ricxvi2 = int.Parse(textBox1.Text);
ricxvi4 = int.Parse(textBox2.Text);
ricxvi1 = obieqti.Metodi(ricxvi2, out ricxvi5);
label1.Text = ricxvi1.ToString() + " " + ricxvi5.ToString();
ricxvi3 = obieqti.Metodi(ref ricxvi4);
label2.Text = ricxvi3.ToString();
}

```

კონსტრუქტორების გადატვირთვა

მეთოდების მსგავსად შესაძლებელია კონსტრუქტორების გადატვირთვაც. ეს საშუალებას გვაძლევს ობიექტები შევქმნათ სხვადასხვა საშუალებებით. მოყვანილ პროგრამაში ხდება გადატვირთვადი კონსტრუქტორის დემონსტრირება.

```

// პროგრამა 6.13
// პროგრამაში ხდება გადატვირთვადი კონსტრუქტორის დემონსტრირება
class ChemiKlasi
{
    public int ricxvi;
    public ChemiKlasi()
    {
        ricxvi = 0;
    }
    public ChemiKlasi(int par1)
    {
        ricxvi = par1 * par1;
    }
    public ChemiKlasi(double par2)
    {

```

```

    ricxvi = ( int ) par2 + 10;
}
public ChemiKlasi(int par3, int par4)
{
    ricxvi = par3 * par4;
}
}

private void button2_Click(object sender, EventArgs e)
{
    int ricxvi = Convert.ToInt32(textBox1.Text);

    ChemiKlasi obieqti1 = new ChemiKlasi();
    label1.Text = "" + obieqti1.ricxvi.ToString();
    ChemiKlasi obieqti2 = new ChemiKlasi(ricxvi);
    label2.Text = "" + obieqti2.ricxvi.ToString();
    ChemiKlasi obieqti3 = new ChemiKlasi(28.89);
    label3.Text = "" + obieqti3.ricxvi.ToString();
    ChemiKlasi obieqti4 = new ChemiKlasi(3, 5);
    label4.Text = "" + obieqti4.ricxvi.ToString();
}

```

პროგრამაში მოყვანილია ChemiKlasi() მეთოდის ოთხი ვერსია. საჭირო მათგანის არჩევა ხდება არგუმენტების მიხედვით.

მეთოდების გადატვირთვა იძლევა იმის საშუალებას, რომ ერთი ობიექტის ინიციალიზაციისათვის კონსტრუქტორმა მეორე ობიექტი გამოიყენოს. მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

// პროგრამა 6.14

// პროგრამაში ერთი ობიექტის ინიციალიზაციისათვის კონსტრუქტორი მეორე

// ობიექტს იყენებს

```

class Shekreba
{
    public int jami;
    // ობიექტის შესაქმნელად კონსტრუქტორი იყენებს int ტიპის პარამეტრს
    public Shekreba(int raodenoba)
    {
        jami = 0;
        for ( int indexi = 1; indexi <= raodenoba; indexi++ )
            jami += indexi;
    }
    // ობიექტის ინიციალიზაციისათვის კონსტრუქტორი იყენებს სხვა ობიექტს,
    // რომელიც მას გადაეცემა, როგორც პარამეტრი
    public Shekreba(Shekreba obieqti)
    {
        jami = obieqti.jami;
    }
}

private void button1_Click(object sender, System.EventArgs e)

```

```

{
int ricxvi1 = int.Parse(textBox1.Text);
Shekreba obieqti1 = new Shekreba(ricxvi1);
Shekreba obieqti2 = new Shekreba(obieqti1);

label1.Text = "რიცხვების ჯამი 0-დან " + ricxvi1.ToString() + "-მდე = " + obieqti1.jami.ToString();
label2.Text = "ცვლადის მნიშვნელობა = " + obieqti2.jami.ToString();
}

```

პროგრამაში პირველი კონსტრუქტორის არგუმენტია მთელი რიცხვი, მეორე კონსტრუქტორის კი - ობიექტი. პირველი კონსტრუქტორი ანგარიშობს ჯამს ნულიდან textBox1 კომპონენტში შეტანილ რიცხვამდე. მეორე კონსტრუქტორი jami ცვლადს ანიჭებს მითითებული ობიექტის შესაბამისი ცვლადის მნიშვნელობას და აღარ ხდება საჭირო ჯამის მეორეჯერ გამოთვლა.

გადატვირთვადი კონსტრუქტორის გამოძახება this სიტყვის გამოყენებით

რიგ შემთხვევებში გადატვირთვადი კონსტრუქტორის გამოძახებისას სასარგებლოა გამოვიყენოთ ერთი კონსტრუქტორის მიერ მეორის გამოძახება. C# ენაში ეს რეალიზდება this სიტყვის გამოყენებით. მისი სინტაქსია:

კონსტრუქტორის სახელი(პარამეტრების_სია) : this(არგუმენტების_სია)

```
{
```

კონსტრუქტორის კოდი

```
}
```

გამომძახებელი კონსტრუქტორის შესრულებისას ჯერ გამოიძახება ის გადატვირთვადი კონსტრუქტორი, რომელშიც პარამეტრების სია ემთხვევა არგუმენტების სიას, შემდეგ კი შესრულდება გამომძახებელი კონსტრუქტორის დანარჩენი ოპერატორები. განვიხილოთ პროგრამა.

```
// პროგრამა 6.15
```

```
// პროგრამაში ხდება გადატვირთვადი კონსტრუქტორის გამოძახება this
```

```
// სიტყვის გამოყენებით
```

```
class ChemiKlasi
```

```
{
```

```
public int cvladi1, cvladi2;
```

```
public ChemiKlasi() : this(0,0)
```

```
{
```

```
MessageBox.Show("ეს მეთოდი განსაზღვრულია ChemiKlasi() კონსტრუქტორში");
```

```
}
```

```
public ChemiKlasi(ChemiKlasi obieqti) : this(obieqti.cvladi1, obieqti.cvladi2)
```

```
{
```

```
MessageBox.Show("ეს მეთოდი განსაზღვრულია ChemiKlasi(obieqti.cvladi1, obieqti.cvladi2) კონსტრუქტორში");
```

```
}
```

```
public ChemiKlasi(int par1, int par2)
```

```
{
```

```
MessageBox.Show("ეს მეთოდი განსაზღვრულია ChemiKlasi(int, int) კონსტრუქტორში");
```

```

cvladi1 = par1;
cvladi2 = par2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      ხდება this სიტყვის გამოყენების დემონსტრირება ერთი კონსტრუქტორის
//      გამოსაძახებლად მეორის მიერ
int ricxvi1 = int.Parse(textBox1.Text);
int ricxvi2 = int.Parse(textBox2.Text);
ChemiKlasi obieqti1 = new ChemiKlasi();
ChemiKlasi obieqti2 = new ChemiKlasi(ricxvi1, ricxvi2);
ChemiKlasi obieqti3 = new ChemiKlasi(obieqti2);

label1.Text = obieqti1.cvladi1.ToString() + " " + obieqti1.cvladi2.ToString();
label2.Text = obieqti2.cvladi1.ToString() + " " + obieqti2.cvladi2.ToString();
label3.Text = obieqti3.cvladi1.ToString() + " " + obieqti3.cvladi2.ToString();
}

```

ChemiKlasi კლასში ერთადერთი კონსტრუქტორია - ChemiKlasi(int, int). დანარჩენი ორი კონსტრუქტორი იყენებს this სიტყვას და ახდენენ ChemiKlasi(int, int) კონსტრუქტორის გამოძახებას. მაგალითად, obieqti1 ობიექტის შექმნისას გამოიძახება მისი კონსტრუქტორი - ChemiKlasi(), რომელიც თავის მხრივ გამოიძახებს this(0,0) კონსტრუქტორს. ე.ი. this სიტყვა ეთითება ChemiKlasi(0,0) კონსტრუქტორის სრული სახელის ნაცვლად. შემდეგ შესრულდება ChemiKlasi() კონსტრუქტორის კოდი, კერძოდ, MessageBox() მეთოდი. ანალოგიურად იქმნება obieqti2 და obieqti3 ობიექტები.

გადატვირთვადი კონსტრუქტორების გამოძახებისათვის this სიტყვის გამოყენების ერთ-ერთი მიზეზი არის კოდის დუბლირების თავიდან აცილების შესაძლებლობა. უკანასკნელ მაგალითში this მიმართვის გამოყენების გარეშე მოგვიწევდა მაინიციალიზებული ოპერატორების მიმდევრობის (cvladi1 = par1; cvladi2 = par2) გამეორება სამივე კონსტრუქტორში.

რეკურსია

მეთოდს შეუძლია თავისი თავის გამოძახება. ამ პროცესს რეკურსია ეწოდება, ხოლო მეთოდს, რომელიც თავის თავს იძახებს - რეკურსიული. რეკურსიული მეთოდის საკვანძო კომპონენტი არის ოპერატორი, რომელიც ამავე მეთოდს იძახებს.

რეკურსიის კლასიკური მაგალითია ფაქტორიალის გამოთვლა. N რიცხვის ფაქტორიალი აღინიშნება N!-ით და არის რიცხვების ნამრავლი 1-დან N-მდე. მაგალითად, 4-ის ფაქტორიალი არის $1 \times 2 \times 3 \times 4 = 24$. ქვემოთ მოყვანილ პროგრამაში წარმოდგენილი რეკურსიული მეთოდი განკუთვნილია რიცხვის ფაქტორიალის გამოთვლისათვის.

```

//      პროგრამა 6.16
//      პროგრამაში ხდება რეკურსიის დემონსტრირება
class Factorial
{
//      რეკურსიული მეთოდი
public int Factor(int ricxvi)
{

```

```

int shedegi;
if ( ricxvi == 1 ) return 1;
shedegi = Factor(ricxvi -1) * ricxvi;
return shedegi;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Factorial obieqti = new Factorial();
int number = Convert.ToInt32(textBox1.Text);
int fact1;

fact1 = obieqti.Factor(number);
label1.Text = fact1.ToString();
}

```

რეკურსიული მეთოდის შედგენისას სწორად უნდა განისაზღვროს რეკურსიის დამთავრების პირობა. ამ დროს უნდა შეწყდეს მეთოდის გამოძახება და მან უნდა დააბრუნოს მნიშვნელობა. ჩვენს შემთხვევაში, რეკურსიული გამოძახება მთავრდება მაშინ, როცა არგუმენტი 1-ის ტოლი ხდება. ამ დროს მეთოდი აბრუნებს 1-ს.

რეკურსიული მეთოდების გამოყენებას აქვს როგორც დადებითი, ისე უარყოფითი მხარეები. უარყოფითი მხარეა ის, რომ რეკურსიული პროგრამები შედარებით ნელა სრულდება. გარდა ამისა, თუ რეკურსიული გამოძახებების რაოდენობა ძალიან დიდია, მაშინ შეიძლება აღიძრას განსაკუთრებული სიტუაცია (შეცდომა). ეს ძირითადად მაშინ ხდება, როცა არასწორად არის განსაზღვრული რეკურსიის დამთავრების პირობა: მეთოდის გამოძახების ნაცვლად უნდა მოხდეს მნიშვნელობის დაბრუნება.

დადებითი მხარეა ის, რომ რიგი ალგორითმებისა რეკურსიულად შეიძლება რეალიზებული იყოს უფრო ეფექტურად და მარტივად. მაგალითად, სწრაფი დახარისხების, კატალოგებში ფაილების ძებნისა და ა.შ. რეკურსიული მიდგომა ხშირად გამოიყენება, აგრეთვე, ხელოვნური ინტელექტის სფეროში.

სტატიკური კომპონენტები

static მოდიფიკატორი

ზოგჯერ საჭიროა კლასის ისეთი წევრის განსაზღვრა, რომელიც გამოყენებული იქნება ამავე კლასის ნებისმიერი ობიექტისაგან დამოუკიდებლად. ჩვეულებრივ, კლასის წევრთან მიმართვა ხორციელდება ამ კლასის ობიექტის გამოყენებით, მაგრამ არსებობს კლასის წევრთან მიმართვის შესაძლებლობა ობიექტთან მიმართვის გარეშეც. კლასის ასეთ წევრებს სტატიკური წევრები ეწოდებათ და მათი გამოცხადებისათვის გამოიყენება static სიტყვა. თუ კლასის წევრი განისაზღვრება static მოდიფიკატორით, მაშინ ის მისაწვდომი ხდება ამ კლასის ობიექტების შექმნამდე. static სიტყვა შეგვიძლია გამოვიყენოთ როგორც მეთოდების, ისე ცვლადების გამოცხადებისათვის.

კლასის სტატიკურ წევრთან მიმართვისათვის უნდა მივუთითოთ კლასის სახელი, ოპერატორი წერტილი (.) და წევრის სახელი. სტატიკურ მეთოდთან მიმართვა ანალოგიურად სრულდება. სტატიკური მეთოდი ჩვეულებრივი მეთოდისაგან იმით განსხვავდება, რომ ის შეიძლება გამოძახებული იყოს მისი კლასის სახელის გამოყენებით ამ კლასის რაიმე ობიექტის შექმნის გარეშე.

სტატიკური ცვლადი ფაქტიურად გლობალური ცვლადია. მისი კლასის ობიექტების გამოცხადებისას არ იქმნება ასეთი ცვლადის ასლი. სამაგიეროდ კლასის ყველა ობიექტი ერთობლივად იყენებს სტატიკურ ცვლადს, რომლის ინიციალიზება ხდება მისი კლასის ჩატვირთვისას. თუ საწყისი მნიშვნელობა არ არის აშკარად მითითებული, გაჩუმებით ასეთ ცვლადს ენიჭება მნიშვნელობა 0 (რიცხვითი ტიპის ცვლადისთვის), null მნიშვნელობა (მიმართვითი ტიპის ცვლადისთვის) ან false მნიშვნელობა (bool ტიპის ცვლადისთვის). ამრიგად, static მოდიფიკატორიან ცვლადს ყოველთვის აქვს მნიშვნელობა.

ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია სტატიკურ ცვლადთან და მეთოდთან მუშაობა.

// **პროგრამა 6.17**

// **პროგრამაში ხდება სტატიკურ ცვლადთან და მეთოდთან მუშაობის დემონსტრირება**

```
class ChemiKlasi
{
// სტატიკური ცვლადის გამოცხადება
public static int Statikuri_cvladi = 100;
// სტატიკური მეთოდის გამოცხადება
public static int Statikuri_Metodi()
{
return Statikuri_cvladi / 2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
label1.Text = ChemiKlasi.Statikuri_cvladi.ToString();
// სტატიკურ ცვლადს მნიშვნელობა ენიჭება
ChemiKlasi.Statikuri_cvladi = int.Parse(textBox1.Text);
label2.Text = ChemiKlasi.Statikuri_cvladi.ToString();
label3.Text = ChemiKlasi.Statikuri_Metodi().ToString();
}
```

როგორც ვხედავთ სტატიკური ცვლადის ინიციალიზება ხდება მისი კლასის რაიმე ობიექტის შექმნის გარეშე.

სტატიკურ მეთოდებს შემდეგი შეზღუდვები აქვთ:

- არ აქვთ this მიმართვა;
- შეუძლიათ უშუალოდ გამოიძახონ მხოლოდ სტატიკური მეთოდები და არ შეუძლიათ გამოიძახონ თავიანთი კლასის ობიექტის მეთოდები;
- შეუძლიათ პირდაპირ მიმართონ მხოლოდ სტატიკურ მონაცემებს.

სტატიკურ მეთოდს მხოლოდ თავისი კლასის ობიექტის გამოყენებით შეუძლია მიმართოს თავისი კლასის ობიექტის მეთოდებს და ცვლადებს (ე.ი. ობიექტის ცვლადთან ან მეთოდთან მიმართვისათვის საჭიროა ამ ობიექტის სახელის მითითება). ქვემოთ მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

// **პროგრამა 6.18**

// **პროგრამაში ხდება სტატიკური მეთოდის მიერ ჩვეულებრივი მეთოდის**

// **გამოძახების დემონსტრირება**

```
class ChemiKlasi
{
int chveulebrivi_cvladi; // ჩვეულებრივი ცვლადის გამოცხადება
static int statikuri_cvladi = 555; // სტატიკური ცვლადის გამოცხადება
// ჩვეულებრივი მეთოდი
```

```

void ChveulebriviMetodi(Label lab1)
{
    chveulebrivi_cvladi = 50;
    statikuri_cvladi = 70;
    lab1.Text = "ჩვეულებრივი ცვლადი - " + chveulebrivi_cvladi.ToString() +
        "\nსტატიკური ცვლადი - " + statikuri_cvladi.ToString();
}
// სტატიკური მეთოდი
public static void StatikuriMetodi(ChemiKlasi objeqti, Label lab2)
{
    objeqti.ChveulebriviMetodi(lab2); // მეთოდის ეს გამოძახება ნამდვილია
}
}
private void button1_Click(object sender, System.EventArgs e)
{
    ChemiKlasi obj = new ChemiKlasi();

    ChemiKlasi.StatikuriMetodi(obj, label1);
}

```

მუდმივები

მუდმივას გამოცხადებისათვის გამოიყენება const საკვანძო სიტყვა. კლასი შეიძლება შეიცავდეს მუდმივებს, რომლებიც ავტომატურად არიან სტატიკური. მუდმივას აუცილებლად უნდა მივანიჭოთ მნიშვნელობა. ამის შემდეგ, მისი მნიშვნელობის შეცვლა აღარ შეიძლება.

მოყვანილ პროგრამაში ხდება მუდმივასთან მუშაობის დემონსტრირება.

```

// პროგრამა 6.19
// პროგრამაში ხდება მუდმივასთან მუშაობის დემონსტრირება
class ChemiKlasi
{
    public const int mudmiva = 5; // მუდმივას გამოცხადება
}
{
    label1.Text = ChemiKlasi.mudmiva.ToString();
}

```

მხოლოდწაკითხვადი ველები

მხოლოდწაკითხვადი ველი შეიძლება ეკუთვნოდეს კლასს ან მის ეგზემპლარს. თუ მხოლოდწაკითხვადი ველი ეკუთვნის კლასს, მაშინ ის უნდა გამოვაცხადოთ როგორც სტატიკური. თუ მხოლოდწაკითხვადი ველი ეკუთვნის კლასის ეგზემპლარს (ობიექტს), მაშინ მას მნიშვნელობა კონსტრუქტორმა უნდა მიანიჭოს.

მოყვანილ პროგრამაში ხდება მხოლოდწაკითხვად ველთან მუშაობის დემონსტრირება.

```

// პროგრამა 6.20
// პროგრამაში ხდება მხოლოდწაკითხვად ველთან მუშაობის დემონსტრირება
class ChemiKlasi
{

```

```

// მხოლოდწაკითხვადი ველის გამოცხადება
public readonly int MxolodWakitxvadiVeli;
// სტატიკური მხოლოდწაკითხვადი ველის გამოცხადება
public static readonly int mudmiva = 5;
// კონსტრუქტორის გამოცხადება
public ChemiKlasi(int par1)
    {
        MxolodWakitxvadiVeli = par1;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = ChemiKlasi.mudmiva.ToString();
    int ricxvi = int.Parse(textBox1.Text);
    ChemiKlasi obieqti = new ChemiKlasi(ricxvi);
    label2.Text = obieqti.MxolodWakitxvadiVeli.ToString();
}

```


თავი 7. მემკვიდრეობითობა. ვირტუალური მეთოდები

მემკვიდრეობითობის საფუძვლები

როგორც ვიცით, ობიექტზე ორიენტირებული პროგრამირების ერთ-ერთი პრინციპია მემკვიდრეობითობა. მისი გამოყენებით შეგვიძლია ახალი კლასების შექმნა, რომლებიც წარმოადგენენ საბაზო კლასის მემკვიდრე კლასებს. მემკვიდრე კლასს დამატებული აქვს საბაზო კლასისაგან განსხვავებული თვისებები. მაგალითად, შეგვიძლია შევექმნათ კლასი „ავტომობილი“, რომელსაც აქვს ისეთი საერთო მახასიათებლები, როგორიცაა ძრავის სიმძლავრე, საწვავის ხარჯი 100 კილომეტრზე, მაქსიმალური სიჩქარე. ამ კლასს შეიძლება ჰქონდეს ორი მემკვიდრე კლასი - „სატვირთო“, რომელშიც ჩნდება მახასიათებელი - ტვირთამწეობა, და „ავტობუსი“, რომელშიც ჩნდება მახასიათებელი - გადასაყვანი მგზავრების რაოდენობა.

C# ენაში კლასს, რომლის ცვლადები და მეთოდები ავტომატურად ხდება სხვა ახალი შესაქმნელი კლასის წევრები, წინაპარი (საბაზო) კლასი ეწოდება. კლასს, რომელიც მემკვიდრეობით იღებს წინაპარი კლასის არსებულ წევრებს და მათ ახალ წევრებს უმატებს, მემკვიდრე კლასი ეწოდება. ამრიგად, მემკვიდრე კლასი მემკვიდრეობით იღებს წინაპარ კლასში განსაზღვრულ ყველა ცვლადს, მეთოდს, თვისებასა და ინდექსატორს და მათ უმატებს თავის საკუთარ წევრებს.

მემკვიდრე კლასის გამოცხადების სინტაქსია:

```
class მემკვიდრე_კლასის_სახელი : წინაპარი_კლასის_სახელი
{
კლასის_კოდი
}
```

როგორც ვხედავთ, მემკვიდრე კლასის განსაზღვრისას ორი წერტილის შემდეგ ეთითება წინაპარი კლასის სახელი. თითოეული მემკვიდრე კლასისათვის შეგვიძლია მივუთითოთ მხოლოდ ერთი წინაპარი კლასი. ასეთი გზით შესაძლებელია მემკვიდრეობითობის იერარქიის შექმნა, რომელშიც მემკვიდრე კლასს თვითონ შეუძლია გახდეს წინაპარი სხვა კლასისთვის. არც ერთი კლასი არ შეიძლება იყოს თავისი თავის წინაპარი.

მემკვიდრეობითობის უპირატესობა იმაშია, რომ კლასის შექმნის შემდეგ, რომელიც განსაზღვრავს საერთო მახასიათებლებს, ის შეგვიძლია გამოვიყენოთ ნებისმიერი რაოდენობის მემკვიდრე კლასების შესაქმნელად, რომლებიც დამატებით შეიცავენ უნიკალურ მახასიათებლებს. შედეგად, მემკვიდრე კლასში აღარ ხდება საჭირო წინაპარ კლასში გამოცხადებული წევრების განმეორებით გამოცხადება.

განვიხილოთ პროგრამა, რომელშიც ხდება მემკვიდრეობითობის დემონსტრირება. მასში იქმნება Sibrtye საბაზო კლასი, რომელიც ინახავს გეომეტრიული ფიგურის გვერდების მნიშვნელობებს და მეთოდს, რომელიც გასცემს მათ მნიშვნელობებს. იქმნება Sibrtye საბაზო კლასის Samkutxedi და Otxkutxedi მემკვიდრე კლასები. ამ კლასებში შესაბამისად გამოითვლება სამკუთხედისა და ოთხკუთხედის პერიმეტრი და ფართობი.

```
// პროგრამა 7.1
// პროგრამაში ხდება მემკვიდრეობითობის დემონსტრირება
// საბაზო კლასი შეიცავს გეომეტრიული ფიგურის გვერდებს
class Sibrtye
{
public double gverdi1;
public double gverdi2;
public double gverdi3;
```

```

public string Naxva()
{
return "პირველი და მეორე გვერდი \n" + gverdi1.ToString()+ " " + gverdi2.ToString();
}
}
//      Samkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
class Samkutxedi : Sibrtye          //      Samkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
{
public string tipi;
public double Partobi()
{
//      Samkutxedi კლასის წევრი შეიძლება მიმართავდეს Sibrtye კლასის წევრებს
return gverdi1 * gverdi2 / 2;
}
public double Perimetri()
{
return gverdi1 + gverdi2 + gverdi3;
}
public string TipisNaxva()
{
return "სამკუთხედის სახე - " + tipi.ToString();
}
}
class Otxkutxedi : Sibrtye          //      Otxkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
{
public double Partobi()
{
//      Otxkutxedi კლასის წევრი შეიძლება მიმართავდეს Sibrtye კლასის წევრებს
return gverdi1 * gverdi2;
}
public double Perimetri()
{
return ( gverdi1 + gverdi2 ) * 2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Samkutxedi samkutxedi = new Samkutxedi();
Otxkutxedi otxkutxedi = new Otxkutxedi();

samkutxedi.gverdi1 = Convert.ToDouble(textBox1.Text);
samkutxedi.gverdi2 = Convert.ToDouble(textBox2.Text);
samkutxedi.gverdi3 = Convert.ToDouble(textBox3.Text);
samkutxedi.tipi = "ტოლგვერდა";
//      ინფორმაციის გამოტანა samkutxedi ობიექტის შესახებ
label1.Text = samkutxedi.TipsisNaxva();
label2.Text = samkutxedi.Naxva();
}

```

```

label3.Text = samkutxedi.Partobi().ToString();
label4.Text = samkutxedi.Perimetri().ToString();
//
otxkutxedi.gverdi1 = Convert.ToDouble(textBox1.Text);
otxkutxedi.gverdi2 = Convert.ToDouble(textBox2.Text);
// ინფორმაციის გამოტანა otxkutxedi ობიექტის შესახებ
label5.Text = otxkutxedi.Naxva();
label6.Text = otxkutxedi.Partobi().ToString();
label7.Text = otxkutxedi.Perimetri().ToString();
}

```

Sibrtye კლასში განსაზღვრულია გეომეტრიული ფიგურის გვერდები. ფიგურა შეიძლება იყოს სამკუთხედი, კვადრეტი, მართკუთხედი და ა.შ. Samkutxedi კლასში, რომელიც არის Sibrtye კლასის მემკვიდრე, განისაზღვრება გეომეტრიული ფიგურის კონკრეტული ტიპი, ჩვენს შემთხვევაში - სამკუთხედი. Samkutxedi კლასში ჩართულია Sibrtye კლასის ყველა წევრი და დამატებულია tipi ცვლადი, აგრეთვე, Perimetri(), Partobi() და TipisNaxva() მეთოდები. სამკუთხედის ტიპი ინახება tipi ცვლადში, ხოლო TipisNaxva() მეთოდი კი აბრუნებს სამკუთხედის ტიპს.

რადგან Samkutxedi კლასში ჩართულია Sibrtye წინაპარი კლასის ყველა წევრი, ამიტომ მის Partobi() მეთოდს შეუძლია მიმართოს gverdi1 და gverdi2 ცვლადებს. გარდა ამისა, წერტილი (.) ოპერატორის გამოყენებით ვუთითებთ რა figura ობიექტისა და gverdi1 ან gverdi2 ცვლადის სახელს, შეიძლება უშუალოდ მეთოდის შიგნით მივმართოთ ამ ცვლადების ასლებს.

მიუხედავად იმისა, რომ Sibrtye კლასი არის Samkutxedi კლასის წინაპარი, ის არის მთლიანად ავტონომიური კლასი, რომელიც შეგვიძლია დამოუკიდებლად გამოვიყენოთ. მაგალითად, სწორია შემდეგი კოდი:

```

{
Sibrtye figura = new Sibrtye();
Figura.gverdi1 = 15;
Figura.gverdi2 = 25;
Figura.Naxva();
}

```

Sibrtye კლასის ობიექტს არ შეუძლია მიმართოს მემკვიდრე კლასის წევრებს.

protected მოდიფიკატორი

მემკვიდრეობითობის დროს ძალაშია კლასის დახურულ წევრებთან მიმართვისას არსებული შეზღუდვები. შედეგად, მემკვიდრე კლასის მეთოდს არ შეუძლია მიმართოს წინაპარი კლასის დახურულ წევრებს. ამ პრობლემის გადაწყვეტის ერთი გზაა protected მოდიფიკატორის გამოყენება. მეორე გზაა დახურულ მონაცემებთან მიმართვისათვის თვისებების გამოყენება.

C# ენაში არსებობს კლასის დაცული წევრების შექმნის შესაძლებლობა. ისინი გახსნილია მხოლოდ მემკვიდრე კლასებისათვის და დახურულია სხვა კლასებისათვის. კლასის დაცული წევრი იქმნება protected მოდიფიკატორის საშუალებით.

ქვემოთ მოყვანილია პროგრამა, რომელშიც გამოყენებულია protected მოდიფიკატორი.

```

// პროგრამა 7.2
// პროგრამაში ხდება protected მოდიფიკატორის გამოყენების დემონსტრირება
class Sibrtye
{

```

```

protected int gverdi1, gverdi2, gverdi3;    // ეს ცვლადები ღიაა Samkutxedi კლასის წევრებისთვის
public void Inicializacia(int a, int b, int c)
{
    gverdi1 = a;
    gverdi2 = b;
    gverdi3 = c;
}
public string Naxva()
{
return gverdi1.ToString() + " " + gverdi2.ToString() + " " + gverdi3.ToString();
}
}
class Samkutxedi : Sibrtye
{
int shedegi;
//     Partobi() მეთოდს შეუძლია მიმართოს Sibrtye კლასში
//     განსაზღვრულ gverdi1, gverdi2 და gverdi3 ცვლადებს
public void Partobi()
{
shedegi = gverdi1 * gverdi2 / 2;
}
public string PartobisNaxva()
{
return shedegi.ToString();
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//     პროგრამაში ხდება protected მოდიფიკატორის გამოყენება
Samkutxedi obieqti = new Samkutxedi();
int ricxvi1 = int.Parse(textBox1.Text);
int ricxvi2 = int.Parse(textBox2.Text);
int ricxvi3 = int.Parse(textBox3.Text);

//     Inicializacia() და Naxva() მეთოდები მისაწვდომია Samkutxedi კლასის ობიექტიდან
obieqti.Inicializacia(ricxvi1, ricxvi2, ricxvi3);
label1.Text = obieqti.Naxva();
obieqti.Partobi();
label2.Text = obieqti.PartobisNaxva();
}
}

```

კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა

წინაპარ და მემკვიდრე კლასებს შეიძლება ჰქონდეთ საკუთარი კონსტრუქტორები. ამ დროს წამოიჭრება კითხვა, თუ რომელი კონსტრუქტორი ქმნის მემკვიდრე კლასის ობიექტს. თითოეული კონსტრუქტორი ქმნის (ინიციალიზებას უკეთებს) ობიექტის თავის ნაწილს, ე.ი.

ობიექტი ნაწილ-ნაწილ იქმნება. ეს იმიტომ ხდება, რომ წინაპარი კლასის კონსტრუქტორს არ შეუძლია მიმართოს მემკვიდრე კლასის წევრებს და მოახდინოს მათი ინიციალიზება. თუ კონსტრუქტორი არ არის განსაზღვრული წინაპარ კლასში და განსაზღვრულია მემკვიდრე კლასში, მაშინ მემკვიდრე კლასის ობიექტის კონსტრუქცია სრულდება მისი კონსტრუქტორის მიერ, ხოლო წინაპარ კლასში განსაზღვრული ობიექტის ნაწილი კი იქმნება მისი ავტომატური კონსტრუქტორის მიერ.

თუ კონსტრუქტორები განსაზღვრულია როგორც წინაპარ, ისე მემკვიდრე კლასებში, მაშინ ობიექტის შექმნის პროცესი რამდენადმე რთულდება, რადგან ამ დროს გამოიძახება ორივე კონსტრუქტორი. ასეთ სიტუაციაში უნდა გამოვიყენოთ base საკვანძო სიტყვა, რომელიც გამოიყენება წინაპარი კლასის კონსტრუქტორის გამოსაძახებლად. ის, აგრეთვე, გამოიყენება წინაპარი კლასის დამალულ წევრებთან მიმართვისათვის.

base საკვანძო სიტყვა ეთითება მემკვიდრე კლასის კონსტრუქტორის გამოცხადების გაფართოებულ ფორმაში. ამ ფორმის სინტაქსია:

```
მემკვიდრე_კლასის_კონსტრუქტორის_სახელი (პარამეტრების_სია) : base (არგუმენტების_სია)
{
კონსტრუქტორის კოდი
}
```

არგუმენტების_სია არის წინაპარი კლასის კონსტრუქტორისთვის გადასაცემი მნიშვნელობების სია.

ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია base საკვანძო სიტყვის გამოყენება.

// პროგრამა 7.3

// პროგრამაში ხდება კონსტრუქტორების მემკვიდრეობითობის დემონსტრირება

```
class Kvadrati
{
    protected int gverdi1;
    public Kvadrati(int par1)
    {
        gverdi1 = par1;
    }
}
class Martkutxedi : Kvadrati
{
    public int gverdi2;
    public Martkutxedi(int par1, int par2) : base(par1)
    {
        gverdi2 = par2;
    }
}
class Samkutxedi1 : Martkutxedi
{
    int gverdi3;
    public Samkutxedi1(int par1, int par2, int par3) : base(par1, par2)
    {
        gverdi3 = par3;
    }
    public void gamotana(Label label)
    {
        label.Text = gverdi1.ToString() + " " + gverdi2.ToString() + " " + gverdi3.ToString();
    }
}
```

```

}
}
private void button2_Click(object sender, EventArgs e)
{
    int gverdi1 = int.Parse(textBox1.Text);
    int gverdi2 = int.Parse(textBox2.Text);
    int gverdi3 = int.Parse(textBox3.Text);
    Samkutxedi1 obj1 = new Samkutxedi1(gverdi1, gverdi2, gverdi3);
    obj1.gamotana(label1);
}

```

ამ პროგრამაში Samkutxedi კლასის კონსტრუქტორი base სიტყვის გამოყენებით იძახებს Martkutxedi კლასის კონსტრუქტორს და გადასცემს მას par1 და par2 პარამეტრებს. თავის მხრივ, Martkutxedi კლასის კონსტრუქტორი base სიტყვის გამოყენებით იძახებს Kvadrati კლასის კონსტრუქტორს და გადასცემს მას par1 პარამეტრს. შედეგად, gverdi1 ცვლადს ენიჭება par1 პარამეტრი. შემდეგ, gverdi2 ცვლადს ენიჭება par2 პარამეტრი. ბოლოს კი - gverdi3 ცვლადს ენიჭება par3 პარამეტრი.

წინაპარ კლასს შეიძლება ჰქონდეს რამდენიმე კონსტრუქტორი, რომელთაგან თითოეული შეგვიძლია გამოვიძახოთ base საკვანძო სიტყვის გამოყენებით. გამოიძახება ის კონსტრუქტორი, რომლის პარამეტრების სიაც შეესაბამება არგუმენტების სიას.

მემკვიდრე კლასის კონსტრუქტორში base საკვანძო სიტყვის მითითებისას გამოიძახება უახლოესი წინაპარი კლასის კონსტრუქტორი. ე.ი. base საკვანძო სიტყვა ყოველთვის მიმართავს უშუალო წინაპარ კლასს. ეს ძალაშია კლასების მრავალდონიანი იერარქიისთვისაც. თუ მემკვიდრე კლასის კონსტრუქტორში არ არის მითითებული base საკვანძო სიტყვა, მაშინ გამოიძახება წინაპარი კლასის ავტომატური კონსტრუქტორი.

წევრების დამალვა მემკვიდრეობითობის დროს

წინაპარი და მემკვიდრე კლასების წევრებს შეიძლება ერთი და იგივე სახელი ჰქონდეთ. ამ შემთხვევაში წინაპარი კლასის წევრი დამალულია მემკვიდრე კლასის სხვა წევრებისათვის და კომპილატორს ეკრანზე გამოაქვს გამაფრთხილებელი შეტყობინება ამის შესახებ. თუ ჩვენ განზრახ ვმალავთ წინაპარი კლასის წევრს და არ გვინდა, რომ კომპილატორმა გამოიტანოს გამაფრთხილებელი შეტყობინება, მაშინ მემკვიდრე კლასის შესაბამისი წევრის წინ უნდა მივუთითოთ new საკვანძო სიტყვა. ამ შემთხვევაში new ოპერატორი არ ახდენს ობიექტის შექმნას.

იმისათვის, რომ შევძლოთ წინაპარი კლასის დამალულ წევრებთან მიმართვა უნდა გამოვიყენოთ base საკვანძო სიტყვა. მისი სინტაქსია:

base.წევრის_სახელი

აქ წევრის_სახელი არის ობიექტის მეთოდის ან ცვლადის სახელი. ასეთი გამოყენების დროს base საკვანძო სიტყვა ისეთივე ფუნქციებს ასრულებს, როგორსაც this სიტყვა, იმ განსხვავებით, რომ base სიტყვა ყოველთვის მიმართავს წინაპარ კლასს.

მოყვანილ პროგრამაში ხდება base საკვანძო სიტყვის გამოყენება წინაპარი კლასის დამალულ ცვლადთან მიმართვისათვის.

// პროგრამა 7.4

// პროგრამაში ხდება base სიტყვის გამოყენება წინაპარი კლასის

// დამალულ ცვლადთან მიმართვისათვის

```
class SabazoKlasi
```

```
{
```

```

public int cvladi = 0;
}
// მემკვიდრე კლასის შექმნა
class MemkvidreKlasi : SabazoKlasi
{
new int cvladi; // cvladi ცვლადი მალავს SabazoKlasi კლასში გამოცხადებულ cvladi ცვლადს
public MemkvidreKlasi(int par1, int par2)
{
base.cvladi = par1; // მიმართვა SabazoKlasi კლასში გამოცხადებულ cvladi ცვლადთან
// base საკვანძო სიტყვის საშუალებით
// MemkvidreKlasi კლასში გამოცხადებული cvladi ცვლადის ინიციალიზება
cvladi = par2;
}
public void Naxva(Label lab1)
{
// SabazoKlasi კლასში განსაზღვრული cvladi ცვლადის ეკრანზე გამოტანა
lab1.Text = "cvladi ცვლადის მნიშვნელობა წინაპარ კლასში - " + base.cvladi.ToString();
// MemkvidreKlasi კლასში განსაზღვრული cvladi ცვლადის ეკრანზე გამოტანა
lab1.Text += "\ncvladi ცვლადის მნიშვნელობა მემკვიდრე კლასში - " + cvladi.ToString();
}
}
private void button1_Click(object sender, EventArgs e)
{
int ricxvi1 = int.Parse(textBox1.Text);
int ricxvi2 = int.Parse(textBox2.Text);
MemkvidreKlasi obieqti = new MemkvidreKlasi(ricxvi1, ricxvi2);

obieqti.Naxva(label1);
}

```

obieqti ობიექტის cvladi ცვლადი მალავს SabazoKlasi კლასში განსაზღვრულ cvladi ცვლადს. base საკვანძო სიტყვის გამოყენება უზრუნველყოფს მიმართვას წინაპარ კლასში განსაზღვრულ cvladi ცვლადთან. base სიტყვის გამოყენებით შეგვიძლია, აგრეთვე, გამოვიძახოთ დამალული მეთოდებიც. მოყვანილ პროგრამაში ხდება base სიტყვის გამოყენების დემონსტრირება დამალულ მეთოდებთან მიმართვისათვის.

// პროგრამა 7.5

// პროგრამაში ხდება base სიტყვის გამოყენება წინაპარი კლასის

// მეთოდთან მიმართვისათვის

```

class SabazoKlasi
{
public int cvladi = 0;
public void Naxva(Label lab1)
{
lab1.Text = " cvladi ცვლადის მნიშვნელობა წინაპარ კლასში - " + cvladi.ToString();
}
}
// მემკვიდრე კლასის შექმნა
class MemkvidreKlasi : SabazoKlasi

```

```

{
// ეს cvladi ცვლადი მალავს SabazoKlasi კლასში გამოცხადებულ cvladi ცვლადს
new int cvladi;
public MemkvidreKlasi(int par1, int par2)
{
base. cvladi = par1; // მიმართვა SabazoKlasi კლასში განსაზღვრულ cvladi ცვლადთან base
// საკვანძო სიტყვის გამოყენებით
cvladi = par2; // cvladi ცვლადის ინიციალიზება MemkvidreKlasi კლასში
}
// ეს Naxva() მეთოდი მალავს SabazoKlasi კლასში გამოცხადებულ Naxva() მეთოდს
new public void Naxva(Label lab1)
{
base.Naxva(lab1); // SabazoKlasi კლასში განსაზღვრული Naxva() მეთოდის გამოძახება
lab1.Text += "\ncvladi ცვლადის მნიშვნელობა მემკვიდრე კლასში - " + cvladi.ToString();
}
}
private void button1_Click(object sender, System.EventArgs e)
{
MemkvidreKlasi obieqti = new MemkvidreKlasi(5, 10);

obieqti.Naxva(label1);
}

```

კლასების მრავალდონიანი იერარქიის შექმნა

აქამდე ჩვენ ვიყენებდით კლასების ორდონიან იერარქიას, რომელიც შედგებოდა წინაპარი და მემკვიდრე კლასებისაგან. C# ენაში შეგვიძლია, შევქმნათ კლასების მრავალდონიანი იერარქია. ამ შემთხვევაში მემკვიდრე კლასი შეგვიძლია გამოვიყენოთ როგორც საბაზო შემდგომი მემკვიდრეობითობისათვის. მაგალითად, Klasi3 კლასი შეიძლება იყოს Klasi2 კლასის მემკვიდრე, რომელიც თავის მხრივ შეიძლება იყოს Klasi1 კლასის მემკვიდრე. ამ შემთხვევაში Klasi3 კლასი მემკვიდრეობით იღებს Klasi1 და Klasi2 კლასების ყველა წევრს.

ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია კლასების მრავალდონიანი იერარქიის შექმნა. აქ Sibrtye არის საბაზო კლასი. მისი მემკვიდრეა Samkutxedi კლასი, რომელიც არის საბაზო FeradiSamkutxedi კლასისათვის. FeradiSamkutxedi კლასი მემკვიდრეობით იღებს Sibrtye და Samkutxedi კლასების ყველა წევრს. FeradiSamkutxedi კლასში გამოცხადებულია ახალი peri ცვლადი, რომელიც შეიცავს ინფორმაციას სამკუთხედის ფერის შესახებ. რადგან FeradiSamkutxedi კლასის შექმნისას გამოიყენება მრავალდონიანი მემკვიდრეობითობა, ამ კლასის ობიექტებს შეუძლიათ მიმართონ Sibrtye და Samkutxedi კლასების წევრებს.

// პროგრამა 7.6

// პროგრამაში ხდება კლასების მრავალდონიანი იერარქიის დემონსტრირება

```

class Sibrtye
{
protected double sigrdze; // დახურული ცვლადების გამოცხადება
protected double sigane;
// ავტომატური კონსტრუქტორი
public Sibrtye()

```



```

    {
        sigrdze = sigane = 0.0;
    }
// კონსტრუქტორი ორი პარამეტრით
public Sibrtye(double par1, double par2)
{
    sigrdze = par1;
    sigane = par2;
}
// კონსტრუქტორი, განკუთვნილი ობიექტების შესაქმნელად, რომელთა sigane და simagle
// თვისებებს ენიჭებათ ერთი და იგივე მნიშვნელობები
public Sibrtye(double par3)
{
    sigrdze = sigane = par3;
}
public string Naxva()
{
    return "გეომეტრიული ფიგურის სიგრძე და სიგანე \n" + sigrdze.ToString() +
        " " + sigane.ToString();
}
}
// Samkutxedi არის Sibrtye კლასის მემკვიდრე
class Samkutxedi : Sibrtye
{
    string tipi;
// ავტომატური კონსტრუქტორი. ის იძახებს Sibrtye კლასის ავტომატურ კონსტრუქტორს
public Samkutxedi()
{
    tipi = "უტიპო";
}
// კონსტრუქტორი სამი პარამეტრით
public Samkutxedi(string par1, double par2, double par3) : base(par2, par3)
{
    tipi = par1;
}
// კონსტრუქტორი ერთი პარამეტრით
public Samkutxedi(double par4) : base(par4)
{
    tipi = "ტოლფერდა";
}
public double Partobi()
{
    return sigrdze * sigane / 2;
}
public string TipisNaxva()
{
    return "სამკუთხედის სახე - " + tipi;
}
}

```

```

    }
}
// FeradiSamkutxedi კლასი არის Samkutxedi კლასის მემკვიდრე
class FeradiSamkutxedi : Samkutxedi
{
    string peri;
    public FeradiSamkutxedi(string par1, string par2, double par3, double par4)
: base(par2, par3, par4)
    {
        peri = par1;
    }
    // ინფორმაციის გამოტანა სამკუთხედის ფერის შესახებ
    public string PerisNaxva()
    {
        return "სამკუთხედის ფერი - " + peri;
    }
}
private void button1_Click(object sender, System.EventArgs e)
{
    FeradiSamkutxedi sam1 = new FeradiSamkutxedi("მწვანე, ", "სწორკუთხა ", 5.0, 15.0);
    label1.Text = sam1.TipisNaxva();
    label2.Text = sam1.Naxva();
    label3.Text = sam1.PerisNaxva();
    label4.Text = sam1.Partobi().ToString();
}

```

მოცემული პროგრამიდან ჩანს, რომ base საკვანძო სიტყვა ყოველთვის მიმართავს უახლოესი წინაპარი კლასის კონსტრუქტორს. თუ base საკვანძო სიტყვა ეთითება Samkutxedi კლასის კონსტრუქტორთან, მაშინ გამოიძახება Sibirtye კლასის კონსტრუქტორი. თუ base სიტყვა ეთითება FeradiSamkutxedi კლასის კონსტრუქტორთან, მაშინ გამოიძახება Samkutxedi კლასის კონსტრუქტორი. თუ წინაპარი კლასის კონსტრუქტორს სჭირდება რაიმე პარამეტრები, მაშინ კლასების მრავალდონიანი იერარქიის შექმნის წესების შესაბამისად ყველა მემკვიდრე კლასი ამ პარამეტრებს უნდა გადასცემდეს "ზევით" შესაბამის დონეზე, მიუხედავად იმისა ეს პარამეტრები სჭირდება თუ არა წინაპარ კლასს.

კონსტრუქტორების გამოძახების რიგითობა

კლასების იერარქიაში კონსტრუქტორები გამოიძახება იმ რიგითობით, რა რიგითობითაც სრულდება მემკვიდრეობითობა - წინაპარი კლასიდან მემკვიდრე კლასისკენ. უფრო მეტიც, ეს რიგითობა რჩება უცვლელი იმისგან დამოუკიდებლად, გამოიყენება თუ არა base საკვანძო სიტყვა. თუ ეს საკვანძო სიტყვა არ არის მითითებული, მაშინ გამოიძახება თითოეული წინაპარი კლასის ავტომატური კონსტრუქტორი. კონსტრუქტორის გამოძახების რიგითობა ნაჩვენებია მოყვანილ პროგრამაში.

```
// პროგრამა 7.7
```

```
// პროგრამაში ხდება კონსტრუქტორების გამოძახების რიგითობის დემონსტრირება
```

```
class Klasi1
{
public Klasi1()
{
```

```

MessageBox.Show("Klasi1 კლასში განსაზღვრული მეთოდი");
}
}
// Klasi1 კლასის მემკვიდრე კლასის შექმნა
class Klasi2 : Klasi1
{
public Klasi2()
{
MessageBox.Show("Klasi2 კლასში განსაზღვრული მეთოდი");
}
}
//
class Klasi3 : Klasi2
{
public Klasi3()
{
MessageBox.Show("Klasi3 კლასში განსაზღვრული მეთოდი");
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Klasi3 obieqti = new Klasi3();
}

```

მიმართვები წინაპარი და მემკვიდრე კლასების ობიექტებზე

ერთი კლასის მიმართვით ცვლადს არ შეიძლება მიენიჭოს სხვა კლასის ობიექტზე მიმართვა. მაგრამ წინაპარი კლასის მიმართვით ცვლადს შეიძლება მიენიჭოს ნებისმიერი მემკვიდრე კლასის ობიექტზე მიმართვა. ქვემოთ მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

```

// პროგრამა 7.8
// პროგრამაში ხდება წინაპარი და მემკვიდრე კლასების ობიექტებზე
// მიმართვების დემონსტრირება
class SabazoKlasi
{
public int ricxvi1;
public SabazoKlasi(int par1)
{
ricxvi1 = par1;
}
}
class MemkvidreKlasi : SabazoKlasi
{
public int ricxvi2;
public MemkvidreKlasi(int par2, int par3) : base(par3)
{

```

```

ricxvi2 = par2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
    int ricxvi = Convert.ToInt32(textBox1.Text);
    SabazoKlasi Sabazo_Obieqti1 = new SabazoKlasi(ricxvi);
    SabazoKlasi Mimartviti_Cvladi;
    MemkvidreKlasi Memkvidre_Obieqti = new MemkvidreKlasi(2, 5);
    // მინიჭების ეს ოპერატორი ნამდვილია, რადგან მასში მონაწილეობს
    // ერთი ტიპის მიმართვითი ცვლადები
    Mimartviti_Cvladi = Sabazo_Obieqti1;
    label1.Text = Mimartviti_Cvladi.ricxvi1.ToString();
    // ასევე ნამდვილია მინიჭების ეს ოპერატორი, რადგან MemkvidreKlasi არის
    // SabazoKlasi-ის მემკვიდრე
    Mimartviti_Cvladi = Memkvidre_Obieqti;
    Mimartviti_Cvladi.ricxvi1 = 35; // ეს მიმართვა ნამდვილია
    label2.Text = Mimartviti_Cvladi.ricxvi1.ToString();
    // Mimartviti_Cvladi.ricxvi2 = 50; // შეცდომა! ricxvi2 ცვლადი არ არის
    // გამოცხადებული SabazoKlasi კლასში
}

```

ამ პროგრამაში MemkvidreKlasi კლასი არის SabazoKlasi კლასის მემკვიდრე. ამიტომ, Sabazo_Obieqti2 ცვლადს შეგვიძლია მივანიჭოთ Memkvidre_Obieqti ობიექტზე მიმართვა.

ყურადღება მივაქციოთ იმას, რომ კლასის წევრებთან მიმართვის შესაძლებლობა დამოკიდებულია მიმართვითი ცვლადის ტიპზე, და არა იმ ობიექტის ტიპზე, რომელსაც ის მიმართავს. თუ მიმართვით ცვლადს, რომელსაც აქვს წინაპარი კლასის ტიპი, ენიჭება მემკვიდრე კლასის ობიექტზე მიმართვა, მაშინ ამ ცვლადის საშუალებით შეგვეძლება მხოლოდ იმ წევრებთან მიმართვა, რომლებიც განსაზღვრული იყო წინაპარ კლასში. სწორედ ამიტომ მიმართვითი Sabazo_Obieqti2 ცვლადი ვერ მიმართავს ricxvi2 ცვლადს მას შემდეგაც კი, რაც Sabazo_Obieqti2 ცვლადს მიენიჭა MemkvidreKlasi კლასის ობიექტზე მიმართვა. ასეთ შეზღუდვას აზრი აქვს, რადგან წინაპარმა კლასმა არაფერი არ "იცის" კლასის იმ წევრების შესახებ, რომლებიც დაემატა მემკვიდრე კლასს.

ვირტუალური მეთოდები

ენაში ვირტუალური მეთოდები უზრუნველყოფენ პოლიმორფიზმს პროგრამის შესრულების დროს. პოლიმორფიზმი საშუალებას იძლევა წინაპარ კლასში განვსაზღვროთ მეთოდები, რომლებიც საერთო იქნება ყველა მემკვიდრე კლასისათვის. ამასთან, ამ მეთოდების კოდი მემკვიდრე კლასებში შეიძლება შეიცვალოს ან იგივე დარჩეს. უცვლელი რჩება მეთოდის სახელი და მიმართვითი ცვლადი. მეთოდების ხელახალი განსაზღვრა წარმოადგენს პოლიმორფიზმის "ერთი ინტერფეისი - რამდენიმე მეთოდი" პრინციპის რეალიზების ერთ-ერთ საშუალებას.

მეთოდს, რომლის განსაზღვრისას წინაპარ კლასში მითითებული იყო virtual საკვანძო სიტყვა, და რომელიც ხელახლა იყო განსაზღვრული ერთ ან მეტ მემკვიდრე კლასში, ვირტუალური მეთოდი ეწოდება. თითოეულ მემკვიდრე კლასს შეიძლება ჰქონდეს ვირტუალური მეთოდის საკუთარი ვერსია. ენაში ვირტუალური მეთოდის ვერსიის არჩევა

ხორციელდება იმ ობიექტის ტიპის შესაბამისად, რომელსაც მიმართვითი ცვლადი მიმართავს. ეს არჩევა ხორციელდება პროგრამის შესრულების დროს. მიმართვითი ცვლადი შეიძლება მიმართავდეს სხვადასხვა ტიპის ობიექტებს, ამიტომ, შეიძლება გამოძახებული იყოს ვირტუალური მეთოდების სხვადასხვა ვერსიები. სხვა სიტყვებით რომ ვთქვათ, სწორედ ობიექტის ტიპი, რომელზეც მიუთითებს მიმართვა (და არა მიმართვითი ცვლადის ტიპი), განსაზღვრავს გამოსაძახებელი ვირტუალური მეთოდის ვერსიას. ამრიგად, თუ კლასი შეიცავს ვირტუალურ მეთოდს და ამ კლასიდან მემკვიდრეობით მიღებული იყო სხვა კლასები, რომლებშიც განსაზღვრულია ვირტუალური მეთოდის საკუთარი ვერსიები, მაშინ სხვადასხვა ტიპის ობიექტებზე წინაპარი კლასის ტიპის მქონე ცვლადის მიმართვის დროს შესრულდება ვირტუალური მეთოდის შესაბამისი ვერსია.

წინაპარ კლასში ვირტუალური მეთოდის განსაზღვრისას დასაბრუნებელი მნიშვნელობის ტიპის წინ ეთითება virtual საკვანძო სიტყვა, ხოლო მემკვიდრე კლასში ვირტუალური მეთოდის ხელახალი განსაზღვრისას გამოიყენება override მოდიფიკატორი. მემკვიდრე კლასში ვირტუალური მეთოდის განსაზღვრის პროცესს, როცა ნაწილობრივ ან მთლიანად იცვლება მეთოდის ტანი, ხოლო მეთოდის სახელი, პარამეტრები და მათი ტიპები იგივე რჩება, მეთოდის ხელახალი განსაზღვრა ეწოდება. ვირტუალური მეთოდი არ შეიძლება განსაზღვრული იყოს static ან abstract მოდიფიკატორის გამოყენებით (ისინი მომდევნო თავში განიხილება).

მეთოდის ხელახალი განსაზღვრა საფუძვლად უდევს გამოსაძახებელი მეთოდის დინამიური არჩევის კონცეფციას. ეს არის მექანიზმი, რომლის საშუალებითაც გამოსაძახებელი ხელახლა განსაზღვრული მეთოდის არჩევა ხორციელდება პროგრამის შესრულების და არა პროგრამის კომპილირების დროს.

ქვემოთ მოყვანილია პროგრამა, რომელშიც ნაჩვენებია ვირტუალური მეთოდისა და მისი ხელახლა განსაზღვრული ვერსიების გამოყენება.

// **პროგრამა 7.9**

// **პროგრამაში ხდება ვირტუალურ მეთოდებთან მუშაობის დემონსტრირება**

```
class SabazoKlasi
{
// ვირტუალური მეთოდის შექმნა წინაპარ კლასში
public virtual string Naxva() // ვირტუალური მეთოდის განსაზღვრა
{
return "SabazoKlasi კლასში განსაზღვრული Naxva() მეთოდი";
}
}
class MemkvidreKlasi1 : SabazoKlasi
{
// ვირტუალური მეთოდის ხელახალი განსაზღვრა
public override string Naxva()
{
return "MemkvidreKlasi1 კლასში განსაზღვრული Naxva() მეთოდი";
}
}
class MemkvidreKlasi2 : SabazoKlasi
{
// ვირტუალური მეთოდის კიდევ ერთი ხელახალი განსაზღვრა
public override string Naxva()
{
return "MemkvidreKlasi2 კლასში განსაზღვრული Naxva() მეთოდი";
}
```

```

}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება ვირტუალური მეთოდის გამოყენების დემონსტრირება
SabazoKlasi Sabazo_Obieqti = new SabazoKlasi();
MemkvidreKlasi1 Memkvidre_Obieqti1 = new MemkvidreKlasi1();
MemkvidreKlasi2 Memkvidre_Obieqti2 = new MemkvidreKlasi2();
//      წინაპარი კლასის ტიპის მიმართებითი ცვლადის გამოცხადება
SabazoKlasi Mimartviti_Cvladi;

Mimartviti_Cvladi = Sabazo_Obieqti;
label1.Text = Mimartviti_Cvladi.Naxva();

Mimartviti_Cvladi = Memkvidre_Obieqti1;
label2.Text = Mimartviti_Cvladi.Naxva();

Mimartviti_Cvladi = Memkvidre_Obieqti2;
label3.Text = Mimartviti_Cvladi.Naxva();
}

```

ამ პროგრამაში იქმნება SabazoKlasi წინაპარი კლასი, აგრეთვე, ორი მემკვიდრე კლასი MemkvidreKlasi1 და MemkvidreKlasi2. SabazoKlasi კლასში გამოცხადებულია Naxva() მეთოდი, რომელიც შემდეგ ხელახლა განისაზღვრება მემკვიდრე კლასებში. პროგრამაში გამოცხადებულია SabazoKlasi, MemkvidreKlasi1 და MemkvidreKlasi2 ტიპის ობიექტები, აგრეთვე, გამოცხადებულია SabazoKlasi ტიპის Mimartviti_Cvladi მიმართებითი ცვლადი. ამ ცვლადს რიგრიგობით ენიჭება მიმართებები სამივე ტიპის ობიექტზე. ეს მიმართებები გამოიყენება Naxva() მეთოდის გამოძახებისათვის. როგორც პროგრამის შესრულების შედეგიდან დავინახავთ, გამოსაძახებელი Naxva() მეთოდის ვერსიის არჩევა დამოკიდებულია იმ ობიექტის ტიპზე, რომელსაც მიმართავს Mimartviti_Cvladi ცვლადი.

ვირტუალური მეთოდის გადატვირთვა არ არის აუცილებელი. თუ მემკვიდრე კლასს არ აქვს ვირტუალური მეთოდის საკუთარი ვერსია, მაშინ გამოიძახება წინაპარი კლასის მეთოდი. ქვემოთ მოყვანილ პროგრამაში ხდება ვირტუალურ მეთოდებთან მუშაობის დემონსტრირება.

// **პროგრამა 7.10**

// **პროგრამაში მემკვიდრე კლასს არ აქვს ვირტუალური მეთოდის საკუთარი ვერსია,**

// **ამიტომ გამოიძახება წინაპარი კლასის მეთოდი**

```

class SabazoKlasi
{
//      ვირტუალური მეთოდის შექმნა წინაპარ კლასში
public virtual void Naxva(Label lab1) //      ვირტუალური მეთოდის გამოცხადება
{
lab1.Text = "SabazoKlasi კლასში განსაზღვრული Naxva() მეთოდი";
}
}
class MemkvidreKlasi1 : SabazoKlasi
{
//
public override void Naxva(Label lab1) // ვირტუალური მეთოდის ხელახალი განსაზღვრა

```

```

    {
        lab1.Text = "MemkvidreKlasi1 კლასში განსაზღვრული Naxva() მეთოდი";
    }
}
class MemkvidreKlasi2 : SabazoKlasi
{
    // ამ კლასში არ ხდება ვირტუალური მეთოდის ხელახალი განსაზღვრა
}
private void button1_Click(object sender, System.EventArgs e)
{
    // პროგრამაში ხდება ვირტუალური მეთოდის გამოყენების დემონსტრირება
    SabazoKlasi Sabazo_Objeqti = new SabazoKlasi();
    MemkvidreKlasi1 Memkvidre_Objeqti1 = new MemkvidreKlasi1();
    MemkvidreKlasi2 Memkvidre_Objeqti2 = new MemkvidreKlasi2();
    // წინაპარი კლასის ტიპის მიმართებით ცვლადის გამოცხადება
    SabazoKlasi Mimartviti_Cvladi;

    Mimartviti_Cvladi = Sabazo_Objeqti;
    Mimartviti_Cvladi.Naxva(label1);

    Mimartviti_Cvladi = Memkvidre_Objeqti1;
    Mimartviti_Cvladi.Naxva(label2);

    // SabazoKlasi კლასში განსაზღვრული Naxva() მეთოდის
    // გამოძახება Mimartviti_Cvladi.Naxva();
    Mimartviti_Cvladi = Memkvidre_Objeqti2;
    Mimartviti_Cvladi.Naxva(label3);
}

```

აქ MemkvidreKlasi2 კლასში არ ხდება Naxva() მეთოდის ხელახალი განსაზღვრა, ამიტომ, MemkvidreKlasi2 ტიპის ობიექტის Naxva() მეთოდის გამოძახებისას გამოიძახება SabazoKlasi კლასში განსაზღვრული Naxva() მეთოდი.

განვიხილოთ კიდევ ერთი მაგალითი. Sibrtye კლასში განვსაზღვროთ ვირტუალური Partobi() მეთოდი. ის შეგვიძლია ხელახლა განვსაზღვროთ თითოეულ მემკვიდრე კლასში. ეს შესაძლებლობას მოგვცემს გამოვთვალოთ ფართობი გეომეტრიული ფიგურის ტიპზე დამოკიდებულებით (სამკუთხედი, კვადრატი და ა.შ.). ქვემოთ მოყვანილია ამ პროგრამის კოდი.

```

// პროგრამა 7.11
// პროგრამაში ხდება გეომეტრიული ფიგურის ფართობის გამოთვლა
// ფიგურის ტიპზე დამოკიდებულებით
class Sibrtye
{
    public double sigrdze; // ცვლადების გამოცხადება
    public double sigane;
    public string saxeli;
    // ავტომატური კონსტრუქტორის განსაზღვრა
    public Sibrtye()
    {
        sigrdze = sigane = 0.0;
    }
}

```

```

    saxeli = "";
}
// კონსტრუქტორი სამი პარამეტრით
public Sibrtye(double sigrdze, double sigane, string saxeli)
{
    this.sigrdze = sigrdze;
    this.sigane = sigane;
    this.saxeli = saxeli;
}
// კონსტრუქტორი ორი პარამეტრით
public Sibrtye(double sigrdze_sigane, string saxeli)
{
    sigrdze = sigane = sigrdze_sigane;
    this.saxeli = saxeli;
}
// ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
public Sibrtye(Sibrtye obieqti)
{
    sigrdze = obieqti.sigrdze;
    sigane = obieqti.sigane;
    saxeli = obieqti.saxeli;
}
public void Naxva()
{
    MessageBox.Show("გეომეტრიული ფიგურის სიგრძე და სიგანე" +
        sigrdze.ToString() + " " + sigane.ToString());
}
public virtual double Partobi()
{
    MessageBox.Show("ეს მეთოდი ხელახლაა განსაზღვრული მემკვიდრე კლასში");
    return 0.0;
}
}
// Samkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
class Samkutxedi : Sibrtye
{
    string tipi;
    // ავტომატური კონსტრუქტორის განსაზღვრა
    public Samkutxedi()
    {
        tipi = "";
    }
    // კონსტრუქტორი ორი პარამეტრით, რომელიც
    // იძახებს წინაპარი კლასის კონსტრუქტორს
    public Samkutxedi(string tipi, double sigrdze, double sigane)
        : base(sigrdze, sigane, "სამკუთხედი")
    {

```



```

    this.tipi = tipi;
}
// კონსტრუქტორი ერთი პარამეტრით, რომელიც იძახებს
// წინაპარი კლასის კონსტრუქტორს
public Samkutxedi(double sigrdze_sigane)
    : base(sigrdze_sigane, " სამკუთხედი ")
{
    tipi = "ტოლფერდა";
}
// ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
public Samkutxedi(Samkutxedi obieqti) : base(obieqti)
{
    tipi = obieqti.tipi;
}
// Partobi() მეთოდის ხელახალი განსაზღვრა Samkutxedi კლასში
public override double Partobi()
{
    return sigrdze * sigane / 2;
}
public void TipisNaxva()
{
    MessageBox.Show("სამკუთხედის სახე - " + tipi);
}
}
// Otxkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
class Otxkutxedi : Sibrtye
{
    // კონსტრუქტორი ორი პარამეტრით, რომელიც
    // იძახებს წინაპარი კლასის კონსტრუქტორს
    // მახასიათებლები, დამახასიათებელი ოთხკუთხედისთვის
    public Otxkutxedi(double sigrdze, double sigane) : base(sigrdze, sigane, " ოთხკუთხედი ")
    {
        //
    }
    // კონსტრუქტორი ერთი პარამეტრით, რომელიც
    // იძახებს წინაპარი კლასის კონსტრუქტორს
    public Otxkutxedi(double sigrdze) : base(sigrdze, "კვადრატი") { }
    // ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
    public Otxkutxedi(Otxkutxedi obieqti)
        : base(obieqti)
    {
        //
    }
    public bool ArisKvadrati()
    {
        if (sigrdze == sigane ) return true;
        return false;
    }
}

```

```

    }
    //      Partobi() მეთოდის ხელახალი განსაზღვრა Otxkuxedi კლასში
    public override double Partobi()
    {
        return sigrdze * sigane;
    }
}

private void button1_Click(object sender, System.EventArgs e)
{
    label1.Text = "";
    //      Sibrtye ტიპის მქონე figurebi ობიექტების მასივის გამოცხადება
    Sibrtye[] figurebis_masivi = new Sibrtye[5];
    figurebis_masivi[0] = new Samkuxedi("სწორკუთხა", 5.0, 15.0);
    figurebis_masivi[1] = new Otxkuxedi(15);
    figurebis_masivi[2] = new Otxkuxedi(15, 5);
    figurebis_masivi[3] = new Samkuxedi(5.5);
    figurebis_masivi[4] = new Sibrtye(15, 25, "საბაზო");

    for ( int indexi = 0; indexi < figurebis_masivi.Length; indexi++ )
    {
        label1.Text += figurebis_masivi[indexi].saxeli + " - ფართობი = " +
            figurebis_masivi[indexi].Partobi().ToString() + '\n';
    }
}

```

ამ პროგრამის Sibrtye კლასში Partobi() მეთოდი გამოცხადებულია როგორც ვირტუალური. ის ხელახლა განისაზღვრება Samkuxedi და Otxkuxedi კლასებში. Sibrtye კლასის Partobi() მეთოდი არ ახდენს გეომეტრიული ფიგურის ფართობის გამოთვლას. მისი შესრულების შედეგად უბრალოდ გაიცემა ინფორმაცია იმის შესახებ, რომ ეს მეთოდი ხელახლა უნდა იყოს განსაზღვრული მემკვიდრე კლასებში. ამიტომ, Partobi() მეთოდის ყოველი ხელახალი განსაზღვრისას, მასში სრულდება შესაბამისი გეომეტრიული ფიგურის ფართობის გამოთვლა.

მთავარ პროგრამაში გამოცხადებულია figurebi მასივი, რომელიც შეიცავს Sibrtye კლასის ობიექტებს. ამ მასივის ელემენტებს ენიჭებათ Samkuxedi, Otxkuxedi და Sibrtye ტიპის ობიექტებზე მიმართვები. ასეთი მინიჭებები სწორია, რადგან წინაპარი კლასის ტიპის მიმართვითი ცვლადები შეიძლება მიმართავდნენ მემკვიდრე კლასის ობიექტებს.

აბსტრაქტული კლასები

ზოგჯერ საჭიროა ისეთი წინაპარი კლასის შექმნა, რომელშიც განსაზღვრულია მეთოდების მხოლოდ ზოგიერთი მახასიათებლები, როგორცაა დასაბრუნებელი მნიშვნელობის ტიპი, მეთოდის სახელი და პარამეტრების სია. ამ მახასიათებლების ერთობლიობას მეთოდის ხელმოწერა ეწოდება. ასეთი მეთოდების ტანი ცარიელია, რადგან ძნელია წინასწარ განვსაზღვროთ, თუ რა მოქმედებები უნდა შეასრულოს მეთოდმა მემკვიდრე კლასის ობიექტებში. ამიტომ, ამ მეთოდების რეალიზება (მეთოდის კოდის განსაზღვრა) მემკვიდრე კლასებში უნდა შესრულდეს.

ასეთ შემთხვევებში საბაზო კლასში შეგვიძლია განვსაზღვროთ აბსტრაქტული მეთოდები, რომლებსაც ცარიელი ტანი აქვთ. აბსტრაქტული მეთოდის გამოცხადებისათვის გამოიყენება

abstract მოდიფიკატორი. თუ კლასის წევრების სიაში გვხვდება მეთოდი ამ მოდიფიკატორით, მაშინ მისი რეალიზება (განსაზღვრა) უნდა მოვახდინოთ მემკვიდრე კლასში. აბსტრაქტული მეთოდი ავტომატურად ხდება ვირტუალური ისე, რომ virtual სიტყვის გამოყენება ასეთი მეთოდის გამოცხადებისას არაა საჭირო. უფრო მეტიც, virtual და abstract მოდიფიკატორების ერთობლივი გამოყენება დაუშვებელია.

აბსტრაქტული მეთოდის გამოცხადების სინტაქსია:

abstract ტიპი მეთოდის_სახელი (პარამეტრების_სია);

ამ გამოცხადებაში არ არის მეთოდის კოდი. abstract მოდიფიკატორი შეგვიძლია გამოვიყენოთ მხოლოდ ჩვეულებრივი მეთოდების მიმართ და არ შეიძლება გამოვიყენოთ სტატიკური მეთოდების მიმართ.

კლასი, რომელიც ერთ ან მეტ აბსტრაქტულ მეთოდს შეიცავს, გამოცხადებული უნდა იყოს როგორც აბსტრაქტული, ე.ი. მისი გამოცხადებისას უნდა მივუთითოთ abstract მოდიფიკატორი. რადგან, აბსტრაქტული კლასი არ არის განსაზღვრული ბოლომდე, ამიტომ შეუძლებელია ამ კლასის ობიექტის შექმნა, და აბსტრაქტული კლასის შექმნის მცდელობა new საკვანძო სიტყვის გამოყენებით გამოიწვევს კომპილირების შეცდომას.

როცა კლასი არის აბსტრაქტული კლასის მემკვიდრე, მან უნდა მოახდინოს წინაპარი კლასის ყველა აბსტრაქტული მეთოდის რეალიზება. წინააღმდეგ შემთხვევაში, ასეთი მემკვიდრე კლასი გამოცხადებული უნდა იყოს abstract მოდიფიკატორით. ამრიგად, abstract ატრიბუტი მემკვიდრეობით გადაიცემა მანამ, სანამ მეთოდები და ე.ი. თვით კლასი, არ იქნება მთლიანად რეალიზებული.

ქვემოთ მოყვანილ პროგრამაში Sibrtye კლასი გამოცხადებულია როგორც აბსტრაქტული. მასში გამოცხადებულია აბსტრაქტული Partobi() მეთოდი, რომელიც განკუთვნილია ორგანზომილებიანი გეომეტრიული ფიგურის ფართობის გამოსათვლელად. ამიტომ, Sibrtye კლასის მემკვიდრე კლასების შექმნისას მათში ხელახლა უნდა იყოს განსაზღვრული Partobi() მეთოდი.

// პროგრამა 7.12

// პროგრამაში ხდება აბსტრაქტულ კლასებთან მუშაობის დემონსტრირება

```
abstract class Sibrtye // აბსტრაქტული კლასის გამოცხადება
{
    public double sigrdze; // დახურული ცვლადების გამოცხადება
    public double sigane;
    public string saxeli;
    // ავტომატური კონსტრუქტორის განსაზღვრა
    public Sibrtye()
    {
        sigrdze = sigane = 0.0;
        saxeli = "";
    }
    // კონსტრუქტორი პარამეტრებით
    public Sibrtye(double sigrdze, double sigane, string saxeli)
    {
        this.sigrdze = sigrdze;
        this.sigane = sigane;
        this.saxeli = saxeli;
    }
    // კონსტრუქტორი, განკუთვნილი ობიექტის შესაქმნელად, რომლის sigane და
    // simagle ცვლადებს ერთნაირი მნიშვნელობები ენიჭებათ
    public Sibrtye(double sigrdze_sigane, string saxeli)
```

```

{
    sigrdze = sigane = sigrdze_sigane;
    this.saxeli = saxeli;
}
//      ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
public Sibrtye(Sibrtye obieqti)
{
    sigrdze = obieqti.sigrdze;
    sigane = obieqti.sigane;
    saxeli = obieqti.saxeli;
}
public void Naxva()
{
    MessageBox.Show("გეომეტრიული ფიგურის სიგრძე და სიგანე " +
        sigrdze.ToString() + " " + sigane.ToString());
}
//      აბსტრაქტული Partobi() მეთოდის გამოცხადება
public abstract double Partobi();
}
// მემკვიდრე კლასის გამოცხადება
class Samkutxedi : Sibrtye
{
    string tipi;
    //      ავტომატური კონსტრუქტორის განსაზღვრა
    public Samkutxedi()
    {
        tipi = "";
    }
    //      Samkutxedi კლასის კონსტრუქტორი
    public Samkutxedi(string tipi, double sigrdze, double sigane)
        : base(sigrdze, sigane, "სამკუთხედი")
    {
        this.tipi = tipi;
    }
    //      კონსტრუქტორი, რომელიც იძახებს წინაპარი კლასის კონსტრუქტორს
    public Samkutxedi(double sigane)
        : base(sigane, "სამკუთხედი")
    {
        tipi = "ტოლფერდა";
    }
    //      ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
    public Samkutxedi(Samkutxedi obieqti)
        : base(obieqti)
    {
        tipi = obieqti.tipi;
    }
    //      Partobi() მეთოდის ხელახალი განსაზღვრა Samkutxedi კლასში
    public override double Partobi()
    {

```

```

        return sigrdze * sigane / 2;
    }
    public void TipisNaxva()
    {
        MessageBox.Show("სამკუთხედის სახე - " + tipi);
    }
}
// Otxkutxedi კლასი არის Sibrtye კლასის მემკვიდრე
class Otxkutxedi : Sibrtye
{
    // კონსტრუქტორი პარამეტრებით, რომელიც იძახებს
    // წინაპარი კლასის კონსტრუქტორს
    public Otxkutxedi(double sigrdze, double sigane)
        : base(sigrdze, sigane, "სწორკუთხედი")
    {
        // ამ მეთოდის კოდი არ არის განსაზღვრული
    }
    // კონსტრუქტორი, განსაზღვრული ობიექტის შესაქმნელად, რომელშიც
    // განსაზღვრულია მახასიათებლები, დამახასიათებელი კვადრატისათვის
    public Otxkutxedi(double x)
        : base(x, "კვადრატი")
    {}
    // ობიექტის კონსტრუირება სხვა ობიექტის საფუძველზე
    public Otxkutxedi(Otxkutxedi obieqti)
        : base(obieqti)
    {
        //
    }
    public bool ArisKvadrati()
    {
        if ( sigrdze == sigane ) return true;
        return false;
    }
    // Partobi() მეთოდის ხელახალი განსაზღვრა Otxkutxedi კლასში
    public override double Partobi()
    {
        return sigrdze * sigane;
    }
}
private void button1_Click(object sender, System.EventArgs e)
{
    label2.Text = "";
    Sibrtye[] figurebis_masivi = new Sibrtye[4];
    figurebis_masivi[0] = new Samkutxedi("სწორკუთხა", 5.0, 15.0);
    figurebis_masivi[1] = new Otxkutxedi(15);
    figurebis_masivi[2] = new Otxkutxedi(15, 5);
    figurebis_masivi[3] = new Samkutxedi(5.5);
}

```

```

for ( int indexi = 0; indexi < figurebis_masivi.Length; indexi++ )
{
    label2.Text += figurebis_masivi[indexi].saxeli + " - ფართობი = " +
        figurebis_masivi[indexi].Partobi().ToString() + "\n";
}
figurebis_masivi[0].Naxva();
}

```

მართალია, Sibrtye ტიპის ობიექტების შექმნა შეუძლებელია, მაგრამ შესაძლებელია Sibrtye ტიპის მიმართებითი ცვლადის შექმნა. ამიტომ, წინა პროგრამისაგან განსხვავებით, პროგრამის ამ ვერსიაში გამოცხადებულია მასივი, რომელიც შედგება ოთხი და არა ხუთი ელემენტისაგან.

მემკვიდრეობითობის აკრძალვა

ზოგჯერ წამოიჭრება ისეთი კლასების შექმნის აუცილებლობა, რომელთა მემკვიდრეობით გადაცემა არ შეიძლება. ასეთ შემთხვევებში გამოიყენება sealed საკვანძო სიტყვა. ის ეთითება კლასის სახელის წინ. ასეთ კლასებს დალუქულ კლასებს უწოდებენ. ქვემოთ მოყვანილი კოდის ფრაგმენტი ახდენს ამის დემონსტრირებას:

```

// პროგრამა 7.13
// პროგრამაში ხდება მემკვიდრეობითობის აკრძალვის დემონსტრირება
sealed class Klas1
{
    // Klas1 კლასის კოდი
}
// ამ კლასის გამოცხადება არ არის ნამდვილი
class Klas2 : Klas1 // შეცდომა! შეუძლებელია Klas1 კლასის მემკვიდრე კლასის შექმნა
{
    // Klas2 კლასის კოდი
}

```

რადგან Klas1 კლასის გამოცხადებისას გამოყენებულია sealed საკვანძო სიტყვა, ამიტომ მისი მემკვიდრეობით გადაცემა დაუშვებელია. არ შეიძლება, აგრეთვე, abstract და sealed საკვანძო სიტყვების ერთდროულად გამოყენება.

პოლიმორფიზმის აკრძალვა

sealed საკვანძო სიტყვა შეგვიძლია, აგრეთვე, გამოვიყენოთ მეთოდის მიმართ მემკვიდრე კლასებში მათი ხელახალი განსაზღვრის აკრძალვის მიზნით. ასეთ მეთოდებს დალუქულ მეთოდებს უწოდებენ.

ქვემოთ მოყვანილი კოდის ფრაგმენტი ახდენს ამის დემონსტრირებას:

```

// პროგრამა 7.14
// პროგრამაში ხდება პოლიმორფიზმის აკრძალვის დემონსტრირება
class Sabazo
{
    public virtual void Metodi(Label lab1)
    {
        lab1.Text = "დალუქული საბაზო მეთოდი";
    }
}

```

```

    }
}
class Memkvidre : Sabazo
{
    sealed public override void Metodi(Label lab1)
    {
        lab1.Text += "\nდალუქული მემკვიდრე მეთოდი";
    }
}
private void button1_Click(object sender, EventArgs e)
{
    // პოლიმორფიზმის აკრძალვის დემონსტრირება
    Memkvidre obieqti = new Memkvidre();
    obieqti.Metodi(label1);
}

```

თუ შევექმნით Memkvidre კლასის მემკვიდრე კლასს, რომელშიც გამოვაცხადებთ Metodi სახელის მქონე მეთოდს, მაშინ კომპილატორი გასცემს შეტყობინებას შეცდომის შესახებ, რადგან sealed სიტყვა კრძალავს ამ მეთოდის გადატვირთვას.

ობიექტების ტიპების დაყვანა

თუ ხდება მემკვიდრე კლასის ობიექტის ტიპის დაყვანა საბაზო კლასის ტიპზე, მაშინ სრულდება დაყვანა *ზევით*. თუ ხდება საბაზო კლასის ობიექტის ტიპის დაყვანა მემკვიდრე კლასის ტიპზე, მაშინ სრულდება დაყვანა *ქვევით*.

ერთი კლასის ობიექტები შეიძლება გარდაექმნათ მეორე კლასის ობიექტების ტიპად იმ შემთხვევაში, თუ ეს კლასები *თავსებადია*. მემკვიდრე კლასი თავსებადია მის საბაზო კლასთან. მაგალითად, თუ Klasi3 არის Klasi2-ის მემკვიდრე, ხოლო Klasi2 არის Klasi1-ის მემკვიდრე, მაშინ Klasi3 თავსებადია Klasi1 და Klasi2 კლასებთან. შედეგად, Klasi1, Klasi2 და Klasi3 კლასების ობიექტები შეიძლება დაყვანილი იყოს ამ კლასების რომელიმე ტიპზე. ობიექტის ტიპის დაყვანის შემთხვევაში, მიმართვა შეგვეძლება მხოლოდ იმ წევრებთან, რომლებიც განსაზღვრულია იმ კლასში, რომლის ტიპზეც ხდება დაყვანა.

კლასები შეუთავსებია, თუ ერთი კლასი არ არის მეორე კლასის მემკვიდრე. მაგალითად, თუ Klasi2 არის Klasi1-ის მემკვიდრე და Klasi3 არის Klasi1-ის მემკვიდრე, მაშინ Klasi2 და Klasi3 შეუთავსებია. მოყვანილ პროგრამაში ხდება ზევით და ქვევით დაყვანის დემონსტრირება.

```

// პროგრამა 7.15
// პროგრამაში ხდება ზევით და ქვევით დაყვანის დემონსტრირება
class Sabazo
{
    public int ricxvi1;
}
class Memkvidre1 : Sabazo
{
    public int ricxvi2;
}
class Memkvidre2 : Sabazo
{

```

```

public int ricxvi3;
}
private void button1_Click(object sender, EventArgs e)
{
Memkvidre1 memkvidre_obj1 = new Memkvidre1();
// დაყვანა ზევით. სრულდება memkvidre_obj1-ის დაყვანა Sabazo ტიპზე
Sabazo sabazo_obj1 = ( Sabazo ) memkvidre_obj1;
sabazo_obj1.ricxvi1 = int.Parse(textBox1.Text);
label1.Text = sabazo_obj1.ricxvi1.ToString();

Memkvidre2 memkvidre_obj2 = new Memkvidre2();
// დაყვანა ზევით. სრულდება memkvidre_obj2-ის დაყვანა Sabazo ტიპზე
Sabazo sabazo_obj2 = ( Sabazo ) memkvidre_obj2;
sabazo_obj2.ricxvi1 = int.Parse(textBox2.Text);
label2.Text = sabazo_obj2.ricxvi1.ToString();

// დაყვანა ქვევით. სრულდება sabazo_obj2-ის დაყვანა Memkvidre2 ტიპზე
Memkvidre2 memkvidre_obj3 = ( Memkvidre2 ) sabazo_obj2;
memkvidre_obj3.ricxvi3 = int.Parse(textBox3.Text);
label3.Text = memkvidre_obj3.ricxvi3.ToString();
}

```

object კლასი

C# ენაში განსაზღვრულია სპეციალური object კლასი, რომელიც არის საბაზო ყველა სხვა კლასისათვის. სხვა სიტყვებით რომ ვთქვათ, ყველა კლასი არის object კლასის მემკვიდრე, ე.ი. object ტიპის მქონე მიმართვითი ცვლადი შეიძლება მიმართავდეს ნებისმიერი სხვა ტიპის ობიექტს. object სახელი - ესაა System.Object სახელის შემოკლებული ფორმა, რომელიც არის .NET Framework კლასების ბიბლიოთეკის შემადგენელი ნაწილი.

თავი 8. თვისებები, ინდექსატორები და ოპერატორების გადატვირთვა

თვისებები

თვისება არის კლასის წევრი, რომელიც საშუალებას გვაძლევს მეთოდების საშუალებით მივიღოთ ან შევცვალოთ ცვლადების (ველები) მნიშვნელობები. თვისებების გამოყენება მოხერხებულია მაშინ, როცა საჭიროა ცვლადის მნიშვნელობებზე შესასრულებელი ოპერაციების კონტროლი (იმის კონტროლი თუ რომელი ოპერაცია დასაშვებია ცვლადის მნიშვნელობებზე და რომელი - არა). მაგალითად, შეიძლება საჭირო გახდეს მნიშვნელობების დიაპაზონის შეზღუდვა, რომელიც ამ ცვლადს ენიჭება. ბუნებრივია, ასეთი მიზნის მისაღწევად შეგვიძლია გამოვიყენოთ პრივატული ცვლადი და მეთოდი, რომელიც მასთან მუშაობს. მაგრამ გაცილებით მარტივი და უკეთესია პრივატული ცვლადებისა და თვისებების ერთობლივი გამოყენება.

თვისება შედგება სახელისა და მიმართვის `get` და `set` მეთოდებისაგან. მიმართვის `get` მეთოდი გამოიყენება ცვლადის მნიშვნელობის მისაღებად, `set` მეთოდი კი ცვლადისთვის მნიშვნელობის მისანიჭებლად. `set` მეთოდს ავტომატურად გადაეცემა `value` არაცხადი პარამეტრი, რომელიც შეიცავს ცვლადისათვის მისანიჭებელ მნიშვნელობას. თვისების ძირითადი ღირსებაა ის, რომ მისი სახელი შეიძლება გამოვიყენოთ გამოსახულებებში და მინიჭების ოპერაციებში, როგორც ჩვეულებრივი ცვლადი. ამ დროს ავტომატურად გამოიძახება მიმართვის `get` და `set` მეთოდები. გარდა ამისა, თვისება მართავს ცვლადთან მიმართვას. თვისებაში არ ხდება ცვლადის გამოცხადება. საჭიროების მიხედვით თვისებაში შეიძლება განისაზღვროს მხოლოდ `get` ან `set` მეთოდი, ან ორივე ერთად.

თვისების სინტაქსია:

მიმართვის_მოდული_კატორი ტიპი თვისების_სახელი

```
{
get
{
get მეთოდის კოდი
}
set
{
set მეთოდის კოდი
}
}
```

აქ ტიპი თვისების ტიპია (მაგალითად, `int` ტიპი). თვისების განსაზღვრის შემდეგ მისი სახელის გამოყენება, იწვევს მიმართვის შესაბამისი მეთოდის გამოძახებას. თუ თვისების სახელი მითითებულია მინიჭების ოპერატორის მარცხნივ, მაშინ გამოიძახება `set` მეთოდი. თუ თვისების სახელი მითითებულია მინიჭების ოპერატორის მარჯვნივ, მაშინ გამოიძახება `get` მეთოდი.

ქვემოთ მოყვანილ პროგრამაში განისაზღვრება `ChemTviseba` თვისება, რომელიც უზრუნველყოფს `Tviseba` ცვლადთან მიმართვას. მოცემულ შემთხვევაში თვისება უშვებს ცვლადისათვის მხოლოდ დადებითი მნიშვნელობების მინიჭებას.

```
// პროგრამა 8.1
// პროგრამაში ხდება თვისებასთან მუშაობის დემონსტრირება
class MartiviTviseba
{
int cvladi;
```

```

public MartiviTviseba()
{
    cvladi = 0;
}
//      ChemiTviseba თვისების გამოცხადება
public int ChemiTviseba
{
    get
    {
        return cvladi;
    }
    set
    {
        //      tviseba ცვლადს ენიჭება value მნიშვნელობა თუ ის დადებითია
        if ( value >= 0 ) cvladi = value;
    }
}
private void button1_Click(object sender, System.EventArgs e)
{
    //
    MartiviTviseba obieqti = new MartiviTviseba();
    int ricxvi = Convert.ToInt32(textBox1.Text);

    label1.Text = obieqti.ChemiTviseba.ToString();
    //      ChemiTviseba თვისებას ენიჭება ricxvi ცვლადის დადებითი მნიშვნელობა
    obieqti.ChemiTviseba = ricxvi;          //      ricxvi ცვლადის მნიშვნელობა ავტომატურად
                                           //      ენიჭება value ცვლადს
    label2.Text = obieqti.ChemiTviseba.ToString();
    //      obieqti.ChemiTviseba თვისებისთვის უარყოფითი მნიშვნელობის მინიჭების მცდელობა
    obieqti.ChemiTviseba = -ricxvi;
    label3.Text = obieqti.ChemiTviseba.ToString();
}

```

ახლა დაწვრილებით განვიხილოთ პროგრამა. მასში განისაზღვრება ერთი პრივატული cvladi ცვლადი და ChemiTviseba თვისება, რომელიც მართავს ამ ცვლადთან მიმართებას. რადგან cvladi ცვლადი არის დახურული, მასთან მიმართვა შესაძლებელია მხოლოდ ChemiTviseba თვისების გამოყენებით.

ChemiTviseba თვისება განისაზღვრება როგორც public, ამიტომ მასთან მიმართვა შეიძლება შესრულდეს სხვა მეთოდების მხრიდან. მიმართვის get მეთოდი აბრუნებს ცვლადის მნიშვნელობას, ხოლო მიმართვის set მეთოდი კი cvladi ცვლადს ანიჭებს მნიშვნელობას მხოლოდ იმ შემთხვევაში, თუ ის არის დადებითი. ასეთი გზით, ChemiTviseba თვისება აკონტროლებს cvladi ცვლადისათვის მნიშვნელობების მინიჭებას.

თვისებების გამოყენებისას უნდა დავიცვათ შემდეგი შეზღუდვები:

- თვისება არ შეიძლება გადაეცეს მეთოდს, როგორც ref ან out პარამეტრი.
- თვისება არ შეიძლება იყოს გადატვირთული.
- თვისებამ არ უნდა შეცვალოს ცვლადის მნიშვნელობა get მეთოდის გამოყენებით.

ინდექსატორები

ინდექსატორი საშუალებას გვაძლევს ობიექტში მოთავსებულ ცვლადებს, აგრეთვე, მასივის ელემენტებს მივმართოთ ობიექტის სახელისა და ინდექსის საშუალებით. ინდექსატორების გამოყენების ძირითადი სფეროა სპეციალიზებული მასივების შექმნა, რომლებსაც ედებათ გარკვეული შეზღუდვები. ინდექსატორების სინტაქსი მასივების სინტაქსის მსგავსია. ინდექსატორებს შეიძლება ჰქონდეთ ერთი ან მეტი განზომილება.

ერთგანზომილებიანი ინდექსატორები

ერთგანზომილებიანი ინდექსატორის სინტაქსია:

მიმართვის_მოდული_კატორი ელემენტის_ტიპი this[ინდექსის_ტიპი ინდექსი]

```
{
მიმართვის get მეთოდი
get
{
get მეთოდის კოდი
}
მიმართვის set მეთოდი
set
{
set მეთოდის კოდი
}
}
```

აქ *ელემენტის_ტიპი* არის ინდექსატორის მიერ გაცემული ცვლადის ტიპი.

ინდექსატორის შემადგენლობაში განსაზღვრულია მიმართვის ორი მეთოდი `get` და `set`. ჩვეულებრივი მეთოდისაგან განსხვავებით მიმართვის მეთოდისათვის არ ეთითება დასაბრუნებელი მნიშვნელობის ტიპი და პარამეტრები. მიმართვის ორივე მეთოდი ინდექსატორის გამოყენებისას ავტომატურად გამოიძახება და პარამეტრებად იღებენ მნიშვნელობებს, რომლებიც მითითებულია ინდექსის ნაცვლად. თუ ინდექსატორი მითითებულია მინიჭების ოპერატორის მარცხენა ნაწილში, გამოიძახება მიმართვის `set` მეთოდი. თუ ინდექსატორი იმყოფება მინიჭების ოპერატორის მარჯვენა ნაწილში, მაშინ გამოიძახება მიმართვის `get` მეთოდი. საჭიროების მიხედვით ინდექსატორში შეიძლება განსაზღვრული იყოს მხოლოდ `get` ან მხოლოდ `set` მეთოდი, ან ორივე ერთად.

ქვემოთ მოყვანილ პროგრამაში ინდექსატორი გამოიყენება ობიექტის ორ ცვლადთან სამუშაოდ.

```
// პროგრამა 8.2
```

```
// პროგრამაში ინდექსატორი გამოიყენება ობიექტის ხუთ ცვლადთან სამუშაოდ
```

```
class Klasi
{
private int ricxvi1;
private int ricxvi2;
private int ricxvi3;
private int ricxvi4;
private int ricxvi5;
// ინდექსატორის გამოცხადება
public int this[int index]
```

```

{
get
{
switch ( index )
{
case 0: return ricxvi1;
case 1: return ricxvi2;
case 2: return ricxvi3;
case 3: return ricxvi4;
case 4: return ricxvi5;
default: return 0;
}
}
set
{
switch ( index )
{
case 0: this.ricxvi1 = value; break;
case 1: this.ricxvi2 = value; break;
case 2: this.ricxvi3 = value; break;
case 3: this.ricxvi4 = value; break;
case 4: this.ricxvi5 = value; break;
}
}
}
}
}

```

```

private void button1_Click(object sender, EventArgs e)
{
label1.Text = "";
Klasi obj1 = new Klasi();
int ind;
// obj1 ობიექტის ცვლადებისთვის მნიშვნელობების მინიჭება ინდექსატორის გამოყენებით
for ( ind = 0; ind < 5; ind++ )
obj1[ind] = ind + 10;
// obj1 ობიექტის ცვლადების მნიშვნელობების ეკრანზე გამოტანა
for ( ind = 0; ind < 5; ind++ )
label1.Text += obj1[ind].ToString() + " ";
}

```

ინდექსატორის გამოყენების ერთ-ერთი უპირატესობა არის ის, რომ ის იძლევა მასივთან მიმართვის ზუსტი კონტროლის შესაძლებლობას. კერძოდ, შეგვიძლია ვაკონტროლოთ გადის თუ არა მითითებული ინდექსი მასივის საზღვრებს გარეთ. ამის დემონსტრირება ხდება ქვემოთ მოყვანილ პროგრამაში.

// **პროგრამა 8.3**

// **პროგრამაში ხდება ინდექსატორის გამოყენება მასივთან სამუშაოდ**

```

class ChemiKlasi
{

```

```

int[] masivi;           // masivi არის მასივზე მითითებული
public int Sigrdze;
// გადის თუ არა ინდექსი დიაპაზონის გარეთ
public ChemiKlasi(int zoma)
{
    masivi = new int[zoma];
    Sigrdze = zoma;
}
// ინდექსატორის გამოცხადება
public int this[int index]
{
    // მიმართვის get მეთოდი
    get
    {
        if ( ( index >= 0 ) && ( index < Sigrdze ) )
            return masivi[index];
        else return 0;
    }
    // მიმართვის set მეთოდი
    set
    {
        if ( ( index >= 0 ) && ( index < Sigrdze ) )
            masivi[index] = value;
        }
    }
}
private void button1_Click(object sender, System.EventArgs e)
{
    label1.Text = "";
    Random rand_obj = new Random();
    ChemiKlasi obieqti = new ChemiKlasi(5);
    int indexi;
    // set მეთოდის გამოძახება
    for ( indexi = 0; indexi < 5; indexi++ )
        obieqti[indexi] = rand_obj.Next(10);
    // get მეთოდის გამოძახება
    for ( indexi = 0; indexi < 5; indexi++ )
        label1.Text += obieqti[indexi].ToString() + " ";
}

```

პროგრამაში გამოიყენება ინდექსატორი, რომელიც ამოწმებს ინდექსის მნიშვნელობას და თავიდან გვაცხადებს ზღვრული მნიშვნელობების გარეთ გასვლას. ინდექსატორის გამოცხადება იწყება სტრიქონიდან:

```
public int this[int index]
```

ის ოპერაციებს ასრულებს int ტიპის ელემენტებზე.

ინდექსის სისწორეს Shemowmeba მეთოდი ამოწმებს. თუ ინდექსის მნიშვნელობა დასაშვებ საზღვრებშია, მაშინ მეთოდი გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას.

მიმართვის get მეთოდი იძახებს Shemowmeba მეთოდს. თუ ინდექსი იმყოფება საზღვრების შიგნით, გაიცემა ამ ინდექსის შესაბამისი ელემენტი. თუ ინდექსი გადის მასივის საზღვრებს გარეთ, მაშინ შესრულდება return 0; ოპერატორი, ე.ი. მასივის ელემენტის ნაცვლად გაიცემა 0. alami ცვლადში ინახება შემოწმების შედეგი. ეს ცვლადი შემდეგ მოწმდება გამომძახებელ პროგრამაში მასივთან ყოველი მიმართვის შემდეგ მიმართვის წარმატებულობის შესამოწმებლად და შეცდომის შესახებ ინფორმაციის გამოსატანად.

მიმართვის set მეთოდიც იძახებს Shemowmeba მეთოდს ინდექსის შესამოწმებლად. თუ index პარამეტრი არ გადის მასივის საზღვრებს გარეთ, მაშინ მნიშვნელობა, რომელიც გადაიცემა value პარამეტრით, ენიჭება შესაბამის ელემენტს. წინააღმდეგ შემთხვევაში მნიშვნელობის მინიჭება არ ხდება.

ინდექსატორების გამოყენებისას არსებობს ერთი მნიშვნელოვანი შეზღუდვა: მათთვის არ შეიძლება ref ან out მოდიფიკატორების გამოყენება.

მრავალგანზომილებიანი ინდექსატორები

ინდექსატორები შეიძლება შეიქმნას მრავალგანზომილებიანი მასივებისთვისაც. ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია ორგანზომილებიან ინდექსატორთან მუშაობის მაგალითი.

```
// პროგრამა 8.4
// პროგრამაში ხდება ორგანზომილებიან ინდექსატორთან მუშაობის დემონსტრირება
class ChemiKlasi
{
    int[,] masivi;
    int striqoni, sveti;
    public int Length;
    // მითითებული პარამეტრების მქონე მასივის შექმნა
    public ChemiKlasi(int par1, int par2)
    {
        striqoni = par1;
        sveti = par2;
        masivi = new int[striqoni, sveti];
        Length = striqoni * sveti;
    }
    // ორგანზომილებიანი ინდექსატორის გამოცხადება
    public int this[int index1, int index2]
    {
        // მიმართვის get მეთოდი
        get
        {
            if ( ( ( index1 >= 0 ) && ( index1 < striqoni ) ) && ( ( index2 >= 0 ) && ( index2 < sveti ) ) )
                return masivi[index1, index2];
            else return 0;
        }
        // მიმართვის set მეთოდი
        set
        {
            if ( ( ( index1 >= 0 ) && ( index1 < striqoni ) ) && ( ( index2 >= 0 ) && ( index2 < sveti ) ) )
                masivi[index1, index2] = value;
        }
    }
}
```

```

    }
}
private void button1_Click(object sender, System.EventArgs e)
{
    label1.Text = "";
    Random rand_obj = new Random();
    ChemiKlasi obieqti = new ChemiKlasi(3, 3);
    int striqoni, sveti;
    //      მუშაობს set მეთოდი
    for ( striqoni = 0; striqoni < 3; striqoni++ )
        for ( sveti = 0; sveti < 3; sveti++ )
            obieqti[striqoni, sveti] = rand_obj.Next(10);
    //      მუშაობს get მეთოდი
    for ( striqoni = 0; striqoni < 3; striqoni++ )
    {
        for ( sveti = 0; sveti < 3; sveti++ )
            label1.Text += obieqti[striqoni, sveti].ToString() + " ";
        label1.Text += "\n";
    }
}
}
}

```

ოპერატორების გადატვირთვა

C# ენაში შეგვიძლია განვსაზღვროთ მოქმედებები (ისინი განსხვავდება ოპერატორის მიერ ავტომატურად შესრულებული მოქმედებებისაგან), რომლებსაც ოპერატორი შეასრულებს ჩვენს მიერ შექმნილი კლასის მიმართ. ამას ეწოდება ოპერატორის გადატვირთვა ან ხელახალი განსაზღვრა. ოპერატორის ფუნქციები განსხვავდებიან იმის და მიხედვით, თუ რომელი კლასის მიმართ იქნებიან გამოყენებული. მაგალითად, კლასში, რომელიც განსაზღვრავს ბმულ სიას, + ოპერატორი გამოიყენება სიისათვის ობიექტის დასამატებლად. კლასში, რომელიც ახდენს სტეკის რეალიზებას, ეს ოპერატორი გამოიყენება სტეკში ობიექტის მოსათავსებლად და ა.შ.

უნდა გვახსოვდეს, რომ გადატვირთვის შესრულებისას ოპერატორის საწყისი დანიშნულება არ იკარგება. უბრალოდ მის არსენალს ემატება ახალი ფუნქციები, რომლებიც გამოიყენებული იქნება გარკვეული კლასის მიმართ. მაგალითად, + ოპერატორის გადატვირთვის შემთხვევაში მის არსენალს დაემატება ახალი ფუნქცია, აგრეთვე, შენარჩუნებული იქნება მისი საწყისი დანიშნულება - შეასრულოს მთელი რიცხვების შეკრება.

ოპერატორის გადატვირთვის გამოყენების ძირითადი უპირატესობა იმაში მდგომარეობს, რომ ადვილად შეიძლება პროგრამაში კლასის ახალი ტიპის ინტეგრირება. თუ კლასისათვის განსაზღვრულია ოპერატორები, მაშინ ამ კლასის ობიექტებზე ოპერაციები შეიძლება შევასრულოთ ჩვეულებრივი სინტაქსის გამოყენებით, ე.ი. კლასის ცვლადებთან სამუშაოდ საკმარისია შევიტანოთ ასეთი სტრიქონი: ობიექტი + ობიექტი.

ოპერატორის გადატვირთვისთვის გამოიყენება operator სიტყვა. ის განსაზღვრავს ოპერატორის მეთოდს, რომელიც აღწერს ოპერატორის მიერ შესასრულებელ მოქმედებებს.

ოპერატორის მეთოდი

C# ენაში ძირითადად გამოიყენება ოპერატორების ორი სახე - უნარული და ბინარული. უნარულია ოპერატორი, რომელსაც ერთი ოპერანდი აქვს. ბინარულია ოპერატორი, რომელსაც ორი ოპერანდი აქვს.

უნარული ოპერატორის გადატვირთვისათვის გამოყენებული მეთოდის სინტაქსია:
მიმართვის_მოდიფიკატორი static შედეგის_ტიპი operator ოპერატორი(ტიპი ოპერანდი)
{
მეთოდის კოდი
}

ბინარული ოპერატორის გადატვირთვისათვის გამოყენებული მეთოდის სინტაქსია:
მიმართვის_მოდიფიკატორი static შედეგის_ტიპი operator ოპერატორი(ტიპი1 ოპერანდი1, ტიპი2 ოპერანდი2)
{
მეთოდის კოდი
}

აქ ოპერატორი არის გადატვირთვადი ოპერატორი, მაგალითად, + ან -. შედეგის_ტიპი არის იმ მნიშვნელობის ტიპი, რომელიც გაიცემა მოცემული მეთოდის მიერ.

უნარული ოპერატორების გადატვირთვისას ოპერანდი უნდა ეკუთვნოდეს იმ კლასის ტიპს, რომლისთვისაც განისაზღვრება ოპერატორი. ბინარული ოპერატორების გადატვირთვისას ერთ-ერთი ოპერანდის ტიპი უნდა ემთხვეოდეს იმ კლასის ტიპს, რომლისთვისაც ხდება ოპერატორის ხელახალი განსაზღვრა.

C# ენაში არ შეიძლება ოპერატორების გადატვირთვა სტანდარტული ტიპის მქონე ობიექტებისათვის. მაგალითად, არ შეიძლება + ოპერატორის გადატვირთვა უკვე არსებული ტიპებისათვის, როგორცაა მაგალითად, int ან string ტიპი. ოპერატორების გადატვირთვა შეიძლება მხოლოდ ჩვენს მიერ შექმნილი კლასების ობიექტებისათვის. გარდა ამისა, ოპერატორის პარამეტრები არ უნდა შეიცავდეს ref ან out მოდიფიკატორს.

ბინარული ოპერატორების გადატვირთვა

განვიხილოთ პროგრამა, რომელშიც ხდება + ბინარული ოპერატორის გადატვირთვა. თავდაპირველად იქმნება Sivrcე კლასი, რომელიც შეიცავს სამგანზომილებიანი ობიექტის კოორდინატებს. გადატვირთული + ოპერატორი ასრულებს Sivrcე კლასის ორი ობიექტის შესაბამისი კოორდინატების შეკრებას.

```
// პროგრამა 8.5
// პროგრამაში ხდება + ბინარული ოპერატორის გადატვირთვა
class Sivrcე
{
int x, y, z; // ობიექტის კოორდინატები
public Sivrcე()
{
x = y = z = 0;
}
public Sivrcე(int a, int b, int c)
{
x = a;
y = b;
z = c;
}
// + ბინარული ოპერატორის გადატვირთვა
```



```

public static Sivrce operator + (Sivrce operand1, Sivrce operand2)
{
    Sivrce shedegi = new Sivrce();
    // კოორდინატების შეკრება და შედეგის დაბრუნება
    shedegi.x = operand1.x + operand2.x; // ამ სამ სტრიქონში სრულდება მთელი
    shedegi.y = operand1.y + operand2.y; // რიცხვების შეკრების ოპერაცია,
    shedegi.z = operand1.z + operand2.z; // რომლებისთვისაც + ოპერატორი ინახავს
    // თავის საწყის მნიშვნელობას

    return shedegi;
}
// x, y და z კოორდინატების მნიშვნელობების გამოტანა
public string show()
{
    return "x = " + x.ToString() + " y = " + y.ToString() + " z = " + z.ToString();
}
}
private void button1_Click(object sender, System.EventArgs e)
{
    // პროგრამაში ხდება გადატვირთული ოპერატორების გამოყენების დემონსტრირება
    Sivrce obieqti1 = new Sivrce(3, 4, 5);
    Sivrce obieqti2 = new Sivrce(5, 5, 5);
    Sivrce obieqti3 = new Sivrce();

    label1.Text = obieqti1.show(); // obieqti1-ის კოორდინატების გამოტანა
    label2.Text = obieqti2.show(); // obieqti2-ის კოორდინატების გამოტანა
    // obieqti1-სა და obieqti2-ის შეკრების ოპერაცია
    obieqti3 = obieqti1 + obieqti2;
    // obieqti1-სა და obieqti2-ის შეკრების ოპერაციის შედეგი
    label3.Text = obieqti3.show();
    // obieqti1-ის, obieqti2-სა და obieqti3-ის შეკრების ოპერაცია
    obieqti3 = obieqti1 + obieqti2 + obieqti3;
    label4.Text = obieqti3.show();
}

```

operator +() მეთოდის შიგნით ცალკეული კოორდინატების მნიშვნელობების შეკრება დაიყვანება მთელი რიცხვების შეკრებაზე. x, y და z კოორდინატებს აქვთ მთელი ტიპი, ამიტომ + ოპერატორის გადატვირთვა Sivrce კლასის ობიექტებისათვის გავლენას არ მოახდენს მოქმედებებზე, რომლებიც ამ ოპერატორის მიერ სრულდება მთელ რიცხვებზე.

- ოპერატორი ისევე მოქმედებს, როგორც +, მაგრამ მისი გადატვირთვისას უნდა გავითვალისწინოთ ოპერანდების მიმდევრობა. გავიხსენოთ, რომ შეკრება კომუტაციურია, ხოლო გამოკლება - არა (ე.ი. $A - B$ გამოსახულება არაა $B - A$ გამოსახულების იგივეური). ყველა ბინარული ოპერატორისთვის პირველი პარამეტრი არის მარცხენა ოპერანდი, მეორე კი - მარჯვენა. არაკომუტაციური ოპერატორების გადატვირთვისას უნდა გავითვალისწინოთ ოპერანდების ურთიერთგანლაგება, კომუტაციური ოპერატორების გადატვირთვისას კი - არა.

უნარული ოპერატორების გადატვირთვა

უნარული ოპერატორების გადატვირთვა ხორციელდება ბინარული ოპერატორების გადატვირთვის ანალოგიურად, მაგრამ, ამ შემთხვევაში მხოლოდ ერთი ოპერანდი გვაქვს.

უნარული ოპერატორის გადატვირთვა განვიხილოთ ინკრემენტისა და დეკრემენტის ოპერატორების მაგალითზე. როგორც ვიცით მათი დანიშნულებაა შესაბამისი ოპერანდების მნიშვნელობების ერთით გაზრდა და შემცირება, ამიტომ ამ ოპერატორების გადატვირთულმა ვერსიებმაც უნდა შეასრულონ იგივე მოქმედებები.

ქვემოთ მოყვანილი პროგრამაში ხდება ++ უნარული ოპერატორის გადატვირთვის დემონსტრირება.

```
//      პროგრამა 8.6
//      პროგრამაში ხდება ++ უნარული ოპერატორის გადატვირთვა
class Sivrcce
{
int x, y, z;           //      ობიექტის კოორდინატები
public Sivrcce()
{
x = y = z = 0;
}
public Sivrcce(int a, int b, int c)
{
x = a;
y = b;
z = c;
}
//      ++ უნარული ოპერატორის გადატვირთვა
public static Sivrcce operator ++(Sivrcce operandi)
{
//      სრულდება არგუმენტის ცვლილება
operandi.x++;
operandi.y++;
operandi.z++;
return operandi;
}
//      x, y და z კოორდინატების მნიშვნელობების გამოტანა
public void show()
{
DialogResult pasuxi = MessageBox.Show("x="+x.ToString()+"\ny="+y.ToString()+"\nz="+z.ToString());
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება გადატვირთული ოპერატორების გამოყენების დემონსტრირება
Sivrcce obieqti = new Sivrcce(3, 4, 5);

obieqti.show();           //      obieqti-ის კოორდინატების გამოტანა
//      გადატვირთული ++ ოპერატორის შესრულება
obieqti++;
obieqti.show();
}
}
```

როგორც ვიცით, ++ და -- ოპერატორებს აქვთ პრეფიქსური და პოსტფიქსური ფორმა. მათი გადატვირთვის შედეგად ორივე ფორმისათვის გამოიძახება ერთი და იგივე მეთოდი.

გადატვირთვის შესრულებისას შეუძლებელია მივუთითოთ განსხვავება ამ ოპერატორების პრეფიქსურ და პოსტფიქსურ ფორმებს შორის.

შედარების ოპერატორების გადატვირთვა

შედარების ოპერატორები, როგორცაა ==, !=, <, >, <= ან >= შეიძლება იყოს გადატვირთული. როგორც წესი, შედარების გადატვირთული ოპერატორები აბრუნებენ true ან false მნიშვნელობას. ამიტომ, რჩება ამ ოპერატორების ჩვეულებრივი გამოყენების შესაძლებლობა, ხოლო შედარების გადატვირთული ოპერატორები შეგვიძლია გამოვიყენოთ პირობის გამოსახულებებში.

ქვევით მოყვანილია Sivrce კლასის ვერსია, რომელშიც სრულდება < და > ოპერატორების გადატვირთვა. ამ ოპერატორების გადატვირთული ვერსიების თანახმად ერთი ობიექტი ითვლება მეორეზე დიდად, თუ ობიექტის ყველა კოორდინატი მეტია მეორე ობიექტის შესაბამის კოორდინატებზე. შესაბამისად, ერთი ობიექტი ითვლება მეორეზე ნაკლებად თუ მისი კოორდინატები ნაკლებია მეორე ობიექტის კოორდინატებზე.

// პროგრამა 8.7

// პროგრამაში ხდება > და < შედარების ოპერატორების გადატვირთვა

```
class Sivrce
{
int x, y, z;           // ობიექტის კოორდინატები
public Sivrce()
{
    x = y = z = 0;
}
public Sivrce(int a, int b, int c)
{
    x = a;
    y = b;
    z = c;
}
// < ოპერატორის გადატვირთვა
public static bool operator < (Sivrce operand1, Sivrce operand2)
{
if ( ( operand1.x < operand2.x ) && ( operand1.y < operand2.y ) &&
    ( operand1.z < operand2.z ) ) return true;
    else return false;
}
// > ოპერატორის გადატვირთვა
public static bool operator > (Sivrce operand1, Sivrce operand2)
{
if ( ( operand1.x > operand2.x ) && ( operand1.y > operand2.y ) &&
    ( operand1.z > operand2.z ) ) return true;
    else return false;
}
// x, y და z კოორდინატების მნიშვნელობების გამოტანა
public void Show()
{
```

```

DialogResult pasuxi =
MessageBox.Show("x = " + x.ToString() + "\ny = " + y.ToString()+ "\nz = " + z.ToString());
}
private void button1_Click(object sender, System.EventArgs e)
{
//      პროგრამაში ხდება გადატვირთული ოპერატორების გამოყენების დემონსტრირება
Sivrcе obieqti1 = new Sivrcе(3, 4, 5);
Sivrcе obieqti2 = new Sivrcе(5, 5, 5);
obieqti1.Show();           // obieqti1-ის კოორდინატები
obieqti2.Show();           // obieqti2-ის კოორდინატები
if ( obieqti1 > obieqti2 )
    label1.Text = "გამოსახულება obieqti1 > obieqti2 სამართლიანია";
if ( obieqti1 < obieqti2 )
    label1.Text = "გამოსახულება obieqti1 < obieqti2 სამართლიანია";
}

```

შედარების ოპერატორების გადატვირთვა უნდა შევასრულოთ წყვილობრივად. მაგალითად, თუ გადაიტვირთება < ოპერატორი, მაშინ, უნდა გადაიტვირთოს > ოპერატორიც და პირიქით. C# ენაში ისევე, როგორც დაპროგრამების სხვა ენებში, არსებობს შედარების ოპერატორების შემდეგი წყვილები:

```

==      !=
<       >
<=     >=

```

ოპერატორების გადატვირთვისას არსებული შეზღუდვები

ოპერატორების გადატვირთვა დაკავშირებულია გარკვეულ შეზღუდვებთან. კერძოდ, არ შეიძლება ოპერაციების პრიორიტეტებისა და ოპერატორის ოპერანდების რაოდენობის შეცვლა. ამას გარდა, არ შეიძლება გადატვირთული იყოს მინიჭების ოპერატორი და მინიჭების შედგენილი ოპერატორები. აგრეთვე, ქვემოთ მოყვანილი ოპერატორები:

```

&&      ||      []      ()      new      is
sizeof   typeof ?  ->     .      =

```

თავი 9. დელეგატები, მოვლენები, ინტერფეისები და სახელების სივრცე

დელეგატები

დელეგატი - ესაა ობიექტი, რომელიც მეთოდს მიმართავს. დელეგატები გამოიყენება მეთოდების გამოძახებისათვის. ამასთან, გამოსაძახებელი მეთოდის არჩევა ხდება დინამიურად, პროგრამის შესრულების დროს. ამიტომ, დელეგატები იმ შემთხვევაში გამოიყენება, როცა წინასწარ არ ვიცით, თუ რომელი მეთოდი უნდა გამოვიძახოთ. დელეგატების გამოყენების კიდევ ერთი უპირატესობა იმაში მდგომარეობს, რომ ისინი უზრუნველყოფენ მოვლენებს.

დელეგატი ინახავს გამოსაძახებელი მეთოდის სახელს, მის მიერ დაბრუნებული შედეგის ტიპსა და პარამეტრების სიას, ანუ მეთოდის სიგნატურას (ხელმოწერას). ამრიგად, მეთოდის სიგნატურა - ესაა მეთოდის სახელი, მის მიერ დაბრუნებული მნიშვნელობის ტიპი და პარამეტრების სია.

როგორც ცნობილია, მეთოდისათვის მეხსიერებაში გარკვეული ზომის უბანი გამოიყოფა. ამ უბნის მისამართი - ესაა მეთოდის შესვლის წერტილი. სწორედ ამ მისამართის ინიცირება ხდება მეთოდის გამოძახებისას. მეთოდის მისამართი ენიჭება დელეგატს. თუ დელეგატი მიმართავს მეთოდს, მაშინ მოცემული მეთოდი შეგვიძლია გამოვიძახოთ ამ დელეგატის საშუალებით. გარდა ამისა, ერთი და იგივე დელეგატი შეგვიძლია გამოვიყენოთ სხვადასხვა მეთოდის გამოძახებისათვის. ამისათვის, მას უნდა მივანიჭოთ ამ მეთოდებზე მიმართვა.

დელეგატის გამოცხადებისათვის უნდა გამოვიყენოთ `delegate` საკვანძო სიტყვა. დელეგატის გამოცხადების სინტაქსია:

`delegate ტიპი დელეგატის_სახელი(პარამეტრების_სია);`

აქ ტიპი არის იმ მნიშვნელობის ტიპი, რომელსაც ის მეთოდი აბრუნებს, რომელიც დელეგატის მიერ იქნება გამოძახებული.

როგორც უკვე აღვნიშნეთ, დელეგატს შეუძლია გამოიძახოს ის მეთოდი, რომლის სიგნატურა შეესაბამება დელეგატის სიგნატურას. ეს საშუალებას იძლევა პროგრამის შესრულების დროს ავირჩიოთ გამოსაძახებელი მეთოდი. ასეთი მეთოდი შეიძლება იყოს ობიექტის მეთოდი ან კლასში განსაზღვრული სტატიკური მეთოდი. მეთოდი მხოლოდ მაშინ შეიძლება გამოვიძახოთ, როცა მისი ხელმოწერა შეესაბამება დელეგატის ხელმოწერას.

ქვემოთ მოყვანილ პროგრამაში ხდება დელეგატთან მუშაობის დემონსტრირება:

// **პროგრამა 9.1**

// **პროგრამაში ხდება დელეგატთან მუშაობის დემონსტრირება**

```
delegate string Delegati(string striqoni1);
class ChemiKlasi
{
// მეთოდი სტრიქონში ინტერვალებს დეფისებით ცვლის
public static string Shecvla(string striqoni2)
{
return striqoni2.Replace(' ', '-');
}
// მეთოდი სტრიქონიდან შლის ინტერვალებს
public static string Washla(string striqoni2)
{
string temp = "";
int index;
```

```

for (index = 0; index < striqoni2.Length; index++ )
    if ( striqoni2[index] != ' ' ) temp += striqoni2[index];
return temp;
}
//      მეთოდი უკუ მიმდევრობით ალაგებს სტრიქონის სიმბოლოებს
public static string Shebruneba(string striqoni2)
{
string temp = "";
int index;

for (index = striqoni2.Length - 1; index >= 0; index-- )
    temp += striqoni2[index];
return temp;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      დელეგატის შექმნა
Delegati obieqti = new Delegati(ChemiKlasi.Shecvla);
string striqoni;

//      Shecvla() მეთოდის გამოძახება დელეგატის საშუალებით
striqoni = obieqti("მეთოდების მუშაობის შემოწმება.");
label1.Text = striqoni;

obieqti = new Delegati(ChemiKlasi.Washla);
//      Washla() მეთოდის გამოძახება დელეგატის საშუალებით
striqoni = obieqti("მეთოდების მუშაობის შემოწმება.");
label2.Text = striqoni;

obieqti = new Delegati(ChemiKlasi.Shebruneba);
//      Shebruneba() მეთოდის გამოძახება დელეგატის საშუალებით
striqoni = obieqti("მეთოდების მუშაობის შემოწმება.");
label3.Text = striqoni;
}

```

პროგრამაში გამოცხადებულია Delegati დელეგატი, რომელსაც აქვს string ტიპის ერთი პარამეტრი და აბრუნებს string ტიპის მნიშვნელობას. ChemiKlasi კლასში გამოცხადებულია სამი სტატიკური მეთოდი დელეგატის შესაბამისი სიგნატურით.

პროგრამაში იქმნება Delegati ტიპის obieqti ობიექტი, რომელსაც ენიჭება Shecvla() მეთოდზე მიმართვა.

```
Delegati obieqti = new Delegati(Shecvla);
```

აქ Shecvla() მეთოდი გადაეცემა დელეგატის კონსტრუქტორს პარამეტრის სახით. გამოიყენება მხოლოდ მეთოდის სახელი პარამეტრის მითითების გარეშე, ე.ი. დელეგატის ეგზემპლარის შექმნისას ეთითება იმ მეთოდის სახელი, რომელზეც უნდა მიუთითებდეს დელეგატი. მეთოდის ხელმოწერა უნდა შეესაბამებოდეს დელეგატის გამოცხადებაში განსაზღვრულ ხელმოწერას. თუ ეს პირობა არ სრულდება, მაშინ პროგრამის კომპილირებისას ადგილი ექნება შეცდომას.

მომდევნო სტრიქონში obieqti დელეგატი გამოიყენება Shecvla() მეთოდის გამოძახებისათვის. შემდეგ, obieqti დელეგატს ენიჭება მიმართვა Washla() მეთოდზე და სრულდება Washla() მეთოდის გამოძახება.

პროგრამის ბოლოს obieqti დელეგატს ენიჭება Shebruneba() მიმართვა და სრულდება შესაბამისი მეთოდის გამოძახება.

დელეგატებს, სტატიკურ მეთოდებზე მიმართვის გარდა, შეიძლება მიენიჭოს აგრეთვე, ობიექტის მეთოდებზე მიმართვა. ამის დემონსტრირება ხდება ქვემოთ მოყვანილ პროგრამაში.

// **პროგრამა 9.2**

// **პროგრამაში ხდება დელეგატისათვის ობიექტის მეთოდზე მიმართვის მინიჭება**

```
delegate string Delegati(string striqoni1);
class ChemiKlasi
{
public string Shecvla(string striqoni2)
{
return striqoni2.Replace(' ', '-');
}
public string Washla(string striqoni2)
{
string temp = "";
int i;

for ( i = 0; i < striqoni2.Length; i++ )
    if (striqoni2[i] != ' ') temp += striqoni2[i];
return temp;
}
public string Shebruneba(string striqoni2)
{
string temp = "";
int i;

for ( i = striqoni2.Length - 1; i >= 0; i-- )
    temp += striqoni2[i];
return temp;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti = new ChemiKlasi();
// დელეგატის ეგზემპლარის შექმნა
Delegati delegati_obieqti = new Delegati(obieqti.Shecvla);
string striqoni;
// მეთოდის გამოძახება დელეგატის საშუალებით
striqoni = delegati_obieqti("მეთოდების მუშაობის შემოწმება");
label1.Text = striqoni;

delegati_obieqti = new Delegati(obieqti.Washla);
striqoni = delegati_obieqti("მეთოდების მუშაობის შემოწმება");
```

```

label2.Text = striqoni;

delegati_obieqti = new Delegati(obieqti.Shebruneba);
striqoni = delegati_obieqti("მეთოდების მუშაობის შემოწმება");
label3.Text = striqoni;
}

```

პროგრამის მუშაობის შედეგად ეკრანზე გამოიკვამ იგივე სტრიქონები, რომლებიც წინა პროგრამაში. მაგრამ, ამ შემთხვევაში დელეგატი მეთოდებს მიმართავს ChemiKlasi კლასის ობიექტის გამოყენებით.

დელეგატების მრავალმისამართიანობა

დელეგატს შეუძლია შეინახოს რამდენიმე მეთოდის მისამართი. ამ მისამართების მიმდევრობით ინიციალიზების გზით დელეგატს შეუძლია ერთმანეთის მიყოლებით გამოიძახოს შესაბამისი მეთოდები. დელეგატის ამ თვისებას მრავალმისამართიანობა ეწოდება. გამოსაძახებელი მეთოდების მისამართების ასეთი მიმდევრობა, ანუ მეთოდებზე მიმართვების მწკრივი, შემდეგნაირად იქმნება. თავდაპირველად იქმნება დელეგატი (დელეგატი-ობიექტი), შემდეგ კი გამოიყენება += ოპერატორი მწკრივისათვის მეთოდების მისამართების დასამატებლად. შედეგად, დელეგატი მრავალმისამართიანი ხდება. მწკრივიდან მეთოდის მისამართების წასაშლელად გამოიყენება -= ოპერატორი. ერთადერთი შეზღუდვაა ის, რომ დელეგატს, რომელიც ინახავს რამდენიმე მიმართვას, უნდა ჰქონდეს დასაბრუნებელი მნიშვნელობის void ტიპი.

ქვემოთ მოყვანილია პროგრამა, რომელიც ახდენს მრავალმისამართიანი დელეგატის გამოყენების დემონსტრირებას. რადგან, მეთოდის მიერ დაბრუნებული შედეგის ტიპია void, ამიტომ გამოიძახებელი პროგრამისათვის შეცვლილი სტრიქონის დასაბრუნებლად გამოიყენება ref მოდიფიკატორი.

```

// პროგრამა 9.3
// პროგრამაში ხდება მრავალმისამართიან დელეგატთან მუშაობის დემონსტრირება
delegate void Delegati(ref string striqoni1);
class ChemiKlasi
{
public void Shecvla(ref string striqoni2)
{
striqoni2 = striqoni2.Replace(' ', '-');
}
public void Washla(ref string striqoni2)
{
string temp = "";
int i;

for ( i = 0; i < striqoni2.Length; i++ )
    if (striqoni2[i] != ' ') temp += striqoni2[i];
striqoni2 = temp;
}
public void Shebruneba(ref string striqoni2)
{
string temp = "";
int i;

```



```

for ( i = striqoni2.Length - 1; i >= 0; i-- )
    temp += striqoni2[i];
striqoni2 = temp;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti = new ChemiKlasi();
Delegati delegati_obieqti;
Delegati Shecvla_mimartva = new Delegati(obieqti.Shecvla);
Delegati Washla_mimartva = new Delegati(obieqti.Washla);
Delegati Shebruneba_mimartva = new Delegati(obieqti.Shebruneba);
string striqoni = " მეთოდების მუშაობის შემოწმება";
label1.Text = striqoni;
// დელეგატს ორი მიმართვა ენიჭება
delegati_obieqti = Shecvla_mimartva;
delegati_obieqti += Shebruneba_mimartva; // მიმართვების მწკრივის შექმნა
// მრავალმისამართიანი დელეგატის გამოძახება
delegati_obieqti(ref striqoni);
label2.Text = striqoni;
// მიმართვების მწკრივიდან Shecvla მიმართვის წაშლა და Washla მიმართვის დამატება
delegati_obieqti -= Shecvla_mimartva;
delegati_obieqti += Washla_mimartva; // მიმართვების ახალი მწკრივის შექმნა
striqoni = " მეთოდების მუშაობის შემოწმება";
// მრავალმისამართიანი დელეგატის გამოძახება
delegati_obieqti(ref striqoni);
label3.Text = striqoni;
}

```

პროგრამაში იქმნება ოთხი დელეგატი (დელეგატი-ობიექტი). delegati_obieqti დელეგატს აქვს null მნიშვნელობა, რადგან შექმნისას ის არ იყო ინიციალიზებული. დანარჩენი სამი დელეგატი მიმართავს შესაბამის მეთოდებს. შემდეგ, პროგრამაში იქმნება მრავალმისამართიანი დელეგატი, რომელიც იძახებს Shecvla() და Shebruneba() მეთოდებს. ეს ხორციელდება სამი ოპერატორის საშუალებით:

```

delegati_obieqti = Shecvla_mimartva;
delegati_obieqti += Shebruneba_mimartva;
delegati_obieqti(ref striqoni);

```

პირველ ოპერატორში delegati_obieqti დელეგატს ენიჭება Shecvla_mimartva მიმართვა. შემდეგ += ოპერატორის საშუალებით მწკრივს ემატება Shebruneba_mimartva მიმართვა. delegati_obieqti დელეგატთან მიმართვისას ჯერ შესრულდება Shecvla() მეთოდი, შემდეგ კი Shebruneba() მეთოდი. delegati_obieqti -= Shecvla_mimartva; ოპერატორი მწკრივიდან წაშლის Shecvla_mimartva მიმართვას, ხოლო delegati_obieqti += Washla_mimartva; ოპერატორი მწკრივს დაუმატებს Washla_mimartva მიმართვას. შედეგად, delegati_obieqti დელეგატთან მიმართვისას ჯერ შესრულდება Shebruneba() მეთოდი, შემდეგ კი Washla() მეთოდი.

ანონიმური მეთოდები

ანონიმური მეთოდი, რომელიც არ არსებობს როგორც ტრადიციული მეთოდი, ე.ი. არ არის რაიმე კლასის მეთოდი. ანონიმური მეთოდი იქმნება მხოლოდ იმისათვის, რომ გამოვიყენოთ როგორც მიზნობრივი მეთოდი დელეგატისთვის. ანონიმური მეთოდის შექმნის სინტაქსია:

delegate (პარამეტრები)

```
{
// ანონიმური მეთოდის კოდი
};
```

პარამეტრები უნდა შეესაბამებოდეს დელეგატის პარამეტრებს.

ანონიმურ მეთოდს ახასიათებს ის, რომ ის ლოკალურია იმ ბლოკის მიმართ, რომელშიც ის არის მოთავსებული და შეუძლია მიმართოს ამ ბლოკში გამოცხადებულ ყველა ლოკალურ ცვლადს. ასეთი ცვლადის გამოყენების შემთხვევაში, ის ხდება გარე. გარე ცვლადები არ იშლება მოქმედების უზნის გარეთ გასვლისას, როგორც ეს მოსდით ლოკალურ ცვლადებს. გარე ცვლადები აგრძელებენ არსებობას მანამ, სანამ დაამთავრებს მუშაობას ის მეთოდი, რომელშიც ისი გამოიყენება.

მოყვანილი პროგრამით ხდება ანონიმურ მეთოდთან მუშაობის დემონსტრირება:

```
delegate void Delegati(ref string striqoni1);
private void button2_Click(object sender, EventArgs e)
{
Delegati delegati_obieqti;
string striqoni = " მეთოდების მუშაობის შემოწმება";
label1.Text = striqoni;
// დელეგატს ანონიმური მეთოდი უკავშირდება
delegati_obieqti = delegate(ref string par)
{
string temp = "";
for ( int i = 0; i < striqoni.Length; i++ )
if ( striqoni[i] != ' ' ) temp += striqoni[i];
striqoni = temp;
};
// დელეგატის გამოძახება
delegati_obieqti(ref striqoni);
label2.Text = striqoni;
}
```

როგორც პროგრამიდან ჩანს, ანონიმურ მეთოდს ერთი სტრიქონული ტიპის პარამეტრი აქვს - par, რომელიც ამ მეთოდის კოდში არ გამოიყენება. ის მითითებულია იმიტომ, რომ ანონიმური მეთოდის პარამეტრები უნდა შეესაბამებოდეს დელეგატის პარამეტრებს.

დელეგატი შეიძლება შეიცავდეს როგორც ჩვეულებრივი მეთოდის მისამართს, ისე ანონიმურ მეთოდს. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
delegate void Delegati(ref string striqoni1);
class ChemiKlasi
{
public void Shebruneba(ref string striqoni2)
{
string temp = "";
int i;
for ( i = striqoni2.Length - 1; i >= 0; i-- )
```

```

        temp += striqoni2[i];
striqoni2 = temp;
}
}
private void button1_Click(object sender, EventArgs e)
{
ChemiKlasi obieqti = new ChemiKlasi();
Delegati delegati_obieqti;
Delegati Shebruneba_mimartva = new Delegati(obieqti.Shebruneba);
string striqoni = " მეთოდების მუშაობის შემოწმება";
label1.Text = striqoni;
// დელეგატს ერთი მიმართვა ენიჭება
delegati_obieqti = Shebruneba_mimartva;
// დელეგატს ანონიმური მეთოდი უკავშირდება
delegati_obieqti += delegate(ref string striqoni3)
{
string temp = "";
for ( int i = 0; i < striqoni.Length; i++ )
    if ( striqoni[i] != ' ' ) temp += striqoni[i];
striqoni = temp;
};
// დელეგატის გამოძახება
delegati_obieqti(ref striqoni);
label2.Text = striqoni;
}

```

მოვლენები

მოვლენა არის ავტომატური უწყება რაიმე მომხდარ მოქმედებაზე. მოვლენის საშუალებით ერთი ობიექტი მეორე ობიექტს ატყობინებს, რომ რაღაც მოხდა. მოვლენის მაგალითებია: კლავიატურის კლავიშის დაჭერა, კლავიატურის კლავიშის აშვება, თაგვის მარცხენა კლავიშის დაჭერა, თაგვის მარჯვენა კლავიშის დაჭერა, თაგვის კლავიშის აშვება და ა.შ.

ობიექტისთვის, რომელიც მისი განსაზღვრის (კოდის) მიხედვით უნდა რეაგირებდეს რაიმე მოვლენაზე, რეგისტრირდება ამ მოვლენის დამამუშავებელი (მეთოდი). მოვლენების დამამუშავებლები იქმნება დელეგატების საფუძველზე.

მოვლენები წარმოადგენენ კლასის წევრებს და მათი გამოცხადება ხდება event საკვანძო სიტყვის გამოყენებით. მოვლენების გამოცხადების სინტაქსია:

event დელეგატის სახელი მოვლენის სახელი;

აქ დელეგატის სახელი იმ დელეგატის სახელია, რომელიც გამოიყენება მოვლენის დასამუშავებლად, ხოლო მოვლენის სახელი არის შესაქმნელი მოვლენის (მოვლენა-ობიექტის) სახელი.

ქვემოთ მოყვანილი პროგრამით ხდება მოვლენასთან მუშაობის დემონსტრირება.

```

// პროგრამა 9.4
// პროგრამაში ხდება მოვლენასთან მუშაობის დემონსტრირება
// დელეგატის გამოცხადება, რომლის საფუძველზეც განსაზღვრული იქნება მოვლენა
delegate void ChemiDelegati();

```

```

// კლასის გამოცხადება, რომელშიც ხდება მოვლენის ინიცირება
class Klasi1
{
public event ChemiDelegati ChemiMovlena; // მოვლენის გამოცხადება
// ამ მეთოდში ინიცირდება მოვლენა
public void Metodi1()
{
// პირობის განსაზღვრა მოვლენის ინიციალიზაციისათვის
if ( ChemiMovlena != null ) ChemiMovlena();
}
}
class Klasi2
{
public void MovlenisDamamushavebeli()
{
MessageBox.Show("აღიძვრა მოვლენა");
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Klasi2 obieqti = new Klasi2();
Klasi1 obieqti_movlena = new Klasi1(); // მოვლენის ეგზემპლარის შექმნა
//
obieqti_movlena.ChemiMovlena += new ChemiDelegati(obieqti.MovlenisDamamushavebeli);
// მოვლენის ინიცირება
obieqti_movlena.Metodi1();
}

```

პროგრამა იწყება დელეგატის გამოცხადებით, რომელიც ინახავს მიმართვას მოვლენის დამამუშავებელზე:

```
delegate void ChemiDelegati();
```

მოვლენების ყველა დამამუშავებელი აქტიურდება დელეგატის საშუალებით. შედეგად, დელეგატი განსაზღვრავს მოვლენის ხელმოწერას. მოცემულ შემთხვევაში მოვლენას არ აქვს პარამეტრები, თუმცა მათი არსებობა დასაშვებია. რადგან, მოვლენა შეიძლება იყოს მრავალმისამართიანი, ამიტომ მოვლენის დამამუშავებლისათვის უნდა მიეთითოს დასაბრუნებელი მნიშვნელობის void ტიპი.

შემდეგ, პროგრამაში იქმნება Klasi1 კლასი, რომელშიც გამოცხადებულია ChemiMovlena მოვლენა. Metodi1() მეთოდი პროგრამის მიერ გამოძახებული იქნება მოვლენის ინიციალიზაციისათვის. პირობა, როცა სრულდება მოვლენის ინიციალიზება, განსაზღვრულია if ოპერატორით

```
if ( ChemiMovlena != null ) ChemiMovlena();
```

აქედან გამომდინარე, მოვლენის დამამუშავებელი მეთოდი მხოლოდ იმ შემთხვევაში გამოიძახება თუ ChemiMovlena მოვლენასთან ასოცირებულ დელეგატს აქვს null-საგან განსხვავებული მნიშვნელობა.

Klasi2 კლასის შიგნით იქმნება მოვლენების დამამუშავებელი MovlenisDamamushavebeli() მეთოდი. ამ მაგალითში მოვლენების დამამუშავებელს უბრალოდ გამოაქვს შეტყობინება. შემდეგ, პროგრამაში იქმნება Klasi1 ტიპის obieqti_movlena ობიექტი, ხოლო MovlenisDamamushavebeli() მეთოდი რეგისტრირდება როგორც ამ კლასში გამოცხადებული

მოვლენის დამამუშავებელი:

```
Klasi1 obieqti_movlena = new Klasi1();  
obieqti_movlena.ChemiMovlena += new ChemiDelegati(obieqti.MovlenisDamamushavebeli);  
უკანასკნელი გამოსახულება იმას ნიშნავს, რომ როცა აღიძვრება ChemiMovlena, მაშინ მის  
დასამუშავებლად გამოძახებული იქნება MovlenisDamamushavebeli() მეთოდი.
```

მოვლენის დამამუშავებლის დამატება ხდება += ოპერატორის გამოყენებით, წაშლა კი -= ოპერატორის გამოყენებით.

პროგრამის ბოლოს ხდება მოვლენის ინიცირება

```
obieqti_movlena.Metodi1();
```

Metodi1() მეთოდის გამოძახება იწვევს მოვლენის დამამუშავებლის გამოძახებას.

ახლა განვიხილოთ შემთხვევა, როცა პარამეტრი გადაეცემა როგორც მოვლენის აღმძვრელ, ისე მოვლენის დამამუშავებელ მეთოდს. მოვლენის აღმძვრელ მეთოდს გადავცეთ რიცხვი და თუ ის დადებითია, მაშინ აღვძრათ მოვლენა. შესაბამისი შეტყობინება Label კომპონენტში გამოვიტანოთ. მოყვანილი პროგრამით ხდება ამის დემონსტრირება:

```
delegate void ChemiDelegati(Label lab);
```

```
// კლასის გამოცხადება, რომელშიც ხდება მოვლენის ინიცირება
```

```
class Klasi1
```

```
{
```

```
public event ChemiDelegati ChemiMovlena; // მოვლენის გამოცხადება
```

```
// ამ მეთოდში ინიცირდება მოვლენა
```

```
public void Metodi1(int par1, Label lab)
```

```
{
```

```
// პირობის განსაზღვრა მოვლენის ინიცირებისათვის
```

```
if ( par1 >= 0 ) ChemiMovlena(lab);
```

```
}
```

```
public void MovlenisDamamushavebeli(Label lab)
```

```
{
```

```
lab.Text = "აღიძვრა მოვლენა, რიცხვი დადებითია";
```

```
}
```

```
}
```

```
//
```

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
int ricxvi = int.Parse(textBox1.Text);
```

```
Klasi1 obieqti_movlena = new Klasi1(); // მოვლენის ეგზემპლარის შექმნა
```

```
//
```

```
obieqti_movlena.ChemiMovlena +=
```

```
new ChemiDelegati(obieqti_movlena.MovlenisDamamushavebeli);
```

```
// მოვლენის ინიცირება
```

```
obieqti_movlena.Metodi1(ricxvi, label1);
```

```
}
```

მოცემულ შემთხვევაში გამოიძახება მხოლოდ ერთი დამამუშავებელი, თუმცა ისინი შეიძლება რამდენიმე იყოს.

ფართოსამაუწყებლო მოვლენა

როგორც ადრე აღვნიშნეთ მოვლენა იქმნება დელეგატების საფუძველზე. რადგანაც, დელეგატი შეიძლება იყოს მრავალმისამართიანი, ამიტომ მოვლენებს შეუძლიათ რამდენიმე

დამამუშავებლის გააქტიურება, რომელთა ნაწილი შეიძლება განსაზღვრული იყოს სხვა ობიექტებში. ასეთ მოვლენებს ფართოსამაუწყებლო ეწოდებათ. მათი გამოყენება ბევრ ობიექტს საშუალებას აძლევს რეაგირება მოახდინოს მოვლენის შესახებ უწყებაზე. ქვემოთ მოყვანილ პროგრამაში ხდება ფართოსამაუწყებლო მოვლენასთან მუშაობის დემონსტრირება.

```
//      პროგრამა 9.5
//      პროგრამაში ხდება ფართოსამაუწყებლო მოვლენასთან მუშაობის დემონსტრირება
delegate void ChemiDelegati();
//      კლასის გამოცხადება, რომელშიც ინიცირდება მოვლენა
class Klasi1
{
public event ChemiDelegati ChemiMovlena;
//      მოვლენის ინიციალიზება
public void Metodi1()
{
if ( ChemiMovlena != null ) ChemiMovlena();
}
}
class Damamushavebeli1
{
public void Movlenis_Damamushavebeli_2()
{
MessageBox.Show("მოვლენა მიღებულია Damamushavebeli1 კლასის ობიექტის მიერ");
}
}
class Damamushavebeli2
{
public void Movlenis_Damamushavebeli_3()
{
MessageBox.Show("მოვლენა მიღებულია Damamushavebeli2 კლასის ობიექტის მიერ");
}
}
class Klasi2
{
public void Movlenis_Damamushavebeli_1()
{
MessageBox.Show("მოვლენა მიღებულია Klasi2 კლასის ობიექტის მიერ");
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Klasi2 obieqti = new Klasi2();
Klasi1 obieqti_movlena = new Klasi1();
Damamushavebeli1 obieqti1 = new Damamushavebeli1();
Damamushavebeli2 obieqti2 = new Damamushavebeli2();
//      მოვლენების დამამუშავებლების მწკრივი: Movlenis_Damamushavebeli_1(),
//      Movlenis_Damamushavebeli_2() და Movlenis_Damamushavebeli_3()
//      მეთოდების დამატება
```

```

obieqti_movlena.ChemiMovlena += new ChemiDelegati(obieqti.Movlenis_Damamushavebeli_1);
obieqti_movlena.ChemiMovlena += new ChemiDelegati(obieqti1.Movlenis_Damamushavebeli_2);
obieqti_movlena.ChemiMovlena += new ChemiDelegati(obieqti2.Movlenis_Damamushavebeli_3);
// მოვლენის ინიცირება
obieqti_movlena.Metodi1();

// მწკრივიდან ერთი დამამუშავებლის წაშლა
obieqti_movlena.ChemiMovlena -= new ChemiDelegati(obieqti1.Movlenis_Damamushavebeli_2);
obieqti_movlena.Metodi1();
}

```

ამ პროგრამაში იქმნება ორი კლასი Damamushavebeli1 და Damamushavebeli2. მათში განსაზღვრულია მოვლენების დამამუშავებლები, რომლებიც თავიანთი ხელმოწერით თავსებადია ChemiDelegati ტიპთან. შედეგად, ეს დამამუშავებლები ხდებიან მწკრივის ნაწილი და გამოიძახებიან ცალ-ცალკე.

გენერირებული მოვლენები ცალ-ცალკე გადაეცემა თითოეულ ობიექტს. შედეგად, მოვლენის შესახებ უწყების მისაღებად წინასწარ უნდა იყოს დარეგისტრირებული (განსაზღვრული) ამ მოვლენის დამამუშავებელი თითოეული ობიექტისათვის. მაგალითად, ქვემოთ მოყვანილ პროგრამაში აღძრული მოვლენა (Metodi1() მეთოდის გამოძახება) ახდენს Klas1 კლასის ორი ობიექტის დამამუშავებლების ინიცირებას.

// პროგრამა 9.6

// პროგრამაში ხდება ორი ობიექტის დამამუშავებლების ინიცირება

```

delegate void ChemiDelegati();
//
class ChemiKlasi
{
public event ChemiDelegati ChemiMovlena;
//
public void Metodi1()
{
if ( ChemiMovlena != null ) ChemiMovlena();
}
}
class Klas1
{
int ricxvi;
public Klas1(int par)
{
ricxvi = par;
}
public void Damamushavebeli()
{
MessageBox.Show("მოვლენა მიღებულია ობიექტის მიერ, რომლის # = " + ricxvi.ToString());
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti_movlena = new ChemiKlasi();

```

```

Klasi1 obieqti1 = new Klasi1(1);
Klasi1 obieqti2 = new Klasi1(2);

obieqti_movlena.ChemiMovlena += new ChemiDelegati(obieqti1.Damamushavebeli);
obieqti_movlena.ChemiMovlena += new ChemiDelegati(obieqti2.Damamushavebeli);
// მოვლენის ინიცირება
obieqti_movlena.Metodi1();
}

```

როგორც ვხედავთ, თითოეული ობიექტისათვის ცალცალკე რეგისტრირდება მოვლენების დამამუშავებელი, და შესაბამისად, თითოეული ობიექტი იღებს ცალკე უწყებას მომხდარი მოვლენის შესახებ.

ინტერფეისები

C# ენის ერთ-ერთი მნიშვნელოვანი ელემენტია ინტერფეისი. ის განსაზღვრავს მეთოდების, თვისებების, ინდექსატორებისა და მოვლენების ნაკრებს, რომელთა რეალიზება შესაძლებელია კლასის საშუალებით. პროგრამირების დროს ზოგჯერ წამოიჭრება კლასის მიერ შესრულებული მოქმედებების განსაზღვრის აუცილებლობა, მათი რეალიზების გზის მითითების გარეშე. ჩვენ ადრე განვიხილეთ მსგავსი მაგალითი - აბსტრაქტული მეთოდი. ის განსაზღვრავს მეთოდის ინტერფეისს, და არა რეალიზებას. C# ენაში შეიძლება მთლიანად განვაცალკევოთ კლასის ინტერფეისი ამ კლასის რეალიზაციისაგან. ინტერფეისის განსაზღვრის შემდეგ მისი რეალიზება შესაძლებელია ერთი ან მეტი კლასის საშუალებით. ამასთან, ასეთი კლასი დამატებით შეიძლება შეიცავდეს საკუთარ ელემენტებს.

ინტერფეისის გამოცხადების სინტაქსია:

```

interface ინტერფეისის_სახელი
{
ინტერფეისის ტანი
}

```

ინტერფეისის ტანი შეიძლება შეიცავდეს მეთოდებს, თვისებებს, ინდექსატორებსა და მოვლენებს. ინტერფეისი არ შეიძლება შეიცავდეს ცვლადებს (ველებს), სტატიკურ წევრებს, კონსტრუქტორებს, დესტრუქტორებსა და ოპერატორულ მეთოდებს. ინტერფეისში შეიძლება გამოცხადებული იყოს მეთოდების, თვისებების, ინდექსატორებისა და მოვლენების სიგნატურები. მეთოდისათვის დამატებით ეთითება დასაბრუნებელი მნიშვნელობის ტიპი. გარდა ამისა, მეთოდის სიგნატურა ზუსტად უნდა შეესაბამებოდეს interface განსაზღვრაში მითითებულ სიგნატურას.

ინტერფეისების რეალიზება

როგორც კი ინტერფეისი განისაზღვრება, ის შეიძლება რეალიზებული იყოს ერთი ან მეტი კლასის მიერ. ინტერფეისის რეალიზაციისათვის კლასის სახელის შემდეგ უნდა მივუთითოთ ორი წერტილი და ინტერფეისის სახელი.

თუ კლასი ახდენს ინტერფეისის რეალიზებას, ის ვალდებულია მისი რეალიზება მოახდინოს მთლიანად.

კლასებს, რომლებიც ახდენს ინტერფეისების რეალიზებას, შეუძლიათ განსაზღვრონ, აგრეთვე, დამატებითი წევრები. ქვემოთ მოყვანილი პროგრამით ხდება ინტერფეისის რეალიზება, რომელშიც ორი მეთოდია გამოცხადებული.

```

// პროგრამა 9.7

```



```

//      პროგრამაში რეალიზებულია ინტერფეისი, რომელშიც გამოცხადებულია მეთოდები
public interface ChemiInterpeisi
{
    int Perimetri();
    double Fartobi();
}
class Samkutxedi : ChemiInterpeisi
{
    int gverdi1;
    int gverdi2;
    int gverdi3;
    public Samkutxedi(int par1, int par2, int par3)
    {
        gverdi1 = par1;
        gverdi2 = par2;
        gverdi3 = par3;
    }
    public int Perimetri()
    {
        return gverdi1 + gverdi2 + gverdi3;
    }
    public double Fartobi()
    {
        return ( gverdi1 * gverdi2 ) / 2;
    }
    public void Naxva(Label lab1)
    {
        lab1.Text = gverdi1.ToString() + " " + gverdi2.ToString() + " " + gverdi3.ToString();
    }
}

private void button1_Click(object sender, EventArgs e)
{
    int gverdi1 = int.Parse(textBox1.Text);
    int gverdi2 = int.Parse(textBox2.Text);
    int gverdi3 = int.Parse(textBox3.Text);
    Samkutxedi Obieqti = new Samkutxedi(gverdi1, gverdi2, gverdi3);

    int perimetri = Obieqti.Perimetri();
    double fartobi = Obieqti.Fartobi();
    Obieqti.Naxva(label1);
    label2.Text = "სამკუთხედის პერიმეტრი = " + perimetri.ToString();
    label3.Text = "სამკუთხედის ფართობი = " + fartobi.ToString();
}

```

ერთ კლასს შეუძლია რამდენიმე ინტერფეისის რეალიზება. ამ შემთხვევაში, კლასში უნდა შესრულდეს ყველა ინტერფეისის წევრების რეალიზება. მოყვანილი პროგრამით ხდება ერთი კლასის მიერ ორი ინტერფეისის რეალიზება.

```

//      პროგრამა 9.8
//      კლასი ახდენს ორი ინტერფეისის რეალიზებას
public interface Interpeisi1
{
    int Perimetri();
}
public interface Interpeisi2
{
    double Fartobi();
}
class Samkutxedi : Interpeisi1, Interpeisi2
{
    int gverdi1;
    int gverdi2;
    int gverdi3;
    public Samkutxedi(int par1, int par2, int par3)
    {
        gverdi1 = par1;
        gverdi2 = par2;
        gverdi3 = par3;
    }
    public int Perimetri()
    {
        return gverdi1 + gverdi2 + gverdi3;
    }
    public double Fartobi()
    {
        return ( gverdi1 * gverdi2 ) / 2;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    int gverdi1 = int.Parse(textBox1.Text);
    int gverdi2 = int.Parse(textBox2.Text);
    int gverdi3 = int.Parse(textBox3.Text);
    Samkutxedi obieqti = new Samkutxedi(gverdi1, gverdi2, gverdi3);
    int Perimetri = obieqti.Perimetri();
    double Fartobi = obieqti.Fartobi();
    label1.Text = "სამკუთხედის პერიმეტრია " + Perimetri.ToString();
    label2.Text = "სამკუთხედის ფართობია " + Fartobi.ToString();
}

```

კლასების მემკვიდრეობითობა და ინტერფეისის რეალიზება

კლასებს შეუძლიათ ერთზე მეტი ინტერფეისის რეალიზება. ამ შემთხვევაში, ინტერფეისების სახელები ერთმანეთისაგან მძიმეებით უნდა გამოიყოს. კლასი შეიძლება იყოს, აგრეთვე, საბაზო კლასის მემკვიდრე და ახდენდეს ერთი ან მეტი ინტერფეისის რეალიზებას. ამ შემთხვევაში სიაში, წინაპარი კლასის სახელი უნდა იყოს პირველი. მოყვანილი პროგრამით

ხდება ყოველივე ამის დემონსტრირება.

```
//   პროგრამა 9.9
//   პროგრამაში ხდება ინტერფეისის რეალიზება მემკვიდრეობითობის შემთხვევაში
//   Interpeisi_A ინტერფეისის გამოცხადება
//   Interpeisi_A ინტერფეისის გამოცხადება
public interface Interpeisi_A
{
    //   მეთოდების გამოცხადება
    int Perimetri();
}
//   Interpeisi_B ინტერფეისის გამოცხადება
public interface Interpeisi_B
{
    //   მეთოდების გამოცხადება
    double Fartobi();
}
public class ChemiKlasi
{
    public int gverdi1;
    public int gverdi2;
    public int gverdi3;
    public ChemiKlasi(int par1, int par2, int par3)
    {
        gverdi1 = par1;
        gverdi2 = par2;
        gverdi3 = par3;
    }
}
public class Samkutxedi : ChemiKlasi, Interpeisi_A, Interpeisi_B
{
    //   საბაზო კლასის კონსტრუქტორის გამოძახება
    public Samkutxedi(int par4, int par5, int par6) : base(par4, par5, par6)
    {
        //   ცარიელი ტანი
    }
    //   Interpeisi_A ინტერფეისის Perimetri() მეთოდის რეალიზება
    public int Perimetri()
    {
        return gverdi1 + gverdi2 + gverdi3;
    }
    //   Interpeisi_B ინტერფეისის Fartobi() მეთოდის რეალიზება
    public double Fartobi()
    {
        return ( gverdi1 * gverdi2 ) / 2;
    }
}
```

```

private void button1_Click(object sender, EventArgs e)
{
    int gverdi1 = int.Parse(textBox1.Text);
    int gverdi2 = int.Parse(textBox2.Text);
    int gverdi3 = int.Parse(textBox3.Text);
    Samkutxedi obj = new Samkutxedi(gverdi1, gverdi2, gverdi3);
    int perimetri = obj.Perimetri();
    double fartobi = obj.Fartobi();
    label1.Text = "სამკუთხედის პერიმეტრი = " + perimetri.ToString();
    label2.Text = "სამკუთხედის ფართობი = " + fartobi.ToString();
}

```

წარმოებული ინტერფეისები

კლასების მსგავსად, ინტერფეისებიც შეიძლება იყოს წარმოებული (მიღებული) მემკვიდრეობით. კლასებისაგან განსხვავებით, ინტერფეისი შეიძლება მიღებული იყოს ერთზე მეტი ინტერფეისისაგან. განვიხილოთ ორი მაგალითი. პირველ მაგალითში Interfeisi_B ინტერფეისი მიღებულია Interfeisi_A ინტერფეისისაგან, მეორე მაგალითში კი Interfeisi_C ინტერფეისი მიღებულია Interfeisi_A და Interfeisi_B ინტერფეისებისაგან.

// პროგრამა 9.10

// პროგრამაში Interfeisi_B ინტერფეისი მიიღება Interfeisi_A ინტერფეისისაგან

```

public interface Interfeisi_A
{
    int Perimetri();
}
public interface Interfeisi_B : Interfeisi_A
{
    double Fartobi();
}
class Samkutxedi : Interfeisi_B
{
    private int gverdi1;
    private int gverdi2;
    private int gverdi3;
    public Samkutxedi(int par1, int par2, int par3)
    {
        gverdi1 = par1;
        gverdi2 = par2;
        gverdi3 = par3;
    }
    public int Perimetri()
    {
        return gverdi1 + gverdi2 + gverdi3;
    }
    public double Fartobi()
    {
        return ( gverdi1 + gverdi2 ) / 2;
    }
}

```

```

public void Naxva(Label lab1)
{
    lab1.Text = gverdi1.ToString() + " " + gverdi2.ToString() + " " + gverdi3.ToString();
}
}

```

```

private void button1_Click(object sender, EventArgs e)
{
    int gverdi1 = int.Parse(textBox1.Text);
    int gverdi2 = int.Parse(textBox2.Text);
    int gverdi3 = int.Parse(textBox3.Text);
    Samkutxedi obieqti = new Samkutxedi(gverdi1, gverdi2, gverdi3);
    int perimetri = obieqti.Perimetri();
    double fartobi = obieqti.Fartobi();
    obieqti.Naxva(label3);
    label1.Text = perimetri.ToString();
    label2.Text = fartobi.ToString();
}

```

მეორე მაგალითი.

// **პროგრამა 9.11**

// **პროგრამაში Interfeisi_C ინტერფეისი მიიღება Interfeisi_A და Interfeisi_B**

// **ინტერფეისებისაგან**

```

public interface Interfeisi_A
{
    int Perimetri();
}
public interface Interfeisi_B
{
    double Fartobi();
}
public interface Interfeisi_C : Interfeisi_A, Interfeisi_B
{
    void Naxva(Label lab1);
}
class Samkutxedi : Interfeisi_C
{
    private int gverdi1;
    private int gverdi2;
    private int gverdi3;
    public Samkutxedi(int par1, int par2, int par3)
    {
        gverdi1 = par1;
        gverdi2 = par2;
        gverdi3 = par3;
    }
    public int Perimetri()
    {

```

```

    return gverdi1 + gverdi2 + gverdi3;
}
public double Fartobi()
{
    return ( gverdi1 * gverdi2 ) / 2;
}
public void Naxva(Label lab1)
{
    lab1.Text = gverdi1.ToString() + " " + gverdi2.ToString() + " " + gverdi3.ToString();
}
}

```

```

private void button1_Click(object sender, EventArgs e)
{
    int gverdi1 = int.Parse(textBox1.Text);
    int gverdi2 = int.Parse(textBox2.Text);
    int gverdi3 = int.Parse(textBox3.Text);
    Samkutxedi obieqti = new Samkutxedi(gverdi1, gverdi2, gverdi3);
    obieqti.Naxva(label3);
    int shedegi1 = obieqti.Perimetri();
    label1.Text = shedegi1.ToString();
    double shedegi2 = obieqti.Fartobi();
    label2.Text = shedegi2.ToString();
}

```

ინტერფეისული მიმართვები

ჩვენ შეგვიძლია გამოვაცხადოთ მიმართვითი ცვლადი, რომელსაც აქვს ინტერფეისული ტიპი, ანუ შეგვიძლია შევქმნათ ინტერფეისული მიმართვითი ცვლადი. ასეთი ცვლადი შეიძლება მიმართავდეს ნებისმიერ ობიექტს, რომელიც ახდენს მისი ინტერფეისის რეალიზებას. ინტერფეისული მიმართვის საშუალებით ობიექტის მეთოდის გამოძახებისას გამოიძახება ის მეთოდი, რომლის რეალიზებას მოცემული ობიექტი ახდენს.

მოყვანილ პროგრამაში ნაჩვენებია ინტერფეისული მიმართვის გამოყენება. აქ გამოცხადებულია obieqti ინტერფეისული ცვლადი Samkutxedi და Otxkutxedi ობიექტების მეთოდების გამოსაძახებლად.

// **პროგრამა 9.12**

```

public interface ChemiInterpeisi
{
    int Perimetri();
    double Fartobi();
}
class Samkutxedi : ChemiInterpeisi
{
    int gverdi1;
    int gverdi2;
    int gverdi3;
    public Samkutxedi(int par1, int par2, int par3)
    {

```

```

gverdi1 = par1;
gverdi2 = par2;
gverdi3 = par3;
}
public int Perimetri()
{
return gverdi1 + gverdi2 + gverdi3;
}
public double Fartobi()
{
return ( gverdi1 * gverdi2 ) / 2;
}
}
class Otxkutxedi : ChemiInterpeisi
{
int gverdi1;
int gverdi2;
public Otxkutxedi(int par1, int par2)
{
gverdi1 = par1;
gverdi2 = par2;
}
public int Perimetri()
{
return ( gverdi1 + gverdi2 ) * 2;
}
public double Fartobi()
{
return gverdi1 * gverdi2;
}
}

```

```

private void button1_Click(object sender, EventArgs e)
{
int gverdi1 = int.Parse(textBox1.Text);
int gverdi2 = int.Parse(textBox2.Text);
int gverdi3 = int.Parse(textBox3.Text);
Samkutxedi Samkutxedi = new Samkutxedi(gverdi1, gverdi2, gverdi3);
Otxkutxedi Otxkutxedi = new Otxkutxedi(gverdi1, gverdi2);
ChemiInterpeisi obieqti;

obieqti = Samkutxedi;
label1.Text = "სამკუთხედის პერიმეტრი = " + obieqti.Perimetri().ToString() +
"\nსამკუთხედის ფართობი = " + obieqti.Fartobi();

obieqti = Otxkutxedi;
label2.Text = "ოთხკუთხედის პერიმეტრი = " + obieqti.Perimetri().ToString() +

```

```

        "\u0540\u0561\u0562\u0563\u0564\u0565 \u0566\u0567\u0568 = " + obieqti.Fartobi());
    }

```

პროგრამაში obieqti ობიექტი გამოცხადებულია როგორც მიმართვა ChemiInterpeisi ინტერფეისზე. ეს იმას ნიშნავს, რომ ის შეიძლება გამოყენებულ იქნას ნებისმიერ იმ ობიექტზე მიმართვების შენახვის მიზნით, რომელიც ახდენს ChemiInterpeisi ინტერფეისის რეალიზებას. მოცემულ შემთხვევაში ის გამოიყენება Samkutxedi და Otxkutxedi ობიექტებზე მიმართვის შენახვისათვის, რომლებიც წარმოადგენენ Samkutxedi და Otxkutxedi ტიპის ობიექტებს შესაბამისად. ორივე ეს ობიექტი ახდენს ChemiInterpeisi ინტერფეისის რეალიზებას. obieqti ინტერფეისულმა მიმართვამ იცის მხოლოდ იმ მეთოდების შესახებ, რომლებიც გამოცხადებულია ინტერფეისში.

ინტერფეისის თვისებები

ისევე, როგორც მეთოდების შემთხვევაში, ინტერფეისში თვისებების განსაზღვრისას არ ეთითება ტანი. ინტერფეისის თვისების სინტაქსია:

ტიპი თვისების_სახელი

```

{
get;
set;
}

```

ქვემოთ მოყვანილია ChemiInterpeisi ინტერფეისი, აგრეთვე, ChemiKlasi კლასის კოდი, რომელიც თვისებას იყენებს ნაკრების მომდევნო ელემენტის გაცემისა და შეცვლისათვის.

// პროგრამა 9.13

// პროგრამაში ხდება ინტერფეისის თვისებასთან მუშაობის დემონსტრირება

```

public interface ChemiInterpeisi
{
// ინტერფეისის თვისება
int shemdegi
{
get; // ნაკრების მომდევნო რიცხვის მიღება
set; // მომდევნო რიცხვის დაყენება
}
}
class ChemiKlasi : ChemiInterpeisi
{
int cvladi;
public ChemiKlasi()
{
cvladi = 0;
}
// მნიშვნელობის მიღება და დაყენება
public int shemdegi
{
get
{
cvladi += 5;
return cvladi;
}
}
}

```



```

}
set
{
    cvladi = value;
}
}
}
private void button1_Click(object sender, System.EventArgs e)
{
    // ინტერფეისის თვისებების დემონსტრირება
    ChemiKlasi obieqti = new ChemiKlasi();

    for ( int i = 0; i < 5; i++ )
        label1.Text += "მომდევნო მნიშვნელობა - " +
            obieqti.shemdegi.ToString() + '\n';
    obieqti .shemdegi = 30;
    for ( int i = 0; i < 5; i++ )
        label2.Text += " მომდევნო მნიშვნელობა - " + obieqti.shemdegi.ToString() + '\n';
}

```

ცხადი რეალიზება

ცხადი რეალიზება არის შემთხვევა, როცა რეალიზაციის დროს ინტერფეისის ელემენტის წინ ეთითება ინტერფეისის სახელი. კლასს შეუძლია ორი ინტერფეისის რეალიზება, რომლებშიც შეიძლება გამოცხადებული იყოს ერთნაირი სახელის ელემენტები. წამოიჭრება პრობლემა - როგორ განვასხვავოთ ასეთი ელემენტები. ეს პრობლემა წყდება ცხადი რეალიზაციის გამოყენებით.

ქვემოთ მოყვანილ პროგრამაში გამოცხადებულია ორი ინტერფეისი, რომლებიც შეიცავენ metodi() მეთოდს. იმისათვის, რომ განვასხვავოთ ერთნაირი სახელის მქონე ეს მეთოდები უნდა გამოვიყენოთ ცხადი რეალიზება:

```

// პროგრამა 9.14
// პროგრამაში ხდება ცხადი რეალიზების გამოყენების დემონსტრირება
interface Interpeisi_A
{
    int Metodi(int x);
}
interface Interpeisi_B
{
    int Metodi(int x);
}
// ChemiKlasi კლასში ხდება ორივე ინტერფეისის რეალიზება
class ChemiKlasi : Interpeisi_A, Interpeisi_B
{
    Interpeisi_A a_obieqti;
    Interpeisi_B b_obieqti;
    // ორივე metodi() მეთოდის აშკარა რეალიზება
    int Interpeisi_A.metodi(int x)
    {

```

```

return x + x;
}
int Interpeisi_B.metodi(int x)
{
return x * x;
}
//      metodi() მეთოდის გამოძახება ინტერფეისული მიმართვის საშუალებით
public int Metodi_A(int x)
{
a_obieqti = this;           //      a_obieqti ობიექტს ენიჭება გამომძახებელ ობიექტზე მიმართვა
return a_obieqti.Metodi(x); //      Interpeisi_A მეთოდის გამოძახება
}
public int Metodi_B(int x)
{
b_obieqti = this;           //      b_obieqti ობიექტს ენიჭება გამომძახებელ ობიექტზე მიმართვა
return b_obieqti.Metodi(x); //      Interpeisi_B მეთოდის გამოძახება
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      აშკარა რეალიზაციის გამოყენება არაერთგვაროვნობის აღმოსაფხვრელად
ChemiKlasi obieqti = new ChemiKlasi();
int ricxvi = int.Parse(textBox1.Text);

//      Interpeisi_A.Metodi() მეთოდის გამოძახება
label1.Text = obieqti.Metodi_A(ricxvi).ToString();
//      Interpeisi_B.Metodi() მეთოდის გამოძახება
label2.Text = obieqti.Metodi_B(ricxvi).ToString();
}

```

ყურადღება მივაქციოთ იმას, რომ Metodi() მეთოდს აქვს ერთი და იგივე სიგნატურა Interpeisi_A და Interpeisi_B ინტერფეისებში. ამიტომ, იმ შემთხვევაში თუ ChemiKlasi ახდენს ორივე ამ ინტერფეისის რეალიზებას, მაშინ მან აშკარად უნდა მოახდინოს თითოეული მათგანის ცალ-ცალკე რეალიზება, მათი სახელების სრული განსაზღვრით. რადგან ერთადერთი საშუალება, რომლის საშუალებითაც შეიძლება გამოვიძახოთ აშკარად რეალიზებული მეთოდი არის ინტერფეისული მიმართვა, ამიტომ ChemiKlasi კლასში გამოცხადებულია ორი მიმართვა, რომელთაგან ერთი განკუთვნილია Interpeisi_A, მეორე კი - Interpeisi_B ინტერფეისისათვის. შემდეგ სრულდება კლასის ორი მეთოდი გამოიძახება ინტერფეისის მეთოდების საშუალებით. რადგან, Metodi() მეთოდი ცხადად არის რეალიზებული, ამიტომ ის მიუწვდომელია ChemiKlasi კლასის გარეთ. ამ კლასის შიგნით Metodi() მეთოდთან მიმართვა შესაძლებელია მხოლოდ ინტერფეისული a_obieqti და b_obieqti ობიექტების საშუალებით.

ინტერფეისის წევრების დამალვა

კლასების ანალოგიურად, მემკვიდრეობითობის დროს, ინტერფეისების შემთხვევაშიც ადგილი აქვს წევრების დამალვას. დავუშვათ, Interfeisi_A ინტერფეისი არის საბაზო და შეიცავს Minicheba() მეთოდს, ხოლო Interfeisi_B ინტერფეისი არის მისგან მიღებული და შეიცავს Minicheba() მეთოდს. თუ გამოვაცხადებთ Klasi1 კლასს, რომელიც მოახდენს Interfeisi_B ინტერფეისის რეალიზებას, მაშინ ამ კლასში ერთ-ერთი Minicheba() მეთოდი უნდა

გამოვაცხადოთ Interfeisi_A ან Interfeisi_B ინტერფეისის სახელის ცხადი მითითების გზით. მოყვანილი პროგრამით ხდება ამის დემონსტრირება.

```
// პროგრამა 9.15
// პროგრამაში ხდება ინტერფეისის წევრების დამალვის დემონსტრირება
public interface Interfeisi_A
{
int Minicheba(int x);
}
public interface Interfeisi_B : Interfeisi_A
{
new int Minicheba(int x);
}
class Klasi1 : Interfeisi_B
{
private int ricxvi;
public int Minicheba(int par1)
{
ricxvi = par1;
return ricxvi * ricxvi;
}
int Interfeisi_B.Minicheba(int par1)
{
ricxvi = par1;
return ricxvi + ricxvi;
}
}
private void button1_Click(object sender, EventArgs e)
{
int cvladi = int.Parse(textBox1.Text);
Klasi1 obieqti = new Klasi1();
Interfeisi_B interfeisi_obj = ( Interfeisi_B ) obieqti;

int shedegi1 = obieqti.Minicheba(cvladi);
label1.Text = shedegi1.ToString();
int shedegi2 = interfeisi_obj.Minicheba(cvladi);
label2.Text = shedegi2.ToString();
}
```

სახელების სივრცე

სახელების სივრცე თავიდან გვაცხადებს პროგრამის კლასების სახელებს შორის კონფლიქტს. დავუშვათ, რომ ჩვენი პროგრამა იყენებს სხვა პროგრამისტის მიერ შემუშავებულ კლასს, რომლის სახელია Manqana. დავუშვათ, ასევე, რომ ჩვენც დაგვჭირდა ამავე სახელის მქონე კლასის განსაზღვრა. ასეთ შემთხვევაში, მოგვიწევს სხვა სახელის გამოყენება. სწორედ ასეთი პრობლემების გადასაწყვეტად გამოიყენება სახელების სივრცე.

სახელების სივრცე განსაზღვრავს გამოცხადების უბანს, რომელიც საშუალებას გვაძლევს

შევინახოთ სახელების თითოეული ნაკრები სხვა ნაკრებებისაგან ცალკე. სახელების ერთ სივრცეში გამოცხადებული სახელები არ არიან კონფლიქტში სახელების სხვა სივრცეში გამოცხადებულ ასეთივე სახელებთან. System სახელების სივრცეში მრავალი კლასია განსაზღვრული, ამიტომ თითოეული პროგრამა იწყება დირექტივით: using System;

სახელების სივრცის გამოცხადება

სახელების სივრცის გამოცხადება ხდება namespace საკვანძო სიტყვის საშუალებით. მისი სინტაქსია:

```
namespace სახელების_სივრცის_სახელი
{
    სახელების სივრცის წევრები
}
```

სახელების სივრცეში განსაზღვრული ყველა ელემენტი, როგორცაა კლასები, დელეგატები, ჩამოთვლები, ინტერფეისები ან სხვა სახელების სივრცეები, იმყოფება ამ სახელების სივრცის მხედველობის უბნის საზღვრებში.

ქვემოთ მოყვანილ პროგრამაში ხდება სახელების სივრცესთან მუშაობის დემონსტრირება. Bmw და Opel სახელების სივრცე უნდა მივუთითოთ ცალკე ფაილში, როგორც ამას კლასებისთვის ვაკეთებთ. ამისათვის, ჯერ შევასრულოთ Project მენიუს Add New Item ბრძანება. შემდეგ, მოვნიშნოთ Visual C# Items განყოფილების Code File ელემენტი, Name ველში შევიტანოთ ამ ფაილის სახელი, მაგალითად Sicrceebi.cs, დავაჭიროთ Add კლავიშს და დავიწყოთ სახელების სივრცის შეტანა.

```
// პროგრამა 9.16
// პროგრამაში ხდება სახელების სივრცესთან მუშაობის დემონსტრირება
// Bmw სახელების სივრცის გამოცხადება
namespace Bmw
{
    public class Manqana
    {
        public string ManqanisMarka;
    }
}
// Opel სახელების სივრცის გამოცხადება
namespace Opel
{
    public class Manqana
    {
        public string ManqanisMarka;
    }
}
private void button1_Click(object sender, System.EventArgs e)
{
    Bmw.Manqana ChemiManqana1 = new Bmw.Manqana();
    ChemiManqana1.ManqanisMarka = "BMW";
    label1.Text = ChemiManqana1.ManqanisMarka;

    Opel.Manqana ChemiManqana2 = new Opel.Manqana();
    ChemiManqana2.ManqanisMarka = "Opel";
}
```

```
label2.Text = ChemiManqana2.ManqanisMarka;
}
```

პროგრამაში განსაზღვრულია ორი კლასი, რომლებსაც ერთნაირი სახელები აქვთ - Manqana. იმისათვის, რომ ეს ორი კლასი ერთმანეთისაგან განვასხვავოთ საჭიროა კლასის სახელის წინ სახელების სივრცის მითითება. მაგალითად, მოყვანილ სტრიქონში ხდება იმ Manqana კლასის ობიექტის შექმნა, რომელიც BMW სახელების სივრცეშია გამოცხადებული:

```
Bmw.Manqana ChemiManqana1 = new Bmw.Manqana();
```

ობიექტის შექმნის შემდეგ არ არის აუცილებელი სახელების სივრცის მითითება ობიექტის სახელის ან მისი წევრის წინ, მაგალითად,
 ChemiManqana1.ManqanisMarka = "BMW";

using დირექტივა

ძალზე დამღვლია ყოველთვის მივუთითოთ სახელების სივრცე, როცა პროგრამაში ხდება ხშირი მიმართვები ამ სახელების სივრცის წევრებთან. ასეთ შემთხვევებში, სასურველია using დირექტივის გამოყენება. ჩვენს მიერ განხილული ყველა პროგრამისათვის System სახელების სივრცე ხილული ხდება using დირექტივაში მისი მითითების შემდეგ. using დირექტივა შეგვიძლია, აგრეთვე, გამოვიყენოთ იმისათვის, რომ ხილული გავხადოთ ჩვენს მიერ შექმნილი სახელების სივრცე.

using დირექტივის სინტაქსია:

using სახელების_სივრცის_სახელი

ყველა წევრი, განსაზღვრული მითითებული სახელების სივრცის ფარგლებში, ხდება მისაწვდომი, ე.ი. ხდება მიმდინარე სახელების სივრცის ნაწილი, და შეიძლება გამოყენებული იყოს ადგილმდებარეობის სრული მითითების გარეშე. using დირექტივა მითითებული უნდა იყოს პროგრამის საწყისი კოდის შემცველი ფაილის დასაწყისში, ყველა სხვა გამოცხადების წინ.

ქვემოთ მოყვანილია პროგრამა, რომელშიც using დირექტივა გამოიყენება იმისთვის, რომ System და Opel სახელების სივრცე გავხადოთ ხილული.

// პროგრამა 9.17

// პროგრამაში ხდება using დირექტივასთან მუშაობის დემონსტრირება

```
using System;
```

```
using Opel;
```

```
// Opel სახელების სივრცის გამოცხადება
```

```
namespace Opel
```

```
{
```

```
public class Manqana
```

```
{
```

```
public string ManqanisMarka;
```

```
}
```

```
}
```

```
private void button1_Click(object sender, System.EventArgs e)
```

```
{
```

```
// Opel სიტყვის მითითება არ არის აუცილებელი ChemiManqana2 ობიექტის შექმნისას
```

```
Manqana ChemiManqana = new Manqana();
```

```
ChemiManqana.ManqanisMarka = "Opel";
```

```
label1.Text = ChemiManqana.ManqanisMarka;
```

```
}
```

ყურადღება მივაქციოთ იმას, რომ პროგრამაში ერთი სახელების სივრცე არ მაღავეს მეორე სახელების სივრცეს. სახელების სივრცის გამოცხადებისას ის თავის სახელებს უმატებს მოცემულ

მომენტში სხვა მისაწვდომ სახელებს. ამრიგად, ორივე სახელების სივრცე System და Opel, ხდება ხილული ამ პროგრამის კოდისათვის. გარდა ამისა, თუ სახელების ორ სივრცეში გამოცხადებულია ერთი და იგივე სახელის მქონე კლასები, მაშინ ამ კლასებთან მიმართვისას აუცილებლად უნდა მივუთითოთ სახელების სივრცის სახელი. წინააღმდეგ შემთხვევაში, კომპილატორი ვერ გაარკვევს თუ რომელ კლასთან უნდა შესრულდეს მიმართვა და გაიცემა შეტყობინება შეცდომის შესახებ.

ჩადგმული სახელების სივრცე

ერთი სახელების სივრცე შეიძლება მოვათავსოთ მეორის შიგნით. ასეთი გზით შეგვიძლია შევექმნათ სახელების სივრცის იერარქია. სახელების სივრცის იერარქია შეიძლება გადანაწილებული იყოს სხვადასხვა პროგრამაში (სხვადასხვა ფაილში). ასეთი გადანაწილების უპირატესობა იმაშია, რომ პროგრამისტებს შეუძლიათ ცალცალკე იმუშაონ ამ პროგრამებზე და შემდეგ ერთდროულად შეასრულონ მათი კომპილაცია.

ქვემოთ მოყვანილია ორი პროგრამა, რომლებშიც გადანაწილებულია Saxelebis_Sivrce სახელების სივრცე.

```
// პროგრამა 9.18
// პროგრამაში ხდება ჩადგმულ სახელების სივრცესთან მუშაობის დემონსტრირება
namespace Saxelebis_Sivrce
{
// Chadgmuli_Sivrce1 სახელების სივრცე არის ჩადგმული
namespace Chadgmuli_Sivrce1
{
public class ChemiKlasi
{
public string Metodi()
{
return " Chadgmuli_Sivrce1 ";
}
}
}
namespace Saxelebis_Sivrce.Chadgmuli_Sivrce2
{
public class ChemiKlasi
{
public string Metodi()
{
return " Chadgmuli_Sivrce2 ";
}
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Saxelebis_Sivrce.Chadgmuli_Sivrce1.ChemiKlasi obieqti1 =
new Saxelebis_Sivrce.Chadgmuli_Sivrce1.ChemiKlasi();
Saxelebis_Sivrce.Chadgmuli_Sivrce2.ChemiKlasi obieqti2 =
new Saxelebis_Sivrce.Chadgmuli_Sivrce2.ChemiKlasi();
```

```
label1.Text = obieqti1.Metodi();  
label2.Text = obieqti2.Metodi();  
}
```

პროგრამაში, Saxelebis_Sivrce სახელების სივრცეში ჩადგმულია Chadgmuli_Sivrce1 და Chadgmuli_Sivrce2 სახელების სივრცეები. სახელების ჩადგმული სივრცე შეგვიძლია, აგრეთვე გამოვაცხადოთ ერთ სტრიქონში. ასეა გამოცხადებული Chadgmuli_Sivrce2 სახელების სივრცე.

ChemiKlasi კლასთან მიმართვისას სრულად უნდა მივუთითოთ მისი ადგილმდებარეობა. ამისათვის, ჯერ უნდა მივუთითოთ ზედა დონის სახელების სივრცის სახელი, შემდეგ წერტილი და ბოლოს, ქვედა დონის სახელების სივრცის სახელი, მაგალითად, Saxelebis_Sivrce.Chadgmuli_Sivrce1.ChemiKlasi.

თუ გვინდა, რომ Chadgmuli_Sivrce1 სახელების სივრცეში გამოცხადებული ChemiKlasi კლასი მისაწვდომი გახდეს მისი ადგილმდებარეობის სრული მითითების გარეშე, using დირექტივა უნდა ჩავწეროთ შემდეგნაირად: using Saxelebis_Sivrce.Chadgmuli_Sivrce1; თუ using დირექტივას ასე ჩავწეროთ: using Saxelebis_Sivrce; , მაშინ Chadgmuli_Sivrce1 სახელების სივრცეში გამოცხადებული ChemiKlasi კლასი არ იქნება მისაწვდომი მისი ადგილმდებარეობის სრული მითითების გარეშე.

ავტომატურად განსაზღვრული სახელების სივრცე

თუ პროგრამაში არ არის გამოცხადებული სახელების სივრცე, მაშინ გამოიყენება ავტომატურად განსაზღვრული სახელების სივრცე. სახელების ასეთი სივრცე მოხერხებულია გამოვიყენოთ მოკლე და მარტივი პროგრამებისთვის. დიდი გამოყენებითი პროგრამების კოდი უნდა შედიოდეს სახელების რომელიმე სივრცეში. კოდი უნდა იყოს ინკაფსულირებული სახელების სივრცეში სახელების კონფლიქტების თავიდან აცილების მიზნით.

თავი 10. ინფორმაციის შეტანა-გამოტანა

შეტანა-გამოტანის ნაკადების კლასები

შეტანა-გამოტანა .NET გარემოში ეფუძნება ორ ერთმანეთთან დაკავშირებულ ცნებას - ნაკადებსა და მონაცემების სათავსოს. ნაკადი არის მონაცემების მიმდევრობა. მონაცემების სათავსო არის ადგილი, სადაც მონაცემები ინახება. მონაცემების სათავსო შეიძლება იყოს ფაილი, ქსელური შეერთება, მისამართი ინტერნეტში და მეხსიერების უბანი. არსებობს ნაკადების შემდეგი კლასები: Stream, FileStream, NetworkStream, MemoryStream, BufferedStream და CryptoStream. ჩვენ დაწვრილებით განვიხილავთ FileStream კლასს.

C#-პროგრამები მონაცემების შეტანა-გამოტანას ნაკადების საშუალებით ახდენენ. ნაკადის ქვეშ იგულისხმება მონაცემების მიმდევრობა. ნაკადი ფიზიკურ მოწყობილობას შეტანა-გამოტანის სისტემის საშუალებით უკავშირდება.

C# ენაში შეტანა-გამოტანის ოპერაციები ყველაზე დაბალ დონეზე ბაიტებთან ოპერირებენ, რადგან შეტანა-გამოტანის მოწყობილობების უმრავლესობა არის ბაიტ-ორიენტირებული. მეორეს მხრივ, კომპიუტერთან ურთიერთობისას ადამიანებისათვის მოხერხებულია სიმბოლოებთან მუშაობა. ამიტომ, შეტანა-გამოტანის ოპერაციების შესრულების დროს საჭირო ხდება ბაიტების გარდაქმნა სიმბოლოებად და პირიქით.

არსებობს სიმბოლოების ASCII და Unicode ნაკრები. ASCII სიმბოლოს ზომაა 1 ბაიტი, ხოლო Unicode სიმბოლოს ზომა კი - 2 ბაიტი. გავიხსენოთ, რომ C# ენაში char ტიპის სიმბოლო იკავებს 2 ბაიტს, byte ტიპის მნიშვნელობა კი - 1 ბაიტს. ამიტომ, ბაიტების ნაკადების გამოყენება არც ისე მოხერხებულია სიმბოლოების შეტანა-გამოტანის შესრულებისას. ამ ამოცანის გადასაწყვეტად C# ენაში განსაზღვრულია რამდენიმე კლასი, რომლებიც ახდენენ ბაიტების ნაკადების გარდაქმნას სიმბოლოების ნაკადებად, ე.ი. ავტომატურად გარდაქმნიან byte ტიპის მონაცემებს char ტიპის მონაცემებად და პირიქით.

C# ენაში არსებობს ბაიტებისა და სიმბოლოების ნაკადების კლასები. ნაკადების კლასები განსაზღვრულია System.IO კლასში. ამიტომ, ნებისმიერი პროგრამის დასაწყისში, რომელიც იყენებს ნაკადების კლასებს, უნდა ჩავრთოთ ინსტრუქცია using System.IO .

ბაიტების ნაკადები

C# ენაში ნაკადები იქმნება System.IO.Stream კლასის საშუალებით. Stream კლასი წარმოადგენს ბაიტების ნაკადებს და არის საბაზო ყველა სხვა კლასებისათვის. ის, აგრეთვე, აბსტრაქტულია, ამიტომ, არ არსებობს Stream ობიექტის რეალიზება. Stream კლასში განსაზღვრულია ნაკადებზე სტანდარტული ოპერაციები. ცხრილში 10.1 მოყვანილია Stream კლასში განსაზღვრული რამდენიმე ხშირად გამოყენებადი მეთოდი.

Stream კლასში განსაზღვრულია მეთოდები, განკუთვნილი მონაცემების ჩაწერისა და წაკითხვისათვის. მაგრამ, ყველა ნაკადი არ უზრუნველყოფს მონაცემების ჩაწერისა და წაკითხვის ოპერაციებს, აგრეთვე, პოზიციონირებას Seek() მეთოდის გამოყენებით. კონკრეტული ნაკადის შესაძლებლობების გასაგებად შეგვიძლია გამოვიყენოთ Stream კლასის თვისებები (ცხრილში 10.2).

Stream კლასს აქვს ბაიტების ნაკადების სამი მემკვიდრე კლასი, რომლებიც მოყვანილია ცხრილში 10.3.

ცხრილი 10.1. Stream კლასში განსაზღვრული ზოგიერთი მეთოდი.

მეთოდი	აღწერა
void Close()	ნაკადს ხურავს
void Flush()	ნაკადის შემცველობას ფიზიკურ მოწყობილობაზე წერს
int ReadByte()	შესასვლელი ნაკადიდან კითხულობს 1 ბაიტს და გასცემს მის მთელრიცხვა წარმოდგენას
int Read(byte[] მასივი, int პოზიცია, int ბაიტების_რაოდენობა)	შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის ბაიტს და მათ ათავსებს მასივში დაწყებული მითითებული პოზიციიდან. გაიცემა წარმატებით წაკითხული ბაიტების რაოდენობა
long Seek(long პოზიცია, SeekOrigin გადათვლის_დასაწყისი)	გასცემს ნაკადში მიმდინარე ბაიტის პოზიციას, გადათვლილს მითითებული პოზიციიდან
void WriteByte(byte ცვლადი)	გამოსასვლელ ნაკადში წერს ერთ ბაიტს
int Write(byte[] მასივი, int პოზიცია, int ბაიტების_რაოდენობა)	გამოსასვლელ ნაკადში მასივიდან წერს მითითებული რაოდენობის ბაიტს, დაწყებული მითითებული პოზიციიდან. გაიცემა ჩაწერილი ბაიტების რაოდენობა

ცხრილი 10.2. Stream კლასის თვისებები.

თვისება	აღწერა
bool CanRead	თვისებას აქვს true მნიშვნელობა, თუ შესაძლებელია ნაკადიდან წაკითხვა. თვისება განკუთვნილია მხოლოდ წასაკითხად.
bool CanSeek	თვისებას აქვს true მნიშვნელობა, თუ ნაკადის ელემენტის მიმდინარე პოზიციის განსაზღვრა შესაძლებელია. თვისება განკუთვნილია მხოლოდ წასაკითხად.
bool CanWrite	თვისებას აქვს true მნიშვნელობა, თუ შესაძლებელია ნაკადში ჩაწერა. თვისება განკუთვნილია მხოლოდ წასაკითხად.
long Length	თვისება განსაზღვრავს ნაკადის სიგრძეს. თვისება განკუთვნილია მხოლოდ წასაკითხად.
long Position	თვისება განსაზღვრავს ნაკადის მიმდინარე ელემენტის პოზიციას. თვისება განსაზღვრულია როგორც წაკითხვის, ისე ჩაწერისათვის.

ცხრილი 10.3. Stream კლასის მემკვიდრე კლასები.

კლასი	აღწერა
BufferedStream	უზრუნველყოფს ბაიტების ნაკადის ბუფერიზებას
FileStream	ბაიტების ნაკადია, განკუთვნილი ფაილში შეტანა-გამოტანის ოპერაციების შესასრულებლად
MemoryStream	ბაიტების ნაკადია, რომელიც გამოიყენება მეხსიერებასთან სამუშაოდ

ცხრილი 10.4. TextReader კლასის შეტანის მეთოდები.

მეთოდი	აღწერა
void Close()	ხურავს შესასვლელ ნაკადს
int Peek()	კითხულობს შესასვლელი ნაკადის მომდევნო სიმბოლოს, მაგრამ არ შლის მას. გასცემს -1-ს თუ არ არის მომდევნო სიმბოლო
int Read()	კითხულობს შესასვლელი ნაკადის სიმბოლოს და გასცემს მის მთელრიცხვა წარმოდგენას. გასცემს -1-ს ნაკადის დასასრულის მიღწევას
int Read(char[] მასივი, int პოზიცია, int სიმბოლოების_რაოდენობა)	შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ მასივში დაწყებული მითითებული პოზიციიდან. გაიცემა წარმატებით წაკითხული სიმბოლოების რაოდენობა
int ReadBlock(char[] მასივი, int პოზიცია, int სიმბოლოების_რაოდენობა)	შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ მასივში დაწყებული მითითებული პოზიციიდან. ამასთან, გაიცემა წარმატებით წაკითხული სიმბოლოების რაოდენობა
string ReadLine()	კითხულობს ტექსტის სტრიქონს და გასცემს მას C# ენის სტრიქონის სახით. ნულოვანი მნიშვნელობა იმ შემთხვევაში გაიცემა, თუ წაკითხული იქნა ფაილის დასასრულის სიმბოლო
string ReadToEnd()	კითხულობს ნაკადის დარჩენილ სიმბოლოებს და გასცემს მათ C# ენის სტრიქონის სახით

ცხრილი 10.5. TextWriter კლასის გამოტანის მეთოდები.

მეთოდი	აღწერა
void Write(int ცვლადი)	int ტიპის მნიშვნელობის ჩაწერა
void Write(double ცვლადი)	double ტიპის მნიშვნელობის ჩაწერა
void Write(bool ცვლადი)	bool ტიპის მნიშვნელობის ჩაწერა
void WriteLine(string ცვლადი)	სტრიქონის ჩაწერა, რომელიც მთავრდება მომდევნო სტრიქონზე გადასვლის სიმბოლოთი
void WriteLine(uint ცვლადი)	uint ტიპის მნიშვნელობისა და მომდევნო სტრიქონზე გადასვლის სიმბოლოს ჩაწერა
void WriteLine(char ცვლადი)	სიმბოლოს ჩაწერა, რომელსაც მოსდევს მომდევნო სტრიქონზე გადასვლის სიმბოლო
virtual void Close()	ნაკადს ხურავს
virtual void Flush()	ახდენს ბუფერში დარჩენილი მონაცემების ჩაწერას გამოსასვლელ ნაკადში

ცხრილი 10.6. სიმბოლოების ნაკადების კლასები.

ნაკადის კლასი	აღწერა
StreamReader	ბაიტების ნაკადიდან კითხულობს სიმბოლოებს
StreamWriter	სიმბოლოებს წერს ბაიტების ნაკადში
StringReader	სტრიქონიდან კითხულობს სიმბოლოებს
StringWriter	სტრიქონში წერს სიმბოლოებს

სიმბოლოების ნაკადები

სიმბოლოების ნაკადებთან სამუშაოდ გამოიყენება `TextReader` და `TextWriter` აბსტრაქტული კლასები. მათში განსაზღვრული მეთოდები უზრუნველყოფენ შეტანა-გამოტანის ოპერაციებს სიმბოლოების ნაკადებისათვის.

ცხრილში 10.4 მოყვანილია `TextReader` კლასის შეტანის მეთოდები. აღვნიშნოთ, რომ `ReadLine()` მეთოდის გამოყენება მოხერხებულია შესასვლელი ნაკადიდან იმ მონაცემების წასაკითხად, რომელიც ინტერვალებს შეიცავს. ცხრილში 10.5 მოყვანილია `TextWriter` კლასში განსაზღვრული `Write()` და `WriteLine()` მეთოდების სხვადასხვა ვერსიები.

`TextReader` და `TextWriter` კლასების რეალიზება ხდება ცხრილში 10.6 მოყვანილი სიმბოლოების ნაკადების კლასების საშუალებით. ეს ნაკადები უზრუნველყოფენ `TextReader` და `TextWriter` კლასებში განსაზღვრულ მეთოდებსა და თვისებებს.

ორობითი ნაკადები

C# ენაში განსაზღვრულია აგრეთვე, ორობითი ნაკადების ორი კლასი `BinaryReader` და `BinaryWriter`. ისინი შეგვიძლია გამოვიყენოთ ორობითი მონაცემების ჩაწერისა და წაკითხვისათვის.

FileStream კლასი

ეს კლასი გამოიყენება ფაილებში ბაიტ-ორიენტირებული შეტანა-გამოტანის შესასრულებლად, ანუ ფაილში ბაიტების ჩაწერა-წაკითხვის ოპერაციების შესასრულებლად. რადგან ფაილის ყველაზე გავრცელებული ტიპია დისკური ფაილი, ამიტომ შემდგომში ამ ტიპის ფაილებს განვიხილავთ.

ფაილის გახსნა და დახურვა

ფაილთან დაკავშირებული ბაიტების ნაკადის შესაქმნელად საჭიროა `FileStream` ობიექტის ფორმირება. მის ფორმირებას ახდენს კონსტრუქტორი, რომლის სინტაქსია:

`FileStream(string ფაილის_სახელი, FileMode.რეჟიმი);`

სადაც, `ფაილის_სახელი` არის გასახსნელი ფაილის სახელი, რომელიც შეიძლება შეიცავდეს, აგრეთვე, ამ ფაილისაკენ გზას, მაგალითად, `C:\My Documents\A1\File1.txt`. რეჟიმი პარამეტრი განსაზღვრავს თუ როგორ უნდა გაიხსნას ფაილი. მისი მნიშვნელობები ანუ ფაილის გახსნის რეჟიმები მოყვანილია ცხრილში 10.7.

ქვემოთ მოყვანილ პროგრამაში სრულდება `testfile.dat` ფაილის გახსნა და დახურვა. არ უნდა დაგვავიწყდეს პროგრამის დასაწყისში `using` გამოცხადებების შემდეგ `using System.IO` ინსტრუქციის ჩართვა. ეს უნდა გავაკეთოთ ამ თავში მოყვანილი ყველა პროგრამისათვის.

```
{
//   პროგრამა 10.1
//   პროგრამაში ხდება ფაილის გახსნა და დახურვა
FileStream file_in;

file_in = new FileStream("file1.txt", FileMode.Open);
file_in.Close();
}
```

ცხრილი 10.7. mode პარამეტრის მნიშვნელობები.

მნიშვნელობა	აღწერა
FileMode.Append	მონაცემები დაემატება ფაილის ბოლოში
FileMode.Create	იქმნება ახალი გამოსასვლელი ფაილი. ამავე სახელის მქონე ადრე შექმნილი ფაილი იშლება
FileMode.CreateNew	იქმნება ახალი გამოსასვლელი ფაილი
FileMode.Open	იხსნება არსებული ფაილი
FileMode.OpenCreate	თუ ფაილი არსებობს, მაშინ ის გაიხსნება, თუ არ არსებობს, მაშინ ის შეიქმნება
FileMode.Truncate	იხსნება არსებული ფაილი, მაგრამ მისი სიგრძე ჩამოიჭრება ნულამდე

პროგრამის პირველი ორი სტრიქონი შეგვიძლია შევცვალოთ ერთით:

```
FileStream file_in = new FileStream("file1.dat", FileMode.Open);
```

პროგრამაში მითითებულია ფაილის მხოლოდ სახელი, ამიტომ მისი ძებნა შესრულდება მიმდინარე კატალოგში. თუ ფაილი სხვა კატალოგშია მოთავსებული, მაშინ უნდა მივუთითოთ გზა ამ ფაილისაკენ:

```
file_in = new FileStream("C:\\My Documents\\Visual Studio Projects\\ A1\\file1.txt", FileMode.Open);
```

თუ ფაილთან მიმართვა უნდა შეიზღუდოს მხოლოდ წაკითხვით ან ჩაწერით, მაშინ უნდა მივუთითოთ ფაილთან მიმართვის სახე:

FileStream(string ფაილის_სახელი, რეჟიმი, FileAccess მიმართვის_სახე)

აქ მიმართვის_სახე იღებს მნიშვნელობებს: FileAccess.Read, FileAccess.Write და FileAccess.ReadWrite. მაგალითად,

```
FileStream file_in = new FileStream("file1.txt",FileMode.Open,FileAccess.Read);
```

აქ file1.txt ფაილი იხსნება წაკითხვის რეჟიმში.

ფაილთან მუშაობის დასამთავრებლად აუცილებელია მისი დახურვა Close() მეთოდის გამოყენებით. ფაილის დახურვისას თავისუფლდება მისთვის გამოყოფილი ოპერაციული სისტემის რესურსები.

ფაილიდან ბაიტების წაკითხვა

FileStream კლასში განსაზღვრულია ორი მეთოდი, რომლებიც ფაილიდან ბაიტებს კითხულობენ: ReadByte() და Read(). ReadByte() მეთოდის სინტაქსია:

int ReadByte()

ამ მეთოდის გამოძახებისას ხდება ერთი ბაიტის წაკითხვა ფაილიდან და გაიცემა ბაიტის კოდი (მთელრიცხვა მნიშვნელობა). თუ წაკითხული იქნა ფაილის დასასრულის სიმბოლო (EOF, End Of File), მაშინ გაიცემა -1.

ბაიტების ბლოკის წასაკითხად უნდა გამოვიყენოთ Read() მეთოდი. მისი სინტაქსია:

int Read(byte[] მასივი, int პოზიცია, int ბაიტების_რაოდენობა)

Read() მეთოდი ცდილობს ფაილიდან წაკითხოს მითითებული რაოდენობის ბაიტი და მოათავსოს ისინი მასივში დაწყებული მითითებული პოზიციიდან. გაიცემა რეალურად წაკითხული ბაიტების რაოდენობა.

მოყვანილ პროგრამაში სრულდება ბაიტების წაკითხვა ReadByte() მეთოდის გამოყენებით და მათი ეკრანზე გამოტანა. დავუშვათ filetext.txt ფაილში Notepad პროგრამის გამოყენებით ჩაწერილია ციფრები: 1234567890 .

```
{
// პროგრამა 10.2
// პროგრამაში ხდება ბაიტების წაკითხვის დემონსტრირება
```

```
// ReadByte() მეთოდის გამოყენებით
int ricxvi;
FileStream file1 = new FileStream("filetext.txt", FileMode.Open);

// მონაცემების წაკითხვა ფაილიდან მანამ, სანამ არ შეგვხვდება EOF სიმბოლო
for ( ; ; )
{
    ricxvi = file1.ReadByte();
    if ( ricxvi != -1 ) label1.Text += ( char ) ricxvi;
    else break;
}
file1.Close();
}
```

მოყვანილ პროგრამაში სრულდება ბაიტების წაკითხვა Read() მეთოდის გამოყენებით და მათი ეკრანზე გამოტანა. დავუშვათ filetext.txt ფაილში Notepad პროგრამის გამოყენებით ჩაწერილია ციფრები: 1234567890 .

```
{
// პროგრამა 10.3
// პროგრამაში ხდება ბაიტების წაკითხვის დემონსტრირება Read() მეთოდის გამოყენებით
int wakitxuli_baitebis_raodenoba;
int pozicia = int.Parse(textBox1.Text),
    raodenoba = int.Parse(textBox2.Text);
byte[] masivi = new byte[10];
FileStream file1 = new FileStream("filetext.txt", FileMode.Open);
label1.Text = "";

// masivi მასივში ჩაიწერება 4 ბაიტი დაწყებული მე-3 ელემენტიდან
wakitxuli_baitebis_raodenoba = file1.Read(masivi, pozicia, raodenoba);
file1.Close();
for (int indexi = 0; indexi < masivi.Length; indexi++)
    label1.Text += masivi[indexi].ToString() + " ";
label2.Text = wakitxuli_baitebis_raodenoba.ToString();
}
```

როგორც ვიცი Read() მეთოდს სამი არგუმენტი აქვს: მასივი, პოზიცია და ბაიტების რაოდენობა. შესაბამისად, პროგრამაში გამოცხადებულია masivi მასივი, pozicia და raodenoba ცვლადები. დავუშვათ, pozicia ცვლადს მივანიჭეთ 2, raodenoba ცვლადს კი - 4. შედეგად, filetext.txt ფაილიდან შესრულდება 4 ბაიტის წაკითხვა და masivi მასივში მესამე პოზიციიდან (masivi[2]) ჩაწერა. masivi მასივი მიიღებს სახეს:

```
0 0 49 50 51 52 0 0 0 0
```

Read() მეთოდი, აგრეთვე, გაცემს რეალურად წაკითხული ბაიტების რაოდენობას, რომელსაც მოვათავსებთ wakitxuli_baitebis_raodenoba ცვლადში.

ფაილში ბაიტების ჩაწერა

FileStream კლასში განსაზღვრულია ორი მეთოდი, რომლებიც ფაილში წერენ ბაიტებს: WriteByte() და Write(). WriteByte() მეთოდი გამოიყენება ფაილში ბაიტების ჩასაწერად. მისი სინტაქსია:

```
void WriteByte(byte ცვლადის_სახელი)
```

Write() მეთოდი გამოიყენება ფაილში ბაიტების ბლოკის (მასივის) ჩასაწერად. მისი სინტაქსია:

int Write(byte[] მასივი, int პოზიცია, int ბაიტების_რაოდენობა)

Write() მეთოდი მასივიდან ფაილში ჩაწერს მითითებული რაოდენობის ბაიტს, დაწყებული მითითებული პოზიციიდან. გაიცემა რეალურად ჩაწერილი ბაიტების რაოდენობა.

ფაილში ჩაწერის დროს ხშირად არ ხდება მონაცემების მაშინვე ჩაწერა ფიზიკურ მოწყობილობაზე. ამის ნაცვლად, ოპერაციული სისტემა ახდენს გამოტანის ბუფერიზებას, ანუ ჩასაწერი ბაიტების ბუფერში დაგროვებას. ბუფერი არის მეხსიერების გარკვეული უბანი. ბუფერიზება სრულდება მანამ, სანამ არ დაგროვდება მონაცემების საკმარისი რაოდენობა. ბუფერის შევსებისთანავე მონაცემები ჩაიწერება დისკზე. შედეგად, იზრდება ოპერაციული სისტემის მწარმოებლურობა. საქმე ის არის, რომ დისკზე მონაცემების ჩაწერა ხდება სექტორებად, რომელთა ზომები იცვლება დაწყებული 128 ბაიტიდან ზევით. გამოტანის შედეგების ბუფერიზება ხდება მანამ, სანამ დისკზე შესაძლებელი არ გახდება მთელი სექტორის ჩაწერა.

რიგ შემთხვევებში საჭიროა მონაცემების დისკზე ჩაწერა მიუხედავად იმისა ბუფერი შევსებულია თუ არა. ასეთ შემთხვევაში უნდა გამოვიყენოთ Flush() მეთოდი. მისი სინტაქსია:

void Flush()

ფაილთან მუშაობის დამთავრებისას ის უნდა დაიხუროს Close() მეთოდის გამოყენებით. ამ დროს ბუფერში მოთავსებული მონაცემები, დისკზე აუცილებლად ჩაიწერება. ამიტომ, Close() მეთოდის გამოძახების წინ არაა საჭირო Flush() მეთოდის გამოძახება.

მოყვანილ პროგრამაში სრულდება ერთი ფაილის მეორეში გადაწერა.

```
{
//   პროგრამა 10.4
//   პროგრამა ასრულებს ერთი ფაილის მეორეში გადაწერას
label1.Text = "";
int ricxvi;
FileStream file_in;
FileStream file_out;
//   საწყისი ფაილის გახსნა
file_in = new FileStream("s_file.txt", FileMode.Open);
//   მიმღები ფაილის გახსნა
file_out = new FileStream("d_file.txt", FileMode.Create);
//   ბაიტების გადაწერა file_in ფაილიდან file_out ფაილში
for ( ; ; )
{
//   ბაიტების წაკითხვა file_in ფაილიდან
ricxvi = file_in.ReadByte();
//   ბაიტების ჩაწერა file_out ფაილში
if ( ricxvi != -1 )
{
file_out.WriteByte(( byte ) ricxvi);
label1.Text += ( char ) ricxvi + " ";
}
else break;
}
//   ფაილების დახურვა
file_in.Close();
file_out.Close();
```

```

}
    მოყვანილ პროგრამაში Write() მეთოდის გამოიყენებით სრულდება filetext.txt ფაილში
    ბაიტების ჩაწერა და მათი ეკრანზე გამოტანა.
{
//    პროგრამა 10.5
//    პროგრამაში ხდება მასივიდან ფაილში ბაიტების ჩაწერა
byte[] masivi = new byte[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
FileStream file_out;
file_out = new FileStream("d_file.txt", FileMode.Create);

//    ბაიტების ჩაწერა d_file.txt ფაილში
file_out.Write(masivi, 2, 4);

file_out.Close();
}
    პროგრამის შესრულების შედეგად filetext.txt ფაილში ჩაიწერება masivi მასივის 4
    ელემენტი დაწყებული მესამე ელემენტიდან (masivi[2]).

```

ფაილში სიმბოლოების შეტანა-გამოტანა

ფაილში სიმბოლოების ჩასაწერად და წასაკითხად სიმბოლოების ნაკადები გამოიყენება. ისინი Unicode სიმბოლოებზე ოპერირებენ. სიმბოლოურ ფაილებთან მუშაობისას FileStream ობიექტი უნდა ჩავრთოთ StreamReader ან StreamWriter კლასის შემადგენლობაში. ეს კლასები უზრუნველყოფენ ბაიტების ნაკადების სიმბოლოების ნაკადებად ავტომატურ გარდაქმნას და პირიქით.

StreamWriter კლასი არის TextWriter კლასის მემკვიდრე. StreamReader კლასი არის TextReader კლასის მემკვიდრე. შედეგად, StreamWriter და StreamReader კლასის წევრებს შეუძლიათ მიმართონ TextWriter და TextReader კლასების მეთოდებსა და თვისებებს.

StreamWriter კლასი

სიმბოლოების გამოსასვლელი ნაკადი გამოიყენება სიმბოლოების ფაილში მონაცემების ჩასაწერად. ამისათვის, FileStream ობიექტი უნდა ჩავრთოთ StreamWriter კლასის შემადგენლობაში. ამ კლასში განსაზღვრულია რამდენიმე კონსტრუქტორი. ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

StreamWriter(Stream ნაკადის_სახელი)

ობიექტის შექმნისთანავე StreamWriter კლასი ავტომატურად გარდაქმნის სიმბოლოებს ბაიტებად.

მოყვანილი პროგრამა ახდენს textBox კომპონენტში შეტანილი ტექსტის ჩაწერას ფაილში.

```

{
//    პროგრამა 10.6
//    პროგრამაში ხდება ფაილში ჩაწერის დემონსტრირება StreamWriter კლასის გამოყენებით
string striqoni = textBox1.Text;
FileStream file_out = new FileStream("text.txt", FileMode.Create);
//
StreamWriter file_stream_out = new StreamWriter(file_out);

```

```
//      textBox კომპონენტში შეტანილი ტექსტის მინიჭება striqoni ცვლადისათვის
file_stream_out.Write(striqoni);      //      text.txt ფაილში striqoni სტრიქონის ჩაწერა
file_stream_out.Close();
}
```

StreamReader კლასი

სიმბოლოების შესასვლელი ნაკადი გამოიყენება სიმბოლოების ნაკადიდან მონაცემების წასაკითხად. ამისათვის, FileStream ობიექტი უნდა ჩავრთოთ StreamReader კლასის შემადგენლობაში. ამ კლასში განსაზღვრულია რამდენიმე კონსტრუქტორი. ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

StreamReader(Stream ნაკადის_სახელი)

StreamReader ობიექტის შექმნისთანავე ავტომატურად სრულდება ბაიტების გარდაქმნა სიმბოლოებად.

მოყვანილი პროგრამა კითხულობს ტექსტურ ფაილს და მისი შემცველობა ეკრანზე გამოაქვს. Notepad პროგრამის გამოყენებით ფაილში ჩავწეროთ რამდენიმე სტრიქონი და შემდეგ შევასრულოთ მოყვანილი პროგრამა.

```
{
//      პროგრამა 10.7
//      პროგრამაში ხდება ტექსტური ფაილიდან წაკითხვის დემონსტრირება
//      StreamReader კლასის გამოყენებით
label1.Text = "";
FileStream file_in;
string striqoni;
file_in = new FileStream("file1.txt", FileMode.Open );
StreamReader file_stream_in = new StreamReader(file_in);
//
for ( ; ( striqoni = file_stream_in.ReadLine() ) != null; )
    label1.Text += striqoni + '\n';
file_stream_in.Close();
}
```

პროგრამაში ფაილიდან სტრიქონების წაკითხვა მთავრდება მაშინ, როცა ReadLine() მეთოდის მიერ გაცემული სტრიქონი მიიღებს ნულოვან მნიშვნელობას. სწორედ ეს მიუთითებს ფაილის დასასრულზე.

ორობითი მონაცემების წაკითხვა და ჩაწერა

ფაილში ბაიტებისა და სიმბოლოების წაკითხვისა და ჩაწერის გარდა შესაძლებელია სხვა ტიპის (ორობითი) მონაცემების წაკითხვა და ჩაწერა, როგორცაა int, double და ა.შ. მათი წაკითხვისა და ჩაწერისათვის გამოიყენება BinaryReader და BinaryWriter ნაკადები. ასეთი მონაცემების წაკითხვა და ჩაწერა ხდება შიდა ორობით ფორმატში და არა ტექსტური ფორმით.

BinaryWriter კლასი

BinaryWriter ნაკადი გამოიყენება ფაილში ორობითი მონაცემების ჩასაწერად. ამ კლასის ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

BinaryWriter(Stream გამოსასვლელი_ნაკადი)

აქ გამოსასვლელი_ნაკადი პარამეტრი განსაზღვრავს იმ ნაკადს, რომელშიც სრულდება

მონაცემების ჩაწერა. ის არის ობიექტი, რომელსაც ქმნის FileStream კონსტრუქტორი.

BinaryWriter კლასში განსაზღვრულია მეთოდები, რომლებიც უზრუნველყოფენ C# ენის ყველა ჩადგმული ტიპის მონაცემების ჩაწერას. ზოგიერთი მეთოდი მოყვანილია ცხრილში 10.8. ამავე კლასში განსაზღვრულია, აგრეთვე, Close() და Flush() მეთოდები.

ცხრილი 10.8. BinaryWriter ნაკადის ზოგიერთი მეთოდი.

მეთოდი	აღწერა
void Write(sbyte ცვლადი)	ნიშნის ბაიტის ჩაწერა
void Write(byte ცვლადი)	უნიშნო ბაიტის ჩაწერა
void Write(byte[] მასივი)	ბაიტების მასივის ჩაწერა
void Write(short ცვლადი)	მოკლე მთელი რიცხვის ჩაწერა
void Write(ushort ცვლადი)	უნიშნო მოკლე მთელი რიცხვის ჩაწერა
void Write(int ცვლადი)	მთელი რიცხვის ჩაწერა
void Write(uint ცვლადი)	უნიშნო მთელი რიცხვის ჩაწერა
void Write(long ცვლადი)	გრძელი მთელი რიცხვის ჩაწერა
void Write(ulong ცვლადი)	უნიშნო გრძელი მთელი რიცხვის ჩაწერა
void Write(float ცვლადი)	float ტიპის მნიშვნელობის ჩაწერა
void Write(double ცვლადი)	double ტიპის მნიშვნელობის ჩაწერა
void Write(char ცვლადი)	სიმბოლოს ჩაწერა
void Write(char[] მასივი)	სიმბოლოების მასივის ჩაწერა
void Write(string ცვლადი)	სტრიქონის ჩაწერა

ცხრილი 10.9. BinaryReader ნაკადის ხშირად გამოყენებადი მეთოდები.

მეთოდი	აღწერა
bool ReadBoolean()	bool ტიპის მნიშვნელობის წაკითხვა
byte ReadByte()	byte ტიპის მნიშვნელობის წაკითხვა
sbyte ReadSByte()	sbyte ტიპის მნიშვნელობის წაკითხვა
byte[] ReadBytes(int ბაიტების_რაოდენობა)	მითითებული რაოდენობის ბაიტის წაკითხვა და მათი გაცემა მასივის სახით
char ReadChar()	char ტიპის მნიშვნელობის წაკითხვა
char[] ReadChars(int სიმბოლოების_რაოდენობა)	მითითებული რაოდენობის სიმბოლოს წაკითხვა და მათი გაცემა მასივის სახით
double ReadDouble()	double ტიპის მნიშვნელობის წაკითხვა
float ReadSingle()	float ტიპის მნიშვნელობის წაკითხვა
short ReadInt16()	short ტიპის მნიშვნელობის წაკითხვა
int ReadInt32()	int ტიპის მნიშვნელობის წაკითხვა
long ReadInt64()	long ტიპის მნიშვნელობის წაკითხვა
ushort ReadUInt16()	ushort ტიპის მნიშვნელობის წაკითხვა
uint ReadUInt32()	uint ტიპის მნიშვნელობის წაკითხვა
ulong ReadUInt64()	ulong ტიპის მნიშვნელობის წაკითხვა
string ReadString()	სტრიქონის წაკითხვა

BinaryReader კლასი

BinaryReader ნაკადი გამოიყენება ფაილიდან ორობითი მონაცემების წასაკითხად. ამ კლასის ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

BinaryReader(Stream შესასვლელი_ნაკადი)

აქ შესასვლელი_ნაკადი პარამეტრი განსაზღვრავს იმ ნაკადს, საიდანაც სრულდება მონაცემების წაკითხვა. ის არის ობიექტი, რომელსაც ქმნის FileStream კონსტრუქტორი.

BinaryReader კლასში განსაზღვრულია მეთოდები, რომლებიც ახდენენ C# ენის ყველა მარტივი ტიპის მონაცემების წაკითხვას. ხშირად გამოყენებადი მეთოდები მოყვანილია ცხრილში 10.9. ამავე კლასში განსაზღვრულია, აგრეთვე, Read() მეთოდის სამი ვერსია:

int Read() - კითხულობს შესასვლელი ნაკადის მომდევნო სიმბოლოს და გასცემს მის მთელრიცხვად წარმოდგენას. ფაილის დასასრულის მიღწევისას გაიცემა -1.

int Read(byte[] მასივი, int პოზიცია, int ბაიტების_რაოდენობა) - შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის ბაიტს და ათავსებს მათ მასივში დაწყებული მითითებული პოზიციიდან. გაიცემა წარმატებით წაკითხული ბაიტების რაოდენობა.

int Read(char[] მასივი, int პოზიცია, int სიმბოლოების_რაოდენობა) - შესასვლელი ნაკადიდან კითხულობს მითითებული რაოდენობის სიმბოლოს და ათავსებს მათ მასივში დაწყებული მითითებული პოზიციიდან. გაიცემა წარმატებით წაკითხული სიმბოლოების რაოდენობა.

ამავე ნაკადში განსაზღვრულია, აგრეთვე, Close() მეთოდი.

ქვემოთ მოყვანილ პროგრამაში გამოყენებულია BinaryReader და BinaryWriter ნაკადები. პროგრამაში ჯერ სრულდება ფაილში სხვადასხვა ტიპის მონაცემების ჩაწერა, შემდეგ კი წაკითხვა.

```
{
//   პროგრამა 10.8
//   პროგრამაში ხდება ფაილში ორობითი მონაცემების ჩაწერა-წაკითხვის დემონსტრირება
BinaryReader file_in;
BinaryWriter file_out;
int  mteli1 = 10, mteli2;
double wiladi1 = 1001.47, wiladi2, wiladi3;
bool  b1 = true, b2;
file_out = new BinaryWriter(new FileStream("file1.dat", FileMode.Create));
//   ფაილში მთელი რიცხვის ჩაწერა
file_out.Write(mteli1);
//   ფაილში წილადის ჩაწერა
file_out.Write(wiladi1);
//   ფაილში ლოგიკური მონაცემის ჩაწერა
file_out.Write(b1);
//   ფაილში იწერება 10.2 * 2.3 გამოსახულების გამოთვლის შედეგი
file_out.Write(10.2 * 2.3);
file_out.Close();
//   ფაილიდან წაკითხვა
file_in = new BinaryReader(new FileStream("file1.dat", FileMode.Open));
//   მთელი რიცხვის წაკითხვა ფაილიდან
mteli2= file_in.ReadInt32();
label1.Text = mteli2.ToString();
//   წილადის წაკითხვა ფაილიდან
wiladi2 = file_in.ReadDouble();
label2.Text = wiladi2.ToString();
//   ლოგიკური მონაცემის წაკითხვა ფაილიდან
```

```

b2 = file_in.ReadBoolean();
label3.Text = b2.ToString();
// წილადის წაკითხვა ფაილიდან
wiladi3 = file_in.ReadDouble();
label4.Text = wiladi3.ToString();
file_in.Close();
}

```

ფაილებთან პირდაპირი მიმართვა

აქამდე ვიყენებდით ფაილებთან მიმდევრობით მიმართვას. ამ დროს მიმდევრობით სრულდება ფაილში მონაცემების ჩაწერა და ფაილიდან მონაცემების წაკითხვა. ფაილთან შესაძლებელია, აგრეთვე, პირდაპირი (ნებისმიერი) მიმართვა. ამისათვის, გამოიყენება საფაილო მიმთითებელი. ის არის ფაილის მიმდინარე ელემენტის პოზიციის ნომერი. საფაილო მიმთითებლის მნიშვნელობის შესაცვლელად გამოიყენება Seek() მეთოდი. მისი სინტაქსია:

long Seek(long პოზიციის_ნომერი, SeekOrigin გადათვლის_დასაწყისი)

პოზიციის_ნომერი პარამეტრი მიუთითებს საფაილო მიმთითებლის ახალ პოზიციას (ბაიტებში) გადათვლილს გადათვლის_დასაწყისი პარამეტრით განსაზღვრული ადგილიდან. გადათვლის_დასაწყისი პარამეტრი იღებს SeekOrigin ჩამონათვალში განსაზღვრული მნიშვნელობებიდან ერთ-ერთს (ცხრილი 10.10).

ცხრილი 10.10. გადათვლის_დასაწყისი პარამეტრის მნიშვნელობები.

მნიშვნელობა	აღწერა
Begin	გადათვლა ფაილის დასაწყისიდან
Current	გადათვლა მიმდინარე პოზიციიდან
End	გადათვლა ფაილის დასასრულიდან

Seek() მეთოდის გამოძახების შემდეგ წაკითხვის ან ჩაწერის ოპერაციები სრულდება საფაილო მიმთითებლის მიერ განსაზღვრული ახალი პოზიციიდან.

ქვემოთ მოყვანილ პროგრამაში, ხდება შეტანა-გამოტანის ოპერაციის დემონსტრირება ფაილთან ნებისმიერი მიმართვით. ფაილში ჯერ ჩაიწერება ანბანის ასოები, შემდეგ კი ხდება მათი სხვადასხვა მიმდევრობით წაკითხვა.

```

{
// პროგრამა 10.9
// პროგრამაში ხდება ფაილში ჩაწერა-წაკითხვის ოპერაციების შესრულება
// პირდაპირი მიმართვით
FileStream file;
file = new FileStream("file.txt",FileMode.Create);
char simbolo;
// ასოების ჩაწერა ანბანის მიხედვით
for ( int i = 0; i < 26; i++ )
{
file.WriteByte( ( byte ) ( 'a' + i ) );
}
// ასოების წაკითხვა ფაილიდან
file.Seek(0, SeekOrigin.Begin); // პირველი ბაიტის არჩევა
simbolo = (char) file.ReadByte(); // პირველი ბაიტის წაკითხვა
}

```

```

label1.Text = "პირველი მნიშვნელობა " + simbolo + '\n';
file.Seek(4, SeekOrigin.Begin); // მეხუთე ბაიტის არჩევა
simbolo = ( char ) file.ReadByte(); // მეხუთე ბაიტის წაკითხვა
label1.Text += "მეხუთე მნიშვნელობა " + simbolo + '\n';
file.Close();
}

```

ფაილის შესახებ ინფორმაციის მიღება

System.IO სახელების სივრცეში განსაზღვრულია, აგრეთვე, File და FileInfo კლასები. File კლასში განსაზღვრულია სტატიკური მეთოდები (ცხრილი 10.11), FileInfo კლასში კი - ჩვეულებრივი მეთოდები (ცხრილი 10.12). რადგან File კლასის მეთოდები სტატიკურია, ამიტომ მათთან სამუშაოდ არ არის საჭირო ობიექტის შექმნა, ხოლო FileInfo კლასის მეთოდებთან სამუშაოდ კი საჭიროა ობიექტის შექმნა. ცხრილში 10.13 მოყვანილია FileInfo კლასის თვისებები.

ცხრილი 10.11. File კლასში განსაზღვრული სტატიკური მეთოდები

მეთოდი	აღწერა
AppendText()	არსებულ ფაილს ტექსტს უმატებს
Copy()	ახდენს ფაილის გადაწერას
Create()	ქმნის ფაილს
CreateText()	ქმნის ფაილს და ხსნის მას ტექსტის ჩასაწერად
Delete()	შლის ფაილს
Exists()	ამოწმებს ფაილის არსებობას
GetAttributes()	გასცემს ფაილის ატრიბუტებს
GetCreationTime()	გასცემს ფაილის შექმნის თარიღს
GetLastAccessTime()	გასცემს ფაილთან უკანასკნელი მიმართვის თარიღს
GetLastWriteTime()	გასცემს ფაილში უკანასკნელი ჩაწერის თარიღს
Move()	ახდენს ფაილის გადაადგილებას
Open()	ხსნის ფაილს
OpenRead()	ხსნის ფაილს წაკითხვისათვის
OpenText()	ხსნის ფაილს ტექსტის წაკითხვისათვის
OpenWrite()	ხსნის ფაილს ჩაწერისათვის
SetAttributes()	აყენებს ფაილის ატრიბუტებს
SetCreationTime()	აყენებს ფაილის შექმნის თარიღს
SetLastAccessTime()	აყენებს ფაილთან უკანასკნელი მიმართვის თარიღს
SetLastWriteTime()	აყენებს ფაილში უკანასკნელი ჩაწერის თარიღს

როგორც ცხრილებიდან ჩანს, თითოეული კლასი იძლევა ფაილის შექმნის, წაშლის, გადატანის, გახსნის და ა.შ. საშუალებას. ბუნებრივია, იბადება კითხვა - თუ რომელი კლასი უნდა გამოვიყენოთ ფაილებთან სამუშაოდ. ასეთ დროს უნდა გავითვალისწინოთ რამდენიმე მოსაზრება. ჯერ ერთი, FileInfo კლასში განსაზღვრულია ისეთი ოპერაციები, რომლებიც არ არის განსაზღვრული File კლასში. მეორე, File კლასი უფრო სწრაფად მუშაობს ფაილებზე ერთი ოპერაციის შესრულებისას, რადგან დრო არ იხარჯება ობიექტის შექმნაზე. FileInfo კლასი უფრო სწრაფად მუშაობს ფაილზე რამდენიმე ოპერაციის შესრულებისას, რადგან ფაილთან მიმართვის უფლებები მოწმდება ერთხელ მისი შექმნის დროს, და არა მასთან ყოველი მიმართვისას.

ცხრილი 10.12. FileInfo კლასში განსაზღვრული მეთოდები

მეთოდი	აღწერა
AppendText()	არსებულ ფაილს ტექსტს უმატებს
CopyTo()	ახდენს ფაილის გადაწერას
Create()	ქმნის ფაილს
CreateText()	ქმნის ტექსტურ ფაილს
Delete()	შლის ფაილს
MoveTo()	ახდენს ფაილის გადატანას
Open()	ხსნის ფაილს
OpenText()	ხსნის ფაილს ტექსტის წასაშლელად
OpenWrite()	ხსნის ფაილს ჩაწერისათვის

ცხრილი 10.13. FileInfo კლასში განსაზღვრული თვისებები

თვისება	ტიპი	აღწერა
Attributes	FileAttributes	გასცემს ან აყენებს ფაილის ატრიბუტებს
CreationTime	DateTime	გასცემს ან აყენებს ფაილის შექმნის თარიღს
Directory	DirectoryInfo	გასცემს კატალოგს, რომელშიც ფაილია მოთავსებული
DirectoryName	string	გასცემს კატალოგის სახელს, რომელშიც ფაილია მოთავსებული
Exists	bool	თუ ფაილი არსებობს, მაშინ გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი false მნიშვნელობას
Extension	string	გასცემს ფაილის გაფართოებას
FullName	string	გასცემს ფაილისკენ გზას და ფაილის სახელს
LastAccessTime	DateTime	გასცემს ან აყენებს ფაილთან უკანასკნელი მიმართვის თარიღს
LastWriteTime	DateTime	გასცემს ან აყენებს ფაილში უკანასკნელი ჩაწერის თარიღს
Length	long	გასცემს ფაილის ზომას
Name	string	გასცემს ფაილის სახელს

განვიხილოთ მაგალითები. მოყვანილ პროგრამაში ხდება File კლასის AppendText მეთოდთან მუშაობის დემონსტრირება.

```

{
//   პროგრამა 10.10
//   პროგრამაში ხდება File კლასის AppendText() მეთოდთან მუშაობის დემონსტრირება
//   ფაილისათვის ხდება სტრიქონების დამატება
string path = "file1.txt";
using ( StreamWriter faili = File.AppendText(path) )
{
    faili.WriteLine("რომან");
    faili.WriteLine("სამხარაძე");
    faili.WriteLine("C# დაპროგრამების ენაა");
    faili.Close();
}
}

```

```
}
```

აქ using არის ოპერატორი, რომელიც განსაზღვრავს მოქმედების უბანს, რომლის ბოლოსაც faili ობიექტი წაიშლება.

მოყვანილ პროგრამაში ხდება File კლასის OpenText მეთოდთან მუშაობის დემონსტრირება.

```
{
```

```
// პროგრამა 10.11
```

```
// პროგრამაში ხდება File კლასის OpenText() მეთოდთან მუშაობის დემონსტრირება
```

```
// ფაილის გახსნა წაკითხვის მიზნით
```

```
label1.Text = "";
```

```
using ( StreamReader faili = File.OpenText("file1.txt") )
```

```
{
```

```
    string striqoni = "";
```

```
//
```

```
    for ( ; ( striqoni = faili.ReadLine() ) != null; )
```

```
        label1.Text += striqoni + " ";
```

```
}
```

```
}
```

მოყვანილ პროგრამაში ხდება File კლასის CreateText და Exists მეთოდთან მუშაობის დემონსტრირება.

```
{
```

```
// პროგრამა 10.12
```

```
// პროგრამაში ხდება File კლასის CreateText() მეთოდთან მუშაობის დემონსტრირება
```

```
string path = "file1.txt";
```

```
// მოწმდება ფაილის არსებობა
```

```
if ( !File.Exists(path) )
```

```
{
```

```
// ფაილის შექმნა მასში სტრიქონების ჩაწერის მიზნით
```

```
using ( StreamWriter faili = File.CreateText(path) )
```

```
{
```

```
    faili.WriteLine("ანა ");
```

```
    faili.WriteLine("და ");
```

```
    faili.WriteLine("საბა ");
```

```
    faili.WriteLine("სამხარაძეები");
```

```
}
```

```
}
```

```
}
```

მოყვანილ პროგრამაში ხდება File კლასის Delete და Copy მეთოდებთან მუშაობის დემონსტრირება.

```
{
```

```
// პროგრამა 10.13
```

```
// პროგრამაში ხდება File კლასის Delete() და Copy() მეთოდთან მუშაობის დემონსტრირება
```

```
string path1 = "file1.txt";
```

```
string path2 = "file1.tmp";
```

```
// file1.txt ფაილის შექმნა
```

```
using ( FileStream faili_obj = File.Open(path1, FileMode.Create) ) { }
```

```

//      file1.tmp ფაილის წაშლა
File.Delete(path2);

//      file1.txt ფაილის გადაწერა file1.tmp ფაილში
File.Copy(path1, path2);
label1.Text = path1 + " ფაილი გადაიწერა " + path2 + "ფაილში";
}

მოყვანილ პროგრამაში ხდება File კლასის GetAttributes() მეთოდთან მუშაობის დემონსტრირება.
{
//      პროგრამა 10.14
//      პროგრამაში ხდება File კლასის GetAttributes() მეთოდთან მუშაობის დემონსტრირება
{
FileAttributes faili_obj = File.GetAttributes("file1.txt");
label1.Text = faili_obj.ToString();
}

მოყვანილ პროგრამაში ხდება FileInfo კლასის თვისებებთან მუშაობის დემონსტრირება.
{
//      პროგრამა 10.15
//      პროგრამაში ხდება FileInfo კლასის თვისებებთან მუშაობის დემონსტრირება
System.IO.FileInfo obieqti = new System.IO.FileInfo("file1.txt");

label1.Text += obieqti.Directory.ToString() + '\n';
label1.Text += obieqti.DirectoryName + '\n';
label1.Text += obieqti.FullName + '\n';
label1.Text += obieqti.Name + '\n';
label1.Text += obieqti.Extension + '\n';
label1.Text += obieqti.Length.ToString() + '\n';
label1.Text += obieqti.Exists.ToString() + '\n';
label1.Text += obieqti.CreationTime.ToString() + '\n';
label1.Text += obieqti.LastAccessTime.ToString() + '\n';
label1.Text += obieqti.LastWriteTime.ToString() + '\n';
}
}

```

კატალოგებთან მუშაობა

System.IO სახელების სივრცეში განსაზღვრულია, აგრეთვე, Directory და DirectoryInfo კლასები. File კლასის ანალოგიურად Directory კლასში განსაზღვრულია სტატიკური მეთოდები (ცხრილი 10.14), DirectoryInfo კლასში კი - ჩვეულებრივი მეთოდები (ცხრილი 10.15). რადგან Directory კლასის მეთოდები სტატიკურია, ამიტომ მათთან სამუშაოდ არ არის საჭირო ობიექტის შექმნა, ხოლო DirectoryInfo კლასის მეთოდებთან სამუშაოდ კი საჭიროა ობიექტის შექმნა. ცხრილში 10.16 მოყვანილია DirectoryInfo კლასის თვისებები.

ცხრილი 10.14. Directory კლასში განსაზღვრული მეთოდები

მეთოდი	აღწერა
--------	--------

CreateDirectory()	ქმნის ახალ კატალოგს
Delete()	შლის კატალოგს მის შემცველობასთან ერთად
Exists()	განსაზღვრავს არსებობს თუ არა კატალოგი
GetCreationTime()	გასცემს კატალოგის შექმნის თარიღს
GetCurrentDirectory()	გასცემს მიმდინარე კატალოგს
GetDirectories()	გასცემს მიმდინარე კატალოგის ქვეკატალოგების სახელებს
GetDirectoryRoot()	მითითებული კატალოგისათვის გასცემს ძირითადი კატალოგის სახელს
GetFiles()	გასცემს მითითებულ კატალოგში მოთავსებული ყველა ფაილის სახელს
GetFileSystemEntries()	გასცემს მითითებულ კატალოგში მოთავსებული ყველა ქვეკატალოგისა და ფაილის სახელს
GetLastAccessTime()	გასცემს კატალოგთან უკანასკნელი მიმართვის თარიღს
GetLastWriteTime()	გასცემს კატალოგში უკანასკნელი ჩაწერის თარიღს
GetLogicalDrives()	გასცემს ლოგიკური დისკების სიას
GetParent()	გასცემს მშობელი კატალოგის სახელს
Move()	აზღვევს კატალოგის გადატანას
SetCreationTime()	აყენებს კატალოგის შექმნის თარიღს
SetCurrentDirectory()	აყენებს მიმდინარე კატალოგს
SetLastAccessTime()	აყენებს კატალოგთან უკანასკნელი მიმართვის თარიღს
SetLastWriteTime()	აყენებს კატალოგში უკანასკნელი ჩაწერის თარიღს

ცხრილი 10.15. DirectoryInfo კლასში განსაზღვრული მეთოდები

მეთოდი	აღწერა
Create()	ქმნის ახალ კატალოგს
CreateSubdirectory()	ქმნის ახალ ქვეკატალოგს
Delete()	შლის კატალოგს
GetDirectories()	გასცემს მიმდინარე კატალოგის ქვეკატალოგების სიას
GetFiles()	გასცემს მიმდინარე კატალოგის ფაილების სიას
MoveTo()	აზღვევს კატალოგის გადატანას

მოვიყვანოთ რამდენიმე პროგრამა, რომლებშიც სრულდება კატალოგებთან მუშაობის დემონსტრირება Directory კლასის მეთოდების გამოყენებით. ქვემოთ მოყვანილ პროგრამაში ხდება ეკრანზე მითითებული კატალოგის სახელის, მასში მოთავსებული ფაილებისა და კატალოგების სახელების გაცემა, კატალოგის გახსნა, ეკრანზე დისკების სახელების გამოტანა. პროგრამაში კატალოგის სახელის შეტანა ხდება textBox1.Text კომპონენტში, მაგალითად, C:\\Windows. ამ დროს ბრჭყალების გამოყენება საჭირო არ არის.

ცხრილი 10.16. DirectoryInfo კლასში განსაზღვრული თვისებები

მეთოდი	ტიპი	აღწერა
Attributes	FileAttributes	გასცემს ან აყენებს კატალოგის ატრიბუტებს
CreationTime	DateTime	გასცემს ან აყენებს კატალოგის შექმნის დროს
Exist	bool	ამოწმებს კატალოგის არსებობას
Extension	string	გასცემს კატალოგის გაფართოებას
FullName	string	გასცემს გზას კატალოგისკენ და კატალოგის სახელს
LastAccessTime	DateTime	გასცემს ან აყენებს კატალოგთან უკანასკნელი მიმართვის დროს
LastWriteTime	DateTime	გასცემს ან აყენებს კატალოგში უკანასკნელი ჩაწერის დროს
Name	string	გასცემს კატალოგის სახელს
Parent	string	გასცემს მშობელი კატალოგის სახელს
Root	string	გასცემს ძირითადი კატალოგის სახელს მითითებული კატალოგისათვის

```

{
//   პროგრამა 10.16
//   პროგრამაში ხდება მიმდინარე და მშობელი კატალოგების სახელების, აგრეთვე,
//   მითითებულ კატალოგში მოთავსებული ფაილებისა და კატალოგების სახელების
//   გამოტანა ეკრანზე
//   მიმდინარე კატალოგის სახელის გაცემა
label1.Text = Directory.GetCurrentDirectory() + '\n';
//   მშობელი კატალოგის სახელის გაცემა
label1.Text += Directory.GetParent(textBox1.Text).ToString()+ '\n';
//   მითითებული კატალოგის გახსნა
Directory.SetCurrentDirectory(textBox1.Text);
//   ეკრანზე ფაილების გამოტანა
string[] striqonebis_masivi_1 = Directory.GetFiles(textBox1.Text);
foreach ( string s in striqonebis_masivi_1 )
    label2.Text += s + '\n';
//   ეკრანზე კატალოგების გამოტანა
string[] striqonebis_masivi_2 = Directory.GetDirectories(textBox1.Text);
foreach ( string s in striqonebis_masivi_2 )
    label1.Text += s + '\n';
//   ეკრანზე დისკების გამოტანა
string[] striqonebis_masivi_3 = Directory.GetLogicalDrives();
foreach ( string s in striqonebis_masivi_3 )
    label1.Text += s + '\n';
}

```

ქვემოთ მოყვანილ პროგრამაში სრულდება კატალოგის შექმნის, წაშლისა და გადატანის დემონსტრირება. Move() მეთოდში ჯერ ეთითება საწყისი კატალოგის სახელი, შემდეგ კი - საბოლოო კატალოგის სახელი. ორივე კატალოგი ერთ დისკზე უნდა იყოს მოთავსებული.

```

{
//   პროგრამა 10.17
//   პროგრამაში ხდება კატალოგის შექმნა, წაშლისა და გადატანის დემონსტრირება

```

```

label2.Text = "";
// კატალოგის შექმნა
Directory.CreateDirectory(textBox1.Text);
// კატალოგის წაშლა. წასაშლელი კატალოგი ცარიელი უნდა იყოს
Directory.Delete(textBox2.Text);
// კატალოგის გადატანა. Move მეთოდში ჯერ ეთითება საწყისი კატალოგი,
// შემდეგ კი - მიმღები.
// მიმღები კატალოგი არ უნდა არსებობდეს
Directory.Move(textBox3.Text, "C:\\Katalogi2");
}

```

ქვემოთ მოყვანილ პროგრამაში სრულდება კატალოგებთან მუშაობის დემონსტრირება DirectoryInfo კლასის მეთოდების გამოყენებით. MoveTo() მეთოდი ახდენს Katalogi3 კატალოგში მოთავსებული ფაილებისა და კატალოგების გადატანას Katalogi4 კატალოგში. გადატანის შემდეგ Katalogi3 კატალოგი იშლება.

```

{
// პროგრამა 10.18
// პროგრამაში ხდება ეკრანზე მითითებული კატალოგის სახელისა და ატრიბუტების
// გამოტანა, აგრეთვე, ერთი კატალოგის შემცველობის გადატანა მეორე კატალოგში
label1.Text = "";
DirectoryInfo obj1 = new DirectoryInfo("C:\\Zprint");
label1.Text += obj1.Name + '\n';
label1.Text += obj1.Attributes.ToString() + '\n';
}

```

ფაილების არჩევა

ამ თავში განხილულ პროგრამებთან მუშაობისას ფაილის სახელის მითითებისათვის საჭიროა პროგრამის კოდის შეცვლა, რაც ბუნებრივია, გარკვეულ წილად ართულებს სხვადასხვა ფაილთან მუშაობას. ამ პრობლემის გადასაწყვეტად გამოიყენება OpenFileDialog კლასი, რომელიც განსაზღვრულია System.Windows.Forms სახელების სივრცეში. მისი ShowDialog() მეთოდი გამოიყენება ფაილის გახსნის სტანდარტული დიალოგური ფანჯრის გამოსატანად ეკრანზე. ამ მეთოდთან მუშაობის დემონსტრირება ხდება ქვემოთ მოყვანილ პროგრამაში.

```

{
// პროგრამა 10.19
// პროგრამაში ხდება Open ფანჯრის გახსნის დემონსტრირება
label1.Text = "";
OpenFileDialog FailisGaxsna = new OpenFileDialog();
if ( FailisGaxsna.ShowDialog() == DialogResult.OK )
{
    FileInfo faili = new FileInfo(FailisGaxsna.FileName);
    label1.Text += faili.FullName + '\n';
    label1.Text += faili.Length.ToString() + '\n';
    label1.Text += faili.LastAccessTime.ToString() + '\n';
    label1.Text += faili.DirectoryName + '\n';
}
}

```

ქვემოთ მოყვანილ პროგრამაში ხდება ფაილიდან მონაცემების წაკითხვის დემონსტრირება Open დიალოგური ფანჯრის გამოყენებით. FailisGaxsna ობიექტის ShowDialog() მეთოდი ხსნის Open დიალოგურ ფანჯრას. მოვნიშნავთ საჭირო ფაილს და ვაჭერთ Open კლავიშს. მონიშნული ფაილის სახელი მიენიჭება FailisGaxsna ობიექტის FileName თვისებას.

```
{
//   პროგრამა 10.20
//   პროგრამაში ხდება ფაილიდან წაკითხვის დემონსტრირება Open დიალოგური
//   ფანჯრის საშუალებით
label1.Text = "";
FileStream file_in;
string striqoni;
OpenFileDialog FailisGaxsna = new OpenFileDialog();
if ( FailisGaxsna.ShowDialog() == DialogResult.OK )
{
    file_in = new FileStream(FailisGaxsna.FileName, FileMode.Open );
    StreamReader file_stream_in = new StreamReader(file_in);
    for ( ; ( striqoni = file_stream_in.ReadLine() ) != null; )
        label1.Text += striqoni + "\n";
    file_stream_in.Close();
}
}
```

ქვემოთ მოყვანილ პროგრამაში ხდება ფაილში მონაცემების ჩაწერის დემონსტრირება Save დიალოგური ფანჯრის გამოყენებით.

```
{
//   პროგრამა 10.21
//   პროგრამაში ხდება ფაილში ჩაწერის დემონსტრირება Save დიალოგური
//   ფანჯრის საშუალებით
string striqoni;
FileStream file_out;
SaveFileDialog FailisGaxsna = new SaveFileDialog();
if ( FailisGaxsna.ShowDialog() == DialogResult.OK )
{
    file_out = new FileStream(FailisGaxsna.FileName, FileMode.Create);
    StreamWriter file_stream_out = new StreamWriter(file_out);
    striqoni = textBox1.Text;
    //   ფაილში striqoni სტრიქონის ჩაწერა
    file_stream_out.Write(striqoni);
    file_stream_out.Close();
}
}
```

კატალოგების ხეზე ძებნა

კატალოგების ხეზე ფაილებისა და კატალოგების ძებნის შესასრულებლად რეკურსია გამოიყენება. მოყვანილი პროგრამით ხდება ამის დემონსტრირება:

```
public static void ShowDirectory(DirectoryInfo dirinfo, int interval, ListBox lb)
```

```

{
// კატალოგის სახელის გამოტანა, რომელსაც
// დასაწყისში ექნება 2 * interval ინტერვალი
string intervalebi = new String(' ', 2 * interval);
lb.Items.Add(intervalebi.ToString() + dirinfo.Name + '\n');

// ქვეკატალოგების მიღება და ამ მეთოდის რეკურსიული გამოძახება
foreach (DirectoryInfo diChild in dirinfo.GetDirectories())
    ShowDirectory(diChild, interval + 1, lb);
}

```

```

private void button1_Click(object sender, EventArgs e)
{
    DirectoryInfo dirinfo = new DirectoryInfo("D:\\");
    ShowDirectory(dirinfo, 0, listBox1);
}

```

თუ კატალოგების რაოდენობა ძალიან დიდია, მაშინ listBox1 კომპონენტში მათ გამოტანას შეიძლება რამდენიმე წუთი დასჭირდეს.

NetworkStream კლასი

ეს კლასი გამოცხადებულია System.Net.Sockets სახელების სივრცეში. ის პროცესს ქსელის საშუალებით უგზავნის და მისგან იღებს ბაიტებს. პროცესი შეიძლება მუშაობდეს ქსელის ნებისმიერ კომპიუტერზე და შეიძლება, აგრეთვე, წარმოადგენდეს ფაილს ან მონაცემების სხვა სათავსოს.

ქსელური შეერთებები ეფუძნება *სოკეტებს*. სოკეტი შეიცავს ჰოსტის (კომპიუტერის) მისამართს და პორტის ნომერს. ჰოსტის მისამართი შეიძლება იყოს TCP/IP მისამართი ან ჰოსტის სახელი. რაც შეეხება პორტის ნომერს, ისინი სამ ჯგუფად იყოფა:

- 0÷1023 პორტები დაკავებული აქვს ცნობილ სამსახურებს. მაგალითად, მე-80 პორტი გამოიყენება მონაცემების გადასაცემად HTTP პროტოკოლის საშუალებით.
- 1024÷49151 პორტები დაკავებული აქვს ნაკლებად ცნობილ სამსახურებს.
- 49152÷65536 პორტები განკუთვნილია მომხმარებლების მიერ შემუშავებული პროგრამა-დანართებისთვის და დინამიური გამოყენებისთვის.

იმისათვის, რომ ორი პროგრამა ერთმანეთს დაუკავშირდეს ქსელის საშუალებით, თითოეულმა მათგანმა უნდა შექმნას სოკეტი, რომელსაც მიმართავს მეორე პროგრამა. ამ მიზნისათვის შეგვიძლია გამოვიყენოთ Socket კლასი, რომელიც განსაზღვრულია System.Net.Sockets სახელების სივრცეში. TopListener კლასი ხსნის სოკეტს და ელოდება მასთან კლიენტის მიერთებას. TopClient კლასი ხსნის სოკეტს და უერთდება სამსახურს დაშორებულ კომპიუტერზე.

მოყვანილი პროგრამა ახდენს სერვერის რეალიზებას. სერვერი ელოდება კლიენტის მიერთებას. კლიენტის მიერთების შემთხვევაში სერვერი მას გადასცემს ბუფერში მოთავსებულ მონაცემებს. პროგრამაში გამოიყენება IPAddress კლასი, რომელიც განსაზღვრულია System.Net სახელების სივრცეში. ამიტომ, using დირექტივების ბლოკში უნდა დავუმატოთ using System.Net; და using System.Net.Sockets; დირექტივები.

```

// პროგრამა 10.22
// პროგრამაში ხდება სერვერის რეალიზება

```

```

class Serveri
{
//      Listen მეთოდი ელოდება შეერთებას
public void Listen()
{
IPAddress Lokaluri_Host = IPAddress.Parse("127.0.0.1");
//      50025 პორტის მოსმენა
TcpListener obj_tcp = new TcpListener(Lokaluri_Host, 50025);
obj_tcp.Start();
//      კლიენტის შეერთების ლოდინი
for ( ; ; )
{
//      კლიენტის სოკეტთან შეერთება
Socket Axali_Socketi = obj_tcp.AcceptSocket();
if ( Axali_Socketi.Connected )
{
//      NetworkStream ობიექტის შექმნა სოკეტის გახსნისათვის
System.Net.Sockets.NetworkStream Qseluri_Nakadi = new
System.Net.Sockets.NetworkStream(Axali_Socketi);

//      მონაცემების გადაგზავნა
byte[] buferi = { ( byte ) 's', ( byte ) 'a', ( byte ) 'b', ( byte ) 'a' };
Qseluri_Nakadi.Write(buferi, 0, 4);
Qseluri_Nakadi.Flush();
Qseluri_Nakadi.Close();
}
//      რესურსების გათავისუფლება
Axali_Socketi.Close();
}
}
}
private void button1_Click(object sender, EventArgs e)
{
//      შესრულების ნაკადის გაშვება, რომელიც პორტს უსმენს
Serveri Serveri_1 = new Serveri();
Serveri_1.Listen();
}

//      TcpListener კლასის Start() მეთოდი იწყებს პორტის მოსმენას ქსელში ახალი შეერთების
//      მოლოდინში, AcceptSocket() მეთოდი კი - ასრულებს კლიენტის რეალურ შეერთებას, ანუ ქმნის
//      ახალ სოკეტს შეერთებისათვის. შექმნილ სოკეტს გამოვიყენებთ კლიენტისათვის მონაცემების
//      გადასაგზავნად. მოყვანილი პროგრამა ახდენს კლიენტის რეალიზებას.
//      პროგრამა 10.23
//      პროგრამაში ხდება კლიენტის რეალიზება
{
label1.Text = "";
//      კლიენტის სოკეტის შექმნა
TcpClient Klientis_Soketi = new TcpClient("127.0.0.1", 50025);
//      NetworkStream ობიექტის შექმნა ჰოსტიდან მონაცემების წასაკითხად

```

```

System.Net.Sockets.NetworkStream Qseluri_Nakadi = Klientis_Soketi.GetStream();
//      ნაკადიდან ბაიტების მასივის წაკითხვა
byte[] buferi = new byte[100];
Qseluri_Nakadi.Read(buferi, 0, 100);
//      ბაიტების გარდაქმნა სიმბოლოებად და ეკრანზე გამოტანა
char[] buferi2 = new char[100];
for ( int i = 0 ; i < 100 ; i++ ) buferi2[i] = ( char ) buferi[i];
for ( int i = 0 ; i < 100 ; i++ ) label1.Text += buferi2[i].ToString();
//      ნაკადის დახურვა
Qseluri_Nakadi.Close();
}

```

როგორც ვხედავთ, TcpClient კლასის კონსტრუქტორს გადაეცემა ორი პარამეტრი: ჰოსტის მისამართი ან სახელი და პორტის ნომერი. კონსტრუქტორი ცდილობს დაუკავშირდეს შესაბამის სოკეტს ქსელში. შეერთების დამყარების შემდეგ ვიყენებთ GetStream() მეთოდს, რომელიც გასცემს NetworkStream კლასის ობიექტს. ამის შემდეგ, ვიყენებთ NetworkStream კლასის Read() მეთოდს სოკეტიდან მონაცემების მისაღებად.

ახლა, ჯერ გავუშვათ Serveri პროგრამა. ის დაელოდება კლიენტის მხრიდან შეერთებას. შემდეგ გავუშვათ Clienti პროგრამა. კლიენტი მიუერთდება სერვერს და label1 კომპონენტში გამოჩნდება სტრიქონი „საბა“. ამის შემდეგ, სერვერი და კლიენტი მუშაობას ამთავრებენ. თუ ჯერ გავუშვებთ კლიენტს და შემდეგ სერვერს, მაშინ გაიცემა შეტყობინება შეცდომის შესახებ (აღიძვრება SocketException განსაკუთრებული სიტუაცია), რადგან ვერ იქნება მოძებნილი სერვერი, რომელიც უსმენს 50025 პორტს.

MemoryStream და BufferedStream კლასები

MemoryStream და BufferedStream კლასების გამოყენება მონაცემების გადასაგზავნად ზრდის კოდის ეფექტურობას. საქმე ის არის, რომ ჩვენ იშვიათად გვიწევს მონაცემების იმ რაოდენობასთან მუშაობა, რომელსაც ოპერაციული სისტემა ეფექტურად დაამუშავებს. დავუშვათ, ფაილიდან გვინდა წავიკითხოთ 6 ბაიტი. ოპერაციული სისტემა ერთი ოპერაციის შედეგად რეალურად წავიკითხავს 4, 8 ან მეტ კილობაიტს. თუ გამოვიყენებთ BufferedStream კლასს, მაშინ ოპერაციული სისტემა მონაცემებს წარმოგვიდგენს ბლოკების სახით, რაც უზრუნველყოფს მაღალ მწარმოებლურობას. BufferedStream კლასი პროგრამას გადასცემს მხოლოდ იმ მონაცემებს, რომლებიც მოთხოვნილი იყო. გარდა ამისა, BufferedStream კლასი ოპერატიულ მეხსიერებაში შეიძლება ინახავდეს ჩაწერის რამდენიმე ოპერაციის შედეგს და ისინი მაშინ ჩაწეროს დისკზე, როცა ჩაწერა ყველაზე ეფექტური იქნება. ასეთი შიდა ბუფერირების გამო უნდა გამოვიყენოთ Flush() მეთოდი BufferedStream ობიექტის შემცველობის რეალურად ჩასაწერად მასთან დაკავშირებული მონაცემების სათავსოში.

MemoryStream კლასის ნაკადისათვის მონაცემების სათავსოა ოპერატიული მეხსიერების უბანი. მოყვანილ პროგრამაში ხდება მონაცემების გადაცემის დემონსტრირება.

```

{
//      პროგრამა 10.24
//      პროგრამაში ხდება მონაცემების გადაცემის დემონსტრირება
//      MemoryStream Write მეთოდი ინახავს MemoryStream ობიექტის
//      შემცველობას ფაილის სახით
public static void MemoryStreamWrite(System.IO.MemoryStream Nakadi, string FailisSaxeli)
{

```

```

FileStream Gamosasvleli_Nakadi = File.OpenWrite(FailisSaxeli);
Nakadi.WriteTo(Gamosasvleli_Nakadi);
Gamosasvleli_Nakadi.Flush();
Gamosasvleli_Nakadi.Close();
}
private void button1_Click(object sender, EventArgs e)
{
string failis_saxeli = "mexsiereba";
// ფაილის წაკითხვა MemoryStream ობიექტში
FileStream Shesasvleli_Nakadi = File.OpenRead(failis_saxeli + ".txt");
System.IO.MemoryStream Mexsierebis_Nakadi = new System.IO.MemoryStream();
// მონაცემების გადაცემა სათავსოში
Mexsierebis_Nakadi.SetLength(Shesasvleli_Nakadi.Length);
Shesasvleli_Nakadi.Read(Mexsierebis_Nakadi.GetBuffer(), 0, (int)Shesasvleli_Nakadi.Length);
// ჩაწერის კორექტულად დამთავრება და რესურსების გათავისუფლება
Mexsierebis_Nakadi.Flush();
Shesasvleli_Nakadi.Close();
// სათავსოს გადაცემა მეთოდისათვის ჩაწერის მიზნით
MemoryStream Write(Mexsierebis_Nakadi, failis_saxeli + ".bak");
Mexsierebis_Nakadi.Close();
}

```

MemoryStream კლასის ავტომატური კონსტრუქტორი ქმნის გაფართოებად ბუფერს, რომლის საწყისი ტევადობა ნულის ტოლია. ამ ბუფერის ზომა შეგვიძლია გავზარდოთ ამავე კლასის SetLength() მეთოდის გამოყენებით. ბუფერთან პირდაპირი მიმართვისათვის გამოიყენება GetBuffer() მეთოდი.

მონაცემების გადაგზავნა შესაძლებელია, აგრეთვე, BufferedStream კლასის გამოყენებით. მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

```

// პროგრამა 10.25
// პროგრამაში ხდება მონაცემების გადაცემის დემონსტრირება
{
string failis_saxeli = "buferi";
// საფაილო ნაკადების შექმნა
FileStream Shesasvleli_Faili = File.OpenRead(failis_saxeli + ".txt");
FileStream Gamosasvleli_Faili = File.OpenWrite(failis_saxeli + ".bak");
// ბუფერირების დამატება
BufferedStream Shemavali_Nakadi = new BufferedStream(Shesasvleli_Faili);
BufferedStream Gamomavali_Nakadi = new BufferedStream(Gamosasvleli_Faili);
byte[] buferi = new byte[4096];
int WakitxuliBaitebisRaodenoba;
// მონაცემების გადაწერა შესასვლელი ნაკადიდან გამოსასვლელ ნაკადში
// ბუფერირების გამოყენებით
for ( ; ( WakitxuliBaitebisRaodenoba = Shemavali_Nakadi.Read(buferi, 0, 4096) ) > 0; )
Gamomavali_Nakadi.Write(buferi, 0, WakitxuliBaitebisRaodenoba);
// გადაწერის კორექტულად დამთავრება და რესურსების გათავისუფლება
Gamomavali_Nakadi.Flush();
Gamomavali_Nakadi.Close();
}

```

```
Shemavali_Nakadi.Close();
Gamosasvleli_Faili.Close();
Shesasvleli_Faili.Close();
}
```

როგორც პროგრამიდან ჩანს, BufferedStream კლასის კონსტრუქტორს პარამეტრად გადაეცემა FileStream კლასის ობიექტი (ნაკადი). მიღებული ობიექტები გამოიყენება მონაცემების წასაკითხად და ჩასაწერად.

CryptoStream კლასს მე-16 თავში განვიხილავთ.

მონაცემების ასინქრონული შეტანა-გამოტანა

ზემოთ განხილულ მაგალითებში ყველგან გამოიყენებოდა მონაცემების სინქრონული შეტანა-გამოტანა. ამ შემთხვევაში, პროგრამის შესრულება დროებით ჩერდება შეტანა-გამოტანის ოპერაციის დამთავრებამდე. ბუნებრივია, ეს იწვევს პროგრამების არასასურველ დაყოვნებებსა და მოცდენებს, განსაკუთრებით იმ შემთხვევებში, როცა პროგრამა კითხულობს ან წერს მონაცემების დიდ მასივს. ხშირ შემთხვევაში, პროგრამას შეუძლია შესრულების გაგრძელება მონაცემების შეტანა-გამოტანის ოპერაციის პარალელურად.

მონაცემების ასინქრონული შეტანა-გამოტანის შემთხვევაში შეტანა-გამოტანის ოპერაციების შესრულებისათვის შესრულების ცალკე ნაკადი იქმნება (შესრულების ნაკადებს მე-15 თავში განვიხილავთ). შეტანა-გამოტანის ოპერაციის დამთავრებისთანავე უკუგამომდახების ფუნქციის მიერ პროგრამა იღებს შესაბამის შეტყობინებას. უკუგამომდახების ფუნქცია გამოიძახება დელეგატის მიერ შეტანა-გამოტანის ოპერაციის დამთავრებისთანავე.

მოყვანილ რეპროგრამაში ხდება ასინქრონული შეტანა-გამოტანის დემონსტრირება.

```
// პროგრამა 10.26
// პროგრამაში ხდება მონაცემების ასინქრონული
// შეტანა-გამოტანის დემონსტრირება
public class Asinqronuli
{
// წაკითხვის ნაკადი
public static FileStream ShesasvleliNakadi;
// დელეგატი უკუგამომდახების ფუნქციის აღწერისათვის
public static AsyncCallback Delegati;
// დიდი ზომის ბუფერის გამოყოფა მონაცემების წაკითხვისათვის
public static byte[] buferi = new byte[500000];
// უკუგამომდახების ფუნქცია, რომელიც გამოიძახება კითხვის დამთავრებისას
public static void Damtavreba(IAsyncResult asyncResult)
{
int bytesRead = ShesasvleliNakadi.EndRead(asyncResult);
MessageBox.Show("წაკითხულია " + bytesRead.ToString() + " ბაიტი");
}
}
private void button1_Click(object sender, EventArgs e)
{
label1.Text = "";
string failis_saxeli = "asincronuli.txt";
// ფაილის გახსნა
```



```

Asinqronuli.ShesasvleliNakadi = new FileStream(failis_saxeli, FileMode.Open, FileAccess.Read,
FileShare.None, 2048, true);
// დელეგატისთვის უკუგამომდახების ფუნქციის მინიჭება
Asinqronuli.Delegati = new AsyncCallback(Asinqronuli.Damtavreba);
// ასინქრონული წაკითხვა
Asinqronuli.ShesasvleliNakadi.BeginRead(Asinqronuli.buferi, 0, 500000, Asinqronuli.Delegati, null);
// პარალელური გამოთვლები
for (int i = 0; i < 500; i++)
    label1.Text += i.ToString();
}

```

როგორც პროგრამიდან ჩანს, გამოყენებულია FileStream() კონსტრუქტორის გადატვირთული ვერსია, რომელსაც ექვსი პარამეტრი აქვს. პირველი სამი პარამეტრის დანიშნულება ჩვენთვის ცნობილია. მეოთხე პარამეტრია FileShare, რომელიც მიუთითებს ფაილთან მუშაობა შესრულდება ერთობლივი თუ ექსკლუზიური გამოყენების რეჟიმში. მეხუთე პარამეტრი განსაზღვრავს შიდა ბუფერის ზომას. მეექვსე პარამეტრი მიუთითებს ფაილი გახსნილი იქნება თუ არა ასინქრონულ რეჟიმში სამუშაოდ. იმისათვის, რომ შევამოწმოთ ოპერაციული სისტემა უზრუნველყოფს თუ არა ასინქრონულ შეტანა-გამოტანას უნდა შევამოწმოთ FileStream ობიექტის IsAsync თვისება:

```

if ( Asinqronuli.ShesasvleliNakadi.IsAsync == true )
    label2.Text = "სისტემა უზრუნველყოფს ასინქრონულ შეტანა-გამოტანას";
else label2.Text = "სისტემა არ უზრუნველყოფს ასინქრონულ შეტანა-გამოტანას";

```

პროგრამაში ფაილის გახსნის შემდეგ იქმნება Delegati დელეგატი, რომელიც შეტანა-გამოტანის ოპერაციის დამთავრების შემდეგ გამოიძახებს Damtavreba() მეთოდს. შემდეგ იწყება მონაცემების ასინქრონული კითხვა, რომელსაც BeginRead() მეთოდი ასრულებს. ეს მეთოდი ქმნის შესრულების ახალ ნაკადს, რომელშიც შესრულდება კითხვის ოპერაცია. კითხვის დამთავრების შემდეგ BeginRead() მეთოდი გამოიძახებს უკუგამომდახების Damtavreba() მეთოდს. ამ მეთოდის კოდში გამოყენებული EndRead() მეთოდი გასცემს წაკითხული ბაიტების რაოდენობას. BeginRead() მეთოდის მუშაობის პარალელურად სრულდება მის შემდეგ მოთავსებული კოდი, კერძოდ კი for ციკლი.

ბოლოს, შევნიშნოთ, რომ თუ ფაილი გახსნილია სინქრონული მიმართვისათვის, მაშინ ასინქრონული შეტანა-გამოტანის მეთოდები მასთან მაინც სინქრონულ რეჟიმში იმუშავებენ.

თავი 11. განსაკუთრებული სიტუაციები

განსაკუთრებული სიტუაციების დამუშავების საფუძვლები

განსაკუთრებული სიტუაცია არის სიტუაცია, რომელიც გამოწვეულია პროგრამის შესრულების დროს აღძრული შეცდომის მიერ. ასეთი სიტუაციებია: ნულზე გაყოფა, გადავსება, მასივის ინდექსის გასვლა დიაპაზონის გარეთ, არარსებული ფაილის გახსნის მცდელობა და ა.შ. ეს შეცდომები იწვევენ პროგრამის შესრულების ავარიულად დამთავრებას. C# ენაში არსებობს განსაკუთრებული სიტუაციების დამუშავების საშუალებები, რომელთა გამოყენებით შესაძლებელია ასეთი შეცდომების დამუშავება ისე, რომ არ შეფერხდეს პროგრამის შესრულება.

C# ენაში განსაკუთრებული სიტუაციები წარმოდგენილია კლასებით. განსაკუთრებული სიტუაციების ყველა კლასი მემკვიდრეობით არის მიღებული განსაკუთრებული სიტუაციის Exception კლასისაგან, რომელიც წარმოადგენს System სახელების სივრცის ნაწილს. C# ენაში განსაზღვრულია განსაკუთრებული სიტუაციების ორი კატეგორია: გენერირებული C# ენის შესრულების გარემოს (CLR) მიერ და გენერირებული პროგრამა-დანართების (Application) მიერ. პირველი მათგანი აღწერილია System.Exception კლასში, მეორე კი - System.ApplicationException კლასში.

ცხრილი 11.1. ხშირად აღძრული განსაკუთრებული სიტუაციები.

განსაკუთრებული სიტუაცია	მნიშვნელობა
ArrayTypeMismatchException	მნიშვნელობის ტიპი შეუთავსებია მასივის ტიპთან
Divide ByZeroException	ნულზე გაყოფის შეცდომა
IndexOutOfRangeException	მასივის ინდექსი გადის დიაპაზონის გარეთ
InvalidCastException	არაკორექტული გარდაქმნა პროგრამის შესრულების პროცესში
OutOfMemoryException	new ოპერატორის გამოძახება იყო არაკორექტული მეხსიერების უკმარისობის გამო
OverflowException	გადავსება არითმეტიკული ოპერაციის შესრულების დროს
StackOverflowException	სტეკის გადავსება

ცხრილი 11.2. Exception კლასის თვისებები.

თვისება	ტიპი	მნიშვნელობა
HelpLink	string	ამ თვისებას შეიძლება მივანიჭოთ ფაილის სახელი ან წებ-მისამართი, რომელიც შეიცავს დამატებით ინფორმაციას განსაკუთრებული სიტუაციის შესახებ
Message	string	შეიცავს განსაკუთრებული სიტუაციის აღწერას
Source	string	შეიცავს განსაკუთრებული სიტუაციის გამომწვევი პროგრამის სახელს
StackTrace	string	შეიცავს განსაკუთრებული სიტუაციის გამომწვევი მეთოდისა და კლასის სახელს
TargetSite	MethodBase	შეიცავს იმ მეთოდის სახელს, რომლიდანაც იყო გამოძახებული განსაკუთრებული სიტუაცია

System სახელების სივრცეში განსაზღვრულია ბევრი სტანდარტული ჩადგმული განსაკუთრებული სიტუაცია. ცხრილში 11.1 მოყვანილია ხშირად აღძრული განსაკუთრებული სიტუაციები. Exception კლასის თვისებები მოყვანილია ცხრილში 11.2.

განსაკუთრებული სიტუაციების დამუშავება სრულდება ოთხი საკვანძო სიტყვის გამოყენებით: try, catch, throw და finally. ისინი, ხშირ შემთხვევაში, ერთობლივად გამოიყენება. try ბლოკში მოთავსებულია პროგრამის ის ოპერატორები, რომელთა შესრულებასაც თვალყური უნდა ვადევნოთ. გენერირებულ განსაკუთრებულ სიტუაციას იჭერს და ამუშავებს catch ბლოკი. throw სიტყვა იწვევს განსაკუთრებული სიტუაციის ხელოვნურად გენერირებას. finally ბლოკში მოთავსებულია კოდი, რომელიც ყოველთვის შესრულდება try ბლოკიდან გამოსვლისას.

try და catch ბლოკები

განსაკუთრებული სიტუაციების დამუშავება ეფუძნება try და catch ბლოკების გამოყენებას. ეს ბლოკები ერთობლივად მუშაობენ. მათი სინტაქსია:

```
try
{
// კოდი, რომლისთვისაც სრულდება შეცდომების მონიტორინგი
}
catch ( ტიპი_1 ობიექტი )
{
// ExceptType1 განსაკუთრებული სიტუაციის დამუშავება
}
catch ( ტიპი_2 ობიექტი )
{
// ExceptType2 განსაკუთრებული სიტუაციის დამუშავება
...
}
```

როგორც სინტაქსიდან ჩანს, შესამოწმებელი კოდი მოთავსებული უნდა იყოს ფიგურულ ფრჩხილებში. ტიპი პარამეტრები განსაზღვრავენ დასამუშავებელი განსაკუთრებული სიტუაციების ტიპებს. როცა catch ოპერატორი იჭერს განსაკუთრებული სიტუაციას, მაშინ შესაბამისი ობიექტი იღებს მის მნიშვნელობას. თუ განსაკუთრებული სიტუაციის დამამუშავებელი არ მიმართავს ობიექტს, რაც პრაქტიკულად ხშირად ხდება, მაშინ არ არის აუცილებელი მისი მითითება.

განსაკუთრებული სიტუაციის გენერირების შემდეგ მისი დაჭერა ხდება შესაბამისი catch ოპერატორის მიერ, რომელიც ახდენს ამ განსაკუთრებული სიტუაციის დამუშავებას. try ბლოკთან შეიძლება დაკავშირებული იყოს რამდენიმე catch ოპერატორი. განსაკუთრებული სიტუაციის ტიპი განსაზღვრულია catch ოპერატორში. თუ გენერირებული განსაკუთრებული სიტუაცია შეესაბამება catch ოპერატორში მითითებულ ტიპს, მაშინ ის დაიჭერს ამ განსაკუთრებულ სიტუაციას და შესრულდება ამ catch ოპერატორის კოდი. დანარჩენი catch ბლოკები იგნორირდება.

იმ შემთხვევაში თუ განსაკუთრებული სიტუაციის გენერირება არ ხდება, მაშინ try ბლოკის ოპერატორების მუშაობა ჩვეულებრივი რიგითობით მთავრდება და მისი შესაბამისი catch ოპერატორები არ შესრულდება. პროგრამის შესრულება გაგრძელდება უკანასკნელი catch ოპერატორის შემდეგ მოთავსებული ოპერატორიდან. ამრიგად, catch ოპერატორები გამოიძახება მხოლოდ განსაკუთრებული სიტუაციების გენერირების შემთხვევაში.

როგორც ცნობილია, როცა ხდება ინდექსთან მიმართვა, რომელიც გადის მასივის საზღვრებს გარეთ, აღიბვრება შეცდომა. ამ დროს გენერირდება განსაკუთრებული სიტუაცია.

მოყვანილი პროგრამა ახდენს ამ განსაკუთრებული სიტუაციის გენერირებისა და დაჭერის დემონსტრირებას.

```
{
//      პროგრამა 11.1
//      პროგრამაში ხდება მასივის საზღვრებს გარეთ ინდექსის გასვლის შეცდომის დამუშავება
int[] masivi = new int[5];
    try
    {
        label1.Text = "ეს სტრიქონი გამოჩნდება განსაკუთრებული სიტუაციის გენერირებამდე";
        masivi[6] = 20; // ინდექსის მნიშვნელობა გადის დიაპაზონის გარეთ
        label2.Text = "ეს სტრიქონი არ გამოჩნდება";
    }
//      განსაკუთრებული სიტუაციის დაჭერა
catch (IndexOutOfRangeException)
{
    label3.Text = "ინდექსი დიაპაზონის გარეთაა ";
}
    label4.Text = "ეს ოპერატორი შესრულდება";
}
```

ამ პროგრამაში განსაკუთრებული სიტუაციის გენერირების შემთხვევაში მისი დაჭერა ხდება catch ოპერატორის მიერ. ამ მომენტიდან დაწყებული მუშაობას იწყებს catch ბლოკი, ხოლო try ბლოკი კი მუშაობას ამთავრებს. შედეგად,

label3.Text = "ეს სტრიქონი არ გამოჩნდება";

ოპერატორი, რომელიც მოსდევს ინდექსის არასწორად გამოყენების ოპერატორს, არ შესრულდება. catch ბლოკის შესრულების შემდეგ მართვა გადაეცემა ოპერატორს, რომელიც მოსდევს ამ catch ბლოკს. ამრიგად, მიუხედავად აღძრული შეცდომისა, პროგრამა აგრძელებს მუშაობას და ავარიულად არ მთავრდება.

catch ოპერატორში შეიძლება არ იყოს მითითებული არც ერთი პარამეტრი. პარამეტრი მოითხოვება მხოლოდ მაშინ, როცა აუცილებელია განსაკუთრებული სიტუაციის ობიექტთან მიმართვა. რიგ შემთხვევებში განსაკუთრებული სიტუაციის ობიექტის მნიშვნელობა შეიძლება გამოყენებული იქნას განსაკუთრებული სიტუაციების დამამუშავებლის მიერ შეცდომის შესახებ დამატებითი ინფორმაციის მიღების მიზნით.

თუ არ მოხდა განსაკუთრებული სიტუაციის გენერირება, მაშინ catch ოპერატორი არ გამოიძახება და მართვა გადაეცემა catch ოპერატორის შემდეგ მყოფ ოპერატორს.

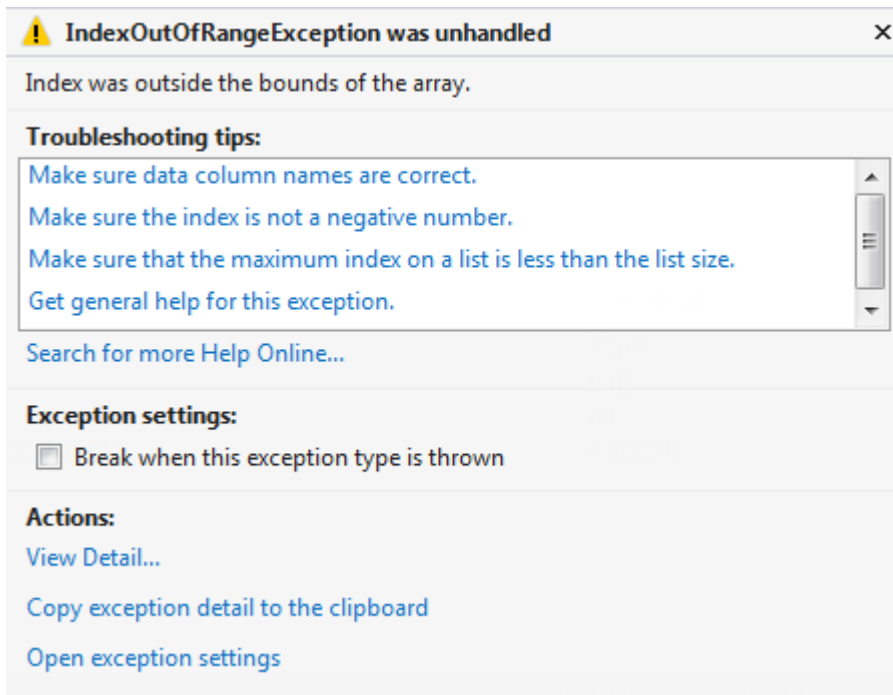
სტანდარტული განსაკუთრებული სიტუაციის დაჭერა და დამუშავება თავიდან აგვაცილებს პროგრამის ავარიულად დამთავრებას. თუ პროგრამა არ ახდენს განსაკუთრებული სიტუაციის დაჭერას, მაშინ განსაკუთრებული სიტუაციას დაიჭერს C#-ის შესრულების სისტემა. პრობლემა, რომელსაც ამ შემთხვევაში აქვს ადგილი, იმაში მდგომარეობს, რომ შესრულების სისტემას ეკრანზე გამოაქვს შეტყობინება შეცდომის შესახებ და ავარიულად ამთავრებს პროგრამის შესრულებას. ქვემოთ მოყვანილ პროგრამაში განსაკუთრებული სიტუაციის დაჭერა პროგრამის მიერ არ ხდება.

```
{
//      პროგრამა 11.2
//      პროგრამაში არ ხდება განსაკუთრებული სიტუაციის დაჭერა
int[] masivi = new int[5];
```

```
masivi[0] = 5; // ეს ოპერატორი შესრულდება განსაკუთრებული სიტუაციის გენერირებამდე
masivi[6] = 20; // ინდექსის მნიშვნელობა გადის დიაპაზონის გარეთ
```

```
}
```

ამ შემთხვევაში პროგრამის შესრულება ავარიულად შეწყდება და გამოჩნდება შეტყობინება შეცდომის შესახებ:



ნახ.11.1

მართალია, ეს შეტყობინება საკმაოდ სასარგებლოა პროგრამის გამართვისას, მაგრამ ის არ უნდა გამოჩნდეს პროგრამის ექსპლუატაციის დროს. ამიტომ, პროფესიულ დონეზე შედგენილმა პროგრამამ თვითონ უნდა დაამუშაოს განსაკუთრებული სიტუაციები.

როგორც ვიცით, განსაკუთრებული სიტუაციის ტიპი უნდა შეესაბამებოდეს catch ოპერატორში მითითებულ ტიპს. წინააღმდეგ შემთხვევაში, განსაკუთრებული სიტუაცია არ იქნება დაჭერილი. ქვემოთ მოყვანილ პროგრამაში ხდება ნულზე გაყოფით გამოწვეული განსაკუთრებული სიტუაციის (DivideByZeroException) დაჭერა. როგორც კი ინდექსის მნიშვნელობა გავა დიაპაზონის გარეთ, მოხდება IndexOutOfRangeException განსაკუთრებული სიტუაციის გენერირება, რომელსაც ეს catch ოპერატორი ვერ დაიჭერს. შედეგად, პროგრამა ავარიულად დაამთავრებს მუშაობას.

```
{  
//   პროგრამა 11.3  
//   პროგრამაში ხდება ნულზე გაყოფით გამოწვეული შეცდომის დაჭერა და  
//   არ ხდება დიაპაზონის გარეთ ინდექსის გასვლის შეცდომის დაჭერა  
int ricxvi;  
int[] masivi = new int[5];  
  
try  
{  
label1.Text = "ეს სტრიქონი გამოჩნდება განსაკუთრებული სიტუაციის გენერირებამდე";  
masivi[6] = 20;           //   ინდექსის მნიშვნელობა გადის დიაპაზონის გარეთ  
ricxvi = 10;             //   ეს ოპერატორი არ შესრულდება  
}  
//   განსაკუთრებული სიტუაციის დაჭერა  
catch ( DivideByZeroException )
```

```
{
label2.Text = "ინდექსი დიაპაზონის გარეთაა";
}
}
```

განსაკუთრებული სიტუაციების დამუშავების ერთ-ერთი მთავარი უპირატესობა ის არის, რომ პროგრამას შეუძლია მოახდინოს რეაგირება შეცდომაზე და შემდეგ გააგრძელოს შესრულება. ქვემოთ მოყვანილ პროგრამაში ერთი მასივის ელემენტები იყოფა მეორე მასივის ელემენტებზე. ნულზე გაყოფის შემთხვევაში გენერირდება DivideByZeroException განსაკუთრებული სიტუაცია. პროგრამა ახდენს მის დამუშავებას გასცემს რა შესაბამის შეტყობინებას. ამრიგად, ნულზე გაყოფის შეცდომა არ გამოიწვევს პროგრამის ავარიულ გაჩერებას. ამის ნაცვლად, label კომპონენტში გამოჩნდება შესაბამისი შეტყობინება.

```
{
//   პროგრამა 11.4
//   პროგრამაში ხდება ნულზე გაყოფის შეცდომის დაჭერა მასივების გაყოფის დროს
int[] masivi1 = { 1, 3, 5, 7, 9 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[5];

label1.Text = ""; label2.Text = "";
for ( int ind = 0; ind < masivi1.Length; ind++ )
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text += masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
catch ( DivideByZeroException )
{
label2.Text += "ადგილი აქვს ნულზე გაყოფას!" + '\n';
}
}
```

ამ პროგრამაში, try ბლოკი მოთავსებულია ციკლის ტანში, ამიტომ, შესაძლებელი ხდება გამეორებადი შეცდომების დამუშავება.

try და catch ბლოკების გამოყენების მაგალითები

მოყვანილ პროგრამაში სრულდება ფაილის გახსნისა და ფაილიდან წაკითხვის ოპერაციების შემოწმება.

```
{
//   პროგრამა 11.5
//   პროგრამაში ხდება ფაილის გახსნისა და ფაილიდან წაკითხვის ოპერაციების
//   მონიტორინგი
int ricxvi = 0;
FileStream file1;
//   try ბლოკი ამოწმებს ფაილის გახსნის ოპერაციას
try
```

```

{
    file1 = new FileStream("filetext.txt", FileMode.Open);
}
catch ( FileNotFoundException arg1 )
{
    label1.Text = arg1.Message;
    return;
}
// მონაცემების წაკითხვა ფაილიდან მანამ, სანამ არ შეგვხვდება EOF სიმბოლო
for ( ; ricxvi != -1; ) // თუ ricxvi = -1, მაშინ მიღწეულია ფაილის დასასრული
{
    // try ბლოკი ამოწმებს ფაილიდან წაკითხვის ოპერაციას
    try
    {
        ricxvi = file1.ReadByte();
    }
    catch ( IOException arg1 )
    {
        label2.Text = arg1.Message;
        return;
    }
    if ( ricxvi != -1 ) label3.Text += ( char ) ricxvi;
}
file1.Close();
}

```

მოყვანილ პროგრამაში სრულდება ფაილის შექმნისა და ფაილში ჩაწერის ოპერაციების შემოწმება.

```

{
// პროგრამა 11.6
// პროგრამაში ხდება ფაილის შექმნისა და ფაილში ჩაწერის ოპერაციების მონიტორინგი
BinaryReader file_in;
BinaryWriter file_out;
int ricxvii1 = 10, ricxvii2;
double wiladi1 = 1001.47, wiladi2, wiladi3;
bool b1 = true, b2;
// try ბლოკი ამოწმებს ფაილის შექმნის ოპერაციას
try
{
file_out = new BinaryWriter(new FileStream("file1.dat", FileMode.Create));
}
catch ( IOException arg1 )
{
MessageBox.Show(arg1.Message + "\n შეუძლებელია ფაილის შექმნა");
return;
}
// try ბლოკი ამოწმებს ფაილში ჩაწერის ოპერაციებს
try

```

```

{
//      ფაილში მთელი რიცხვის ჩაწერა
file_out.Write(ricxvii1);
//      ფაილში წილადის ჩაწერა
file_out.Write(wiladi1);
//      ფაილში ლოგიკური მონაცემის ჩაწერა
file_out.Write(b1);
//      ფაილში იწერება 10.2 * 2.3 გამოსახულების გამოთვლის შედეგი
file_out.Write(10.2 * 2.3);
}
catch ( IOException arg1 )
{
MessageBox.Show(arg1.Message + "\n ჩაწერის შეცდომა");
return;
}
file_out.Close();
//      ფაილიდან წაკითხვა
try
{
file_in = new BinaryReader(new FileStream("file1.dat",FileMode.Open));
}
catch ( IOException arg1 )
{
MessageBox.Show(arg1.Message + "\n შეუძლებელია ფაილის გახსნა");
return;
}
try
{
//      მთელი რიცხვის წაკითხვა ფაილიდან
ricxvii2 = file_in.ReadInt32();
label1.Text = ricxvii2.ToString();
//      წილადის წაკითხვა ფაილიდან
wiladi2 = file_in.ReadDouble();
label2.Text = wiladi2.ToString();
//      ლოგიკური მონაცემის წაკითხვა ფაილიდან
b2 = file_in.ReadBoolean();
label3.Text = b2.ToString();
//      წილადის წაკითხვა ფაილიდან
wiladi3 = file_in.ReadDouble();
label4.Text = wiladi3.ToString();
}
catch ( IOException arg1 )
{
MessageBox.Show(arg1.Message + "\n წაკითხვის შეცდომა");
return;
}
file_in.Close();

```



```

}
    მოყვანილ პროგრამაში სრულდება ფაილის შექმნისა და ფაილში ჩაწერა-წაკითხვის
ოპერაციების შემოწმება.
{
//    პროგრამა 11.7
//    პროგრამაში ხდება ფაილის შექმნისა და ფაილში ჩაწერა-წაკითხვის ოპერაციების
//    მონიტორინგი
label1.Text = "";
FileStream file;
//    try ბლოკი ამოწმებს ფაილის შექმნის ოპერაციას
try
{
file = new FileStream("file.dat", FileMode.Create);
}
catch ( FileNotFoundException arg1 )
{
label2.Text = arg1.Message;
return;
}
char simbolo;
//    ასოების ჩაწერა ანბანის მიხედვით
for ( int i = 0; i < 26; i++ )
{
//    try ბლოკი ამოწმებს ფაილში ჩაწერის ოპერაციას
try
{
file.WriteByte((byte) ('a' + i));
}
catch ( IOException arg1 )
{
label2.Text = arg1.Message;
return;
}
}
//    try ბლოკი ამოწმებს ფაილიდან წაკითხვის ოპერაციას
try
{
file.Seek(0, SeekOrigin.Begin);           //    პირველი ბაიტის არჩევა
simbolo = (char) file.ReadByte();         //    პირველი ბაიტის წაკითხვა
label1.Text += "პირველი მნიშვნელობა  " + simbolo + '\n';
file.Seek(4, SeekOrigin.Begin);         //    მეხუთე ბაიტის არჩევა
simbolo = ( char ) file.ReadByte();     //    მეხუთე ბაიტის წაკითხვა
label1.Text += "მეხუთე მნიშვნელობა  " + simbolo + '\n';
}
catch ( IOException arg1 )
{
label3.Text = arg1.Message;
}
}

```

```

return;
}
file.Close();
}

```

რამდენიმე catch ბლოკის გამოყენება

ერთ try ბლოკთან შეიძლება დაკავშირებული იყოს რამდენიმე catch ბლოკი. მაგრამ, თითოეული catch ბლოკი იჭერს კონკრეტული ტიპის განსაკუთრებული სიტუაციას. მოყვანილ პროგრამაში ხდება ინდექსის მნიშვნელობის დიაპაზონის გარეთ გასვლისა და ნულზე გაყოფის შეცდომების დაჭერა.

```

{
//   პროგრამა 11.8
//   პროგრამაში ხდება რამდენიმე catch ოპერატორის გამოყენების დემონსტრირება
label1.Text = "";
int[] masivi1 = { 1, 3, 5, 7, 9, 11, 13 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[7];

for ( int ind = 0; ind < masivi1.Length; ind++ )
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text += masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
//   ეს catch ოპერატორი იჭერს ნულზე გაყოფის შეცდომას
catch ( DivideByZeroException )
{
label2.Text = "ადგილი აქვს ნულზე გაყოფას!";
}
//   ეს catch ოპერატორი იჭერს მასივის ინდექსის დიაპაზონის გარეთ გასვლის შეცდომას
catch ( IndexOutOfRangeException )
{
label3.Text = "ინდექსი დიაპაზონის გარეთაა!";
}
}

```

catch ოპერატორები მოწმდება იმ მიმდევრობით, როგორც არიან პროგრამაში მითითებული. შესრულდება ის ბლოკი, რომლის catch ოპერატორში მითითებული ტიპი დაემთხვევა განსაკუთრებული სიტუაციის ტიპს. დანარჩენი catch ბლოკები არ შესრულდება.

ყველა განსაკუთრებული სიტუაციის დაჭერა

ზოგჯერ საჭიროა ყველა განსაკუთრებული სიტუაციის დაჭერა მათი ტიპის მიუხედავად. ასეთ შემთხვევაში, გამოიყენება catch ოპერატორი პარამეტრების გარეშე. უპარამეტრებო catch

ოპერატორი იჭერს ყველა განსაკუთრებული სიტუაციას. მოყვანილ პროგრამაში IndexOutOfRangeException და DivideByZeroException განსაკუთრებული სიტუაციის დაჭერა და დამუშავება სრულდება ერთი catch ოპერატორის მიერ.

```
{
//   პროგრამა 11.9
//   პროგრამაში ხდება ყველა განსაკუთრებული სიტუაციის დაჭერა
label1.Text = "";
int[] masivi1 = { 1, 3, 5, 7, 9, 11, 13 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[7];

for ( int ind = 0; ind < masivi1.Length; ind++ )
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text += masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
//   ეს catch ოპერატორი იჭერს ყველა შეცდომას
catch
{
label2.Text = "ადგილი აქვს განსაკუთრებული სიტუაციის გენერირებას!";
}
}
```

ყველა განსაკუთრებული სიტუაციის დაჭერა შესაძლებელია აგრეთვე, catch ოპერატორში Exception ტიპის არგუმენტის გამოცხადებით. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
{
label1.Text = ""; label2.Text = "";
int[] masivi1 = { 1, 3, 5, 7, 9, 11, 13 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[7];

for ( int ind = 0; ind < masivi1.Length; ind++ )
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label2.Text += masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
catch ( Exception arg )
{
label1.Text += arg.Message + '\n';
}
}
```

ჩადგმული try ბლოკები

შესაძლებელია try ბლოკების ერთმანეთში მოთავსება. შიდა try ბლოკში გენერირებული განსაკუთრებული სიტუაცია, რომელიც ვერ დაიჭირა ამ შიდა try ბლოკის შესაბამისმა catch ბლოკმა, ვრცელდება გარე try ბლოკზე. მოყვანილ მაგალითში გენერირებული IndexOutOfRangeException განსაკუთრებული სიტუაცია, რომელიც არ იყო დაჭერილი შიდა catch ბლოკის მიერ, დაჭერილი იქნება გარე catch ბლოკის მიერ.

```
{
//   პროგრამა 11.10
//   პროგრამაში ხდება ჩადგმული try ბლოკების გამოყენების დემონსტრირება
label1.Text = "";
int[] masivi1 = { 1, 3, 5, 7, 9, 11, 13 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[7];

try
{
for ( int ind = 0; ind < masivi1.Length; ind++ )
{
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text += masivi1[ind].ToString() + " / " +
                masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + "\n";
}
catch ( DivideByZeroException )
{
label2.Text = "ადგილი აქვს ნულზე გაყოფას!";
}
}
}
catch ( IndexOutOfRangeException )
{
label3.Text = "ადგილი აქვს ინდექსის გასვლას დიაპაზონის გარეთ!";
}
}
```

პროგრამაში ნულზე გაყოფის შედეგად გენერირებული განსაკუთრებული სიტუაცია მუშავდება შიდა try ბლოკში. ამასთან, პროგრამა აგრძელებს მუშაობას. დიაპაზონის გარეთ ინდექსის გასვლისას შეცდომას ხელში იგდებს გარე try ბლოკის შესაბამისი catch ოპერატორი.

ხშირად, ჩადგმული try ბლოკები გამოიყენება სხვადასხვა სახის შეცდომების დასამუშავებლად. გარე try ბლოკი შეგვიძლია გამოვიყენოთ სერიოზული შეცდომების დასაჭერად. ხოლო შიდა try ბლოკები ნაკლებად სერიოზული შეცდომების დასამუშავებლად. შეგვიძლია, აგრეთვე, გამოვიყენოთ გარე try ბლოკი უპარამეტრებო catch ოპერატორით იმ შეცდომების დასაჭერად, რომლებიც არ მუშავდებიან შიდა try ბლოკში.

განსაკუთრებული სიტუაციის იძულებით გენერირება

წინა მაგალითებში ხდებოდა C# სისტემის მიერ ავტომატურად გენერირებული განსაკუთრებული სიტუაციის დაჭერა. მაგრამ, განსაკუთრებული სიტუაციის გენერირება შეიძლება, აგრეთვე, **throw** ოპერატორის საშუალებით. მისი სინტაქსია:

throw ობიექტი:

აქ ობიექტი არის განსაკუთრებული სიტუაციის კლასის ობიექტი. მოყვანილ მაგალითში ხდება DivideByZeroException განსაკუთრებული სიტუაციის გენერირება throw ოპერატორის მიერ.

```
{
//      პროგრამა 11.11
//      პროგრამაში ხდება throw ოპერატორთან მუშაობის დემონსტრირება
int ricxvi1, ricxvi2, shedegi;
try
{
//      ეს ოპერატორები შესრულდება throw ოპერატორის წინ
ricxvi1 = int.Parse(textBox1.Text);
ricxvi2 = int.Parse(textBox2.Text);
//      განსაკუთრებული სიტუაციის გენერირება
if ( ricxvi2 == 0 ) throw new DivideByZeroException();
shedegi = ricxvi1 / ricxvi2;
label1.Text = shedegi.ToString();
}
catch ( DivideByZeroException )
{
label1.Text = "ადგილი აქვს ნულზე გაყოფას";
}
//      try-catch ბლოკის დასასრული
}
```

throw ოპერატორი ოპერირებს ობიექტებზე. ამიტომ, მის გამოყენებამდე საჭიროა ობიექტის შექმნა new ოპერატორით. ჩვენს შემთხვევაში, DivideByZeroException ობიექტის შესაქმნელად გამოიყენება ავტომატური კონსტრუქტორი.

მოყვანილ მაგალითში ხდება DivideByZeroException კლასის nulze_fayofa ობიექტის შექმნა და მისი მითითება throw ოპერატორის შემდეგ.

```
{
//      პროგრამა 11.12
//      პროგრამაში ხდება throw ოპერატორთან მუშაობის დემონსტრირება
int ricxvi1, ricxvi2, shedegi;
DivideByZeroException nulze_fayofa = new DivideByZeroException();
nulze_fayofa.HelpLink = "დამატებითი ინფორმაცია იხილეთ Readme.txt ფაილში";
nulze_fayofa.Source = "შეცდომის გამომწვევია - პროგრამა 11.12";
try
{
//      ეს ოპერატორები შესრულდება throw ოპერატორის წინ
ricxvi1 = int.Parse(textBox1.Text);
ricxvi2 = int.Parse(textBox2.Text);
//      განსაკუთრებული სიტუაციის გენერირება
if ( ricxvi2 == 0 ) throw nulze_fayofa;
shedegi = ricxvi1 / ricxvi2;
```

```

label1.Text = shedegi.ToString();
}
catch ( DivideByZeroException arg)
{
label1.Text = "განსაკუთრებული სიტუაცია დაჭერილია";
label2.Text = arg.HelpLink;
label3.Text = arg.Source;
}
}

```

finally ბლოკი

როგორც ვიცით, განსაკუთრებული სიტუაციის გენერირება იწვევს მიმდინარე მეთოდის შესრულების ავარიულ შეწყვეტას. მაგრამ, ამ მეთოდმა შეიძლება გახსნას ფაილი ან ქსელური შეერთება, რომლებიც უნდა იყოს დახურული. ასეთ შემთხვევებში უნდა გამოვიყენოთ **finally** ბლოკი. ის ეთითება try/catch მიმდევრობის ბოლოში. try/catch/finally კონსტრუქციის სინტაქსია:

```

try
{
//      try ბლოკის კოდი, რომლისთვისაც სრულდება შეცდომის მონიტორინგი
}
catch ( ტიპი_1 ობიექტი1)
{
//      ტიპი_1 განსაკუთრებული სიტუაციის დამუშავება
}
catch ( ტიპი_2 ობიექტი2)
{
//      ტიპი_2 განსაკუთრებული სიტუაციის დამუშავება
}
finally
{
finally ბლოკის კოდი
}

```

finally ბლოკი გამოიძახება იმისგან დამოუკიდებლად გენერირდება თუ არა განსაკუთრებული სიტუაცია. ამრიგად, არა აქვს მნიშვნელობა try ბლოკი როგორ დამთავრდება - ნორმალურად თუ აღიძვრება განსაკუთრებული სიტუაცია. finally ბლოკის კოდი ყოველთვის შესრულდება. მოყვანილ პროგრამაში ხდება finally ბლოკის გამოყენების დემონსტრირება.

```

//      პროგრამა 11.13
//      პროგრამაში ხდება finally ბლოკის გამოყენების დემონსტრირება
class finally1
{
public static void gen_finally(int par1)
{
int ricxvi;
int[] masivi = new int[5];

MessageBox.Show(par1.ToString() + "-ის მიღება");

```

```

try
{
switch ( par1 )
{
case 0 : ricxvi = 10 / par1;           //   ნულზე გაყოფის შეცდომა
        break;
case 1 : masivi[7] = 7;               //   დიაპაზონის გარეთ ინდექსის გასვლის შეცდომა
        break;
case 2 : return;                     //   try ბლოკიდან გამოსვლა
}
}
catch ( DivideByZeroException )
{
MessageBox.Show("ნულზე გაყოფის შეცდომა!");
return;
}
catch ( IndexOutOfRangeException )
{
MessageBox.Show(" დიაპაზონის გარეთ ინდექსის გასვლის შეცდომა!");
return;
}
finally
{
MessageBox.Show("try ბლოკიდან გამოსვლა");
}
}
private void button1_Click(object sender, System.EventArgs e)
{
for ( int indeqsi = 0; indeqsi < 3; indeqsi++ )
{
finally1.gen_finally(indeqsi);
}
}
}

```

საკუთარი განსაკუთრებული სიტუაციის შექმნა

ჩვენ შეგვიძლია შევქმნათ განსაკუთრებული სიტუაციების საკუთარი კლასები, მაგალითად, იმისათვის, რომ HelpLink და Source თვისებებს მივანიჭოთ ჩვენთვის საჭირო მნიშვნელობები. შედეგად, თავიდან ავიცილებთ ამ თვისებების განმეორებით ინიციალიზებას შექმნილი კლასის ყოველი ახალი ობიექტის შექმნის დროს. განსაკუთრებული სიტუაციების ჩვენს მიერ შექმნილი კლასი უნდა იყოს System.ApplicationException კლასის მემკვიდრე და ამ მემკვიდრე კლასის კონსტრუქტორიდან უნდა გამოვიძახოთ საბაზო კლასის კონსტრუქტორი.

```

//   პროგრამა 11.14
//   პროგრამაში ხდება საკუთარი განსაკუთრებული სიტუაციის შექმნის
//   დემონსტრირება

```

```

public class Gansakutrebuli_Situacia : ApplicationException
{
public Gansakutrebuli_Situacia(string Message) : base(Message)
{
this.HelpLink = "დამატებითი ინფორმაცია იხილეთ Readme.txt ფაილში";
this.Source = "შეცდომის გამომწვევია - პროგრამა 11.14";
}
}
private void button1_Click(object sender, EventArgs e)
{
try
{
throw new Gansakutrebuli_Situacia("ჩემი შეტყობინება Gansakutrebuli_Situacia კლასში");
}
catch ( Gansakutrebuli_Situacia arg)
{
label1.Text = arg.HelpLink;
label2.Text = arg.Message;
label3.Text = arg.Source;
label4.Text = arg.StackTrace;
label5.Text = arg.TargetSite.ToString();
}
}

```

checked და unchecked ოპერატორები

ართმეტიკული ოპერაციების შესრულებისას შეიძლება ადგილი ჰქონდეს გადავსების შეცდომებს. მაგალითად, მოყვანილ პროგრამაში ricxv1 და ricxvi2 მნიშვნელობების ნამრავლი აჭარბებს byte ტიპის მქონე მნიშვნელობის დასაშვებ დიაპაზონს. შედეგად, ადგილი ექნება გადავსების შეცდომას.

```

{
//   პროგრამა 11.15
byte ricxv1, ricxvi2, shedegi;

ricxv1 = 127;
ricxvi2 = 127;
shedegi = ( byte ) ( ricxv1 * ricxvi2 );
label1.Text = shedegi.ToString();
}

```

C# ენაში გადავსების შემთხვევაში შესაძლებელია განსაკუთრებული სიტუაციის გენერირება checked და unchecked ოპერატორების გამოყენებით. checked ოპერატორი გამოიყენება იმ გამოსახულების მისათითებლად, რომელიც უნდა შემოწმდეს გადავსების არსებობაზე. გადავსების იგნორირებისათვის გამოიყენება unchecked ოპერატორი. ამ შემთხვევაში მოხდება შედეგის ჩამოჭრა იმ მიზნით, რომ მისი ტიპი დაემთხვეს საბოლოო გამოსახულების ტიპს. ეს კი გამოიწვევს შედეგის დამახინჯებას.

checked ოპერატორს ორი ძირითადი ფორმა აქვს. პირველი ფორმა გამოიყენება

მითითებული გამოსახულების შემოწმებისათვის. მისი სინტაქსია:

checked (*გამოსახულება*)

აქ სრულდება გამოსახულების შემოწმება. თუ მისი გამოთვლისას ადგილი აქვს გადავსებას, მაშინ გენერირდება `OverflowException` განსაკუთრებული სიტუაცია.

მეორე ფორმა გამოიყენება ოპერატორების ბლოკის შესამოწმებლად. მისი სინტაქსია:

checked

{

შესამოწმებელი ოპერატორები

}

`unchecked` ოპერატორს აქვს ორი ძირითადი ფორმა. პირველი ფორმის გამოყენებისას იგნორირდება გადავსება მითითებული გამოსახულებისათვის. მისი სინტაქსია:

unchecked (*გამოსახულება*)

აქ გამოსახულება არ შემოწმდება გადავსების არსებობაზე. თუ მისი გამოთვლისას ადგილი ექნება გადავსებას, მაშინ მოხდება შედეგის ჩამოჭრა.

მეორე ფორმა ახდენს გადავსების იგნორირებას ოპერატორების ბლოკისათვის. მისი სინტაქსია:

unchecked

{

შესამოწმებელი ოპერატორები

}

ქვემოთ მოყვანილ პროგრამაში ხდება `checked` და `unchecked` ოპერატორების გამოყენება გამოსახულების მიმართ.

{

// **პროგრამა 11.16**

byte ricxv1, ricxv2, shedegi;

ricxv1 = 127;

ricxv2 = 127;

try

{

shedegi = unchecked((byte) (ricxv1 * ricxv2));

label1.Text = shedegi.ToString();

shedegi = checked((byte) (ricxv1 * ricxv2));

label2.Text = shedegi.ToString();

}

catch (`OverflowException` exc) // ეს catch ოპერატორი იჭერს გადავსების შეცდომას

{

label3.Text = exc.ToString();

}

}

მოყვანილ პროგრამაში სრულდება `checked` და `unchecked` ოპერატორების გამოყენება ოპერატორების ბლოკის მიმართ.

{

// **პროგრამა 11.17**

byte ricxv1, ricxv2, shedegi;

ricxv1 = ricxv2 = 127;

```

try
{
unchecked
{
ricxvi1 = ricxvi2 = 127;
shedegi = unchecked( ( byte ) ( ricxvi1 * ricxvi2 ) );
label1.Text = shedegi.ToString();
ricxvi1 = 120;
ricxvi2 = 10;
shedegi = unchecked( ( byte ) ( ricxvi1 * ricxvi2 ) );
label2.Text = shedegi.ToString();
}
checked
{
ricxvi1 = 2;
ricxvi2 = 7;
shedegi = checked( ( byte ) ( ricxvi1 * ricxvi2 ) );
label3.Text = shedegi.ToString();
ricxvi1 = 127;
ricxvi2 = 127;
shedegi = checked( ( byte ) ( ricxvi1 * ricxvi2 ) );
label4.Text = shedegi.ToString();
}
}
catch ( OverflowException exc ) // ეს catch ოპერატორი იჭერს გადავსების შეცდომას
{
label5.Text = exc.ToString();
}
}

```

გამართვა

.NET გარემოს შემადგენლობაში შედის პროგრამა გამმართველი. ის საშუალებას გვაძლევს პროგრამის კოდი შევასრულოთ ბიჯობრივ რეჟიმში და ვნახოთ პროგრამის ცვლადების მნიშვნელობები. ეს, თავის მხრივ, აადვილებს პროგრამის კოდში შეცდომების პოვნას.

მანამ, სანამ დავიწყებდეთ გამმართველთან მუშაობას, შევექმნათ ახალი პროექტი. ფორმაზე მოვათავსოთ ერთი button და ერთი label კომპონენტი. button კომპონენტს მივაბათ პროგრამა, რომელიც ახდენს მასივის ელემენტების შეკრებას:

```

{
int jami = 0;
int[] masivi = new int[5] { 3, 1, 8, -4, 1 };
for (int indexi = 0; indexi < masivi.Length; indexi++)
    jami += masivi[indexi];
label1.Text = jami.ToString();
}

```

წყვეტის წერტილების შექმნა

წყვეტის წერტილი არის პროგრამის კოდის ის ადგილი (სტრიქონი), სადაც წყდება პროგრამის შესრულება. ამ სტრიქონში მოთავსებული გამოსახულება (ოპერატორი ან მეთოდი) არ შესრულდება და მართვა მომხმარებელს გადაეცემა. წყვეტის წერტილის შესაქმნელად საჭირო გამოსახულების გასწვრივ მარცხენა ველში უნდა მოვათავსოთ კურსორი და დავაჭიროთ თავის მარცხენა კლავიშს. გამოჩნდება მუქი წრე. წყვეტის წერტილის შესაქმნელად შეგვიძლია, აგრეთვე, კურსორი მოვათავსოთ საჭირო გამოსახულების შიგნით და დავაჭიროთ F9 კლავიშს. პროგრამის კოდში შეგვიძლია შევქმნათ წყვეტის რამდენიმე წერტილი. ჩვენი პროგრამის კოდში კურსორი მოვათავსოთ for ოპერატორის შემცველ სტრიქონში და დავაჭიროთ F9 კლავიშს. შეიქმნება წყვეტის წერტილი (ნახ. 11.2).

წყვეტის წერტილის გასაუქმებლად უნდა დავაჭიროთ მუქ წრეს ან F9 კლავიშს. წყვეტის წერტილთან პროგრამის კოდის შესრულება წყდება. ამის შემდეგ, პროგრამის კოდი შეგვიძლია ბიჯობრივ რეჟიმში შევასრულოთ და დავაკვირდეთ ცვლადების მნიშვნელობებს.

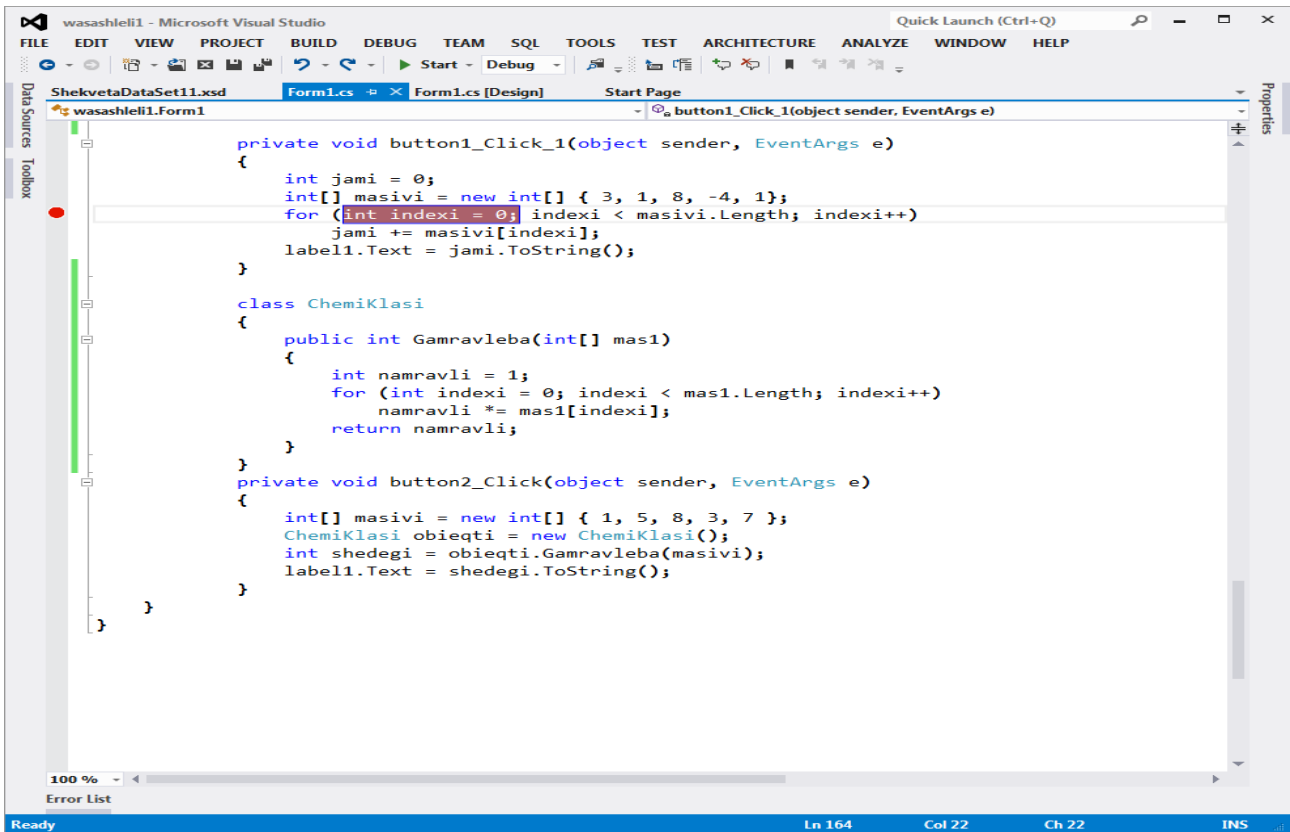
გამმართველის გაშვება ხდება F5 კლავიშით. თუ გვინდა კოდის შესრულება გამმართველის გარეშე, მაშინ უნდა დავაჭიროთ Ctrl+F5 კლავიშებს. დავაჭიროთ F5 კლავიშს. გამოჩნდება ფორმა. შემდეგ დავაჭიროთ button კლავიშს. გამმართველი კოდს წყვეტის წერტილამდე ასრულებს და მართვას მომხმარებელს გადასცემს. შესაბამისი გამოსახულების (int indexi = 0;) გასწვრივ მარცხენა ველში გამოჩნდება ისარი (ნახ. 11.3). ეს გამოსახულება ჯერ არ არის შესრულებული.

ცვლადების მნიშვნელობების ნახვა

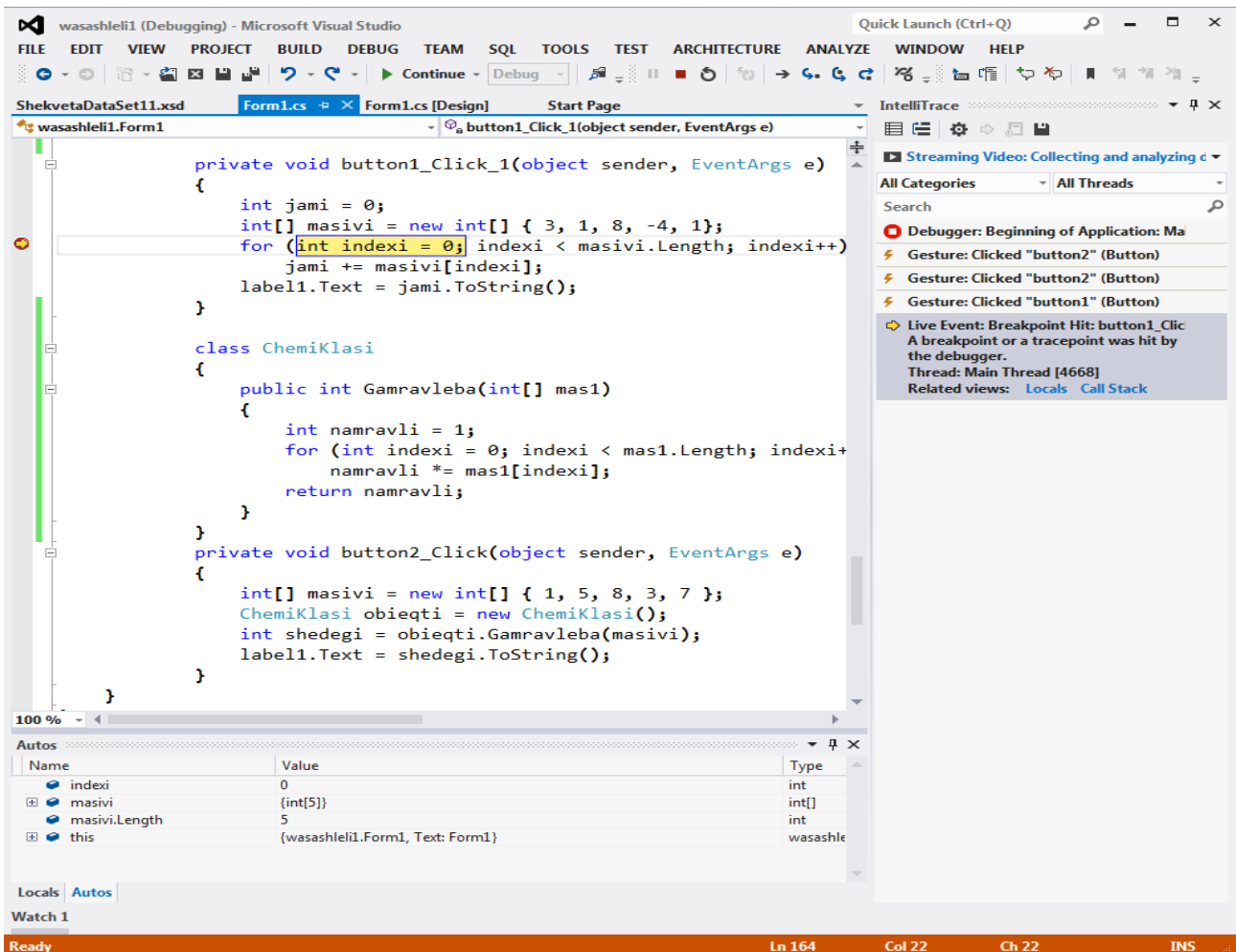
როგორც აღვნიშნეთ, წყვეტის წერტილთან გამმართველი წყვეტს კოდის შესრულებას. ამ დროს შეგვიძლია ვნახოთ ცვლადების მნიშვნელობები.

Autos ფანჯარა. ეს ფანჯარა მოთავსებულია გამმართველის ფანჯრის ქვემოთ (ნახ. 11.3). თუ ის არ ჩანს, მაშინ ვხსნით Debug მენიუს, Windows ქვემენიუს და ვასრულებთ Autos ბრძანებას (Debug→Windows→Autos). Windows ქვემენიუს ბრძანებები გამოჩნდება წყვეტის წერტილთან პროგრამის შეჩერების შემდეგ. ამ ფანჯარაში ჩანს იმ ცვლადების მნიშვნელობები, რომლებიც გამოიყენება მიმდინარე და წინა გამოსახულებებში. ამ ეტაპზე მასში ჩანს indexi, masivi და masivi.Length ცვლადების მნიშვნელობები. ცვლადის მნიშვნელობა შეგვიძლია ვნახოთ, აგრეთვე, თუ პროგრამის კოდში მის სახელზე გავაჩერებთ კურსორს. თუ Autos ფანჯარაში დავაჭერთ masivi სახელის მარცხნივ მოთავსებულ + ნიშანზე, მაშინ გამოჩნდება ამ მასივის ელემენტების მნიშვნელობები.

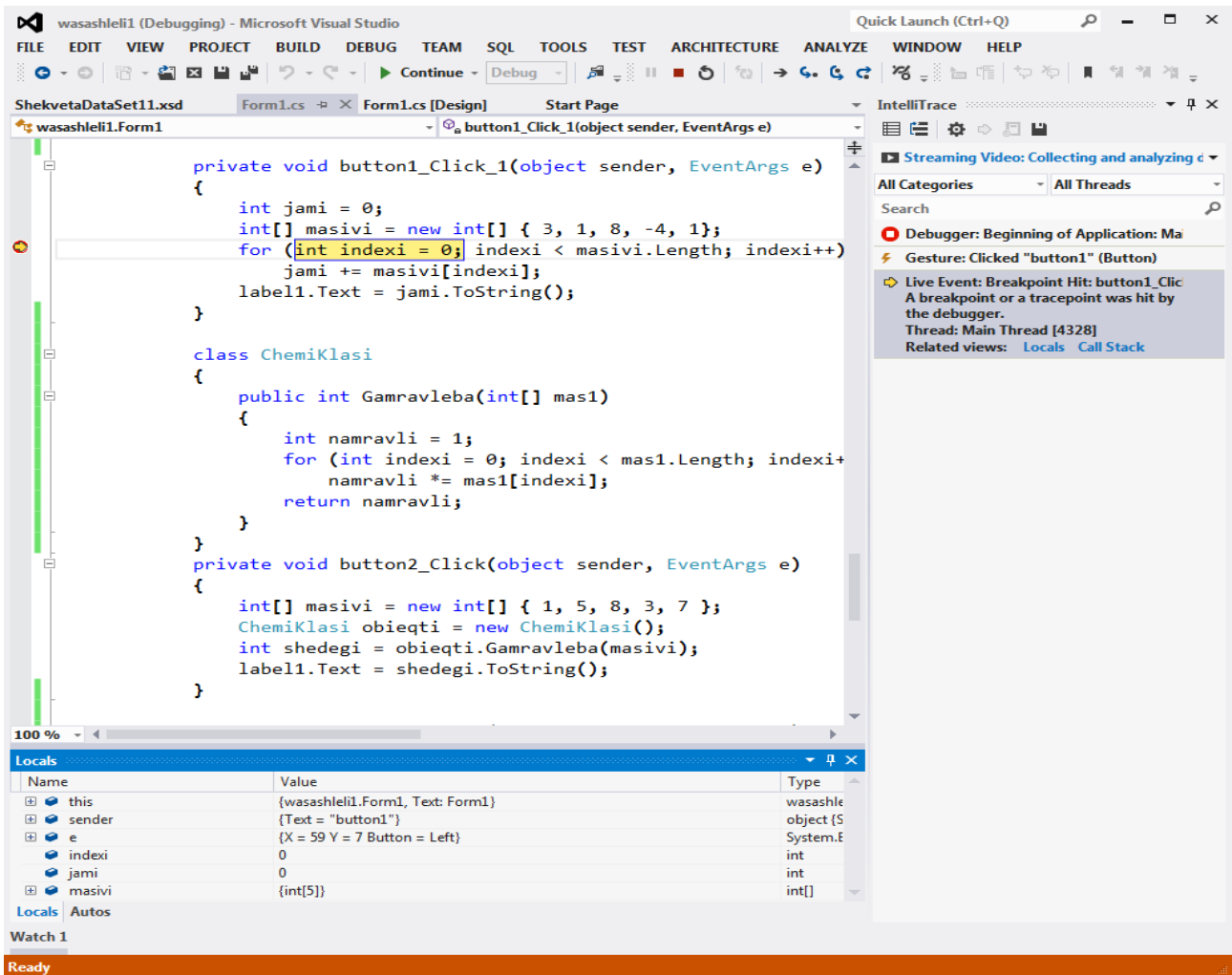
Locals ფანჯარა. ეს ფანჯარა მოთავსებულია გამმართველის ფანჯრის ქვემოთ (ნახ. 11.4). მის გასახსნელად უნდა დავაჭიროთ Locals ჩანართს. თუ ეს ფანჯარა არ ჩანს, მაშინ Debug→Windows→Locals. მასში ჩანს იმ ცვლადების მნიშვნელობები, რომლებიც იმყოფებიან ხილვადობის მოცემულ უბანში. ამ მომენტისათვის ხილვადობის უბანში იმყოფება indexi, jami და masivi ცვლადები.



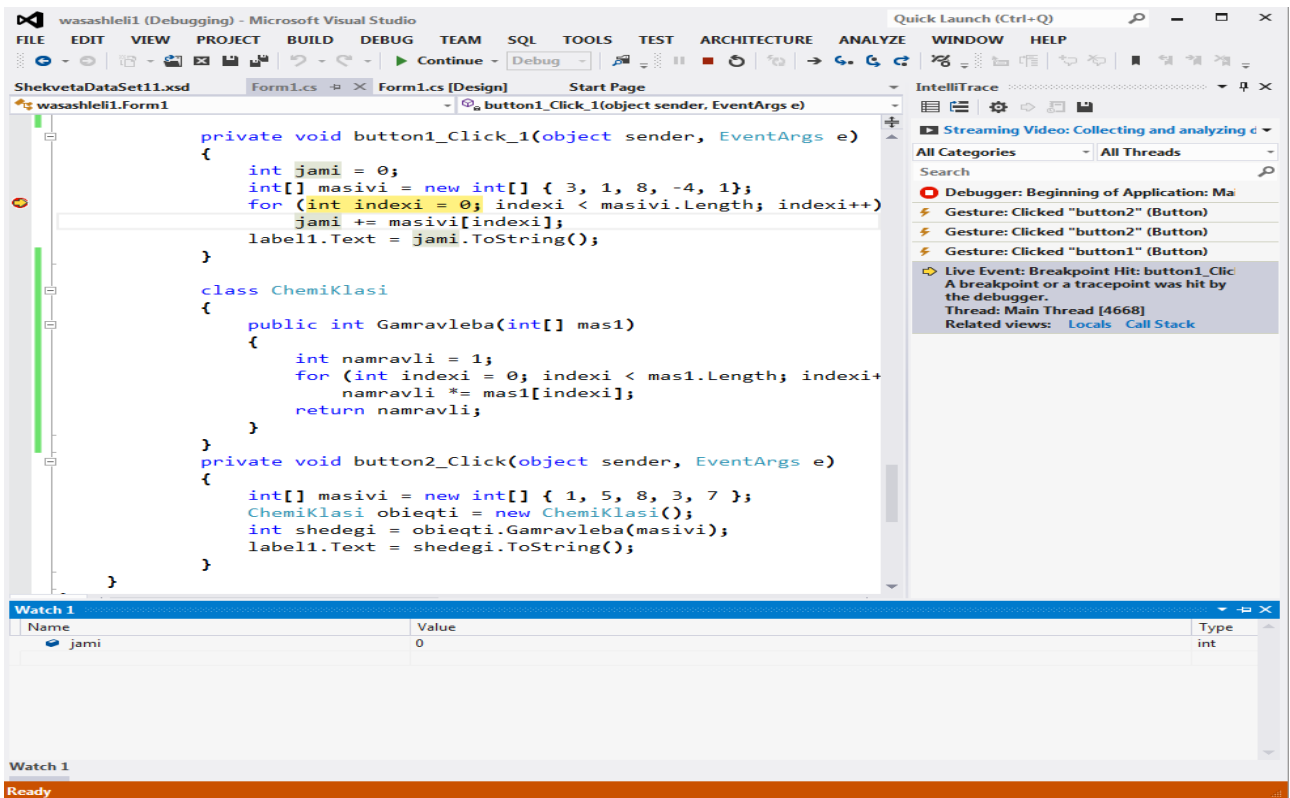
6sb. 11.2.



6sb. 11.3.



6sb. 11.4.



6sb. 11.5.

Watch ფანჯარა. ეს ფანჯარა მოთავსებულია გამმართველის ფანჯრის ქვემოთ (ნახ. 11.5). მის გასახსნელად უნდა დავაჭიროთ Watch ჩანართს. თუ ეს ფანჯარა არ ჩანს, მაშინ Debug→Windows→Watch→Watch1. მასში ჩანს სპეციალურად მითითებული ცვლადების მნიშვნელობები. ეს საჭიროა მაშინ, როცა ცვლადების დიდი რაოდენობიდან გვინტერესებს მხოლოდ რამდენიმე. Watch ფანჯარაში ცვლადის დასამატებლად პროგრამის კოდში ცვლადის სახელზე უნდა დავაჭიროთ თავის მარჯვენა კლავიშს და კონტექსტური მენიუდან შევასრულოთ Add Watch ბრძანება. ასეთი გზით jami ცვლადი დავუმატოთ Watch ფანჯარას.

პროგრამის კოდის შესრულება ბიჯობრივ რეჟიმში

პროგრამის გამართვის დროს შეგვიძლია ცვლადების მნიშვნელობების ნახვა და ოპერატორების ბიჯობრივ რეჟიმში შესრულება. მიმდინარე გამოსახულების შესასრულებლად შეგვიძლია დავაჭიროთ F10 კლავიშს ან შევასრულოთ Debug→Step Over ბრძანება. თუ გვინდა მეთოდის კოდის ბიჯობრივ რეჟიმში შესრულება, მაშინ უნდა დავაჭიროთ F11 კლავიშს ან შევასრულოთ Debug→Step Into ბრძანება. მეთოდის კოდიდან გამოსასვლელად ვასრულებთ Debug→Step Out ბრძანებას ან ვაჭერთ Shift+F11 კლავიშებს. მეთოდიდან გამოსვლისას მართვა გადაეცემა ამ მეთოდის გამომძახებელ გამოსახულებას.

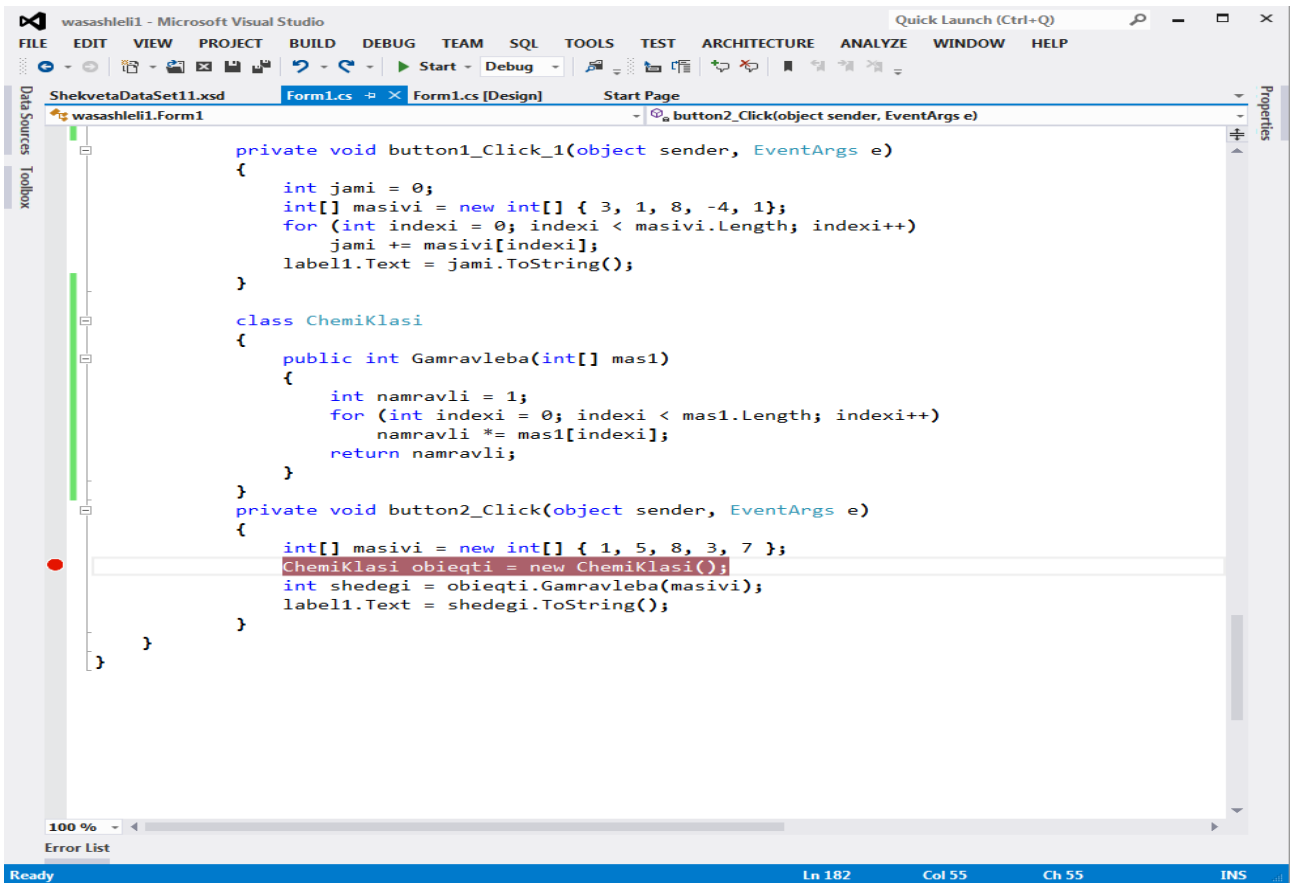
ახლა ჩვენს პროგრამას დავუმატოთ ChemiKlasi კლასი, რომელიც Gamravleba() მეთოდს შეიცავს. ეს მეთოდი ასრულებს მასივის ელემენტების ერთმანეთზე გამრავლებას და ნამრავლის გაცემას. კლასს შემდეგი სახე აქვს:

```
class ChemiKlasi
{
public int Gamravleba(int[ ] mas1)
{
int namravli = 1;
for ( int indexi = 0; indexi < mas1.Length; indexi++ )
    namravli *= mas1[indexi];
return namravli;
}
}
```

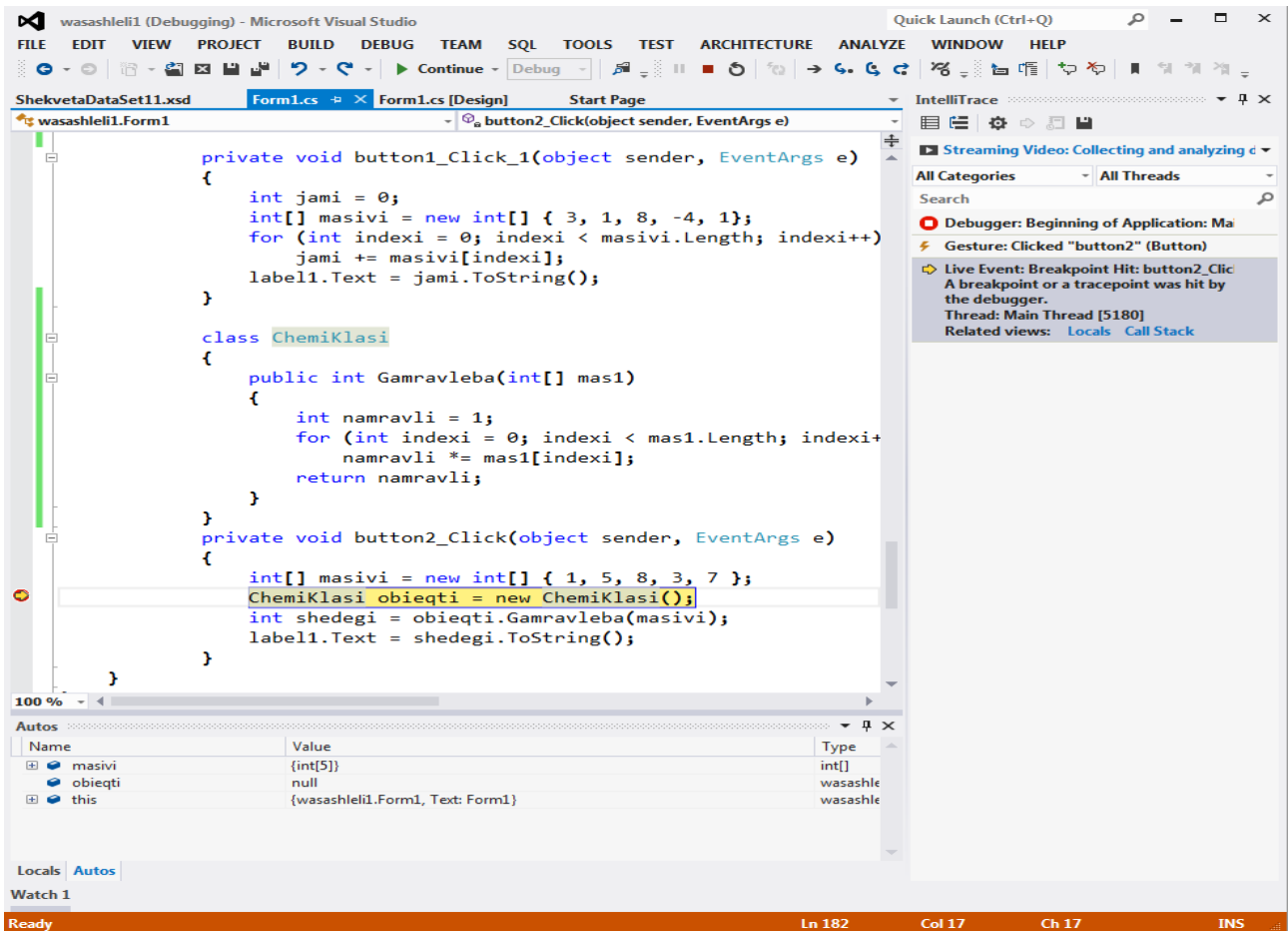
ფორმაზე მოვათავსოთ მეორე button კომპონენტი. მას მივაბათ პროგრამის კოდი, რომელიც Gamravleba() მეთოდს გამოიძახებს:

```
{
int[] masivi = new int[] { 1, 5, 8, 3, 7 };
ChemiKlasi obieqti = new ChemiKlasi();
int shedegi = obieqti.Gamravleba(masivi);
label1.Text = shedegi.ToString();
}
```

ახლა სტრიქონში, რომელშიც ხდება Gamravleba() მეთოდის გამოიძახება შევქმნათ წყვეტის წერტილი (ნახ. 11.6). დავაჭიროთ F5 კლავიშს. გაიხსნება ფორმა. დავაჭიროთ მეორე button კლავიშს. პროგრამის შესრულება შეწყდება წყვეტის წერტილში. ამის შემდეგ, თუ გვინდა Gamravleba() მეთოდის შესრულება ბიჯობრივ რეჟიმში უნდა დავაჭიროთ F11 კლავიშს (ნახ. 11.8). ამ კლავიშზე ყოველი დაჭერის შემდეგ შეგვიძლია ვნახოთ ცვლადების მნიშვნელობები.



б.б. 11.6.



б.б. 11.7.

თავი 12. სტრიქონები

სტრიქონები

სტრიქონები გამოიყენება Unicode სიმბოლოების მიმდევრობის შესანახად. ერთი Unicode სიმბოლო იკავებს 2 ბაიტს ანუ 16 ბიტს. სტრიქონები წარმოადგენენ System.String კლასის მიმართებით ტიპის მქონე ობიექტებს. ისინი აღიწერება string სიტყვის საშუალებით. სტრიქონის (string ტიპის ობიექტის) შექმნის ყველაზე მარტივი საშუალებაა სტრიქონული ლიტერალის გამოყენება. მაგალითად,

```
string striqoni = "ეს არის სტრიქონული ლიტერალი";
```

აქ striqoni ცვლადი არის string ტიპის მიმართებით ცვლადი, რომელსაც ენიჭება სტრიქონულ ლიტერალზე ("ეს არის სტრიქონული ლიტერალი") მიმართვა. ამ შემთხვევაში striqoni ცვლადის ინიციალიზება ხდება სიმბოლოების მიმდევრობით - "ეს არის სტრიქონული ლიტერალი".

ისევე როგორც მასივებში, აქაც პირველი ელემენტის (სიმბოლოს) ინდექსია 0. ინდექსი გამოიყენება მხოლოდ სიმბოლოს მისაღებად. მაგალითად,

```
string striqoni = "კომპიუტერი";
```

```
label1.Text = striqoni[2];
```

ორივე ოპერატორის შესრულების შედეგად label კომპონენტში გამოჩნდება ასო "მ". რადგან, სტრიქონები ობიექტებს წარმოადგენენ, ამიტომ მათ Length თვისება აქვთ. მასში ინახება სტრიქონის სიგრძე (სტრიქონში სიმბოლოების რაოდენობა).

სტრიქონი შეიძლება შეიცავდეს მმართველ სიმბოლოებს. მაგალითად,

```
string striqoni = "საბა \n ანა";
```

```
label1.Text = striqoni;
```

ამ სტრიქონების შესრულების შედეგად label1 კომპონენტის პირველ სტრიქონში გამოჩნდება "საბა", მეორე სტრიქონში კი - "ანა".

შეგვიძლია შევქმნათ ისეთი სტრიქონებიც, რომლებშიც მმართველი სიმბოლოები არ იმუშავებენ. ასეთი სტრიქონები @ სიმბოლოთი იწყება. მაგალითად,

```
string striqoni = @" საბა \n ანა";
```

```
label1.Text = striqoni;
```

ამ სტრიქონების შესრულების შედეგად label1 კომპონენტში გამოჩნდება სტრიქონი - "საბა \n ანა".

სტრიქონებთან სამუშაო მეთოდები

String კლასის თვისება და მეთოდები მოყვანილია ცხრილებში 12.1 და 12.2.

მოყვანილ პროგრამაში ხდება Length თვისებასთან მუშაობის დემონსტრირება.

```
{  
string striqoni1 = @" საბა \n ანა \n";  
label1.Text = striqoni1.Length.ToString() + '\n'; // გაიცემა 15  
string striqoni2 = " საბა \n ანა \n";  
label2.Text = striqoni2.Length.ToString() + '\n'; // გაიცემა 13  
}
```

ცხრილი 12.1. String კლასის თვისება.

თვისება	ტიპი	აღწერა
Length	int	შეიცავს სტრიქონში სიმბოლოების რაოდენობას

ცხრილი 12.2. String კლასის მეთოდები.

მეთოდი	აღწერა
int Compare(string სტრიქონი1, string სტრიქონი2)	ადარებს ორ სტრიქონს. შედარების დროს გაითვალისწინება ენისა და კულტურის თავისებურებები
CompareOrdinal(string სტრიქონი1, string სტრიქონი2)	ადარებს ორ სტრიქონს. შედარების დროს არ გაითვალისწინება ენისა და კულტურის თავისებურებები
string Concat(string სტრიქონი1, string სტრიქონი2)	გასცემს ორი ან მეტი სტრიქონის გაერთიანებას
string Copy(string სტრიქონი)	გასცემს მითითებული სტრიქონის ასლს
bool Equals(string სტრიქონი1, string სტრიქონი2)	გასცემს true მნიშვნელობას, თუ სტრიქონები ტოლია, წინააღმდეგ შემთხვევაში false მნიშვნელობას
string Format(string სტრიქონი, object ობიექტი)	გასცემს მითითებული ფორმატის მიხედვით დაფორმატებულ სტრიქონს
string Intern(string სტრიქონი)	გასცემს სისტემურ მიმართვას მითითებულ სტრიქონზე
string IsInterned(string სტრიქონი)	გასცემს მიმართვას მითითებულ სტრიქონზე
string Join(string გამყოფი, string[] მასივი)	მეთოდი გამყოფს ათავსებს მასივის ელემენტებს შორის და გასცემს მიღებულ სტრიქონს
int CompareTo(string სტრიქონი)	გამომძახებელ სტრიქონს ადარებს მითითებულ სტრიქონთან
void CopyTo(int საწყისი_ინდექსი, char[] სიმბოლოების_მასივი, int საბოლოო_ინდექსი, int რაოდენობა)	გამომძახებელი სტრიქონის მითითებული პოზიციიდან სიმბოლოების მასივის მითითებულ პოზიციიაში გადაწერს მითითებული რაოდენობის Unicode სიმბოლოს
bool EndsWith(string სტრიქონი)	ამოწმებს მთავრდება თუ არა გამომძახებელი სტრიქონი მითითებული სტრიქონით
Type GetType()	გასცემს მიმდინარე ეგზემპლარის ტიპს
int IndexOf(string სტრიქონი)	გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც პირველად იყო ნაპოვნი მითითებული ქვესტრიქონი
int IndexOfAny(char[] მასივი)	გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც პირველად იყო ნაპოვნი სიმბოლოების მასივის ნებისმიერი სიმბოლო
string Insert(int ინდექსი, string სტრიქონი)	გამომძახებელ სტრიქონში ინდექსი პოზიციიდან დაწყებული ჩასვამს მითითებულ სტრიქონს

ცხრილი 12.2. (გაგრძელება)

int LastIndexOf(string სტრიქონი)	გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც უკანასკნელად იყო ნაპოვნი მითითებული ქვესტრიქონი
int LastIndexOfAny(char[] მასივი)	გასცემს გამომძახებელი სტრიქონის იმ პოზიციის ნომერს, სადაც უკანასკნელად იყო ნაპოვნი სიმბოლოების მასივის ნებისმიერი სიმბოლო
string PadLeft(int სიგანე, char სიმბოლო)	სტრიქონის სიმბოლოებს ასწორებს მარცხენა საზღვარზე, უმატებს რა სტრიქონის დასაწყისს ინტერვალებს ან მითითებულ სიმბოლოს იმდენჯერ, რომ სტრიქონის სიგრძე გახდეს მითითებული სიგანის ტოლი
string PadRight(int სიგანე, char სიმბოლო)	სტრიქონის სიმბოლოებს ასწორებს მარჯვენა საზღვარზე, უმატებს რა სტრიქონის ბოლოს ინტერვალებს ან მითითებულ სიმბოლოს იმდენჯერ, რომ სტრიქონის სიგრძე გახდეს მითითებული სიგანის ტოლი
string Remove(int ინდექსი, int რაოდენობა)	გამომძახებელ სტრიქონში ინდექსი პოზიციიდან დაწყებული წამლის მითითებული რაოდენობის სიმბოლოს
string Replace(string ძველი_სტრიქონი, string ახალი_სტრიქონი)	გამომძახებელ სტრიქონში ძველ სტრიქონს ახალი სტრიქონით შეცვლის
string[] Split(params char[] გამყოფი)	სტრიქონს ჰყოფს სტრიქონების მასივად მითითებული გამყოფის გამოყენებით.
bool StartsWith(string სტრიქონი)	ამოწმებს იწყება თუ არა გამომძახებელი სტრიქონი მითითებული სტრიქონით
string Substring (int ინდექსი, int რაოდენობა)	გამომძახებელი სტრიქონის ინდექსი პოზიციიდან გასცემს მითითებული რაოდენობის სიმბოლოს
char[] ToCharArray()	სტრიქონის სიმბოლოებს გადაწერს სიმბოლოების მასივში
string ToLower()	გამომძახებელი სტრიქონის ყველა სიმბოლო გადაყავს ქვედა რეგისტრში
string ToString()	გამომძახებელ ობიექტს სტრიქონად გარდაქმნის
string ToUpper()	გამომძახებელი სტრიქონის ყველა სიმბოლო გადაყავს ზედა რეგისტრში
string Trim()	გამომძახებელ სტრიქონში შლის საწყის და ბოლო ინტერვალებს
string TrimEnd(char[] სიმბოლოები)	გამომძახებელი სტრიქონის ბოლოში შლის სიმბოლოების მასივში მითითებულ სიმბოლოებს
string TrimStart(char[] სიმბოლოები)	გამომძახებელი სტრიქონის დასაწყისში შლის სიმბოლოების მასივში მითითებულ სიმბოლოებს

განვიხილოთ ზოგიერთი მეთოდი.

ქვემოთ მოყვანილ პროგრამაში ხდება **Insert**, **Remove** და **Replace** მეთოდების გამოყენების დემონსტრირება.

```
{
//   პროგრამა 12.1
//   პროგრამაში ხდება Insert, Remove და Replace მეთოდებთან მუშაობის დემონსტრირება
string str1, str2 = "C# დაპროგრამების ენაა", str3 = "თანამედროვე ";
str1 = str2.Insert(17, str3);      // მიიღება სტრიქონი "C# დაპროგრამების თანამედროვე ენაა"
label1.Text = str1;
str1 = str2.Remove(3, 14);        // მიიღება სტრიქონი "C# ენაა"
label2.Text = str1;
str1 = str2.Replace("დაპროგრამების ", str3);    // მიიღება სტრიქონი "C# თანამედროვე ენაა"
label3.Text = str1;
}
```

პროგრამის `str1 = str2.Insert(17, str3)`; სტრიქონში `str2` არის გამომძახებელი სტრიქონის სახელი. მის მარჯვნივ წერტილის დასმის შემდეგ გაიხსნება სტრიქონებთან სამუშაო მეთოდების სია. ვირჩევთ `Insert` მეთოდს. მისი შესრულების შედეგად `str2` სტრიქონის მე-17 პოზიციიდან მოხდება `str3` სტრიქონის ჩასმა, ანუ "თანამედროვე" სტრიქონის ჩასმა. შედეგად, მიიღება სტრიქონი "C# დაპროგრამების თანამედროვე ენაა", რომელიც მიენიჭება `str1` ცვლადს.

+ ოპერატორი გამოიყენება სტრიქონების კონკატენაციის (შეერთების) ოპერაციის შესასრულებლად. განვიხილოთ კოდის ფრაგმენტი:

```
string striqoni1 = "ეს არის ";
string striqoni2 = "კომ";
string striqoni3 = "პიუტერი";
string striqoni = striqoni1 + striqoni2 + striqoni3;
```

ამ ოპერატორების შესრულების შედეგად `striqoni` ცვლადს მიენიჭება სტრიქონი - "ეს არის კომპიუტერი".

ორი სტრიქონის შესადარებლად შეგვიძლია == ოპერატორის გამოყენება.

მოყვანილ პროგრამაში ხდება **Copy**, **CompareTo**, **IndexOf**, **LastIndexOf**, **Substring** მეთოდების მუშაობის დემონსტრირება.

```
{
//   პროგრამა 12.2
//   პროგრამაში ხდება Copy, CompareTo, IndexOf, LastIndexOf, Substring
//   მეთოდებთან მუშაობის დემონსტრირება
string striqoni1, striqoni2 = "ერთი ორი სამი ერთი ორი სამი",
    striqoni3 = textBox1.Text;
int shedegi, indexi;

//   striqoni2 სტრიქონი გადაიწერება striqoni1 სტრიქონში
striqoni1 = string.Copy(striqoni2);
label1.Text = striqoni1;

striqoni1 = striqoni2.Substring(9, 13);    // მიიღება სტრიქონი "ერთი ორი სამი"
label2.Text = striqoni1;

indexi = striqoni2.IndexOf("ორი");        // გაიცემა 5
label3.Text = indexi.ToString();
indexi = striqoni2.LastIndexOf("ორი");    // გაიცემა 19
}
```

```
label4.Text = indexi.ToString();
```

```
//      striqoni2 და striqoni4 სტრიქონების შედარება  
shedegi = striqoni2.CompareTo(striqoni3);  
if ( shedegi == 0 ) label5.Text = "სტრიქონები ერთმანეთის ტოლია";  
    else if ( shedegi > 0 ) label5.Text = " striqoni2 > striqoni3";  
        else label5.Text = " striqoni2 < striqoni3";  
}
```

Compare() და **Equals()** მეთოდები. **Equals()** მეთოდი ორ სტრიქონს ადარებს და გასცემს true მნიშვნელობას თუ სტრიქონები ტოლია, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას. მისი სინტაქსია:

Equals(სტრიქონი1, სტრიქონი2)

მაგალითი.

```
{  
string striqoni1 = "საბა ანა ლიკა რომანი",  
    striqoni2 = "ანა რომანი საბა ლიკა";  
bool shedegi = striqoni1.Equals(striqoni2);  
if ( shedegi == true ) label1.Text = "სტრიქონები ერთნაირია";  
    else label1.Text = " სტრიქონები ერთნაირი არ არის";  
}
```

Compare() მეთოდი ორ სტრიქონს ადარებს ენის, ეროვნული და კულტურული თავისებურებების გათვალისწინებით. თუ პირველი სტრიქონი მეტია მეორეზე, მაშინ გაიცემა 1. თუ პირველი სტრიქონი ნაკლებია მეორეზე, მაშინ გაიცემა -1. თუ სტრიქონები ტოლია, მაშინ გაიცემა 0. ეს მეთოდი გადატვირთულია და ამიტომ აქვს რამდენიმე ვერსია. ჩვენ განვიხილავთ რამდენიმე მათგანს.

Compare() მეთოდის უმარტივესი ვერსიის სინტაქსია:

String.Compare(სტრიქონი1, სტრიქონი2)

მაგალითი.

```
{  
string striqoni1 = "საბა ანა ლიკა რომანი",  
    striqoni2 = "ანა რომანი საბა ლიკა";  
int shedegi = String.Compare(striqoni1, striqoni2);  
if ( shedegi == 0 ) label1.Text = "სტრიქონები ერთნაირია";  
    else label1.Text = " სტრიქონები ერთნაირი არ არის";  
}
```

Compare() მეთოდის მეორე ვერსია შედარების დროს ითვალისწინებს სიმბოლოების რეგისტრს. მისი სინტაქსია:

String.Compare(სტრიქონი1, სტრიქონი2, რეგისტრი)

თუ რეგისტრი იღებს true მნიშვნელობას, მაშინ შედარების დროს რეგისტრი არ იქნება გათვალისწინებული. თუ ის იღებს false მნიშვნელობას, მაშინ შედარების დროს რეგისტრი იქნება გათვალისწინებული.

მაგალითი.

```
{  
string striqoni1 = "საბა ანა ლიკა რომანი",  
    striqoni2 = "ანა რომანი საბა ლიკა";  
int shedegi = String.Compare(striqoni1, striqoni2, true);  
if ( shedegi == 0 ) label1.Text = "სტრიქონები ერთნაირია";  
}
```

```

else label1.Text = " სტრიქონები ერთნაირი არ არის";
}

```

Compare() მეთოდის მესამე ვერსია საშუალებას გვაძლევს შევადაროთ ორი სტრიქონის ნაწილები (ქვესტრიქონები). მისი სინტაქსია:

String.Compare(სტრიქონი1, ინდექსი1, სტრიქონი2, ინდექსი2, სიმბოლოების_რაოდენობა)
 მოყვანილ პროგრამაში ხდება ამ მეთოდთან მუშაობის დემონსტრირება.

```

{
string striqoni1 = "საბა ანა ლიკა რომანი",
    striqoni2 = "ანა რომანი საბა ლიკა";
int shedegi = String.Compare(striqoni1, 5, striqoni2, 5, 4);
if ( shedegi == 0 ) label1.Text = "სტრიქონები ერთნაირია";
    else label1.Text = " სტრიქონები ერთნაირი არ არის";
}

```

მოყვანილ პროგრამაში ხდება **Concat()** მეთოდთან მუშაობის დემონსტრირება.

```

{
string striqoni1 = "საბა";
string striqoni2 = string.Concat(striqoni1, " ანა");
string striqoni3 = striqoni2 + " ლიკა";
label1.Text = striqoni3;
}

```

როგორც პროგრამის კოდიდან ჩანს სტრიქონების კონკატენაციისათვის შეიძლება გამოვიყენოთ + ოპერატორიც.

Format() მეთოდი სტრიქონის ფორმატირებისათვის იყენებს ფორმატირების სიმბოლოებს, რომლებიც მოყვანილია ცხრილში 12.3.

ცხრილი 12.3. სტრიქონის დაფორმატების სიმბოლოები.

დაფორმატების სიმბოლო	აღწერა
f ან F	რიცხვს აფორმატებს როგორც წილადს
e ან E	რიცხვს აფორმატებს როგორც რიცხვს ექსპონენციალური წარმოდგენით
p ან P	რიცხვს აფორმატებს როგორც პროცენტს
n ან N	რიცხვს აფორმატებს როგორც რიცხვს თანრიგების გამყოფებით
c ან C	რიცხვს აფორმატებს როგორც თანხას ადგილობრივ ვალუტაში
d ან D	რიცხვს აფორმატებს როგორც ათობით რიცხვს
g ან G	რიცხვს აფორმატებს როგორც წილადს ან როგორც რიცხვს ექსპონენციალური წარმოდგენით
x ან X	მთელი რიცხვი გადაჰყავს თექვსმეტობით წარმოდგენაში

მოყვანილ პროგრამაში ხდება **Format()** მეთოდის გამოყენების დემონსტრირება.

```

{
//   პროგრამა 12.3
//   პროგრამაში ხდება Format() მეთოდის გამოყენების დემონსტრირება
label1.Text = "";
string striqoni;
int mteli = 12;
double wiladi_d = 1234.56789;
float wiladi_f = 1234.56789f;
}

```

```

decimal valuta = 12345.67m;
striqoni = String.Format("{0:D5}\n", mteli); // 00012
label1.Text += striqoni;
striqoni = String.Format("{0:X}\n", mteli); // C
label1.Text += striqoni;
striqoni = String.Format("{0:P3}\n", mteli); // 1 200,000%
label1.Text += striqoni;
striqoni = String.Format("{0:C3}\n", valuta); // 12 345,670 Lari
label1.Text += striqoni;
striqoni = String.Format("{0:F3}\n", wiladi_d); // 1234,568
label1.Text += striqoni;
striqoni = String.Format("{0:F3}\n", wiladi_f); // 1234,568
label1.Text += striqoni;
striqoni = String.Format("{0:P3}\n", wiladi_d); // 123 456,789%
label1.Text += striqoni;
striqoni = String.Format("{0:E3}\n", wiladi_d); // 1,235E+003
label1.Text += striqoni;
striqoni = String.Format("{0:N3}\n", wiladi_d); // 1 234,568
label1.Text += striqoni;
striqoni = String.Format("{0:G3}\n", wiladi_d); // 1,23E+03
label1.Text += striqoni;
}

```

{0:F3} ფორმატში პირველი სიმბოლო "0" მიუთითებს, რომ უნდა დაფორმატდეს რიგით პირველი ცვლადი. მეორე სიმბოლო მიუთითებს, რომ ათწილადში წილად ნაწილს უნდა დაეთმოს 3 თანრიგი. მოყვანილ მაგალითში დაფორმატდება რიგით მეორე ცვლადი - wiladi_f, რადგან ფორმატში მითითებულია 1.

```

{
double wiladi_d = 9234.56781;
float wiladi_f = 1234.56789f;
string striqoni = String.Format("{1:F3}\n", wiladi_d, wiladi_f); // 1234,568
label1.Text = striqoni;
}

```

თუ ფორმატში არ მივუთითებთ თანრიგების რაოდენობას წილადი ნაწილისათვის, მაშინ ის აიღება ორის ტოლი. მაგალითად,

```

{
float wiladi_f = 1234.56789f;
string striqoni = String.Format("{0:F}\n", wiladi_f); // 1234,56
label1.Text = striqoni;
}

```

მოყვანილ პროგრამაში ხდება Join() მეთოდთან მუშაობის დემონსტრირება.

```

{
string[] striqonebis_masivi = { "საბა", "ანა", "ლიკა", "რომანი" };
string striqoni = String.Join("-", striqonebis_masivi);
label17.Text = striqoni; // striqoni = "საბა-ანა-ლიკა-რომანი"
}

```

მოყვანილ პროგრამაში ხდება Split() მეთოდთან მუშაობის დემონსტრირება.

```

{

```

```

label1.Text = "";
string striqoni = "რომანი, ანა:საბა ლიკა.ნატა";
string[] shedegi = striqoni.Split(new Char[] { ' ', ',', '!', ':' });

foreach ( string sityva in shedegi )
{
if ( sityva.Trim() != "" )
    label1.Text += sityva + '\n';           //    shedegi = "რომანი ანა საბა ლიკა ნატა"
}
}

```

მოყვანილ პროგრამაში ხდება **StartWith()** და **EndWith()** მეთოდთან მუშაობის დემონსტრირება.

```

{
//    პროგრამა 12.4
//    პროგრამაში ხდება StartWith() და EndWith() მეთოდებთან მუშაობის
//    დემონსტრირება
string striqoni1, striqoni2;
striqoni1 = textBox1.Text;
striqoni2 = textBox2.Text;
if ( striqoni1.StartsWith(striqoni2) )
    label1.Text = "სტრიქონი იწყება " + striqoni2 + " სტრიქონით";
    else label1.Text = " სტრიქონი არ იწყება " + striqoni2 + " სტრიქონით";
if ( striqoni1.EndsWith(striqoni2) )
    label2.Text = "სტრიქონი მთავრდება " + striqoni2 + " სტრიქონით";
    else label2.Text = " სტრიქონი არ მთავრდება " + striqoni2 + " სტრიქონით";
}

```

IndexOfAny() და **LastIndexOfAny()** მეთოდები გადატვირთულია და ამიტომ აქვთ რამდენიმე ვერსია. ყველაზე მარტივი ვერსიის სინტაქსია:

```

int IndexOfAny(char[ ] სიმბოლოების_მასივი)
int LastIndexOfAny(char[ ] სიმბოლოების_მასივი)

```

ამ მეთოდების მეორე ვერსია საშუალებას გვაძლევს ძებნა დავიწყოთ მითითებული ინდექსიდან. ამ ვერსიის სინტაქსია:

```

int IndexOfAny(char[ ] სიმბოლოების_მასივი, ინდექსი)
int LastIndexOfAny(char[ ] სიმბოლოების_მასივი, ინდექსი)

```

ამ მეთოდების მესამე ვერსია საშუალებას გვაძლევს მივუთითოთ შესამოწმებელი სიმბოლოების რაოდენობა. ამ ვერსიის სინტაქსია:

```

int IndexOfAny(char[ ] სიმბოლოების_მასივი, ინდექსი, სიმბოლოების_რაოდენობა)
int LastIndexOfAny(char[ ] სიმბოლოების_მასივი, ინდექსი, სიმბოლოების_რაოდენობა)

```

მოყვანილ პროგრამაში ხდება **IndexOfAny()** და **LastIndexOfAny()** მეთოდებთან მუშაობის დემონსტრირება.

```

{
//    პროგრამა 12.5
//    პროგრამაში ხდება IndexOfAny() და LastIndexOfAny() მეთოდებთან მუშაობის
//    დემონსტრირება
char[] simboloebis_masivi = { 'ნ', 'ა' };
string striqoni1 = "ლიკა, ანა და რომანი";

```

```

int indexi1 = striqoni1.IndexOfAny(simboloebis_masivi); // indexi1 = 3
int indexi2 = striqoni1.LastIndexOfAny(simboloebis_masivi); // indexi2 = 17
label1.Text = indexi1.ToString() + " " + indexi2.ToString();

int indexi3 = striqoni1.IndexOfAny(simboloebis_masivi, 5); // indexi3 = 6
int indexi4 = striqoni1.LastIndexOfAny(simboloebis_masivi, 5); // indexi4 = 3
label2.Text = indexi3.ToString() + " " + indexi4.ToString();

int indexi5 = striqoni1.IndexOfAny(simboloebis_masivi, 5, 6); // indexi5 = 6
int indexi6 = striqoni1.LastIndexOfAny(simboloebis_masivi, 5, 6); // indexi5 = 3
label3.Text = indexi5.ToString() + " " + indexi6.ToString(); // 6 3
}

```

PadLeft() და **PadRight()** მეთოდებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```

{
string striqoni2 = "საბა";
string striqoni1 = striqoni2.PadLeft(10, '*'); // striqoni1 = "*****საბა"
string striqoni3 = striqoni2.PadRight(10, '*'); // striqoni3 = "საბა*****"
label1.Text = striqoni1.ToString();
label2.Text = striqoni3.ToString();
}

```

Trim(), **TrimStart()** და **TrimEnd()** მეთოდები გადატვირთულია. **Trim()** მეთოდის ყველაზე მარტივი ვერსია, რომელშიც არ ეთითება პარამეტრი, ახდენს ინტერვალების წაშლას სტრიქონის დასაწყისში და ბოლოში. **TrimStart()** მეთოდის ყველაზე მარტივი ვერსია ახდენს ინტერვალების წაშლას სტრიქონის დასაწყისში. **TrimEnd()** მეთოდის ყველაზე მარტივი ვერსია ახდენს ინტერვალების წაშლას სტრიქონის ბოლოში. ამ მეთოდების სინტაქსია:

```

string Trim()
string TrimStart()
string TrimEnd()

```

ამ მეთოდების მეორე ვერსია საშუალებას გვაძლევს სტრიქონს დასაწყისში და ბოლოში მოვაცილოთ მითითებული სიმბოლოები. ამ ვერსიის სინტაქსია:

```

string Trim(char[ ] სიმბოლოების_მასივი)
string TrimStart(char[ ] სიმბოლოების_მასივი)
string TrimEnd(char[ ] სიმბოლოების_მასივი)

```

მოყვანილ პროგრამაში ხდება **Trim()**, **TrimStart()** და **TrimEnd()** მეთოდებთან მუშაობის დემონსტრირება.

```

{
string striqoni2, striqoni3, striqoni4 = ";;რომანი.,";
char[] simboloebi = { '.', ',', ' ' };

label1.Text = striqoni4.Trim(simboloebi);
striqoni2 = striqoni4.TrimStart(simboloebi); // striqoni2 = "რომანი.,"
striqoni3 = striqoni4.TrimEnd(simboloebi); // striqoni3 = ";;რომანი"
label2.Text = striqoni2.ToString();
label3.Text = striqoni3.ToString();
}

```

მოყვანილ პროგრამაში ხდება **ToLower()** და **ToUpper()** მეთოდებთან მუშაობის

დემონსტრირება.

```
{
string striqoni1 = "ANA SABA";
string striqoni2 = "lika romani";

label15.Text = striqoni1.ToLower();           // "ana saba"
label17.Text = striqoni2.ToUpper();         // "LIKA ROMANI"
}
```

სტრიქონების მასივები

სტრიქონების მასივის გამოცხადებისათვის string სიტყვის შემდეგ კვადრატული ფრჩხილები უნდა მივუთითოთ. ქვემოთ მოყვანილ პროგრამაში ხდება str სტრიქონების მასივის გამოცხადება და ინიციალიზება. ის სამ სტრიქონულ ლიტერალს შეიცავს.

```
{
// პროგრამა 12.6
// პროგრამაში ხდება სტრიქონების მასივთან მუშაობის დემონსტრირება
string[] striqonebis_masivi = { "სტრიქონული ", "მასივის ", "მაგალითი" };
// მასივის გამოტანა label კომპონენტში
for ( int indexi = 0; indexi < striqonebis_masivi.Length; indexi++ )
    label1.Text += striqonebis_masivi[indexi] + " ";
// სტრიქონული მასივის მნიშვნელობების შეცვლა
striqonebis_masivi[0] = "მეორე ";
striqonebis_masivi[1] = "სტრიქონული ";
striqonebis_masivi[2] = textBox1.Text;
label1.Text += "\n";
// მასივის გამოტანა label კომპონენტში
foreach ( string striqoni in striqonebis_masivi )
    label1.Text += striqoni + " ";
}
```

სტრიქონების უცვლელობა

string ტიპის ობიექტების ერთ-ერთი მნიშვნელოვანი თავისებურებაა ის, რომ სტრიქონში ერთხელ შექმნილი სიმბოლოების მიმდევრობა აღარ შეიცვლება, ე.ი. არ შეიძლება ინდექსის გამოყენება სტრიქონის რომელიმე სიმბოლოსათვის ახალი მნიშვნელობის მისანიჭებლად. მაგალითად, დაუშვებელია ასეთი მინიჭება:

```
striqoni1[1] = 'ს';
```

იმისათვის, რომ სტრიქონში სიმბოლოები შევცვალოთ, უნდა შევქმნათ ახალი სტრიქონი, რომელიც საჭირო ცვლილებებს შეიცავს. ამისათვის, შეგვიძლია Replace ან Substring მეთოდის გამოყენება. ქვემოთ მოყვანილ პროგრამაში ხდება სტრიქონის შეცვლის დემონსტრირება.

```
{
// პროგრამა 12.7
// პროგრამაში ხდება სტრიქონის შეცვლის დემონსტრირება
string str1, str2, str3;
str1 = "დაპროგრამება";
// str1[0] = 'ბ'; // ასეთი მინიჭება დაუშვებელია
str2 = str1.Replace('ა', 'ბ'); // str2 სტრიქონში ჩაიწერება შეცვლილი სტრიქონი
str3 = str1.Substring(2, 3); // str3 სტრიქონში ჩაიწერება ახალი სტრიქონი
}
```

```

str1 = str1.Replace("პროგ", "აზგ"); // ასეთი მინიჭება დასაშვებია. ამ შემთხვევაში
// str1 სტრიქონშივე ჩაიწერება შეცვლილი სტრიქონი

label1.Text = str3;
label2.Text = str1;
}

```

დინამიური სტრიქონები

დინამიურ სტრიქონებთან სამუშაოდ გამოიყენება System.Text.StringBuilder კლასი. StringBuilder კლასის მეთოდები მუშაობენ უფრო სწრაფად, რადგან ისინი არ ქმნიან სტრიქონის ასლს და უშუალოდ სტრიქონთან მუშაობენ. დინამიურ სტრიქონებში, String კლასის სტრიქონებისაგან განსხვავებით, შესაძლებელია სიმბოლოების პირდაპირ შეცვლა. პროგრამის დასაწყისში, რომელიც დინამიურ სტრიქონებთან მუშაობს, უნდა მოვათავსოთ დირექტივა:

```
using System.Text;
```

დინამიური სტრიქონის შექმნა

StringBuilder კლასის კონსტრუქტორი გადატვირთულია, ამიტომ არსებობს ამ კლასის კონსტრუქტორის რამდენიმე ვარიანტი. ჩვენ განვიხილავთ დინამიური სტრიქონის შექმნის რამდენიმე გზას. ერთ-ერთი გზაა ცარიელი დინამიური სტრიქონის შექმნა:

```
StringBuilder DinamiuriStriqoni1 = new StringBuilder();
```

თუ კოდის კომპილაციის დროს გამოჩნდა შეტყობინება იმის შესახებ, რომ StringBuilder კლასი არ არის განსაზღვრული, მაშინ ის უნდა მივუთითოთ შემდეგნაირად:

```
System.Text.StringBuilder
```

შექმნის დროს დინამიური სტრიქონის ტევადობაა 16 სიმბოლო. ახალი სიმბოლოს დამატების შემდეგ დინამიური სტრიქონის ტევადობა ავტომატურად იზრდება. ტევადობა შეგვიძლია მივუთითოთ დინამიური სტრიქონის შექმნის დროს. მაგალითად,

```

{
int Tevadoba = 25;
StringBuilder DinamiuriStriqoni2 = new StringBuilder(Tevadoba);
}

```

დინამიური სტრიქონის შექმნისას კონსტრუქტორს შეგვიძლია გადავცეთ, აგრეთვე, მაქსიმალური ტევადობა. მაგალითად,

```

{
int Tevadoba = 25;
int MaxTevadoba = 100;
StringBuilder DinamiuriStriqoni3 = new StringBuilder(Tevadoba, MaxTevadoba);
}

```

შექმნის დროს დინამიურ სტრიქონს ავტომატურად ენიჭება 2147483647-ის ტოლი მაქსიმალური ტევადობა.

დინამიური სტრიქონის შექმნისას კონსტრუქტორს შეგვიძლია გადავცეთ სტრიქონი, რომელიც დინამიურ სტრიქონში უნდა ჩაიწეროს. მაგალითად,

```

{
string Striqoni = "ანა და საბა";
StringBuilder DinamiuriStriqoni4 = new StringBuilder(Striqoni);
}

```

დინამიური სტრიქონის შექმნის დროს კონსტრუქტორს შეგვიძლია გადავცეთ სტრიქონი;

საწყისი ინდექსი, საიდანაც უნდა დაიწყოს სიმბოლოების გადაწერა; გადასაწერი სიმბოლოების რაოდენობა და დინამიური სტრიქონის ტევადობა. მაგალითად,

```
{
string Striqoni = "ანა და საბა";
int Tevadoba = 50;
int SawyisiIndexi = 4;
int SimboloebisRaodenoba = 7;
StringBuilder DinamiuriStriqoni6 = new
    StringBuilder(Striqoni, SawyisiIndexi, SimboloebisRaodenoba, Tevadoba);
label1.Text = DinamiuriStriqoni6;           //   DinamiuriStriqoni6 = "და საბა"
}
```

ცხრილი 12.4. StringBuilder კლასის თვისებები.

თვისება	ტიპი	აღწერა
Capacity	int	გასცემს ან აყენებს დინამიური სტრიქონის ტევადობას
Length	int	გასცემს ან აყენებს დინამიურ სტრიქონში სიმბოლოების რაოდენობას
MaxCapacity	int	გასცემს დინამიური სტრიქონის მაქსიმალურ ტევადობას

ცხრილი 12.5. StringBuilder კლასის მეთოდები.

მეთოდი	დაბრუნებული მნიშვნელობის ტიპი	აღწერა
Append()	StringBuilder	ობიექტის სტრიქონულ წარმოდგენას ამატებს დინამიური სტრიქონის ბოლოში
AppendFormat()	StringBuilder	დაფორმატებულ სტრიქონს ამატებს დინამიური სტრიქონის ბოლოში
EnsureCapacity()	int	ამოწმებს, ტოლია თუ მეტი დინამიური სტრიქონის მიმდინარე ტევადობა მითითებულ მნიშვნელობაზე
Equals()	bool	გასცემს true მნიშვნელობას თუ დინამიური სტრიქონი მითითებული ობიექტის ტოლია, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას
GetType()	Type	გასცემს ობიექტის ტიპს
Insert()	StringBuilder	მითითებული ობიექტის სტრიქონულ წარმოდგენას ამატებს დინამიურ სტრიქონში დაწყებული მითითებული პოზიციიდან
Remove()	StringBuilder	დინამიური სტრიქონიდან შლის მითითებული რაოდენობის სიმბოლოს დაწყებული მითითებული პოზიციიდან
Replace()	StringBuilder	მითითებულ სიმბოლოს ან ქვესტრიქონს დინამიურ სტრიქონში ყველგან ცვლის მითითებული სიმბოლოთი ან ქვესტრიქონით.
ToString()	string	დინამიურ სტრიქონს გარდაქმნის სტრიქონად

StringBuilder კლასის თვისებები და მეთოდები

ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 12.4 და 12.5.

განვიხილოთ ზოგიერთი მათგანი.

Append() და **AppendFormat()** მეთოდები. Append() მეთოდი ახდენს მითითებული ობიექტის სტრიქონად გარდაქმნას და დინამიური სტრიქონისათვის მის დამატებას. ობიექტს შეიძლება ჰქონდეს ნებისმიერი ჩადგმული ტიპი, როგორცაა, bool, byte, int, double და ა.შ. Append() მეთოდის უმარტივეს ვერსიას აქვს შემდეგი სინტაქსი:

StringBuilder.Append(მნიშვნელობა)

აქ stringBuilder არის StringBuilder კლასის ობიექტი, სიდიდე კი დინამიური სტრიქონისათვის დასამატებელი მნიშვნელობა.

მოყვანილ პროგრამაში ხდება სხვადასხვა ტიპის ობიექტების დინამიური სტრიქონისათვის დამატების დემონსტრირება.

```
{
//   პროგრამა 12.8
//   პროგრამაში ხდება დინამიური სტრიქონისათვის ელემენტების დამატება
string striqoni = "საბა და ანა";
int  ricxvi = 5;
double wiladi = 2.8765;
bool logikuri = true;
StringBuilder DinamiuriStriqoni = new StringBuilder();
DinamiuriStriqoni.Append(striqoni);           //   DinamiuriStriqoni = "საბა და ანა"
DinamiuriStriqoni.Append(wiladi);           //   DinamiuriStriqoni = "საბა და ანა2.8765"
DinamiuriStriqoni.Append(logikuri);         //   DinamiuriStriqoni = "საბა და ანა2.8765true"
DinamiuriStriqoni.Append(ricxvi);           //   DinamiuriStriqoni = "საბა და ანა2.8765true5"
label1.Text = DinamiuriStriqoni.ToString();
}
```

Append() მეთოდის მეორე ვერსია საშუალებას გვაძლევს დინამიურ სტრიქონს დავუმატოთ გამეორებადი სიმბოლოს სერია. მეთოდის სინტაქსია:

StringBuilder.Append(სიმბოლო, გამეორების_რაოდენობა)

მაგალითად, DinamiuriStriqoni.Append('ა', 10); მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს დაემატება 10 ცალი 'ა' სიმბოლო.

დინამიურ სტრიქონს შეგვიძლია დავუმატოთ მითითებული სტრიქონის მითითებული სიმბოლოები. შესაბამისი მეთოდის სინტაქსია:

StringBuilder.Append(სტრიქონი, საწყისი_ინდექსი, სიმბოლოების_რაოდენობა)

მაგალითად, DinamiuriStriqoni.Append("ანა და საბა", 4, 7); ფუნქციის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს დაემატება სტრიქონი "და საბა".

AppendFormat() მეთოდი ახდენს დინამიური სტრიქონისათვის დაფორმატებული სტრიქონის დამატებას. მისი სინტაქსია:

AppendFormat(string ფორმატი, object ობიექტი)

აქ ობიექტი წარმოდგენილი იქნება ფორმატის მიხედვით და შემდეგ დაემატება დინამიურ სტრიქონს. მაგალითად:

```
{
double wiladi = 2.8765;
DinamiuriStriqoni.AppendFormat("{0, 5:f2}", wiladi);
label1.Text = DinamiuriStriqoni.ToString();
}
```

AppendFormat() მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს დაემატება სტრიქონი " 2.88". ფორმატში რიცხვი 5 არის ათწილადში თანრიგების რაოდენობა მძიმის ჩათვლით, რიცხვი 2 კი - თანრიგების რაოდენობა წილად ნაწილში. შედეგად, ათწილადის

მთელ ნაწილს დაეთმობა 2 თანრიგი, მძიმეს ერთი, ხოლო წილად ნაწილს კი - 2. 0 მიუთითებს, რომ უნდა დაფორმატდეს პირველი ცვლადი, კერძოდ კი - wiladi.

Insert() მეთოდი ახდენს ობიექტის სტრიქონად გარდაქმნას და დინამიურ სტრიქონში მითითებული პოზიციიდან ჩამატებას. Insert() მეთოდის უმარტივეს ვერსიას აქვს შემდეგი სინტაქსი:

StringBuilder.Insert(ინდექსი, სტრიქონი)

მაგალითად, DinamiuriStriqoni.Insert(5, "საბა, "); მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს მე-5 პოზიციიდან ჩამატება "საბა, " სტრიქონი.

დინამიურ სტრიქონში შეგვიძლია ჩავსვათ, აგრეთვე, ჩასასმელი სტრიქონის რამდენიმე ასლი. Insert() მეთოდის ამ ვერსიის სინტაქსია:

StringBuilder.Insert(ინდექსი, სტრიქონი, გამეორების_რაოდენობა)

მაგალითად, DinamiuriStriqoni.Insert(5, "საბა, ", 4); მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიურ სტრიქონს მე-5 პოზიციიდან 4-ჯერ ჩამატება "საბა, " სტრიქონი.

Remove() მეთოდი. ის დინამიური სტრიქონის მითითებული პოზიციიდან შლის მითითებული რაოდენობის სიმბოლოს. მეთოდის სინტაქსია:

StringBuilder.Remove(ინდექსი, სიმბოლოების_რაოდენობა)

მაგალითი.

```
{
StringBuilder DinamiuriStriqoni1 = new StringBuilder("ანა და საბა");
DinamiuriStriqoni1.Remove(3, 4);           // DinamiuriStriqoni1 = "ანასაბა"
label1.Text = "DinamiuriStriqoni1 = " + DinamiuriStriqoni1.ToString();
}
```

Replace() მეთოდი. ის მითითებულ სიმბოლოს ან სტრიქონს დინამიურ სტრიქონში ყველგან ცვლის ახალი სიმბოლოთი ან სტრიქონით. მეთოდის სინტაქსია:

StringBuilder.Replace(ძველი_მნიშვნელობა, ახალი_მნიშვნელობა)

```
{
StringBuilder DinamiuriStriqoni1 = new StringBuilder("ანა და საბა");
DinamiuriStriqoni1.Replace('ა', 'რ');     // DinamiuriStriqoni1 = "რნრ დრ სრბრ"
label1.Text += "DinamiuriStriqoni1 = " + DinamiuriStriqoni1.ToString();
}
```

ToString() მეთოდი. ის დინამიურ სტრიქონს გარდაქმნის ჩვეულებრივ სტრიქონად. მეთოდის სინტაქსია:

StringBuilder.ToString()

მაგალითად, string striqoni = DinamiuriStriqoni.ToString(); მეთოდის შესრულების შედეგად DinamiuriStriqoni დინამიური სტრიქონი გარდაიქმნება ჩვეულებრივ სტრიქონად.

II ნაწილი. C# ენის სხვა შესაძლებლობები

თავი 13. თარიღი, დრო და დროის ინტერვალები

თარიღი და დრო

თარიღთან და დროსთან სამუშაოდ გამოიყენება System.DateTime სტრუქტურა. მისი თვისებები და მეთოდები მოყვანილია ცხრილებში 13.1 და 13.2. რაც შეეხება Ticks თვისებას, მის ქართულ შესატყვისად შევარჩიე სიტყვა "წიკი", გამომდინარე საათის წიკწიკიდან (რუსულად тик). წიკი არის 100-ნანოწამიანი ინტერვალი.

თავდაპირველად ვნახოთ თუ როგორ ხდება DateTime სტრუქტურის ეგზემპლარის შექმნა, შემდეგ კი განვიხილოთ მისი ზოგიერთი თვისება და მეთოდი. არსებობს DateTime სტრუქტურის კონსტრუქტორის რამდენიმე ვერსია. შესაბამისად, DateTime სტრუქტურის ეგზემპლარი შეგვიძლია რამდენიმე საშუალებით შევქმნათ. ერთ-ერთია კონსტრუქტორისთვის წლის, თვის და დღის გადაცემა. მაგალითი.

```
{  
int weli = 2005;  
int tve = 3;  
int dge = 19;  
// იქმნება Tarigi სტრუქტურა, რომელიც შეიცავს წელს, თვესა და დღეს  
DateTime Tarigi = new DateTime(weli, tve, dge);  
label1.Text = Tarigi.Year.ToString() + " " + Tarigi.Month.ToString() + " " + Tarigi.Day.ToString();  
}
```

კონსტრუქტორს შეიძლება, აგრეთვე, გადავცეთ საათი, წუთი, წამი და მილიწამი. მაგალითი.

```
{  
int weli = 2005;  
int tve = 3;  
int dge = 19;  
int saati = 20;  
int wuti = 30;  
int wami = 25;  
int miliwami = 45;  
// იქმნება Tarigi სტრუქტურა, რომელიც შეიცავს  
// წელს, თვეს, დღეს, საათს, წუთს, წამს და მილიწამს  
DateTime Tarigi = new DateTime(weli, tve, dge, saati, wuti, wami, miliwami);  
label1.Text = Tarigi.Year.ToString() + " " + Tarigi.Month.ToString() + " " + Tarigi.Day.ToString() + " " +  
Tarigi.Hour.ToString() + " " + Tarigi.Minute.ToString() + " " +  
Tarigi.Second.ToString() + " " + Tarigi.Millisecond.ToString();  
}
```

ცხრილი 13.1. DateTime სტრუქტურის თვისებები.

თვისება	ტიპი	აღწერა
Now (სტატიკურია)	DateTime	შეიცავს მიმდინარე თარიღსა და დროს
Today (სტატიკურია)	DateTime	შეიცავს მიმდინარე თარიღს. დროის ველები განულებულია 00:00:00
UtcNow() (სტატიკურია)	DateTime	შეიცავს მიმდინარე თარიღსა და დროს სასაათო სარტყლის გათვალისწინებით
Date	DateTime	შეიცავს თარიღს. დროის ველები განულებულია 00:00:00
Day	int	შეიცავს თვის დღის მნიშვნელობას 1÷31 დიაპაზონში
DayOfWeek	DayOfWeek	შეიცავს კვირის დღის მნიშვნელობას 0 (კვირა) ÷ 6 (შაბათი) დიაპაზონში
DayOfYear	int	შეიცავს წლის დღის ნომერს 1÷366 დიაპაზონში
Hour	int	შეიცავს საათის მნიშვნელობას 0÷23 დიაპაზონში
Millisecond	int	შეიცავს მილიწამის მნიშვნელობას 0÷999 დიაპაზონში
Minute	int	შეიცავს წუთის მნიშვნელობას 0÷59 დიაპაზონში
Month	int	შეიცავს თვის მნიშვნელობას 1÷12 დიაპაზონში
Second	int	შეიცავს წამის მნიშვნელობას 0÷59 დიაპაზონში
Ticks	long	შეიცავს წიკების (100-ნანოწამიანი ინტერვალების) რაოდენობას, რომელიც გავიდა 0001 წლის 1 იანვრიდან ეგზემპლარში დაყენებულ დრომდე
TimeOfDay	TimeSpan	შეიცავს შუალამიდან გასულ დროს
Year	int	შეიცავს წლის მნიშვნელობას 1÷9999 დიაპაზონში

კონსტრუქტორს უკანასკნელ პარამეტრად შეიძლება გადავცეთ კალენდარი - System.Globalization.Calendar კლასის ობიექტი. Calendar კლასი უზრუნველყოფს დროის დაყოფას ინტერვალებად, კერძოდ, წლებად, თვეებად და კვირებად. არსებობს რამდენიმე კლასი, რომლებიც ახდენენ Calendar კლასის მეთოდების რეალიზებას. ეს კლასებია: EastAsianLunisolarCalendar, GregorianCalendar, HebrewCalendar, HijriCalendar, JapaneseCalendar, JapaneseLunisolarCalendar, JulianCalendar, KoreanCalendar, KoreanLunisolarCalendar, PersianCalendar, TaiwanCalendar, TaiwanLunisolarCalendar, ThaiBuddhistCalendar, UmAlQuraCalendar. მოყვანილ მაგალითში კონსტრუქტორს გადაეცემა JulianCalendar კლასის ობიექტი:

```
{
System.Globalization.JulianCalendar Kalendari = new System.Globalization.JulianCalendar();
System.DateTime Tarigi = new System.DateTime(2005, 3, 19, Kalendari);
label1.Text = Tarigi.Year.ToString() + " " + Tarigi.Month.ToString() + " " + Tarigi.Day.ToString();
}
```

ცხრილი 13.2. DateTime სტრუქტურის მეთოდები.

მეთოდი	დასაბრუნებელი ტიპი	აღწერა
Compare() (სტატიკურია)	int	ადარებს DateTime სტრუქტურის ორ ეგზემპლარს.
DaysInMonth() (სტატიკურია)	int	გასცემს დღეების რაოდენობას მითითებული წლის მითითებული თვისათვის
Equals()	bool	ტოლობაზე ადარებს ორ DateTime სტრუქტურას
FromFileTime() (სტატიკურია)	DateTime	გასცემს DateTime სტრუქტურის ეგზემპლარს, რომელიც ფაილის დროითი ჭდის ეკვივალენტურია
FromOADate() (სტატიკურია)	DateTime	გასცემს DateTime სტრუქტურის ეგზემპლარს, რომელიც OLE თარიღის ეკვივალენტურია
IsLeapYear() (სტატიკურია)	bool	გასცემს true-ს, თუ წელი ნაკიანია, წინააღმდეგ შემთხვევაში false-ს
Parse() (სტატიკურია)	DateTime	თარიღისა და დროის სტრიქონულ წარმოდგენას გარდაქმნის DateTime სტრუქტურის ეკვივალენტურ ეგზემპლარად
ParseExact() (სტატიკურია)	DateTime	თარიღისა და დროის სტრიქონულ წარმოდგენას გარდაქმნის DateTime სტრუქტურის ეკვივალენტურ ეგზემპლარად ფორმატის მიხედვით
Add()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს TimeSpan კლასის ეგზემპლარს
AddDays()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს დღეების მითითებულ რაოდენობას
AddHours()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს საათების მითითებულ რაოდენობას
AddMilliseconds()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს მილიწამების მითითებულ რაოდენობას
AddMinutes()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს წუთების მითითებულ რაოდენობას
AddMonths()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს თვეების მითითებულ რაოდენობას
AddSeconds()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს წამების მითითებულ რაოდენობას
AddTicks()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს წიკების მითითებულ რაოდენობას
AddYears()	DateTime	DateTime სტრუქტურის ეგზემპლარს უმატებს წლების მითითებულ რაოდენობას
CompareTo()	int	DateTime სტრუქტურის ეგზემპლარს ადარებს მითითებულ ობიექტთან.
GetDateTimeFormats()	string[]	DateTime სტრუქტურის ეგზემპლარს გარდაქმნის სტრიქონების მასივად, რომელიც შეიცავს ამ სტრუქტურის ყველა შესაძლო ფორმატს
GetType()	Type	გასცემს მიმდინარე ობიექტის ტიპს

ცხრილი 13.2. (გაგრძელება)

Subtract()	TimeSpan	DateTime სტრუქტურის ეგზემპლარს აკლებს ამავე სტრუქტურის ეგზემპლარს ან TimeSpan კლასის ეგზემპლარს
ToFileTime()	long	DateTime სტრუქტურის ეგზემპლარს გარდაქმნის ფაილის დროით ჭდედ
ToLocalTime()	DateTime	გრინვიჩის დროს გარდაქმნის ადგილობრივ დროდ
ToLongDateString()	string	DateTime სტრუქტურის ეგზემპლარის თარიღს გარდაქმნის გრძელი თარიღის შემცველ სტრიქონად
ToLongTimeString()	string	DateTime სტრუქტურის ეგზემპლარის დროს გარდაქმნის გრძელი დროის შემცველ სტრიქონად
ToOADate()	double	DateTime სტრუქტურის ეგზემპლარს გარდაქმნის ეკვივალენტურ OLE თარიღად
ToShortDateString()	string	DateTime სტრუქტურის ეგზემპლარის თარიღს გარდაქმნის მოკლე თარიღის შემცველ სტრიქონად
ToShortTimeString()	string	DateTime სტრუქტურის ეგზემპლარის დროს გარდაქმნის მოკლე დროის შემცველ სტრიქონად
ToString()	string	DateTime სტრუქტურის ეგზემპლარს გარდაქმნის ეკვივალენტურ სტრიქონად
ToUniversalTime()	DateTime	ადგილობრივ დროს გარდაქმნის გრინვიჩის დროდ

ახლა განვიხილოთ ზოგიერთი თვისება და მეთოდი.

Now და **UtcNow** თვისებები. Now თვისება შეიცავს კომპიუტერის სისტემური საათიდან აღებულ თარიღსა და დროს. UtcNow თვისება შეიცავს თარიღსა და დროს UTC (Universal Coordinated Time) ფორმატში, რომელსაც, აგრეთვე, გრინვიჩის დროს (GMT - Greenwich Mean Time) უწოდებენ. ის შეესაბამება ინგლისის ერთ-ერთი დაბის - გრინვიჩის სასაათო სარტყელს. სტანდარტული წყნარი ოკეანის დრო (PST - Pacific Standard Time) გრინვიჩის დროს რვა საათით ჩამორჩება. მოყვანილ პროგრამაში ხდება ამ თვისებებთან მუშაობის დემონსტრირება.

```
{
//   პროგრამა 13.1
//   პროგრამაში ხდება Now და UtcNow თვისებებთან მუშაობის დემონსტრირება
DateTime Tariqi_Dro1 = DateTime.Now;
DateTime Tariqi_Dro2 = DateTime.UtcNow;
label1.Text = Tariqi_Dro1.Day.ToString() + " " + Tariqi_Dro1.Month.ToString() + " " +
Tariqi_Dro1.Year.ToString() + '\n';
label1.Text += Tariqi_Dro2.Day.ToString() + " " + Tariqi_Dro2.Month.ToString() + " " +
Tariqi_Dro2.Year.ToString();
}
```

მოყვანილ პროგრამაში ხდება **Date**, **Day**, **DayOfWeek**, **DayOfYear** და **TimeOfDay** თვისებებთან მუშაობის დემონსტრირება.

```
{
//   პროგრამა 13.2
//   პროგრამაში ხდება Date, Day, DayOfWeek, DayOfYear, TimeOfDay
//   თვისებებთან მუშაობის დემონსტრირება
DateTime Tariqi_Dro1 = DateTime.Now;
label1.Text = Tariqi_Dro1.Date.ToString();
label1.Text += Tariqi_Dro1.Day.ToString();
}
```

```

label1.Text += Tarigi_Dro1.DayOfWeek.ToString();
label1.Text += Tarigi_Dro1.DayOfYear.ToString();
label1.Text += Tarigi_Dro1.TimeOfDay.ToString();
}

```

Compare() მეთოდი ადარებს DateTime სტრუქტურის ორ ეგზემპლარს. თუ პირველი ეგზემპლარი მეტია მეორეზე, მაშინ გაიცემა 1. თუ პირველი ეგზემპლარი ნაკლებია მეორეზე, მაშინ გაიცემა -1. თუ ეგზემპლარები ტოლია, მაშინ გაიცემა 0. მეთოდის სინტაქსია:

DateTime.Compare(თარიღი_დრო1, თარიღი_დრო2)

აქ თარიღი_დრო1 და თარიღი_დრო2 არის DateTime სტრუქტურის ეგზემპლარები, რომელთა შედარებაც ხდება. მაგალითი.

```

{
DateTime Tarigi_Dro1 = DateTime.Now;
DateTime Tarigi_Dro2 = new DateTime(2007, 1, 27);
int Shedegi = DateTime.Compare(Tarigi_Dro1, Tarigi_Dro2);
switch ( Shedegi )
{
case -1 : label1.Text = "თარიღები ტოლია"; break;
case 0 : label1.Text = "პირველი თარიღი მეტია მეორეზე"; break;
case 1 : label1.Text = " პირველი თარიღი ნაკლებია მეორეზე "; break;
}
}

```

DateTime სტრუქტურის ეგზემპლარების შედარება შეიძლება, აგრეთვე, შედარების გადატვირთული ოპერატორებით: >, >=, <, <=, ==, !=. მაგალითად,

```

if ( Tarigi_Dro1 == Tarigi_Dro2 ) label1.Text = "თარიღები ტოლია";
else label1.Text = "თარიღები არ არის ტოლი";

```

Equals() მეთოდი ადარებს DateTime სტრუქტურის ორ ეგზემპლარს. თუ ისინი ტოლია, მაშინ გაიცემა true, წინააღმდეგ შემთხვევაში - false. ამ მეთოდს აქვს როგორც სტატიკური, ისე ჩვეულებრივი ვერსია. სტატიკური ვერსიის სინტაქსია:

DateTime.Equals(თარიღი_დრო1, თარიღი_დრო2)

აქ თარიღი_დრო1 და თარიღი_დრო2 არის DateTime სტრუქტურის ეგზემპლარები. მაგალითი.

```

{
DateTime Tarigi_Dro1 = DateTime.Now;
DateTime Tarigi_Dro2 = new DateTime(2007, 1, 27);
bool Shedegi = DateTime.Equals(Tarigi_Dro1, Tarigi_Dro2);
if (Shedegi == true) label1.Text = "თარიღები ტოლია";
else label1.Text = "თარიღები არ არის ტოლი";
}

```

Equals() მეთოდის ჩვეულებრივი ვერსიის სინტაქსია:

თარიღი_დრო1.Equals(თარიღი_დრო2)

მაგალითი.

```

{
DateTime Tarigi_Dro1 = DateTime.Now;
DateTime Tarigi_Dro2 = new DateTime(2007, 1, 27);
bool Shedegi = Tarigi_Dro1.Equals(Tarigi_Dro2);
if ( Shedegi == true ) label1.Text = "თარიღები ტოლია";
else label1.Text = "თარიღები არ არის ტოლი";
}

```

DaysInMonth() მეთოდი გასცემს მითითებულ თვეში დღეების რაოდენობას. მისი სინტაქსია:

DateTime.DaysInMonth(წელი, თვე)

მაგალითი.

```
{
int DgeebisRaodenoba = DateTime.DaysInMonth(2007, 2);
label1.Text = DgeebisRaodenoba.ToString(); // DgeebisRaodenoba = 28
}
```

IsLeapYear() მეთოდი გასცემს true-ს თუ თვე ნაკიანია და false-ს წინააღმდეგ შემთხვევაში. მისი სინტაქსია:

DateTime.IsLeapYear(წელი)

მაგალითი.

```
{
bool ArisNakiani = DateTime.IsLeapYear(2007);
if ( ArisNakiani == true ) label1.Text = "წელი ნაკიანია";
else label1.Text = "წელი არ არის ნაკიანი";
}
```

Parse() მეთოდი თარიღისა და დროის შემცველ სტრიქონს გარდაქმნის DateTime სტრუქტურის ეკვივალენტურ ეგზემპლარად. მისი სინტაქსია:

DateTime.Parse(სტრიქონი)

მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Parse("10/5/2000");
DateTime Tarigi_Dro2 = DateTime.Parse("10/5/2000 8:30:35");
label1.Text = Tarigi_Dro1.ToString() + " " + Tarigi_Dro2.ToString();
}
```

Add() და **Subtract()** მეთოდები. Add() მეთოდი DateTime სტრუქტურის ეგზემპლარს უმატებს TimeSpan კლასის ობიექტს. Subtract() მეთოდი DateTime სტრუქტურის ეგზემპლარს აკლებს TimeSpan კლასის ობიექტს. ამ მეთოდების სინტაქსია:

თარიღი_დრო.Add(დროის_ინტერვალი)

თარიღი_დრო.Subtract(დროის_ინტერვალი)

აქ თარიღი_დრო არის DateTime სტრუქტურის ეგზემპლარი, დროის_ინტერვალი კი - TimeSpan კლასის ობიექტი. მაგალითი.

```
{
TimeSpan Drois_Intervali = new TimeSpan(1, 5, 10, 20);
DateTime Tarigi_Dro1 = new DateTime(2007, 5, 15, 17, 35, 30);
DateTime Tarigi_Dro2 = Tarigi_Dro1.Add(Drois_Intervali); // 16.05.2007 22:45:50
DateTime Tarigi_Dro3 = Tarigi_Dro1.Subtract(Drois_Intervali); // 14.05.2007 12:25:10
label1.Text = Tarigi_Dro2.ToString() + " " + Tarigi_Dro3.ToString();
}
```

თავდაპირველად პროგრამში იქმნება დროის ინტერვალი: 1 დღე, 5 საათი, 10 წუთი და 20 წამი. შემდეგ იქმნება Tarigi_Dro1 სტრუქტურა: 2007 წლის 15 მაისი, 17 საათი, 35 წუთი და 30 წამი. Add() მეთოდი Tarigi_Dro1 სტრუქტურას დაუმატებს Drois_Intervali დროის ინტერვალს. შედეგად, Tarigi_Dro2 სტრუქტურას მიენიჭება მნიშვნელობა - 16.05.2007 22:45:50. Subtract() მეთოდი Tarigi_Dro1 სტრუქტურას გამოაკლებს Drois_Intervali დროის ინტერვალს. შედეგად, Tarigi_Dro3 სტრუქტურას მიენიჭება მნიშვნელობა - 14.05.2007 12:25:10.

ცხრილი 13.3. საფორმატო სტრიქონის კომპონენტები.

ფორმატის ელემენტი	აღწერა
d	თვის დღე წინა ნულის გარეშე დღის ნომრებისათვის, რომლებიც ერთი ციფრისგან შედგება
Dd	თვის დღე წინა ნულით დღის ნომრებისათვის, რომლებიც ერთი ციფრისგან შედგება
ddd	კვირის დღის შემოკლებული დასახელება
dddd	კვირის დღის სრული დასახელება
f	წამის ნაწილი ერთი ციფრის სიზუსტით. სიზუსტის გასაზრდელად შეგვიძლია გამოვიყენოთ შვიდი "f" სიმბოლო. დანარჩენი ციფრები არ გამოჩნდება.
M	თვის ნომერი წინა ნულის გარეშე თვის ნომრებისათვის, რომლებიც ერთი ციფრისგან შედგება
MM	თვის ნომერი წინა ნულით თვის ნომრებისათვის, რომლებიც ერთი ციფრისგან შედგება
MMM	თვის შემოკლებული დასახელება
MMMM	თვის სრული დასახელება
y	წელი ასწლეულის გარეშე. 10-ზე ნაკლები მნიშვნელობა აისახება წინა ნულის გარეშე
Yy	წელი ასწლეულის გარეშე. 10-ზე ნაკლები მნიშვნელობა აისახება წინა ნულით
yyyy	წლის სრული ნომერი შემდგარი ოთხი ციფრისგან
Gg	ერა. ეს ელემენტი იგნორირდება, თუ თარიღი არ შეიცავს ერას
h	საათი თორმეტსაათიანი წარმოდგენით, წინა ნულის გარეშე, ერთი ციფრისგან შემდგარი საათებისთვის
Hh	საათი თორმეტსაათიანი წარმოდგენით, წინა ნულით, ერთი ციფრისგან შემდგარი საათებისთვის
H	საათი, ოცდაოთხსაათიანი წარმოდგენით, წინა ნულის გარეშე, ერთი ციფრისგან შემდგარი საათებისთვის
HH	საათი, ოცდაოთხსაათიანი წარმოდგენით, წინა ნულით, ერთი ციფრისგან შემდგარი საათებისთვის
m	წუთები, წინა ნულის გარეშე, ერთი ციფრისგან შემდგარი წუთებისთვის
mm	წუთები, წინა ნულით, ერთი ციფრისგან შემდგარი წუთებისთვის
s	წამები, წინა ნულის გარეშე, ერთი ციფრისგან შემდგარი წამებისთვის
ss	წამები, წინა ნულით, ერთი ციფრისგან შემდგარი წამებისთვის
t	A.M./P.M მიმთითებლის პირველი სიმბოლო
tt	A.M./P.M სრული მიმთითებელი
zz	სასაათო სარტყლის წანაცვლება, წინა ნულით, ერთი ციფრისგან შემდგარი წანაცვლებისთვის
zzz	სასაათო სარტყლის წანაცვლება შემდგარი საათებისა და წუთებისაგან. ერთი ციფრისგან შემდგარი მნიშვნელობები შეივსება ნულით

იგივე ოპერაციები შეგვიძლია შევასრულოთ გადატვირთული "+" და "-" ოპერატორების გამოყენებით. მაგალითი.

DateTime Tarigi_Dro4 = Tarigi_Dro1 + Drois_Intervali;

DateTime Tarigi_Dro5 = Tarigi_Dro1 - Drois_Intervali;

AddYears(), AddMonths(), AddDays(), AddHours() და **AddMinutes()** მეთოდები. ეს მეთოდები

DateTime სტრუქტურის ეგზემპლარს შესაბამისად უმატებს წლებს, თვეებს, დღეებს, საათებსა და წუთებს. ამ მეთოდებს აქვთ double ტიპის პარამეტრი. მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
label1.Text = Tarigi_Dro1.ToString();
Tarigi_Dro1 = Tarigi_Dro1.AddYears(3);
Tarigi_Dro1 = Tarigi_Dro1.AddMonths(2);
Tarigi_Dro1 = Tarigi_Dro1.AddDays(10);
Tarigi_Dro1 = Tarigi_Dro1.AddHours(7);
Tarigi_Dro1 = Tarigi_Dro1.AddMinutes(40);
label1.Text += Tarigi_Dro1.ToString();
}
```

მოყვანილ პროგრამაში ხდება **ToLongDateString()**, **ToShortDateString()**, **ToLongTimeString()** და **ToShortTimeString()** მეთოდების გამოყენების დემონსტრირება.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
label1.Text = Tarigi_Dro1.ToLongDateString() + '\n';
label1.Text += Tarigi_Dro1.ToShortDateString()+ '\n';
label1.Text += Tarigi_Dro1.ToLongTimeString() + '\n';
label1.Text += Tarigi_Dro1.ToShortTimeString();
}
```

String() მეთოდი. მისი გადატვირთული ვერსია ახდენს DateTime სტრუქტურის ეგზემპლარის გარდაქმნას ეკვივალენტურ სტრიქონად. მისი სინტაქსია:

თარიღი_დრო.ToString()

მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
label1.Text = Tarigi_Dro1.ToString();
}
```

ToString() მეთოდს შეგვიძლია პარამეტრად გადავცეთ ფორმატი (სტრიქონი), რომლის მიხედვითაც შესრულდება DateTime სტრუქტურის ეგზემპლარის გარდაქმნა. ToString() მეთოდის ამ ვერსიის სინტაქსია:

თარიღი_დრო.ToString(ფორმატი)

მაგალითი.

```
{
DateTime Tarigi_Dro1 = DateTime.Now;
label1.Text = Tarigi_Dro1.ToString() + '\n';
label1.Text += Tarigi_Dro1.ToString("MM/dd/yy");
label1.Text += Tarigi_Dro1.ToString("D");
}
```

პროგრამის შესრულების შედეგად მიიღება სტრიქონი, მაგალითად, "02.03.07". აქ 02 არის თვე, 03 - დღე, 07 კი - წელი. კოდის უკანასკნელ სტრიქონში გამოყენებულია ჩადგმული ფორმატებიდან ერთ-ერთი, კერძოდ კი - "D".

ცხრილებში 13.3 და 13.4 შესაბამისად მოყვანილია საფორმატო სტრიქონის კომპონენტები და თარიღისა და დროის ჩადგმული ფორმატები.

ცხრილი 13.4. თარიღისა და დროის ჩადგმული ფორმატები.

სიმბოლო	ფორმატი	მაგალითი
d	MM/dd/yyyy	03.19.2007
D	dddd, MMMM dd, yyyy	2007 წლის 04 02, კვირა
f	dddd, MMMM dd, yyyy HH:mm	2007 წლის 04 02, კვირა 9:36
F	dddd, MMMM dd, yyyy HH:mm:ss	2007 წლის 04 02, კვირა 9:36:51
g	MM/dd/yyyy/ HH:mm	04.02.2007 9:36:51
G	MM/dd/yyyy HH:mm:ss	04.02.2007 9:36
m, M	MMMM dd	04 02
r, R	ddd, dd MM yyyy HH':'mm':'ss 'GMT'	Sun, 04 Feb 2007 09:42:17 GMT
s	yyyy'-'MM'-'dd'T'HH':'mm':'ss	2007-02-04T09:42:17
t	HH:mm	9:42
T	HH:mm:ss	9:42:17
u	yyyy'-'MM'-'dd HH':'mm':'ss'Z'	2007-02-04 09:42:17Z
U	dddd, MMMM dd, yyyy HH:mm:ss	2007 წლის 04 02, კვირა 5:42:17
y, Y	yyyy, MMMM	თებერვალი 2007

დროის ინტერვალები

თარიღისა და დროის გარდა შეგვიძლია, აგრეთვე, ვიმუშაოთ დროის ინტერვალებთან. დროის ინტერვალებთან სამუშაოდ გამოიყენება System.TimeSpan კლასი. ამ კლასის კონსტრუქტორს აქვს რამდენიმე გადატვირთული ვერსია. მათი სინტაქსია:

TimeSpan(int საათი, int წუთი, int წამი)

TimeSpan(int დღე, int საათი, int წუთი, int წამი)

TimeSpan(int დღე, int საათი, int წუთი, int წამი, int მილიწამი)

TimeSpan(int წიკი)

მაგალითი.

```
{
// დროითი ინტერვალის შექმნა საათის, წუთისა და წამის მითითებით
TimeSpan Drois_Interval1 = new TimeSpan(5, 15, 45);
// დროითი ინტერვალის შექმნა დღის, საათის, წუთისა და წამის მითითებით
TimeSpan Drois_Interval2 = new TimeSpan(2, 5, 15, 45);
// დროითი ინტერვალის შექმნა დღის, საათის, წუთის, წამისა და მილიწამის მითითებით
TimeSpan Drois_Interval3 = new TimeSpan(2, 5, 15, 45, 80);
// დროითი ინტერვალის შექმნა წიკის მითითებით
TimeSpan Drois_Interval4 = new TimeSpan(1000);
}
```

TimeSpan კლასის ობიექტი შეგვიძლია დავუმატოთ DateTime სტრუქტურის ეგზემპლარს.

მაგალითად

```
{
DateTime Tarigi_Dro = new DateTime(2007, 04, 12);
TimeSpan Drois_Interval1 = new TimeSpan(2, 5, 15, 45);
Tarigi_Dro += Drois_Interval1; // 14.04.2007 5:15:45
label1.Text = Tarigi_Dro.ToString();
}
```

განვიხილოთ ზოგიერთი თვისება და მეთოდი.

Days, Hours, Minutes, Seconds, Milliseconds და **Ticks** თვისებებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```
{
//   პროგრამა 13.3
//   პროგრამაში ხდება Days, Hours, Minutes, Seconds, Milliseconds და
//   Ticks თვისებებთან მუშაობის დემონსტრირება
TimeSpan Drois_Interval1 = new TimeSpan(5, 15, 45);
label1.Text = Drois_Interval1.Days.ToString() + '\n';
label1.Text += Drois_Interval1.Hours.ToString() + '\n';
label1.Text += Drois_Interval1.Minutes.ToString() + '\n';
label1.Text += Drois_Interval1.Seconds.ToString() + '\n';
label1.Text += Drois_Interval1.Milliseconds.ToString() + '\n';
label1.Text += Drois_Interval1.Ticks.ToString() + '\n';
}
```

შედეგში Days და Milliseconds თვისებებს ექნებათ ნულოვანი მნიშვნელობები, რადგან Drois_Interval1 ობიექტის შექმნის დროს მათი მნიშვნელობები არ იყო მითითებული.

TimeSpan კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 13.5 და 13.6.

ცხრილი 13.5. TimeSpan კლასის თვისებები.

თვისება	ტიპი	აღწერა
Days	int	შეიცავს დღეების რაოდენობას TimeSpan კლასის ობიექტში
Hours	int	შეიცავს საათების რაოდენობას TimeSpan კლასის ობიექტში
Milliseconds	int	შეიცავს მილიწამების რაოდენობას TimeSpan კლასის ობიექტში
Minutes	int	შეიცავს წუთების რაოდენობას TimeSpan კლასის ობიექტში
Seconds	int	შეიცავს წამების რაოდენობას TimeSpan კლასის ობიექტში
Ticks	long	შეიცავს წიკების რაოდენობას TimeSpan კლასის ობიექტში
TotalDays	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას დღეებში
TotalHours	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას საათებში
TotalMilliseconds	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას მილიწამებში
TotalMinutes	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას წუთებში
TotalSeconds	double	შეიცავს TimeSpan კლასის ობიექტის ხანგრძლივობას წამებში

ცხრილი 13.6. TimeSpan კლასის მეთოდები.

მეთოდი	დაბრუნებული მნიშვნელობის ტიპი	აღწერა
Compare() (სტატიკური)	int	ადარებს TimeSpan კლასის ორ ობიექტს. გასცემს შესაბამის რიცხვს
Equals()	bool	ადარებს TimeSpan კლასის ორ ობიექტს. გასცემს შესაბამის ლოგიკურ მნიშვნელობას
FromDays() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს დღეების მითითებულ რაოდენობას
FromHours() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს საათების მითითებულ რაოდენობას

ცხრილი 13.6. (გაგრძელება)

FromMilliseconds() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს მილიწამების მითითებულ რაოდენობას
FromMinutes() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს წუთების მითითებულ რაოდენობას
FromSeconds() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს წამების მითითებულ რაოდენობას
FromTicks() (სტატიკური)	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომელიც შეიცავს წიკების მითითებულ რაოდენობას
Parse() (სტატიკური)	TimeSpan	დროის ინტერვალის მითითებულ სტრიქონულ წარმოდგენას გარდაქმნის TimeSpan კლასის ეკვივალენტურ ობიექტად
Add()	TimeSpan	TimeSpan კლასის მოცემულ ობიექტს უმატებს ამავე კლასის ობიექტს
CompareTo()	int	ადარებს TimeSpan კლასის ორ ობიექტს. გასცემს შესაბამის რიცხვს
Duration()	TimeSpan	გასცემს დადებითი ხანგრძლივობის TimeSpan კლასის ობიექტს, რომელიც სიდიდით მიმდინარე ობიექტის ხანგრძლივობის ტოლია
GetType()	Type	გასცემს მიმდინარე ობიექტის ტიპს
Negate()	TimeSpan	გასცემს TimeSpan კლასის ობიექტს, რომლის მნიშვნელობა მიმდინარე ობიექტის მნიშვნელობის საწინააღმდეგოა
Subtract()	TimeSpan	TimeSpan კლასის მოცემულ ობიექტს აკლებს ამავე კლასის ობიექტს
ToString()	string	TimeSpan კლასის მოცემულ ობიექტს გარდაქმნის ეკვივალენტურ სტრიქონად

TotalDays, **TotalHours**, **TotalMinutes**, **TotalSeconds** და **TotalMilliseconds** თვისებებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```

{
//   პროგრამა 13.4
//   პროგრამაში ხდება TotalDays, TotalHours, TotalMinutes, TotalSeconds და
//   TotalMilliseconds თვისებებთან მუშაობის დემონსტრირება
TimeSpan Drois_Interval1 = new TimeSpan(5, 15, 45);
label1.Text = Drois_Interval1.TotalDays.ToString() + '\n';
label1.Text += Drois_Interval1.TotalHours.ToString() + '\n';
label1.Text += Drois_Interval1.TotalMinutes.ToString() + '\n';
label1.Text += Drois_Interval1.TotalSeconds.ToString() + '\n';
label1.Text += Drois_Interval1.TotalMilliseconds.ToString() + '\n';
}

```

FromDays(), **FromHours()**, **FromMinutes()**, **FromSeconds()**, **FromMilliseconds()** და **FromTicks()** მეთოდების პარამეტრებს აქვთ double ტიპი. ამ მეთოდებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```

{

```



```
// პროგრამა 13.5
// პროგრამაში ხდება FromDays(), FromHours(), FromMinutes(), FromSeconds(),
// FromMilliseconds() და FromTicks() მეთოდებთან მუშაობის დემონსტრირება
TimeSpan Drois_Intervali1 = TimeSpan.FromDays(7);
TimeSpan Drois_Intervali2 = TimeSpan.FromHours(7);
TimeSpan Drois_Intervali3 = TimeSpan.FromMinutes(7);
TimeSpan Drois_Intervali4 = TimeSpan.FromSeconds(7);
TimeSpan Drois_Intervali5 = TimeSpan.FromMilliseconds(7);
TimeSpan Drois_Intervali6 = TimeSpan.FromTicks(7);
label1.Text = Drois_Intervali1.ToString() + '\n';
label1.Text += Drois_Intervali2.ToString() + '\n';
label1.Text += Drois_Intervali3.ToString() + '\n';
label1.Text += Drois_Intervali4.ToString() + '\n';
label1.Text += Drois_Intervali5.ToString() + '\n';
label1.Text += Drois_Intervali6.ToString() + '\n';
}
```

ცხრილი 13.7. Parse() მეთოდის ფორმატის ელემენტები.

ელემენტი	აღწერა
ws	არააუცილებელი " " (ინტერვალის სიმბოლო)
-	არააუცილებელი "-", რომელიც აღნიშნავს, რომ ინტერვალის უარყოფითია
d	დღეების არააუცილებელი რაოდენობა
hh	საათი ოცდაოთხსაათიან ფორმატში
mm	წუთები
ss	წამები
ff	წამის არააუცილებელი ნაწილები. ის შეიძლება შეიცავდეს 1÷7 სიმბოლოს

Parse() მეთოდი. მისთვის პარამეტრად გადაცემულ სტრიქონს უნდა ჰქონდეს შემდეგი ფორმატი:

[ws] [-] [d.]hh:mm:ss[.ff][ws]

კვადრატულ ფრჩხილებში მოთავსებული ელემენტების მითითება არ არის აუცილებელი. ფორმატის ელემენტები მოყვანილია ცხრილში 13.7. მაგალითი.

```
{
// პროგრამა 13.6
// პროგრამაში ხდება Parse() მეთოდის ფორმატის ელემენტებთან მუშაობის
// დემონსტრირება
// იქმნება Drois_Intervali1 ობიექტი, რომლის ხანგრძლივობაა 9 საათი, 15 წუთი და 45 წამი
TimeSpan Drois_Intervali1 = TimeSpan.Parse("9:15:45");
// იქმნება Drois_Intervali2 ობიექტი, რომლის ხანგრძლივობაა
// 2 დღე, 9 საათი, 15 წუთი და 45.50 წამი
TimeSpan Drois_Intervali2 = TimeSpan.Parse("2.9:15:45.50");
label1.Text = Drois_Intervali1.ToString() + '\n';
label1.Text += Drois_Intervali2.ToString() + '\n';
}
```

Add() და **Subtract()** მეთოდებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```

{
TimeSpan Drois_Intervali1 = new TimeSpan(2, 15, 25);
TimeSpan Drois_Intervali2 = new TimeSpan(12, 20, 30);
TimeSpan Drois_Intervali3 = Drois_Intervali2.Add(Drois_Intervali1);
TimeSpan Drois_Intervali4 = Drois_Intervali2.Subtract(Drois_Intervali1);
label1.Text = Drois_Intervali3.ToString() + '\n';           // შედეგი - 14:35:55
label1.Text += Drois_Intervali4.ToString() + '\n';         // შედეგი - 10:05:05
}

```

Duration() და **Negate()** მეთოდებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```

{
TimeSpan Drois_Intervali1 = new TimeSpan(2, 15, 25);
TimeSpan Drois_Intervali2 = Drois_Intervali1.Duration();
TimeSpan Drois_Intervali3 = Drois_Intervali1.Negate();
label1.Text = Drois_Intervali2.ToString() + '\n';         // შედეგი - 02:15:25
label1.Text += Drois_Intervali3.ToString() + '\n';         // შედეგი - -02:15:25
}

```

თავი 14. კოლექციები

მე-4 თავში ჩვენ განვიხილეთ მასივები. ისინი ფართოდ გამოიყენებიან სხვადასხვა ხასიათის ამოცანების გადაწყვეტად. ამავე დროს, მათ აქვთ შემდეგი შეზღუდვები:

- შექმნის შემდეგ მასივს ენიჭება ფიქსირებული ზომა, რომელსაც ვეღარ შევცვლით.
- შეუძლებელია მასივში ელემენტის ჩამატება ან წაშლა.
- მასივის ელემენტებთან მიმართვა შესაძლებელია მხოლოდ ინდექსის საშუალებით.
- მასივის ელემენტებს ერთი და იგივე ტიპი აქვთ.

ეს შეზღუდვები მოხსნილია კოლექციებში. ჩვენ განვიხილავთ შემდეგ კოლექციებს: დინამიური მასივები, ჰეშ-ცხრილები, დახარისხებული სიები, რიგები, სტეკები და ბიტების მასივები. კოლექციების აღმწერი კლასები გამოცხადებულია System.Collections სახელების სივრცეში.

ამ განყოფილებაში მოყვანილი ყველა პროგრამის დასაწყისში უნდა მოვათავსოთ დირექტივა using System.Collections;

დინამიური მასივები

ჩვეულებრივი მასივებისაგან განსხვავებით, დინამიური მასივი მასში ელემენტების დამატების ან წაშლის შემთხვევაში ზომას იცვლის. დინამიურ მასივებში ელემენტების ნუმერაცია ნულიდან იწყება. ArrayList კლასში განსაზღვრულია თვისებები და მეთოდები დინამიური მასივების ელემენტებთან სამუშაოდ. ზოგიერთი მათგანი მოყვანილია ცხრილებში 14.1 და 14.2.

დინამიური მასივის შექმნა. ArrayList კონსტრუქტორს სამი გადატვირთული ვერსია აქვს. შესაბამისად, დინამიური სტრიქონი სამი გზით შეგვიძლია შევქმნათ. ამ ვერსიების სინტაქსია:

ArrayList()

ArrayList(ICollection კოლექცია)

ArrayList(int ტევადობა)

მაგალითი.

```
{
string[] Striqoni = { "ანა", "საბა", "ლიკა", "რომანი" };
int[] masivi = new int[] { 1, 2, 3, 4, 5 };
ArrayList DinamiuriMasivi1 = new ArrayList();
ArrayList DinamiuriMasivi2 = new ArrayList(masivi);
ArrayList DinamiuriMasivi3 = new ArrayList(Striqoni);
ArrayList DinamiuriMasivi4 = new ArrayList(50);
foreach (int x in DinamiuriMasivi2)
    label1.Text += x.ToString() + " ";
foreach (string x in DinamiuriMasivi3)
    label2.Text += x + " ";
}
```

ცხრილი 14.1. ArrayList კლასის თვისებები

თვისება	ტიპი	აღწერა
Capacity	int	განსაზღვრავს დინამიური მასივის ტევადობას, ანუ ელემენტების მაქსიმალურ რაოდენობას
Count	int	შეიცავს დინამიური მასივში ელემენტების რაოდენობას
IsFixedSize	bool	მიუთითებს, არის თუ არა დინამიური მასივი ფიქსირებული ზომის
IsReadOnly	bool	მიუთითებს, არის თუ არა დინამიური მასივი მხოლოდ წაკითხვადი
Item	object	გასცემს ან აყენებს მითითებული ინდექსის მქონე ელემენტის მნიშვნელობას

ცხრილი 14.2. ArrayList კლასის მეთოდები

მეთოდი	დაბრუნებული მნიშვნელობის ტიპი	აღწერა
ReadOnly()	ArrayList	გასცემს დინამიურ მასივს მხოლოდ წაკითხვის სათვის
Repeat()	ArrayList	გასცემს დინამიური მასივს, რომლის ყველა ელემენტი გადაცემული მნიშვნელობების ტოლია
Add()	int	დინამიურ მასივს ბოლოში უმატებს ელემენტს
AddRange()	void	დინამიურ მასივს ბოლოში უმატებს სხვა კოლექციის ელემენტებს
BinarySearch()	int	დახარისხებულ მასივში ასრულებს მითითებული ელემენტის ორობით ძებნას.
Clear()	void	დინამიური მასივიდან შლის ყველა ელემენტს
Contains()	bool	განსაზღვრავს შეიცავს თუ არა დინამიური მასივი მითითებულ ელემენტს
CopyTo()	void	დინამიური მასივის ელემენტებს გადაწერს ერთ-განზომილებიან მასივში
Equals()	bool	განსაზღვრავს ტოლია თუ არა ორი დინამიური მასივი
GetEnumerator()	IEnumerator	გასცემს ჩამომთვლელს
GetRange()	ArrayList	გასცემს დინამიური მასივს, რომელიც შეიცავს საწყისი მასივის ელემენტების დიაპაზონს
GetType()	Type	გასცემს მიმდინარე ობიექტის ტიპს
IndexOf()	int	გასცემს ნაპოვნი ელემენტის პოზიციის ნომერს
Insert()	void	ახდენს დინამიურ მასივში ელემენტის ჩასმას მითითებული პოზიციიდან
InsertRange()	void	ახდენს დინამიურ მასივში კოლექციის ელემენტების ჩასმას მითითებული პოზიციიდან
LastIndexOf()	int	გასცემს ნაპოვნი ელემენტის უკანასკნელი პოზიციის ნომერს

ცხრილი 14.2 (გაგრძელება)

Remove()	void	დინამიური მასივიდან შლის პირველ ნაპოვნ ელემენტს
RemoveAt()	void	დინამიური მასივიდან შლის მითითებული პოზიციის მქონე ელემენტს
RemoveRange()	void	დინამიური მასივიდან შლის ელემენტების დიაპაზონს დაწყებული მითითებული პოზიციიდან
Reverse()	void	დინამიური მასივის ელემენტებს უკუ მიმდევრობით ალაგებს
SetRange()	void	დინამიური მასივის შესაბამის დიაპაზონში ახდენს კოლექციის ელემენტების ჩასმას
Sort()	void	ახდენს დინამიური მასივის ან მისი დიაპაზონის დახარისხებას
ToArray()	object[]	დინამიური მასივის ელემენტებს გადაწერს ჩვეულებრივ მასივში
ToString()	string	მიმდინარე ობიექტს გარდაქმნის სტრიქონად
TrimToSize()	void	ამცირებს დინამიური მასივის ტევადობას ელემენტების იმ რაოდენობამდე, რომელიც აქვს ამ მასივს მეთოდის გამოძახების მომენტში

ელემენტების დამატება და ჩამატება. **Add()** მეთოდი დინამიურ მასივს ბოლოში უმატებს მითითებულ ელემენტს, რომლის ტიპია object. მისი სინტაქსია:

ArrayList.Add(სიდიდე)

აქ *ArrayList* აღნიშნავს *ArrayList* ტიპის ობიექტს.

მოყვანილ პროგრამაში ხდება Add მეთოდთან მუშაობის დემონსტრირება.

```

{
//   პროგრამა 14.1
//   პროგრამაში ხდება Add მეთოდთან მუშაობის დემონსტრირება
ArrayList DinamiuriMasivi = new ArrayList();
//
DinamiuriMasivi.Add("საბა");
DinamiuriMasivi.Add("ანა");
DinamiuriMasivi.Add("ლიკა");
DinamiuriMasivi.Add("რომანი");
//
for ( int indexi = 0; indexi < DinamiuriMasivi.Count; indexi++ )
    label1.Text += "DinamiuriMasivi [" + indexi.ToString() + "] = " +
        DinamiuriMasivi[indexi].ToString() + "\n";
}

```

დინამიურ მასივს შეგვიძლია დავუმატოთ სხვადასხვა ტიპის მონაცემები. მაგალითი.

```

{
//
ArrayList DinamiuriMasivi = new ArrayList();
//
DinamiuriMasivi.Add(1);
DinamiuriMasivi.Add(2.2);
DinamiuriMasivi.Add(true);
}

```

```
DinamiuriMasivi.Add("რომანი");
for (int indexi = 0; indexi < DinamiuriMasivi.Count; indexi++)
    label1.Text += "DinamiuriMasivi[" + indexi.ToString() + "] = " +
        DinamiuriMasivi[indexi].ToString() + "\n";
}
```

AddRange() მეთოდი დინამიურ მასივს ბოლოში უმატებს ელემენტების დიაპაზონს. მისი სინტაქსია:

arrayList.Insert(კოლექცია)

მეთოდის შესულების შედეგად კოლექცია დაემატება arrayList დინამიურ მასივს. მაგალითი.

```
{
string[] striqoni1 = { "ანა", "საბა", "ლიკა", "რომანი", "გიორგი" };
string[] striqoni2 = { "ნატა", "ბექა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni1);
DinamiuriMasivi1.AddRange(striqoni2); // DinamiuriMasivi1 = "ანა", "საბა", "ლიკა",
// "რომანი", "გიორგი", "ნატა", "ბექა"
for (int index = 0; index < DinamiuriMasivi1.Count; index++)
    label1.Text += DinamiuriMasivi1[index] + " ";
}
```

Insert() მეთოდი ახდენს მითითებული ელემენტის ჩასმას მითითებულ პოზიციაში. მისი სინტაქსია:

arrayList.Insert(ინდექსი, სიდიდე)

მაგალითი.

```
{
string[] striqoni1 = { "ანა", "საბა", "ლიკა", "რომანი", "გიორგი" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni1);
DinamiuriMasivi1.Insert(2, "ლევანი"); // DinamiuriMasivi1 = "ანა", "საბა",
// "ლევანი", "ლიკა", "რომანი", "გიორგი"
for ( int index = 0; index < DinamiuriMasivi1.Count; index++ )
    label1.Text += DinamiuriMasivi1[index] + " ";
}
```

InsertRange() მეთოდი ახდენს კოლექციის ჩამატებას მითითებული პოზიციიდან. მისი სინტაქსია:

arrayList.InsertRange(ინდექსი, კოლექცია)

მაგალითი.

```
{
string[] striqoni1 = { "ანა", "საბა", "ლიკა", "რომანი", "გიორგი" };
string[] striqoni2 = { "ნატა", "ბექა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni1);
DinamiuriMasivi1.InsertRange(2, striqoni2); // DinamiuriMasivi1 = "ანა", "საბა",
// "ნატა", "ბექა", "ლიკა", "რომანი", "გიორგი"
for ( int index = 0; index < DinamiuriMasivi1.Count; index++ )
    label2.Text += DinamiuriMasivi1[index] + " ";
}
```

SetRange() მეთოდი ახდენს დინამიური მასივის ელემენტების დიაპაზონის შეცვლას კოლექციის ელემენტებით დაწყებული მითითებული ინდექსიდან. მისი სინტაქსია:

arrayList.SetRange(ინდექსი, კოლექცია)

მაგალითი.

```
{
string[] striqoni1 = { "ანა", "საბა", "ლიკა", "რომანი", "გიორგი" };
string[] striqoni2 = { "ნატა", "ბექა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni1);
DinamiuriMasivi1.SetRange(2, striqoni2);           // DinamiuriMasivi1 = "ანა", "საბა",
                                                    // "ნატა", "ბექა", "გიორგი"
for ( int index = 0; index < DinamiuriMasivi1.Count; index++ )
    label1.Text += DinamiuriMasivi1[index] + " ";
}
```

ელემენტების ძებნა. **Contains()** მეთოდი ამოწმებს შეიცავს თუ არა დინამიური მასივი მითითებულ ელემენტს. თუ ელემენტი სიაში არსებობს, მაშინ მეთოდი გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას. მისი სინტაქსია:

arrayList.Contains(სიდიდე)

მაგალითი.

```
{
bool elementi_aris;
string[] striqoni = { "რომანი", "ანა", "საბა", "ლიკა", " რომანი ", "საბა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
elementi_aris = DinamiuriMasivi1.Contains("ანა");
if ( elementi_aris == true ) label1.Text = "ელემენტი არის მასივში";
    else label1.Text = "ელემენტი არ არის მასივში";
}
```

IndexOf() მეთოდი დინამიურ მასივში ეძებს მითითებულ ელემენტს. თუ ეს ელემენტი რამდენიმეა მასივში, მაშინ გაიცემა პირველი მათგანის ინდექსი, წინააღმდეგ შემთხვევაში კი - - 1. მისი სინტაქსია:

arrayList.IndexOf(სიდიდე)

მაგალითი.

```
{
int index1;
string[] striqoni = { "რომანი", "ანა", "საბა", "ლიკა", " რომანი ", "საბა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
index1 = DinamiuriMasivi1.IndexOf("საბა");           // index1 = 2
label1.Text = " პირველი " + DinamiuriMasivi1[index1] +
    " ელემენტის ინდექსია " + index1.ToString();
}
```

LastIndexOf() მეთოდი დინამიურ მასივში ეძებს მითითებულ ელემენტს. თუ ეს ელემენტი რამდენიმეა მასივში, მაშინ გაიცემა უკანასკნელი მათგანის ინდექსი, წინააღმდეგ შემთხვევაში კი - -1. მისი სინტაქსია:

arrayList.LastIndexOf(სიდიდე)

მაგალითი.

```
{
int index1;
string[] striqoni = { "რომანი", "ანა", "საბა", "ლიკა", " რომანი ", "საბა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
index1 = DinamiuriMasivi1.LastIndexOf("რომანი");     // index1 = 4
label1.Text = " უკანასკნელი " + DinamiuriMasivi1[index1] +
```

```
" ელემენტის ინდექსია " + index1.ToString());
```

```
}
```

ელემენტების წაშლა. **RemoveAt()** მეთოდი. ის დინამიური მასივიდან შლის მითითებული ინდექსის მქონე ელემენტს. მისი სინტაქსია:

arrayList.RemoveAt(ინდექსი)

მაგალითი.

```
{
```

```
string[] striqoni = { "ანა", "საბა", "ლიკა", "რომანი" };
```

```
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
```

```
DinamiuriMasivi1.RemoveAt(1); // DinamiuriMasivi1 = "ანა", "ლიკა", "რომანი"
```

```
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + " " ;
```

```
}
```

Remove() მეთოდი. ის დინამიური მასივიდან შლის მითითებულ ელემენტს. მისი სინტაქსია:

arrayList.Remove(სიდიდე)

მაგალითი.

```
{
```

```
string[] striqoni = { "ანა", "საბა", "ლიკა", "რომანი" };
```

```
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
```

```
DinamiuriMasivi1.Remove("რომანი"); // DinamiuriMasivi1 = "ანა", "საბა", "ლიკა"
```

```
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + " " ;
```

```
}
```

RemoveRange() მეთოდი. ის დინამიური მასივიდან შლის ელემენტების დიაპაზონს. მისი სინტაქსია:

arrayList.RemoveRange(საწყისი_ინდექსი, ელემენტების_რაოდენობა)

მაგალითი.

```
{
```

```
string[] striqoni = { "ანა", "საბა", "ლიკა", "რომანი" };
```

```
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
```

```
DinamiuriMasivi1.RemoveRange(1, 2); // DinamiuriMasivi1 = "ანა", "რომანი"
```

```
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + " " ;
```

```
}
```

ელემენტების დახარისხება, ძებნა და მიმდევრობის შებრუნება. **Sort()** მეთოდი დინამიური მასივის ელემენტებს ზრდადობის მიხედვით ალაგებს. მისი სინტაქსია:

arrayList.Sort()

მაგალითი.

```
{
```

```
string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა" };
```

```
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
```

```
DinamiuriMasivi1.Sort(); // DinamiuriMasivi1 = "ანა", "ლიკა", "რომანი", "საბა"
```

```
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + " " ;
```

```
}
```

BinarySearch() მეთოდი დახარისხებულ დინამიურ მასივში ეძებს მითითებულ ელემენტს. პოვნის შემთხვევაში გაიცემა მისი ინდექსი. მისი სინტაქსია:

arrayList.BinarySearch(ინდექსი)

მაგალითი.

```
{
```



```

int indexi;
string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
DinamiuriMasivi1.Sort();
indexi = DinamiuriMasivi1.BinarySearch("რომანი");           //      indexi = 2
label1.Text = indexi.ToString();
}

```

Reverse() მეთოდი ახდენს დინამიური მასივის ელემენტების მიმდევრობის შებრუნებას. მისი სინტაქსია:

arrayList.Reverce()

მაგალითი.

```

{
string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
foreach ( string str1 in DinamiuriMasivi1 ) label1.Text += str1 + " ";
DinamiuriMasivi1.Reverse();           //      DinamiuriMasivi1 = "ანა", "რომანი", "ლიკა", "საბა"
foreach ( string str1 in DinamiuriMasivi1 ) label2.Text += str1 + " ";
}

```

დინამიური მასივის ტევადობის შემცირება და ელემენტების დიაპაზონის მიღება. **TrimToSize()** მეთოდი დინამიური მასივის ტევადობას ამცირებს მასივში რეალურად არსებული ელემენტების რაოდენობამდე. მისი სინტაქსია:

arrayList.TrimToSize()

GetRange() მეთოდი გასცემს დინამიური მასივის მითითებული რაოდენობის ელემენტს დაწყებული მითითებული პოზიციიდან. მისი სინტაქსია:

arrayList.GetRange(საწყისი_ინდექსი, ელემენტების_რაოდენობა)

მაგალითი.

```

{
string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა", "ნატა" };
ArrayList DinamiuriMasivi1 = new ArrayList(striqoni);
ArrayList DinamiuriMasivi2 = DinamiuriMasivi1.GetRange(1, 3);
//      DinamiuriMasivi2 = "ლიკა", "რომანი", "ანა"
foreach ( string str1 in DinamiuriMasivi2 ) label1.Text += str1 + " ";
}

```

ჩამომთვლელების გამოყენება. ვნახოთ თუ როგორ შეიძლება ჩამომთვლელების გამოყენება დინამიური მასივიდან ელემენტების წაკითხვისათვის. ზემოთ აღნიშნული ყველა სახის კოლექცია ახდენს IEnumerable ინტერფეისის რეალიზებას. ამ ინტერფეისში განსაზღვრულია Current თვისება და GetEnumerator(), MoveNext() და Reset() მეთოდები.

GetEnumerator() მეთოდი გასცემს ჩამომთვლელს, რომელიც IEnumerator ობიექტს წარმოადგენს და გამოიყენება კოლექციის ელემენტებთან მიმართვისათვის. ჩვენ განვიხილავთ მისი ორი ვერსიის სინტაქსს:

arrayList.GetEnumerator()

arrayList.GetEnumerator(საწყისი_ინდექსი, ელემენტების_რაოდენობა)

MoveNext() მეთოდი ჩამომთვლელს ათავსებს კოლექციის მომდევნო ელემენტზე და გასცემს true მნიშვნელობას თუ მომდევნო ელემენტი არსებობს, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას.

Reset() მეთოდი ჩამომთვლელს ათავსებს საწყის მდგომარეობაში, ანუ კოლექციის პირველი ელემენტის წინ.

მოყვანილ პროგრამაში ხდება დინამიურ მასივთან ჩამომთვლელის საშუალებით მუშაობის დემონსტრირება.

```
{
//   პროგრამა 14.2
//   პროგრამაში ხდება დინამიურ მასივთან ჩამომთვლელის საშუალებით
//   მუშაობის დემონსტრირება
    label1.Text = "";
    string[] striqoni = { "საბა", "ლიკა", "რომანი", "ანა", "ნატა" };
    ArrayList DinamiuriMasivi = new ArrayList(striqoni);
    IEnumerator Chamomtvleli = DinamiuriMasivi.GetEnumerator();
    for ( ; Chamomtvleli.MoveNext(); )
        label1.Text += "myEnumerator.Current = " + Chamomtvleli.Current.ToString() + '\n';
    //
    Chamomtvleli.Reset();
    Chamomtvleli.MoveNext();
    label1.Text += "myEnumerator.Current = " +
        Chamomtvleli.Current.ToString() + '\n';           //   გამოჩნდება - "საბა"
}
```

საკუთარი კლასების ობიექტების დინამიური მასივები. დინამიური მასივები ჩვეულებრივი ტიპის მქონე ცვლადების გარდა შეიძლება შეიცავდეს ჩვენი საკუთარი კლასების ობიექტებს. თუ ვქმნით დინამიურ მასივს, რომელიც ჩვენს მიერ შექმნილი კლასის ობიექტებს შეიცავს, და ვაპირებთ მათ დახარისხებას, მაშინ ჩვენს მიერ შექმნილი კლასი უნდა უზრუნველყოფდეს IComparable ინტერფეისს. ეს ინტერფეისი ჩვენს მიერ შექმნილი კლასისაგან ითხოვს CompareTo() მეთოდის რეალიზებას. გარდა ამისა, ჩვენი კლასი უნდა ახდენდეს IComparer ინტერფეისის Compare() მეთოდის რეალიზებას. მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

```
//   პროგრამა 14.3
//   პროგრამაში ხდება საკუთარი კლასის ობიექტების დინამიურ მასივთან მუშაობის
//   დემონსტრირება
public class Manqana : IComparable
{
    public string Modeli;
    public int GamoshvebisWeli;
    public Manqana(string par_model, int par_gamoshvebisweli)
    {
        this.Modeli = par_model;
        this.GamoshvebisWeli = par_gamoshvebisweli;
    }
    //   გადატვირთული ToString() მეთოდი
    public override string ToString()
    {
        return "მოდელი არის " + Modeli + ", გამოშვების წელი არის " + GamoshvebisWeli;
    }
    //   IComparer ინტერფეისის Compare()მეთოდის რეალიზება
    public int Compare(object obj1, object obj2)
    {
        Manqana obj1Manqana = (Manqana)obj1;
```

```

Manqana obj2Manqana = ( Manqana ) obj2;
if (obj1Manqana.GamoshvebisWeli < obj2Manqana.GamoshvebisWeli)
{
return -1;
}
else if (obj1Manqana.GamoshvebisWeli > obj2Manqana.GamoshvebisWeli)
{
return 1;
}
else
{
return 0;
}
}
//      IComparer ინტერფეისის CompareTo() მეთოდის რეალიზება
public int CompareTo(object obj1)
{
return Compare(this, obj1);
}
}
public void MasivisGamotana(string DinamiuriMasivisSaxeli, ArrayList DinamiuriMasivi)
{
for (int indexi = 0; indexi < DinamiuriMasivi.Count; indexi++)
label1.Text += DinamiuriMasivisSaxeli + "[" + indexi.ToString() + "] = " +
DinamiuriMasivi[indexi].ToString() + '\n';
}
private void button1_Click(object sender, EventArgs e)
{
ArrayList DinamiuriMasivi = new ArrayList();
//      სისთვის ოთხი ელემენტის დამატება
Manqana myOpel = new Manqana("Opel", 2006);
Manqana myFord = new Manqana("Ford", 2007);
Manqana myBMW = new Manqana("BMW", 2005);
Manqana myToiota = new Manqana("Toiota", 2000);
DinamiuriMasivi.Add(myOpel);
DinamiuriMasivi.Add(myFord);
DinamiuriMasivi.Add(myBMW);
DinamiuriMasivi.Add(myToiota);
MasivisGamotana("DinamiuriMasivi", DinamiuriMasivi);
//
if ( DinamiuriMasivi.Contains(myFord) )
{
int index = DinamiuriMasivi.IndexOf(myFord) + '\n'; //      index = 11
label1.Text += "myFord ობიექტის ინდექსი = " + index.ToString() + '\n';
}
//
DinamiuriMasivi.Remove(myFord);

```

```

MasivisGamotana("DinamiuriMasivi", DinamiuriMasivi);
//
DinamiuriMasivi.Sort();
MasivisGamotana("DinamiuriMasivi", DinamiuriMasivi);
//
int index2 = DinamiuriMasivi.BinarySearch(myBMW);           //      index2 = 1
label1.Text += "myBMW ობიექტის ინდექსია = " + index2.ToString() + "\n";
//
ArrayList MeoreDinamiuriMasivi = DinamiuriMasivi.GetRange(1, 2);
MasivisGamotana("MeoreDinamiuriMasivi", MeoreDinamiuriMasivi);
//
IEnumerator Chamomtvleli = DinamiuriMasivi.GetEnumerator();
for ( ; Chamomtvleli.MoveNext(); )
    label1.Text += " Chamomtvleli.Current = " + Chamomtvleli.Current.ToString() + "\n";
//
Chamomtvleli.Reset();
Chamomtvleli.MoveNext();
label1.Text += "Chamomtvleli.Current = " + Chamomtvleli.Current.ToString() + "\n";
}

```

ჰემ-ცხრილები

ჰემ-ცხრილები ინახავენ წყვილებს - გასაღები-მნიშვნელობა, ანუ ჰემ-ცხრილის თითოეული ელემენტი შედგება გასაღებისა და მნიშვნელობისაგან. გასაღები გამოიყენება შესაბამისი მნიშვნელობის მისაღებად. ჰემ-ცხრილები განსაზღვრულია Hashtable კლასში. ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.3 და 14.4. ჰემ-ცხრილში გასაღები და მნიშვნელობა შეიძლება იყოს ნებისმიერი ტიპის ობიექტი. თუმცა, ჩვეულებრივ, გასაღების როლში ხშირად სტრიქონებს იყენებენ.

ცხრილი 14.3. Hashtable კლასის თვისებები

თვისებები	ტიპი	აღწერა
Count	int	შეიცავს ჰემ-ცხრილში შენახული ელემენტების რაოდენობას
IsFixedSize	bool	ამოწმებს აქვს თუ არა ჰემ-ცხრილს ფიქსირებული სიგრძე
IsReadOnly	bool	ამოწმებს არის თუ არა ჰემ-ცხრილი მხოლოდ წაკითხვადი
Item	object	გასცემს ან აყენებს მითითებული გასაღების მქონე ელემენტის მნიშვნელობას
Keys	ICollection	გასცემს კოლექციას, რომელიც ჰემ-ცხრილის გასაღებებისაგან შედგება
Values	ICollection	გასცემს კოლექციას, რომელიც ჰემ-ცხრილის მნიშვნელობებისაგან შედგება

ელემენტების დამატება, წაშლა და მიღება. **Add()** მეთოდი გამოიყენება ჰემ-ცხრილში ელემენტების დასამატებლად. მისი სინტაქსია:

Hashtable.Add(გასაღები, მნიშვნელობა)

Remove() მეთოდი გამოიყენება ჰეშ-ცხრილში ელემენტების წასაშლელად. მისი სინტაქსია: **Hashtable.Remove(გასაღები)**

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

ცხრილი 14.4. Hashtable კლასის მეთოდები

მეთოდები	დაბრუნებული ტიპი	აღწერა
Add()	void	ჰეშ-ცხრილს უმატებს მითითებული გასაღებისა და მნიშვნელობის მქონე ელემენტს
Clear()	void	ჰეშ-ცხრილიდან შლის ყველა ელემენტს
Clone()	object	ქმნის ჰეშ-ცხრილის ასლს
Contains()	bool	განსაზღვრავს, შეიცავს თუ არა ჰეშ-ცხრილი მითითებულ გასაღებს
ContainsKey()	bool	განსაზღვრავს, შეიცავს თუ არა ჰეშ-ცხრილი მითითებულ გასაღებს
ContainsValue()	bool	განსაზღვრავს, შეიცავს თუ არა ჰეშ-ცხრილი მითითებულ მნიშვნელობას
CopyTo()	void	ერთგანზომილებიან მასივში ჰეშ-ცხრილიდან გადაწერს გასაღებებს ან მნიშვნელობებს
Equals()	bool	ამოწმებს, ტოლია თუ არა ორი ჰეშ-ცხრილი
GetEnumerator()	IEnumerator	გაცემს ჩამომთვლელს, რომელიც გამოიყენება ჰეშ-ცხრილის ელემენტებთან სამუშაოდ
GetType()	Type	გაცემს მიმდინარე ობიექტის ტიპს
Remove()	void	ჰეშ-ცხრილიდან შლის მითითებული გასაღების მქონე ელემენტს
ToString()	string	გაცემს მიმდინარე ობიექტის შესაბამის სტრიქონს

```

{
//   პროგრამა 14.4
//   პროგრამაში ხდება Add() და Remove() მეთოდებთან მუშაობის დემონსტრირება
Hashtable HeshCxrili = new Hashtable();
//
HeshCxrili.Add("კა", "კახეთი");
HeshCxrili.Add("იმ", "იმერეთი");
HeshCxrili.Add("თუ", "თუშეთი");
HeshCxrili.Add("ხე", "ხევსურეთი");
foreach ( string Gasagebi in HeshCxrili.Keys )
    label1.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in HeshCxrili.Values )
    label1.Text += Mnishvneloba.ToString() + '\n';
//
HeshCxrili.Remove("თუ");
foreach ( string Gasagebi in HeshCxrili.Keys )
    label2.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in HeshCxrili.Values )
    label2.Text += Mnishvneloba.ToString() + '\n';
}

```

```
string Regioni = (string)HeshCxrili["იმ"];
label3.Text = Regioni;
}
```

როგორც პროგრამიდან ჩანს, ჰეშ-ცხრილიდან საჭირო მნიშვნელობის მისაღებად კვადრატულ ფრჩხილებში უნდა მივუთითოთ მისი შესაბამისი გასაღები, მაგალითად, HeshCxrili["იმ"]. გასაღებების მისაღებად უნდა გამოვიყენოთ Keys თვისება, ხოლო მნიშვნელობების მისაღებად კი - Values თვისება.

გასაღებისა და მნიშვნელობის ძებნა. **ContainsKey()** მეთოდი გამოიყენება ჰეშ-ცხრილში მითითებული გასაღების მოსაძებნად. პოვნის შემთხვევაში გაიცემა true, წინააღმდეგ შემთხვევაში - false. მისი სინტაქსია:

Hashtable.ContainsKey(გასაღები)

ContainsValue() მეთოდი გამოიყენება ჰეშ-ცხრილში მითითებული მნიშვნელობის მოსაძებნად. პოვნის შემთხვევაში გაიცემა true, წინააღმდეგ შემთხვევაში - false. მისი სინტაქსია:

Hashtable.ContainsValue(მნიშვნელობა)

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

```
{
//   პროგრამა 14.5
//   პროგრამაში ხდება ContainsKey() და ContainsValue() მეთოდებთან
//   მუშაობის დემონსტრირება
Hashtable HeshCxrili = new Hashtable();
//
HeshCxrili.Add("კა", "კახეთი");
HeshCxrili.Add("იმ", "იმერეთი");
HeshCxrili.Add("თუ", "თუშეთი");
HeshCxrili.Add("ხე", "ხევსურეთი");
foreach ( string Gasagebi in HeshCxrili.Keys )
    label1.Text += Gasagebi.ToString() + "\n";
foreach ( string Mnishvneloba in HeshCxrili.Values )
    label1.Text += Mnishvneloba.ToString() + "\n";
//
if ( HeshCxrili.ContainsKey("კა") )
    label2.Text = "ჰეშ-ცხრილი შეიცავს მითითებულ გასაღებს";
else label2.Text = "ჰეშ-ცხრილი არ შეიცავს მითითებულ გასაღებს";
//
if ( HeshCxrili.ContainsValue("ხევსურეთი") )
    label3.Text = "ჰეშ-ცხრილი შეიცავს მითითებულ მნიშვნელობას";
else label3.Text = "ჰეშ-ცხრილი არ შეიცავს მითითებულ მნიშვნელობას";
}
```

გასაღებებისა და მნიშვნელობების გადაწერა ერთგანზომილებიან მასივში. **CopyTo()** მეთოდი გამოიყენება ჰეშ-ცხრილიდან გასაღებებისა და მნიშვნელობების გადასაწერად ერთგანზომილებიან მასივში. მისი სინტაქსია:

Hashtable.Keys.CopyTo(მასივი, ინდექსი)

მეთოდი დაწყებული ინდექსი პოზიციიდან მითითებულ მასივში გადაწერს ჰეშ-ცხრილის გასაღებებს.

Hashtable.Values.CopyTo(მასივი, ინდექსი)

მეთოდი დაწყებული ინდექსი პოზიციიდან მითითებულ მასივში გადაწერს ჰეშ-ცხრილის მნიშვნელობებს.

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

```
{
//   პროგრამა 14.6
//   პროგრამაში ხდება CopyTo() მეთოდთან მუშაობის დემონსტრირება
Hashtable HeshCxrili = new Hashtable();
HeshCxrili.Add("კა", "კახეთი");
HeshCxrili.Add("იმ", "იმერეთი");
HeshCxrili.Add("თუ", "თუშეთი");
HeshCxrili.Add("ხე", "ხევსურეთი");
string[] Gasagebebi = new string[HeshCxrili.Count];
string[] Mnishvnelobebi = new string[HeshCxrili.Count];

foreach ( string Gasagebi in HeshCxrili.Keys )
    label1.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in HeshCxrili.Values )
    label1.Text += Mnishvneloba.ToString() + '\n';
//
HeshCxrili.Keys.CopyTo(Gasagebebi, 0);
foreach ( string Gasagebi in Gasagebebi )
    label2.Text += Gasagebi + '\n';
//
HeshCxrili.Values.CopyTo(Mnishvnelobebi, 0);
foreach ( string Mnishvneloba in Mnishvnelobebi )
    label3.Text += Mnishvneloba + '\n';
}
```

დახარისხებული სიები

დახარისხებული სია არის დინამიური მასივისა და ჰეშ-ცხრილის კომბინაცია. დახარისხებული სიის ელემენტის მიღება შეიძლება როგორც ინდექსის, ისე გასაღების მიხედვით. გასაღებები ასეთ მასივებში დახარისხებულია ზრდადობის მიხედვით. მასივში ახალი ელემენტის ჩამატებისას ის იმ პოზიციაში მოთავსდება, რომ მასივი დახარისხებული დარჩეს. დახარისხებული სია განსაზღვრულია SortedList კლასში. ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.5 და 14.6.

ელემენტების დამატება, წაშლა და მიღება. **Add()** მეთოდი გამოიყენება დახარისხებულ სიაში ელემენტების დასამატებლად. მისი სინტაქსია:

sortedList.Add(გასაღები, მნიშვნელობა)

Remove() მეთოდი გამოიყენება დახარისხებული სიიდან ელემენტების წასაშლელად. მისი სინტაქსია:

sortedList.Remove(გასაღები)

ცხრილი 14.5. SortedList კლასის თვისებები

თვისება	ტიპი	აღწერა
Capacity	int	განსაზღვრავს დახარისხებული სიის ტევადობას
Count	int	შეიცავს დახარისხებულ სიაში რეალურად არსებული ელემენტების რაოდენობას
IsFixedSize	bool	ამოწმებს, აქვს თუ არა დახარისხებულ მასივს ფიქსირებული სიგრძე
IsReadOnly	bool	ამოწმებს, არის თუ არა დახარისხებული სია მხოლოდ წაკითხვადი
Item	object	გასცემს ან აყენებს მითითებული გასაღების მქონე ელემენტის მნიშვნელობას
Keys	ICollection	გასცემს დახარისხებული სიის გასაღებებისაგან შემდგარ კოლექციას
Values	ICollection	გასცემს დახარისხებული სიის მნიშვნელობებისაგან შემდგარ კოლექციას

ცხრილი 14.6. SortedList კლასის მეთოდები

მეთოდი	დაბრუნებული ტიპი	აღწერა
Add()	void	დახარისხებულ სიას ელემენტს უმატებს
Clear()	void	დახარისხებული სიიდან შლის ყველა ელემენტს
Clone()	object	ქმნის დახარისხებული სიის ასლს
Contains()	bool	მითითებულ გასაღებს ეძებს დახარისხებულ სიაში
ContainsKey()	bool	მითითებულ გასაღებს ეძებს დახარისხებულ სიაში
ContainsValue()	bool	მითითებულ მნიშვნელობას ეძებს დახარისხებულ სიაში
CopyTo()	void	ერთგანზომილებიან მასივში გადაწერს დახარისხებული სიის მნიშვნელობებს ან გასაღებებს
Equals()	bool	ამოწმებს, ტოლია თუ არა ორი ობიექტი
GetByIndex()	object	დახარისხებული სიიდან გასცემს მითითებული ინდექსის მქონე მნიშვნელობას
GetEnumerator()	IDictionaryEnumerator	გასცემს ჩამომთვლელს დახარისხებული სიისათვის
GetKey()	object	გასცემს დახარისხებული სიის მითითებული პოზიციის გასაღებს
GetKeyList()	IList	გასცემს დახარისხებული სიის ყველა გასაღებს
GetType()	Type	გასცემს მოცემული ობიექტის ტიპს
GetValueList()	IList	გასცემს დახარისხებული სიის ყველა მნიშვნელობას
IndexOfKey()	int	გასცემს მითითებული გასაღების ინდექსს

ცხრილი 14.6. (გაგრძელება)

IndexOfValue()	int	გასცემს მითითებული მნიშვნელობის ინდექსს
Remove()	void	დახარისხებული სიიდან შლის მითითებული გასაღების შემცველ ელემენტს
RemoveAt()	void	დახარისხებული სიიდან შლის მითითებული მნიშვნელობის შემცველ ელემენტს
SetByIndex()	void	ცვლის მნიშვნელობას ინდექსით მითითებულ პოზიციაში
ToString()	string	გასცემს მიმდინარე ობიექტის შესაბამის სტრიქონს
TrimToSize()	void	დახარისხებული სიის ზომას ამცირებს მასში რეალურად არსებული ელემენტების რაოდენობამდე

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

```

{
//   პროგრამა 14.7
//   პროგრამაში ხდება Add() და Remove() მეთოდებთან მუშაობის დემონსტრირება
SortedList DaxarisxebuliSia = new SortedList ();
//
DaxarisxebuliSia.Add("კა", "კახეთი");
DaxarisxebuliSia.Add("იმ", "იმერეთი");
DaxarisxebuliSia.Add("თუ", "თუშეთი");
DaxarisxebuliSia.Add("ხე", "ხევსურეთი");
foreach ( string Gasagebi in DaxarisxebuliSia.Keys )
    label1.Text += Gasagebi.ToString() + '\n';
foreach ( string MniSvneloba in DaxarisxebuliSia.Values )
    label1.Text += MniSvneloba.ToString() + '\n';
//
DaxarisxebuliSia.Remove("თუ");
foreach ( string Gasagebi in DaxarisxebuliSia.Keys )
    label2.Text += Gasagebi.ToString() + '\n';
foreach ( string MniSvneloba in DaxarisxebuliSia.Values )
    label2.Text += MniSvneloba.ToString() + '\n';
string Regioni = (string)DaxarisxebuliSia["იმ"];
label3.Text += Regioni;
}

```

როგორც პროგრამიდან ჩანს, დახარისხებული სიიდან საჭირო მნიშვნელობის მისაღებად კვადრატულ ფრჩხილებში უნდა მივუთითოთ მისი შესაბამისი გასაღები, მაგალითად, DaxarisxebuliSia["იმ"]. გასაღებების მისაღებად უნდა გამოვიყენოთ Keys თვისება, ხოლო მნიშვნელობების მისაღებად კი - Values თვისება.

მნიშვნელობის მიღება და შეცვლა ინდექსის მიხედვით. **GetByIndex()** მეთოდი გამოიყენება დახარისხებული სიიდან საჭირო მნიშვნელობის მისაღებად ინდექსის მიხედვით. მისი სინტაქსია:

```

sortedList.GetByIndex(ინდექსი)
    მაგალითი.

```

string region1 = (string) DaxarisxebuliSia.GetByIndex(2);

SetByIndex() მეთოდი გამოიყენება დახარისხებული სიაში მნიშვნელობის შესაცვლელად ინდექსის მიხედვით. მისი სინტაქსია:

sortedList.SetByIndex(ინდექსი, მნიშვნელობა)

მაგალითი.

DaxarisxebuliSia.SetByIndex(2, "რაჭა");

გასაღების, გასაღებისა და მნიშვნელობის ინდექსის მიღება. **GetKey()** მეთოდი გამოიყენება დახარისხებული სიიდან გასაღების მისაღებად ინდექსის მიხედვით. მისი სინტაქსია:

sortedList.GetKey(ინდექსი)

მაგალითი.

string Gasagebi = (string) DaxarisxebuliSia. GetKey (2);

IndexOfKey() მეთოდი გამოიყენება დახარისხებული სიიდან მითითებული გასაღების ინდექსის მისაღებად. მისი სინტაქსია:

sortedList.IndexOfKey(გასაღები)

მაგალითი.

int GasagebisIndeqsi = DaxarisxebuliSia.IndexOfKey("იმ");

IndexOfValue() მეთოდი გამოიყენება დახარისხებული სიიდან მითითებული მნიშვნელობის ინდექსის მისაღებად. მისი სინტაქსია:

sortedList.IndexOfValue(მნიშვნელობა)

მაგალითი.

int MnishvnelobisIndeqsi = DaxarisxebuliSia.IndexOfValue("იმერეთი");

გასაღებებისა და მნიშვნელობების სიის მიღება. **GetKeyList()** მეთოდი გამოიყენება დახარისხებული სიიდან გასაღებების სიის გასაცემად. მეთოდი გასცემს IList ინტერფეისის ობიექტს. მეთოდის სინტაქსია:

sortedList.GetKeyList()

მაგალითი.

```
{  
IList GasagebebisSia = DaxarisxebuliSia.GetKeyList();  
foreach ( string Gasagebi in GasagebebisSia )  
    label1.Text += Gasagebi + " ";  
}
```

GetValueList() მეთოდი გამოიყენება დახარისხებული სიიდან მნიშვნელობების სიის გასაცემად. მეთოდი გასცემს IList ინტერფეისის ობიექტს. მეთოდის სინტაქსია:

sortedList. GetValueList()

მაგალითი.

```
{  
IList MnishvnelobebisSia = DaxarisxebuliSia.GetValueList();  
foreach ( string Gasagebi in MnishvnelobebisSia )  
    label1.Text += Gasagebi + " ";  
}
```

გასაღებისა და მნიშვნელობის ძებნა. **ContainsKey()** მეთოდი გამოიყენება დახარისხებულ სიაში მითითებული გასაღების მოსაძებნად. პოვნის შემთხვევაში გაიცემა true მნიშვნელობა, წინააღმდეგ შემთხვევაში კი - false. მისი სინტაქსია:

sortedList.ContainsKey(გასაღები)

ContainsValue() მეთოდი გამოიყენება დახარისხებულ სიაში მითითებული მნიშვნელობის მოსაძებნად. პოვნის შემთხვევაში გაიცემა true, წინააღმდეგ შემთხვევაში - false. მისი სინტაქსია:

sortedList.ContainsValue(მნიშვნელობა)

მოყვანილ პროგრამაში ხდება ამ მეთოდებთან მუშაობის დემონსტრირება.

```
{
//   პროგრამა 14.8
//   პროგრამაში ხდება ContainsKey() და ContainsValue()
//   მეთოდებთან მუშაობის დემონსტრირება
SortedList DaxarisxebuliSia = new SortedList();
//
DaxarisxebuliSia.Add("კა", "კახეთი");
DaxarisxebuliSia.Add("იმ", "იმერეთი");
DaxarisxebuliSia.Add("თუ", "თუშეთი");
DaxarisxebuliSia.Add("ხე", "ხევსურეთი");
foreach ( string Gasagebi in DaxarisxebuliSia.Keys )
    label1.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in DaxarisxebuliSia.Values )
    label1.Text += Mnishvneloba.ToString() + '\n';
//
if ( DaxarisxebuliSia.ContainsKey("კა") )
    label2.Text = "დახარისხებული სია შეიცავს მითითებულ გასაღებს";
else    label2.Text = "დახარისხებული სია არ შეიცავს მითითებულ გასაღებს";
//
if ( DaxarisxebuliSia.ContainsValue("ხევსურეთი") )
    label2.Text = "დახარისხებული სია შეიცავს მითითებულ მნიშვნელობას";
else    label2.Text = "დახარისხებული სია არ შეიცავს მითითებულ მნიშვნელობას";
}
```

გასაღებებისა და მნიშვნელობების გადაწერა ერთგანზომილებიან მასივში. **CopyTo()** მეთოდი გამოიყენება დახარისხებული სიიდან გასაღებებისა და მნიშვნელობების გადასაწერად ერთგანზომილებიან მასივში. მისი სინტაქსია:

sortedList.Keys.CopyTo(მასივი, ინდექსი)

sortedList.Values.CopyTo(მასივი, ინდექსი)

Keys თვისების CopyTo მეთოდი დაწყებული ინდექსი პოზიციიდან მითითებულ მასივში გადაწერს დახარისხებული სიის გასაღებებს. Values თვისების CopyTo მეთოდი დაწყებული ინდექსი პოზიციიდან მითითებულ მასივში გადაწერს დახარისხებული სიის მნიშვნელობებს.

მოყვანილ პროგრამაში ხდება ამის დემონსტრირება.

```
{
//   პროგრამა 14.9
//   პროგრამაში ხდება CopyTo() მეთოდთან მუშაობის დემონსტრირება
SortedList DaxarisxebuliSia = new SortedList();
//
DaxarisxebuliSia.Add("კა", "კახეთი");
DaxarisxebuliSia.Add("იმ", "იმერეთი");
DaxarisxebuliSia.Add("თუ", "თუშეთი");
DaxarisxebuliSia.Add("ხე", "ხევსურეთი");
string[] Gasagebebi = new string[DaxarisxebuliSia.Count];
string[] Mnishvnelobebebi = new string[DaxarisxebuliSia.Count];
foreach ( string Gasagebi in DaxarisxebuliSia.Keys )
    label1.Text += Gasagebi.ToString() + '\n';
foreach ( string Mnishvneloba in DaxarisxebuliSia.Values )
```

```

    label1.Text += Mnishvneloba.ToString() + '\n';
//
DaxarisxebuliSia.Keys.CopyTo(Gasagebebi, 0);
foreach ( string Gasagebi in Gasagebebi )
    label2.Text += Gasagebi + '\n';
//
DaxarisxebuliSia.Values.CopyTo(Mnishvnelobebi, 0);
foreach ( string Mnishvneloba in Mnishvnelobebi )
    label3.Text += Mnishvneloba + '\n';
}

```

რიგები

რიგი არის მეხსიერების უბანი, რომელიც მუშაობს პრინციპით "პირველი მოვიდა - პირველი წავიდა" (FIFO - First In, First Out). რიგი განსაზღვრულია Queue კლასში. ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.7 და 14.8.

მოყვანილ პროგრამაში ხდება **Enqueue()**, **Dequeue()** და **Peek()** მეთოდებთან მუშაობის დემონსტრირება.

```

{
//   პროგრამა 14.10
//   პროგრამაში ხდება Enqueue(), Dequeue() და Peek()
//   მეთოდებთან მუშაობის დემონსტრირება
//   რიგის შექმნა
Queue Rigi = new Queue();
//   რიგში ელემენტების დამატება
Rigi.Enqueue("ანა");
Rigi.Enqueue("საბა");
Rigi.Enqueue("ლიკა");
Rigi.Enqueue("რომანი");
//   რიგის ელემენტების გამოტანა
foreach ( string striqoni in Rigi )
    label1.Text += striqoni + '\n';
//   რიგში ელემენტების რაოდენობის მიღება
int ElementebisRaodenoba = Rigi.Count;
for ( int count = 0; count < ElementebisRaodenoba; count++ )
{
    label2.Text += "Peek() = " + Rigi.Peek() + '\n';
    label2.Text += "Dequeue() = " + Rigi.Dequeue() + '\n';
}
}

```

ცხრილი 14.7. Queue კლასის თვისებები

თვისება	ტიპი	აღწერა
---------	------	--------

Count	int	შეიცავს რიგში რეალურად არსებული ელემენტების რაოდენობას
IsReadOnly	bool	ამოწმებს, არის თუ არა რიგი მხოლოდ წაკითხვადი

ცხრილი 14.8. Queue კლასის მეთოდები

თვისება	დაბრუნებული ტიპი	აღწერა
Clear()	void	რიგიდან ყველა ელემენტს შლის
Clone()	object	ქმნის რიგის ასლს
Contains()	bool	ამოწმებს, არის თუ არა რიგში მითითებული ელემენტი
CopyTo	void	რიგის ელემენტებს გადაწერს ერთგანზომილებიან მასივში
Dequeue()	object	გასცემს რიგის პირველი ელემენტის მნიშვნელობას და მას რიგიდან შლის
Enqueue()	void	რიგს ბოლოში უმატებს ელემენტს
Equals()	bool	ამოწმებს, ტოლია თუ არა ორი რიგი
GetEnumerator()	IEnumerator	გასცემს ჩამომთვლელს რიგის ელემენტების გასწვრივ იტერაციისათვის
GetType()	Type	გასცემს რიგის მიმდინარე ელემენტის ტიპს
Peek()	object	გასცემს რიგის პირველი ელემენტის მნიშვნელობას, მაგრამ არ შლის მას რიგიდან
ToArray()	object[]	რიგის ელემენტებს გადაწერს მასივში
ToString()	string	გასცემს რიგის მიმდინარე ელემენტის შესაბამის სტრიქონს

სტეკები

სტეკი არის მეხსიერების უბანი, რომელიც მუშაობს პრინციპით "უკანასკნელი მოვიდა - პირველი გავიდა" (LIFO - Last In, First Out). სტეკი განსაზღვრულია Stack კლასში. ამ კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.9 და 14.10.

ცხრილი 14.9. Stack კლასის თვისებები

თვისება	ტიპი	აღწერა
Count	int	შეიცავს სტეკში რეალურად არსებული ელემენტების რაოდენობას
IsReadOnly	bool	ამოწმებს, არის თუ არა სტეკი მხოლოდ წაკითხვადი

ცხრილი 14.10. Stack კლასის მეთოდები

თვისება	დაბრუნებული	აღწერა
---------	-------------	--------

	ტიპი	
Clear()	void	სტეკიდან ყველა ელემენტს შლის
Clone()	object	ქმნის სტეკის ასლს
Contains()	bool	ამოწმებს, არის თუ არა სტეკში მითითებული ელემენტი
CopyTo	void	სტეკის ელემენტებს გადაწერს ერთგანზომილებიან მასივში
Equals()	bool	ამოწმებს, ტოლია თუ არა ორი სტეკი
GetEnumerator()	IEnumerator	გასცემს ჩამომთვლელს სტეკის ელემენტების გასწვრივ იტერაციისათვის
GetType()	Type	გასცემს სტეკის მიმდინარე ელემენტის ტიპს
Peek()	object	გასცემს სტეკის უკანასკნელ (მწვერვალზე მყოფ) ობიექტს, მაგრამ არ შლის მას სტეკიდან
Pop()	object	გასცემს სტეკის უკანასკნელ (მწვერვალზე მყოფ) ობიექტს და შლის მას სტეკიდან
Push()	void	ამატებს ელემენტს სტეკის მწვერვალზე
ToArray()	object[]	სტეკის ელემენტებს გადაწერს მასივში
ToString()	string	გასცემს სტეკის მიმდინარე ელემენტის შესაბამის სტრიქონს

მოყვანილ პროგრამაში ხდება **Push()**, **Pop()** და **Peek()** მეთოდებთან მუშაობის დემონსტრირება.

```

{
//   პროგრამა 14.11
//   პროგრამაში ხდება Push(), Pop() და Peek() მეთოდებთან მუშაობის დემონსტრირება
//   სტეკის შექმნა
Stack Steki = new Stack();
//   სტეკში ელემენტების დამატება
Steki.Push("ანა");
Steki.Push("საბა");
Steki.Push("ლიკა");
Steki.Push("რომანი");
//   სტეკის ელემენტების გამოტანა
foreach ( string striqoni in Steki )
    label1.Text += striqoni + '\n';
//   სტეკის ელემენტების რაოდენობის მიღება
int ElementebisRaodenoba = Steki.Count;
label1.Text += "ელემენტების რაოდენობა - " + ElementebisRaodenoba.ToString() + '\n';
for ( int count = 0; count < ElementebisRaodenoba; count++ )
{
    label2.Text += "Peek() = " + Steki.Peek() + '\n';
    label2.Text += "Pop() = " + Steki.Pop() + '\n';
}
}

```

ბიტების მასივები

ბიტების მასივი არის ბულის ცვლადების (true და false) მასივი. მასში true მნიშვნელობას შეესაბამება 1, false მნიშვნელობას კი - 0. ბიტების მასივი განსაზღვრულია BitArray კლასში. ამ კლასში წარმოდგენილ მეთოდებს შორის აღსანიშნავია „და“, „ან“, „არა“ და „გამომრიცხავი ან“ მეთოდები. BitArray კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 14.11 და 14.12.

განვიხილოთ ზოგიერთი მეთოდი.

Not() მეთოდი გამოიყენება ბიტების მასივის ელემენტების ინვერტირებისათვის (იხ. ბიტობრივი ოპერატორები, გვ. 62). მისი სინტაქსია:

bitArray1.Not()

bitArray1 არის BitArray ტიპის ობიექტი.

Or() მეთოდი გამოიყენება ან ოპერაციის შესასრულებლად მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის. მისი სინტაქსია:

bitArray1.Or(bitArray2)

And() მეთოდი გამოიყენება და ოპერაციის შესასრულებლად მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის. მისი სინტაქსია:

bitArray1.And(bitArray2)

Xor() მეთოდი გამოიყენება გამომრიცხავი ან ოპერაციის შესასრულებლად მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის. მისი სინტაქსია:

bitArray1.Xor(bitArray2)

ცხრილი 14.11. BitArray კლასის თვისებები

თვისება	ტიპი	აღწერა
Count	int	შეიცავს ბიტების მასივში რეალურად არსებული ელემენტების რაოდენობას
IsReadOnly	bool	ამოწმებს, არის თუ არა ბიტების მასივი მხოლოდ წაკითხვადი
Item	object	განსაზღვრავს ელემენტის მნიშვნელობას მითითებული ინდექსის მიხედვით. ეს თვისება არის BitArray კლასის ინდექსატორი
Length	object	განსაზღვრავს ელემენტების რაოდენობას, რომლებიც შეიძლება შევიწინახოთ ბიტების მასივში

ცხრილი 14.12. BitArray კლასის მეთოდები

მეთოდი	დაბრუნებული ტიპი	აღწერა
And()	BitArray	ასრულებს და ოპერაციას მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის
Clone()	object	ქმნის ბიტების მასივის ასლს
CopyTo()	void	ბიტების მასივის ელემენტებს ერთგანზომილებიან მასივში გადაწერს
Equals()	bool	ამოწმებს ტოლია თუ არა ბიტების ორი მასივი
Get()	bool	გასცემს ბიტების მასივის მითითებული პოზიციის ბიტის მნიშვნელობას
GetEnumerator()	IEnumerator	გასცემს ჩამომთვლელს ბიტების მასივისათვის
GetType()	Type	გასცემს მიმდინარე ობიექტის ტიპს
Not()	BitArray	ახდენს ბიტების მასივის ელემენტების ინვერსიას
Or()	BitArray	ასრულებს ან ოპერაციას მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის
Set()	void	ბიტების მასივის მითითებული პოზიციის ბიტს ანიჭებს მითითებულ მნიშვნელობას
SetAll()	void	მითითებულ მნიშვნელობას ანიჭებს ბიტების მასივის ყველა ელემენტს
ToString()	string	გასცემს მიმდინარე ელემენტის შესაბამის სტრიქონს
Xor()	BitArray	ასრულებს გამომრიცხავ ან ოპერაციას მიმდინარე ბიტების მასივის ელემენტებსა და პარამეტრად გადაცემული ბიტების მასივის ელემენტებს შორის

მოყვანილ პროგრამაში ხდება ამ ფუნქციების მუშაობის დემონსტრირება.

```

{
//   პროგრამა 14.12
//   პროგრამაში ხდება Not(),Or(), And() და Xor() მეთოდებთან მუშაობის დემონსტრირება
label1.Text = "BitebisMasivi";
label2.Text = "BitebisMasivi1";
label3.Text = "NOT";
label4.Text = "OR";
label5.Text = "AND";
label6.Text = "XOR";
BitArray BitebisMasivi = new BitArray(4);
BitebisMasivi[0] = true;
BitebisMasivi[1] = false;
BitebisMasivi[2] = true;
BitebisMasivi[3] = false;
//   ელემენტების გამოტანა
label1.Text += '\n';
label2.Text += '\n';
label3.Text += '\n';

```



```

label4.Text += '\n';
label5.Text += '\n';
label6.Text += '\n';
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
    label1.Text += "BitebisMasivi[" + indexi.ToString() + "] = " + BitebisMasivi[indexi].ToString() + '\n';
//
BitArray BitebisMasivi1 = new BitArray(BitebisMasivi);
BitebisMasivi1[0] = true;
BitebisMasivi1[1] = true;
BitebisMasivi1[2] = false;
BitebisMasivi1[3] = false;
// ელემენტების გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi1.Count; indexi++ )
    label2.Text += "BitebisMasivi1[" + indexi.ToString() + "] = " +
        BitebisMasivi1[indexi].ToString() + '\n';
// Not() მეთოდი ახდენს BitebisMasivi მასივის ელემენტების ინვერსიას
BitebisMasivi.Not(); // BitebisMasivi = { false, true, false, true } ელემენტების გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
    label3.Text += "BitebisMasivi[" + indexi.ToString() + "] = " + BitebisMasivi[indexi].ToString() + '\n';
// Or() მეთოდი BitebisMasivi და BitebisMasivi1 მასივის ელემენტებს შორის ასრულებს ან
// ოპერაციას
BitebisMasivi.Or(BitebisMasivi1); // BitebisMasivi = { true,true,false,true } ელემენტების გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
    label4.Text += "BitebisMasivi[" + indexi.ToString() + "] = " + BitebisMasivi[indexi].ToString() + '\n';
// And() მეთოდი BitebisMasivi და BitebisMasivi1 მასივის ელემენტებს შორის ასრულებს და
// ოპერაციას
BitebisMasivi.And(BitebisMasivi1); // BitebisMasivi = { true,true,false,false } ელემენტების
// გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
    label5.Text += "BitebisMasivi[" + indexi.ToString() + "] = " + BitebisMasivi[indexi].ToString() + '\n';
// Xor() მეთოდი BitebisMasivi და BitebisMasivi1 მასივის ელემენტებს შორის ასრულებს
// გამომრიცხავ ან ოპერაციას
BitebisMasivi.Xor(BitebisMasivi1); // BitebisMasivi = {false,false,false,false} ელემენტების
// გამოტანა
for ( int indexi = 0; indexi < BitebisMasivi.Count; indexi++ )
    label6.Text += "BitebisMasivi[" + indexi.ToString() + "] = " + BitebisMasivi[indexi].ToString() + '\n';
}

```

თავი 15. შესრულების ნაკადები

შესრულების ნაკადის განსაზღვრა

პროცესი არის შესრულების სტადიაში მყოფი პროგრამა (პროგრამა-დანართი). მაგალითად, თუ ჩვენ გავუშვებთ Word ტექსტურ რედაქტორს, მაშინ შეიქმნება მისი შესაბამისი პროცესი. ოპერაციული სისტემები პროცესებს იყენებენ სხვადასხვა პროგრამა-დანართების, როგორცაა Word, Excel და ა.შ., განცალკევებისათვის. თითოეული პროცესი ქმნის ერთ ან მეტ ნაკადს პროგრამის კოდის ნაწილების შესრულების მართვის მიზნით და ა.შ. ნაკადი არის ძირითადი ერთეული, რომლისთვისაც ოპერაციული სისტემა პროცესორის დროს გამოყოფს. პროცესები და ნაკადები დაწვრილებითაა განხილული ლიტერატურაში [8] და [9].

.NET Framework სისტემა პროცესს ყოფს ადვილად სამართავ ქვეპროცესებად, რომლებსაც პროგრამა-დანართების დომენები ეწოდებათ. ეს არის იზოლირებული გარემო, რომელშიც პროგრამა-დანართი სრულდება. პროგრამა-დანართების დომენები, რომლებიც წარმოდგენილი არიან AppDomain ობიექტებით, უზრუნველყოფენ იზოლირებისა და უსაფრთხოების საზღვრებს შესრულების პროცესში მყოფი კოდისთვის. ერთი პროცესის შიგნით ერთი ან მეტი ნაკადი შეიძლება შესრულდეს ერთ ან მეტ პროგრამა-დანართის დომენში.

ნაკადებთან სამუშაო კლასები განსაზღვრულია System.Threading სახელების სივრცეში. ნაკადები ორგვარია: შესრულების ნაკადები და მონაცემების ნაკადები. მონაცემების ნაკადები ესაა ნაკადები, რომელთა გადაადგილება ხდება ერთი მოწყობილობიდან მეორეზე. ამ თავში დაწვრილებით განვიხილავთ შესრულების ნაკადებს.

ცხრილი 15.1. Thread კლასის თვისებები

თვისება	ტიპი	აღწერა
ApartmentState	ApartmentState	განსაზღვრავს ნაკადი ერთნაკადურ თუ მრავალ-ნაკადურ გარემოში სრულდება
CurrentCulture	CultureInfo	ნაკადისათვის აყენებს ან გასცემს რეგიონალურ პარამეტრებს
CurrentThread	Thread	გასცემს ნაკადს, რომელიც მოცემულ მომენტში სრულდება
IsAlive	bool	გასცემს true მნიშვნელობას თუ ნაკადი იქნა გაშვებული და არ იქნა დამთავრებული
IsBackground	bool	ნაკადისთვის აყენებს ან გასცემს ფონური შესრულების ატრიბუტს
IsThreadPoolThread	bool	გასცემს true მნიშვნელობას თუ ნაკადი იმყოფება ნაკადების სისტემურ პულში
Name	string	აყენებს ან გასცემს ნაკადის სახელს. თუ სახელი მითითებული არ არის, მაშინ ის არის null-ის ტოლი
Priority	ThreadPriority	გასცემს ან აყენებს ნაკადის პრიორიტეტს
ThreadState	ThreadState	გასცემს ნაკადის მდგომარეობას

ხშირ შემთხვევაში, კომპიუტერში რამოდენიმე პროგრამა ერთდროულად სრულდება. მაგალითად, ჩვენ შეგვიძლია ერთმანეთის მიყოლებით შესასრულებლად გავუშვათ Excel, Paint და Internet Explorer პროგრამები. თითოეული მათგანი წარმოდგენილი იქნება შესაბამისი პროცესით. პროცესორი ამ პროცესებს შორის ისე ახდენს სწრაფად გადართვას, რომ ჩვენ გვეჩვენება თითქოს სამივე პროგრამა ერთდროულად სრულდება. სინამდვილეში, პროცესორი

ასრულებს პირველი პროცესის კოდის მცირე ნაწილს, შემდეგ მეორე პროცესის კოდის მცირე ნაწილს და ა.შ. თითოეული პროცესი წარმოქმნის ძირითად ნაკადს, რომელშიც ხდება შესაბამისი პროგრამის შესრულება.

ახლა დავუშვათ, რომ Excel პროგრამასთან მუშაობის დროს ჩვენ შევასრულეთ პრინტერზე ბეჭდვის ბრძანება. ამ შემთხვევაში პროცესი წარმოქმნის მეორე ნაკადს, რომელშიც შესრულდება პრინტერზე მონაცემების ბეჭდვა. ამავე დროს ძირითად ნაკადში ჩვენ შეგვიძლია გავაგრძელოთ ცხრილის რედაქტირება. პროცესორი მოახდენს ამ ორ ნაკადს შორის გადართვას.

Thread კლასში განსაზღვრული თვისებები და მეთოდები მოყვანილია ცხრილებში 15.1 და 15.2.

ცხრილი 15.2. Thread კლასის მეთოდები

მეთოდი	აღწერა
Abort()	იწყებს ნაკადის შესრულების შეწყვეტას
AllocateDataSlot() (სტატიკურია)	ყველა ნაკადისთვის გამოყოფს მონაცემების არასახელდებულ უბანს
AllocateNamedDataSlot() (სტატიკურია)	ყველა ნაკადისთვის გამოყოფს მონაცემების სახელდებულ უბანს
Equal()	ამოწმებს, ტოლია თუ არა ორი ობიექტი
FreeNamedDataSlot() (სტატიკურია)	ათავისუფლებს ადრე გამოყოფილ მონაცემების სახელდებულ უბანს
GetData() (სტატიკურია)	გასცემს მონაცემების უბნის მნიშვნელობას
GetDomain() (სტატიკურია)	გასცემს დომენს, რომელშიც ნაკადი სრულდება
GetDomainID() (სტატიკურია)	გასცემს იმ დომენის იდენტიფიკატორს, რომელშიც ნაკადი სრულდება
GetNamedDataSlot() (სტატიკურია)	გასცემს მონაცემების სახელდებულ უბნის მნიშვნელობას
GetType()	გასცემს მიმდინარე ობიექტის ტიპს
Interrupt()	წყვეტს ნაკადის შესრულებას
Join()	ბლოკავს ნაკადის შესრულებას სხვა ნაკადის დამთავრებამდე
ResetAbort() (სტატიკურია)	აუქმებს მიმდინარე ნაკადის შეწყვეტის მოთხოვნას
Resume()	აგრძელებს შეჩერებული ნაკადის შესრულებას
SetData() (სტატიკურია)	მონაცემების უბანს მნიშვნელობას ანიჭებს
Sleep()	აჩერებს ნაკადის შესრულებას მითითებული დროის განმავლობაში
SpinWait() (სტატიკურია)	ნაკადს აიძულებს გამოტოვოს ციკლების მითითებული რაოდენობა
Start()	იწყებს ნაკადის შესრულებას
Suspend()	დროებით აჩერებს ნაკადის შესრულებას
ToString()	გასცემს მიმდინარე ნაკადის შესაბამის სტრიქონს

შესრულების ნაკადის შექმნა

ნაკადის შესაქმნელად უნდა შევქმნათ Thread კლასის ობიექტი. ამ კლასის კონსტრუქტორს

უნდა გადავცეთ დელეგატი. ეს დელეგატი მიმართავს იმ მეთოდს, რომელიც უნდა შესრულდეს ნაკადის ქვეშ. პროგრამის საწყისი კოდი, რომელშიც სრულდება ნაკადებთან მუშაობა, უნდა იწყებოდეს დირექტივით:

using System.Threading;

ამ განყოფილებაში შევადგენთ კონსოლურ პროგრამა-დანართებს, რადგან ასეთ პროგრამებთან მუშაობის დროს ნაკადების შესრულების შედეგები უფრო მკაფიოდ ჩანს. კონსოლური პროგრამის შესაქმნელად ნახ. 2.2-ზე ნაჩვენები ფანჯრის Templates ზონაში უნდა მოვნიშნოთ Console Application ელემენტი, Name: ველში შევიტანოთ პროგრამის სახელი, კატალოგი ავირჩიოთ Browse კლავიშის საშუალებით და დავაჭიროთ OK კლავიშს. გახსნილ ფანჯარაში შეგვაქვს ჩვენი პროგრამის კოდი ისე, როგორც ეს ქვემოთაა ნაჩვენები.

// **პროგრამა 15.1**

// **პროგრამაში ხდება ორ ნაკადთან მუშაობის დემონსტრირება**

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Text;
```

```
using System.Threading;
```

```
namespace Ori_Nakadi
```

```
{
```

```
    class ChemiKlasi
```

```
    {
```

```
        // ნაკადში შესასრულებელი მეთოდის გამოძახება
```

```
        public static void Datvla_Luwi()
```

```
        {
```

```
            for ( int index2 = 2; index2 <= 50; index2 += 2 )
```

```
                Console.WriteLine(index2.ToString() + " ");
```

```
                Thread.Sleep(5000);
```

```
            }
```

```
        static void Main(string[] args)
```

```
        {
```

```
            // Nakadi1 ნაკადის შექმნა, რომელშიც შესრულდება Datvla_Luwi() მეთოდი  
            Thread Nakadi1 = new Thread(new ThreadStart(Datvla_Luwi));
```

```
            // Nakadi1 ნაკადში შესრულებას იწყებს Datvla_Luwi() მეთოდი
```

```
            Nakadi1.Start();
```

```
            // პირველ, ძირითად ნაკადში შესრულებას იწყებს Datvla_Luwi() მეთოდი
```

```
            Datvla_Luwi();
```

```
        }
```

```
    }
```

```
}
```

პროგრამაში იქმნება ობიექტი-ნაკადი Nakadi1. კონსტრუქტორს გადაეცემა Datvla_Luwi დელეგატი, რომელიც წარმოადგენს Datvla_Luwi() მეთოდზე მიმთითებელს. ამ მეთოდის შესრულების დასაწყებად გამოიძახება Nakadi1 ობიექტის Start() მეთოდი. ამის შემდეგ, მომდევნო სტრიქონში ვიძახებთ Datvla_Luwi() მეთოდს, რომელიც პირველ ძირითად ნაკადში შესრულდება. შედეგად, ორ სხვადასხვა ნაკადში ერთდროულად სრულდება Datvla_Luwi() მეთოდი. პროგრამის მუშაობის ერთ-ერთი შესაძლო შედეგი შეიძლება ასეთი იყოს:

```
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42  
44 46 48 50 36 38 40 42 44 46 48 50
```

აქ ორი ნაკადის მუშაობის შედეგები ერთმანეთშია არეული. ეს იმიტომ მოხდა, რომ ორივე ნაკადს შედეგები ერთ მოწყობილობაზე გამოაქვს. ჩვენს შემთხვევაში ეს მოწყობილობაა - მონიტორი. როგორც ვხედავთ, პირველმა ნაკადმა, რომელშიც Datvla_Luwi() მეთოდი სრულდება, მონაცემები გამოიტანა 2-დან 34-ის ჩათვლით. შემდეგ პროცესორი გადაერთო Nakadi1-ის შესრულებაზე. ამ ნაკადში მუშაობს Datvla_Luwi() მეთოდი. ამ ნაკადმა მონაცემები თავიდან ბოლომდე გამოიტანა ეკრანზე. შემდეგ პროცესორმა ისევ გააგრძელა პირველი ნაკადის შესრულება, რომელმაც დაამთავრა ეკრანზე მონაცემების გამოტანა.

აქვე შევნიშნოთ, რომ ამ პროგრამის ყოველი გაშვებისას ეკრანზე შეიძლება სხვადასხვა შედეგები მივიღოთ. საქმე ისაა, რომ წინასწარ უცნობია თუ როდის მოახდენს პროცესორი ერთი ნაკადიდან მეორეზე გადართვას.

რაც შეეხება Thread.Sleep(5000) მეთოდს, ის საშუალებას გვაძლევს შედეგები ეკრანზე დავავიწოთ 5 წამის განმავლობაში (1000 შეესაბამება 1 წამს).

იმისათვის, რომ პროგრამის შესრულების შედეგები უფრო თვალსაჩინო იყოს, ქვემოთ მოყვანილი პროგრამის ChemiKlasi კლასში გამოვაცხადოთ ორი მეთოდი: Datvla_Kenti() და Datvla_Luwi(), რომლებიც შესაბამისად, კენტ და ლუწ რიცხვებს გასცემენ.

// პროგრამა 15.2

// პროგრამაში ხდება ორი სხვადასხვა მეთოდის ორ სხვადასხვა ნაკადში

// შესრულების დემონსტრირება

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading;

namespace Ori_Sxvadasxva_Metodi

{

class ChemiKlasi

{

public static void Datvla_Kenti()

{

for (int index1 = 1; index1 <= 50; index1 += 2)

Console.Write(index1.ToString() + " ");

Thread.Sleep(5000);

}

public static void Datvla_Luwi()

{

for (int index2 = 2; index2 <= 50; index2 += 2)

Console.Write(index2.ToString() + " ");

Thread.Sleep(5000);

}

static void Main()

{

//

Thread Nakadi1 = new Thread(new ThreadStart(Datvla_Luwi));

// Datvla_Luwi() მეთოდი შესრულებას იწყებს Nakadi1 ნაკადში

Nakadi1.Start();

// Datvla_Kenti() მეთოდი შესრულებას იწყებს ძირითად ნაკადში

Datvla_Kenti();

```

    }
}
}

```

ძირითად პროგრამაში იქმნება ობიექტი-ნაკადი Nakadi1. კონსტრუქტორს გადაეცემა დელეგატი - Datvla_Luwi, რომელიც წარმოადგენს Datvla_Luwi() მეთოდზე მიმთითებელს. შემდეგ, Datvla_Luwi მეთოდის შესრულების დასაწყებად გამოიძახება Nakadi1 ობიექტის Start() მეთოდი. ამის შემდეგ, მომდევნო სტრიქონში ვიძახებთ Datvla_Kenti() მეთოდს, რომელიც პირველ, ძირითად ნაკადში შესრულდება. შედეგად, ორ სხვადასხვა ნაკადში ერთდროულად სრულდება Datvla_Kenti() და Datvla_Luwi() მეთოდები. პროგრამის მუშაობის ერთ-ერთი შესაძლო შედეგი შეიძლება ასეთი იყოს:

```

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44
46 48 50 33 35 37 39 41 43 45 47 49

```

შესრულების ნაკადის პრიორიტეტები

შექმნისას თითოეულ ნაკადს ენიჭება Normal პრიორიტეტი. სულ არსებობს პრიორიტეტების 5 დონე:

Lowest - უდაბლესი

BelowNormal - ნორმალურზე დაბალი

Normal - ნორმალური

AboveNormal - ნორმალურზე მაღალი

Highest - უმაღლესი

მოყვანილ პროგრამაში ხდება ნაკადებისათვის პრიორიტეტების მინიჭების დემონსტრირება.

// პროგრამა 15.3

// პროგრამაში ხდება ნაკადებისათვის პრიორიტეტების მინიჭების დემონსტრირება

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Text;
```

```
using System.Threading;
```

```
namespace Ori_Metodi_Prioriteti
```

```
{
```

```
    class ChemiKlasi
```

```
    {
```

```
        public static void Datvla_Kenti()
```

```
        {
```

```
            for ( int index1 = 1; index1 <= 50; index1 += 2 )
```

```
                Console.Write(index1.ToString() + " ");
```

```
                Thread.Sleep(5000);
```

```
        }
```

```
        public static void Datvla_Luwi()
```

```
        {
```

```
            for ( int index2 = 2; index2 <= 50; index2 += 2 )
```

```
                Console.Write(index2.ToString() + " ");
```

```
                Thread.Sleep(5000);
```

```

}
static void Main(string[] args)
{
    // Nakadi1 ნაკადს შექმნის დროს ენიჭება Normal პრიორიტეტი
    Thread Nakadi1 = new Thread(new ThreadStart(Datvla_Kenti));
    // მიმდინარე ნაკადს ენიჭება Lowest პრიორიტეტი
    Thread.CurrentThread.Priority = ThreadPriority.Lowest;
    // Nakadi1 ნაკადს ენიჭება Highest პრიორიტეტი
    Nakadi1.Priority = ThreadPriority.Highest;
    Nakadi1.Start();
    Datvla_Luwi();
}
}
}

```

პროგრამაში Nakadi1 ნაკადს უმაღლესი პრიორიტეტი ენიჭება, ხოლო ძირითად ნაკადს, რომელშიც Datvla_Luwi() მეთოდი სრულდება - უდაბლესი. აქ Thread კლასის CurrentThread სტატიკური თვისება გასცემს მიმდინარე ნაკადს, ანუ იმ ნაკადს, რომელიც მოცემულ მომენტში სრულდება, ხოლო Thread ობიექტის CurrentThread თვისება კი - მიმდინარე ნაკადზე მიმთითებელს.

როგორც პროგრამის შესრულების შედეგებიდან ჩანს:

```

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 2 4 6 8 10 12 14 16 18 20 22 24 26 28
30 32 34 36 38 40 42 44 46 48 50

```

ჯერ მუშაობას ამთავრებს პრიორიტეტული ნაკადი, შემდეგ კი - დაბალი პრიორიტეტის მქონე ნაკადი. თუმცა არ არსებობს იმის გარანტია, რომ მაღალპრიორიტეტული ნაკადი უფრო ადრე დაამთავრებს შესრულებას, ვიდრე დაბალპრიორიტეტული ნაკადი. საქმე ის არის, რომ, ჯერ ერთი, ყველა ოპერაციული სისტემა რესურსებს ერთნაირად არ ანაწილებს. მეორეც, პრიორიტეტები ხშირად განსაზღვრავენ პროცესორული დროის ხანგრძლივობას და არა პროცესების შესრულების რიგითობას. შედეგად, მაღალპრიორიტეტულმა პროცესმა, რომელიც დიდი მოცულობის გამოთვლებს ასრულებს, მუშაობა შეიძლება უფრო გვიან დაამთავროს, ვიდრე დაბალპრიორიტეტულმა პროცესმა, რომელიც მცირე მოცულობის გამოთვლებს ასრულებს.

შესრულების ნაკადის მდგომარეობები

ნაკადის მიმდინარე მდგომარეობის მისაღებად გამოიყენება ThreadState თვისება. ნაკადი არსებობის მანძილზე შეიძლება რამდენიმე მდგომარეობაში იმყოფებოდეს. ეს მდგომარეობები მოყვანილია ცხრილში 15.3.

შექმნის შემდეგ ნაკადს აქვს Unstarted მდგომარეობა. Start() მეთოდის გამოძახების შემდეგ ნაკადი Running მდგომარეობაში გადადის. თუ ნაკადის IsBackground თვისებას true მნიშვნელობას მივანიჭებთ, მაშინ ის ფონურ რეჟიმში იმუშავებს. ნაკადი დროის ერთსა და იმავე მომენტში შეიძლება რამდენიმე მდგომარეობაში იმყოფებოდეს, მაგალითად, WaitSleepJoin და AbortRequested მდგომარეობებში.

ცხრილი 15.3. ნაკადის მდგომარეობები.

მდგომარეობა	მდგომარეობის აღწერა
-------------	---------------------

Unstarted	ნაკადს შესრულება არ დაუწყია
Running	ნაკადი სრულდება
Background	ნაკადი ფონურ რეჟიმში სრულდება
WaitSleepJoin	ნაკადი ბლოკირებულია Wait, Sleep ან Join მეთოდის მიერ
SuspendRequested	მოთხოვნილია ნაკადის დროებით შეჩერება
Suspended	ნაკადი დროებით შეჩერებულია
StopRequested	მოთხოვნილია ნაკადის გაჩერება
Stopped	ნაკადი გაჩერებულია
AbortRequested	მოთხოვნილია ნაკადის გაუქმება
Aborted	ნაკადი გაუქმებულია

მოყვანილ პროგრამაში ხდება ThreadState თვისებასთან მუშაობის დემონსტრირება.

// **პროგრამა 15.4**

// **პროგრამაში ხდება ThreadState თვისებასთან მუშაობის დემონსტრირება**

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Nakadis_Mdgomareobis_Analizi
{
class ChemiKlasi
{
public static void Datvla()
{
for ( int index = 1; index <= 50; index++ )
    Console.WriteLine(index.ToString() + " ");
Console.WriteLine();
Thread.Sleep(5000);
}
public static void MdgomareobisNaxva(Thread Nakadi)
{
if ( ( Nakadi.ThreadState & ThreadState.Aborted ) == ThreadState.Aborted )
    Console.WriteLine("ნაკადი შეწყვეტილია");
if ( ( Nakadi.ThreadState & ( ThreadState.Stopped | ThreadState.Unstarted | ThreadState.Aborted ) ) == 0)
    Console.WriteLine("ნაკადი სრულდება");
else Console.WriteLine("ნაკადი არ სრულდება");
}
static void Main(string[] args)
{
//
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
MdgomareobisNaxva(Nakadi1);
// ნაკადის გაშვება
Nakadi1.Start();
MdgomareobisNaxva(Nakadi1);
}
```



```
// ამავე დროს სრულდება მეორე ნაკადი
Datvla();
// nakadi1 ნაკადის შეწყვეტა
Nakadi1.Abort();
MdgomareobisNaxva(Nakadi1);
}
}
}
```

შედეგი შეიძლება ასეთი იყოს:

```
nakadi ar sruldeba
```

```
nakadi sruldeba
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 1 2 3 4 5 6
7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

შესრულების ნაკადის ლოკალური მონაცემები

გადასაწყვეტი ამოცანიდან გამომდინარე ნაკადებმა შეიძლება გამოიყენონ ოპერატიული მეხსიერების ერთი და იგივე ან სხვადასხვა უბნები. იმისათვის, რომ რამოდენიმე ნაკადმა ერთობლივად გამოიყენოს მონაცემები, საჭიროა ამ მონაცემების გამოცხადება ნაკადებისათვის საერთო ხილვადობის უბანში. იმისათვის, რომ პროცესებმა თავიანთ ლოკალურ მონაცემებთან იმუშაონ, ანუ გამოიყენონ ოპერატიული მეხსიერების სხვადასხვა უბნები, უნდა გამოვიყენოთ LocalDataStoreSlot ობიექტები.

მოყვანილ პროგრამაში ხდება ორი პროცესის მიერ საკუთარ ლოკალურ უბნებთან მუშაობის დემონსტრირება.

```
// პროგრამა 15.5
```

```
// პროგრამაში ხდება ორი პროცესის მიერ საკუთარ ლოკალურ უბნებთან
```

```
// მუშაობის დემონსტრირება
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Text;
```

```
using System.Threading;
```

```
namespace Lokaluri_Monacemebi
```

```
{
```

```
class ChemiKlasi
```

```
{
```

```
public static void MonacemebisGamotana()
```

```
{
```

```
Console.WriteLine("შემთხვევითი რიცხვი = " +
```

```
Thread.GetData(Thread.GetNamedDataSlot("შემთხვევითი რიცხვი - "));
```

```
Console.WriteLine("ნაკადის სახელი = " +
```

```
Thread.GetData(Thread.GetNamedDataSlot("ნაკადის სახელი - "));
```

```
Thread.Sleep(5000);
```

```
}
```

```
public static void MonacemebisChatsera1()
```

```

{
Random Shemtxveviti_Ricxvi = new Random();
Thread.SetData(Thread.GetNamedDataSlot("შემთხვევითი რიცხვი - "),
                Shemtxveviti_Ricxvi.Next(100));
Thread.SetData(Thread.GetNamedDataSlot("ნაკადის სახელი - "),
                Thread.CurrentThread.Name);
MonacemebisGamotana();
}
public static void MonacemebisChatsera2()
{
Random Shemtxveviti_Ricxvi = new Random();
Thread.SetData(Thread.GetNamedDataSlot("შემთხვევითი რიცხვი - "),
                Shemtxveviti_Ricxvi.NextDouble());
Thread.SetData(Thread.GetNamedDataSlot("ნაკადის სახელი - "),
                Thread.CurrentThread.Name);
MonacemebisGamotana();
}
static void Main(string[] args)
{
// სახელდებული უბნების გამოყოფა
Thread.AllocateNamedDataSlot("შემთხვევითი რიცხვი - ");
Thread.AllocateNamedDataSlot("ნაკადის სახელი - ");
// მეორე ნაკადის შექმნა და გაშვება
Thread nakadi1 = new Thread(new ThreadStart(MonacemebisChatsera1));
nakadi1.Name = "nakadi1";
nakadi1.Start();
// მესამე ნაკადის შექმნა და გაშვება
Thread nakadi2 = new Thread(new ThreadStart(MonacemebisChatsera2));
nakadi2.Name = "nakadi2";
nakadi2.Start();
// მონაცემების უბნის გასუფთავება
Thread.FreeNamedDataSlot("შემთხვევითი რიცხვი- ");
Thread.FreeNamedDataSlot("ნაკადის სახელი - ");
}
}
}

```

ძირითად პროგრამაში ხდება მეხსიერების ორი უბნის გამოყოფა AllocateNamedDataSlot() მეთოდის საშუალებით. ამ უბნების სახელებია: "შემთხვევითი რიცხვი - " და "ნაკადის სახელი - ". შემდეგ იქმნება ორი ნაკადი: nakadi1 და nakadi2. მათ ენიჭება შესაბამისი სახელები. nakadi1 ნაკადში ხდება MonacemebisChatsera1() მეთოდის შესრულება, nakadi2 ნაკადში კი - MonacemebisChatsera2() მეთოდის შესრულება. თითოეული მეთოდი "შემთხვევითი რიცხვი - " უბანში წერს შემთხვევით რიცხვს, "ნაკადის სახელი - " უბანში კი - შესაბამისი ნაკადის სახელს. შემდეგ ხდება MonacemebisGamotana() მეთოდის გამოძახება, რომელსაც ეკრანზე გამოაქვს "შემთხვევითი რიცხვი - " და "ნაკადის სახელი - " უბნებში მოთავსებული მონაცემები. ძირითადი პროგრამის დასასრულს ხდება მეხსიერების გამოყოფილი უბნების გათავისუფლება და ოპერაციული სისტემისათვის გადაცემა. პროგრამის შესრულების შედეგები შეიძლება ასეთი იყოს:

```

Shemtxveviti ricxvi = 4
Nakadis saxeli      = nakadi1
Shemtxveviti ricxvi = 0,400937242154468
Nakadis saxeli      = nakadi2

```

მიუხედავად იმისა, რომ ორივე ნაკადი იყენებს ერთი და იგივე სახელის მქონე მეხსიერების უბნებს, მათ მაინც გამოეყოფათ მეხსიერების სხვადასხვა ლოკალური უბნები.

შესრულების ნაკადების მართვა

ჩვეულებრივ, ჩვენ არ ვიცით თუ რომელი ნაკადი როდის შესრულდება და ვერ განვსაზღვრავთ მათი შესრულების მიმდევრობას. ამ განყოფილებაში ჩვენ განვიხილავთ ნაკადების მართვის ხერხებს.

Timer კლასი

Timer კლასი საშუალებას გვაძლევს ნაკადი დროის გარკვეული გამეორებადი ინტერვალების შემდეგ, ანუ გარკვეული პერიოდულობით შევასრულოთ. Timer ტიპის ობიექტის შექმნისას კონსტრუქტორს ოთხი პარამეტრი უნდა გადავცეთ:

- callback. ესაა TimeCallback ტიპის დელეგატი, რომელიც მიმართავს იმ მეთოდს, რომელსაც ტაიმერი გამოიძახებს.
 - state. ესაა ობიექტი, რომელიც TimeCallback მეთოდს გადაეცემა. ეს პარამეტრი იმიტომ ეთითება, რომ ტაიმერისათვის მისაწვდომი იყოს რაიმე მუდმივი მდგომარეობა, წინააღმდეგ შემთხვევაში ეთითება null.
 - dueTime. ესაა მილიწამების რაოდენობა ტაიმერის პირველ ამუშავებამდე.
 - Period. ესაა მილიწამების რაოდენობა ტაიმერის ამუშავებებს შორის.
- მოყვანილ პროგრამაში ხდება Timer კლასთან მუშაობის დემონსტრირება.

```

// პროგრამა 15.6
// პროგრამაში ხდება Timer კლასთან მუშაობის დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Timer
{
    class ChemiKlasi
    {
        public static void Drois_Gacema(Object Mdgomareoba)
        {
            Console.WriteLine(DateTime.Now);
        }
        static void Main(string[] args)
        {
            // დელეგატის შექმნა, რომელსაც Timer ობიექტი გამოიძახებს
            TimerCallback delegati = new TimerCallback(Drois_Gacema);
            // ტაიმერის შექმნა, რომელიც წამში ორჯერ იმუშავებს
            // პირველი გაშვება ერთი წამის შემდეგ შესრულდება

```

```

System.Threading.Timer Taimer = new System.Threading.Timer(delegati, null, 1000, 2000);
//   პროგრამის შესრულების დასამთავრებლად ვაჭერთ Enter კლავიშს
Console.WriteLine("დააჭირეთ ENTER კლავიშს დასამთავრებლად");
int ricxvi = Console.Read();
//   რესურსების გათავისუფლება
Taimer.Dispose();
Taimer = null;
}
}
}

```

პროგრამის შესრულების შედეგი:

```

daaWired ENTER klaviSs dasamTavreblad

```

```

23.11.2010 13:35:16

```

```

23.11.2010 13:35:17

```

```

23.11.2010 13:35:18

```

```

23.11.2010 13:35:19

```

```

23.11.2010 13:35:20

```

პროგრამაში იქმნება `delegati` დელეგატი, რომელიც გამოიყენება `Drois_Gacema()` მეთოდის გამოსაძახებლად. შემდეგ, იქმნება `Taimer` ობიექტი-ტაიმერი, რომელიც პირველად ამუშავდება 1 წამის შემდეგ, შემდეგ კი იმუშავებს წამში ორჯერ, ანუ წამში ორჯერ მოხდება `Drois_Gacema()` მეთოდის გამოძახება. პროგრამის ბოლოს `Dispose()` მეთოდი ათავისუფლებს ოპერაციული სისტემის რესურსებს, რომელიც ნაკადს გამოეყო შექმნის დროს.

Join() მეთოდი

`Join()` მეთოდი გამოიყენება ერთი ნაკადის მისაბმელად მეორე ნაკადისათვის. შედეგად, მიზნული ნაკადი დაელოდება პირველი ნაკადის შესრულების დამთავრებას და ამის შემდეგ დაიწყებს მუშაობას. ამის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```

//   პროგრამა 15.7

```

```

//   პროგრამაში ხდება ერთი ნაკადის მეორისათვის მიზნის დემონსტრირება

```

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Join
{
    class ChemiKlasi
    {
        public static void Datvla_Kenti()
        {
            for ( int indexi = 1; indexi <= 10; indexi += 2 )
                Console.Write(indexi.ToString() + " ");
            Thread.Sleep(3000);
        }
        public static void Datvla_Luwi()
        {
            for (int indexi = 2; indexi <= 10; indexi += 2)

```

```

        Console.WriteLine(indexi.ToString() + " ");
Thread.Sleep(3000);
}
static void Main(string[] args)
{
Thread nakadi2 = new Thread(new ThreadStart(Datvla_Kenti));
//
nakadi2.Start();
//    nakadi2-ის ბლოკირება პირველი ნაკადის დამთავრებამდე
nakadi2.Join();
Datvla_Luwi();
}
}
}

```

პროგრამაში ხდება nakadi2 ნაკადის ბლოკირება მანამ, სანამ არ დამთავრდება Datvla_Luwi() მეთოდის შესრულება. პროგრამის შესრულების შედეგი:
2 4 6 8 10 1 3 5 7 9

Join() მეთოდი საშუალებას გვაძლევს, აგრეთვე, მივუთითოთ ლოდინის მაქსიმალური დრო. მაგალითად, nakadi2.Join(4000) მეთოდის შესრულების შედეგად nakadi2 ნაკადი მეორე ნაკადის შესრულების დამთავრებას დაელოდება არა უმეტეს 4 წამისა.

lock საკვანძო სიტყვა

ჩვეულებრივ, მრავალნაკადიანი კოდის შესრულებისას ადგილი აქვს ბევრ პრობლემას. ერთ-ერთი მათგანი ნაჩვენებია მოყვანილ პროგრამაში, რომელშიც ორი ნაკადი ერთსა და იმავე ცვლადთან მუშაობს.

```

//    პროგრამა 15.8
//    პროგრამაში ხდება პრობლემის დემონსტრირება, როცა ორი ნაკადი
//    ერთსა და იმავე ცვლადთან მუშაობს
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Lock
{
class ChemiKlasi
{
private int Saerto_Mtvleli = 0;
public void Datvla()
{
for ( ; Saerto_Mtvleli < 10; )
{
int Temp = Saerto_Mtvleli;
Temp++;
Console.WriteLine(Thread.CurrentThread.Name + " " + Temp);
Thread.Sleep(500);
Saerto_Mtvleli = Temp;
}
}
}

```

```

}
Thread.Sleep(5000);
}
static void Main(string[] args)
{
ChemiKlasi obieqti = new ChemiKlasi();
obieqti.Nakadebis_Shesruleba();
}
public void Nakadebis_Shesruleba()
{
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
Nakadi1.Name = "Nakadi1";
Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
Nakadi2.Name = "Nakadi2";
Nakadi1.Start();
Nakadi2.Start();
}
}
}

```

პროგრამაში Nakadi1 და Nakadi2 ნაკადები მუშაობენ Saerto_Mtvleli ცვლადთან. ამ ცვლადის მნიშვნელობას ზრდის Datvla() მეთოდი. შესაბამისად, შედეგი უნდა იყოს 1, 2, 3, ..., 10. რადგან ორი ნაკადი ერთდროულად ცვლის ერთი და იგივე ცვლადის მნიშვნელობას, ამიტომ შედეგი შეიძლება იყოს ჩვენთვის არასასურველი. მაგალითად, ასეთი:

```

Nakadi1 1
Nakadi2 1
Nakadi1 2
Nakadi2 2
Nakadi1 3
Nakadi2 3
Nakadi1 4
Nakadi2 4
Nakadi1 5
Nakadi2 5
Nakadi1 6
Nakadi2 6
Nakadi1 7
Nakadi2 7
Nakadi1 8
Nakadi2 8
Nakadi1 9
Nakadi2 9
Nakadi1 10
Nakadi2 10

```

არასწორი შედეგი მიიღება იმ მოქმედებების შედეგად, რომლებსაც პროგრამა ასრულებს:

1. Nakadi1 ნაკადი Temp ცვლადს ანიჭებს Saerto_Mtvleli ცვლადის მნიშვნელობას, ანუ 0-ს. შემდეგ, Temp ცვლადის მნიშვნელობა ერთით იზრდება და ხდება 1-ის ტოლი.
2. Nakadi1 ნაკადი დროებით აჩერებს შესრულებას 0,5 წამის განმავლობაში.

3. ოპერაციული სისტემის დამგეგმავი იწყებს Nakadi2 ნაკადის შესრულებას.
4. Nakadi2 ნაკადი Temp ცვლადს ანიჭებს Saerto_Mtvleli ცვლადის მნიშვნელობას, ანუ 0-ს. როგორც ვხედავთ, Temp ცვლადის მნიშვნელობა, რომელიც 1-ის ტოლი გახდა Nakadi1 ნაკადის შესრულების დროს, ისევ 0-ის ტოლი ხდება Nakadi2 ნაკადის შესრულების შედეგად. შემდეგ, Temp ცვლადის მნიშვნელობა ერთით იზრდება და ხდება 1-ის ტოლი.
5. შემდეგ, Nakadi2 ნაკადი დროებით აჩერებს შესრულებას 0,5 წამის განმავლობაში.
6. Nakadi1 ნაკადი ისევ აგრძელებს შესრულებას და Saerto_Mtvleli ცვლადს ანიჭებს Temp ცვლადის მნიშვნელობას, კერძოდ, 1-ს.
7. Nakadi2 ნაკადი აგრძელებს შესრულებას და Saerto_Mtvleli ცვლადს ახალ მნიშვნელობას (Temp ცვლადის მნიშვნელობას) ანიჭებს, ცვლის რა Nakadi1 ნაკადის მიერ ამ ცვლადისთვის მინიჭებულ მნიშვნელობას.
8. ციკლი მეორდება.

როგორც ვხედავთ, პრობლემას ადგილი აქვს იმის გამო, რომ სხვა პროცესის შესრულებაზე გადართვა ხდება ციკლის შუა ნაწილში. ამის თავიდან ასაცილებლად შეგვიძლია გამოვიყენოთ lock საკვანძო სიტყვა. ის განსაზღვრავს კოდის იმ ფრაგმენტს, რომლის ბლოკირებაც უნდა მოხდეს. ბლოკირებული ფრაგმენტის შესრულების დროს არ მოხდება სხვა პროცესის შესრულებაზე გადართვა. lock სიტყვის გამოყენებით შეიძლება ნებისმიერი ობიექტის ბლოკირება. მასთან ერთად მოყვანილ პროგრამაში გამოყენებულია this სიტყვა. შედეგად, მოხდება მთელი კლასის კოდის ბლოკირება.

// **პროგრამა 15.9**

// **პროგრამაში ხდება lock სიტყვის გამოყენების დემონსტრირება**

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Lock
{
    class ChemiKlasi
    {
        private int Saerto_Mtvleli = 0;
        public void Datvla()
        {
            for (; Saerto_Mtvleli < 10;)
            {
                lock (this)
                {
                    int Temp = Saerto_Mtvleli;
                    Temp++;
                    Console.WriteLine(Thread.CurrentThread.Name + " " + Temp);
                    Thread.Sleep(500);
                    Saerto_Mtvleli = Temp;
                }
            }
            Thread.Sleep(5000);
        }
        static void Main(string[] args)
```

```

{
ChemiKlasi obieqti = new ChemiKlasi();
obieqti.Nakadebis_Shesruleba();
}
public void Nakadebis_Shesruleba()
{
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
Nakadi1.Name = "Nakadi1";
Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
Nakadi2.Name = "Nakadi2";
Nakadi1.Start();
Nakadi2.Start();
}
}
}

```

პროგრამის შესრულების შედეგი იქნება ისეთი, როგორც ჩვენ გვჭირდება:

```

Nakadi1 1
Nakadi2 2
Nakadi1 3
Nakadi2 4
Nakadi1 5
Nakadi2 6
Nakadi1 7
Nakadi2 8
Nakadi1 9
Nakadi2 10
Nakadi1 11

```

უკანასკნელი მნიშვნელობა 11 მივიღეთ იმიტომ, რომ როცა Nakadi1 ნაკადმა აღმოაჩინა რა Saerto_Mtvleli ცვლადის ახალი მნიშვნელობა 9, დაიწყო ახალი ციკლი და შემდეგ გადავიდა ბლოკირებულ მდგომარეობაში. ამასობაში Nakadi2 ნაკადმა ამ ცვლადის მნიშვნელობა შეცვალა და გახადა 10-ის ტოლი. როცა Nakadi1 ნაკადს მოეხსნება ბლოკირება, მაშინ ის მუშაობას აგრძელებს უკვე ამ ცვლადის შეცვლილ მნიშვნელობასთან.

Interlocked კლასი

Interlocked კლასი გამოიყენება ცვლადების უსაფრთხო ინკრემენტისა და დეკრემენტისათვის. ამ კლასის მეთოდები ნაჩვენებია ცხრილში 15.4.

მოყვანილ პროგრამაში ხდება Decrement() მეთოდის გამოყენების დემონსტრირება.

```

// პროგრამა 15.10
// პროგრამაში ხდება Decrement() მეთოდის გამოყენების დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Interlocked
{
class ChemiKlasi

```



```

{
private int Saerto_Mtvleli = 10;
public void Datvla()
{
for ( ; Saerto_Mtvleli > 0; )
{
    System.Threading.Interlocked.Decrement(ref Saerto_Mtvleli);
    Console.WriteLine(Thread.CurrentThread.Name + " " + Saerto_Mtvleli);
    Thread.Sleep(500);
}
Thread.Sleep(5000);
}
static void Main(string[] args)
{
ChemiKlasi obieqti = new ChemiKlasi();
obieqti.Nakadebis_Shesruleba();
}
public void Nakadebis_Shesruleba()
{
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
Nakadi1.Name = "Nakadi1";
Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
Nakadi2.Name = "Nakadi2";
Nakadi1.Start();
Nakadi2.Start();
}
}
}

```

ცხრილი 15.4. Interlocked კლასის მეთოდები.

მეთოდი	აღწერა
Decrement() (სტატეკური)	ცვლადის მნიშვნელობას ერთით ამცირებს
Exchange() (სტატეკური)	ცვლადს მნიშვნელობას ანიჭებს
Increment() (სტატეკური)	ცვლადის მნიშვნელობას ერთით ზრდის

პროგრამის შესრულების შედეგი:

```

Nakadi1    9
Nakadi2    8
Nakadi1    7
Nakadi2    6
Nakadi1    5
Nakadi2    4
Nakadi1    3
Nakadi2    2

```

```
Nakadi1    1
Nakadi2    0
```

თუ Datvla() მეთოდში Decrement() მეთოდს შევცვლით Increment() მეთოდით, მაშინ Saerto_Mtvleli ცვლადის მნიშვნელობა ერთით გაიზრდება ამ მეთოდის ყოველი შესრულებისას. Exchange() მეთოდის გამოყენების შემთხვევაში Datvla() მეთოდი შემდეგ სახეს მიიღებს.

```
public void Datvla()
{
    Console.WriteLine(Thread.CurrentThread.Name + " " + Saerto_Mtvleli);
    System.Threading.Interlocked.Exchange(ref Saerto_Mtvleli, 55);
    Console.WriteLine(Thread.CurrentThread.Name + " " + Saerto_Mtvleli);
    Thread.Sleep(5000);
}
```

Monitor კლასი

Monitor კლასი გამოიყენება კოდის ნებისმიერ ობიექტთან სინქრონიზებული მიმართვის განხორციელებისათვის. ამ კლასის მეთოდები მოყვანილია ცხრილში 15.5.

ცხრილი 15.5. Monitor კლასის მეთოდები.

მეთოდი	აღწერა
Enter() (სტატიკური)	ბლოკავს მონიტორისათვის გადაცემულ ობიექტს. თუ სხვა ნაკადმა უკვე დაბლოკა ეს ობიექტი, მაშინ მიმდინარე ნაკადის შესრულება დროებით შეწყდება მანამ, სანამ სხვა ნაკადი არ გაათავისუფლებს ამ ობიექტს
Exit() (სტატიკური)	ახდენს ობიექტის დებლოკირებას
Pulse() (სტატიკური)	მომდევნო მომლოდინე ნაკადს აუწყებს იმის შესახებ, რომ მონიტორმა დროებით დაამთავრა მუშაობა ობიექტთან და ნაკადს შეუძლია მუშაობის გაგრძელება
PulseAll() (სტატიკური)	ყველა ნაკადს აუწყებს იმის შესახებ, რომ ობიექტი მალე გაათავისუფლდება
TryEnter() (სტატიკური)	ცდილობს დაბლოკოს გადაცემული ობიექტი. თუ ობიექტის ბლოკირება შესაძლებელია, მაშინ გაიცემა true, წინააღმდეგ შემთხვევაში - false
Wait() (სტატიკური)	ათავისუფლებს ყველა ბლოკირებას და დროებით აჩერებს მიმდინარე ნაკადის შესრულებას მანამ, სანამ სხვა ნაკადი არ გამოიძახებს Pulse() მეთოდს

მოყვანილ პროგრამაში ხდება Monitor კლასის გამოიყენების დემონსტრირება.

```
// პროგრამა 15.11
// პროგრამაში ხდება Monitor კლასის გამოიყენების დემონსტრირება
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Monitor
{
    class ChemiKlasi
```

```

{
private int Saerto_Mtvleli = 0;
public void Datvla()
{
while ( Saerto_Mtvleli < 10 )
{
    System.Threading.Monitor.Enter(this);
    int Temp = Saerto_Mtvleli;
    Temp++;
    Console.WriteLine(Thread.CurrentThread.Name + " " + Temp);
    Thread.Sleep(500);
    Saerto_Mtvleli = Temp;
    System.Threading.Monitor.Exit(this);
}
Thread.Sleep(5000);
}
static void Main(string[] args)
{
ChemiKlasi obieqti = new ChemiKlasi();
obieqti.Nakadebi_Shesruleba();
}
public void Nakadebi_Shesruleba()
{
Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
Nakadi1.Name = "Nakadi1";
Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
Nakadi2.Name = "Nakadi2";
Nakadi1.Start();
Nakadi2.Start();
}
}
}

```

პროგრამის შესრულების შედეგები ნაჩვენებია ნახ. 15.11-ზე.

```

Nakadi1      1
Nakadi2      2
Nakadi1      3
Nakadi2      4
Nakadi1      5
Nakadi2      6
Nakadi1      7
Nakadi2      8
Nakadi1      9
Nakadi2     10
Nakadi1     11

```

Mutex კლასი

ნაკადების სინქრონიზება შესაძლებელია, აგრეთვე, Mutex კლასის, ანუ სემაფორების

გამოიყენებით. სემაფორი არის ობიექტი, რომელსაც დროის განსაზღვრულ მომენტში შეიძლება იყენებდეს მხოლოდ ერთი ნაკადი. ჩვენ გამოვიყენებთ Mutex კლასის კონსტრუქტორს, რომელსაც ორი პარამეტრი აქვს. პირველი პარამეტრი ბულის ტიპისაა და განსაზღვრავს ობიექტი-სემაფორი უნდა ეკუთვნოდეს თუ არა მის შემქმნელ ნაკადს. მეორე პარამეტრი არის სემაფორის სახელი.

ნაკადს შეუძლია WaitOne() მეთოდის გამოძახება სემაფორის მისაღებად. ეს ნაკადი რჩება ბლოკირებულ მდგომარეობაში მანამ, სანამ სემაფორი სხვა ნაკადს უკავია. როგორც კი ნაკადი სემაფორს გაათავისუფლებს, სემაფორი მის მომთხოვნ ნაკადს დაეთმობა. როცა ნაკადი სემაფორთან მუშაობას დაამთავრებს, მაშინ მას შეუძლია ReleaseMutex() მეთოდის გამოძახება სემაფორის გასათავისუფლებლად.

მოყვანილ პროგრამაში ხდება ნაკადების სინქრონიზებისათვის სემაფორის გამოყენების დემონსტრირება.

// პროგრამა 15.12

// პროგრამაში ხდება სემაფორების გამოყენების დემონსტრირება

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Mutex
{
    class ChemiKlasi
    {
        private static int Saerto_Mtvleli = 0;
        // სემაფორის შექმნა
        static System.Threading.Mutex Semapori;
        public static void Datvla()
        {
            while ( Saerto_Mtvleli < 10 )
            {
                // სემაფორის მიღება
                Semapori.WaitOne();
                int Temp = Saerto_Mtvleli;
                Temp++;
                Console.WriteLine(Thread.CurrentThread.Name + " " + Temp);
                Thread.Sleep(500);
                Saerto_Mtvleli = Temp;
                // სემაფორის გაათავისუფლება
                Semapori.ReleaseMutex();
            }
            Thread.Sleep(5000);
        }
        static void Main()
        {
            Semapori = new System.Threading.Mutex(false, "SemaporisMushaoba");
            Thread Nakadi1 = new Thread(new ThreadStart(Datvla));
            Nakadi1.Name = "Nakadi1";
            Thread Nakadi2 = new Thread(new ThreadStart(Datvla));
```

```
Nakadi2.Name = "Nakadi2";  
Nakadi1.Start();  
Nakadi2.Start();  
}  
}  
}
```

პროგრამის მუშაობის შედეგი:

Nakadi1	1
Nakadi1	2
Nakadi1	3
Nakadi1	4
Nakadi1	5
Nakadi2	6
Nakadi2	7
Nakadi2	8
Nakadi2	9
Nakadi2	10
Nakadi1	11

თავი 16. საბაზო ბიბლიოთეკის სხვა კლასები

გლობალიზაცია

.NET გარემო შეიცავს გლობალიზაციის გაფართოებულ უზრუნველყოფას. System.Globalization სახელების სივრცეში განსაზღვრულია კლასები (ცხრილი 16.1), რომლებიც საშუალებას გვაძლევს პროგრამის კოდი სწრაფად გავაწყოთ ჩვენთვის საჭირო ეროვნული კულტურის ელემენტების გამოყენებაზე. ეროვნული კულტურის ელემენტებია: ეროვნული ენა, ვალუტის სიმბოლო, დროის, თარიღისა და რიცხვის ფორმატი და ა.შ.

.NET გარემოში თითოეული ქვეყნის კულტურას შეესაბამება აღინიშნა, რომელიც შედგება კულტურის კოდისგან, რომელსაც მოსდევს ერთი ან მეტი ქვეკულტურის კოდი. შეთანხმების თანახმად კულტურის კოდები პატარა ასოებით აღინიშნება, ხოლო ქვეკულტურის კოდები - დიდი ასოებით. მაგალითად, en-CA კოდი განსაზღვრავს ინგლისურ კულტურას კანადაში. ეს კულტურა შეიცავს ინფორმაციას კანადაში მცხოვრები ინგლისელებისათვის. pt-BR კოდი განსაზღვრავს პორტუგალიურ კულტურას ბრაზილიაში, ka-GE კოდი განსაზღვრავს ქართულ კულტურას, bs-BA-Latn კოდი განსაზღვრავს ბოსნიის კულტურას იმ მომხმარებლებისათვის, რომლებიც ლათინურ ანბანს იყენებენ და ა.შ. აქვე შევნიშნოთ, რომ .NET გარემოში ამერიკული ინგლისური ენა და ბრიტანული ინგლისური ენა სხვადასხვა კულტურებს მიეკუთვნება, რადგან ამ ქვეყნების ვალუტის სიმბოლოები და თარიღის ფორმატები ერთმანეთისაგან განსხვავდება.

ცხრილი 16.1. System.Globalization სახელების სივრცის კლასები.

კლასი	აღწერა
Calendar	აბსტრაქტული კლასია, რომელიც გვაძლევს საერთო წარმოდგენას კულტურისათვის დამახასიათებელი კალენდრის შესახებ
CompareInfo	სტრიქონების შედარების მეთოდების კულტურისათვის დამახასიათებელი კოლექციაა
CultureInfo	შეიცავს ინფორმაციას კულტურის ყველა ასპექტის შესახებ
DateTimeFormatInfo	შეიცავს ინფორმაციას დროისა და თარიღის ფორმატის შესახებ
DaylightTime	შეიცავს ინფორმაციას ზაფხულისა და ზამთრის დროზე გადასვლის შესახებ
NumberFormatInfo	შეიცავს ინფორმაციას რიცხვის წარმოდგენის ფორმატის შესახებ
RegionInfo	შეიცავს ინფორმაციას განსაზღვრული რეგიონის ან ქვეყნის შესახებ
SortKey	შეიცავს ინფორმაციას, რომელიც აუცილებელია სტრიქონების დახარისხებისათვის
StringInfo	წარმოგვიდგენს მეთოდებს სტრიქონების დაყოფისათვის ელემენტებად და ამ ელემენტების გადარჩევისათვის
TextInfo	შეიცავს ინფორმაციას ტექსტის დაფორმატების შესახებ

ცხრილი 16.2. CultureInfo კლასის თვისებები.

თვისება	ტიპი	აღწერა
Calendar	Calendar	შეიცავს კალენდარს, რომელიც მოცემულ კულტურაში ავტომატურად გამოიყენება
CompareInfo	CompareInfo	შეიცავს ინფორმაციას მოცემულ კულტურაში სტრიქონების შედარების შესახებ
CurrentCulture (სტატიკურია)	CurrentCulture	შეიცავს მოცემული ნაკადის მიმდინარე კულტურას
CurrentUICulture (სტატიკურია)	CultureInfo	შეიცავს სამომხმარებლო ინტერფეისის მიმდინარე კულტურას
DateTimeFormat	DateTimeFormat-Info	შეიცავს ინფორმაციას მოცემული კულტურის თარიღისა და დროის შესახებ
DisplayName	string	შეიცავს კულტურის ასახვად სახელს
EnglishName	string	შეიცავს კულტურის ინგლისურ დასახელებას
InstalledUICulture (სტატიკურია)	CultureInfo	შეიცავს ოპერაციულ სისტემასთან ერთად დაინსტალირებულ კულტურას
InvariantCulture (სტატიკურია)	CultureInfo	შეიცავს ინვარიანტულ კულტურას
IsNeutralCulture	bool	მიუთითებს CultureInfo ობიექტი წარმოადგენს თუ არა ნეიტრალურ კულტურას
IsReadOnly	bool	მიუთითებს CultureInfo ობიექტი არის თუ არა მხოლოდ წაკითხვადი
LCID	string	შეიცავს კულტურის იდენტიფიკატორს CultureInfo ობიექტისთვის
Name	string	შეიცავს კულტურის დასახელებას
NativeName	string	შეიცავს კულტურის ეროვნულ დასახელებას
NumberFormat	NumberFormat-Info	შეიცავს ინფორმაციას რიცხვის ფორმატის შესახებ მოცემული კულტურისათვის
OptionalCalendars	Calendar[]	შეიცავს ალტერნატიული კალენდრების სიას მოცემული კულტურისათვის
Parent	CultureInfo	შეიცავს კულტურას, რომელიც არის მშობელი მოცემული კულტურისათვის
TextInfo	TextInfo	შეიცავს ინფორმაციას ტექსტის დაფორმატების შესახებ მოცემული კულტურისათვის
ThreeLetterISO-LanguageName	string	შეიცავს ISO ენის კოდს მოცემული კულტურის ენისათვის
TwoLetterISO-LanguageName	string	შეიცავს ISO ენის ორ ასოიან კოდს მოცემული კულტურის ენისათვის
UseUserOverride	bool	მიუთითებს CultureInfo ობიექტი იყენებს თუ არა მომხმარებლის მიერ არჩეულ გაწყობებს

ცხრილი 16.3. CultureInfo კლასის მეთოდები.

მეთოდი	აღწერა
ClearCachedData()	აახლებს ქეშირებულ ინფორმაციას კულტურის შესახებ
CreateSpecificCulture() (სტატიკურია)	კულტურის დასახელებიდან ქმნის CultureInfo ობიექტს
GetCultures() (სტატიკურია)	გასცემს კულტურების სიას
ReadOnly() (სტატიკურია)	გასცემს CultureInfo ობიექტის ვერსიას, რომლის მხოლოდ წაკითხვაა შესაძლებელი

ამ განყოფილებაში მოყვანილი ყველა პროგრამის დასაწყისში უნდა მოვათავსოთ დირექტივა **using System.Globalization;**

CultureInfo კლასის თვისებები და მეთოდები მოყვანილია ცხრლებში 16.2 და 16.3.

მოყვანილ პროგრამაში ხდება CultureInfo და მასთან დაკავშირებული კლასების გამოყენების დემონსტრირება.

```

{
//   პროგრამა 16.1
//   პროგრამაში სრულდება ქართულ კულტურასთან მუშაობა
//   იქმნება ქართული კულტურა
CultureInfo QartuliKultura = new CultureInfo("ka-GE");
label1.Text = QartuliKultura.NativeName;           //   გაიცემა - ქართული(საქართველო)
label2.Text = QartuliKultura.EnglishName;         //   გაიცემა - Georgian(Georgia)
//   ქართული კულტურისათვის განისაზღვრება თარიღისა და დროის ფორმატი
DateTimeFormatInfo TarigiDroisFormati = QartuliKultura.DateTimeFormat;
label3.Text = TarigiDroisFormati.LongDatePattern;
//   გაიცემა - yyyy'წლის'dd MM, dddd
//   ქართული კულტურისათვის განისაზღვრება ვალუტისა და რიცხვის ფორმატი
NumberFormatInfo RicxvisFormati = QartuliKultura.NumberFormat;
label4.Text = RicxvisFormati.CurrencySymbol;       //   გაიცემა - Lari
label5.Text = RicxvisFormati.NumberDecimalSeparator; //   გაიცემა - ,
}

```

პროგრამაში იქმნება CultureInfo კლასის QartuliKultura ობიექტი, რომელიც ქართულ კულტურას წარმოადგენს. შემდეგ, გაიცემა ამ კულტურის სახელი ქართულ და ინგლისურ ენებზე, თარიღის გრძელი ფორმატი, ვალუტის სიმბოლო და მთელისა და წილადის გამყოფი.

CurrentCulture და CurrentUICulture თვისებები. CurrentCulture თვისება გამოიყენება კულტურის ისეთი მონაცემების ფორმატირებისათვის, როგორცაა რიცხვების, თარიღისა და დროის ფორმატი, ვალუტის სიმბოლო, დახარისხების რიგითობა და სტრიქონების შედარების წესები. CurrentUICulture თვისება გამოიყენება სამომხმარებლო ინტერფეისის ენისათვის. ის ახდენს შესაბამისი ფაილიდან კულტურაზე დამოკიდებული რესურსების, პირველ რიგში კი სამომხმარებლო ინტერფეისის ტექსტის, მიღებას. მოყვანილ პროგრამაში ხდება ამ თვისებებთან მუშაობის დემონსტრირება.

```

{
//   პროგრამა 16.2
//   პროგრამაში ხდება CurrentCulture და CurrentUICulture თვისებებთან მუშაობის
//   დემონსტრირება
//   იქმნება ქართული კულტურა
Thread.CurrentThread.CurrentCulture = new CultureInfo("ka-GE");

```



```

label1.Text = Thread.CurrentThread.CurrentCulture.Name + '\n';
// იქმნება დიდი ბრიტანეთის კულტურა
CultureInfo InglisisKultura1 = new CultureInfo("en-GB");
label1.Text += CultureInfo.CurrentCulture.Name + '\n';
// სამომხმარებლო ინტერფეისისათვის მიმდინარე კულტურის დაყენება
// მიმდინარე ნაკადისათვის განისაზღვრება იტალიის კულტურა
Thread.CurrentThread.CurrentUICulture = new CultureInfo("it-IT");
label1.Text += Thread.CurrentThread.CurrentUICulture.Name + '\n';
// იქმნება დიდი ბრიტანეთის კულტურა
CultureInfo InglisisKultura2 = new CultureInfo("en-GB");
label1.Text += CultureInfo.CurrentCulture.Name;
}

```

.NET გარემო ორ სხვადასხვა ნაკადში ახდენს ორ სხვადასხვა კულტურასთან მუშაობას. როგორც პროგრამიდან ჩანს, მიმდინარე ნაკადისათვის მიმდინარე კულტურად განისაზღვრა საქართველო - "ka-GE", CultureInfo ობიექტის შექმნის გზით. ამ ობიექტზე მიმართვა ენიჭება Thread.CurrentThread.CurrentCulture თვისებას. შემდეგ, იქმნება დამოუკიდებელი InglisisKultura1 ობიექტი, რომელიც განსაზღვრავს ინგლისის კულტურას - "en-GB". მაგრამ, ეს კულტურა არ არის მიმდინარე, ამიტომ ეკრანზე ისევ გამოჩნდება "ka-GE" კულტურა.

კულტურების ჩამონათვალი. CultureInfo ობიექტის GetCultures() მეთოდი გასცემს კულტურების სიას. მეთოდს პარამეტრად გადაეცემა CultureTypes პარამეტრი, რომელიც შეიცავს კულტურების ნაკრებს. ცხრილში 16.4 მოყვანილია ამ პარამეტრის მნიშვნელობები.

ცხრილი 16.4. CultureTypes პარამეტრის მნიშვნელობები.

მნიშვნელობა	აღწერა
AllCultures	ყველა კულტურა, ამოცნობილი .NET გარემოს მიერ
InstalledWin32Cultures	ყველა კულტურა, დაყენებული ოპერაციულ სისტემაში
NeutralCultures	ყველა ნეიტრალური კულტურა
SpecificCultures	ყველა სპეციფიკური კულტურა

მოყვანილ პროგრამაში ხდება თითოეული კულტურის დასახელების ეკრანზე გამოტანა.

```

{
// პროგრამა 16.3
// პროგრამაში ხდება თითოეული კულტურის დასახელების ეკრანზე გამოტანა
foreach ( CultureInfo Kultura in CultureInfo.GetCultures(CultureTypes.AllCultures) )
    listBox1.Items.Add(Kultura.EnglishName + "-" + Kultura.Name);
}

```

პროგრამაში listBox1 კომპონენტი გამოიყენება კულტურების სახელების გამოსატანად. ის შეგიძლიათ ჩასვათ Toolbox პანელიდან.

ინვარიანტული კულტურა. ინვარიანტული კულტურა (Invariant Language, Invariant Country) გამოიყენება იმ მონაცემებთან სამუშაოდ, რომლებიც არ იქნება ასახული რომელიმე კულტურის ელემენტების მიხედვით. ის დაკავშირებულია ინგლისურ ენასთან და არა რომელიმე ქვეყანასთან ან რეგიონთან. CultureInfo ობიექტი, რომელიც წარმოადგენს ინვარიანტულ კულტურას, შეგიძლია ორი სხვადასხვა საშუალებით შევქმნათ:

```

CultureInfo InvariantuliKultura1 = new CultureInfo("");
CultureInfo InvariantuliKultura2 = CultureInfo.InvariantCulture;

```

მოყვანილ პროგრამაში ხდება თარიღისა და ვალუტის დაფორმატება საქართველოსა და დიდი ბრიტანეთის კულტურებისათვის.

```

{
//      პროგრამა 16.4
//      პროგრამაში ხდება თარიღისა და ვალუტის დაფორმატება
DateTime Tarigi = DateTime.Now;
Double Valuta = 12345.67;
//      თარიღისა და ვალუტის დაფორმატება ქართული
//      კულტურის პარამეტრების შესაბამისად
CultureInfo SaqartvelosKultura = new CultureInfo("ka-GE");
string QartuliTarigi = Tarigi.ToString("d", SaqartvelosKultura);
string QartuliValuta = Valuta.ToString("c", SaqartvelosKultura);
label1.Text = QartuliTarigi + '\n' + QartuliValuta + '\n';
//      თარიღისა და ვალუტის დაფორმატება ინგლისის
//      კულტურის პარამეტრების შესაბამისად
CultureInfo InglisisKultura = new CultureInfo("en-GB");
string InglisuriTarigi = Tarigi.ToString("d", InglisisKultura);
string InglisuriValuta = Valuta.ToString("c", InglisisKultura);
label1.Text += InglisuriTarigi + '\n' + InglisuriValuta + '\n';
}

```

პროგრამაში გამოიყენება ToString() მეთოდის გადატვირთული ვერსია, რომელიც ახდენს თარიღისა და ვალუტის დაფორმატებას.

გრაფიკა

.NET გარემოში განსაზღვრულია გრაფიკის კლასები, რომლებიც განკუთვნილია GDI+ ინტერფეისთან სამუშაოდ (Graphic Device Interface, გრაფიკული მოწყობილობების ინტერფეისი). ეს კლასები საშუალებას გვაძლევენ დავხატოთ გეომეტრიული ფიგურები, ვიმუშაოთ გამოსახულების შემცველ ფაილებთან და ტიპოგრაფიულ შრიფტებთან. GDI+ არის კლასების ბიბლიოთეკა, რომლის დანიშნულებაა გრაფიკული ობიექტების ასახვა ჩვენი კომპიუტერის აპარატურული საშუალებებით. მისი მიზანია პროგრამა-დანართების შემმუშავებლების იზოლირება სხვადასხვა გრაფიკული მოწყობილობების რეალიზების დეტალებისაგან. ჩვენს კოდს შეუძლია გამოიძახოს GDI+ ბიბლიოთეკაში შემავალი კლასები. ის ინსტალირდება Windows XP, Windows .NET Server ან .NET გარემოს ინსტალირების დროს. GDI+ კლასები სამ ფუნქციას ასრულებენ:

- ვექტორული გრაფიკის ისეთი ელემენტების ასახვა, როგორცაა წერტილები, ხაზები და გეომეტრიული ფიგურები.
- გრაფიკული ობიექტების შემცველი ფაილების ასახვა, მაგალითად, bmp, gif, tif, jpeg და png ფორმატის მქონე.
- შრიფტების ასახვა.

ცხრილი 16.5. Graphics კლასის თვისებები.

თვისება	ტიპი	აღწერა
Clip	Region	განსაზღვრავს ხატვის უბანს
ClipBounds	Rectangle	შეიცავს Rectangle ობიექტს, რომელიც განსაზღვრავს ხატვის უბანს
CompositingMode	CompositingMode	შედგენილი გამოსახულებებისათვის მიუთითებს ხატვის რეჟიმს
CompositingQuality	CompositingQuality	შეიცავს გამოსახულებების ხარისხს
DpiX	float	განსაზღვრავს ჰორიზონტალური გარჩევის უნარს Graphics ობიექტისთვის
DpiY	float	განსაზღვრავს ვერტიკალური გარჩევის უნარს Graphics ობიექტისთვის
InterpolatingMode	InterpolatingMode	განსაზღვრავს შუალედური წერტილების გამოთვლის რეჟიმს
IsClipEmpty	bool	მიუთითებს, არის თუ არა კვეთის უბანი ცარიელი
PageScale	float	შეიცავს მასშტაბირების კოეფიციენტს Graphics ობიექტისთვის
PageUnit	GraphicsUnit	განსაზღვრავს გამოყენებულ ზომის ერთეულს
PixelOffsetMode	PixelOffsetMode	განსაზღვრავს პიქსელების წანაცვლების საშუალებას ხატვის დროს
RenderingOrigin	Point	განსაზღვრავს კოორდინატების დასაწყისს ფუნჯისა და შტრიხებისათვის
SmoothingMode	SmoothingMode	მართავს ხაზის სიგლუვეს Graphics ობიექტზე მათი ხატვის დროს
TextContrast	int	განსაზღვრავს კონტრასტის მნიშვნელობას ტექსტისათვის
TextRenderingHint	TextRenderingHint	განსაზღვრავს შემხსენებელ შეტყობინებას ტექსტისათვის
Transform	Matrix	განსაზღვრავს გეომეტრიულ გარდაქმნას
VisibleClipbounds	Rectangle	შეიცავს კვეთის ხილულ უბანს Graphics ობიექტისთვის

ცხრილი 16.6. Graphics კლასის მეთოდები.

მეთოდი	აღწერა
AddMetafileComment()	Metafile ობიექტს უმატებს კომენტარს
BeginContainer()	ხსნის ახალ გრაფიკულ კონტეინერს
Clear()	ასუფთავებს სახატავ ზედაპირს
DrawArc()	ხატავს რკალს
DrawBezier()	ხატავს ბაზიეს მრუდს
DrawBeziers()	ხატავს ბაზიეს მრუდების სერიას
DrawClosedCurve()	ხატავს დახურულ ფუნდამენტურ მრუდს
DrawCurve()	ხატავს ფუნდამენტურ მრუდს
DrawEllipse()	ხატავს ელიფსს
DrawIcon()	ხატავს პიქტოგრამას
DrawIconUnstretched()	ხატავს პიქტოგრამას მასშტაბირების გარეშე
DrawImage()	ხატავს გამოსახულებას
DrawImageUnscaled()	ხატავს გამოსახულებას მასშტაბირების გარეშე
DrawLine()	ხატავს ხაზს
DrawLines()	ხატავს ხაზების სერიას
DrawPath()	ხატავს GraphicsPath ობიექტს
DrawPie()	ხატავს სექტორს
DrawPolygon()	ხატავს მრავალკუთხედს
DrawRectangle()	ხატავს სწორკუთხედს
DrawRectangles()	ხატავს სწორკუთხედების სერიას
DrawString()	ხატავს ტექსტის სტრიქონს
EndContainer()	ხურავს გრაფიკულ კონტეინერს
EnumerateMetafile()	ახდენს მეტაფაილის ვიზუალიზებას
ExcludeClip()	სწორკუთხედს უმატებს კვეთის უბანს
FillClosedCurve()	ფერით ავსებს ჩაკეტილ მრუდს
FillEllipse()	ფერით ავსებს ელიფსს
FillPath()	ფერით ავსებს GraphicsPath ობიექტს
FillPie()	ფერით ავსებს სექტორს
FillPolygon()	ფერით ავსებს მრავალკუთხედს
FillRectangle()	ფერით ავსებს სწორკუთხედს
Fill Rectangles()	ფერით ავსებს სწორკუთხედების სერიას
FillRegion()	ფერით ავსებს Region ობიექტს
Flush()	იწვევს რიგში მყოფი ოპერაციების იძულებით შესრულებას
FromImage() (სტატიკურია)	Image ობიექტიდან ქმნის Graphics ობიექტს
GetNearestColor()	გასცემს მითითებულ ფერთან ახლოს მყოფ ფერს
IntersectClip()	ცვლის კვეთის უბანს სწორკუთხედთან მისი გადაკვეთის გზით
IsVisible()	გასცემს true-ს თუ მითითებული წერტილი ჩანს, წინააღმდეგ შემთხვევაში კი - false-ს
MeasureCharacter- Ranges()	გასცემს ინფორმაციას ტექსტური სტრიქონის შესახებ

ცხრილი 16.6. (გაგრძელება)

MeasureString()	გასცემს სტრიქონის ზომას, თუ ის დახატულია მითითებული შრიფტით
ResetClip()	აუქმებს კვეთის უბანს
ResetTransform()	აუქმებს Graphics ობიექტის გარდაქმნას
Restore()	ადადგენს GraphicsState ობიექტში შენახულ მდგომარეობას
RotateTransform()	ახდენს Graphics ობიექტის მობრუნებას
Save()	მდგომარეობას ინახავს GraphicsState ობიექტში
ScaleTransform()	ცვლის Graphics ობიექტის გარდაქმნის მასშტაბს
SetClip()	აყენებს კვეთის უბანს

ცხრილი 16.7. Pen კლასის თვისებები.

თვისება	ტიპი	აღწერა
Alignment	PenAlignment	შეიცავს Pen ობიექტის საზღვარზე გასწორების სახეს
Brush	Brush	შეიცავს Brush ობიექტს, რომელსაც იყენებს მოცემული Pen ობიექტი
Color	Color	შეიცავს Pen ობიექტის ფერს
CustomEndCap	CustomLineCap	შეიცავს ხაზის დაბოლოების სტილს
CustomStartCap	CustomLineCap	შეიცავს ხაზის დასაწყისის სტილს
DashCap	DashCap	შეიცავს პუნქტირის მონაკვეთების დაბოლოებების სტილს
DashOffset	Float	შეიცავს მანძილს ხაზის დასაწყისიდან პუნქტირის შაბლონამდე
DashPattern	Float []	პუნქტირის მონაკვეთებისა და მათ შორის მანძილების მასივია
DashStyle	DashStyle	შეიცავს პუნქტირის მონაკვეთების სტილს
EndCap	LineCap	შეიცავს ხაზის დაბოლოების სტილს
LineJoin	LineJoin	შეიცავს რამდენიმე ხაზს შორის შეერთების სტილს
MiterLimit	Float	შეიცავს ჩამოჭრილი კუთხეების სისქეს რამდენიმე ხაზის შეერთებისას
PenType	PenType	შეიცავს სტილს, რომელიც გამოიყენება Pen ობიექტის მიერ ხაზების გავლებისას
StartCap	LineCap	შეიცავს ხაზის დასაწყისის სტილს
Width	Float	ხაზის სისქეა პიქსელებში

ცხრილი 16.8. Pen კლასის მეთოდები.

მეთოდი	აღწერა
Clone()	ქმნის Pen ობიექტის ასლს
ResetTransform()	ახდენს გარდაქმნის დაყვანას საწყის მდგომარეობაზე
RotateTransform()	ახდენს გარდაქმნის მობრუნებას
ScaleTransform()	ახდენს გარდაქმნის მასშტაბირებას
SetLineCap()	აყენებს StartCap და EndCap თვისებებს
TranslateTransform()	ახდენს გარდაქმნის ტრანსლაციას

Graphics კლასი განსაზღვრავს სახატავ უბანს (ფურცელს, ტილოს), Pen კლასი გამოიყენება ხაზებისა და გრაფიკული ფიგურების დასახატად, Brush კლასი გამოიყენება ფიგურების შესავსებად არჩეული ფერით. Rectangle კლასი განსაზღვრავს სახატავი უბნის საზღვრებს. Graphics კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 16.5 და 16.6. Pen კლასის თვისებები და მეთოდები მოყვანილია ცხრილებში 16.7 და 16.8.

სიგრძის ერთეული ყველა გრაფიკული ობიექტისათვის არის პიქსელი. კოორდინატების დასაწყისად ითვლება Graphics ობიექტის ზედა მარცხენა კუთხე.

Graphics ობიექტი წარმოადგენს ხატვის ზედაპირს და შეგვიძლია განვიხილოთ როგორც სახატავი ფურცელი (ტილო). Pen ობიექტი შეიცავს დასახატი ხაზის თვისებებს.

გეომეტრიული ფიგურების ხატვა

განვიხილოთ Graphics კლასის ზოგიერთი მეთოდი.

Graphics კლასის ობიექტი იქმნება CreateGraphics() მეთოდის საშუალებით. მაგალითად, Graphics SaxataviPurceli = CreateGraphics();

DrawArc() მეთოდი გამოიყენება ელიფსის ნაწილის (რკალის) დასახატად. მისი სინტაქსია:
void Graphics.DrawArc(Pen ფუნჯი, Rectangle სწორკუთხედი, float საწყისი_კუთხე, float მოღუნვის_კუთხე);

მაგალითი.

```
{
// სახატავი ფურცლის შექმნა
Graphics SaxataviPurceli = CreateGraphics();
// კალმის შექმნა
Pen Kalami = new Pen(Color.Red, 2);
// ხატვის უბნის შექმნა
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
// რკალის ხატვა
SaxataviPurceli.DrawArc(Kalami, Otxkutxedi, 0, 90);
}
```

როგორც პროგრამიდან ჩანს, ჯერ იქმნება სახატავი ფურცელი. შემდეგ იქმნება წითელი ფერის კალამი, რომლის სისქეა 2 პიქსელი. შემდეგ იქმნება ხატვის უბანი, რომელშიც უნდა მოთავსდეს რკალი. ამ უბნის საწყისი წერტილია (0,0), საბოლოო წერტილი კი - (200,300). ბოლოს სრულდება რკალის ხატვა.

DrawRectangle() მეთოდი გამოიყენება სწორკუთხედის დასახატად. მისი სინტაქსია:
void Graphics.DrawRectangle(Pen ფუნჯი, Rectangle სწორკუთხედი);

მაგალითი.

```
{
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
SaxataviPurceli.DrawRectangle(Kalami, Otxkutxedi);
}
```

DrawEllipse() მეთოდი გამოიყენება ელიფსის დასახატად. მისი სინტაქსია:
void Graphics.DrawEllipse(Pen ფუნჯი, Rectangle სწორკუთხედი);

მაგალითი.

```
{
Graphics SaxataviPurceli = CreateGraphics();
```

```

Pen Kalami = new Pen(Color.Red, 2);
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
SaxataviPurceli.DrawEllipse(Kalami, Otxkutxedi);
}

```

DrawPie() მეთოდი გამოიყენება სექტორის დასახატად. მისი სინტაქსია:

```

void Graphics.DrawPie(Pen ფუნჯი, Rectangle სწორკუთხედი,
float საწყისი_კუთხე, float მოღუნვის_კუთხე);

```

მაგალითი.

```

{
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
SaxataviPurceli.DrawPie(Kalami, Otxkutxedi, 0, 90);
}

```

DrawLine() მეთოდი გამოიყენება ხაზის გასავლებად საწყისი წერტილიდან საბოლოო წერტილამდე. მისი სინტაქსია:

```

void Graphics.DrawLine(Pen ფუნჯი, float x1, float y1, float x2, float y2);

```

აქ x1 და y1 არის საწყისი წერტილის კოორდინატები, ხოლო x2 და y2 კი - საბოლოო წერტილის. მაგალითი.

```

{
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
Rectangle Otxkutxedi = new Rectangle(0, 0, 200, 300);
SaxataviPurceli.DrawLine(Kalami, 0, 0, 100, 100);
}

```

DrawPolygon() მეთოდი გამოიყენება მრავალკუთხედის ასაგებად. მისი სინტაქსია:

```

void Graphics.DrawPolygon(Pen ფუნჯი, Point [ ] წერტილები);

```

მაგალითი.

```

{
// წერტილების შექმნა, რომლებიც მრავალკუთხედს განსაზღვრავენ
Point wertili1 = new Point(50, 50);
Point wertili2 = new Point(100, 25);
Point wertili3 = new Point(130, 5);
Point wertili4 = new Point(170, 50);
Point wertili5 = new Point(210, 100);
Point wertili6 = new Point(50, 50);
Point[] WertilebisMasivi = {
                                wertili1,
                                wertili2,
                                wertili3,
                                wertili4,
                                wertili5,
                                wertili6
};
}

```

```

Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
SaxataviPurceli.DrawPolygon(Kalami, WertilebisMasivi);

```

```

}
    DrawLines() მეთოდი გამოიყენება ხაზების გასავლებად საწყისი წერტილიდან საბოლოო
წერტილამდე. მისი სინტაქსია:
void Graphics.DrawLines(Pen ფუნჯი, Point [ ] წერტილები);
    მაგალითი.
{
//    წერტილების შექმნა ხაზებისათვის
Point wertili1 = new Point(50, 50);
Point wertili2 = new Point(100, 25);
Point wertili3 = new Point(130, 5);
Point wertili4 = new Point(170, 50);
Point wertili5 = new Point(210, 100);
Point wertili6 = new Point(50, 50);
Point[] WertilebisMasivi = { wertili1,
                             wertili2,
                             wertili3,
                             wertili4,
                             wertili5,
                             wertili6 };
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 2);
SaxataviPurceli.DrawLines(Kalami, WertilebisMasivi);
}
    Pen კლასის კონსტრუქტორს აქვს 4 გადატვირთული ვერსია. პირველი ვერსიის სინტაქსია:
Pen ობიექტის_სახელი = new Pen(Brush ფუნჯი);
    მაგალითი.
{
Brush PunjiSolid = new SolidBrush(Color.Red);
Pen Kalami = new Pen(PunjiSolid);
}
    მეორე ვერსიის სინტაქსია:
Pen ობიექტის_სახელი = new Pen(Color ფერი);
    მაგალითი.
{
Pen Kalami = new Pen(Color.Red);
}
    მესამე ვერსიის სინტაქსია:
Pen ობიექტის_სახელი = new Pen(Brush ფუნჯი, float სიგანე);
    მაგალითი.
{
Brush PunjiSolid = new SolidBrush(Color.Red);
Pen Kalami = new Pen(PunjiSolid, 3);
}
    მეოთხე ვერსიის სინტაქსია:
Pen ობიექტის_სახელი = new Pen(Color ფერი, float სიგანე);
    მაგალითი.
{

```



```
Pen Kalami = new Pen(Color.Red, 3);
}
```

მოყვანილ პროგრამაში ხდება ხაზისა და ისრის გავლების დემონსტრირება.

```
{
// პროგრამა 16.5
// პროგრამაში ხდება ხაზის გავლების დემონსტრირება
int x1 = 5, y1 = 5, x2 = 200, y2 = 200;
Graphics SaxataviPurceli = CreateGraphics();
Pen Kalami = new Pen(Color.Red, 5);
SaxataviPurceli.DrawLine(Kalami, x1, y1, x2, y2);
//
x1 = 5; y1 = 200; x2 = 200; y2 = 5;
Kalami.StartCap = LineCap.ArrowAnchor; // ხაზის დასაწყისს ექნება ისრის ფორმა
Kalami.EndCap = LineCap.Round; // ხაზის დაბოლოება იქნება მომრგვალებული
SaxataviPurceli.DrawLine(Kalami, x1, y1, x2, y2);
// რესურსების გათავისუფლება
Kalami.Dispose();
SaxataviPurceli.Dispose();
}
```

პროგრამაში ჯერ იქმნება Graphics კლასის SaxataviPurceli ობიექტი. შემდეგ იქმნება Pen კლასის Kalami ობიექტი, რომელსაც ექნება წითელი ფერი, სისქე კი - 5 პიქსელი. თუ სისქე არ არის მითითებული, მაშინ ის აიღება 1-ის ტოლი. DrawLine მეთოდი (0,0) წერტილიდან გავლებს წითელ ხაზს (200, 200) წერტილამდე. შემდეგ პირველი წერტილის კოორდინატი ხდება (5, 200), მეორე წერტილის კი - (200, 5). ხაზის დასაწყისს ექნება ისრის ფორმა, ბოლოს კი - მომრგვალებული სახე. ამის შემდეგ, ისარი გაივლება (5, 200) წერტილიდან (200, 5) წერტილამდე. პროგრამის ბოლოს ხდება Kalami და SaxataviPurceli ობიექტების მიერ დაკავებული რესურსების გათავისუფლება.

ფიგურების შევსება ფერით

Brush კლასი გამოიყენება სხვადასხვა ფერით და შაბლონით ფიგურების შესავსებად. ის აბსტრაქტულია და ამიტომ ამ კლასის ობიექტებს ვერ შევქმნით. მისგან წარმოებული კლასები მოყვანილია ცხრილში 16.9.

ცხრილი 16.9. Brush კლასიდან წარმოებული კლასები.

კლასი	გამოყენება
HatchBrush	გამოიყენება უზნის შესავსებად სტანდარტული შაბლონების ნაკრებიდან ამორჩეული შაბლონის მიხედვით
LinearGradientBrush	გამოიყენება უზნის შესავსებად ორ ფერს შორის თანაბარი გადასვლით სწორი ხაზის გასწვრივ
PathGradientBrush	გამოიყენება უზნის შესავსებად ორ ფერს შორის თანაბარი გადასვლით ნებისმიერი ხაზის გასწვრივ
SolidBrush	გამოიყენება უზნის შესავსებად ერთტონიანი ფერით
TextureBrush	გამოიყენება უზნის შესავსებად ფაილიდან წაკითხული გამოსახულებით

მოყვანილ პროგრამაში ხდება სხვადასხვა გეომეტრიული ფიგურის სხვადასხვა სტილით შევსების დემონსტრირება, კერძოდ, სწორკუთხედისთვის გამოიყენება გრადიენტული შევსება,

ელიფსისთვის - შაბლონის მიხედვით შევსება, ხოლო სექტორისთვის კი - თანაბარი ფერით შევსება.

```
{
//   პროგრამა 16.6
//   პროგრამაში ხდება სხვადასხვა გეომეტრიულ ფიგურის სხვადასხვა სტილით შევსება
int x = 0;
int y = 0;
int Sigane = 200;
int Simagle = 300;
//   სახატავი ფურცლის შექმნა
Graphics SaxataviPurceli = CreateGraphics();
//   სახატავი უბნის შექმნა
Rectangle Otxkuxtedi = new Rectangle(x, y, Sigane, Simagle);
//   ფუნჯის შექმნა თანაბარი ფერით შევსებისათვის
Brush PunjiSolid = new SolidBrush(Color.Red);
//   ფუნჯის შექმნა შაბლონის მიხედვით შევსებისათვის
Brush PunjiHatch = new HatchBrush(HatchStyle.Horizontal, Color.Blue, Color.Yellow);
//   ფუნჯის შექმნა გრადიენტული შევსებისათვის
Brush PunjiGradient = new LinearGradientBrush(Otxkuxtedi, Color.Black, Color.Aqua, 45, false);
//   ოთხკუთხედის, ელიფსისა და სექტორის შევსება ფერით
SaxataviPurceli.FillRectangle(PunjiGradient, Otxkuxtedi);
SaxataviPurceli.FillEllipse(PunjiHatch, 5, 110, 100, 100);
SaxataviPurceli.FillPie(PunjiSolid, 60, 60, 120, 120, 270, 90);
//   რესურსების გათავისუფლება
PunjiSolid.Dispose();
PunjiHatch.Dispose();
PunjiGradient.Dispose();
SaxataviPurceli.Dispose();
}
```

სტრიქონების ხატვა

DrawString() მეთოდი გამოიყენება სტრიქონების დასახატად. მისი სინტაქსია:

```
void Graphics.DrawString(string სტრიქონი, Font შრიფტი, Brush ფუნჯი, float x, float y);
```

მოყვანილ პროგრამაში ხდება ჰორიზონტალური სტრიქონის ხატვის დემონსტრირება.

```
{
//   პროგრამა 16.7
//   პროგრამაში ხდება ჰორიზონტალური სტრიქონის ხატვა
float x = 10.0f;
float y = 10.0f;
string DasaxatiStriqoni = "ანა და საბა";
//   სახატავი ფურცლის შექმნა
Graphics SaxataviPurceli = CreateGraphics();
//   შრიფტის განსაზღვრა სტრიქონისთვის
Font Shrifti = new Font("AcadNusx", 14);
//   ფუნჯის შექმნა
SolidBrush Punji = new SolidBrush(Color.Black);
//   ჰორიზონტალური სტრიქონის ხატვა
```

```

SaxataviPurceli.DrawString(DasaxatiStriqoni, Shrifti, Punji, x, y);
// რესურსების გათავისუფლება
Shrifti.Dispose();
Punji.Dispose();
SaxataviPurceli.Dispose();
}

```

მოყვანილ პროგრამაში ხდება ვერტიკალური სტრიქონის ხატვის დემონსტრირება.

```

{
// პროგრამა 16.8
// პროგრამაში ხდება ვერტიკალური სტრიქონის ხატვა
float x = 10.0f;
float y = 10.0f;
string DasaxatiStriqoni = "საბა და ანა";
// სახატავი ფურცლის შექმნა
Graphics SaxataviPurceli = this.CreateGraphics();
// შრიფტის განსაზღვრა სტრიქონისთვის
Font Shrifti = new Font("AcadNusx", 14);
// ფუნჯის შექმნა
SolidBrush Punji = new SolidBrush(Color.Black);
// სტრიქონისთვის ვერტიკალური მიმართულების არჩევა
StringFormat StriqonisFormati = new StringFormat(StringFormatFlags.DirectionVertical);
// ვერტიკალური სტრიქონის ხატვა
SaxataviPurceli.DrawString(DasaxatiStriqoni, Shrifti, Punji, x, y, StriqonisFormati);
// რესურსების გათავისუფლება
Shrifti.Dispose();
Punji.Dispose();
SaxataviPurceli.Dispose();
}

```

გამოსახულებასთან მუშაობა

გამოსახულებასთან სამუშაოდ გამოიყენება Bitmap ობიექტი. ის მუშაობს bmp, gif, jpeg, png და tif ფაილებთან. GDI+ ინტერფეისი ავტომატურად ცვლის გამოსახულების ზომას ისე, რომ ის მოთავსდეს ხატვის უბანში. ცხრილებში 16.10 და 16.11 მოყვანილია Bitmap კლასის თვისებები და მეთოდები.

Bitmap კლასის კონსტრუქტორს აქვს 12 გადატვრთული ვერსია. ჩვენ მოვიყვანთ ორი მათგანის სინტაქსს:

Bitmap(string ფაილის_სახელი);

Bitmap(int სიგანე, int სიმაღლე);

მოყვანილ პროგრამაში ხდება მითითებული ფაილიდან გამოსახულების ასახვის დემონსტრირება.

```

{
// პროგრამა 16.9
// პროგრამაში ხდება გამოსახულების ასახვა მითითებული ფაილიდან
Graphics SaxataviPurceli = CreateGraphics();
Bitmap Gamosaxuleba2 = new Bitmap("ani2.jpg");
SaxataviPurceli.DrawImage(Gamosaxuleba2, 5, 5, 250, 250);
}

```

ცხრილი 16.10. Bitmap კლასის თვისებები.

თვისება	ტიპი	აღწერა
Flags	int	ალამი-ატრიბუტების ნაკრები
FrameDimensionsList	Guid []	მასივი, რომელიც მიუთითებს გამოსახულებაში კადრების განზომილებებს
Height	int	გამოსახულების სიმაღლე მასშტაბირების გარეშე
HorizontalResolution	float	ჰორიზონტალური გარჩევის უნარი პიქსელი/დუიმიზე
Palette	ColorPalette	ფერების პალიტრა
PhysicalDimension	Size	გამოსახულების სიგანე და სიმაღლე
PixelFormat	PixelFormat	გამოსახულების პიქსელების ფორმატი
PropertyIDList	int []	თვისებების იდენტიფიკატორების მასივი, რომლებიც გამოსახულებასთან ერთადაა შენახული
PropertyItems	PropertyItem []	PropertyItem ობიექტების მასივი, რომელიც მოცემულ გამოსახულებას აღწერს
RawFormat	ImageFormat	გამოსახულების ფორმატი
Size	Size	გამოსახულების სიგანე და სიმაღლე
VerticalResolution	float	ვერტიკალური გარჩევის უნარი პიქსელი/დუიმიზე
Width	int	გამოსახულების სიგანე მასშტაბირების გარეშე

მოყვანილ პროგრამაში ხდება დიალოგურ ფანჯარაში არჩეული ფაილიდან გამოსახულების ასახვის დემონსტრირება.

```

{
//   პროგრამა 16.10
//   პროგრამაში ხდება გამოსახულების ასახვა დიალოგურ ფანჯარაში არჩეული ფაილიდან
Graphics SaxataviPurceli = CreateGraphics();
OpenFileDialog Surati_Failidan = new OpenFileDialog();
Bitmap Gamosaxuleba;
if ( Surati_Failidan.ShowDialog() == DialogResult.OK )
{
    Gamosaxuleba = new Bitmap(Surati_Failidan.FileName);
    SaxataviPurceli.DrawImage(Gamosaxuleba, 5, 5, 250, 250);
}
}

```

SetPixel() მეთოდი გამოიყენება მითითებული პიქსელისთვის (მონიტორის ეკრანის წერტილისთვის) ფერის მისანიჭებლად. მისი სინტაქსია:

```
void SetPixel(int x, int y, Color ფერი);
```

ცხრილი 16.11. Bitmap კლასის მეთოდები.

მეთოდი	აღწერა
Clone() (სტატიკური)	ქმნის Bitmap ობიექტის ასლს
FromHicon()	ქმნის Bitmap ობიექტს Windows-ის პიქტოგრამის დესკრიპტორიდან
FromResource()	ქმნის Bitmap ობიექტს Windows-ის რესურსიდან
GetBounds()	გასცემს სწორკუთხა უბანს, რომელშიც უნდა მოთავსდეს გამოსახულება
GetEncoderParameterList()	გასცემს ინფორმაციას გამოსახულების შენახვის პარამეტრების შესახებ
GetFrameCount()	გასცემს გამოსახულებაში კადრების რაოდენობას
GetHbitmap()	გასცემს Windows-ის ბიტების კარტის დესკრიპტორს მოცემული გამოსახულებისათვის
GetHicon()	გასცემს Windows-ის პიქტოგრამის დესკრიპტორს მოცემული გამოსახულებისათვის
GetPixel()	გასცემს გამოსახულების მითითებული პიქსელის ფერს
GetPropertyItem()	გასცემს PropertyItem ობიექტს
GetThumbnailImage()	გასცემს მოცემული გამოსახულების პიქტოგრამას
LockBits()	გამოსახულებას აფიქსირებს ოპერატიულ მეხსიერებაში
MakeTransparent()	გამოსახულების ნაწილებს გამჭვირვალეს ხდის
RemovePropertyItem()	მოცემული გამოსახულებიდან შლის PropertyItem ობიექტს
RotateFlip()	გამოსახულებს აბრუნებს ან სარკისებურად ასახავს
Save()	გამოსახულებას ნაკადში ინახავს
SaveAdd()	ახდენს ორი გამოსახულების კომბინირებას
SelectActiveFrame()	მრავალკადრიანი გამოსახულებიდან ირჩევს ერთ კადრს
SetPixel()	აყენებს გამოსახულების მითითებული პიქსელის ფერს
SetPropertyItem()	აყენებს ობიექტის მნიშვნელობას
SetResolution()	აყენებს გამოსახულების გარჩევის უნარს
UnlockBits()	აუქმებს ოპერატიულ მეხსიერებაში გამოსახულების ფიქსაციას

მოყვანილ პროგრამაში ხდება SetPixel() მეთოდის გამოყენებით გამოსახულების ასახვის დემონსტრირება.

```

{
//   პროგრამა 16.11
//   პროგრამაში ხდება გამოსახულების ასახვა SetPixel() მეთოდის გამოყენებით
Graphics SaxataviPurceli = CreateGraphics();
Bitmap Gamosaxuleba1 = new Bitmap(90, 90);           //   სიგანე = სიმაღლე = 90
//   თეთრი ოთხკუთხედის ხატვა
for ( int x = 0; x < Gamosaxuleba1.Height; ++x )
    for ( int y = 0; y < Gamosaxuleba1.Width; ++y )
        Gamosaxuleba1.SetPixel(x, y, Color.White);
//   წითელი ხაზის ხატვა
for ( int x = 0; x < Gamosaxuleba1.Height; ++x )
    Gamosaxuleba1.SetPixel(x, x, Color.Red);
//   გამოსახულების გამოტანა

```

```
SaxataviPurceli.DrawImage(Gamosaxuleba1, 0, 0, 100, 100);
}
```

ხატვა შესაძლებელია, აგრეთვე, pictureBox კომპონენტის შიგნით. მასში გავავლოთ ორი ხაზი. ამის დემონსტრირება ხდება შემდეგი პროგრამით.

```
{
Graphics g= pictureBox1.CreateGraphics();
Pen pen1 = new Pen(Color.Blue, 2);
Pen pen2 = new Pen(Color.Red, 2);
g.DrawLine(pen1, 0, 0, 200, 200);
g.DrawLine(pen2, 0, 0, 200, 500);
}
```

შეგვიძლია pictureBox კომპონენტის ფონად ავირჩიოთ რაიმე სურათი, რისთვისაც უნდა გამოვიყენოთ BackgroundImage თვისება, და შემდეგ დავხატოთ რაიმე სურათი.

მონაცემების დაშიფვრა და გაშიფვრა

მონაცემების დაცვის ერთ-ერთი სახეა მათი დაშიფვრა. .NET გარემო შეიცავს კლასებს, რომლებიც უზრუნველყოფენ როგორც სიმეტრიულ, ისე ასიმეტრიულ კრიპტოგრაფიას. სიმეტრიულ კრიპტოგრაფიას, ზოგჯერ უწოდებენ კრიპტოგრაფიას დახურული გასაღებით, ხოლო ასიმეტრიულ კრიპტოგრაფიას კი - კრიპტოგრაფიას ღია გასაღებით.

სიმეტრიული კრიპტოგრაფიის შემთხვევაში მონაცემების დაშიფვრისა და გაშიფვრისათვის ერთი და იგივე გასაღები გამოიყენება. .NET გარემოში გამოყენებულია სიმეტრიული კრიპტოგრაფიის ოდნავ გართულებული ფორმა, რომელსაც დაშიფრული ბლოკების გადაბმა ეწოდება. მონაცემების გასაშიფრად ამ ფორმის გამოყენების შემთხვევაში უნდა ვიცოდეთ არა მხოლოდ გასაღები, არამედ მაინიაცილებელი ვექტორიც. სიმეტრიული კრიპტოგრაფიის მიმართულებით შექმნილია ბევრი ალგორითმი, რომლებიც ორიენტირებულია იმაზე, რომ რთული იყოს გასაღების გარეშე მონაცემების გაშიფვრა. .NET გარემოში გამოყენებულია სიმეტრიული კრიპტოგრაფიის ოთხი ალგორითმი: DES, RC2, Rijndael და Triple DES.

ასიმეტრიული კრიპტოგრაფიის შემთხვევაში მონაცემების დაშიფვრისა და გაშიფვრისათვის გამოიყენება ორი გასაღები: ღია და დახურული. ღია გასაღების გამოყენებით დაშიფრული მონაცემების გაშიფვრა შესაძლებელია მხოლოდ შესაბამისი დახურული გასაღებით. .NET გარემოში გამოყენებულია ასიმეტრიული კრიპტოგრაფიის ორი ალგორითმი: RSA და DSA.

თუ გასაღები არ გვაქვს, მაშინ ასიმეტრიული შიფრის "გატეხვა" უფრო რთულია, ვიდრე სიმეტრიულის. "გატეხვა" არის პროცესი, როცა გასაღები არ გვაქვს და სხვადასხვა ხერხებით ვცდილობთ დაშიფრული მონაცემების გაშიფვრას. გარდა ამისა, მონაცემების ასიმეტრიული დაშიფვრა მოითხოვს უფრო მეტ გამოთვლებს, ვიდრე სიმეტრიული. კრიპტოგრაფიის კლასები მოთავსებულია **System.Security.Cryptography** სახელების სივრცეში.

სიმეტრიული კრიპტოგრაფია

სიმეტრიული კრიპტოგრაფიისათვის გამოიყენება TripleDESCryptoServiceProvider კლასი. მისი თვისებები მოყვანილია ცხრილში 16.12, მეთოდები კი - ცხრილში 16.13.

მონაცემების სიმეტრიული დაშიფვრისათვის გამოიყენება ნაკადები. დაშიფვრა ხორციელდება CryptoStream ობიექტის საშუალებით. რადგან ერთი ნაკადის გამოსასვლელი შეიძლება მივაწოდოთ მეორე ნაკადის შესასვლელს, ამიტომ CryptoStream კლასი შეგვიძლია

გამოვიყენოთ FileStream კლასთან ერთად ფაილში დაშიფრული მონაცემების ჩასაწერად და წასაკითხად, აგრეთვე, NetworkStream კლასთან ერთად ქსელში გადასაცემი მონაცემების დასაშიფრად.

მოყვანილი პროგრამა ასრულებს ფაილში ჩასაწერი მონაცემების დაშიფვრას.

```
{
//   პროგრამა 16.12
//   პროგრამაში ხდება ფაილში ჩასაწერი მონაცემების დაშიფვრა
FileStream SafailoNakadi1 = File.Create("temp.txt");
//   ახალი კრიპტოგრაფიული პროვაიდერის შექმნა
TripleDESCryptoServiceProvider CriptoProvaideri = new TripleDESCryptoServiceProvider();
//   დამშიფრავი ნაკადის შექმნა საფაილო ნაკადში
CryptoStream CriptoNakadi = new CryptoStream(SafailoNakadi1, CriptoProvaideri.CreateEncryptor(),
CryptoStreamMode.Write);
StreamWriter ChawerisNakadi = new StreamWriter(CriptoNakadi);
ChawerisNakadi.WriteLine("საბა და ანა");
ChawerisNakadi.WriteLine("ლიკა და რომანი");
ChawerisNakadi.Close();
//   გასაღების და მანიციალიზებული ვექტორის შენახვა
FileStream SafailoNakadi2 = File.Create("encrypted.key");
BinaryWriter OrobitiNakadi = new BinaryWriter(SafailoNakadi2);
OrobitiNakadi.Write(CriptoProvaideri.Key);
OrobitiNakadi.Write(CriptoProvaideri.IV);
OrobitiNakadi.Close();
}
```

ცხრილი 16.12. TripleDESCryptoServiceProvider კლასის თვისებები

თვისება	ტიპი	აღწერა
BlockSize	Int	დასაშიფრი ან გასაშიფრი ბაიტების რაოდენობა ერთი ოპერაციის შესრულებისას
FeedbackSize	Int	მონაცემების რაოდენობა თითოეული ბლოკიდან, რომლებიც გამოიყენება მომდევნო ბლოკის შიფრაციისათვის
IV	Byte	მანიციალიზებული ვექტორი მოცემული ალგორითმისთვის
Key	Byte	დაშიფვრის გასაღები მოცემული ალგორითმისთვის
KeySize	Int	გასაღებში ბიტების რაოდენობა
LegalBlockSizes	KeySizes[]	ბლოკების ზომები, რომლებსაც მოცემული ალგორითმი უზრუნველყოფს
LegalKeySizes	KeySizes[]	გასაღებების ზომები, რომლებსაც მოცემული ალგორითმი უზრუნველყოფს
Mode	CipherMode	განსაზღვრავს სიმეტრიული ალგორითმის ოპერაციის რეჟიმს
Padding	PaddingMode	განსაზღვრავს ბაიტებს, რომლებიც დაემატება უკანასკნელ ბლოკს, რომელსაც არ აქვს საკმარისი სიგრძე

პროგრამაში ჯერ იქმნება TripleDESCryptoServiceProvider კლასის ობიექტი მონაცემების დამშიფრავი სამსახურის უზრუნველყოფისათვის. შემდეგ იქმნება CryptoStream კლასის ობიექტი, რომელიც მოახდენს მონაცემების დაშიფვრას. CryptoStream კლასის კონსტრუქტორს სამი პარამეტრი გადაეცემა. პირველია ობიექტი-ნაკადი, რომელიც გამოიყენება ფაილში მონაცემების ჩასაწერად ან წასაკითხად. მეორე პარამეტრია მეთოდი, რომელიც ახდენს მონაცემების დაშიფვრას ან გაშიფვრას. მესამე პარამეტრია რეჟიმი, რომელიც განსაზღვრავს CryptoStream ობიექტი კითხულობს თუ წერს მონაცემებს.

პროგრამაში StreamWriter ობიექტიდან მონაცემები გადაეცემა CryptoStream ობიექტს, რომელიც ახდენს ამ მონაცემების დაშიფვრას, შემდეგ კი დაშიფრული მონაცემები გადაეცემა FileStream ობიექტს, რომელიც ამ მონაცემებს ფაილში ჩაწერს. ბოლოს პროგრამა ფაილში წერს გასაღებსა და მაინიალიზებულ ვექტორს.

ცხრილი 16.13. TripleDESCryptoServiceProvider კლასის მეთოდები

მეთოდი	აღწერა
Clear()	ათავისუფლებს ალგორითმის მიერ დაკავებულ რესურსებს
CreateDecryptor()	ქმნის ობიექტს, რომელიც შეიძლება გამოყენებული იყოს გაშიფვრის ოპერაციის შესასრულებლად
CreateEncryptor()	ქმნის ობიექტს, რომელიც შეიძლება გამოყენებული იყოს დაშიფვრის ოპერაციის შესასრულებლად
GenerateIV()	ახდენს ახალი შემთხვევითი მაინიციალიზებელი ვექტორის გენერირებას
GenerateKey()	ახდენს ახალი შემთხვევითი გასაღების გენერირებას
ValidateKeySize()	განსაზღვრავს, აქვს თუ არა მიმდინარე გასაღებს კორექტული ზომა

მოყვანილი პროგრამა ასრულებს ფაილიდან წაკითხული მონაცემების გაშიფვრას.

```

{
//   პროგრამა 16.13
//   პროგრამაში ხდება ფაილიდან წაკითხული მონაცემების გაშიფვრა
TripleDESCryptoServiceProvider CriptoProvaideri = new TripleDESCryptoServiceProvider();
FileStream SafailoNakadi1 = File.OpenRead("encrypted.key");
BinaryReader OrobitiWamkitxavi = new BinaryReader(SafailoNakadi1);
//   ფაილიდან გასაღებისა და ვექტორის წაკითხვა
CriptoProvaideri.Key = OrobitiWamkitxavi.ReadBytes(24);
CriptoProvaideri.IV = OrobitiWamkitxavi.ReadBytes(8);
FileStream SafailoNakadi2 = File.OpenRead("temp.txt");
//   ფაილიდან წაკითხული მონაცემების გაშიფვრა
CryptoStream CriptoNakadi = new CryptoStream(SafailoNakadi2,
                                             CriptoProvaideri.CreateDecryptor(), CryptoStreamMode.Read);
StreamReader WakitxvisNakadi = new StreamReader(CriptoNakadi);
label1.Text = WakitxvisNakadi.ReadToEnd().ToString() + '\n' +
              "გასაღების ზომა = " + CriptoProvaideri.KeySize.ToString() +
              '\n' + "ბლოკის ზომა = " + CriptoProvaideri.BlockSize.ToString();
label2.Text = "რეჟიმი = " + CriptoProvaideri.Mode.ToString();
WakitxvisNakadi.Close();
}

```

პროგრამაში ხდება დაშიფრული მონაცემების წაკითხვა FileStream ობიექტის მიერ, მათი

გამიფვრა CryptoStream ობიექტის მიერ, შემდეგ კი მათი გადაცემა StreamReader ობიექტისათვის და ეკრანზე გამოტანა.

ასიმეტრიული კრიპტოგრაფია

ასიმეტრიული კრიპტოგრაფიის ალგორითმები განსაზღვრულია არა ნაკადებთან, არამედ, მცირე ზომის მონაცემებთან სამუშაოდ. მოყვანილ პროგრამაში ხდება მონაცემების დაშიფვრისა და გაშიფვრის დემონსტრირება ასიმეტრიული კრიპტოგრაფიის მეთოდების გამოყენებით.

```
{
//   პროგრამა 16.14
//   პროგრამაში ხდება მონაცემების დაშიფვრისა და გაშიფვრის დემონსტრირება
//   ასიმეტრიული კრიპტოგრაფიის მეთოდების გამოყენებით
label1.Text = "";
//   ახალი კრიპტოგრაფიული პროვაიდერის შექმნა
RSACryptoServiceProvider CriptoProvaideri = new RSACryptoServiceProvider();
Byte[] DasashifriMasivi = { 1, 2, 3, 4, 5 };
//   მონაცემების დაშიფვრა
Byte[] DashifruliMasivi = CriptoProvaideri.Encrypt(DasashifriMasivi, false);
label1.Text = "დაშიფრული მონაცემები: \n";
for ( int indexi = 0; indexi < DashifruliMasivi.GetLength(0); indexi++ )
    label1.Text += DashifruliMasivi[indexi].ToString() + " ";
label1.Text += '\n';
//   მონაცემების გაშიფვრა
Byte[] GashifruliMasivi = CriptoProvaideri.Decrypt(DashifruliMasivi, false);
label1.Text += "გაშიფრული მონაცემები: \n";
for ( int indexi = 0; indexi < GashifruliMasivi.GetLength(0); indexi++ )
    label1.Text += GashifruliMasivi[indexi].ToString() + " ";
label2.Text += "ალგორითმის სახელი - " + CriptoProvaideri.KeyExchangeAlgorithm.ToString() + '\n'
    + "გასაღების ზომა = " + CriptoProvaideri.KeySize.ToString() + '\n';
}
```

სერიულ პორტებთან მუშაობა

მკითხველმა უნდა იცოდეს სერიული პორტის (COM პორტის) მუშაობის პრინციპები, რადგან წიგნის მიზანი არ არის მათი დეტალური განხილვა. სერიულ პორტებთან მუშაობას პროგრამისტის კუთხით განვიხილავთ. პორტებთან სამუშაოდ SerialPort კლასი გამოიყენება. მისი ზოგიერთი თვისება და მეთოდი მოყვანილია ცხრილებში 16.14-16.15.

სერიულ პორტთან მუშაობა კონკრეტულ მაგალითზე განვიხილოთ. დავუშვათ, კომპიუტერში გვინდა მივიღოთ მიკროპროცესორული მოწყობილობის ორი რეგისტრის შემცველობა. მიკროპროცესორული მოწყობილობა მუშაობს MODBUS პროტოკოლით და კომპიუტერს COM პორტის საშუალებით უკავშირდება. აქვე შევნიშნოთ, რომ პროტოკოლის არჩევა კონკრეტულ მოწყობილობაზეა დამოკიდებული. მოწყობილობის თითოეული რეგისტრი ოთხბაიტანია. ამ პროტოკოლის მიხედვით კომპიუტერმა მოწყობილობას უნდა გაუგზავნოს ინფორმაცია, რომელსაც შემდეგი ფორმატი აქვს:

პირველი ბაიტი - მოწყობილობის მისამართი;

მეორე ბაიტი - წაკითხვის ბრძანება;

მესამე და მეოთხე ბაიტები - მოწყობილობის იმ რეგისტრის მისამართი, საიდანაც ინფორმაცია უნდა მივიღოთ;

მეხუთე და მეექვსე ბაიტები - რეგისტრების რაოდენობა;

მეშვიდე და მერვე ბაიტები - საკონტროლო ჯამი.

ამ ბრძანების საპასუხოდ მოწყობილობა დაახლოებით 600 მილიწამის შემდეგ კომპიუტერს უგზავნის ინფორმაციას, რომელსაც პროტოკოლის მიხედვით, შემდეგი ფორმატი აქვს:

პირველი ბაიტი - მოწყობილობის მისამართი;

მეორე ბაიტი - წაკითხვის ბრძანება;

მესამე ბაიტი - ბაიტების რაოდენობა;

მეოთხე, მეხუთე, მეექვსე და მეშვიდე ბაიტები - პირველი რეგისტრის მნიშვნელობები;

მერვე, მეცხრე, მეათე და მეთერთმეტე ბაიტები - მეორე რეგისტრის მნიშვნელობები;

მეთორმეტე და მეცამეტე ბაიტები - საკონტროლო ჯამი.

ოთხბაიტის რეგისტრიდან მიღებული მონაცემი მთელ რიცხვად შეგვიძლია შემდეგნაირად გარდავქმნათ:

$$Y = X_1 * 256^3 + X_2 * 256^2 + X_3 * 256 + X_4 * 356^0$$

აქ Y - გარდაქმნის შედეგად მიღებული მთელი რიცხვია, X₁ - რეგისტრის უფროსი ბაიტია, X₂ - რეგისტრის მომდევნო ბაიტი და ა.შ. გარდაქმნის შედეგად მიღებული მთელი რიცხვები შეგვიძლია ჩავწეროთ ფაილში ან მონაცემთა ბაზაში (შემდგომი დამუშავების მიზნით), ან ავსახოთ გრაფიკის სახით.

კონკრეტულად, თუ რომელი პორტი გამოყო ოპერაციულმა სისტემამ მიკროპროცესორულ მოწყობილობასთან სამუშაოდ, შეგვიძლია ვნახოთ Device Manager ფანჯარაში. სწორედ, ამ გამოყოფილი პორტის სახელი უნდა მივუთითოთ პროგრამაში პორტის პარამეტრების განსაზღვრის დროს.

ზემოთ თქმულის დემონსტრირება ხდება მოყვანილ პროგრამაში. პროგრამის შეტანამდე using დირექტივების ბლოკს უნდა დავუმატოთ დირექტივა using System.IO.Ports; .

{

// **პროგრამა 16.15**

// **პროგრამაში ხდება სერიულ პორტთან მუშაობის დემონსტრირება**

BinaryWriter file_out = new BinaryWriter(new FileStream("faii.dat", FileMode.Create));

long ricxvi1, ricxvi2;

int wakitxuli_baitebis_raodenoba;

byte[] buf8 = new byte[8]; // ბუფერი მონაცემების გასაგზავნად

byte[] buf13 = new byte[13]; // ბუფერი მონაცემების მისაღებად

// მოწყობილობისთვის გასაგზავნი მონაცემების მომზადება

buf8[0] = 2; // მოწყობილობის მისამართი

buf8[1] = 3; // მონაცემების წაკითხვის ბრძანება

buf8[2] = 29; // რეგისტრის მისამართი

buf8[3] = 79;

buf8[4] = 0; // რეგისტრების რაოდენობა

buf8[5] = 2;

buf8[6] = 243; // საკონტროლო ჯამი

buf8[7] = 131;

SerialPort COM_Port1 = new SerialPort();

// პორტის პარამეტრების მომზადება

COM_Port1.BaudRate = 38400; // პორტის სწრაფქმედება

COM_Port1.DataBits = 8; // ბაიტში მონაცემების ბიტების რაოდენობა

COM_Port1.Parity = Parity.None; // პარიტეტი არ გვაქვს

COM_Port1.PortName = textBox1.Text; // პორტის სახელი

```

COM_Port1_1.Open(); // პორტის გახსნა
// პორტში მონაცემების გაგზავნა
COM_Port1_1.Write(buf8, 0, 8);
Thread.Sleep(600); // დაყოვნება 600 მილიწამი
// პორტიდან მონაცემების მიღება
wakitxuli_baitebis_raodenoba = COM_Port1_1.Read(buf13, 0, 13);
// რეგისტრებიდან მიღებული მონაცემების გარდაქმნა
ricxvi1 = buf13[3] * 256 * 256 * 256 + buf13[4] * 256 * 256 + buf13[5] * 256 + buf13[6];
ricxvi2 = buf13[7] * 256 * 256 * 256 + buf13[8] * 256 * 256 + buf13[9] * 256 + buf13[10];
// მონაცემების ჩაწერა ფაილში
file_out.Write(ricxvi1);
file_out.Write(ricxvi2);
file_out.Close();
//
label1.Text = COM_Port1_1.PortName; // პორტის სახელის ეკრანზე გამოტანა
label2.Text = wakitxuli_baitebis_raodenoba.ToString();
label3.Text = "წაკითხული ბაიტების რაოდენობა = " + COM_Port1_1.BytesToRead.ToString() +
            "\nჩაწერილი ბაიტების რაოდენობა = " + COM_Port1_1.BytesToWrite.ToString();
// გარდაქმნილი მონაცემების ეკრანზე გამოტანა
label4.Text = ricxvi1.ToString();
label5.Text = ricxvi2.ToString();
//
COM_Port1_1.Close(); // პორტის დახურვა
}

```

ცხრილი 16.14. SerialPort კლასის თვისებები

თვისება	აღწერა
int BaudRate	სერიული პორტის მუშაობის სიხშირე
int BytesToRead	განსაზღვრავს მონაცემების ბაიტების რაოდენობას მიმდებ ბუფერში
int BytesToWrite	განსაზღვრავს მონაცემების ბაიტების რაოდენობას გადაცემის ბუფერში
int DataBits	განსაზღვრავს მონაცემების ბიტების სტანდარტულ რაოდენობას 1 ბაიტში. მისი მნიშვნელობაა 5÷8
Encoding Encoding	განსაზღვრავს ბაიტის კოდირებას ტექსტის გადაცემამდე ან გადაცემის შემდეგ გარდაქმნისთვის
Handshake Handshake	განსაზღვრავს პროტოკოლს სერიული პორტით მონაცემების გადაცემისათვის. იღებს მნიშვნელობებს: None, XonXoff, RequestToSend, RequestToSendXonXoff
bool IsOpen	შეიცავს true-ს, თუ სერიული პორტი ღიაა და false-ს, თუ პორტი დახურულია
string NewLine	განსაზღვრავს სიდიდეს, რომელიც გამოიყენება გამოძახების დასასრულის ინტერპრეტირებისათვის ReadLine() და WriteLine() მეთოდებში
Parity Parity	განსაზღვრავს პარიტეტის შემოწმების პროტოკოლს. იღებს მნიშვნელობებს: None, Odd, Even, Mark, Space
string PortName	განსაზღვრავს სერიული პორტის სახელს

ცხრილი 16.14. SerialPort კლასის თვისებები (გაგრძელება)

int ReadBufferSize	განსაზღვრავს სერიული პორტის შესასვლელი (მიმღები) ბუფერის ზომას
int ReadTimeout	განსაზღვრავს მილიწამების რაოდენობას, რომელიც უნდა გავიდეს მანამ, სანამ ტაიმ-აუტი აღიმგრება, როცა კითხვის ოპერაცია არ არის დამთავრებული
StopBits StopBits	განსაზღვრავს სტოპ-ბიტების რაოდენობას. ის იღებს მნიშვნელობებს: None, One, Two, OnePointFive
int WriteBufferSize	განსაზღვრავს სერიული პორტის გამოსასვლელი (გადაცემის) ბუფერის ზომას
int WriteTimeout	განსაზღვრავს მილიწამების რაოდენობას, რომელიც უნდა გავიდეს მანამ, სანამ ტაიმ-აუტი აღიმგრება, როცა ჩაწერის ოპერაცია არ არის დამთავრებული

ცხრილი 16.15. SerialPort კლასის მეთოდები

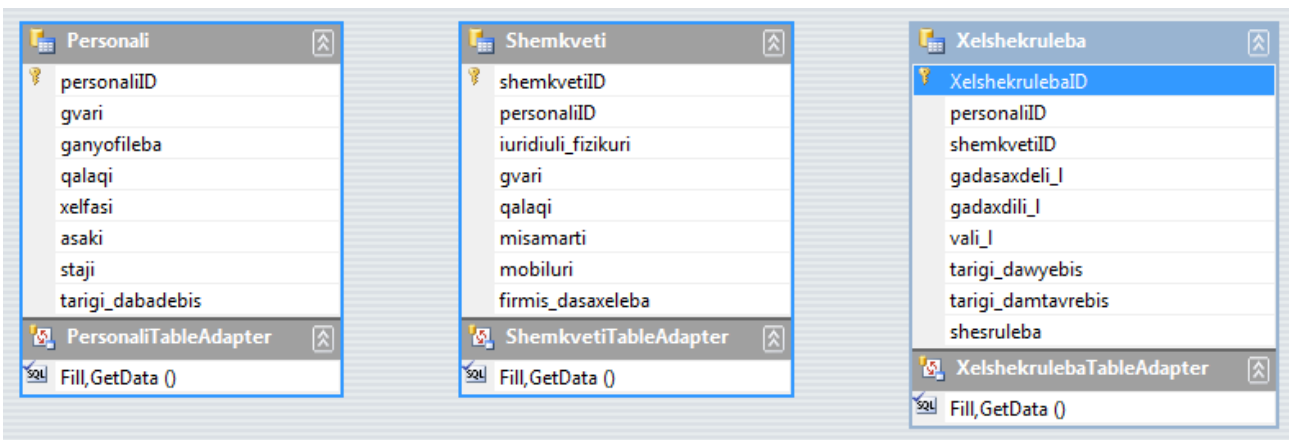
მეთოდი	აღწერა
void Close()	ხურავს სერიულ პორტთან შეერთებას, IsOpen თვისებას false მნიშვნელობას ანიჭებს და ათავისუფლებს შესაბამის რესურსებს
void Dispose()	ათავისუფლებს პორტის მიერ დაკავებულ უმართავ რესურსებს
string[] GetPortNames()	გასცემს კომპიუტერის სერიული პორტების სახელებს
void Open()	ხსნის ახალ შეერთებას სერიულ პორტთან
int Read(byte[] ბუფერი, int წანაცვლება, int რაოდენობა)	სერიული პორტიდან კითხულობს მითითებული რაოდენობის ბაიტებს და ათავსებს მათ ბუფერში დაწყებული წანაცვლებით განსაზღვრული პოზიციიდან
int Read(char[] ბუფერი, int წანაცვლება, int რაოდენობა)	სერიული პორტიდან კითხულობს მითითებული რაოდენობის სიმბოლოებს და ათავსებს მათ ბუფერში დაწყებული წანაცვლებით განსაზღვრული პოზიციიდან
int ReadByte()	სინქრონულად კითხულობს ერთ ბაიტს შესასვლელი ბუფერიდან
int ReadChar()	სინქრონულად კითხულობს ერთ სიმბოლოს შესასვლელი ბუფერიდან
string ReadExisting()	შესასვლელ ნაკადში ათავსებს კოდირებაზე დაფუძნებულ ყველა უშუალოდ მისაწვდომ ბაიტს
string ReadLine()	მონაცემებს კითხულობს შესასვლელი ბუფერიდან NewLine სიმბოლომდე
string ReadTo(string სტრიქონი)	კითხულობს სტრიქონს მითითებულ სიმბოლომდე
void Write(string სტრიქონი)	მითითებულ სტრიქონს უგზავნის სერიულ პორტს
void Write(byte[] ბუფერი, int წანაცვლება, int რაოდენობა)	ბუფერიდან სერიულ პორტს უგზავნის მითითებული რაოდენობის ბაიტს
void Write(char[] ბუფერი, int წანაცვლება, int რაოდენობა)	ბუფერიდან სერიულ პორტს უგზავნის მითითებული რაოდენობის სიმბოლოს
void WriteLine(string სტრიქონი)	მითითებულ სტრიქონს და NewLine სიმბოლოს ათავსებს გამოსასვლელ ბუფერში

III ნაწილი. მონაცემთა ბაზები თავი 17. ADO.NET კლასები

მონაცემთა ბაზასთან მიმართვა Visual Studio .NET-ის საშუალებებით

ამ თავში ვნახავთ თუ როგორ უნდა ვიმუშაოთ SQL სერვერზე მოთავსებულ მონაცემთა ბაზასთან C# პროგრამიდან Active Data Objects ბიბლიოთეკის კლასების გამოყენებით .NET Framework პლატფორმისათვის (შემოკლებით ADO.NET). აქვე შევნიშნავ, რომ მკითხველს უნდა ჰქონდეს მინიმალური ცოდნა SQL სერვერის შესახებ. ქართულ ენაზე SQL სერვერი აღწერილია წიგნში [1].

დავუშვათ, სერვერზე შევქმენით Shekveta მონაცემთა ბაზა, რომელიც შემდეგ ობიექტებს შეიცავს: Personali, Shemkveti და Xelshekruleba ცხრილები, View_3 წარმოდგენა, Chemi_Proc და Myproc_Par შენახული პროცედურები და Maqsimaluri_Xelfasi ფუნქცია. ცხრილების სტრუქტურას აქვს შემდეგი სახე:



თითოეულ ცხრილს უნდა ჰქონდეს პირველადი გასაღები. ეს ცხრილები შევავსოთ მონაცემებით. Chemi_Proc შენახული პროცედურა გასცემს Personali ცხრილის სტრიქონებს. Myproc_Par შენახულ პროცედურას გამოაქვს ინფორმაცია მითითებული განყოფილების თანამშრომლების შესახებ. Maqsimaluri_Xelfasi ფუნქცია გასცემს მითითებული განყოფილების თანამშრომლების მაქსიმალურ ხელფასს.

SQL სერვერზე მოთავსებულ მონაცემთა ბაზასთან მიმართვა ADO.NET ბიბლიოთეკის კლასების გამოყენებით შემდეგნაირად ხორციელდება: C# პროგრამა უერთდება სერვერზე მოთავსებულ მონაცემთა ბაზას, იღებს საჭირო ობიექტებს, როგორცაა ცხრილები, წარმოდგენები, ფუნქციები და შენახული პროცედურები, და მათ იმ კომპიუტერის (კლიენტის) მეხსიერებაში ათავსებს, რომელზეც ეს პროგრამა მუშაობს. ამ ობიექტების სიმრავლეს მონაცემების ლოკალური ნაკრები ეწოდება. ამის შემდეგ, ჩვენ შეგვეძლება ამ ობიექტებთან მუშაობა. კერძოდ, ცხრილების სტრიქონების ცვლილება, ფუნქციებისა და შენახული პროცედურების გამოძახება და ა.შ. C# პროგრამა პერიოდულად უნდა დავუკავშიროთ მონაცემთა ბაზას იმ ცვლილებების შესანახად, რომელიც ჩვენ შევიტანეთ მონაცემების ლოკალურ ნაკრებში (ასლში).


მონაცემთა ბაზა შეიძლება იმყოფებოდეს იმ კომპიუტერზე, რომელზეც ჩვენი პროგრამა მუშაობს. თუმცა, ყველაფერი დამოკიდებულია გადასაწყვეტი ამოცანის მოთხოვნებზე. უმჯობესია მონაცემთა ბაზის მოთავსება უფრო მძლავრ კომპიუტერზე (სერვერზე), რომელსაც შეეძლება ერთდროულად მოემსახუროს იმ პროგრამებსა და მომხმარებლებს, რომლებიც SQL მოთხოვნებს აგზავნიან.

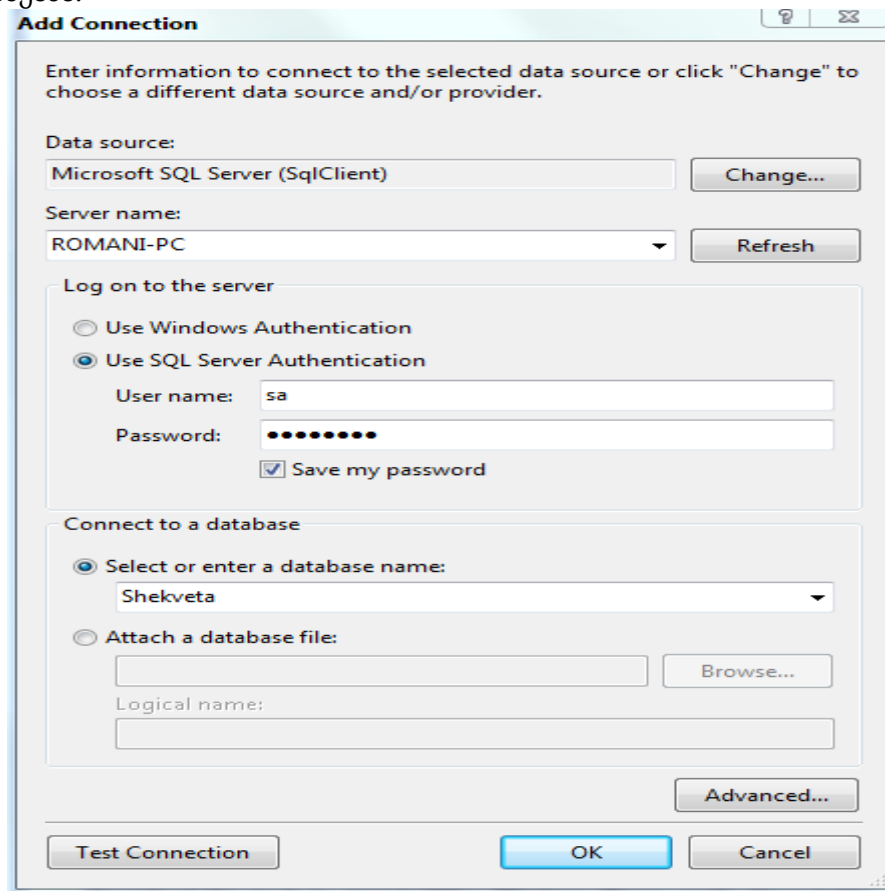
Visual Studio .NET გარემოს აქვს ფუნქციები, რომლებიც საშუალებას იძლევიან

დავუკავშირდეთ მონაცემთა ბაზას და ვიმუშაოთ მის ცხრილებთან. ეს ფუნქციები თავმოყრილია Server Explorer ფანჯარაში.

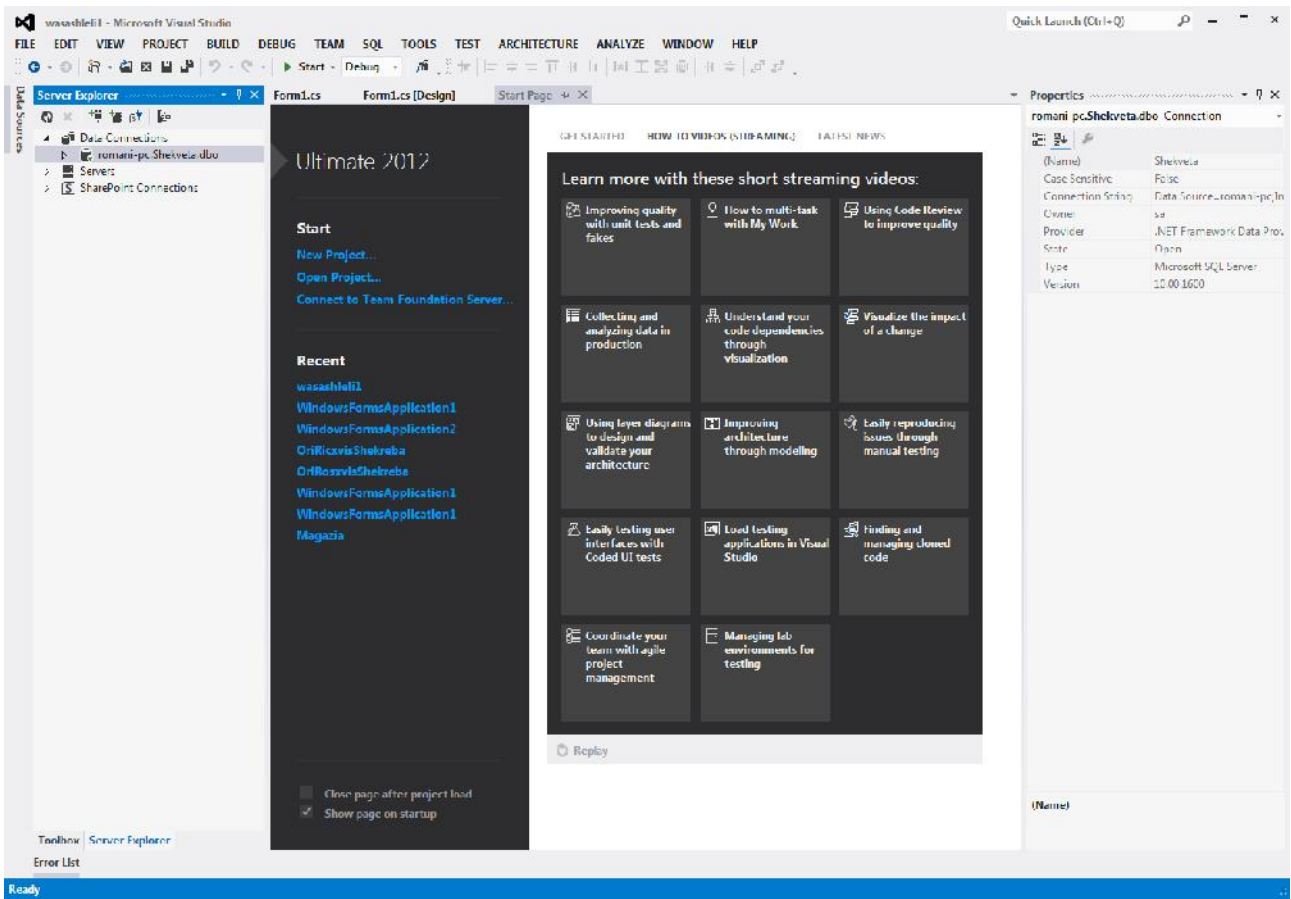
უწინარეს ყოვლისა, მონაცემთა ბაზასთან უნდა დავამყაროთ შეერთება (კავშირი). ამისათვის, ჯერ უნდა გავუშვათ Visual Studio .NET სისტემა, შემდეგ კი შევასრულოთ Tools მენიუს Connect to Database ბრძანება. გაიხსნება Add Connection ფანჯარა (ნახ. 17.1). Server name ველში შეგვაქვს სერვერის (კომპიუტერის) სახელი. შემდეგ უნდა ჩავრთოთ Use SQL Server Authentication გადამრთველი და User name და Password ველებში შევიტანოთ შესაბამისი ინფორმაცია. საჭირო ბაზას ვირჩევთ Select or enter a database name ჩამოშლადი სიდიდან. ვაჭერთ Ok კლავიშს.

მონაცემთა ბაზასთან შეერთების დამყარების შემდეგ, შეგვეძლება ცხრილების ნახვა და ცვლილება. ცხრილებთან მიმართვისათვის ვხსნით Server Explorer ფანჯარას (ნახ. 17.2), რისთვისაც ერთხელ ვაჭერთ გახსნილი ფანჯრის მარცხნივ მოთავსებულ Server Explorer პანელის პიქტოგრამას. თუ ეს პანელი არ ჩანს, მაშინ უნდა შევასრულოთ View მენიუს Server Explorer ბრძანება. ვხსნით ამ ფანჯრის Data Connections განშტოებას. გამოჩნდება მონაცემთა ბაზების სახელები. შემდეგ, ვხსნით ROMANI-PC.Shekveta.dbo განშტოებას, ბოლოს კი Tables განშტოებას (ნახ. 17.3). Personal ცხრილიდან სტრიქონების მისაღებად მიმთითებელს ვათავსებთ ამ ცხრილის პიქტოგრამაზე, ვხსნით კონტექსტურ მენიუს და ვასრულებთ Show Table Data ბრძანებას. ნახ. 17.4-ზე ნაჩვენებია Personal ცხრილის სტრიქონები.

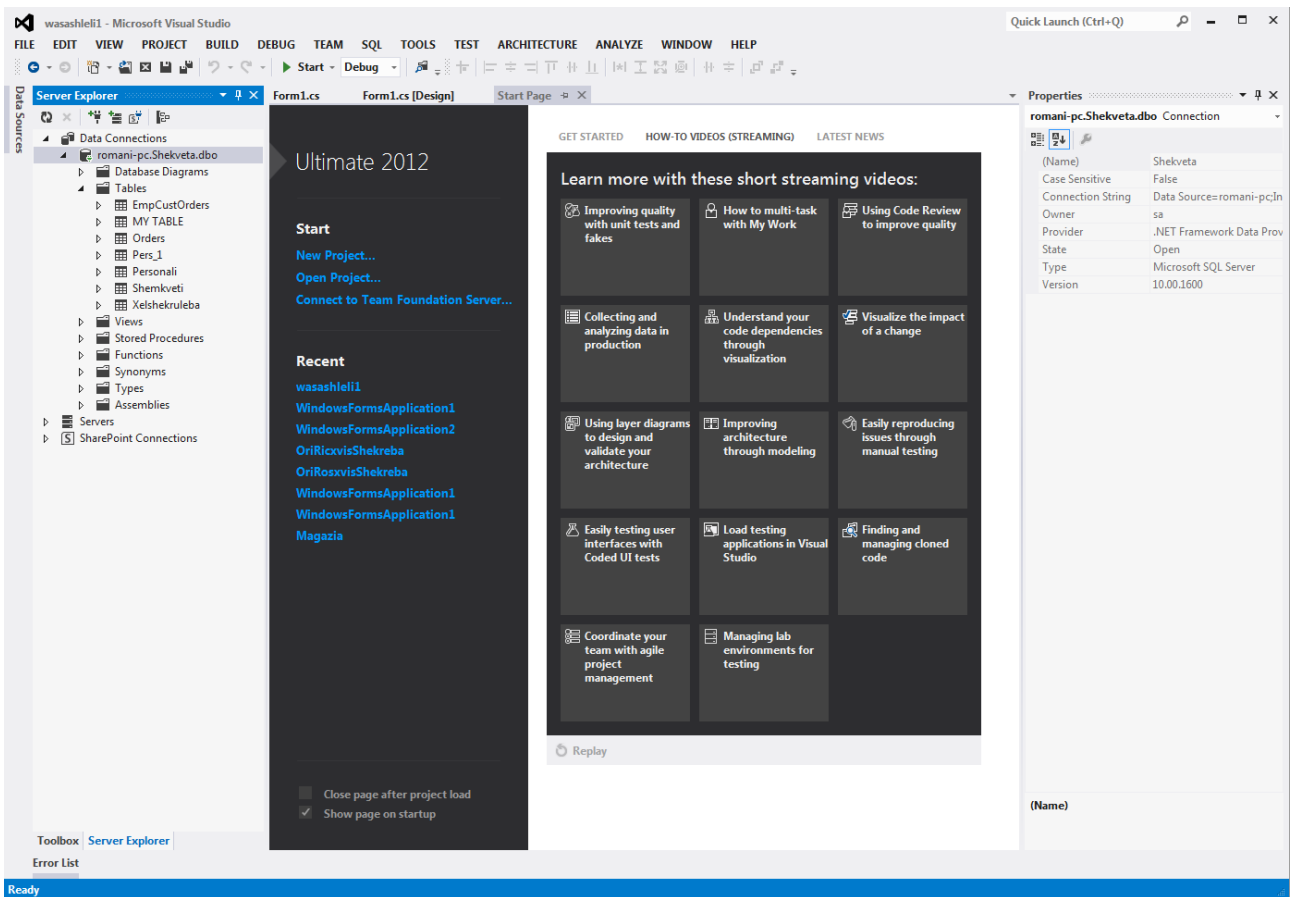
SQL მოთხოვნის შესატანად ვაჭერთ ინსტრუმენტების პანელის Show SQL Pane კლავიშს (ნახ. 17.5). მოთხოვნის შესასრულებლად ვაჭერთ Ctrl+R კლავიშებს ან ინსტრუმენტების პანელის  კლავიშს. SQL მოთხოვნა შეგვიძლია, აგრეთვე ავაგოთ ვიზუალურად თუ დავაჭერთ Show Diagram Pane კლავიშს (ნახ. 17.6). შეგვიძლია სვეტების თვისებების ნახვა თუ გავხსნით Personal განშტოებას, მოვნიშნავთ საჭირო სვეტს, გავხსნით კონტექსტურ მენიუს და შევასრულებთ Properties ბრძანებას.



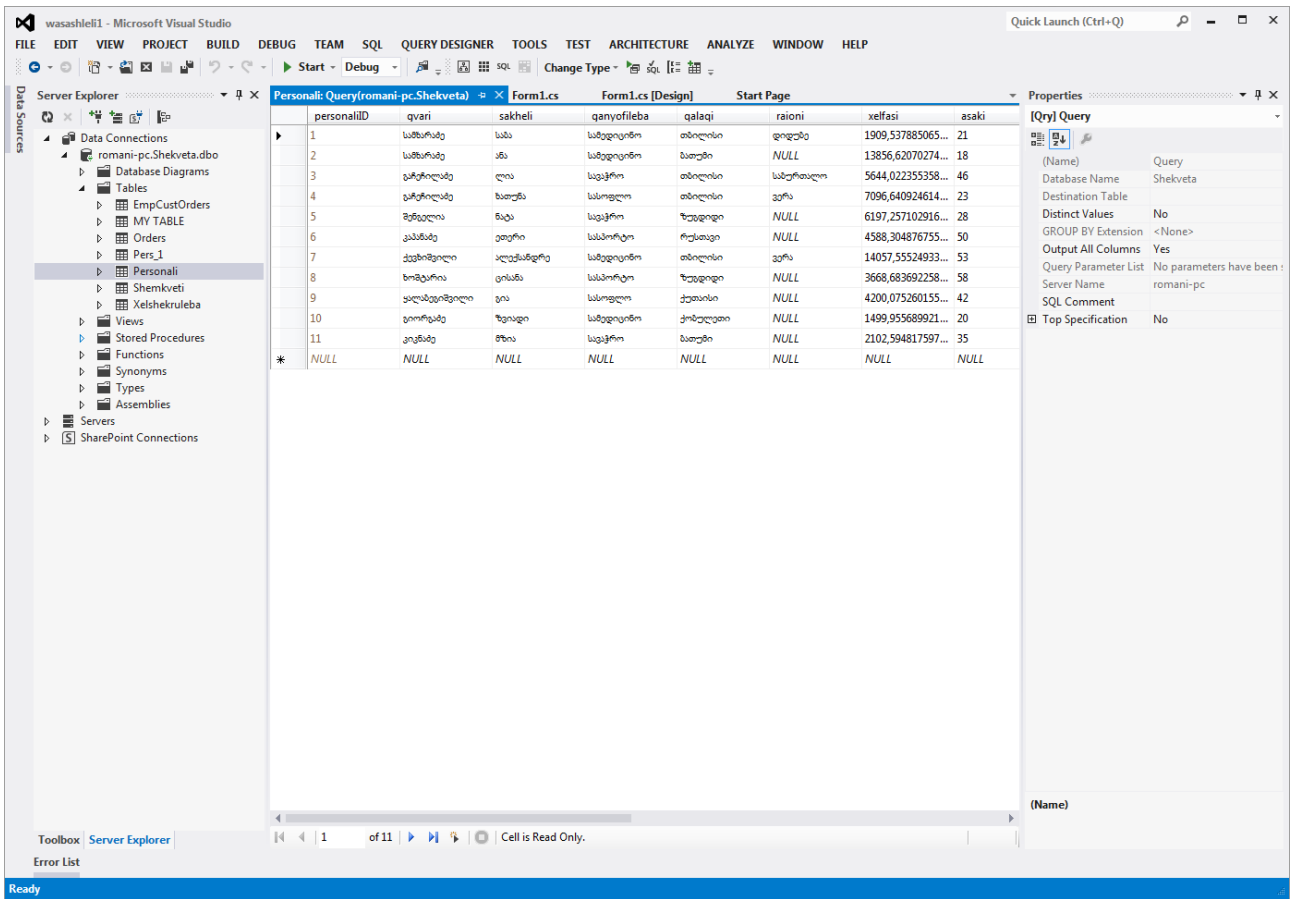
ნახ. 17.1.



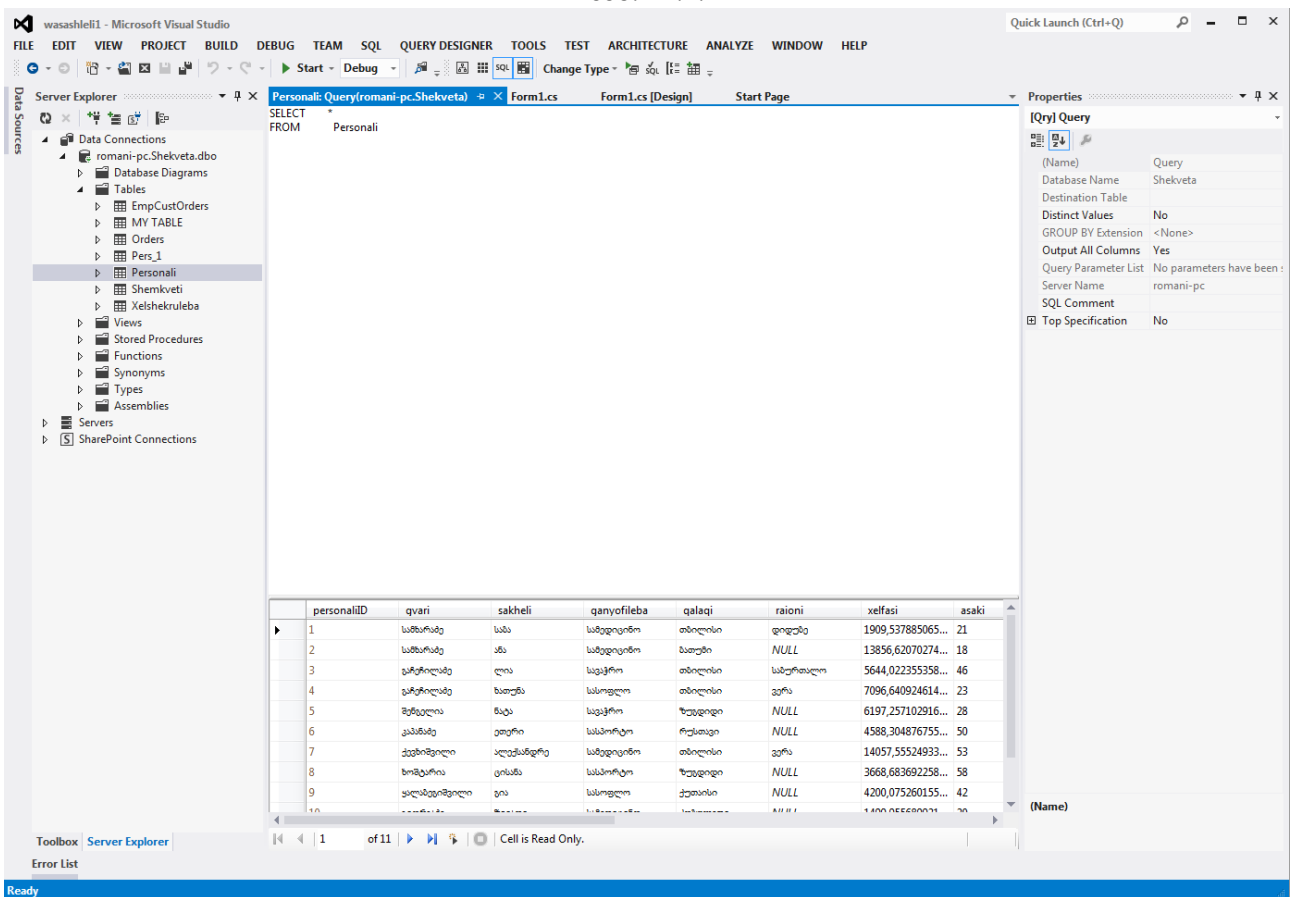
Յժ. 17.2.



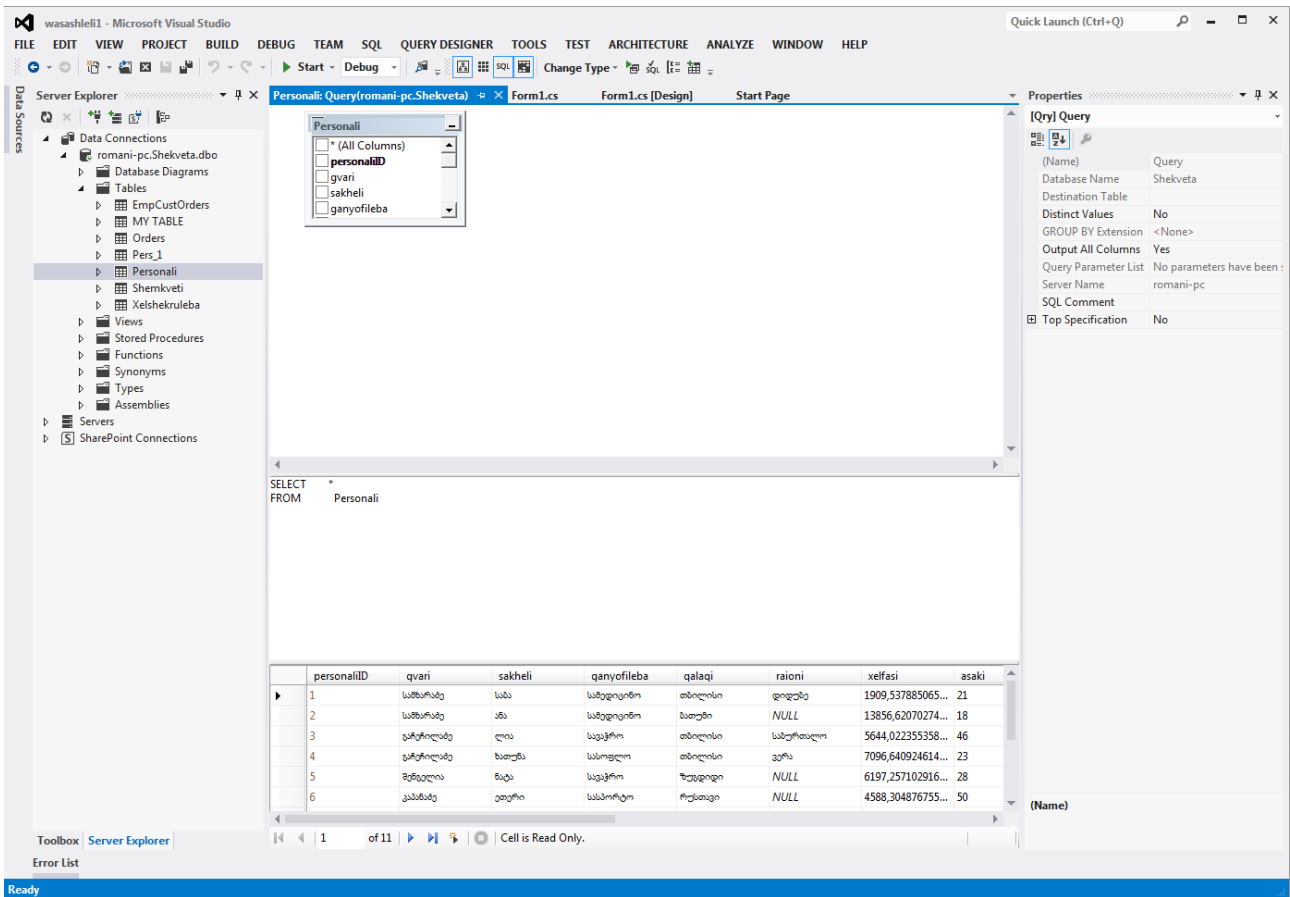
Յժ. 17.3.



ნახ. 17.4.



ნახ. 17.5.



ნახ. 17.6.

მონაცემთა ბაზასთან მუშაობა ვიზუალური და არავიზუალური კომპონენტებით

ამ განყოფილებაში მოყვანილ მაგალითში ნაჩვენებია თუ როგორ უნდა დავამყაროთ კავშირი SQL სერვერზე მოთავსებულ Shekveta მონაცემთა ბაზასთან და როგორ ავსახოთ ცხრილების სტრიქონები ფორმაზე მოთავსებულ კომპონენტებში. აქვე შევნიშნოთ, რომ ანალოგიურად ხორციელდება წარმოდგენებთან მუშაობა.

გავხსნათ ახალი პროექტი. გამოჩნდება Form1 კომპონენტი. მასზე ორჯერ სწრაფად დავაწკაპუნოთ და using დირექტივების ბლოკში ჩავუმატოთ using System.Data.SqlClient; დირექტივა. შემდეგ, ისევ გავხსნათ Form1.cs [Design] განყოფილება და Toolbox ფანჯრის AllWindowsForms განყოფილებიდან BindingSource კომპონენტი ფორმაზე მოვათავსოთ. მას დაერქმევა bindingSource1 სახელი და გამოჩნდება ფორმის ქვედა ზონაში, რადგან ის არავიზუალური კომპონენტია. მოვნიშნოთ ის. გავხსნათ Properties ფანჯრის DataSource სია და დავაჭიროთ Add Project Data Source მიმართვას.

გაიხსნება ფანჯარა (ნახ. 17.7), რომელშიც ვირჩევთ Database მონაცემთა წყაროს და ვაჭერთ Next კლავიშს. გახსნილ ფანჯარაში (ნახ. 17.8) ვირჩევთ მონაცემთა ბაზის მოდელს, კერძოდ Dataset-ს.

მომდევნო ფანჯარაში (ნახ. 17.9) ვხსნით Which data connection ... სიას და ვირჩევთ მონაცემთა ბაზას. თუ ამ სიაში არ არის ჩვენთვის საჭირო ბაზა, მაშინ ვაჭერთ New Connection კლავიშს. გახსნილი ფანჯრის (ნახ. 17.10) Server name სვეტში შევიტანოთ სერვერის (კომპიუტერის) სახელი. ჩავრთოთ Use SQL Server Authentication გადამრთველი. User name და

Password სვეტებში შევიტანოთ მომხმარებლის სახელი და პაროლი. Select or enter a database name სიიდან ავირჩიოთ საჭირო მონაცემთა ბაზის სახელი და დავაჭიროთ OK კლავიშს.

ვუბრუნდებით წინა ფანჯარას (ნახ. 17.11). დავაჭიროთ Connection string... სტრიქონის მარცხნივ მოთავსებულ + ნიშანს. გამოჩნდება მონაცემთა ბაზასთან შეერთების სტრიქონი, რომელშიც სამი პარამეტრი ჩანს: სერვერის (კომპიუტერის) სახელი - ROMANI-PC, მონაცემთა ბაზის სახელი - Shekveta და მომხმარებლის სახელი - sa. ჩავრთოთ Yes, include sensitive data in the connection string გადამრთველი (ნახ. 17.12). შედეგად, შეერთების სტრიქონს დაემატება მეოთხე პარამეტრი - პაროლი. ვაჭერთ Next კლავიშს.





გახსნილ ფანჯარაში (ნახ. 17.13) გამოჩნდება შეერთების სახელი. სურვილის შემთხვევაში, შეგვიძლია მისი შეცვლა. თუ Yes, save the connection as გადამრთველს ჩართულს დავტოვებთ, მაშინ შეერთების სტრიქონი შეინახება ჩვენი პროგრამის საკონფიგურაციო ფაილში. ვაჭერთ Next კლავიშს.

მომდევნო ფანჯარაში ვქმნით (ნახ. 17.14) ShekvetaDataSet მონაცემთა ნაკრებს, რომელიც მოთავსებული იქნება ჩვენს ლოკალურ კომპიუტერზე. მასში შეგვიძლია მოვათავსოთ ისეთი ობიექტები, როგორცაა ცხრილები, წარმოდგენები, შენახული პროცედურები და ფუნქციები. მათ გახსნილი ფანჯრიდან ვირჩევთ. ავირჩიოთ Personali, Shemkveti, Xelshekruleba, View_3, Myproc_Par და Sashuialo_Xelfasi ობიექტები. Finish კლავიშზე დაჭერის შემდეგ, ფორმის ქვეშ გამოჩნდება ShekvetaDataSet არავიზუალური კომპონენტი (ნახ. 17.15).


შემდეგ, ისევ მოვნიშნოთ bindingSource1 კომპონენტი. Properties ფანჯარაში გავხსნათ DataMember სია და ავირჩიოთ Personali ცხრილი. შედეგად, შეიქმნება PersonaliTableAdapter არავიზუალური კომპონენტი (ნახ. 17.16).


ამის შემდეგ, ფორმაზე მოვათავსოთ dataGridView1 კომპონენტი (ნახ. 17.17). მის მარჯვნივ გაიხსნება ფანჯარა. Choose Data Source სიიდან ავირჩიოთ bindingSource1. ცხრილის სტრიქონების სანახავად ჯერ დავაჭიროთ ამავე ფანჯრის Preview Data მიმართვას, შემდეგ კი გახსნილი ფანჯრის Preview კლავიშს. დავხუროთ ეს ფანჯარა და dataGridView1 კომპონენტი მოვათავსოთ ფორმის საჭირო ადგილზე და მივცეთ მას საჭირო ზომა. dataGridView1 კომპონენტისთვის bindingSource1 მონაცემების წყარო შეგვიძლია, აგრეთვე, ავირჩიოთ Properties ფანჯრის DataSource სიიდანაც.

შემდეგ, ფორმაზე მოვათავსოთ bindingNavigator1 კომპონენტი (ნახ. 17.18). ის ავტომატურად მოთავსდება ფორმის ზედა ნაწილში. გავხსნათ Properties ფანჯრის Dock თვისება და ავირჩიოთ None. შედეგად, შევძლებთ ნავიგატორის მოთავსებას ფორმის ნებისმიერ ადგილზე. ახლა ნავიგატორი დავუკავშიროთ Personali ცხრილს. ამისათვის, Properties ფანჯრის BindingSource სიიდან ავირჩიოთ bindingSource1.

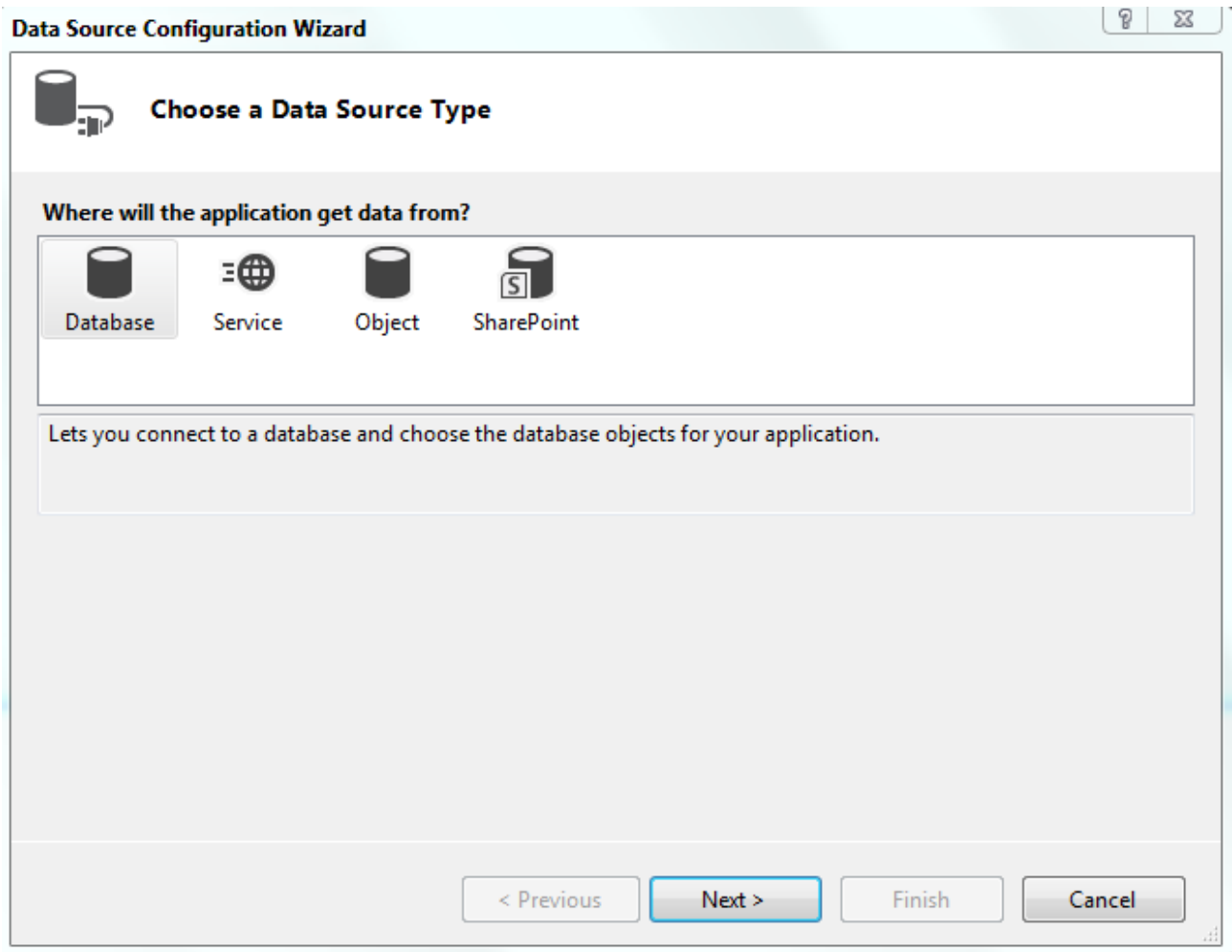
bindingNavigator1 კომპონენტის  კლავიშში გამოიყენება DataSet მონაცემთა ნაკრების Personali ცხრილში სტრიქონის ჩასამატებლად,  კლავიშში კი - სტრიქონის წასაშლელად. იმისათვის, რომ შევძლოთ შეცვლილი სტრიქონების სერვერზე გადაგზავნა და სერვერიდან სტრიქონების წაკითხვა, ამ კომპონენტს უნდა დავუმატოთ შესაბამისი კლავიშები  და . ეს შეგვიძლია ორი გზით გავაკეთოთ. პირველი გზა შემდეგში მდგომარეობს. მოვნიშნოთ bindingNavigator1 კომპონენტი და დავაჭიროთ მარჯვენა ზედა კუთხეში მოთავსებულ მცირე ზომის შავ ისარს. გახსნილ ფანჯარაში ვაჭერთ Insert Standard Items მიმართვაზე. შედეგად, კომპონენტს სტანდარტული კლავიშები დაემატება.

მეორე გზა შემდეგში მდგომარეობს. მოვნიშნოთ bindingNavigator1 კომპონენტი და Properties ფანჯარაში Items თვისების გასწვრივ დავაჭიროთ შესაბამის კლავიშს. გახსნილ ფანჯარაში (ნახ. 17.19) Select item ... სიიდან ვირჩევთ დასამატებელ ელემენტს და ვაჭერთ Add კლავიშს. შემდეგ შეგვიძლია დამატებული კლავიშის თვისებების შეცვლა ამავე ფანჯრის მარჯვნივ მოთავსებული თვისებების სიის საშუალებით. bindingNavigator1 კომპონენტის

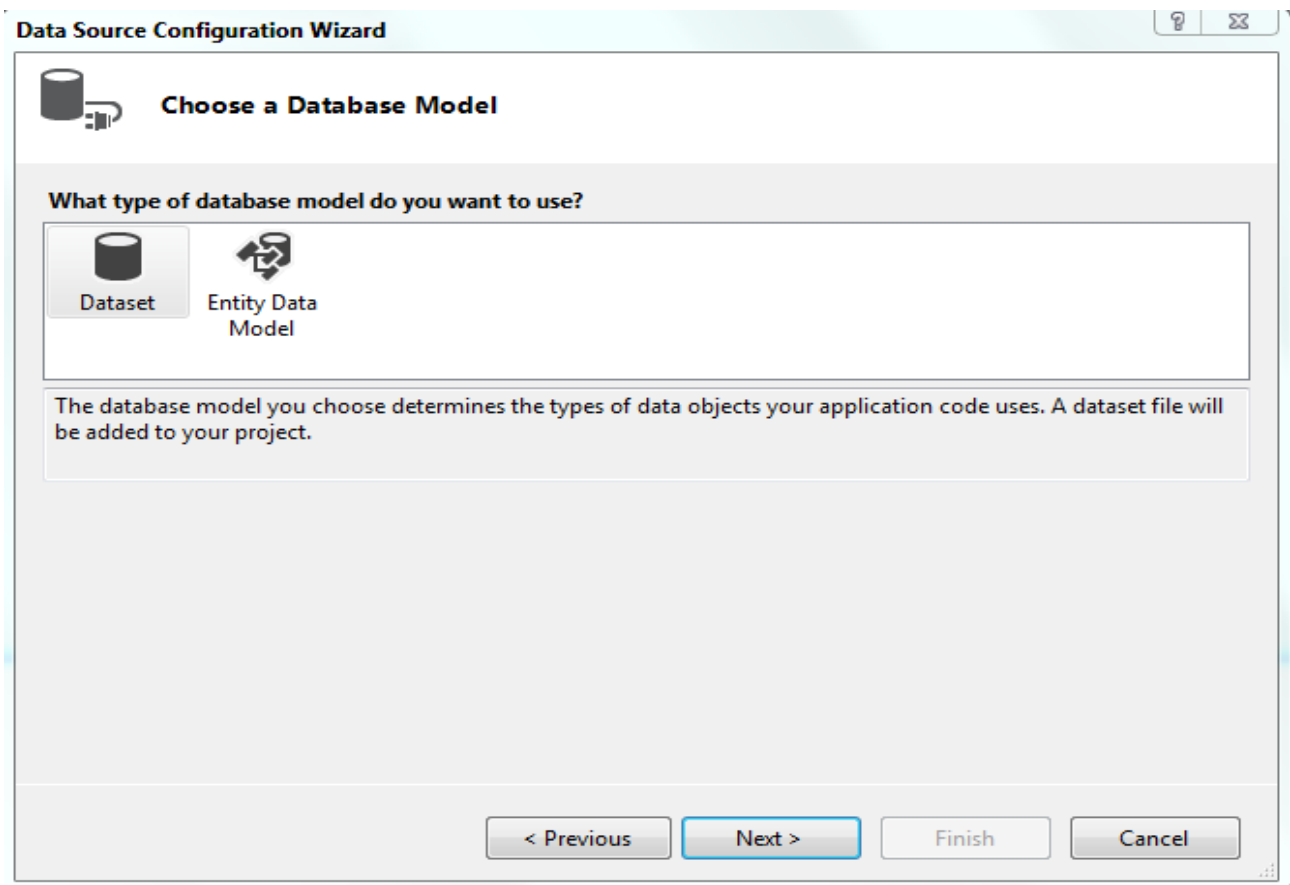
რომელიმე ელემენტის წასაშლელად, ჯერ ის უნდა მოვნიშნოთ, შემდეგ კი დავაჭიროთ  კლავიშს. ელემენტებს ადგილები შეგვიძლია შევუცვალოთ ზედა და ქვედა ისრების საშუალებით.

bindingNavigator1 კომპონენტზე საჭირო კლავიშების მოთავსების შემდეგ ორჯერ სწრაფად დავაწკაპუნოთ  კლავიშზე და შევიტანოთ კოდი, რომელიც სერვერზე შეინახავს Personal ცხრილში ჩვენს მიერ შესრულებულ ცვლილებებს (სტრიქონების დამატება, წაშლა და შეცვლა). ცვლილებები შეგვიძლია dataGridView1 კომპონენტში შევიტანოთ და შემდეგ დავაჭიროთ აღნიშნულ კლავიშს. კოდს აქვს სახე:

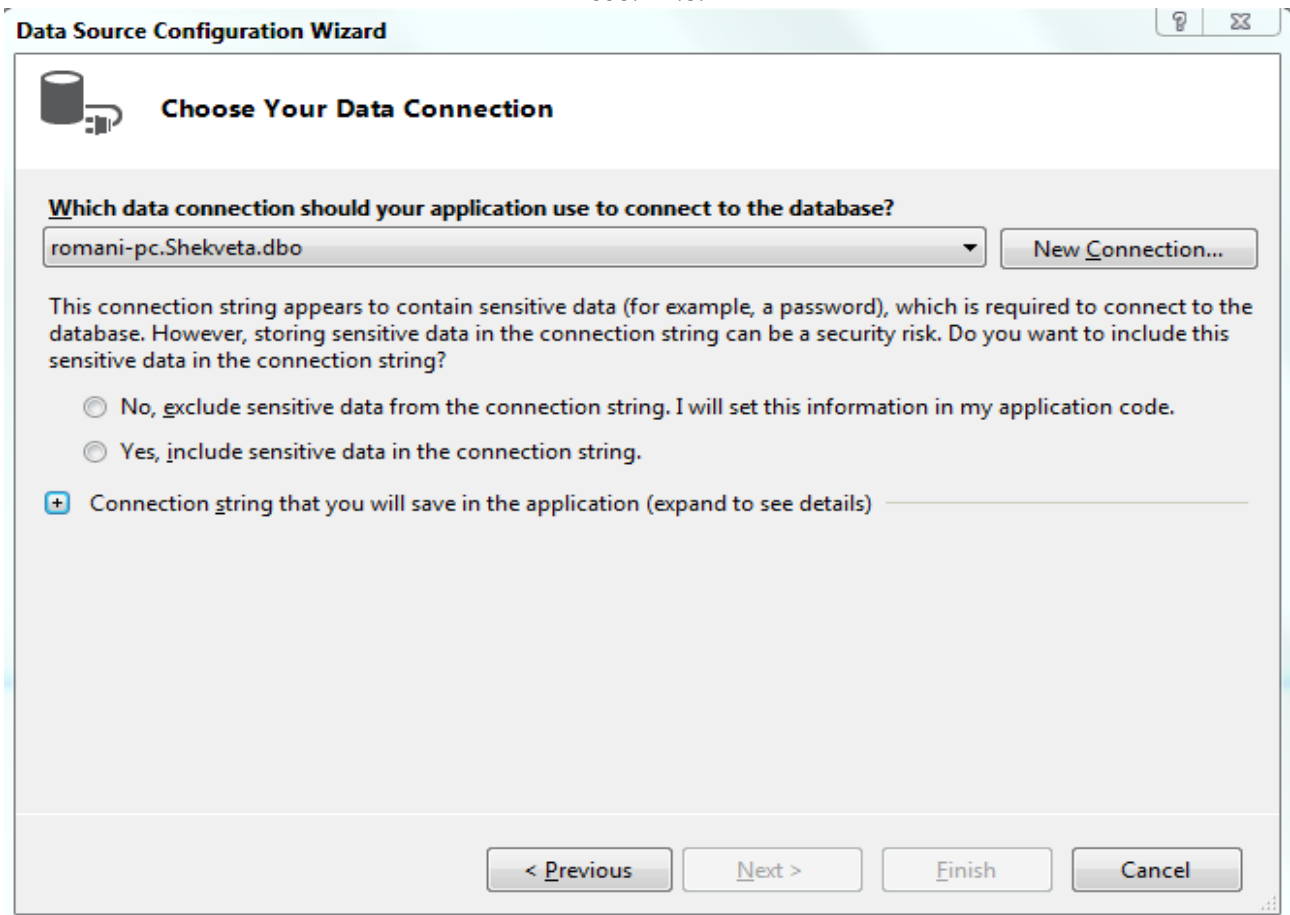
```
{  
this.Validate();  
this.bindingSource1.EndEdit();  
this.personaliTableAdapter.Update(this.ShekvetaDataSet.Personali);  
}
```



ნახ. 17.7.



бсб. 17.8.



бсб. 17.9

Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server (SqlClient) Change...

Server name:
ROMANI-PC Refresh

Log on to the server

Use Windows Authentication

Use SQL Server Authentication

User name: sa

Password: ●●●●●●

Save my password

Connect to a database

Select or enter a database name:
Shekveta

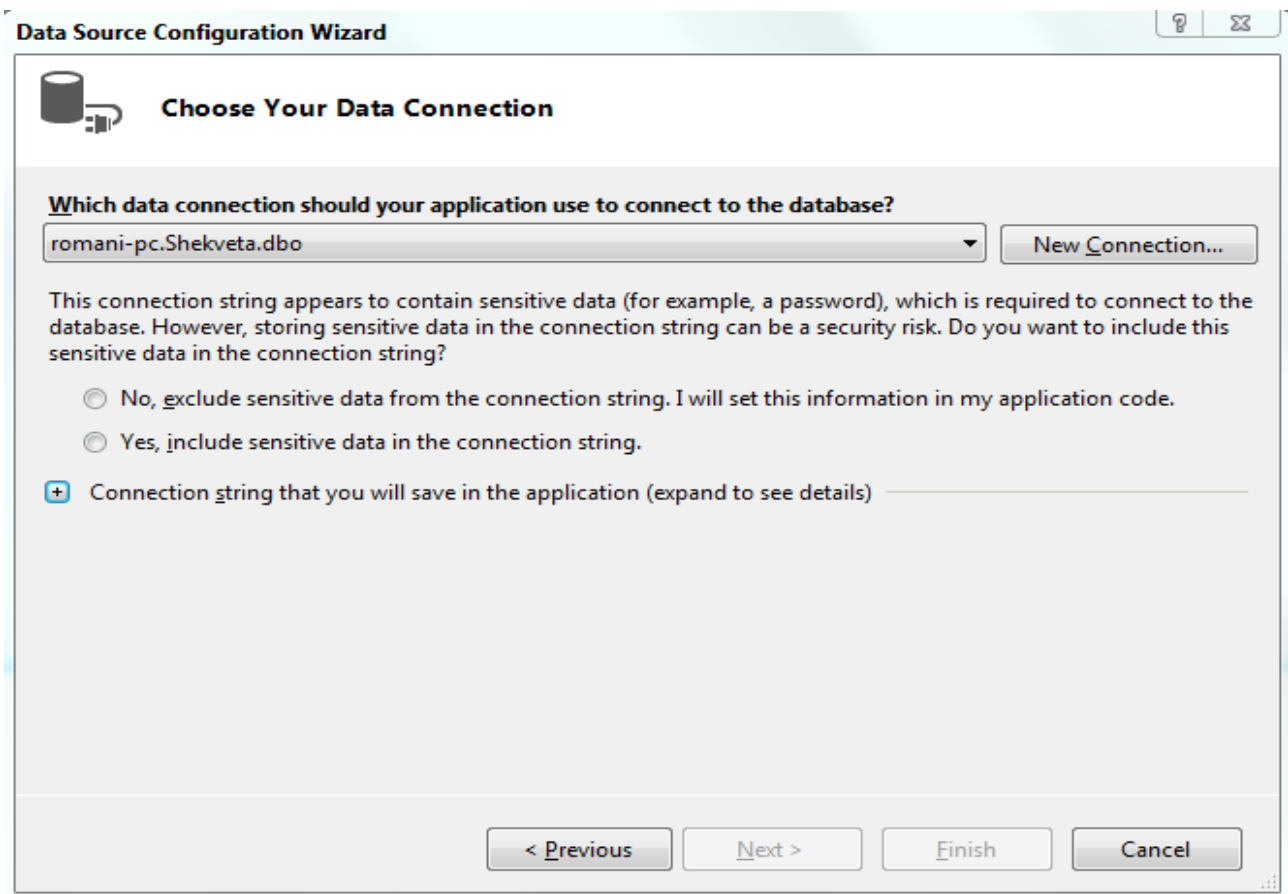
Attach a database file:
Browse...

Logical name:

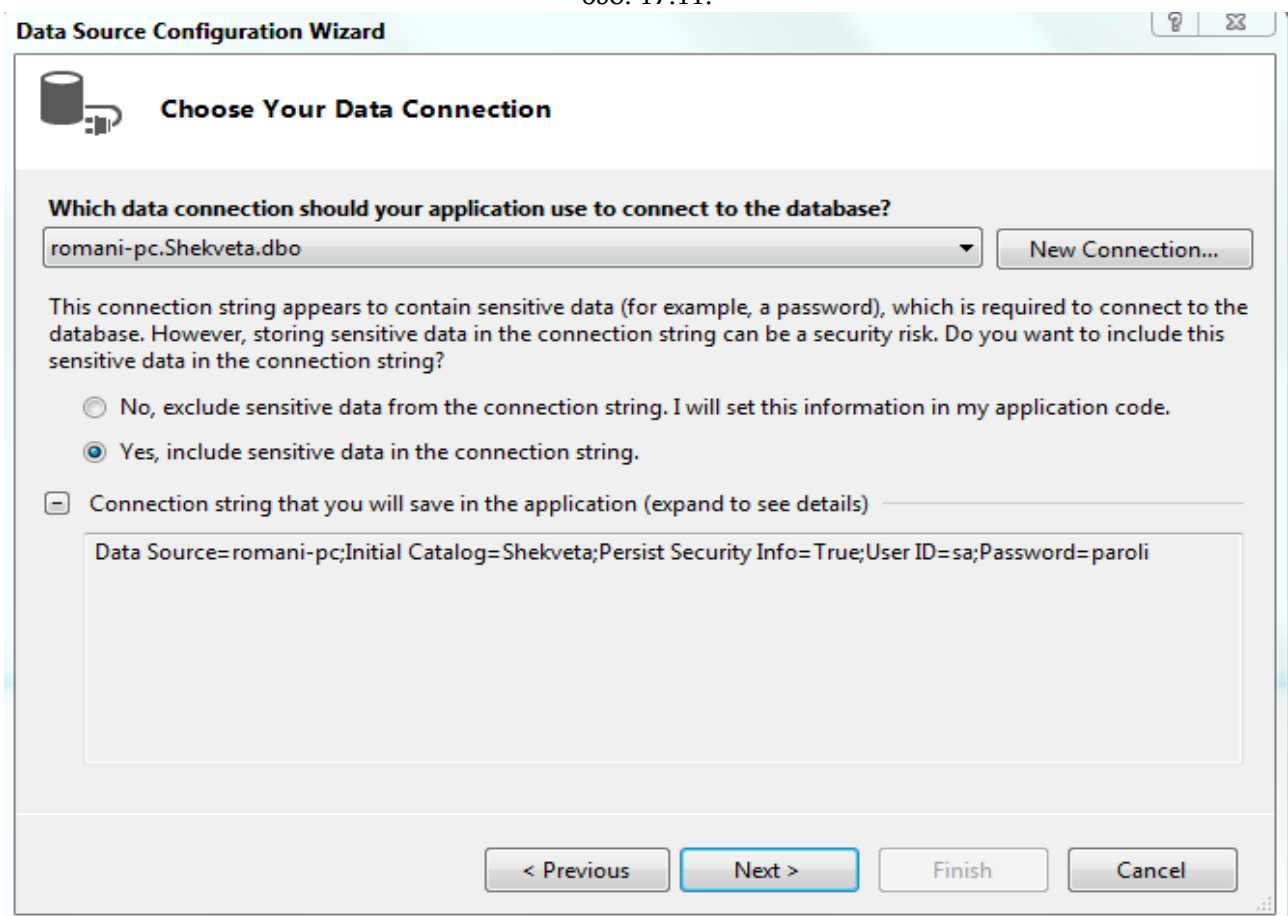
Advanced...

Test Connection OK Cancel

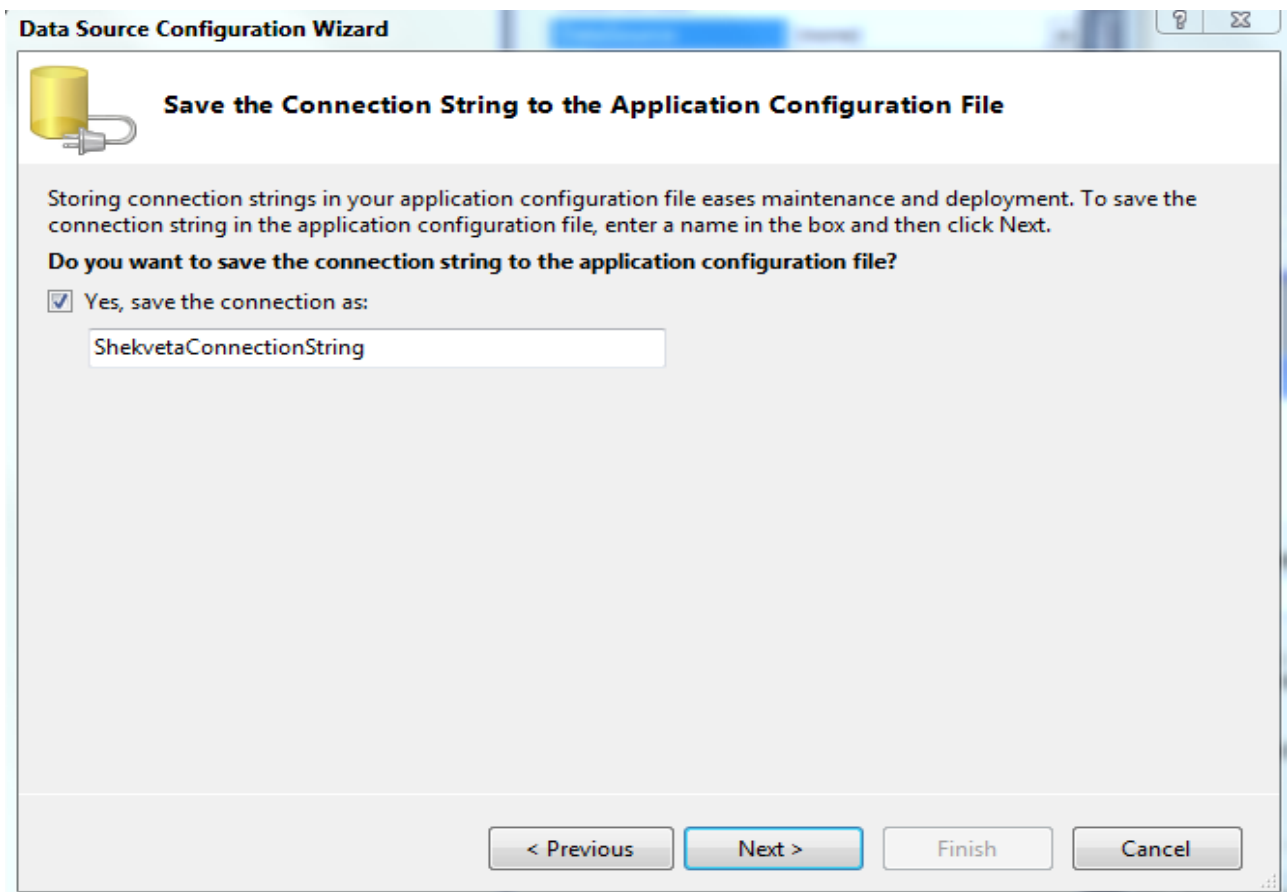
6sb. 17.10



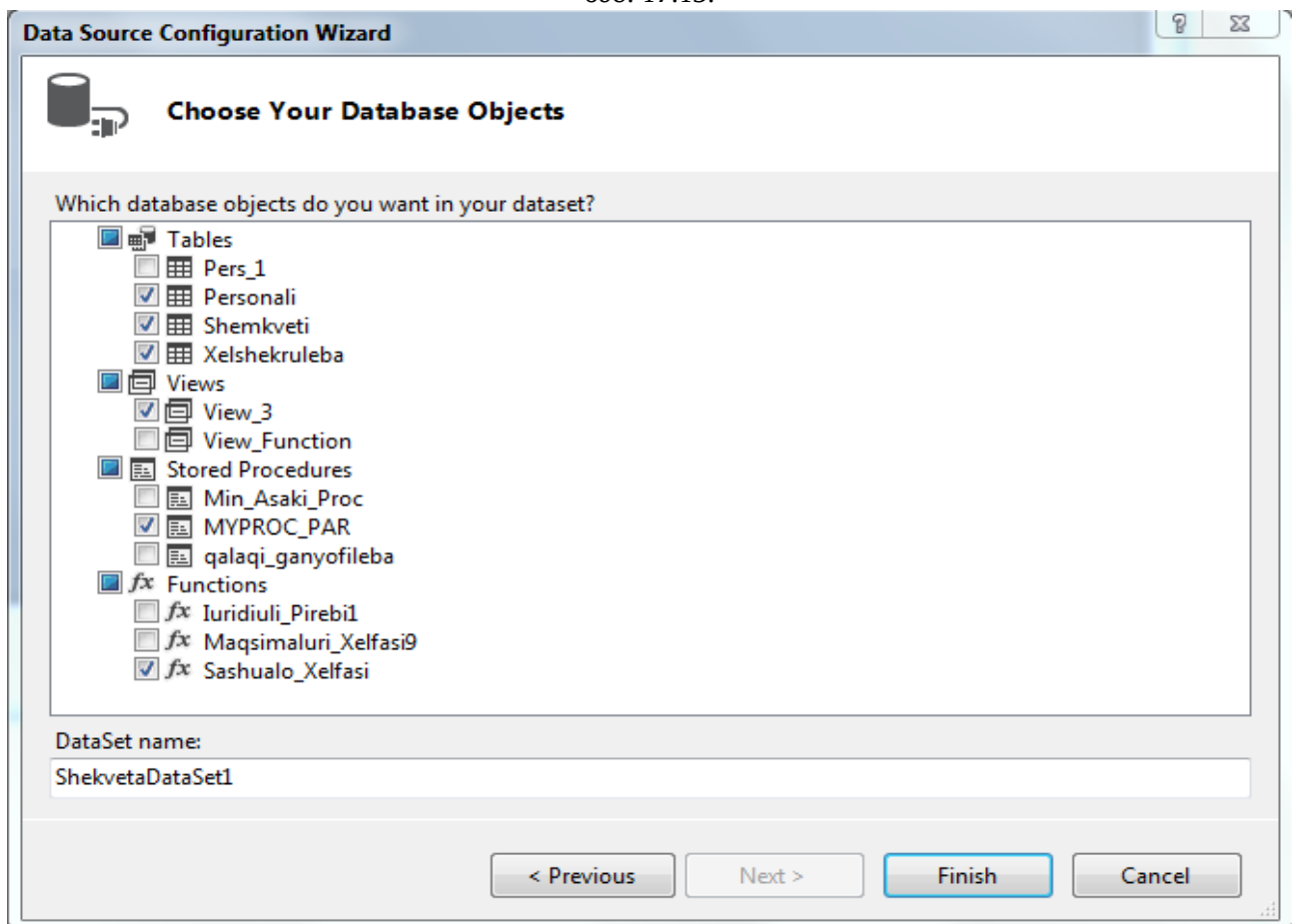
Біб. 17.11.



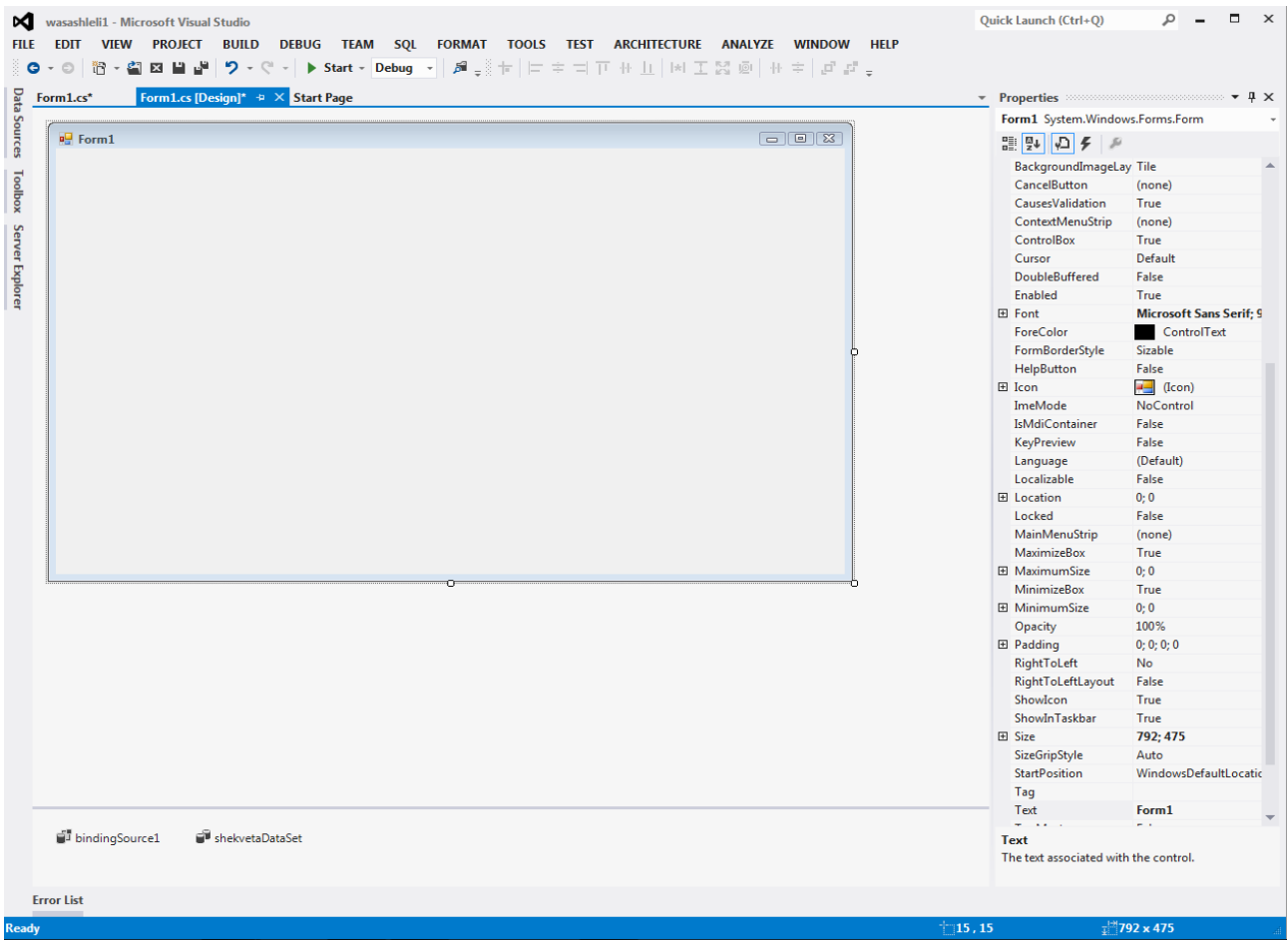
Біб. 17.12.



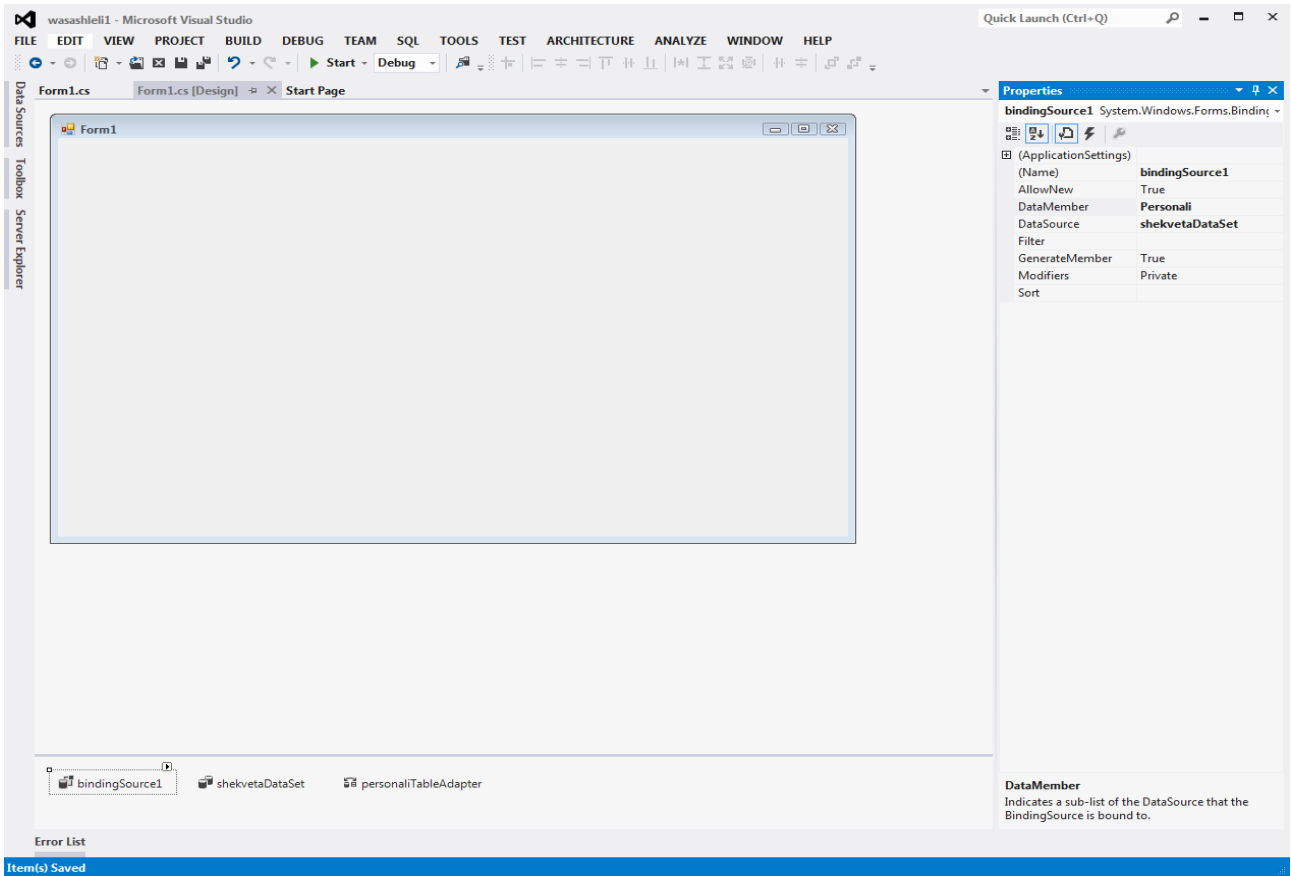
Бсб. 17.13.



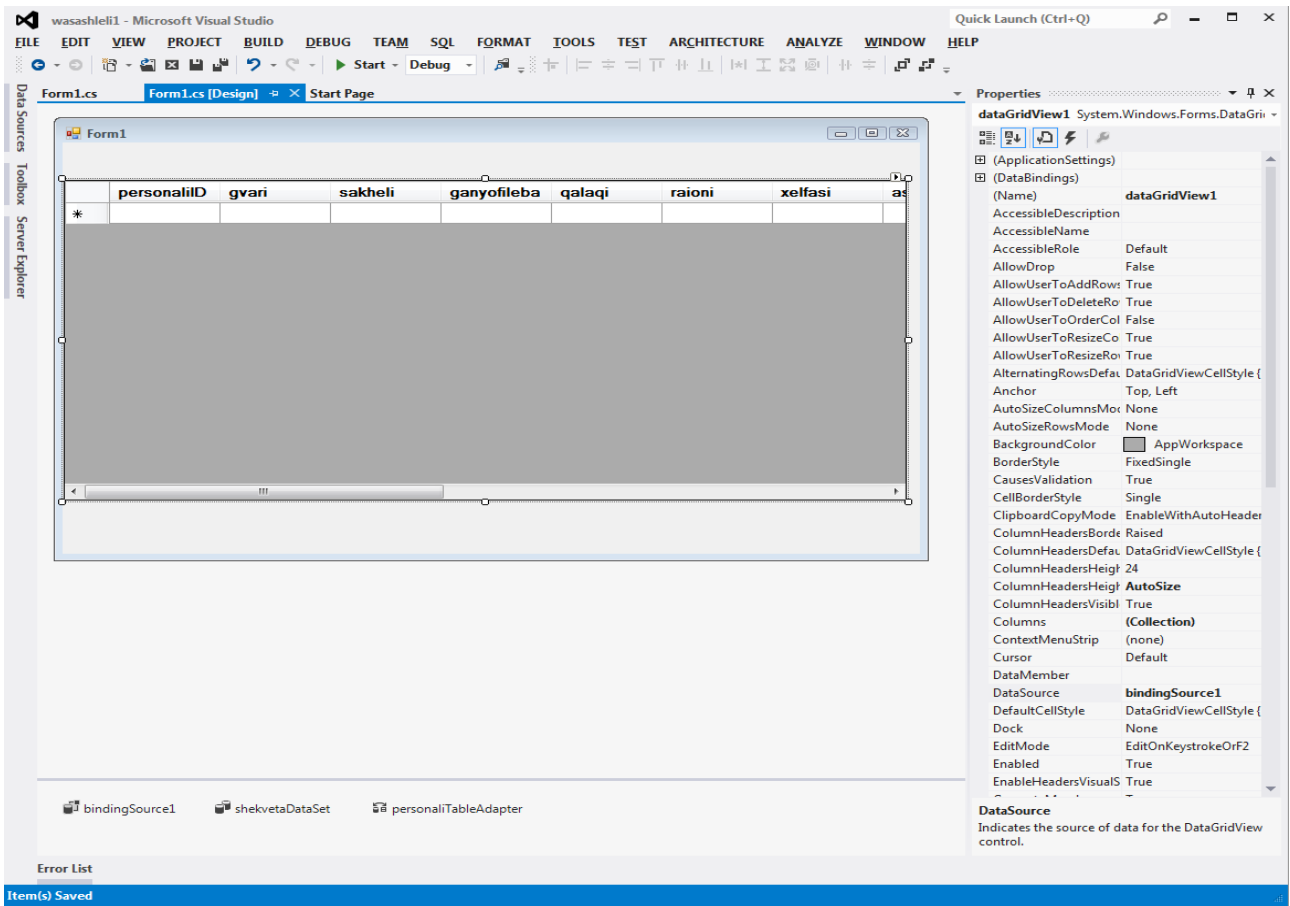
Бсб. 17.14.



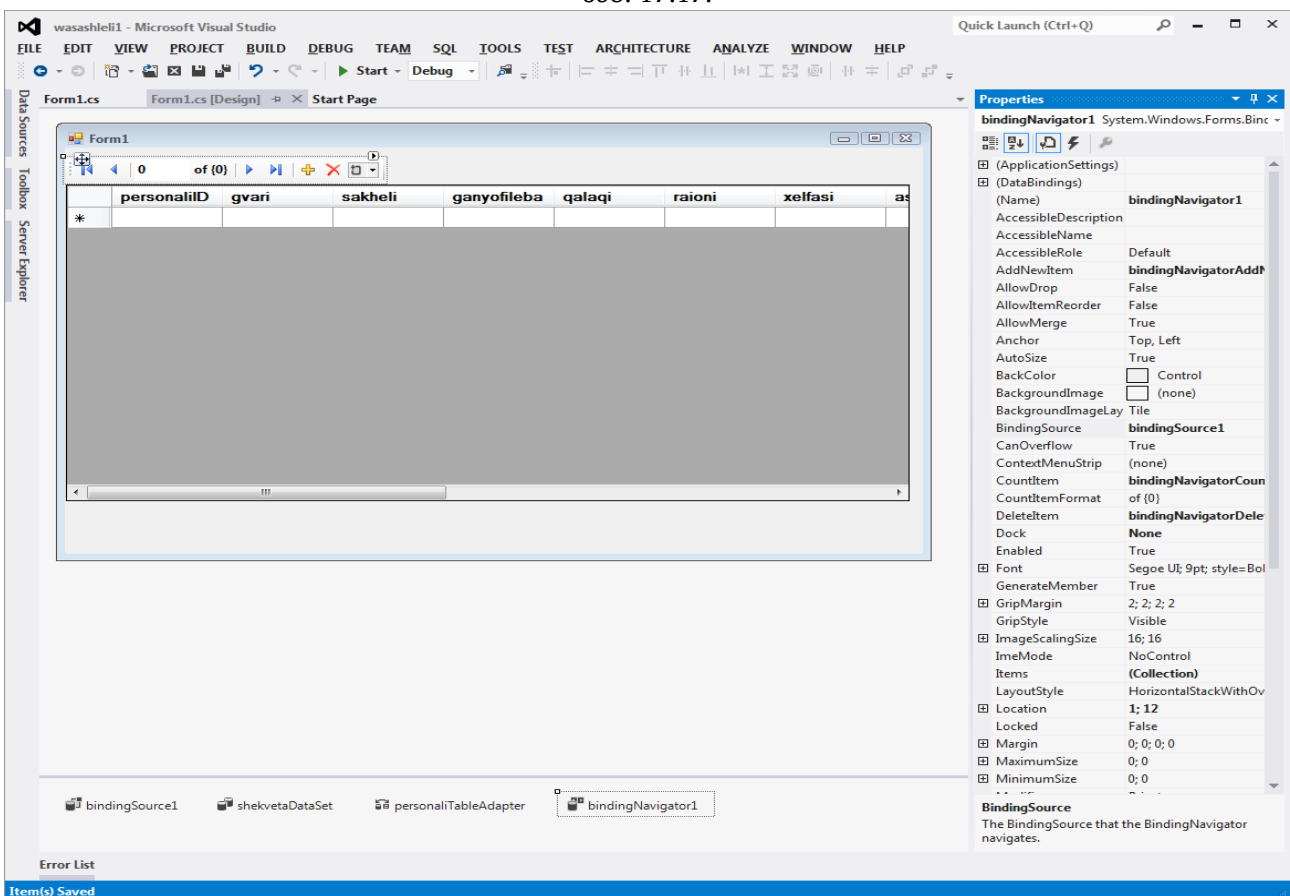
Біб. 17.15.



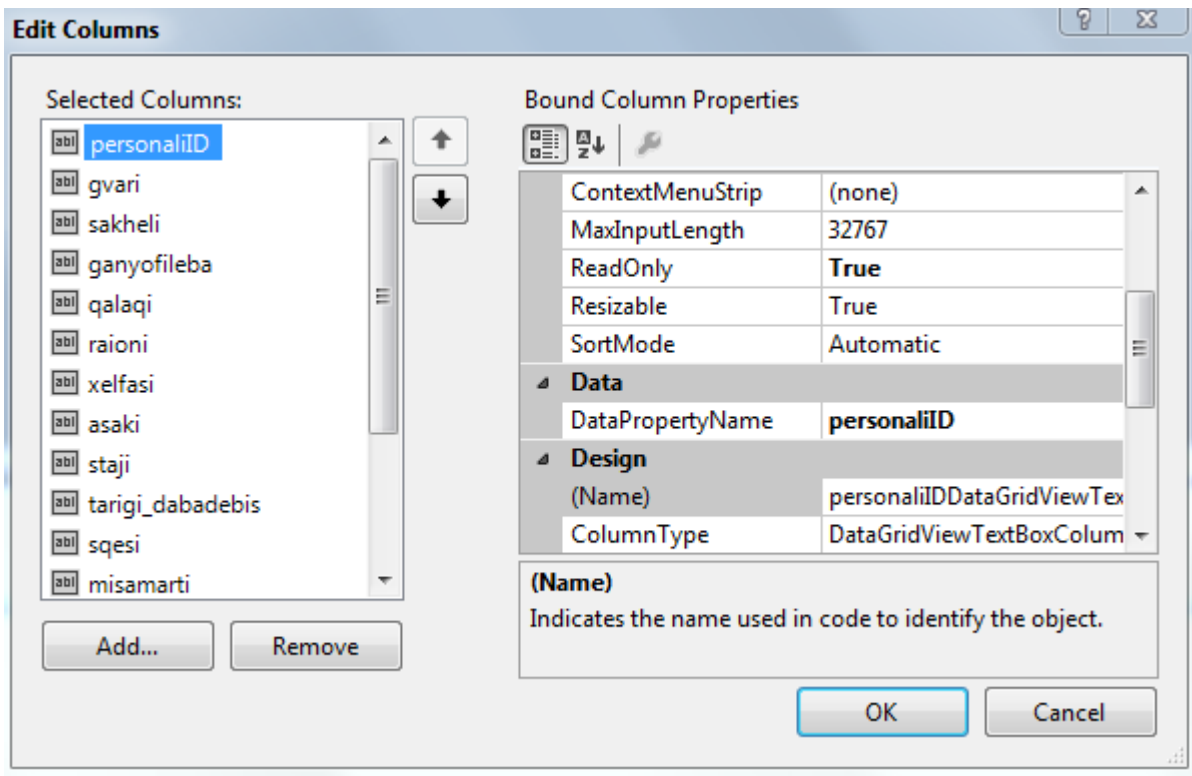
Біб. 17.16.



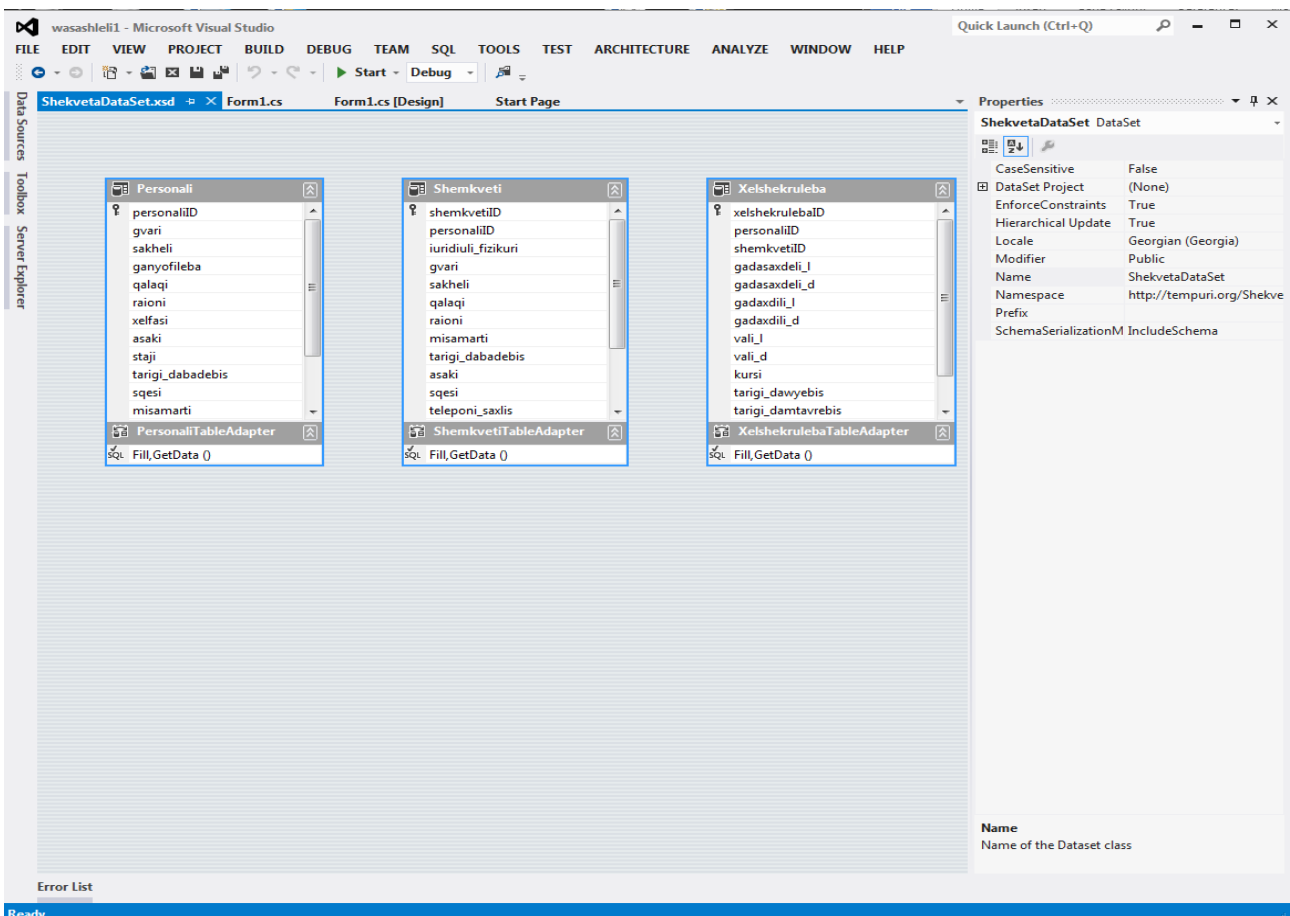
ბიბ. 17.17.



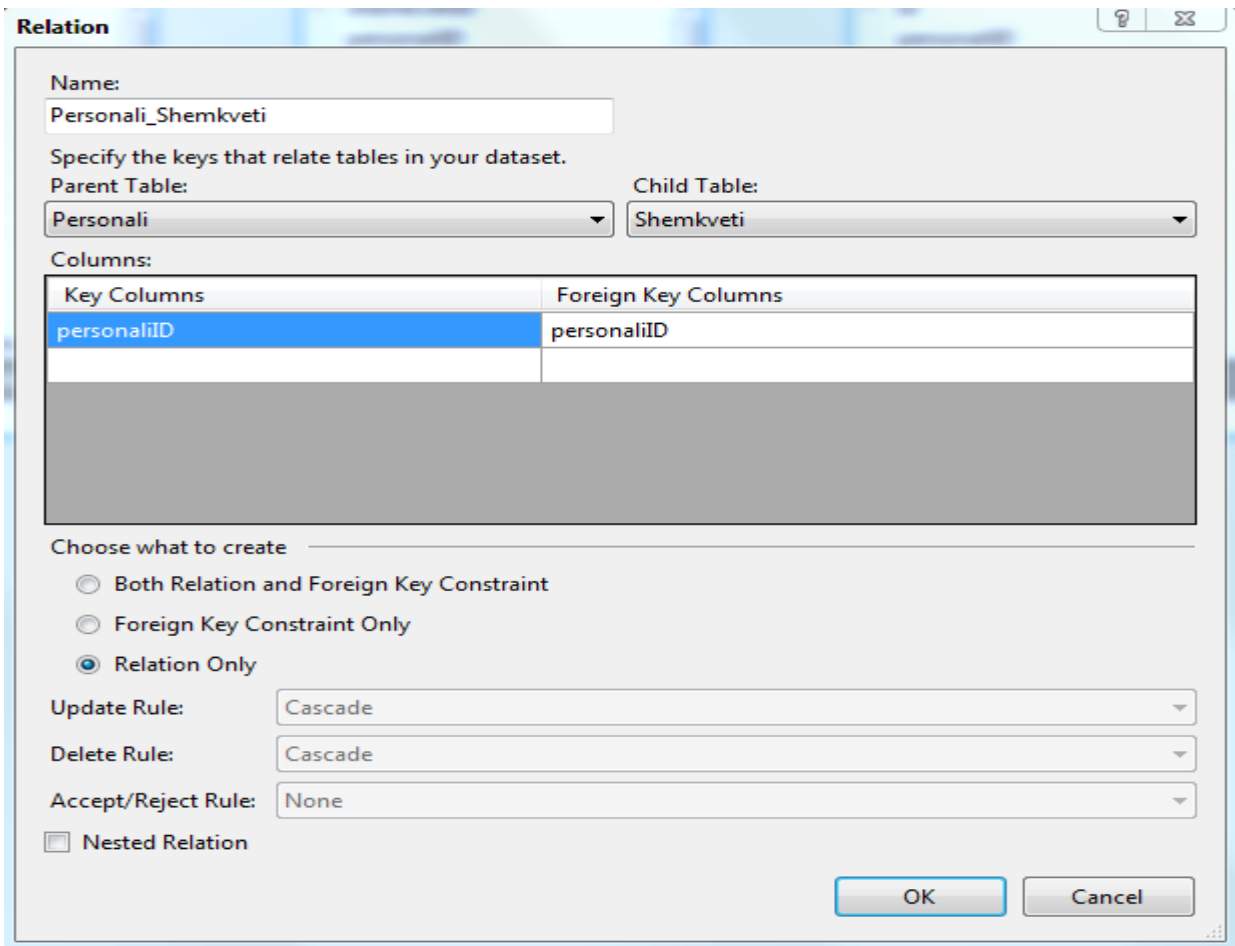
ბიბ. 17.18.



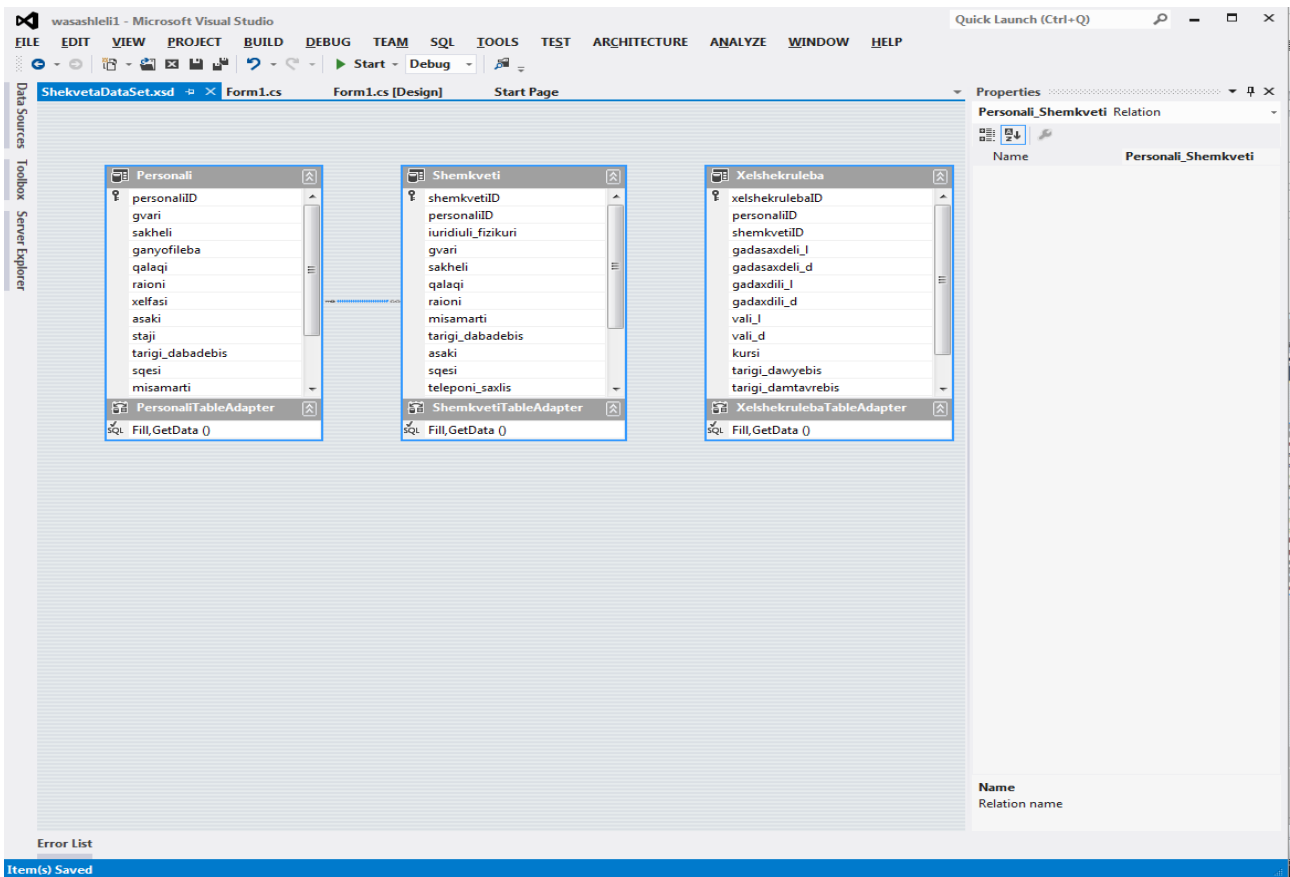
ბსბ. 17.19.



ბსბ. 17.20.



ფიგ. 17.21.



ფიგ. 17.22.

აქვე შევნიშნოთ, რომ Update() მეთოდი ითხოვს, რომ PersonalI ცხრილს ჰქონდეს პირველადი გასაღები. სერვერიდან ჩვენს ლოკალურ მონაცემთა ნაკრებში ცხრილის გადმოსაწერად და dataGridView1 კომპონენტში გამოსატანად ორჯერ სწრაფად დავაწკაპუნოთ





კლავიშზე და შევიტანოთ შემდეგი კოდი:


```
{  
this.personaliTableAdapter.Fill(this.shekvetaDataSet.Personali);  
}
```

ამ კლავიშს უნდა დავაჭიროთ ცვლილებების შენახვის შემდეგ, იმაში დასარწმუნებლად, რომ ცვლილებები მართლაც დაფიქსირდა სერვერზე.

ახლა მონაცემების შეტანის მიზნით ფორმაზე მოვათავსოთ textBox1 კომპონენტი და მივაბათ ის PersonalI ცხრილის gvari სვეტს. ამისათვის, Properties ფანჯარაში DataBindings თვისების მარცხნივ დავაჭიროთ + ნიშანს, გავხსნათ Text სია, დავაჭიროთ bindingSource1 სახელის მარცხნივ მოთავსებულ + ნიშანს და ავირჩიოთ gvari სვეტი. შევასრულოთ ჩვენი პროგრამა. ნავიგაციის დროს textBox1 კომპონენტში გამოჩნდება gvari სვეტის მნიშვნელობები. ამავე კომპონენტში შეგვიძლია შევიტანოთ, აგრეთვე, gvari სვეტის ახალი მნიშვნელობებიც.

მონაცემების შეტანა შემდეგნაირად ხორციელდება. ვაჭერთ bindingNavigator1 კომპონენტის  კლავიშს და textBox1 კომპონენტში შეგვაქვს ახალი მნიშვნელობა, მაგალითად "რომანი".

შეტანილი მნიშვნელობის შესანახად ვაჭერთ bindingNavigator1 კომპონენტის  კლავიშს. ანალოგიური გზით, შეგვიძლია შევიტანოთ ყველა სვეტის მნიშვნელობა. იმის შესამოწმებლად,

რომ მართლაც შესრულდა მონაცემების შენახვა სერვერზე უნდა დავაჭიროთ  კლავიშს. თუ სვეტი შეიცავს თარიღს, მაშინ მისი მნიშვნელობა შეგვიძლია შევიტანოთ dateTimePicker კომპონენტიდან.

ცხრილებს შორის ბმების შექმნა

ახლა შევექმნათ კავშირი (რელაცია, ბმა) PersonalI და Shemkveti ცხრილებს შორის. ამისათვის, მოვნიშნოთ shekvetaDataSet1 კომპონენტი, გავხსნათ კონტექსტური მენიუ და შევასრულოთ Edit in DataSet Designer ბრძანება. გაიხსნება ShekvetaDataSet.xsd განყოფილება (ნახ 17.20). ცარიელ ადგილზე მოვათავსოთ კურსორი, გავხსნათ კონტექსტური მენიუ, Add ქვემენიუ და შევასრულოთ Relation ბრძანება. გაიხსნება ფანჯარა (ნახ. 17.21), რომლის Name სვეტში ჩანს კავშირის სახელი - PersonalI_Shemkveti. სურვილის შემთხვევაში შეგვიძლია მისი შეცვლა. Parent Table სიიდან ვირჩევთ მთავარი (მშობელი) ცხრილის სახელს - PersonalI, ხოლო Child Table სიიდან კი - დამოკიდებული (შვილობილი) ცხრილის სახელს - Shemkveti. Key Columns სიიდან ვირჩევთ მთავარი ცხრილის პირველად გასაღებს. ესაა PersonalI ცხრილის PersonalIID სვეტი. Foreign Key Columns სიიდან ვირჩევთ დამოკიდებული ცხრილის გარე გასაღებს. ესაა Shemkveti ცხრილის PersonalIID სვეტი. ჩავრთოთ Relation Only გადამრთველი და დავაჭიროთ OK კლავიშს. ShekvetaDataSet.xsd განყოფილებაში გამოჩნდება ცხრილებს შორის კავშირი. (ნახ 17.22).

ამრიგად, ჩვენ სამუშაოდ მოვამზადეთ bindingSource1 კომპონენტი. ის დაუკავშირეთ კონკრეტულ მონაცემთა ბაზასა და კონკრეტულ ცხრილს, და შევექმნით კავშირი ცხრილებს შორის. ახლა ფორმაზე მოვათავსოთ bindingSource2 კომპონენტი. Properties ფანჯრის DataSource სიიდან ავირჩიოთ bindingSource1, ხოლო DataMember სიიდან კი - კავშირის სახელი - PersonalI_Shemkveti. შემდეგ, ფორმაზე მოვათავსოთ dataGridView2 კომპონენტი და მივაბათ ის bindingSource2 მონაცემთა წყაროს. თუ კომპონენტი ფორმაზე არ ეტევა, მაშინ ფორმის ზომები შესაბამისად გავზარდოთ. ეს შესაძლებელია ბუქსირის ოპერაციის გამოყენებით ან Properties ფანჯრის Size თვისებაში სიმაღლისა და სიგანის შეცვლის გზით. ამის შემდეგ, დავაჭიროთ F5 კლავიშს ჩვენი პროგრამის შესასრულებლად.

მონაცემების გაფილტვრა

მონაცემების გასაფილტრად შეგვიძლია dataView კომპონენტის გამოყენება. ამისათვის, ჯერ შევქმნათ ახალი პროექტი. შემდეგ ფორმაზე მოვათავსოთ bindingSource1 კომპონენტი და დავუკავშიროთ Shekveta მონაცემთა ბაზის Personalი ცხრილს. ფორმაზე მოვათავსოთ bindingNavigator1 კომპონენტი და დავუკავშიროთ bindingSource1 კომპონენტს. ამის შემდეგ, Toolbox ფანჯრის All Windows Forms განყოფილებიდან ავირჩიოთ dataView1 კომპონენტი და მოვათავსოთ ფორმაზე. თუ ეს კომპონენტი არ არის სიაში, მაშინ შევასრულოთ Tool მენიუს Choose Toolbox Items... ბრძანება, გახსნილი ფანჯრის .NET Framework Components ჩანართში მოვნიშნოთ DataView ელემენტი და დავაჭიროთ OK კლავიშს. ამის შემდეგ, ის გამოჩნდება All Windows Forms განყოფილებაში. მის Table თვისებაში ავირჩიოთ shekvetaDataSet განშტოების Personalი ცხრილი. შემდეგ, ფორმაზე მოვათავსოთ dataGridView1 კომპონენტი და მის DataSource თვისებაში ავირჩიოთ dataView1 კომპონენტი. ახლა შეგვიძლია შევასრულოთ მონაცემების გაფილტვრა Personalი ცხრილში. დავუშვათ, გვინტერესებს ინფორმაცია სამედიცინო განყოფილებაში მომუშავე იმ თანამშრომლების შესახებ, რომელთა ასაკი აღემატება 30 წელს. ფორმაზე მოვათავსოთ button კომპონენტი და მივაბათ შემდეგი კოდი:

```
{
dataView1.RowFilter = "ganyofileba = 'სამედიცინო' AND asaki > 30";
dataGridView1.DataSource = dataView1;
}
```

ფილტრის შესატანად შეგვიძლია textBox კომპონენტის გამოყენებაც:

```
{
dataView1.RowFilter = textBox1.Text;
dataGridView1.DataSource = dataView1;
}
```

ამ შემთხვევაში, textBox კომპონენტში ფილტრი შეგვაქვს ბრჭყალების გარეშე.

თუ გვინტერესებს ის თანამშრომლები, რომლებიც დაიბადნენ 1990 წლის 1 იანვრის შემდეგ, უნდა შევასრულოთ შემდეგი კოდი

```
{
dataView1.RowFilter = "tarigi_dabadebis > '01.01.1990'"
dataGridView1.DataSource = dataView1;
}
```

თარიღის შეტანა შეგვიძლია dateTimePicker1 კომპონენტიდანაც:

```
{
dataView1.RowFilter = "tarigi_dabadebis > " + dateTimePicker1.Value.ToString() + """;
dataGridView1.DataSource = dataView1;
}
```

ფილტრის გაუქმება შემდეგნაირად შეგვიძლია:

```
{
dataView1.RowFilter = "";
dataGridView1.DataSource = dataView1;
}
```

მონაცემების დახარისხება

dataView კომპონენტის გამოყენებით შეგვიძლია მონაცემების დახარისხებაც. მოყვანილი კოდით ხდება Personalი ცხრილის სტრიქონების დახარისხება asaki სვეტის მნიშვნელობების კლების მიხედვით:

```
{
dataView1.Sort = "asaki DESC";
dataGridView1.DataSource = dataView1;
}
```

ცხრილის დახარისხება შესაძლებელია რამდენიმე სვეტის მნიშვნელობების მიხედვითაც:

```
{
dataView1.Sort = "ganyofileba, asaki DESC";
dataGridView1.DataSource = dataView1;
}
```

დახარისხების გასაუქმებლად უნდა შევასრულოთ კოდი:

```
{
dataView1.Sort = "";
dataGridView1.DataSource = dataView1;
}
```

მონაცემების დახარისხებისათვის შეგვიძლია dataGridView1 კომპონენტის გამოყენებაც:

```
{
dataGridView1.Sort(dataGridView1.Columns[1],
System.ComponentModel.ListSortDirection.Descending);
}
```

მოყვანილ კოდში შესრულდება dataGridView1 კომპონენტის სტრიქონების დახარისხება კლებადობით იმ სვეტის მნიშვნელობების მიხედვით, რომლის ინდექსია 1. ასეთია gvari სვეტი.

ქვემოთ მოცემულ კოდში სრულდება dataGridView1 კომპონენტის სტრიქონების დახარისხება ზრდადობით იმ სვეტის მნიშვნელობების მიხედვით, რომლის ინდექსია 0. ასეთია PersonalIID სვეტი:

```
{
dataGridView1.Sort(dataGridView1.Columns[0],
System.ComponentModel.ListSortDirection.Ascending);
}
```

მონაცემების ძებნა

ცხრილში მონაცემების მოსაძებნად შეგვიძლია გამოვიყენოთ bindingSource1 კომპონენტის Find მეთოდი. მისი პირველი პარამეტრი არის იმ სვეტის სახელი, რომელშიც უნდა შესრულდეს ძებნა, მეორე პარამეტრი კი მიუთითებს სამეზნ სიდიდეს. გაიცემა მოძებნილი მნიშვნელობის შემცველი სტრიქონის ინდექსი. თუ მნიშვნელობა ვერ მოიძებნა, მაშინ გაიცემა -1. ნაპოვნი სტრიქონის სვეტების მნიშვნელობები ენიჭებათ ცვლადებს, რომლებიც შემდეგ შეგვიძლია სხვადასხვა მიზნებისთვის გამოვიყენოთ.

მოყვანილი კოდით gvari სვეტში სრულდება "სამხარაძე რომანი" მნიშვნელობის ძებნა. label1 კომპონენტში გამოგვაქვს ნაპოვნი სტრიქონის ინდექსი. იმისათვის, რომ dataGridView1 კომპონენტში მონიშნოს მოძებნილი სტრიქონი, მისი ინდექსის მნიშვნელობა bindingSource1 კომპონენტის Position თვისებას უნდა მივანიჭოთ. კოდს აქვს სახე:

```
{
int index = bindingSource1.Find("gvari", "სამხარაძე ანა");
if ( index != -1 )
{
string gvari = shekvetaDataSet.Personali.Rows[index]["gvari"].ToString();
string ganyofileba = shekvetaDataSet.Personali.Rows[index]["ganyofileba"].ToString();
string qalaqi = shekvetaDataSet.Personali.Rows[index]["qalaqi"].ToString();
}
```

```

double xelfasi = Convert.ToDouble(shekvetaDataSet.Personali.Rows[index]["xelfasi"]);
int asaki = Convert.ToInt32(shekvetaDataSet.Personali.Rows[index]["asaki"]);
int staji = Convert.ToInt32(shekvetaDataSet.Personali.Rows[index]["staji"]);
DateTime tarigi_dabadebis =
    Convert.ToDateTime(shekvetaDataSet.Personali.Rows[index]["tarigi_dabadebis"]);
//
bindingSource1.Position = index;
label1.Text = index.ToString();
}
else label1.Text = "მნიშვნელობა ვერ მოიძებნა";
}

```

უმჯობესი და მოქნილი იქნება, თუ Find მეთოდის პარამეტრებს textBox კომპონენტების საშუალებით შევიტანთ:

```
int index = bindingSource1.Find(textBox3.Text, textBox4.Text);
```

შედეგად, შევძლებთ სხვადასხვა სვეტში სხვადასხვა მნიშვნელობების ძებნას.

ნავიგაცია

ნავიგაციისთვის, ანუ ცხრილის სტრიქონების, უფრო სწორად, dataGridView კომპონენტის სტრიქონების გასწვრივ წინ და უკან გადაადგილებისათვის bindingNavigator კომპონენტის გარდა შეგვიძლია bindingSource1 კომპონენტის გამოყენებაც. კერძოდ, მომდევნო სტრიქონზე გადასასვლელად bindingSource1.MoveNext() მეთოდი გამოიყენება, წინა სტრიქონზე გადასასვლელად - bindingSource1.MovePrevious() მეთოდი, უკანასკნელ სტრიქონზე გადასასვლელად - bindingSource1.MoveLast() მეთოდი, ხოლო პირველ სტრიქონზე გადასასვლელად კი - bindingSource1.MoveFirst() მეთოდი. შეგვიძლია ეს მეთოდები button კომპონენტებს მივაბათ და შევასრულოთ ნავიგაცია ცხრილის სტრიქონების გასწვრივ. თუმცა ამ მიზნით უმჯობესია bindingNavigator კომპონენტის გამოყენება.

თუ მოვნიშნავთ dataGridView1 კომპონენტის რომელიმე უჯრას, მაშინ მისი შემცველი სტრიქონის ინდექსი ავტომატურად მიენიჭება RowIndex თვისებას. ინდექსის საშუალებით შეგვიძლია მივმართოთ არჩეული სტრიქონის ნებისმიერ სვეტს. მოყვანილი პროგრამით ხდება ამის დემონსტრირება:

```

{
int CurrentRowIndex = dataGridView1.CurrentRow.Index;
int staji = Convert.ToInt32(shekvetaDataSet.Personali.Rows[CurrentRowIndex]["staji"]);
string qalaqi = dataGridView1.CurrentRow.Cells[3].Value.ToString();
label1.Text = qalaqi + " " + staji.ToString();
}

```

გამოთვლები

გამოთვლების შესასრულებლად შეგვიძლია გამოვიყენოთ DataSet კომპონენტის Compute მეთოდი. მისი პირველი პარამეტრია შესასრულებელი ფუნქცია, მეორე კი - ფილტრი. ის საშუალებას გვაძლევს მითითებული სვეტის მნიშვნელობებზე შემდეგი ფუნქციები შევასრულოთ: SUM, AVG, COUNT, MAX, MIN. მოყვანილ კოდში სრულდება Personali ცხრილის xelfasi სვეტის მნიშვნელობების შეკრება:

```

{
object sumObject = shekvetaDataSet.Personali.Compute("SUM(xelfasi)", "");
label1.Text = sumObject.ToString();
}

```

Compute მეთოდის პირველი პარამეტრი შეიცავს შესასრულებელ ფუნქციას, მეორე კი ფილტრს. მოყვანილ პროგრამაში გამოითვლება asaki სვეტის საშუალო არითმეტიკული იმ სტრიქონებისთვის, რომლებშიც xelfasi სვეტის მნიშვნელობა 700-ს აღემატება.

```
{
object avgObject = shekvetaDataSet.Personali.Compute("AVG(asaki)", "xelfasi > 700");
label1.Text = avgObject.ToString();
}
```

ანალოგიურად, შეგვიძლია დანარჩენი ფუნქციების მნიშვნელობების გამოთვლა:

```
{
object countObject = shekvetaDataSet.Personali.Compute("COUNT(asaki)", "");
object maxObject = shekvetaDataSet.Personali Compute("MAX(asaki)", "");
object minObject = shekvetaDataSet.Personali.Compute("MIN(asaki)", "");
label1.Text = countObject.ToString();
label2.Text = maxObject.ToString();
label3.Text = minObject.ToString();
}
```

გამოთვლების შესრულება შეიძლება, აგრეთვე DataTable ობიექტის გამოყენების გზით:

```
{
DataTable cxrili_obj = shekvetaDataSet.Tables["Personali"];
object sumObject = cxrili_obj.Compute("SUM(xelfasi)", "");
label1.Text = sumObject.ToString();
}
```

გამოთვლების შესრულება შეიძლება, აგრეთვე dataGridViewView ობიექტის გამოყენების გზით. მოყვანილ პროგრამაში სრულდება არჩეული სტრიქონის xelfasi სვეტის მნიშვნელობის გაორმაგება:

```
{
int CurrentRowIndex = dataGridView1.CurrentCell.RowIndex;
double xelfasi = Convert.ToDouble(shekvetaDataSet.Personali.Rows[CurrentRowIndex]["xelfasi"]);
double gaormagebuli_xelfasi = xelfasi * 2;
label1.Text = gaormagebuli_xelfasi.ToString();
//
double xelfasi2 = Convert.ToInt32(dataGridView1.CurrentRow.Cells[5].Value);
label2.Text = gaormagebuli_xelfasi.ToString() + " " + xelfasi2.ToString();
}
```

DataSet ობიექტის Rows თვისების გამოყენებით შეგვიძლია სათითაოდ მივმართოთ ცხრილის თითოეულ სტრიქონს და მივიღოთ ან შევცვალოთ რომელიმე სვეტის მნიშვნელობა. მოყვანილ პროგრამაში ხდება masivi მასივის შევსება Personali ცხრილის იმ სვეტის მონაცემებით, რომლის ინდექსია 5 (xelfasi სვეტი):

```
{
label1.Text = "";
double[] masivi = new double[shekvetaDataSet.Personali.Rows.Count];
for (int striqonis_indexi = 0; striqonis_indexi < shekvetaDataSet.Personali.Rows.Count;
    striqonis_indexi++)
{
masivi[striqonis_indexi] =
    Convert.ToDouble(shekvetaDataSet.Personali.Rows[striqonis_indexi][5].ToString()) + 10;
}
```



```

for (int ind = 0; ind < masivi.Length; ind++)
    label1.Text += Math.Round(masivi[ind],2).ToString() + '\n';
}

```

სვეტის ინდექსის ნაცვლად შეგვიძლია მისი სახელის მითითებაც. მაგალითად, 10-ით გავზარდოთ xelfasi სვეტის მნიშვნელობები:

```

{
for (int striqonis_indexi = 0; striqonis_indexi < shekvetaDataSet.Personali.Rows.Count;
striqonis_indexi++)
{
shekvetaDataSet.Personali.Rows[striqonis_indexi]["xelfasi"] =
    Convert.ToDouble(shekvetaDataSet.Personali.Rows[striqonis_indexi]["xelfasi"]) + 10;
}
// სერვერზე ცვლილებების შენახვა
Validate();
bindingSource1.EndEdit();
personaliTableAdapter.Update(shekvetaDataSet.Personali);
}

```

ცხრილებს შორის არსებული ზმებით მანიპულირება

შევქმნათ ახალი პროექტი და ზემოთ აღწერილი გზით შევქმნათ ბმა Personali და Shemkveti ცხრილებს შორის. მთავარი ცხრილია Personali, დამოკიდებული კი - Shemkveti. ბმის სახელია Personali_Shemkveti. შემდეგ, შევქმნათ ბმა ამავე ცხრილებს შორის, ოღონდ მთავარი ცხრილი იქნება Shemkveti, დამოკიდებული კი - Personali. ბმის სახელია Shemkveti_Personali. ფორმაზე მოვათავსოთ DataGridView1 კომპონენტი. მისი DataSource თვისება დავუკავშიროთ bindingSource1 წყაროს. შემდეგ, ფორმაზე მოვათავსოთ DataGridView2 კომპონენტი. მისი DataSource თვისება დავუკავშიროთ bindingSource2 წყაროს. ფორმაზე მოვათავსოთ button კომპონენტი და მივაბათ მოყვანილი კოდი, რომელიც Personali და Shemkveti ცხრილებს შორის ბმას ამყარებს. მთავარია Personali ცხრილი, დამოკიდებული კი - Shemkveti:

```

{
bindingSource2.DataSource = bindingSource1;
bindingSource2.DataMember = "Personali_Shemkveti";
}

```

ამ ცხრილებს შორის ბმის გასაუქმებლად უნდა შევასრულოთ შემდეგი კოდი:

```

{
bindingSource1.DataSource = null;
bindingSource1.DataSource = shekvetaDataSet;
bindingSource1.DataMember = shekvetaDataSet.Personali.ToString();
bindingSource2.DataSource = shekvetaDataSet;
bindingSource2.DataMember = shekvetaDataSet.Shemkveti.ToString();
}

```

ქვემოთ მოყვანილი კოდი ბმას ამყარებს ამავე ცხრილებს შორის, ოღონდ ახლა მთავარია Shemkveti ცხრილი, დამოკიდებული კი Personali:

```

{
bindingSource1.DataSource = null;
bindingSource2.DataSource = null;
bindingSource2.DataSource = shekvetaDataSet;
bindingSource2.DataMember = "Shemkveti";
}

```

```
bindingSource1.DataSource = bindingSource2;
bindingSource1.DataMember = "Shemkveti_Personali";
}
```

ADO.NET კლასების მომიხილვა

ADO.NET არის კლასების სიმრავლე, რომელიც გამოიყენება C# და .NET Framework გარემოსთან ერთად სხვადასხვა ფორმატის მონაცემთა ბაზასთან სამუშაოდ. ADO.NET ტექნოლოგია ინტეგრირებულია .NET Framework გარემოსთან და ორიენტირებულია .NET-ის ნებისმიერ ენასთან, განსაკუთრებით კი - C#-თან სამუშაოდ. ის მოიცავს System.Data სახელების სივრცეს და მასში შემავალ ჩადგმულ სახელების სივრცეებს: System.Data.SqlClient, System.Data.Linq და ა.შ.

ADO.NET უზრუნველყოფს მონაცემთა რელაციურ ბაზებთან, როგორც Microsoft SQL Server და Microsoft Access, მარტივ მიმართვას. მისი კლასების საშუალებით შეგვიძლია განვსაზღვროთ ცხრილები, სვეტები და სტრიქონები. მასში განსაზღვრულია, აგრეთვე DataSet კლასი, რომელიც წარმოადგენს რელაციური ცხრილების ერთობლიობას, მათ შორის არსებულ კავშირებთან ერთად.

სხვადასხვა ფორმატის მონაცემთა ბაზებთან მუშაობის გასამარტივებლად 90-იანი წლების დასაწყისში Microsoft-მა შეიმუშავა ODBC (Open Database Connectivity - მონაცემთა ბაზებთან ურთიერთქმედების ღია ინტერფეისი) ინტერფეისი. მას აქვს ფუნქციების სრული ნაკრები, რომლების გამოყენებაც შეიძლება სხვადასხვა ფორმატის მონაცემთა ბაზის მიმართ. ODBC ტექნოლოგია კარგად მუშაობს ტრადიციულ მონაცემთა ბაზებთან, მაგრამ ის არ იძლევა ისეთ მონაცემთან მიმართვის შესაძლებლობას, რომლებიც არ არის წარმოდგენილი სტრიქონებისა და სვეტების სახით ან არ აქვს რეგულარული სტრუქტურა.

ამ პრობლემის გადასაჭრელად შეიმუშავეს OLE DB ტექნოლოგია, რომელიც მუშაობს ODBC-ის ანალოგიურად. ის წარმოადგენს შუალედურ რგოლს კლიენტის პროგრამასა და მონაცემთა ბაზას შორის და კლიენტის პროგრამას მონაცემებს წარუდგენს ცხრილური სახით. ADO.NET უზრუნველყოფს როგორც OLE DB, ისე ODBC ტექნოლოგიას. შემდეგ, შემუშავებულ იქნა ActiveX Data Objects (ADO) ტექნოლოგია. ADO პროგრამული სისტემაა, რომელიც მაღალი დონის ენებზე დაწერილ პროგრამებს საშუალებას აძლევს მიმართონ OLE DB მონაცემებს.

ADO.NET კლასები და ობიექტები

ADO.NET საბაზო კლასები შეგვიძლია ორ ნაწილად დავყოთ: კლასები, რომლებიც აღწერენ **ობიექტ-მომხმარებლებს** (ესაა ობიექტები, რომლებიც მონაცემებს იღებენ) და კლასები, რომლებიც აღწერენ **ობიექტ-მიმწოდებლებს** (ესაა ობიექტები, რომლებიც გასცემენ მონაცემებს) (ნახ. 17. 23). ობიექტი-მიმწოდებლები ახდენენ მონაცემების კითხვას მონაცემთა ბაზებიდან და მათ მიწოდებას ობიექტი-მომხმარებლებისათვის. ამიტომ, ობიექტი-მიმწოდებლები ითხოვენ სერვერთან აქტიური შეერთების არსებობას. ობიექტ-მომხმარებლებს კი მონაცემების მიღების შემდეგ ამ მონაცემებთან მუშაობა შეუძლიათ ავტონომურ (ოფლაინ) რეჟიმში, ანუ რეჟიმში, როცა მონაცემთა ბაზასთან შეერთება დახურულია.

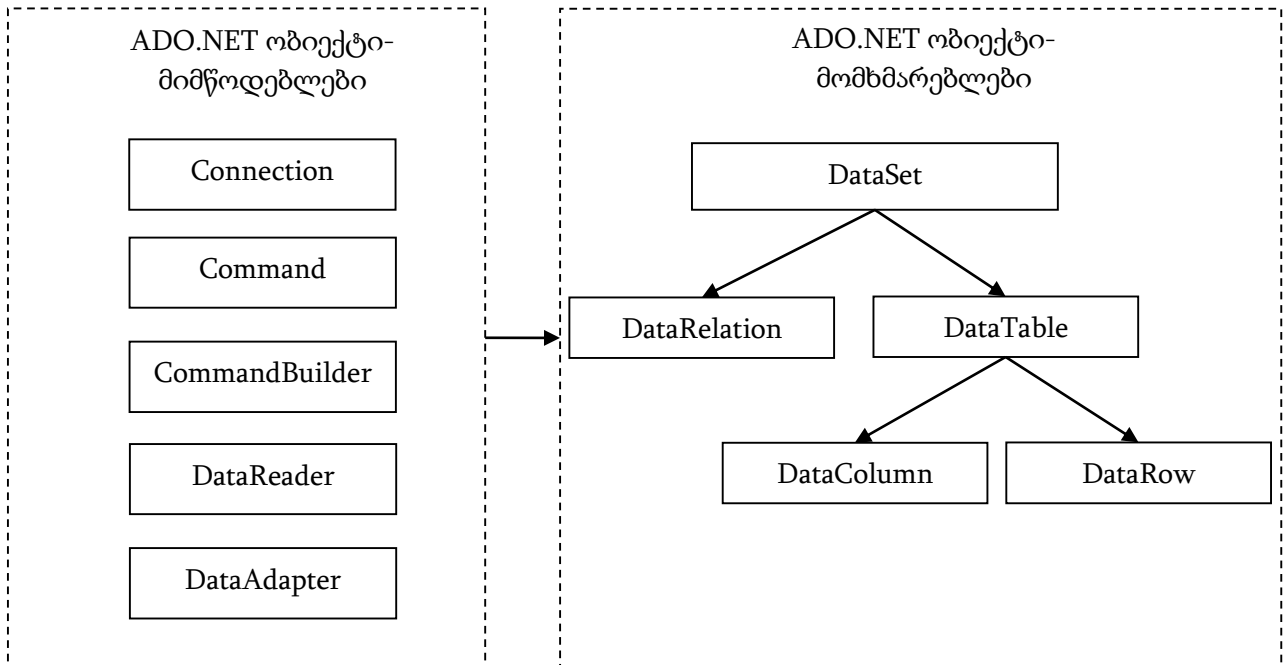
ობიექტი-მომხმარებლები და შესაბამისი კლასები

მოკლედ აღვწეროთ ობიექტ-მომხმარებლებთან სამუშაო კლასები.

DataSet კლასი. ამ კლასის ობიექტი გამოიყენება მონაცემთა ბაზაში მოთავსებული მონაცემების ლოკალური ასლების შესანახად კლიენტის კომპიუტერზე. ეს ობიექტებია:

ცხრილები, ცხრილებს შორის კავშირები (ბმები), წარმოდგენები, შენახული პროცედურები და ფუნქციები. მათთან მუშაობა სრულდება ავტონომურ რეჟიმში. DataSet ობიექტები შეიძლება, აგრეთვე, გამოვიყენოთ მონაცემების წარმოსადგენად XML ფაილებიდან.

DataTable კლასი. ამ კლასის ობიექტი გამოიყენება ერთი ცხრილის შესანახად, მაგალითად Personal-ის ცხრილის. ერთ DataSet ობიექტში შეიძლება რამდენიმე DataTable ობიექტის ანუ რამდენიმე ცხრილის მოთავსება.



ნახ. 17.23. ADO.NET საბაზო კლასები

DataRow კლასი. ამ კლასის ობიექტი გამოიყენება ცხრილის ერთი სტრიქონის შესანახად. ერთ DataTable ობიექტში შეიძლება რამდენიმე DataRow ობიექტის ანუ რამდენიმე სტრიქონის მოთავსება.

DataColumn კლასი. ამ კლასის ობიექტი გამოიყენება ერთი სვეტის შესანახად. ერთ DataRow ობიექტში შეიძლება რამდენიმე DataColumn ობიექტის ანუ რამდენიმე სვეტის მოთავსება.

DataRelation კლასი. ამ კლასის ობიექტი გამოიყენება ორ DataTable ობიექტს ანუ ორ ცხრილს შორის კავშირის დასამყარებლად. DataRelation ობიექტები შეგვიძლია გამოვიყენოთ მონაცემთა ბაზაში ორ ცხრილს შორის "მთავარი-დამოკიდებული" კავშირის შესაქმნელად. ერთ DataSet ობიექტში შეიძლება რამდენიმე DataRelation ობიექტის მოთავსება.

Constraint კლასი. ამ კლასის ობიექტი გამოიყენება ცხრილის შეზღუდვების შესანახად, როგორცაა სვეტის მნიშვნელობის შეზღუდვა უნიკალურობაზე ან გარე გასაღების შეზღუდვა, რომელიც სხვა ცხრილს მიმართავს. ერთ DataTable ობიექტში შეიძლება რამდენიმე Constraint ობიექტი მოვათავსოთ.

DataView კლასი. ამ კლასის ობიექტი გამოიყენება DataTable ობიექტში მოთავსებული ცხრილის გაფილტვრისა და დახარისხებისათვის. ერთი DataSet ობიექტი შეიძლება რამდენიმე DataView ობიექტს შეიცავდეს.

DataSet, DataTable, DataRow, DataColumn, DataRelation, Constraint და DataView კლასები

განსაზღვრულია **System.Data** სახელების სივრცეში.

ობიექტი-მიმწოდებლები და შესაბამისი კლასები

მოკლედ აღვწეროთ ობიექტი-მიმწოდებლებთან სამუშაო კლასები.

SqlConnection, OleDbConnection, OdbcConnection კლასები. SqlConnection კლასის ობიექტი გამოიყენება SQL სერვერზე მონაცემთა ბაზასთან შეერთების დასამყარებლად. OleDbConnection კლასის ობიექტი გამოიყენება იმ მონაცემთა ბაზის მართვის სისტემასთან მისაერთებლად, რომელიც OLEDB ტექნოლოგიას (Object Linking and Embedding for Databases - ობიექტის დაკავშირება და ჩადგმა მონაცემთა ბაზებისათვის) უზრუნველყოფს. ასეთი სისტემებია Oracle და Access. OdbcConnection კლასის ობიექტი გამოიყენება იმ მონაცემთა ბაზის მართვის სისტემასთან მისაერთებლად, რომელიც ODBC (Open Database Connectivity - მონაცემთა ბაზასთან მიმართვის ღია ინტერფეისი) ტექნოლოგიას უზრუნველყოფს. ყველა ძირითადი მონაცემთა ბაზა უზრუნველყოფს ODBC ტექნოლოგიას, მაგრამ მისი საშუალებით მონაცემთა ბაზასთან მიმართვა .NET სისტემაში, ჩვეულებრივ, ნელა სრულდება, ვიდრე წინა ორი საშუალებით.

SqlCommand, OleDbCommand, OdbcCommand კლასები. ამ კლასების ობიექტები გამოიყენება SQL მოთხოვნის ფორმირებისათვის ან შენახული პროცედურის გამოძახებისათვის. ფორმირებული მოთხოვნა ან პროცედურის გამოძახება შეიძლება შესრულდეს წინა აბზაცში ჩამოთვლილი კლასებიდან ერთ-ერთის საშუალებით.

SqlDataReader, OleDbDataReader, OdbcDataReader კლასები. ამ კლასების ობიექტები გამოიყენება ინფორმაციის წასაკითხად მონაცემთა ბაზიდან, რომელიც მოთავსებულია SQL სერვერზე ან მუშაობს იმ მონაცემთა ბაზის მართვის სისტემის ქვეშ, რომელიც უზრუნველყოფს OLEDB ან ODBC ტექნოლოგიას. ეს ობიექტები გამოიყენება მონაცემების მხოლოდ წასაკითხად მონაცემების წყაროდან და წარმოადგენენ DataSet ობიექტის ალტერნატივას. მონაცემების წაკითხვა ამ ობიექტების საშუალებით, როგორც წესი, უფრო სწრაფად სრულდება, ვიდრე DataSet ობიექტის გამოყენებით.

SqlDataAdapter, OleDbDataAdapter, OdbcDataAdapter კლასები. ამ კლასების ობიექტები გამოიყენება სტრიქონების გადასაადგილებლად DataSet ობიექტსა და მონაცემთა ბაზას შორის, რომელიც მოთავსებულია SQL სერვერზე, ან მუშაობს იმ მონაცემთა ბაზის მართვის სისტემის ქვეშ, რომელიც უზრუნველყოფს OLEDB ან ODBC ტექნოლოგიას.

მართვადი კლასები SQL სერვერისათვის გამოცხადებულია **System.Data.SqlClient** სახელების სივრცეში. კლასები მონაცემთა ბაზებისათვის, რომლებიც უზრუნველყოფენ ODBC ტექნოლოგიას, გამოცხადებულია **System.Data.Odbc** სახელების სივრცეში.

System.Data სახელების სივრცე

C# პროგრამაში ADO.NET კლასების გამოყენებამდე using დირექტივების ბლოკში უნდა მოვათავსოთ დირექტივა:

```
using System.Data
```

სწორედ ამ სახელების სივრცეშია მოთავსებული ADO.NET კლასები. ამის შემდეგ, ჩვენ უნდა ავირჩიოთ .NET მონაცემების კონკრეტული მიმწოდებელი, ჩვენს მიერ გამოყენებული მონაცემთა კონკრეტული წყაროსთვის:

- თუ ვმუშაობთ SQL სერვერზე მოთავსებულ მონაცემთა ბაზასთან, მაშინ using დირექტივების ბლოკში უნდა მოვათავსოთ დირექტივა: using System.Data.SqlClient
- თუ ვიყენებთ SQL სერვერისა და Oracle-საგან განსხვავებული მონაცემთა ბაზას, მაგალითად Access, მაშინ using დირექტივების ბლოკში უნდა მოვათავსოთ დირექტივა: using System.Data.OleDb

- თუ ვიყენებთ მონაცემთა წყაროებს, რომლებსაც არ არსებობს „მშობლიური“ OLE DB მომწოდებელი. using დირექტივების ბლოკში უნდა მოვათავსოთ დირექტივა: using System.Data.Odbc

მოთხოვნების შესრულება ADO.NET კლასების გამოყენებით

მონაცემების წაკითხვა DataReader ობიექტის საშუალებით

DataReader ობიექტი SELECT ბრძანებაში (მოთხოვნაში) მითითებული ცხრილიდან თითო სტრიქონს კითხულობს. წაკითხვის პროცესი შემდეგი მოქმედებებისაგან შედგება:

1. მონაცემების წყაროსთან მიერთება. ამისათვის, ვქმნით შეერთების ობიექტს:

```
SqlConnection myConnection =
```

```
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
```

myConnection ობიექტს ენიჭება შეერთების სტრიქონი, რომელიც ოთხ პარამეტრს შეიცავს: **კომპიუტერის (სერვერის) სახელი**, რომელზეც მუშაობს SQL სერვერი. ეს სახელი ეთითება შეერთების სტრიქონის server პარამეტრში. თუ SQL სერვერი ქსელურ კომპიუტერზეა მოთავსებული, მაშინ უნდა მივუთითოთ კომპიუტერის სახელი. თუ სერვერი ჩვენს (ლოკალურ) კომპიუტერზე მუშაობს, მაშინ შეგვიძლია მივუთითოთ როგორც კომპიუტერის სახელი, ისე localhost სიტყვა, server = localhost. **მონაცემთა ბაზის სახელი** database პარამეტრში ეთითება, database = Shekveta. **მომხმარებლის სახელი** შეერთების სტრიქონის uid პარამეტრში ეთითება, uid = sa. **პაროლი**, მონაცემთა ბაზის მოცემული მომხმარებლისათვის, pwd პარამეტრში ეთითება, pwd=paroli.

2. შეერთების გახსნა:

```
myConnection.Open();
```

3. SELECT ბრძანების ფორმირება:

```
myCommand.CommandText = "SELECT gvari, ganyofileba, asaki, tarigi_dabadebis FROM Personali";
```

ამ SELECT ბრძანების შესრულების შედეგად მიღებულ სტრიქონში მოთავსებული იქნება მხოლოდ მითითებული სვეტები. თუ გვინდა ყველა სვეტის მიღება, მაშინ SELECT სიტყვის შემდეგ სვეტების სახელების ნაცვლად უნდა მივუთითოთ '*'.
4. SqlDataReader ტიპის ობიექტის საშუალებით მონაცემების წაკითხვა და ეკრანზე მათი გამოტანა.

myCommand ობიექტის ExecuteReader() მეთოდის საშუალებით იქმნება SqlDataReader ტიპის myReader ობიექტი. ExecuteReader() მეთოდი ასრულებს SELECT ბრძანებას და ქმნის ობიექტ-წამკითხველს გენერირებული შედეგების წაკითხვისათვის. ეს ობიექტი-წამკითხველი ენიჭება myReader ობიექტს.

ობიექტი-წამკითხველიდან შედეგების მიღების ერთ-ერთი გზაა myReader ობიექტის Read() მეთოდის გამოყენება. ეს მეთოდი კითხულობს ერთ სტრიქონს, რომელიც მიღებული იყო SELECT ბრძანების შესრულების შედეგად და გასცემს true-ს, თუ კიდევ არის წასაკითხი სტრიქონები, წინააღმდეგ შემთხვევაში - false-ს. კოდის ფრაგმენტს აქვს სახე:

```
SqlDataReader myReader = myCommand.ExecuteReader();
```

```
for (; myReader.Read();)
```

```
    label10.Text += myReader["gvari"] + " " + myReader["ganyofileba"] + " " +  
    myReader["asaki"].ToString() + " " + myReader["tarigi_dabadebis"] + "\n";
```

როგორც, კოდის ამ ფრაგმენტიდან ჩანს, ციკლი სრულდება მანამ, სანამ Read() მეთოდი გასცემს true მნიშვნელობას. ყურადღება მივაქციოთ იმას, რომ საჭირო სვეტთან მიმართვისთვის მისი სახელი მითითებულია myReader სახელის მარჯვნივ კვადრატულ ფრჩხილებში, მაგალითად myReader["gvari"]. სვეტის სახელის ნაცვლად შეიძლება მისი ინდექსის მითითებაც - myReader[0]. "gvari" სვეტის ინდექსია 0, რადგან ეს პირველი სვეტია SELECT მოთხოვნაში, მეორეა "ganyofileba"

სვეტი, ამიტომ მისი ინდექსია 1 და ა.შ.

5. DataReader-ისა და შეერთების დახურვა:

```
myReader.Close();
```

```
myConnection.Close();
```

მოყვანილი პროგრამით ხდება განხილული ეტაპების რეალიზება, კერძოდ SQL სერვერის Shekveta ბაზის Personali ცხრილიდან სტრიქონების წაკითხვა:

```
// პროგრამა 17.1
```

```
// SQL სერვერის Shekveta ბაზის Personali ცხრილიდან სტრიქონების წაკითხვა
```

```
{
```

```
label1.Text = "";
```

```
// შეერთების ობიექტის შექმნა
```

```
SqlConnection myConnection =
```

```
new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
```

```
// შეერთების გახსნა
```

```
myConnection.Open();
```

```
// ამ შეერთებისთვის ბრძანების შექმნა
```

```
SqlCommand myCommand = myConnection.CreateCommand();
```

```
// SQL მოთხოვნის განსაზღვრა ამ ბრძანებისთვის
```

```
myCommand.CommandText = "SELECT gvari, ganyofileba, asaki, tarigi_dabadebis FROM Personali";
```

```
// მოცემული ბრძანებისთვის DataReader-ის შესრულება
```

```
SqlDataReader myReader = myCommand.ExecuteReader();
```

```
// კითხვა მანამ, სანამ არის წასაკითხი სტრიქონები
```

```
for ( ; myReader.Read(); )
```

```
label1.Text += myReader["gvari"] + " " + myReader["ganyofileba"] + " " +
```

```
myReader["asaki"].ToString() + " " + myReader["tarigi_dabadebis"] + "\n";
```

```
// მკითხველის დახურვა
```

```
myReader.Close();
```

```
// შეერთების დახურვა
```

```
myConnection.Close();
```

```
}
```

Access მონაცემთა ბაზიდან სტრიქონების წასაკითხად, ჩვენს პროგრამას დასაწყისში ჯერ დავუმატოთ

```
using System.Data.OleDb;
```

დირექტივა, შემდეგ კი 17.19 პროგრამაში SqlConnection, SqlCommand და SqlDataReader ობიექტები შევცვალოთ OleDbConnection, OleDbCommand და OleDbDataReader ობიექტებით. უნდა შევცვალოთ, აგრეთვე შეერთების სტრიქონი, რომელსაც ექნება შემდეგი სახე:

```
@ "Provider=Microsoft.Jet.OLEDB.4.0;
```

```
DataSource=C:\Users\Romani\Documents\ROMANI\Access\db5.mdb"
```

შეერთების სტრიქონის წინ მოთავსებული @ სიმბოლო აუქმებს სტრიქონის მმართველი სიმბოლოების მოქმედებას. თუ ეს სიმბოლო არ არის მითითებული, მაშინ ` სიმბოლოს ნაცვლად შეერთების სტრიქონში უნდა მივუთითოთ `\' სიმბოლოები. შეერთების სტრიქონში Provider კონსტრუქცია განსაზღვრავს OLE DB მიმწოდებელს Access მონაცემთა ბაზისთვის, DataSource კონსტრუქცია კი განსაზღვრავს მონაცემთა ბაზის კონკრეტულ ფაილს, რომელთანაც უნდა შესრულდეს მიმართვა.

მოყვანილი პროგრამით სრულდება Access მონაცემთა ბაზიდან, კერძოდ db5.mdb მონაცემთა ბაზის Personali ცხრილიდან სტრიქონების წაკითხვა.

```
{
```

```

//      პროგრამა 17.2
//      Access სისტემის Shekveta ბაზის Personalი ცხრილიდან სტრიქონების წაკითხვა
label1.Text = "";
//      შეერთების ობიექტის შექმნა
OleDbConnection myConnection = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;
                                     Data Source=C:\Users\Romani\Documents\ROMANI\Access\db5.mdb");
//      შეერთების გახსნა
myConnection.Open();
//      ბრძანების შექმნა ამ შეერთებისთვის
OleDbCommand myCommand = myConnection.CreateCommand();
//      SQL მოთხოვნის განსაზღვრა ამ ბრძანებისთვის
myCommand.CommandText = "SELECT gvარი, ganyofileba, asaki, tarigi_dabadebis FROM Personalი";
//      მოცემული ბრძანებისთვის DataReader-ის შესრულება
OleDbDataReader myReader = myCommand.ExecuteReader();
//      სანამ არის წასაკითხი სტრიქონები
for ( ; myReader.Read(); )
    label1.Text += myReader["gvარი"] + " " + myReader["ganyofileba"] + " " +
        myReader["asaki"].ToString() + " " + myReader["tarigi_dabadebis"] + "\n";
//      მკითხველის დახურვა
myReader.Close();
//      შეერთების დახურვა
myConnection.Close();
}

```

მონაცემთა ბაზასთან მუშაობა DataSet ობიექტის გამოყენებით

მონაცემების კითხვა

ახლა ვნახოთ, თუ როგორ შეიძლება მონაცემების წაკითხვა მონაცემთა ბაზიდან DataSet ობიექტის გამოყენებით. როგორც ნახ. 17.24-დან ჩანს, DataSet ობიექტი მოიცავს DataRelation და DataTable ობიექტებს. თავის მხრივ, DataTable ობიექტი მოიცავს DataRow ობიექტებს, DataRow ობიექტი კი - DataColumn ობიექტებს. აქედან, გამომდინარე, DataSet ობიექტში შეგვიძლია მოვათავსოთ ერთი ან მეტი ცხრილი, მაგალითად Personalი, Shemkveti და Xelshekruleba, და მათ შორის ბმები (რელაციები, კავშირები). მხოლოდ ამის შემდეგ შეგვეძლება ამ ცხრილებთან მუშაობა. ცხრილებით DataSet ობიექტის შესავსებად გამოიყენება ადაპტერის Fill() მეთოდი. ადაპტერი არის ობიექტი, რომელიც გამოიყენება მონაცემების გადასაგზავნად კლიენტიდან (ჩვენი კომპიუტერიდან) სერვერზე და პირიქით.

DataSet ობიექტს აქვს Tables თვისება, რომელიც წარმოადგენს DataTable ტიპის ობიექტების კოლექციას. ეს თვისება საშუალებას გვაძლევს ცხრილებს მივმართოთ როგორც სახელის, ისე ინდექსის მიხედვით. მაგალითად,

```

myDataSet.Tables["Personalი"]; ან
myDataSet.Tables[0];

```

შედეგად, DataTable ობიექტში შეგვიძლია მოვათავსოთ არჩეული ცხრილი:

```

DataTable personaliTable = new DataTable();
personaliTable = myDataSet.Tables["Personalი"];

```

შევნიშნოთ, რომ ცხრილების ნუმერაცია (ინდექსაცია) ნულიდან იწყება.

Tables თვისებას, თავის მხრივ აქვს Rows თვისება, რომელიც არის DataRow ობიექტების

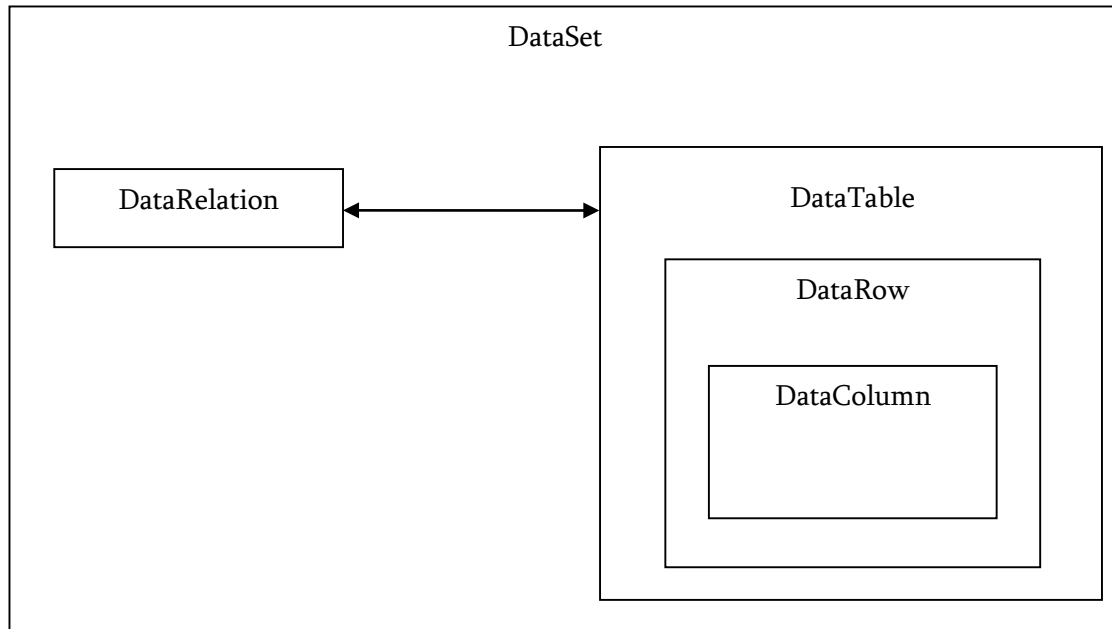
კოლექცია. ეს თვისება საშუალებას გვაძლევს ცხრილის სტრიქონს მივმართოთ ინდექსის მიხედვით. მაგალითად,

```
myDataSet.Tables["Personali"].Rows[2];
```

შედეგად, DataRow ობიექტში შეგვიძლია მოვათავსოთ არჩეული სტრიქონი:

```
DataRow myRow = myDataSet.Tables["Personali"].Rows[3];
```

შევნიშნოთ, რომ სტრიქონების ინდექსაცია 0-დან იწყება.



ნახ. 17.24. DataSet ობიექტის სტრუქტურა

Rows თვისება საშუალებას გვაძლევს, აგრეთვე მივმართოთ სტრიქონის სვეტებს სახელის ან ინდექსის მიხედვით. მაგალითად,

```
object gvari_1 = myDataSet.Tables["Personali"].Rows[2]["gvari"];
```

```
object gvari_2 = myDataSet.Tables["Personali"].Rows[2][1];
```

შეგვიძლია, აგრეთვე კონკრეტულ ცვლადში ჩავწეროთ არჩეული სვეტის მნიშვნელობა

```
string gvari_3 = myDataSet.Tables["Personali"].Rows[2]["gvari"].ToString();
```

უნდა გვახსოვდეს, რომ სვეტების ნუმერაცია 0-დან იწყება. რაც შეეხება იმ საკითხს, სახელის გამოყენება ჯობია თუ ინდექსის, უმჯობესია სახელის გამოყენება, რადგან მომავალში სვეტებმა ცხრილში შეიძლება მიმდევრობა შეიცვალოს.

მოყვანილი პროგრამით ხდება სერვერზე მოთავსებული Shekveta მონაცემთა ბაზის Personali ცხრილის სტრიქონების წაკითხვა და მათი მოთავსება myDataset ობიექტში:

```

{
// პროგრამა 17.3
// პროგრამით ხდება Personali ცხრილის მოთავსება myDataset მონაცემთა ნაკრებში
// შეერთების ობიექტის შექმნა
SqlConnection myConnection =
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
// შეერთების გახსნა
myConnection.Open();
// SqlDataAdapter ობიექტის შექმნა
SqlDataAdapter myAdapter = new
SqlDataAdapter("SELECT gvari, ganyofileba, asaki, tarigi_dabadebis FROM Personali", myConnection);
  
```



```
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personalი ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personalი");
// Personalი ცხრილის სტრიქონების გამოტანა ეკრანზე
dataGridView4.DataSource = myDataset;
dataGridView4.DataMember = "Personalი";
// შეერთების დახურვა
myConnection.Close();
}
```

სტრიქონში:

```
myAdapter.Fill(myDataset, "Personalი");
```

myDataset ობიექტში იქმნება DataTable ტიპის Personalი ობიექტი, რომელიც ივსება სტრიქონებით Shekveta მონაცემთა ბაზის Personalი ცხრილიდან. Fill() მეთოდი ასრულებს myAdapter ობიექტთან დაკავშირებულ SELECT მოთხოვნას. ამრიგად, Personalი ობიექტი მოთავსებულია არა სერვერზე, არამედ კლიენტის კომპიუტერზე (ჩვენს კომპიუტერზე) myDataset ობიექტში. კლიენტისა და სერვერის სტრუქტურა ნაჩვენებია ნახ. 17. 25-ზე.

ცხრილებით DataSet ობიექტის შევსების შემდეგ, ამ ცხრილებთან მუშაობა შეგვეძლება ავტონომურ (ოფლაინ) რეჟიმში ანუ სერვერთან კავშირის გარეშე.

მონაცემთა ბაზასთან შეერთების ამკარად გახსნა ან დახურვა აუცილებელი არ არის, რადგან ამას ადაპტერი ავტომატურად აკეთებს. myAdapter ობიექტი-ადაპტერი, საჭიროების მიხედვით, ხსნის და ხურავს შეეთებას მონაცემთა ბაზასთან. ადაპტერი შეერთების მდგომარეობას უცვლელად ინარჩუნებს. ამიტომ, თუ შეერთება გახსნილი იყო ადაპტერის მუშაობის დაწყებამდე, მაშინ ადაპტერის მუშაობის დამთავრების შემდეგ შეერთება ისევ დარჩება გახსნილი. ამისგან, განსხვავებით, DataReader ითხოვს მონაცემთა ბაზასთან შეერთების არსებობას მისი მუშაობის მთელი პერიოდის განმავლობაში.

Personalი ცხრილის სტრიქონები შეგვიძლია გამოვიტანოთ label კომპონენტშიც:

```
foreach ( DataRow theRow in myDataset.Tables["Personalი"].Rows )
```

```
label1.Text += theRow["gvari"] + " " + theRow["ganyofileba"] + " " +
```

```
theRow["asaki"].ToString() + " " + theRow["tarigi_dabadebis"] + "\n";
```

თუმცა dataGridView კომპონენტში ცხრილის სტრიქონების გამოტანა გაცილებით სწრაფად სრულდება.

თუ ადაპტერის შექმნისას SELECT ბრძანებას აქვს შემდეგი სახე:

```
SqlDataAdapter personaliAdapter = new SqlDataAdapter("SELECT * FROM Personalი", myConnection);
```

მაშინ Fill() მეთოდის შესრულებისას გაიცემა Personalი ცხრილის ყველა სვეტი და ყველა სტრიქონი. ასეთი მიდგომა შეგვიძლია გამოვიყენოთ მცირე ზომის ცხრილებისათვის, რომლებშიც სვეტებისა და სტრიქონების რაოდენობა მცირეა. როცა ცხრილში სტრიქონების რაოდენობა აღემატება 100 000-ს, მაშინ ასეთი მიდგომა კარგად ვერ იმუშავებს. ასეთ შემთხვევებში, საჭიროა გამოვიყენოთ როგორც ვერტიკალური, ისე ჰორიზონტალური ფილტრი. ვერტიკალური ფილტრი გვექნება მაშინ, როცა SELECT ბრძანებაში მივუთითებთ ცხრილის მხოლოდ იმ სვეტებს, რომლებთანაც რეალურად ვაპირებთ მუშაობას, მაგალითად:

```
SqlDataAdapter personaliAdapter = new SqlDataAdapter(
```

```
"SELECT gvari, staji, xelfasi FROM Personalი", myConnection);
```

მაგრამ, ასეთი მიდგომა არ გამოდგება იმ შემთხვევაში, როცა ცხრილს სტრიქონებს ვუმატებთ, რადგან ამ დროს სტრიქონის თითოეულ სვეტს უნდა მივანიჭოთ შესაბამისი მნიშვნელობა.

ჰორიზონტალური ფილტრის შემთხვევაში, SELECT ბრძანებაში უნდა მივუთითოთ

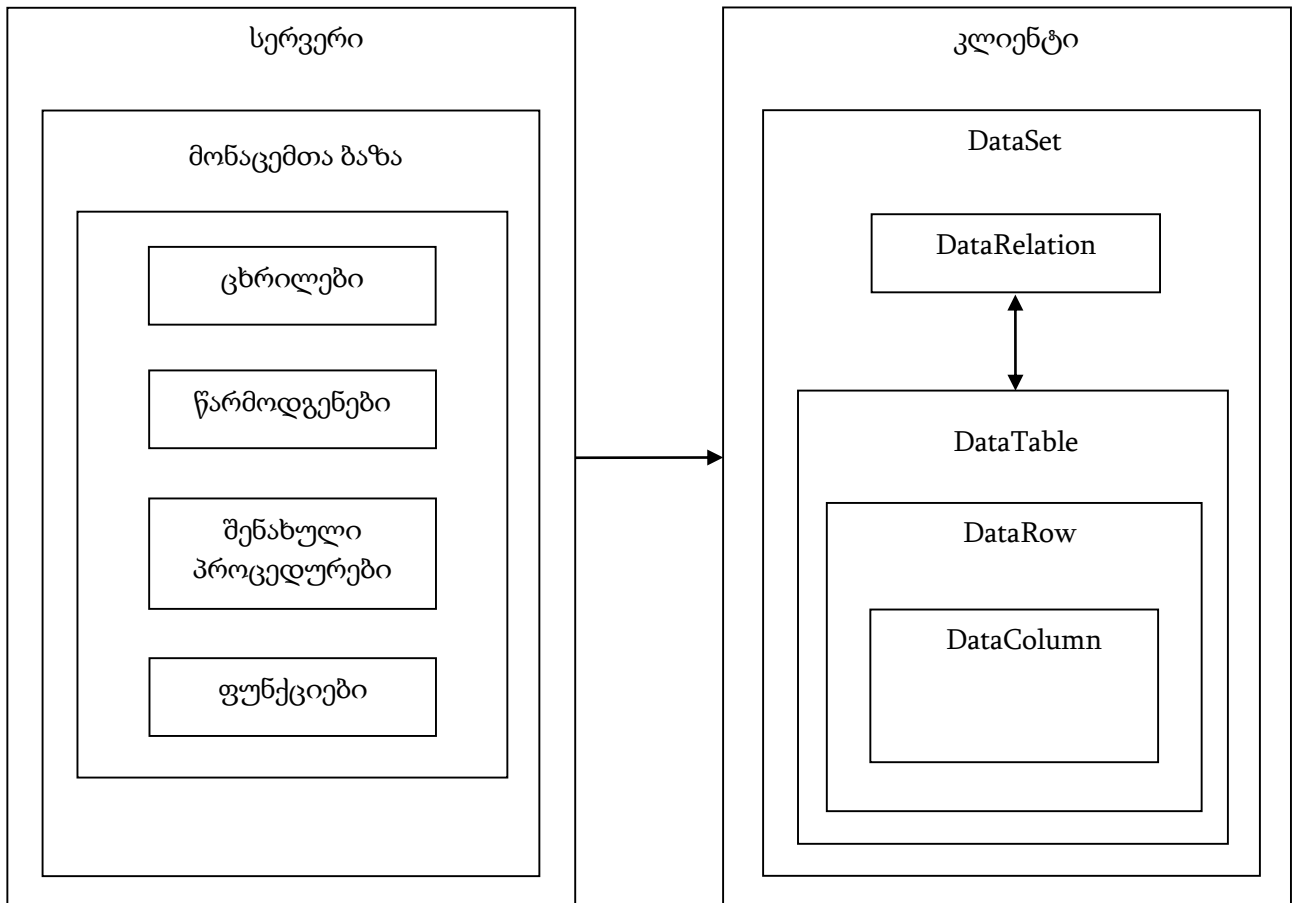
WHERE განყოფილება. შედეგად, ამოირჩევა ცხრილის ერთი ან მეტი სტრიქონი, რომლებთანაც რეალურად ვაპირებთ მუშაობას. დავუშვათ, გვინდა იმ თანამშრომლებზე ინფორმაცია, რომელთა სტაჟი აღემატება 20 წელს. მაშინ SELECT მოთხოვნას ენება სახე:

```
SELECT * FROM Personal WHERE staji >= 20
```

შეგვიძლია მივუთითოთ სვეტებიც:

```
SELECT gvari, asaki, staji, xelfasi FROM Personal WHERE staji >= 20
```

ცუდ მიდგომად ითვლება მთელი ცხრილის გადმოწერა სერვერიდან DataSet ობიექტში და შემდეგ ამ ობიექტში ძებნის შესრულება. ასეთ დროს უმჯობესია SELECT ბრძანების გამოყენება WHERE კონსტრუქციასთან ერთად.



ნახ. 17. 25. კლიენტისა და სერვერის სტრუქტურა

როგორც ვიცით, DataSet ობიექტში შეგვიძლია რამდენიმე ცხრილის მოთავსება. მოყვანილი პროგრამით ხდება myDataSet ობიექტში Personal, Shemkveti და Xelshekruleba ცხრილების მოთავსება:

```
{
// პროგრამა 17.4
// პროგრამით ხდება Personal, Shemkveti და Xelshekruleba ცხრილების
// მოთავსება myDataset მონაცემთა ნაკრებში
// შეერთების ობიექტის შექმნა
SqlConnection myConnection =
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
// შეერთების გახსნა
```

```

myConnection.Open();
//      DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//      DataAdapter ობიექტების შექმნა თითოეული ცხრილისთვის
SqlDataAdapter personaliAdapter = new SqlDataAdapter("SELECT * FROM Personali", myConnection);
SqlDataAdapter shemkvetiAdapter = new
        SqlDataAdapter("SELECT * FROM Shemkveti", myConnection);
SqlDataAdapter xelshekrulebaAdapter = new
        SqlDataAdapter("SELECT * FROM Xelshekruleba", myConnection);
//      myDataset ობიექტის ცხრილებით შევსება
personaliAdapter.Fill(myDataset, "Personali");
shemkvetiAdapter.Fill(myDataset, "Shemkveti");
xelshekrulebaAdapter.Fill(myDataset, "Xelshekruleba");
//      ეკრანზე ცხრილების გამოტანა
dataGridView4.DataSource = myDataset;
dataGridView4.DataMember = "Personali";
dataGridView5.DataSource = myDataset;
dataGridView5.DataMember = "Shemkveti";
dataGridView6.DataSource = myDataset;
dataGridView6.DataMember = "Xelshekruleba";
//      შეერთების დახურვა
myConnection.Close();
}

```

მონაცემების გაფილტვრა

მონაცემების გასაფილტრად შეგვიძლია DataView ობიექტის გამოყენება. დავუშვათ, გვინტერესებს ინფორმაცია სამედიცინო განყოფილებაში მომუშავე იმ თანამშრომლების შესახებ, რომელთა ასაკი აღემატება 30 წელს. მოცემული კოდით ხდება ამის დემონსტრირება:

```

{
//      პროგრამა 17.5
//      პროგრამით ხდება Personali ცხრილის სტრიქონების გაფილტვრა
//      dataView ობიექტის გამოყენებით
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection =
        new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
myConnection.Open();
//      DataAdapter ობიექტის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM Personali", myConnection);
//      DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//      myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
//      შეერთების დახურვა
myConnection.Close();
//      dataView1 ობიექტის დაკავშირება Personali ცხრილთან
DataView dataView1 = new DataView(myDataset.Tables["Personali"]);

```

```
// სტრიქონების გაფილტვრა
dataView1.RowFilter = "ganyofileba = 'სავაჭრო' AND asaki > 30";
// გაფილტვრული სტრიქონების ასახვა ეკრანზე
dataGridView1.DataSource = dataView1;
}
```

როგორც პროგრამიდან ჩანს dataView1 ობიექტის RowFilter თვისებას უნდა მივანიჭოთ ფილტრი, რომელიც სტრიქონს წარმოადგენს. ფილტრის გასაუქმებლად უნდა შევასრულოთ კოდი:

```
dataView1.RowFilter = "";
// ფილტრის შესატანად შეგვიძლია textBox კომპონენტის გამოყენებაც:
dataView1.RowFilter = textBox1.Text;
// ამ შემთხვევაში, textBox კომპონენტში ფილტრი შეგვაქვს ბრჭყალების გარეშე.
// თუ გვინტერესებს ის თანამშრომლები, რომლებიც დაიბადნენ 1990 წლის 1 იანვრის
// შემდეგ, მაშინ dataView1 ობიექტის RowFilter თვისებას უნდა მივანიჭოთ შემდეგი ფილტრი:
dataView1.RowFilter = "tarigi_dabadebis > '01.01.1990'";
// თარიღის შეტანა შეგვიძლია dateTimePicker1 კომპონენტიდანაც:
dataView1.RowFilter = "tarigi_dabadebis > " + dateTimePicker1.Value.ToString() + "";
```

მონაცემების დახარისხება

მონაცემების დასახარისხებლად შეგვიძლია DataView ობიექტის გამოყენება. ამისათვის გამოიყენება მისი Sort თვისება. მოყვანილი კოდით ხდება Personalი ცხრილის სტრიქონების დახარისხება ganyofileba სვეტის მნიშვნელობების ზრდის მიხედვით და asaki სვეტის მნიშვნელობების კლების მიხედვით:

```
{
// პროგრამა 17.6
// პროგრამით ხდება Personalი ცხრილის სტრიქონების დახარისხება
// dataView ობიექტის გამოყენებით
// შეერთების ობიექტის შექმნა
SqlConnection myConnection =
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
// შეერთების გახსნა
myConnection.Open();
// DataAdapter ობიექტის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM Personal", myConnection);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personalი ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personal");
// შეერთების დახურვა
myConnection.Close();
// dataView1 ობიექტის დაკავშირება Personalი ცხრილთან
DataView dataView1 = new DataView(myDataset.Tables["Personal"]);
// სტრიქონების დახარისხება
dataView1.Sort = "ganyofileba, asaki DESC";
// დახარისხებული სტრიქონების ასახვა ეკრანზე
dataGridView1.DataSource = dataView1;
}
```

დახარისხების გაუქმება შემდეგნაირად შეგვიძლია:
dataView1.Sort = "";

სტრიქონების ძებნა

ცხრილში სტრიქონის მისაძებნად Find() მეთოდი გამოიყენება. ეს მეთოდი ითხოვს ცხრილში პირველადი გასაღების დაყენებას. საჭირო მნიშვნელობა იძებნება პირველად გასაღებში. პირველადი გასაღები შეიძლება შედგებოდეს ერთი ან მეტი სვეტისგან, რომელთა მნიშვნელობები ცალსახად განსაზღვრავენ ერთ სტრიქონს. ამიტომ, ძებნის შედეგი ყოველთვის იქნება ცხრილის ერთი სტრიქონი.

მოყვანილი პროგრამით ხდება PersonalI ცხრილში სტრიქონის ძებნის დემონსტრირება:

```
{
//   პროგრამა 17.7
//   პროგრამით ხდება PersonalI ცხრილში სტრიქონის ძებნა
//   შეგვაქვს პირველადი გასაღების მნიშვნელობა
int find_key = Convert.ToInt32(textBox1.Text);
//   შეერთების ობიექტის შექმნა
SqlConnection myConnection =
        new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//   შეერთების გახსნა
myConnection.Open();
//   ადაპტერის მომზადება სამუშაოდ
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM PersonalI", myConnection);
//   SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
//   DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//   myDataset ობიექტის შევსება PersonalI ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "PersonalI");
//   გასაღებების მასივის გამოცხადება პირველადი გასაღების განსაზღვრისათვის
DataColumn[] keys = new DataColumn[1];
keys[0] = myDataset.Tables["PersonalI"].Columns["PersonalIID"];
myDataset.Tables["PersonalI"].PrimaryKey = keys;
//   findRow ობიექტს ენიჭება ნაპოვნი სტრიქონი
DataRow findRow = myDataset.Tables["PersonalI"].Rows.Find(find_key);
if ( findRow != null )
{
    label1.Text = "სტრიქონი მოიძებნა: ";
    label1.Text += findRow["gvari"] + " " + findRow["xelfasi"].ToString() + " ";
}
else label1.Text = "სტრიქონი ვერ მოიძებნა";
//   შეერთების დახურვა
myConnection.Close();
}
```

მოყვანილ პროგრამაში საძებნი მნიშვნელობა textBox1.Text კომპონენტიდან ენიჭება find_key ცვლადს. პირველადი გასაღებია PersonalI ცხრილის PersonalIID სვეტი. მისი განსაზღვრა შემდეგნაირად ხდება:

```
DataColumn[] keys = new DataColumn[1];
```

```
keys[0] = myDataset.Tables["Personali"].Columns["PersonaliID"];
myDataset.Tables["Personali"].PrimaryKey = keys;
```

როგორც ვხედავთ, ჯერ იქმნება DataColumn ტიპის მქონე keys ობიექტი, რომელშიც უნდა მოვათავსოთ პირველადი გასაღები. რადგან პირველადი გასაღები შეიძლება იყოს ერთი ან მეტი სვეტი, ამიტომ keys არის DataColumn ტიპის ობიექტების მასივი. შემდეგ, ამ მასივის პირველ ელემენტს ენიჭება Personali ცხრილის PersonaliID სვეტი. ბოლოს კი - DataTable ტიპის Personali ობიექტის PrimaryKey თვისებას ენიჭება keys ობიექტი.

პირველადი გასაღების განსაზღვრის შემდეგ შეგვიძლია შევასრულოთ ძებნა Rows თვისების Find() მეთოდის გამოყენებით. ეს მეთოდი გასცემს DataRow ობიექტს (სტრიქონს), რომელიც ენიჭება ამავე ტიპის findRow ობიექტს. მეთოდს არგუმენტად გადაეცემა საძებნი სიდიდე. თუ პირველადი გასაღები ორი და მეტი სვეტისგან შედგება, მაშინ Find() მეთოდს არგუმენტად უნდა გადავცეთ ობიექტების მასივი. ჩვენს შემთხვევაში, პირველადი გასაღები ერთი სვეტისგან შედგება, ამიტომ მეთოდს ერთ მნიშვნელობას გადავცემთ. საძებნი მნიშვნელობა find_key ცვლადს ენიჭება textBox1 კომპონენტიდან პროგრამის დასაწყისში.

Find() მეთოდი სტრიქონის პოვნის შემთხვევაში ამ სტრიქონს გასცემს DataRow ობიექტის სახით, წინააღმდეგ შემთხვევაში, გასცემს null მიმართვას. ამ შემოწმებას ასრულებს if ოპერატორი:

```
if ( findRow != null )
```

ასეთივე გზით შეგვიძლია ჯერ მოვძებნოთ საჭირო სტრიქონი და შემდეგ შევცვალოთ მისი სვეტები ან წავშალოთ ნაპოვნი სტრიქონი.

მონაცემთა ბაზაში მონაცემების შეცვლა

ამ განყოფილებაში ვნახავთ, თუ როგორ შეიძლება შევცვალოთ ცხრილებში მოთავსებული მონაცემები. ამისათვის უნდა შევასრულოთ შემდეგი მოქმედებები:

- DataSet ობიექტის შევსება იმ ცხრილებით, რომლებშიც გვინდა მონაცემების შეცვლა.
- DataSet ობიექტში ცხრილების შეცვლა.
- DataSet ობიექტში მოთავსებული ცხრილების გადაგზავნა სერვერზე მონაცემთა ბაზაში.

მოყვანილი პროგრამით ხდება Personali ცხრილის მე-4 სტრიქონში (მისი ინდექსია 3) "asaki" სვეტის მნიშვნელობის შეცვლა. SELECT ბრძანებაში პირველადი გასაღების (personaliID) მითითება აუცილებელია.

```
{
// პროგრამა 17.8
// პროგრამით ხდება Personali ცხრილის სტრიქონში სვეტის მნიშვნელობის შეცვლა
// შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
    SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
// შეერთების გახსნა
myConnection.Open();
// ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter(
    "SELECT personaliID, gvari, ganyofileba, asaki, tarigi_dabadebis FROM Personali", myConnection);
// SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
```

```

// მონაცემების გამოტანა ცვლილებამდე
label1.Text = myDataset.Tables["Personali"].Rows[3]["asaki"].ToString() + "\n";
// Personalis ცხრილის მე-3 სტრიქონში ასაკის შეცვლა
myDataset.Tables["Personali"].Rows[3]["asaki"] = Convert.ToInt32(textBox1.Text);
// Update ბრძანების გამოძახება ცვლილებების დაფიქსირებისათვის ცხრილში
myAdapter.Update(myDataset, "Personali");
// ეკრანზე ცვლილებების გამოტანა
label2.Text = myDataset.Tables["Personali"].Rows[3]["asaki"].ToString();
// Personalis ცხრილის სტრიქონების გამოტანა ეკრანზე
dataGridView4.DataSource = myDataset;
dataGridView4.DataMember = "Personali";
// შეერთების დახურვა
myConnection.Close();
}

```

პროგრამაში ადაპტერის შექმნის შემდეგ უნდა შევქმნათ კორექტული SQL ბრძანებები: INSERT, UPDATE და DELETE, ცხრილებში მონაცემების შეცვლის მიზნით. ამას, ავტომატურად აკეთებს SqlCommandBuilder კლასი. მის კონსტრუქტორს არგუმენტად გადაეცემა myAdapter ობიექტი. myBuilder ობიექტის შექმნისას ხდება SQL ბრძანებების გენერირება და ასოცირება (დაკავშირება) myAdapter ობიექტთან კონსტრუქტორის მიერ.

myAdapter ობიექტის Update() მეთოდი ასრულებს myDataset ობიექტიდან Personalis ცხრილის გადაგზავნას სერვერზე Shekveta მონაცემთა ბაზაში. ეს მეთოდი ამოწმებს Personalis ცხრილის თითოეულ სტრიქონს, შეიცვალა თუ არა ის. Rows კოლექციის თითოეულ სტრიქონს აქვს RowState თვისება, რომელიც განსაზღვრავს სტრიქონი იყო თუ არა წაშლილი, დამატებული, შეცვლილი თუ დარჩა უცვლელი. თითოეული ცვლილება აისახება Update() მეთოდის მიერ მონაცემთა ბაზაში. უნდა გვახსოვდეს, რომ Update() მეთოდი ითხოვს ცხრილში პირველადი გასაღების არსებობას.

როგორც პროგრამიდან ჩანს, იმისათვის, რომ ცხრილის რომელიმე სტრიქონის ველს მივანიჭოთ საჭირო მნიშვნელობა, უნდა მივუთითოთ სტრიქონის ინდექსი (ნომერი) და სვეტის სახელი ან ინდექსი:

```
myDataset.Tables["Personali"].Rows[3]["asaki"] = Convert.ToInt32(textBox1.Text);
```

ეს ცვლილება ეხება ჩვენს ლოკალურ კომპიუტერში მოთავსებულ myDataset ობიექტის Personalis ცხრილს და არა სერვერზე მოთავსებული Shekveta მონაცემთა ბაზის Personalis ცხრილს.

დავუბრუნდეთ სტრიქონს:

```
label2.Text += myDataset.Tables["Personali"].Rows[3]["asaki"].ToString();
```

ეს სტრიქონი ეკვივალენტურია შემდეგი სტრიქონების:

```
DataTable personaliTable = myDataset.Tables["Personali"];
DataRow myRow = personaliTable.Rows[3];
object asaki = myRow["asaki"];
label2.Text = asaki.ToString();
```

```
object asaki = myRow["asaki"];
label2.Text = asaki.ToString();
```

```
object asaki = myRow["asaki"];
label2.Text = asaki.ToString();
```

```
object asaki = myRow["asaki"];
label2.Text = asaki.ToString();
```

რომელ კოდს გამოვიყენებთ, ჩვენი გადასაწყვეტია. ზოგადად, უმჯობესია იმ კოდის გამოყენება, რომელიც ჩვენთვის უფრო გასაგებია.

სტრიქონის სვეტებისთვის მნიშვნელობების მისანიჭებლად, მოცემული სტრიქონების ნაცვლად:

```
myDataset.Tables["Personali"].Rows[3]["gvari"] = textBox1.Text;
```

```
myDataset.Tables["Personali"].Rows[3]["asaki"] = Convert.ToInt32(textBox2.Text);
```

```
myDataset.Tables["Personali"].Rows[3]["ganyofileba"] = textBox3.Text;
```

```
myDataset.Tables["Personali"].Rows[3]["tarigi_dabadebis"] = dateTimePicker1.Value;
```

შეგვიძლია, შემდეგი კოდის გამოყენება:

```
DataRow myRow = personaliTable.Rows[3];
myRow["gvari"] = textBox1.Text;
myRow["asaki"] = Convert.ToInt32(textBox2.Text);
myRow["ganyofileba"] = textBox3.Text;
myRow["tarigi_dabadebis"] = dateTimePicker1.Value;
```

რადგან, ხშირ შემთხვევაში, პირველადი გასაღებების ცოდნა შეუძლებელია, ამიტომ შესაცვლელი სტრიქონი შეგვიძლია მოვძებნოთ თანამშრომლის გვარის მიხედვით. ამისათვის, გამოვიყენებთ BindingSource ობიექტის Find() მეთოდს. მისი პირველი პარამეტრია იმ სვეტის სახელი, რომელშიც უნდა შესრულდეს ძებნა, მეორე პარამეტრია საძებნი სიდიდე. თუ საძებნი სიდიდე მოიძებნა, მაშინ გაიცემა ნაპოვნი სტრიქონის ინდექსი, წინააღმდეგ შემთხვევაში კი - -1. მოცემული პროგრამით ხდება Personali ცხრილიდან სტრიქონის შეცვლის დემონსტრირება:

```
{
//
//
label1.Text = "";
// შეერთების ობიექტის შექმნა
SqlConnection thisConnection =
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd= paroli");
// DataAdapter ობიექტის შექმნა
SqlDataAdapter thisAdapter =
new SqlDataAdapter("SELECT personaliID, gvari, ganyofileba, asaki, tarigi_dabadebis FROM Personali",
thisConnection);
// SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder thisBuilder = new SqlCommandBuilder(thisAdapter);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet thisDataset = new DataSet();
// thisDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
thisAdapter.Fill(thisDataset, "Personali");
// bind ობიექტის დაკავშირება Personali ცხრილთან
BindingSource bind = new BindingSource(thisDataset, thisDataset.Tables["Personali"].ToString());
// index ცვლადში იწერება მოძებნილი სტრიქონის ნომერი
int index = bind.Find("gvari", textBox15.Text);
label1.Text += index.ToString() + "\n";
// პირველადი გასაღების ფორმირება
DataColumn[] keys = new DataColumn[1];
keys[0] = thisDataset.Tables["Personali"].Columns["PersonaliID"];
thisDataset.Tables["Personali"].PrimaryKey = keys;
if ( index != -1 )
{
// ნაპოვნი სტრიქონის მინიჭება findRow ობიექტისთვის
DataRow findRow = thisDataset.Tables["Personali"].Rows[index];
// მოწმდება სტრიქონის არსებობა
if ( findRow != null )
{
// მონაცემების გამოტანა ცვლილებამდე
label1.Text += findRow["PersonaliID"].ToString() + " " + findRow["gvari"] + " " +
    findRow["asaki"].ToString() + "\n";
```



```

//      Personalis ცხრილის სტრიქონში ასაკის შეცვლა
findRow["asaki"] = Convert.ToInt32(textBox7.Text);
thisAdapter.Update(thisDataset, "Personali");
label1.Text += " სტრიქონი შეიცვალა\n";
//      ეკრანზე ცვლილებების გამოტანა
label1.Text += findRow["asaki"].ToString();
}
else label1.Text += "შესაცვლელი სტრიქონი ვერ მოიძებნა\n";
}
//      Personalis ცხრილის სტრიქონების გამოტანა ეკრანზე
dataGridView4.DataSource = thisDataset;
dataGridView4.DataMember = "Personali";
//      შეერთების დახურვა
thisConnection.Close();
}

```

SELECT, UPDATE, INSERT და DELETE ბრძანებების ნახვა

CommandBuilder ობიექტი, ახდენს SQL ბრძანებების გენერირებას, რომლებიც გამოიყენება მონაცემების მოდიფიცირებისთვის. ეს ბრძანებებია: INSERT, UPDATE და DELETE. მათი გენერირება ხდება ადაპტერში მითითებული SELECT ბრძანების საფუძველზე. გენერირებული ბრძანებების მისაღებად და სანახავად შეგვიძლია შესაბამისად გამოვიყენოთ CommandBuilder ობიექტის GetInsertCommand(), GetUpdateCommand() და GetDeleteCommand() მეთოდები. მოყვანილი პროგრამით ხდება ამის დემონსტრირება:

```

{
//      პროგრამა 17.9
//      პროგრამით ხდება INSERT, UPDATE და DELETE ბრძანებების კოდის გამოტანა ეკრანზე
label1.Text = "";
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
        SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
myConnection.Open();
//      ადაპტერის შექმნა
SqlDataAdapter myAdapter = new
        SqlDataAdapter("SELECT personaliID, gvari FROM Personalis", myConnection);
//      SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
label1.Text += myAdapter.SelectCommand.CommandText + "\n";
//      UPDATE ბრძანების ეკრანზე გამოტანა
SqlCommand updateCommand = myBuilder.GetUpdateCommand();
label1.Text += updateCommand.CommandText + "\n";
//      INSERT ბრძანების ეკრანზე გამოტანა
SqlCommand insertCommand = myBuilder.GetInsertCommand();
label1.Text += insertCommand.CommandText + "\n";
//      DELETE ბრძანების ეკრანზე გამოტანა
SqlCommand deleteCommand = myBuilder.GetDeleteCommand();

```

```

label1.Text += deleteCommand.CommandText + "\n";
// შერეობის დახურვა
myConnection.Close();
}

შედეგს ექნება სახე:
SELECT personaliID, gvari FROM Personali
UPDATE [Personali] SET [gvari] = @p1 WHERE ((([personaliID] = @p2) AND ((@p3 = 1 AND [gvari] IS NULL) OR ([gvari] = @p4)))
INSERT INTO [Personali] ([gvari]) VALUES (@p1)
DELETE FROM [Personali] WHERE (([personaliID] = @p1) AND ((@p2 = 1 AND [gvari] IS NULL) OR ([gvari] = @p3)))

```

ცხრილში სტრიქონების დამატება

ცხრილში ახალი სტრიქონების დასამატებლად შემდეგი მოქმედებები უნდა შევასრულოთ:

- ახალი DataRow ობიექტის შექმნა.
- მონაცემებით მისი შევსება.
- შევსებული სტრიქონის დამატება DataSet ობიექტის Rows კოლექციისთვის.
- ადაპტერის Update() მეთოდის შესრულება.

მოყვანილი პროგრამით ხდება Personali ცხრილში ახალი სტრიქონის დამატება:

```

{
// პროგრამა 17.10
// პროგრამით ხდება Personali ცხრილში ახალი სტრიქონის დამატება
// შერეობის ობიექტის შექმნა
SqlConnection myConnection = new
    SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
// შერეობის გახსნა
myConnection.Open();
// ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter(
    "SELECT personaliID, gvari, xelfasi, staji, tarigi_dabadebis FROM Personali", myConnection);
// SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
// სტრიქონების რაოდენობა ცვლილებამდე
label1.Text = myDataset.Tables["Personali"].Rows.Count.ToString() + "\n";
// ახალი სტრიქონის შექმნა
DataRow myRow = myDataset.Tables["Personali"].NewRow();
// ახალი სტრიქონის სვეტებისთვის მნიშვნელობების მინიჭება
myRow["gvari"] = textBox1.Text;
myRow["staji"] = textBox2.Text;
myRow["xelfasi"] = textBox3.Text;
myRow["tarigi_dabadebis"] = dateTimePicker1.Value.ToString();
myDataset.Tables["Personali"].Rows.Add(myRow);
// სტრიქონების რაოდენობა ცვლილების შემდეგ
label1.Text += myDataset.Tables["Personali"].Rows.Count.ToString() + "\n";

```

```
// Update ბრძანების გამოძახება ცვლილებების დაფიქსირებისათვის ცხრილში
myAdapter.Update(myDataset, "Personali");
// Personali ცხრილის სტრიქონების გამოტანა ეკრანზე
dataGridView4.DataSource = myDataset;
dataGridView4.DataMember = "Personali";
// შეერთების დახურვა
myConnection.Close();
}
```

როგორც პროგრამის მოცემული სტრიქონიდან ჩანს:

```
label1.Text = myDataset.Tables["Personali"].Rows.Count.ToString() + "\n";
```

Rows კოლექციას აქვს Count თვისება, რომელშიც ავტომატურად იწერება Personali ცხრილში არსებული სტრიქონების რაოდენობა. ამ ცხრილში ახალი სტრიქონის დამატების შემდეგ, ეს მნიშვნელობა ერთით გაიზრდება.

შემდეგ, იქმნება DataRow ტიპის myRow ობიექტი myDataset ობიექტის NewRow() მეთოდის გამოყენებით. myRow ობიექტის სტრუქტურა ემთხვევა Personali ცხრილის სტრუქტურას ანუ შეიცავს ამ ცხრილის მხოლოდ იმ სვეტებს, რომლებიც ადაპტერში განსაზღვრულ SELECT ბრძანებაშია მოთავსებული. ეს სვეტები ცარიელია და მათ უნდა მივანიჭოთ კონკრეტული მნიშვნელობები. რადგან, ცხრილის თითოეულ სვეტს აქვს object ტიპი, ამიტომ, მაგალითად "staji" სვეტს მნიშვნელობა შეგვიძლია მივანიჭოთ ან ასე:

```
myRow["staji"] = textBox1.Text;
```

ან ასე:

```
myRow["xelfasi"] = Convert.ToDouble(textBox1.Text);
```

შედეგი ერთი და იგივე იქნება.

როცა ყველა სვეტს მივანიჭებთ საჭირო მნიშვნელობებს, მხოლოდ ამის შემდეგ უნდა დავუმატოთ myRow ობიექტი Personali ცხრილს Rows კოლექციის Add() მეთოდის გამოყენებით: myDataset.Tables["Personali"].Rows.Add(myRow);

ამის შემდეგ, ვასრულებთ ადაპტერის Update() მეთოდს ცვლილებების შესანახად სერვერზე მოთავსებულ მონაცემთა ბაზაში.

პროგრამაში მნიშვნელობები მიენიჭა Personali ცხრილის personaliID, gvari, xelfasi, staji, tarigi_dabadebis სვეტებს. ამ ცხრილის დანარჩენ სვეტებში ჩაიწერება null მნიშვნელობა. იმისათვის, რომ სხვა სვეტებსაც მივანიჭოთ მნიშვნელობები საჭიროა, რომ ეს სვეტები მივუთითოთ SELECT მოთხოვნაში, რომელიც ეთითება ადაპტერის შექმნისას. მაგალითად, SELECT მოთხოვნას დავუმატოთ ganyofileba სვეტი:

```
SqlDataAdapter myAdapter = new SqlDataAdapter(
"SELECT personaliID, gvari, xelfasi, staji, tarigi_dabadebis, ganyofileba FROM Personali",
myConnection);
```

ამის შემდეგ, შეგვიძლია აღნიშნულ სვეტს მნიშვნელობა მივანიჭოთ:

```
myRow["ganyofileba"] = Convert.ToDouble(textBox1.Text);
```

ეს იმას ნიშნავს, რომ ჩვენ მნიშვნელობები შეგვიძლია მივანიჭოთ მხოლოდ იმ სვეტებს, რომლებიც მოთავსებულია SELECT მოთხოვნაში ობიექტი-ადაპტერის შექმნისას. ზოგად შემთხვევაში, უნდა მივუთითოთ ის სვეტები, რომლებთანაც მუშაობას ვაპირებთ. თუ სვეტების სახელების ნაცვლად მივუთითებთ '*' სიმბოლოს, მაშინ აირჩევა ყველა სვეტი.

სტრიქონების წაშლა

ახლა ვნახთ თუ როგორ ხდება ცხრილიდან სტრიქონის წაშლა. ამისთვის გამოიყენება DataRow ობიექტის Delete() მეთოდი. მოყვანილ პროგრამაში, ჯერ იძებნება წასაშლელი სტრიქონი, შემდეგ კი ხდება მისი წაშლა. Delete() მეთოდის შემდეგ აუცილებელია Update()

მეთოდის შესრულება. საქმე ის არის, რომ Delete() მეთოდი არ შლის სტრიქონს, არამედ მონიშნავს მას შემდგომი წაშლისთვის. Update() მეთოდი კი ახდენს ცვლილებების რეალურად ფიქსირებას, ჩვენს შემთხვევაში, წასაშლელად მონიშნული სტრიქონის წაშლას.

```

{
//      პროგრამა 17.11
//      პროგრამით ხდება Personali ცხრილიდან სტრიქონის წაშლა
//      find_key ცვლადში უნდა მოვათავსოთ საძებნი მნიშვნელობა
int find_key = Convert.ToInt32(textBox1.Text);
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection =
            new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
myConnection.Open();
//      ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter(
            "SELECT personaliID, gvari, xelfasi, staji, tarigi_dabadebis FROM Personali", myConnection);
//      SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
//      DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//      myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
//      პირველადი გასაღების ფორმირება
DataColumn[] keys = new DataColumn[1];
keys[0] = myDataset.Tables["Personali"].Columns["PersonaliID"];
myDataset.Tables["Personali"].PrimaryKey = keys;
//      findRow ობიექტს ენიჭება ნაპოვნი სტრიქონი
DataRow findRow = myDataset.Tables["Personali"].Rows.Find(find_key);
//      მოწმდება სტრიქონის არსებობა
if ( findRow != null )
{
            //      სტრიქონის წაშლა
            findRow.Delete();
            myAdapter.Update(myDataset, "Personali");
            label1.Text = "სტრიქონი წაიშალა";
}
else label1.Text = "წასაშლელი სტრიქონი ვერ მოიძებნა";
//      შეერთების დახურვა
myConnection.Close();
}

```

რადგან, ხშირ შემთხვევაში, პირველადი გასაღებების ცოდნა შეუძლებელია, ამიტომ შეგვიძლია სტრიქონი მოვძებნოთ თანამშრომლის გვარის მიხედვით. ამისათვის, გამოვიყენებთ BindingSource ობიექტის Find() მეთოდს. მისი პირველი პარამეტრია იმ სვეტის სახელი, რომელშიც უნდა შესრულდეს ძებნა, მეორე პარამეტრია საძებნი სიდიდე. თუ საძებნი სიდიდე მოიძებნა, მაშინ გაიცემა ნაპოვნი სტრიქონის ინდექსი, წინააღმდეგ შემთხვევაში კი - -1. მოცემული პროგრამით ხდება Personali ცხრილიდან სტრიქონის წაშლის დემონსტრირება:

```

{

```

```

//      პროგრამა 17.12
//      პროგრამით ხდება PersonalI ცხრილიდან სტრიქონის წაშლა
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection =
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
myConnection.Open();
//      ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter(
    "SELECT personaliID, gvari, xelfasi, staji, tarigi_dabadebis FROM PersonalI", myConnection);
//      SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
//      DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//      myDataset ობიექტის შევსება PersonalI ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "PersonalI");
//      bind ობიექტის დაკავშირება PersonalI ცხრილიდან
BindingSource bind = new BindingSource(myDataset, myDataset.Tables["PersonalI"].ToString());
//      index ცვლადში იწერება მოძებნილი სტრიქონის ნომერი
int index = bind.Find("gvari", textBox1.Text);
label1.Text = index.ToString() + "\n";
//      პირველადი გასაღების ფორმირება
DataColumn[] keys = new DataColumn[1];
keys[0] = myDataset.Tables["PersonalI"].Columns["PersonalIID"];
myDataset.Tables["PersonalI"].PrimaryKey = keys;
if ( index != -1 )
{
//      ნაპოვნი სტრიქონის მინიჭება findRow ობიექტისთვის
DataRow findRow = myDataset.Tables["PersonalI"].Rows[index];
label1.Text += findRow["PersonalIID"].ToString() + " " + findRow["gvari"];
//      მოწმდება სტრიქონის არსებობა
if ( findRow != null )
{
//      სტრიქონის წაშლა
findRow.Delete();
myAdapter.Update(myDataset, "PersonalI");
label1.Text += " სტრიქონი წაიშალა";
}
else label1.Text += "წასაშლელი სტრიქონი ვერ მოიძებნა";
}
//      შეერთების დახურვა
myConnection.Close();
}

```

ორ ცხრილს შორის ბმის შექმნა

ორ ცხრილს შორის ბმის შესაქმნელად გამოიყენება DataRelation ობიექტი. ის კავშირს

ამყარებს DataTable ობიექტებს შორის. ბმა დავამყაროთ Personali და Shemkveti ცხრილებს შორის. Personali ცხრილში მოთავსებულია მონაცემები ფორმის თანამშრომლების შესახებ, Shemkveti ცხრილში კი - შემკვეთების შესახებ. თითოეულ თანამშრომელს შეიძლება ჰყავდეს ერთი ან მეტი შემკვეთი. მთავარია Personali ცხრილი. მისი პირველადი გასაღებია personaliID სვეტი. Shemkveti ცხრილი დამოკიდებულია. მისი გარე გასაღებია personaliID სვეტი. ამ ცხრილებს შორის კავშირის დასამყარებლად Personali ცხრილის პირველადი გასაღები უნდა დავუკავშიროთ Shemkveti ცხრილის გარე გასაღებს.

DataRelation ობიექტის შესაქმნელად გამოიყენება Relations ობიექტის Add() მეთოდი:

```
DataRelation personaliShemkvetiRelation = myDataset.Relations.Add("PersonaliShemkveti",
    myDataset.Tables["Personali"].Columns["PersonaliID"],
    myDataset.Tables["Shemkveti"].Columns["PersonaliID"]);
```

Add() მეთოდის პირველი პარამეტრი ბმის სახელია, მეორე - პირველადი გასაღები, მესამე კი - გარე გასაღები.

მოყვანილი პროგრამით ხდება ორ ცხრილს შორის კავშირის დამყარება:

```
{
//    პროგრამა 17.13
//    Personali და Shemkveti ცხრილებს შორის კავშირის დამყარება
label1.Text = "";
//    შერტების ობიექტის შექმნა
SqlConnection myConnection = new
    SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//    შერტების გახსნა
myConnection.Open();
//    ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter(
    "SELECT personaliID, gvari, xelfasi, tarigi_dabadebis FROM Personali", myConnection);
//    SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
//    DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//    DataAdapter ობიექტების შექმნა თითოეული ცხრილისთვის და მათი შევსება
SqlDataAdapter personaliAdapter = new SqlDataAdapter("SELECT * FROM Personali", myConnection);
SqlDataAdapter shemkvetiAdapter = new
    SqlDataAdapter("SELECT * FROM Shemkveti", myConnection);
//    myDataset ობიექტის შევსება Personali და Shemkveti ცხრილებით
personaliAdapter.Fill(myDataset, "Personali");
shemkvetiAdapter.Fill(myDataset, "Shemkveti");
//    კავშირის დამყარება ცხრილებს შორის
DataRelation personaliShemkvetiRelation = myDataset.Relations.Add("PersonaliShemkveti",
    myDataset.Tables["Personali"].Columns["PersonaliID"],
    myDataset.Tables["Shemkveti"].Columns["PersonaliID"]);
//    ეკრანზე სტრიქონების გამოტანა
foreach ( DataRow personaliRow in myDataset.Tables["Personali"].Rows )
{
label1.Text += personaliRow["PersonaliID"].ToString() + " " + personaliRow["gvari"] + "\n";
    foreach ( DataRow shemkvetiRow in personaliRow.GetChildRows(personaliShemkvetiRelation) )
        label1.Text += " " + shemkvetiRow["ShemkvetiID"].ToString() + " " + shemkvetiRow["gvari"] + "\n";
}
}
```

```

}
// შერტების დახურვა
myConnection.Close();
}

```

როგორც პროგრამიდან ჩანს, ორივე ცხრილისთვის იქმნება შესაბამისი ადაპტერები: `personaliAdapter` და `shemkvetiAdapter`. შემდეგ ხდება `myDataset` ობიექტის შევსება ორივე ცხრილით. ცხრილებს შორის ბმის დასამყარებლად იქმნება `personaliShemkvetiRelation` ობიექტი. გარე `foreach` ციკლს `label` კომპონენტში გამოაქვს `Personali` ცხრილის სტრიქონები, შიდა `foreach` ციკლს კი `Shemkveti` ცხრილის სტრიქონები. იმისათვის, რომ `Personali` ცხრილის თითოეული სტრიქონისთვის გამოვიტანოთ შესაბამისი ბმული სტრიქონები `Shemkveti` ცხრილიდან, უნდა გამოვიყენოთ `DataRow` ობიექტის `GetChildRows()` მეთოდი, რომელსაც პარამეტრად უნდა გადავცეთ `personaliShemkvetiRelation` ობიექტი.

ორ ცხრილს შორის ბმა შეგვიძლია `dataGridView` კომპონენტებში ავსახოთ. მოყვანილი პროგრამით ხდება `Personali` და `Shemkveti` ცხრილებს შორის ბმის დემონსტრირება.

```

{
// შერტების ობიექტის შექმნა
SqlConnection myConnection = new
SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=romaroma");
// შერტების გახსნა
myConnection.Open();
// ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter(
"SELECT personaliID, gvari, xelfasi, tarigi_dabadebis FROM Personali", myConnection);
// SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// DataAdapter ობიექტების შექმნა თითოეული ცხრილისთვის და მათი შევსება
SqlDataAdapter personaliAdapter = new SqlDataAdapter("SELECT * FROM Personali",
myConnection);
SqlDataAdapter shemkvetiAdapter = new SqlDataAdapter("SELECT * FROM Shemkveti",
myConnection);
// myDataset ობიექტის შევსება Personali და Shemkveti ცხრილებით
personaliAdapter.Fill(myDataset, "Personali");
shemkvetiAdapter.Fill(myDataset, "Shemkveti");
// კავშირის დამყარება ცხრილებს შორის
DataRelation personaliShemkvetiRelation = myDataset.Relations.Add("PersonaliShemkveti",
myDataset.Tables["Personali"].Columns["PersonaliID"],
myDataset.Tables["Shemkveti"].Columns["PersonaliID"]);
// ეკრანზე სტრიქონების გამოტანა
//myDataset.Relations.Add(personaliShemkvetiRelation);
// შერტების დახურვა
myConnection.Close();
BindingSource bindingSource4 = new BindingSource(myDataset, "Personali");
BindingSource bindingSource5 = new BindingSource(myDataset, "Shemkveti");
bindingSource5.DataSource = bindingSource4;
bindingSource5.DataMember = "PersonaliShemkveti";
}

```

```

dataGridView4.DataSource = bindingSource4;
dataGridView5.DataSource = bindingSource5;
}
}
    ზმის გასაუქმებლად შეგვიძლია გამოვიყენოთ კოდი:
{
string connectionString = "server=ROMANI-PC; database=Shekveta; uid=sa; pwd=romaroma";
SqlConnection mySqlConnection = new SqlConnection(connectionString);
String selectString = "SELECT * FROM Personali ";
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
mySqlCommand.CommandText = selectString;
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
DataSet myDataSet = new DataSet();
mySqlConnection.Open();
//
mySqlDataAdapter.Fill(myDataSet, "Personali");
//
selectString = "SELECT * FROM Shemkveti ";
mySqlCommand.CommandText = selectString;
mySqlDataAdapter.SelectCommand = mySqlCommand;
mySqlDataAdapter.Fill(myDataSet, "Shemkveti");
//
/*+dataGridView4.DataSource = myDataSet;
dataGridView4.DataMember = "Personali";
dataGridView5.DataSource = myDataSet;
dataGridView5.DataMember = "Shemkveti";*/
// შეერთების დახურვა
mySqlConnection.Close();
//
BindingSource bindingSource4 = new BindingSource(myDataSet, "Personali");
BindingSource bindingSource5 = new BindingSource(myDataSet, "Shemkveti");
//
dataGridView4.DataSource = bindingSource4;
dataGridView5.DataSource = bindingSource5;
}
}

```

ამ პროგრამაში ცხრილებს შორის ზმის გასაუქმებლად შეგვიძლია

```

BindingSource bindingSource4 = new BindingSource(myDataSet, "Personali");
BindingSource bindingSource5 = new BindingSource(myDataSet, "Shemkveti");
//
dataGridView4.DataSource = bindingSource4;
dataGridView5.DataSource = bindingSource5;
კოდის ნაცვლად გამოვიყენოთ კოდი:
dataGridView4.DataSource = myDataSet;
dataGridView4.DataMember = "Personali";
dataGridView5.DataSource = myDataSet;
dataGridView5.DataMember = "Shemkveti";

```


SQL ბრძანებების უშუალო შესრულება

თუ საჭიროა ოპერაციების შესრულება სტრიქონების ჯგუფზე, მაგალითად, სტრიქონების წაშლა ან სტრიქონებში სვეტების მნიშვნელობების შეცვლა, მაშინ დიდი ცხრილებისთვის გაცილებით ეფექტური იქნება ამის გაკეთება ერთი SQL-ბრძანებით. SQL-ბრძანების შესასრულებლად შეგვიძლია გამოვიყენოთ SqlCommand და OleDbCommand კლასების ობიექტები. ამ ობიექტებს აქვს მეთოდები, რომლებიც SQL-ბრძანებებს უშუალოდ ასრულებენ.

ერთი მნიშვნელობის მიღება

როცა საჭიროა SQL მოთხოვნისაგან ერთი შედეგის მიღება, მაგალითად პერსონალის საშუალო ასაკი, მაქსიმალური ხელფასი და ა.შ. მაშინ უმჯობესია SqlCommand ობიექტის ExecuteScalar() მეთოდის გამოყენება. ის გასცემს სკალარულ მნიშვნელობას. მოყვანილი პროგრამის შესრულებით გაიცემა ფორმაში მომუშავე თანამშრომლების რაოდენობა:

```
{
//   პროგრამა 17.14
//   პროგრამით გაიცემა Personalი ცხრილის სტრიქონების რაოდენობა
//   შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
    SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//   შეერთების გახსნა
myConnection.Open();
//   myCommand ობიექტის შექმნა მიმდინარე შეერთებისთვის
SqlCommand myCommand = myConnection.CreateCommand();
//   SELECT ბრძანების ფორმირება
myCommand.CommandText = "SELECT COUNT(*) FROM Personal";
//   მიღებული შედეგის მოთავსება countResult ობიექტში
Object countResult = myCommand.ExecuteScalar();
//   შედეგის ეკრანზე გამოტანა
label1.Text = countResult.ToString();
//   შეერთების დახურვა
myConnection.Close();
}
```

პროგრამაში იქმნება myCommand ობიექტი, რომლის CommandText თვისებას ვანიჭებთ შესასრულებელ SELECT მოთხოვნას. ამ მოთხოვნის შესასრულებლად უნდა გამოვიძახოთ myCommand ობიექტის ExecuteScalar() მეთოდი. მისი შესრულების შედეგად გაიცემა ფორმის თანამშრომლების რაოდენობა.

თუ გვინტერესებს xelfasi სვეტის მაქსიმალური მნიშვნელობა, myCommand ობიექტის CommandText თვისებას უნდა მივანიჭოთ შესაბამისი SELECT ბრძანება:

```
myCommand.CommandText = "SELECT MAX(xelfasi) FROM Personal";
```

თუ გვინტერესებს იმ თანამშრომლების, საშუალო ასაკი, რომლებიც სამედიცინო განყოფილებაში მუშაობენ, SELECT მოთხოვნა უნდა ჩავწეროთ შემდეგნაირად:

```
myCommand.CommandText =
    "SELECT AVG(asaki) FROM Personal WHERE ganyofileba = N'სამედიცინო';
```

UPDATE მოთხოვნის უშუალოდ შესრულება

როგორც აღვნიშნეთ INSERT, UPDATE და DELETE ბრძანებები ახდენენ ცხრილში მონაცემების შეცვლას. მათ შესასრულებლად გამოიყენება SqlCommand ობიექტის ExecuteNonQuery() მეთოდი, რომელიც, აგრეთვე გასცემს ამ ბრძანებებით შეცვლილი სტრიქონების რაოდენობას. მოყვანილი პროგრამით ხდება ფორმის თანამშრომლების ხელფასის გაზრდა 10%-ით.

```
{
//   პროგრამა 17.15
//   პროგრამით ხდება UPDATE ბრძანების უშუალოდ შესრულება
//   შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
        SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//   შეერთების გახსნა
myConnection.Open();
//   myCommand ობიექტის შექმნა
SqlCommand myCommand = myConnection.CreateCommand();
//   UPDATE ბრძანების მომზადება შესასრულებლად
myCommand.CommandText = "UPDATE Personal SET xelfasi = xelfasi * 1.1";
//   UPDATE ბრძანების უშუალოდ შესრულება
int rowsAffected = myCommand.ExecuteNonQuery();
//   დამუშავებული სტრიქონების რაოდენობის ეკრანზე გამოტანა
label1.Text = rowsAffected.ToString();
//   შეერთების დახურვა
myConnection.Close();
}
```

პროგრამაში იქმნება myCommand ობიექტი, რომლის CommandText თვისებას ენიჭება შესასრულებელი UPDATE ბრძანება. ამ ბრძანებას ასრულებს ამ ობიექტის ExecuteNonQuery() მეთოდი, რომელიც გასცემს შეცვლილი სტრიქონების რაოდენობას. ეს რაოდენობა მიენიჭება rowsAffected ცვლადს. ცვლილებების სანახავად ფორმაზე მოვათავსოთ Button კომპონენტი და მას მივაბათ პროგრამა 17.3.

თუ UPDATE ბრძანებას დავუმატებთ WHERE განყოფილებას, მაშინ შეგვეძლება რამდენიმე სტრიქონში მონაცემების შეცვლა. მაგალითად, სამედიცინო განყოფილების პერსონალისთვის ხელფასის გასაზრდელად 10%-ით UPDATE ბრძანება ასე უნდა შევიტანოთ:

```
myCommand.CommandText =
        "UPDATE Personal SET xelfasi = xelfasi * 1.1 WHERE ganyofileba = N'სამედიცინო'";
```

ახლა განვიხილოთ შემთხვევა, როცა UPDATE ბრძანებას გადაეცემა პარამეტრები. მოყვანილი პროგრამით ხდება Shekveta მონაცემთა ბაზის Personal ცხრილის იმ სტრიქონში მონაცემების შეცვლა, რომლის პირველადი გასაღების მნიშვნელობა დაემთხვევა პარამეტრად გადაცემულ მნიშვნელობას:

```
{
//   პროგრამა 17.16
//   პროგრამაში ხდება UPDATE მოთხოვნის შესრულების დემონსტრირება
//   შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
        SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//   შეერთების გახსნა
myConnection.Open();
//   mySqlCommand ობიექტის შექმნა
```

```

SqlCommand mySqlCommand = myConnection.CreateCommand();
//      UPDATE ბრძანების მომზადება შესასრულებლად
mySqlCommand.CommandText = "UPDATE PersonalI SET gvari = @gvari, ganyofileba = @ganyofileba,
qalaqi = @qalaqi, " + "xelfasi = @xelfasi, asaki = @asaki, staji = @staji, tarigi_dabadebis = @tarigi_dabadebis
" + "WHERE (personaliID = @personaliID)";
//      პარამეტრების დამატება
mySqlCommand.Parameters.Add("@personaliID", SqlDbType.Int, 4);
mySqlCommand.Parameters.Add("@gvari", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@qalaqi", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@xelfasi", SqlDbType.Float, 8);
mySqlCommand.Parameters.Add("@asaki", SqlDbType.Int, 4);
mySqlCommand.Parameters.Add("@staji", SqlDbType.Int, 4);
mySqlCommand.Parameters.Add("@tarigi_dabadebis", SqlDbType.DateTime, 8);
//      პარამეტრებისთვის მნიშვნელობების მინიჭება
mySqlCommand.Parameters["@personaliID"].Value = Convert.ToInt32(textBox7.Text);
mySqlCommand.Parameters["@gvari"].Value = textBox1.Text;
mySqlCommand.Parameters["@ganyofileba"].Value = textBox2.Text;
mySqlCommand.Parameters["@qalaqi"].Value = textBox3.Text;
mySqlCommand.Parameters["@xelfasi"].Value = Convert.ToDouble(textBox4.Text);
mySqlCommand.Parameters["@asaki"].Value = Convert.ToInt32(textBox5.Text);
mySqlCommand.Parameters["@staji"].Value = Convert.ToInt32(textBox6.Text);
mySqlCommand.Parameters["@tarigi_dabadebis"].Value = dateTimePicker1.Value;
//      UPDATE ბრძანების შესრულება
int rowsAffected = mySqlCommand.ExecuteNonQuery();
//      დამუშავებული სტრიქონების რაოდენობის გამოტანა ეკრანზე
label1.Text = rowsAffected.ToString();
//      შეერთების დახურვა
myConnection.Close();
}

```

როგორც ვხედავთ, პარამეტრის დამატების დროს პარამეტრის სახელის გარდა მითითებულია მისი ტიპიც (ცხრილი 17.1).

შესაცვლელი სტრიქონი შეგვიძლია მოვძებნოთ bindingSource1 კომპონენტის Find() ბრძანების საშუალებით და გამოვიყენოთ ნაპოვნი სტრიქონის პირველადი გასაღები. პროგრამას ექნება შემდეგი სახე:

```

{
//      პროგრამა 17.17
//      პროგრამაში ხდება UPDATE მოთხოვნის შესრულების დემონსტრირება
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection =
        new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=romaroma");
//      შეერთების გახსნა
myConnection.Open();
//      ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM PersonalI", myConnection);
//      SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);

```

```

//      DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//      myDataset ობიექტის შევსება Personalი ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personalი");
//      personaliBind ობიექტის დაკავშირება Personalი ცხრილთან
BindingSource personaliBind = new BindingSource(myDataset, "Personalი");
//      ნაპოვნი სტრიქონის ინდექსის მიღება
int index = personaliBind.Find("gvari", textBox1.Text);
if (index != -1)
{
//      MySqlCommand ობიექტის შექმნა
SqlCommand mySqlCommand = myConnection.CreateCommand();
//      UPDATE ბრძანების მომზადება შესასრულებლად
mySqlCommand.CommandText =
"UPDATE Personalი SET gvari = @gvari, ganyofileba = @ganyofileba, qalaqi = @qalaqi, " +
"xelfasi = @xelfasi, asaki = @asaki, staji = @staji, tarigi_dabadebis = @tarigi_dabadebis " +
"WHERE (personaliID = @personaliID)";
//      პარამეტრების დამატება
mySqlCommand.Parameters.Add("@personaliID", SqlDbType.Int, 4);
mySqlCommand.Parameters.Add("@gvari", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@qalaqi", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@xelfasi", SqlDbType.Float, 8);
mySqlCommand.Parameters.Add("@asaki", SqlDbType.Int, 4);
mySqlCommand.Parameters.Add("@staji", SqlDbType.Int, 4);
mySqlCommand.Parameters.Add("@tarigi_dabadebis", SqlDbType.DateTime, 8);
//      პარამეტრებისთვის მნიშვნელობების მინიჭება
mySqlCommand.Parameters["@personaliID"].Value =
Convert.ToInt32(myDataset.Tables["Personalი"].Rows[index][0]);
mySqlCommand.Parameters["@gvari"].Value = textBox1.Text;
mySqlCommand.Parameters["@ganyofileba"].Value = textBox2.Text;
mySqlCommand.Parameters["@qalaqi"].Value = textBox3.Text;
mySqlCommand.Parameters["@xelfasi"].Value = Convert.ToDouble(textBox4.Text);
mySqlCommand.Parameters["@asaki"].Value = Convert.ToInt32(textBox5.Text);
mySqlCommand.Parameters["@staji"].Value = Convert.ToInt32(textBox6.Text);
mySqlCommand.Parameters["@tarigi_dabadebis"].Value = dateTimePicker1.Value;
//      UPDATE ბრძანების შესრულება
int rowsAffected = mySqlCommand.ExecuteNonQuery();
label3.Text = rowsAffected.ToString();
}
else label3.Text = "სტრიქონი არ არსებობს";
//      შეერთების დახურვა
myConnection.Close();
}

```

შესაძლებელია, აგრეთვე რამდენიმე სტრიქონის შეცვლა. მოყვანილ პროგრამაში ხდება მითითებული განყოფილების თანამშრომლების ხელფასის გაზრდა მითითებული თანხით. განყოფილებისა და ხელფასის ნაზრდის ზომა შეგვიძლია textBox კომპონენტებიდან შევიტანოთ.

```

{
//      პროგრამა 17.18
//      პროგრამაში ხდება UPDATE მოთხოვნის შესრულების დემონსტრირება
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection =
        new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
myConnection.Open();
//      mySqlCommand ობიექტის შექმნა
SqlCommand mySqlCommand = myConnection.CreateCommand();
//      UPDATE ბრძანების მომზადება შესასრულებლად
mySqlCommand.CommandText =
        "UPDATE Personal SET xelfasi = xelfasi + @xelfasi WHERE (ganyofileba = @ganyofileba)";
//      პარამეტრების დამატება
mySqlCommand.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@xelfasi", SqlDbType.Float, 8);
//      პარამეტრებისთვის მნიშვნელობების მინიჭება
mySqlCommand.Parameters["@xelfasi"].Value = Convert.ToDouble(textBox4.Text);
mySqlCommand.Parameters["@ganyofileba"].Value = textBox2.Text;
//      UPDATE ბრძანების შესრულება
int rowsAffected = mySqlCommand.ExecuteNonQuery();
label1.Text = rowsAffected.ToString();
//      შეერთების დახურვა
myConnection.Close();
}

```

ცხრილი 17.1. SqlDbTypeType ჩამოთვლადი ტიპის ელემენტები.

ელემენტი	აღწერა
BigInt	64-ბიტური ნიშნის მთელი რიცხვი, რომელიც იღებს მნიშვნელობას -263÷263-1 (-9223372036854775808÷9223372036854775807)
Binary	ბაიტების მასივი სიგრძით 8000 ელემენტამდე
Bit	უნიშნო მთელი რიცხვი, რომელიც იღებს მნიშვნელობებს 0, 1 ან null
Char	სტრიქონი სიგრძით 8000 სიმბოლომდე, რომელიც არ შეიცავს Unicode სიმბოლოებს
DateTime	დრო და თარიღი 1753 წლის 1 იანვრიდან 9999 წლის 31 დეკემბრამდე
Decimal	რიცხვი მითითებული სიზუსტით და მასშტაბით მნიშვნელობით -10 ³⁸ -1÷10 ³⁸ -1 (-79228162514264337593543950335÷79228162514264337593543950335)
Float	მცურავწერტილიანი რიცხვი დიაპაზონში -1.79E+308÷1.79E+308
Image	ბაიტების მასივი სიგრძით 2 ³¹ -1 (2147483647) ელემენტამდე
Int	32-ბიტური მთელი რიცხვი ნიშნით, დიაპაზონში -2 ³¹ ÷2 ³¹ -1 (-2147483648÷2147483647)
Money	ფული დიაპაზონში -922337203685477.5808÷922337203685477.5807
NChar	Unicode სიმბოლოების სტრიქონი მაქსიმალური სიგრძით 4000 სიმბოლომდე
NText	Unicode სიმბოლოების სტრიქონი მაქსიმალური სიგრძით 2 ³⁰ -1 (1073741823) სიმბოლომდე
NVarChar	Unicode სიმბოლოების სტრიქონი მაქსიმალური სიგრძით 4000 სიმბოლომდე
Real	მცურავწერტილიანი რიცხვი დიაპაზონში -3.40E+38÷3.40E+38
SmallDateTime	დრო და თარიღი 1900 წლის 1 იანვრიდან 2079 წლის 6 ივნისამდე
SmallInt	16-ბიტური მთელი რიცხვი ნიშნით
SmallMoney	ფული დიაპაზონში -214748.3648÷214748.3647
Text	სტრიქონი მაქსიმალური სიგრძით 2 ³¹ -1 (2147483647) სიმბოლომდე, რომელიც არ შეიცავს Unicode სიმბოლოებს
Timestamp	თარიღი და დრო ფორმატში yyyyymmddhhmmss (წელი/თვე/დღე/საათი/წუთები/წამები)
TinyInt	8-ბიტური უნიშნო მთელი რიცხვი დიაპაზონში 0÷2 ⁸ -1 (255)
UniqueIdentifier	GUID (Globally Unique Identifier - გლობალური უნიკალური იდენტიფიკატორი)
VarBinary	ბაიტების მასივი მაქსიმალური სიგრძით 8000 ელემენტამდე
VarChar	სტრიქონი მაქსიმალური სიგრძით 8000 სიმბოლომდე, რომელიც არ შეიცავს Unicode სიმბოლოებს
Variant	მონაცემების ტიპი, რომელიც შეიძლება შეიცავდეს რიცხვებს, სტრიქონებს, ბაიტებსა და თარიღებს

INSERT მოთხოვნის უშუალოდ შესრულება

Shekveta მონაცემთა ბაზის Personali ცხრილს შეგვიძლია დავუმატოთ ახალი სტრიქონი INSERT მოთხოვნის უშუალოდ შესრულების გზით. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
{
//   პროგრამა 17.19
//   პროგრამაში ხდება INSERT მოთხოვნის შესრულების დემონსტრირება
//   შეერთების ობიექტის შექმნა
SqlConnection myConnection =
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//   შეერთების გახსნა
myConnection.Open();
//   mySqlCommand ობიექტის შექმნა
SqlCommand mySqlCommand = myConnection.CreateCommand();
//   INSERT ბრძანების მომზადება შესასრულებლად
mySqlCommand.CommandText =
    "INSERT INTO Personali (gvari, ganyofileba, qalaqi, xelfasi, asaki, staji, tarigi_dabadebis) " +
    "VALUES (@gvari, @ganyofileba, @qalaqi, @xelfasi, @asaki, @staji, @tarigi_dabadebis)";
//   პარამეტრების დამატება
mySqlCommand.Parameters.Add("@gvari", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@qalaqi", SqlDbType.NVarChar, 20);
mySqlCommand.Parameters.Add("@xelfasi", SqlDbType.Float, 8);
mySqlCommand.Parameters.Add("@asaki", SqlDbType.Int, 4);
mySqlCommand.Parameters.Add("@staji", SqlDbType.Int, 4);
mySqlCommand.Parameters.Add("@tarigi_dabadebis", SqlDbType.DateTime, 8);
//   პარამეტრებისთვის მნიშვნელობების მინიჭება
mySqlCommand.Parameters["@gvari"].Value = textBox1.Text;
mySqlCommand.Parameters["@ganyofileba"].Value = textBox2.Text;
mySqlCommand.Parameters["@qalaqi"].Value = textBox3.Text;
mySqlCommand.Parameters["@xelfasi"].Value = Convert.ToDouble(textBox4.Text);
mySqlCommand.Parameters["@asaki"].Value = Convert.ToInt32(textBox5.Text);
mySqlCommand.Parameters["@staji"].Value = Convert.ToInt32(textBox6.Text);
mySqlCommand.Parameters["@tarigi_dabadebis"].Value = dateTimePicker1.Value;
//   INSERT ბრძანების შესრულება
int rowsAffected = mySqlCommand.ExecuteNonQuery();
label1.Text = rowsAffected.ToString();
//   შეერთების დახურვა
myConnection.Close();
}
```

როგორც ვხედავთ, INSERT მოთხოვნაში არ არის მითითებული Personali ცხრილის პირველადი გასაღები personaliID. ეს იმიტომ, რომ ამ სვეტისთვის Identity თვისება დაყენებულია Yes მდგომარეობაში და სტრიქონის დამატების დროს ავტომატურად ხდება პირველადი გასაღებისთვის სწორი მნიშვნელობის გენერირება.

DELETE მოთხოვნის უშუალოდ შესრულება

Shekveta მონაცემთა ბაზის PersonalI ცხრილიდან წავშალოთ ერთი სტრიქონი DELETE მოთხოვნის უშუალოდ შესრულების გზით. წასაშლელი სტრიქონის პირველადი გასაღების მნიშვნელობა შეგვაქვს textBox კომპონენტიდან. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
{
//      პროგრამა 17.20
//      პროგრამაში ხდება DELETE მოთხოვნის შესრულების დემონსტრირება
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection =
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
myConnection.Open();
//      mySqlCommand ობიექტის შექმნა
SqlCommand mySqlCommand = myConnection.CreateCommand();
//      DELETE ბრძანების მომზადება შესასრულებლად
mySqlCommand.CommandText = "DELETE FROM PersonalI WHERE (PersonalIID = @PersonalIID)";
//      mySqlCommand ობიექტისთვის პარამეტრების დამატება
mySqlCommand.Parameters.Add("@PersonalIID", SqlDbType.Int, 4);
mySqlCommand.Parameters["@PersonalIID"].Value = Convert.ToInt32(textBox1.Text);
//      DELETE ბრძანების შესრულება
int rowsAffected = mySqlCommand.ExecuteNonQuery();
label1.Text = rowsAffected.ToString();
//      შეერთების დახურვა
myConnection.Close();
}
```

წასაშლელი სტრიქონი შეგვიძლია მოვძებნოთ bindingSource1 კომპონენტის Find() ბრძანების საშუალებით და გამოვიყენოთ ნაპოვნი სტრიქონის პირველადი გასაღები. დავუშვათ, PersonalI ცხრილიდან გვინდა წავშალოთ მითითებული გვარის შემცველი სტრიქონი. ამისათვის, ძებნა უნდა შევასრულოთ "gvari" სვეტში, საძებნი გვარი კი შევიტანოთ textBox1 კომპონენტში. ამის დემონსტრირება ხდება მოცემული პროგრამით:

```
{
//      პროგრამა 17.21
//      პროგრამაში ხდება DELETE მოთხოვნის შესრულების დემონსტრირება
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection =
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
myConnection.Open();
//      DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//      ადაპტერის შექმნა და დაკავშირება SELECT მოთხოვნასთან
SqlDataAdapter personalIAdapter = new SqlDataAdapter("SELECT * FROM PersonalI", myConnection);
//      myDataset ობიექტის შევსება PersonalI ცხრილით
personalIAdapter.Fill(myDataset, "PersonalI");
//      bind1 ობიექტის შექმნა და დაკავშირება myDataset ობიექტის PersonalI ცხრილთან
BindingSource bind1 = new BindingSource(myDataset, "PersonalI");
}
```



```

//      ნაპოვნი სტრიქონის ინდექსის მიღება
int index = bind1.Find("gvari", textBox1.Text);
if (index != -1)
{
SqlCommand mySqlCommand = myConnection.CreateCommand();
//      DELETE ბრძანების მომზადება შესასრულებლად
mySqlCommand.CommandText = "DELETE FROM PersonalI WHERE (PersonalIID = @PersonalIID)";
//      mySqlCommand ობიექტისთვის პარამეტრის დამატება
mySqlCommand.Parameters.Add("@PersonalIID", SqlDbType.Int, 4);
mySqlCommand.Parameters["@PersonalIID"].Value =
        Convert.ToInt32(myDataset.Tables["PersonalI"].Rows[index][0]);
//      DELETE ბრძანების შესრულება
int rowsAffected = mySqlCommand.ExecuteNonQuery();
label1.Text = rowsAffected.ToString();
}
//      შეერთების დახურვა
myConnection.Close();
}

```

შესაძლებელია, აგრეთვე რამდენიმე სტრიქონის წაშლა. მოყვანილ პროგრამაში ხდება მონაცემების (სტრიქონების) წაშლა იმ თანამშრომლების შესახებ, რომლებიც დაბადებული არიან მითითებულ თარიღზე ადრე. თარიღის შესატანად შეგვიძლია dateTimePicker კომპონენტის გამოყენება.

```

{
//      პროგრამა 17.22
//      პროგრამაში ხდება DELETE მოთხოვნის შესრულების დემონსტრირება
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
        SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
myConnection.Open();
//      DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//      ადაპტერის შექმნა და მისთვის SELECT მოთხოვნის დაკავშირება
SqlDataAdapter personaliAdapter = new SqlDataAdapter("SELECT * FROM PersonalI", myConnection);
//      myDataset ობიექტის შევსება PersonalI ცხრილით
personaliAdapter.Fill(myDataset, "PersonalI");
//      mySqlCommand ობიექტის შექმნა
SqlCommand mySqlCommand = myConnection.CreateCommand();
//      DELETE ბრძანების მომზადება შესასრულებლად
mySqlCommand.CommandText =
        "DELETE FROM PersonalI WHERE (tarigi_dabadebis < @tarigi_dabadebis)";
//      mySqlCommand ობიექტისთვის პარამეტრის დამატება
mySqlCommand.Parameters.Add("@tarigi_dabadebis", SqlDbType.DateTime, 8);
mySqlCommand.Parameters["@tarigi_dabadebis"].Value = dateTimePicker1.Value;
//      DELETE ბრძანების შესრულება
int rowsAffected = mySqlCommand.ExecuteNonQuery();
label1.Text = rowsAffected.ToString();
}

```

```
// შერტების დახურვა
myConnection.Close();
}
```

ტრანზაქციებთან მუშაობა ADO.NET საშუალებებით

ADO.NET გარემოში ტრანზაქციებთან სამუშაოდ SqlTransaction ობიექტი გამოიყენება. დავუშვათ, გვინდა რომ Personal და Shemkveta ცხრილებს INSERT ბრძანების გამოყენებით თითო სტრიქონი დავუმატოთ და ეს მოქმედებები ერთ ტრანზაქციაში გავაერთიანოთ. ამისათვის, შემდეგი მოქმედებები უნდა შევასრულოთ:

1. SqlTransaction ობიექტის შექმნა და ტრანზაქციის დაწყება SqlConnection ობიექტის BeginTransaction() მეთოდის გამოძახების გზით.
2. SqlCommand ობიექტის შექმნა SQL მოთხოვნის შესანახად.
3. SqlCommand ობიექტის Transaction თვისების დაყენება.
4. პირველი INSERT მოთხოვნის შემცველი სტრიქონის ფორმირება.
5. SqlCommand ობიექტის CommandText თვისებისათვის პირველი INSERT მოთხოვნის შემცველი სტრიქონის მინიჭება.
6. პირველი INSERT მოთხოვნის შესრულება SqlCommand ობიექტის ExecuteNonQuery() მეთოდის საშუალებით.
7. მეორე INSERT მოთხოვნის შემცველი სტრიქონის ფორმირება.
8. SqlCommand ობიექტის CommandText თვისებისათვის მეორე INSERT მოთხოვნის შემცველი სტრიქონის მინიჭება.
9. მეორე INSERT მოთხოვნის შესრულება SqlCommand ობიექტის ExecuteNonQuery() მეთოდის საშუალებით.
10. ტრანზაქციის დაფიქსირება SqlTransaction ობიექტის Commit() მეთოდის გამოყენებით.
ქვემოთ მოყვანილია პროგრამა, რომელშიც აღწერილი მოქმედებები სრულდება:

```
{
// პროგრამა 17.23
// პროგრამაში ხდება ტრანზაქციასთან მუშაობის დემონსტრირება
// შერტების ობიექტის შექმნა
SqlConnection myConnection =
new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd= paroli");
// შერტების გახსნა
myConnection.Open();
// mySqlConnection ობიექტის შექმნა ტრანზაქციის დასაწყებად
SqlTransaction mySqlTransaction = myConnection.BeginTransaction();
// mySqlCommand ობიექტის შექმნა
SqlCommand mySqlCommand = myConnection.CreateCommand();
// mySqlTransaction ობიექტის დაკავშირება mySqlCommand ობიექტთან
mySqlCommand.Transaction = mySqlTransaction;
// INSERT ბრძანების მომზადება შესასრულებლად
mySqlCommand.CommandText =
"INSERT INTO Personal (gvari, ganyofileba, qalaqi, xelfasi, asaki, staji, tarigi_dabadebis) " +
"VALUES (N'კირვალიძე ნინო', N'სასპორტო', N'ონი', 555.55, 50, 5, '2005.11.23')";
// INSERT ბრძანების შესრულება
int rowsAffected = mySqlCommand.ExecuteNonQuery();
```

```

label1.Text = rowsAffected.ToString() + " ";
//      INSERT ბრძანების მომზადება შესასრულებლად
mySqlCommand.CommandText =
"INSERT INTO Shemkveti (gvari, iuridiuli_fizikuri, qalaqi, firmis_dasaxeleba) " +
"VALUES (N'ჭუმბურიძე ნინო', N'იურიდიული', N'ბათუმი', N'ჯეონეთი)";
//      INSERT ბრძანების შესრულება
rowsAffected = mySqlCommand.ExecuteNonQuery();
label1.Text += rowsAffected.ToString();
//      ტრანზაქციის დაფიქსირება
mySqlConnection.Commit();
//      შეერთების დახურვა
mySqlConnection.Close();
}

```

შენახული პროცედურის გამოძახება

SQL-ბრძანებების შესრულების გარდა, შეგვიძლია შესასრულებლად გამოვიძახოთ შენახული პროცედურა და ფუნქცია და საჭიროების შემთხვევაში გადავცეთ პარამეტრები. თუ შენახული პროცედურა გასცემს სკალარულ შედეგს, მაშინ ამ პროცედურის შესასრულებლად უნდა გამოვიყენოთ ExecuteScalar() და ExecuteNonQuery() მეთოდები. თუ პროცედურა გასცემს სტრიქონებს, მაშინ მის შესასრულებლად უნდა გამოვიყენოთ ExecuteReader() მეთოდი.

დავუშვათ, გვინდა შევასრულოთ შენახული პროცედურა, რომელიც გასცემს მითითებული განყოფილების თანამშრომლების შესახებ მონაცემებს. მოყვანილი პროგრამით ხდება ამ შენახული პროცედურის გამოძახება და მისთვის ერთი სტრიქონული ტიპის პარამეტრის გადაცემა, რომელიც შეიცავს განყოფილების სახელს:

```

{
//      პროგრამა 17.24
//      პროგრამით ხდება შენახული პროცედურის შესრულების დემონსტრირება
label1.Text = "";
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
        SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
myConnection.Open();
//      myCommand ობიექტის შექმნა
SqlCommand myCommand = myConnection.CreateCommand();
//      myCommand ობიექტის დაკავშირება Myproc_Par პროცედურასთან
myCommand.CommandType = CommandType.StoredProcedure;
myCommand.CommandText = "Myproc_Par";
//      myCommand ობიექტისთვის პარამეტრის დამატება
myCommand.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 30);
myCommand.Parameters["@ganyofileba"].Value = textBox1.Text;
//      Myproc_Par შენახული პროცედურის შესრულება და
//      მიღებული სტრიქონების ჩაწერა myReader ობიექტში
SqlDataReader myReader = myCommand.ExecuteReader();
//      myReader ობიექტიდან სტრიქონების გამოტანა ეკრანზე

```

```

for ( ; myReader.Read(); )
    label1.Text += myReader["gvari"].ToString() + " " + myReader["qalaqi"] + " " +
        myReader["staji"].ToString() + "\n";
// წამკითხველის დახურვა
myReader.Close();
// შეერთების დახურვა
myConnection.Close();
}

```

როგორც პროგრამიდან ჩანს, Myproc_Par პროცედურის სახელი ეთითება myCommand ობიექტის CommandText თვისებაში. რაც შეეხება პარამეტრს, ის უნდა დავუმატოთ ამავე ობიექტის პარამეტრების კოლექციას Add() მეთოდის გამოყენებით:

```
myCommand.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 30);
```

შემდეგ, ამ პარამეტრს უნდა მივანიჭოთ მნიშვნელობა უშუალოდ ან textBox კომპონენტიდან. ამის შემდეგ, შეგვიძლია შევასრულოთ myCommand ობიექტის ExecuteReader() მეთოდი.

შენახული პროცედურის შესრულება შეიძლება, აგრეთვე ადაპტერის Fill() მეთოდის გამოყენებით. მოყვანილი პროგრამით ხდება შენახული პროცედურის შესრულების დემონსტრირება, რომელსაც ერთი პარამეტრი აქვს. პროცედურას ეკრანზე გამოაქვს მონაცემები მითითებული განყოფილების თანამშრომლების შესახებ:

```

{
// პროგრამა 17.25
// პროგრამაში ხდება Myproc_Par შენახული პროცედურის გამოძახება და მისთვის
// პარამეტრის გადაცემა
// შეერთების შექმნა
SqlConnection mySqlConnection =
    new SqlConnection("server=ROMANI-PC;database=Shekveta;uid=sa;pwd=paroli");
// შეერთების გახსნა
mySqlConnection.Open();
// mySqlCommand ობიექტის შექმნა
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
// mySqlCommand ობიექტისთვის პარამეტრების დამატება
mySqlCommand.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 30);
mySqlCommand.Parameters["@ganyofileba"].Value = textBox1.Text;
// mySqlCommand ობიექტის მომზადება შენახული პროცედურის შესასრულებლად
mySqlCommand.CommandText = "dbo.Myproc_Par";
mySqlCommand.CommandType = CommandType.StoredProcedure;
// ადაპტერის შექმნა და მომზადება სამუშაოდ
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
// myDataSet ობიექტის შექმნა
DataSet myDataSet = new DataSet();
// Myproc_Par შენახული პროცედურის შესრულება
mySqlDataAdapter.Fill(myDataSet, "Myproc_Par");
// შედეგის ეკრანზე გამოტანა
dataGridView1.DataSource = myDataSet;
dataGridView1.DataMember = "Myproc_Par";
// შეერთების დახურვა

```

```
mySqlConnection.Close();
}
```

გამოვიძახოთ შენახული პროცედურა, რომელიც გასცემს მითითებული განყოფილების თანამშრომლების რაოდენობას. ამისათვის გამოვიყენოთ SqlCommand ობიექტის ExecuteScalar() მეთოდი. მოყვანილი პროგრამით ხდება ამის დემონსტრირება:

```
{
//   პროგრამა 17.26
//   პროგრამით ხდება პროცედურის გამოძახების დემონსტრირება ExecuteScalar() მეთოდით
//   შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//   შეერთების გახსნა
myConnection.Open();
//   myCommand ობიექტის შექმნა
SqlCommand myCommand = myConnection.CreateCommand();
//   ბრძანების ტიპის არჩევა
myCommand.CommandType = CommandType.StoredProcedure;
myCommand.CommandText = "Myproc_Par";
//   myCommand ობიექტისთვის პარამეტრის დამატება
myCommand.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 30);
myCommand.Parameters["@ganyofileba"].Value = textBox1.Text;
//   Myproc_Par შენახული პროცედურის შესრულება
Object countResult = myCommand.ExecuteScalar();
//   შედეგის ეკრანზე გამოტანა
label1.Text = countResult.ToString();
//   შეერთების დახურვა
myConnection.Close();
}
```

ფუნქციის გამოძახება

შენახული პროცედურის შესრულების გარდა, შეგვიძლია შესასრულებლად გამოვიძახოთ ფუნქცია და მას პარამეტრები გადავცეთ. შევასრულოთ Scalar ფუნქცია, რომელიც გასცემს მითითებული განყოფილების თანამშრომლების მაქსიმალურ ხელფასს. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
{
//   პროგრამა 17.27
//   პროგრამით ხდება Scalar ფუნქციის გამოძახებისა და
//   მისთვის პარამეტრის გადაცემის დემონსტრირება
//   შეერთების ობიექტის შექმნა Personal ცხრილის სტრიქონების რაოდენობა
SqlConnection myConnection =
    new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//   შეერთების გახსნა
myConnection.Open();
//   myCommand ობიექტის შექმნა მიმდინარე შეერთებისთვის
SqlCommand myCommand = myConnection.CreateCommand();
```

```

myCommand.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 30);
myCommand.Parameters["@ganyofileba"].Value = textBox1.Text;
//      SELECT ბრძანების ფორმირება
myCommand.CommandText = "SELECT dbo.Maqsimaluri_Xelfasi(@ganyofileba)";
//      მიღებული შედეგის მოთავსება countResult ობიექტში
Object countResult = myCommand.ExecuteScalar();
//      შედეგის ეკრანზე გამოტანა
label1.Text = countResult.ToString();
//      შეერთების დახურვა
myConnection.Close();
}

ახლა მოვიყვანოთ Inline ფუნქციის გამოძახების მაგალითი. ფუნქცია გასცემს თითოეული
განყოფილების საშუალო ხელფასს.
{
//      პროგრამა 17.28
//      პროგრამით ხდება Inline ფუნქციის გამოძახების დემონსტრირება
SqlConnection mySqlConnection =
        new SqlConnection("server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli");
//      შეერთების გახსნა
mySqlConnection.Open();
//      mySqlCommand ობიექტის შექმნა
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
//      SELECT მოთხოვნის ფორმირება
mySqlCommand.CommandText = "SELECT * FROM Sashualo_Xelfasi() ";
//      ადაპტერის ფორმირება
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
//      SELECT ბრძანების მომზადება შესასრულებლად
mySqlDataAdapter.SelectCommand = mySqlCommand;
//      Maqsimaluri_Xelfasi ფუნქციის შესრულება
DataSet myDataSet = new DataSet();
//      Maqsimaluri_Xelfasi ფუნქციის შესრულება
mySqlDataAdapter.Fill(myDataSet, "Sashualo_Xelfasi");
//      შედეგის ეკრანზე გამოტანა
dataGridView1.DataSource = myDataSet;
dataGridView1.DataMember = "Sashualo_Xelfasi";
//      შეერთების დახურვა
mySqlConnection.Close();
}

```

თავი 18. ინტეგრირებული მოთხოვნების ენა LINQ

var საკვანძო სიტყვა

var საკვანძო სიტყვა გამოიყენება როგორც ტიპის ალტერნატივა. მისი სინტაქსია:

```
var ცვლადის_სახელი = მნიშვნელობა;
```

აქ ცვლადის_სახელი იღებს მნიშვნელობის ტიპს. მოყვანილი პროგრამით ხდება var საკვანძო სიტყვის გამოყენების დემონსტრირება:

```
{
var cvladi1 = 15;
int cvladi2 = 10;
var cvladi3 = cvladi2;
var cvladi4 = cvladi1 + cvladi2;
var cvladi5 = cvladi1 + 0.5;
label1.Text = cvladi1.ToString() + " " + cvladi3.ToString() + " " + cvladi2.ToString() + " " +
cvladi4.ToString() + " " + cvladi5.ToString();
var masivi1 = new[] { 1, 2, 3, 4, 5 };
}
```

პროგრამაში cvladi1 არის int ტიპის ცვლადი და არა var ტიპის. როცა ვიყენებთ var სიტყვას, ჩვენ არ ვქმნით უტიპო ცვლადს ან ტიპს, რომელიც შეიძლება შეიცვალოს. კომპილატორი თვითონ განსაზღვრავს ცვლადის ტიპს.

```
var cvladi1 = 15;
```

სტრიქონში cvladi1 ცვლადს ენიჭება int ტიპი, რადგან 15 მთელი რიცხვია.

```
int cvladi2 = 10;
```

```
var cvladi3 = cvladi2;
```

ამ მინიჭებების შედეგად, cvladi3 ცვლადს ენიჭება int ტიპი, რადგან cvladi2 ცვლადის ტიპია int. თუ cvladi2 ცვლადს არ მივანიჭებთ საწყის მნიშვნელობას, მაშინ კომპილატორი cvladi3 ცვლადს ვერ მიანიჭებს საწყის მნიშვნელობას და გასცემს შეტყობინებას შეცდომის შესახებ.

```
var cvladi5 = cvladi1 + 0.5;
```

სტრიქონში cvladi5 ცვლადს ენიჭება double ტიპი, რადგან cvladi1 + 0.5 გამოსახულების მნიშვნელობას წილადი ტიპი აქვს.

```
var masivi1 = new[] { 1, 2, 3, 4, 5 };
```

სტრიქონში masivi1 ცვლადს ენიჭება int[] ტიპი. ამ შემთხვევაში, მასივის ყველა ელემენტს ერთნაირი ტიპი უნდა ჰქონდეს, ან ჰქონდეს მიმართვითი ტიპი ან null, ან უნდა შეიძლებოდეს თითოეული მათგანის დაყვანა ერთ ტიპზე. მაგალითად, კომპილატორი შეცდომას გასცემს შემდეგი კოდის კომპილირებისას:

```
var masivi1 = new[] { 5, „რომან სამხარაძე”, 15 }; // „რომან სამხარაძე”-ის ნაცვლად უნდა
// იყოს მთელი რიცხვი
```

```
var masivi2 = new[] { 5, null, 15 }; // null-ის ნაცვლად უნდა იყოს მთელი
// რიცხვი
```

არ არის რეკომენდებული var სიტყვის გამოყენება როგორც კლასის სახელი, რადგან ასეთ შემთხვევაში ის თავის სტანდარტულ ფუნქციას ვეღარ შეასრულებს.

LINQ მოთხოვნის სინტაქსი

ინტეგრირებული მოთხოვნების ენა (LINQ, Language-Integrated Query) არის C# ენის გაფართოება, რომელიც ინტეგრირებულია ამ ენაში. ის არის დიდი მოცულობის მონაცემებთან მუშაობის მოხერხებული, სწრაფი და ეფექტური მექანიზმი. LINQ გვათავისუფლებს მონაცემების გაფილტვრისა და დახარისხების რთული და გრძელი კოდების წერისაგან. ის არის მოთხოვნების ენა, რომელიც საშუალებას გვაძლევს მიღებული შედეგები ადვილად დავახარისხოთ, გავფილტროთ და შევასრულოთ გამოთვლები. LINQ საშუალებას გვაძლევს ეფექტურად ვიმუშაოთ დიდი ზომის მონაცემთა ბაზებთან და კომპლექსურ XML დოკუმენტებთან, რომლებშიც მილიონზე მეტი ჩანაწერია.

LINQ ტექნოლოგია შეგვიძლია გამოვიყენოთ მონაცემთა Object, SQL და XML ტიპების მიმართ:

1. წარმოქმნის მოთხოვნებს, რომლებიც გამოიყენება მასივების, სიებისა და სხვა კოლექციების მიმართ.
2. წარმოქმნის მოთხოვნებს, რომლებიც გამოიყენება მონაცემთა რელაციური ბაზების მიმართ, რომლებიც იყენებენ სტანდარტულ SQL ენას. ასეთია Microsoft SQL Server, Oracle და ა.შ.
3. წარმოქმნის მოთხოვნებს, რომლებიც გამოიყენება XML ობიექტებთან სამუშაოდ.

LINQ მოთხოვნა ოთხი ნაწილისაგან შედგება: var, from, where და select. მისი სინტაქსია:

```
var შედეგი =  
from ცვლადი in კოლექცია  
[ where პირობა ]  
select ცვლადი;
```

var განყოფილებაში მითითებულია ცვლადი, რომელიც შეიცავს მოთხოვნის მიერ გაცემულ შედეგს. from განყოფილებაში ეთითება დასამუშავებელი კოლექცია ან მასივი. კოლექცია არ უნდა შეიცავდეს მხოლოდ ერთ ობიექტს ან ცვლადს. where განყოფილებაში ეთითება პირობა ჩვენი მოთხოვნისათვის. ამ განყოფილების მითითება აუცილებელი არ არის. select განყოფილებაში მითითებული ცვლადის მნიშვნელობა უნდა გამოჩნდეს შედეგობრივ ნაკრებში.

იმისათვის, რომ შევძლოთ LINQ მოთხოვნების შესრულება, using დირექტივების ბლოკში უნდა მოვათავსოთ **using System.Linq**; დირექტივა. ჩვეულებრივ, ამას Visual Studio ავტომატურად აკეთებს.

განვიხილოთ LINQ მოთხოვნის გამოყენების მარტივი მაგალითი. მოყვანილი პროგრამით ხდება სტრიქონების მასივიდან იმ სტრიქონების ამორჩევა, რომლებიც „ს“ ასოთი იწყება. მოცემული პროგრამა დაწერილია LINQ მოთხოვნის სინტაქსის გამოყენებით.

```
// პროგრამა 18.1  
// სტრიქონების მასივიდან იმ სტრიქონების ამორჩევა, რომლებიც „ს“ ასოთი იწყება  
{  
string[] saxelebi = { "რომანი", "ანა", "გიორგი", "არჩილი", "სანდრო", "დავითი", "ვახტანგი", "ბექა",  
"ალექსანდრე", "საბა", "სიმონი", "ლია", "ლუკა" };  
// LINQ მოთხოვნის ფორმირება  
var shedegi =  
from cvladi in saxelebi  
where cvladi.StartsWith("ს")  
select cvladi;  
// შედეგების ეკრანზე გამოტანა  
label1.Text = "სახელები, რომლებიც იწყება „ს“ ასოთი:\n";  
foreach ( var elementi in shedegi )
```



```
label1.Text += elementi.ToString() + "\n";
}
```

როგორც პროგრამიდან ჩანს, LINQ მოთხოვნის მიერ გაცემული სტრიქონები მოთავსდება shedegi ცვლადში. saxelebi მასივიდან ამორჩეული თითოეული სტრიქონი შემოწმდება იწყება თუ არა ის „ს“ ასოთი. თუ კი, მაშინ ის ამორჩევა და მოთავსდება shedegi ცვლადში. შედეგად, label1 კომპონენტში გამოჩნდება ის სტრიქონები, რომლებიც „ს“ ასოთი იწყება.

LINQ მეთოდის სინტაქსი და ლამბდა გამოსახულებები

როგორც აღვნიშნეთ, პროგრამა 18.1 დაიწერა LINQ მოთხოვნის სინტაქსის გამოყენებით. ახლა იგივე პროგრამა დავეწერთ LINQ მეთოდის სინტაქსის გამოყენებით. LINQ არის განხორციელებული როგორც კოლექციების, მასივების, მოთხოვნების შედეგების და სხვა ობიექტების, რომლებიც უზრუნველყოფენ IEnumerable ინტერფეისს, გაფართოებული მეთოდების სერია. LINQ-ს გაფართოებული მეთოდები გამოჩნდება, თუ 18.1 პროგრამაში saxelebi ცვლადის შემდეგ შევიტანთ „.“ წერტილს. გაიხსნება სია, რომელშიც გამოჩნდება LINQ-ს გაფართოებული მეთოდები: Aggregate<>, All<>, Any<>, Average<>, First<>, Last<>, Contains<>, Take<>, Where<>, Reverse<> Sum<> და ა.შ. თუ using System.Linq დირექტივას გავაკომენტარებთ, მაშინ სიაში ეს მეთოდები აღარ გამოჩნდება.

უნდა გვახსოვდეს, რომ მოთხოვნის სინტაქსი უნდა გამოვიყენოთ ყოველთვის, როცა ეს შესაძლებელია, ხოლო მეთოდის სინტაქსი კი მაშინ, როცა ეს აუცილებელია.

LINQ მეთოდების უმრავლესობა, რომლებიც იყენებენ მეთოდის სინტაქსს, ითხოვს, რომ გადავცეთ მეთოდი ან ფუნქცია მოთხოვნის გამოსახულების შესაფასებლად. მეთოდი/ფუნქცია პარამეტრი გადაიცემა დელეგატის ფორმით, რომელიც მიმართავს ანონიმურ მეთოდს. ჩვენ ვქმნით მეთოდს/ფუნქციას C# ენის სპეციალური კონსტრუქციის გამოყენებით, რომელსაც ლამბდა გამოსახულება ეწოდება. მისი საშუალებით ვწერთ ანონიმურ მეთოდს, რომელიც გამოიყენება მოთხოვნის გამოსახულების შესაფასებლად. უმარტივესი ლამბდა გამოსახულების მაგალითია:

```
cvladi => cvladi > 25
```

აქ => არის ლამბდა ოპერატორი. მოცემულ გამოსახულებაში, ლამბდა გამოსახულება განსაზღვრავს ფუნქციას, რომელიც იღებს cvladi პარამეტრს და გასცემს true-ს, თუ cvladi > 25 პირობა სრულდება, წინააღმდეგ შემთხვევაში, გასცემს false მნიშვნელობას. ეს ფუნქცია ანონიმური მეთოდია, რომელსაც სახელი არ აქვს და გამოიყენება როცა ლამბდა გამოსახულება გადაეცემა LINQ მეთოდს.

პროგრამა 18.1 ლამბდა გამოსახულების გამოყენებით ასე ჩაიწერება:

```
// პროგრამა 18.2
// სტრიქონების მასივიდან იმ სტრიქონების ამორჩევა, რომლებიც „ს“ ასოთი იწყება
{
string[] saxelebi = { "რომანი", "ანა", "გიორგი", "არჩილი", "სანდრო", "დავითი", "ვახტანგი",
                    "ბექა", "ალექსანდრე", "საბა", "სიმონი", "ლია", "ლუკა" };
// LINQ მოთხოვნის ფორმირება
var shedegi = saxelebi.Where(cvladi => cvladi.StartsWith("ს"));
// შედეგების ეკრანზე გამოტანა
label1.Text = "სახელები, რომლებიც იწყება 'ს' ასოთი:\n";
foreach ( var elementi in shedegi )
    label1.Text += elementi.ToString() + "\n";
}
```

```
}
```

კომპილატორი cvladi => cvladi.StartsWith("ს") ლამზდა გამოსახულებას გარდაქმნის ანონიმურ მეთოდად, რომელიც სრულდება Where() მეთოდის მიერ saxelebi მასივის თითოეულ ელემენტზე.

მოთხოვნის შედეგების დახარისხება

LINQ მოთხოვნის მიერ გაცემული შედეგების დასახარისხებლად გამოიყენება orderby განყოფილება. მოყვანილი პროგრამით ხდება მოთხოვნის მიერ გაცემული შედეგების დახარისხება ზრდადობით.

```
// პროგრამა 18.3
```

```
// შედეგის დახარისხება ზრდადობით
```

```
{
```

```
string[] saxelebi = { "რომანი", "ანა", "გიორგი", "არჩილი", "სანდრო", "დავითი", "ვახტანგი", "ბექა",  
                    "ალექსანდრე", "საბა", "სიმონი", "ლია", "ლუკა" };
```

```
// LINQ მოთხოვნის ფორმირება
```

```
var shedegi = from cvladi in saxelebi
```

```
where cvladi.StartsWith("ს")
```

```
orderby cvladi
```

```
select cvladi;
```

```
// შედეგების ეკრანზე გამოტანა
```

```
label1.Text = "ზრდადობით დალაგებული სახელები, რომლებიც იწყება 'ს' ასოთი:\n";
```

```
foreach (var elementi in shedegi)
```

```
    label1.Text += elementi.ToString() + "\n";
```

```
}
```

იმისათვის, რომ კლებადობით დავახარისხოთ მოთხოვნის მიერ გაცემული შედეგები, LINQ მოთხოვნა ასე უნდა ჩავწეროთ:

```
var shedegi = from cvladi in saxelebi where cvladi.StartsWith("ს")
```

```
orderby cvladi descending select cvladi;
```

თუ გვინდა, რომ სახელები დავახარისხოთ უკანასკნელი ასოების მიხედვით, მაშინ LINQ მოთხოვნა ასე უნდა ჩავწეროთ:

```
var shedegi = from cvladi in saxelebi where cvladi.StartsWith("ს")
```

```
orderby cvladi.Substring(cvladi.Length - 1) select cvladi;
```

მოთხოვნის მიერ გაცემული შედეგების დახარისხებისათვის შეგვიძლია, აგრეთვე გამოვიყენოთ მეთოდზე დაფუძნებული LINQ მოთხოვნა, ანუ LINQ მეთოდის სინტაქსი. მოყვანილი პროგრამით ხდება ამის დემონსტრირება:

```
// პროგრამა 18.4
```

```
// შედეგის დახარისხება ზრდადობით
```

```
{
```

```
string[] saxelebi = { "რომანი", "ანა", "გიორგი", "არჩილი", "სანდრო", "დავითი", "ვახტანგი", "ბექა",  
                    "ალექსანდრე", "საბა", "სიმონი", "ლია", "ლუკა" };
```

```
// LINQ მოთხოვნის ფორმირება
```

```
var shedegi = saxelebi.OrderBy(cvladi => cvladi).Where(cvladi => cvladi.StartsWith("ს"));
```

```
// შედეგების ეკრანზე გამოტანა
```

```
label1.Text = "ზრდადობით დალაგებული სახელები, რომლებიც იწყება „ს“ ასოთი:\n";
```

```
foreach (var elementi in shedegi)
```

```

        label1.Text += elementi.ToString() + "\n";
    }

```

თუ გვინდა შედეგის დახარისხება კლებადობის მიხედვით, მაშინ LINQ მოთხოვნა ასე უნდა ჩავწეროთ:

```

var shedegi = saxelebi.OrderByDescending(cvladi => cvladi).Where(cvladi => cvladi.StartsWith("ს"));

```

თუ გვინდა, რომ სახელები დავახარისხოთ უკანასკნელი ასოების მიხედვით, მაშინ LINQ მოთხოვნა ასე უნდა ჩავწეროთ:

```

var shedegi =
saxelebi.OrderBy(cvladi => cvladi.Substring(cvladi.Length - 1)).Where(cvladi => cvladi.StartsWith("ს"));

```

მონაცემთა დიდი ზომის ნაკრებთან მუშაობა

LINQ მოთხოვნა შეგვიძლია, აგრეთვე გამოვიყენოთ დიდი ზომის რიცხვითი მასივიდან რიცხვების ამოსაჩვენად. მოყვანილი პროგრამით ხდება ამის დემონსტრირება.

```

// პროგრამა 18.5

```

```

// შედეგიდან 5500-ზე ნაკლები მნიშვნელობის მქონე რიცხვების ამორჩევა

```

```

label1.Text = "";
Random Shemtxveviti_Ricxvebi = new Random();
int[] masivi = new int[1000000];
for ( int i = 0; i < masivi.Length; i++ )
masivi[i] = Shemtxveviti_Ricxvebi.Next();
// LINQ მოთხოვნის ფორმირება
var shedegi =
from cvladi in masivi
where cvladi < 5500
select cvladi;
// შედეგების ეკრანზე გამოტანა
label1.Text = "5500-ზე ნაკლები რიცხვები:\n";
foreach (var elementi in shedegi)
    label1.Text += elementi.ToString() + " ";
}

```

ძირითადი პროგრამიდან ხდება RixvebisGenerireba() მეთოდის გამოძახება და მისთვის 1000000-ის გადაცემა. შემთხვევითი გზით წარმოიქმნება ამდენივე რიცხვი, რომლებიც ჩაიწერება masivi მასივში. LINQ მოთხოვნის გამოყენებით ამ მასივიდან ამოირჩევა ის რიცხვები, რომლებიც 5500-ზე ნაკლებია.

LINQ მოთხოვნის გამოყენებით შეიძლება მასივიდან კენტი რიცხვების ამორჩევა. ამის დემონსტრირება ხდება მოყვანილი პროგრამით.

```

// პროგრამა 18.6

```

```

// შედეგიდან კენტი რიცხვების ამორჩევა

```

```

{
label1.Text = "";
//
int[] numbers = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
// LINQ მოთხოვნის ფორმირება
var numQuery =
from num in numbers

```

```

where ( num % 2 == 1 )
select num;
// შედეგების ეკრანზე გამოტანა
foreach (int num in numQuery)
    label1.Text += num.ToString() + " ";
}

```

თუ გვინდა კენტი რიცხვების ამორჩევა 5-15 დიაპაზონში, მაშინ მოთხოვნა ასე უნდა ჩავწეროთ:

```

where ( (num % 2 == 1 ) && ( num >= 5 && num <= 15) )

```

აგრეგირების ოპერატორები

LINQ მოთხოვნას აქვს აგრეგირების ოპერატორები, რომლებიც იძლევიან შედეგის ანალიზის საშუალებას. ხშირად გამოყენებადი ოპერატორებია:

Average() – გასცემს შედეგში მოთავსებული რიცხვების საშუალო არითმეტიკულს;

Count() – გასცემს მონაცემების რაოდენობას შედეგში;

Max() – გასცემს შედეგში მოთავსებული მონაცემების მაქსიმალურ მნიშვნელობას;

Min() – გასცემს შედეგში მოთავსებული მონაცემების მინიმალურ მნიშვნელობას;

Sum() - გასცემს შედეგში მოთავსებული რიცხვების ჯამს.

ამ ოპერატორების მუშაობის დემონსტრირება ხდება მოყვანილი პროგრამით.

```

// პროგრამა 18.7

```

```

// პროგრამით ხდება Average(), Count(), Max(), Min(), Sum()

```

```

// ოპერატორების მუშაობის დემონსტრირება

```

```

Random Shemtxveviti_Ricxvebi = new Random();

```

```

int[] masivi = new int[12345678];

```

```

for ( int i = 0; i < masivi.Length; i++ )

```

```

    masivi[i] = Shemtxveviti_Ricxvebi.Next();

```

```

// LINQ მოთხოვნის ფორმირება

```

```

var shedegi =

```

```

    from cvladi in masivi

```

```

    where cvladi > 9000

```

```

    select cvladi;

```

```

// შედეგების ეკრანზე გამოტანა

```

```

label1.Text = "იმ რიცხვების რაოდენობა, რომელთა მნიშვნელობები > 9000" + " ";

```

```

label1.Text += shedegi.Count().ToString() + '\n';

```

```

label1.Text += "მაქსიმალური რიცხვი, რომელიც აღემატება 9000-ს" + " ";

```

```

label1.Text += shedegi.Max().ToString() + '\n';

```

```

label1.Text += "მაქსიმალური რიცხვი, რომელიც ნაკლებია 9000-ზე" + " ";

```

```

label1.Text += shedegi.Min().ToString() + '\n';

```

```

label1.Text += "იმ რიცხვების საშუალო არითმეტიკული, რომელთა მნიშვნელობები > 9000" + " ";

```

```

label1.Text += shedegi.Average().ToString() + '\n';

```

```

label1.Text += "იმ რიცხვების ჯამი, რომელთა მნიშვნელობები > 9000" + " ";

```

```

label1.Text += shedegi.Sum(cvladi => (long) cvladi);

```

```

}

```

რთულ ობიექტებთან მუშაობა

ამ განყოფილებაში ვნახავთ თუ როგორ შეიძლება LINQ მოთხოვნის გამოყენება რთული ობიექტების მიმართ. მაგალითისთვის შევქმნათ სტუდენტის კლასი.

```
// პროგრამა 18.8
// პროგრამით ხდება LINQ მოთხოვნის გამოყენებით დემონსტრირება
// რთული ობიექტების მიმართ
class Student1
{
public string gvari;
public int asaki;
public double tanxa;
public int kursi;
public string fakulteti;
public Student1(string par1, int par2, double par3, int par4, string par5)
{
gvari = par1;
asaki = par2;
tanxa = par3;
kursi = par4;
fakulteti = par5;
}
}
private void button1_Click(object sender, EventArgs e)
{
Student1[] obj = new Student1[10];
obj[0] = new Student1("სამხარაძე", 21, 1200.50, 4, "ინფორმატიკის");
obj[1] = new Student1("კაპანაძე", 20, 1250.50, 4, "ენერგეტიკის");
obj[2] = new Student1("კირვალიძე", 18, 1300.50, 4, "ინფორმატიკის");
obj[3] = new Student1("ჭუმბურიძე", 19, 1100.50, 4, "ინფორმატიკის");
obj[4] = new Student1("ხუციშვილი", 20, 1400.50, 4, "სამშენებლო");
obj[5] = new Student1("კიკნაძე", 21, 1200.50, 4, "ინფორმატიკის");
obj[6] = new Student1("ნონიაშვილი", 20, 1250.50, 4, "ენერგეტიკის");
obj[7] = new Student1("ქევიშვილი", 18, 1300.50, 4, "ინფორმატიკის");
obj[8] = new Student1("ხომტარია", 19, 1100.50, 4, "ინფორმატიკის");
obj[9] = new Student1("ქარცივაძე", 20, 1400.50, 4, "სამშენებლო");
// LINQ მოთხოვნის ფორმირება
var shedegi =
from cvladi in obj
where cvladi.fakulteti == "ინფორმატიკის"
select cvladi;
// ეკრანზე შედეგების გამოტანა
label1.Text = "ინფორმატიკის ფაკულტეტის სტუდენტები:\n
გვარი ფაკულტეტი ასაკი კურსი თანხა\n";
foreach (Student1 cvladi in shedegi)
label1.Text += cvladi.gvari + " " + cvladi.fakulteti + " " + cvladi.asaki.ToString() + " " +
cvladi.kursi.ToString() + " " + cvladi.tanxa.ToString() + "\n";
}
```

თითოეული ობიექტის შესაქმნელად კონსტრუქტორის გამოძახების ნაცვლად უფრო სწრაფი და მარტივია List<T> ტიპის გამოყენება. მისი საშუალებით შეგვიძლია შევქმნათ Studenti ტიპის ობიექტების კოლექცია.

// პროგრამა 18.9

// პროგრამით ხდება List<T> ტიპის გამოყენების დემონსტრირება LINQ მოთხოვნაში

```
class Studenti
{
public string gvari;
public int asaki;
public double tanxa;
public int kursi;
public string fakulteti;
}
private void button1_Click(object sender, EventArgs e)
{
List<Studenti> Studentebi = new List<Studenti>
{
new Studenti { gvari = "სამხარაძე", asaki = 21, tanxa = 1600, kursi = 4, fakulteti = "ინფორმატიკის" },
new Studenti { gvari = "კაპანაძე", asaki = 18, tanxa = 1770.50, kursi = 3, fakulteti = "ენერგეტიკის" },
new Studenti { gvari = "ქეციშვილი", asaki = 20, tanxa = 1830.50, kursi = 1, fakulteti = "სამშენებლო" },
new Studenti { gvari = "ყალაბეგიშვილი", asaki=19,tanxa=2150.00,kursi=2,fakulteti="ჰუმანიტარული"},
new Studenti { gvari = "ხომტარია", asaki = 21, tanxa = 2050.50, kursi = 2, fakulteti = "ინფორმატიკის" },
new Studenti { gvari = "მამიაშვილი", asaki = 19, tanxa = 1950.50, kursi = 3, fakulteti = "ინფორმატიკის" },
new Studenti { gvari = "ნონიაშვილი", asaki = 18, tanxa = 1900.00, kursi = 4, fakulteti = "სამშენებლო" },
new Studenti { gvari = "ასანიძე", asaki = 20, tanxa = 1500.50, kursi = 4, fakulteti = "ჰუმანიტარული" },
new Studenti { gvari = "ჭუმბურიძე", asaki = 19, tanxa = 2000.50, kursi = 4, fakulteti = "ენერგეტიკის" },
new Studenti { gvari = "კირვალიძე", asaki = 18, tanxa = 1275.00, kursi=2, fakulteti="ინფორმატიკის" }
};
// LINQ მოთხოვნის ფორმირება
var shedegi =
from cvladi in Studentebi
where cvladi.fakulteti == "ინფორმატიკის"
select cvladi;
// ეკრანზე შედეგების გამოტანა
label1.Text = "ინფორმატიკის ფაკულტეტის სტუდენტები:\n
გვარი ფაკულტეტი ასაკი კურსი თანხა\n";
foreach (Studenti cvladi in shedegi)
label1.Text += cvladi.gvari + " " + cvladi.fakulteti + " " + cvladi.asaki.ToString() + " " +
cvladi.kursi.ToString() + " " + cvladi.tanxa.ToString() + "\n";
}
```

მოთხოვნაში ახალი ობიექტის შექმნა

LINQ მოთხოვნაში ჩვენ Studentebi სიიდან შეგვიძლია ამოვირჩიოთ მხოლოდ ერთი, რომელიმე ველი, მაგალითად gvari. ამისათვის მოთხოვნა ასე უნდა ჩავწეროთ:

```
var shedegi =
from cvladi in Studentebi
```

```
where cvladi.fakulteti == "ინფორმატიკის"
```

```
select cvladi.gvari;
```

თუ გვინდა რამდენიმე ველის ამორჩევა, მაშინ select განყოფილებაში უნდა მივუთითოთ new საკვანძო სიტყვა. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
// პროგრამა 18.10
```

```
// პროგრამით ხდება new საკვანძო სიტყვის გამოყენების დემონსტრირება
```

```
class Studenti
```

```
{  
    public string gvari;  
    public int asaki;  
    public double tanxa;  
    public int kursi;  
    public string fakulteti;
```

```
}  
private void button1_Click(object sender, EventArgs e)
```

```
{  
    label1.Text = "";
```

```
List<Studenti> Studentebi = new List<Studenti>
```

```
{  
    new Studenti { gvari = "სამხარაძე", asaki = 21, tanxa = 1600, kursi = 4, fakulteti = "ინფორმატიკის" },  
    new Studenti { gvari = "კაპანაძე", asaki = 18, tanxa = 1770.50, kursi = 3, fakulteti = "ენერგეტიკის" },  
    new Studenti { gvari = "ქეციშვილი", asaki = 20, tanxa = 1830.50, kursi = 1, fakulteti = "სამშენებლო" },  
    new Studenti { gvari = "ყალაბეგიშვილი", asaki=19,tanxa=2150.00,kursi=2,fakulteti="ჰუმანიტარული"},  
    new Studenti { gvari = "ხომტარია", asaki = 21, tanxa = 2050.50, kursi = 2, fakulteti = "ინფორმატიკის" },  
    new Studenti { gvari = "ქარცივაძე", asaki = 19, tanxa = 1950.50, kursi = 3, fakulteti = "ინფორმატიკის"},  
    new Studenti { gvari = "ნონიაშვილი", asaki = 18, tanxa = 1900.00, kursi = 4, fakulteti = "სამშენებლო" },  
    new Studenti { gvari = "გასიტაშვილი", asaki = 20, tanxa = 2150.50, kursi = 1, fakulteti = "ინფორმატიკის"},  
    new Studenti { gvari = "კიკნაძე", asaki = 20, tanxa = 1500.50, kursi = 4, fakulteti = "ჰუმანიტარული" },  
    new Studenti { gvari = "სიდამონიძე", asaki = 21, tanxa = 1700.00, kursi = 3, fakulteti = "ინფორმატიკის"},  
    new Studenti { gvari = "ჭუმბურიძე", asaki = 19, tanxa = 2000.50, kursi = 4, fakulteti = "ენერგეტიკის" },  
    new Studenti { gvari = "ბარბაქაძე", asaki = 19, tanxa = 2200.50, kursi = 3, fakulteti = "ჰუმანიტარული"},  
    new Studenti { gvari = "ბერძენიშვილი", asaki = 18, tanxa = 1275.00, kursi = 2, fakulteti = "ინფორმატიკის"}  
};
```

```
// LINQ მოთხოვნის ფორმირება
```

```
var shedegi = from cvladi in Studentebi where cvladi.fakulteti == "ინფორმატიკის"
```

```
select new { cvladi.gvari, cvladi.tanxa, cvladi.asaki };
```

```
// ეკრანზე შედეგების გამოტანა
```

```
foreach (var elementi in shedegi)
```

```
    label1.Text += elementi.ToString() + "\n";
```

```
}
```

პროექცია ნიშნავს მონაცემთა ახალი ტიპის შესაქმნას LINQ მოთხოვნაში სხვა მონაცემთა ტიპიდან. select საკვანძო სიტყვა პროექციის ოპერატორია. პროგრამაში ვიყენებთ ანონიმური ტიპის შექმნის სინტაქსს პირდაპირ select განყოფილებაში ობიექტის ახალი უსახელო ტიპის შესაქმნელად, რომელსაც აქვს gvari, tanxa და asaki თვისებები. select განყოფილება ქმნის ახალ ობიექტს. ამრიგად, მხოლოდ ეს სამი თვისება (ან ცვლადი) იქნება დუბლირებული და დაჭერილი მოთხოვნის დამუშავების სხვადასხვა საფეხურზე.

ახალი ობიექტის შექმნა შეგვიძლია, აგრეთვე ლამბდა გამოსახულების გამოყენებით. მოთხოვნას ექნება სახე:

```
var shedegi1 = Studentebi.Where(cvladi => cvladi.fakulteti == "ინფორმატიკის")
.Select(cvladi => new { cvladi.gvari, cvladi.fakulteti, cvladi.kursi });
```

როგორც მოთხოვნიდან ჩანს, ჯერ სრულდება Where() მეთოდი და შემდეგ, მიღებულ შედეგებზე შესრულდება Select() მეთოდი. შეგვიძლია, ჯერ შევასრულოთ Select() მეთოდი და შემდეგ, მიღებულ შედეგებზე შევასრულოთ Where() მეთოდი. მივიღებთ იგივე შედეგს. მოთხოვნას ექნება სახე:

```
var shedegi3 = Studentebi.Select(cvladi => new { cvladi.gvari, cvladi.fakulteti, cvladi.kursi })
.Where(cvladi => cvladi.fakulteti == "ინფორმატიკის");
```

```
// ეკრანზე შედეგების გამოტანა
foreach (var elementi in shedegi3)
    label2.Text += elementi.ToString() + "\n";
```

LINQ მეთოდები

Distinct() მეთოდი

არაგამეორებადი მნიშვნელობების მისაღებად LINQ მოთხოვნაში უნდა გამოვიყენოთ Distinct() მეთოდი. 18.10 პროგრამაში LINQ მოთხოვნა შევცვალოთ შემდეგი მოთხოვნით:

```
var shedegi = Studentebi.Select(cvladi => cvladi.kursi).Distinct();
```

ეკრანზე გამოჩნდება სტუდენტების კურსების მნიშვნელობები: 1, 2, 3, 4. თუ Distinct() მეთოდს არ მივუთითებთ, მაშინ ეკრანზე გამოჩნდება კურსების გამეორებადი მნიშვნელობები განმეორებით იქნება გამოტანილი თითოეული სტუდენტისათვის.

Any() და All() მეთოდები

Any() მეთოდი გასცემს true მნიშვნელობას, თუ ერთი მონაცემი მაინც აკმაყოფილებს მითითებულ პირობას, წინააღმდეგ შემთხვევაში - false-ს. All() მეთოდი გასცემს true მნიშვნელობას, თუ ყველა მონაცემი აკმაყოფილებს მითითებულ პირობას, წინააღმდეგ შემთხვევაში - false-ს. მაგალითად, იმისათვის, რომ დავადგინოთ ინფორმატიკის ფაკულტეტზე არის თუ არა ერთი სტუდენტი მაინც, რომელმაც გადაიხადა 2000 ლარზე მეტი, LINQ მოთხოვნა 18.10 პროგრამაში შევცვალოთ შემდეგნაირად:

```
bool shedegi1 = Studentebi.Any(cvladi => cvladi.tanxa >= 2000);
if ( shedegi1 ) label1.Text = "ინფორმატიკის ფაკულტეტზე არიან სტუდენტები,
    \nრომლებმაც გადაიხადეს 2000 ლარზე მეტი";
else label1.Text = "ინფორმატიკის ფაკულტეტზე არ არიან სტუდენტები,
    \nრომლებმაც გადაიხადეს 2000 ლარზე მეტი";
```

იმისათვის, რომ დავადგინოთ ინფორმატიკის ფაკულტეტზე ყველა სტუდენტმა გადაიხადა თუ არა 2000 ლარზე მეტი, LINQ მოთხოვნა უნდა შევიტანოთ შემდეგნაირად:

```
bool shedegi2 = Studentebi.All(cvladi => cvladi.tanxa >= 2000);
if ( shedegi2 ) label2.Text = "ინფორმატიკის ფაკულტეტზე ყველა სტუდენტმა
    გადაიხადა 2000 ლარზე მეტი";
else label2.Text = "ინფორმატიკის ფაკულტეტზე ყველა სტუდენტმა
    არ გადაიხადა 2000 ლარზე მეტი";
```


დახარისხება რამდენიმე ველის მიხედვით

მოთხოვნის შედეგების დახარისხება ჩვენ უკვე შევისწავლეთ (პროგრამა 18.3). იქ დახარისხება სრულდებოდა ერთი მნიშვნელობის მიხედვით. მაგრამ, მოთხოვნის შედეგების დახარისხება შესაძლებელია, აგრეთვე რამდენიმე ველის მიხედვით. დავუშვათ, გვინდა, რომ შედეგში მოთავსებული მონაცემები დავახარისხოთ ჯერ ფაკულტეტის, შემდეგ კურსის, ასაკის, გვარისა და გადახდილი თანხის მიხედვით, LINQ მოთხოვნა უნდა შევიტანოთ შემდეგნაირად:

```
var shedegi = from cvladi in Studentebi
              orderby cvladi.fakulteti, cvladi.kursi, cvladi.asaki descending, cvladi.gvari, cvladi.tanxa
              select new { cvladi.fakulteti, cvladi.kursi, cvladi.asaki, cvladi.gvari, cvladi.tanxa };
// ეკრანზე შედეგების გამოტანა
foreach (var elementi in shedegi)
    label1.Text += elementi.ToString() + "\n";
```

როგორც მოთხოვნიდან ჩანს, მონაცემები ასაკის მიხედვით დახარისხდება კლებადობით, ხოლო ფაკულტეტის, კურსის, გვარისა და თანხის მიხედვით კი - ზრდადობით. ყურადღება მივაქციოთ იმას, რომ select განყოფილებაში გამოიყენება new საკვანძო სიტყვა.

ახლა იგივე გავაკეთოთ ლამბდა გამოსახულების გამოყენებით. ამ შემთხვევაში, უნდა გამოვიყენოთ OrderBy(), ThenBy() და ThenByDescending() მეთოდები. LINQ მოთხოვნას ექნება სახე:

```
var shedegi2 = Studentebi.OrderBy(cvladi => cvladi.fakulteti)
                  .ThenByDescending(cvladi => cvladi.asaki)
                  .ThenBy(cvladi => cvladi.kursi)
                  .Select(cvladi => new { cvladi.fakulteti, cvladi.kursi, cvladi.asaki, cvladi.gvari, cvladi.tanxa });
// ეკრანზე შედეგების გამოტანა
foreach (var elementi in shedegi2)
    label1.Text += elementi.ToString() + "\n";
```

ჯგუფური მოთხოვნა

ჯგუფური მოთხოვნა მონაცემებს ჯგუფებად ჰყოფს და იძლევა ამ ჯგუფების მიხედვით მონაცემების დახარისხების, გამოთვლებისა და შედარების შესრულების საშუალებას. დავუშვათ, გვინტერესებს ფაკულტეტების მიხედვით გადახდილი თანხები. LINQ მოთხოვნას ექნება სახე:

```
var shedegi = from cvladi in Studentebi
              group cvladi by cvladi.fakulteti into jgufi
              select new { Fakulteti = jgufi.Key, Tanxa = jgufi.Sum(cvladi => cvladi.tanxa) };
// LINQ მოთხოვნის ფორმირება
var shedegi_jgufi = from jgufi in shedegi
                   orderby jgufi.Tanxa descending
                   select jgufi;
// ეკრანზე შედეგების გამოტანა
label1.Text = "ფაკულტეტი    თანხა\n";
foreach ( var elementi in shedegi_jgufi )
```

```
    label1.Text += elementi.Fakulteti.ToString() + " - " + elementi.Tanxa.ToString() + "\n";
```

პირველ მოთხოვნაში საკვანძოა fakulteti ველი. მისი მნიშვნელობების მიხედვით შესრულდება მონაცემების დაჯგუფება jgufi სიმრავლეში. select განყოფილებაში იქმნება ახალი ანონიმური ტიპი და ჯგუფის საკვანძო მნიშვნელობა, რომელსაც მივმართავთ ჯგუფის გასაღების საშუალებით - Fakulteti = jgufi.Key. Key თვისების გამოყენება საჭიროა, რადგან ის გვიჩვენებს კრიტერიუმს, რომლის მიხედვითაც უნდა შეიქმნას თითოეული ჯგუფი. ფაკულტეტების

მიხედვით დაჯგუფებულ მონაცემებზე ვასრულებთ სტუდენტების მიერ გადახდილი თანხების შეკრების ოპერაციას.

Take() და Skip() მეთოდები

Take() მეთოდი მოთხოვნის შესრულების შედეგიდან გასცემს მითითებული რაოდენობის მონაცემს დაწყებული დასაწყისიდან. Skip() მეთოდი მოთხოვნის შესრულების შედეგიდან გამოტოვებს მითითებული რაოდენობის მონაცემს დაწყებული დასაწყისიდან და გასცემს დარჩენილ მონაცემებს.

დავუშვათ, გვინტერესებს მიღებული შედეგის პირველი 2 მონაცემი. LINQ მოთხოვნას ექნება სახე:

```
var shedegi = from cvladi in Studentebi
orderby cvladi.tanxa descending
select new { cvladi.gvari, cvladi.fakulteti, cvladi.kursi, cvladi.asaki };
// ეკრანზე შედეგების გამოტანა
label1.Text = "პირველი ორი სტუდენტი\n";
foreach (var elementi in shedegi.Take(2))
```

```
    label1.Text += elementi.ToString() + "\n";
```

თუ გვინტერესებს მიღებული შედეგის ყველა მონაცემი გარდა პირველი 4 მონაცემისა, მაშინ უნდა შევასრულოთ კოდი:

```
label1.Text += "სტუდენტების სია პირველი ოთხის გარდა\n";
foreach (var elementi in shedegi.Skip(4))
    label1.Text += elementi.ToString() + "\n";
```

First() და FirstOrDefault() მეთოდები

First() მეთოდი გასცემს პირველ ნაპოვნ მონაცემს მოთხოვნის შესრულების შედეგიდან, რომელიც ძებნის პირობას აკმაყოფილებს. FirstOrDefault() მეთოდი აკეთებს იგივეს, ოღონდ თუ საძებნი მნიშვნელობა არ მოიძებნა, მაშინ გასცემს სიის ნაგულისხმევ ელემენტს, რომელიც უნდა იყოს null. წინააღმდეგ შემთხვევაში, გაიცემა შეტყობინება შეცდომის შესახებ. დავუშვათ, გვინტერესებს ინფორმატიკის ფაკულტეტის პირველი სტუდენტის გვარი. მოთხოვნას ექნება სახე:

```
var shedegi = from cvladi in Studentebi
                select new { cvladi.fakulteti, cvladi.gvari, cvladi.tanxa, cvladi.kursi };
// ეკრანზე შედეგების გამოტანა
label1.Text = "ინფორმატიკის ფაკულტეტის სტუდენტია:\n";
label1.Text += shedegi.First(cvladi => cvladi.fakulteti == "ინფორმატიკის").ToString() + "\n";
label1.Text += "ჟურნალისტიკის ფაკულტეტის სტუდენტია:\n";
label1.Text += shedegi.FirstOrDefault(cvladi => cvladi.fakulteti == "ჟურნალისტიკის").ToString();
```

Intersect(), Except() და Union() მეთოდები

LINQ უზრუნველყოფს სტანდარტულ ოპერაციებს სიმრავლეებზე. Intersect() მეთოდი მოთხოვნის შედეგში ჩართავს იმ ელემენტების სიმრავლეს, რომელიც ორივე კოლექციაში გვხვდება. Except() მეთოდი მოთხოვნის შედეგში ჩართავს ერთი კოლექციის ელემენტებს, რომლებიც მეორე კოლექციაში არ გვხვდება. Union() მეთოდი გასცემს ორივე კოლექციის ელემენტებს, რომლებიც არ მეორდება. მოყვანილი პროგრამით ხდება ამ მეთოდებთან მუშაობის დემონსტრირება:

```
// პროგრამა 18.11
// პროგრამით ხდება Intersect(), Except() და Union()
```

```

// მეთოდების გამოყენების დემონსტრირება
class Leqtori
{
public string gvari;
public int staji;
}
List<Leqtori> Leqtorebi = new List<Leqtori>
{
new Leqtori { gvari="კაპანაძე", staji=10 },
new Leqtori { gvari="ნონიაშვილი", staji=15 },
new Leqtori { gvari="სამხარაძე", staji=30 },
new Leqtori { gvari="კიკნაძე", staji=20 },
new Leqtori { gvari="ნემსაძე", staji=10 },
new Leqtori { gvari="გაჩეჩილაძე", staji=15 },
new Leqtori { gvari="ლომიძე", staji=20 }
};
{
// LINQ მოთხოვნის ფორმირება
var studenti_shedegi = from cvladi_studenti in Studentebi select cvladi_studenti.gvari;
var leqtori_shedegi = from cvladi_leqtori in Leqtorebi select cvladi_leqtori.gvari;
// LINQ მოთხოვნის ფორმირება
var Leqtorebi_Studentebi = studenti_shedegi.Intersect(leqtori_shedegi);
label1.Text = "ერთნაირი გვარის ლექტორები და სტუდენტები:\n";
foreach ( var elementi in Leqtorebi_Studentebi )
    label1.Text += elementi.ToString() + "\n";
//
var Studentebi_1 = leqtori_shedegi.Except(studenti_shedegi);
label2.Text = "სხვა გვარის სტუდენტები:\n";
foreach (var elementi in Studentebi_1)
    label2.Text += elementi.ToString() + "\n";
// LINQ მოთხოვნის ფორმირება
var Studentebi_da_Leqtorebi = leqtori_shedegi.Union(studenti_shedegi);
label3.Text = "ლექტორები და სტუდენტები:\n";
foreach (var elementi in Studentebi_da_Leqtorebi)
    label3.Text += elementi.ToString() + "\n";
}

```

Joins() მეთოდი

Joins() მეთოდი იძლევა ორი რელაციური (ბმული) მონაცემების დაკავშირების საშუალებას საკვანძო ველის გამოყენებით. Studentebi სია (მთავარი) შეგვიძლია დავუკავშიროთ Leqtorebi სიას (დამოკიდებული) გვარი საკვანძო ველის საშუალებით. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```

{
// პროგრამა 18.12
// პროგრამით ხდება Joins() მეთოდის გამოყენების დემონსტრირება
label1.Text = "";
// LINQ მოთხოვნის ფორმირება

```

```

var shedegi = from cvladi_studenti in Studentebi
join cvladi_leqtori in Leqtorebi on cvladi_studenti.gvari equals cvladi_leqtori.gvari
select new
{
    cvladi_studenti.gvari,
    cvladi_studenti.asaki,
    Kursi = cvladi_studenti.kursi,
    Leqtoris_Gvari = cvladi_leqtori.gvari,
    Leqtoris_Stajii = cvladi_leqtori.staji,
    Leqtoris_Asaki_Staji = cvladi_studenti.asaki + cvladi_leqtori.staji
};
// ეკრანზე შედეგების გამოტანა
foreach (var elementi in shedegi)
    label1.Text += elementi.ToString() + "\n";
}

```

LINQ to SQL

LINQ-ს აქვს SQL მონაცემთა ბაზებთან მუშაობის ყველა საშუალება. ის LINQ მოთხოვნას ავტომატურად გარდაქმნის SQL ოპერატორებად და ჩვენს პროგრამებს შეეძლება C# ობიექტებთან მარტივად მუშაობა.

ობიექტურ-რელაციური ასახვა

როგორც ვიცით, SQL მონაცემთა ბაზები, ანუ რელაციური მონაცემთა ბაზები მონაცემებს ინახავენ ცხრილებში, რომლებიც წარმოადგენენ სტრიქონებისა და სვეტების ჯგუფებს. C# ენაში კი - მონაცემები ინახება მესხიერების ობიექტებში. LINQ to SQL ქმნის ობიექტურ-რელაციური ასახვის (object-relation mapping, ORM) ფენას რელაციური ბაზის ცხრილებსა და C# პროგრამის ობიექტებს შორის (ნახ. 18.1). LINQ to SQL ასახვაში ავტომატურად იქმნება მონაცემთა ბაზის ცხრილების შესაბამისი კლასები თვით მონაცემთა ბაზიდან.

ახლა შევადგინოთ LINQ მოთხოვნა, რომელიც Shekveta მონაცემთა ბაზის Personali ცხრილიდან თბილისელი თანამშრომლების შესახებ ინფორმაციას dataGridView კომპონენტში გამოიტანს. ამისათვის, ჯერ შევქმნათ ახალი პროექტი. შემდეგ, მას დავუმატოთ LINQ to SQL კლასი, რისთვისაც შევასრულოთ Project მენიუს Add Class (ან Add New Item) ბრძანება. გახსნილ ფანჯარაში ავირჩიოთ LINQ to SQL Classes ელემენტი და Name ველში შევიტანოთ მისი სახელი Shekveta. დავაჭიროთ Add კლავიშს. გამოჩნდება Shekveta.dbml ჩანართი (ნახ. 18. 2). შემდეგ, გავხსნათ Server Explorer ფანჯარა და გავხსნათ Tables განყოფილება (იხილეთ 17.2 და 17.3 ფანჯრები). ამის შემდეგ, Shekveta.dbml ჩანართში ბუქსირის ოპერაციის გამოყენებით გადავიტანოთ Personali, Shemkveti და Xelshekruleba ცხრილები (ნახ. 18.4). გახსნილ ფანჯარაში (ნახ. 18.3.) ვაჭერთ Yes კლავიშს. შედეგად, შეერთება დაიმასსოვრება პაროლთან ერთად. ამის შემდეგ, ფორმაზე მოვათავსოთ button, dataGridView და bindingNavigator კომპონენტები (ნახ. 18.5). button კომპონენტს მივაბათ კოდი, რომელსაც Personali ცხრილიდან ეკრანზე გამოაქვს მონაცემების თბილისელი თანამშრომლების შესახებ:

```

{
// პროგრამა 18.13
//
ShekvetaDataContext shekvetaDataContext = new ShekvetaDataContext();

```

```
// LINQ მოთხოვნის ფორმირება
var shedegebi = from cvladi in shekvetaDataContext.Personalis
                where cvladi.qalaqi == "თბილისი"
                select new
                {
                    PersonaliID = cvladi.personaliID,
                    Gvari = cvladi.gvari,
                    Ganyofileba = cvladi.ganyofileba,
                    Asaki = cvladi.asaki,
                    Qalaqi = cvladi.qalaqi
                };
dataGridView1.DataSource = shedegebi;
}
```

პროგრამის შესრულების შედეგები dataGridView1 კომპონენტში გამოჩნდება. სურვილის შემთხვევაში, შედეგები შეგვიძლია label1 კომპონენტში გამოვიტანოთ:

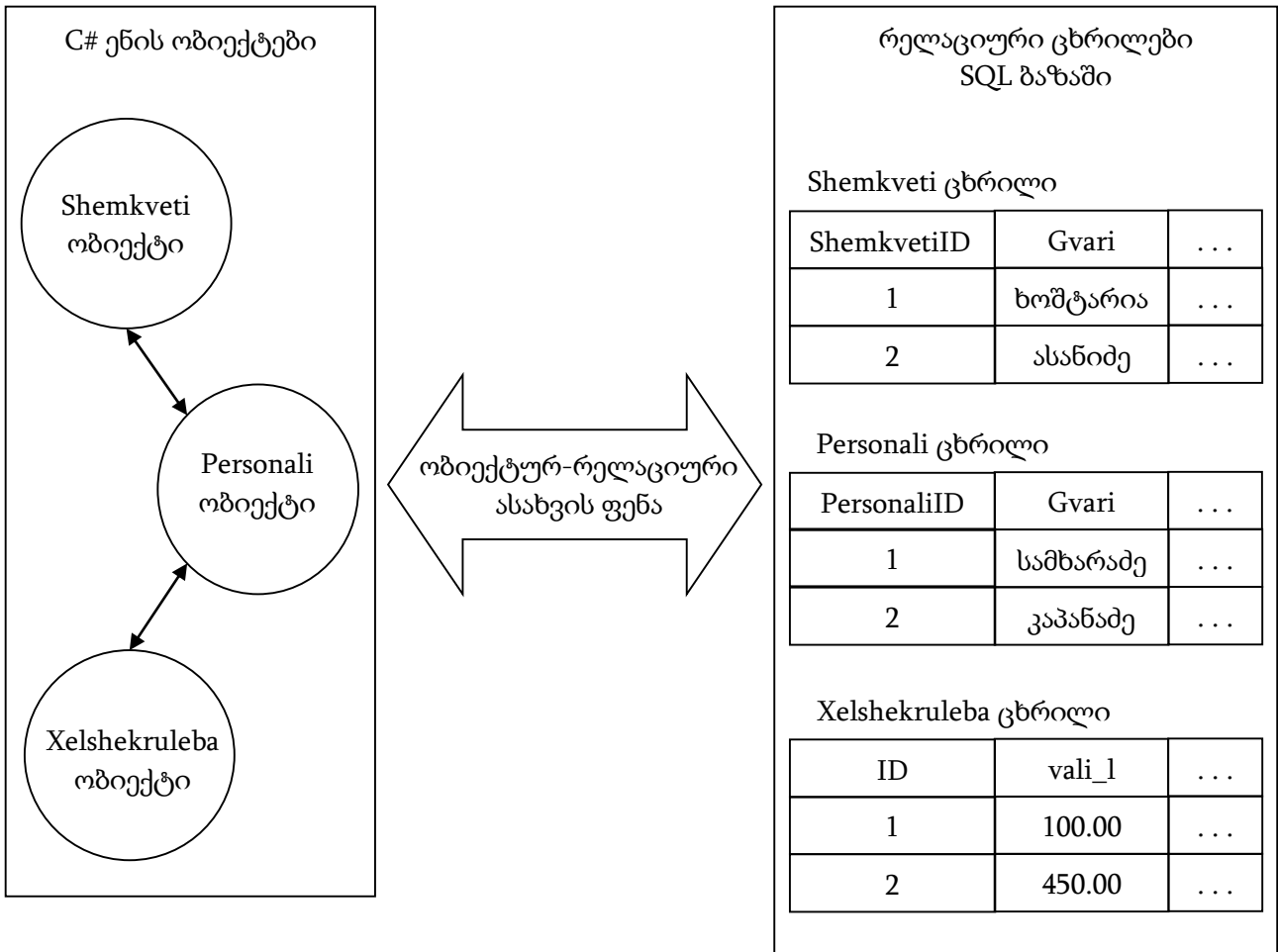
```
foreach ( var elementi in shedegebi )
    label1.Text += elementi.ToString() + "\n";
```

ობიექტურ-რელაციური დამპროექტებელი აღმოაჩენს, რომ Personalis ცხრილის სახელი წარმოადგენს არსებითი სახელის მრავლობით ფორმას და ავტომატურად ქმნის Personali სახელს მხოლოდ რიცხვში გენერირებული ობიექტისთვის. თუ გვინდა ამ რეჟიმის გამორთვა, უნდა შევასრულოთ Tools→Options→Database Tools→O/R Designer ბრძანება. გახსნილი ფანჯარის Enabled ველში უნდა ავირჩიოთ false მნიშვნელობა.

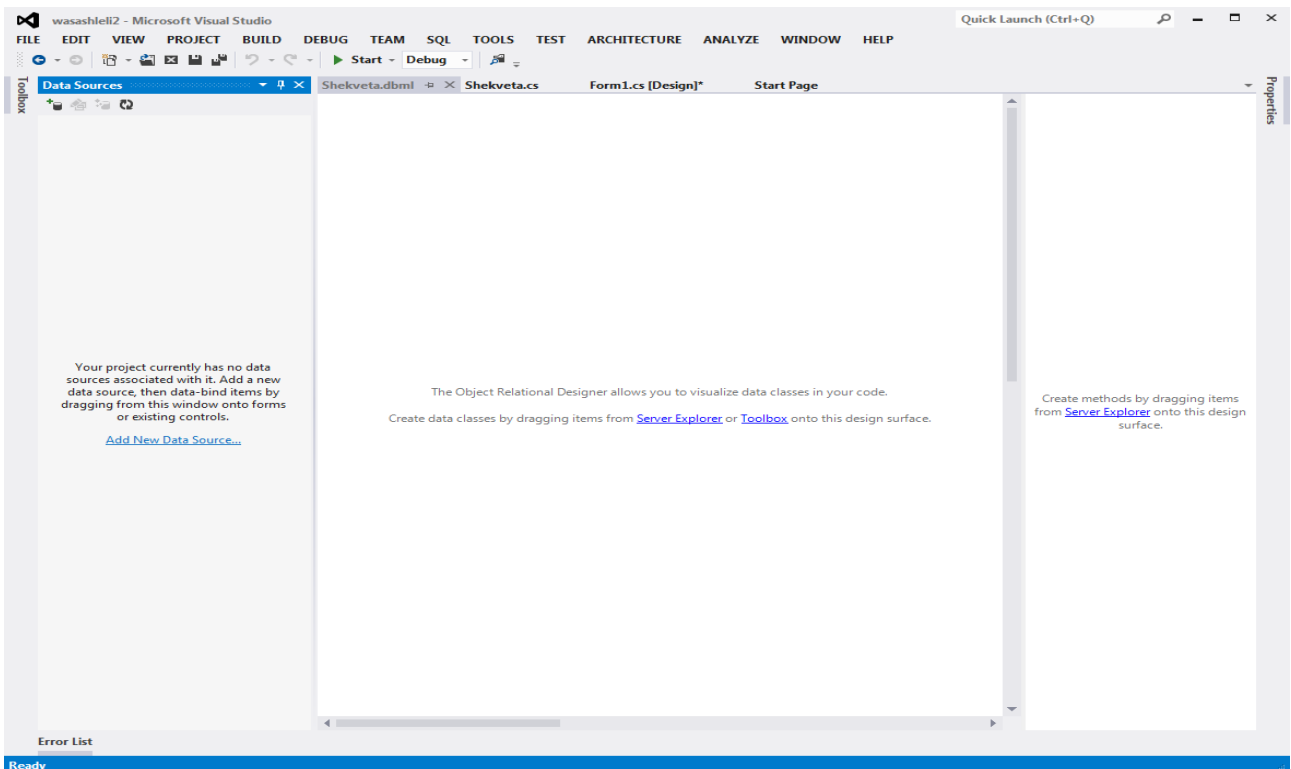
LINQ მოთხოვნის შესრულებისას მონაცემების წყაროდ გამოიყენება shekvetaDataContext ობიექტის Personalis წევრი. Personalis – LINQ-ის ტიპიზებული ცხრილია (System.Data.Linq.Table<Personali>), რომელიც Personali ობიექტების ტიპიზებული კოლექციის მსგავსია (როგორც List<Table>), მაგრამ რეალიზებულია LINQ to SQL-თვის და ავტომატურად ივსება მონაცემებით მონაცემთა ბაზიდან. ის ახდენს IEnumerable/IQueryable ინტერფეისების რეალიზებას და ამიტომ, შეიძლება გამოიყენებული იყოს როგორც LINQ მონაცემთა წყარო from კონსტრუქციაში, როგორც ნებისმიერი სხვა კოლექცია ან მასივი. ამრიგად, ჩვენ შევქმენით LINQ to SQL საბაზო მოთხოვნა, რომელიც შეგვიძლია გამოვიყენოთ როგორც საფუძველი სხვა უფრო რთული მოთხოვნების ასაგებად.

ნავიგაცია LINQ to SQL ბმების მიმართ

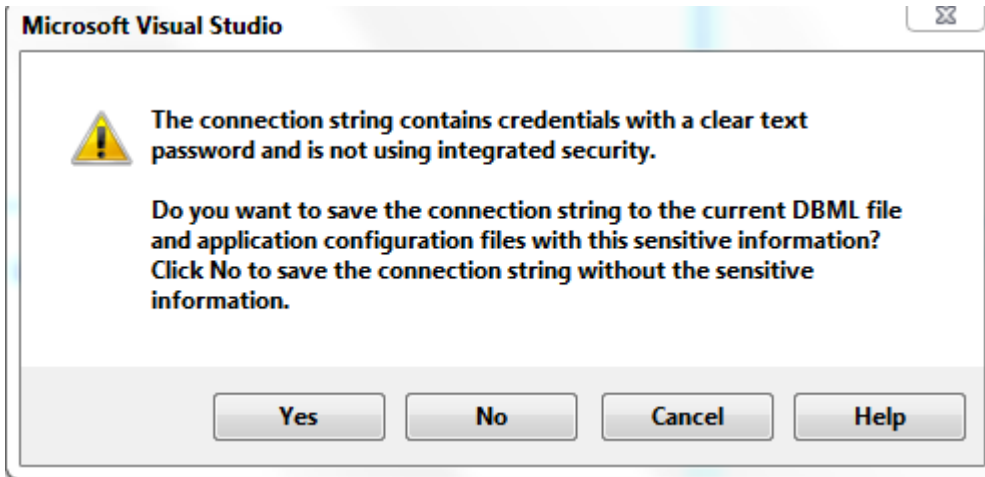
ობიექტურ-რელაციური დამპროექტებლის ერთ-ერთი შესაძლებლობაა LINQ to SQL ობიექტების ავტომატურად შექმნა. მათი საშუალებით შევძლებთ ნავიგაციას ბმულ ცხრილების გასწვრივ. წინა პროექტს დავუმატოთ ბმა (რელაცია) Personali და Shemkveti ცხრილებს შორის. ამისათვის, გავხსნათ Shekveta.dbml ჩანართი. მოვნიშნოთ Personali ცხრილის personaliID სვეტი და Properties ფანჯარაში Primary Key ველის მნიშვნელობა დავაყენოთ true მდგომარეობაში (თუ ის false მდგომარეობაშია). იგივე გავაკეთოთ Shemkveti და Xelshekruleba ცხრილებისთვის. შემდეგ, ცარიელ ადგილზე გავხსნათ კონტექსტური მენიუ და შევასრულოთ Add ქვემენიუს Association ბრძანება. Association Editor ფანჯარაში (ნახ. 18.6) ვირჩევთ მთავარ და დამოკიდებულ ცხრილებსა და შესაბამის სვეტებს. Ok კლავიშზე დაჭერის შემდეგ გამოჩნდება ბმა ამ ცხრილებს შორის (ნახ. 18.7). ასეთივე გზით შევქმნათ ბმა Personali და Xelshekruleba ცხრილებს შორის.



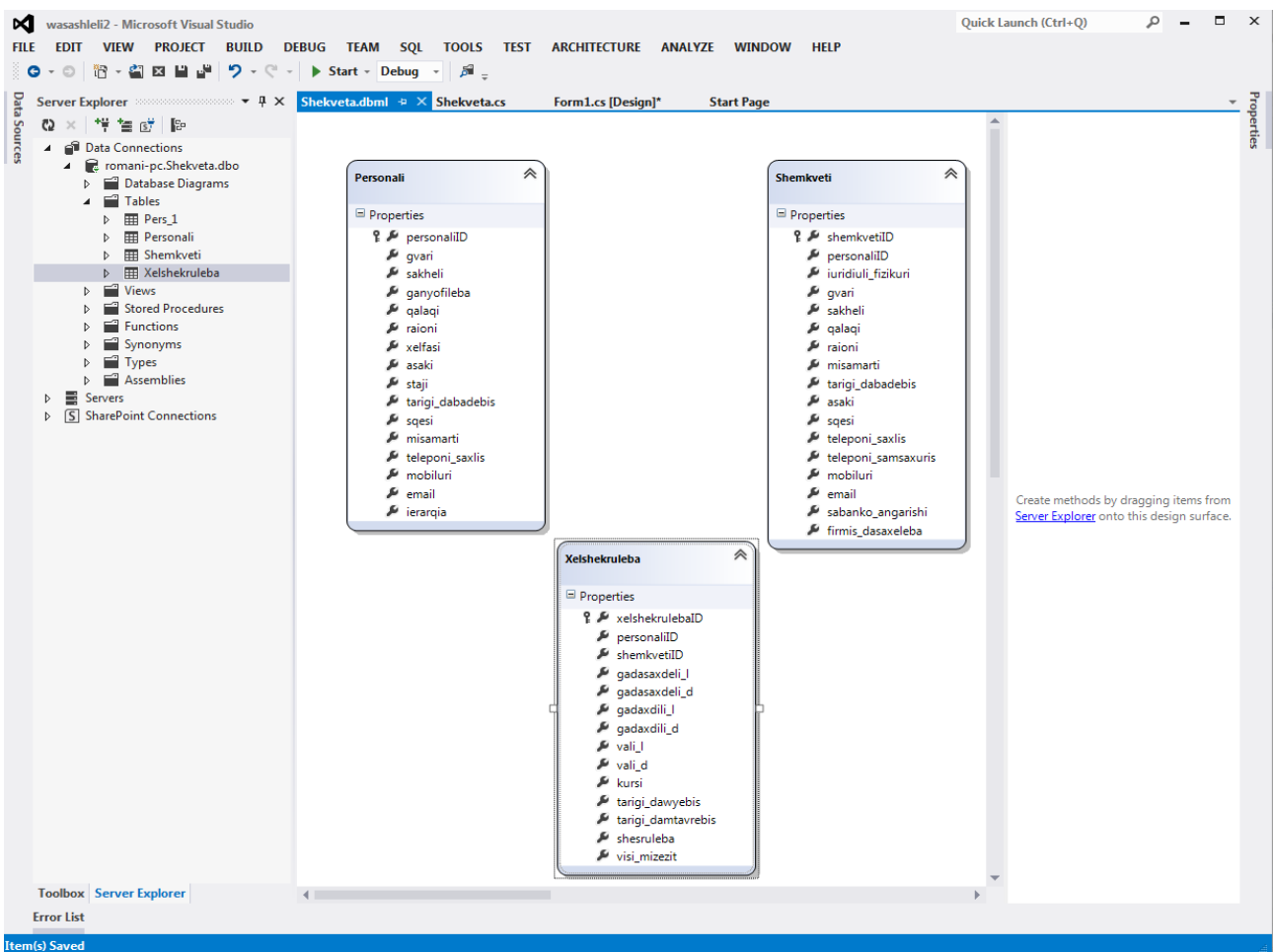
ნახ. 18.1. ობიექტურ-რელაციური ასახვის ფენა



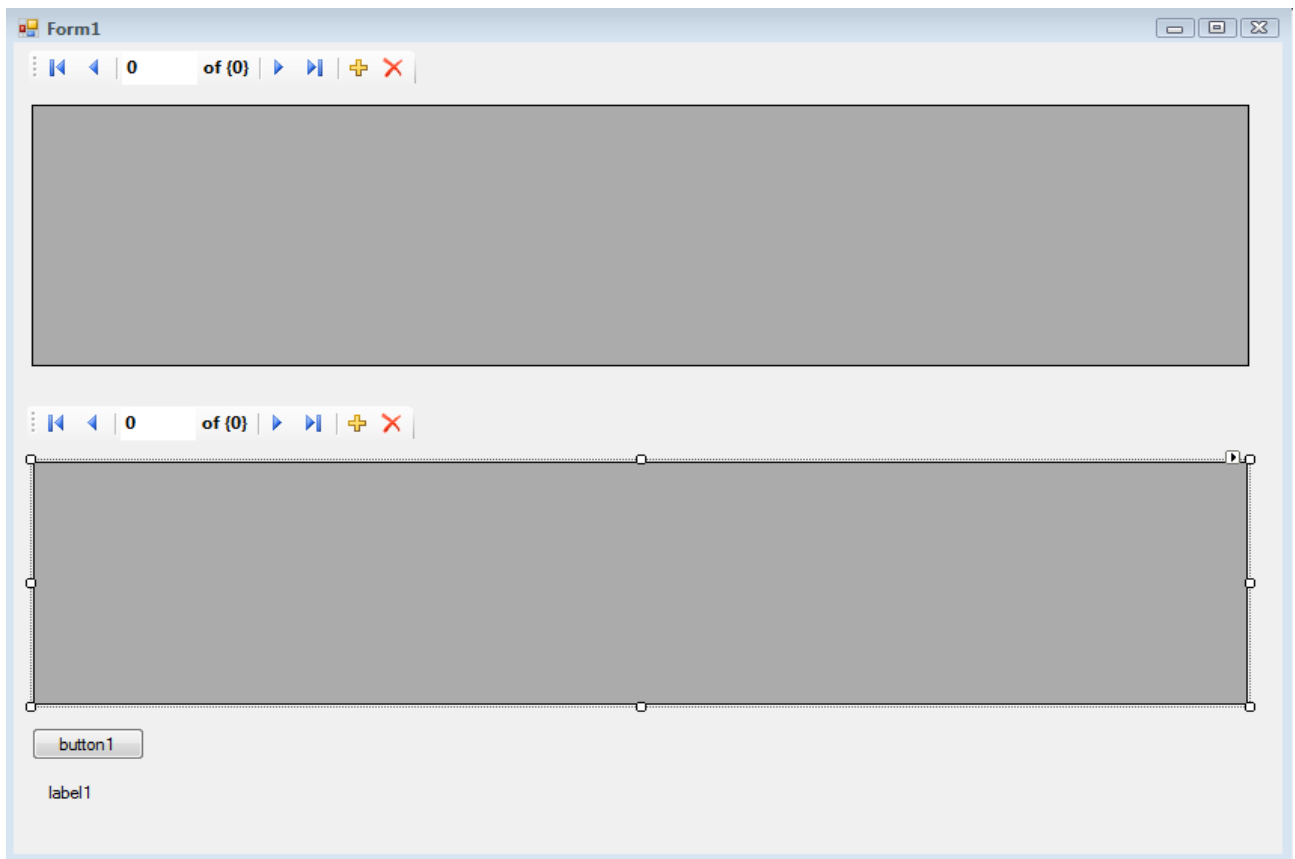
ნახ. 18.2.



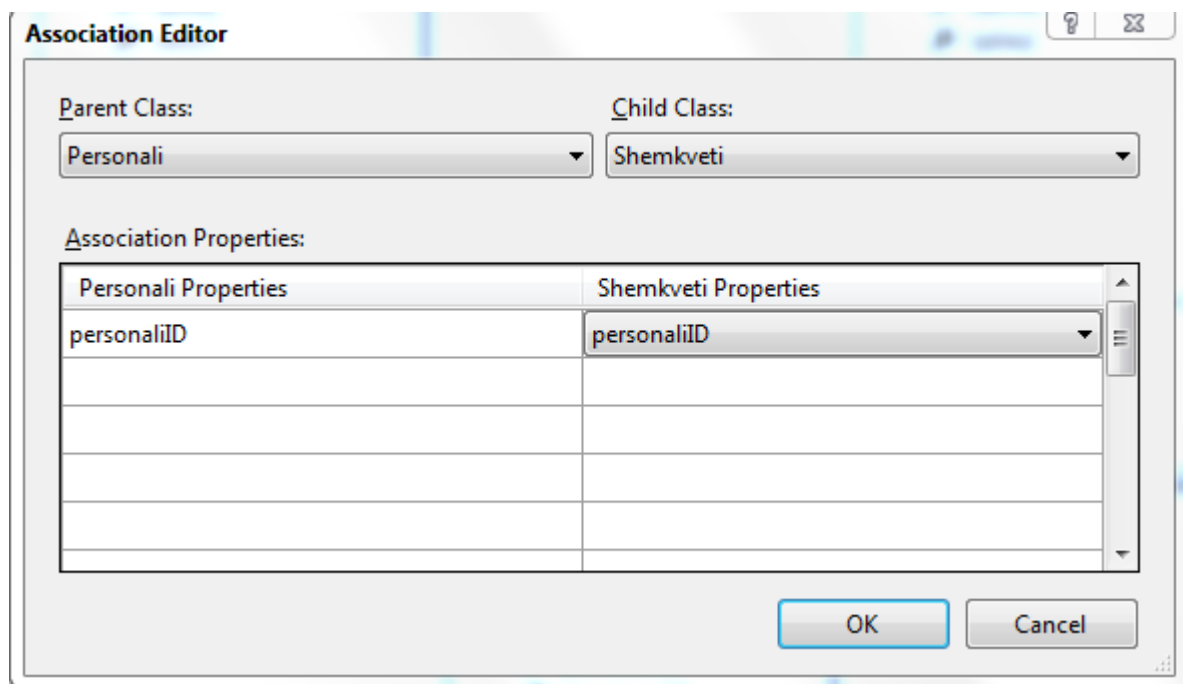
ბსბ. 18. 3.



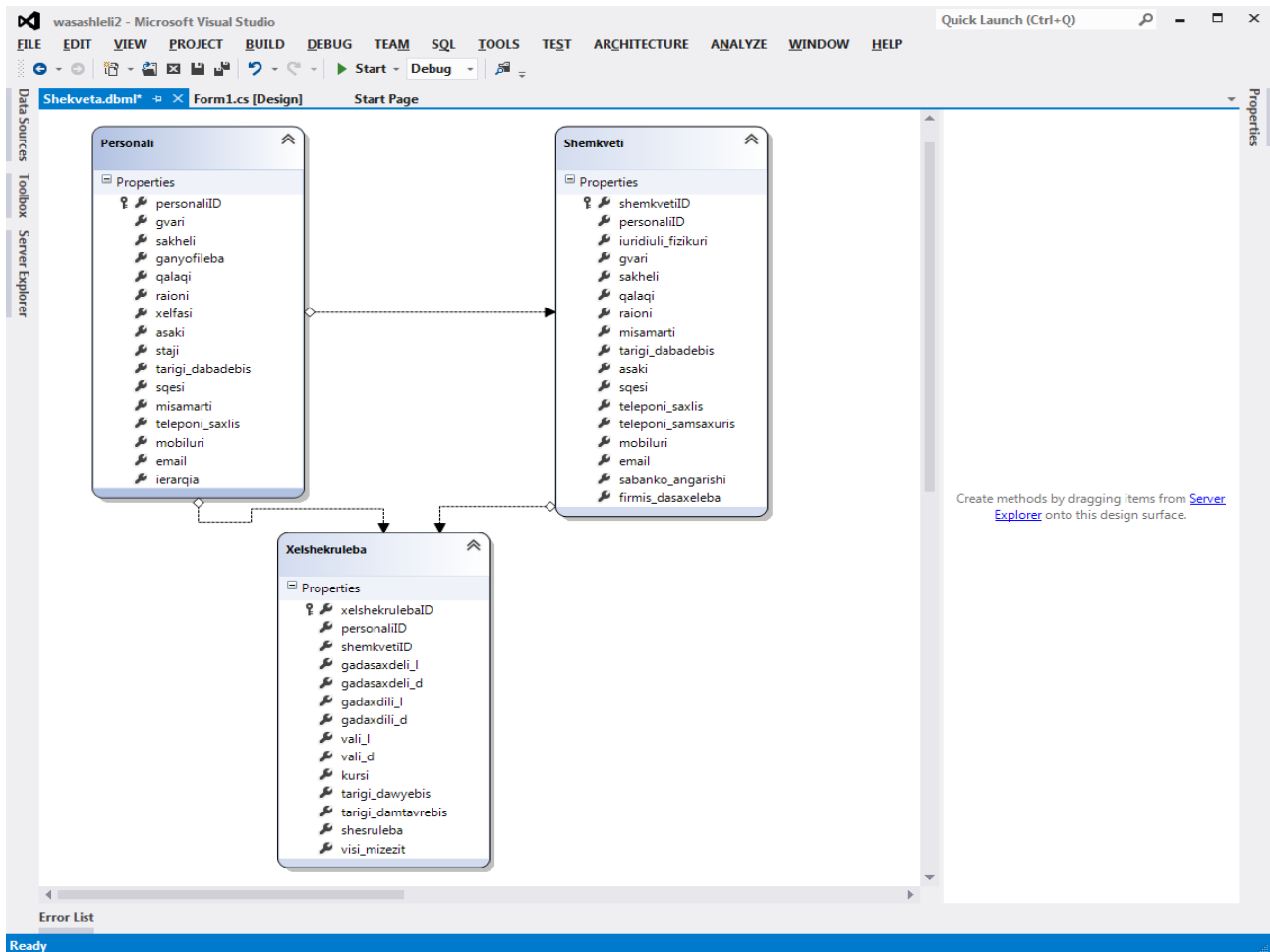
ბსბ. 18. 4.



ბსბ. 18.5.



ბსბ. 18.6.



ნახ. 18.7.

მოყვანილი პროგრამით ეკრანზე გამოგვაქვს მონაცემები თბილისელი თანამშრომლების შესახებ:

```

{
//   პროგრამა 18.14
//   პროგრამით ეკრანზე გამოჩნდება მონაცემები თბილისელი თანამშრომლების შესახებ
label1.Text = "";
ShekvetaDataContext shekvetaDataContext = new ShekvetaDataContext();
//   LINQ მოთხოვნის ფორმირება
var shedegebi = from cvladi in shekvetaDataContext.Personalis
                where cvladi.qalaqi == "თბილისი"
                select new
                {
                    PersonaliID = cvladi.personaliID,
                    Gvari = cvladi.gvari,
                    Ganyofileba = cvladi.ganyofileba,
                    Asaki = cvladi.asaki,
                    Qalaqi = cvladi.qalaqi,
                    Shemkvetis = cvladi.Shemkvetis
                };
//   ეკრანზე შედეგების გამოტანა
foreach (var element1 in shedegebi)

```

```

{
label1.Text += element1.PersonaliID.ToString() + " - " + element1.Gvari + " "
                    + element1.Ganyofileba + "\n";
foreach (Shemkveti element2 in element1.Shemkvetis)
    label1.Text += " " + element2.personaliID.ToString() + " - " + element2.gvari + "\n";
}
dataGridView1.DataSource = shedegebi;
}

```

ახლა დავთვალოთ Personali ცხრილის თითოეული თანამშრომლის შემკვეთების რაოდენობა და გამოვიტანოთ თითოეული ხელშეკრულების ვალის მნიშვნელობა. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```

{
// პროგრამა 18.15
//
label1.Text = "";
ShekvetaDataContext shekvetaDataContext = new ShekvetaDataContext();
// LINQ მოთხოვნის ფორმირება
var shedegebi = from cvladi in shekvetaDataContext.Personalis
                where cvladi.qalaqi == "თბილისი"
                select new
                {
                    PersonaliID = cvladi.personaliID,
                    Gvari = cvladi.gvari,
                    Ganyofileba = cvladi.ganyofileba,
                    Tariqi_Dabadebis = cvladi.tariqi_dabadebis,
                    Shemkvetis = cvladi.Shemkvetis
                };
// ეკრანზე შედეგების გამოტანა
foreach ( var element1 in shedegebi )
{
label1.Text += element1. PersonaliID.ToString() + " --- " + element1.Gvari + " " +
                element1. Ganyofileba + " " + element1.Tariqi_Dabadebis.Value.ToShortDateString() +
                " " + " შემკვეთების რაოდენობა = " + element1.Shemkvetis.Count.ToString() + "\n";
foreach ( Shemkveti element2 in element1.Shemkvetis )
    label1.Text += "ID = " + element2.shemkvetiID.ToString() + " ვალი = " +
                element2.Xelshekrulebas.Sum(tanxa => tanxa.vali_1).ToString() + "\n";
}
// ეკრანზე შედეგების გამოტანა
dataGridView1.DataSource = shedegebi;
}

```

Group და Orderby გაფართოებული მოთხოვნები

LINQ to SQL-ის გაფართოებული მოთხოვნებია: orderby და group.

orderby საშუალებას გვაძლევს შედეგები დავახარისხოთ ჩვენთვის საჭირო სვეტის მიხედვით. მოყვანილი პროგრამით ხდება შედეგების დახარისხება ზრდადობით gvari სვეტის მიხედვით:

```

{

```

```

//      პროგრამა 18.16
//
label1.Text = "";
ShekvetaDataContext shekvetaDataContext = new ShekvetaDataContext();
//      LINQ მოთხოვნის ფორმირება
var shedegebi = from cvladi in shekvetaDataContext.Personalis
                where cvladi.qalaqi == "თბილისი"
                orderby cvladi.gvari
                select cvladi;
//      ეკრანზე შედეგების გამოტანა
dataGridView1.DataSource = shedegebi;
//
foreach (var element1 in shedegebi)
    label1.Text += element1.gvari + " " + element1.ganyofileba + " " + element1.qalaqi + " " +
                element1.xelfasi.ToString() + "\n";
}
    orderby საშუალებას გვაძლევს, აგრეთვე შედეგები დავახარისხოთ რამდენიმე სვეტის
მიხედვით როგორც ზრდადობით, ისე კლებადობით. მაგალითად,
orderby cvladi.qalaqi, cvladi.gvari descending
კონსტრუქცია შედეგებს ახარისხებს qalaqi სვეტის მიხედვით ზრდადობით და gvari სვეტის
მიხედვით კლებადობით.
    group იძლევა მონაცემების დაჯგუფების საშუალებას ჩვენთვის საჭირო სვეტის მიხედვით.
ვიპოვოთ თითოეული თანამშრომლის ხელშეკრულებების თანხების ჯამი. პროგრამას ექნება
სახე:
{
//      პროგრამა 18.17
//
label1.Text = "";
ShekvetaDataContext shekvetaDataContext = new ShekvetaDataContext();
//      LINQ მოთხოვნის ფორმირება
var shedegebi1 = from cvladi1 in shekvetaDataContext.Personalis
                select new
                {
                    PersonaliID = cvladi1.personaliID,
                    Gvari = cvladi1.gvari,
                    Ganyofileba = cvladi1.ganyofileba,
                    Xelshekruleba = cvladi1.Xelshekrulebas
                };
var shedegebi2 = from cvladi2 in shedegebi1
                group cvladi2 by cvladi2.PersonaliID into cvladi3
                select new
                {
                    Tanxa = cvladi3.Sum(cvladi2 => cvladi2.Xelshekruleba.Sum( cvladi4 => cvladi4.gadasaxdeli_1)),
                    PersonaliID = cvladi3.Key,
                };
//      ეკრანზე შედეგების გამოტანა
dataGridView1.DataSource = shedegebi2;
//

```

```
foreach (var elementi2 in shedegebi2)
    label1.Text += elementi2.PersonaliID + " --- " + elementi2. Tanxa.ToString() + "\n";
}
```

როგორც ვხედავთ, შედეგში მონაცემების დაჯგუფება სრულდება PersonaliID სვეტის მნიშვნელობების მიხედვით.

მონაცემების მიბმა LINQ to SQL-თან

ცხრილებს შორის კავშირების დამყარება შესაძლებელია, აგრეთვე ფორმების ვიზუალური კონსტრუქტორების საშუალებით. ამისათვის, ვქმნით ShekvetaDataContext კლასს, მასში ვათავსებთ Personali, Shemkveti და Xelshekruleba ცხრილებს და ვქმნით მათ შორის ბმებს. ამის შემდეგ, ვირჩევთ მონაცემთა ახალ წყაროს, რისთვისაც ვასრულებთ Project მენიუს Add New Data Source ბრძანებას. გახსნილ ფანჯარაში (ნახ. 18.8) მოვნიშნავთ Object ელემენტს და ვაჭერთ Next კლავიშს. გახსნილ ფანჯარაში (ნახ. 18.9) ვირჩევთ Personali, ShekvetaDataContext, Shemkveti და Xelshekruleba ელემენტებს და ვაჭერთ Finish კლავიშს. ახლა ეკრანის მარცხენა ნაწილში გამოვაჩინოთ Data Sources ფანჯარა. გადავიდეთ ფორმაზე (Form1.cs [Design]), Data Sources ფანჯარაში გავხსნათ Personali სია და ავირჩიოთ Details ელემენტი როგორც ნაგულისხმევი მართვის ელემენტის ტიპი ამ ცხრილის შექმნისთვის. მიმთითებელი მოვათავსოთ Personali სტრიქონზე და ბუქსირის საშუალებით გადმოვათრიოთ ის ფორმაზე (ნახ. 18.10). ფორმაზე გამოჩნდება შესაბამისი ველები და შეიქმნება ორი ობიექტი: personaliBindingSource და personaliBindingNavigator. ამის შემდეგ, Data Sources ფანჯარიდან ფორმაზე ბუქსირის ოპერაციის გამოყენებით გადმოვათრიოთ Shemkvetის ცხრილი (ნახ. 18.11). ფორმაზე გამოჩნდება shemkvetისDataGridView კომპონენტი და შეიქმნება shemkvetისBindingSource ობიექტი. ამის შემდეგ, ფორმაზე ორჯერ სწრაფად დავაწკაპუნოთ და

```
public partial class Form1 : Form
{
```

სტრიქონების შემდეგ შევიტანოთ კოდი:

```
ShekvetaDataContext shekvetaDataContext = new ShekvetaDataContext();
```

ხოლო

```
private void Form1_Load(object sender, EventArgs e)
```

```
{
```

სტრიქონების შემდეგ კი კოდი:

```
personaliBindingSource.DataSource = shekvetaDataContext.Personalis;
```

მივიღებთ კოდს:

```
namespace WindowsFormsApplicatuin1
```

```
{
```

```
    public partial class Form1 : Form
```

```
    {
```

```
        ShekvetaDataContext shekvet1DataContext = new ShekvetaDataContext();
```

```
        public Form1()
```

```
        {
```

```
            InitializeComponent();
```

```
        }
```

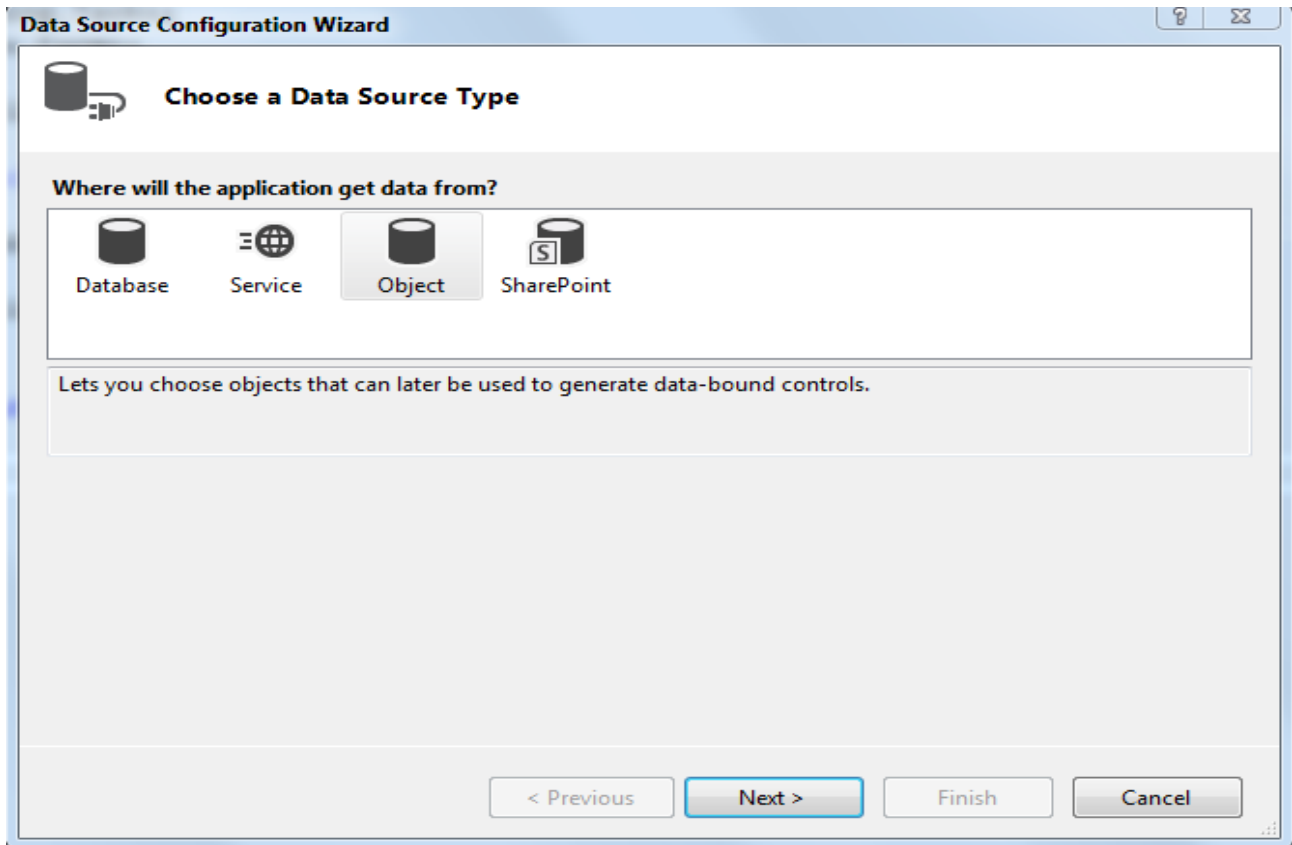
```
        private void Form1_Load(object sender, EventArgs e)
```

```
        {
```

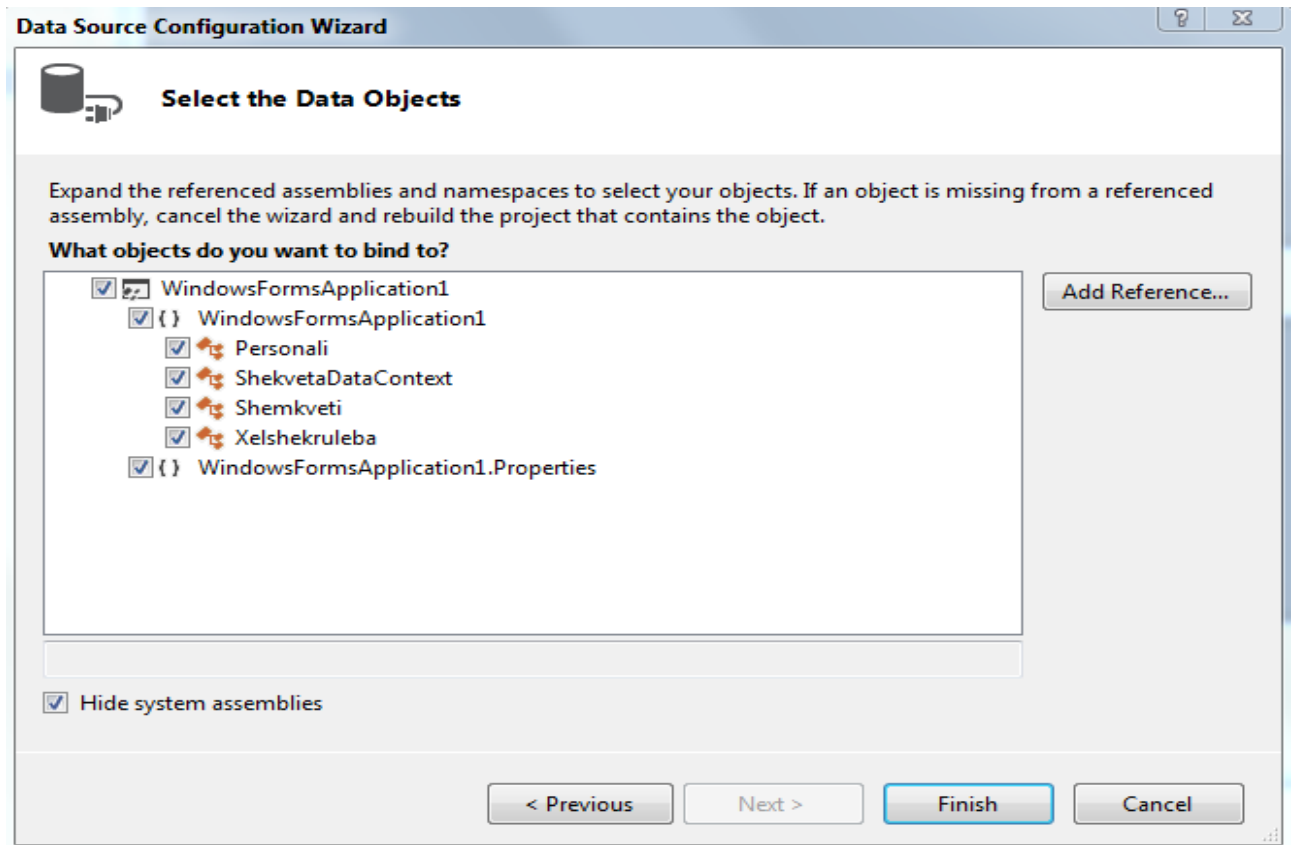
```
            personaliBindingSource.DataSource = shekvetaDataContext.Personalis;
```

```
        }
```

```
}  
}
```



6sb. 18.8.



6sb. 18.9.

Form1

0 of {0}

asaki:

email:

ganyofileba:

gvari:

ierarqia:

misamarti:

mobiluri:

personali ID:

qalaqi:

raioni:

sakheli:

sqesi:

staji:

targi dabadebis: 2013 წლის 04 09, მთხმანათი

teleponi saxlis:

xelfasi:

ნახ. 18.10.

Form1

0 of {0}

asaki:

email:

ganyofileba:

gvari:

ierarqia:

misamarti:

mobiluri:

personali ID:

qalaqi:

raioni:

sakheli:

sqesi:

staji:

targi dabadebis: 2013 წლის 04 09, მთხმანათი

teleponi saxlis:

xelfasi:

	shemkvetiID	personaliID	iuriduli_fizikuri	gvari	sakheli	qalaqi	raioni
*							

ნახ. 18.11.

ახლა შევასრულოთ პროექტი. ნავიგაციის დროს, ველებში აისახება Personali ცხრილის სტრიქონები, shemkvetisDataGridView კომპონენტში კი - Shemkveti ცხრილის სტრიქონები.

ორივე ცხრილში მონაცემების შესაცვლელად, personaliBindingNavigator კომპონენტში მოვნიშნოთ Save კლავიში და Properties ფანჯრის Enabled თვისებას მივანიჭოთ True მნიშვნელობა. ორჯერ სწრაფად დავაწკაპუნოთ ამ კლავიშზე და შევიტანოთ კოდი: shekvetaDataContext.SubmitChanges();

ADO.NET და LINQ

LINQ მოთხოვნები შეგვიძლია, აგრეთვე გამოვიყენოთ ADO.NET-ის მიმართ. ასეთი მიდგომა სასარგებლოა მაშინ, როცა პროგრამას ჭირდება შეასრულოს გაფართოებული მანიპულაციები გამორთულ მონაცემთა ნაკრებთან. გარდა ამისა, LINQ უმატებს მოთხოვნების ოპერატორების სრულ ნაკრებს იმ მოთხოვნების შეზღუდულ შესაძლებლობებს, რომლებიც ჩადგმულია DataSet საწყის კლასში.

დავუშვათ, გვინტერესებს იმ თანამშრომლების გვარები, რომელთა ასაკი აღემატება 30 წელს. პროგრამას ექნება სახე:

```
{
//   პროგრამა 18.18
//
label1.Text = "";
string connectionstring = "server=ROMANI-PC; database=Shekveta; uid=sa; pwd=paroli";
SqlConnection mySqlConnection = new SqlConnection(connectionstring);
string selectString = "SELECT * FROM Personali ";
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
mySqlCommand.CommandText = selectString;
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
DataSet myDataSet = new DataSet();
mySqlConnection.Open();
myDataSet.Clear();
mySqlDataAdapter.Fill(myDataSet, "Personali");
//   LINQ მოთხოვნის ფორმირება
var Personalis = myDataSet.Tables["Personali"].AsEnumerable();
var shedegi_Personali = from cvladi in Personalis
                        where Convert.ToInt32(cvladi["asaki"]) > 30
                        orderby cvladi["gvari"] descending
                        select cvladi;
//   ეკრანზე შედეგების გამოტანა
foreach (var cvladi1 in shedegi_Personali)
    label1.Text += cvladi1["gvari"] + " " + cvladi1["asaki"].ToString() + "\n";
mySqlConnection.Close();
}
```

დავუშვათ, გვინტერესებს ის თანამშრომლები, რომელთაც ერთზე მეტი შემკვეთი ჰყავთ. პროგრამას ექნება სახე:

```
{
```

```

//      პროგრამა 18.19
//
label1.Text = "";
string connectionString = "server=ROMANI-PC;database=Shekveta;uid=sa;pwd=paroli";
string selectString1 = "SELECT * FROM PersonalI";
string selectString2 = "SELECT * FROM Shemkveti";
SqlConnection mySqlConnection = new SqlConnection(connectionString);
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
mySqlCommand.CommandText = selectString1;
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
DataSet myDataSet = new DataSet();
mySqlConnection.Open();
mySqlDataAdapter.Fill(myDataSet, "PersonalI");
mySqlCommand.CommandText = selectString2;
mySqlDataAdapter.SelectCommand = mySqlCommand;
mySqlDataAdapter.Fill(myDataSet, "Shemkveti");
//
DataColumn parentColumn = myDataSet.Tables["PersonalI"].Columns["personalIID"];
DataColumn childColumn = myDataSet.Tables["Shemkveti"].Columns["personalIID"];
//      რელაციის შექმნა
DataRelation PersonalI_Shemkveti =
                new DataRelation("PersonalI_Shemkveti", parentColumn, childColumn);
//
myDataSet.Relations.Add(PersonalI_Shemkveti);
//
BindingSource bindingSource4 = new BindingSource(myDataSet, "PersonalI");
BindingSource bindingSource5 = new BindingSource(myDataSet, "Shemkveti");
bindingSource5.DataSource = bindingSource4;
bindingSource5.DataMember = "PersonalI_Shemkveti";
//
var Relation_Parents = myDataSet.Tables["PersonalI"].AsEnumerable();
var Relation_Childes = myDataSet.Tables["Shemkveti"].AsEnumerable();
//      LINQ მოთხოვნის ფორმირება
var shedegi_Relation_Parent = from cvladi in Relation_Parents
                                where cvladi.GetChildRows("PersonalI_Shemkveti").Length > 1
                                orderby cvladi.GetChildRows("PersonalI_Shemkveti").Length
                                select cvladi;
//      ეკრანზე შედეგების გამოტანა
foreach (var cvladi1 in shedegi_Relation_Parent)
{
label1.Text += cvladi1["gvari"] + " ხელფასი = " + cvladi1["xelfasi"].ToString() + "\n";
label1.Text += " შემკვეთების რაოდენობა = " +
                cvladi1.GetChildRows("PersonalI_Shemkveti").Length.ToString() + "\n";
}
mySqlConnection.Close();
}

```


Personali_Shemkveti ობიექტი გადაეცემა GetChildRows() მეთოდს, რომელიც გასცემს DataRowCollection ობიექტს, რომელიც შეიცავს მოცემულ თანამშრომელთან დაკავშირებულ Shemkveti ცხრილის ობიექტებს.

დანართი 1. სავარჯიშოები თავების მიხედვით

თავი 2. C# ენის საფუძვლები

არიტმეტიკული გამოსახულებები

შეადგინეთ პროგრამა, რომელიც გამოთვლის ქვემოთ მოყვანილი ფორმულების მნიშვნელობებს:

$$2.1.1. \mathbf{y} = \mathbf{1} + \mathbf{x} + \frac{\mathbf{x}^2}{2}$$

$$2.1.2. \mathbf{y} = \frac{\mathbf{x}^2 + \mathbf{z}^2}{\mathbf{1} - \frac{\mathbf{x}^2 - \mathbf{z}^2}{2}}$$

$$2.1.3. \mathbf{y} = \sqrt{\mathbf{1} + \sqrt{|\mathbf{x}|}}$$

$$2.1.4. \mathbf{y} = (\mathbf{1} + \mathbf{x}) \cdot \frac{\mathbf{x} + \frac{\sqrt{\mathbf{x}-1}}{4}}{\mathbf{e}^{\mathbf{x}-1}}$$

$$2.1.5. \mathbf{y} = \mathbf{\sin x}^3 + (\mathbf{\cos x}^4)^3 - \mathbf{e}^{\sqrt{\mathbf{x}}}$$

$$2.1.6. \mathbf{y} = \mathbf{1} + |\mathbf{x}| + |\mathbf{1} + \mathbf{x}|$$

$$2.1.7. \mathbf{y} = \frac{|\mathbf{x}| - |\mathbf{z}|}{\mathbf{1} + \mathbf{\ln x}}$$

შეადგინეთ პროგრამა, რომელიც:

2.1.6. გამოთვლის დენის ძალას, $\mathbf{J} = \frac{\mathbf{U}}{\mathbf{R}}$.

2.1.7. გამოთვლის ხუთი რიცხვის საშუალო არითმეტიკულს.

2.1.8. გამოთვლის ოთხი რიცხვის საშუალო გეომეტრიულს.

2.1.9. გამოთვლის მართკუთხა სამკუთხედის ფართობს, $\mathbf{S} = \frac{\mathbf{ah}}{2}$.

2.1.10. გამოთვლის სამკუთხედის ფართობს ჰერონის ფორმულის გამოყენებით,

$$\mathbf{S} = \sqrt{\mathbf{p(p - a)(p - b)(p - c)}}.$$

2.1.11. გამოთვლის სამკუთხედის პერიმეტრს, $\mathbf{P} = \mathbf{a} + \mathbf{b} + \mathbf{c}$.

2.1.12. გამოთვლის კვადრატის ფართობს.

2.1.13. გამოთვლის მართკუთხედის პერიმეტრს.

2.1.14. გამოთვლის წრეწირის სიგრძეს, $\mathbf{C} = \mathbf{2 \cdot \pi r}$.

2.1.15. გამოთვლის წრეწირის ფართობს, $\mathbf{S} = \mathbf{\pi r^2}$.

2.1.16. გამოთვლის ტრაპეციის ფართობს, $\mathbf{S} = \frac{\mathbf{a+b}}{2} \cdot \mathbf{h}$.

2.1.17. გამოთვლის ტრაპეციის პერიმეტრს.

2.1.18. გამოთვლის მართკუთხედის დიაგონალს, $\mathbf{d} = \sqrt{\mathbf{a^2 + b^2}}$.

2.1.19. გამოთვლის რომბის ფართობს, $S = a^2 \sin \alpha$.

2.1.20. გამოთვლის კვადრატის დიაგონალს, $d = a\sqrt{2}$.

2.1.21. გამოთვლის პარალელოგრამის ფართობს, $S = ab \sin \alpha$.

2.1.22. იპოვის A რიცხვის P პროცენტს, $B = \frac{AP}{100}$.

2.1.23. იპოვის A რიცხვს, თუ მისი P პროცენტი B რიცხვის ტოლია, $A = \frac{B \cdot 100}{P}$.

2.1.24. 1 ფუტი = 30,48 სმ. შეადგინეთ პროგრამა, რომელიც ფუტს გადაიყვანს სანტიმეტრში და პირიქით.

2.1.25. 1 ფუნტი = 409,5 გრამი. შეადგინეთ პროგრამა, რომელიც ფუნტს გადაიყვანს გრამში და პირიქით.

2.1.26. 1 დუიმი = 2,54 სმ. შეადგინეთ პროგრამა, რომელიც დუიმს გადაიყვანს სანტიმეტრში და პირიქით.

2.1.27. 1 იარდი = 0,9144 მ. შეადგინეთ პროგრამა, რომელიც იარდს გადაიყვანს მეტრში და პირიქით.

2.1.28. 1 არშინი = 0,711 მ. შეადგინეთ პროგრამა, რომელიც არშინს გადაიყვანს მეტრში და პირიქით.

2.1.29. 1 საჟენი = 2,1336 მ. შეადგინეთ პროგრამა, რომელიც საჟენს გადაიყვანს მეტრში და პირიქით.

2.1.30. შეადგინეთ პროგრამა, რომელიც დაადგენს a, b და c მონაკვეთებით შეიძლება თუ არა სამკუთხედის აგება. სამი მონაკვეთით სამკუთხედის აგებისთვის აუცილებელია, რომ ნებისმიერი ორი გვერდის ჯამი მესამეს აღემატებოდეს.

2.1.31. შეადგინეთ პროგრამა, რომელიც დაადგენს a, b და c მონაკვეთებით შეიძლება თუ არა მართკუთხა სამკუთხედის აგება. მართკუთხა სამკუთხედი, თუ მისი გვერდები აკმაყოფილებენ პითაგორას თეორემას: $a^2 + b^2 = c^2$.

ლოგიკური გამოსახულებები

შეადგინეთ პროგრამა, რომელიც გამოთვლის ქვემოთ მოყვანილი ლოგიკური გამოსახულებების მნიშვნელობებს:

2.2.1. $y = x1 \& \&x2 \parallel x3 \& \&!x4$, სადაც $x1=true, x2=false, x3=true, x4=false$

2.2.2. $y = x1 \parallel !x2 \& \&x3 \parallel x4$, სადაც $x1=false, x2=false, x3=true, x4=true$

2.2.3. $y = !x1 \& \&!x2 \& \&x3 \parallel x4$, სადაც $x1=true, x2=true, x3=false, x4=false$

2.2.4. $y = !x1 \parallel x2 \parallel !x3 \& \&x4$, სადაც $x1=false, x2=true, x3=true, x4=false$

თავი 3. მმართველი ოპერატორები

if ოპერატორი

შეადგინეთ პროგრამა, რომელიც:

3.1.1. დაადგენს რიცხვი დადებითია თუ უარყოფითი. რიცხვი textBox კომპონენტიდან შეიტანეთ. label კომპონენტში გამოიტანეთ შესაბამისი შეტყობინება.

3.1.2. დაადგენს რიცხვი კენტია თუ ლუწი.

3.1.3. დაადგენს რიცხვი არის თუ არა 5-ის ჯერადი.

3.1.4. დაადგენს რიცხვი არის თუ არა 30-ის ტოლი.

3.1.5. დაადგენს რიცხვი მეტია თუ არა 10-ზე.

- 3.1.6. დაადგენს რიცხვი ნაკლებია თუ არა -2,01-ზე.
- 3.1.7. დაადგენს რიცხვი არის თუ არა მეტი ან ტოლი 15-ზე.
- 3.1.8. დაადგენს რიცხვი არის თუ არა ნაკლები ან ტოლი 23-ზე.
- 3.1.9. ორ რიცხვს შორის იპოვის მაქსიმალურს.
- 3.1.10. ორ რიცხვს შორის იპოვის მინიმალურს.

ჩადგმული if ოპერატორი

შეადგინეთ პროგრამა, რომელიც:

- 3.2.1. სამ რიცხვს შორის იპოვის მაქსიმალურს.
- 3.2.2. სამ რიცხვს შორის იპოვის მინიმალურს.
- 3.2.3. დაადგენს ორ რიცხვს შორის არის თუ არა დადებითი.
- 3.2.4. დაადგენს ორ რიცხვს შორის არის თუ არა უარყოფითი.
- 3.2.5. დაადგენს ორ რიცხვს შორის არის თუ არა 7-ის ჯერადი.
- 3.2.6. დაადგენს სამ რიცხვს შორის არის თუ არა ლუწი.
- 3.2.7. დაადგენს სამ რიცხვს შორის არის თუ არა დადებითი.
- 3.2.8. დაადგენს სამ რიცხვს შორის არის თუ არა უარყოფითი.
- 3.2.9. დათვლის სამ რიცხვს შორის რამდენია კენტი.
- 3.2.10. დათვლის სამ რიცხვს შორის რამდენია 5-ის ტოლი.
- 3.2.11. დათვლის სამ რიცხვს შორის რამდენია 10-ზე მეტი.
- 3.2.12. ორ რიცხვს შორის რომელია 5-ის ჯერადი.
- 3.2.13. სამ რიცხვს შორის რომელია 50-ზე ნაკლები.

ლოგიკა

შეადგინეთ პროგრამა, რომელიც:

- 3.3.1. დაადგენს x რიცხვი არის თუ არა $0 < x \leq 17$ დიაპაზონში მოთავსებული.
- 3.3.2. დაადგენს x რიცხვი არის თუ არა $10 \leq x < 19$ დიაპაზონში მოთავსებული.
- 3.3.3. დაადგენს x რიცხვი არის თუ არა 25-ზე ნაკლები ან 100-ზე მეტი.
- 3.3.4. დაადგენს x რიცხვი არის თუ არა 30-ის ტოლი ან 5-ზე ნაკლები.
- 3.3.5. დაადგენს ორივე რიცხვი არის თუ არა დადებითი.
- 3.3.6. დაადგენს სამივე რიცხვი არის თუ არა 7-ის ჯერადი.

for, while, do-while ოპერატორები

შეადგინეთ პროგრამა, რომელიც გამოთვლის შემდეგი გამოსახულებების მნიშვნელობებს:

- 3.4.1. $y = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + \dots + n^2$
- 3.4.2. $y = 1^1 + 2^2 + 3^3 + 4^4 + 5^5 + \dots + n^n$
- 3.4.3. $y = 1 + 3 + 5 + \dots + (2n - 1)$
- 3.4.4. $y = 2 + 4 + 6 + \dots + 2n$
- 3.4.5. $y = 1 + 3 + 5 + \dots + (2n + 1)$
- 3.4.6. $y = 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \dots + n \cdot 2^n$
- 3.4.7. $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$

$$3.4.8. y = a(a+1)(a+2) \dots (a+n-1)$$

$$3.4.9. y = (a-2)(a-4) \dots (a-2n)$$

3.4.10. ფობონახის რიცხვები: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$3.4.11. y = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \dots$$

$$3.4.12. y = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots + \frac{1}{n}$$

$$3.4.13. y = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \dots$$

$$3.4.14. y = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9}$$

$$3.4.15. y = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2}$$

$$3.4.16. y = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2}$$

$$3.4.17. y = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \dots + \frac{1}{(n-1) \cdot n}$$

$$3.4.18. y = \frac{1 \cdot 2}{3 \cdot 4} + \frac{3 \cdot 4}{5 \cdot 6} + \frac{5 \cdot 6}{7 \cdot 8} + \frac{n \cdot (n+1)}{(n+2) \cdot (n+3)}$$

$$3.4.19. y = \frac{1}{a} + \frac{1}{a^2} + \frac{1}{a^4} + \dots + \frac{1}{a^{2n}}$$

$$3.4.20. y = \frac{1}{1 \cdot 3} + \frac{1}{2 \cdot 4} + \frac{1}{3 \cdot 5} + \dots + \frac{1}{n \cdot (n+2)}$$

$$3.4.21. y = \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \frac{1}{3 \cdot 4 \cdot 5} + \dots + \frac{1}{n \cdot (n+1) \cdot (n+2)}$$

$$3.4.22. y = \left(1 + \frac{1}{1^2}\right) \left(1 + \frac{1}{2^2}\right) \dots \left(1 + \frac{1}{n^2}\right)$$

$$3.4.23. y = \left(1 + \frac{1}{1^n}\right) \left(1 + \frac{1}{2^n}\right) \dots \left(1 + \frac{1}{n^n}\right)$$

$$3.4.24. y = \left(1 + \frac{1}{1}\right)^n \left(1 + \frac{1}{2}\right)^n \dots \left(1 + \frac{1}{n}\right)^n$$

$$3.4.25. y = 1 - x + x^2 - x^3 + x^4 - x^5$$

$$3.4.26. y = 1 + x + x^2 + x^3 + x^4 + x^5$$

$$3.4.27. y = x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} + \dots$$

$$3.4.28. y = 1 + \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \frac{1}{x^{2n-1}}$$

$$3.4.29. y = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5}$$

$$3.4.30. y = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \frac{x^5}{5}$$

$$3.4.31. y = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9}$$

$$3.4.32. y = 1 + \frac{1}{2}x - \frac{1}{4}x^2 + \frac{1}{8}x^3$$

$$3.4.33. y = 1 - \frac{1}{2}x + \frac{3}{4}x^2 - \frac{5}{8}x^3$$

$$3.4.34. y = x + \frac{1}{2 \cdot 3}x^3 + \frac{1 \cdot 3}{3 \cdot 4 \cdot 5}x^5$$

$$3.4.35. y = \frac{x-1}{x} + \frac{(x-1)^2}{2 \cdot x^2} + \frac{(x-1)^3}{3 \cdot x^3} + \dots + \frac{(x-1)^n}{n \cdot x^n}$$

$$3.4.36. y = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7}\right)$$

შეადგინეთ პროგრამა, რომელიც:

3.4.37. დათვლის ლუწი რიცხვების რაოდენობას 5-დან 27-მდე დიაპაზონში.

3.4.38. შეკრებს კენტ რიცხვებს 10-დან 20-მდე დიაპაზონში.

3.4.39. დათვლის 8-ის ჯერადი რიცხვების რაოდენობას 20-დან 50-მდე დიაპაზონში.

3.4.40. იპოვის 10-დან 70-მდე იმ რიცხვების ჯამს, რომლებიც არ არიან 8-ს ჯერადი.

3.4.41. შეკრებს ლუწ რიცხვებს 1-დან 20-მდე დიაპაზონში.

3.4.42. დათვლის კენტ რიცხვებს 1-დან 20-მდე დიაპაზონში.

3.4.43. დათვლის 4-სა და 7-ის ჯერადი რიცხვების რაოდენობას 1-დან 100-მდე დიაპაზონში.

3.4.44. იპოვის ყველა სამნიშნა რიცხვს დიაპაზონში 200÷500, რომელიც უნაშთოდ იყოფა საკუთარი ციფრების ჯამზე.

3.4.45. იპოვის მოცემული მთელი რიცხვის ციფრების ჯამს.

3.4.46. მოცემული მთელი მნიშვნელობის მქონე ფართობისათვის იპოვის ყველა განსხვავებულ ტოლდიდ მართკუთხედს, რომელთა გვერდებიც მთელი რიცხვებია. მართკუთხედებს ეწოდებათ ტოლდიდი, თუ მათი ფართობები ტოლია. გვერდების გაცვლით მიღებული მართკუთხედები განსხვავებულად არ ჩაითვლება.

3.4.47. მოცემულ ნატურალურ რიცხვს ($n < 1000$) მარტივ მამრავლებად დაშლის.

3.4.48. იპოვის ორი მოცემული ნატურალური რიცხვის უდიდესი საერთო გამყოფს.

continue, break ოპერატორები

შეადგინეთ პროგრამა, რომელიც:

3.5.1. შეკრებს 1-დან 20-მდე რიცხვებს მანამ, სანამ ჯამი არ გახდება 24-ზე მეტი.

3.5.2. დათვლის 4-ის ჯერადი რიცხვებს 6-დან 79-მდე დიაპაზონში მანამ, სანამ 4-ის ჯერადი რიცხვი არ გახდება 50-ზე მეტი.

3.5.3. შეკრებს 9-ის ჯერად რიცხვებს 2-დან 47-მდე დიაპაზონში მანამ, სანამ 9-ის ჯერადი რიცხვი იქნება 30-ზე ნაკლები.

თავი 4. მასივები, სტრიქონები და ბიტობრივი ოპერაციები

ერთგანზომილებიანი მასივები

შეადგინეთ პროგრამა, რომელიც:

4.1.1. იპოვის მასივის ელემენტების ჯამს.

4.1.2. იპოვის მასივის ელემენტების ნამრავლს.

4.1.3. იპოვის მასივის ელემენტების კვადრატების ჯამს.

4.1.4. იპოვის მასივის მინიმალურ ელემენტს.

4.1.5. იპოვის მასივის მაქსიმალური ელემენტის ინდექსს.

4.1.6. იპოვის მასივის ლუწი ელემენტების ჯამს.

4.1.7. იპოვის მასივის კენტი ელემენტების რაოდენობას.

4.1.8. იპოვის მასივის დადებითი ელემენტების ჯამს.

4.1.9. იპოვის მასივის უარყოფითი ელემენტების რაოდენობას.

4.1.10. იპოვის მასივის არანულოვანი ელემენტების რაოდენობას.

4.1.11. იპოვის მასივის ნულოვანი ელემენტების რაოდენობას.

4.1.12. იპოვის მასივის ლუწი ინდექსის მქონე ელემენტების ჯამს.

4.1.13. იპოვის მასივის კენტი ინდექსის მქონე ელემენტების რაოდენობას.

4.1.14. იპოვის მაქსიმალურ სხვაობას M_{10} და N_{10} მასივების ერთნაირი ინდექსის მქონე ელემენტებს შორის.

4.1.15. იპოვის მასივის პირველ უარყოფით ელემენტს და მის ინდექსს.

4.1.16. იპოვის მასივის პირველი დადებითი ელემენტის ინდექსს.

4.1.17. იპოვის მასივის უარყოფითი ელემენტებიდან უდიდესის ინდექსს.

4.1.18. გამოთვლის მასივის ელემენტების საშუალო არითმეტიკულს.

4.1.19. გამოთვლის მასივის 3-ის ჯერადი ელემენტების საშუალო არითმეტიკულს.

4.1.20. დათვლის მასივის იმ ელემენტების რაოდენობას, რომლებიც მეტია x რიცხვზე.

4.1.21. იპოვის მასივის იმ ელემენტების ჯამს, რომლებიც ნაკლებია x რიცხვზე.

4.1.22. განსაზღვრავს თუ რამდენი ელემენტია მოთავსებული მასივის ორ მოცემულ ელემენტს შორის.

4.1.23. იპოვის მასივის დადებითი ელემენტებიდან უმცირესს.

4.1.24. იპოვის მასივის უარყოფითი ელემენტებიდან უდიდესს.

4.1.25. იპოვის მასივის იმ ელემენტების ჯამს, რომლებიც მოთავსებულია ორ მოცემულ მნიშვნელობას შორის.

4.1.26. იპოვის იმ ელემენტებს შორის მაქსიმალურს, რომლებიც მოთავსებულია მასივის ორ მოცემულ ელემენტს შორის.

- 4.1.27. დათვლის მასივის იმ ელემენტების რაოდენობას, რომელთა მნიშვნელობები 20-ზე მეტია.
- 4.1.28. ერთი მასივიდან მეორეში გადაწერს იმ ელემენტებს, რომელთა მნიშვნელობები 20-ზე ნაკლებია.
- 4.1.29. მასივში იპოვის იმ ელემენტების რაოდენობას, რომელთა მნიშვნელობები აღემატება საკუთარ ინდექსს.
- 4.1.30. მასივის დადებით ელემენტებს გადაწერს მეორე მასივში.
- 4.1.31. მასივის 5-ის ჯერად ელემენტებს გადაწერს მეორე მასივში.
- 4.1.32. მასივის არანულოვან ელემენტებს გადაწერს მეორე მასივში.
- 4.1.33. მასივის კენტი ინდექსის მქონე ელემენტებს გადაწერს მეორე მასივში.
- 4.1.34. მასივის ლუწი ინდექსის მქონე ელემენტებს გადაწერს მეორე მასივში.
- 4.1.35. მასივის ლუწინდექსიან ელემენტებს გადაწერს მეორე მასივში, ხოლო კენტინდექსიან ელემენტებს კი - მესამე მასივში.
- 4.1.36. მასივის დადებითი ელემენტების ინდექსებს გადაწერს N_{10} მასივში.
- 4.1.37. მასივის ლუწ ელემენტებს გადაწერს მეორე მასივში.
- 4.1.38. მასივის კენტ ელემენტებს გადაწერს მეორე მასივში.
- 4.1.39. მასივის 3-ის ჯერადი ელემენტების ინდექსებს გადაწერს მეორე მასივში.
- 4.1.40. მოახდენს ზრდადობის მიხედვით დალაგებული ორი მასივის შერწყმას და შექმნის ახალ ზრდადობის მიხედვით დალაგებულ მესამე მასივს.
- 4.1.41. მასივში განსაზღვრავს გაორმაგებული კენტი რიცხვების რაოდენობას და ჩაწერს მათ მეორე მასივში.
- 4.1.42. მასივის იმ ელემენტებს, რომლებიც 5-ზე გაყოფისას იძლევიან 1-დან 4-ის ტოლ ნაშთს, გადაწერს მეორე მასივში.
- 4.1.43. მასივის უარყოფით ელემენტებს შეცვლის 0-ებით.
- 4.1.44. მასივის კენტ ელემენტებს შეცვლის მათი კვადრატებით.
- 4.1.45. მასივის ელემენტებს უკუ (შებრუნებული) თანმიმდევრობით დაალაგებს.
- 4.1.46. მასივის ელემენტებს დაალაგებს ისე, რომ დასაწყისში განლაგდეს ყველა უარყოფითი რიცხვი რიგითობის შენარჩუნებით, შემდეგ კი დადებითი რიცხვები რიგითობის შენარჩუნებით. დამხმარე მასივი არ გამოიყენოთ.
- 4.1.47. შექმნის მეორე მასივს, რომელშიც ჯერ მოთავსებული იქნება პირველი მასივის ნულოვანი ელემენტები, შემდეგ კი - არანულოვანი.
- 4.1.48. მასივში ელემენტებს დაალაგებს ისე, რომ ჯერ მოთავსდეს არანულოვანი ელემენტები, შემდეგ კი - ნულოვანი.
- 4.1.49. მასივის ელემენტებს ციკლურად დამრავს მარჯვნივ n ელემენტით.
- 4.1.50. მასივის ელემენტებს ციკლურად დამრავს მარცხნივ n ელემენტით.
- 4.1.51. მასივში ადგილებს შეუცვლის კენტი და ლუწი ინდექსის მქონე ელემენტებს.
- 4.1.52. შეამოწმებს ემთხვევა თუ არა მასივის პირველი ნახევარი მის მეორე ნახევარს.
- 4.1.53. შეამოწმებს განსხვავდება თუ არა ორი მოცემული მასივი.
- 4.1.54. განსაზღვრავს შეიცავს თუ არა ერთი მასივი მეორეს.
- 4.1.55. განსაზღვრავს მასივის ერთმანეთის მიყოლებით მდებარე ნულის ტოლი ელემენტების ყველაზე გრძელ მიმდევრობას.
- 4.1.56. დათვლის მასივის განსხვავებული ელემენტების რაოდენობას.
- 4.1.57. დაადგენს არის თუ არა მასივში ერთნაირი ელემენტები.
- 4.1.58. მასივში იპოვის ზრდადობით დალაგებულ ყველაზე გრძელ მიმდევრობას.
- 4.1.59. მასივში იპოვის კლებადობით დალაგებულ ყველაზე მოკლე მიმდევრობას.
- 4.1.60. დათვლის მასივის მეზობელ ელემენტებს შორის ნიშანცვლათა რაოდენობას.
- 4.1.61. იპოვის მასივის იმ ელემენტების ინდექსებს, რომელთა შორის სხვაობა უდიდესია.
- 4.1.62. იპოვის მასივის იმ ელემენტების ინდექსებს, რომელთა შორის სხვაობა უმცირესია.

- 4.1.63. მასივში იპოვის იმ 5-ელემენტთან მიმდევრობას, რომლის ელემენტების ჯამი მინიმალურია.
- 4.1.64. მასივში იპოვის: ა. კენტი ინდექსის მქონე რიცხვებს შორის უდიდესს; ბ. ლუწი ინდექსის მქონე რიცხვებს შორის უმცირესს.
- 4.1.65. დაადგენს შეიცავს თუ არა მასივი უარყოფითი ლუწი რიცხვს.
- 4.1.66. დაადგენს შეიცავს თუ არა მასივი 9-ის ჯერად რიცხვს.
- 4.1.67. დაადგენს არის თუ არა მასივი დალაგებული ზრდადობით.
- 4.1.68. დაადგენს არის თუ არა მასივი დალაგებული კლებადობით.
- 4.1.69. დაადგენს არის თუ არა მასივში მეზობლად მდებარე ორი ერთნაირი ელემენტი.
- 4.1.70. დაადგენს არის თუ არა მასივში ყველა ელემენტი ერთნაირი.
- 4.1.71. დაადგენს რომელი რიცხვი გვხდება მასივში ყველაზე მეტად და რამდენჯერ.

ორგანზომილებიანი მასივები

შეადგინეთ პროგრამა, რომელიც:

- 4.2.1. იპოვის მასივის მაქსიმალურ ელემენტს.
- 4.2.2. იპოვის მასივის მინიმალური ელემენტის ინდექსებს.
- 4.2.3. იპოვის მასივის ელემენტების ჯამს.
- 4.2.4. იპოვის მასივის ელემენტების ნამრავლს.
- 4.2.5. იპოვის მასივის დადებითი ელემენტების ჯამს.
- 4.2.6. იპოვის მასივის უარყოფითი ელემენტების რაოდენობას.
- 4.2.7. იპოვის მასივის კენტი ელემენტების ჯამს.
- 4.2.8. იპოვის მასივის ლუწი ელემენტების რაოდენობას.
- 4.2.9. იპოვის მასივის არანულოვანი ელემენტების ნამრავლს.
- 4.2.10. იპოვის მასივის ნულოვანი ელემენტების რაოდენობას.
- 4.2.11. იპოვის მასივის მთავარი დიაგონალის ელემენტების ჯამს.
- 4.2.12. იპოვის მასივის არამთავარი დიაგონალის ელემენტების ნამრავლს.
- 4.2.13. იპოვის მასივის მთავარი დიაგონალის მინიმალურ ელემენტს.
- 4.2.14. იპოვის მასივის არამთავარი დიაგონალის მაქსიმალური ელემენტის ინდექსებს.
- 4.2.15. იპოვის მასივის მთავარი დიაგონალის 6-ის ჯერადი ელემენტების რაოდენობას.
- 4.2.16. იპოვის მასივის არამთავარი დიაგონალის დადებითი ელემენტების ჯამს.
- 4.2.17. იპოვის მასივის 4-ის ჯერადი ელემენტების რაოდენობას.
- 4.2.18. დათვლის მასივის იმ ელემენტების რაოდენობას, რომელთა მნიშვნელობები 20-ზე მეტია.
- 4.2.19. ერთი მასივიდან მეორეში მასივში გადაწერს იმ ელემენტებს, რომელთა მნიშვნელობები 20-ზე ნაკლებია.
- 4.2.20. მასივის ლუწი ელემენტებს გადაწერს ერთგანზომილებიან მასივში.
- 4.2.21. მასივის არანულოვან ელემენტებს გადაწერს ერთგანზომილებიან მასივში.
- 4.2.22. მასივის თითოეული სვეტის ელემენტების ჯამს ჩაწერს ერთგანზომილებიან მასივში.
- 4.2.23. მასივის თითოეული სტრიქონის ელემენტების ნამრავლს ჩაწერს ერთგანზომილებიან მასივში.
- 4.2.24. დათვლის მასივის თითოეული სვეტის დადებითი ელემენტების რაოდენობას და ჩაწერს მათ ერთგანზომილებიან მასივში.
- 4.2.25. იპოვის მასივის თითოეული სტრიქონის კენტი ელემენტების ჯამს და ჩაწერს მათ ერთგანზომილებიან მასივში.
- 4.2.26. იპოვის მასივის თითოეული სვეტის მაქსიმალურ ელემენტს და ჩაწერს მათ ერთგანზომილებიან მასივში.
- 4.2.27. იპოვის მასივის თითოეული სტრიქონის მინიმალურ ელემენტს და ჩაწერს მათ ერთგანზომილებიან მასივში.

- 4.2.28.** იპოვის მასივის მთავარი დიაგონალის მაქსიმალურ ელემენტს და ამ ელემენტის შემცველ სტრიქონს გადაწერს ერთგანზომილებიან მასივში.
- 4.2.29.** იპოვის მასივის არამთავარი დიაგონალის მინიმალურ ელემენტს და ამ ელემენტის შემცველ სვეტს გადაწერს ერთგანზომილებიან მასივში.
- 4.2.30.** იპოვის მასივის იმ სტრიქონს, რომელიც შეიცავს ლუწი ელემენტების მაქსიმალურ რაოდენობას და ამ სტრიქონს გადაწერს ერთგანზომილებიან მასივში.
- 4.2.31.** იპოვის მასივის იმ სვეტს, რომელიც შეიცავს კენტი ელემენტების მინიმალურ რაოდენობას და ამ სვეტს გადაწერს ერთგანზომილებიან მასივში.
- 4.2.32.** იპოვის მასივის მაქსიმალური ელემენტის შემცველ სტრიქონს და გადაწერს მას ერთგანზომილებიან მასივში.
- 4.2.33.** იპოვის მასივის მინიმალური ელემენტის შემცველ სვეტს და გადაწერს მას ერთგანზომილებიან მასივში.
- 4.2.34.** მასივის მთავარი დიაგონალის ელემენტებს ერთგანზომილებიან მასივში გადაწერს.
- 4.2.35.** მასივის არამთავარი დიაგონალის ელემენტებს ერთგანზომილებიან მასივში გადაწერს.
- 4.2.36.** იანგარიშებს მასივის ლუწი ნომრების მქონე სვეტების ელემენტების საშუალო არითმეტიკულს და ჩაწერს მათ ერთგანზომილებიან მასივში.
- 4.2.37.** იანგარიშებს მასივის კენტი ნომრების მქონე სტრიქონების ელემენტების საშუალო გეომეტრიულს და ჩაწერს მათ ერთგანზომილებიან მასივში.
- 4.2.38.** იპოვის მასივის იმ სტრიქონების ინდექსებს, რომელთა ყველა ელემენტი ნულის ტოლია. ინდექსები ჩაწერეთ ერთგანზომილებიან მასივში.
- 4.2.39.** იპოვის მასივის იმ სვეტების ინდექსებს, რომელთა ყველა ელემენტი კენტია. ინდექსები ჩაწერეთ ერთგანზომილებიან მასივში.
- 4.2.40.** იპოვის მასივის იმ სტრიქონების ინდექსებს, რომელთა ელემენტები ქმნიან მონოტონურად ზრდად მიმდევრობას. ინდექსები ჩაწერეთ ერთგანზომილებიან მასივში.
- 4.2.41.** იპოვის მასივის იმ სვეტების ინდექსებს, რომელთა ელემენტები ქმნიან მონოტონურად კლებად მიმდევრობას. ინდექსები ჩაწერეთ ერთგანზომილებიან მასივში.
- 4.2.42.** ერთიანებით შეავსებს მასივის იმ სვეტსა და სტრიქონს, რომელთა გადაკვეთაზე მდებარეობს მასივის მინიმალური ელემენტი.
- 4.2.43.** ერთიანებით შეავსებს მასივის ნულების შემცველ სტრიქონს.
- 4.2.44.** ერთიანებით შეავსებს მასივის ნულების შემცველ სვეტს.
- 4.2.45.** მასივის ნულოვან ელემენტებს შეცვლის 1-ებით.
- 4.2.46.** მასივში ადგილებს შეუცვლის მითითებული ნომრის მქონე სვეტსა და უკანასკნელ სვეტს.
- 4.2.47.** მასივში ადგილებს შეუცვლის მითითებული ნომრის მქონე სტრიქონსა და პირველ სტრიქონს.
- 4.2.48.** შეასრულებს მასივის სტრიქონების ძვრას ზევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ მეორე სტრიქონი დაიკავებს პირველის ადგილს, მესამე სტრიქონი დაიკავებს მეორის ადგილს და ა.შ. უკანასკნელი სტრიქონი შეივსება ნულებით.
- 4.2.49.** შეასრულებს მასივის სტრიქონების ძვრას ზევით ორი სტრიქონით.
- 4.2.50.** შეასრულებს მასივის სტრიქონების ძვრას ქვევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ პირველი სტრიქონი დაიკავებს მეორის ადგილს, მეორე სტრიქონი დაიკავებს მესამის ადგილს და ა.შ. პირველი სტრიქონი შეივსება ნულებით.
- 4.2.51.** შეასრულებს მასივის სტრიქონების ძვრას ქვევით ორი სტრიქონით.
- 4.2.52.** შეასრულებს მასივის სტრიქონების ციკლისებურ ძვრას ზევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ მეორე სტრიქონი დაიკავებს პირველის ადგილს, მესამე სტრიქონი დაიკავებს მეორის ადგილს და ა.შ. პირველი სტრიქონი დაიკავებს უკანასკნელის ადგილს.
- 4.2.53.** შეასრულებს მასივის სტრიქონების ციკლისებურ ძვრას ზევით ორი სტრიქონით.
- 4.2.54.** შეასრულებს მასივის სტრიქონების ციკლისებურ ძვრას ქვევით ერთი სტრიქონით. ეს იმას

ნიშნავს, რომ პირველი სტრიქონი დაიკავებს მეორის ადგილს, მეორე სტრიქონი დაიკავებს მესამის ადგილს და ა.შ. უკანასკნელი სტრიქონი დაიკავებს პირველის ადგილს.

4.2.55. შეასრულებს მასივის სტრიქონების ციკლისებურ ძვრას ქვევით ორი სტრიქონით.

4.2.56. შეასრულებს მასივის სვეტების ძვრას მარჯვნივ ერთი სვეტით.

4.2.57. შეასრულებს მასივის სვეტების ძვრას მარჯვნივ ორი სვეტით.

4.2.58. შეასრულებს მასივის სვეტების ძვრას მარცხნივ ერთი სვეტით.

4.2.59. შეასრულებს მასივის სვეტების ძვრას მარცხნივ ორი სვეტით.

4.2.60. შეასრულებს მასივის სვეტების ციკლისებურ ძვრას მარცხნივ ერთი სვეტით.

4.2.61. შეასრულებს მასივის სვეტების ციკლისებურ ძვრას მარცხნივ ორი სვეტით.

4.2.62. შეასრულებს მასივის სვეტების ციკლისებურ ძვრას მარჯვნივ ერთი სვეტით.

4.2.63. შეასრულებს მასივის სვეტების ციკლისებურ ძვრას მარჯვნივ ორი სვეტით.

4.2.64. იპოვის მასივის იმ სტრიქონის ინდექსს, რომელიც შეიცავს 8-ის ჯერადი ელემენტების მაქსიმალურ რაოდენობას.

4.2.65. იპოვის მასივის იმ სვეტის ინდექსს, რომელიც შეიცავს დადებითი ელემენტების მინიმალურ რაოდენობას.

4.2.66. იპოვის მასივის იმ ელემენტების ჯამს, რომლებიც მთავარი დიაგონალის ზემოთაა მოთავსებული.

4.2.67. იპოვის მასივის იმ ელემენტების ნამრავლს, რომლებიც მთავარი დიაგონალის ქვემოთაა მოთავსებული.

4.2.68. იპოვის მასივის ორი ტოლი ელემენტის ინდექსებს.

4.2.69. იპოვის მასივის იმ ელემენტს, რომელიც ყველა სტრიქონში შედის.

4.2.70. იპოვის მასივის იმ ელემენტს, რომელიც ყველა სვეტში შედის.

4.2.71. იპოვის მასივის არამთავარი დიაგონალის ზემოთ მოთავსებული ელემენტების ჯამს.

4.2.72. იპოვის მასივის არამთავარი დიაგონალის ქვემოთ მოთავსებული ელემენტების ჯამს.

4.2.73. იპოვის მასივის იმ სტრიქონის ინდექსს, რომლის ელემენტების ჯამი აღემატება დანარჩენი სტრიქონების ელემენტების ჯამს.

4.2.74. იპოვის მასივის იმ სვეტის ნომერს, რომლის ელემენტების ჯამი აღემატება დანარჩენი სვეტების ელემენტების ჯამს.

4.2.75. განსაზღვრავს არის თუ არა მასივში ერთნაირი სვეტები.

4.2.76. განსაზღვრავს არის თუ არა მასივში ერთნაირი სტრიქონები.

4.2.77. ერთიანებით შეავსებს მასივის იმ სვეტსა და სტრიქონს, რომელთა გადაკვეთაზე მდებარეობს მთავარი დიაგონალის მაქსიმალური ელემენტი.

4.2.78. მასივის რომელიმე სვეტსა და სტრიქონს შეავსებს ნულებით მათ გადაკვეთაზე მდებარე ელემენტის გარდა.

4.2.79. იპოვის მასივის უნაგირა წერტილს. უნაგირა წერტილი არის მასივის ის ელემენტი, რომელიც უმცირესია თავის სტრიქონში და უდიდესია თავის სვეტში.

4.2.80. იპოვის მასივის იმ ელემენტების ჯამს, რომლებიც არამთავარი დიაგონალის ქვემოთაა მოთავსებული.

4.2.81. იპოვის მასივის იმ ელემენტების ჯამს, რომლებიც არამთავარი დიაგონალის ზემოთაა მოთავსებული.

4.2.82. დაადგენს შეიცავს თუ არა მასივი უარყოფითი ლუწ რიცხვს.

4.2.83. დაადგენს შეიცავს თუ არა მასივი 9-ის ჯერად რიცხვს.

4.2.84. დაადგენს არის თუ არა მასივის რომელიმე სტრიქონი დალაგებული ზრდადობით.

4.2.85. დაადგენს არის თუ არა მასივის რომელიმე სვეტი დალაგებული კლებადობით.

4.2.86. დაადგენს არის თუ არა მასივის რომელიმე სტრიქონში მეზობლად მდებარე ორი ერთნაირი ელემენტი.

4.2.87. დაადგენს არის თუ არა მასივის რომელიმე სვეტში მეზობლად მდებარე ორი ერთნაირი

ელემენტი.

4.2.88. დაადგენს არის თუ არა მასივის რომელიმე სტრიქონში ყველა ელემენტი ერთნაირი.

4.2.89. დაადგენს არის თუ არა მასივის რომელიმე სვეტში ყველა ელემენტი ერთნაირი.

4.2.90. ჭადრაკის დაფა წარმოვადგინოთ სიმბოლური (char ტიპის) ორგანოზომილებიანი მასივის სახით 8×8 . მოცემულია ორი მთელი რიცხვი p და q ($1 \leq p \leq 8, 1 \leq q \leq 8$). p არის სვეტის ინდექსი, q კი - სტრიქონის. ისინი განსაზღვრავენ ლაზიერის ადგილმდებარეობას. მასივის შესაბამის ელემენტს მივანიჭოთ 'ლ' (ლაზიერი) მნიშვნელობა. ველები, რომლებსაც ლაზიერი ემუქრება "*" - ით აღვნიშნოთ, დანარჩენი ველები კი - '0' - ებით. Label კომპონენტში გამოვიტანოთ მიღებული მასივი. იგივე ამოცანა გადავწყვიტოთ ეტლისთვის, მხედრისთვის და კუსთვის.

თავი 5. კლასები, ინკაფსულაცია, მეთოდები

კლასები. ინკაფსულაცია

5.1.1. შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ბაკის ტევადობა და მანძილი, რომელსაც თვითმფრინავი 1 ლიტრი საწვავით გაიფრენს; ღია ცვლადებს: მგზავრების რაოდენობა და გაყიდული ბილეთების რაოდენობა.

5.1.2. შექმენით სტუდენტის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: გვარი, სახელი და ასაკი; ღია ცვლადებს: უნივერსიტეტის დასახელება და კურსი.

5.1.3. შექმენით მატარებლის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ვაგონების რაოდენობასა და მგზავრების რაოდენობას 1 ვაგონში; ღია ცვლადებს: ბილეთების ფასს და გაყიდული ბილეთების რაოდენობას.

5.1.4. შექმენით მართკუთხედის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: პერიმეტრი და ფართობი; ღია ცვლადებს: მართკუთხედის გვერდებს.

5.1.5. შექმენით ავტომანქანის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: მანქანის ფერი და კარების რაოდენობა; ღია ცვლადებს: მფლობელის გვარი და გამომშვები ფირმა.

მეთოდები

5.2.1. შექმენით სტუდენტის კლასი, რომელიც შეიცავს ღია მეთოდს. მეთოდს გადაეცემა სტუდენტის ნიშნები, რომლებიც წარმოადგენენ 10-ელემენტიან მთელრიცხვა მასივს, მეთოდი გამოთვლის და აბრუნებს ნიშნების საშუალო არითმეტიკულს.

5.2.2. შექმენით სტუდენტის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: გვარი, სახელი და ასაკი; პრივატულ მეთოდს, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; ღია მეთოდებს: პირველი მეთოდი იძახებს პრივატულ მეთოდს და გადასცემს მნიშვნელობებს პრივატული ცვლადებისათვის მისანიჭებლად; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები.

5.2.3. შექმენით მატარებლის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ვაგონების რაოდენობასა და მგზავრების რაოდენობას 1 ვაგონში; ღია ცვლადებს: ბილეთების ფასს და გაყიდული ბილეთების რაოდენობას. შეიცავს ღია მეთოდებს: პირველი მეთოდი პრივატულ და ღია ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული და ღია ცვლადების მნიშვნელობები; მესამე მეთოდი გამოთვლის და აბრუნებს ბილეთების გაყიდვით მიღებულ თანხას.

5.2.4. შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს ღია ცვლადებს: ბაკის ტევადობა და მანძილი, რომელსაც გაიფრენს თვითმფრინავი 1 ლიტრი საწვავით; ღია მეთოდებს: პირველი მეთოდი ღია ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს ღია ცვლადების მნიშვნელობები; მესამე მეთოდი გამოთვლის და აბრუნებს თვითმფრინავის მიერ სახვე ბაკით გავლილ მანძილს.

5.2.5. შექმენით მართკუთხედის კლასი, რომელიც შეიცავს ღია ცვლადებს: მართკუთხედის ოთხივე გვერდის ზომებს; პრივატულ მეთოდს, რომელიც ღია ცვლადებს მნიშვნელობებს ანიჭებს; ღია მეთოდებს: პირველი მეთოდი იძახებს პრივატულ მეთოდს, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები; მესამე მეთოდი გამოთვლის და აბრუნებს მართკუთხედის ფართობს.

კონსტრუქტორი

5.3.1. შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ბაკის ტევადობა და მანძილი, რომელსაც გაიფრენს თვითმფრინავი 1 ლიტრი საწვავით; ღია მეთოდებს: პირველი მეთოდი კონსტრუქტორი, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები; მესამე მეთოდი გამოთვლის და აბრუნებს თვითმფრინავის მიერ სავსე ბაკით გავლილ მანძილს.

5.3.2. შექმენით სამკუთხედის კლასი, რომელიც შეიცავს ღია ცვლადებს: სამკუთხედის სამივე გვერდის ზომებს; პრივატულ ცვლადებს: სამკუთხედის პერიმეტრსა და ფართობს; ღია მეთოდებს: პირველი მეთოდი კონსტრუქტორი, რომელიც ღია და პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები.

5.3.3. შექმენით მართკუთხედის კლასი, რომელიც შეიცავს ღია ცვლადებს: მართკუთხედის ოთხივე გვერდის ზომებს; პრივატულ ცვლადებს: მართკუთხედის ფართობს და პერიმეტრს; ღია მეთოდებს: პირველი მეთოდი კონსტრუქტორი, რომელიც ღია და პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები.

5.3.4. შექმენით ავტომანქანის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: მანქანის ფერი და კარებების რაოდენობა; ღია ცვლადებს: მფლობელის გვარი და გამომშვები ფირმა; ღია მეთოდებს: პირველი მეთოდი კონსტრუქტორი, რომელიც ღიას და პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები.

this საკვანძო სიტყვა

5.4.1. შექმენით კლასი, რომლის ცვლადებსა და მეთოდის პარამეტრებს ერთნაირი სახელები აქვთ. პარამეტრების მნიშვნელობები მიაჩნით კლასის ამავე სახელის მქონე ცვლადებს. მეთოდი აბრუნებს კლასის ცვლადების ჯამს.

5.4.2. შექმენით კლასი, რომლის ცვლადებსა და მეთოდის პარამეტრებს ერთნაირი სახელები აქვთ. პარამეტრების მნიშვნელობები მიაჩნით კლასის ამავე სახელის მქონე ცვლადებს. მეთოდი აბრუნებს კლასის ცვლადების ნამრავლს.

5.4.3. შექმენით კლასი, რომელშიც განსაზღვრულ მასივსა და მეთოდის პარამეტრს ერთნაირი სახელები აქვთ. პარამეტრის მნიშვნელობა მიაჩნით კლასის ამავე სახელის მქონე მასივს. მეთოდი აბრუნებს მასივის პირველ უარყოფით ელემენტს.

5.4.4. შექმენით კლასი, რომელშიც განსაზღვრულ მასივსა და მეთოდის პარამეტრს ერთნაირი სახელები აქვთ. პარამეტრის მნიშვნელობა მიაჩნით კლასის ამავე სახელის მქონე მასივს. მეთოდი აბრუნებს მასივის ელემენტებს შორის მინიმალურს.

შემთხვევითი რიცხვები

შეადგინეთ პროგრამა, რომელიც:

5.5.1. მოახდენს [1,100] ინტერვალში მოთავსებული 20 შემთხვევითი რიცხვის გენერირებას.

5.5.2. დათვლის თუ რამდენჯერ გვხვდება q რიცხვი [1,1000] ინტერვალში 50 მცდელობის შემდეგ.

5.5.3. ჯერ მოახდენს 100 მთელი რიცხვის გენერირებას [1,100] ინტერვალში, შემდეგ კი დათვლის თუ რამდენჯერ გამოჩნდა თითოეული რიცხვი ამ ინტერვალში.

5.5.4. განსაზღვრავს თუ რამდენი მთელი რიცხვის გენერირება უნდა მოვახდინოთ [1,30]

ინტერვალიდან, რომ მოცემული R რიცხვი მათ შორის შეგვხვდეს 7-ჯერ.

5.5.5. განსაზღვრავს თუ რამდენი ასანთი უნდა ამოვიღოთ ასანთის 5 ყუთიდან იმისათვის, რომ დაცარიელდეს ერთ-ერთი. თითო ყუთში 20 ასანთია.

თავი 6. პოლიმორფიზმი

მეთოდისათვის ობიექტის გადაცემა

6.1.1. შექმენით Samkutxedi კლასი. მასში შექმენით მეთოდი, რომელსაც გადაეცემა ამავე ტიპის ობიექტი და რომელიც შეასრულებს ამ ობიექტის ფართობის გამოთვლას.

6.1.2. შექმენით Otxkutxedi კლასი. მასში შექმენით მეთოდი, რომელსაც გადაეცემა ამავე ტიპის ობიექტი და რომელიც შეასრულებს ამ ობიექტის პერიმეტრის გამოთვლას.

6.1.3. შექმენით Kvadrati კლასი. მასში შექმენით მეთოდი, რომელსაც გადაეცემა ამავე ტიპის ობიექტი და რომელიც შეასრულებს ამ ობიექტის პერიმეტრის გამოთვლას.

არგუმენტების გადაცემის ხერხები. ref, out და params მოდიფიკატორები

6.2.1. ref მოდიფიკატორის საშუალებით მეთოდს გადაეცით ორი მთელი ტიპის ცვლადი. მეთოდი მეორე ცვლადს პირველი ცვლადის კვადრატს ანიჭებს.

6.2.2. კლასში გამოაცხადეთ ერთი მეთოდი, რომელსაც სამი პარამეტრი აქვს. პირველი და მეორე პარამეტრებია მთელრიცხვა მასივები, მესამე პარამეტრი მთელი ტიპისაა და აქვს ref მოდიფიკატორი. მეთოდი პირველი მასივის დადებით ელემენტებს მეორე მასივში გადაწერს. ref მოდიფიკატორიან პარამეტრს უნდა მიენიჭოს გადაწერილი ელემენტების რაოდენობა.

6.2.3. out მოდიფიკატორის საშუალებით მეთოდიდან დააბრუნეთ სამკუთხედის პერიმეტრი, return ოპერატორით კი - ფართობი.

6.2.4. out მოდიფიკატორის საშუალებით მეთოდიდან დააბრუნეთ ერთგანზომილებიანი მასივის დადებითი და უარყოფითი ელემენტების რაოდენობა, აგრეთვე, ლუწი ელემენტების ჯამი, return ოპერატორით კი - კენტი ელემენტების ჯამი.

6.2.5. მეთოდს params მოდიფიკატორის საშუალებით გადაეცით სხვადასხვა რაოდენობის მთელი ცვლადი. მეთოდი აბრუნებს მათ შორის დადებითი რიცხვების რაოდენობას.

6.2.6. მეთოდს params მოდიფიკატორის საშუალებით გადაეცით სხვადასხვა რაოდენობის მთელი ცვლადი. მეთოდი აბრუნებს მათ შორის ლუწი რიცხვების რაოდენობას.

6.2.7. მეთოდს params მოდიფიკატორის საშუალებით გადაეცით სხვადასხვა რაოდენობის მთელი ცვლადი. მეთოდი აბრუნებს მათ შორის მაქსიმალურს.

ობიექტის დაბრუნება

6.3.1. შექმენით Kvadrati_1 კლასი. მასში გამოაცხადეთ ფართობი და პერიმეტრი, და კონსტრუქტორი რომელიც ამ ცვლადებს მნიშვნელობებს ანიჭებს. შექმენით Kvadrati_2 კლასი. მასში გამოაცხადეთ მეთოდი, რომელიც ანგარიშობს კვადრატის ფართობს და პერიმეტრს, და აბრუნებს ამ პარამეტრების მქონე Kvadrati_1 ტიპის ობიექტს.

6.3.2. შექმენით ChemiKlasi_1 კლასი. მასში გამოაცხადეთ ორი ცვლადი მასივის მინიმალური და მაქსიმალური ელემენტების მოსათავსებლად და კონსტრუქტორი, რომელიც ამ ცვლადების ინიციალიზებას ასრულებს. შექმენით ChemiKlasi_2 კლასი. მასში გამოაცხადეთ მეთოდი, რომლის პარამეტრია ერთგანზომილებიანი მასივი და რომელიც პოულობს ამ მასივის ელემენტებს შორის მინიმალურს და მაქსიმალურს და აბრუნებს ამ პარამეტრების მქონე ChemiKlasi_1 ტიპის ობიექტს.

მეთოდების გადატვირთვა. პოლიმორფიზმი

6.4.1. შეადგინეთ სამკუთხედის კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 2 მეთოდი. პირველ მეთოდს 2 მთელირიცხვა პარამეტრი აქვს: სამკუთხედის სიმაღლე და ფუძე და აბრუნებს მართკუთხა სამკუთხედის ფართობს. მეორე მეთოდს 3 მთელირიცხვა პარამეტრი აქვს: სამკუთხედის გვერდები და აბრუნებს სამკუთხედის პერიმეტრს.

6.4.2. შეადგინეთ ფიგურის კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 3 მეთოდი. პირველ მეთოდს ერთი მთელირიცხვა პარამეტრი აქვს და აბრუნებს კვადრატის პერიმეტრს. მეორე მეთოდს ორი მთელირიცხვა პარამეტრი აქვს და აბრუნებს ოთხკუთხედის პერიმეტრს. მესამე მეთოდს სამი მთელირიცხვა პარამეტრი აქვს და აბრუნებს სამკუთხედის პერიმეტრს.

6.4.3. შექმენით ავტომობილის კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 2 მეთოდი. პირველ მეთოდს 2 მთელირიცხვა პარამეტრი აქვს: ბაკის ტევადობა და მანძილი, რომელსაც გაივლის ავტომობილი 1 ლიტრი საწვავით, და აბრუნებს ავტომობილის მიერ სავსე ბაკით გავლილ მანძილს. მეორე მეთოდს 2 წილადი პარამეტრი აქვს: მაქსიმალური სიჩქარე და მოძრაობის დრო, და აბრუნებს მანძილს, რომელსაც გაივლის ავტომობილი მითითებული დროის განმავლობაში მაქსიმალური სიჩქარით მოძრაობისას.

6.4.4. შეადგინეთ მატარებლის კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 2 მეთოდი. პირველ მეთოდს 2 მთელირიცხვა პარამეტრი აქვს: ვაგონების რაოდენობა და ერთ ვაგონში მგზავრების რაოდენობა, და აბრუნებს მგზავრების საერთო რაოდენობას. მეორე მეთოდს 2 წილადი პარამეტრი აქვს: 1 კილომეტრის გავლისას დახარჯული ელექტროენერგია და გავლილი მანძილი, და აბრუნებს მატარებლის მიერ მითითებული მანძილის გავლისას დახარჯულ ელექტროენერგიას.

კონსტრუქტორების გადატვირთვა

6.5.1. შექმენით კლასი, რომელიც მთელი ტიპის Min ცვლადს შეიცავს. კლასის I კონსტრუქტორი, რომელსაც გადაეცემა ერთგანზომილებიანი მთელირიცხვა მასივი, Min ცვლადს ანიჭებს მასივის მინიმალურ ელემენტს. კლასის II კონსტრუქტორს პარამეტრად გადაეცემა ამავე კლასის ობიექტი, რომლის საშუალებით მოხდება ამავე კლასის Min ცვლადის ინიციალიზება.

6.5.2. შექმენით კლასი, რომელიც სამ გვერდს, პერიმეტრს, ფართობსა და სამ კონსტრუქტორს შეიცავს. I კონსტრუქტორს 1 მთელი ტიპის პარამეტრი აქვს და ქმნის კვადრატს (გამოთვლის პერიმეტრსა და ფართობს). II კონსტრუქტორს 2 მთელი ტიპის პარამეტრი აქვს და ქმნის მართკუთხედს. III კონსტრუქტორს 3 მთელი ტიპის პარამეტრი აქვს და ქმნის სამკუთხედს.

გადატვირთვადი კონსტრუქტორის გამოძახება this სიტყვის გამოყენებით

6.6.1. შექმენით კვადრატის კლასი, რომელიც სამ კონსტრუქტორს შეიცავს. I კონსტრუქტორს 1 მთელი ტიპის პარამეტრი აქვს, რომელსაც ამავე კლასის ცვლადს ანიჭებს. II კონსტრუქტორს პარამეტრები არ აქვს, იძახებს I კონსტრუქტორს და მას 0-ს გადასცემს. III კონსტრუქტორი, რომლის პარამეტრია ამავე კლასის ობიექტი, იძახებს I კონსტრუქტორს და მას გადასცემს ობიექტის ცვლადის მნიშვნელობას. II და III კონსტრუქტორებში სრულდება კვადრატის პერიმეტრის გამოთვლა.

6.6.2. შექმენით მართკუთხედის კლასი, რომელიც სამ კონსტრუქტორს შეიცავს. I კონსტრუქტორს 2 მთელი ტიპის პარამეტრი აქვს, რომლებსაც ამავე კლასის ცვლადებს ანიჭებს. II კონსტრუქტორს პარამეტრები არ აქვს, იძახებს I კონსტრუქტორს და მას 0-ებს გადასცემს. III კონსტრუქტორი, რომლის პარამეტრია ამავე კლასის ობიექტი, იძახებს I კონსტრუქტორს და მას გადასცემს ობიექტის ორივე ცვლადის მნიშვნელობას. II და III კონსტრუქტორებში სრულდება მართკუთხედის ფართობის გამოთვლა.

static მოდიფიკატორი

6.7.1. შექმენით Otxkutxedi კლასი, რომელიც შეიცავს: სტატიკურ ცვლადებს - ოთხკუთხედის გვერდებს, რომელთა ინიციალიზებას კონსტრუქტორი ასრულებს; ჩვეულებრივ მეთოდს, რომელიც ოთხკუთხედის პერიმეტრს ანგარიშობს; სტატიკურ მეთოდს, რომელიც ჩვეულებრივ მეთოდს იძახებს.

6.7.2. შექმენით ChemiKlasi კლასი, რომელიც შეიცავს: ერთგანზომილებიან სტატიკურ მასივს, რომლის ინიციალიზებას კონსტრუქტორი ახდენს; ჩვეულებრივ მეთოდს, რომელიც ერთგანზომილებიანი სტატიკური მასივის კენტი ელემენტების ჯამს გასცემს; სტატიკურ მეთოდს, რომელიც ჩვეულებრივ მეთოდს იძახებს და გასცემს შედეგს.

6.7.3. შექმენით ChemiKlasi კლასი, რომელიც შეიცავს: ორგანზომილებიან სტატიკურ მასივს, რომლის ინიციალიზებას კონსტრუქტორი ახდენს; ჩვეულებრივ მეთოდს, რომელიც ორგანზომილებიანი სტატიკური მასივის კენტი ელემენტების რაოდენობას გასცემს; სტატიკურ მეთოდს, რომელიც ჩვეულებრივ მეთოდს იძახებს და გასცემს შედეგს.

თავი 7. მემკვიდრეობითობა

კლასები. მემკვიდრეობითობა. protected მოდიფიკატორი

7.1.1. შექმენით სამკუთხედის საბაზო კლასი, რომელიც შეიცავს სამ დაცულ ცვლადს - სამკუთხედის გვერდებს. შექმენით სამკუთხედის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს პრივატულ ცვლადს - სამკუთხედის პერიმეტრს და ორ ღია მეთოდს. პირველი მეთოდი გამოთვლის და აბრუნებს სამკუთხედის ფართობს. მეორე მეთოდი გამოთვლის და გასცემს სამკუთხედის პერიმეტრს.

7.1.2. შექმენით მართკუთხედის საბაზო კლასი, რომელიც შეიცავს დაცულ ცვლადს: მართკუთხედის ფუძეს. შექმენით მართკუთხედის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს: პრივატულ ცვლადს - მართკუთხედის სიმაღლეს; ღია მეთოდს, რომელიც გამოთვლის და აბრუნებს მართკუთხედის ფართობს.

7.1.3. შექმენით მართკუთხედის საბაზო კლასი, რომელიც შეიცავს დაცულ ცვლადს: მართკუთხედის ფუძეს; ღია მეთოდებს: პირველი მეთოდი არის კონსტრუქტორი, რომელიც პრივატულ ცვლადს მნიშვნელობას ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადის მნიშვნელობა. შექმენით მართკუთხედის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს: პრივატულ ცვლადს - მართკუთხედის სიმაღლეს; ღია მეთოდს, რომელიც გამოთვლის და აბრუნებს მართკუთხედის ფართობს.

7.1.4. შექმენით პიროვნების საბაზო კლასი, რომელიც შეიცავს დაცულ ცვლადებს: გვარს, სახელსა და ასაკს; ღია მეთოდებს: პირველი მეთოდი არის კონსტრუქტორი, რომელიც დაცულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები. შექმენით ექიმის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს: ღია ცვლადებს: ექიმის ასაკს, განყოფილების დასახელებას, თანამდებობას, საავადმყოფოს დასახელებას, სტაჟს.

კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა

7.2.1. შექმენით გეომეტრიული ფიგურის Figura საბაზო კლასი, რომელშიც ორი კონსტრუქტორია. შექმენით მისი მემკვიდრე Samkutxedi და Kvadrati კლასები. Samkutxedi კლასის კონსტრუქტორი იძახებს წინაპარი კლასის კონსტრუქტორს გვერდების ზომების განსაზღვრისათვის. თვით ამ კონსტრუქტორში განისაზღვრება სამკუთხედის პერიმეტრი. Kvadrati კლასის კონსტრუქტორი იძახებს წინაპარი კლასის კონსტრუქტორს გვერდების ზომების განსაზღვრისათვის. თვით ამ კონსტრუქტორში ხდება კვადრატის ფართობის გამოთვლა.

7.2.2. შექმენით მატარებლის საბაზო კლასი - Matarebeli. შექმენით Matarebeli კლასის მემკვიდრე კლასი - Memkvidre_1, რომლის კონსტრუქტორი იძახებს წინაპარი კლასის კონსტრუქტორს შემდეგი პარამეტრების განსაზღვრის მიზნით: მატარებელი 1 სთ-ში რამდენ კილომეტრს გადის და რამდენ საათს მოძრაობს. თვით ამ კონსტრუქტორში განისაზღვრება მატარებლის მიერ გავლილი მანძილი. შექმენით Matarebeli_1 კლასის მემკვიდრე კლასი - Memkvidre_2, რომლის რომლის კონსტრუქტორი იძახებს წინაპარი კლასის კონსტრუქტორს შემდეგი პარამეტრების განსაზღვრის მიზნით: მატარებელი 1 კმ-ის გავლისას რამდენ ელექტროენერგიას ხარჯავს და რამდენ კილომეტრს გადის. თვით ამ კონსტრუქტორში განისაზღვრება მატარებლის მიერ დახარჯული ელექტროენერგია.

7.2.3. შექმენით ტელევიზორის საბაზო კლასი - Televizori. შექმენით Televizori კლასის მემკვიდრე კლასი - Memkvidre_1, რომლის კონსტრუქტორი იძახებს წინაპარი კლასის კონსტრუქტორს შემდეგი პარამეტრების განსაზღვრის მიზნით: ტელევიზორი 1 სთ-ში რამდენ ვატს მოიხმარს და რამდენი საათის განმავლობაშია ჩართული. თვით ამ კონსტრუქტორში განისაზღვრება ტელევიზორის მიერ ნამუშევარი საათების რაოდენობა. შექმენით Memkvidre_1 კლასის მემკვიდრე კლასი - Memkvidre_2, რომლის რომლის კონსტრუქტორი იძახებს წინაპარი კლასის კონსტრუქტორს შემდეგი პარამეტრების განსაზღვრის მიზნით: ტელევიზორი 1 სთ-ის მუშაობის შედეგად რამდენ ვატს დახარჯავს და რამდენ საათს იმუშავებს. თვით ამ კონსტრუქტორში განისაზღვრება ტელევიზორის მიერ დახარჯული ვატების რაოდენობა.

წევრების დამალვა მემკვიდრეობითობის დროს

7.3.1. შექმენით საბაზო კლასი, რომელშიც სამკუთხედის სამივე გვერდია განსაზღვრული. შექმენით მისი მემკვიდრე კლასი, რომელშიც სამკუთხედის სამივე გვერდია განსაზღვრული, რომელთა სახელები ემთხვევა წინაპარი კლასის ცვლადების სახელებს. მემკვიდრე კლასი ორ მეთოდს შეიცავს: I მეთოდი გამოთვლის წინაპარი კლასის სამკუთხედის პერიმეტრს, II მეთოდი კი - მემკვიდრე კლასის სამკუთხედის პერიმეტრს.

7.3.2. შექმენით საბაზო კლასი, რომელშიც ერთგანზომილებიანი მასივია განსაზღვრული. შექმენით მისი მემკვიდრე კლასი, რომელშიც ერთგანზომილებიანი მასივია განსაზღვრული, რომლის სახელი ემთხვევა წინაპარი კლასის მასივის სახელს. მემკვიდრე კლასი ორ მეთოდს შეიცავს: I მეთოდი გასცემს წინაპარი კლასის მასივის ელემენტებს შორის მინიმალურს, II მეთოდი კი - მემკვიდრე კლასის მასივის ელემენტებს შორის მაქსიმალურს.

7.3.3. შექმენით საბაზო კლასი, რომელშიც ორგანზომილებიანი მასივია განსაზღვრული. შექმენით მისი მემკვიდრე კლასი, რომელშიც ორგანზომილებიანი მასივია განსაზღვრული, რომლის სახელი ემთხვევა წინაპარი კლასის მასივის სახელს. მემკვიდრე კლასი ორ მეთოდს შეიცავს: I მეთოდი გასცემს წინაპარი კლასის მასივის დადებით ელემენტებს შორის მინიმალურს, II მეთოდი კი - მემკვიდრე კლასის მასივის დადებით ელემენტებს შორის მაქსიმალურს.

ვირტუალური მეთოდები

7.4.1. შექმენით გეომეტრიული ფიგურის საბაზო კლასი. ის შეიცავს Perimetri მეთოდს, რომელიც გეომეტრიული ფიგურის პერიმეტრს ანგარიშობს. შექმენით საბაზო კლასის მემკვიდრე კლასი. მასში შექმენით Perimetri მეთოდის გადატვირთული ვერსია, რომელიც ოთხკუთხედის პერიმეტრს გამოთვლის.

7.4.2. შექმენით საბაზო კლასი. ის შეიცავს Metodi1 მეთოდს, რომელიც გასცემს ერთგანზომილებიანი მასივის ელემენტების ჯამს. შექმენით მემკვიდრე კლასი. მასში შექმენით Metodi1 მეთოდის გადატვირთული ვერსია, რომელიც გამოთვლის და გასცემს ერთგანზომილებიანი მასივის ელემენტების ნამრავლს.

7.4.3. შექმენით საბაზო კლასი. ის შეიცავს Metodi1 მეთოდს, რომელიც გასცემს სტრიქონში სასვენი ნიშნების რაოდენობას. შექმენით მემკვიდრე კლასი. მასში შექმენით Metodi1 მეთოდის

გადატვირთული ვერსია, რომელიც გამოთვლის და გასცემს სტრიქონში ხმოვნების რაოდენობას.

7.4.4. შექმენით საბაზო კლასი. ის შეიცავს Metodi1 მეთოდს, რომელიც გასცემს ორგანზომილებიანი მასივის უარყოფით ელემენტებს შორის მინიმალურს. შექმენით მემკვიდრე კლასი. მასში შექმენით Metodi1 მეთოდის გადატვირთული ვერსია, რომელიც გამოთვლის და გასცემს ორგანზომილებიანი მასივის უარყოფით ელემენტებს შორის მაქსიმალურს.

აბსტრაქტული კლასები და მეთოდები

7.5.1. შექმენით გეომეტრიული ფიგურის აბსტრაქტული საბაზო კლასი, რომელიც შეიცავს Perimetri აბსტრაქტულ მეთოდს. შექმენით საბაზო კლასის პირველი მემკვიდრე კლასი. მასში მოხდინეთ Perimetri აბსტრაქტული მეთოდის რეალიზება, რომელიც ოთხკუთხედის პერიმეტრს გამოთვლის. შექმენით საბაზო კლასის მეორე მემკვიდრე კლასი. მასში მოხდინეთ Perimetri აბსტრაქტული მეთოდის რეალიზება, რომელიც სამკუთხედის პერიმეტრს გამოთვლის.

7.5.2. შექმენით მატარებლის აბსტრაქტული საბაზო კლასი, რომელიც შეიცავს Gamotvla აბსტრაქტულ მეთოდს. შექმენით საბაზო კლასის პირველი მემკვიდრე კლასი. მასში მოხდინეთ Gamotvla აბსტრაქტული მეთოდის რეალიზება, რომელიც გამოთვლის მატარებლის მიერ გავლილ მანძილს (ვიცით 1 სთ-ში რამდენ კილომეტრს გადის და რამდენ საათს მოძრაობდა). შექმენით საბაზო კლასის მეორე მემკვიდრე კლასი. მასში მოხდინეთ Gamotvla აბსტრაქტული მეთოდის რეალიზება, რომელიც გამოთვლის მატარებლის მიერ დახარჯულ ელექტროენერგიას (ვიცით 1 კმ-ის გავლისას რამდენ ენერგიას ხარჯავს და რამდენი კილომეტრი გაიარა).

7.5.3. შექმენით ტელევიზორის აბსტრაქტული საბაზო კლასი, რომელიც შეიცავს Metodi აბსტრაქტულ მეთოდს. შექმენით საბაზო კლასის მემკვიდრე კლასი. მასში მოხდინეთ Metodi აბსტრაქტული მეთოდის რეალიზება, რომელიც გამოთვლის ტელევიზორის მიერ დახარჯულ ელექტროენერგიას (ვიცით 1 სთ-ის მუშაობისას რამდენ ენერგიას ხარჯავს და რამდენი საათი იმუშავა).

7.5.4. შექმენით თანამშრომლის აბსტრაქტული საბაზო კლასი, რომელიც შეიცავს Metodi აბსტრაქტულ მეთოდს. შექმენით საბაზო კლასის მემკვიდრე კლასი. მასში მოხდინეთ Metodi აბსტრაქტულ მეთოდის რეალიზება, რომელიც გამოთვლის თანამშრომლის მიერ 1 წლის განმავლობაში აღებული ხელფასის ჯამს (ვიცით თანამშრომლის ყოველთვიური ხელფასი).

თავი 8. ოპერატორების გადატვირთვა, თვისებები და ინდექსატორები

თვისებები

8.1.1. შექმენით კლასი, რომელშიც პრივატული მთელი ტიპის ცვლადია გამოცხადებული. მასთან მიმართვისათვის გამოიყენეთ თვისება, რომელშიც set და get მეთოდებია განსაზღვრული. თვისება პრივატულ ცვლადს დადებით მნიშვნელობებს ანიჭებს.

8.1.2. შექმენით კლასი, რომელშიც პრივატული მთელი ტიპის ცვლადია გამოცხადებული. მასთან მიმართვისათვის გამოიყენეთ თვისება, რომელშიც set და get მეთოდებია განსაზღვრული. თვისება პრივატულ ცვლადს კენტ მნიშვნელობებს ანიჭებს.

8.1.3. შექმენით კლასი, რომელშიც პრივატული მთელი ტიპის ცვლადია გამოცხადებული. მასთან მიმართვისათვის გამოიყენეთ თვისება, რომელშიც set და get მეთოდებია განსაზღვრული. თვისება პრივატულ ცვლადს 5-ის ჯერად მნიშვნელობებს ანიჭებს.

ინდექსატორები. ერთგანზომილებიანი ინდექსატორები

8.2.1. შექმენით კლასი, რომელშიც გამოცხადებულია 5 მთელი ტიპის ცვლადი. მათ მიანიჭეთ მნიშვნელობები ინდექსატორის გამოყენებით.

8.2.2. შექმენით კლასი, რომელშიც გამოცხადებულია 5 წილადი ტიპის ცვლადი. მათ მიანიჭეთ

მნიშვნელობები ინდექსატორის გამოყენებით.

8.2.3. შექმენით კლასი, რომელშიც გამოცხადებულია 5 ლოგიკური ცვლადი. მათ მიანიჭეთ მნიშვნელობები ინდექსატორის გამოყენებით.

8.2.4. შექმენით კლასი, რომელშიც გამოცხადებულია 5 სტრიქონი. მათ მიანიჭეთ მნიშვნელობები ინდექსატორის გამოყენებით.

8.2.5. შექმენით კლასი, რომელშიც გამოცხადებულია მთელი რიცხვა ერთგანზომილებიანი მასივი. ინდექსატორის გამოყენებით იპოვეთ მასივის ელემენტების ჯამი.

8.2.6. შექმენით კლასი, რომელშიც გამოცხადებულია წილადების ორგანზომილებიანი მასივი. ინდექსატორის გამოყენებით იპოვეთ მასივის მინიმალური ელემენტი.

8.2.7. შექმენით კლასი, რომელშიც გამოცხადებულია 5 მთელი ტიპის ცვლადი. ცვლადებს მიანიჭეთ ლუწი მნიშვნელობები ინდექსატორის გამოყენებით.

თავი 9. დელეგატები, მოვლენები, ინტერფეისები და სახელების სივრცე

დელეგატები. დელეგატების მრავალმისამართიანობა

9.1.1. დელეგატს დაუმატეთ 2 მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით I მეთოდის მისამართი და დაუმატეთ III მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით.

9.1.2. დელეგატს დაუმატეთ 2 მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით II მეთოდის მისამართი და დაუმატეთ III მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით.

9.1.3. დელეგატს დაუმატეთ 3 მეთოდის მისამართი. სამივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით I და III მეთოდების მისამართები და დაუმატეთ IV მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით.

9.1.4. დელეგატს დაუმატეთ 3 მეთოდის მისამართი. სამივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით II და III მეთოდების მისამართები და დაუმატეთ IV მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით.

მოვლენები

9.2.1. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ, როცა პირველი რიცხვი მეორეზე მეტია. დამამუშავებელს გამოაქვს შესაბამისი შეტყობინება.

9.2.2. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ, როცა ორივე რიცხვი ლუწია. დამამუშავებელს გამოაქვს შესაბამისი შეტყობინება.

9.2.3. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ როცა ორი სტრიქონი ერთნაირი არ არის.

9.2.4. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ, როცა მასივის ყველა ელემენტი 50-ზე ნაკლებია.

9.2.5. შექმენით მოვლენა, რომელიც აღიძვრება ნულზე გაყოფის შემთხვევაში.

9.2.6. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ როცა 5-ელემენტიან მთელი რიცხვა მასივში მხოლოდ ნულებია.

9.2.7. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ როცა 25-სიმბოლოიან სტრიქონში მხოლოდ '*'-ებია.

9.2.8. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ, როცა შეტანილია ლუწი რიცხვი. დამამუშავებელი გასცემს ლუწი რიცხვის კვადრატს.

ინტერფეისები. რეალიზება

9.3.1. შექმენით ინტერფეისი, რომელიც ორ მეთოდს შეიცავს. I მეთოდი გამოთვლის და აბრუნებს პარამეტრის კვადრატს, II კი - პარამეტრის კუბს. მოახდინეთ ამ ინტერფეისის რეალიზება.

9.3.2. შექმენით ინტერფეისი, რომელიც ერთ თვისებას შეიცავს. თვისება პრივატულ ცვლადს ლუწ მნიშვნელობებს ანიჭებს. მოახდინეთ ამ ინტერფეისის რეალიზება.

9.3.3. შექმენით ინტერფეისი, რომელიც ერთ ინდექსატორს შეიცავს. ინდექსატორი 5 მთელ რიცხვთან მუშაობს, ანიჭებს მათ კენტ მნიშვნელობებს და გასცემს მათ მნიშვნელობებს. მოახდინეთ ამ ინტერფეისის რეალიზება.

9.3.4. შექმენით ინტერფეისი, რომელიც ერთ მოვლენას შეიცავს. მოვლენა აღიძვრება მაშინ, როცა ერთგანზომილებიანი მასივის მაქსიმალური ელემენტი 25-ს გადააჭარბებს. მოახდინეთ ამ ინტერფეისის რეალიზება.

9.3.5. შექმენით ინტერფეისი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი გამოთვლის და აბრუნებს პარამეტრის კუბს; შეიცავს ერთ თვისებას, რომელიც პრივატულ ცვლადს დადებით მნიშვნელობებს ანიჭებს; შეიცავს ერთ ინდექსატორს, რომელიც 4 მთელ რიცხვთან მუშაობს და ანიჭებს მათ 7-ის ჯერად მნიშვნელობებს; შეიცავს ერთ მოვლენას, რომელიც აღიძვრება მაშინ, როცა ერთგანზომილებიანი მასივის მინიმალური ელემენტი იქნება 10-ზე ნაკლები.

ინტერფეისები. მემკვიდრეობითობა

9.4.1. შექმენით საბაზო კლასი. შექმენით ინტერფეისი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი გამოთვლის სამკუთხედის პერიმეტრს. მოახდინეთ ამ ინტერფეისის რეალიზება იმ კლასში, რომელიც საბაზო კლასის მემკვიდრეა. საბაზო კლასი შეიცავს მეთოდს, რომელიც ახდენს სამკუთხედის ფართობის გამოთვლას.

9.4.2. შექმენით საბაზო კლასი. შექმენით ინტერფეისი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი პოულობს მასივის მინიმალურ ელემენტს. მოახდინეთ ამ ინტერფეისის რეალიზება იმ კლასში, რომელიც საბაზო კლასის მემკვიდრეა. საბაზო კლასი შეიცავს მეთოდს, რომელიც მეთოდი პოულობს მასივის მაქსიმალურ ელემენტს.

ინტერფეისები. ცხადი რეალიზება

9.5.1. შექმენით ორი ინტერფეისი, რომლებსაც ერთნაირი სახელის მქონე მეთოდი აქვთ. პირველი ინტერფეისის მეთოდი გამოთვლის და აბრუნებს სამკუთხედის პერიმეტრს. მეორე ინტერფეისის მეთოდი გამოთვლის და აბრუნებს სამკუთხედის ფართობს. შეასრულეთ ამ ინტერფეისების რეალიზება.

9.5.2. შექმენით ორი ინტერფეისი, რომლებსაც ერთნაირი სახელის მქონე თვისება აქვთ. პირველი ინტერფეისის თვისება პრივატულ ცვლადს დადებით მნიშვნელობებს ანიჭებს და აბრუნებს პრივატული ცვლადის მნიშვნელობას. მეორე ინტერფეისის თვისება პრივატულ ცვლადს უარყოფით მნიშვნელობებს ანიჭებს და აბრუნებს პრივატული ცვლადის მნიშვნელობას. შეასრულეთ ამ ინტერფეისების რეალიზება.

სახელების სივრცე. using დირექტივა. ჩადგმული სახელების სივრცე

9.6.1. შექმენით სახელების 2 სივრცე - Sivrc_1 და Sivrc_2 რომლებიც შეიცავენ Samkutedi კლასს. Sivrc_1 სივრცის Samkutedi კლასი შეიცავს Fartobi მეთოდს, რომელიც სამკუთხედის ფართობს ანგარიშობს, Sivrc_2 სივრცის Samkutedi კლასი კი შეიცავს Perimetri მეთოდს, რომელიც სამკუთხედის პერიმეტრს ანგარიშობს.

9.6.2. შექმენით სახელების 2 სივრცე - Sivrc_1 და Sivrc_2 რომლებიც შეიცავენ Otkutedi კლასს. Sivrc_1 სივრცის Otkutedi კლასი შეიცავს Fartobi მეთოდს, რომელიც ოთხკუთხედის ფართობს ანგარიშობს, Sivrc_2 სივრცის Otkutedi კლასი კი შეიცავს Perimetri მეთოდს, რომელიც ოთხკუთხედის პერიმეტრს ანგარიშობს.

9.6.3. შექმენით სახელების 2 სივრცე - `Sivrc_1` და `Sivrc_2` რომლებიც შეიცავენ `Masivi` კლასს. `Sivrc_1` სივრცის `Masivi` კლასი შეიცავს `Metodi1` მეთოდს, რომელიც გამოთვლის და დააბრუნებს ერთგანზომილებიანი მასივის დადებითი ელემენტების ჯამს, `Sivrc_2` სივრცის `Masivi` კლასი კი შეიცავს `Metodi2` მეთოდს, რომელიც გამოთვლის და დააბრუნებს ერთგანზომილებიანი მასივის უარყოფითი ელემენტების ნამრავლს.

თავი 10. ინფორმაციის შეტანა-გამოტანა

ფაილში ბაიტების შეტანა-გამოტანა. `FileStream` კლასი

შეადგინეთ პროგრამა, რომელიც:

- 10.1.1. ერთი ფაილიდან წაიკითხავს `byte` ტიპის 5 მთელ რიცხვს, მოათავსებს მათ ერთგანზომილებიან მასივში და ამ მასივის ელემენტების ჯამს დაუმატებს ამავე ფაილს.
- 10.1.2. ერთი ფაილიდან წაიკითხავს `byte` ტიპის 5 მთელ რიცხვს, მოათავსებს მათ ერთგანზომილებიან მასივში და ამ მასივის ელემენტებს და მათ ჯამს ჩაწერს მეორე ფაილში.
- 10.1.3. ერთი ფაილიდან მეორეში სათითაოდ გადაწერს `byte` ტიპის 10 მთელ რიცხვს.

ფაილში სიმბოლოების შეტანა-გამოტანა. `StreamReader` და `StreamWriter` კლასები

შეადგინეთ პროგრამა, რომელიც:

- 10.2.1. ერთი ფაილიდან წაიკითხავს 10 სიმბოლოს, მოათავსებს მათ სიმბოლოების მასივში და ამ მასივის ელემენტებს ჩაწერს მეორე ფაილში.
- 10.2.2. ერთი ფაილიდან წაიკითხავს 10 სიმბოლოს, მოათავსებს მათ სიმბოლოების მასივში და ამ მასივის ელემენტებს დაუმატებს ამავე ფაილს.
- 10.2.3. ერთი ფაილიდან წაიკითხავს სტრიქონს და ჩაწერს მას მეორე ფაილში.
- 10.2.4. ფაილში ჩაწერს რამდენიმე სტრიქონს.

ფაილში ორობითი მონაცემების შეტანა-გამოტანა. `BinaryReader` და `BinaryWriter` ნაკადები

შეადგინეთ პროგრამა, რომელიც:

- 10.3.1. ფაილიდან წაიკითხავს 10 მთელ რიცხვს და ჩაწერს მათ ფაილში.
- 10.3.2. ფაილიდან წაიკითხავს 10 წილად რიცხვს და ჩაწერს მათ ფაილში.
- 10.3.3. ფაილიდან წაიკითხავს 20 სიმბოლოიან სტრიქონს და ჩაწერს მათ ფაილში.
- 10.3.4. ფაილიდან წაიკითხავს 10 მთელ რიცხვს და დაუმატებს მათ ამავე ფაილს.
- 10.3.5. ფაილიდან წაიკითხავს 10 წილად რიცხვს და დაუმატებს მათ ამავე ფაილს.
- 10.3.6. ფაილიდან წაიკითხავს 15 მთელ რიცხვს და მოათავსებს მათ ერთგანზომილებიან მასივში.

ფაილებთან პირდაპირი მიმართვა. `Seek` მეთოდი

შეადგინეთ პროგრამა, რომელიც:

- 10.4.1. ფაილის მე-2 პოზიციიდან წაიკითხავს 10 ბაიტს და მოათავსებს მათ ერთგანზომილებიან მასივში.
- 10.4.2. ფაილის მე-5 პოზიციიდან ჩაწერს 10 ბაიტს ერთგანზომილებიანი მასივიდან.
- 10.4.3. ფაილის მე-3 პოზიციიდან ჩაწერს 9 სიმბოლოს.
- 10.4.4. ფაილის მე-6 პოზიციიდან წაიკითხავს 9 სიმბოლოს.

ფაილის შესახებ ინფორმაციის მიღება. `FileInfo` კლასი

- 10.5.1. მიიღეთ მითითებული ფაილის ატრიბუტები.
- 10.5.2. მიიღეთ მითითებული ფაილის შექმნის თარიღი.
- 10.5.3. მიიღეთ იმ კატალოგის სახელი, რომელშიც მითითებული ფაილია მოთავსებული.

- 10.5.4. შეამოწმეთ მითითებული ფაილი არსებობს თუ არა.
- 10.5.5. მიიღეთ მითითებული ფაილის გაფართოება.
- 10.5.6. მიიღეთ მითითებული ფაილის სრული სახელი.
- 10.5.7. მიიღეთ მითითებულ ფაილთან უკანასკნელი მიმართვის თარიღი.
- 10.5.8. მიიღეთ მითითებულ ფაილში უკანასკნელი ჩაწერის თარიღი.
- 10.5.9. მიიღეთ მითითებული ფაილის ზომა.

კატალოგებთან მუშაობა. DirectoryInfo კლასი

- 10.6.1. მიიღეთ მითითებული კატალოგის ატრიბუტები.
- 10.6.2. მიიღეთ მითითებული კატალოგის შექმნის თარიღი.
- 10.6.3. მიიღეთ მითითებული კატალოგის სრული სახელი.
- 10.6.4. მიიღეთ მითითებულ კატალოგთან უკანასკნელი მიმართვის დრო.
- 10.6.5. მიიღეთ მითითებულ კატალოგში უკანასკნელი ჩაწერის დრო.
- 10.6.6. შეამოწმეთ მითითებული კატალოგის არსებობა.
- 10.6.7. მიიღეთ მითითებული კატალოგის მშობელი კატალოგის სახელი.
- 10.6.8. მიიღეთ მითითებული კატალოგის ძირითადი კატალოგის სახელი.
- 10.6.9. შექმენით კატალოგი.
- 10.6.10. შექმენით ქვეკატალოგი.
- 10.6.11. წაშალეთ კატალოგი.
- 10.6.12. მიიღეთ მიმდინარე კატალოგის ქვეკატალოგების სია.
- 10.6.13. მიიღეთ მიმდინარე კატალოგის ფაილების სია.
- 10.6.14. ქვეკატალოგი გადაიტანეთ ერთი კატალოგიდან მეორეში.

სხვადასხვა

შეადგინეთ პროგრამა, რომელიც:

- 10.7.1. ფაილიდან წაიკითხავს მთელ რიცხვებს და დათვლის მათ შორის უარყოფითი, ნულოვანი და დადებითი რიცხვების რაოდენობას.
- 10.7.2. ფაილიდან წაიკითხავს წილად რიცხვებს, დათვლის მათ რაოდენობას და გამოთვლის მათ ჯამს.
- 10.7.3. წაიკითხავს ფაილში ჩაწერილი წილადი რიცხვების შუაში მოთავსებულ რიცხვს. ფაილში ჩაწერილია კენტი რაოდენობის რიცხვი.
- 10.7.4. ფაილიდან წაიკითხავს მთელ რიცხვებს, დაალაგებს მათ ზრდადობის მიხედვით და ჩაწერს ამავე ფაილში.
- 10.7.5. ფაილიდან წაიკითხავს რიცხვებს და მათ შორის პირველსა და უკანასკნელს მეორე ფაილში გადაწერს.
- 10.7.6. ფაილიდან წაიკითხავს რიცხვებს და მეორე ფაილში გადაწერს:
 - ა. მათ შორის მინიმალურსა და მაქსიმალურს;
 - ბ. კენტი ინდექსის მქონე რიცხვებს შორის უდიდესს;
 - გ. ლუწი ინდექსის მქონე რიცხვებს შორის უმცირესს.
 - დ. ლუწი და კენტი რიცხვების რაოდენობას;
- 10.7.7. სიმბოლური ფაილიდან ორ მონაცემს კითხულობს. თუ ეს მონაცემები ციფრებია, მაშინ მათი ჯამი, ნამრავლი და სხვაობა ამავე ფაილს დაუმატეთ.
- 10.7.8. ტექსტური ფაილის თითოეულ სტრიქონში დათვლის სიტყვების რაოდენობას.
- 10.7.9. ფაილიდან წაიკითხავს რიცხვებს და მეორე ფაილში გადაწერს:
 - ა. კენტ რიცხვებს;
 - ბ. ლუწ რიცხვებს;

- გ. 7-ის ჯერად რიცხვებს;
- დ. რიცხვებს, რომლებიც წარმოადგენენ კვადრატებს.
- 10.7.10.** განსაზღვრავს ორი ფაილი შეიცავს თუ არა ერთნაირ რიცხვებს.
- 10.7.11.** ერთი სიმბოლური ფაილიდან მეორეში მონაცემებს ისე გადაწერს, რომ მიმდევრობით მოთავსებული რამდენიმე ინტერვალის შეიცვალოს ერთით.
- 10.7.12.** ფაილის ყოველი სტრიქონის წინ ინტერვალის ნიშანს ჩასვამს და გადაწერს მეორე ფაილში.
- 10.7.13.** ფაილის ყოველი სტრიქონის წინ და ბოლოში მოთავსებულ ინტერვალებს წაშლის და მიღებულ სტრიქონებს ამავე ფაილში ჩაწერს.
- 10.7.14.** სიმბოლურ ფაილში:
- ა. დათვლის 'ა' სიმბოლოს რაოდენობას;
 - ბ. დათვლის 'ს', 'რ' და 'ე' სიმბოლოების რაოდენობას;
 - გ. დაადგენს შეიცავს თუ არა ფაილი "კომპიუტერი" სიმბოლოების მიმდევრობას;
 - დ. დათვლის "კომპიუტერი" სიტყვის რაოდენობას.
- 10.7.15.** სიმბოლური ფაილიდან წაშლის ყველა სამსიმბოლოიან სიტყვას და შედეგს მეორე ფაილში გადაწერს.
- 10.7.16.** სიმბოლურ ფაილში:
- ა. იპოვის ყველაზე გრძელ სიტყვას;
 - ბ. იპოვის ყველაზე მოკლე სიტყვას;
 - გ. იპოვის ყველაზე გრძელ სიტყვას, რომლის მესამე სიმბოლოა 'ფ';
 - დ. დათვლის ორი, სამი და ოთხი სიმბოლოსაგან შემდგარი სიტყვების რაოდენობას.
- 10.7.17.** ერთ სიმბოლურ ფაილს მეორეში გადაწერს.
- 10.7.18.** ერთი სიმბოლური ფაილიდან წაიკითხავს ტექსტს, პატარა ასოებს დიდ ასოებად გარდაქმნის და შედეგს მეორე ფაილში ჩაწერს.
- 10.7.19.** ერთი სიმბოლური ფაილიდან მონაცემებს უკუ მიმდევრობით გადაწერს მეორე ფაილში.
- 10.7.20.** ტექსტურ ფაილში მოძებნის 10 სიტყვას, რომლებიც ყველაზე ხშირად გვხვდება.
- 10.7.21.** ტექსტურ ფაილში დათვლის სტრიქონების რაოდენობას.
- 10.7.22.** ტექსტური ფაილის თითოეულ სტრიქონში დათვლის სიმბოლოების რაოდენობას.
- 10.7.23.** ეკრანზე გამოიტანს ტექსტური ფაილის ყველაზე გრძელ და მოკლე სტრიქონებს.
- 10.7.24.** ტექსტურ ფაილში დათვლის იმ სტრიქონების რაოდენობას, რომლებშიც სიმბოლოების რაოდენობა 60-ს აღემატება.
- 10.7.25.** ფაილში მოთავსებულ ტექსტს ეკრანზე გამოიტანს.
- 10.7.26.** ფაილში ჩაწერს 10 სტუდენტის გვარს, დაბადების თარიღსა და სიმაღლეს.
- 10.7.27.** ფაილის მეხუთე პოზიციიდან 3 ძახილის ნიშანს ჩაწერს.
- 10.7.28.** ერთი ფაილის კენტი ნომრის მქონე სიმბოლოებს მეორე ფაილის ლუწი ნომრის პოზიციებში გადაწერს.
- 10.7.29.** ერთი ფაილიდან სიმბოლოებს დაწყებული პირველი პოზიციიდან გადაწერს მეორე ფაილში დაწყებული უკანასკნელი პოზიციიდან.
- 10.7.30.** მოცემულია ორი ფაილი, რომლებიც შეიცავენ ზრდადობით დალაგებულ რიცხვებს. შეადგინეთ პროგრამა, რომელიც მესამე ფაილს შექმნის და მასში ზრდადობის მიხედვით ჩაწერს ორივე ფაილში მოთავსებულ რიცხვებს.
- 10.7.31.** მოცემულია ორი ტექსტური ფაილი. შეადგინეთ პროგრამა, რომელიც მესამე ფაილში ჯერ ჩაწერს პირველ ფაილში მოთავსებულ ტექსტს, შემდეგ კი - მეორე ფაილში მოთავსებულ ტექსტს.
- 10.7.32.** ტექსტური ფაილი შეიცავს C# ენაზე შედგენილ საწყის კოდს (პროგრამას). კოდის თითოეული ოპერატორი თითო სტრიქონს იკავებს. შეადგინეთ პროგრამა, რომელიც:
- ა. შეამოწმებს გამხსნელი და დამხურავი მრგვალი ფრჩხილების გამოყენების სისწორეს;
 - ბ. შეამოწმებს გამხსნელი და დამხურავი ფიგურული ფრჩხილების გამოყენების სისწორეს;

თავი 11. განსაკუთრებული სიტუაციები

განსაკუთრებული სიტუაციები. try, catch, throw და finally ოპერატორები

- 11.1.1. შეადგინეთ პროგრამა, რომელიც შეკრებს ერთგანზომილებიანი მასივის ელემენტებს. ინდექსის არასწორი მნიშვნელობის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.
- 11.1.2. შეადგინეთ პროგრამა, რომელიც შეკრებს ორგანზომილებიანი მასივის ელემენტებს. რომელიმე ინდექსის არასწორი მნიშვნელობის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.
- 11.1.3. შეადგინეთ პროგრამა, რომელიც ერთი ორგანზომილებიანი მასივის ელემენტებს გაყოფს მეორე ორგანზომილებიანი მასივის შესაბამის ელემენტებზე. ნულზე გაყოფის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.
- 11.1.4. შეადგინეთ პროგრამა, რომელიც მთელრიცხვა ერთგანზომილებიანი მასივის ელემენტებს ფაილში ჩაწერს. შეტანა-გამოტანის შეცდომის აღძვრის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.
- 11.1.5. შეადგინეთ პროგრამა, რომელიც მთელრიცხვა ორგანზომილებიანი მასივის ელემენტებს ფაილიდან წაიკითხავს. შეტანა-გამოტანის შეცდომის აღძვრის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.

თავი 12. სტრიქონები

სტრიქონები

ქვემოთ მოყვანილ ამოცანებში სტრიქონების შესატანად textBox კომპონენტი გამოიყენეთ.

შეადგინეთ პროგრამა, რომელიც:

- 12.1.1. სტრიქონში დათვლის სიმბოლოების რაოდენობას.
- 12.1.2. სტრიქონში დათვლის "ბ" სიმბოლოს რაოდენობას.
- 12.1.3. სტრიქონში დათვლის ციფრების რაოდენობას 0-დან 9-მდე.
- 12.1.4. სტრიქონში დათვლის სასვენი ნიშნების რაოდენობას (, ; ! ? :).
- 12.1.5. სტრიქონში დათვლის ხმოვნების რაოდენობას (ა, ი, ო, უ, ე).
- 12.1.6. სტრიქონში იპოვის "ა" სიმბოლოს პოზიციას.
- 12.1.7. სტრიქონში ყველა "ი" სიმბოლოს "ს" სიმბოლოთი შეცვლის.
- 12.1.8. სტრიქონში ყველა ციფრს "რ" სიმბოლოთი შეცვლის.
- 12.1.9. სტრიქონში "ეს არის ტესტი" ჩაუმატებს "კომპიუტერი" სიტყვას დაწყებული მე-3 პოზიციიდან.
- 12.1.10. სტრიქონიდან "ეს არის ტესტი" წაშლის "არის" სიტყვას.
- 12.1.11. სტრიქონიდან "ეს არის ტესტი" წაშლის 6 სიმბოლოს დაწყებული მე-2 პოზიციიდან.
- 12.1.12. სტრიქონში "ეს არის ტესტი" "ეს" სიტყვას შეცვლის "კომპიუტერი" სიტყვით.
- 12.1.13. სტრიქონში "ეს არის ტესტური პროგრამა" "ეს არის" სიტყვებს შეცვლის "ახლა შევასრულოთ" სიტყვებით.

დინამიკური სტრიქონები

- 12.2.1. შექმენით დინამიკური სტრიქონი. კონსტრუქტორს გადაეცით თქვენი სახელი და გვარი. დინამიური სტრიქონის ტევადობაა 50. დინამიურ სტრიქონს დაუმატეთ წილადი.
- 12.2.2. შექმენით დინამიური სტრიქონი, რომლის ტევადობაა 50, მაქსიმალური ტევადობა კი - 70. დინამიურ სტრიქონს დაუმატეთ თქვენი სახელი, გვარი და მთელი რიცხვი.

12.2.3. შექმენით დინამიური სტრიქონი. კონსტრუქტორს გადაეცით თქვენი სახელი და გვარი. დინამიურ სტრიქონს დაუმატეთ ლოგიკური მნიშვნელობა.

12.2.4. შექმენით დინამიური სტრიქონი. კონსტრუქტორს გადაეცით თქვენი სახელი და გვარი, საწყისი ინდექსი, გადასაწერი სიმბოლოების რაოდენობა და ტევადობა. საწყისი ინდექსია 3, გადასაწერი სიმბოლოების რაოდენობაა 7, ტევადობაა 50. დინამიურ სტრიქონს მე-4 პოზიციიდან ჩაუმატეთ სიტყვა "კომპიუტერი".

სხვადასხვა

შეადგინეთ პროგრამა, რომელიც:

12.3.1. სიმბოლოების $M_{5,5}$ მასივში მოძებნის იმ სტრიქონს, რომელიც შეიცავს "ა" სიმბოლოს მაქსიმალურ რაოდენობას.

12.3.2. სიმბოლოების $M_{5,5}$ მასივში მოძებნის იმ სტრიქონს, რომელიც შეიცავს განსხვავებული სიმბოლოების მაქსიმალურ რაოდენობას.

12.3.3. სიმბოლოების $M_{5,5}$ მასივში მოძებნის იმ სვეტს, რომელიც შეიცავს "ა" სიმბოლოს მინიმალურ რაოდენობას.

12.3.4. სიმბოლოების $M_{5,5}$ მასივში მოძებნის იმ სვეტს, რომელიც შეიცავს განსხვავებული სიმბოლოების მინიმალურ რაოდენობას.

12.3.5. სტრიქონის მე-10 პოზიციიდან 30-ე პოზიციამდე დათვლის "+" სიმბოლოს რაოდენობას.

12.3.6. სტრიქონში დათვლის "რო" სიმბოლოების რაოდენობას.

12.3.7. სტრიქონში დათვლის ინტერვალების რაოდენობას.

12.3.8. განსაზღვრავს შეიცავს თუ არა სტრიქონი "დ" სიმბოლოს.

12.3.9. განსაზღვრავს შეიცავს თუ არა სტრიქონი ერთნაირი სიმბოლოების წყვილებს.

12.3.10. განსაზღვრავს შეიცავს თუ არა სტრიქონი 4 მეზობელ "კ" სიმბოლოს.

12.3.11. სტრიქონში განსაზღვრავს მეზობელი ინტერვალების ყველაზე გრძელი მიმდევრობის ზომას.

12.3.12. სტრიქონიდან წაშლის "ათი" სიტყვებს.

12.3.13. სტრიქონში "ს" სიმბოლოს შემდეგ მოთავსებულ ყველა სიმბოლოს "თ" სიმბოლოთი შეცვლის.

12.3.14. ინგლისური ანბანის ყველა ასოს გამოიტანს.

12.3.15. ქართული ანბანის ყველა ასოს გამოიტანს.

12.3.16. კლავიატურის ყველა სიმბოლოს გამოიტანს.

12.3.17. სტრიქონში დათვლის სიტყვების რაოდენობას.

12.3.18. სტრიქონის უკანასკნელ სიტყვაში "ლ" სიმბოლოს რაოდენობას დათვლის.

12.3.19. სტრიქონში იმ სიტყვების რაოდენობას დათვლის, რომლებიც "ო" სიმბოლოთი იწყებიან.

12.3.20. სტრიქონში იმ სიტყვების რაოდენობას დათვლის, რომლებიც "შვილი" სიმბოლოებით მთავრდება.

12.3.21. სტრიქონში იმ სიტყვების რაოდენობას დათვლის, რომელთა პირველი და უკანასკნელი ასოები ერთმანეთს ემთხვევა.

12.3.22. სტრიქონში ყველაზე გრძელ სიტყვას იპოვის.

12.3.23. სტრიქონში ყველაზე მოკლე სიტყვას იპოვის.

12.3.24. სტრიქონიდან წაშლის ციფრებს.

12.3.25. სტრიქონში პატარა ასოებს დიდი ასოებით შეცვლის.

12.3.26. სტრიქონში დიდ ასოებს პატარა ასოებით შეცვლის.

12.3.27. სტრიქონში წაშლის წერტილის წინ მოთავსებულ რიცხვებს.

12.3.28. სტრიქონში წაშლის წერტილის შემდეგ მოთავსებულ რიცხვებს.

12.3.29. სტრიქონის სიმბოლოებს უკუ მიმდევრობით დაალაგებს.

12.3.30. ტექსტში მრგვალი ფრჩხილების ბალანსს შეამოწმებს. მრგვალი ფრჩხილები

დაბალანსებულია თუ გამხსნელ ფრჩხილს მოსდევს შესაბამისი დამხურავი ფრჩხილი და მათი რაოდენობა ტოლია.

12.3.31. შეამოწმებს არის თუ არა ერთი წინადადება მეორის ნაწილი და პირიქით.

12.3.32. შეამოწმებს ერთნაირია თუ არა ორი წინადადება.

12.3.33. ორ წინადადებაში მოძებნის იმ სიტყვებს, რომლებიც ორივე წინადადებაში შედიან და ამ სიტყვებს ჩაწერს სტრიქონების მასივში.

12.3.34. მოცემულ რიცხვს (1-დან 10-მდე) ჩაწერს სიტყვების საშუალებით.

12.3.35. სტრიქონში მოძებნის იმ სიტყვებს, რომლებიც "მე" სიმბოლოებით მთავრდება და ამ სიტყვებს ჩაწერს სტრიქონების მასივში.

12.3.36. განსაზღვრავს ტექსტში რამდენი სიტყვა შეიცავს 1 ხმოვანს, 2 ხმოვანს, 3 ხმოვანს და ა.შ.

12.3.37. განსაზღვრავს ტექსტში სიტყვების რამდენი პროცენტი იწყება "ლ" სიმბოლოთი.

12.3.38. განსაზღვრავს შეიცავს თუ არა ტექსტი ასოებისაგან განსხვავებულ სიმბოლოებს.

12.3.39. ტექსტში იპოვის იმ სიტყვას, რომელიც შეიცავს ხმოვნების მაქსიმალურ რაოდენობას.

12.3.40. ტექსტში იპოვის იმ სიტყვას, რომელიც შეიცავს თანხმოვნების მინიმალურ რაოდენობას.

12.3.41. ტექსტში იპოვის იმ სიტყვას, რომელიც შეიცავს "გ" სიმბოლოს მაქსიმალურ რაოდენობას.

12.3.42. ტექსტში იპოვის სიტყვას, რომელიც ყველაზე ხშირად გვხვდება.

12.3.43. ტექსტში იპოვის 10 სიტყვას, რომელიც ყველაზე ხშირად გვხვდება და დათვლის თითოეული მათგანის რაოდენობას.

12.3.44. ტექსტში განსაზღვრავს თუ რომელი ასოთი იწყება სიტყვების უმრავლესობა.

12.3.45. ტექსტში განსაზღვრავს თუ რამდენჯერ გვხვდება ანბანის თითოეული ასო.

12.3.46. ტექსტში განსაზღვრავს სიტყვების პირველი ასოების მინიმალურ რაოდენობას, რომელთა მიხედვით შეიძლება განვასხვავოთ სიტყვები.

12.3.47. ტექსტში იპოვის იმ სიტყვებს, რომლებიც შეიცავენ გაორმაგებულ ხმოვნებს.

12.3.48. ტექსტში იპოვის იმ სიტყვებს, რომლებიც შეიცავენ გაორმაგებულ თანხმოვნებს.

12.3.49. ტექსტიდან წაშლის ზედმეტ ინტერვალის ნიშნებს და სიტყვებს შორის დატოვებს ინტერვალის თითო ნიშანს.

12.3.50. მოცემულია ორი სიტყვა. ისინი გააერთიანეთ ისე, რომ წინ მოთავსდეს მოკლე სიტყვა, შემდეგ კი - გრძელი.

12.3.51. მოცემულია ორი სიტყვა. მეორე სიტყვა მოათავსეთ პირველი სიტყვის შუაში.

12.3.52. დაადგინეთ რამდენი პალინდრომია მოცემულ ტექსტში. პალინდრომია, მაგალითად, "აბგგბა", 123321 და ა.შ.

12.3.53. სიტყვას, რომელიც იწყება და მთავრდება ერთი და იგივე სიმბოლოთი უწოდებენ ალმასისმაგვარს. შეადგინეთ პროგრამა, რომელიც სტრიქონში დათვლის ალმასისმაგვარი სიტყვების რაოდენობას.

თავი 14. კოლექციები

დინამიური მასივები, ჰეშ-ცხრილები, დახარისხებული სიები

14.1.1. შექმენით 10-ელემენტოვანი დინამიური მასივი სტრიქონების მასივის საფუძველზე. დინამიურ მასივს დაუმატეთ ორი სტრიქონი. ეკრანზე გამოიტანეთ დინამიურ მასივში ელემენტების რაოდენობა და თვით დინამიური მასივი.

14.1.2. შექმენით 10-ელემენტოვანი დინამიური მასივი სტრიქონების მასივის საფუძველზე. დინამიური მასივიდან წაშალეთ სამი სტრიქონი. ეკრანზე გამოიტანეთ დინამიურ მასივში ელემენტების რაოდენობა და თვით დინამიური მასივი.

14.1.3. შექმენით 10-ელემენტის დინამიური მასივი სტრიქონების მასივის საფუძველზე. დინამიურ მასივს ჩაუმატეთ სტრიქონი მე-2 პოზიციიდან. ეკრანზე გამოიტანეთ დინამიურ მასივში ელემენტების რაოდენობა და თვით დინამიური მასივი.

14.1.4. შექმენით 10-ელემენტის დინამიური მასივი სტრიქონების მასივის საფუძველზე. დინამიურ მასივში მოძებნეთ თქვენი სახელი. ეკრანზე გამოიტანეთ მოძებნილი ელემენტის ინდექსი, დინამიურ მასივში ელემენტების რაოდენობა და თვით დინამიური მასივი.

14.1.5. შექმენით 10-ელემენტის ჰეშ-ცხრილი. ჰეშ-ცხრილს დაუმატეთ ორი ელემენტი. ეკრანზე გამოიტანეთ ჰეშ-ცხრილში ელემენტების რაოდენობა და ყველა გასაღები და ყველა სიდიდე.

14.1.6. შექმენით 10-ელემენტის ჰეშ-ცხრილი. ჰეშ-ცხრილიდან წაშალეთ ორი ელემენტი. ეკრანზე გამოიტანეთ ჰეშ-ცხრილში ელემენტების რაოდენობა და ყველა გასაღები და ყველა სიდიდე.

14.1.7. შექმენით 10-ელემენტის ჰეშ-ცხრილი. ჰეშ-ცხრილში შეასრულეთ ძებნა გასაღებისა და სიდიდის მიხედვით. ნაპოვნი გასაღები და სიდიდე ეკრანზე გამოიტანეთ. ეკრანზე გამოიტანეთ ჰეშ-ცხრილში ელემენტების რაოდენობა და ყველა გასაღები და ყველა სიდიდე.

14.1.8. შეკუმშავს M_{10} მასივს მისი გამეორებადი ელემენტების წაშლის გზით.

14.1.9. შეკუმშავს M_{10} მასივს მისი იმ ელემენტების წაშლის გზით, რომლებიც მასივში ერთხელ გვხვდება.

14.1.10. შეკუმშავს M_{10} მასივს მისი ნულოვანი ელემენტების წაშლის გზით.

რიგები, სტეკები და ბიტების მასივები

14.2.1. შექმენით 10-ელემენტის რიგი. წაიკითხეთ და ეკრანზე გამოიტანეთ რიგის I ელემენტის მნიშვნელობა მისი წაშლის გარეშე. ეკრანზე გამოიტანეთ რიგში არსებული ელემენტების რაოდენობა და რიგის ყველა ელემენტი. რიგიდან ყველა ელემენტი წაშალეთ.

14.2.2. შექმენით 10-ელემენტის რიგი. წაიკითხეთ და ეკრანზე გამოიტანეთ რიგის I ელემენტის მნიშვნელობა მისი წაშლის გარეშე. ეკრანზე გამოიტანეთ რიგში არსებული ელემენტების რაოდენობა და რიგის ყველა ელემენტი. რიგიდან ყველა ელემენტი წაშალეთ.

14.2.3. შექმენით 10-ელემენტის სტეკი. წაიკითხეთ და ეკრანზე გამოიტანეთ სტეკის უკანასკნელი ელემენტის მნიშვნელობა მისი წაშლის გარეშე. ეკრანზე გამოიტანეთ სტეკში არსებული ელემენტების რაოდენობა და სტეკის ყველა ელემენტი. სტეკიდან ყველა ელემენტი წაშალეთ.

14.2.4. შექმენით 10-ელემენტის სტეკი. წაიკითხეთ და ეკრანზე გამოიტანეთ სტეკის უკანასკნელი ელემენტის მნიშვნელობა მისი წაშლის გარეშე. ეკრანზე გამოიტანეთ სტეკში არსებული ელემენტების რაოდენობა და სტეკის ყველა ელემენტი. სტეკიდან ყველა ელემენტი წაშალეთ.

14.2.5. შექმენით ბიტების 2 მასივი, თითოეული 10-ელემენტის. ამ მასივებზე შეასრულეთ და, ან, არა და გამომრიცხავი ან ოპერაციები.

თავი 15. შესრულების ნაკადები

შესრულების ნაკადები

15.1.1. შექმენით 2 ნაკადი. პირველ ნაკადში მუშაობს მეთოდი, რომელიც ანგარიშობს სამკუთხედის პერიმეტრს. მეორე ნაკადში მუშაობს მეთოდი, რომელიც ანგარიშობს სამკუთხედის ფართობს.

15.1.2. შექმენით 3 ნაკადი. პირველ ნაკადში მუშაობს მეთოდი, რომელიც ანგარიშობს სამკუთხედის პერიმეტრს. მეორე ნაკადში მუშაობს მეთოდი, რომელიც ანგარიშობს

ნაკადების ლოკალური მონაცემები

15.4.1. შექმენით მეთოდი, რომელიც ახდენს ერთგანზომილებიანი მთელრიცხვა მასივის ელემენტების შეკრებას. ეს მეთოდი ორ ნაკადში შეასრულეთ. ორივე ნაკადისთვის მეხსიერების ერთი და იგივე უბნები გამოიყენეთ.

15.4.2. შექმენით მეთოდი, რომელიც ახდენს ერთგანზომილებიანი მთელრიცხვა მასივის ელემენტების შეკრებას. ეს მეთოდი სამ ნაკადში შეასრულეთ. სამივე ნაკადისთვის მეხსიერების ერთი და იგივე უბნები გამოიყენეთ.

Timer კლასი

15.5.1. შექმენით მეთოდი, რომელიც ახდენს ერთგანზომილებიანი მთელრიცხვა მასივის დადებითი ელემენტების შეკრებას. ეს მეთოდი ერთ ნაკადში შეასრულეთ. ნაკადი გაუშვით 5 წამის შემდეგ და შეასრულეთ ყოველი 3 წამის შემდეგ.

15.5.2. შექმენით მეთოდი, რომელიც ახდენს ერთგანზომილებიანი მთელრიცხვა მასივის დადებითი ელემენტების შეკრებას. ეს მეთოდი ერთ ნაკადში შეასრულეთ. ნაკადი გაუშვით დაუყოვნებლივ და შეასრულეთ ყოველი 4 წამის შემდეგ.

Join მეთოდი

15.6.1. შექმენით მეთოდი, რომელიც ახდენს ერთგანზომილებიანი მთელრიცხვა მასივის ელემენტების შეკრებას. ეს მეთოდი ორ ნაკადში შეასრულეთ. პირველი ნაკადი მიაბით მეორეს.

15.6.2. შექმენით ორი მეთოდი. პირველი მათგანი ახდენს ერთგანზომილებიანი მთელრიცხვა მასივის ელემენტების შეკრებას, მეორე კი - ერთგანზომილებიანი მთელრიცხვა მასივის ელემენტების გამრავლებას. ორივე მეთოდი ორ ნაკადში შეასრულეთ. პირველი ნაკადი მიაბით მეორეს.

lock სიტყვა

15.7.1. შექმენით მეთოდი, რომელიც ახდენს მთელი რიცხვების შეკრებას 1-დან 20-მდე. ეს მეთოდი ორ ნაკადში შეასრულეთ. ორივე ნაკადი ერთსა და იმავე ცვლადთან მუშაობს. ამ ცვლადთან ნაკადების მუშაობის მართვისათვის გამოიყენეთ lock სიტყვა.

15.7.2. შექმენით მეთოდი, რომელიც გამოთვლის 5-ის ფაქტორიალს. ეს მეთოდი ორ ნაკადში შეასრულეთ. ორივე ნაკადი ერთსა და იმავე ცვლადთან მუშაობს. ამ ცვლადთან ნაკადების მუშაობის მართვისათვის გამოიყენეთ lock სიტყვა.

Interlocked კლასი

15.8.1. შექმენით მეთოდი, რომელიც ახდენს მთელი რიცხვის უსაფრთხო ინკრემენტს. ეს მეთოდი ორ ნაკადში შეასრულეთ. გამოიყენეთ Interlocked სიტყვა.

15.8.2. შექმენით მეთოდი, რომელიც ახდენს მთელი რიცხვის უსაფრთხო დეკრემენტს. ეს მეთოდი ორ ნაკადში შეასრულეთ. გამოიყენეთ Interlocked სიტყვა.

15.8.3. შექმენით მეთოდი, რომელიც ახდენს მთელი რიცხვისათვის მნიშვნელობის უსაფრთხოდ მინიჭებას. ეს მეთოდი ორ ნაკადში შეასრულეთ. გამოიყენეთ Interlocked სიტყვა.

Monitor კლასი

15.9.1. შექმენით მეთოდი, რომელიც ახდენს მთელი რიცხვების შეკრებას 1-დან 20-მდე. ეს მეთოდი ორ ნაკადში შეასრულეთ. მოახდინეთ ორივე ნაკადის მუშაობის სინქრონიზება საერთო ცვლადთან მუშაობის დროს Monitor კლასის გამოყენებით.

15.9.2. შექმენით მეთოდი, რომელიც გამოთვლის 7-ის ფაქტორიალს. ეს მეთოდი ორ ნაკადში შეასრულეთ. მოახდინეთ ორივე ნაკადის მუშაობის სინქრონიზება საერთო ცვლადთან მუშაობის დროს Monitor კლასის გამოყენებით.

Mutex კლასი

15.10.1. შექმენით მეთოდი, რომელიც ახდენს მთელი რიცხვების შეკრებას 1-დან 20-მდე. ეს მეთოდი ორ ნაკადში შეასრულეთ. მოახდინეთ ორივე ნაკადის მუშაობის სინქრონიზება საერთო ცვლადთან მუშაობის დროს Mutex კლასის გამოყენებით.

15.10.2. შექმენით მეთოდი, რომელიც ახდენს ლუწი რიცხვების გამრავლებას 1-დან 20-მდე. ეს მეთოდი ორ ნაკადში შეასრულეთ. მოახდინეთ ორივე ნაკადის მუშაობის სინქრონიზება საერთო ცვლადთან მუშაობის დროს Mutex კლასის გამოყენებით.

თავი 16. საბაზო ბიბლიოთეკის სხვა კლასები

გლობალიზება

16.1.1. შექმენით ქართული კულტურა. ეკრანზე გამოიტანეთ კულტურის ეროვნული და ინგლისური დასახელება, თარიღისა და დროის ფორმატი, რიცხვის ფორმატი, ვალუტის სიმბოლო, გამყოფი სიმბოლო.

16.1.2. შექმენით დანიის კულტურა. ეკრანზე გამოიტანეთ კულტურის ეროვნული და ინგლისური დასახელება, თარიღისა და დროის ფორმატი, რიცხვის ფორმატი, ვალუტის სიმბოლო, გამყოფი სიმბოლო.

16.1.3. შექმენით ლატვიის კულტურა. ეკრანზე გამოიტანეთ კულტურის ეროვნული და ინგლისური დასახელება, თარიღისა და დროის ფორმატი, რიცხვის ფორმატი, ვალუტის სიმბოლო, გამყოფი სიმბოლო.

16.1.4. შექმენით საბერძნეთის კულტურა. ეკრანზე გამოიტანეთ კულტურის ეროვნული და ინგლისური დასახელება, თარიღისა და დროის ფორმატი, რიცხვის ფორმატი, ვალუტის სიმბოლო, გამყოფი სიმბოლო.

დაშიფვრა-გაშიფვრა

16.2.1. ფაილში ჩაწერეთ დაშიფრული სტრიქონი: თქვენი სახელი და გვარი. შემდეგ ამავე ფაილიდან წაიკითხეთ ეს სტრიქონი, მოახდინეთ მისი გაშიფვრა და ეკრანზე გამოტანა. გამოიყენეთ სიმეტრიული კრიპტოგრაფია.

16.2.2. ფაილში ჩაწერეთ დაშიფრული სტრიქონი: თქვენი სახელი და გვარი. შემდეგ ამავე ფაილიდან წაიკითხეთ ეს სტრიქონი, მოახდინეთ მისი გაშიფვრა და ეკრანზე გამოტანა. გამოიყენეთ ასიმეტრიული კრიპტოგრაფია.

16.2.3. ფაილში ჩაწერეთ დაშიფრული ერთგანზომილებიანი მთელრიცხვა მასივი. შემდეგ ამავე ფაილიდან წაიკითხეთ ეს მასივი, მოახდინეთ მისი გაშიფვრა და ეკრანზე გამოტანა. გამოიყენეთ სიმეტრიული კრიპტოგრაფია.

16.2.4. ფაილში ჩაწერეთ დაშიფრული ერთგანზომილებიანი მთელრიცხვა მასივი. შემდეგ ამავე ფაილიდან წაიკითხეთ ეს მასივი, მოახდინეთ მისი გაშიფვრა და ეკრანზე გამოტანა. გამოიყენეთ ასიმეტრიული კრიპტოგრაფია.

სხვადასხვა

კომბინატორიკა

1.1. შეადგინეთ პროგრამა, რომელიც დააფორმირებს ხუთკაციანი გუნდის ყველა შესაძლო ვარიანტს არსებული ცხრა კანდიდატურისაგან.

1.2. შეადგინეთ პროგრამა, რომელიც მოძებნის ისეთ ოთხნიშნა რიცხვებს, რომელთა ციფრების

ჯამი წინასწარ მოცემული N რიცხვის ტოლია.

1.3. შეადგინეთ პროგრამა, რომელიც დააფორმირებს 1,2,3,4 ციფრების ყველა შესაძლო გადაადგილებას.

1.4. შეადგინეთ პროგრამა, რომელიც 1,2,3,4,5,6,7 ციფრებისაგან დააფორმირებს ყველა შესაძლო ოთხთანრიგა რიცხვს.

დახარისხება

2.1. შეადგინეთ პროგრამა, რომელიც ერთგანზომილებიანი მასივის ელემენტებს ზრდადობის მიხედვით დაალაგებს.

2.2. შეადგინეთ პროგრამა, რომელიც ერთგანზომილებიანი მასივის ელემენტებს კლებადობის მიხედვით დაალაგებს.

2.3. შეადგინეთ პროგრამა, რომელიც ორგანზომილებიანი მასივის თითოეული სტრიქონის ელემენტებს კლებადობის მიხედვით დაალაგებს.

2.4. შეადგინეთ პროგრამა, რომელიც ორგანზომილებიანი მასივის თითოეული სვეტის ელემენტებს ზრდადობის მიხედვით დაალაგებს.

2.5. შეადგინეთ პროგრამა, რომელიც შეამოწმებს ერთგანზომილებიანი მასივი არის თუ არა ზრდადობის მიხედვით დალაგებული.

2.6. შეადგინეთ პროგრამა, რომელიც შეამოწმებს ერთგანზომილებიანი მასივი არის თუ არა კლებადობის მიხედვით დალაგებული.

2.7. მასივი დალაგებულია ზრდადობის მიხედვით. შეადგინეთ პროგრამა, რომელიც ამ მასივში ჩასვამს R რიცხვს მასივის ზრდადობის მიხედვით მოწესრიგების დაურღვევლად.

2.8. მასივი დალაგებულია კლებადობის მიხედვით. შეადგინეთ პროგრამა, რომელიც ამ მასივში ჩასვამს R რიცხვს მასივის კლებადობის მიხედვით მოწესრიგების დაურღვევლად.

2.9. მოცემულია ზრდადობის მიხედვით დალაგებული ორი მასივი. შეადგინეთ პროგრამა, რომელიც ამ ორი მასივისაგან შექმნის მესამე მასივს, რომლის ელემენტებიც, ასევე, ზრდადობის მიხედვით იქნება დალაგებული.

2.10. შეადგინეთ პროგრამა, რომელიც მაგისტრების გვარებს ანბანის მიხედვით დაალაგებს.

კალენდარი

შეადგინეთ პროგრამა, რომელიც:

3.1. კვირის დღის ნომრის მიხედვით კვირის შესაბამისი დღის დასახელებას ქართულ ენაზე გამოიტანს ეკრანზე.

3.2. რიცხვის, თვისა და წლის მიხედვით კვირის დღეს განსაზღვრავს.

3.3. ეკრანზე ნაკიან წლებს გამოიტანს. ნაკიანია წელი თუ მისი ნომერი უნაშთოდ იყოფა 4-ზე.

3.4. განსაზღვრავს თუ რამდენჯერ მოხვდა თქვენი დაბადების დღე კვირის თითოეულ დღეზე.

3.5. განსაზღვრავს თვის იმ რიცხვებს, როცა მოუწევს ყოველი თვის უკანასკნელი კვირა დღე.

3.6. შეადგინეთ პროგრამა, რომელსაც თარიღი შემდეგი სახით მიეწოდება - 19.03.05. პროგრამამ ეკრანზე უნდა გამოიტანოს 2005 წლის 19 მარტი.

3.7. შეადგინეთ პროგრამა, რომელსაც თარიღი მიეწოდება. პროგრამამ უნდა განსაზღვროს შესაბამისი დღის ნომერი წლის დასაწყისიდან.

3.8. შეადგინეთ პროგრამა, რომელსაც ორი თარიღი მიეწოდება. პროგრამამ უნდა გამოთვალოს:

ა. დღეების რაოდენობა, რომელიც ამ ორ თარიღს შორის გავიდა;

ბ. თვეების რაოდენობა, რომელიც ამ ორ თარიღს შორის გავიდა;

გ. წლების რაოდენობა, რომელიც ამ ორ თარიღს შორის გავიდა;

3.9. შეადგინეთ პროგრამა, რომელსაც თარიღი მიეწოდება. პროგრამამ უნდა:

ა. შეამოწმოს თარიღის კორექტულობა. მაგალითად, არაკორექტულია 2005 წლის 30 თებერვალი;

ბ. განსაზღვროს თუ რამდენი დღე დარჩა წლის ბოლომდე;

გ. განსაზღვროს თუ რამდენი დღე გავიდა წლის დასაწყისიდან.

3.10. მოცემულია თვე და რიცხვი. შეადინეთ პროგრამა, რომელიც განსაზღვრავს თუ რომელ დღეზე მოდის ეს თარიღი თუ წელი არანაკიანია, 1 იანვარი კი - ხუთშაბათი.

პედაგოგიკა

შეადგინეთ პროგრამა, რომელიც:

4.1. შეამოწმებს გამრავლების ტაბულის ცოდნას.

4.2. შეამოწმებს საისტორიო თარიღების ცოდნას. პროგრამამ ეკრანზე უნდა გამოიტანოს ისტორიული მოვლენის დასახელება, თქვენ კი უნდა შეიტანოთ შესაბამისი თარიღი.

4.3. შეამოწმებს:

ა. გამრავლების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;

ბ. შეკრების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;

გ. გამოკლების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;

დ. გაყოფის ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;

ე. ახარისხების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში;

ვ. კვადრატული ფესვის ამოღების ოპერაციის შესრულების სისწორეს 100-ის ფარგლებში.

დანართი 2. სავარჯიშოების ამოხსნები

თავი 2. C# ენის საფუძვლები

არითმეტიკული გამოსახულებები

2.1.4.

```
{
    double x, y;

    x = Convert.ToDouble(textBox1.Text);
    y = ( 1 + x ) * ( x - Math.Sqrt(x - 1) / 4 ) / ( Math.Exp(x) + 1 / ( Math.Pow(x, 2) + 4 ) );
    label1.Text = Math.Round(y, 2).ToString();
}
```

2.1.10.

```
{
    int a, b, c, Perimetri;
    double Partobi;

    a = Convert.ToInt32(textBox1.Text);
    b = Convert.ToInt32(textBox2.Text);
    c = Convert.ToInt32(textBox3.Text);
    Perimetri = ( a + b + c ) / 2;
    Partobi = Math.Sqrt( Perimetri * ( Perimetri - a ) * ( Perimetri - b ) * ( Perimetri - c ) );
    label1.Text = Math.Round(Partobi, 2).ToString();
}
```

2.1.30.

```
{
    int gverdi1, gverdi2, gverdi3;
    gverdi1 = int.Parse(textBox1.Text);
    gverdi2 = int.Parse(textBox2.Text);
    gverdi3 = int.Parse(textBox3.Text);
    if ((gverdi1 + gverdi2 > gverdi3) && (gverdi1 + gverdi3 > gverdi2) &&
        (gverdi2 + gverdi3 > gverdi1))
        label1.Text = "მოცემული სიგრძის მქონე მონაკვეთებით “ +
            “შესაძლებელია სამკუთხედის აგება”;
    else
        label1.Text = "მოცემული სიგრძის მქონე მონაკვეთებით “ +
            “შეუძლებელია სამკუთხედის აგება”;
}
```

2.1.31.

```
{
    int gverdi1, gverdi2, gverdi3;
    gverdi1 = int.Parse(textBox1.Text);
    gverdi2 = int.Parse(textBox2.Text);
    gverdi3 = int.Parse(textBox3.Text);
    if ((Math.Pow(gverdi1,2) + Math.Pow(gverdi2,2) == Math.Pow(gverdi3,2)) ||
```

```

(Math.Pow(gverdi1,2) + Math.Pow(gverdi3,2) == Math.Pow(gverdi2,2)) ||
(Math.Pow(gverdi2,2) + Math.Pow(gverdi3,2) == Math.Pow(gverdi1,2)))
label1.Text = "მოცემული სიგრძის მქონე მონაკვეთებით “ +
    “შესაძლებელია მართკუთხა სამკუთხედის აგება”;
else label1.Text = "მოცემული სიგრძის მქონე მონაკვეთებით “ +
    “შეუძლებელია მართკუთხა სამკუთხედის აგება”;
}

```

ლოგიკური გამოსახულებები

2.2.3.

```

{
bool y, x1, x2, x3, x4;

x1 = Convert.ToBoolean(textBox1.Text); // x1 = True
x2 = Convert.ToBoolean(textBox2.Text); // x2 = True
x3 = Convert.ToBoolean(textBox3.Text); // x3 = False
x4 = Convert.ToBoolean(textBox4.Text); // x4 = False
y = !x1 && !x2 && x3 || x4; // y = False
label1.Text = y.ToString();
}

```

თავი 3. მმართველი ოპერატორები

if ოპერატორი

3.1.3.

```

{
int ricxvi;

ricxvi = Convert.ToInt32(textBox1.Text);
if ( ricxvi % 5 == 0 ) label1.Text = "რიცხვი 5-ის ჯერადაა";
else label1.Text = "რიცხვი 5-ის ჯერადა არ არის";
}

```

3.1.8.

```

{
int ricxvi1, ricxvi2;

ricxvi1 = Convert.ToInt32(textBox1.Text);
ricxvi2 = Convert.ToInt32(textBox2.Text);
if ( ricxvi1 >= ricxvi2 ) label1.Text = "პირველი რიცხვი მეორეზე მეტია ან მისი ტოლია";
else label1.Text = "მეორე რიცხვი პირველზე მეტია";
}

```

ჩადგმული if ოპერატორი

3.2.1.

```

{

```

```

int ricxvi1 = Convert.ToInt32(textBox1.Text);
int ricxvi2 = Convert.ToInt32(textBox2.Text);
int ricxvi3 = Convert.ToInt32(textBox3.Text);

if ( ricxvi1 >= ricxvi2 ) if ( ricxvi1 >= ricxvi3 ) label1.Text = "პირველი რიცხვი მაქსიმალურია";
    else label1.Text = "მესამე რიცხვი მაქსიმალურია";
    else if ( ricxvi2 >= ricxvi3 ) label1.Text = "მეორე რიცხვი მაქსიმალურია";
    else label1.Text = "მესამე რიცხვი მაქსიმალურია";
}

```

3.2.8.

```

{
int ricxvi1 = Convert.ToInt32(textBox1.Text);
int ricxvi2 = Convert.ToInt32(textBox2.Text);
int ricxvi3 = Convert.ToInt32(textBox3.Text);

if (ricxvi1 < 0) label1.Text = "სამ რიცხვს შორის არის უარყოფითი";
else if (ricxvi2 < 0) label1.Text = "სამ რიცხვს შორის არის უარყოფითი";
    else if (ricxvi3 < 0) label1.Text = "სამ რიცხვს შორის არის უარყოფითი";
    else label1.Text = "სამ რიცხვს შორის არ არის უარყოფითი";
}

```

ეს ამოცანა შეიძლება გადაწყდეს ლოგიკის გამოყენებით.

```

{
int ricxvi1 = Convert.ToInt32(textBox1.Text);
int ricxvi2 = Convert.ToInt32(textBox2.Text);
int ricxvi3 = Convert.ToInt32(textBox3.Text);

if ( ( ricxvi1 < 0 ) || ( ricxvi2 < 0 ) || ( ricxvi3 < 0 ) )
    label1.Text = "სამ რიცხვს შორის არის უარყოფითი";
    else label1.Text = "სამ რიცხვს შორის არ არის უარყოფითი";
}

```

3.2.9.

```

{
int raodenoba = 0;
int ricxvi1 = Convert.ToInt32(textBox1.Text);
int ricxvi2 = Convert.ToInt32(textBox2.Text);
int ricxvi3 = Convert.ToInt32(textBox3.Text);

if ( ricxvi1 % 2 == 1 ) raodenoba++;
if ( ricxvi2 % 2 == 1 ) raodenoba++;
if ( ricxvi3 % 2 == 1 ) raodenoba++;
label1.Text = raodenoba.ToString();
}

```

ლოგიკა

3.3.2.

```

{
int ricxvi = Convert.ToInt32(textBox1.Text);

```

```

if ( ( ricxvi >= 10 ) && ( ricxvi < 19 ) )
    label1.Text = "რიცხვი მოთავსებულია მითითებულ დიაპაზონში";
else label1.Text = "რიცხვი მოთავსებული არ არის მითითებულ დიაპაზონში";
}

```

3.3.3.

```

{
int ricxvi = Convert.ToInt32(textBox1.Text);
if ( ( ricxvi < 25 ) || ( ricxvi > 100 ) )
    label1.Text = "რიცხვი მოთავსებულია მითითებულ დიაპაზონში";
else label1.Text = "რიცხვი მოთავსებული არ არის მითითებულ დიაპაზონში";
}

```

for, while, do-while ოპერატორები

3.4.13.

```

{
double jami = 0;
int mnishvneli, minusi = -1;

for (mnishvneli = 1; mnishvneli <= 5; mnishvneli++)
{
    minusi *= -1;
    jami += minusi * ( (double) 1 / mnishvneli);
}
label1.Text = Math.Round(jami, 2).ToString();
}

```

3.4.36.

```

{
double ricxvi, jami = 0;
int mnishvneli;

ricxvi = Convert.ToDouble(textBox1.Text);
for ( mnishvneli = 1; mnishvneli <= 7; mnishvneli += 2 )
    jami += Math.Pow(ricxvi, mnishvneli) / mnishvneli;
jami *= 2;
label1.Text = Math.Round(jami, 2).ToString();
}

```

3.4.38.

```

{
int jami = 0;
for ( int ricxvi = 10; ricxvi <= 20; ricxvi++ )
    if ( ricxvi % 2 == 1 ) jami += ricxvi;
label1.Text = jami.ToString();
}

```

ამ ამოცანის გადაწყვეტის მეორე გზა

```

{

```

```

int jami = 0;
for ( int ricxvi = 11; ricxvi <= 20; ricxvi += 2 )
    jami += ricxvi;
label1.Text = jami.ToString();
}

```

3.4.44.

```

{
    label1.Text = "";
    int ricxvi, tanrigi1, tanrigi2, tanrigi3;

    for ( ricxvi = 200; ricxvi <= 500; ricxvi++ )
    {
        tanrigi1 = ricxvi % 10;
        tanrigi2 = ( ricxvi % 100 ) / 10;
        tanrigi3 = ricxvi / 100;
        if ( ricxvi % ( tanrigi1 + tanrigi2 + tanrigi3 ) == 0 ) label1.Text += ricxvi.ToString() + " ";
    }
}

```

3.4.45.

```

{
    label1.Text = "";
    int ricxvi, jami = 0;
    ricxvi = int.Parse(textBox1.Text);

    for ( ; ricxvi > 0; )
    {
        jami += ricxvi % 10;
        ricxvi /= 10;
    }
    label1.Text = jami.ToString();
}

```

3.4.46.

```

{
    label1.Text = "";
    int ricxvi, kvadratuli_fesvi;
    ricxvi = int.Parse(textBox1.Text);
    kvadratuli_fesvi = (int) Math.Sqrt(ricxvi);

    for ( int i = 1; i <= kvadratuli_fesvi; i++)
        if ( ricxvi % i == 0 ) label1.Text += i.ToString() + " * " + (ricxvi / i).ToString() + '\n';
}

```

3.4.47.

```

{
    label1.Text = "";
    int ricxvi, nakhevari, mamravli;
    ricxvi = int.Parse(textBox1.Text);
    nakhevari = ricxvi / 2;
}

```

```

for (mamravli = 2; mamravli <= nakhevari; )
    if (ricxvi % mamravli == 0)
    {
        ricxvi /= mamravli;
        label1.Text += mamravli.ToString() + " ";
    }
    else mamravli++;
}

```

3.4.48.

```

{
    int x, y;
    x = int.Parse(textBox1.Text);
    y = int.Parse(textBox2.Text);
    for (; x != y; )
    {
        if (x > y)
            x = x - y;
        else
            y = y - x;
    }
    label1.Text = x.ToString();
}

```

continue, break ოპერატორები

3.5.1.

```

{
    int jami = 0;

    for (int ricxvi = 1; ricxvi <= 20; ricxvi++)
        if (jami > 24) break;
        else jami += ricxvi;
    label1.Text = jami.ToString();
}

```

3.5.2.

```

{
    int raodenoba = 0;

    for ( int ricxvi = 6; ricxvi <= 79; ricxvi++ )
    {
        if ( ricxvi % 4 == 0 ) raodenoba++;
        if ( ricxvi > 50 ) break;
    }
    label1.Text = raodenoba.ToString();
}

```

თავი 4. მასივები, სტრიქონები და ბიტობრივი ოპერაციები

ერთგანზომილებიანი მასივები

4.1.5.

```
{
int[] masivi = new int[] { 1, -5, 20, -2, 4};
int max = masivi[0], maxind = 0;

for ( int indexi = 1; indexi < masivi.Length; indexi++)
    if ( max < masivi[indexi] )
    {
        max = masivi[indexi];
        maxind = indexi;
    }
label1.Text = "მასივის მაქსიმალური ელემენტია - " + max.ToString() + '\n' +
    "მაქსიმალური ელემენტის ინდექსია - " + maxind.ToString();
for ( int indexi = 0; indexi < masivi.Length; indexi++)
    label2.Text += masivi[indexi].ToString() + " ";
}
```

4.1.8.

```
{
int[] masivi = new int[] { 1, -5, 20, -2, 4 };
int jami = 0;

for ( int indexi = 0; indexi < masivi.Length; indexi++ )
    if ( masivi[indexi] >= 0 ) jami += masivi[indexi];
label1.Text = "დადებითი ელემენტების ჯამია - " + jami.ToString();
for ( int indexi = 0; indexi < masivi.Length; indexi++ )
    label2.Text += masivi[indexi].ToString() + " ";
}
```

4.1.9.

```
{
int[] masivi = new int[] { 1, -5, 20, -2, 4 };
int raodenoba = 0;

for ( int indexi = 0; indexi < masivi.Length; indexi++ )
    if ( masivi[indexi] < 0 ) raodenoba++;
label1.Text = "უარყოფითი ელემენტების რაოდენობაა - " + raodenoba.ToString();
for ( int indexi = 0; indexi < masivi.Length; indexi++ )
    label2.Text += masivi[indexi].ToString() + " ";
}
```

4.1.12.

```
{
int[] masivi = new int[] { 1, 5, 20, -2, 4 };
int jami = 0;

for ( int indexi = 0; indexi < masivi.Length; indexi += 2 )
```

```

        if ( masivi[indexi] >= 0 ) jami += masivi[indexi];
        label1.Text = "ლუწი ინდექსის მქონე ელემენტების ჯამია - " + jami.ToString();
        for ( int indexi = 0; indexi < masivi.Length; indexi++ )
            label2.Text += masivi[indexi].ToString() + " ";
    }

```

4.1.15.

```

{
    int[] masivi = new int[] { 1 , 5 , 20 , -2 , 4 , 75 , -3 , 85 , 9 , 21 };
    int indexi;

    for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
        if ( masivi[indexi] < 0 ) break;
    label1.Text = "პირველი უარყოფითი ელემენტია - " +
        masivi[indexi].ToString() + " მისი ინდექსია - " + indexi.ToString();
}

```

4.1.17.

```

{
    int[] masivi = new int[] { 1, 5, -20, -2, 4, 75, -3, -85, 9, -21 };
    int indexi, max = masivi[0], maxind = 0;

    for ( indexi = 0; indexi < masivi.Length; indexi++ )
        if ( masivi[indexi] < 0 ) break;

    max = masivi[indexi];
    maxind = indexi;

    for ( int indexi1 = indexi; indexi1 < masivi.Length; indexi1++ )
        if ( ( masivi[indexi1] < 0 ) && ( max < masivi[indexi1] ) )
        {
            max = masivi[indexi1];
            maxind = indexi1;
        }
}

```

```

label1.Text = "მაქსიმალური უარყოფითი რიცხვია - " +
    max.ToString() + "\nმისი ინდექსია - " + maxind.ToString();
    for ( indexi = 0; indexi < masivi.Length; indexi++ )
        label2.Text += masivi[indexi].ToString() + " ";
}

```

4.1.22.

```

{
    int[] masivi = new int[] { 1, 5, -20, -2, 4, 75, -3, -85, 9, -21 };
    int indexi, min, max, raodenoba = 0;

    min = Convert.ToInt32(textBox1.Text);
    max = Convert.ToInt32(textBox2.Text);
    for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
        if ( ( min <= masivi[indexi] ) && ( max >= masivi[indexi] ) ) raodenoba++;
}

```



```

    label1.Text = raodenoba.ToString();
    for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
        label2.Text += masivi[indexi].ToString() + " ";
    }

```

4.1.25.

```

{
int[] masivi = new int[] { 3, 9, 6, -1, 12, 10, 17, 4, 6, 19 };
int ind, jami = 0;
int min_ind = Convert.ToInt32(textBox1.Text);
int max_ind = Convert.ToInt32(textBox2.Text);
for (ind = min_ind; ind <= max_ind; ind++)
    jami += masivi [ind];
label3.Text = jami.ToString();
}

```

4.1.26.

```

{
int[] masivi = new int[] { 3, 9, 6, -1, 12, 10, 17, 4, 6, 19 };
int ind, max;
//    პირველი ელემენტის ინდექსი
int min_ind = Convert.ToInt32(textBox1.Text);
//    მეორე ელემენტის ინდექსი
int max_ind = Convert.ToInt32(textBox2.Text);
max = masivi[min_ind];
for (ind = min_ind; ind <= max_ind; ind++)
    if ( max < masivi[ind] ) max = masivi[ind];
label3.Text = max.ToString();
}

```

4.1.28.

```

{
label1.Text = "";
int[] masivi1 = new int[] { 3, 29, 6, -1, 12, 40, 17, 64, 6, 79 };
int[] masivi2 = new int[masivi1.Length];
int ind1, ind2 = 0, ricxvi;
ricxvi = Convert.ToInt32(textBox1.Text);
for ( ind1 = 0; ind1 < masivi1.Length; ind1++ )
    if ( masivi1[ind1] < ricxvi ) masivi2[ind2++] = masivi1[ind1];
for ( ind1 = 0; ind1 < ind2; ind1++ )
    label1.Text += masivi2[ind1].ToString() + " ";
}

```

4.1.31.

```

{
    int[] masivi1 = new int[] { 1 , -5 , 20 , -2 , 4, 75, -3, 85, 9, 21 };
    int[] masivi2 = new int[masivi1.Length];
    int indexi1 , indexi2 = 0;

    for ( indexi1 = 0 ; indexi1 < masivi1.Length ; indexi1++ )
        if ( masivi1[indexi1] % 5 == 0 ) masivi2[indexi2++] = masivi1[indexi1];
}

```

```

for ( indexi1 = 0 ; indexi1 < masivi1.Length ; indexi1++ )
    label2.Text += masivi1[indexi1].ToString() + " ";
for ( indexi1 = 0 ; indexi1 < indexi2 ; indexi1++ )
    label3.Text += masivi2[indexi1].ToString() + " ";
}

```

4.1.36.

```

{
int[] masivi1 = new int[] { 1 , -5 , 20 , -2 , 4 , 75 , -3 , 85 , 9 , 21 };
int[] masivi2 = new int[masivi1.Length];
int indexi1 , indexi2 = 0;

for ( indexi1 = 0 ; indexi1 < masivi1.Length ; indexi1++ )
    if ( masivi1[indexi1] >= 0 ) masivi2[indexi2++] = indexi1;

for ( indexi1 = 0 ; indexi1 < masivi1.Length ; indexi1++ )
    label2.Text += masivi1[indexi1].ToString() + " ";
for ( indexi1 = 0 ; indexi1 < indexi2 ; indexi1++ )
    label3.Text += masivi2[indexi1].ToString() + " ";
}

```

4.1.45.

```

{
int[] masivi = new int[] { 1 , -5 , 20 , -2 , 4 , 75 , -3 , 85 , 9 , 21 };
int indexi, temp;

for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
    label2.Text += masivi[indexi].ToString() + " ";
for ( indexi = 0 ; indexi < masivi.Length / 2 ; indexi++ )
{
    temp = masivi[indexi];
    masivi[indexi] = masivi[masivi.Length - 1 - indexi];
    masivi[masivi.Length - 1 - indexi] = temp;
}
for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
    label3.Text += masivi[indexi].ToString() + " ";
}

```

4.1.48.

```

{
int[] masivi = new int[] { 1, -5, 0, -2, 0, 0, -3, 0, 9, 0 };
int indexi, temp, Alami;

for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
    label2.Text += masivi[indexi].ToString() + " ";
M1: Alami = 0;
for ( indexi = 0 ; indexi < masivi.Length - 1; indexi++ )
    if ( ( masivi[indexi] == 0 ) && ( masivi[indexi + 1] != 0 ) )
{

```

```

    temp = masivi[indexi];
    masivi[indexi] = masivi[indexi + 1];
    masivi[indexi + 1] = temp;
    Alami = 1;
}
if ( Alami == 1 ) goto M1;
for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
    label3.Text += masivi[indexi].ToString() + " ";
}

```

4.1.49.

```

{
    int[] masivi = new int[] { 1, 5, 2, 4, 7 };
    int indexi, temp, Zvris_Raodenoba = Convert.ToInt32(textBox1.Text);

    for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
        label2.Text += masivi[indexi].ToString() + " ";
    for ( int i = 1 ; i <= Zvris_Raodenoba ; i++ )
    {
        temp = masivi[masivi.Length - 1];
        for ( indexi = 0 ; indexi < masivi.Length - 1 ; indexi++ )
            masivi[masivi.Length - 1 - indexi] = masivi[masivi.Length - 2 - indexi];
        masivi[0] = temp;
    }
    for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
        label3.Text += masivi[indexi].ToString() + " ";
}

```

4.1.50.

```

{
    int[] masivi = new int[] { 1, 5, 2, 4, 7 };
    int indexi, temp, Zvris_Raodenoba = Convert.ToInt32(textBox1.Text);

    for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
        label2.Text += masivi[indexi].ToString() + " ";
    for ( int i = 1 ; i <= Zvris_Raodenoba ; i++ )
    {
        temp = masivi[0];
        for ( indexi = 0 ; indexi < masivi.Length - 1 ; indexi++ )
            masivi[indexi] = masivi[indexi + 1];
        masivi[masivi.Length - 1] = temp;
    }
    for ( indexi = 0 ; indexi < masivi.Length ; indexi++ )
        label3.Text += masivi[indexi].ToString() + " ";
}

```

4.1.71.

```

{
    int[] masivi1 = new int[] { 1, 2, 1, 3, 4, 2, 5, 3, 7, 1, 3, 8, 4, 1, 2 };
    int[] masivi2 = new int[masivi1.Length];
}

```

```

int max = 0, maxind = 0, indexi;

for (indexi = 0; indexi < masivi1.Length; indexi++)
    masivi2[masivi1[indexi]]++;
for (indexi = 0; indexi < masivi2.Length; indexi++)
    if (masivi2[indexi] > max)
    {
        max = masivi2[indexi];
        maxind = indexi;
    }
label1.Text = max.ToString() + " " + maxind.ToString();
}

```

ორგანზომილებიანი მასივები

4.2.2.

```

{
int[,] masivi = new int[3, 3] { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
int striqoni, sveti, minstriqoni = 0, minsveti = 0, min = masivi[0, 0];

for (striqoni = 0; striqoni < 3; striqoni++)
    for (sveti = 0; sveti < 3; sveti++)
        if (min > masivi[striqoni, sveti])
        {
            min = masivi[striqoni, sveti];
            minstriqoni = striqoni;
            minsveti = sveti;
        }
label1.Text = "მინიმალური ელემენტია - " + min.ToString() +
    "\nსტრიქონის ინდექსია - " + minstriqoni.ToString() +
    "\nსვეტის ინდექსია - " + minsveti.ToString();
for (striqoni = 0; striqoni < 3; striqoni++)
{
    for (sveti = 0; sveti < 3; sveti++)
        label2.Text += masivi[striqoni, sveti].ToString() + " ";
    label2.Text += '\n';
}
}

```

4.2.7.

```

{
int[,] masivi = new int[3, 3] { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
int striqoni, sveti, jami = 0;

for (striqoni = 0; striqoni < 3; striqoni++)
    for (sveti = 0; sveti < 3; sveti++)
        if (masivi[striqoni, sveti] % 2 == 1)
            jami += masivi[striqoni, sveti];
label1.Text = jami.ToString();
for (striqoni = 0; striqoni < 3; striqoni++)

```

```

    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
}

```

4.2.8.

```

{
    int[,] masivi = new int[3, 3] { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
    int striqoni, sveti, raodenoba = 0;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            if ( masivi[striqoni, sveti] % 2 == 0 )
                raodenoba++;
    label1.Text = raodenoba.ToString();
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
}

```

4.2.11.

```

{
    int[,] masivi = new int[3, 3] { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
    int striqoni, sveti, jami = 0;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        jami += masivi[striqoni, striqoni];
    label1.Text = jami.ToString();
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
}

```

4.2.12.

```

int[,] masivi = new int[3, 3] { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
int striqoni, sveti, namravli = 1;

for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    namravli *= masivi[striqoni, 2 - striqoni];
label1.Text = namravli.ToString();
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{

```

```

        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }

```

4.2.18.

```

{
    label1.Text = ""; label2.Text = "";
    int[,] masivi = new int[,] { { 24, 3, -82 }, { 4, 240, 9 }, { 5, 6, -2 } };
    int str, sv, raodenoba = 0, ricxvi = Convert.ToInt32(textBox1.Text);

    for ( str = 0; str < 3; str++ )
        for ( sv = 0; sv < 3; sv++ )
            if ( masivi[str, sv] > ricxvi ) raodenoba++;

    label1.Text = raodenoba.ToString();
    for ( str = 0; str < 3; str++ )
    {
        for ( sv = 0; sv < 3; sv++ )
            label2.Text += masivi[str, sv].ToString() + " ";
        label2.Text += '\n';
    }
}

```

4.2.19.

```

{
    label1.Text = ""; label2.Text = "";
    int[,] masivi = new int[,] { { 24, 3, -82 }, { 4, 240, 9 }, { 5, 6, -2 } };
    int str, sv, raodenoba = 0, ricxvi = Convert.ToInt32(textBox1.Text);

    for ( str = 0; str < 3; str++ )
        for ( sv = 0; sv < 3; sv++ )
            if ( masivi[str, sv] > ricxvi ) raodenoba++;

    label1.Text = raodenoba.ToString();
    for ( str = 0; str < 3; str++ )
    {
        for ( sv = 0; sv < 3; sv++ )
            label2.Text += masivi[str, sv].ToString() + " ";
        label2.Text += '\n';
    }
}

```

4.2.20.

```

{
    int[,] masivi1 = new int[3, 3] { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
    int[] masivi2 = new int[masivi1.Length];
    int indexi2 = 0, striqoni, sveti;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )

```

```

    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        if ( masivi1[striqoni, sveti] % 2 == 0 ) masivi2[indexi2++] = masivi1[striqoni, sveti];
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        label2.Text += masivi1[striqoni, sveti].ToString() + " ";
    label2.Text += '\n';
}
for ( striqoni = 0 ; striqoni < indexi2 ; striqoni++ )
    label3.Text += masivi2[striqoni].ToString() + " ";
}

```

4.2.21.

```

{
    label1.Text = ""; label2.Text = "";
    int[,] masivi1 = new int[,] { { 0, 3, -82 }, { 4, 0, 9 }, { 5, 6, 0 } };
    int[] masivi2 = new int[masivi1.Length];
    int str, sv, indexi2 = 0;

    for ( str = 0; str < 3; str++ )
        for ( sv = 0; sv < 3; sv++ )
            if ( masivi1[str, sv] != 0 ) masivi2[indexi2++] = masivi1[str, sv];

    for ( str = 0; str < indexi2; str++ )
        label1.Text += masivi2[str].ToString() + " ";
    for ( str = 0; str < 3; str++ )
    {
        for ( sv = 0; sv < 3; sv++ )
            label2.Text += masivi1[str, sv].ToString() + " ";
        label2.Text += '\n';
    }
}

```

4.2.22.

```

{
    int[,] masivi1 = new int[3, 3] { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
    int[] masivi2 = new int[masivi1.Length];
    int striqoni, sveti;

    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
            masivi2[sveti] += masivi1[striqoni, sveti];
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi1[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )

```

```

        label3.Text += masivi2[striqoni].ToString() + " ";
    }

```

4.2.23.

```

{
    int[,] masivi1 = new int[3, 3] { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
    int[] masivi2 = new int[3] { 1, 1, 1 };
    int striqoni, sveti;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            masivi2[striqoni] *= masivi1[striqoni, sveti];
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi1[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        label3.Text += masivi2[striqoni].ToString() + " ";
    }
}

```

4.2.24.

```

{
    label1.Text = ""; label2.Text = "";
    int[,] masivi1 = new int[,] { { 24, 3, -82 }, { 4, 0, 9 }, { 5, -6, -2 } };
    int[] masivi2 = new int[3];
    int str, sv, raodenoba = 0;

    for ( sv = 0; sv < 3; sv++ )
    {
        raodenoba = 0;
        for ( str = 0; str < 3; str++ )
            if ( masivi1[str, sv] >= 0 ) raodenoba++;
        masivi2[sv] = raodenoba;
    }

    for ( sv = 0; sv < 3; sv++ )
        label1.Text += masivi2[sv].ToString() + " ";
    for ( str = 0; str < 3; str++ )
    {
        for ( sv = 0; sv < 3; sv++ )
            label2.Text += masivi1[str, sv].ToString() + " ";
        label2.Text += '\n';
    }
}

```

4.2.25.

```

{
    int[,] masivi1 = new int[3, 3] { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };

```



```

int[] masivi2 = new int[3];
int striqoni, sveti;

for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        if ( masivi1[striqoni, sveti] %2 == 1 ) masivi2[striqoni] += masivi1[striqoni, sveti];
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        label2.Text += masivi1[striqoni, sveti].ToString() + " ";
    label2.Text += '\n';
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    label3.Text += masivi2[striqoni].ToString() + " ";
}

```

4.2.26.

```

{
int[, ] masivi1 = new int[3, 3] { { 11, -2, 3 }, { 4, 15, -6 }, { 7, 8, 9 } };
int[] masivi2 = new int[3];
int striqoni, sveti, max;

for ( sveti = 0 ; sveti < 3 ; sveti++ )
{
    max = masivi1[0, sveti];
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        if ( max < masivi1[striqoni, sveti] ) max = masivi1[striqoni, sveti];
    masivi2[sveti] = max;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        label2.Text += masivi1[striqoni, sveti].ToString() + " ";
    label2.Text += '\n';
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    label3.Text += masivi2[striqoni].ToString() + " ";
}

```

4.2.27.

```

{
label1.Text = ""; label2.Text = ""; label3.Text = "";
int[, ] masivi1 = new int[, ] { { 2, 3, -8 }, { 4, 10, 9 }, { 5, 6, -2 } };
int[] masivi2 = new int[3];
int str, sv, min_cvcladi;

for ( str = 0; str < 3; str++ )
{
    min_cvcladi = masivi1[str, 0];
}

```

```

for ( sv = 0; sv < 3; sv++ )
    if ( min_cvladi > masivi1[str, sv] ) min_cvladi = masivi1[str, sv];
masivi2[str] = min_cvladi;
}

```

```

for ( str = 0; str < 3; str++ )
    label1.Text += masivi2[str].ToString() + " ";
for ( str = 0; str < 3; str++ )
{
    for ( sv = 0; sv < 3; sv++ )
        label2.Text += masivi1[str, sv].ToString() + " ";
    label2.Text += '\n';
}
}

```

4.2.28.

```

{
    int[,] masivi1 = new int[3, 3] { { 11, -2, 3 }, { 4, 15, -6 }, { -7, 8, 9 } };
    int[] masivi2 = new int[3];
    int striqoni, sveti, max = masivi1[0,0], maxstriqoni = 0;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        if ( max < masivi1[striqoni, striqoni] )
            {
                max = masivi1[striqoni, striqoni];
                maxstriqoni = striqoni;
            }
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        masivi2[sveti] = masivi1[maxstriqoni, sveti];
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi1[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        label3.Text += masivi2[striqoni].ToString() + " ";
}
}

```

4.2.34.

```

{
    int[,] masivi1 = new int[3, 3] { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
    int[] masivi2 = new int[3];
    int striqoni, sveti;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        masivi2[striqoni] = masivi1[striqoni, striqoni];
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {

```

```

    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        label2.Text += masivi1[striqoni, sveti].ToString() + " ";
    label2.Text += '\n';
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    label3.Text += masivi2[striqoni].ToString() + " ";
}

```

4.2.35.

```

{
    int[,] masivi1 = new int[3, 3] { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
    int[] masivi2 = new int[3];
    int striqoni, sveti;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        masivi2[striqoni] = masivi1[striqoni, 2 - striqoni];
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi1[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        label3.Text += masivi2[striqoni].ToString() + " ";
}

```

4.2.36.

```

{
    int[,] masivi1 = new int[3, 3] { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
    double[] masivi2 = new double[3];
    int striqoni, sveti, indexi2 = 0;
    double sashualo;

    for ( sveti = 0; sveti < 3; sveti += 2 )
    {
        sashualo = 0;
        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
            sashualo += masivi1[striqoni, sveti];
        masivi2[indexi2++] = sashualo / 3;
    }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi1[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
    for ( striqoni = 0 ; striqoni < indexi2 ; striqoni++ )
        label3.Text += masivi2[striqoni].ToString() + " ";
}

```

4.2.37.

```

{
int[,] masivi1 = new int[4, 4] { { 10, 21, 3, 2 }, { 4, 51, 6, 5 }, { 7, 3, 9, 6 }, { 4, 1, 8, 9 } };
double[] masivi2 = new double[4];
int striqoni, sveti, indexi2 = 0;
double sashualo;

for ( striqoni = 1 ; striqoni < 4 ; striqoni += 2 )
{
    sashualo = 1;
    for ( sveti = 0 ; sveti < 4 ; sveti++ )
        sashualo *= masivi1[striqoni, sveti];
    masivi2[indexi2++] = sashualo / 4;
}
for ( striqoni = 0 ; striqoni < 4 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 4 ; sveti++ )
        label2.Text += masivi1[striqoni, sveti].ToString() + " ";
    label2.Text += '\n';
}
for ( striqoni = 0 ; striqoni < indexi2 ; striqoni++ )
    label3.Text += masivi2[striqoni].ToString() + " ";
}

```

4.2.38.

```

{
int[,] masivi1 = new int[4, 4] { { 10, 21, 3, 2 }, { 0, 0, 0, 0 }, { 0, 0, 0, 0 }, { 4, 1, 8, 9 } };
int[] masivi2 = new int[4];
int striqoni, sveti, indexi2 = 0, raodenoba = 0;

for ( striqoni = 0 ; striqoni < 4 ; striqoni++ )
{
    raodenoba = 0;
    for ( sveti = 0 ; sveti < 4 ; sveti++ )
        if ( masivi1[striqoni, sveti] == 0 ) raodenoba++;
    if ( raodenoba == 4 ) masivi2[indexi2++] = striqoni;
}
for ( striqoni = 0 ; striqoni < 4 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 4 ; sveti++ )
        label2.Text += masivi1[striqoni, sveti].ToString() + " ";
    label2.Text += '\n';
}
for ( striqoni = 0 ; striqoni < indexi2 ; striqoni++ )
    label3.Text += masivi2[striqoni].ToString() + " ";
}

```

4.2.39.

```
{
int[,] masivi1 = new int[3, 3] { { 10, 21, 3 }, { 4, 51, 61 }, { 7, 3, 9 } };
int[] masivi2 = new int[3];
int striqoni, sveti, raodenoba = 0, indexi2 = 0;

for ( sveti = 0 ; sveti < 3 ; sveti++ )
{
    raodenoba = 0;
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        if ( masivi1[striqoni, sveti] % 2 == 1 ) raodenoba++;
    if ( raodenoba == 3 ) masivi2[indexi2++] = sveti;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        label2.Text += masivi1[striqoni, sveti].ToString() + " ";
    label2.Text += '\n';
}
for ( striqoni = 0 ; striqoni < indexi2 ; striqoni++ )
    label1.Text += masivi2[striqoni].ToString() + " ";
}
```

4.2.46.

```
{
int[,] masivi = new int[3, 3] { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
int striqoni, sveti, temp;

for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        label2.Text += masivi[striqoni, sveti].ToString() + " ";
    label2.Text += '\n';
}
sveti = Convert.ToInt32(textBox1.Text);
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    temp = masivi[striqoni, sveti];
    masivi[striqoni, sveti] = masivi[striqoni, 2];
    masivi[striqoni, 2] = temp;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        label3.Text += masivi[striqoni, sveti].ToString() + " ";
    label3.Text += '\n';
}
}
```

4.2.47.

```

{
    int[,] masivi = new int[3, 3] { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
    int striqoni, sveti, temp;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
    striqoni = Convert.ToInt32(textBox1.Text);
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
    {
        temp = masivi[striqoni, sveti];
        masivi[striqoni, sveti] = masivi[0, sveti];
        masivi[0, sveti] = temp;
    }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label3.Text += masivi[striqoni, sveti].ToString() + " ";
        label3.Text += '\n';
    }
}

```

4.2.49.

```

{
    label1.Text = ""; label2.Text = "";
    int[,] masivi1 = new int[,] { { 12, 3, -8 }, { 4, 10, 9 }, { 5, 6, -2 } };
    int[] masivi2 = new int[3];
    int str, sv, dzvris_raod = Convert.ToInt32(textBox1.Text);
    //    masivi1 მასივი ძვრად
    for ( str = 0; str < 3; str++ )
    {
        for ( sv = 0; sv < 3; sv++ )
            label1.Text += masivi1[str, sv].ToString() + " ";
        label1.Text += '\n';
    }
    //    masivi1 მასივის ძვრა
    for ( int indexi = 1; indexi <= dzvris_raod; indexi++ )
    {
        for ( sv = 0; sv < 3; sv++ )
            masivi2[sv] = masivi1[0, sv];

        for ( str = 1; str < 3; str++ )
            for ( sv = 0; sv < 3; sv++ )
                masivi1[str - 1, sv] = masivi1[str, sv];
    }
}

```

```

for ( sv = 0; sv < 3; sv++ )
    masivi1[2, sv] = masivi2[sv];
}
//   masivi1 მასივი ძვრის შედეგად
for ( str = 0; str < 3; str++ )
{
    for ( sv = 0; sv < 3; sv++ )
        label2.Text += masivi1[str, sv].ToString() + " ";
    label2.Text += '\n';
}
}

```

4.2.65.

```

{
    int[,] masivi1 = new int[3, 3] { { 11, -2, 3 }, { 4, -15, -6 }, { -7, 8, 9 } };
    int[] masivi2 = new int[3];
    int striqoni, sveti, min, minind, raodenoba = 0;

    for ( sveti = 0 ; sveti < 3 ; sveti++ )
    {
        raodenoba = 0;
        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
            if ( masivi1[striqoni, sveti] >= 0 ) raodenoba++;
        masivi2[sveti] = raodenoba;
    }
    min = masivi2[0];
    minind = 0;
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        if ( min > masivi2[striqoni] )
        {
            min = masivi2[striqoni];
            minind = striqoni;
        }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi1[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
    label1.Text = minind.ToString();
}

```

4.2.66.

```

{
    label2.Text = "";
    int[,] masivi = new int[,] { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };
    int str, sv, jami = 0;
    //

```

```

for ( str = 0; str < 4; str++ )
for ( sv = 0; sv < 4; sv++ )
// მოწმდება მასივის ელემენტი არის თუ არა მთავარი დიაგონალის ზევით
if ( ( str < sv ) && ( str != sv ) ) jami += masivi[str, sv];
label1.Text = jami.ToString();
// ეკრანზე მასივის გამოტანა
for ( str = 0; str < 4; str++ )
{
for ( sv = 0; sv < 4; sv++ )
label2.Text += masivi[str, sv].ToString() + " ";
label2.Text += '\n';
}
}

```

4.2.67.

```

{
label2.Text = "";
int[,] masivi = new int[,] { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };
int str, sv, jami = 0;
//
for ( str = 0; str < 4; str++ )
for ( sv = 0; sv < 4; sv++ )
// მოწმდება მასივის ელემენტი არის თუ არა მთავარი დიაგონალის ქვევით
if ( ( str > sv ) && ( str != sv ) ) jami += masivi[str, sv];
label1.Text = jami.ToString();
// ეკრანზე მასივის გამოტანა
for ( str = 0; str < 4; str++ )
{
for ( sv = 0; sv < 4; sv++ )
label2.Text += masivi[str, sv].ToString() + " ";
label2.Text += '\n';
}
}

```

4.2.71.

```

{
label2.Text = "";
int[,] masivi = new int[,] { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };
int str, sv, jami = 0;
//
for ( str = 0; str < 4; str++ )
for ( sv = 0; sv < 4; sv++ )
// მოწმდება მასივის ელემენტი არის თუ არა არამთავარი დიაგონალის ზევით
if ( str + sv < 3 ) jami += masivi[str, sv];
label1.Text = jami.ToString();
// ეკრანზე მასივის გამოტანა
for ( str = 0; str < 4; str++ )
{

```



```

for ( sv = 0; sv < 4; sv++ )
    label2.Text += masivi[str, sv].ToString() + " ";
label2.Text += '\n';
}
}

```

4.2.72.

```

label2.Text = "";
int[,] masivi = new int[,] { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };
int str, sv, jami = 0;
//
for ( str = 0; str < 4; str++ )
    for ( sv = 0; sv < 4; sv++ )
        // მოწმდება მასივის ელემენტი არის თუ არა არამთავარი დიაგონალის ქვევით
        if ( str + sv > 3 ) jami += masivi[str, sv];
label1.Text = jami.ToString();
// ეკრანზე მასივის გამოტანა
for ( str = 0; str < 4; str++ )
{
    for ( sv = 0; sv < 4; sv++ )
        label2.Text += masivi[str, sv].ToString() + " ";
    label2.Text += '\n';
}
}

```

4.2.73.

```

{
    int[,] masivi1 = new int[3, 3] { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
    int[] masivi2 = new int[3];
    int striqoni, sveti, max, maxind;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        for ( sveti = 0; sveti < 3; sveti++ )
            masivi2[striqoni] += masivi1[striqoni, sveti];
    max = masivi2[0];
    maxind = 0;
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        if ( max < masivi2[striqoni] )
        {
            max = masivi2[striqoni];
            maxind = striqoni;
        }
    label3.Text = maxind.ToString();
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            label2.Text += masivi1[striqoni, sveti].ToString() + " ";
        label2.Text += '\n';
    }
}

```

```

    }
4.2.74.
    {
        int[,] masivi1 = new int[3, 3] { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
        int[] masivi2 = new int[3];
        int striqoni, sveti, max, maxind;

        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
                masivi2[sveti] += masivi1[striqoni, sveti];
        max = masivi2[0];
        maxind = 0;
        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
            if ( max < masivi2[striqoni] )
                {
                    max = masivi2[striqoni];
                    maxind = striqoni;
                }
        label3.Text = maxind.ToString();
        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
            {
                for ( sveti = 0 ; sveti < 3 ; sveti++ )
                    label2.Text += masivi1[striqoni, sveti].ToString() + " ";
                label2.Text += '\n';
            }
    }

```

```

4.2.78.
    {
        int[,] masivi = new int[3, 3] { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
        int striqoni, sveti, temp;
        striqoni = Convert.ToInt32(textBox1.Text);
        sveti = Convert.ToInt32(textBox2.Text);
        temp = masivi[striqoni, sveti];
        for ( int striqoni1 = 0; striqoni1 < 3; striqoni1++ )
            masivi[striqoni1, sveti] = 0;
        for ( int sveti1 = 0 ; sveti1 < 3 ; sveti1++ )
            masivi[striqoni, sveti1] = 0;
        masivi[striqoni, sveti] = temp;
        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
            {
                for ( sveti = 0 ; sveti < 3 ; sveti++ )
                    label2.Text += masivi[striqoni, sveti].ToString() + " ";
                label2.Text += '\n';
            }
    }

```

თავი 5. კლასები, ინკაფსულაცია, მეთოდები

კლასები. ინკაფსულაცია

5.1.1.

```
class Tvitmfrinavi
{
private int banis_tevadoba;
private int manzili;
public int mgzavrebis_raodenoba;
public int gayiduli_bilitebi;
}
private void button1_Click(object sender, EventArgs e)
{
Tvitmfrinavi obieqti_1 = new Tvitmfrinavi();
obieqti_1.mgzavrebis_raodenoba = Convert.ToInt32(textBox1.Text);
obieqti_1.gayiduli_bilitebi = Convert.ToInt32(textBox2.Text);
label1.Text = obieqti_1.mgzavrebis_raodenoba.ToString();
label2.Text = obieqti_1.gayiduli_bilitebi.ToString();
}
```

5.1.2.

```
class Studenti
{
private string gvari;
private string saxeli;
private int asaki;
public string universitetis_dasaxeleba;
public int kursi;
}
private void button2_Click(object sender, EventArgs e)
{
Studenti obieqti_1 = new Studenti();
obieqti_1.universitetis_dasaxeleba = textBox1.Text;
obieqti_1.kursi = Convert.ToInt32(textBox2.Text);
label1.Text = obieqti_1.universitetis_dasaxeleba;
label2.Text = obieqti_1.kursi.ToString();
}
```

მეთოდები

5.2.1.

```
class Studenti_1
{
public double Sashualo_qula(int[] masivi1)
{
int jami = 0;
for (int ind = 0; ind < masivi1.Length; ind++)
jami += masivi1[ind];
return jami / masivi1.Length;
}
```

```

    }
}
private void button3_Click(object sender, EventArgs e)
{
    int [] masivi = new int[] { 60, 87, 71, 90, 94, 58, 83, 57, 70, 65 };
    double shedegi;
    Studenti_1 obieqti_1 = new Studenti_1();
    shedegi = obieqti_1.Sashualo_qula(masivi);
    label1.Text = shedegi.ToString();
}

```

5.2.2.

```

class Studenti_2
{
    string gvari;
    string saxeli;
    int asaki;
    void minicheba(string par1, string par2, int par3)
    {
        gvari = par1;
        saxeli = par2;
        asaki = par3;
    }
    public void gadacema(string par1, string par2, int par3)
    {
        minicheba(par1, par2, par3);
    }
    public void gamotana(Label lab)
    {
        lab.Text = "სტუდენტის გვარი: " + gvari + "\nსტუდენტის სახელი: " + saxeli +
            "\nსტუდენტის ასაკი: " + asaki.ToString();
    }
}
private void button4_Click(object sender, EventArgs e)
{

```

```

    Studenti_2 obieqti_1 = new Studenti_2();
    string gvari = textBox1.Text;
    string saxeli = textBox2.Text;
    int asaki = Convert.ToInt32(textBox3.Text);
    obieqti_1.gadacema(gvari, saxeli, asaki);
    obieqti_1.gamotana(label1);
}

```

5.2.3.

```

class Matarebeli
{
    int vagonebis_raodenoba;
    int mgzavrebis_raodenoba;
    public double biletis_fasi;

```

```

public int gayiduli_biletebis_raodenoba;
public void minicheba(double par1, int par2, int par3, int par4)
{
    biletis_fasi = par1;
    gayiduli_biletebis_raodenoba = par2;
    vagonebis_raodenoba = par3;
    mgzavrebis_raodenoba = par4;
}
public void gamotana()
{
    MessageBox.Show("ბილეთის ფასი - " + biletis_fasi.ToString() +
        "\nგაყიდული ბილეთების რაოდენობა - " + gayiduli_biletebis_raodenoba.ToString() +
        "\nვგაგონების რაოდენობა - " + vagonebis_raodenoba.ToString() +
        "\nმგზავრების რაოდენობა - " + mgzavrebis_raodenoba.ToString());
}
public double mogeba()
{
    return biletis_fasi * gayiduli_biletebis_raodenoba;
}
}
private void button5_Click(object sender, EventArgs e)
{
    int vagonebis_raodenoba = Convert.ToInt32(textBox3.Text);
    int mgzavrebis_raodenoba = Convert.ToInt32(textBox4.Text);
    double shedegi;
    Matarebeli obieqti_1 = new Matarebeli();
    obieqti_1.biletis_fasi = Convert.ToDouble(textBox1.Text);
    obieqti_1.gayiduli_biletebis_raodenoba = Convert.ToInt32(textBox2.Text);
    obieqti_1.minicheba(obieqti_1.biletis_fasi, obieqti_1.gayiduli_biletebis_raodenoba,
        vagonebis_raodenoba, mgzavrebis_raodenoba);
    obieqti_1.gamotana();
    shedegi = obieqti_1.mogeba();
    label1.Text = "ბილეთების გაყიდვით მიღებულ თანხა" + shedegi.ToString();
}

```

კონსტრუქტორი

5.3.1.

```

class Tvitmfrinavi
{
    private int bakis_tevadoba;
    private int manzili_1_litri;
    public Tvitmfrinavi(int par1, int par2)
    {
        bakis_tevadoba = par1;
        manzili_1_litri = par2;
    }
    public void Gamotana(Label lab)

```

```

    {
        lab.Text = "ბაკის ტევადობა = " + bakis_tevadoba.ToString() +
            "\n1 ლიტრი საწვავით გავლილი მანძილი = " + manzili_1_litri.ToString();
    }
    public int Gamotvla()
    {
        return bakis_tevadoba * manzili_1_litri;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    int bakis_tevadoba = Convert.ToInt32(textBox1.Text);
    int manzili_1_litri = Convert.ToInt32(textBox2.Text);
    Tvitmfrinavi obieqti_1 = new Tvitmfrinavi(bakis_tevadoba, manzili_1_litri);
    obieqti_1.Gamotana(label1);
    int shedegi = obieqti_1.Gamotvla();
    label2.Text = "სავსე ბაკით გავლილი მანძილი = " + shedegi.ToString();
}

```

5.3.3.

```

class Martkutxedi
{
    public int gverdi_1;
    public int gverdi_2;
    private int perimetri;
    private int fartobi;
    public Martkutxedi(int par1, int par2)
    {
        gverdi_1 = par1;
        gverdi_2 = par2;
        perimetri = ( gverdi_1 + gverdi_2 ) * 2;
        fartobi = gverdi_1 * gverdi_2;
    }
    public void Gamotana(Label lab1)
    {
        lab1.Text = "პერიმეტრი = " + perimetri.ToString() + "ფართობი = " + fartobi.ToString();
    }
}
private void button2_Click(object sender, EventArgs e)
{
    int gverdi_1 = Convert.ToInt32(textBox1.Text);
    int gverdi_2 = Convert.ToInt32(textBox2.Text);
    Martkutxedi obieqti_1 = new Martkutxedi(gverdi_1, gverdi_2);
    obieqti_1.Gamotana(label1);
}

```

this სავსანძო სოცყვა

5.4.1.

```
class Chemi_Klasi
{
    int ricxvi1, ricxvi2, ricxvi3;
    public int Gamotvla(int ricxvi1, int ricxvi2, int ricxvi3)
    {
        this.ricxvi1 = ricxvi1;
        this.ricxvi2 = ricxvi2;
        this.ricxvi3 = ricxvi3;
        return this.ricxvi1 + this.ricxvi2 + this.ricxvi3;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    Chemi_Klasi obieqti_1 = new Chemi_Klasi();
    int cvladi1 = Convert.ToInt32(textBox1.Text);
    int cvladi2 = Convert.ToInt32(textBox2.Text);
    int cvladi3 = Convert.ToInt32(textBox3.Text);

    int shedegi = obieqti_1.Gamotvla(cvladi1, cvladi2, cvladi3);
    label1.Text = shedegi.ToString();
}
```

5.4.3.

```
class Chemi_Klasi1
{
    int[] masivi2;
    public int Gamotvla(int[] masivi2)
    {
        int ind;
        this.masivi2 = masivi2;
        for (ind = 0 ; ind < masivi2.Length ; ind++)
            if (masivi2[ind] < 0) break;
        return this.masivi2[ind];
    }
}
private void button2_Click(object sender, EventArgs e)
{
    Chemi_Klasi1 obieqti_1 = new Chemi_Klasi1();
    int[] masivi1 = new int[] { 5, 2, -3, 6, -1, 9, -8 };

    int shedegi = obieqti_1.Gamotvla(masivi1);
    label1.Text = shedegi.ToString();
}
```

შემთხვევითი რიცხვები

5.5.4.

```
{
    System.Random obieqti = new System.Random();
    int min_rixcvi = Convert.ToInt32(textBox1.Text), // 1
        max_rixcvi = Convert.ToInt32(textBox2.Text), // 30
        ramdenjer = Convert.ToInt32(textBox3.Text), // 7
        mocemuli_rixcvi = Convert.ToInt32(textBox4.Text), // 10
        raodenoba = 0, rixcvi,
        rixvebis_raodenoba = 0; // გენერირებული რიცხვების რაოდენობა

    for ( ; ; )
    {
        rixcvi = obieqti.Next(min_rixcvi, max_rixcvi);
        rixvebis_raodenoba++;
        if ( rixcvi == mocemuli_rixcvi ) raodenoba++;
        if ( raodenoba == ramdenjer ) break;
    }
    label1.Text = rixvebis_raodenoba.ToString();
}
```

5.5.5.

```
{
    label1.Text = "";
    int[] masivi = new int[] { 20, 20, 20, 20, 20 };
    int shemtxveviti_rixcvi, asantebis_raodenoba = 0;
    System.Random obieqti = new System.Random();

    for ( ; ; )
    {
        shemtxveviti_rixcvi = obieqti.Next(5);
        masivi[shemtxveviti_rixcvi]--;
        asantebis_raodenoba++;
        if ( masivi[shemtxveviti_rixcvi] == 0 ) break;
    }
    label1.Text = "უნდა ამოვიღოთ " + asantebis_raodenoba.ToString() +
        " ასანთი იმისათვის, \nორმ ერთ-ერთი კოლოფი დაცარიელდეს";
}
```

თავი 6. პოლიმორფიზმი

მეთოდისათვის ობიექტის გადაცემა

6.1.1.

```
class Samkutxedi
{
    int gverdi1, gverdi2, gverdi3;
    public Samkutxedi(int par1, int par2, int par3)
```



```

    {
        gverdi1 = par1;
        gverdi2 = par2;
        gverdi3 = par3;
    }
    public double Gamotvla(Samkutxedi Sam1)
    {
        return ( Sam1.gverdi1 * Sam1.gverdi2 ) / 2;
    }
}

```

```

private void button1_Click(object sender, EventArgs e)
{
    int arg1 = Convert.ToInt32(textBox1.Text);
    int arg2 = Convert.ToInt32(textBox2.Text);
    int arg3 = Convert.ToInt32(textBox3.Text);

    Samkutxedi obieqti1 = new Samkutxedi(arg1, arg2, arg3);
    Samkutxedi obieqti2 = new Samkutxedi(arg1 + 10, arg2 + 20, arg3 + 30);
    double shedegi1 = obieqti1.Gamotvla(obieqti1);
    double shedegi2 = obieqti1.Gamotvla(obieqti2);
    label1.Text = shedegi1.ToString();
    label2.Text = shedegi2.ToString();
}

```

არგუმენტების გადაცემის ხერხები. ref, out და params მოდიფიკატორები

6.2.1.

```

class Kvadrati
{
    // მეთოდი ადგილებს უცვლის ორ პარამეტრს
    public void Miniweba(ref int par1, ref int par2)
    {
        par2 = par1 * par1;
    }
}

```

```

private void button1_Click(object sender, EventArgs e)
{
    label1.Text = ""; label2.Text = "";
    Kvadrati obieqti = new Kvadrati();
    int ricxvi1 = Convert.ToInt32(textBox1.Text);
    int ricxvi2 = Convert.ToInt32(textBox2.Text);

    // არგუმენტების მნიშვნელობები გაცვლამდე
    label1.Text = ricxvi1.ToString() + " " + ricxvi2.ToString();
    obieqti.Miniweba(ref ricxvi1, ref ricxvi2);
    // არგუმენტების მნიშვნელობები გაცვლის შემდეგ
}

```

```
label2.Text = ricxvi1.ToString() + " " + ricxvi2.ToString();
```

```
}
```

6.2.4.

```
class Uaryofiti_Dadebiti
```

```
{
```

```
public int Raod_Jami(int[] masivi2, out int raod_dadebiti, out int raod_uaryofiti, out int luwi_jami)
```

```
{
```

```
int kenti_jami = 0;
```

```
luwi_jami = raod_dadebiti = raod_uaryofiti = 0;
```

```
for ( int ind = 0; ind < masivi2.Length; ind++ )
```

```
{
```

```
if ( masivi2[ind] < 0 ) raod_uaryofiti++;
```

```
if ( masivi2[ind] > 0 ) raod_dadebiti++;
```

```
if ( masivi2[ind] % 2 == 0 ) luwi_jami += masivi2[ind];
```

```
if ( masivi2[ind] % 2 == 1 ) kenti_jami += masivi2[ind];
```

```
}
```

```
return kenti_jami;
```

```
}
```

```
}
```

```
private void button2_Click(object sender, EventArgs e)
```

```
{
```

```
int[] masivi = new int[] { 5, -3, 2, 9, -4, -8 };
```

```
Uaryofiti_Dadebiti obieqti = new Uaryofiti_Dadebiti();
```

```
int raod_uaryofiti, raod_dadebiti;
```

```
int luwi_jami, kenti_jami;
```

```
kenti_jami = obieqti.Raod_Jami(masivi, out raod_uaryofiti, out raod_dadebiti, out luwi_jami);
```

```
label1.Text = "უარყოფითი რიცხვების რაოდენობა = " + raod_uaryofiti.ToString() +
```

```
"\nდადებითი რიცხვების რაოდენობა = " + raod_dadebiti.ToString() +
```

```
"\nიკენტი რიცხვების ჯამი = " + kenti_jami.ToString() +
```

```
"\nილუწი რიცხვების ჯამი = " + luwi_jami.ToString();
```

```
}
```

6.2.5.

```
class Chemi_Klasi
```

```
{
```

```
public int Dadebitebis_Raodenoba(params int[] masivi)
```

```
{
```

```
int raodenoba = 0;
```

```
if ( masivi.Length == 0 )
```

```
{
```

```
// რადგან არგუმენტი არ არის მითითებული, ამიტომ მეთოდი აბრუნებს ნულს
```

```
return 0;
```

```
}
```

```
for ( int indexi = 0; indexi < masivi.Length; indexi++ )
```

```
if ( masivi[indexi] >= 0 ) raodenoba++;
```

```
return raodenoba;
```

```
}
```

```

    }
private void button4_Click(object sender, EventArgs e)
{
    Chemi_Klasi obieqti = new Chemi_Klasi();
    int i1 = 15, i2 = -25;
    int i3 = Convert.ToInt32(textBox1.Text);
    int[] masivi = new int[] { 11, 22, -33, -44, 5 };
    int raodenoba;
    // მეთოდისთვის ორი არგუმენტის გადაცემა
    raodenoba = obieqti.Dadebitebis_Raodenoba(i1, i2);
    label1.Text = raodenoba.ToString();
    // მეთოდისთვის ოთხი არგუმენტის გადაცემა
    raodenoba = obieqti.Dadebitebis_Raodenoba(i1, i2, -30, i3);
    label2.Text = raodenoba.ToString();
    // მეთოდისთვის მასივის გადაცემა
    raodenoba = obieqti.Dadebitebis_Raodenoba(masivi);
    label4.Text = raodenoba.ToString();
}

```

ობიექტის დაბრუნება

6.3.2.

```

class ChemiKlasi_1
{
    public int max, min;
    public ChemiKlasi_1(int par1, int par2)
    {
        min = par1;
        max = par2;
    }
}
class ChemiKlasi_2
{
    public ChemiKlasi_1 MaxMin(int[] masivi2)
    {
        int min = masivi2[0];
        int max = masivi2[0];
        for ( int ind = 0; ind < masivi2.Length; ind++ )
        {
            if ( max < masivi2[ind] ) max = masivi2[ind];
            if ( min > masivi2[ind] ) min = masivi2[ind];
        }
        return new ChemiKlasi_1(min, max);
    }
}
private void button2_Click(object sender, EventArgs e)
{
    int[] masivi1 = new int[] { 3, 6, 2, 9, 6, 8 };

```

```

ChemiKlasi_2 obieqti1 = new ChemiKlasi_2();
ChemiKlasi_1 obieqti2;

obieqti2 = obieqti1.MaxMin(masivi1);
label1.Text = "მინიმალური ელემენტი = " + obieqti2.min.ToString() +
            "\nმაქსიმალური ელემენტი = " + obieqti2.max.ToString();
}

```

მეთოდების გადატვირთვა. პოლიმორფიზმი

6.4.2.

```

class Figura
{
public int Perimetri(int par1)
{
return 4 * par1;
}
public int Perimetri(int par1, int par2)
{
return 2 * (par1 + par2);
}
public int Perimetri(int par1, int par2, int par3)
{
return par1 + par2 + par3;
}
}
private void button2_Click(object sender, EventArgs e)
{
Figura obieqti_1 = new Figura();
int gverdi_1 = Convert.ToInt32(textBox1.Text);
int gverdi_2 = Convert.ToInt32(textBox2.Text);
int gverdi_3 = Convert.ToInt32(textBox3.Text);

int kvadratis_perimetri = obieqti_1.Perimetri(gverdi_1);
int otxktxedis_perimetri = obieqti_1.Perimetri(gverdi_1, gverdi_2);
int samktxedis_perimetri = obieqti_1.Perimetri(gverdi_1, gverdi_2, gverdi_3);

label1.Text = "კვადრატის პერიმეტრი = " + kvadratis_perimetri.ToString();
label1.Text += "\nოთხკუთხედის პერიმეტრი = " + otxktxedis_perimetri.ToString();
label1.Text += "\nსამკუთხედის პერიმეტრი = " + samktxedis_perimetri.ToString();
}

```

6.4.3.

```

class Avtomobili
{
public int Metodi_1(int par1, int par2)
{
return par1 * par2;
}
}

```

```

public double Metodi_1(double par1, double par2)
{
return par1 * par2;
}
}
private void button1_Click(object sender, EventArgs e)
{
Avtomobili obieqti_1 = new Avtomobili();
int bakis_tevadoba = Convert.ToInt32(textBox1.Text);
int manzili_1_litri = Convert.ToInt32(textBox2.Text);
double maqsimaluri_sichqare = Convert.ToDouble(textBox3.Text);
double dro = Convert.ToDouble(textBox4.Text);

int manzili_savse_baki = obieqti_1.Metodi_1(bakis_tevadoba, manzili_1_litri);
double manZili_dro = obieqti_1.Metodi_1(dro, maqsimaluri_sichqare);

label1.Text = "სავსე ბაკით გავლილი მანძილი = " + manzili_savse_baki.ToString();
label2.Text = "მაქსიმალური სიჩქარით მოძრაობისას გავლილი მანძილი = " +
manZili_dro.ToString();
}

```

კონსტრუქტორების გადატვირთვა

6.5.1.

```

class ChemiKlasi
{
int Min;
public ChemiKlasi(int[] masivi1)
{
Min = masivi1[0];
for ( int ind = 1; ind < masivi1.Length; ind++ )
if ( masivi1[ind] < Min ) Min = masivi1[ind];
}
public ChemiKlasi(ChemiKlasi obj)
{
Min = obj.Min;
}
public void Naxva(Label lab)
{
lab.Text = Min.ToString();
}
}
private void button1_Click(object sender, EventArgs e)
{
int[] masivi2 = new int[] { 6, 2, 9, 4, -7, 1 };
ChemiKlasi obieqti1 = new ChemiKlasi(masivi2);
ChemiKlasi obieqti2 = new ChemiKlasi(obieqti1);
}

```

```
        obieqti2.Naxva(label1);
```

```
    }
```

6.5.2

```
class Chemi_Klasi
```

```
{
```

```
    int gverdi1;
```

```
    int gverdi2;
```

```
    int gverdi3;
```

```
    int perimetri;
```

```
    double fartobi;
```

```
    public Chemi_Klasi(int par1)
```

```
    {
```

```
        gverdi1 = par1;
```

```
        perimetri = 4 * par1;
```

```
        fartobi = Math.Pow(gverdi1, 2);
```

```
    }
```

```
    public Chemi_Klasi(int par1, int par2)
```

```
    {
```

```
        gverdi1 = par1;
```

```
        gverdi2 = par2;
```

```
        perimetri = 2 * ( par1 + par2 );
```

```
        fartobi = par1 * par2;
```

```
    }
```

```
    public Chemi_Klasi(int par1, int par2, int par3)
```

```
    {
```

```
        gverdi1 = par1;
```

```
        gverdi2 = par2;
```

```
        gverdi3 = par3;
```

```
        perimetri = par1 + par2 + par3;
```

```
        fartobi = ( par1 * par2 ) / 2;
```

```
    }
```

```
    public void Naxva(Label lab)
```

```
    {
```

```
        lab.Text = "ფიგურის პერიმეტრი = " + perimetri.ToString() +
```

```
        "\nფიგურის ფართობი = " + fartobi.ToString();
```

```
    }
```

```
}
```

```
private void button2_Click(object sender, EventArgs e)
```

```
{
```

```
    int gverdi1 = Convert.ToInt32(textBox1.Text);
```

```
    int gverdi2 = Convert.ToInt32(textBox2.Text);
```

```
    int gverdi3 = Convert.ToInt32(textBox3.Text);
```

```
    Chemi_Klasi obieqti1 = new Chemi_Klasi(gverdi1);
```

```
    Chemi_Klasi obieqti2 = new Chemi_Klasi(gverdi1, gverdi2);
```

```
    Chemi_Klasi obieqti3 = new Chemi_Klasi(gverdi1, gverdi2, gverdi3);
```

```
    obieqti1.Naxva(label1);
```

```
    obieqti2.Naxva(label2);
```

```

    obieqti3.Naxva(label3);
}

```

გადატვირთვადი კონსტრუქტორის გამოძახება this სიტყვის გამოყენებით

6.6.2.

```

class Otxkutxedi
{
    public int gverdi1, gverdi2;
    int fartobi;
    public Otxkutxedi(int par1, int par2)
    {
        gverdi1 = par1;
        gverdi2 = par2;
    }
    public Otxkutxedi() : this (20, 50)
    {
        fartobi = gverdi1 * gverdi2;
    }
    public Otxkutxedi(Otxkutxedi obj) : this (obj.gverdi1, obj.gverdi2)
    {
        fartobi = gverdi1 * gverdi2;
    }
    public void Naxva(Label lab)
    {
        lab.Text = "მართკუთხედის პირველი გვერდი = " + gverdi1.ToString() +
            "\nმართკუთხედის მეორე გვერდი = " + gverdi2.ToString() +
            "\nმართკუთხედის ფართობი = " + fartobi.ToString();
    }
}
private void button1_Click(object sender, EventArgs e)
{
    Otxkutxedi obieqti2 = new Otxkutxedi();
    obieqti2.Naxva(label2);
    obieqti2.gverdi1 = Convert.ToInt32(textBox1.Text);
    obieqti2.gverdi2 = Convert.ToInt32(textBox2.Text);
    Otxkutxedi obieqti3 = new Otxkutxedi(objiect2);
    obieqti3.Naxva(label3);
}

```

static მოდიფიკატორი

6.7.1.

```

class ChemiKlasi
{
    int ricxvi1; // ჩვეულებრივი ცვლადის გამოცხადება
    static int ricxvi2; // სტატიკური ცვლადის გამოცხადება
    intimetri;
    public ChemiKlasi(int par1, int par2)

```

```

{
    ricxvi1 = par1;
    ricxvi2 = par2;
}
//      ჩვეულებრივი მეთოდი
int ChveulebriviMetodi()
{
    perimetri = ricxvi1 + ricxvi2;
    return perimetri;
}
//      სტატიკური მეთოდი
public static int StatikuriMetodi(ChemiKlasi obieqti)
{
    int shedegi = obieqti.ChveulebriviMetodi();
    return shedegi;
}
}
private void button1_Click(object sender, EventArgs e)
{
    int gverdi1 = Convert.ToInt32(textBox1.Text);
    int gverdi2 = Convert.ToInt32(textBox2.Text);
    ChemiKlasi obj = new ChemiKlasi(gverdi1, gverdi2);

    int shedegi = ChemiKlasi.StatikuriMetodi(obj);
    label1.Text = shedegi.ToString();
}

```

6.7.2

class ChemiKlasi_2

```

{
    public static int[] masivi_1;
    public ChemiKlasi_2(int[] masivi_2)
    {
        masivi_1 = masivi_2;
    }
    int Kenti_Jami()
    {
        int jami = 0;
        for ( int ind = 0 ; ind < masivi_1.Length ; ind++ )
            if ( masivi_1[ind] % 2 == 1 ) jami += masivi_1[ind];
        return jami;
    }
    public static int StatikuriMetodi(ChemiKlasi_2 obieqti)
    {
        return obieqti.Kenti_Jami();
    }
}
private void button2_Click(object sender, EventArgs e)

```



```

{
    int[] masivi = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
    ChemiKlasi_2 obj1 = new ChemiKlasi_2(masivi);
    int shedegi = ChemiKlasi_2.StatikuriMetodi(obj1);
    label1.Text = shedegi.ToString();
}

```

თავი 7. მემკვიდრეობითობა

კლასები. მემკვიდრეობითობა. protected მოდიფიკატორი

7.1.2.

```

class Martkutxedi
{
    protected int fudze;
}
class Memkvidre : Martkutxedi
{
    int simagle;
    public Memkvidre()
    {
        simagle = 0;
    }
    public Memkvidre(int par1, int par2)
    {
        simagle = par2;
        fudze = par1;
    }
    public int Fartobi()
    {
        return fudze * simagle;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    int ricxvi1 = Convert.ToInt32(textBox1.Text);
    int ricxvi2 = Convert.ToInt32(textBox2.Text);
    Memkvidre obj2 = new Memkvidre(ricxvi1, ricxvi2);
    int shedegi = obj2.Fartobi();
    label1.Text = shedegi.ToString();
}

```

7.1.3.

```

class Martkutxedi_1
{
    protected int fudze;
    public Martkutxedi_1()
    {
        fudze = 0;
    }
}

```

```

    }
    public Martkutxedi_1(int par1)
    {
        fudze = par1;
    }
    public void Naxva(Label lab1)
    {
        lab1.Text = fudze.ToString();
    }
}
class Memkvidre_1 : Martkutxedi_1
{
    int simagle;
    public Memkvidre_1()
    {
        simagle = 0;
    }
    public Memkvidre_1(int par1, int par2)
    {
        simagle = par2;
        fudze = par1;
    }
    public int Fartobi()
    {
        return fudze * simagle;
    }
}
private void button2_Click(object sender, EventArgs e)
{
    int ricxvi1 = Convert.ToInt32(textBox1.Text);
    int ricxvi2 = Convert.ToInt32(textBox2.Text);
    Martkutxedi_1 obj1 = new Martkutxedi_1(ricxvi1);
    obj1.Naxva(label2);
    Memkvidre_1 obj2 = new Memkvidre_1(obj1.fudze, ricxvi2);
    int shedegi = obj2.Fartobi();
    label1.Text = shedegi.ToString();
}

```

კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა

7.2.3.

```

class Televizori
{
    public int namushevari_saatebis_raodenoba;
    public int wati_erti_saati;
    public Televizori(int par1)
    {
        namushevari_saatebis_raodenoba = par1;
    }
}

```

```

    }
    public Televizori(int par1, int par2)
    {
        wati_erti_saati = par1;
        namushevari_saatebis_raodenoba = par2;
    }
}
class Memkvidre_3 : Televizori
{
    public int saatebis_raodenoba;
    public Memkvidre_3(int par5) : base(par5)
    {
        saatebis_raodenoba = namushevari_saatebis_raodenoba;
    }
    public void Naxva_3(Label lab3)
    {
        lab3.Text = saatebis_raodenoba.ToString();
    }
}
class Memkvidre_4 : Televizori
{
    public int watebis_raodenoba;
    public Memkvidre_4(int par3, int par4) : base(par3, par4)
    {
        watebis_raodenoba = wati_erti_saati * namushevari_saatebis_raodenoba;
    }
    public void Naxva_4(Label lab4)
    {
        lab4.Text = watebis_raodenoba.ToString();
    }
}
private void button1_Click(object sender, EventArgs e)
{
    int saati = Convert.ToInt32(textBox1.Text);
    int wati = Convert.ToInt32(textBox2.Text);
    Memkvidre_3 obj_3 = new Memkvidre_3(saati);
    Memkvidre_4 obj_4 = new Memkvidre_4(saati, wati);
    obj_3.Naxva_3(label1);
    obj_4.Naxva_4(label2);
}

```

ცვლადების დამალვა მემკვიდრეობითობის დროს

7.3.1.

```

class Samkutxedi_Sabazo
{
    public int gverdi1;
    public int gverdi2;
}

```

```

    public int gverdi3;
}
class Samkutxedi_Memkvidre : Samkutxedi_Sabazo
{
    new public int gverdi1;
    new public int gverdi2;
    new public int gverdi3;
    public int Perimetri_Sabazo(int par1, int par2, int par3)
    {
        base.gverdi1 = par1 + 10;
        base.gverdi2 = par2 + 20;
        base.gverdi3 = par3 + 30;
        return base.gverdi1 + base.gverdi2 + base.gverdi3;
    }
    public int Perimetri_Memkvidre(int par1, int par2, int par3)
    {
        gverdi1 = par1;
        gverdi2 = par2;
        gverdi3 = par3;
        return gverdi1 + gverdi2 + gverdi3;
    }
}
private void button2_Click(object sender, EventArgs e)
{
    int gverdi1 = Convert.ToInt32(textBox1.Text);
    int gverdi2 = Convert.ToInt32(textBox2.Text);
    int gverdi3 = Convert.ToInt32(textBox3.Text);
    Samkutxedi_Memkvidre obj_Memkvidre = new Samkutxedi_Memkvidre();
    int perimetri_sabazo = obj_Memkvidre.Perimetri_Memkvidre(gverdi1, gverdi2, gverdi3);
    int perimetri_memkvidre = obj_Memkvidre.Perimetri_Sabazo(gverdi1, gverdi2, gverdi3);
    label1.Text = perimetri_sabazo.ToString();
    label2.Text = perimetri_memkvidre.ToString();
}

```

7.3.3.

```

class SabazoKlasi
{
    public int[,] masivi1 = new int[4, 4]
        { { -10, -20, 30, 40 }, { 50, 60, -70, -80 }, { -90, 100, -110, 120 }, { 130, -140, 150, -160 } };
}
class Memkvidre : SabazoKlasi
{
    new public int[,] masivi1 = new int[4, 4]
        { { -1, -2, 3, 4 }, { 5, 6, -7, -8 }, { -9, 10, -11, 12 }, { 13, -14, 15, -16 } };
    public int Min_Metodi()
    {
        int min = 0;
        for ( int striqoni = 0; striqoni < 4; striqoni++ )

```

```

        for ( int sveti = 0; sveti < 4; sveti++ )
            if ( masivil[striqoni, sveti] > 0 )
            {
                min = masivil[striqoni, sveti];
                goto M1;
            }
M1:
for ( int striqoni = 0; striqoni < 4; striqoni++ )
    for ( int sveti = 0; sveti < 4; sveti++ )
        if ( ( min > masivil[striqoni, sveti] ) && ( masivil[striqoni, sveti] > 0 ) )
            min = masivil[striqoni, sveti];
return min;
}
public int Max_Metodi()
{
    int max = 0;
    for ( int striqoni = 0; striqoni < 4; striqoni++ )
        for ( int sveti = 0; sveti < 4; sveti++ )
            if ( masivil[striqoni, sveti] > 0 )
            {
                max = masivil[striqoni, sveti];
                goto M1;
            }
M1:
    for ( int striqoni = 0; striqoni < 4; striqoni++ )
        for ( int sveti = 0; sveti < 4; sveti++ )
            if ( ( max < base.masivil[striqoni, sveti] ) && ( base.masivil[striqoni, sveti] > 0 ) )
                max = base.masivil[striqoni, sveti];
    return max;
}
}
private void button1_Click(object sender, EventArgs e)
{
    Memkvidre obj_Memkvidre = new Memkvidre();
    int shedegi_max = obj_Memkvidre.Max_Metodi();
    int shedegi_min = obj_Memkvidre.Min_Metodi();
    label1.Text = shedegi_max.ToString();
    label2.Text = shedegi_min.ToString();
}

```

ვირტუალური მეთოდები

7.4.1.

```

class Figura
{
    public int gverdi1, gverdi2, gverdi3, gverdi4;
    public Figura()
    {}
}

```

```

public Figura(int par1, int par2, int par3, int par4)
{
    gverdi1 = par1;
    gverdi2 = par2;
    gverdi3 = par3;
    gverdi4 = par4;
}
public virtual int Perimetri()
{
    return gverdi1 + gverdi2 + gverdi3 + gverdi4;
}
}
class Otxkutxedi : Figura
{
    public override int Perimetri()
    {
        return ( gverdi1 + gverdi2 ) * 2;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    int gverdi1 = Convert.ToInt32(textBox1.Text);
    int gverdi2 = Convert.ToInt32(textBox2.Text);
    int gverdi3 = Convert.ToInt32(textBox3.Text);
    int gverdi4 = Convert.ToInt32(textBox4.Text);
    Otxkutxedi obj_memkvidre = new Otxkutxedi();
    obj_memkvidre.gverdi1 = gverdi1;
    obj_memkvidre.gverdi2 = gverdi2;
    int shedegi = obj_memkvidre.Perimetri();
    label1.Text = shedegi.ToString();
    Figura obj_WInapari = new Figura(gverdi1, gverdi2, gverdi3, gverdi4);
    int shedegi1 = obj_WInapari.Perimetri();
    label2.Text = shedegi1.ToString();
}

```

7.4.4.

```

class Sabazo_Klasi
{
    public virtual int Metodi1(int[,] masivi1)
    {
        int min = 0;
        for ( int striqoni = 0 ; striqoni < 4 ; striqoni++ )
            for ( int sveti = 0 ; sveti < 4 ; sveti++ )
                if ( masivi1[striqoni, sveti] < 0 )
                {
                    min = masivi1[striqoni, sveti];
                    goto M1;
                }
    }
}

```

```

M1:
for ( int striqoni = 0; striqoni < 4; striqoni++ )
    for ( int sveti = 0; sveti < 4; sveti++ )
        if ( ( masivil[striqoni, sveti] < min ) && ( masivil[striqoni, sveti] < 0 ) )
            min = masivil[striqoni, sveti];
return min;
}
}
class Memkvidre_Klasi : Sabazo_Klasi
{
public override int Metodil(int[,] masivi1)
{
    int max = 0;
    for ( int striqoni = 0 ; striqoni < 4 ; striqoni++ )
        for ( int sveti = 0 ; sveti < 4 ; sveti++ )
            if ( masivi1[striqoni, sveti] < 0 )
                {
                    max = masivi1[striqoni, sveti];
                    goto M1;
                }
}
M1:
for ( int striqoni = 0; striqoni < 4; striqoni++ )
    for ( int sveti = 0; sveti < 4; sveti++ )
        if ( ( masivil[striqoni, sveti] > max ) && ( masivil[striqoni, sveti] < 0 ) )
            max = masivil[striqoni, sveti];
return max;
}
}
private void button2_Click(object sender, EventArgs e)
{
    int[,] masivi = new int[4, 4] { { 1, -2, -3, 4 }, { -5, 6, 7, -8 }, { 9, 10, -11, -12 }, { -13, -14, 15, 16 } };
    Sabazo_Klasi obj_Sabazo = new Sabazo_Klasi();
    Memkvidre_Klasi obj_Memkvidre = new Memkvidre_Klasi();
    int min = obj_Sabazo.Metodil(masivi);
    int max = obj_Memkvidre.Metodil(masivi);
    label1.Text = min.ToString();
    label2.Text = max.ToString();
}

```

აბსტრაქტული კლასები და მეთოდები

7.5.3.

```

abstract class Televizori
{
    public abstract int Metodil();
}
class Memkvidre_1 : Televizori
{

```

```

int Wati_Erti_Saati;
int Namushevari_Saatebi;
public Memkvidre_1(int par1, int par2)
{
    Wati_Erti_Saati = par1;
    Namushevari_Saatebi = par2;
}
public override int Metodi()
{
    return Wati_Erti_Saati * Namushevari_Saatebi;
}
}
private void button1_Click(object sender, EventArgs e)
{
    int wati = Convert.ToInt32(textBox1.Text);
    int saati = Convert.ToInt32(textBox2.Text);
    Memkvidre_1 obj_Memkvidre = new Memkvidre_1(wati, saati);
    int shedegi = obj_Memkvidre.Metodi();
    label1.Text = shedegi.ToString();
}

```

7.5.4.

```

abstract class Tanamshromeli
{
    public abstract int Metodi();
}
class Memkvidre_2 : Tanamshromeli
{
    int xelfasi_erti_tvis;
    public Memkvidre_2(int par1)
    {
        xelfasi_erti_tvis = par1;
    }
    public override int Metodi()
    {
        return xelfasi_erti_tvis * 12;
    }
}
private void button2_Click(object sender, EventArgs e)
{
    int erti_tvis_xelfasi = Convert.ToInt32(textBox1.Text);
    Memkvidre_2 obj_memkvidre = new Memkvidre_2(erti_tvis_xelfasi);
    int shedegi = obj_memkvidre.Metodi();
    label1.Text = shedegi.ToString();
}

```


თავი 8. თვისებები, ინდექსატორები და ოპერატორების გადატვირთვა

თვისებები

8.1.3.

```
class ChemiKlasi
{
    private int cvladi;
    //    თვისების გამოცხადება
    public int ChemiTviseba
    {
        get
        {
            return cvladi;
        }
        set
        {
            // cvladi ცვლადს ენიჭება value მნიშვნელობა თუ ის 5-ის ჯერადია
            if (value % 5 == 0) cvladi = value;
        }
    }
}
private void button1_Click(object sender, EventArgs e)
{
    ChemiKlasi obieqti = new ChemiKlasi();
    obieqti.ChemiTviseba = Convert.ToInt32(textBox1.Text);
    label1.Text = obieqti.ChemiTviseba.ToString();
}
```

ინდექსატორები. ერთგანზომილებიანი ინდექსატორები

8.2.1.

```
class Chemi_Klasi
{
    int ricxvi1;
    int ricxvi2;
    int ricxvi3;
    int ricxvi4;
    int ricxvi5;
    public int this[int index]
    {
        get
        {
            switch (index)
            {
                case 0: return ricxvi1;
                case 1: return ricxvi2;
                case 2: return ricxvi3;
                case 3: return ricxvi4;
            }
        }
    }
}
```

```

        case 4: return ricxvi5;
        default: return 0;
    }
}
set
{
    switch (index)
    {
        case 0: this.ricxvi1 = value; break;
        case 1: this.ricxvi2 = value; break;
        case 2: this.ricxvi3 = value; break;
        case 3: this.ricxvi4 = value; break;
        case 4: this.ricxvi5 = value; break;
    }
}
}
}
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    Chemi_Klasi obieqti = new Chemi_Klasi();
    for ( int ind = 0; ind <= 4; ind++)
        obieqti[ind] = ind * 2 + 10;
    for (int ind = 0 ; ind <= 4 ; ind++)
        label1.Text += obieqti[ind].ToString() + " ";
}

```

8.2.7.

```

class ChemiKlasi
{
    int ricxvi1 = -1;
    int ricxvi2 = -1;
    int ricxvi3 = -1;
    int ricxvi4 = -1;
    int ricxvi5 = -1;
    public int this[int index]
    {
        get
        {
            switch ( index )
            {
                case 0: return ricxvi1;
                case 1: return ricxvi2;
                case 2: return ricxvi3;
                case 3: return ricxvi4;
                case 4: return ricxvi5;
                default: return 0;
            }
        }
    }
}

```

```

    }
    set
    {
        switch ( index )
        {
            case 0: if (value % 2 == 0) { this.ricxvi1 = value; break; } else break;
            case 1: if (value % 2 == 0) { this.ricxvi2 = value; break; } else break;
            case 2: if (value % 2 == 0) { this.ricxvi3 = value; break; } else break;
            case 3: if (value % 2 == 0) { this.ricxvi4 = value; break; } else break;
            case 4: if (value % 2 == 0) { this.ricxvi5 = value; break; } else break;
        }
    }
}
private void button2_Click(object sender, EventArgs e)
{
    label2.Text = "";
    ChemiKlasi obieqti = new ChemiKlasi();
    for (int ind = 0 ; ind <= 4 ; ind++)
    {
        obieqti[ind] = ind;
        label2.Text += obieqti[ind].ToString() + " ";
    }
}

```

თავი 9. დელეგატები, მოვლენები, ინტერფეისები და სახელების სივრცე

დელეგატები. დელეგატების მრავალმისამართიანობა

9.1.1.

```

delegate void ChemiDelegati(ref int ricxvi);
class ChemiKlasi
{
    public void Metodi1(ref int par)
    {
        par = par + 10;
    }
    public void Metodi2(ref int par)
    {
        par = par + 20;
    }
    public void Metodi3(ref int par)
    {
        par = par + par;
    }
}
private void button1_Click(object sender, EventArgs e)

```

```

{
    ChemiDelegati delegati_obieqti;
    ChemiKlasi obieqti = new ChemiKlasi();
    ChemiDelegati Metodi1_Mimartva = new ChemiDelegati(obieqti.Metodi1);
    ChemiDelegati Metodi2_Mimartva = new ChemiDelegati(obieqti.Metodi2);
    ChemiDelegati Metodi3_Mimartva = new ChemiDelegati(obieqti.Metodi3);
    int ricxvi1 = Convert.ToInt32(textBox1.Text);
    //
    delegati_obieqti = Metodi1_Mimartva;
    delegati_obieqti += Metodi2_Mimartva;
    //
    delegati_obieqti(ref ricxvi1);
    label1.Text = ricxvi1.ToString();
    //
    delegati_obieqti -= Metodi1_Mimartva;
    delegati_obieqti += Metodi3_Mimartva;
    //
    delegati_obieqti(ref ricxvi1);
    label2.Text = ricxvi1.ToString();
}

```

9.1.2.

```

delegate void ChemiDelegati2(ref int ricxvi1, ref int ricxvi2);

```

```

class ChemiKlasi2
{
    public void Metodi1(ref int par1, ref int par2)
    {
        par1 = par2 + 10;
    }
    public void Metodi2(ref int par1, ref int par2)
    {
        par1 = par2 + 20;
    }
    public void Metodi3(ref int par1, ref int par2)
    {
        par1 = par2 + par2;
    }
}
private void button2_Click(object sender, EventArgs e)
{
    ChemiDelegati2 delegati_obieqti;
    ChemiKlasi2 obieqti = new ChemiKlasi2();
    ChemiDelegati2 Metodi1_Mimartva = new ChemiDelegati2(obieqti.Metodi1);
    ChemiDelegati2 Metodi2_Mimartva = new ChemiDelegati2(obieqti.Metodi2);
    ChemiDelegati2 Metodi3_Mimartva = new ChemiDelegati2(obieqti.Metodi3);
    int ricxvi1 = Convert.ToInt32(textBox1.Text);
    int ricxvi2 = Convert.ToInt32(textBox2.Text);
    //

```

```

delegati_obieqti = Metodi1_Mimartva;
delegati_obieqti += Metodi2_Mimartva;
//
delegati_obieqti(ref ricxvi1, ref ricxvi2);
label1.Text = ricxvi1.ToString();
//
delegati_obieqti -= Metodi2_Mimartva;
delegati_obieqti += Metodi3_Mimartva;
//
delegati_obieqti(ref ricxvi1, ref ricxvi2);
label2.Text = ricxvi1.ToString();
}

```

მოვლენები

9.2.1.

```

delegate void ChemiDelegati();
class Klasi1
{
    public event ChemiDelegati ChemiMovlena;
    public void metodi(int par1, int par2)
    {
        if ( par1 > par2 ) ChemiMovlena();
    }
    public void MovlenisDamamushavebeli()
    {
        MessageBox.Show("პირველი რიცხვი მეორეზე მეტია");
    }
}
private void button1_Click(object sender, EventArgs e)
{
    Klasi1 obieqti_movlena = new Klasi1();
    //
    obieqti_movlena.ChemiMovlena += new
        ChemiDelegati(obieqti_movlena.MovlenisDamamushavebeli);
    int ricxvi1 = Convert.ToInt32(textBox1.Text);
    int ricxvi2 = Convert.ToInt32(textBox2.Text);
    //
    obieqti_movlena.metodi(ricxvi1, ricxvi2);
}

```

9.2.2.

```

delegate void ChemiDelegati2();
class Klasi2
{
    public event ChemiDelegati2 ChemiMovlena;
    public void metodi(int par1, int par2)

```

```

    {
        if ( ( par1 % 2 == 0 ) && ( par2 % 2 == 0 ) ) ChemiMovlena();
    }
    public void MovlenisDamamushavebeli()
    {
        MessageBox.Show("ორივე რიცხვი ლუწია");
    }
}
private void button2_Click(object sender, EventArgs e)
{
    Klasi2 obieqti_movlena = new Klasi2();
    //
    obieqti_movlena.ChemiMovlena += new
        ChemiDelegati2(obieqti_movlena.MovlenisDamamushavebeli);
    int ricxvi1 = Convert.ToInt32(textBox1.Text);
    int ricxvi2 = Convert.ToInt32(textBox2.Text);
    //
    obieqti_movlena.metodi(ricxvi1, ricxvi2);
}

```

9.2.3.

```

delegate void ChemiDelegati3();
class Klasi3
{
    public event ChemiDelegati3 ChemiMovlena;
    public void metodi(string par1, string par2)
    {
        for ( int index = 0; index < par1.Length; index++ )
            if ( par1[index] != par2[index] )
            {
                ChemiMovlena();
                return;
            }
    }
    public void MovlenisDamamushavebeli()
    {
        MessageBox.Show("სტრიქონები ერთნაირი არ არის");
    }
}
private void button3_Click(object sender, EventArgs e)
{
    Klasi3 obieqti_movlena = new Klasi3();
    //
    obieqti_movlena.ChemiMovlena += new
        ChemiDelegati3(obieqti_movlena.MovlenisDamamushavebeli);
    string striqoni1 = textBox1.Text;
    string striqoni2 = textBox2.Text;
    //
}

```

```

    obieqti_movlena.metodi(striqoni1, striqoni2);
}

```

9.2.4.

```

delegate void ChemiDelegati4();
class Klasi4
{
    public event ChemiDelegati4 ChemiMovlena;
    public void metodi(int[] masivi1)
    {
        int alami = 0;
        for ( int index = 0; index < masivi1.Length; index++ )
            if ( masivi1[index] >= 50 ) alami = 1;
            if ( alami == 0 ) ChemiMovlena();
    }
    public void MovlenisDamamushavebeli()
    {
        MessageBox.Show("მასივში ყველა ელემენტი 50-ზე ნაკლებია");
    }
}
private void button6_Click(object sender, EventArgs e)
{
    Klasi4 obieqti_movlena = new Klasi4();
    //
    obieqti_movlena.ChemiMovlena += new
        ChemiDelegati4(obieqti_movlena.MovlenisDamamushavebeli);
    int[] masivi = new int[] { 1, 2, 3, 4, 5 };
    //
    obieqti_movlena.metodi(masivi);
}

```

9.2.5.

```

delegate void ChemiDelegati5();
class Klasi5
{
    public event ChemiDelegati5 ChemiMovlena;
    public void metodi(int[] masivi1, int[] masivi2, int[] masivi3)
    {
        for ( int index = 0; index < masivi1.Length; index++ )
            if ( masivi2[index] == 0 ) ChemiMovlena();
            else masivi3[index] = masivi1[index] / masivi2[index];
    }
    public void MovlenisDamamushavebeli()
    {
        MessageBox.Show("ადგილი აქვს ნულზე გაყოფას");
    }
}
private void button1_Click(object sender, EventArgs e)
{

```

```

Klasi5 obieqti_movlena = new Klasi5();
//
obieqti_movlena.ChemiMovlena += new
    ChemiDelegati5(obieqti_movlena.MovlenisDamamushavebeli);
int[] masivi1 = new int[] { 92, 20, 37, 44, 69 };
int[] masivi2 = new int[] { 1, 0, 3, 4, 5 };
int[] masivi3 = new int[masivi1.Length];
//
obieqti_movlena.metodi(masivi1, masivi2, masivi3);
for ( int index = 0; index < masivi1.Length; index++ )
    label1.Text += masivi3[index].ToString() + " ";
}

```

9.2.8.

```

delegate void Chemidelegati1(ref int par1);
class klasi
{
    public event Chemidelegati1 movlena;
    public void Metodi(ref int par1)
    {
        if (par1 % 2 == 0) movlena(ref par1);
    }
    public void Damamushavebeli(ref int par1)
    {
        par1 *= par1;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    int ricxvi = Convert.ToInt32(textBox1.Text);
    klasi obj = new klasi();
    obj.movlena += new Chemidelegati1(obj.Damamushavebeli);
    obj.Metodi(ref ricxvi);
    label1.Text = ricxvi.ToString();
}

```

ინტერფეისები. რეალიზება

9.3.1.

```

public interface ChemiInterpeisi
{
    double Kvadrati(int x);
    double Kubi(int x);
}
class ChemiKlasi : ChemiInterpeisi
{
    public double Kvadrati(int x)
    {
        return Math.Pow(x,2);
    }
}

```



```

    }
    public double Kubi(int x)
    {
        return Math.Pow(x, 3);
    }
}
private void button1_Click(object sender, EventArgs e)
{
    ChemiKlasi Obieqti = new ChemiKlasi();
    int ricxvi = Convert.ToInt32(textBox1.Text);
    double kvadrati = Obieqti.Kvadrati(ricxvi);
    double kubi = Obieqti.Kubi(ricxvi);
    label1.Text = kvadrati.ToString() + " " + kubi.ToString();
}

```

9.3.2.

```

public interface ChemiInterpeisi2
{
    int tviseba
    {
        get;
        set;
    }
}
class ChemiKlasi2 : ChemiInterpeisi2
{
    int cvladi;
    public int tviseba
    {
        get
        {
            return cvladi;
        }
        set
        {
            if ( value % 2 == 0 ) cvladi = value;
        }
    }
}
private void button2_Click(object sender, EventArgs e)
{
    // ინტერფეისის თვისებების დემონსტრირება
    ChemiKlasi2 obieqti = new ChemiKlasi2();

    obieqti.tviseba = Convert.ToInt32(textBox1.Text);
    label1.Text += obieqti.tviseba.ToString();
}

```

9.3.3.

```
class Klasi3
{
int ricxvi1;
int ricxvi2;
int ricxvi3;
// ინდექსატორის გამოცხადება
public int this[int index]
{
get
{
switch ( index )
{
case 0 : return this.ricxvi1;
case 1 : return this.ricxvi2;
case 2 : return this.ricxvi3;
default : return 0;
}
}
set
{
if ( value % 2 == 0 ) goto M1;
switch ( index )
{
case 0 : this.ricxvi1 = value; break;
case 1 : this.ricxvi2 = value; break;
case 2 : this.ricxvi3 = value; break;
}
M1: ;
}
}
}

private void button3_Click(object sender, EventArgs e)
{
int ricxvi1 = Convert.ToInt32(textBox1.Text);
int ricxvi2 = Convert.ToInt32(textBox2.Text);
int ricxvi3 = Convert.ToInt32(textBox3.Text);
Klasi3 obieqti = new Klasi3();
label1.Text = obieqti[0].ToString();
label2.Text = obieqti[1].ToString();
label3.Text = obieqti[2].ToString();
obieqti[0] = ricxvi1;
obieqti[1] = ricxvi2;
obieqti[2] = ricxvi3;
label1.Text = obieqti[0].ToString();
label2.Text = obieqti[1].ToString();
label3.Text = obieqti[2].ToString();
}
```

```
}
```

9.3.4.

```
class Klasi1
```

```
{
```

```
    public event ChemiDelegati ChemiMovlena;
```

```
    public void metodi()
```

```
    {
```

```
        if ( ChemiMovlena != null ) ChemiMovlena();
```

```
    }
```

```
    public void MovlenisDamamushavebeli()
```

```
    {
```

```
        MessageBox.Show("მაქსიმალური რიცხვი 25-ზე მეტია");
```

```
    }
```

```
}
```

```
private void button4_Click(object sender, EventArgs e)
```

```
{
```

```
    Klasi1 obieqti_movlena = new Klasi1();
```

```
    //
```

```
    obieqti_movlena.ChemiMovlena += new
```

```
        ChemiDelegati(obieqti_movlena.MovlenisDamamushavebeli);
```

```
    int[] masivi1 = new int[] { 2, 45, 3, 87, 9 };
```

```
    //
```

```
    int max = masivi1[0];
```

```
    for ( int indexi = 0; indexi < masivi1.Length; indexi++ )
```

```
        if ( max < masivi1[indexi] ) max = masivi1[indexi];
```

```
    if ( max > 25 ) obieqti_movlena.metodi();
```

```
}
```

ინტერფეისები. მემკვიდრეობითობა

9.4.1.

```
// Interpeisi_A ინტერფეისის გამოცხადება
```

```
public interface Interpeisi_A
```

```
{
```

```
// მეთოდის გამოცხადება
```

```
    int Perimetri(int par1, int par2, int par3);
```

```
}
```

```
public class SabazoKlasi
```

```
{
```

```
    double fartobi;
```

```
public double Fartobi(int gverdi1, int gverdi2)
```

```
{
```

```
    fartobi = gverdi1 * gverdi2 / 2;
```

```
    return fartobi;
```

```
}
```

```
}
```

```
public class MemkvidreKlasi : SabazoKlasi, Interpeisi_A
```

```

{
int perimetri;
//   Interpeisi_A ინტერფეისის Perimetri() მეთოდის რეალიზება
public int Perimetri(int gverdi1, int gverdi2, int gverdi3)
{
perimetri = gverdi1 + gverdi2 + gverdi3;
return perimetri;
}
}
private void button1_Click(object sender, EventArgs e)
{
    MemkvidreKlasi obieqti = new MemkvidreKlasi();

    int gverdi1 = Convert.ToInt32(textBox1.Text);
    int gverdi2 = Convert.ToInt32(textBox2.Text);
    int gverdi3 = Convert.ToInt32(textBox3.Text);
    int perimetri = obieqti.Perimetri(gverdi1, gverdi2, gverdi3);
    double fartobi = obieqti.Fartobi(gverdi1, gverdi2);
    label1.Text = perimetri.ToString() + " " + fartobi.ToString();
}

```

9.4.2.

```

//   Interpeisi_B ინტერფეისის გამოცხადება
public interface Interpeisi_B
{
    //   მეთოდის გამოცხადება
    int Minimaluri(int[] masivi1);
}
public class SabazoKlasi2
{
    public int Maqsimaluri(int[] masivi2)
    {
        int max = masivi2[0];
        for ( int indeqsi = 0; indeqsi < masivi2.Length; indeqsi++ )
            if ( max < masivi2[indeqsi] ) max = masivi2[indeqsi];
        return max;
    }
}
public class MemkvidreKlasi2 : SabazoKlasi2, Interpeisi_B
{
    public int Minimaluri(int[] masivi3)
    {
        int min = masivi3[0];
        for ( int indeqsi = 0; indeqsi < masivi3.Length; indeqsi++ )
            if ( min > masivi3[indeqsi] ) min = masivi3[indeqsi];
        return min;
    }
}

```

```

private void button2_Click(object sender, EventArgs e)
{
    MemkvidreKlasi2 obieqti = new MemkvidreKlasi2();

    int[] masivi = new int[] { 1, 2, 3, 4, 5 };
    int max = obieqti.Maqsimaluri(masivi);
    int min = obieqti.Minimaluri(masivi);
    labell1.Text = min.ToString() + " " + max.ToString();
}

```

ინტერფეისები. ცხადი რეალიზება

9.5.1.

```

interface Interpeisi_1
{
    int metodi(int gverdi1, int gverdi2, int gverdi3);
}
interface Interpeisi_2
{
    int metodi(int gverdi1, int gverdi2);
}
// ChemiKlasi3 კლასში ხდება ორივე ინტერფეისის რეალიზება
class ChemiKlasi3 : Interpeisi_1, Interpeisi_2
{
    Interpeisi_1 obieqti_1;
    Interpeisi_2 obieqti_2;
    // ორივე metodi() მეთოდის აშკარა რეალიზება
    int Interpeisi_1.metodi(int gverdi1, int gverdi2, int gverdi3)
    {
        return gverdi1 + gverdi2 + gverdi3;
    }
    int Interpeisi_2.metodi(int gverdi1, int gverdi2)
    {
        return gverdi1 * gverdi2 / 2;
    }
    // metodi() მეთოდის გამოძახება ინტერფეისული მიმართვის საშუალებით
    public int metodi_1(int gverdi1, int gverdi2, int gverdi3)
    {
        // a_obieqti ობიექტს ენიჭება გამომძახებელ ობიექტზე მიმართვა
        obieqti_1 = this;
        return obieqti_1.metodi(gverdi1, gverdi2, gverdi3); //Interpeisi_A მეთოდის გამოძახება
    }
    public int metodi_2(int gverdi1, int gverdi2)
    {
        obieqti_2 = this; // b_obieqti ობიექტს ენიჭება გამომძახებელ ობიექტზე მიმართვა
        return obieqti_2.metodi(gverdi1, gverdi2); // Interpeisi_B მეთოდის გამოძახება
    }
}

```

```

private void button3_Click(object sender, EventArgs e)
{
    // აშკარა რეალიზაციის გამოყენება არაერთგვაროვნობის აღმოსაფხვრელად
    ChemiKlasi3 obieqti = new ChemiKlasi3();
    int ricxvi1 = Convert.ToInt32(textBox1.Text);
    int ricxvi2 = Convert.ToInt32(textBox2.Text);
    int ricxvi3 = Convert.ToInt32(textBox3.Text);

    // Interpeisi_A.metodi() მეთოდის გამოძახება
    label1.Text = obieqti.metodi_1(ricxvi1, ricxvi2, ricxvi3).ToString();
    // Interpeisi_B.metodi() მეთოდის გამოძახება
    label2.Text = obieqti.metodi_2(ricxvi1, ricxvi2).ToString();
}

```

9.5.2.

```

public interface Interpeisi11
{ int tviseba { get; set; } }
public interface Interpeisi12
{ int tviseba { get; set; } }
class ChemiKlasi22 : Interpeisi11, Interpeisi12
{
    int cvladi1, cvladi2;
    Interpeisi11 obieqti_1;
    Interpeisi12 obieqti_2;
    int Interpeisi11.tviseba
    {
        get { return cvladi1; }
        set { if ( value >= 0 ) cvladi1 = value; }
    }
    int Interpeisi12.tviseba
    {
        get { return cvladi2; }
        set { if ( value < 0 ) cvladi2 = value; }
    }
    public int tviseba1
    {
        get { obieqti_1 = this;
            return obieqti_1.tviseba; }
        set { obieqti_1 = this;
            if ( value >= 0 ) obieqti_1.tviseba = value; }
    }
    public int tviseba2
    {
        get { obieqti_2 = this;
            return obieqti_2.tviseba; }
        set { obieqti_2 = this;
            if ( value < 0 ) obieqti_2.tviseba = value; }
    }
}

```

```

}
private void button2_Click(object sender, EventArgs e)
{
    ChemiKlasi22 obieqti = new ChemiKlasi22();
    obieqti.tviseba1 = Convert.ToInt32(textBox1.Text);
    obieqti.tviseba2 = Convert.ToInt32(textBox2.Text);
    label1.Text = obieqti.tviseba1.ToString();
    label2.Text = obieqti.tviseba2.ToString();
}

```

სახელების სივრცე. using დირექტივა. ჩადგმული სახელების სივრცე

9.6.1.

```

namespace Sivrc_1
{
    public class Samkutxedi
    {
        public double Fartobi(int par1, int par2)
        {
            return par1 * par2 / 2;
        }
    }
}
namespace Sivrc_2
{
    public class Samkutxedi
    {
        public int Perimetri(int par1, int par2, int par3)
        {
            return par1 + par2 + par3;
        }
    }
}
private void button1_Click(object sender, EventArgs e)
{
    Sivrc_1.Samkutxedi samkutxedi_1 = new Sivrc_1.Samkutxedi();
    Sivrc_2.Samkutxedi samkutxedi_2 = new Sivrc_2.Samkutxedi();
    int gverdi_1 = Convert.ToInt32(textBox1.Text);
    int gverdi_2 = Convert.ToInt32(textBox2.Text);
    int gverdi_3 = Convert.ToInt32(textBox3.Text);
    double fartobi = samkutxedi_1.Fartobi(gverdi_1, gverdi_2);
    int perimetri = samkutxedi_2.Perimetri(gverdi_1, gverdi_2, gverdi_3);
    label1.Text = fartobi.ToString();
    label2.Text = perimetri.ToString();
}

```

9.6.2.

```

namespace Sivrc
{

```

```

namespace Sivrcce_1
{
    public class Masivi
    {
        public int Metodi1(int[] masivi1)
        {
            int jami = 0;
            for ( int indexi = 0; indexi < masivi1.Length; indexi++ )
                if ( masivi1[indexi] >= 0 ) jami += masivi1[indexi];
            return jami;
        }
    }
}
namespace Sivrcce_2
{
    public class Masivi
    {
        public int Metodi2(int[] masivi1)
        {
            int namravli = 1;
            for ( int indexi = 0; indexi < masivi1.Length; indexi++ )
                if ( masivi1[indexi] < 0 ) namravli *= masivi1[indexi];
            return namravli;
        }
    }
}
private void button1_Click(object sender, EventArgs e)
{
    int[] masivi = new int[] { 1, -2, 3, -4, 5, -6 };
    Sivrcce.Sivrcce_1.Masivi obieqti_1 = new Sivrcce.Sivrcce_1.Masivi();
    Sivrcce.Sivrcce_2.Masivi obieqti_2 = new Sivrcce.Sivrcce_2.Masivi();
    int jami = obieqti_1.Metodi1(masivi);
    int namravli = obieqti_2.Metodi2(masivi);
    label1.Text = jami.ToString();
    label2.Text = namravli.ToString();
}

```

თავი 10. ინფორმაციის შეტანა-გამოტანა

ფაილში ბაიტების შეტანა-გამოტანა. FileStream კლასი

10.1.1.

```

{
    label1.Text = "";
    int wakitxuli_baitebis_raodenoba;
    byte jami = 0;

```



```

byte[] masivi = new byte[5];
FileStream file1 = new FileStream("filetext.txt", FileMode.Open);

wakitxuli_baitebis_raodenoba = file1.Read(masivi, 0, 5);

for ( int indexi = 0; indexi < masivi.Length; indexi++ )
    jami += masivi[indexi];
file1.WriteByte(jami);
file1.Close();
for ( int indexi = 0; indexi < masivi.Length; indexi++ )
    label1.Text += masivi[indexi].ToString() + " ";
label1.Text += jami.ToString();
label2.Text = wakitxuli_baitebis_raodenoba.ToString();
}

```

10.1.2.

```

{
    label1.Text = "";
    int wakitxuli_baitebis_raodenoba;
    byte[] masivi = new byte[5];
    FileStream file1 = new FileStream("file1.txt", FileMode.Open);
    FileStream file2 = new FileStream("file2.txt", FileMode.Create);

    wakitxuli_baitebis_raodenoba = file1.Read(masivi, 0, 5);
    file2.Write(masivi, 0, 5);
    file1.Close();
    file2.Close();
    for ( int indexi = 0; indexi < masivi.Length; indexi++ )
        label1.Text += masivi[indexi].ToString() + " ";
    label2.Text = wakitxuli_baitebis_raodenoba.ToString();
}

```

10.1.3.

```

{
    int ricxvi;
    FileStream file_in;
    FileStream file_out;
    //    საწყისი ფაილის გახსნა
    file_in = new FileStream("s_file.txt", FileMode.Open);
    //    მიმღები ფაილის გახსნა
    file_out = new FileStream("d_file.txt", FileMode.Create);
    label1.Text = "";
    //    ბაიტების გადაწერა file_in ფაილიდან file_out ფაილში
    for ( int i = 1; i <= 10; i++ )
    {
        //    ბაიტების წაკითხვა file_in ფაილიდან
        ricxvi = file_in.ReadByte();
        //    ბაიტების ჩაწერა file_out ფაილში
        if ( ricxvi != -1 ) file_out.WriteByte( ( byte ) ricxvi );
    }
}

```

```

label1.Text += ricxvi + " ";
}
// ფაილების დახურვა
file_in.Close();
file_out.Close();
}

```

ფაილში სიმბოლოების შეტანა-გამოტანა. StreamReader და StreamWriter კლასები

10.2.1.

```

{
    label1.Text = "";
    FileStream file_in;
    char[] simboloebis_masivi = new char[14];
    file_in = new FileStream("file1.txt", FileMode.Open);
    StreamReader file_stream_in = new StreamReader(file_in);

    int wakitxuli_simboloebis_raodenoba = file_stream_in.Read(simboloebis_masivi, 0, 14);
    for ( int i = 0; i < simboloebis_masivi.Length; i++ )
        label1.Text += simboloebis_masivi[i].ToString();
    file_stream_in.Close();
    //
    FileStream file_out;
    file_out = new FileStream("file2.txt", FileMode.Create);
    StreamWriter file_stream_out = new StreamWriter(file_out);
    // file2.txt ფაილში simboloebis_masivi სტრიქონის ჩაწერა
    file_stream_out.Write(simboloebis_masivi);
    file_stream_out.Close();
}

```

10.2.2.

```

{
    label1.Text = "";
    FileStream file_in;
    char[] simboloebis_masivi = new char[14];
    file_in = new FileStream("file3.txt", FileMode.Open);
    StreamReader file_stream_in = new StreamReader(file_in);

    int wakitxuli_simboloebis_raodenoba = file_stream_in.Read(simboloebis_masivi, 0, 14);
    for ( int i = 0; i < simboloebis_masivi.Length; i++ )
        label1.Text += simboloebis_masivi[i].ToString();
    file_stream_in.Close();
    FileStream file_out;
    file_out = new FileStream("file3.txt", FileMode.Append);
    StreamWriter file_stream_out = new StreamWriter(file_out);
    // file2.txt ფაილში simboloebis_masivi სტრიქონის ჩაწერა
    file_stream_out.Write();
    file_stream_out.Close();
}

```

10.2.3.

```
{
    label1.Text = ""; label2.Text = "";
    char simbolo;
    FileStream file_in;
    file_in = new FileStream("file3.txt", FileMode.Open);
    StreamReader file_stream_in = new StreamReader(file_in);
    FileStream file_out;
    file_out = new FileStream("file4.txt", FileMode.Create);
    StreamWriter file_stream_out = new StreamWriter(file_out);
    for ( int i = 0; i < file_in.Length; i++ )
    {
        simbolo = ( char ) file_stream_in.Read();
        file_stream_out.Write(simbolo);
        label2.Text += ( char ) simbolo;
    }
    file_stream_in.Close();
    file_stream_out.Close();
}
```

ფაილებთან პირდაპირი მიმართვა. Seek მეთოდი

10.4.2.

```
{
    int seek = Convert.ToInt32(textBox2.Text);
    label1.Text = "";
    FileStream file_out;
    byte[] masivi1 = new byte[10] { 255, 100, 200, 1, 50, 130, 250, 20, 90, 190 };
    byte[] masivi2 = new byte[masivi1.Length];
    file_out = new FileStream("file2.txt", FileMode.Create);
    //
    //StreamWriter file_stream_out = new StreamWriter(file_out);
    file_out.Seek(seek, SeekOrigin.Begin);
    file_out.Write(masivi1, 0, 9);
    file_out.Close();
    FileStream file_in;
    file_in = new FileStream("file2.txt", FileMode.Open);
    file_in.Read(masivi2, 0, 9);
    for ( int index = 0; index < masivi2.Length; index++ )
        label1.Text += masivi2[index].ToString() + " ";
    file_in.Close();
}
```

10.4.4.

```
{
    int seek = Convert.ToInt32(textBox2.Text);
    label1.Text = "";
    FileStream file_in;
    char[] masivi = new char[14];
```

```

file_in = new FileStream("file1.txt", FileMode.Open);
StreamReader file_stream_in = new StreamReader(file_in);
file_in.Seek(seek, SeekOrigin.Begin);
file_stream_in.Read(masivi, 0, 9);
for ( int index = 0; index < masivi.Length; index++ )
label1.Text += masivi[index].ToString();
file_stream_in.Close();
}

```

ფაილის შესახებ ინფორმაციის მიღება. FileInfo კლასი

10.5.1.

```

{
    FileInfo obieqti = new FileInfo("file1.txt");
    label1.Text += obieqti.Attributes.ToString() + '\n';
}

```

10.5.2.

```

{
    FileInfo obieqti = new FileInfo("file1.txt");
    label1.Text += obieqti.CreationTime.ToString() + '\n';
}

```

10.5.3.

```

{
    FileInfo obieqti = new FileInfo("file1.txt");
    label1.Text += obieqti.Directory.ToString() + '\n';
    label1.Text += obieqti.DirectoryName + '\n';
}

```

10.5.4.

```

{
    FileInfo obieqti = new FileInfo("file1.txt");
    label1.Text += obieqti.Exists.ToString() + '\n';
}

```

10.5.5.

```

{
    FileInfo obieqti = new FileInfo("file1.txt");
    label1.Text += obieqti.Extension + '\n';
}

```

10.5.6.

```

{
    FileInfo obieqti = new FileInfo("file1.txt");
    label1.Text += obieqti.FullName + '\n';
}

```

10.5.7.

```

{
    FileInfo obieqti = new FileInfo("file1.txt");
    label1.Text += obieqti.LastAccessTime.ToString() + '\n';
}

```

10.5.8.

```
{
    FileInfo obieqti = new FileInfo("file1.txt");
    label1.Text += obieqti.LastWriteTime.ToString() + '\n';
}
```

10.5.9.

```
{
    FileInfo obieqti = new FileInfo("file1.txt");
    label1.Text += obieqti.Length.ToString() + '\n';
}
```

კატალოგებთან მუშაობა. DirectoryInfo კლასი

10.6.1.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    label1.Text += obj1.Attributes.ToString() + '\n';
}
```

10.6.2.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    label1.Text += obj1.CreationTime.ToString() + '\n';
}
```

10.6.3.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    label1.Text += obj1.FullName.ToString() + '\n';
}
```

10.6.4.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    label1.Text += obj1.LastAccessTime.ToString() + '\n';
}
```

10.6.5.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    label1.Text += obj1.LastWriteTime.ToString() + '\n';
}
```

10.6.6.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    label1.Text += obj1.Exists.ToString() + '\n';
}
```

10.6.7.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    label1.Text += obj1.Parent.ToString() + '\n';
}
```

10.6.8.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    label1.Text += obj1.Root.ToString() + '\n';
}
```

10.6.9.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    DirectoryInfo[] dir = obj1.GetDirectories();
    for ( int ind = 0; ind < dir.Length; ind++ )
        label1.Text += dir[ind].ToString() + '\n';
}
```

10.6.10.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install");
    FileInfo[] dir = obj1.GetFiles();
    for ( int ind = 0; ind < dir.Length; ind++ )
        label1.Text += dir[ind].ToString() + '\n';
}
```

10.6.11.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Katalogi_1");
    //     Katalogi_2 კატალოგი არ უნდა არსებობდეს
    obj1.MoveTo("C:\\Katalogi_2");
}
```

10.6.12.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install_1");
    //     C: დისკზე იქმნება Install1 კატალოგი
    obj1.Create();
}
```

10.6.13.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install_1");
    //     Install1 კატალოგში იქმნება Install2 ქვეკატალოგი
    obj1.CreateSubdirectory("Install_2");
}
```

10.6.14.

```
{
    DirectoryInfo obj1 = new DirectoryInfo("C:\\Install_1");
    obj1.Delete();
}
```

თავი 11. განსაკუთრებული სიტუაციები

განსაკუთრებული სიტუაციები. try, catch, throw და finally ოპერატორები

11.1.1.

```
{
    int[] masivi = new int[] { 1, 2, 3, 4, 5 };
    int index, jami = 0;
    try
    {
        for ( index = 0; index <= masivi.Length; index++ )
            jami += masivi[index];
    }
    catch ( IndexOutOfRangeException )
    {
        jami = 0;
        label2.Text = "ინდექსი გადის დასაშვები მნიშვნელობების დიაპაზონის გარეთ";
    }
    label1.Text = jami.ToString();
}
```

11.1.2.

```
{
    int[,] masivi = new int[3,3] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
    int striqoni, sveti, jami = 0;
    try
    {
        for ( striqoni = 0; striqoni < 4; striqoni++ )
            for ( sveti = 0; sveti < 3; sveti++ )
                jami += masivi[striqoni,sveti];
    }
    catch ( IndexOutOfRangeException )
    {
        jami = 0;
        label2.Text = "ინდექსი გადის დასაშვები მნიშვნელობების დიაპაზონის გარეთ";
    }
    label1.Text = jami.ToString();
}
```

11.1.3.

```
{
    int[] masivi1 = new int[] { 10, 20, 30, 40, 50 };
    int[] masivi2 = new int[] { 1, 0, 3, 0, 5 };
    int[] shedegi = new int[masivi1.Length];
    try
    {
        for ( int index = 0; index < masivi1.Length; index++ )
            shedegi[index] = masivi1[index] / masivi2[index];
    }
    catch ( DivideByZeroException )
    {
    }
}
```

```

    {
        label2.Text = "აღვიწილი აქვს 0-ზე გაცივდას";
    }
    for ( int index = 0; index < masivi1.Length; index++ )
        label1.Text += shedegi[index].ToString() + " ";
}

```

11.1.4.

```

{
    int[] masivi = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int indexi;
    BinaryWriter file_out = new BinaryWriter(new FileStream("file1.txt", FileMode.Create));
    try
    {
        for ( indexi = 0; indexi < masivi.Length; indexi++ )
            file_out.Write(masivi[indexi]);
    }
    catch ( IOException arg1 )
    {
        label1.Text = arg1.Message;
    }
    finally
    {
        file_out.Close();
    }
}

```

11.1.5.

```

{
    int[,] masivi = new int[3,3];
    int striqoni, sveti;
    BinaryReader file_in = new BinaryReader(new FileStream("file1.txt", FileMode.Open));
    try
    {
        for ( striqoni = 0; striqoni < 3; striqoni++ )
        {
            for ( sveti = 0; sveti < 3; sveti++ )
            {
                masivi[striqoni, sveti] = file_in.ReadInt32();
                label1.Text += masivi[striqoni, sveti].ToString() + " ";
            }
            label1.Text += '\n';
        }
    }
    catch ( IOException arg1 )
    {
        label2.Text = arg1.Message;
    }
    finally

```



```

    {
        file_in.Close();
    }
}

```

თავი 12. სტრიქონები

სტრიქონები

12.3.1.

```

{
label1.Text = ""; label2.Text = ""; label3.Text = "";
char[,] masivi = new char[3, 3] { { 'ა', 'კ', 'ა' }, { 'რ', 'ს', 'ო' }, { 'ა', 'ტ', 'პ' } };
int[] masivi2 = new int[3];
int striqoni, sveti, max = masivi2[0], maxind = 0;

for ( striqoni = 0; striqoni < 3; striqoni++ )
    for ( sveti = 0; sveti < 3; sveti++ )
        if ( masivi[striqoni, sveti] == 'ა' ) masivi2[striqoni]++;
//
for ( striqoni = 0; striqoni < masivi2.Length; striqoni++ )
    if ( max < masivi2[striqoni] )
    {
        max = masivi2[striqoni];
        maxind = striqoni;
    }
label1.Text = "ა" სიმბოლოს მაქსიმალურ რაოდენობას - " + max.ToString() +
    "\nშეიცავს ორგანზომილებიანი მასივის სტრიქონი,\n" +
    "რომლის ინდექსია - " + maxind.ToString();
for ( int indexi = 0; indexi < masivi2.Length; indexi++ )
    label2.Text += masivi2[indexi].ToString() + " ";
for ( striqoni = 0; striqoni < 3; striqoni++ )
{
for ( sveti = 0; sveti < 3; sveti++ )
    label3.Text += masivi[striqoni, sveti].ToString() + " ";
label3.Text += '\n';
}
}

```

ლიტერატურა

1. რ. სამხარაძე. Visual C#.NET. საქართველოს ტექნიკური უნივერსიტეტი. 2009. - 455 გვ.
2. რ. სამხარაძე. SQL სერვერი. საქართველოს ტექნიკური უნივერსიტეტი. 2008. - 374 გვ.
3. რ. სამხარაძე. Visual C++/CLI.NET. საქართველოს ტექნიკური უნივერსიტეტი. 2010. - 442 გვ.
4. Шилдт. Г. С# .
5. Смайли Дж. Учимся программировать на С# .
6. Davis. Н. Visual C#.NET Programming.
7. Прайс дж., Гандерлой М. Visual C#.NET .
8. Butow E., Ryan T. С# .
9. O'Reilly. Programming C# .
10. Г. Дейтел. Введение в операционные системы. В 2-х томах. Пер. с англ. - М.: Мир, 1987.
11. Э. Таненбаум. Современные операционные системы. - СПб.: Питер, 2002. - 1040 с.: ил.
12. Saurabh Nandu. C#.NET. Web Developer's Guide.
13. O'Reilly. Programming C#.
14. Разработка Windows-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# -NET. Учебный курс MCAD/MCSD/Пер. с англ. - М.: Издательско-торговый дом "Русская Редакция", 2003. - 512 стр.: ил.
15. E.Butow, T. Ryan. Your visual blueprint for building .NET applications.
16. Harold Davis. Visual C# .NET Programming.
17. A. Hejlsberg, S.Wiltamuth. C# Language Reference.
18. Г. Шилдт. Полный справочник по С#.
19. C# Frequently Asked Questions <<http://blogs.msdn.com/csharpfaq/default.aspx>>.
20. O'REILLY. C# 3.0 Cookbook. 2007.
21. Hejlsberg A., Wiltamuth S. C# Language Reference.
22. Уотсон, Карл и, Нейгел, Кристиан, Педерсен, Якоб Хаммер, Рид, Джон Д., Скиннер, Морган, Уайт, Эрик. Visual C# 2008: базовый курс. : Пер. с англ. - М. : ООО "И.Д. Вильямс", 2009. - 1216 с.: ил. — Парал. тит. англ. ISBN 978-5-8459-1532-0 (рус.)
23. Professional C# 4 and .NET 4. Published by Wiley Publishing, Inc. 10475 Crosspoint Boulevard Indianapolis, IN 46256 www.wiley.com . Copyright © 2010 by Wiley Publishing, Inc., Indianapolis, Indiana. Published simultaneously in Canada. ISBN: 978-0-470-50225-9

რედაქტორი

გადაეცა წარმოებას 01.22.2014. ხელმოწერილია დასაბეჭდად 20.12.2014. ბეჭდვა ოფსეტური.
პირობითი ნაბეჭდი თაბახი 31,44. ტირაჟი 50 ეგზ. შეკვეთა #

გამომცემლობა "ტექნიკური უნივერსიტეტი", თბილისი, კოსტავას 77



Verba volant,
scripta manent