ჯემალ ანთიძე

დაპროგრამება C# ენაზე

თბილისი 2017

ჯემალ ანთიძე

დაპროგრამება C# ენაზე

სახელმძღვანელო კომპიუტერულ მეცნიერებათა მიმართულების სტუდენტებისათვის

თბილისი 2017

წინათქმა

წინამდებარე სახელმძღვანელო განკუთვნილია კომპიუტერულ მეცნიერებათა მიმართულების სტუდენტებისათვის და მათთვის, ვისაც სურს სრულყოფილად დაეფლოს C# დაპროგრამების ენას და მის გამოყენებას სხვადასხვა პრობლემების კომპიუტერული რეალიზაციისათვის. სახელმძღვანელოში მოცემულია როგორც C# ინსტრუქციების სრული ფორმალური აღწერა, ასევე ფართოდაა წაროდგენილი მათი გამოყენება კონკრეტულ პროგრამებში ისე, რომ შემსწავლელს შეუძლია პრაქტიკულად გამოსცადოს მათი შესრულება კომპიუტერზე. დანართის სახით მოცემულია პროგრამები, რომელთა გარჩევა და კომპიუტერზე რეალიზაცია დაეხმარება შემსწავლელს დაოსტატდეს . სახელმძღვანელო წარმოადგენს ჩემს მიერ გადამუშავებულ ვერსიას C# ლექციების კურსისა, რომელიც გამოქვეყნებულია ელექტრონულად თბილისის სახელმწიფო უნივერსიტეტის ჩემს ვებ გვედზე(იხილეთ: http://fpv.science.tsu.ge). ავტორი მრავალი წლის განმავლობაში კითხულობდა ლექციებსა და ატარებდა პრაქტიკულ მეცადინეობებს C# დაპროგრამებაში თბილისის სახელმწიფო უნივერსიტეტსა და სხვა უმაღლეს სასწავლებელში. მიუხედავად ამისა, სახელმძღვანელო არაა დაზღვეული შეცდომებისა თუ ნაკლისაგან. ავტორი მადლიერებით მიიღებს ყველა საქმიან შენიშვნასა თუ სურვილებს მკითხველებისაგან.

ს ა რ ჩ ე ვ ი

1 თავი

ზოგადი ინსტრუქციები

1.1 მუშაობის დაწყება C#-ში

შინაარსი :

პროგრამის ძირითადი სტრუქტურა;

namespace-ის გამოყენება;

Main მეთოდის გამოყენება;

შეტანა comand line-ით;

პროგრამის ძირითადი სტრუქტურა

მარტივი C# პროგრამა

```
// Namespace-ის გამოცხადება
using System;
// პროგრამის დაწყების კლასი
class Cmisalmeba
{
// Main მეთოდი იწყებს პროგრამის შესრულებას
      static void Main()
   {
      // ეკრანზე გამოტანა
      Console.WriteLine("ვიწყებთ C# შესწავლას!");
   }
```

}

ამ პროგრამის კომპილაცია შე იძლება შემდეგი ბრძანებით (იგულისხმება, რომ პროგრამის ტექსტი ჩაწერილია Cmisalmeba.cs ფაილში):

csc.exe Cmisalmeba.cs

კომპილაციის შედეგი ჩაიწერება ფაილში Cmisalmeba.exe სტანდარტულ ბიბლიოთეკაში. თუ სხვაგან გვინდა ჩაწერა, მაშინ უნდა მივუთითოთ სრული გზა . //-ით იწყება ერთსტრიქონიანი კომენტარი. იგი კომპილაციის დროს არ განიხილება და გამოიყენება მხოლოდ როგორც პროგრამის ახსნა-განმარტება. ბრძანებათა სტრიქონი არის ფანჯარა, რომლის საშუალებითაც შეიძლება ბრძანებებისა და პროგრამების გაშვება სათანასო ტექსტის აკრეფით. რომ გამოვიყენოთ ბრძანებათა სტრიქონში მუშაობა, საჭიროა, მაგალითად, WINDOWS XP ოპერციულ სისტემაში შევიდეთ, START ნენიუში ავირჩიოთ RUN და აკრიფოთ CMD. შევნიშნოთ, რომ C# კლავიატურის რეგისტრისადმი მგრძობიარეა და ასხვავებს დიდ და პატარა ასოებს.

Namespace-ის გამოცხადება using System ნიშნავს, რომ თქვენ შეგიძლიათ ისარგებლოთ System-ში მოთავსებული მეთოდების კლასებით ისე, რომ წინ არ მიუწეროთ სახელი System მაშინ, როცა ამ გამოცხადების გარეშე უნდა მიუთითოთ სრული სახელი. მაგალითად, წინა პროგრამაში გამოყენებულია მეთოდი WriteLine System-ის წინ მიწერის გარეშე, რადგან პროგრამაში გამოყენებულია using System. პროგრამაში შემოტანილია კლასი სახელით Cmisalmeba, რომლიც შეიცავს Main მეთოდს და იგი არის პროგრამაში შესასვლელი წერტილი, საიდანაც იწყება პროგრამის შესრულება. ყოველ პროგრამას უნდა ჰქონდეს Main მეთოდი, რომელიც მოთავსებულია ძირითად კლასში. კლასისა და მეთოდის საზღვრები მიეთითება გახსნილი და დახურული ფიგურული ფრჩხილებით. სპეციფიკაცია void მიუთითებს, რომ მეთოდს არ გამოაქვს რაიმე მნიშვნელობა. Main მეთოდს წოველთვის აქვს void სპეციფიკაცია. ამ მაგალითში Main მეთოდს აქვს ერთად-ერთი ინსტრუქცია, რომელიც არის System-ის Console ჯგუფის მეთოდი WriteLine. მას გამოაქვს ტექსტი ეკრანზე და გადადის ახალ სტრიქონზე. Console არის Namespace-ის კლასი და WriteLine არის მისი მეთოდი. კლასებსა და Namespace-ებს დაწვრილებით შევისწავლით შემდეგ ლექციებში. ისინი აქ მოტანილი არიან პროგრამის სტრუქტურის საჩვენებლად. static მოდიფიკატორი მიუთითებს, რომ ამ მეთოდის გამოყენება შეიძლება მხოლდ ამ კლასში.

ბრძანებათა სტრიქონით შეტანა

ბრძანებათა სტრიქონით ჩვენ შეგვიძლია გავხსნათ ზოგადი მოხმარების პროგრამები და ვიმუშაოთ მათში. მაგალითად,

Notepad.exe saxeli.txt

ეს ბრძანება გამოიძახებს Notepad რედაქტორს და მასში გახსნის saxeli.txt ფაილს. რასაკვირველია თუ იგი არსებობს იმ დირექტორიაში, რომელსაც სისტემა იცნობს. ჩვემნ შეგვიძლია ბრძანებათა სტრიქონით არგუმენტები და გამოვიყენოთ ისინი პროგრამაში. თუ როგორ გავაკეთოთ ეს ამისათვის განვიხილოთ მაგალითი:

```
// Namespace-ის გამოცხადება

using System;

// პროგრამის დაწყების კლასი

class Csaxeli1

{

   //Main იწყებს პროგრამის შესრულებას.

   static void Main(string[] args)

   {

      // ეკრანზე გამოტანა

      Console.WriteLine("სალამი, {0}!", args[0]);

      Console.WriteLine("გისურვებთ წარმატებას!");

   }

}
```

აქ, ჩვენ Main მეთოდში მივუთითეთ არგუმენტი string[] args, რაც ნიშნავს, რომ args არის string ტიპის მასივი. მისი მნიშვნელობები აიღება ბრძანებათა სტრიქონიდან. სადაც პირველი სახელი არის პროგრამის ფაილის სახელი და დანარჩენი სახელები არიან args მასივის ელემენტები. მასივის ელემენტების გადათვლა იწყება ნულიდან.

Console.WriteLine("სალამი, {0}!", args[0]) ინსტრუქცია

იღებს ბრძანებათა სტრიქონიდან პროგრამის სახელის შემდეგ სახელს და მას განიხილავს როგორც args[0]-ის მნიშვნელობას. თუ ეს სახელია "პეტრე", მა**შინ პროგრამა გამოიტანს ეკრანზე**

სალამი პეტრე!

გისურვებთ წარმატებას!

დიალოგი ბრძანებათა სტრიქონის საშუალებით .

პროგრამაში სტრიქონის შეტანის მეორე ხერხია მისი შეტანა Console-ის საშუალებით: განვიილოთ პროგრამა:

```
using System;

// პროგრამის დაწყების კლასი

class Csaxeli2

{

   //Main იწყებს პროგრამის შესრულებას

   public static void Main()

{

// შეტანა Console-ის საშუალებით

 // ცვლადის განსაზღვრა

 Console.Write("სალამი, {0}! ", Console.ReadLine());

Console.WriteLine("გისურვებთ წარმატებას!");

}

}
```

Console.Write ახდენს თავისი არგუმენტების გამოტანას ისე, რომ არ გადადის ახალ სტრიქონზე. Console.ReadLine() ელოდება მომხმარებლისაგან სტრიქონის შეტანას კლავიატურიდან. მომხმარებელმა უნდა აკრიოს სტრიქონი და დააჭიროს ხელი ENTER-ს. ამის შემდეგ, {0}-ის ნაცვლად აიღება აკრეფილი სტრიქონი და Console.WriteLine გამოიტანს მიღებულ სტრიქონს.

```
Console.Write("სალამი, {0}! ", Console.ReadLine());
```

შეიძლება შეიცვალოს ინსტრუქციით

```
string name=Console.ReadLine() ;

Console.Write("სალამი, {0}! ",name);
```

შედეგად გამოიტანება ეკრანზე იგივე. ასეთ რეჯიმში პროგრამის გამოტანასა და მომხმარებლის მუშაობას აქვს შემდეგი სახე:

ეკრანზე გამოიტანება - თქვენი სახელი? აკრიფეთ თქვენი სახელი კლავიატურაზე და დააჭირეთ ENTER კლავიშას. ამის შემდეგ ეკრანზე გამოიტანება:

სალამი, <თქვენს მიერ აკრეფილი სახელი>! გისურვებთ წარმატებას!

ამგვარად, ჩვენ შევისწავლეთ თუ როგორია პროგრამის სტრუქტურა C#-ში, მუშაობა ბრძანებათა სტრიქონის გამოყენებით და როგორ გამოვიყენოთ ბრძანებათა სტრიქონის არგუმენტები პროგრამაში.

1.2 ცვლადები, ტიპები და ოპერატორები

ცვლადი გამოიყენება პროგრამაში მნიშვნელობაზე მისათითებლად. ცვლადის გამოცხადება ნიშნავს ცვლადისთვის სახელისა და ტიპის განსაზღვრას, ხოლო ცვლადის განსაზღვრა არის მისთვის მნიშვნელობის მიცემა. მეორენაირად მას ცვლადის ინიციალიზაციას უწოდებენ. ამავე დროს განისაზღვრება ადგილი, სადაც ეს მნიშვნელობა იწერება. გამოყოფილი ადგილის სიგრძე განისაზღვრება ცვლადის ტიპის მიხედვით, ხოლო პროგრამაში მისი გამოყენების არე განისაზღვრება ცვლადის გამოცხადების ადგილით. ტიპი არის მნიშვნელობათა სიმრავლე საიდანაც მნიშვნელობის აღება შეიძლება ცვლადისათვის. არსებობენ ორი სახის ტიპები: პროგრამირების ენაში დაფიქსირებული ტიპები, მორენაირად მათ ჩადგმულ ან მარტივ ტიპებს უწოდებენ და პროგრამისტის მიერ შედგენილი ტიპები, რომლებიც მიიღებიან მარტივი ტიპების კომბინაციით. C#-ში მარტივი ტიპებია: ბულის, ინტეგრალური, ნამდვილი და სტრიქონული ტიპები. განვიხილოთ თითოეული ცალკ-ცალკე. ჭეშმარიტი მნიშვნელობა აღინიშნება true-თი, ხოლო მცდარი მნიშვნელობა false-თი. მათ ეწოდებათ ბულის(ლოგიკური) კონსტანტები. ისინი გამოიყენებიან ლოგიკური გამოსახულების ჭეშმარიტობის დასადგენად. თუ რა არის ლოგიკური გამოსახულება და როგორ უნდა გამოვიყენოთ იგი შევისწავლით მოგვიანებით.

ინტეგრალური ტიპი. C#-ში ინტეგრალურ ტიპს მიეკუთვნებიან მთელი რიცხვები ნიშნით ან უნიშნოთ და char ტიპის სიმბოლოები, რომლებიც წარმოადგენენ Unicode-ის სიმბოლოებს(იხილეთ ინტერნეტში Unicode ცხრილი). განვიხილოთ ცხრილი, სადაც მოცემულია integral ტიპის ქვეტიპები, მეხსიერებაში მათთვის გამოყოფილი ბიტების რაოდენობა და იმ რიცხვების დიაპაზონი, რომლებიც შეიძლება წარმოიდგინოს მოცემული ქვეტიპით.

ქვეტიპი	სიგრძე ბიტებში	დიაპაზონი
sbyte	8	128-დან 127-მდე
byte	8	0-დან 255-მდე
short	16	2768-დან 32767-მდე

ushort	16	0-დან 65535-მდე
int	32	-2147483648-დან 2147483647-მდე
uint	32	0-დან 4294967295-მდე
long	64	-92233720 36854775808-დან 9223372036854775807-მდე
ulong	64	0-დან 18446744073709551615-მდე
char	16	0-დან 65535-მდე

მაგალითად, sbyte ტიპის ცვლადს კომპიუტერის მეხსიერებაში გამოეყოფა 8 ბიტის სიგრძის ადგილი. მას შეიძლება ჰქონდეს ნებისმიერი მთელი მნიშვნელობა -128-დან 127-ის ჩათვლით.

ნამდვილი ტიპი. ნამდვილი ტიპის რიცხვებს მიეკუთვნებიან ნამ**დვილი** რიცხვები მოძრავი წერტილით, ნამდვილი რიცხვები მოძრავი წერტილითა და ორმაგი სიზუსტით და ათწილადი რიცხვები. განვიხილოთ ცხრილი:

ტიპი	**სიგრძე ბიტებში**	**სიზუსტე**	**დიაპაზონი**
float	32	7 ათობითი ციფრი	1.5×10^{-45}-დან 3.4×10^{38}
double	64	15-16 ათობითი ციფრი	5.0×10^{-324} t-დან 1.7×10^{308}
decimal	128	28-29 ათობითი ციფრი	1.0×10^{-28} –დან 7.9×10^{28}

ამ ცხრილში დიაპაზონი ზუსტად გვიჩვენებს თუ რა რიცხვები ეკუთვნიან კონკრეტულ ტიპს. ნამდვილი რიცხვები მოძრავი წერტილით შედგება ნაწილებისაგან: მთელი. რომელსაც მოსდევს წერტილი, შემდეგ ათწილადი ნაწილი და შემდეგ ათი ხარისხად მთელი. ზოგადად, იგი ჩაიწერება ასე: $n.m10^p$, სადაც n და p მთელი რიცხვებია, m ნატურალური რიცხვია. ათწილადი რიცხვი ჩაიწერება n.m სახით. n შეიძლება არ გვქონდეს.

სტრიქონული ტიპი. სტრიქონული ტიპის კონსტანტა წარმოიდგინება ორმაგ ბრჭყალებში მოთავსებული სიმბოლოებით, სადაც სიმბოლოები ეკუთვნიან ASCII სტანდარტს. ზოგიერთი მათგანი არიან არაბეჭდვადი სიმბოლოები. ისინი წარმოიდგინებიან სტრიქონში ზოგიერთი ბეჭდვადი სიმბოლოებით, რომელთაც წინ მიწერილი აქვთ სიმბოლო “დახრილი ხაზი”(\). ასეთ სიმბოლოებს ეწოდებათ სპეციალური სიმბოლოები. მაგალითად, \n არის სპეციალური სიმბოლო, რომელიც აღნიშნავს ახალ სტრიქონზე გადასვლას. სპეციალური სიმბოლოები მოცემულია შემდეგ ცხრილში:

სპეციალური სიმბოლო	მნიშვნელობა
\'	ერთმაგი ბრჭყალები
\"	ორმაგი ბრჭყალები
\\	დახრილი ხაზი
\0	nul-კოდი
\a	ზარი
\b	დახრილი ხაზი
\f	ფორმის ჩატვირთვა
\n	ახალ სტრიქონზე გადასვლა
\r	კარეტის უკან დაბრუნება
\t	ჰორიზონტალური ტაბულაცია
\v	ვერტიკალური ტაბულაცია

გარდა ამისა, C#-ში არსებობს სტრიქონების სპეციალური წარმოდგენა, როცა სტრიქონს წინ ეწერება სიმბოლო @ . ასეთ შემთხვევაში სტრიქონის შიგნით დახრილი ხაზი ხაზი გამოიყენება ჩვეულებრივი აზრით, რაც სტრიქონს ადვილად წასაკითხს ხდის.

მაგალითად, ავიღოთ გზა ფაილების სისტემაში c:\a\b\magaliti.txt სტრიქონში იგი ჩაიწერება ასე: “c:\\a\\b\\magaliti.txt”, ხოლო ცპეციალური ჩაწერით: @”c:\a\b\magaliti.txt”.

ოპერატორები. ოპერატორები გამოიყენებიან გამოსახულებების შესადგენად. C#-ში ასხვავებენ პირველად, ერთადგილიან და ორადგილიან ოპერატორებს იმისდამიხედვით, თუ რამდენ ოპერანდს იყენებს ოპერატორი. გამოსახულებაში, რომ განვსაზღვროთ ორადგილიანი ოპერატორების შესრულების რიგი, ამისათვის გამოიყენება ოპერატორის პრიორიტეტი და ასოციატურობა. გამოსახულებიდან აღებული ორი მეზობელი ოპერატორიდან ის სრულდება პირველად, რომელსაც უფრო მაღალი პრიორიტეტი აქვს, ხოლო ტოლი პრიორიტეტების შემთხვევაში განიხილება ასოციატურობა. თუ ოპერატორი მარჯვნივ ასოციატიურია, მაშინ პირველად სრულდება მარჯვენა ოპერატორი. თუ ოპერატორი მარცხნივ ასოციატურია, მაშინ სრულდება მარცხენა ოპერატორი. თუ ჩვენ გვინდა ორი მეზობელი ოპერატორიდან რომელიმე შესრულდეს პირველად, მაშინ მას ვსვამთ მრგვალ ფრჩხილებში თავისი ოპერანდებიანად. პირველადი და ერთადგილიანი ოპერატორები სრულდებიან მარცხნიდან მარჯვნივ რიგის მიხედვით.

ქვემოთ, მოცემულია ცხრილი, სადაც ჩამოთვლილია ოპერატორთა კატეგორიები მათ პრიორიტეტთა კლების მიხედვით და მოცემულია, აგრეთვე, მათი ასოციატურობა..

კატეგორია პრიორიტეტის მიხედვით	ოპერატორები	ასოციატურობა
პირველადი	x.y, f(x), a[x], x++, x--, new, typeof , default, checked, unchecked, delegate	მარცხნივ
ერთადგილიანი	+, - , !, ~ , ++x, --x, (T)x	მარცხნივ
მულტიპლიკატორული	*, /, %	მარცხნივ
ადიტიური	+, -	მარცხნივ

ძვრის	<<, >>	მარცხნივ
რელაციური	<, >, <=, >=, is, as	მარცხნივ
ტოლობის	=, !=	მარცხნივ
ლოგიკური და	&	მარცხნივ
ლოგიკური XOR	^	მარცხნივ
ლოგიკური ან	|	მარცხნივ
პირობითი და	&&	მარცხნივ
პირობითი ან	||	მარცხნივ
ნულ კოდი	??	მარცხნივ
ტერნალური	?:	მარჯვნივ
მინიჭების	=, *= , /=, %=, +=, -=, <<=, >>=, &=, ^=, |=, =>	მარჯვნივ

განვიხილოთ ერთადგილიანი ოპერატორების გამოყენების მაგალითი:

```
using System;

class Unary

{

    public static void Main()

    {
```

```
int unary = 0;

int preIncrement;

int preDecrement;

int postIncrement;

int postDecrement;

int positive;

int negative;

sbyte bitNot;

bool logNot;

preIncrement = ++unary;

Console.WriteLine("წინასწარ გაზრდა: {0}", preIncrement);

preDecrement = --unary;

Console.WriteLine("წინასწარ შემცირება: {0}", preDecrement);

postDecrement = unary--;

Console.WriteLine("შემდეგ შემცირება: {0}", postDecrement);

postIncrement = unary++;

Console.WriteLine("შემდეგ გაზრდა: {0}", postIncrement);

Console.WriteLine("ერთადგილიანის საბოლოო მნიშვნელობა: {0}", unary);

positive = -postIncrement;

Console.WriteLine("დადებითი: {0}", positive);

negative = +postIncrement;

Console.WriteLine("უარყოფითი: {0}", negative);
```

```
        bitNot = 0;
        bitNot = (sbyte)(~bitNot);
        Console.WriteLine("ბიტური უარყოფა: {0}", bitNot);
        logNot = false;
        logNot = !logNot;
        Console.WriteLine("ლოგიკური უარყოფა: {0}", logNot);
    }
}
```

განვიხილოთ მინიჭების ინსტრუქცია bitNot = (sbyte) (~bitNot); ამ ინსტრუქციაში ოპერატორი ~ ახდენს bitNot-ის ბიტურ უარყოფას. ამიტომ, საჭიროა მისი გარდაქმნა sbyte ტიპში, რაც სრულდება ცხადად (sbyte) გარდაქმნით და შედეგი ენიჭება bitNot-ს, ხოლო ++x ახდენს ჯერ x-ის გაზრდას ერთით და შემდეგ მის გამოყენებას. ანალოგიურად, ხდება x-- და --x-ის შემთხვევაში, მხოლოდ ერთით მცირდება x-ის მნიშვნელობა. ქვემოთ, მოცემულია პროგრამის მუშაობის შედეგი, რომ შეამოწმოთ თქვენი ცოდნა ოპერატორების მუშაობის შესახებ.

```
    წინასწარ გაზრდა:
    წინასწარ შემცირება: 0
    შემდეგ შემცირება: 0
    შემდეგ გაზრდა: -1
    ერთადგილიანის საბოლოო მნიშვნელობა: 0
    დადებითი: 1
    უარყოფითი: -1
    ბიტური უარყოფა: -1
```

ლოგიკური უარყოფა: true

შემდეგი პროგრამა გვიჩვენებს ბინარული ოპერატორების გამოყენებას.

```
using System;

class Binary
{
    public static void Main()
    {
        int x, y, result;
        float floatresult;
        x = 7;
        y = 5;
        result = x+y;
        Console.WriteLine("x+y: {0}", result);
        result = x-y;
        Console.WriteLine("x-y: {0}", result);
        result = x*y;
        Console.WriteLine("x*y: {0}", result);
        result = x/y;
        Console.WriteLine("x/y: {0}", result);
        floatresult = (float)x/(float)y;
        Console.WriteLine("x/y: {0}", floatresult);
        result = x%y;
```

```
        Console.WriteLine("x%y: {0}", result);
        result += x;
        Console.WriteLine("result+=x: {0}", result);
    }
    }
```

ეს პროგრამა შესრულების შედეგად გამოიტანს:

```
    x+y: 12
    x-y: 2
    x*y: 35
    x/y: 1
    x/y: 1.4
    x%y: 2
    result+=x: 9
```

განვიხილოთ ინსტრუქცია floatresult = (float)x/(float)y; რადგან x და y int ტიპისაა და მათზე სრულდება გაყოფის ოპერატორი, საჭიროა ისინი გადავიყვანოთ float ტიპში ცხადად. გაყოფის შედეგი იქნება float და იგი მიენიჭება floatresult-ს. ინსტრუქცია result += x არის result = result + x ინსტრუქციის შემოკლებული ჩაწერა.

მასივის ტიპი. მასივი არის ერთიდაიგივე ტიპის მნიშვნელობათა სასრული მიმდევრობა. მიმდევრობის ელემენტებისათვის მეხსიერებაში ადგილი გამოიყოფა მიმდევრობით. მნიშვნელობაზე მითითება ხდება სპეციალური წესით, რომელიც განისაზღვრება მასივის განზომილებითა და თითოეული განზომილების სიგრძით. ამგვარად, მასივი განისაზღვრება სახელით, ტიპით, განზომილებითა და თითოეული განზომილების სიგრძით. რომ გავეცნოთ მასივის განსაზღვრას, განვიხილოთ პროგრამა და მისი შესრულების შედეგი:

```
using System;

class Array

{

   public static void Main()

   {

      int[] myInts = { 5, 10, 15 };

      bool[][] myBools = new bool[2][];

      myBools[0] = new bool[2];

      myBools[1] = new bool[1];

      double[,] myDoubles = new double[2, 2];

      string[] myStrings = new string[3];

      Console.WriteLine("myInts[0]: {0}, myInts[1]: {1}, myInts[2]: {2}", myInts[0], myInts[1],
myInts[2]);

      myBools[0][0] = true;

      myBools[0][1] = false;

      myBools[1][0] = true;

      Console.WriteLine("myBools[0][0]: {0}, myBools[1][0]: {1}", myBools[0][0],
myBools[1][0]);

      myDoubles[0, 0] = 3.147;

      myDoubles[0, 1] = 7.157;

      myDoubles[1, 1] = 2.117;

      myDoubles[1, 0] = 56.00138917;
```

```
        Console.WriteLine("myDoubles[0, 0]: {0}, myDoubles[1, 0]: {1}", myDoubles[0, 0],
myDoubles[1, 0]);

        myStrings[0] = "Joe";

        myStrings[1] = "Matt";

        myStrings[2] = "Robert";

        Console.WriteLine("myStrings[0]: {0}, myStrings[1]: {1}, myStrings[2]: {2}", myStrings[0],
myStrings[1], myStrings[2]);

    }

}
```

ამ პროგრამის შესრულების შედეგია:

myInts[0]: 5, myInts[1]: 10, myInts[2]: 15

myBools[0][0]: true, myBools[1][0]: true

myDoubles[0, 0]: 3.147, myDoubles[1, 0]: 56.00138917

myStrings[0]: Joe, myStrings[1]: Matt, myStrings[2]: Robert

ავიღოთ ინსტრუქცია: int[] myInts = { 5, 10, 15 } იგი ახდენს myInts ცვლადის გამოცხადებას როგორც მთელი ტიპის ერთგანზომილებიანი მასივის და ამავე დროს ახდენს მის ინიციალიზაციას. მასივის ელემენტების გადანომვრა იწყება ნულიდან. myInts[0] იქნება 5 და ასე შემდეგ. ინსტრუქცია double[,] myDoubles = new double[2, 2] ახდენს myDoubles როგორც double ტიპის ორგანზომილებიანი მასივის გამოცხადებას. new ოპერატორი აძლევს სიგრძეს თითოეულ განზომილებას. მასივის ელემენტებს მნიშვნელობები ენიჭებათ ინსრუქციებით:

myDoubles[0, 0] = 3.147;

myDoubles[0, 1] = 7.157;

myDoubles[1, 1] = 2.117;

C#-ში არსებობს მეორე სახის მასივები, რომელთაც ჰქვიათ დანაწევრებული მასივები. მაგალითად, bool[][] myBools = new bool[2][] არის ორგანზომილებიანი დანაწევრებული bool ტიპის მასივი. რადგანაც, მეორე განზომილების ელემენტები განიხილებიან როგორც ვექტორები და თითოეულს შეგვიძლია მივცეთ სხვადასხვა სიგრძე new ოპერატორით:

```
myBools[0] = new bool[2];

 myBools[1] = new bool[1];
```

ამ ვექტორებს მნიშვნელობები მიენიჭებათ ასე:

```
myBools[0][0] = true;

myBools[0][1] = false;

myBools[1][0] = true;
```

ამგვარად, ამ პარაგრაფში წარმოდგენილია მარტივი ტიპის ცვლადები, ოპერატორები და მასივები - თუ როგორ გამოიყენებიან ისინი C# პროგრამაში.

1.3 პროგრამის მართვის ინსტრუქციები. ამ პარაგრაფში აღწერილია პროგრამის მართვის ინსტრუქციები და მოცემულია მათი გამოყენების მაგალითები:

if ინსტრუქცია. if ინსტრუქცია საშუალებას გვაძლევს მოვახდინოთ პროგრამის შესრულების განშტოება გარკვეული პირობის მიხედვით. განვიხილოთ მაგალითი:

მაგალითი 3.1

```
using System;

class IfSelect

{

   public static void Main()

   {

      string myInput;

      int myInt;

      Console.Write("გთხოვთ, შეიტანოთ რაიმე რიცხვი: ");

      myInput = Console.ReadLine();

      myInt = Int32.Parse(myInput);

      // კონკრეტული გადაწყვეტილება და ბლოკის შესრულება

      if (myInt > 0)

      {

         Console.WriteLine("თქვენი რიცხვი {0} მეტია ნულზე.", myInt);

      }

      // კონკრეტული გადაწყვეტილება და ერთი ინსტრუქციის  შესრულება

      if (myInt < 0)
```

```
    Console.WriteLine("თქვენი რიცხვი {0} ნულზე ნაკლებია.", myInt);
// ტოლია თუ არა
if (myInt != 0)
{
    Console.WriteLine("თქვენი რიცხვი {0} არ უდრის ნულს.", myInt);
}
else
{
    Console.WriteLine("თქვენი რიცხვი {0} ნულის ტოლია.", myInt);
}
// მრავალი ალტერნატივა
if (myInt < 0 || myInt == 0)
{
    Console.WriteLine("თქვენი რიცხვი {0} ნაკლებია ან ტოლი ნულზე.", myInt);
}
else if (myInt > 0 && myInt <= 10)
{
    Console.WriteLine("თქვენი რიცხვი  {0} არის 0-ზე მეტი და ნაკლებია ან ტოლი 10-
ზე.", myInt);
}
else if (myInt > 10 && myInt <= 20)
{
```

```
        Console.WriteLine("თქვენი რიცხვი  {0} არის 10-ზე მეტი და ნაკლებია ან ტოლი 20-ზე.", myInt);
    }
    else if (myInt > 20 && myInt <= 30)
    {
        Console.WriteLine("თქვენი რიცხვი  {0} არის 20-ზე მეტი და ნაკლებია ან ტოლი 30-ზე.", myInt);
    }
    else
    {
        Console.WriteLine("თქვენი რიცხვი  {0} არის 30-ზე მეტი.", myInt);
    }
  }
}
```

if ინსტრუქციას აქვს შემდეგი სახე: გასაღები სიტყვა if შემდეგ (<პირობა>), ბლოკი შემდეგ შეიძლება იყოს გასაღები სიტყვა else და ბლოკი ან elseif და იგივე მეორდება, რაც if შემთხვევაში, მხოლოდ elseif შეიძლება გამეორდეს მრავალჯერ და ასეთი ინსტრუქცია მთავრდება else და ბლოკით. ბლოკი არის ფიგურულ ფრჩხილებში მოთავსებული ერთი ან მეტი ინსტრუქცია. იგი გამოიყენება როცა გვსურს ინსტრუქციის ადგილას დავწეროთ ერთზე მეტი ინსტრუქცია. if ინსტრუქციის განმარტებაში, სადაც გვაქვს ბლოკი შეიძლება შეიცვალოს ინსტრუქციით. <პირობა> არის ლოგიკური გამოსახულება, რომლის მნიშვნელობა შეიძლება იყოს true ან false. true შემთხვევაში სრულდება (<პირობა>)-ს შემდეგ მოთავსებულია ბლოკი. false-ს შემთხვევაში, სრულდება ბლოკის შემდეგ მოთავსებული ინსტრუქცია. განვიხილოთ მაგალითი:

```
myInput = Console.ReadLine();

myInt = Int32.Parse(myInput);
```

პირველი ინსტრუქცია მოითხოვს კლავიატურიდან ტექსტის შეტანას. შევიტანოთ: გთხოვთ შეიტანოთ რაიმე რიცხვი და დავაჭიროთ enter კლავიშას. შედეგად myInput-ს მიენიჭება აკრეფილი სტრიქონი. შემდეგ ინსტრუქციას აკრეფილი სტრიქონი გადაჰყავს მთელ რიცხვადა და ენიჭება myInt-ს. Int32.Parse ფუნქცია ასრულებს ამ გარდაქმნას.

switch(გადამრთველი) ინსტრუქცია. გადამრთველი გამოიყენება, როცა ჩვენ გვსურს რაიმე ცვლადის მნიშვნელობის მიხედვით მოვახდინოთ პროგრამის განშტოება. გადამრთველს აქვს შემდეგი სახე:

```
switch(<ცვლადის სახელი>)

{

case <მნიშვნელობა1> :

<ინსტრუქცია1>

break;

case <მნიშვნელობა2> :

<ინსტრუქცია2>

...

case <მნიშვნელობაn>

<ინსტრუქციაn>

default:

<ინსტრუქცია>

}
```

თუ ცვლადის მნიშვნელობა დაემთხვევა <მნიშვნელობაi>-ს, მაშინ სრულდება <instruqciai> და გადამრთველი ამთავრებს მოშაობას. თუ ცვლადის მნიშვნელობა არ დაემთხვევა არცერთ მნიშვნელობას გადამრთველი ამთავრებს მუშაობას. განვიხილოთ მაგალითი:

```
using System;

class SwitchSelect

{

   public static void Main()

   {

      string myInput;

      int myInt;

      begin:

      Console.Write("გთხოვთ შეიტანოთ რაიმე რიცხვი  1-დან 3-მდე ");

      myInput = Console.ReadLine();

      myInt = Int32.Parse(myInput);

      // გადამრთველი მთელი მნიშვნელობით

      switch (myInt)

      {

         case 1:

            Console.WriteLine("თქვენი რიცხვია {0}.", myInt);

            break;

         case 2:

            Console.WriteLine("("თქვენი რიცხვია {0}.", myInt);
```

```
            break;
        case 3:
            Console.WriteLine("("თქვენი რიცხვია {0}.", myInt);
            break;
        default:
            Console.WriteLine("("თქვენი რიცხვი არაა ერთსა და სამს შორის {0}", myInt);
            break;
    }
    decide:
    Console.Write("აკრიფეთ \"continue\" რომ გააგრძელოთ ან \"quit\" რომ გააჩეროთ პროგრამის შესრულება");        myInput = Console.ReadLine();
    // გადამრთველი სტრიქონული ტიპის მნიშვნელობით
    switch (myInput)
    {
        case "continue":
            goto begin;
        case "quit":
            Console.WriteLine("Bye.");
            break;
        default:
            Console.WriteLine("თქვენი შეტანა არაა სწორი {0} ", myInput);
            goto decide;
```

```
        }
    }
}
```

აქ განხილულია გადამრთველი მთელი და სტრიქონული ტიპის მნიშვნელობებით.

განშტოების ინსტრუქციები

განშტოების ინსტრუქცია	აღწერა
break	გამოდის გადამრთველის ბლოკიდან
continue	აგრძელებს ციკლს, ციკლის პარამეტრის შემდეგი მნიშვნელობისათვის, თუ ციკლის პარამეტრის მნიშვნელობა არ აღემატება ციკლის ბოლო მნიშვნელობას. წინააღმდეგ შემთხვევაში ამთავრებს ციკლს.
goto	გადადის იმ ჭდეზე, რომელიც წერია goto ინსტრუქციაში
return	ამთავრებს მიმ დინარე მეთოდს.
throw	გამოაქვს შეცდომის შეტყობინება.

1.4 ციკლის ინსტრუქციები

while ინსტრუქცია. while ინსტრუქციის სინტაქსია:

while (<პირობითი გამოსახულება>) {<ინსტრუქციები>},

ხოლო while ინსტრუქციის შესრულება ხდება შემდეგნაირად: გამოითვლება <პირობითი გამოსახულება >, თუ მისი მნიშვნელობაა true, მაშინ შესრულდება ფიგურულ ფრჩხილებში მოთავსებული ინსტრუქციები და კვლავ გამოითვლება <პირობითი გამოსახულება >. თუ მისი მნიშვნელობაა false, მთავრდება while ინსტრუქცია. განვიხილოთ მაგალითი:

```
using System;

class WhileLoop

{

   public static void Main()

   {

      int myInt = 0;

      while (myInt < 10)

      {

         Console.Write("{0} ", myInt);

         myInt++;

      }

      Console.WriteLine();

   }

}
```

ამ პროგრამაში myInt-ს ენიჭება ნული და ედარება 10-ს, რადგან myInt ნაკლებია 10-ზე, პირობით გამისახულებას ენიჭება true და ეკრანზე გამოიტანება myInt-ის მნიშვნელობა - ნული. შემდეგ myInt-ის მნიშვნელობა იზრდება ერთით და პროცესი მეორდება, სანამ myInt არ გახდება 10. ამ დროს პირობითი გამოსახულება მიიღებს მნიშვნელობას false და ციკლი მთავრდება. ეკრანზე გამოიტანება ცარიელი სტრიქონი და პროგრამა ამთავრებს მუშაობას.

do ინსტრუქცია. მისი სინტაქსია: do {<ინსტრუქციები>} while (<პირობითი გამოსახულება>). ამ შემთხვევაში ჯერ სრულდება <instruqciebi> და შემდეგ ხდება <პირობითი გამოსახულება>-ს შემოწმება. ამით განსხვავდება იგი while ინსტრუქციისაგან. განვიხილოთ მაგალითი:

```
using System;

class DoLoop

{

   public static void Main()

   {

      string myChoice;

      do

     {

         //menu-ს გამოტანა

         Console.WriteLine("ჩემი მისამართების წიგნი \n");

         Console.WriteLine("A – დაამატე ახალი მისამართი");

         Console.WriteLine("D – წაშალე მისამართი");

         Console.WriteLine("M – შეასწორე მისამართი");

         Console.WriteLine("V – გადაათვალიერე მისამართები");
```

```
        Console.WriteLine("Q – გამოსვლა მისამართების წიგნიდან\n");
        Console.WriteLine("აირჩიე(A,D,M,V,ან Q): ");
        // მომხმარებლის არჩევანის აღება
        myChoice = Console.ReadLine();
        // გადაწყვეტილების მიღება მომხმარებლის არჩევანის მიხედვით
switch(myChoice)
        {
           case "A":
           case "a":
              Console.WriteLine("ახალი მისამართის დამატება.");
              break;
           case "D":
           case "d":
              Console.WriteLine("მასამართის წაშლა.");
              break;
           case "M":
           case "m":
              Console.WriteLine("მისამართის შესწორება.");
              break;
           case "V":
           case "v":
              Console.WriteLine("მისამართების გადათვალიერება.");
```

```
                break;
            case "Q":
            case "q":
                Console.WriteLine("დამთავრება“);
                break;
            default:
                Console.WriteLine("{0} არასწორი არჩევანი", myChoice);
                break;
        }
        // პაუზა, რომ მომხმარებელმა ნახოს შედეგები
        Console.Write("დააჭირეთ Enter-ს, რომ გააგრძელოთ...");
        Console.ReadLine();
        Console.WriteLine();
    } while (myChoice != "Q" && myChoice != "q");
// გაგრძელება
  }
}
```

ამ პროგრამაში მომხმარებელი კონკრეტული კოდის მიხედვით ირჩევს გარკვეულ მოქმედებას მისამართების წიგნში.

for ინსტრუქცია. for ინსტრუქციის სინტაქსია:

for (<ინსტრუქცია>; <პირობითი გამოსახულება>; <საიტერაციო სია>)
{<ინსტრუქციები>}

<ინსტრუქცია> განსაზღვრავს ციკლის პარამეტრს - მის ტიპსა და საწყის მნიშვნელობას. შემდეგ გამოითვლება <პირობითი გამოსახულება>. თუ მისი მნინელობააშ true სრულდება <ინსტრუქციები> და <საიტერაციო სია>-დან განისაზღვრება ციკლის პარამეტრის ახალი მნიშვნელობა. ეს პროცესი მეორდება მანამ, სანამ <პირობითი გამოსახულება> არ მიიღებს false მნიშვნელობას, რაც ამთავრებს ციკლს. განვიხილოთ მაგალითი:

```
using System

class ForLoop

{

   public static void Main()

   {

      for (int i = 0; i < 20; i++)

      {

         if (i == 10)

            break;

         if (i % 2 == 0)

            continue;

         Console.Write("{0} ", i);

      }

      Console.WriteLine();

   }

}
```

ამ პროგრამას გამოაქვს ეკრანზე i-ს კენტი მნიშვნელობები 1-დან 10-მდე. თუ i=10, პროგრამა ამთავრებს ციკლს, ხოლო თუ i კენტია, ეკრანზე გამოაქვს მისი მნიშვნელობა და განიხილება ციკლის პარამეტრის შემდეგი შვნელობა. თუ i ლუწია, განიხილება ციკლის პარამეტრის შემდეგი მნიშვნელობა. ინსტრუქცია break ამთავრებს ციკლს, ხოლო ინსტრუქცია continue აგრძელებს ციკლს ციკლის პარამეტრის შემდეგი მნიშვნელობისათვის.

forich ინსტრუქცია. მისი სინტაქსია:

forich (<ტიპი> <საიტერაციო ცვლადი> in <სია>) {<ინსტრუქციები>}

<ტიპი> მიუთითებს <სია>-ს ელემენტების ტიპს. <საიტერაციო ცვლადი> არის ციკლის პარამეტრი, რომელიც მნიშვნელობებს იღებს <sia>-დან. თუ <სია>-ს ტიპია int[], მაშინ <ტიპი>=int. <საიტერაციო ცვლადი> არის ცვლადის სახელი, რომელიც მნიშვნელობებს იღებს <sia>-დან. ცვლადის ყოველი შვნელობისათვის <სია>-დან სრულდება <ინსტრუქციები>, სანამ არ ამოიწურება <sia>. განვიხილოთ მაგალითი:

```
using System;

class ForEachLoop

{

   public static void Main()

   {

      string[] names = { "Cheryl", "Joe", "Matt", "Robert" };

      foreach (string person in names)

      {

         Console.WriteLine("{0} ", person);

      }

   }
```

```
}
```

ეს პროგრამა გამოიტანს ეკრანზე names მნიშვნელობებს და ამთავრებს მუშაობას.

ამგვარად, ჩვენ ამ პარაგრაფში განვიხილეთ ციკლის ინსტრუქციების ყველა სახე.

1.5 მეთოდები

მეთოდები საშუალებას გვაძლევენ გამოვყოთ პროგრამის ნაწილი, როგორც დამოუკიდებელი ერთეული და მას მივმართოთ პროგრამის სხვადასხვა ადგილიდან. ასევე, შესაძლებელია მისი ტექსტის მოდიფიკაცია. მისი სინტაქსია: <ატრიბუტები> <მოდიფიკატორები> <დასაბრუნებელი მნიშვნელობის ტიპი> <მეთოდის სახელი>(<პარამეტრები>) {<ინსტრუქციები>}

<ატრიბუტები>-ს და <მოდიფიკაორები>-ს განვიხილავთ მოგვიანებით.

<დასაბრუნებელი მნიშვნელობის ტიპი> შეიძლება იყოს c#-ის ნებისმიერი ტიპი, რომელიც აღნიშნავს მეთოდის დასაბრუნებელი მნიშვნელობის ტიპს. <მეთოდის სახელი> არის უნიკალური იდენტიფიკატორი, რომელიც ცალსახად განსაზღვრავს მეთოდს. <პარამეტრები> განსაზღვრავენ მეთოდისათვის გადასაცემ და დასაბრუნებელ ინფორმაციას. <ინსტრუქციები> განსაზღვრავენ იმ მოქმედებებს, რომლებიც უნდა შეასრულოს მეთოდმა. განვიხილოთ მარტივი მეganvixiloT martivi meTodisთოდის მაგალითი.

```
using System;
class OneMethod
{
   public static void Main()
   {
      string myChoice;

OneMethod om = new OneMethod();
      do
     {
         myChoice = om.getChoice();
         // არჩევანი მომხმარებლის მიერ
         switch(myChoice)
         {
            case "A":
            case "a":
```

```
                Console.WriteLine("თქვენ გსურთ დაუმატოთ მისამართი.");
                break;
            case "D":
            case "d":

Console.WriteLine(“თქვენ გსურთ მისამართის წაშლა");
                break;
            case "M":
            case "m":
                Console.WriteLine(“თქვენ გსურთ მისამართის შესწორება.");
                break;
            case "V":
            case "v":
                Console.WriteLine(“თქვენ გსურთ მისამართების დათვალიერება.");
                break;
            case "Q":
            case "q":

                Console.WriteLine(“დამთავრება.");
                break;
            default:
                Console.WriteLine("{0} araswori arCevani", myChoice);
                break;
        }
        // პაუზა შედეგის დასათვალიერებლად

    Console.WriteLine();
        Console.Write(“დააჭირეთ enter-ს მუშაობის გასაგრძელებლად ...");
        Console.ReadLine();
        Console.WriteLine();

} while (myChoice != "Q" && myChoice != "q");

// მუშაობის გაგრძელება, სანამ მომხმარებელი
```

```
//არ დაამთავრებს მუშაობას

}
   string getChoice() {
       string myChoice;
       // მენიუს გამოტანა
       Console.WriteLine(“ჩემი მისამართების წიგნი\n");
       Console.WriteLine("A – ახალი მისამართის დამატება");
 Console.WriteLine("D – მისამართის წაშლა");
Console.WriteLine("M – მისამართის შესწორება");
Console.WriteLine("V  - მისამართების დათვალიერება");
       Console.WriteLine("Q - დამთავრება\n");

Console.Write(“აირჩიეთ (A,D,M,V,ან Q): ");
       // მომხმარებლის არჩევანის აღება
       myChoice = Console.ReadLine();
       Console.WriteLine();


       return myChoice;
   }
}
```

ამ პროგრამაში ძირითადი კლასია OneMethod, რომელში განსაზღვრულია სტატიკური main მეთოდი და getChoice მეთოდი. main მეთოდში, რომ გამოვიყენოთ getChoice მეთოდი, ამისათვის საჭიროა შევქმნათ OneMethod კლასის ობიექტი და გამოვიძახოთ ამ ობიექტის getChoice მეთოდი, რადგან იგი არაა სტატიკური მეთოდი, ამიტომ პირდაპირ ვერ გამოვიძახებთ OneMethod კლასიდან. OneMethod კლასის om ობიექტი იქმნება ინსტრუქციით:

```
OneMethod om = new OneMethod();
```

აქ OneMethod არის ტიპი და OneMethod() არის კლასის კონსტრუქტორი (მეთოდი, რომელსაც იგივე სახელი აქვს, რაც კლასს). ამის შემდეგ om ობიექტის getChoice მეთოდი გამოიძახება om.getChoice()-ით. ეს მეთოდი აბრუნებს მომხმარებლის არჩევანს.

სხვა მხრივ, ეს პროგრამა იგივეა, რაც ადრინდელი პროგრამა. ახლა, განვიხილოთ იგივე ამოცანის პროგრამა მეთოდის პარამეტრების გამოყენებით.

მენიუს ჩვენება და მომხმარებლის არჩევანი.

```
using System;
class Address
{
   public string name;
   public string address;
}
class MethodParams
{
   public static void Main()
   {
      string myChoice;
      MethodParams mp = new MethodParams();
      do
     {
         // მენიუს ჩვენება და მომხმარებლის არჩევანი.
         myChoice = mp.getChoice();

// მომხმარებლის არჩევანის შესრულება
         mp.makeDecision(myChoice);
         // პაუზა შედეგების დასათვალიერებლად
         Console.Write("დააჭირეთ Enter-ს გაგრძელებისათვის");
         Console.ReadLine();
         Console.WriteLine();
      } while (myChoice != "Q" && myChoice != "q");

 // გაგრძელება დამთავრებამდე
   }
   // მენიუს ჩვენება და მომხმარებლის არჩევანი
```

```
    string getChoice()
    {
        string myChoice;
        // მენიუს გამოტანა
        Console.WriteLine("ჩემი მისამართების წიგნი");

Console.WriteLine("A – ახალი მისამართის დამატება");
        Console.WriteLine("D – მისამართის წაშლა");
        Console.WriteLine("M – მისამართის შესწორება");
        Console.WriteLine("V – მისამართების დათვალიერება");
        Console.WriteLine("Q - დამთავრება\n");
        Console.WriteLine("აირჩიეთ (A,D,M,V,ან Q): ");
        // არჩევანის დაფიქსირება
        myChoice = Console.ReadLine();
        return myChoice;
    }
    // გადაწყვეტილების შესრულება
    void makeDecision(string myChoice)
    {
        Address addr = new Address();

switch(myChoice)
        {
            case "A":
            case "a":
                addr.name = "Joe";
                addr.address = "C# Station";
                this.addAddress(ref addr);
                break;
            case "D":
            case "d":
                addr.name = "Robert";
```

```
            this.deleteAddress(addr.name);
            break;
        case "M":
        case "m":
            addr.name = "Matt";
            this.modifyAddress(out addr);
 Console.WriteLine(“saxelia {0}.", addr.name);
            break;
        case "V":
        case "v":
            this.viewAddresses("Cheryl", "Joe", "Matt", "Robert");
            break;
        case "Q":
        case "q":
            Console.WriteLine(“დამთავრება.");
            break;
        default:
            Console.WriteLine("{0} არაა სწორი არჩევანი", myChoice);
            break;
      }
   }

// მისამართის დამატება
   void addAddress(ref Address addr)
   {
      Console.WriteLine(“სახელი: {0}, მისამართი: {1} დაემატა.", addr.name, addr.address);   }
   // მისამართის წაშლა
   void deleteAddress(string name)
   {
      Console.WriteLine(“თქვენ გსურთ {0}-ის მისამართის წაშლა.", name);
   }
```

```
    // მისამართის შეცვლა
    void modifyAddress(out Address addr)
    {
        //Console.WriteLine("სახელი: {0}.", addr.name);
//იწვევს შეცდომას!
        addr = new Address();
        addr.name = "Joe";
        addr.address = "C# Station";
    }
    // მისამართების დათვალიერება
    void viewAddresses(params string[] names)
{
        foreach (string name in names)
        {
            Console.WriteLine("სახელი: {0}", name);
        }
    }
}
```

ეს პროგრამა იმავე ამოცანის რეალიზაციას ახდენს რაც წინა პროგრამა, მაგრამ განსხვავდება ინსტრუქციების გამოყენებით. აქ გამოიყენება პარამეტრიანი მეთოდი და პარამეტრის სხვადასხვა სახეები. პარამეტრი შეიძლება იყოს მნიშვნელობა. ასეთი პარამეტრის შესატყვის არგუმენტს უნდა ჰქონდეს მხოლოდ მნიშვნელობა. პარამეტრს შეიძლება ჰქონდეს მოდიფიკატორი out. ასეთ შემთხვევაში შესტყვის არგუმენტს არ უნდა ჰქონდეს მნიშვნელობა. ასეთი არგუმენტის მნიშვნელობა უნდა განისაზღვროს მეთოდის ტანში და იგი უბრუნდება გამომძახებელ პროგრამას, სადაც შეიძლება ამ მნიშვნელობის გამოყენება. პარამეტრს შეიძლება ჰქონდეს მოდიფიკატორი ref, რაც ნიშნავს, რომ შესატყვისი არგუმენტის მნიშვნელობა უნდა იყოს მისამართი. ან კიდევ პარამეტრს შეიძლება ჰქონდეს მოდიფიკატორი params, რაც ნიშნავს, რომ შესატყვისი

არგუმენტი არის ცვალებადი სიგრძის ერთგანზომილებიანი მასივი. ამ პროგრამაში მოცემულია ყველა ჩამოთვლილი სახის პარამეტრების გამოყენება. მაგალითად,

```
    void addAddress(ref Address addr)
```

aq, Cven gvaqvs addr aris Address tipis parametri modifikatoriT ref.

```
void modifyAddress(out Address addr)
```

აქ, პარამეტრი არის out მოდიფიკატორით. ახლა განვიხილოთ

```
void viewAddresses(params string[] names)
```

აქ, პარამეტრად გვაქვს names, რომელიც არის ერთგანზომილებიანი სტრიქონის ტიპის მასივი მოდიფიკატორით params. ეს ნიშნავს, რომ შესტყვის არგუმენტად შეიძლება ავიღოთ სტრიქონული ტიპის ერთგანზომილებიანი ნებისმიერი სიგრძის მასივი.

1.6 სახელთა არეები (Namespaces). სახელთა არეები გამოიყენება იმისათვის, რომ ზუსტად მივუთითოთ კონკრეტული ობიექტი, როცა განსხვავებულ ობიექტებს აქვთ ერთიდაიგივე სახელი. ამას ვახერხებთ ამ ობიექტების სხვადასხვა სახელთა არეში მოთავსებით. კონკრეტული ობიექტის მითითება ხდება ობიექტის სახელზე სახელთაარის სახელის დამატებით. თუ როგორ ხდება სახელთა არეს შემოტანა, ამისათვის განვიხილოთ მაგალითი: // Namespace-ის გამოძახება

```
using System;
namespace csharp_station
{
   class NamespaceCSS
   {
      public static void Main()
      {
         Console.WriteLine("ეს არის ახალი C# Namespace.");

  }
 }
}
```

სახელთა არე განისაზღვრება გასღები სიტყვით namespace, რომელსაც მოსდევს სახელი და გახსნილი და დახურული ფრჩხილები. ფრჩხილებს შორის მოთავსებული ტექსტი ეკუთვნის სახელთა არეს. ამ მაგალითში შემოტანილია csharp_station სახელთა არე, რომელსაც ეკუთვნის კლასი NamespaceCSS. სახელთა არეები შეიძლება იყოს ერთიმეორეში ჩადგმული. განვიხილოთ მაგალითი:

```
using System;
namespace csharp_station
{
   namespace tutorial
   {
      class NamespaceCSS
      {
         public static void Main()
```

```
        {
                Console.WriteLine("ეს არის სახელთა არეს მაგალითი.");
        }
      }
   }
}
```

ეს მაგალითი შეიძლება გადაიწეროს ასეც:

```
using System;
namespace csharp_station.tutorial
{
   class NamespaceCSS
   {
      public static void Main()
      {
         Console.WriteLine("ეს არის ახალი Namespace.");
      }
   }
;
```

ინსტრუქცია

```
namespace csharp_station.tutorial
```

აღნიშნავს ჩადგმულ სახელთა არეებს. წერტილი მიუთითებს, რომ არეები ერთიმეორეშია ჩადგმული. ახლა, განვიხილოთ სახელთა არეებიდან წევრების გამოძახების მაგალითი:

```
using System;
namespace csharp_station
{
   // ჩადგმული namespace
   namespace tutorial
   {
```

```
    class myExample1
    {
        public static void myPrint1()
        {
            Console.WriteLine("სახელთა არეს წევრის გამოყენების მაგალითი. ");
        }
    }
  }

class NamespaceCalling
  {
    public static void Main()
    {
        tutorial.myExample1.myPrint1();
        tutorial.myExample2.myPrint2();
    }
  }
}
// namespace როგორც ზემოთ ჩადგმული  namespace
namespace csharp_station.tutorial
{
  class myExample2

public static void myPrint2()
    {
        Console.WriteLine("სახელთა არეს წევრის გამოძახების მეორე მაგალითი.);
    }
  }
}
```

ახლა, განვიხილოთ using დირექტივის გამოყენების მაგალითი:

```
using System;
using csharp_station.tutorial;
```

```
class UsingDirective
{
   public static void Main()
   {
      // namespace –ის წევრის გამოყენება
      myExample.myPrint();
   }
}

namespace csharp_station.tutorial
{
   class myExample
   {
      public static void myPrint()
      {
         Console.WriteLine(“using დირექტივის გამოყენების მაგალითი.");
     }
   }
}
```

ამ მაგალითში დირექტივას

```
 using csharp_station.tutorial;
```

შემოჰყავს პროგრამაში სახელთა არე csharp_station.tutorial. ახლა, განვიხილოთ თუ როგორ შევამოკლოთ სახელთა არეს გრძელი მითითება:

```
using System;
using csTut = csharp_station.tutorial.myExample;

// მეტსახელი


class  AliasDirective
{
   public static void Main()
```

```
    {
        // namespace –ის გამოძახება
        csTut.myPrint();
        myPrint();
    }
// შესაძლო ორაზროვანი მეთოდი
    static void myPrint()
    {
        Console.WriteLine("არაა წევრი csharp_station.tutorial.myExample.");
    }
}

namespace csharp_station.tutorial
{
    class myExample
    {
        public static void myPrint()
        {
            Console.WriteLine("ეს არის წევრი -  csharp_station.tutorial.myExample.");
        }}}
```

ინსტრუქციას

```
using csTut = csharp_station.tutorial.myExample;
```

შემოჰყავს შემამოკლებელი სახელი სახელთა არესათვის
csharp_station.tutorial.myExample

სახელთა არეს წევრები შეიძლება იყოს:

კლასები

სტრუქტურები

ინტერფეისები

გადანომვრა

დელეგატები

2 თავი. კლასები და ობიექტები

2.1 კლასები. ჩვენ აქამდე მრავალჯერ გამოვიყენეთ კლასები და ვიცით ზოგი რამ კლასების შესახებ. ახლა, სრულად განვიხილავთ კლასებს. კლასის განსაზღვრა იწყება გასაღები სიტყვით **class**, რომელსაც მოსდევს კლასის სახელი და ფიგურულ ფრჩხილებში მოთავსებული კლასის წევრები. ყოველ კლასს აქვს კონსტრუქტორი, რომელიც არის მეთოდი და აქვს სახელად კლასის სახელი. მას არა აქვს დასაბრუნებელი მნიშვნელობა. იგი გამოიძახება ავტომატურად ყოველთვის, როცა ხდება კლასის ეგზემპლარის შექმნა. დამატებით ჩვენ შეგვიძლია კლასის შიგნით განვსაზღვროთ მრავალი კონსტრუქტორი და გამოვიყენოთ ნებისმიერი მათგანი კლასის ეგზემპლარის შექმნის დროს. კონსტრუქტორები განსხვავდებიან ერთიმეორისაგან პარამეტრებით. კონსტრუქტორის დანიშნულებაა მოახდინოს კლასის წევრების ინიციალიზაცია კლასის ეგზემპლარის შექმნისას. კლასს აქვს აგრეთვე დესტრუქტორები. დესტრუქტორი არის მეთოდი, რომლის სახელია კლასის სახელს წინდართული სიმბოლო ~. იგი გამოიყენება კლასის დახურვის დროს, რომ გავათავისუფლოთ კლასის მიერ დაკავებული რესურსები. კლასს შეიძლება ჰქონდეს მრავალი დესტრუქტორი. ისინი ერთიმეორესგან განსხვავდებიან პარამეტრებით.განვიხილოთმაგალითი:

```
using System;

// დამხმარე კლასი

class OutputClass

{

   string myString;

   // კონსტრუქტორი

   public OutputClass(string inputString)

   {

      myString = inputString;
```

```
    }
    // საობიექტო მეთოდი
    public void printString()
    {
        Console.WriteLine("{0}", myString);
    }
    // დესტრუქტორი
    ~OutputClass()
    {
        // ზოგიერთი რესურსის განთავისუფლება
    }
}
class ExampleClass
{
    public static void Main()
    {
        // OutputClass-ის  ეგზემპლიარი
        OutputClass outCl = new OutputClass("OutputClass-ით გამოიტანება ეს");
        // OutputClass-ის მეთოდის გამოძახება
        outCl.printString();
    }
}
```

ამ პროგრამაში მოცემულია კონსტრუქტორის და დესტრუქტორის მაგალითი. კონსტრუქტორი ინიციალიზაციას უკეთებს კლასის ელემენტს myString-ს. ასევე დესტრუქტორის ტანში შეიძლება იყოს რაიმე რესურსის გათავისუფლება. ამ პროგრამაში იქმნება OutputClass-ის ობიექტი outCl და printString მეთოდს გამოაქვს ეკრანზე myString-ის მნიშვნელობა. კონსტრუქტორებისა და დესტრუქტორების შექმნა პროგრამაში აუცილებელი არ არის. ასეთ შემთხვევაში ასეთ შემთხვევაში ავტომატურად იქმნება გაჩუმების პრინციპით კლასში ერთი კონსტრუქტორი და დესტრუქტორი პარამეტრების გარეშე. ეს კონსრუქტორი გამოიყენება ავტომატურად ობიექტის შექმნისას და ობიექტის მიერ დაკავებული ადგილის გასათავისუფლებლად ნაგავის შემგროვებლის მიერ. ასეთი კონსტრუქტორები არ არიან დიდად სასარგებლო. რომ გავხადოთ კონსტრუქტორები უფრო სასარგებლო, ჩვენ თვითონ შევქმნათ ისინი მაინიციალიზირებელის გამოყენებით:

public OutputClass() : this("Default Constructor String") { }

აქ, კონსტრუქტორის შექმნა ხდება მაინიციალიზირებელის საშუალებით. მისი გამოძახება ხდება :-ით. This უთითებს მიმდინარე ობიექტს. ფრჩხილებში მოთავსებული სტრიქონი არის კონსტრუქტორის არგუმენტი, რომელიც ინიციალიზაციას უკეთებს ობიექტის ველს. ასეთი გზით შესაძლებელია სხვადასხვა კონსტრუქტორების გამოძახება, თუ რომელი კონსტრუქტორი იქნება გამოძახებული, დამოკიდებულია პარამეტრების ტიპსა და რაოდენობაზე. კლასის წევრები შეიძლება იყვნენ ეგზემპლიარები ან სტატიკურები. სტატიკური წევრები გამოიძახებიან ასე: <კლასის-სახელი> . <სტატიკური წევრი >. ვთქვათ, OutputClass შეიცავს სტატიკურ მეთოდს

```
public static void staticPrinter()
   {
      Console.WriteLine("There is only one of me.");
   }
```

მაშინ, იგი შეიძლება გამოვიძახოთ main() მეთოდიდან ასე:

```
OutputClass.staticPrinter();
```

შეიძლება გვქონდეს სტსტიკური კონსტრუქტორი. იგი გამოიყენება სტატიკური ველის ინიციალიზაციისათვის. იგი უნდა იყოს გამოძახებული ობიექტის შექმნამდე და ველის გამოყენებამდე.

კლასის წევრები შეიძლება იყვნენ:

ჩადგმული კლასები

ველები

მეთოდები

კონსტრუქტორები

დესტრუქტორები

მაინდექსირებლები

თვისებები

ხდომილებები

დელეგატები

2.2 მემკვიდრეობითობა. ამ ლექციაში შეისწავლით როგორ შექმნათ ძირითადი და წარმოებული კლასები, როგორ მოვახდინოთ წარმოებული კლასიდან ძირითადი კლასის ინიციალიზაცია, როგორ გამოიძახოთ და დამალოთ წარმოებული კლასიდან ძირითადი კლასი.

მემკვიდრეობითობა არის ობიექტზე ორიენტირებული დაპროგრამების ძირითადი თვისება, რომელიც საშუალებას გვაძლევს ხელმეორედ გამოვიყენოთ უკვე შექმნილი კოდი და ამით დავზოგოთ პრობლემის დაპროგრამირებისათვის საჭირო დრო. განვიხილოთ მაგალითი:

```
using System;

public class ParentClass

{

   public ParentClass()

   {

      Console.WriteLine("წინაპარი კონსტრუქტორი.");

   }

   public void print()

   {

      Console.WriteLine("წინაპარი კლასი.");

   }

}

public class Childclass : ParentClass

{

   public ChildClass()

   {
```

```
        Console.WriteLine("შვილი კონსტრუქტორი.");
    }
    public static void Main()
    {
        ChildClass child = new ChildClass();
        child.print();
    }
}
```

ამ მაგალითში შემოტანილია ParentClass წინაპარი კლასი და შემოტანილია მისი კონსტრუქტორი, განსაზღვრულია მეთოდი print() და შემდეგ მოდის public class ChildClass : ParentClass . ამით შემოდის მემკვიდრე კლასი Childclass და მისი წინაპარი კლასია ParentClass . განვიხილოთ ტექსტი:

```
  public static void Main()
    {
        ChildClass child = new ChildClass();


        child.print();
    }
```

რადგან ეს ტექსტი ეკუთვნის Childclass ამიტომ ამ კლასში განისაზღვრა პროგრამა და მასში იქმნება ობიექტი child. child.print() საშუალებით ხდება წინაპარი კლასის მეთოდის გამოყენება. ამგვარად, წინაპარი კლასი არის ძირითადი კლასი და ჩაილდჩლასს არის მისი მემკვიდრე კლასი ან რაც იგივეა წარმოებული კლასი. წარმოებული კლასის კონსტრუქტორი გამოიძახება ძირითადი კლასის კონსტრუქტორის გამოძახების შემდეგ. ახლა განვიხილოთ მაგალითი თუ როგორ უკავშირდება წარმოებული კლასი ძირითად კლასს:

```
using System;


public class Parent
{
    string parentString;
    public Parent()
    {
        Console.WriteLine("ძირითადი კონსტრუქტორი.");
    }
    public Parent(string myString)
    {
        parentString = myString;
        Console.WriteLine(parentString);
    }
    public void print()
   {
        Console.WriteLine("ძირითადი კლასი");
    }
}
public class Child : Parent
{
    public Child() : base("აღებულია წარმოებული კლასიდან")
```

```
    {
        Console.WriteLine("წარმოებული კლასის კონსტრუქტორი");
    }
    public new void print()
    {
        base.print();
        Console.WriteLine("წარმოებული კლასი");
    }
    public static void Main()
    {
        Child child = new Child();
        child.print();
        ((Parent)child).print();
    }
}
```

public class Child : Parent ეს ინსტრუქცია აცხადებს წარმოებულ კლასს, ხოლო ინსტრუქცია

public Child() : base("warmoebuli klasidan")

იძახებს ძირითადი კლასის კონსტრუქტორს, რომელსაც პარამეტრად აქვს სტრიქონული ტიპის ცვლადი. ეს პარამეტრი შეიცვლება არგუმენტით "წარმოებული კლასიდან". ამას უთითებს : და გასაღები სიტყვა ბასე. ზოგჯერ შეიძლება დაგვჭირდეს, რომ შევქმნათ წარმოებულ კლასში, საკუთარი პრინტ მეთოდი და გამოვიყენოთ საჭიროების მიხედვით ხან ძირითადი კლასის პრინტ მეთოდი და ხან წარმოებული კლასის print მეთოდი.

public new void print();

გასაღები სიტყვა new საშუალებას იძლევა წარმოებულ კლასში გამოვაცხადოთ ახალი print მეთოდი და დავმალოთ ძირითადი კლასის print მეთოდი .

child.print()

იძახებს ცხადად წარმოებული კლასის print მეთოდს, ხოლო

((Parent)child).print();

იძახებს ძირითადი კლასის print მეთოდს. ((Parent)child).print(); ახდენს წარმოებული კლასის print მეთოდის შეცვლას ძირითადი კლასის print მეთოდით.

ამგვარად, ჩვენ უკვე ვიცით როგორ შევქმნათ ძირითადი და წარმოებული კლასები და როგორ გამოვიძახოთ მათი წევრები. ამავე დროს, უნდა გვახსოვდეს, რომ წამოებული კლასი არის მისი ძირითადი კლასის დაზუსტება.

2.3. პოლიმორფიზმი. ამ პარაგრაფში შეისწავლით რა არის პოლიმორფიზმი, როგორ შექმნათ წარმოებული მეთოდი, როგორ გადაფაროთ იგი და როგორ გამოიყენოთ პოლიმორფიზმი პროგრამაში.

პოლიმორფიზმი არის ობიექტზე ორიენტირებული დაპროგრამების ერთერთი ძირითადი ცნება. იგი საშუალებას გვაძლევს გამოვიძახოთ წარმოებული კლასის მეთოდები ძირითადი კლასის მიმთითებლების საშუალებით პროგრამის თვლის პროცესში. ეს არის მოხერხებული საშუალება, როცა გჭირდებათ მასივის ელემენტებად აიღოთ ობიექტები და გამოიძახოთ ამ ობიექტების მეთოდები. არაა აუცილებელი ობიექტები იყვნენ ერთიდაიგივე ტიპის. მაგრამ თუ ისინი დაკავშირებული არიან მემკვიდრეობითობით, მაშინ შეგიძლიათ ისინი დაუმატოთ მასივს, როგორც წარმოებული ტიპი და ასევე შესაძლებელია მათ ჰქონდეთ საერთო მეთოდები. შესაძლებელია მათი გამოძახება ისე, რომ არ წარმოიშვას ომონიმია. განვიხილოთ მაგალითი:

```
using System;

public class DrawingObject

{

   public virtual void Draw()

   {

      Console.WriteLine("ძირითადი დასახაზი ობიექტი.");

   }

}
```

ამ მაგალითში მოცემულია ვირტუალური მეთოდი რაწ, რომელიც შეიძლება გადაფარული იყოს წარმოებული კლასის მეთოდებით. ვირტუალური მეთოდი შემოგვაქვს მოდიფიკატორით virtual. ახლა, განვიხილოთ მაგალითი, სადაც წარმოებული კლასის მეთოდები ახდენენ ძირითადი კლასის ვირტუალური მეთოდის გადაფარვას:

```
using System;

public class Line : DrawingObject
```

```
{
    public override void Draw()
    {
        Console.WriteLine("ეს არის სწორი ხაზი.");
    }
}
public class Circle : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("ეს არის წრეწირი.");
    }
}
public class Square : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("ეს არის კვადრატი.");
    }
}
```

ვირტუალური მეთოდის გადაფარვა ხდება მოდიფიკატორით override. გადამფარავ მეთოდებს უნდა ჰქონდეთ იგივე სიგნატურა და პარამეტრები რაც ვირტუალურ მეთოდს. ახლა, განვიხილოთ პოლიმორფიზმის მაგალითი:

```
using System;

public class DrawDemo

{

   public static int Main()

   {

      DrawingObject[] dObj = new DrawingObject[4];

      dObj[0] = new Line();

      dObj[1] = new Circle();

      dObj[2] = new Square();

      dObj[3] = new DrawingObject();

      foreach (DrawingObject drawObj in dObj)

      {

         drawObj.Draw();

      }

      return 0;

   }

}
```

ამ მაგალითში შემოგვაქვს ოთხელემენტიანი მასივი და მასივის ელემენტებს ვანიჭებთ ობიექტებს. შემდეგ, ციკლში ვიყენებთ ობიექტების Draw მეთოდს.

2.4 თვისებები. ამ პარაგრაფში შეისწავლით რა არის თვისებები და რისთვის გამოიყენებიან ისინი, როგორ მოვახდინოთ მათი რეალიზაცია. ასევე, შეისწავლით მხოლოდ წაკითხვად და მხოლოდ ჩაწერად თვისებებს და ავტომატურად რეალიზებად თვისებებს. c#-ში თვისებები შემოტანილია იმისათვის, რომ დავიცვათ კლასის ველები მასში ჩაწერისა ან მისი წაკითხვისაგან. სხვა ენებში ეს კეთდება სპეციალური წამკითხველი და ჩამწერი მეთოდების საშუალებით.

თვისებების მეორე უპირატესობა ველებთან შედარებით არის ის, რომ თქვენ შეგიძლიათ დროდადრო შეცვალოთ მისი რეალიზაცია. ველის შემთხვევაში, მისი ტიპი უნდა იყოს ყოველთვის ერთიდაიგივე და თქვენ არ შეგიძლიათ შეცვალოთ იგი, მაშინ როცა თქვენ შეგიძლიათ შეცვალოთ თვისების შიგა რეალიზაცია. თუ კლიენტის ID თავდაპირველად დაყენებულია მთელ მნიშვნელობაზე, თქვენ შეგიძლიათ დააყენოთ მოთხოვნა, რომ მან ვერ მიიღოს უარყოფითი მნიშვნელობა. ამას ვერ გააკეთებთ ველის შემთხვევაში. იმ ენებში, რომლებსაც არ გაჩნიათ თვისებები, თვისებების როლი შეიძლება შეასრულოს სპეციალურად შექმნილმა მეთოდებმა. ახლა განვიხილოთ თვისებების მეთოდებით რეალიზაციის მაგალითი:

```
using System;

public class Customer

{

   private int m_id = -1;

   public int GetID()

   {

      return m_id;

   }

   public void SetID(int id)

   {

      m_id = id;

   }
```

```
    private string m_name = string.Empty;
    public string GetName()
    {
        return m_name;
    }
    public void SetName(string name)
    {
        m_name = name;
    }
}
public class CustomerManagerWithAccessorMethods
{
    public static void Main()
    {
        Customer cust = new Customer();
        cust.SetID(1);
cust.SetName("ჯემალ ანთაძე");
        Console.WriteLine(
            "ID: {0}, saxeli da gvari: {1}",
            cust.GetID(),
            cust.GetName());
        Console.ReadKey();
```

```
    }
}
```

am magaliTSi, instruqciiT

```
private string m_name = string.Empty;
```

m_name-s vaniWebT cariel mniSvnelobas. amisaTvis viyenebT string klasis Empty meTods. axla, vnaxoT rogor gavakeToT igive Tvisebebis gamoyenebiT:

```
using System;
public class Customer
{
    private int m_id = -1;
    public int ID
    {
        get
        {
            return m_id;
        }
        set
        {
            m_id = value;
        }
    }
    private string m_name = string.Empty;
```

```
    public string Name
    {
        get
        {
            return m_name;
        }
        set
        {
            m_name = value;
        }
    }
}
public class CustomerManagerWithProperties
{
    public static void Main()
    {
        Customer cust = new Customer();
        cust.ID = 1;
        cust.Name = "ჯემალ ანთაძე";
        Console.WriteLine(
                "ID: {0}, სახელი და გვარი: {1}",
                cust.ID,
```

```
            cust.Name);
        Console.ReadKey();
    }
}
```

ამ მაგალითში შემოტანილია კლასი Customer, რომელშიც გვაქვს ორი ველი int m_id და string m_name მოდიფიკატორით private. ასევე გვაქვს ორი თვისება: ID და Name, რომელთაც აქვთ get და set. მათი საშუალებით შესაძლებელია ავიღოთ ველის მნიშვნელობა და დავაყენოთ მათი მნიშვნელობები გასაღები სიტყვის value-ს საშუალებით. ინსტრუქცია cust.ID = 1; ანიჭებს id-ს მნიშვნელობას ერთს ID-ის set-ის საშუალებით, ხოლო გამოსახულება cust.ID გვაძლევს id-ის მნიშვნელობას ID-ის გეტ-ის საშუალებით. ამგვარად, ID და Name საშუალებას გვაძლევენ როგორც წავიკითხოთ თვისებების მნიშვნელობები, ასევე დავაყენოთ თვისებების მნიშვნელობები. ახლა, განვიხილოთ მაგალითი, როცა შეგვიძლია მხოლოდ თვისების მნიშვნელობის წაკითხვა:

```
using System;
public class Customer
{
    private int m_id = -1;
    private string m_name = string.Empty;
    public Customer(int id, string name)
    {
        m_id = id;
        m_name = name;
    }
    public int ID
```

```
    {
        get
        {
            return m_id;
        }
    }
    public string Name
    {
        get
        {
            return m_name;
        }
    }
}
public class ReadOnlyCustomerManager
{
    public static void Main()
    {
        Customer cust = new Customer(1, "ჯემალ ანთიძე");
        Console.WriteLine(
            "ID: {0}, სახელი და გვარი: {1}",
            cust.ID,
```

```
            cust.Name);
        Console.ReadKey();
    }
}
```

აქ, ID-ს და Name-ს აქვთ მხოლოდ get. ამოტომ, მათ შეუძლიათ მხოლოდ თვისების მნიშვნელობის წაკითხვა. ანალოგიურად, შემდეგ მაგალითში შეგვიძლია მხოლოდ თვისების მნიშვნელობის დაყენება:

```
using System;
public class Customer
{
    private int m_id = -1;
    public int ID
    {
        set
        {
            m_id = value;
        }
    }
    private string m_name = string.Empty;
    public string Name
    {
        set
```

```
        {
            m_name = value;
        }
    }
    public void DisplayCustomerData()
    {
        Console.WriteLine("ID: {0}, სახელი და გვარი: {1}", m_id, m_name);
    }
}
public class WriteOnlyCustomerManager
{
    public static void Main()
    {
        Customer cust = new Customer();
        cust.ID = 1;
        cust.Name = "ჯემალ ანთიძე";
        cust.DisplayCustomerData();
        Console.ReadKey();
    }
}
```

ახლა, ვნახოთ თუ როგორ წავიკითხოთ და დავაყენოთ თვისების მნიშვნელობა ავტომატურად:

```
using System;

public class Customer

{

   public int ID { get; set; }

   public string Name { get; set; }

}

public class AutoImplementedCustomerManager

{

   static void Main()

   {

      Customer cust = new Customer();

      cust.ID = 1;

      cust.Name = "ჯემალ ანთიძე";

      Console.WriteLine(

         "ID: {0}, სახელი და გვარი: {1}",

         cust.ID,

         cust.Name);

      Console.ReadKey();

   }

}
```

როგორც ხედავთ, ასეთი მიდგომა უფრო კომპაქტურია ტექსტის მოცულობის მიხედვით.

3 თავი

C#-ის სპეციალური საშუალებები

3.1 მაინდექსირებლები. ამ ლექციაში შეისწავლით რა არის მაინდექსირებელი და რისთვის გამოიყენება იგი, როგორ შექმნათ და გადატვირთოთ ისინი, როგორ შექმნათ მრავალპარამეტრიანი მაინდექსირებლები. მაინდექსირებელი საშუალებას აძლევს კლასს გამოყენებული იყოს როგორც მასივი. კლასის შიგნით მაინდექსირებლით შეიძლება მართოთ ნებისმიერი გზით მონაცემთა კოლექციები. მაინდექსირებლით თქვენ შეგიძლიათ მიუთითოთ კლასის ნებისმიერი მონაცემი. განვიხილოთ მაგალითი:

```
using System;

// მარტივი მაინდექსირებლის მაგალითი.

class IntIndexer

{

   private string[] myData;

   public IntIndexer(int size)

   {

      myData = new string[size];

      for (int i = 0; i < size; i++)

      {

         myData[i] = "ცარიელი";

      }

   }

   public string this[int pos]

   {
```

```
    get
    {
        return myData[pos];
    }
    set
    {
        myData[pos] = value;
    }
}
static void Main(string[] args)
{
    int size = 10;
    IntIndexer myInd = new IntIndexer(size);
    myInd[9] = "რაიმე მნიშვნელობა";
    myInd[3] = "სხვა მნიშვნელობა";
    myInd[5] = "ნებისმიერი მნიშვნელობა";
    Console.WriteLine("\nმაინდექსირებლის საშუალებით გამოტანა \n");
    for (int i = 0; i < size; i++)
    {
        Console.WriteLine("myInd[{0}]: {1}", i, myInd[i]);
    }
}
```

}

ამ მაგალითში, პარამეტრიანი კონსტრუქტორით იქმნება მასივი, რომლის სიგრძე მოცემულია პარამეტრით და ამ მასივის ელემენტებს ენიჭება მნიშვნელობა "ცარიელი". შემდეგ, იქმნება მაინდექსირებელი გეტ, სეტ და ვალუე გასაღები სიტყვების გამოყენებით. პარამეტრი უთითებს მასივის ინდექსს. Main მეთოდში ხდება მაინდექსირებლის გამოყენებით კონკრეტული მასივის შექმნა და მის ელემენტებზე მითითება. შემდეგი მაგალითი გვიჩვენებს თუ როგორ შეიძლება მაინდექსირებლის გადატვირთვა:

```
// გადატვირთული მაინდექსირებლის განხორციელება.

class OvrIndexer

{

   private string[] myData;

   private int arrSize;

   public OvrIndexer(int size)

   {

      arrSize = size;

      myData = new string[size];

      for (int i = 0; i < size; i++)

      {

         myData[i] = "ცარიელი";

      }

   }

   public string this[int pos]

   {
```

```
    get
    {
        return myData[pos];
    }
    set
    {
        myData[pos] = value;
    }
}
public string this[string data]
{
    get
    {
        int count = 0;
        for (int i = 0; i < arrSize; i++)
        {
            if (myData[i] == data)
            {
                count++;
            }
        }
        return count.ToString();
```

```
    }
    set
    {
        for (int i = 0; i < arrSize; i++)
        {
            if (myData[i] == data)
            {
                myData[i] = value;
            }
        }
    }
}
static void Main(string[] args)
{
    int size = 10;
    OvrIndexer myInd = new OvrIndexer(size);
    myInd[9] = "რაიმე მნიშვნელობა";
    myInd[3] = "სხვა მნიშვნელობა";
    myInd[5] = "ნებისმიერი მნიშვნელობა";
    myInd["carieli"] = "არ აქვს მნიშვნელობა";
    Console.WriteLine("\nმაინდექსირებლის გამოტანა\n");
    for (int i = 0; i < size; i++)
```

```
    {
        Console.WriteLine("myInd[{0}]: {1}", i, myInd[i]);
    }
    Console.WriteLine("\n \"არ აქვს მნიშვნელობა\" მნიშვნელობათა რაოდენობა: {0}",
myInd["არ აქვს მნიშვნელობა"]);
  }
}
```

პირველი მაინდექსირებელი იგივეა რაც წინა მაგალითში. მეორე მაინდექსირებელში, get-ის დროს, ხდება პარამეტრის მნიშვნელობის მქონე მასივის ელემენტების დათვლა. set-ის დროს, თუ მასივის ელემენტი უდრის პარამეტრის მნიშვნელობას, მაშინ მასივის ელემენტს ენიჭება ახალი მნიშვნელობა.

როცა მაინდექსირებელს მრავალი პარამეტრი აქვს დაიწერება ასე:

```
public object this[int param1, int param2, int param3]
  {
    get
    {
        // მოქმედებები და კლასის ზოგიერთი მონაცემის
// დაბრუნება
    }
    set
    {
            // მოქმედებები და კლასის ზოგიერთი მონაცემის
// დაყენება
```

```
    }

}
```

3.2 სტრუქტურები. იმისათვის, რომ დავინახოთ თუ რა განსხვავება არსებობს სტრუქტურებსა და კლასებს შორის, შევადაროთ ისინი ერთიმეორეს. პირველ რიგში, struct არის მნიშვნელობის ტიპი, ხოლო class არის მიმთითებლის ტიპი. სტრუქტურის განსაზღვრის დროს, მეხსიერებაში გამოიყოფა მისთვის ადგილი, მაშინ როცა კლასის განსაზღვრის დროს მეხსიერებაში გამოიყოფა ადგილი კლასის მიმთითებლისათვის. ახლა, განვიხილოთ მომხმარებლის მიერ განსაზღვრული სტრუქტურის მაგალითი:

```
// კლიენტის struct ტიპი, რომელიც წარმოადგენს მართკუთხოვან ფორმას

struct Rectangle

{

   private int m_width;

      // მართკუთხედის სიგანე

   public int Width

   {

      get

      {

         return m_width;

      }

      set

      {

         m_width = value;

      }

   }

   // მართკუთხედის სიმაღლე
```

```
    private int m_height;
    // მართკუთხედის სიმაღლის დაყენება და მიღება
    public int Height
    {
        get
        {
            return m_height;
        }
        set
        {
            m_height = value;
        }
    }
}
```

ამ მაგალითში განსაზღვრულია სტრუქტურა Rectangle, რომელსაც აქვს ორი თვისება. სტრუქტურის განსაზღვრა იწყება სიტყვით სტრუცტ, რომელსაც მოსდევს სტრუქტურის სახელი და ფიგურულ ფრჩხილებში მოთავსებული სტრუქტურის წევრები. რომ გამოვიყენოთ ეს სტრუქტურა, უნდა მოვახდინოთ მისი ინიციალიზაცია და შემდეგ გამოვიყენოთ მისი თვისებები, როგორც ეს მოცემულია შემდეგ მაგალითში:

```
using System;
// სტრუქტურის განსაზვრისა და გამოყენების მაგალითი
class StructExample
{
```

```
    // პროგრამაში შესასვლელი
        static void Main()
    {
        // სტრუქტურის რეალიზაცია
        // სიგრძეა 1 და სიმაღლე 3
        Rectangle rect1 = new Rectangle();
        rect1.Width = 1;
        rect1.Height = 3;
        // rect1-ის სიგრძისა და სიმაღლის ჩვენება
        Console.WriteLine("rect1:  {0}:{1}", rect1.Width, rect1.Height);
Console.ReadKey();
    }
}
```

იგივე შეიძლებაგაკეთდეს ობიექტის ინიციალიზაციით:

```
        Rectangle rect11 = new Rectangle
        {
            Width = 1,
            Height = 3
        };
```

აქ, შექმნილია rect11 მართკუთხედი, გვერდებით 1 და 3.

```
// klientis struct tipi, romelic warmoadgens marTkuTxeds
 struct Rectangle
```

```
{
    // სიგანის მოცემა
    private int m_width;
    // სიგანის დაყენება და მიღება
    public int Width
    {
        get
        {
            return m_width;
        }
        set
        {
            m_width = value;
        }
    }
    // ადგილი სიმაღლისათვის
      private int m_height;
    // სიმაღლის დაყენება და მიღება
        public int Height
    {
        get
        {
```

```
        return m_height;
      }
      set
      {
        m_height = value;
      }
    }
```

რომ გადატვირთოთ სტრუქტურის კონსტრუქტორი უნდა შემოიტანოთ ცხადად პარამეტრებიანი კონსტრუქტორი, როგორც ეს ნაჩვენებია შემდეგ მაგალითში:

```
    // ახალი მართკუთხედის შექმნა პარამეტრებიანი  კონსტრუქტორით
    public Rectangle(int width, int height)
    {
      m_width = width;
      m_height = height;
    }}
```

სტრუქტურის ინიციალიზაცია კონსტრუქტორის გადატვირთვით:

```
using System;
class StructExample
{
    static void Main()
    {
            // rect2 სტრუქტურის შექმნა
```

```
Rectangle rect2 = new Rectangle(5, 7);


        Console.WriteLine("rect2:  {0}:{1}", rect2.Width, rect2.Height);
        Console.ReadKey();
    }
}
```

სტრუქტურაში მეთოდის შემოტანა:

```
struct Rectangle
{
    private int m_width;
    public int Width
    {
        get
        {
            return m_width;
        }
        set
        {
            m_width = value;
        }
    }
    private int m_height;
```

```
    public int Height
    {
        get
        {
            return m_height;
        }
        set
        {
            m_height = value;
        }
    }
// პარამეტრებიანი კონსტრუქტორი
    public Rectangle(int width, int height)
    {
        m_width = width;
        m_height = height;
    }
            // მოცემული მართკუთხედის ზომის გაზრდა კონკრეტული
//მართკუთხედის ზომით
    public Rectangle Add(Rectangle rect)
    {
            // newRect მართკუთხედის შექმნა
```

```
                        Rectangle newRect = new Rectangle();
        // მართკუთხედის ზომის გაზრდა
        newRect.Width = Width + rect.Width;
        newRect.Height = Height + rect.Height;
        // newRect-is dabruneba
                        return newRect;
    }
}
```

სტრუქტურის მეთოდის გამოძახება:

```
using System;
// სტრუქტურის გამოცხადებისა და გამოყენების მაგალითი
class StructExample
{
        static void Main()
    {
        // რეცტ1 მართკუთხედის შექმნა და ინიციალიზაცია
        Rectangle rect1 = new Rectangle();
        rect1.Width = 1;
        rect1.Height = 3;
        Console.WriteLine("rect1:  {0}:{1}", rect1.Width, rect1.Height);
            // rect2 მართკუთხედის შექმნა და ინიციალიზაცია
Rectangle rect2 = new Rectangle(5, 7);
```

```
        Console.WriteLine("rect2:  {0}:{1}", rect2.Width, rect2.Height);

    // rect3 მართკუთხედის  შექმნა    და  ინიციალიზაცია rect1.Add-ით

Rectangle rect3 = rect1.Add(rect2);

        Console.WriteLine("მართკუთხედი3:  {0}:{1}", rect3.Width, rect3.Height);

        Console.ReadKey();

    }

}

        Console.ReadKey();

    }

}
```

3.3 ინტერფეისები.

ამ პარაგრაფში შეისწავლით რა არის ინტერფეისი, როგორ შექმნათ და გამოიყენოთ იგი. ინტერფეისს, ისევე როგორც კლასს, აქვს სახელი და იგი შეიძლება შეიცავდეს ხდომილებების, მაინდექსირებლების, მეთოდებისა თვისებების გამოცხადებებს რეალიზაციის გარეშე. შეთანხმებით მისი სახელი იწყება ყოველთვის I ასოთი. განვიხილოთ მაგალითი:

```
interface IMyInterface

{

   void MethodToImplement();

}
```

ამ მაგალითში შემოტანილია ინტერფეისი IMyInterface , რომელიც შეიცავს ერთ სარეალიზაციო მეთოდს MethodToImplemnt(); ტანის გარეშე. ამ ინტერფეისის მემკვიდრე კლასში უნდა იყოს ამ მეთოდის განსაზღვრება და მათ უნდა ჰქონდეთ ერთნაირი სიგნატურა. ახლა, ვნახოთ ამ ინტერფეისის გამოყენება:

```
class InterfaceImplementer : IMyInterface

{

   static void Main()

   {

      InterfaceImplementer iImp = new InterfaceImplementer();

      iImp.MethodToImplement();

   }

   public void MethodToImplement()

   {

      Console.WriteLine("MethodToImplement() გამოძახება.");
```

```
    }
}
```

აქ, ჩვენ გვაქვს წინა ინტერფეისის მემკვიდრე კლასი, რომელიც რეალიზაციას უკეთებს ინტერფეისის მეთოდს და იყენებს ამ მეთოდს. ყურადღება მიაქციეთ, რომ ინტერფეისის მეთოდს და რეალიზებულ მეთოდს აქვთ ერთნაირი სიგნატურა. ეს სავალდებულოა, წინააღმდეგ შემთხვევაში კომპილატორი მოგვცემს შეცდომას. ინტერფეისის მემკვიდრე კლასი უნდა შეიცავდეს ინტერფეისის ყველა წევრის რეალიზაციას. ერთი ინტერფეისი შეიძლება იყოს მეორე ინტერფეისის მემკვიდრე. განვიხილოთ მაგალითი:

```
 using System;
interface IParentInterface
{
    void ParentInterfaceMethod();
}
interface IMyInterface : IParentInterface
{
    void MethodToImplement();
}


class InterfaceImplementer : IMyInterface
{
    static void Main()
    {
        InterfaceImplementer iImp = new InterfaceImplementer();
```

```
        iImp.MethodToImplement();
        iImp.ParentInterfaceMethod();
    }
    public void MethodToImplement()
    {
        Console.WriteLine("MethodToImplement() გამოძახება.");
    }
    public void ParentInterfaceMethod()
    {
        Console.WriteLine("ParentInterfaceMethod() გამოძახება.");
    }
}
```

ამგვარად, ამ ლექციაში ვაჩვენეთ როგორ შევქმნათ ინტერფეისი, როგორ გამოვიყენოთ შექმნილი ინტერფეისი მის მემკვიდრე კლასში და როგორ გავახორციელოთ ინტერფეისების მემკვიდრეობითობა.

3.4 დელეგატები და ხდომილებები. ამ პარაგრაფში შეისწავლით რა არის დელეგატი და ხდომილება და როგორ გამოიყენოთ ისინი. C#-ში გვაქვს მონაცემების ორი ტიპი – მნიშვნელობები და მიმთითებლები. აქამდე გაეცანით კლასებსა და ინტერფეისებს. ახლა გაეცნობით დელეგატებს, რომლებიც არიან მეთოდებზე მიმთითებლები.

```
using System;

// ეს არის დელეგატის გამოცხადება

public delegate int Comparer(object obj1, object obj2);

public class Name

{

   public string FirstName = null;

   public string LastName = null;

   public Name(string first, string last)

   {

      FirstName = first;

      LastName = last;

   }

   // ეს არის delegate მეთოდის დამამუშავებელი

   public static int CompareFirstNames(object name1, object name2)

   {

      string n1 = ((Name)name1).FirstName;

      string n2 = ((Name)name2).FirstName;

      if (String.Compare(n1, n2) > 0)
```

```
        {
            return 1;
        }
        else if (String.Compare(n1, n2) < 0)
        {
            return -1;
        }
        else
        {
            return 0;
        }
    }
    public override string ToString()
    {
        return FirstName + " " + LastName;
    }
}
class SimpleDelegate
{
    Name[] names = new Name[5];
    public SimpleDelegate()
    {
```

```
    names[0] = new Name("ილია", "ჭავჭავაძე");
    names[1] = new Name("აკაკი", "წერეთელი");
    names[2] = new Name("ვაჟა", "ფშაველა");
    names[3] = new Name("ეგნატე", "ნინოშვილი");
    names[4] = new Name("ჭოლა", "ლომთათიძე");
}
static void Main(string[] args)
{
    SimpleDelegate sd = new SimpleDelegate();
    // ეს არის delegate-ის  მყისიერი მდგომარეობა
    Comparer cmp = new Comparer(Name.CompareFirstNames);
    Console.WriteLine("\nდახარისხებამდე: \n");
    sd.PrintNames();
    // დაათვალიერეთ delegate-ს არგუმენტი
    sd.Sort(cmp);
    Console.WriteLine("\nდახარისხების შემდეგ: \n");
    sd.PrintNames();
}

// delegate-ის პარამეტრი
public void Sort(Comparer compare)
{
```

```
    object temp;
    for (int i=0; i < names.Length; i++)
    {
        for (int j=i; j < names.Length; j++)
        {
            // delegate "compare" გამოყენება
                        // ჩვეულებრივი მეთოდის მსგავსად
            if ( compare(names[i], names[j]) > 0 )
            {
                temp = names[i];
                names[i] = names[j];
                names[j] = (Name)temp;
            }
        }
    }
}
public void PrintNames()
{
    Console.WriteLine("სახელები: \n");
    foreach (Name name in names)
    {
        Console.WriteLine(name.ToString());
```

```
        }
    }
}
 public static int CompareFirstNames(object name1, object name2)
    {
        ...
    }
using System;
using System.Drawing;
using System.Windows.Forms;
//  საკუთარი  delegate
public delegate void Startdelegate();
class Eventdemo : Form
{
    // საკუთარი event
    public event Startdelegate StartEvent;
    public Eventdemo()
    {
        Button clickMe = new Button();
        clickMe.Parent = this;
        clickMe.Text = "Click Me";
        clickMe.Location = new Point(
```

```
        (ClientSize.Width - clickMe.Width) /2,
        (ClientSize.Height - clickMe.Height)/2);
    // ხდომილების დამამუშავებელი Delegate დანიშნულია
    // ღილაკზე დაწკაპუნების ხდომილებისათვის
    clickMe.Click += new EventHandler(OnClickMeClicked);
    // ჩვენი custom "Startdelegate" delegateდაენიშნა
    // ჩვენს custom "StartEvent" ხდომილებას.
    StartEvent += new Startdelegate(OnStartEvent);
    // custom event-ის ჩართვა
    StartEvent();
}
    // ეს მეთოდი გამოიძახება როცა "clickMe" ღილაკს დააჭერთ
public void OnClickMeClicked(object sender, EventArgs ea)
{
    MessageBox.Show("You Clicked My Button!");
}
//ეს მეთოდი გამოიძახება, როცა "StartEvent" ხდომილება ჩაირთვება
public void OnStartEvent()
{
    MessageBox.Show("მე დავიწყე!");
}
static void Main(string[] args)
```

```
    {
        Application.Run(new Eventdemo());
    }
}
public void OnClickMeClicked(object sender, EventArgs ea)
    {
        MessageBox.Show("თქვენ დააჭირეთ ღილაკს!");
    }
```

3.5 განსაკუთრებულ შემთხვევათა დამუშავება. ამ პარაგრაფში შეისწავლით რა არის განსაკუთრებული შემთხვევა, როგორ განახორციელოთ შეცდომების დამუშავების სტანდარტი try/catch ბლოკით და როგორ გაათავისუფლოთ რესურსები finally ბლოკით.

განსაკუთრებული შემთხვევები არიან გაუთვალისწინებელი შეცდომები, რომლებიც შეიძლება პროგრამაში წარმოიშვნენ. უმეტეს შემთხვევებში შესაძლებელია იწინასწარმეტყველოთ სად შეიძლება წარმოიშვას შეცდომა და მიიღოთ სათანადო ზომები, რომ აიცილოთ იგი. მაგალითად, შეამოწმოთ აბრუნებს თუ არა ფუნქცია იმ მნიშვნელობას რასაც ელოდებით. ზოგ შემთხვევებში შეუძლებელია იწინასწარმეტყველოთ იგი. მაგალითად, ფაილის წაკითხვის შემთხვევაში და მონაცემთა ბაზასთან მუშაობის დროს. System.Exeption კლასი ითვალისწინებს სხვადასხვა მეთოდებსა და თვისებებს, რომ მიიღოთ ინფორმაცია იმის შესახებ თუ რა არ არის სწორი პროგრამაში. მაგალითად, Message თვისება ითვალისწინებს ინფორმაციას თუ რა შეცდომა წარმოიშვა. Stacktrace თვისება იძლევა ინიფორმაციას იმის შესახებ თუ რა ადგილზე წარმოიშვა შეცდომა და გადაფარული ToString() იძლევა დაწვრილებით ინფორმაციას მთელი შეცდომის შესახებ. თუ რა განსაკუთრებული შემთხვევები შეიძლება წარმოიშვას დამოკიდებულია იმაზე, თუ რა მოქმედება სრულდებოდა განსაკუთრებული შემთხვევის წარმოშობის მომენტში. მაგალითად, System.IO.File.OpenRead() მეთოდის შესრულებისას შეიძლება წამოიშვას შემდეგი განსაკუთრებული შემთხვევები:

SecurityException

ArgumentException

PathTooLongException

DirectoryNotFoundException

FileNotFoundException

NotSupportedException

რომ დაადგინოთ, კონკრეტულმა მეთოდმა რა განსაკუთრებული შემთხვევები შეიძლება წარმოშვას, უნდა შეხვიდეთ .Net Frmework SDK დოკუმენტაციაში,

გადახვიდეთ Reference/Class Library სექციაში და ნახოთ Namespace/Class/Method დოკუმენტაცია. მას შემდეგ, რაც იცით თუ რა განსაკუთრებული შემთხვევები შეიძლება წარმოიშვას კონკრეტული მეთოდის გამოყენების შემთხვევაში, თქვენ უნდა დაამუშაოთ ყოველი განსაკუთრებული შემთხვევა try/catch ბლოკებში. try ბლოკში უნდა შეიტანოთ გამოსაცდელი კოდი და catch ბლოკში შესაძლო განსაკუთრებული შემთხვევების დამამუშავებელი კოდი. განვიხილოთ მაგალითი:

Listing 15-1. Using a try/catch Blocks: tryCatchDemo.cs

```
using System;

using System.IO;

class tryCatchDemo

{

   static void Main(string[] args)

   {

      try

      {

         File.OpenRead("NonExistentFile");

      }

      catch(Exception ex)

      {

         Console.WriteLine(ex.ToString());

      }

   }

}
```

ამ მაგალითში საცდელი მეთოდია File.OpenRead, ხოლო catch ბლოკში დაჭერა ხდება Exception განსაკუთრებული შემთხვევის. რადგან Exception არის ბაზური განსაკუთრებული შემთხვევის ტიპი (ყველა სხვა განსაკუთრებული შემთხვევა არის მისგან წარმოებული ტიპი), ამიტომ ნებისმიერი განსაკუთრებული შემთხვევა იქნება დაჭერილი. ახლა ვნახოთ, თუ როგორ დავიჭიროთ კონკრეტული განსაკუთრებული შემთხვევა:

```
catch(FileNotFoundException fnfex)
    {
       Console.WriteLine(fnfex.ToString());
    }
    catch(Exception ex)
    {
       Console.WriteLine(ex.ToString());
    }
```

ამ მაგალითში გვაქვს ორი განსაუთრებული შემთხვევის დაჭერა. რადგან FileNotFoundException არის წარმოებული ტიპი ამიტომ პირველ რიგში მოხდება მისი დაჭერა. სხვა შეცდომის შემთხვევაში მოხდება Exception-ის დაჭერა. განსაკუთრებული შემთხვევის დროს საჭიროა მისი შედეგების ლიკვიდაცია. ამისათვის გამოიყენება finally ბლოკი. იგი სრულდება მეთოდის დამთავრების წინ. მასში უნდა მოთავსდეს ის მოქმედებები, რაც საჭიროა განსაკუთრებული შემთხვევის შედეგების აღმოსაფხვრელად. განვიხილოთ მაგალითი:

Listing 15-2. საბოლოო ბლოკის რეალიზაცია: FinallyDemo.cs

```
using System;
using System.IO;
class FinallyDemo
{
```

```
static void Main(string[] args)
{
    FileStream outStream = null;
    FileStream inStream = null;
    try
    {
        outStream = File.OpenWrite("DestinationFile.txt");
        inStream = File.OpenRead("BogusInputFile.txt");
    }
    catch(Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
    finally
    {
        if (outStream != null)
        {
            outStream.Close();
            Console.WriteLine("გარე ნაკადი დაიხურა.");
        }
        if (inStream != null)
        {
```

```
            inStream.Close();

            Console.WriteLine("inStream დაიხურა.");

        }

    }

}
```

ამ მაგალითში ნაკადური შეტანა/გამოტანის შემთხვევაში წარმოშობილი შეცდომისას ხდება სათანადო ნაკადის დახურვა.

3.6 ატრიბუტების გამოყენება. ამ პარაგრაფში შეისწავლით: რა არის ატრიბუტი და რატომ უნდა გამოიყენოთ იგი; უპარამეტრო და მრავალპარამეტრიანი ატრიბუტების გამოყენებას; ასამბლირებული კოდის, ტიპის წევრების და ტიპის დონის ატრიბუტების გამოყენებას. ატრიბუტები არიან ელემენტები, რომლებიც საშუალებას გაძლევენ დაუმატოთ დეკლარაციული ინფორმაცია თქვენს პროგრამებს. ეს დეკლარაციული ინფორმაცია გამოიყენება სხვადასხვა დანიშნულებით პროგრამის შესრულების დროს და პროგრამის შექმნისას გამოყენებითი პროგრამების შემქმნელი საშუალებების მიერ. მაგალითად, არსებობენ ატრიბუტები როგორიცაა DllImportAttribute , რომლითაც თქვენ შეგიძლიათ დაუკავშირდეთ WIN32 ბიბლიოთეკებს. მეორე ატრიბუტს ObsoleteAttribute-ს კომპილაციის დროს გამოაქვს გაფრთხილება, რომ კონკრეტული მეთოდი არ შეიძლება გამოყენებულ იქნეს. ატრიბუტები არიან კლასები, რომლებიც შეიძლება დაიწერონ C#-ზე და გამოყენებულ იქნენ თქვენი კოდის გასალამაზებლად დეკლარაციული ინფორმაციით. ამ ლექციაში ვაჩვენებთ როგორ გამოვიყენოთ წინასწარ არსებული ატრიბუტები პროგრამაში. რაც დაგეხმარებათ გამოიყენოთ მრავალი სხვა წინასწარ არსებული ატრიბუტები, რომლებიც არსებობენ .NET class ბიბლიოთეკებში. ატრიბუტი იწერება ტიპის წინ და ჩასმულია კვადრატულ ფრჩხილებში. მაგალითად, [ObsoluteAtribute], ან [Obsolute] სიტყვა Atribute გარეშე. მრავალ ატრიბუტს აქვს პარამეტრების სია, რაც საშუალებას გვაძლევს ჩავრთოთ დამატებითი ინფორმაცია ატრიბუტში. განვიხილოთ მაგალითი:

```
using System;


class BasicAttributeDemo

{

   [Obsolete]

   public void MyFirstdeprecatedMethod()
```

```
{
    Console.WriteLine("gamoZaxebulia MyFirstdeprecatedMethod().");
}
[ObsoleteAttribute]
public void MySecondDeprecatedMethod()
{
    Console.WriteLine("გამოძახებულია MySecondDeprecatedMethod().");
}
[Obsolete("ar unda gamoiyenoT es meTodi")]
public void MyThirdDeprecatedMethod()
{
    Console.WriteLine("გამოძახებულია  MyThirdDeprecatedMethod().");
}
// უსაფრთხო პროგრამის ძაფის გაკეთება COM-ისთვის
[STAThread]
static void Main(string[] args)
{
    BasicAttributeDemo attrDemo = new BasicAttributeDemo();

    attrDemo.MyFirstdeprecatedMethod();
    attrDemo.MySecondDeprecatedMethod();
    attrDemo.MyThirdDeprecatedMethod();
```

```
    }
}
 [Obsolete]
    public void MyFirstdeprecatedMethod()
    ...
    [ObsoleteAttribute]
    public void MySecondDeprecatedMethod()
    ...
```

Listing 16-2. პოზიციური და სახელიანი ატრიბუტი პარამეტრების გამოყენება: AttributeParamsDemo.cs

```
using System;
using System.Runtime.InteropServices;
class AttributeParamsDemo
{
    [DllImport("User32.dll", EntryPoint="MessageBox")]
    static extern int MessageDialog(int hWnd, string msg, string caption, int msgType);
    [STAThread]
    static void Main(string[] args)
    {
        MessageDialog(0, "MessageDialog Called!", "DllImport Demo", 0);
    }
}
```

ატრიბუტს შეიძლება ჰქონდეს პოზიციური და სახელდებული პარამეტრები. პოზიციური პარამეტრები წინ უნდა უსწრებდეს სახელდებულ პარამეტრებს:

Listing 16-3. პოზიციური და ატრიბუტი პარამეტრების გამოყენება:
AttributeTargetdemo.cs

```
using System;

[assembly:CLSCompliant(true)]

public class AttributeTargetdemo

{

   public void NonClsCompliantMethod(uint nclsParam)

   {

      Console.WriteLine("გამოძახაNonClsCompliantMethod().");

   }

   [STAThread]

   static void Main(string[] args)

   {

      uint myUint = 0;

      AttributeTargetdemo tgtdemo = new AttributeTargetdemo();

      tgtdemo.NonClsCompliantMethod(myUint);

   }

}
```

4 თავი.
სხვა საშუალებები ეფექტური პროგრამების შესაქმნელად

4.1 გადანომვრა. ამ პარაგრაფის მიზანია: შევისწავლოთ რა არის გადანომრვა; შეგეძლოთ შექმნათ ახალი გადანომრის ტიპები; შევისწავლოთ როგორ გამოვიყენოთ ისინი და გავეცნოთ System.Enum ტიპის მეთოდებს. გადანომრვები არიან მკაცრად ტიპიზირებული კონსტანტები. ისინი არსებითად არიან უნიკალური ტიპები, რომლებიც საშუალებას გაძლევთ სიმბოლური სახელების ნაცვლად გამოიყენოთ მთელი რიცხვები. ნებისმიერი ახალი აღნიშვნა გადანომრვის ტიპსა და მთელ რიცხვს შორის მოითხოვს ცხად გარდაქმნას, რადგან ისინი სხვადასხვა ტიპის არიან. სიმბოლური აღნიშვნები არიან შინაარსობლივი, მაგრამ არიან მოუხერხებელი მისათითებლად ვიდრე მთელი რიცხვები. მაგალითად, თუ გვაქვს სახელის მქონე მნიშვნელობები: ორშაბათი, სამშაბათი და ასე შემდეგ კვირა და მათ შევუსაბამებთ მთელ რიცხვებს 0, 1 და ასე შემდეგ 6, ეს რიცხვები შემდგომში მიუთითებენ კვირის დღეებს. გადანომრვები არიან აღნიშნული enum ან Enum გასაღები სიტყვებით. C#-ის enum ტიპი ასე Class Library (BCL)-ის Enum ტიპის მემკვიდრეა. გამოიყენეთ C#-ის enum, რომ განსაზღვროთ ახალი გადანომრვის ტიპები და გამოიყენეთ Enum, რომ განახორციელოთ static enum მეთოდები. განვიხილოთ მაგალითი, თუ როგორ განსაზღვროთ enum ტიპი და გამოიყენოთ იგი switch ინსტრუქციაში:

```
using System;

// აცხადებს enum-ს

public enum Volume

{

  Low,

  Medium,

  High

}

// enum-ის გამოყენება
```

```
class EnumSwitch
{
  static void Main()
  {
    // enum ტიპის ობიექტის შექმნა
    Volume myVolume = Volume.Medium;
    // გადაწყვეტილების მიღება ენუმ მნიშვნელობის მიხედვით
    switch (myVolume)
    {
      case Volume.Low:
        Console.WriteLine("ტომი გამორთულია.");
        break;
      case Volume.Medium:
        Console.WriteLine("ტომის ნახევარი.");
        break;
      case Volume.High:
        Console.WriteLine("ტომი ჩაირთო.");
        break;
    }
}
```

ახლა განვიხილოთ როგორ დავაყენოთ Enum Base და როგორ მოვახდინოთ მისი წევრების ინიციალიზაცია:

```
using System;
// აცხდებს  enum-ს
public enum Volume : byte
{
   Low = 1,
   Medium,
   High
}
class EnumBaseAndMembers
{
   static void Main()
   {
            // შექმნა და ინიციალიზაცია
            // enum ტიპის ობიექტის შექმნა
      Volume myVolume = Volume.Low;
      // enum  მნიშვნელობის მიხედვით გადაწყვეტილების მიღება
      switch (myVolume)
      {
         case Volume.Low:
            Console.WriteLine("ტომი გამოირთო.");
            break;
         case Volume.Medium:
```

```
            Console.WriteLine("ტომის ნახევარი.");
            break;
        case Volume.High:
            Console.WriteLine("ტომი ჩაირთო.");
            break;
      }
      Console.ReadLine();
   }
}
```

enum ბაზური ტიპი გაჩუმებით არის int, მაგრამ შეიძლება შეიცვალოს სხვა ბაზური ტიპით – byte, როგორც ეს მოცემულია ინსტრუქციაში:

```
public enum Volume : byte
```

ჩვეულებრივ, საწყისი მნიშვნელობა იწყება ნულით, შემდეგი მნიშვნელობა არის ერთი და ასე შემდეგ. მაგრამ, საწყისი მნიშვნელობა დავიწყოთ სხვა რიცხვით, როგორც ეს მოცემულია ინსტრუქციაში:

```
Low = 1
```

ახლა, განვიხილოთ Enum გარდაქმნები და System.Enum გამოყენების მაგალითი:

Listing 17-3. Enum გარდაქმნები და using the System.Enum ტიპი: Enumtricks.cs

```
using System;
// enum-ის გამოცხადება
public enum Volume : byte
{
   Low = 1,
```

```
    Medium,
    High
}

// გადანომრვებთან მუშაობის სხვა ხერხი
class Enumtricks
{
    static void Main(string[] args)
    {
            // ობიექტის შექმნა
        Enumtricks enumtricks = new Enumtricks();
        // გვიჩვენებს ცხად გარდაქმნას int-ის Volume-ში
        enumtricks.GetEnumFromUser();
        //იტერაცია სახელით
        enumtricks.ListEnumMembersByName();
            // იტერაცია მიშვნელობით
        enumtricks.ListEnumMembersByValue();
        Console.ReadLine();
    }
    public void GetEnumFromUser()
    {
        Console.WriteLine("\n----------------");
```

```
        Console.WriteLine("ზომის დაყენება:");
        Console.WriteLine("----------------\n");
        Console.Write(@"1 – მცირე  2 – საშუალო 3 – დიდი აირჩიეთ ერთერთი (1, 2, ან 3): ");
```

```
            // მომხმარებლის მიერ განსაზღვრული მნიშვნელობის მიღება
        string volString = Console.ReadLine();
// ცხადი გარდაქმნა
        int volInt = Int32.Parse(volString);
        // ცხადი გარდაქმნა
         Volume myVolume = (Volume)volInt;
        Console.WriteLine();
        switch (myVolume)
        {
           case Volume.Low:
              Console.WriteLine("ტომი გამოირთო.");
              break;
           case Volume.Medium:
              Console.WriteLine("ტომის ნახევარი.");
              break;
           case Volume.High:
              Console.WriteLine("ტომი ჩაირთო.");
```

```
            break;
        }
        Console.WriteLine();
    }
    // სახელით იტერაცია
    public void ListEnumMembersByName()
    {
        Console.WriteLine("\n----------------------------- ");
        Console.WriteLine("ტომის წევრების გადანომრვა სახელით:");
        Console.WriteLine("-----------------------------\n");
        foreach (string volume in Enum.GetNames(typeof(Volume)))
        {
            Console.WriteLine("ტომის წევრი: {0}\n მნიშვნელობა: {1}",
                volume, (byte)Enum.Parse(typeof(Volume), volume));
        }
    }
    // მიშვნელობით იტერაცია
    public void ListEnumMembersByValue()
    {
        Console.WriteLine("\n----------------------------- ");
        Console.WriteLine("ტომის წევრების გადანომრვა მნიშვნელობით:");
        Console.WriteLine("-----------------------------\n");
```

```
        // ყველა მნიშვნელობის (რიცხვითი მნიშვნელობების) მიღება ტომიდან
        // enum ტიპი, წევრის სახელის ფორმირება და გამოტანა
        foreach (byte val in Enum.GetValues(typeof(Volume)))
        {
            Console.WriteLine("ტომის მნიშვნელობა: {0}\n წევრი: {1}",
                val, Enum.GetName(typeof(Volume), val));
        }
    }
}
// მომხმარებლის მიერ გათვალისწინებული მნიშვნელობის მიღება
        string volString = Console.ReadLine();
        int volInt = Int32.Parse(volString);
        // ცხადი გარდაქმნის შესრულება int-დან enum ტიპში
        Volume myVolume = (Volume)volInt;
  // წევრების სახელების სიის მიღება enum ტომიდან,
        // რიცხვითი მნიშვნელობის ფორმირება და გამოტანა
        foreach (string volume in Enum.GetNames(typeof(Volume)))
        {
            Console.WriteLine("ტომის წევრი: {0}\n მნიშვნელობა: {1}",
                volume, (byte)Enum.Parse(typeof(Volume), volume));
        }
  // ყველა მნიშვნელობის (რიცხვითი მნიშვნელობების) მიღება Volume
```

```
// enum ტიპიდან, წევრის სახელის ფორმირება და გამოტანა
foreach (byte val in Enum.GetValues(typeof(Volume)))
{
    Console.WriteLine("Volume Value: {0}\n Member: {1}",
        val, Enum.GetName(typeof(Volume), val));
}
```

4.2 ოპერატორების გადატვირთვა. ამ პარაგრაფში შეისწავლით: რა არის ოპერატორის გადატვირთვა; როდის არის მიზანშეწონილი ოპერატორის გადატვირთვა; როგორ გადატვირთოთ ოპერატორი და გაეცნობით ოპერატორების გადატვირთვის წესებს. C#-ში ოპერატორები გამოიყენებიან ბაზურ ტიპებზე მათემატიკური ოპერაციების შესასრულებლად. ოპერატორის გადატვირთვა ნიშნავს, გამოვიყენოთ კონკრეტული ოპერატორი მომხმარებლის მიერ განსაზღვრულ ტიპებზე ოპერატორის სიმბოლოს შეუცვლელად და შეცვალოთ კოდი, რომელიც ოპერატორმა უნდა შეასრულოს. ამისათვის, მომხმარებელმა უნდა შექმნას ტიპი და მასში განსაზღვროს static მეთოდი. ამ მეთოდის სახელია ოპერატორის სიმბოლო და წინ ეწერება გასაღები სიტყვა operator. პარამეტრების და დასაბრუნებელი მნიშვნელობის ტიპი უნდა იყოს მომხმარებლის მიერ განსაზღვრული ტიპი. მეთოდის ტანი უნდა იყოს ოპერატორის შესასრულებელი კოდი. ოპერატორის გადატვირთვა ყოველთვის არ არის მიზანშეწონილი. მაგალითად, თუ ოპერატორის სემანტიკა მკვეთრად განსხვავებულია გადატვირთული ოპერატორის სემანტიკისაგან. ახლა, მოვიყვანოთ ოპერატორის გადატვირთვისა და მისი გამოყენების მაგალითი:

```
using System;

class Matrix

{

        public const int DimSize = 3;

        private double[,] m_matrix = new double[DimSize, DimSize];

        // ინიციალიზაცია გამომძახებლის მიერ

        public double this[int x, int y]

        {

                get { return m_matrix[x, y]; }

                set { m_matrix[x, y] = value; }

        }
```

```
        // მატრიცების შეკრება
        public static Matrix operator +(Matrix mat1, Matrix mat2)
        {
                Matrix newMatrix = new Matrix();
                for (int x=0; x < DimSize; x++)
                        for (int y=0; y < DimSize; y++)
                                newMatrix[x, y] = mat1[x, y] + mat2[x, y];
                return newMatrix;
        }
}
class MatrixTest
{
        // მატრიცის ინიციალიზაცია შემთხვევითი რიცხვებით
        public static Random m_rand = new Random();
        // Matrix-ის ტესტირება
        static void Main()
        {
                Matrix mat1 = new Matrix();
                Matrix mat2 = new Matrix();
                // მატრიცების ინიციალიზაცია
                InitMatrix(mat1);
                InitMatrix(mat2);
```

```
        // მატრიცების გამოტანა
        Console.WriteLine("Matrix 1: ");
        PrintMatrix(mat1);
        Console.WriteLine("Matrix 2: ");
        PrintMatrix(mat2);
        // მატრიცების შეკრება და შედეგის გამოტანა
        Matrix mat3 = mat1 + mat2;
        Console.WriteLine();
        Console.WriteLine("Matrix 1 + Matrix 2 = ");
        PrintMatrix(mat3);
        Console.ReadLine();
}
//მატრიცის ინიციალიზაცია
public static void InitMatrix(Matrix mat)
{
        for (int x=0; x < Matrix.DimSize; x++)
                for (int y=0; y < Matrix.DimSize; y++)
                        mat[x, y] = m_rand.NextDouble();
}
// მატრიცის გამოტანა
public static void PrintMatrix(Matrix mat)
```

```
        {
            Console.WriteLine();
            for (int x=0; x < Matrix.DimSize; x++)
            {
                Console.Write("[ ");
                for (int y=0; y < Matrix.DimSize; y++)
                {
                    // გამოტანის დაფორმატება
                    Console.Write("{0,8:#.000000}", mat[x, y]);
                    if ((y+1 % 2) < 3)
                        Console.Write(", ");
                }
                Console.WriteLine(" ]");
            }
            Console.WriteLine();
        }
}
```

ჩ№ მოითხოვს ოპერატორების გადატვირთვის დროს შესრულდეს შემდეგი წესები: გადატვირთულმა ოპერატორმა უნდა იმუშაოს მომხმარებლის მიერ განსაზღვრულ ტიპზე; ოპერატორის გადატვირთვისას უნდა გადაიტვირთოს მისი შებრუნებული ოპერატორია galc. მაგალითად, == ოპერატორის გადატვირთვის დროს უნდა გადაიტვირთოს != ოპერატორიც; გადატვირთული ოპერატორი მუშაობს აგრეთვე მის შედგენილ ოპერატორზეც. მაგალითად, გადატვირთული + ოპერატორი მუშაობს += ოპერატორზეც.

4.3 ჩადგმა (ინკაფსულაცია). ამ პარაგრაფში თქვენ შეისწავლით: ინკაფსულაციის ობიექტზე ორიენტირებულ მეთოდს; კლასის წევრებზე წვდომის მოდიფიკატორებს; ობიექტის მდგომარეობის დაცვას თვისებებით; მეთოდებზე კონტროლირებულ წვდომას; ასამბლირებული კოდის ტიპების დაზუსტებას ინკაფსულაციით. ობიექტებს, რომლებსაც თქვენ ქმნით აქვთ მდგომარეობა ქცევა. მდგომარეობა არის ობიექტის კონკრეტული მონაცემი ან ინფორმაცია, რომელსაც იგი შეიცავს, ხოლო ქცევა ხშირად წამოდგენილია ობიექტის მეთოდებით. აგრეთვე, გათვალისწინებული უნდა იყოს, თუ როგორ იქნება გამოყენებული ობიექტი. ინკაფსულაცია არის ობიექტის შიგა სტრუქტურის, მონაცემებისა და მეთოდების რეალიზაციის დაფარვა პროგრამის სხვა ნაწილისაგან, ე.ი. ობიექტში მისთვის საჭირო ინფორმაციის ჩართვა ისე, რომ სხვა ობიექტებს არ დასჭირდეთ მისი შინაგანი სტრუქტურის ცოდნა. სხვა ობიექტების დაკავშირება მასთან ხდება მხოლოდ მისი ინტერფეისის საშუალებით. გარე კოდის კომუნიკაცია ობიექტთან შეიძლება განხორციელდეს ტიპის მეთოდებით ისე, რომ გარე კოდს არ დასჭირდეს ობიექტის შიგა სტრუქტურის ცოდნა. ობიექტის ხილვადობა იმართება ტიპსა და ტიპის წევრებზე წვდომის მოდიფიკატორებით. პირველ რიგში, გავეცნოთ ტიპის წევრებზე წვდომის მოდიფიკატორებს და როგორ გავლენას ახდენენ ისინი ხილვადობაზე. საზოგადოდ, თქვენ უნდა დაფაროთ ობიექტის შიგა მდგომარეობა გარე კოდის მიერ მასზე პირდაპირი წვდომისაგან. შემდეგ, უნდა განახორციელოთ სხვა წევრები, როგორიც არიან მეთოდები და თვისებები, რომლებიც ფუთავენ ამ მდგომარეობას. ეს საშუალებას გვაძლევს შიგა მდგომარეობა ვცვალოთ სურვილისამებრ, მაშინ როცა მეთოდები და თვისებები ფუთავენ რა შიგა მდგომარეობას, აბრუნებენ მის გარე წამოდგენას უცვლელს. გარდა ამისა, გარე კოდს არ შეუძლია გააფუჭოს ობიექტის შიგა მდგომარეობა. ობიექტის მდგომარეობის ინკაფსულაციისათვის, განვსაზღვროთ რა სახის წვდომა შეიძლება ჰქონდეს გარე კოდს ტიპის წევრებზე. ამას ადგენს ტიპის წევრებზე წვდომის მოდიფიკატორი. შემდეგ ცხრილში ჩამოთვლილია ტიპის წევრებზე წვდომის მოდიფიკატორები და მათი მნიშვნელობები.

წვდომის მოდიფიკატორი	აღწერა
private	წვდომა აქვთ მხოლოდ ამ ტიპის წევრებს. გაჩუმების პრინციპით ეს მოდიფიკატორი იგულისხმება ტიპის წევრებისათვის.

protected წვდომა აქვთ მხოლოდ ამ ტიპისა და წარმოებული ტიპის წევრებს.

internal წვდომა აქვთ ნებისმიერ კოდს ასამბლირებული კოდის შიგნით. გაჩუმების პრინციპით ეს მოდიფიკატორი იგულისხმება ტიპისათვის.

protected internal წვდომა აქვს კოდს წარმოებული ტიპიდან ან კოდს იგივე ასამბლირებული კოდიდან.

public წვდომა აქვს ნებისმიერ კოდს.

განვიხილოთ პუბლიც მოდიფიკატორის გამოყენების მაგალითი:

```
using System;

class BankAccountPublic

{

   public decimal GetAmount()

   {

      return 1000.00m;

   }

}
```

private მოდიფიკატორის გამოყენების მაგალითი:

```
class BankAccountPrivate

{

   private string m_name;

   public string CustomerName

   {
```

```
        get { return m_name; }
        set { m_name = value; }
    }
}
```

protected მეთოდის მაგალითი:

```
using System;
class BankAccountProtected
{
    public void CloseAccount()
    {
        ApplyPenalties();
        CalculateFinalInterest();
        DeleteAccountFromDB();
    }
    protected virtual void ApplyPenalties()
    {
        //  account-დან გამოყვანა
    }
    protected virtual void CalculateFinalInterest()
    {
        //  account-ის დამატება
    }
```

```
    protected virtual void DeleteAccountFromDB()
    {
        //  data entry personnel-ისთვის შეტყობინების გაგზავნა
    }
}
```

წარმოებული კლასის წევრები იყენებენ ბაზური კლასის protected წევრებს:

```
using System;
class SavingsAccount : BankAccountProtected
{
    protected override void ApplyPenalties()
    {
        Console.WriteLine("დანაზოგების ანგარიში ჯარიმების გამოყენებით");
    }
    protected override void CalculateFinalInterest()
    {
        Console.WriteLine("დანაზოგების ანგარიში საბოლოო მოგების გამოანგარიშებით ");
    }
    protected override void DeleteAccountFromDB()
    {
        base.DeleteAccountFromDB();
        Console.WriteLine("Savings Account Deleting Account from DB");
    }
```

```
}
```

წარმოებული კლასის წევრები იყენებენ ბაზური კლასის protected წევრებს:

```
using System;
class CheckingAccount : BankAccountProtected
{
   protected override void ApplyPenalties()
   {
      Console.WriteLine("Checking Account Applying Penalties");
   }
   protected override void CalculateFinalInterest()
   {
      Console.WriteLine("დანაზოგის შემოწმებასაბოლოო მოგების გამოთვლით ");
   }
   protected override void DeleteAccountFromDB()
   {
      base.DeleteAccountFromDB();
      Console.WriteLine("ანგარიშისშემოწმება და ამოგდება ბაზიდან ");
   }
}
BankAccountProtected[] bankAccts = new BankAccountProtected[2];
bankAccts[0] = new SavingsAccount();
bankAccts[1] = new CheckingAccount();
```

```
foreach (BankAccountProtected acct in bankAccts)
{
    // public method-ის გამოძახება, რომელიც იძახებს protected virtual method-ებს
    acct.CloseAccount();
}
```

კლასებს აქვთ მხოლოდ ორი წვდომის მოდიფიკატორი: internal და public. internal იგულისხმება გაჩუმებით და public ნიშნავს, რომ ამ კლასს შეიძლება მივწვდეთ ბიბლიოთეკის გარედან.

4.4 ძირითადი კოლექციები

```
public class Customer
  {
     public Customer(int id, string name)
     {
        ID = id;
        Name = name;
     }
     private int m_id;
     public int ID
     {
        get { return m_id; }
        set { m_id = value; }
     }
     private string m_name;
     public string Name
     {
        get { return m_name; }
        set { m_name = value; }
     }
  }
```

```
Dictionary<int, Customer> customers = new Dictionary<int, Customer>();
```

```
    Customer cust1 = new Customer(1, "Cust 1");
    Customer cust2 = new Customer(2, "Cust 2");
    Customer cust3 = new Customer(3, "Cust 3");
    customers.Add(cust1.ID, cust1);
    customers.Add(cust2.ID, cust2);
    customers.Add(cust3.ID, cust3);
    foreach (KeyValuePair<int, Customer> custKeyVal in customers)
    {
        Console.WriteLine(
            "Customer ID: {0}, Name: {1}",
            custKeyVal.Key,
            custKeyVal.Value.Name);
    }
```

Listing 20-1. Introduction to Using Generic Collections with an Example of the List<T> and Dictionary<TKey, TValue> Generic Collections

```
using System;
using System.Collections.Generic;
public class Customer
{
    public Customer(int id, string name)
    {
        ID = id;
```

```
        Name = name;
    }
    private int m_id;
    public int ID
    {
        get { return m_id; }
        set { m_id = value; }
    }
    private string m_name;
    public string Name
    {
        get { return m_name; }
        set { m_name = value; }
    }
}
class Program
{
    static void Main(string[] args)
    {
        List<int> myInts = new List<int>();
        myInts.Add(1);
        myInts.Add(2);
```

```
    myInts.Add(3);
    for (int i = 0; i < myInts.Count; i++)
    {
        Console.WriteLine("MyInts: {0}", myInts[i]);
    }
    Dictionary<int, Customer> customers = new Dictionary<int, Customer>();
    Customer cust1 = new Customer(1, "Cust 1");
    Customer cust2 = new Customer(2, "Cust 2");
    Customer cust3 = new Customer(3, "Cust 3");
    customers.Add(cust1.ID, cust1);
    customers.Add(cust2.ID, cust2);
    customers.Add(cust3.ID, cust3);
    foreach (KeyValuePair<int, Customer> custKeyVal in customers)
    {
        Console.WriteLine(
            "Customer ID: {0}, Name: {1}",
            custKeyVal.Key,
            custKeyVal.Value.Name);
    }
    Console.ReadKey();
}
```

4.5 ანონიმური მეთოდები.

Listing 21-1. ანონიმური მეთოდის რეალიზაცია

```
using System.Windows.Forms;

public partial class Form1 : Form

{

   public Form1()

   {

      Button btnHello = new Button();

      btnHello.Text = "სალამი";

      btnHello.Click +=

         delegate

         {

            MessageBox.Show("სალამი");

         };

      Controls.Add(btnHello);

   }

}
```

Listing 21-2. პარამეტრების გამოყენება ანონიმურ მეთოდებით

```
using System;

using System.Windows.Forms;

public partial class Form1 : Form

{
```

```
    public Form1()
    {
        Button btnHello = new Button();
        btnHello.Text = "სალამი";
        btnHello.Click +=
            delegate
            {
                MessageBox.Show("სალამი");
            };
        Button btnGoodBye = new Button();
        btnGoodBye.Text = "მშვიდობით";
        btnGoodBye.Left = btnHello.Width + 5;
        btnGoodBye.Click +=
            delegate(object sender, EventArgs e)
            {
                string message = (sender as Button).Text;
                MessageBox.Show(message);
            };
        Controls.Add(btnHello);
        Controls.Add(btnGoodBye);
    }
}
```

4.6 C#-ის ტიპების შესახებ. ამ პარაგრაფში თქვენ შეისწავლით: რისთვის არის საჭირო ტიპების უსაფრთხოება; როგორ გარდაქმნათ ერთი ტიპი მეორეში; უფრო დეტალურად შეისწავლით მიმთითებლიან და მნიშვნელობიან ტიპებს და შინაარსობლივ განსხვავებას მათ შორის. თუ ერთი ტიპის ცვლადს მიანიჭებთ მეორე ტიპის ცვლადს ასეთ შემთხვევაში შეიძლება წარმოიშვას პრობლემები. ზოგი პრობლემა შეიძლება იყოს აცილებადი და ზოგი არა. აცილებადი პრობლემის შემთხვევაში იყენებენ არაცხად ან ცხად გარდაქმნებს. ცხადი გარდაქმნის შემთხვევაში იყენებენ გარდაქმნის ოპერატორს (X), ხოლო არაცხადი გარდაქმნა ნიშნავს, რომ ცხადი გარდაქმნის ოპერატორი არაა გამოყენებული და ასეთ შემთხვევაში კომპილატორი ცდილობს თვითონ შეასრულოს გარდაქმნა და თუ ეს შეუძლებელია კომპილატორი იძლევა შეცდომის შეტყობინებას. ზოგჯერ ცხადი გარდაქმნა არაა მიზანშეწონილი, რადგან შეიძლება გამოიწვიოს სიზუსტის დაკარგვა. მაგალითად, double ტიპის მნიშვნელობის int ტიპის მნიშვნელობად გარდაქმნის დროს. განვიხილოთ ცხადი და არაცხადი გარდაქმნის მაგალითი:

```
using System;

class Program

{

   static void Main()

   {

      float lengthFloat = 7.35f;

      // სიზუსტის დაკარგვის შესაძლებლობა

      int lengthInt = (int)lengthFloat;

      // უპრობლემო არაცხადი გარდაქმნა

      double lengthDouble = lengthInt;

      Console.WriteLine("lengthInt =

         " + lengthInt);
```

```
        Console.WriteLine("lengthDouble = " + lengthDouble);
        Console.ReadKey();
    }
}
```

ერთი მიმთითებლის მინიჭებისას მეორესთვის ხდება მისამართების და არა ობიექტების კოპირება. განვიხილოთ მაგალითი:

```
using System;
class Employee
{
    private string m_name;
    public string Name
    {
        get { return m_name; }
        set { m_name = value; }
    }
}
class Program
{
    static void Main()
    {
        Employee joe = new Employee();
        joe.Name = "Joe";
```

```
        Employee bob = new Employee();

        bob.Name = "Bob";

        Console.WriteLine("Original Employee Values:");

        Console.WriteLine("joe = " + joe.Name);

        Console.WriteLine("bob = " + bob.Name);

            // ცვლადისათვის მიმთითებლის მინიჭება

        bob = joe;

        Console.WriteLine();

        Console.WriteLine("Values After Reference Assignment:");

        Console.WriteLine("joe = " + joe.Name);

        Console.WriteLine("bob = " + bob.Name);

        joe.Name = "Bobbi Jo";

        Console.WriteLine();

        Console.WriteLine("Values After Changing One Instance:");

        Console.WriteLine("joe = " + joe.Name);

        Console.WriteLine("bob = " + bob.Name);

        Console.ReadKey();

    }

}
```

მნიშვნელობიანი ტიპის ცვლადების ერთიმეორეზე მინიჭებისას ხდება ობიექტის მნიშნელობის კოპირება და ორივე ცვლადი მიიღებს ერთიდაიგივე მნიშვნელობას. განვიხილოთ მაგალითი:

```
using System;

struct Height

{

   private int m_inches;

   public int Inches

   {

      get { return m_inches; }

      set { m_inches = value; }

   }

}

class Program

{

   static void Main()

   {

      Height joe = new Height();

      joe.Inches = 71;

      Height bob = new Height();

      bob.Inches = 59;

      Console.WriteLine("საწყისი სიმაღლის

         მნიშვნელობები:");

      Console.WriteLine("joe = " + joe.Inches);

      Console.WriteLine("bob = " + bob.Inches);
```

```
            bob = joe;
        Console.WriteLine();
        Console.WriteLine("Values After
           Value Assignment:");
        Console.WriteLine("joe = " + joe.Inches);
        Console.WriteLine("bob = " + bob.Inches);
        joe.Inches = 65;
        Console.WriteLine();
        Console.WriteLine("Values After
           Changing One Instance:");
        Console.WriteLine("joe = " + joe.Inches);
        Console.WriteLine("bob = " + bob.Inches);
        Console.ReadKey();
    }
}
```

4.7 null ტიპის გამოყენება. ამ პარაგრაფში შეისწავლით: რა არის ცარიელმნიშვნელობიანი ტიპი;

როგორ გამოაცხადოთ და გამოიყენოთ იგი. პრობლემა მდგომარეობს იმაში, რომ როცა რაიმე ცვლადს არა აქვს მინიჭებული მნიშვნელობა და მას იყენებთ გამოსახულებაში, ასეთ შემთხვევაში თვლის დროს წამოიშვება შეცდომა და თქვენი პროგრამა განიცდის წყვეტას. ამის აცილება იქნებოდა შესაძლებელი, თქვენ რომ შეგძლებოდათ შეგემოწმებინათ აქვს თუ არა ამ ცვლადს მნიშვნელობა. ამის საშუალებას იძლევა ცარიელმნიშვნელობიანი (null) ტიპის შემოტანა C#-ში. მაგალითად, ცარიელმნიშვნელობიანი მთელი ტიპი ნიშნავს, რომ მთელ რიცხვთა სიმრავლეს ემატება სპეციალური მნიშვნელობა null, რომელიც აღიშნავს, რომ ცვლადს არა აქვს მინიჭებული მნიშვნელობა. მაგალითად, იგი გვეხმარება დავადგინოთ მთელი ტიპის ცვლადს აქვს თუ არა მნიშვნელობა და ამით ავიცილოთ ცვლადის გამოყენება გამოსახულებაში, როცა მას არ აქვს მნიშვნელობა. ცარიელმნიშვნელობიანი ტიპის აღსანიშნავად ტიპის სახელს ბოლოში მიეწერება ? ნიშანი. განვიხილოთ მაგალითი:

```
 int a;

int? b;

   if (b == null)

   {

      a = 0;

   }

   else

   {

      a = (int)b;

   }
```

ამ მაგალითში, ა-ს ენიჭება ბ, თუ ბ-ს არა აქვს null მნიშვნელობა, წინააღმდეგ შემთხვევაში ნული. შევნიშნოთ, რომ ცარიელმნიშვნელობიანი ცვლადის

არაცარიელმნიშვნელობიანი შესატყვისი ტიპის ცვლადზე მინიჭებისას აუცილებელია ცხადი გარდაქმნა. განსაკუთრებით მოსახერხებელია ცარიელმნიშვნელობიანი ტიპების გამოყენება მონაცემთა ბაზებთან მუშაობისას. მონაცემთა ბაზის ცარიელმნიშვნელობიან ტიპს უნდა შეუთანადოთ C#-ში ცარიელმნიშვნელობიანი ტიპი. ამით მოგეცემათ საშუალება აიცილოთ პროგამის წყვეტა, როცა მონაცემთა ბაზის ჩანაწერის ველს აქვს ცარიელი მნიშვნელობა. თქვენ შეგიძლიათ ცვლადს მიანიჭოთ ცარიელი მნიშვნელობა:

```
int? a = null;
```

an kidev

```
bull c = a == null;
```

C#-ში არსებობს სპეციალური ორადგილიანი ოპერატორი ?? , რომელიც მუშაობს ასე: თუ მარცხენა ოპერანდის მნიშვნელობაა null, მაშინ გამოითვლება მარჯვენა ოპერანდი, სხვა შემთხვევაში მარცხენა ოპერანდი. მაგალითად:

```
int b = a ?? 0;
```

sadac

```
int? a;
```

ამგვარად, C#-ის ცარიელმნიშვნელობიანი ტიპი გვეხმარება მთელ რიგ შემთხვევებში შევადგინოთ ეფექტური პროგრამა.

დანართი 1

სპეციალური პროგრამების მაგალითები

1. ფაილის შექმნა და წაკითხვა

```
using System;
using System.IO;
class SavarjiSo
{
   public static void Main()
   {
      string path = @"c:\temp\savarjiSo.txt";
      if (!File.Exists(path))
      {
         // ფაილის შექმნა
         using (StreamWriter sw = File.CreateText(path))
         {
            sw.WriteLine("ეს არის საცდელი ფაილი");
            sw.WriteLine("შექმენი,");
            sw.WriteLine("შემდეგ წაიკითხე");
         }
      }
      // ფაილის გახსნა წასაკითხად
```

```
        using (StreamReader sr = File.OpenText(path))
        {
            string s = "";
            while ((s = sr.ReadLine()) != null)
            {
                Console.WriteLine(s);
            }
        }
    }
}
```

2. მასივის გამოყენება

```
// arrays.cs

using System;

class DeclareArraysSample

{

    public static void Main()

    {

        //  ერთგანზომილებიანი  მასივი

        int[] numbers = new int[5];

        //  მრავალგანზომილებიანი  მასივი

        string[,] names = new string[5,4];

        // მასივების მასივი (დაკბილული მასივი)

        byte[][] scores = new byte[5][];

        // დაკბილული მასივის სიგრძის გამოტანა

        for (int i = 0; i < scores.Length; i++)

        {

            scores[i] = new byte[i+3];

        }

        // მასივის სტრიქონის სიგრძის გამოტანა

        for (int i = 0; i < scores.Length; i++)

        {

            Console.WriteLine("{0} სტრიქონის სიგრძეა {1}", i, scores[i].Length);
```

```
        }

    }

}
```

3. ატრიბუტების გამოყენება

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// AttributesTutorial.cs

// ეს მაგალითი გვიჩვენებს კლასისა და მეთოდის ატრიბუტების გამოყენებას.

using System;

using System.Reflection;

using System.Collections;

//  IsTested არის მომხმარებლის მიერ განსაზღვრული custom attribute

// class.

// იგი შეიძლება იყოს გამოყენებული ნებისმიერ გამოცხადებაში struct, class,

// enum, delegate ტიპების ჩათვლით.

//  methods, fields, events, properties, indexers წევრები არიან გამოყენებული
არგუმენტების გარეშე.

public class IsTestedAttribute : Attribute

{

   public override string ToString()

   {

      return "ტესტირებულია";

   }

}
```

```
// AuthorAttribute class არის მომხმარებლის მიერ გამოცხადებული
// attribute class.
// იგი შეიძლება იყოს გამოყენებული კლასებისა და სტრუქტურების გამოცხადებებში
// მხოლოდ. იგი იღებს უსახელო სტრიქონული ტიპის არგუმენტსდა აქვს
// არასავალდებულო სახელიანი მთელი ტიპის არგუმენტი
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct)]
public class AuthorAttribute : Attribute
{
   // ეს კონსტრუქტორი აზუსტებს უსახელო არგუმენტებს attribute class-ში.
   public AuthorAttribute(string name)
   {
      this.name = name;
      this.version = 0;
   }
   // This property is readonly (it has no set accessor)
   // so it cannot be used as a named argument to this attribute.
   public string Name
   {
      get
      {
         return name;
      }
```

```
    }
    // ეს თვისება არის read-write (მას აქვს set accessor)
    // ამგვარად, იგი შეიძლება იყოს გამოყენებული, როგორც სახელიანი არგუმენტი,
// როცა ვიყენებთ ამ კლასს როგორც attribute კლასი.
    public int Version
    {
        get
        {
            return version;
        }
        set
        {
            version = value;
        }
    }
    public override string ToString()
    {
        string value = "Author : " + Name;
        if (version != 0)
        {
            value += " Version : " + Version.ToString();
        }
```

```
        return value;
    }
    private string name;
    private int version;
}
 [Author("Joe Programmer")]
class Account
{
    // IsTestedAttribute custom attribute-ის დართვა მეთოდზე.    [IsTested]
    public void AddOrder(Order orderToAdd)
    {
        orders.Add(orderToAdd);
    }

    private ArrayList orders = new ArrayList();
}

// AuthorAttribute და IsTestedAttribute custom attributes დართვა
// კლასზე.
// to this class.
// შევნიშნოთ 'Version' სახელიანი არგუმენტის გამოყენება .
[Author("Jane Programmer", Version = 2), IsTested()]
```

```
class Order
{
    // add stuff here ...
}
class MainClass
{
   private static bool IsMemberTested(MemberInfo member)
   {
       foreach (object attribute in member.GetCustomAttributes(true))
       {
          if (attribute is IsTestedAttribute)
          {
             return true;
          }
       }
     return false;
   }
    private static void DumpAttributes(MemberInfo member)
    {
       Console.WriteLine("Attributes for : " + member.Name);
       foreach (object attribute in member.GetCustomAttributes(true))
       {
```

```
        Console.WriteLine(attribute);
    }
}
public static void Main()
{
    // გამოაქვს ატრიბუტები  Account class-ისათვის.
    DumpAttributes(typeof(Account));
    // გამოაქვს ტესტირებული წევრების სია.
    foreach (MethodInfo method in (typeof(Account)).GetMethods())
    {
        if (IsMemberTested(method))
        {
            Console.WriteLine("წევრი {0} ტესტირებულია!", method.Name);
        }
        else
        {
            Console.WriteLine("წევრი {0} არაა ტესტირებული!", method.Name);
        }
    }
    Console.WriteLine();
    // გამოაქვს ატრიბუტები  Order class-ისათვის.
    DumpAttributes(typeof(Order));
```

```
        // გამოაქვს ატრიბუტები მეთოდებისათვის    Order class-ში.
        foreach (MethodInfo method in (typeof(Order)).GetMethods())
        {
          if (IsMemberTested(method))
          {
             Console.WriteLine("წევრი {0} ტესტირებულია!", method.Name);
          }
          else
          {
             Console.WriteLine("წევრი {0} არაა ტესტირებული!", method.Name);
          }
        }
        Console.WriteLine();
    }
}
```

4. სიმბოლოების გამოცნობა

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// tokens.cs

using System;

// გავხადოთ შესაძლებელი პროგრამაში System.Collection  namespace-ის

// გამოყენება.

using System.Collections;

// გამოვაცხადოთ Tokens class, როგორც IEnumerable-ის ქვეკლასი:

public class Tokens : IEnumerable

{

//  შემოგვაქვს სტრიქონული ტიპის ერთგანზომილებიანი მასივი

  private string[] elements;

  Tokens(string source, char[] delimiters)

  {

    // გადავაქციოთ სტრიქონი სიმბოლოდ:

    elements = source.Split(delimiters);

  }

  // IEnumerable Interface-ის რეალიზაცია:

  // GetEnumerator() მეთოდის გამოცხადება,

  //   რომელიც მოითხოვება  IEnumerable-ისთვის

  public IEnumerator GetEnumerator()

  {
```

```
    return new TokenEnumerator(this);
  }
  // ახორციელებს  IEnumerator interface-ს:
  private class TokenEnumerator : IEnumerator
  {
    private int position = -1;
    private Tokens t;
    public TokenEnumerator(Tokens t)
    {
      this.t = t;
    }
    // MoveNext მეთოდის გამოცხადება, რომელიც მოითხოვება                    //
IEnumerator -ისთვის:
    public bool MoveNext()
    {
      if (position < t.elements.Length - 1)
      {
        position++;
        return true;
      }
      else
      {
```

```
            return false;
        }
    }
    //  Reset მეთოდის გამოცხადება, რომელიც მოითხოვება
// IEnumerator-ისთვის:
    public void Reset()
    {
        position = -1;
    }
    //  Current თვისების გამოცხადება, რომელიც მოითხოვება
// IEnumerator-ისთვის:
    public object Current
    {
        get
        {
            return t.elements[position];
        }
    }
}
// სიმბოლოების ტესტირება
static void Main()
{
```

```
    // სიმბოლოების ტესტირება სტრიქონის სიმბოლოებად დახლეჩით:
    Tokens f = new Tokens("ესაა კარგად დაწერილი პროგრამა.",
       new char[] {' ','-'});
    foreach (string item in f)
    {
       Console.WriteLine(item);
    }
  }
}
```

5. სტრუქტურების გამოყენება

```
        //Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// struct1.cs
using System;
struct SimpleStruct
{
   private int xval;
   public int X
   {
      get
      {
         return xval;
      }
      set
      {
         if (value < 100)
            xval = value;
      }
   }
   public void DisplayX()
   {
      Console.WriteLine("დამახსოვრებული მნიშვნელობა: {0}", xval);
```

```
    }
}


class TestClass
{
    public static void Main()
    {
        SimpleStruct ss = new SimpleStruct();
        ss.X = 5;
        ss.DisplayX();
    }
}
```

6. თვისებების გამოყენება

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// person.cs

using System;

class Person

{

   private string myName ="N/A";

   private int myAge = 0;

   // string ტიპის თვისების გამოცხადება

   public string Name

   {

      get

      {

         return myName;

      }

      set

      {

         myName = value;

      }

   }

   // int ტიპის თვისების გამოცხადება:

   public int Age
```

```
{
    get
    {
        return myAge;
    }
    set
    {
        myAge = value;
    }
}
public override string ToString()
{
    return "Name = " + Name + ", Age = " + Age;
}
public static void Main()
{
    Console.WriteLine("მარტივი თვისებები");
    // ახალი ობიექტის შექმნა
    Person person = new Person();
    // name და age მნიშვნელობების გამოტანა
    Console.WriteLine("Person details - {0}", person);
    // person ობიექტის მნიშვნელობის დაყენება
```

```
        person.Name = "Joe";
        person.Age = 99;
        Console.WriteLine("პიროვნების დეტალები - {0}", person);
        // Age თვისების მნიშვნელობის გაზრდა
        person.Age += 1;
        Console.WriteLine("პიროვნების დეტალები  - {0}", person);
    }
}
```

7. მაინდექსირებლების გამოყენება

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// indexer.cs

// არგუმენტები: indexer.txt

using System;

using System.IO;

// Class, რომელიც გულისხმობს დიდ მასივზე წვდომას, თითქოს ის იყოს ბაიტების

// მასივი

public class FileByteArray

{

   Stream stream;

   public FileByteArray(string fileName)

   {

      stream = new FileStream(fileName, FileMode.Open);

   }

   // ნაკადის დახურვა

   public void Close()

   {

      stream.Close();

      stream = null;

   }

   // მაიდექსირებელი read/write წვდომისათვის ფაილზე.
```

```
  public byte this[long index]
  {
     // ბაიტის წაკითხვა offset index-ზე და მისი დაბრუნება
get
     {
        byte[] buffer = new byte[1];
        stream.Seek(index, SeekOrigin.Begin);
        stream.Read(buffer, 0, 1);
        return buffer[0];
     }
     // ბაიტის ჩაწერა  offset index-ში და მისი დაბრუნება.
     set
     {
        byte[] buffer = new byte[1] {value};
        stream.Seek(index, SeekOrigin.Begin);
        stream.Write(buffer, 0, 1);
     }
  }
  // ფაილის სიგრძის მიღება.
  public long Length
  {
     get
```

```
        {
            return stream.Seek(0, SeekOrigin.End);
        }
    }
}
// FileByteArray class-ის დემონსტრაცია.
// ფაილში ბაიტების შებრუნება.
public class Reverse
{
    public static void Main(String[] args)
    {
                    // არგუმენტების არსებობის შემოწმება.
                    if (args.Length != 1)
                    {
                            Console.WriteLine("გამოყენება : Indexer <filename>");
                            return;
                    }
                    // ფაილის არსებობის შემოწმება.
                    if (!System.IO.File.Exists(args[0]))
                    {
                            Console.WriteLine("ფაილი " + args[0] + " ვერ იპოვნა.");
                            return;
```

```
                }

                FileByteArray file = new FileByteArray(args[0]);

                long len = file.Length;

                for (long i = 0; i < len / 2; ++i)

                {

                        byte t;

                        // შევნიშნოთ, რომ  "file" ცვლადი იძახებს

// მაინდექსირებელს FileByteStream კლასში, რომელიც წერს ბაიტებს ფაილში.

                                                t = file[i];

                        file[i] = file[len - i - 1];

                        file[len - i - 1] = t;

                }

                file.Close();

    }

}
```

8. დელეგატების გამოყენება

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// bookstore.cs

using System;

// კლასების სიმრავლე წიგნების საწყობის დასამუშავებლად.

namespace Bookstore

{

  using System.Collections;

  // აღწერს წიგნს წიგნების სიაში:

  public struct Book

  {

    public string Title;       // წიგნის სათაური

    public string Author;      // სხვა წიგნი

    public decimal Price;      // წიგნის ფასი

    public bool Paperback;

    public Book(string title, string author, decimal price, bool paperBack)

    {

      Title = title;

      Author = author;

      Price = price;

      Paperback = paperBack;

    }
```

```
}

//   delegate-ის გამოცხადება  წიგნის დასამუშავებლად.

public delegate void ProcessBookDelegate(Book book);

// წიგნების მონაცემთა ბაზა

public class BookDB

{

  // წიგნების სია მონაცემთა ბაზაში.

  ArrayList list = new ArrayList();

  // ბაზაში წიგნის ჩამატება

  public void AddBook(string title, string author, decimal price, bool paperBack)

  {

    list.Add(new Book(title, author, price, paperBack));

  }

  //  delegate-ს გამოძახება წიგნის დასამუშავებლად

  public void ProcessPaperbackBooks(ProcessBookDelegate processBook)

  {

    foreach (Book b in list)

    {

      if (b.Paperback)

      // დელეგატის გამოძახება

        processBook(b);

    }
```

```
    }
  }
}
//   Bookstore კლასის გამოყენება
namespace BookTestClient
{
  using Bookstore;
  // Class წიგნების საერთო და საშუალო ფასებისათვის
  class PriceTotaller
  {
    int countBooks = 0;
    decimal priceBooks = 0.0m;
    internal void AddBookToTotal(Book book)
    {
      countBooks += 1;
      priceBooks += book.Price;
    }
    internal decimal AveragePrice()
    {
      return priceBooks / countBooks;
    }
  }
```

```
  // კლასი წიგნების მონაცემთა ბაზის ტესტირებისათვის.
  class Test
  {
    // წიგნის სათაურის გამოტანა.
    static void PrintTitle(Book b)
    {
      Console.WriteLine("   {0}", b.Title);
    }
    // აქ იწყება გამოთვლა.
    static void Main()
    {
      BookDB bookDB = new BookDB();
      // მონაცემთა ბაზის ინიციალიზაცია
      AddBooks(bookDB);
      // წიგნების საშაურების გამოტანა
      Console.WriteLine("Paperback Book Titles:");
      // ახალი delegate ობიექტის შექმნა, რომელიც დაკავშირებულია
// Test.PrintTitle მეთოდთან.
      bookDB.ProcessPaperbackBooks(new ProcessBookDelegate(PrintTitle));
      // რბილყდიანი წიგნების საშუალო ფასის მიღება მთლიანი ფასის
// ობიექტის გამოყენებით
      PriceTotaller totaller = new PriceTotaller();
```

```
        // ახალი დელეგატი ობიექტის შექმნა
        bookDB.ProcessPaperbackBooks(new ProcessBookDelegate(totaller.AddBookToTotal));
        Console.WriteLine("Average Paperback Book Price: ${0:#.##}",
          totaller.AveragePrice());
      }
      // წიგნების მონაცემთა ბაზის ინიციალიზაცია )ზოგიერთი სატესტო
// წიგნებით
      static void AddBooks(BookDB bookDB
      {
        bookDB.AddBook("C დაპროგრამების ენა",
          "Brian W. Kernighan and Dennis M. Ritchie", 19.95m, true);
        bookDB.AddBook("The Unicode Standard 2.0",
          "The Unicode Consortium", 39.95m, true);
        bookDB.AddBook("The MS-DOS Encyclopedia",
          "Ray Duncan", 129.95m, false);
        bookDB.AddBook("Dogbert's Clues for the Clueless",
          "Scott Adams", 12.00m, true);
      }
    }
}
  int availableUnits;
    if (unitsInStock == null)
```

```
{
    availableUnits = 0;
}
else
{
    availableUnits = (int)unitsInStock;
}
```

9. ველები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

using System;

using System.Collections.Generic;

using System.Text;

namespace Yield

{

   class Yield

   {

      public static class NumberList

      {

         // მთელი ტიპის მასივის შექმნა.

         public static int[] ints = { 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377 };

         // თვისების განსაზღვრა, რომელიც აბრუნებს მხოლოდ წყვილ

// რიცხვებს.

         public static IEnumerable<int> GetEven()

         {

            //ველის განსაზღვრა სიაში წყვილი რიცხვების დასაბრუნებლად

foreach (int i in ints)

               if (i % 2 == 0)

                  yield return i;

         }
```

```
        // თვისების განსაზღვრა. რომელიც აბრუნებს მხოლოდ წყვილ
// რიცხვებს.
        public static IEnumerable<int> GetOdd()
        {
            // ველის განსაზღვრა მხოლოდ კენტი რიცხვების
// დასაბრუნებლად.
            foreach (int i in ints)
                if (i % 2 == 1)
                    yield return i;
        }
    }
    static void Main(string[] args)
    {
        // ლუწი რიცხვების გამოტანა.
        Console.WriteLine("ლუწი რიცხვები");
        foreach (int i in NumberList.GetEven())
            Console.WriteLine(i);
        // წყვილი რიცხვების გამოტანა.
        Console.WriteLine("Odd numbers");
        foreach (int i in NumberList.GetOdd())
            Console.WriteLine(i);
    }
```

```
    }

}
```

10. გარდაქმნები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// conversion.cs

using System;

struct RomanNumeral

{

    public RomanNumeral(int value)

    {

       this.value = value;

    }

    // მთელი რიცხვის რომაულ ციფრებში გადაყვანა. ეს არის გარდაქმნის

// ოპერატორი სახელით RomanNumeral:

    static public implicit operator RomanNumeral(int value)

    {

       return new RomanNumeral(value);

    }

    // ცხადი გარდაქმნის გამოცხადება  RomanNumeral-დან მთელ რიცხვად.

 static public explicit operator int(RomanNumeral roman)

    {

       return roman.value;

    }
```

```
    // RomanNumeral-დან სტრიქონად არაცხადი გარდაქმნის გამოცხადება.
    static public implicit operator string(RomanNumeral roman)
    {
       return("გარდაქმნა ჯერ არაა რეალიზებული ");
    }
    private int value;
}
class Test
{
    static public void Main()
    {
        RomanNumeral numeral;
        numeral = 10;
        Console.WriteLine((int)numeral);
// სტრიქონად არაცხადი გარდაქმნის გამოძახება.
        Console.WriteLine(numeral);
// ცხადი გარდაქმნის გამოძახება ციფრიდან მთელში და შემდეგ ცხადი გარდაქმნა
// მთელიდან  short-ში.
        short s = (short)numeral;
        Console.WriteLine(s);
    }
}
```

```
 //Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// structconversion.cs
using System;
struct RomanNumeral
{
   public RomanNumeral(int value)
   {
      this.value = value;
   }
   static public implicit operator RomanNumeral(int value)
   {
      return new RomanNumeral(value);
   }
   static public implicit operator RomanNumeral(BinaryNumeral binary)
   {
      return new RomanNumeral((int)binary);
   }
   static public explicit operator int(RomanNumeral roman)
   {
       return roman.value;
   }
   static public implicit operator string(RomanNumeral roman)
```

```
    {
        return("Conversion not yet implemented");
    }
    private int value;
}
struct BinaryNumeral
{
    public BinaryNumeral(int value)
    {
        this.value = value;
    }
    static public implicit operator BinaryNumeral(int value)
    {
        return new BinaryNumeral(value);
    }
    static public implicit operator string(BinaryNumeral binary)
    {
        return("Conversion not yet implemented");
    }
    static public explicit operator int(BinaryNumeral binary)
    {
        return(binary.value);
```

```
    }
    private int value;
}
class Test
{
    static public void Main()
    {
        RomanNumeral roman;
        roman = 10;
        BinaryNumeral binary;
        // RomanNumeral-დან  BinaryNumeral-ში გარდაქმნის შესრულება.
        binary = (BinaryNumeral)(int)roman;
        // BinaryNumeral-დან RomanNumeral-ში გარდაქმნის შესრულება.
              roman = binary;
        Console.WriteLine((int)binary);
        Console.WriteLine(binary);
    }
}
```

11. ოპერატორების გადატვირთვა

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// dbbool.cs

using System;

public struct DBBool

{

   // სამი შესაძლებელი DBBool მნიშვნელობები.

   public static readonly DBBool dbNull = new DBBool(0);

   public static readonly DBBool dbFalse = new DBBool(-1);

   public static readonly DBBool dbTrue = new DBBool(1);

   // Private ველი, რომელიც ინახავს -1, 0, 1 dbFalse, dbNull, dbTrue-

// სთვის.

   int value;

   // Private constructor. value პარამეტრი უნდა იყოს -1, 0, ან 1:

   DBBool(int value)

   {

      this.value = value;

   }

   // არაცხადი გარდაქმნა bool-დან DBBool-ში. გადასახავს true-ს

   // DBBool.dbTrue-ში და false-ს DBBool.dbFalse-ში.

   public static implicit operator DBBool(bool x)
```

```
    {
        return x? dbTrue: dbFalse;
    }
    // ცხადი გარდაქმნა DBBool-დან bool-ში. იწვევს განსაკუთრებულ
// შემთხვევას  თუ  მოცემული DBBool არის dbNull, სხვა შემთხვევებში აბრუნებს
    // true ან false-ს.
    public static explicit operator bool(DBBool x)
    {
        if (x.value == 0) throw new InvalidOperationException();
        return x.value > 0;
    }
    // ექვივალენტობის ოპერატორი აბრუნებს dbNull-ს, თუ რომელიმე ოპერანდი
//  არის dbNull, სხვა შემთხვევებში აბრუნებს dbTrue-ს ან dbFalse-ს.
        public static DBBool operator ==(DBBool x, DBBool y)
    {
        if (x.value == 0 || y.value == 0) return dbNull;
        return x.value == y.value? dbTrue: dbFalse;
    }
    // არაექვივალენტობის ოპერატორი აბრუნებს dbNull-ს, თუ რომელიმე
// ოპერანდი არის
    // dbNull, სხვა შემთხვევებში აბრუნებს dbTrue-ს ან dbFalse-ს:
    public static DBBool operator !=(DBBool x, DBBool y)
```

```
    {
      if (x.value == 0 || y.value == 0) return dbNull;
      return x.value != y.value? dbTrue: dbFalse;
    }
    // ლოგიკური უარყოფის ოპერატორი აბრუნებს dbTrue-ს, თუ ოპერანდი არის
    // dbFalse, dbNull თუ ოპერანდი არის dbNull, ან dbFalse თუ
    // ოპერანდი არის dbTrue:
    public static DBBool operator !(DBBool x)
    {
      return new DBBool(-x.value);
    }
    // ლოგიკური AND ოპერატორი. აბრუნებს dbFalse თუ რომელიმე
  // ოპერანდებიდან არის
    // dbFalse, აბრუნებს dbNull თუ რომელიმე ოპერანდი არის dbNull, სხვა
  // შემთხვევებში აბრუნებს dbTrue:
    public static DBBool operator &(DBBool x, DBBool y)
    {
      return new DBBool(x.value < y.value? x.value: y.value);
    }
    // ლოგიკური OR ოპერატორი აბრუნებს dbTrue-ს, თუ რომელიმე ოპერანდი
  // არის
    // dbTrue; აბრუნებს dbNull-ს, თუ რომელიმე ოპერანდი არის dbNull, სხვა
```

```
// შემთხვევებში აბრუნებს
// dbFalse:
  public static DBBool operator |(DBBool x, DBBool y)
  {
    return new DBBool(x.value > y.value? x.value: y.value);
  }
  // და ბოლოს true ოპერატორი აბრუნებს true-ს, თუ ოპერანდი არის
  // dbTrue, false-ს სხვა შემთხვევებში.
  public static bool operator true(DBBool x)
  {
    return x.value > 0;
  }
  // false ოპერატორი აბრუნებს true-ს, თუ ოპერანდი არის
  // dbFalse, false-ს სხვა შემთხვევებში.
  public static bool operator false(DBBool x)
  {
    return x.value < 0;
  }
  // გარდაქმნის გადატვირთვა DBBool-დან string-ში:
  public static implicit operator string(DBBool x)
  {
    return x.value > 0 ? "dbTrue"
```

```
        : x.value < 0 ? "dbFalse"
        : "dbNull";
}
// Object.Equals(object o) method-ის გადაფარვა.
public override bool Equals(object o)
{
  try
  {
    return (bool) (this == (DBBool) o);
  }
  catch
  {
    return false;
  }
}
// Object.GetHashCode() method-ის გადაფარვა:
public override int GetHashCode()
{
  return value;
}
// ToString method-ის გადაფარვა.
public override string ToString()
```

```
  {
    switch (value)
    {
      case -1:
        return "DBBool.False";
      case 0:
        return "DBBool.Null";
      case 1:
        return "DBBool.True";
      default:
        throw new InvalidOperationException();
    }
  }
}
class Test
{
  static void Main()
  {
    DBBool a, b;
    a = DBBool.dbTrue;
    b = DBBool.dbNull;
    Console.WriteLine( "!{0} = {1}", a, !a);
```

```
      Console.WriteLine( "!{0} = {1}", b, !b);
      Console.WriteLine( "{0} & {1} = {2}", a, b, a & b);
      Console.WriteLine( "{0} | {1} = {2}", a, b, a | b);
      // true ოპერატორის გამოძახება, რომ განვსაზღვროთ DBBool ცვლადის
//  ბულის მნიშვნელობა.
      if (b)
         Console.WriteLine("b is definitely true");
      else
         Console.WriteLine("b არაა true");
   }
}
using System; //Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// complex.cs

public struct Complex
{
   public int real;
   public int imaginary;
   public Complex(int real, int imaginary)
   {
      this.real = real;
      this.imaginary = imaginary;
```

```
  }
  public static Complex operator +(Complex c1, Complex c2)
  {
    return new Complex(c1.real + c2.real, c1.imaginary + c2.imaginary);
  }
  // ToString მეთოდის გადაფარვა, რომ გამოვიტანოთ კომპლექსური რიცხვი
// სასურველ ფორმატში.
 public override string ToString()
  {
    return(String.Format("{0} + {1}i", real, imaginary));
  }
  public static void Main()
  {
    Complex num1 = new Complex(2,3);
    Complex num2 = new Complex(3,4);
    // ორი კომპლექსური ოპერატორის (num1 და num2) დამატება
// გადატვირთული plus  ოპერატორით
    Complex sum = num1 + num2;
   // რიცხვების დაბეჭდვა და ჯამი გადაფარული ToString  მეთოდის
// გამოყენებით
Console.WriteLine("პირველი კომპლექსური რიცხვი:  {0}",num1);
    Console.WriteLine("მეორე კომპლექსური რიცხვი: {0}",num2);
```

```
        Console.WriteLine("ორი რიცხვის ჯამი: {0}",sum);

    }

}
```

12. nul ტიპები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

using System;

class NullableOperator

{

   static int? GetNullableInt()

   {

      return null;

   }

   static string GetStringValue()

   {

      return null;

   }

   static void Main()

   {

      // ?? ოპერატორის მაგალითი.

      int? x = null;

      // y = x, წინააღმდეგ შემთხვევაში x არის ნული და მაშინ  y = -1.

      int y = x ?? -1;

      Console.WriteLine("y == " + y);

      int i = GetNullableInt() ?? default(int);
```

```
        Console.WriteLine("i == " + i);
        // ?? გამოიყენება აგრეთვე ტიპებზე მიმთითებელი
string s = GetStringValue();
        // გამოვიტანოთ s-ის მნიშვნელობა, თუ არ არის ნული. თუ ნულია, მაშინ
// გამოვიტანოთ "Unspecified".
        Console.WriteLine("s = {0}", s ?? "null");
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
using System;
class NullableBoxing
{
    static void Main()
    {
        int? a;
        object oa;
        // Assigns a to Nullable<int> (value = default(int),
// hasValue = false).
        a = null;
        // მივანიჭოთ oa-ს ნული (რადგანაც x==null), არაბოქსირებული
// მთელი "int?".
        oa =  Console.WriteLine("Testing 'a' and 'boxed a' for null...");
```

```
    // ნულმნიშვნელობიანი ცვლადები შეიძლება შევადაროთ ნულს.
if (a == null)
    {
        Console.WriteLine("  a == null");
    }
    if (oa == null)
    {
        Console.WriteLine("  oa == null");
    }
    Console.WriteLine("Unboxing a nullable type...");
    int? b = 10;
    object ob = b;
    int? unBoxedB = (int?)ob;
    Console.WriteLine("  b={0}, unBoxedB={0}", b, unBoxedB);
    int? unBoxedA = (int?)oa;
    if (oa == null && unBoxedA == null)
    {
        Console.WriteLine("  a და unBoxedA არიან null");
    }
    Console.WriteLine("Attempting to unbox into non-nullable type...");
    try
    {
```

```
            int unBoxedA2 = (int)oa;
        }
        catch (Exception e)
        {
            Console.WriteLine("  {0}", e.Message);
        }
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
using System;
class NullableBasics
{
    static void DisplayValue(int? num)
    {
        if (num.HasValue == true)
        {
            Console.WriteLine("num = " + num);
        }
        else
        {
            Console.WriteLine("num = null");
        }
```

```
        // num.Value-ს გამოაქვს InvalidOperationException, თუ
// num.HasValue არის false
        try
        {
            Console.WriteLine("value = {0}", num.Value);
        }
        catch (InvalidOperationException e)
        {
            Console.WriteLine(e.Message);
        }
    }
    static void Main()
    {
        DisplayValue(1);
        DisplayValue(null);
    }
}
```

13. ბიბლიოთეკები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// FunctionClient.cs

// შედგენა /reference:DigitCounter.dll;Factorial.dll-ით

// არგუმენტები: 3 5 10

using System;

// შემდეგი using დირექტივა ხდის ტიპებს განსაზღვრულს ფუნქციებში.

using Functions;

class FunctionClient

{

   public static void Main(string[] args)

   {

      Console.WriteLine("Function Client");

      if ( args.Length == 0 )

      {

         Console.WriteLine("Usage: FunctionTest ... ");

         return;

      }

      for ( int i = 0; i < args.Length; i++ )

      {

         int num = Int32.Parse(args[i]);
```

```
            Console.WriteLine(
              "The Digit Count for String [{0}] is [{1}]",
              args[i],
              // NumberOfDigits static მეთოდის გამოძახება
              // DigitCount class-ში:
              DigitCount.NumberOfDigits(args[i]));
            Console.WriteLine(
              "The Factorial for [{0}] is [{1}]",
               num,
              // იძახებს Calc static მეთოდს Factorial class-იდან:
               Factorial.Calc(num) );
        }
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// Factorial.cs
using System;
// namespaceის გამოცხადება. თქვენ გჭირდებათ შეფუთოთ თქვენი ბიბლიოთეკები
// თავიანთი    namespace-ის შესატყვისად ისე, რომ   .NET runtime-მა შეძლოს
// სწორად  ჩატვირთოს კლასები.
namespace Functions
{
```

```
   public class Factorial
   {
// "Calc" static მეთოდი ითვლის ფაქტორიალის მნიშვნელობას მიწოდებული
// მთელისათვის.
      public static int Calc(int i)
      {
         return((i <= 1) ? 1 : (i * Calc(i-1)));
      }
   }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// DigitCounter.cs
//  /target:library-ით კომპილირება
using System;
namespace Functions
{
   public class DigitCount
   {
      // NumberOfDigits static მეთოდი ითვლის ციფრების რაოდენობას
// გადაცემულ სტრიქონში
      public static int NumberOfDigits(string theString)
      {
```

```
        int count = 0;
        for ( int i = 0; i < theString.Length; i++ )
        {
            if ( Char.IsDigit(theString[i]) )
            {
                count++;
            }
        }
        return count;
    }
  }
}
```

14. ცხადი ინტერფეისი

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// explicit1.cs

interface IDimensions

{

   float Length();

   float Width();

}

class Box : IDimensions

{

   float lengthInches;

   float widthInches;

   public Box(float length, float width)

   {

      lengthInches = length;

      widthInches = width;

   }

      float IDimensions.Length()

   {

      return lengthInches;

   }
```

```
  float IDimensions.Width()
  {
    return widthInches;
  }
  public static void Main()
  {
    // "myBox"ინტერფეისის გამოცხადება:
    Box myBox = new Box(30.0f, 20.0f);
    // "myDimensions"ინტერფეისის გამოცხადება:
    IDimensions myDimensions = (IDimensions) myBox;
    // box-ის განზომილების დაბეჭდვა:
    /* The following commented lines would produce compilation
      errors because they try to access an explicitly implemented
      interface member from a class instance:              */
    System.Console.WriteLine("სიგრძე: {0}", myBox.Length());
    System.Console.WriteLine("სიგანე: {0}", myBox.Width());
    /* Print out the dimensions of the box by calling the methods
      from an instance of the interface:                   */
    System.Console.WriteLine("სიგრძე: {0}", myDimensions.Length());
    System.Console.WriteLine("სიგანე: {0}", myDimensions.Width());
  }
}
```

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// explicit2.cs
// ინგლისური ერთეულების ინტერფეისის გამოცხადება:
interface IEnglishDimensions
{
   float Length();
   float Width();
}
// მეტრიკული ერთეულების ინტერფეისის გამოცხადება:
interface IMetricDimensions
{
   float Length();
   float Width();
}
// კლასის გამოცხადება, რომლებიც ახორციელებენ ორ ინტერფეისს:
 // IEnglishDimensions და IMetricDimensions:
class Box : IEnglishDimensions, IMetricDimensions
{
   float lengthInches;
   float widthInches;
   public Box(float length, float width)
```

```
  {
    lengthInches = length;
    widthInches = width;
  }
// IEnglishDimensions-ის წევრების ცხადად რეალიზაცია:
  float IEnglishDimensions.Length()
  {
    return lengthInches;
  }
  float IEnglishDimensions.Width()
  {
    return widthInches;
  }
// IMetricDimensions-ის წევრების ცხადად რეალიზაცია:
  float IMetricDimensions.Length()
  {
    return lengthInches * 2.54f;
  }
  float IMetricDimensions.Width()
  {
    return widthInches * 2.54f;
  }
```

```
   public static void Main()
   {
      // "myBox"ობიექტის გამოცხადება:
      Box myBox = new Box(30.0f, 20.0f);
// ინგლისური ერთეულების ინტერფეისის გამოცხადება:
      IEnglishDimensions eDimensions = (IEnglishDimensions) myBox;
      // მეტრიკული ერთეულების ინტერფეისის გამოცხადება
      IMetricDimensions mDimensions = (IMetricDimensions) myBox;
      // განზომილებების დაბეჭდვა ინგლისურ ერთეულებში
System.Console.WriteLine("Length(in): {0}", eDimensions.Length());
      System.Console.WriteLine("Width (in): {0}", eDimensions.Width());
      // განზომილებების დაბეჭდვა ინგლისურ ერთეულებში:
      System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());
      System.Console.WriteLine("Width (cm): {0}", mDimensions.Width());
```

15. ხდომილებები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// events1.cs

using System;

namespace MyCollections

{

  using System.Collections;

  // A delegate type for hooking up change notifications.

  public delegate void ChangedEventHandler(object sender, EventArgs e);

  // კლას, რომელიც მუშაობს ArrayList-ის მსგავსად, მაგრამ აგზავნის

// ხდომილების შეტყობინებებს, როცა სია იცვლება

  public class ListWithChangedEvent: ArrayList

  {

    // ხდომილება, რომელიც კლიენტებს შეუძლიათ გამოიყენონ, რომ იყვნენ

//ინფორმირებული,  როცა სიის ელემენტები იცვლება

    public event ChangedEventHandler Changed;

    // შეცვლილი ხდომილების გამოძახება, როცა სია იცვლება

 protected virtual void OnChanged(EventArgs e)

    {

      if (Changed != null)

        Changed(this, e);
```

```
  }
  // ზოგიერთი მეთოდების გადაფარვა, რომელთაც შეუძლიათ შეცვალონ სია
  public override int Add(object value)
  {
    int i = base.Add(value);
    OnChanged(EventArgs.Empty);
    return i;
  }
  public override void Clear()
  {
    base.Clear();
    OnChanged(EventArgs.Empty);
  }
  public override object this[int index]
  {
    set
    {
      base[index] = value;
      OnChanged(EventArgs.Empty);
    }
  }
}
```

```
}
namespace TestEvents
{
  using MyCollections;
  class EventListener
  {
    private ListWithChangedEvent List;
    public EventListener(ListWithChangedEvent list)
    {
      List = list;
      // შეცვლილ ხდომილებას დავუმატოთ "ListChanged" სიაში
      List.Changed += new ChangedEventHandler(ListChanged);
    }
    // ეს შეიძლება იყოს გამოძახებული, როცა სია იცვლება
    private void ListChanged(object sender, EventArgs e)
    {
      Console.WriteLine("ეს გამოიძახება, როცა ხდომილება გაისვრის.");
    }
    public void Detach()
    {
      // ხდომილების მოცილება და სიის წაშლა
      List.Changed -= new ChangedEventHandler(ListChanged);
```

```
        List = null;
      }
    }
    class Test
    {
      // ListWithChangedEvent კლასის ტესტირება.
      public static void Main()
      {
      // ახალი სიის შექმნა.
      ListWithChangedEvent list = new ListWithChangedEvent();
      // კლასის შექმნა, რომელიც უსმენს სიის შეცვლის ხდომილებას.
// სიაზე ელემენტის დამატება და ამოგდება
EventListener listener = new EventListener(list);
      // ერთეულების დამატება და ამოგდება სიიდან.
      list.Add("item 1");
      list.Clear();
      listener.Detach();
      }
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// events2.cs
```

```
using System;

namespace MyCollections

{

  using System.Collections;

  // კლასი, რომელიც მუშაობს  ArrayList-ის მსგავსად, მაგრამ აგზავნის

// ხდომილების შეტყობინებას, როცა სია იცვლება

  public class ListWithChangedEvent: ArrayList

  {

    // ხდომილება, რომელიც კლიენტებმა შეიძლება გამოიყენონ შეტობინების

// მისაღებად, როცა სიის ელემენტები იცვლება.

    public event EventHandler Changed;

    // შეცვლილი ხდომილების გამოძახება, როცა სია იცვლება.

 protected virtual void OnChanged(EventArgs e)

    {

      if (Changed != null)

        Changed(this,e);

    }

    // ზოგიერთი მეთოდის გადაფარვა, რომლებსაც შეუძლიათ სიის შეცვლა

        public override int Add(object value)

    {

      int i = base.Add(value);

      OnChanged(EventArgs.Empty);
```

```
        return i;
      }
      public override void Clear()
      {
        base.Clear();
        OnChanged(EventArgs.Empty);
      }
      public override object this[int index]
      {
        set
        {
          base[index] = value;
          OnChanged(EventArgs.Empty);
        }
      }
    }
}
namespace TestEvents
{
    using MyCollections;
    class EventListener
    {
```

```
  private ListWithChangedEvent List;
  public EventListener(ListWithChangedEvent list)
  {
    List = list;
    // დავუმატოთ  "ListChanged" შეცვლილხდომილებას "List"-ზე:
    List.Changed += new EventHandler(ListChanged);
  }
  // ეს გამოიძახება ყველგან, სადაც სია შეიცვლება:
  private void ListChanged(object sender, EventArgs e)
  {
    Console.WriteLine("ეს გამოიძახება როცა ხდომილება გაისვრის.");
  }
  public void Detach()
  {
    // ხდომილების მოშორება და სიის ამოგდება:
    List.Changed -= new EventHandler(ListChanged);
    List = null;
  }
}
class Test
{
  // ListWithChangedEvent კლასის ტესტირება:
```

```
        public static void Main()
        {
        // ახალი სიის შექმნა:
        ListWithChangedEvent list = new ListWithChangedEvent();
        // კლასის შექმნა, რომელიც მოუსმენს სიის ცვლილების ხდომილებას:
        EventListener listener = new EventListener(list);
        // ერთეულების დამატება და ამოგდება სიიდან:
        list.Add("item 1");
        list.Clear();
        listener.Detach();
        }
    }
}
```

16. ბრძანებათა სტრიქონი

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// cmdline1.cs

// არგუმენტები: A B C

using System;

public class CommandLine

{

  public static void Main(string[] args)

  {

    // მასივის სიგრძის მისაღებად Length თვისების გამოყენება.

    // შევნიშნოთ, რომ Length არის მხოლოდ წაკითხვადი თვისება:

    Console.WriteLine("ბრძანებათა სტრიქონის პარამეტრების რაოდენობა = {0}",

      args.Length);

    for(int i = 0; i < args.Length; i++)

    {

      Console.WriteLine("Arg[{0}] = [{1}]", i, args[i]);

    }

  }

}

//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// cmdline2.cs
```

```
// არგუმენტები: John Paul Mary

using System;

public class CommandLine2

{

  public static void Main(string[] args)

  {

    Console.WriteLine("ბრძანებათა სტრიქონის პარამეტრების რაოდენობა = {0}",

      args.Length);

    foreach(string s in args)

    {

      Console.WriteLine(s);

    }

  }

}
```

17. კოლექციები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// tokens.cs

using System;

// System.Collections namespace გავხადოთ წვდომადი:

using System.Collections;

// Tokens კლასის გამოცხადება:

public class Tokens : IEnumerable

{

  private string[] elements;

  Tokens(string source, char[] delimiters)

  {

    // სტრიქონის გარჩევა token-ებში:

    elements = source.Split(delimiters);

  }

  // IEnumerable ინტერფეისი:

  //GetEnumerator() მეთოდის გამოცხადება მოითხოვება  IEnumerable-ის

// მიერ

  public IEnumerator GetEnumerator()

  {

    return new TokenEnumerator(this);

  }
```

```
// შიდა კლასი ახორციელებს IEnumerator ინტერფეისს:
private class TokenEnumerator : IEnumerator
{
  private int position = -1;
  private Tokens t;
  public TokenEnumerator(Tokens t)
  {
    this.t = t;
  }
  // MoveNext მეთოდის გამოცხადება, საჭიროა IEnumerator-ისათვის.
  public bool MoveNext()
  {
    if (position < t.elements.Length - 1)
    {
      position++;
      return true;
    }
    else
    {
      return false;
    }
  }
```

```
    // Reset მეთოდის გამოცხადება, საჭიროა IEnumerator-ისათვის:
    public void Reset()
    {
      position = -1;
    }
    // მიმდინარე თვისების გამოცხადება IEnumerator-ისათვის:
    public object Current
    {
      get
      {
        return t.elements[position];
      }
    }
  }
  // Token-ების და TokenEnumerator-ის ტესტირება
  static void Main()
  {
    // Token-ების ტესტირება სტრიქონის  token-ებად დანაწილებით
Tokens f = new Tokens("This is a well-done program.",
      new char[] {' ','-'});
    foreach (string item in f)
    {
```

```
        Console.WriteLine(item);
      }
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// tokens2.cs
using System;
using System.Collections;
public class Tokens: IEnumerable
{
  private string[] elements;
  Tokens(string source, char[] delimiters)
  {
    elements = source.Split(delimiters);
  }
  // IEnumerable Interface განხორციელება:
  public TokenEnumerator GetEnumerator()
  {
    return new TokenEnumerator(this);
  }
  IEnumerator IEnumerable.GetEnumerator()
  {
```

```
    return (IEnumerator) new TokenEnumerator(this);
}
// შიგა კლასი ახორციელებს IEnumerator ინტერფეისს:
public class TokenEnumerator: IEnumerator
{
    private int position = -1;
    private Tokens t;
    public TokenEnumerator(Tokens t)
    {
        this.t = t;
    }
    public bool MoveNext()
    {
        if (position < t.elements.Length - 1)
        {
            position++;
            return true;
        }
        else
        {
            return false;
        }
```

```
    }
    public void Reset()
    {
      position = -1;
    }
    public string Current
    {
      get
      {
        return t.elements[position];
      }
    }
    object IEnumerator.Current
    {
      get
      {
        return t.elements[position];
      }
    }
  }
  // Token-ების და  TokenEnumerator-ის ტესტირება
```

```
  static void Main()
  {
    Tokens f = new Tokens("ეს არის კარგად დაწერილი პროგრამა.",
      new char [] {' ','-'});
    foreach (string item in f)
    {
      Console.WriteLine(item);
    }
  }
}
```

18. პირობითი მეთოდები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// tokens.cs

using System;

// System.Collections namespace გავხადოთ წვდომადი:

using System.Collections;

// Tokens კლასის გამოცხადება:

public class Tokens : IEnumerable

{

  private string[] elements;

  Tokens(string source, char[] delimiters)

  {

    // სტრიქონის გარჩევა  token-ებად:

    elements = source.Split(delimiters);

  }

  // IEnumerable Interface განხორციელება:

  //GetEnumerator() მეთოდის  გამოცხადება, მოითხოვება IEnumerable-ის

// მიერ:

  public IEnumerator GetEnumerator()

  {

    return new TokenEnumerator(this);
```

```
}

// შიგა კლასი ახორციელებს IEnumerator ინტერფეისს:

private class TokenEnumerator : IEnumerator

{

  private int position = -1;

  private Tokens t;

  public TokenEnumerator(Tokens t)

  {

    this.t = t;

  }

  // MoveNext მრთოდის გამოცხადება, მოითხოვება IEnumerator-ისთვის:

  public bool MoveNext()

  {

    if (position < t.elements.Length - 1)

    {

      position++;

      return true;

    }

    else

    {

      return false;

    }
```

```
  }
  // Reset მეთოდის გამოცხადება, მოითხოვება IEnumerator-ისთვის:
  public void Reset()
  {
    position = -1;
  }
  // Current თვისების გამოცხადება, მოითხოვება IEnumerator-ისთვის:
  public object Current
  {
    get
    {
      return t.elements[position];
    }
  }
}
//  Token-ებისა და TokenEnumerator-ის ტესტირება
static void Main()
{
  //  Token-ების ტესტირება სტრიქონის token-ებად დანაწილებით:
  Tokens f = new Tokens("ეს არის კარგად დაწერილი პროგრამა.",
    new char[] {' ','-'});
  foreach (string item in f)
```

```
    {
      Console.WriteLine(item);
    }
  }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// tokens2.cs
using System;
using System.Collections;
public class Tokens: IEnumerable
{
  private string[] elements;
  Tokens(string source, char[] delimiters)
  {
    elements = source.Split(delimiters);
  }
  // IEnumerable Interface-ის განხორციელება:
  public TokenEnumerator GetEnumerator()
  {
    return new TokenEnumerator(this);
  }
  IEnumerator IEnumerable.GetEnumerator()
```

```
// IEnumerable ვერსია
  {
    return (IEnumerator) new TokenEnumerator(this);
  }
  // შიგა კლასი ახორციელებს IEnumerator ინტერფეისს:
  public class TokenEnumerator: IEnumerator
  {
    private int position = -1;
    private Tokens t;
    public TokenEnumerator(Tokens t)
    {
      this.t = t;
    }
    public bool MoveNext()
    {
      if (position < t.elements.Length - 1)
      {
        position++;
        return true;
      }
      else
      {
```

```
            return false;
        }
    }
    public void Reset()
    {
        position = -1;
    }
    public string Current
    {
get{
            return t.elements[position];
        }
    }
    object IEnumerator.Current
// IEnumerator version: აბრუნებს ობიექტს
    {
        get
        {
            return t.elements[position];
        }
    }
  }
```

```
  // Token-ებისა და TokenEnumerator-ის ტესტირება
  static void Main()
  {
    Tokens f = new Tokens("This is a well-done program.",
      new char [] {' ','-'});
    foreach (string item in f)
    {
      Console.WriteLine(item);
    }
  }
}
```

xml დოკუმენტი

```
//Copyright (C) Microsoft Corporation.  All rights reserved.
// XMLsample.cs
//   /doc:XMLsample.xml-ით კომპილაცია
using System;
/// <summary>
/// Class level summary documentation goes here.</summary>
/// <remarks>
/// Longer comments can be associated with a type or member
/// through the remarks tag</remarks>
```

```
public class SomeClass
{
  /// <summary>
  /// Store for the name property</summary>
  private string myName = null;
  /// <summary>
  /// The class constructor. </summary>
  public SomeClass()
  {
     // TODO: Add Constructor Logic here
  }
  /// <summary>
  /// Name property </summary>
  /// <value>
  /// A value tag is used to describe the property value</value>
  public string Name
  {
    get
    {
      if ( myName == null )
      {
        throw new Exception("Name is null");
```

```
      }
      return myName;
    }
}
/// <summary>
/// Description for SomeMethod.</summary>
/// <param name="s"> Parameter description for s goes here</param>
/// <seealso cref="String">
/// You can use the cref attribute on any tag to reference a type or member
/// and the compiler will check that the reference exists. </seealso>
public void SomeMethod(string s)
{
}
/// <summary>
/// Some other method. </summary>
/// <returns>
/// Return results are described through the returns tag.</returns>
/// <seealso cref="SomeMethod(string)">
/// Notice the use of the cref attribute to reference a specific method </seealso>
public int SomeOtherMethod()
{
   return 0;
```

```
    }

    /// <summary>

    /// The entry point for the application.

    /// </summary>

    /// <param name="args"> A list of command line arguments</param>

    public static int Main(String[] args)

    {

      // TODO: Add code to start application here

       return 0;

    }

}
```

19. ვერსიები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// versioning.cs

// მოსალოდნელია CS0114.

public class MyBase

{

  public virtual string Meth1()

  {

    return "MyBase-Meth1";

  }

  public virtual string Meth2()

  {

    return "MyBase-Meth2";

  }

  public virtual string Meth3()

  {

    return "MyBase-Meth3";

  }

}

class MyDerived : MyBase

{
```

```
   // virtual method Meth1-ის გადაფარვა override გასაღები სიტყვის
// გამოყენებით:
   public override string Meth1()
   {
      return "MyDerived-Meth1";
   }
   // Meth2 ვირტუალური მეთოდის ცხადად დამალვა  new გასაღები სიტყვის
// გამოყენებით:
      public new string Meth2()
   {
      return "MyDerived-Meth2";
   }
   // რადგანაც არაა გამოყენებული არავითარი გასაღები სიტყვა მომდევნო
//გამოცხადებაში, ამიტომ გამოიტანება გამაფრთხილებელი შეტყობინება
//პროგრამისტისათვის, რომ ეს მეთოდი ფარავს MyBase.Meth3()  მემკვიდრე წევრს:
   public string Meth3()
   {
      return "MyDerived-Meth3";
   }
   public static void Main()
   {
      MyDerived mD = new MyDerived();
```

```
    MyBase mB = (MyBase) mD;

    System.Console.WriteLine(mB.Meth1());

    System.Console.WriteLine(mB.Meth2());

    System.Console.WriteLine(mB.Meth3());

}
```

20. დაუცველი

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// readfile.cs

// კომპილირებულია: /unsafe არგუმენტებით readfile.txt

// გამოიყენეთ ეს პროგრამა, რომ წაიკითხოთ და გამოიტანოთ რაიმე ტექსტური
//ფაილი:

using System;

using System.Runtime.InteropServices;

using System.Text;

class FileReader

{

        const uint GENERIC_READ = 0x80000000;

        const uint OPEN_EXISTING = 3;

        IntPtr handle;

        [DllImport("kernel32", SetLastError=true)]

        static extern unsafe IntPtr CreateFile(

                string FileName,

                uint DesiredAccess,

                uint ShareMode,

                uint SecurityAttributes,

                uint CreationDisposition,

                uint FlagsAndAttributes,
```

```
        int hTemplateFile);
[DllImport("kernel32", SetLastError=true)]
static extern unsafe bool ReadFile(
        IntPtr hFile,
        void* pBuffer,
        int NumberOfBytesToRead,
        int* pNumberOfBytesRead,
        int Overlapped);
[DllImport("kernel32", SetLastError=true)]
static extern unsafe bool CloseHandle(
        IntPtr hObject);
public bool Open(string FileName)
{
        // არსებული ფაილის გახსნა წაკითხვისათვის:
        handle = CreateFile(
                FileName,
                GENERIC_READ,
                0,
                0,
                OPEN_EXISTING,
                0,
                0);
```

```
            if (handle != IntPtr.Zero)
                return true;
            else
                return false;
        }
        public unsafe int Read(byte[] buffer, int index, int count)
        {
            int n = 0;
            fixed (byte* p = buffer)
            {
                if (!ReadFile(handle, p + index, count, &n, 0))
                    return 0;
            }
            return n;
        }
        public bool Close()
        {
            //ფაილის სახელურის დახურვა
            return CloseHandle(handle);
        }
}
```

```
class Test
{
	public static int Main(string[] args)
	{
		if (args.Length != 1)
		{
			Console.WriteLine("Usage : ReadFile <FileName>");
			return 1;
		}

		if (! System.IO.File.Exists(args[0]))
		{
			Console.WriteLine("File " + args[0] + " not found.");
			return 1;
		}

		byte[] buffer = new byte[128];
		FileReader fr = new FileReader();

		if (fr.Open(args[0]))
		{
```

```
                // იგულისხმება, რომ ვკითხულობთ ASCII ფაილს:
                ASCIIEncoding Encoding = new ASCIIEncoding();
                int bytesRead;
                do
                {
                        bytesRead = fr.Read(buffer, 0, buffer.Length);
                        string content = Encoding.GetString(buffer,0,bytesRead);
                        Console.Write("{0}", content);
                }
                while ( bytesRead > 0);

                fr.Close();
                return 0;
            }
            else
            {
                Console.WriteLine("Failed to open requested file");
                return 1;
            }
        }
}
```

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// printversion.cs
// კომპილაცია  /unsafe-ით:
using System;
using System.Reflection;
using System.Runtime.InteropServices;
// ამლევს ამ assembly-ის ვერსიის ნომერს: [assembly:AssemblyVersion("4.3.2.1")]

public class Win32Imports
{
        [DllImport("version.dll")]
        public static extern bool GetFileVersionInfo (string sFileName,
                int handle, int size, byte[] infoBuffer);
        [DllImport("version.dll")]
        public static extern int GetFileVersionInfoSize (string sFileName,
                out int handle);
        // მე-3 პარამეტრი - "out string pValue" - არის ავტომატურად //გადაყვანილი Ansi-
დან Unicode-ში:
        [DllImport("version.dll")]
        unsafe public static extern bool VerQueryValue (byte[] pBlock,
                string pSubBlock, out string pValue, out uint len);
```

```
        // ეს VerQueryValue overload არის მონიშნული, როგორც 'unsafe', //რადგანაც იგი
იყენებს short*-ს:
        [DllImport("version.dll")]
        unsafe public static extern bool VerQueryValue (byte[] pBlock,
                string pSubBlock, out short *pValue, out uint len);
}
public class C
{
        // Main მონიშნულია, როგორც 'unsafe', რადგანაც იგი იყენებს //მიმთითებლებს:
        unsafe public static int Main ()
        {
                try
                {
                        int handle = 0;
                                                int size =
                                Win32Imports.GetFileVersionInfoSize("printversion.exe",
                                out handle);
                        if (size == 0) return -1;
                        byte[] buffer = new byte[size];
                        if (!Win32Imports.GetFileVersionInfo("printversion.exe", handle, size,
buffer))
                        {
```

```
                Console.WriteLine("Failed to query file version information.");
                return 1;
            }
            short *subBlock = null;
            uint len = 0;
            if (!Win32Imports.VerQueryValue (buffer, @"\VarFileInfo\Translation",
out subBlock, out len))
            {
                Console.WriteLine("Failed to query version information.");
                return 1;
            }
            string spv = @"\StringFileInfo\" + subBlock[0].ToString("X4") +
subBlock[1].ToString("X4") + @"\ProductVersion";
            byte *pVersion = null;
            // Get the ProductVersion value for this program:
            string versionInfo;
            if (!Win32Imports.VerQueryValue (buffer, spv, out versionInfo, out
len))
            {
                Console.WriteLine("Failed to query version information.");
                return 1;
            }
```

```
                Console.WriteLine ("ProductVersion == {0}", versionInfo);
            }
            catch (Exception e)
            {
                Console.WriteLine ("Caught unexpected exception " + e.Message);
            }
            return 0;
        }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// fastcopy.cs
// კომპილაცია /unsafe-ით:
using System;
class Test
{
   // unsafe გასაღები სიტყვა საშუალებას იძლევა მიმთითებლები იყვნენ
//გამოყენებულნი შემდეგი მეთოდების შიგნით:
   static unsafe void Copy(byte[] src, int srcIndex,
      byte[] dst, int dstIndex, int count)
   {
      if (src == null || srcIndex < 0 ||
         dst == null || dstIndex < 0 || count < 0)
```

```
        {
            throw new ArgumentException();
        }
        int srcLen = src.Length;
        int dstLen = dst.Length;
        if (srcLen - srcIndex < count ||
            dstLen - dstIndex < count)
        {
            throw new ArgumentException();
        }
        // შემდეგი ინსტრუქციები აფიქსირებენ src და dst ობიექტების
// ადგილს, რომ არ იყვნენ ისინი ამოგდებული ნაგავის შეგროვების დროს:
        fixed (byte* pSrc = src, pDst = dst)
        {
            byte* ps = pSrc;
            byte* pd;
            for (int n = 0; n < count / 4; n++)
            {
                *((int*)pd) = *((int*)ps);
                pd += 4;
                ps += 4;
            }
```

```
            for (int n = 0; n < count % 4; n++)
            {
                *pd = *ps;
                pd++;
                ps++;
            }
        }
    }
    static void Main(string[] args)
    {
        byte[] a = new byte[100];
        byte[] b = new byte[100];
        for (int i = 0; i < 100; ++i)
            a[i] = (byte)i;
        Copy(a, 0, b, 0, 100);
        Console.WriteLine("The first 10 elements are:");
        for (int i = 0; i < 10; ++i)
            Console.Write(b[i] + " ");
        Console.WriteLine("\n");
    }
}
```

21. ძაფები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

using System;

using System.Threading;

using System.Collections;

using System.Collections.Generic;

// The thread synchronization events are encapsulated in this

// class to allow them to easily be passed to the Consumer and

// Producer classes.

public class SyncEvents

{

    public SyncEvents()

    {

        // AutoResetEvent is used for the "new item" event because

        // we want this event to reset automatically each time the

        // consumer thread responds to this event.

        _newItemEvent = new AutoResetEvent(false);

        // ManualResetEvent is used for the "exit" event because

        // we want multiple threads to respond when this event is

        // signaled. If we used AutoResetEvent instead, the event

        // object would revert to a non-signaled state with after
```

```
    // a single thread responded, and the other thread would
    // fail to terminate.
    _exitThreadEvent = new ManualResetEvent(false);
    // The two events are placed in a WaitHandle array as well so
    // that the consumer thread can block on both events using
    // the WaitAny method.
    _eventArray = new WaitHandle[2];
    _eventArray[0] = _newItemEvent;
    _eventArray[1] = _exitThreadEvent;
}
// Public properties allow safe access to the events.
public EventWaitHandle ExitThreadEvent
{
    get { return _exitThreadEvent; }
}
public EventWaitHandle NewItemEvent
{
    get { return _newItemEvent; }
}
public WaitHandle[] EventArray
{
    get { return _eventArray; }
```

```
    }

    private EventWaitHandle _newItemEvent;
    private EventWaitHandle _exitThreadEvent;
    private WaitHandle[] _eventArray;
}

// The Producer class asynchronously (using a worker thread)
// adds items to the queue until there are 20 items.
public class Producer
{
    public Producer(Queue<int> q, SyncEvents e)
    {
        _queue = q;
        _syncEvents = e;
    }
    public void ThreadRun()
    {
        int count = 0;
        Random r = new Random();
        while (!_syncEvents.ExitThreadEvent.WaitOne(0, false))
        {
```

```
            lock (((ICollection)_queue).SyncRoot)
            {
                while (_queue.Count < 20)
                {
                    _queue.Enqueue(r.Next(0, 100));
                    _syncEvents.NewItemEvent.Set();
                    count++;
                }
            }
        }
        Console.WriteLine("Producer thread: produced {0} items", count);
    }
    private Queue<int> _queue;
    private SyncEvents _syncEvents;
}
// The Consumer class uses its own worker thread to consume items
// in the queue. The Producer class notifies the Consumer class
// of new items with the NewItemEvent.
public class Consumer
{
    public Consumer(Queue<int> q, SyncEvents e)
    {
```

```
        _queue = q;
        _syncEvents = e;
    }
    public void ThreadRun()
    {
        int count = 0;
        while (WaitHandle.WaitAny(_syncEvents.EventArray) != 1)
        {
            lock (((ICollection)_queue).SyncRoot)
            {
                int item = _queue.Dequeue();
            }
            count++;
        }
        Console.WriteLine("Consumer Thread: consumed {0} items", count);
    }
    private Queue<int> _queue;
    private SyncEvents _syncEvents;
}
public class ThreadSyncSample
{
    private static void ShowQueueContents(Queue<int> q)
```

```
{
    // Enumerating a collection is inherently not thread-safe,
    // so it is imperative that the collection be locked throughout
    // the enumeration to prevent the consumer and producer threads
    // from modifying the contents. (This method is called by the
    // primary thread only.)
    lock (((ICollection)q).SyncRoot)
    {
        foreach (int i in q)
        {
            Console.Write("{0} ", i);
        }
    }
    Console.WriteLine();
}
static void Main()
{
    // Configure struct containing event information required
    // for thread synchronization.
    SyncEvents syncEvents = new SyncEvents();
    // Generic Queue collection is used to store items to be
    // produced and consumed. In this case 'int' is used.
```

```
Queue<int> queue = new Queue<int>();

// Create objects, one to produce items, and one to

// consume. The queue and the thread synchronization

// events are passed to both objects.

Console.WriteLine("Configuring worker threads...");

Producer producer = new Producer(queue, syncEvents);

Consumer consumer = new Consumer(queue, syncEvents);

// Create the thread objects for producer and consumer

// objects. This step does not create or launch the

// actual threads.

Thread producerThread = new Thread(producer.ThreadRun);

Thread consumerThread = new Thread(consumer.ThreadRun)

// Create and launch both threads.

Console.WriteLine("Launching producer and consumer threads...");

producerThread.Start();

consumerThread.Start();

// Let producer and consumer threads run for 10 seconds.

// Use the primary thread (the thread executing this method)

// to display the queue contents every 2.5 seconds.

for (int i = 0; i < 4; i++)

{

    Thread.Sleep(2500);
```

```
            ShowQueueContents(queue);
        }
        // Signal both consumer and producer thread to terminate.
        // Both threads will respond because ExitThreadEvent is a
        // manual-reset event--so it stays 'set' unless explicitly reset.
        Console.WriteLine("Signaling threads to terminate...");
        syncEvents.ExitThreadEvent.Set();
        // Use Join to block primary thread, first until the producer thread
        // terminates, then until the consumer thread terminates.
        Console.WriteLine("main thread waiting for threads to finish...");
        producerThread.Join();
        consumerThread.Join();
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
using System;
using System.Threading;
public class Worker
{
    // This method will be called when the thread is started.
    public void DoWork()
    {
```

```
        while (!_shouldStop)
        {
            Console.WriteLine("worker thread: working...");
        }
        Console.WriteLine("worker thread: terminating gracefully.");
    }
    public void RequestStop()
    {
        _shouldStop = true;
    }
    // Volatile is used as hint to the compiler that this data
    // member will be accessed by multiple threads.
    private volatile bool _shouldStop;
}
public class WorkerThreadExample
{
    static void Main()
    {
        // Create the thread object. This does not start the thread.
        Worker workerObject = new Worker();
        Thread workerThread = new Thread(workerObject.DoWork);
```

```
        // Start the worker thread.
        workerThread.Start();
        Console.WriteLine("main thread: Starting worker thread...");
        // Loop until worker thread activates.
        while (!workerThread.IsAlive);
        // Put the main thread to sleep for 1 millisecond to
        // allow the worker thread to do some work:
        Thread.Sleep(1);
        // Request that the worker thread stop itself:
        workerObject.RequestStop();
        // Use the Join method to block the current thread
        // until the object's thread terminates.
        workerThread.Join();
        Console.WriteLine("main thread: Worker thread has terminated.");
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
using System;
using System.Threading;
// The Fibonacci class provides an interface for using an auxiliary
// thread to perform the lengthy Fibonacci(N) calculation.
// N is provided to the Fibonacci constructor, along with an
```

```
// event that the object signals when the operation is complete.
// The result can then be retrieved with the FibOfN property.
public class Fibonacci
{
    public Fibonacci(int n, ManualResetEvent doneEvent)
    {
        _n = n;
        _doneEvent = doneEvent;
    }
    // Wrapper მეთოდი  ძაფების ერთობლიობის გამოყენებისათვის.
    public void ThreadPoolCallback(Object threadContext)
    {
        int threadIndex = (int)threadContext;
        Console.WriteLine("thread {0} started...", threadIndex);
        _fibOfN = Calculate(_n);
        Console.WriteLine("thread {0} result calculated...", threadIndex);
        _doneEvent.Set();
    }
    // რეკურსიული მეთოდი, რომელიც ითვლის ფიბონაჩის მე-n-ე რიცხვს.
    public int Calculate(int n)
    {
        if (n <= 1)
```

```
        {
            return n;
        }
        else
        {
            return Calculate(n - 1) + Calculate(n - 2);
        }
    }
    public int N { get { return _n; } }
    private int _n;
    public int FibOfN { get { return _fibOfN; } }
    private int _fibOfN;
    ManualResetEvent _doneEvent;
}
public class ThreadPoolExample
{
    static void Main()
    {
        const int FibonacciCalculations = 10;
        // ყოველი Fibonacci ობიექტისათვის გამოიყენება თითო ხდომილება:
        ManualResetEvent[] doneEvents = new ManualResetEvent[FibonacciCalculations];
        Fibonacci[] fibArray = new Fibonacci[FibonacciCalculations];
```

```
        Random r = new Random();
        // ძაფების შექმნა და გაშვება ThreadPool-ის გამოყენებით:
        Console.WriteLine("launching {0} tasks...", FibonacciCalculations);
        for (int i = 0; i < FibonacciCalculations; i++)
        {
            doneEvents[i] = new ManualResetEvent(false);
            Fibonacci f = new Fibonacci(r.Next(20,40), doneEvents[i]);
            fibArray[i] = f;
            ThreadPool.QueueUserWorkItem(f.ThreadPoolCallback, i);
        }
        // ელოდება ყველა ძაფს pool-ში გამოსათვლელად...
        WaitHandle.WaitAll(doneEvents);
        Console.WriteLine("Calculations complete.");
        // შედეგების გამოტანა...
        for (int i= 0; i<FibonacciCalculations; i++)
        {
            Fibonacci f = fibArray[i];
            Console.WriteLine("Fibonacci({0}) = {1}", f.N, f.FibOfN);
        }
    }
}
```

22. უსაფრთხოება

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// ImperativeSecurity.cs

using System;

using System.Security;

using System.Security.Permissions;

using System.Runtime.InteropServices;

class NativeMethods

{

   // ეს არის უმართავი კოდი. ამ მეთოდის შესრულება მოითხოვს    UnmanagedCode
//უსაფრთხოების ნებართვას. ამ ნებართვის გარეშე ამ მეთოდის გამოძახების ცდა
//გამოიწვევს Security განსაკუთრებულ შემთხვევას.

   [DllImport("msvcrt.dll")]

   public static extern int puts(string str);

   [DllImport("msvcrt.dll")]

   internal static extern int _flushall();

}

class MainClass

{

   private static void CallUnmanagedCodeWithoutPermission()

   {

      // შექმენით უსაფრთხო წვდომის ობიექტი,რომ აღწეროთ UnmanagedCode
```

```
// ნებართვა:
  SecurityPermission perm =
        new SecurityPermission(SecurityPermissionFlag.UnmanagedCode);
      // აკრძალეთ UnmanagedCode.
      // ნებისმიერი მეთოდს, რომელიც გამოიძახება ამ ძაფში აეკრძალება წვდომა
 // უმართავ კოდზე.
      // Even though the CallUnmanagedCodeWithPermission method
      // is called from a stack frame that already
      // calls Assert for unmanaged code, you still cannot call native
      // code. Because you use Deny here, the permission gets
      // overwritten.
      perm.Deny();
      try
      {
         Console.WriteLine("უმართავი კოდის გამოძახების ცდა ნებართვის გარეშე.");
         NativeMethods.puts("Hello World!");
         NativeMethods._flushall();
         Console.WriteLine("გამოძახებულ იქნა უმართავი კოდი ნებართვის გარეშე.");
      }
      catch (SecurityException)
      {
```

```
            Console.WriteLine("დაჭერილ იქნა Security Exception უმართავი კოდის
გამოძახების ცდის დროს.");
        }
    }
    private static void CallUnmanagedCodeWithPermission()
    {
        // შექმენით უსაფრთხოების ნებართვის ობიექტი, რომ აღწეროთ
//UnmanagedCode ნებართვა:
        SecurityPermission perm =
          new SecurityPermission(SecurityPermissionFlag.UnmanagedCode);
        // შეამოწმეთ გაქვთ თუ არა უმართავ კოდზე წვდომის ნებართვა.
        // თუ თქვენ არა გაქვთ უმართავ კოდზე წვდომის ნებართვა, მაშინ ეს გამოძახება
// გამოიწვევს SecurityExcept
        perm.Assert();
        try
        {
            Console.WriteLine("უმართავი კოდის გამოძახების ცდა ნებართვით.");
            NativeMethods.puts("Hello World!");
            NativeMethods._flushall();
            Console.WriteLine("გამოძახებულ იქნა უმართავი კოდი ნებართვით.");
        }
        catch (SecurityException)
```

```
        {
            Console.WriteLine("Security განსაკუთრებული შემთხვევის დაჭერა უმართავი
კოდის გამოძახების ცდის დროს.");
        }
    }
    public static void Main()
    {
        // ეს მეთოდი თვითონ გამოიძახებს უსაფრთხოების ნებართვის აკრძალვას
// უმართავი კოდისათვის, რომელიც გადაფარავს Assert ნებართვასამ სტეკის ფრეიმში.
        SecurityPermission perm = new
            SecurityPermission(SecurityPermissionFlag.UnmanagedCode);
        perm.Assert();
        CallUnmanagedCodeWithoutPermission();

        // ეს მეთოდი თვითონ გამოიძახებს უსაფრთხოების ნებართვას უმართავი //
კოდისათვის, რომელიც გადაფარავს აკრძალვას ამ სტეკის ფრეიმში.
        perm.Deny();
        CallUnmanagedCodeWithPermission();
    }
}
```

```
//Copyright (C) Microsoft Corporation. ყველა უფლება დაცულია.

// DeclarativeSecurity.cs

using System;

using System.Security;

using System.Security.Permissions;

using System.Runtime.InteropServices;

class NativeMethods

{

    // ეს არის უმართავი კოდის გამოძახება. ამ მეთოდის შესრულება მოითხოვს

// UnmanagedCode უსაფრთხოების ნებართვას. ამ მეთოდის გამოძახების ცდა ნებართვის
// გარეშე გამოიწვევს security permission-ს განსაკუთრებულ შემთხვევას.

    [DllImport("msvcrt.dll")]

    public static extern int puts(string str);

    [DllImport("msvcrt.dll")]

    internal static extern int _flushall();

}

class MainClass

{

    // ამ მეთოდზე მიბმული უსაფრთხოების ნებართვა გამოიწვევს  UnmanagedCode

// ნებართვის აკრძალვას ამ მეთოდის დამთავრებამდე: ნებართვა იქნება გადაფარული

    [SecurityPermission(SecurityAction.Deny, Flags =

      SecurityPermissionFlag.UnmanagedCode)]
```

```
    private static void CallUnmanagedCodeWithoutPermission()
    {
        try
        {
            Console.WriteLine("უმართავი კოდის გამოძახების ცდა ნებართვის გარეშე.");
            NativeMethods.puts("Hello World!");
            NativeMethods._flushall();
            Console.WriteLine("გამოძახებულ იქნა უმართავი კოდი ნებართვის გარეშე.");
        }
        catch (SecurityException)
        {
            Console.WriteLine("დაჭერილ იქნა უსაფრთხოების დარღვევის განსაკუთრებული
შემთხვევა უმართავი კოდის გამოძახების გამო.");
        }
    }
    // ამ მეთოდზე მიბმული უსაფრთხოების ნებართვა აიძულებს შესრულებას
// შეამოწმოს  უმართავი კოდის ნებართვა, როცა ეს მეთოდი გამოიძახება. თუ
// გამომძახებელს არა აქვს უმართავი კოდის ნებართვა, მაშინ გამოძახება გამოიმუშავებს
// განსაკუთრებულ შემთხვევას.
    [SecurityPermission(SecurityAction.Assert, Flags =
      SecurityPermissionFlag.UnmanagedCode)]
    private static void CallUnmanagedCodeWithPermission()
```

```
    {
        try
        {
            Console.WriteLine("უმართავი კოდის ნებართვით გამოძახების ცდა.");
            NativeMethods.puts("Hello World!");
            NativeMethods._flushall();
            Console.WriteLine("უმართავი კოდი გამოძახებულ იქნა ნებართვით.");
        }
        catch (SecurityException)
        {
            Console.WriteLine("დაიჭირა განსაკუთრებული შემთხვევა, რომელიც ცდილობდა
უმართავი კოდის გამოძახებას. ");
        }
    }
    public static void Main()
    {
        SecurityPermission perm = new
            SecurityPermission(SecurityPermissionFlag.UnmanagedCode);
        perm.Assert();
        CallUnmanagedCodeWithoutPermission();
        perm.Deny();
        CallUnmanagedCodeWithPermission();
```

```
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// SuppressSecurity.cs
using System;
using System.Security;
using System.Security.Permissions;
using System.Runtime.InteropServices;
class NativeMethods
{
    // ეს არის უმართავი კოდის გამოძახება. ამ მეთოდის შესრულება მოითხოვს
// უმართავი კოდის შესრულების ნებართვას. ამ ნებართვის გარეშე გამოძახების ცდა
// გამოიწვევს განსაკუთრებულ შემთხვევას.
    [SuppressUnmanagedCodeSecurityAttribute()]
    [DllImport("msvcrt.dll")]
    internal static extern int puts(string str);
    [SuppressUnmanagedCodeSecurityAttribute()]
    [DllImport("msvcrt.dll")]
    internal static extern int _flushall();
}
class MainClass
{
```

```
[SecurityPermission(SecurityAction.Deny, Flags =
  SecurityPermissionFlag.UnmanagedCode)]
private static void CallUnmanagedCodeWithoutPermission()
{
   try
   {
      UIPermission uiPermission =
        new UIPermission(PermissionState.Unrestricted);
      uiPermission.Demand();
      Console.WriteLine("უმართავი კოდის გამოძახების ცდა ნებართვის გარეშე.");
      NativeMethods.puts("Hello World!");
      NativeMethods._flushall();
      Console.WriteLine("გამოძახებულ იქნა უმართავი კოდი ნებართვის გარეშე.");
   }
   catch (SecurityException)
   {
      Console.WriteLine("დაჭერილ იქნა უსაფრთხოების განსაკუთრებული შემთხვევა,
რომელიც ცდილობდა უმართავი კოდის გამოძახებას. ");
   }
}
[SecurityPermission(SecurityAction.Assert, Flags =
  SecurityPermissionFlag.UnmanagedCode)]
```

```
    private static void CallUnmanagedCodeWithPermission()
    {
        try
        {
            Console.WriteLine("უმართავი კოდის გამოძახების ცდა ნებართვით.");
            NativeMethods.puts("Hello World!");
            NativeMethods._flushall();
            Console.WriteLine("გამოძახებულ იქნა უმართავი კოდი ნებართვით.");
        }
        catch (SecurityException)
        {
            Console.WriteLine("დაჭერილ იქნა უსაფრთხოების განსაკუთრებული შემთხვევა,
რომელიც ცდილობდა უმართავი კოდის გამოძახებას.");
        }
    }
    public static void Main()
    {
        SecurityPermission perm = new
            SecurityPermission(SecurityPermissionFlag.UnmanagedCode);
        perm.Assert();
        CallUnmanagedCodeWithoutPermission();
        perm.Deny();
```

```
        CallUnmanagedCodeWithPermission();

    }

}
```

23. pinvoke

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// Marshal.cs
using System;
using System.Runtime.InteropServices;
class PlatformInvokeTest
{
   [DllImport("msvcrt.dll")]
   public static extern int puts(
      [MarshalAs(UnmanagedType.LPStr)]
      string m);
   [DllImport("msvcrt.dll")]
   internal static extern int _flushall();
   public static void Main()
   {
      puts("Hello World!");
      _flushall();
   }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// pinvoke.cs
```

```
// კომპილაცია  /addmodule:logfont.netmodule-ით

// csc pinvoke.cs /addmodule:logfont.netmodule

using System;

using System.Runtime.InteropServices;

class PlatformInvokeTest

{

        [DllImport("gdi32.dll", CharSet=CharSet.Auto)]

        public static extern IntPtr CreateFontIndirect(

                [In, MarshalAs(UnmanagedType.LPStruct)]

                LOGFONT lplf

                );

        [DllImport("gdi32.dll")]

        public static extern bool DeleteObject(

                IntPtr handle

                );

        public static void Main()

        {

                LOGFONT lf = new LOGFONT();

                lf.lfHeight = 9;

                lf.lfFaceName = "Arial";

                IntPtr handle = CreateFontIndirect(lf);

                if (IntPtr.Zero == handle)
```

```
                {
                        Console.WriteLine("არ შეიძლება ლოგიკური ფონტის შექმნა.");
                }
                else
                {

                        if (IntPtr.Size == 4)
                                Console.WriteLine("{0:X}", handle.ToInt32());
                        else
                                Console.WriteLine("{0:X}", handle.ToInt64());

                        // ლოგიკური ფონტის ამოგდების შექმნა.
                        if (!DeleteObject(handle))
                                Console.WriteLine("ლოგიკური ფონტის ამოგდება არ
შეიძლება");

                }

        }
}
```

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.


// logfont.cs
// კომპილაცია /target:module-ით
using System;
using System.Runtime.InteropServices;


[StructLayout(LayoutKind.Sequential)]
public class LOGFONT
{
    public const int LF_FACESIZE = 32;
    public int lfHeight;
    public int lfWidth;
    public int lfEscapement;
    public int lfOrientation;
    public int lfWeight;
    public byte lfItalic;
    public byte lfUnderline;
    public byte lfStrikeOut;
    public byte lfCharSet;
    public byte lfOutPrecision;
    public byte lfClipPrecision;
```

```
    public byte lfQuality;
    public byte lfPitchAndFamily;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst=LF_FACESIZE)]
    public string lfFaceName;
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// PInvokeTest.cs
using System;
using System.Runtime.InteropServices;
class PlatformInvokeTest
{
    [DllImport("msvcrt.dll")]
    public static extern int puts(string c);
    [DllImport("msvcrt.dll")]
    internal static extern int _flushall();
    public static void Main()
    {
        puts("Test");
        _flushall();
    }
}
```

24. კერძო ტიპები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

using System;

using System.Collections.Generic;

using System.Text;

namespace PartialClassesExample

{

   class PartialClassesMain

   {

      static void Main(string[] args)

      {

         if (args.Length != 1)

         {

            Console.WriteLine("მოითხოვება ერთი არგუმენტი.");

            return;

         }

         // CharValues არის კერძო კლასი -- ორი მისი მეთოდი განსაზღვრულია

         // CharTypesPublic.cs-ში და სხვა ორი კი CharTypesPrivate.cs-ში.

         int aCount = CharValues.CountAlphabeticChars(args[0]);

         int nCount = CharValues.CountNumericChars(args[0]);

         Console.Write("შესასვლელი არგუმენტი შეიცავს {0} ალფაბეტურ და {1} რიცხვით სიმბოლოებს", aCount, nCount);
```

```
        }
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
using System;
using System.Collections.Generic;
using System.Text;
namespace PartialClassesExample
{
    // კერძო გასაღები სიტყვა,  დამატებით, ამ კლასის მეთოდებს, ველებს და თვისებებს
// ნებას რთავს  იყვნენ განსაზღვრულნი სხვაა   .cs ფაილებში.
    // ეს ფაილი შეიცავს public მეთოდებს განსაზღვრულს CharValue-ებით.
    partial class CharValues
    {
        public static int CountAlphabeticChars(string str)
        {
            int count = 0;
            foreach (char ch in str)
            {
                // IsAlphabetic განსაზღვრულია CharTypesPrivate.cs-ში
                if (IsAlphabetic(ch))
                    count++;
```

```
            }
            return count;
        }
        public static int CountNumericChars(string str)
        {
            int count = 0;
            foreach (char ch in str)
            {
                // IsNumeric განსაზღვრულია CharTypesPrivate.cs-ში
                if (IsNumeric(ch))
                    count++;
            }
            return count;
        }
    }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
using System;
using System.Collections.Generic;
using System.Text;
namespace PartialClassesExample
{
```

```
    // ეს partial გასაღები სიტყვა ნებას რთავს ამ კლასის დამატებით მეთოდებს, ველებს და თვისებებს იყვნენ განსაზღვრულნი სხვა  .cs ფაილებში.

    // ეს ფაილი შეიცავს  private მეთოდებს განსაზღვრულს CharValue-ებით.

    partial class CharValues

    {

        private static bool IsAlphabetic(char ch)

        {

            if (ch >= 'a' && ch <= 'z')

                return true;

            if (ch >= 'A' && ch <= 'Z')

                return true;

            return false;

        }

        private static bool IsNumeric(char ch)

        {

            if (ch >= '0' && ch <= '9')

                return true;

            return false;

        }

    }

}
```

25. oledb

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// OleDbSample.cs

// რომ შევქნათ ეს მაგალითი command line-ით, გამოიყენეთ ბრძანება:

// csc oledbsample.cs

using System;

using System.Data;

using System.Data.OleDb;

using System.Xml.Serialization;

public class MainClass

{

        public static void Main ()

        {

                // დასვით Access connection და აირჩიეთ სტრიქონები.

                // გზა BugTypes.MDB-ისკენ უნდა შეცვალოთ თუთქვენ აგებთ მაგალითს

                // command line-ში:

#if USINGPROJECTSYSTEM

                string strAccessConn = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=..\\..\\BugTypes.MDB";

#else

                string strAccessConn = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=BugTypes.MDB";
```

```
#endif
                string strAccessSelect = "SELECT * FROM Categories";
                // შექმენით dataset და დაუმატეთ მას Categories ცხრილი:
                DataSet myDataSet = new DataSet();
                OleDbConnection myAccessConn = null;
                try
                {
                        myAccessConn = new OleDbConnection(strAccessConn);
                }
                catch(Exception ex)
                {
                        Console.WriteLine("შეცდომა: ვერ შეიქმნა database დაკავშირება.
\n{0}", ex.Message);
                        return;
                }
                try
                {
                        OleDbCommand myAccessCommand = new
OleDbCommand(strAccessSelect,myAccessConn);
                        OleDbDataAdapter myDataAdapter = new
OleDbDataAdapter(myAccessCommand);
                        myAccessConn.Open();
                        myDataAdapter.Fill(myDataSet,"Categories");
```

```
                }
                catch (Exception ex)
                {
                        Console.WriteLine("შეცდომა: წარუმატებლად დამთავრდა საჭიროო
მონაცემების პოვნა მონაცემთა ბაზაში.\n{0}", ex.Message);
                        return;
                }
                finally
                {
                        myAccessConn.Close();
                }
                // dataset შეიძლება შეიცავდეს ჯერად ცხრილებს, შემოვიტანოთ ყველა
// მასივში:
                DataTableCollection dta = myDataSet.Tables;
                foreach (DataTable dt in dta)
                {
                        Console.WriteLine("ნაპოვნია მონაცემთა ცხრილი
 {0}", dt.TableName);
                }
                // შემდეგი ორი სტრიქონი გვიჩვენებს ცხრილების დათვლის ორ
// განსხვავებულ ხერხს dataset-ში:
```

```
            Console.WriteLine("{0} ცხრილები მონაცემთა სიმრავლეში",
myDataSet.Tables.Count);

            Console.WriteLine("{0} ცხრილები მონაცემთა სიმრავლეში ", dta.Count);

            // შემდეგი რამოდენიმე სტრიქონი გვიჩვენებს, როგორ მივიღოთ
// ინფორმაცია კონკრეტული ცხრილის შესახებ dataset-იდან:

            Console.WriteLine("{0} სტრიქონები კატეგორიის ცხრილში",
myDataSet.Tables["Categories"].Rows.Count);

            Console.WriteLine("{0} სვეტები კატეგორიის ცხრილში ",
myDataSet.Tables["Categories"].Columns.Count);

            DataColumnCollection drc = myDataSet.Tables["Categories"].Columns;

            int i = 0;

            foreach (DataColumn dc in drc)

            {

                  Console.WriteLine("სვეტის სახელი[{0}] არის {1}, {2} ტიპის",i++ ,
dc.ColumnName, dc.DataType);

            }

            DataRowCollection dra = myDataSet.Tables["კატეგორიები"].Rows;

            foreach (DataRow dr in dra)

            {

                  Console.WriteLine("CategoryName[{0}] არის {1}", dr[0], dr[1]);

            }


      } }
```

26. ინდექსირებული თვისებები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// indexedproperty.cs

using System;

public class Document

{

   // ტიპი, რომელიც ნებას რთავს დოკუმენტს იყოს მიმოხილული სიტყვების მასივის

// მსგავსად:

   public class WordCollection

   {

      readonly Document document;

      internal WordCollection(Document d)

      {

        document = d;

      }

      // Helper function -- search character array "text", starting at

      // character "begin", for word number "wordCount." Returns false

      // if there are less than wordCount words. Sets "start" and

      // length" to the position and length of the word within text:

      private bool GetWord(char[] text, int begin, int wordCount,

                           out int start, out int length)
```

```
{
    int end = text.Length;
    int count = 0;
    int inWord = -1;
    start = length = 0;
    for (int i = begin; i <= end; ++i)
    {
        bool isLetter = i < end && Char.IsLetterOrDigit(text[i]);
        if (inWord >= 0)
        {
            if (!isLetter)
            {
                if (count++ == wordCount)
                {
                    start = inWord;
                    length = i - inWord;
                    return true;
                }
                inWord = -1;
            }
        }
        else
```

```
        {
            if (isLetter)
                inWord = i;
        }
    }
    return false;
}
        public string this[int index]
{
    get
    {
        int start, length;
        if (GetWord(document.TextArray, 0, index, out start,
                                        out length))
            return new string(document.TextArray, start, length);
        else
            throw new IndexOutOfRangeException();
    }
    set
    {
        int start, length;
        if (GetWord(document.TextArray, 0, index, out start,
```

```
                                out length))
{
   if (length == value.Length)
   {
      Array.Copy(value.ToCharArray(), 0,
            document.TextArray, start, length);
   }
   else
   {
      char[] newText =
         new char[document.TextArray.Length +
                  value.Length - length];
      Array.Copy(document.TextArray, 0, newText,
                              0, start);
      Array.Copy(value.ToCharArray(), 0, newText,
                     start, value.Length);
      Array.Copy(document.TextArray, start + length,
            newText, start + value.Length,
            document.TextArray.Length - start
                              - length);
      document.TextArray = newText;
   }
```

```
            }
            else
                throw new IndexOutOfRangeException();
        }
    }

    // დოკუმენტის სიტყვების რაოდენობის მიღება:
    public int Count
    {
        get
        {
            int count = 0, start = 0, length = 0;
            while (GetWord(document.TextArray, start + length, 0,
                                out start, out length))
                ++count;
            return count;
        }
    }
}

// ტიპი, რომელიც საშუალებას აძლევს დოკუმენტს განხილული იყოს, როგორც
// სიმბოლოების მასივი
```

```
public class CharacterCollection
{
   readonly Document document;
   internal CharacterCollection(Document d)
   {
     document = d;
   }
   // მაინდექსირებელი დოკუმენტში სიმბოლოების მისაღებად:
   public char this[int index]
   {
      get
      {
         return document.TextArray[index];
      }
      set
      {
         document.TextArray[index] = value;
      }
   }
   // სიმბოლოების რაოდენობის მიღება დოკუმენტში:
   public int Count
   {
```

```
            get
            {
                return document.TextArray.Length;
            }
        }
    }
    // რადგანაც ველების თვისებებს აქვთ მაინდექსირებლები, ეს ველები გვრვლინება,
// როგორც "ინდექსირებულითვისებები":
    public WordCollection Words;
    public CharacterCollection Characters;
    private char[] TextArray;
    public Document(string initialText)
    {
        TextArray = initialText.ToCharArray();
        Words = new WordCollection(this);
        Characters = new CharacterCollection(this);
    }
    public string Text
    {
        get
        {
            return new string(TextArray);
```

```
        }
    }
}
class Test
{
    static void Main()
    {
        Document d = new Document(
          "peter piper picked a peck of pickled peppers. How many pickled peppers did peter piper
pick?"
        );
        // შეცვალეთ სიტყვა "peter" "penelope"-ით:
        for (int i = 0; i < d.Words.Count; ++i)
        {
            if (d.Words[i] == "peter")
                d.Words[i] = "penelope";
        }
        // შეცვალეთ ასო "p"   "P"-ით
        for (int i = 0; i < d.Characters.Count; ++i)
        {
            if (d.Characters[i] == 'p')
                d.Characters[i] = 'P';
```

```
        }

        Console.WriteLine(d.Text);

    }

}
```

27. generics

```
// Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

namespace Generics_CSharp

{

   //Type parameter T in angle brackets.

   public class MyList<T> : IEnumerable<T>

   {

      protected Node head;

      protected Node current = null;

      // ჩადგმული ტიპები არიან  generic T-ზე

      protected class Node

      {

         public Node next;

         //T როგორც private წევრი მონაცემთა ტიპი.

         private T data;

         //T გამოყენებული არა-generic კონსტრუქტორში.

         public Node(T t)
```

```
        {
            next = null;
            data = t;
        }
        public Node Next
        {
            get { return next; }
            set { next = value; }
        }
        //T როგორც თვისების დასაბრუნებელი ტიპი
public T Data
        {
            get { return data; }
            set { data = value; }
        }
    }

    public MyList()
    {
        head = null;
    }
```

```
//T როგორც მეთოდის პარამეტრის ტიპი.
public void AddHead(T t)
{
    Node n = new Node(t);
    n.Next = head;
    head = n;
}

// GetEnumerator-ის რეალიზაცია, დააბრუნოს  IEnumerator<T>, რომ ნება დართოს
// foreach იტერაციას ჩვენს სიაში. შევნიშნოთ, რომ C# 2.0
// თქვენ არ მოგეთხოვებათ რეალიზაცია გაუკეთოთ Current და MoveNext.
// კომპილატორი შექმნის კლასს, რომელიც განახორციელებს IEnumerator<T>-ს.
public IEnumerator<T> GetEnumerator()
{
    Node current = head;

    while (current != null)
    {
        yield return current.Data;
        current = current.Next;
    }
}
```

```
        // თქვენ უნდა განახორციელოთ ეს მეთოდი, რადგან IEnumerable<T>
// მემკვიდრეობით იღებს IEnumerable-ს.
        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }
    public class SortedList<T> : MyList<T> where T : IComparable<T>
    {
        public void BubbleSort()
        {
            if (null == head || null == head.Next)
                return;
            bool swapped;
            do
            {
                Node previous = null;
                Node current = head;
                swapped = false;
                while (current.next != null)
                {
                    //  Because we need to call this method, the SortedList
```

```
//  class is constrained on IEnumerable<T>
if (current.Data.CompareTo(current.next.Data) > 0)
{
   Node tmp = current.next;
   current.next = current.next.next;
   tmp.next = current;
   if (previous == null)
   {
      head = tmp;
   }
   else
   {
      previous.next = tmp;
   }
   previous = tmp;
   swapped = true;
}
else
{
   previous = current;
   current = current.next;
}
```

```
            }
        } while (swapped);
    }
}
// A simple class that implements IComparable<T>
// using itself as the type argument. This is a
// common design pattern in objects that are
// stored in generic lists.
public class Person : IComparable<Person>
{
    string name;
    int age;
    public Person(string s, int i)
    {
        name = s;
        age = i;
    }
    // ეს გამოიწვევს სიის ელემენტების დალაგებას ასაკის მნიშვნელობების მიხედვით.
    public int CompareTo(Person p)
    {
        return age - p.age;
    }
```

```
    public override string ToString()
    {
        return name + ":" + age;
    }
    // უნდა განახორციელოს ტოლობები.
    public bool Equals(Person p)
    {
        return (this.age == p.age);
    }
}
class Generics
{
    static void Main(string[] args)
    {
        //აცხადებს და ინიციალიზაციას უკეთებს ახალ   generic SortedList კლასს.
        //Person არის ტიპის არგუმენტი.
        SortedList<Person> list = new SortedList<Person>();
        //ჰქმნის name და age მნიშვნელობებს Person ობიექტების ინიციალიზაციისათვის.
        string[] names = new string[] { "Franscoise", "Bill", "Li", "Sandra", "Gunnar", "Alok",
"Hiroyuki", "Maria", "Alessandro", "Raul" };
        int[] ages = new int[] { 45, 19, 28, 23, 18, 9, 108, 72, 30, 35 };
        //სიის პოპულაცია.
```

```
            for (int x = 0; x < names.Length; x++)
            {
                list.AddHead(new Person(names[x], ages[x]));
            }
            Console.WriteLine("Unsorted List:");
            //დაულაგებელი სიის დაბეჭდვა.
            foreach (Person p in list)
            {
                Console.WriteLine(p.ToString());
            }
            //სიის დახარისხება.
            list.BubbleSort();
            Console.WriteLine(String.Format("{0}Sorted List:", Environment.NewLine));
            //დახარისხებული სიის დაბეჭდვა.
            foreach (Person p in list)
            {
                Console.WriteLine(p.ToString());
            }
            Console.WriteLine("Done");
        }
    }
}
```

28. cominterop2

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

// CSharpServer.cs

using System;

using System.Runtime.InteropServices;

namespace CSharpServer

{

  // რადგანაც .NET Framework interface და coclass უნდა მოიქცნენ როგორც

  // COM ობიექტები, ცვენ უნდა მივცეთ მათ მეგზურები.

  [Guid("DBE0E8C4-1C61-41f3-B6A4-4E2F353D3D05")]

  public interface IManagedInterface

  {

    int PrintHi(string name);

  }

  [Guid("C6659361-1625-4746-931C-36014B146679")]

  public class InterfaceImplementation : IManagedInterface

  {

    public int PrintHi(string name)

    {

      Console.WriteLine("Hello, {0}!", name);

      return 33;
```

```
    }

  }

}
```

29. cominterop1

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// interop2.cs
// კომპილაცია "csc interop2.cs"-ით
using System;
using System.Runtime.InteropServices;
namespace QuartzTypeLib
{
        // გამოაცხადე IMediaControl როგორც COM ინტერფეისი, რომელიც ნაწარმოებია
        // IDispatch ინტერფეისიდან:
        [Guid("56A868B1-0AD4-11CE-B03A-0020AF0BA770"),
        InterfaceType(ComInterfaceType.InterfaceIsDual)]
        interface IMediaControl
        {
                // შევნიშნოთ, რომ IUnknown Interface წევრები არ არიან ჩამოთვლილნი
// აქ:
                void Run();
                void Pause();
                void Stop();
                void GetState( [In] int msTimeout, [Out] out int pfs);
```

```
        void RenderFile(
                [In, MarshalAs(UnmanagedType.BStr)] string strFilename);
        void AddSourceFilter(
                [In, MarshalAs(UnmanagedType.BStr)] string strFilename,
                [Out, MarshalAs(UnmanagedType.Interface)]
                out object ppUnk);
        [return: MarshalAs(UnmanagedType.Interface)]
        object FilterCollection();
        [return: MarshalAs(UnmanagedType.Interface)]
        object RegFilterCollection();
        void StopWhenReady();
    }
    // გამოაცხადე FilgraphManager როგორც COM თანაკლასი:
    [ComImport, Guid("E436EBB3-524F-11CE-9F53-0020AF0BA770")]
    class FilgraphManager
// არ შეიძლება ჰქონდეს ძირითადი კლასი ან
        // interface სია აქ.
    {
        // არ შეიძლება ჰქონდეს რაიმე წევრები აქ.
        // შევნიშნოთ, რომ კომპილატორი დაუმატებს გაჩუმებით უპარამეტრო
// კონსტრუქტორს
    }
```

```
}

class MainClass
{
        /*****************************************************
        რეზიუმე: This method collects the filename of an AVI to show
        then creates an instance of the Quartz COM object.
        To show the AVI, the program calls RenderFile and Run on
        IMediaControl. Quartz uses its own thread and window to
        display the AVI. The main thread blocks on a ReadLine until
        the user presses ENTER.
                Input Parameters: the location of the avi file it is going to display
                Returns: void
        *****************************************************/

        public static void Main(string[] args)
        {
                                if (args.Length != 1)
                {
                        DisplayUsage();
                        return;
                }
```

```
if (args[0] == "/?")
{
        DisplayUsage();
        return;
}
String filename = args[0];
// შეამოწმე არსებობს თუ არა ეს ფაილი
if (!System.IO.File.Exists(filename))
{
        Console.WriteLine("ფაილი " + filename + " ვერ იპოვნა.");
        DisplayUsage();
        return;
}
// (იძახებს CoCreateInstance(E436EBB3-524F-11CE-9F53-0020AF0BA770,
//  NULL, CLSCTX_ALL, IID_IUnknown,
//  &graphManager).):
try
{
        QuartzTypeLib.FilgraphManager graphManager =
                new QuartzTypeLib.FilgraphManager();
        // QueryInterface   IMediaControl interface-ისთვის:
        QuartzTypeLib.IMediaControl mc =
```

```
                    (QuartzTypeLib.IMediaControl)graphManager;

                // ზოგიერთი მეთოდების გამოძახება COM ინტერფეისში.

                // ფაილის გადაცემა RenderFile მეთოდისთვის COM ობიექტში.

                mc.RenderFile(filename);


                // ფაილის ჩვენება.

                mc.Run();

            }

            catch(Exception ex)

            {

                Console.WriteLine("Unexpected COM exception: " + ex.Message);

            }

            // დასრულების ლოდინი.

            Console.WriteLine("დააჭირეთ Enter-ს გასაგრძელებლად.");

            Console.ReadLine();

        }

        private static void DisplayUsage()

        {

            // მომხმარებელმა არ მიაწოდა საკმარისიპარამეტრები.          //
Display usage.

            Console.WriteLine("ვიდეოპლეიერი უკრავს AVI ფაილებს.");

            Console.WriteLine("გამოყენება: VIDEOPLAYER.EXE ფაილის სახელი");
```

```
            Console.WriteLine("როცა ფაილის სახელი არის სრული გზა");
            Console.WriteLine("file name of the AVI-ის ფაილის სახელი
გამოსატანად.");
        }
}
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.
// interop1.cs
// ააგე "csc /R:QuartzTypeLib.dll interop1.cs"-ით
using System;
class MainClass
{
        /*****************************************************
        რეზიუმე: This method collects the filename of an AVI to show
        then creates an instance of the Quartz COM object.
        To show the AVI, the program calls RenderFile and Run on
        IMediaControl. Quartz uses its own thread and window to display
        the AVI.The main thread blocks on a ReadLine until the user presses
        Enter.
            Input Parameters: the location of the avi file it is going to display
            Returns: void
        *****************************************************/
        public static void Main(string[] args)
```

```
        {
                // შეამოწმე მომხმარებელმა გადასცა თუ არა ფაილის სახელი          if
(args.Length != 1)
                {
                        DisplayUsage();
                        return;
                }
                if (args[0] == "/?")
                {
                        DisplayUsage();
                        return;
                }
                string filename = args[0];
                // შეამოწმე არსებობს თუ არა ფაილი
                if (!System.IO.File.Exists(filename))
                {
                        Console.WriteLine("File " + filename + " not found.");
                        DisplayUsage();
                        return;
                }
                // შექმენი Quartz ობიექტი
                // (იძახებს CoCreateInstance(E436EBB3-524F-11CE-9F53-0020AF0BA770,
```

```
                // NULL, CLSCTX_ALL, IID_IUnknown, &graphManager).):
                try
                {
                        QuartzTypeLib.FilgraphManager graphManager =
                                new QuartzTypeLib.FilgraphManager();
                        // QueryInterface IMediaControl interface-ისთვის:
                        QuartzTypeLib.IMediaControl mc =
                                (QuartzTypeLib.IMediaControl)graphManager;
                        // ზოგიერთი მეთოდების გამოძახება COM interface-ში
                        // გადაეცი in file RenderFile მეთოდს   COM ობიექტში.
                        mc.RenderFile(filename);
                        // აჩვენე ფაილი.
                        mc.Run();
                }
                catch(Exception ex)
                {
                        Console.WriteLine("მოულოდნელი COM განსაკუთრებული
//შემთხვევა: " + ex.Message);
                }
                // დასრულების ლოდინი.
                Console.WriteLine("დააჭირეთ  Enter-ს გასაგრძელებლად.");
                Console.ReadLine();
```

```
        }

        private static void DisplayUsage()

        {

                // მომხმარებელმა არ მიაწოდა საკმარისი პარამეტრები.

                // Display-ის გამოყენება:

                Console.WriteLine("VideoPlayer: დაუკარი AVI ფაილები.");

                Console.WriteLine("გამოყენება: VIDEOPLAYER.EXE ფაილის სახელი");

                Console.WriteLine("სადაც filename არის სრული გზა და");

                Console.WriteLine("AVI-ის ფაილის სახელი გამოსატანად to display.");

        }

}
```

30. ანონიმური დელეგატები

```
//Copyright (C) Microsoft Corporation.  ყველა უფლება დაცულია.

using System;

using System.Collections.Generic;

using System.Text;

namespace AnonymousDelegate_Sample

{

    // delegate მეთოდის განსაზღვრა.

    delegate decimal CalculateBonus(decimal sales);

    // განსაზღვრე Employee ტიპი.

    class Employee

    {

        public string name;

        public decimal sales;

        public decimal bonus;

        public CalculateBonus calculation_algorithm;

    }

    class Program

    {

        // ეს კლასი განსაზღვრავს ორ დელეგატს, რომლებიც ასრულებენ გამოთვლებს.

        // პირველი იქნება სახელიანი მეთოდი, ხოლო მეორე ანონიმური დელეგატი.
```

```
        // ეს არის სახელიანი მეთოდი.
        // იგი განსაზღვრავს ბონუსის გამოთვლის ალგორითმის შესაძლო რეალიზაციას.
        static decimal CalculateStandardBonus(decimal sales)
        {
            return sales / 10;
        }
        static void Main(string[] args)
        {
            // ბონუსის გამოთვლაში გამოყენებული მნიშვნელობა.
            // შენიშვნა: ეს ლოკალური ცვლადი გახდება "დაჭერილი გარე ცვლადი".
            decimal multiplier = 2;
            // ეს დელეგატი განსაზღვრულია როგორც სახელიანი მეთოდი.
            CalculateBonus standard_bonus = new CalculateBonus(CalculateStandardBonus);
            // ეს დელეგატი ანონიმიურია.
            // იგი განსაზღვრავს ბონუსის გამოთვლის ალგორითმის ალტერნატივას.
    CalculateBonus enhanced_bonus = delegate(decimal sales) { return multiplier * sales / 10; };
            // განსაზღვრე  Employee ობიექტები.
            Employee[] staff = new Employee[5];
            // მოსამსახურეების მასივის პოპულაცია.
            for (int i = 0; i < 5; i++)
                staff[i] = new Employee();
            // დაუნიშნე საწყისი მნიშვნელობები მოსამსახურეებს.
```

```
    staff[0].name = "Mr Apple";
    staff[0].sales = 100;
    staff[0].calculation_algorithm = standard_bonus;
    staff[1].name = "Ms Banana";
    staff[1].sales = 200;
    staff[1].calculation_algorithm = standard_bonus;
    staff[2].name = "Mr Cherry";
    staff[2].sales = 300;
    staff[2].calculation_algorithm = standard_bonus;
    staff[3].name = "Mr Date";
    staff[3].sales = 100;
    staff[3].calculation_algorithm = enhanced_bonus;
    staff[4].name = "Ms Elderberry";
    staff[4].sales = 250;
    staff[4].calculation_algorithm = enhanced_bonus;
    // Calculate bonus for all Employees
    foreach (Employee person in staff)
        PerformBonusCalculation(person);
    // ყველა მოსამსახურის დეტალების გამოტანა.
    foreach (Employee person in staff)
        DisplayPersonDetails(person);
}
```

```
        public static void PerformBonusCalculation(Employee person)
        {
            // ეს მეთოდი იყენებს person ობიექტში შენახულ დელეგატს, რომ შეასრულოს გამოთვლები.
            person.bonus = person.calculation_algorithm(person.sales);
        }
        public static void DisplayPersonDetails(Employee person)
        {
            Console.WriteLine(person.name);
            Console.WriteLine(person.bonus);
            Console.WriteLine("----------------");
        }
    }
}
```

დანართი 2

გამოყენებული ინგლისურ-ქართული ტერმინოლოგიური სიტყვარი

absolute address	აბსოლი ტური მისამართი
align	ხაზზე გასწორება
anchor	ღუზა
assotiative array	ასოციატიური მასივი
autoescape	ავტომატური აცილება
auto load	ავტომატური ჩატვირთვა
batton	ღილაკი
broser	ბროუზერი
built-in functions	ჩადგმული ფუნქციები
bynary	ორობითი
call by name	არგუმენტის სახელით გამოძახება
call conventions	გამოძახების შეთანხმებები
class	კლასი
clickable image button	სურათზე დასაწკაპუნებელი ღილაკი
coding	კოდირება
concatenation	კონკატენაცია (შეერთება)
cookie	როცესის ინდიკატორი
data base	მონაცემთა ბაზა

dynamic pages	დინამიკური გვერდები
debuger	გამმართველი
debugging	გამართვა
default	გაჩუმება
default button	ღილაკი გაჩუმებით
directory	დირექტორია (კატალოგი)
dumping	რაიმეს მყისიერი მდგომარეობის ამობეჭდვა
dynamic pages	დინამიკურად შექმნილი გვერდები
environment	გარემო
errors	შეცდომები
file handle	ფაილის სახელური
form	ფორმა
field	ველი
file	ფაილი
format	ფორმატი
frame	ჩარჩო
Gateway	ჭიშკარი
gests book	სტუმართა წიგნი
GMT	გრინვიჩის მერიდიანიდან ათვლილი დროითი ზონები
header field	სათაურის ველი
host	მასპინძელი
handler	დამამუშავებელი

hardware	ტექნიკური უზრუნველყოფა
hash	ხეში
hidden field	ფარული ველი
key	გასაღები
keyword	გასაღები სიტყვა
line break	სტრიქონის გაწყვეტა
list	სია
load	ჩატვირთვა
magic kookie	ჯადოქრული სიგნალი
menu	მენიუ
name argument	სახელიანი არგუმენტი
named parameter	სახელიანი პარამეტრი
namespace	სახელთა არე
object	ობიექტი
object oriented style	ობიექტზე ორიენტირებული სტილი
parameter	პარამეტრი
parameters call by list	პარამეტრების სიით გამოძახება
password field	პასპორტის ველი
patch	გზა
pipe	არხი
pointer	მიმთითებელი
popup menu	სტეკური მენიუ

pragmas	მიმთითებელი ტრანსლიატორისათვის
proxy – server	proxy – სერვერი
query	შეკითხვა
query form	შეკითხვის ფორმა
radio button	რადიო ღილაკი
reference	მითითება
related checkbooxes	დაკავშირებული მოსანიშნი უჯრები
relative address	ფარდობითი მისამართი
remote user	შორეული მომხმარებელი
reset button	ღილაკის აღდგენა
script	სკრიპტი
scripting	სკრიპტების გამოყენება
scrolling list	გასასრიალებელი სია
shortcats	შემოკლებები
site	საიტი
software	პროგრამული უზრუნველყოფა
space	ხარვეზისნიშანი
standalone checkbox	სწრაფად მოსანიშნი უჯრაKA
submit	დადასტურება, წარდგენა
table	ცხრილი
tag	ჭდე
text field	ტექსტის ველი

URL	მისამართის მითითება ინტერნეტში
user	მომხმარებელი
utility	უტილიტა (დამხმარე პროგრამა)
variable	ცვლადი
web page	ვებ გვერდი
window	ფანჯარა (ეკრანზე გამოსატანი, ეკრანზე გამოტანილი)

C# ლიტერატურა

1. C# programming guide. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/index
2. C# Tutorial. https://www.tutorialspoint.com/csharp/
3. The complete C# Tutorial. http://csharp.net-tutorials.com/
4. C#Tutorial. http://www.tutorialsteacher.com/csharp/csharp-tutorials
5. Selenium C Sharp. http://toolsqa.com/selenium-c-sharp/
6. Free C# NET Course. http://www.homeandlearn.co.uk/csharp/csharp.html
7. LEARN VISUAL C# PROGRAMMING. HTTP://WWW.C-SHARPCORNER.COM/BEGINNERS/
8. Joe Mayo. C# succinctly
9. Object-Oriented Programming Concepts. http://www.blackwasp.co.uk/ObjectOrientedConcepts.aspx
10. FUNDAMENTALS OF COMPUTER PROGRAMMING WITH C# http://www.introprogramming.info/wp-content/uploads/2013/07/Books/CSharpEn/Fundamentals-of-Computer-Programming-with-CSharp-Nakov-eBook-v2013.pdf
11. C# Tutorial. http://a4academics.com/tutorials/86-c-sharp
12. C# Tutorial. http://www.guru99.com/c-tutorial.html
13. Advanced C#. H.Mössenböck University of Linz, Austria, moessenboeck@ssw.uni-linz.ac.at, http://www.ssw.uni-linz.ac.at/Teaching/Lectures/CSharp/Tutorial/Part2.pdf
14. C# programming guide. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/index
15. C# Reference. https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/
16. C# Tutorial. https://www.tutorialspoint.com/csharp/
17. The complete C# Tutorial. http://csharp.net-tutorials.com/
18. C#Tutorial. http://www.tutorialsteacher.com/csharp/csharp-tutorials
19. Selenium C Sharp. http://toolsqa.com/selenium-c-sharp/
20. Free C# NET Course. http://www.homeandlearn.co.uk/csharp/csharp.html
21. LEARN VISUAL C# PROGRAMMING. HTTP://WWW.C-SHARPCORNER.COM/BEGINNERS/