პრაქტიკული #6-2
თვისებები და ინდექსატორები

თვისებები

თვისება არის კლასის წევრი, რომელიც საშუალებას გვაძლევს მეთოდების საშუალებით მივიღოთ ან შევცვალოთ ცვლადების (ველების) მნიშვნელობები. თვისებების გამოყენება მოხერხებულია მაშინ, როცა საჭიროა ცვლადის მნიშვნელობებზე შესასრულებელი ოპერაციების კონტროლი (იმის კონტროლი თუ რომელი ოპერაციებია დასაშვები ცვლადის მნიშვნელობებზე და რომელი - არა). მაგალითად, შეიძლება საჭირო გახდეს მნიშვნელობების დიაპაზონის შეზღუდვა, რომელიც ამ ცვლადს ენიჭება. ბუნებრივია, ასეთი მიზნის მისაღწევად შეგვიძლია გამოვიყენოთ პრივატული ცვლადი და მეთოდი, რომელიც მასთან მუშაობს. მაგრამ გაცილებით მარტივი და უკეთესია პრივატული ცვლადებისა და თვისებების ერთობლივი გამოყენება.

თვისება შედგება სახელისა და მიმართვის get და set მეთოდებისაგან. მიმართვის get მეთოდი გამოიყენება ცვლადის მნიშვნელობის მისაღებად, set მეთოდი კი ცვლადისთვის მნიშვნელობის მისანიჭებლად. set მეთოდს ავტომატურად გადაეცემა value არაცხადი პარამეტრი, რომელიც შეიცავს ცვლადისათვის მისანიჭებელ მნიშვნელობას. თვისების ძირითადი ღირსებაა ის, რომ მისი სახელი შეიძლება გამოვიყენოთ გამოსახულებებში და მინიჭების ოპერაციებში, როგორც ჩვეულებრივი ცვლადი. ამ დროს ავტომატურად გამოიძახება მიმართვის get და set მეთოდები. გარდა ამისა, თვისება მართავს ცვლადთან მიმართვას. თვისებაში არ ხდება ცვლადის გამოცხადება. საჭიროების მიხედვით თვისებაში შეიძლება განისაზღვროს მხოლოდ get ან set მეთოდი, ან ორივე ერთად.

თვისების სინტაქსია:

მიმართვის_მოდიფიკატორი ტიპი თვისების_სახელი
{
get
{
get *მეთოდის კოდი*
}
set
{
set *მეთოდის კოდი*
}
}

აქ *ტიპი* თვისების ტიპია (მაგალითად, int ტიპი). თვისების განსაზღვრის შემდეგ მისი სახელის გამოყენება, იწვევს მიმართვის შესაბამისი მეთოდის გამოძახებას. თუ თვისების სახელი მითითებულია მინიჭების ოპერატორის მარცხნივ, მაშინ გამოიძახება set მეთოდი. თუ თვისების სახელი მითითებულია მინიჭების ოპერატორის მარჯვნივ, მაშინ გამოიძახება get მეთოდი.

ქვემოთ მოყვანილ პროგრამაში განისაზღვრება ChemiTviseba თვისება, რომელიც უზრუნველყოფს Tviseba ცვლადთან მიმართვას. მოცემულ შემთხვევაში თვისება უშვებს ცვლადისათვის მხოლოდ დადებითი მნიშვნელობების მინიჭებას.

```
class MartiviTviseba
{
int cvladi;
public MartiviTviseba()
{
cvladi = 0;
}
```

```
//      ChemiTviseba თვისების გამოცხადება
public int ChemiTviseba
{
get
{
        return cvladi;
}
set
{
//      tviseba ცვლადს ენიჭება value მნიშვნელობა თუ ის დადებითია
        if ( value >= 0 ) cvladi = value;
}
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//
MartiviTviseba obieqti = new MartiviTviseba();
int ricxvi = Convert.ToInt32(textBox1.Text);

label1.Text = obieqti.ChemiTviseba.ToString();
//      ChemiTviseba თვისებას ენიჭება ricxvi ცვლადის დადებითი მნიშვნელობა
obieqti.ChemiTviseba = ricxvi;          //      ricxvi ცვლადის მნიშვნელობა ავტომატურად
                                        //      ენიჭება  value ცვლადს
label2.Text = obieqti.ChemiTviseba.ToString();
//      obieqti.ChemiTviseba თვისებისთვის უარყოფითი მნიშვნელობის მინიჭების მცდელობა
obieqti.ChemiTviseba = -ricxvi;
label3.Text = obieqti.ChemiTviseba.ToString();
}
```

ახლა დაწვრილებით განვიხილოთ პროგრამა. მასში განისაზღვრება ერთი პრივატული cvladi ცვლადი და ChemiTviseba თვისება, რომელიც მართავს ამ ცვლადთან მიმართვას. რადგან cvladi ცვლადი არის დახურული, მასთან მიმართვა შესაძლებელია მხოლოდ ChemiTviseba თვისების გამოყენებით.

ChemiTviseba თვისება განისაზღვრება როგორც public, ამიტომ მასთან მიმართვა შეიძლება შესრულდეს სხვა მეთოდების მხრიდან. მიმართვის get მეთოდი აბრუნებს ცვლადის მნიშვნელობას, ხოლო მიმართვის set მეთოდი კი cvladi ცვლადს ანიჭებს მნიშვნელობას მხოლოდ იმ შემთხვევაში, თუ ის არის დადებითი. ასეთი გზით, ChemiTviseba თვისება აკონტროლებს cvladi ცვლადისათვის მნიშვნელობების მინიჭებას.

თვისებების გამოყენებისას უნდა დავიცვათ შემდეგი შეზღუდვები:

- თვისება არ შეიძლება გადაეცეს მეთოდს, როგორც ref ან out პარამეტრი.
- თვისება არ შეიძლება იყოს გადატვირთული.
- თვისებამ არ უნდა შეცვალოს ცვლადის მნიშვნელობა get მეთოდის გამოყენებით.

ინდექსატორები

ინდექსატორი საშუალებას გვაძლევს ობიექტში მოთავსებულ ცვლადებს, აგრეთვე, მასივის ელემენტებს მივმართოთ ობიექტის სახელისა და ინდექსის საშუალებით. ინდექსატორების გამოყენების ძირითადი სფეროა სპეციალიზებული მასივების შექმნა, რომლებსაც ედებათ გარკვეული შეზღუდვები. ინდექსატორების სინტაქსი მასივების სინტაქსის

მსგავსია. ინდექსატორებს შეიძლება ჰქონდეთ ერთი ან მეტი განზომილება.

ერთგანზომილებიანი ინდექსატორები

ერთგანზომილებიანი ინდექსატორის სინტაქსია:

მიმართვის_მოდიფიკატორი ელემენტის_ტიპი this[*ინდექსის_ტიპი ინდექსი*]
{
მიმართვის get *მეთოდი*
get
{
get *მეთოდის კოდი*
}
მიმართვის set *მეთოდი*
set
{
set *მეთოდის კოდი*
}
}

აქ *ელემენტის_ტიპი* არის ინდექსატორის მიერ გაცემული ცვლადის ტიპი.

ინდექსატორის შემადგენლობაში განსაზღვრულია მიმართვის ორი მეთოდი get და set. ჩვეულებრივი მეთოდისაგან განსხვავებით მიმართვის მეთოდისათვის არ ეთითება დასაბრუნებელი მნიშვნელობის ტიპი და პარამეტრები. მიმართვის ორივე მეთოდი ინდექსატორის გამოყენებისას ავტომატურად გამოიძახება და პარამეტრებად იღებენ მნიშვნელობებს, რომლებიც მითითებულია ინდექსის ნაცვლად. თუ ინდექსატორი მითითებულია მინიჭების ოპერატორის მარცხენა ნაწილში, გამოიძახება მიმართვის set მეთოდი. თუ ინდექსატორი იმყოფება მინიჭების ოპერატორის მარჯვენა ნაწილში, მაშინ გამოიძახება მიმართვის get მეთოდი. საჭიროების მიხედვით ინდექსატორში შეიძლება განსაზღვრული იყოს მხოლოდ get ან მხოლოდ set მეთოდი, ან ორივე ერთად.

ქვემოთ მოყვანილ პროგრამაში ინდექსატორი გამოიყენება ობიექტის ორ ცვლადთან სამუშაოდ.

```
class Klasi
{
 private int ricxvi1;
 private int ricxvi2;
 private int ricxvi3;
 private int ricxvi4;
 private int ricxvi5;
 //      ინდექსატორის გამოცხადება
 public int this[int index]
 {
  get
  {
   switch ( index )
   {
    case 0: return ricxvi1;
    case 3: return ricxvi4;
```

```
     case 4: return ricxvi5;
     default: return 0;
    }
   }
   set
   {
    switch ( index )
    {
     case 0: this.ricxvi1 = value; break;
     case 1: this.ricxvi2 = value; break;
     case 2: this.ricxvi3 = value; break;
     case 3: this.ricxvi4 = value; break;
     case 4: this.ricxvi5 = value; break;
    }
   }
  }
 }

  private void button1_Click(object sender, EventArgs e)
 {
   label1.Text = "";
   Klasi obj1 = new Klasi();
   int ind;
   //      obj1 ობიექტის ცვლადებისთვის მნიშვნელობების მინიჭება ინდექსატორის გამოყენებით
   for ( ind = 0; ind < 5; ind++ )
    obj1[ind] = ind + 10;
   //      obj1 ობიექტის ცვლადების მნიშვნელობების ეკრანზე გამოტანა
   for ( ind = 0; ind < 5; ind++ )
    label1.Text += obj1[ind].ToString() + "   ";
}
```

ინდექსატორის გამოყენების ერთ-ერთი უპირატესობა არის ის, რომ ის იძლევა მასივთან მიმართვის ზუსტი კონტროლის შესაძლებლობას. კერძოდ, შეგვიძლია ვაკონტროლოთ გადის თუ არა მითითებული ინდექსი მასივის საზღვრებს გარეთ. ამის დემონსტრირება ხდება ქვემოთ მოყვანილ პროგრამაში.

```
 class ChemiKlasi
 {
  int[] masivi;                    //          masivi არის მასივზე მითითებელი
  public int Sigrdze;
  //       გადის თუ არა ინდექსი დიაპაზონის გარეთ
  public ChemiKlasi(int zoma)
  {
   masivi = new int[zoma];
   Sigrdze = zoma;
  }
  //       ინდექსატორის გამოცხადება
  public int this[int index]
  {
```

```
   //      მიმართვის get მეთოდი
   get
   {
    if ( ( index >= 0 ) && ( index < Sigrdze ) )
     return masivi[index];
    else return 0;
   }
   //      მიმართვის set მეთოდი
   set
   {
    if ( ( index >= 0 ) && ( index < Sigrdze ) )
     masivi[index] = value;
   }
  }
 }
private void button1_Click(object sender, System.EventArgs e)
{
label1.Text = "";
Random rand_obj = new Random();
ChemiKlasi obieqti = new ChemiKlasi(5);
int indexi;
//          set მეთოდის გამოძახება
for ( indexi = 0; indexi < 5; indexi++ )
          obieqti[indexi] = rand_obj.Next(10);
//          get მეთოდის გამოძახება
for ( indexi = 0; indexi < 5; indexi++ )
          label1.Text += obieqti[indexi].ToString() + "    ";
}
```

ინდექსატორების გამოყენებისას არსებობს ერთი მნიშვნელოვანი შეზღუდვა: მათთვის არ შეიძლება ref ან out მოდიფიკატორების გამოყენება.

მრავალგანზომილებიანი ინდექსატორები

ინდექსატორები შეიძლება შეიქმნას მრავალგანზომილებიანი მასივებისთვისაც. ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია ორგანზომილებიან ინდექსატორთან მუშაობის მაგალითი.

```
class ChemiKlasi
 {
  int[,] masivi;
  int striqoni, sveti;
  public int Length;
  //      მითითებული პარამეტრების მქონე მასივის შექმნა
  public ChemiKlasi(int par1, int par2)
  {
   striqoni = par1;
   sveti = par2;
   masivi = new int[striqoni, sveti];
   Length = striqoni * sveti;
  }
```

```
  //      ორგანზომილებიანი ინდექსატორის გამოცხადება
  public int this[int index1, int index2]
  {
   //      მიმართვის get მეთოდი
   get
   {
    if ( ( ( index1 >= 0 ) && ( index1 < striqoni ) ) && ( ( index2 >= 0 ) && ( index2 < sveti ) ) )
     return masivi[index1, index2];
    else return 0;
   }
   //      მიმართვის set მეთოდი
   set
   {
    if ( ( ( index1 >= 0 ) && ( index1 < striqoni ) ) && ( ( index2 >= 0 ) && ( index2 < sveti ) ) )
     masivi[index1, index2] = value;
   }
  }
 }
private void button1_Click(object sender, System.EventArgs e)
{
label1.Text = "";
Random rand_obj = new Random();
ChemiKlasi obieqti = new ChemiKlasi(3, 3);
int striqoni, sveti;
//      მუშაობს set მეთოდი
for ( striqoni = 0; striqoni < 3; striqoni++ )
        for ( sveti = 0; sveti < 3; sveti++ )
                obieqti[striqoni, sveti] = rand_obj.Next(10);
//      მუშაობს get მეთოდი
for ( striqoni = 0; striqoni < 3; striqoni++ )
{
for ( sveti = 0; sveti < 3; sveti++ )
        label1.Text += obieqti[striqoni, sveti].ToString() + "   ";
   label1.Text += "\n";
}
}
}
```

სავარჯიშოები

1. შექმენით კლასი, რომელშიც პრივატული მთელი ტიპის ცვლადია გამოცხადებული. მასთან მიმართვისათვის გამოიყენეთ თვისება, რომელშიც set და get მეთოდებია განსაზღვრული. თვისება პრივატულ ცვლადს დადებით მნიშვნელობებს ანიჭებს.
2. შექმენით კლასი, რომელშიც პრივატული მთელი ტიპის ცვლადია გამოცხადებული. მასთან მიმართვისათვის გამოიყენეთ თვისება, რომელშიც set და get მეთოდებია განსაზღვრული. თვისება პრივატულ ცვლადს კენტ მნიშვნელობებს ანიჭებს.
3. შექმენით კლასი, რომელშიც პრივატული მთელი ტიპის ცვლადია გამოცხადებული. მასთან მიმართვისათვის გამოიყენეთ თვისება, რომელშიც set და get მეთოდებია განსაზღვრული. თვისება პრივატულ ცვლადს 5-ის ჯერად მნიშვნელობებს ანიჭებს.
4. შექმენით კლასი, რომელშიც გამოცხადებულია 5 მთელი ტიპის ცვლადი. მათ მიანიჭეთ მნიშვნელობები ინდექსატორის გამოყენებით.
5. შექმენით კლასი, რომელშიც გამოცხადებულია 5 წილადი ტიპის ცვლადი. მათ მიანიჭეთ მნიშვნელობები ინდექსატორის გამოყენებით.
6. შექმენით კლასი, რომელშიც გამოცხადებულია 5 ლოგიკური ცვლადი. მათ მიანიჭეთ მნიშვნელობები ინდექსატორის გამოყენებით.
7. შექმენით კლასი, რომელშიც გამოცხადებულია 5 სტრიქონი. მათ მიანიჭეთ მნიშვნელობები ინდექსატორის გამოყენებით.
8. შექმენით კლასი, რომელშიც გამოცხადებულია მთელრიცხვა ერთგანზომილებიანი მასივი. ინდექსატორის გამოყენებით იპოვეთ მასივის ელემენტების ჯამი.
9. შექმენით კლასი, რომელშიც გამოცხადებულია წილადების ორგანზომილებიანი მასივი. ინდექსატორის გამოყენებით იპოვეთ მასივის მინიმალური ელემენტი.

პრაქტიკული მეცადინეობა #6-1

სტატიკური წევრები. static მოდიფიკატორი. მუდმივები

სტატიკური წევრები. static მოდიფიკატორი

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

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

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

ქვემოთ მოყვანილ პროგრამაში ნაჩვენებია სტატიკურ ცვლადთან და მეთოდთან მუშაობა.

//პროგრამაში ხდება სტატიკურ ცვლადთან და მეთოდთან მუშაობის დემონსტრირება

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

```
}
```

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

სტატიკურ მეთოდებს შემდეგი შეზღუდვები აქვთ:

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

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

```
//      პროგრამაში ხდება სტატიკური მეთოდის მიერ ჩვეულებრივი მეთოდის
//      გამოძახების დემონსტრირება
class ChemiKlasi
{
  int chveulebrivi_cvladi;                  //ჩვეულებრივი ცვლადის გამოცხადება
  static int statikuri_cvladi = 555;        //სტატიკური ცვლადის გამოცხადება
  //     ჩვეულებრივი მეთოდი
  void ChveulebriviMetodi(Label lab1)
  {
   chveulebrivi_cvladi = 50;
   statikuri_cvladi = 70;
   lab1.Text = "ჩვეულებრივი ცვლადი - " + chveulebrivi_cvladi.ToString() +
       "\nსტატიკური ცვლადი - " + statikuri_cvladi.ToString();
  }
  //     სტატიკური მეთოდი
  public static void StatikuriMetodi(ChemiKlasi obieqti, Label lab2)
  {
   obieqti.ChveulebriviMetodi(lab2);        //მეთოდის ეს გამოძახება ნამდვილია
  }
}
private void button1_Click(object sender, System.EventArgs e)
{
  ChemiKlasi obj = new ChemiKlasi();

  ChemiKlasi.StatikuriMetodi(obj, label1);
}
```

მუდმივები

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

მოყვანილ პროგრამაში ხდება მუდმივასთან მუშაობის დემონსტრირება.

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

მხოლოდწაკითხვადი ველი

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

მოყვანილ პროგრამაში ხდება მხოლოდწაკითხვად ველთან მუშაობის დემონსტრირება.

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

სავარჯიშოები

სტატიკური კომპონენტები. static მოდიფიკატორი

1. შექმენით კლასი. მასში გამოაცხადეთ პრივატული მთელი ტიპის ჩვეულებრივი და სტატიკური ცვლადები. გამოაცხადეთ ღია ჩვეულებრივი მეთოდი, რომელსაც ორი მთელი ტიპის პარამეტრი აქვს. მეთოდი პარამეტრების მნიშვნელობებს ანიჭებს ობიექტის ცვლადებს და აბრუნებს მათ ჯამს. გამოაცხადეთ ღია სტატიკური მეთოდი, რომელსაც სამი პარამეტრი აქვს: პირველი პარამეტრია ამავე კლასის ობიექტი, მეორე და მესამე კი - მთელი ტიპის რიცხვები. ეს მეთოდი იძახებს ჩვეულებრივ მეთოდს. მთავარ პროგრამაში შექმენით ამ კლასის ობიექტი. გამოიძახეთ სტატიკური მეთოდი. გაცემული შედეგი მიანიჭეთ ცვლადს და გამოიტანეთ Label ვიზუალურ ელემენტში. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი.
2. შექმენით Martkutxedi კლასი, რომელიც შეიცავს სტატიკურ ცვლადებს - მართკუთხედის გვერდებს და ჩვეულებრივ ცვლადს - მართკუთხედის პერიმეტრს. შეიცავს კონსტრუქტორს, რომელიც სტატიკური ცვლადების ინიციალიზებას ასრულებს და პრივატულ ჩვეულებრივ მეთოდს, რომელიც ანგარიშობს და გასცემს მართკუთხედის პერიმეტრის მნიშვნელობას. შეიცავს ღია სტატიკურ მეთოდს, რომელიც ჩვეულებრივ მეთოდს იძახებს და გასცემს პერიმეტრის მნიშვნელობას. მთავარ პროგრამაში შექმენით ობიექტი კონსტრუქტორის გამოყენებით და გამოიძახეთ სტატიკური მეთოდი.
3. შექმენით Klasi_1 კლასი, რომელიც შეიცავს: მთელი ტიპის ერთგანზომილებიან სტატიკურ მასივს, რომლის ინიციალიზებას კონსტრუქტორი ახდენს; პრივატულ ჩვეულებრივ მეთოდს, რომელიც ერთგანზომილებიანი სტატიკური მასივის კენტი ელემენტების ჯამს გასცემს; ღია სტატიკურ მეთოდს, რომელიც ჩვეულებრივ მეთოდს იძახებს და გასცემს შედეგს. მთავარ პროგრამაში შექმენით ობიექტი კონსტრუქტორის გამოყენებით და გამოიძახეთ სტატიკური მეთოდი.
4. შექმენით Klasi_2 კლასი, რომელიც შეიცავს: მთელი ტიპის ორგანზომილებიან სტატიკურ მასივს, რომლის ინიციალიზებას კონსტრუქტორი ახდენს; პრივატულ ჩვეულებრივ მეთოდს, რომელიც ორგანზომილებიანი სტატიკური მასივის კენტი ელემენტების რაოდენობას გასცემს; ღია სტატიკურ მეთოდს, რომელიც ჩვეულებრივ მეთოდს იძახებს და გასცემს შედეგს. მთავარ პროგრამაში შექმენით ობიექტი კონსტრუქტორის გამოყენებით და გამოიძახეთ სტატიკური მეთოდი.

პრაქტიკული სამუშაო #5-2

შემთხვევითი რიცხვების გენერატორი. სტრიქონები

შემთხვევითი რიცხვების გენერატორი

C# ენაში განსაზღვრულია System.Random კლასი შემთხვევით რიცხვებთან სამუშაოდ. ცხრილში 1 მოყვანილია ამ კლასის მეთოდები, რომლებიც ახდენენ შემთხვევითი რიცხვების გენერირებას.

ცხრილი 1. Random კლასის მეთოდები.

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

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

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

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

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

```
{
//      პროგრამაში ხდება შემთხვევითი რიცხვების გენერატორის გამოყენება
//      ერთგანზომილებიანი მასივის შესავსებად
label1.Text="";
byte[] masivi = new byte[5];
System.Random obieqti = new System.Random();
```

```
obieqti.NextBytes(masivi);
for ( int i = 0; i < masivi.Length; i++ )
        label1.Text += masivi[i].ToString() + "   ";
}
```

სტრიქონები

სტრიქონები გამოიყენება Unicode სიმბოლოების მიმდევრობის შესანახად. ერთი Unicode სიმბოლო იკავებს 2 ბაიტს ანუ 16 ბიტს. სტრიქონები წარმოადგენენ System.String კლასის მიმართვითი ტიპის მქონე ობიექტებს. ისინი აღიწერება string სიტყვის საშუალებით. სტრიქონის (string ტიპის ობიექტის) შექმნის ყველაზე მარტივი საშუალებაა სტრიქონული ლიტერალის გამოყენება. მაგალითად,

```
string striqoni = "ეს არის სტრიქონული ლიტერალი";
```

აქ striqoni ცვლადი არის string ტიპის მიმართვითი ცვლადი, რომელსაც ენიჭება სტრიქონულ ლიტერალზე ("ეს არის სტრიქონული ლიტერალი") მიმართვა. ამ შემთხვევაში striqoni ცვლადის ინიციალიზება ხდება სიმბოლოების მიმდევრობით - "ეს არის სტრიქონული ლიტერალი".

ისევე როგორც მასივებში, აქაც პირველი ელემენტის (სიმბოლოს) ინდექსია 0. ინდექსი გამოიყენება მხოლოდ სიმბოლოს მისაღებად. მაგალითად,

```
string striqoni = "კომპიუტერი";
label1.Text = striqoni[2];
```

ორივე ოპერატორის შესრულების შედეგად label კომპონენტში გამოჩნდება ასო "მ". რადგან, სტრიქონები ობიექტებს წარმოადგენენ, ამიტომ მათ Length თვისება აქვთ. მასში ინახება სტრიქონის სიგრძე (სტრიქონში სიმბოლოების რაოდენობა).

სტრიქონი შეიძლება შეიცავდეს მმართველ სიმბოლოებს. მაგლითად,

```
string striqoni = "საბა \n ანა";
label1.Text = striqoni;
```

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

შეგვიძლია შევქმნათ ისეთი სტრიქონებიც, რომლებშიც მმართველი სიმბოლოები არ იმუშავებენ. ასეთი სტრიქონები @ სიმბოლოთი იწყება. მაგალითად,

```
string striqoni = @" საბა \n ანა";
label1.Text = striqoni;
```

ამ სტრიქონების შესრულების შედეგად label1 კომპონენტში გამოჩნდება სტრიქონი - "საბა \n ანა".

მოყვანილ პროგრამაში ხდება Length თვისებასთან მუშაობის დემონსტრირება.

```
{
string striqoni1 = @" საბა \n ანა \n";
label1.Text = striqoni1.Length.ToString() + '\n';                    //      გაიცემა 15
string striqoni2 = " საბა \n ანა \n";
label2.Text = striqoni2.Length.ToString() + '\n';                    //      გაიცემა 13
}
```

განვიხილოთ ზოგიერთი მეთოდი.

ქვემოთ მოყვანილ პროგრამაში ხდება **Insert, Remove** და **Replace** მეთოდების გამოყენების დემონსტრირება.

```
{
// პროგრამაში ხდება Insert, Remove და Replace მეთოდებთან მუშაობის დემონსტრირება
string str1, str2 = "C# პროგრამირების ენაა", str3 = "თანამედროვე ";
```

```
str1 = str2.Insert(17, str3);          //      მიიღება სტრიქონი "C# პროგრამირების თანამედროვე ენაა"
label1.Text = str1;
str1 = str2.Remove(3, 14);                                  //      მიიღება სტრიქონი "C# ენაა"
label2.Text = str1;
str1 = str2.Replace("პროგრამირების ", str3);          //   მიიღება სტრიქონი "C# თანამედროვე ენაა"
label3.Text = str1;
}
```

პროგრამის str1 = str2.Insert(17, str3); სტრიქონში str2 არის გამომძახებელი სტრიქონის სახელი. მის მარჯვნივ წერტილის დასმის შემდეგ გაიხსნება სტრიქონებთან სამუშაო მეთოდების სია. ვირჩევთ Insert მეთოდს. მისი შესრულების შედეგად str2 სტრიქონის მე-17 პოზიციიდან მოხდება str3 სტრიქონის ჩასმა, ანუ "თანამედროვე" სტრიქონის ჩასმა. შედეგად, მიიღება სტრიქონი "C# პროგრამირების თანამედროვე ენაა", რომელიც მიენიჭება str1 ცვლადს.

+ ოპერატორი გამოიყენება სტრიქონების კონკატენაციის (შეერთების) ოპერაციის შესასრულებლად. განვიხილოთ კოდის ფრაგმენტი:

```
string striqoni1 = "ეს არის ";
string striqoni2 = "კომ";
string striqoni3 = "პიუტერი";
string striqoni   = striqoni1 + striqoni2 + striqoni3;
```

ამ ოპერატორების შესრულების შედეგად striqoni ცვლადს მიენიჭება სტრიქონი - "ეს არის კომპიუტერი".

ორი სტრიქონის შესადარებლად შეგვიძლია == ოპერატორის გამოყენება.

მოყვანილ პროგრამაში ხდება **Copy, CompareTo, IndexOf, LastIndexOf, Substring** მეთოდების მუშაობის დემონსტრირება.

```
{
//      პროგრამაში ხდება Copy, CompareTo, IndexOf, LastIndexOf, Substring
//      მეთოდებთან მუშაობის დემონსტრირება
string striqoni1, striqoni2 = "ერთი ორი სამი ერთი ორი სამი",
        striqoni3 = textBox1.Text;
int shedegi, indexi;

//      striqoni2  სტრიქონი გადაიწერება striqoni1 სტრიქონში
striqoni1 = string.Copy(striqoni2);
label1.Text = striqoni1;

striqoni1 = striqoni2.Substring(9, 13);          //      მიიღება სტრიქონი "სამი ერთი ორი"
label2.Text = striqoni1;

indexi = striqoni2.IndexOf("ორი");               //      გაიცემა 5
label3.Text = indexi.ToString();
indexi = striqoni2.LastIndexOf("ორი");                   //      გაიცემა 19
label4.Text = indexi.ToString();

//      striqoni2  და striqoni4 სტრიქონების შედარება
shedegi = striqoni2.CompareTo(striqoni3);
if ( shedegi == 0 ) label5.Text = "სტრიქონები ერთმანეთის ტოლია";
        else if ( shedegi > 0 ) label5.Text = " striqoni2 > striqoni3";
                        else label5.Text = " striqoni2 < striqoni3";
}
```

Compare() და **Equals()** მეთოდები. Equals() მეთოდი ორ სტრიქონს ადარებს და გასცემს true მნიშვნელობას თუ სტრიქონები ტოლია, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას. მაგალითი.

```
{
string striqoni1 = "საბა ანა ლიკა რომანი",
       striqoni2 = "ანა რომანი საბა ლიკა";
bool shedegi = striqoni1.Equals(striqoni2);
if ( shedegi == true ) label1.Text = "სტრიქონები ერთნაირია";
        else label1.Text = " სტრიქონები ერთნაირი არ არის";
}
```

Compare() მეთოდი ორ სტრიქონს ადარებს ენის, ეროვნული და კულტურული თავისებურებების გათვალისწინებით. თუ პირველი სტრიქონი მეტია მეორეზე, მაშინ გაცემა 1. თუ პირველი სტრიქონი ნაკლებია მეორეზე, მაშინ გაიცემა -1. თუ სტრიქონები ტოლია, მაშინ გაიცემა 0. ეს მეთოდი გადატვირთულია და ამიტომ აქვს რამდენიმე ვერსია. ჩვენ განვიხილავთ რამდენიმე მათგანს. მაგალითი.

```
{
string striqoni1 = "საბა ანა ლიკა რომანი",
       striqoni2 = "ანა რომანი საბა ლიკა";
int  shedegi = String.Compare(striqoni1, striqoni2);
if ( shedegi == 0 ) label1.Text = "სტრიქონები ერთნაირია";
        else label1.Text = " სტრიქონები ერთნაირი არ არის";
}
```

Compare() მეთოდის მეორე ვერსია შედარების დროს ითვალისწინებს სიმბოლოების რეგისტრს. თუ რეგისტრი იღებს true მნიშვნელობას, მაშინ შედარების დროს რეგისტრი არ იქნება გათვალისწინებული. თუ ის იღებს false მნიშვნელობას, მაშინ შედარების დროს რეგისტრი იქნება გათვალისწინებული. მაგალითი.

```
{
string striqoni1 = "საბა ანა ლიკა რომანი",
     striqoni2 = "ანა რომანი საბა ლიკა";
int  shedegi = String.Compare(striqoni1, striqoni2, true);
if ( shedegi == 0 ) label1.Text = "სტრიქონები ერთნაირია";
        else label1.Text = " სტრიქონები ერთნაირი არ არის";
}
```

Compare() მეთოდის მესამე ვერსია საშუალებას გვაძლევს შევადაროთ ორი სტრიქონის ნაწილები (ქვესტრიქონები). მოყვანილ პროგრამაში ხდება ამ მეთოდთან მუშაობის დემონსტრირება.

```
{
string striqoni1 = "საბა ანა ლიკა რომანი",
       striqoni2 = "ანა რომანი საბა ლიკა";
int  shedegi = String.Compare(striqoni1, 5, striqoni2, 5, 4);
if ( shedegi == 0 ) label1.Text = "სტრიქონები ერთნაირია";
        else label1.Text = " სტრიქონები ერთნაირი არ არის";
}
```

მოყვანილ პროგრამაში ხდება **Concat()** მეთოდთან მუშაობის დემონსტრირება.

```
{
string striqoni1 = "საბა";
string striqoni2 = string.Concat(striqoni1, " ანა");
string striqoni3 = striqoni2 + " ლიკა";
```

```
label1.Text = striqoni3;
}
```

როგორც პროგრამის კოდიდან ჩანს სტრიქონების კონკატენაციისთვის შეიძლება გამოვიყენოთ + ოპერატორიც.

მოყვანილ პროგრამაში ხდება **StartWith()** და **EndWith()** მეთოდთან მუშაობის დემონსტრირება.

```
{
//      პროგრამაში ხდება StartWith() და EndWith() მეთოდებთან მუშაობის დემონსტრირება
string striqoni1, striqoni2;
striqoni1 = textBox1.Text;
striqoni2 = textBox2.Text;
if ( striqoni1.StartsWith(striqoni2) )
        label1.Text = "სტრიქონი იწყება " + striqoni2 + " სტრიქონით";
    else  label1.Text = " სტრიქონი არ იწყება " + striqoni2 + " სტრიქონით";
if ( striqoni1.EndsWith(striqoni2) )
        label2.Text = "სტრიქონი მთავრდება " + striqoni2 + " სტრიქონით";
    else  label2.Text = " სტრიქონი არ მთავრდება " + striqoni2 + " სტრიქონით";
}
```

IndexOfAny() და **LastIndexOfAny()** მეთოდები გადატვირთულია და ამიტომ აქვთ რამდენიმე ვერსია. ყველაზე მარტივი ვერსიის სინტაქსია:

int IndexOfAny(char[ ] *სიმბოლოების_მასივი*)

int LastIndexOfAny(char[ ] *სიმბოლოების_მასივი*)

ამ მეთოდების მეორე ვერსია საშუალებას გვაძლევს ძებნა დავიწყოთ მითითებული ინდექსდან. ამ ვერსიის სინტაქსია:

int IndexOfAny(char[ ] *სიმბოლოების_მასივი, ინდექსი*)

int LastIndexOfAny(char[ ] *სიმბოლოების_მასივი, ინდექსი*)

ამ მეთოდების მესამე ვერსია საშუალებას გვაძლევს მივუთითოთ შესამოწმებელი სიმბოლოების რაოდენობა. ამ ვერსიის სინტაქსია:

int IndexOfAny(char[ ] *სიმბოლოების_მასივი, ინდექსი, სიმბოლოების_რაოდენობა*)

int LastIndexOfAny(char[ ] *სიმბოლოების_მასივი, ინდექსი, სიმბოლოების_რაოდენობა*)

მოყვანილ პროგრამაში ხდება IndexOfAny() და LastIndexOfAny() მეთოდებთან მუშაობის დემონსტრირება.

```
{
//      პროგრამაში ხდება IndexOfAny() და LastIndexOfAny()მეთოდებთან მუშაობის
//      დემონსტრირება
char[] simboloebis_masivi = { 'ნ', 'ა' };
string striqoni1 = "ლიკა, ანა და რომანი";

int indexi1 = striqoni1.IndexOfAny(simboloebis_masivi);              //      indexi1 = 3
int indexi2 = striqoni1.LastIndexOfAny(simboloebis_masivi);          //      indexi2 = 17
label1.Text = indexi1.ToString() + "   " + indexi2.ToString();

int indexi3 = striqoni1.IndexOfAny(simboloebis_masivi, 5);           //      indexi3 = 6
int indexi4 = striqoni1.LastIndexOfAny(simboloebis_masivi, 5);       //      indexi4 = 3
label2.Text = indexi3.ToString() + "   " + indexi4.ToString();

int indexi5 = striqoni1.IndexOfAny(simboloebis_masivi, 5, 6);        //      indexi5 = 6
int indexi6 = striqoni1.LastIndexOfAny(simboloebis_masivi, 5, 6);    //      indexi5 = 3
```

```
label3.Text = indexi5.ToString() + "   " + indexi6.ToString();      //        6 3
}
```

PadLeft() და **PadRight()** მეთოდებთან მუშაობის დემონსტრირება ხდება მოყვანილ პროგრამაში.

```
{
string striqoni2 = "საბა";
string striqoni1 = striqoni2.PadLeft(10, '*');                    //        striqoni1 = "******საბა"
string striqoni3 = striqoni2.PadRight(10, '*');                   //        striqoni3 = "საბა******"
label1.Text = striqoni1.ToString();
label2.Text = striqoni3.ToString();
}
```

Trim(), TrimStart() და **TrimEnd()** მეთოდები გადატვირთულია. Trim() მეთოდის ყველაზე მარტივი ვერსია, რომელშიც არ ეთითება პარამეტრი, ახდენს ინტერვალების წაშლას სტრიქონის დასაწყისში და ბოლოში. TrimStart() მეთოდის ყველაზე მარტივი ვერსია ახდენს ინტერვალების წაშლას სტრიქონის დასაწყისში. TrimEnd() მეთოდის ყველაზე მარტივი ვერსია ახდენს ინტერვალების წაშლას სტრიქონის ბოლოში.

ამ მეთოდების მეორე ვერსია საშუალებას გვაძლევს სტრიქონს დასაწყისში და ბოლოში მოვაცილოთ მითითებული სიმბოლოები. მოყვანილ პროგრამაში ხდება Trim(), TrimStart() და TrimEnd() მეთოდებთან მუშაობის დემონსტრირება.

```
{
string striqoni2, striqoni3, striqoni4 = ";,რომანი.,;";
char[] simboloebi = { '.', ';', ',' };

label1.Text = striqoni4.Trim(simboloebi);
striqoni2   = striqoni4.TrimStart(simboloebi);                    //        striqoni2   = "რომანი.,;"
striqoni3   = striqoni4.TrimEnd(simboloebi);              //        striqoni3   = ";,რომანი"
label2.Text = striqoni2.ToString();
label3.Text = striqoni3.ToString();
}
```

მოყვანილ პროგრამაში ხდება **ToLower()** და **ToUpper()** მეთოდებთან მუშაობის დემონსტრირება.

```
{
string striqoni1 = "ANA SABA";
string striqoni2 = "lika romani";

label15.Text = striqoni1.ToLower();                               //        "ana saba"
label17.Text = striqoni2.ToUpper();                               //        "LIKA ROMANI"
}
```

სტრიქონების მასივები

სტრიქონების მასივის გამოცხადებისთვის string სიტყვის შემდეგ კვადრატული ფრჩხილები უნდა მივუთითოთ. ქვემოთ მოყვანილ პროგრამაში ხდება str სტრიქონების მასივის გამოცხადება და ინიციალიზება. ის სამ სტრიქონულ ლიტერალს შეიცავს.

```
{
//        პროგრამაში ხდება სტრიქონების მასივთან მუშაობის დემონსტრირება
string[] striqonebis_masivi = { "სტრიქონული ", "მასივის ", "მაგალითი" };
//        მასივის გამოტანა label კომპონენტში
for ( int indexi = 0; indexi < striqonebis_masivi.Length; indexi++ )
```

```
        label1.Text += striqonebis_masivi[indexi] + " ";
//      სტრიქონული მასივის მნიშვნელობების შეცვლა
striqonebis_masivi[0] = "მეორე ";
striqonebis_masivi[1] = "სტრიქონული ";
striqonebis_masivi[2] = textBox1.Text;
label1.Text += '\n';
//      მასივის გამოტანა label კომპონენტში
foreach ( string striqoni in striqonebis_masivi )
        label1.Text += striqoni + " ";
}
```

სტრიქონების უცვლელობა

string ტიპის ობიექტების ერთ-ერთი მნიშვნელოვანი თავისებურებაა ის, რომ სტრიქონში ერთხელ შექმნილი სიმბოლოების მიმდევრობა აღარ შეიცვლება, ე.ი. არ შეიძლება ინდექსის გამოყენება სტრიქონის რომელიმე სიმბოლოსთვის ახალი მნიშვნელობის მისანიჭებლად. მაგალითად, დაუშვებელია ასეთი მინიჭება:

```
striqoni1[1] = 'ს';
```

იმისთვის, რომ სტრიქონში სიმბოლოები შევცვალოთ, უნდა შევქმნათ ახალი სტრიქონი, რომელიც საჭირო ცვლილებებს შეიცავს. ამ მიზნით, შეგვიძლია Replace ან Substring მეთოდის გამოყენება. ქვემოთ მოყვანილ პროგრამაში ხდება სტრიქონის შეცვლის დემონსტრირება.

```
{
//      პროგრამაში ხდება სტრიქონის შეცვლის დემონსტრირება
string str1, str2, str3;
str1 = "პროგრამირება";
//      str1[0] = 'ბ';                  //      ასეთი მინიჭება დაუშვებელია
str2 = str1.Replace('ა', 'ბ');          //      str2 სტრიქონში ჩაიწერება შეცვლილი სტრიქონი
str3 = str1.Substring(2, 3);            //      str3 სტრიქონში ჩაიწერება ახალი სტრიქონი
str1 = str1.Replace("პროგ", "აბგ");     //      ასეთი მინიჭება დასაშვებია. ამ შემთხვევაში
                                        //      str1 სტრიქონშივე ჩაიწერება შეცვლილი სტრიქონი
label1.Text = str3;
label2.Text = str1;
}
```

სავარჯიშოები

შემთხვევითი რიცხვების გენერატორი

შეადგინეთ პროგრამა, რომელიც:

1. მოახდენს [1,100] ინტერვალში მოთავსებული 20 შემთხვევითი რიცხვის გენერირებას.
2. დათვლის თუ რამდენჯერ გვხვდება q რიცხვი [1,1000] ინტერვალში 50 მცდელობის შემდეგ.
3. ჯერ მოახდენს 100 მთელი რიცხვის გენერირებას [1,100] ინტერვალში, შემდეგ კი დათვლის თუ რამდენჯერ გამოჩნდა თითოეული რიცხვი ამ ინტერვალში.
4. განსაზღვრავს თუ რამდენი მთელი რიცხვის გენერირება უნდა მოვახდინოთ [1,30] ინტერვალიდან, რომ მოცემული R რიცხვი მათ შორის შეგვხვდეს 7-ჯერ.
5. განსაზღვრავს თუ რამდენი ასანთი უნდა ამოვიღოთ ასანთის 5 ყუთიდან იმისთვის, რომ დაცარიელდეს ერთ-ერთი. თითო ყუთში 20 ასანთია.
6. შეადგინეთ პროგრამა, რომელიც შემთხვევითი რიცხვების გენერატორის გამოყენებით ერთგანზომილებიან მასივს ჯერ შეავსებს მთელი რიცხვებით 0-დან 20-მდე დიაპაზონში, შემდეგ კი ეკრანზე გამოიტანს.
7. შეადგინეთ პროგრამა, რომელიც შემთხვევითი რიცხვების გენერატორის გამოყენებით ერთგანზომილებიან მასივს ჯერ შეავსებს წილადი რიცხვებით, შემდეგ კი ეკრანზე გამოიტანს.
8. შეადგინეთ პროგრამა, რომელიც შემთხვევითი რიცხვების გენერატორის გამოყენებით ბაიტების ერთგანზომილებიან მასივს, ჯერ შეავსებს შემსეგ კი ეკრანზე გამოიტანს.
9. შეადგინეთ პროგრამა, რომელიც შემთხვევითი რიცხვების გენერატორის გამოყენებით ორგანზომილებიან მასივს ჯერ შეავსებს მთელი რიცხვებით 4-დან 24-მდე დიაპაზონში, შემდეგ კი ეკრანზე გამოიტანს.
10. შეადგინეთ პროგრამა, რომელიც შემთხვევითი რიცხვების გენერატორის გამოყენებით ორგანზომილებიან მასივს ჯერ შეავსებს წილადი რიცხვებით, შემდეგ კი ეკრანზე გამოიტანს.
11. შეადგინეთ პროგრამა, რომელიც შემთხვევითი რიცხვების გენერატორის გამოყენებით შეავსებს ბაიტების ერთგანზომილებიან მასივს, შემდეგ კი ეკრანზე გამოიტანს. ეკრანზე გამოიტანს ასევე, ამ მასივის ელემენტების ჯამს.
12. შეადგინეთ პროგრამა, რომელიც შემთხვევითი რიცხვების გენერატორის გამოყენებით ერთგანზომილებიან მასივს ჯერ შეავსებს მთელი რიცხვებით 7-დან 27-მდე დიაპაზონში, შემდეგ კი ეკრანზე გამოიტანს.
13. შეადგინეთ პროგრამა, რომელიც შემთხვევითი რიცხვების გენერატორის გამოყენებით ერთგანზომილებიან მასივს ჯერ შეავსებს მთელი რიცხვებით -8-დან 28-მდე დიაპაზონში, შემდეგ კი ეკრანზე გამოიტანს.
14. შეადგინეთ პროგრამა, რომელიც შემთხვევითი რიცხვების გენერატორის გამოყენებით ერთგანზომილებიან მასივს ჯერ შეავსებს მთელი რიცხვებით 29-დან 99-მდე დიაპაზონში, შემდეგ კი ეკრანზე გამოიტანს.

სტრიქონები

1. სტრიქონში იპოვის "ა" სიმბოლოს პოზიციას.
2. სტრიქონში ყველა "ი" სიმბოლოს "ს" სიმბოლოთი შეცვლის.
3. სტრიქონში ყველა ციფრს "რ" სიმბოლოთი შეცვლის.
4. სტრიქონში "ეს არის ტესტი" ჩაუმატებს "კომპიუტერი" სიტყვას დაწყებული მე-3 პოზიციიდან.
5. სტრიქონიდან "ეს არის ტესტი" წაშლის "არის" სიტყვას.

6. სტრიქონიდან "ეს არის ტესტი" წაშლის 6 სიმბოლოს დაწყებული მე-2 პოზიციიდან.
7. სტრიქონში "ეს არის ტესტი" "ეს" სიტყვას შეცვლის "კომპიუტერი" სიტყვით.
8. სტრიქონში "ეს არის ტესტური პროგრამა" "ეს არის" სიტყვებს შეცვლის "ახლა შევასრულოთ" სიტყვებით.
9. შეადგინეთ პროგრამა, რომელიც სტრიქონში „C# დაპროგრამების თანამედროვე ენაა“ სტრიქონს „თანამედროვე“ შეცვლის სტრიქონით „მძლავრი“.
10. შეადგინეთ პროგრამა, რომელიც სტრიქონში „C# პროგრამირების ენაა“ ჩაუმატებს სტრიქონს „თანამედროვე “ დაწყებული მე-18 პოზიციიდან.
11. შეადგინეთ პროგრამა, რომელიც სტრიქონიდან „C# პროგრამირების თანამედროვე ენაა“ წაშლის სტრიქონს „თანამედროვე“.
12. შეადგინეთ პროგრამა, რომელიც სტრიქონში „C# პროგრამირების თანამედროვე ენაა“ იპოვის „თანამედროვე“ სტრიქონის ინდექსს.
13. შეადგინეთ პროგრამა, რომელიც სტრიქონში „C# პროგრამირების თანამედროვე ენაა“ იპოვის „თანამედროვე“ სტრიქონის ინდექსს. ძებნა შეასრულეთ სტრიქონის ბოლოდან.

პრაქტიკული სამუშაო #4

მემკვიდრეობითობა

protected მოდიფიკატორი

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

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

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

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

შეადგინეთ Sibrtye საბაზო კლასი, რომელშიც გამოცხადებულია მთელი ტიპის სამი დაცული ცვლადი. გამოცხადებულია კონსტრუქტორი, რომელიც ამ დაცულ ცვლადებს მნიშვნელობებს ანიჭებს. გამოცხადებულია ღია მეთოდი, რომელისაც ამ ცვლადების მნიშვნელობები ეკრანზე გამოაქვს. შექმენით მემკვიდრე Samkutxedi კლასი, რომელშიც გამოცხადებულია პრივატული მთელი ტიპის ცვლადი და ორი ღია მეთოდი: ერთ მეთოდი პროვატულ ცვლადს ანიჭებს მართკუთხა სამკუთხედის ფართობის მნიშვნელობას, მეორე მეთოდი გვიბრუნებს ამ ცვლადის მნიშვნელობას. ძირითად პროგრამაში შექმენით მემკვიდრე კლასის ტიპის მქონე ობიექტი და გამოიძახეთ მემკვიდრე კლასის მეთოდები.

```
class Sibrtye
{
protected int gverdi1, gverdi2, gverdi3; // ეს ცვლადები ღიაა Samkutxedi კლასის წევრებისთვის
public void Inicializacia(int a, int b, int c)
{
gverdi1 = a;
```

```
gverdi2 = b;
gverdi3 = c;
}
public string Naxva()
{
return gverdi1.ToString() + " " + gverdi2.ToString() + " " + gverdi3.ToString();
}
}
class Samkutxedi : Sibrtye
{
int shedegi;
// Partobi() მეთოდს შეუძლია მიმართოს Sibrtye კლასში
// განსაზღვრულ gverdi1, gverdi2 და gverdi3 ცვლადებს
public void Partobi()
{
shedegi = gverdi1 * gverdi2 / 2;
}
public string PartobisNaxva()
{
return shedegi.ToString();
}
}
private void button1_Click(object sender, System.EventArgs e)
{
// პროგრამაში ხდება protected მოდიფიკატორის გამოყენება
Samkutxedi obieqti = new Samkutxedi();
int ricxvi1 = int.Parse(textBox1.Text);
int ricxvi2 = int.Parse(textBox2.Text);
int ricxvi3 = int.Parse(textBox3.Text);
// Inicializacia() და Naxva() მეთოდები მისაწვდომია Samkutxedi კლასის ობიექტიდან
obieqti.Inicializacia(ricxvi1, ricxvi2, ricxvi3);
label1.Text = obieqti.Naxva();
obieqti.Partobi();
label2.Text = obieqti.PartobisNaxva();
}
```

კონსტრუქტორები და მემკვიდრეობითობა. base საკვანძო სიტყვა

წინაპარ და მემკვიდრე კლასებს შეიძლება ჰქონდეთ საკუთარი კონსტრუქტორები. ამ დროს წამოიჭრება კითხვა, თუ რომელი კონსტრუქტორი ქმნის მემკვიდრე კლასის ობიექტს. თითოეული კონსტრუქტორი ქმნის (ინიციალიზებას უკეთებს) ობიექტის თავის ნაწილს, ე.ი. ობიექტი ნაწილ-ნაწილ იქმნება. ეს იმიტომ ხდება, რომ წინაპარი კლასის კონსტრუქტორს არ შეუძლია მიმართოს მემკვიდრე კლასის წევრებს და მოახდინოს მათი ინიციალიზება. თუ კონსტრუქტორი არ არის განსაზღვრული წინაპარ კლასში და განსაზღვრულია მემკვიდრე კლასში, მაშინ მემკვიდრე კლასის ობიექტის კონსტრუირება სრულდება მისი კონსტრუქტორის მიერ, ხოლო წინაპარ კლასში

განსაზღვრული ობიექტის ნაწილი კი იქმნება მისი ავტომატური კონსტრუქტორის მიერ.

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

base საკვანძო სიტყვა ეთითება მემკვიდრე კლასის კონსტრუქტორის გამოცხადების გაფართოებულ ფორმაში.

შეადგინეთ Kvadrati საბაზო კლასი, რომელშიც გამოცხადებულია მთელი ტიპის ერთი დაცული ცვლადი და ღია კონსტრუქტორი, რომელიც ამ ცვლადს კვადრატის გვერდის მნიშვნელობას ანიჭებს. შექმენით Martkutxedi მემკვიდრე კლასი, რომელშიც გამოცხადებულია მთელი ტიპის ერთი ღია გვერდი და კონსტრუქტორი, რომელიც წინაპარი კლასის კონსტრუქტორს იძახებს. შექმენით Martkutxedi კლასის მემკვიდრე Samkutxedi კლასი, რომელშიც გამოცხადებულია მთელი ტიპის ერთი პრივატული გვერდი და კონსტრუქტორი, რომელიც წინაპარი კლასის კონსტრუქტორს იძახებს. ამავე კლასში გამოცხადებულია ღია მეთოდი, რომელიც Label კომპონენტში სამკუთხედის სამივე გვერდის მნიშვნელობა გამოაქვს.

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

```
}
}

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

სავარჯიშოები

protected მოდიფიკატორი

1. შექმენით საბაზო კლასი, რომელიც შეიცავს სამკუთხედის გვერდებს. ერთი გვერდი არის ღია, მეორე და მესამე დაცული. შექმენით სამკუთხედის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს პრივატულ ცვლადებს - სამკუთხედის პერიმეტრს და ფართობს; ასევე ორ ღია მეთოდს. პირველი მეთოდი გამოთვლის და აბრუნებს სამკუთხედის ფართობს. მეორე მეთოდი გამოთვლის და გასცემს სამკუთხედის პერიმეტრს. მთავარ პროგრამაში შექმენით მემკვიდრე კლასის ტიპის მქონე ობიექტი და გამოიძახეთ ორივე მეთოდი.
2. შექმენით საბაზო კლასი, რომელიც შეიცავს დაცულ ცვლადს - მართკუთხედის ფუძეს. შექმენით მართკუთხედის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს: პრივატულ ცვლადებს - მართკუთხედის სიმაღლეს და ფართობს; ღია მეთოდს, რომელიც გამოთვლის და აბრუნებს მართკუთხედის ფართობს. მთავარ პროგრამაში შექმენით მემკვიდრე კლასის ტიპის მქონე ობიექტი და გამოიძახეთ მეთოდი.

base საკვანძო სიტყვა

1. შექმენით გეომეტრიული ფიგურის Figura საბაზო კლასი, რომელშიც სამი დაცული გვერდია გამოცხადებული. განსაზღვრულია ასევე ორი კონსტრუქტორი. პირველ კონსტრუქტორს სამი პარამეტრი აქვს და ახდენს სამივე გვერდის ინიციალიზებას. მეორე კონსტრუქტორს ერთი პარამეტრი აქვს და ახდენს ერთი გვერდის ინიციალიზებას. შექმენით მისი მემკვიდრე Samkutxedi და Kvadrati კლასები. Samkutxedi კლასში გამოცხადებულია პრივატული ცვლადი - პერიმეტრი. Samkutxedi კლასის კონსტრუქტორი იძახებს წინაპარი კლასის კონსტრუქტორს გვერდების ზომების განსაზღვრის მიზნით. თვით ამ კონსტრუქტორში გამოითვლება სამკუთხედის პერიმეტრი. ამავე კლასის მეორე ღია მეთოდი გვიბრუნებს სამკუთხედის პერიმეტრს. Kvadrati კლასში გამოცხადებულია პრივატული ცვლადი - ფართობი. Kvadrati კლასის კონსტრუქტორი იძახებს წინაპარი კლასის კონსტრუქტორს გვერდის ზომის განსაზღვრის მიზნით. თვით ამ კონსტრუქტორში ხდება კვადრატის ფართობის გამოთვლა. ამავე კლასის მეორე ღია მეთოდი გვიბრუნებს კვადრატის ფართობს. მთავარ პროგრამაში შექმენით Samkutxedi და Kvadrati კლასის ტიპის მქონე ობიექტები და გამოიძახეთ მეთოდები.
2. შექმენით ტელევიზორის საბაზო კლასი - Televizori. მასში გამოაცხადეთ დაცული ცვლადები: ტელევიზორი 1 საათში რამდენ ვატს მოიხმარს და რამდენი საათის განმავლობაშია ჩართული, და კონსტრუქტორი, რომელიც დაცულ წევრებს მნიშვნელობებს ანიჭებს. შექმენით Televizori კლასის მემკვიდრე კლასი - Memkvidre_1. მასში გამოაცხადეთ პრივატული ცვლადი - ტელევიზორის მიერ დახარჯული (მოხმარებული) ვატების რაოდენობა. გამოაცხადეთ კონსტრუქტორი, რომელიც იძახებს წინაპარი კლასის კონსტრუქტორს მემკვიდრეობითობით მიღებული ცვლადების ინიციალიზების მიზნით. თვით ეს კონსტრუქტორი ახდენს ამავე კლასის ცვლადის ინიციალიზებას. ამავე კლასში განსაზღვრეთ ღია მეთოდი, რომელსაც ეკრანზე გამოაქვს ყველა ცვლადის მნიშვნელობა. მთავარ პროგრამაში შექმენით მემკვიდრე კლასის ტიპის მქონე ობიექტი და გამოიძახეთ მეთოდი.

3. შექმენით საბაზო კლასი, რომელიც შეიცავს დაცულ ცვლადს: მართკუთხედის ფუძეს; ღია კონსტრუქტორს, რომელიც დაცულ ცვლადს მნიშვნელობას ანიჭებს. შექმენით მართკუთხედის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს ღია ცვლადს - მართკუთხედის სიმაღლეს და პრივატულ ცვლადს - მართკუთხედი ფართობს; ღია მეთოდს, რომელიც გამოთვლის და აბრუნებს მართკუთხედის ფართობის მნიშვნელობას. მთავარ პროგრამაში შექმენით მემკვიდრე კლასის ტიპის მქონე ობიექტი და გამოიძახეთ მეთოდი.
4. შექმენით პიროვნების საბაზო კლასი, რომელიც შეიცავს დაცულ ცვლადებს: გვარს, სახელსა და ასაკს; ღია კონსტრუქტორს, რომელიც დაცულ ცვლადებს მნიშვნელობებს ანიჭებს. შექმენით ექიმის მემკვიდრე კლასი, რომელიც დამატებით შეიცავს ღია ცვლადებს: განყოფილების დასახელებას, თანამდებობას, საავადმყოფოს დასახელებას და სტაჟს. შეიცავს ღია მეთოდებს. პირველი მეთოდია კონსტრუქტორი, რომელიც საწყის მნიშვნელობებს ანიჭებს ღია ცვლადებს და იძახებს წინაპარი კლასის კონსტრუქტორს, მემკვიდრეობითობით მიღებული ცვლადების ინიციალიზების მიზნით. მეორე მეთოდს ეკრანზე გამოაქვს ყველა ცვლადი. მთავარ პროგრამაში შექმენით მემკვიდრე კლასის ტიპის მქონე ობიექტი და გამოიძახეთ მეთოდი.

პრაქტიკული სამუშაო #3

პოლიმორფიზმი

მეთოდის გადატვირთვა

C# ენაში ერთი კლასის შიგნით ორ ან მეტ მეთოდს შეიძლება ერთნაირი სახელი ჰქონდეს იმ პირობით, რომ ან პარამეტრების რაოდენობა უნდა იყოს სხვადასხვა ან პარამეტრებს სხვადასხვა ტიპი უნდა ჰქონდეთ. ასეთ მეთოდებს გადატვირთვადი ეწოდებათ, ხოლო ერთნაირი სახელის მქონე მეთოდების განსაზღვრის პროცესს კი მეთოდის გადატვირთვა. მეთოდების გადატვირთვა არის პოლიმორფიზმის რეალიზების ერთ-ერთი საშუალება. გადატვირთვადი მეთოდის გამოძახებისას სრულდება ის მეთოდი, რომლის პარამეტრებიც ემთხვევა არგუმენტებს. გადატვირთვადი მეთოდები შეიძლება აბრუნებდნენ, სხვადასხვა ტიპის მონაცემებს.

შეადგინეთ კლასი, რომელშიც გამოცხადებული ერთნაირი სახელის მქონე ოთხი მეთოდი: პირველ მეთოდს პარამეტრები არ აქვს და გვიბრუნებს სტრიქონს, მეორე მეთოდს ერთი წილადი ტიპის პარამეტრი აქვს და გვიბრუნებს მის კვადრატს, მესამე მეთოდს მთელი ტიპის ორი პარამეტრი აქვს და გვიბრუნებს მათ ჯამს, მეოთხე მეთოდს ორი წილადი პარამეტრი აქვს და გვიბრუნებს მათ ნამრავლს. ძირითადი პროგრამიდან გამოიძახეთ ეს მეთოდები.

// პროგრამაში ხდება მეთოდის გადატვირთვის დემონსტრირება

```
class Gadatvirtva
{
// მეთოდს პარამეტრები არ აქვს და აბრუნებს სტრიქონს
public string Metodi()
{
return "გამოცხადებული იქნა უპარამეტრებო მეთოდი";
}
// მეთოდს double ტიპის ერთი პარამეტრი აქვს და აბრუნებს მის კვადრატს
public double Metodi(double par1)
{
return Math.Pow(par1, 2);
}
// მეთოდს int ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს
public int Metodi(int par1, int par2)
{
return par1 + par2;
}
// მეთოდს double ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს
public double Metodi(double par1, double par2)
{
```

```
return par1 * par2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Gadatvirtva obieqti = new Gadatvirtva();
int ricxvi1;
double wiladi1, wiladi2;
string striqoni;
// Metodi მეთოდის ყველა ვერსიის გამოძახება
striqoni = obieqti.Metodi();
label1.Text = striqoni;
wiladi2 = obieqti.Metodi(5.7);
label2.Text = "გამოძახებული იქნა ერთპარამეტრიანი მეთოდი. შედეგი = " +
wiladi2.ToString();
ricxvi1 = obieqti.Metodi(5, 10);
label3.Text = "გამოძახებული იქნა ორპარამეტრიანი მეთოდი. შედეგი = " +
ricxvi1.ToString();
wiladi1 = obieqti.Metodi(5.5, 10.10);
label4.Text = " გამოძახებული იქნა ორპარამეტრიანი მეთოდი. შედეგი = " +
wiladi1.ToString();
}
```

კონსტრუქტორის გადატვირთვა

მეთოდების მსგავსად შესაძლებელია კონსტრუქტორების გადატვირთვაც. ეს საშუალებას გვაძლევს ობიექტები შევქმნათ სხვადასხვა საშუალებებით.

შეადგინეთ კლასი, რომელშიც გამოცხადებულია ოთხი კონსტრუქტორი. პირველ კოპნსტრუქტორს პარამეტრები არ აქვს და ობიექტის ცვლადს ნულოვან მნიშვნელობას ანიჭებს. მეორე კონსტრუქტორს ერთი მთელი ტიპის პარამეტრი აქვს და ობიექტის ცვლადს ამ პარამეტრის კვადრატს ანიჭებს. მესამე კონსტრუქტორს ერთი წილადი პარამეტრი აქვს და ობიექტის ცვლადს ანიჭებს 10-ით გაზრდილ პარამეტრის მნიშვნელობას. მეოთხე კონსტრუქტორს ორი მთელი პარამეტრი აქვს, რომელთა ნამრავლს ანიჭებს ობიექტის ცვლადს. ძირითად პროგრამაში ამ კონსტრუქტორების გამოყენებით შექმენით ოთხი ობიექტი:

```
// პროგრამაში ხდება გადატვირთვადი კონსტრუქტორის დემონსტრირება
class ChemiKlasi
{
public int ricxvi;
public ChemiKlasi()
```

```
{
ricxvi = 0;
}
public ChemiKlasi(int par1)
{
ricxvi = par1 * par1;
}
public ChemiKlasi(double par2)
{
ricxvi = ( int ) par2 + 10;
}
public ChemiKlasi(int par3, int par4)
{
ricxvi = par3 * par4;
}
}
private void button2_Click(object sender, EventArgs e)
{
int ricxvi = Convert.ToInt32(textBox1.Text);
ChemiKlasi obieqti1 = new ChemiKlasi();
label1.Text = "" + obieqti1.ricxvi.ToString();
ChemiKlasi obieqti2 = new ChemiKlasi(ricxvi);
label2.Text = "" + obieqti2.ricxvi.ToString();
ChemiKlasi obieqti3 = new ChemiKlasi(28.89);
label3.Text = "" + obieqti3.ricxvi.ToString();
ChemiKlasi obieqti4 = new ChemiKlasi(3, 5);
label4.Text = "" + obieqti4.ricxvi.ToString();
}
```

კონსტრუქტორის გადატვირთვა this სიტყვის გამოყენებით

რიგ შემთხვევებში გადატვირთვადი კონსტრუქტორის გამოძახებისას სასარგებლოა გამოვიყენოთ ერთი კონსტრუქტორის მიერ მეორის გამოძახება. C# ენაში ეს რეალიზდება this სიტყვის გამოყენებით. გამომძახებელი კონსტრუქტორის შესრულებისას ჯერ გამოიძახება ის გადატვირთვადი კონსტრუქტორი, რომელშიც პარამეტრების სია ემთხვევა არგუმენტების სიას, შემდეგ კი შესრულდება გამომძახებელი კონსტრუქტორის დანარჩენი ოპერატორები.

შეადგინეთ კლასი, რომელშიც გამოცხადებულია სამი კონსტრუქტორი. პირველ კოპნსტრუქტორს პარამეტრები არ აქვს, იძახებს ამავე კლასის კონსტრუქტორს, რომელსაც ორი მთელი პარამეტრი აქვს და ეკრანზე გამოაქვს რაიმე სტრიქონი. მეორე

კონსტრუქტორს პარამეტრად აქვს ამავე კლასის ობიექტი, იძახებს ამავე კლასის კონსტრუქტორს, რომელსაც ორი მთელი პარამეტრი აქვს და ეკრანზე გამოაქვს რაიმე სტრიქონი. მესამე კონსტრუქტორს ორი მთელი პარამეტრი აქვს, ობიექტის ცვლადებს ანიჭებს პარამეტრების მნიშვნელობებს და ეკრანზე გამოაქვს რაიმე სტრიქონი. ძირითად პროგრამაში ამ კონსტრუქტორების გამოყენებით შექმენით სამი ობიექტი:

```
// პროგრამაში ხდება გადატვირთვადი კონსტრუქტორის გამოძახება this
// სიტყვის გამოყენებით
class ChemiKlasi
{
public int cvladi1, cvladi2;
public ChemiKlasi() : this(0,0)
{
MessageBox.Show("ეს მეთოდი განსაზღვრულია ChemiKlasi() კონსტრუქტორში");
}
public ChemiKlasi(ChemiKlasi obieqti) : this(obieqti.cvladi1, obieqti.cvladi2)
{
MessageBox.Show("ეს მეთოდი განსაზღვრულია ChemiKlasi(obieqti.cvladi1, obieqti.cvladi2)
კონსტრუქტორში");
}
public ChemiKlasi(int par1, int par2)
{
MessageBox.Show("ეს მეთოდი განსაზღვრულია ChemiKlasi(int, int) კონსტრუქტორში");
cvladi1 = par1;
cvladi2 = par2;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
// ხდება this სიტყვის გამოყენების დემონსტრირება ერთი კონსტრუქტორის
// გამოსაძახებლად მეორის მიერ
int ricxvi1 = int.Parse(textBox1.Text);
int ricxvi2 = int.Parse(textBox2.Text);
ChemiKlasi obieqti1 = new ChemiKlasi();
ChemiKlasi obieqti2 = new ChemiKlasi(ricxvi1, ricxvi2);
ChemiKlasi obieqti3 = new ChemiKlasi(obieqti2);
label1.Text = obieqti1.cvladi1.ToString() + " " + obieqti1.cvladi2.ToString();
label2.Text = obieqti2.cvladi1.ToString() + " " + obieqti2.cvladi2.ToString();
label3.Text = obieqti3.cvladi1.ToString() + " " + obieqti3.cvladi2.ToString();
}
```

სავარჯიშოები

მეთოდის გადატვირთვა

1. შეადგინეთ სამკუთხედის კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 2 მეთოდი. პირველ მეთოდს 2 მთელრიცხვა პარამეტრი აქვს: სამკუთხედის სიმაღლე და ფუძე და აბრუნებს მართკუთხა სამკუთხედის ფართობს. მეორე მეთოდს 3 მთელრიცხვა პარამეტრი აქვს: სამკუთხედის გვერდები და აბრუნებს სამკუთხედის პერიმეტრს. ძირითად პროგრამაში შექმენით შესაბამისი ობიექტი და გამოიძახეთ ორივე მეთოდი.
2. შექმენით ავტომობილის კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 2 მეთოდი. პირველ მეთოდს 2 მთელრიცხვა პარამეტრი აქვს: ბაკის ტევადობა და მანძილი, რომელსაც გაივლის ავტომობილი 1 ლიტრი საწვავით, და აბრუნებს ავტომობილის მიერ სავსე ბაკით გავლილ მანძილს. მეორე მეთოდს 2 წილადი პარამეტრი აქვს: მაქსიმალური სიჩქარე და მოძრაობის დრო, და აბრუნებს მანძილს, რომელსაც გაივლის ავტომობილი მითითებული დროის განმავლობაში მაქსიმალური სიჩქარით მოძრაობისას. ძირითად პროგრამაში შექმენით შესაბამისი ობიექტი და გამოიძახეთ ორივე მეთოდი.
3. შეადგინეთ მატარებლის კლასი, რომელშიც განსაზღვრულია ერთი და იგივე სახელის მქონე 2 მეთოდი. პირველ მეთოდს 2 მთელრიცხვა პარამეტრი აქვს: ვაგონების რაოდენობა და ერთ ვაგონში მგზავრების რაოდენობა, და აბრუნებს მგზავრების საერთო რაოდენობას. მეორე მეთოდს 2 წილადი პარამეტრი აქვს: 1 კილომეტრის გავლისას დახარჯული ელექტროენერგია და გავლილი მანძილი, და აბრუნებს მატარებლის მიერ მითითებული მანძილის გავლისას დახარჯულ ელექტროენერგიას. ძირითად პროგრამაში შექმენით შესაბამისი ობიექტი და გამოიძახეთ ორივე მეთოდი.

კონსტრუქტორის გადატვირთვა

1. შექმენით კლასი, რომელიც მთელი ტიპის Min_Max_1 ცვლადს და წილადი ტიპის Min_Max_2 ცვლადს შეიცავს. კლასის I კონსტრუქტორი, რომელსაც გადაეცემა ერთგანზომილებიანი მთელრიცხვა მასივი, Min_Max_1 ცვლადს ანიჭებს მასივის მინიმალურ ელემენტს. კლასის II კონსტრუქტორს პარამეტრად გადაეცემა ერთგანზომილებიანი წილადი ტიპის მასივი, Min_Max_2 ცვლადს ანიჭებს მასივის მაქსიმალურ ელემენტს. ძირითად პროგრამაში შექმენით ორი ობიექტი კონსტრუქტორების გამოყენებით.
2. შექმენით ფიგურის კლასი, რომელიც სამ გვერდს, პერიმეტრს, ფართობსა და სამ კონსტრუქტორს შეიცავს. I კონსტრუქტორს 1 მთელი ტიპის პარამეტრი აქვს და ქმნის კვადრატს (გამოთვლის პერიმეტრსა და ფართობს). II კონსტრუქტორს 2 მთელი ტიპის პარამეტრი აქვს და ქმნის მართკუთხედს (გამოთვლის პერიმეტრსა და ფართობს). III კონსტრუქტორს 3 მთელი ტიპის პარამეტრი აქვს და ქმნის

სამკუთხედს (გამოთვლის პერიმეტრსა და ფართობს). ძირითად პროგრამაში შექმენით სამი ობიექტი კონსტრუქტორების გამოყენებით.

გადატვირთვადი კონსტრუქტორის გამოძახება this სიტყვის გამოყენებით

1. შექმენით კვადრატის კლასი, რომელიც სამ კონსტრუქტორს შეიცავს. I კონსტრუქტორს 1 მთელი ტიპის პარამეტრი აქვს, რომელსაც ამავე კლასის ცვლადს ანიჭებს. II კონსტრუქტორს პარამეტრები არ აქვს, იძახებს I კონსტრუქტორს და მას 0-ს გადასცემს. III კონსტრუქტორი, რომლის პარამეტრია ამავე კლასის ობიექტი, იძახებს I კონსტრუქტორს და მას გადასცემს ობიექტის ცვლადის მნიშვნელობას. II და III კონსტრუქტორებში სრულდება კვადრატის პერიმეტრის გამოთვლა. ძირითად პროგრამაში შექმენით სამი ობიექტი კონსტრუქტორების გამოყენებით.
2. შექმენით მართკუთხედის კლასი, რომელიც სამ კონსტრუქტორს შეიცავს. I კონსტრუქტორს 2 მთელი ტიპის პარამეტრი აქვს, რომლებსაც ამავე კლასის ცვლადებს ანიჭებს. II კონსტრუქტორს პარამეტრები არ აქვს, იძახებს I კონსტრუქტორს და მას 0-ებს გადასცემს. III კონსტრუქტორი, რომლის პარამეტრია ამავე კლასის ობიექტი, იძახებს I კონსტრუქტორს და მას გადასცემს ობიექტის ორივე ცვლადის მნიშვნელობას. II და III კონსტრუქტორებში სრულდება მართკუთხედის ფართობის გამოთვლა. ძირითად პროგრამაში შექმენით სამი ობიექტი კონსტრუქტორების გამოყენებით.

პრაქტიკული სამუშაო #2

კონსტრუქტორი. this სიტყვა. სახელების სივრცე

კონსტრუქტორი

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

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

int SamkutxedisPerimetri1 = Sam1.Perim();
int SamkutxedisPerimetri2 = Sam2.Perim();
label1.Text = "პირველი სამკუთხედის პერიმეტრი = " +
SamkutxedisPerimetri1.ToString();
label2.Text = " მეორე სამკუთხედის პერიმეტრი = " +
```

```
        SamkutxedisPerimetri2.ToString();
}
```

this სიტყვა

მეთოდს გამოძახებისას ავტომატურად გადაეცემა არაცხადი არგუმენტი, რომელიც წარმოადგენს გამომძახებელ ობიექტზე მიმართვას. ამ მიმართვას ეწოდება this. C# ენის სინტაქსი იძლევა ერთნაირი სახელების გამოყენების საშუალებას პარამეტრებისა და ლოკალური ცვლადებისათვის. ასეთ შემთხვევაში, პარამეტრი მალავს ლოკალურ ცვლადს და მასთან მიმართვა შესაძლებელია მხოლოდ this მიმართვის გამოყენებით. ქვემოთ მოყვანილ პროგრამაში ხდება დამალულ ცვლადთან მიმართვა this სიტყვის გამოყენებით.

```
//      პროგრამაში ხდება დამალულ ცვლადთან მიმართვის დემონსტრირება
class Axarisxeba
{
public double ricxvi;
public int xarisxi;
public double shedegi;
public Axarisxeba(double ricxvi, int xarisxi)
{
this.ricxvi = ricxvi;                   //      ობიექტის this.ricxvi ცვლადს ენიჭება
                                        //      ricxvi პარამეტრი
this.xarisxi = xarisxi;                 //      ობიექტის this.xarisxi ცვლადს ენიჭება
                                        //      xarisxi პარამეტრი
shedegi = 1;

if (this.xarisxi == 0 ) return;
for ( ; this.xarisxi > 0; this.xarisxi-- ) shedegi *= this.ricxvi;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
double ricxvi1 = double.Parse(textBox1.Text);
int   xarisxi1 = int.Parse(textBox2.Text);
Axarisxeba obieqti = new Axarisxeba(ricxvi1, xarisxi1);
label1.Text = obieqti.shedegi.ToString();
}
```

სახელების სივრცე

სახელების სივრცე თავიდან გვაცილებს პროგრამის კლასების სახელებს შორის კონფლიქტს. დავუშვათ, რომ ჩვენი პროგრამა იყენებს სხვა პროგრამისტის მიერ შემუშავებულ კლასს, რომლის სახელია Manqana. დავუშვათ, ასევე, რომ ჩვენც დაგვჭირდა ამავე სახელის მქონე კლასის განსაზღვრა. ასეთ შემთხვევაში, მოგვიწევს სხვა სახელის გამოყენება. სწორედ ასეთი პრობლემების გადასაწყვეტად გამოიყენება სახელების სივრცე.

სახელების სივრცე განსაზღვრავს გამოცხადების უბანს, რომელიც საშუალებას გვაძლევს შევინახოთ სახელების თითოეული ნაკრები სხვა ნაკრებებისაგან ცალკე. სახელების ერთ სივრცეში გამოცხადებული სახელები არ არიან კონფლიქტში სახელების სხვა სივრცეში გამოცხადებულ ასეთივე სახელებთან. System სახელების სივრცეში მრავალი კლასია განსაზღვრული, ამიტომ თითოეული პროგრამა იწყება დირექტივით: using System; სახელების სივრცის გამოცხადება ხდება namespace საკვანძო სიტყვის საშუალებით.

ქვემოთ მოყვანილ პროგრამაში ხდება სახელების სივრცესთან მუშაობის დემონსტრირება. Bmw და Opel სახელების სივრცე უნდა მივუთითოთ ცალკე ფაილში, როგორც ამას კლასებისთვის ვაკეთებთ. ამისათვის, ჯერ შევასრულოთ Project მენიუს Add New Item ბრძანება. შემდეგ, მოვნიშნოთ Visual C# Items განყოფილების Code File ელემენტი, Name ველში შევიტანოთ ამ ფაილის სახელი, მაგალითად Sicrceebi.cs, დავაჭიროთ Add კლავიშს და დავიწყოთ სახელების სივრცის შეტანა.

```
//      პროგრამაში ხდება სახელების სივრცესთან მუშაობის დემონსტრირება
//      Bmw სახელების სივრცის გამოცხადება
namespace Bmw
{
public class Manqana
{
public string ManqanisMarka;
}
}
//      Opel სახელების სივრცის გამოცხადება
namespace Opel
{
public class Manqana
{
public string ManqanisMarka;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
Bmw.Manqana ChemiManqana1 = new Bmw.Manqana();
ChemiManqana1.ManqanisMarka = "BMW";
label1.Text = ChemiManqana1.ManqanisMarka;

Opel.Manqana ChemiManqana2 = new Opel.Manqana();
ChemiManqana2.ManqanisMarka = "Opel";
label2.Text = ChemiManqana2.ManqanisMarka;
}
```

პროგრამაში განსაზღვრულია ორი კლასი, რომლებსაც ერთნაირი სახელები აქვთ - Manqana. იმისათვის, რომ ეს ორი კლასი ერთმანეთისაგან განვასხვავოთ საჭიროა კლასის სახელის წინ სახელების სივრცის მითითება. მაგალითად, მოყვანილ სტრიქონში ხდება იმ Manqana კლასის ობიექტის შექმნა, რომელიც BMW სახელების სივრცეშია გამოცხადებული:

```
Bmw.Manqana ChemiManqana1 = new Bmw.Manqana();
```

ობიექტის შექმნის შემდეგ არ არის აუცილებელი სახელების სივრცის მითითება ობიექტის სახელის ან მისი წევრის წინ, მაგალითად,

```
ChemiManqana1.ManqanisMarka = "BMW";
```

using დირექტივა

ძალზე დამღლელია ყოველთვის მივუთითოთ სახელების სივრცე, როცა პროგრამაში ხდება ხშირი მიმართვები ამ სახელების სივრცის წევრებთან. ასეთ შემთხვევებში, სასურველია using დირექტივის გამოყენება. ჩვენს მიერ განხილული ყველა პროგრამისათვის System სახელების სივრცე ხილული ხდება using დირექტივაში მისი მითითების შემდეგ. using დირექტივა შეგვიძლია, აგრეთვე, გამოვიყენოთ იმისათვის, რომ ხილული გავხადოთ ჩვენს მიერ შექმნილი სახელების სივრცე.

ყველა წევრი, განსაზღვრული მითითებული სახელების სივრცის ფარგლებში, ხდება მისაწვდომი, ე.ი. ხდება მიმდინარე სახელების სივრცის ნაწილი, და შეიძლება გამოყენებული იყოს ადგილმდებარეობის სრული მითითების გარეშე. using დირექტივა მითითებული უნდა იყოს პროგრამის საწყისი კოდის შემცველი ფაილის დასაწყისში, ყველა სხვა გამოცხადების წინ.

ქვემოთ მოყვანილია პროგრამა, რომელშიც using დირექტივა გამოიყენება იმისთვის, რომ System და Opel სახელების სივრცე გავხადოთ ხილული.

```
//      პროგრამაში ხდება using დირექტივასთან მუშაობის დემონსტრირება
using System;
using  Opel;
//      Opel სახელების სივრცის გამოცხადება
namespace Opel
{
public class Manqana
{
public string ManqanisMarka;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      Opel სიტყვის მითითება არ არის აუცილებელი ChemiManqana2 ობიექტის
//      შექმნისას
Manqana ChemiManqana = new Manqana();
ChemiManqana.ManqanisMarka = "Opel";
label1.Text = ChemiManqana.ManqanisMarka;
}
```

ყურადღება მივაქციოთ იმას, რომ პროგრამაში ერთი სახელების სივრცე არ მალავს მეორე სახელების სივრცეს. სახელების სივრცის გამოცხადებისას ის თავის სახელებს უმატებს მოცემულ მომენტში სხვა მისაწვდომ სახელებს. ამრიგად, ორივე სახელების სივრცე System და Opel, ხდება ხილული ამ პროგრამის კოდისათვის. გარდა ამისა, თუ სახელების ორ სივრცეში გამოცხადებულია ერთი და იგივე სახელის მქონე კლასები, მაშინ ამ კლასებთან მიმართვისას აუცილებლად უნდა მივუთითოთ სახელების სივრცის სახელი. წინააღმდეგ შემთხვევაში, კომპილატორი ვერ გაარკვევს თუ რომელ კლასთან

უნდა შესრულდეს მიმართვა და გაიცემა შეტყობინება შეცდომის შესახებ.

ჩადგმული სახელების სივრცე

ერთი სახელების სივრცე შეიძლება მოვათავსოთ მეორის შიგნით. ასეთი გზით შეგვიძლია შევქმნათ სახელების სივრცის იერარქია. სახელების სივრცის იერარქია შეიძლება გადანაწილებული იყოს სხვადასხვა პროგრამაში (სხვადასხვა ფაილში). ასეთი გადანაწილების უპირატესობა იმაშია, რომ პროგრამისტებს შეუძლიათ ცალცალკე იმუშაონ ამ პროგრამებზე და შემდეგ ერთდროულად შეასრულონ მათი კომპილაცია.

ქვემოთ მოყვანილია ორი პროგრამა, რომლებშიც გადანაწილებულია Saxelebis_Sivrce სახელების სივრცე.

```
//      პროგრამაში ხდება ჩადგმულ სახელების სივრცესთან მუშაობის დემონსტრირება
        namespace Saxelebis_Sivrce
        {
        //      Chadgmuli_Sivrce1 სახელების სივრცე არის ჩადგმული
        namespace Chadgmuli_Sivrce1
        {
        public class ChemiKlasi
        {
        public string Metodi()
        {
        return " Chadgmuli_Sivrce1 ";
        }
        }
        }
        }
        namespace Saxelebis_Sivrce.Chadgmuli_Sivrce2
        {
        public class ChemiKlasi
        {
        public string Metodi()
        {
        return " Chadgmuli_Sivrce2 ";
        }
        }
        }
        private void button1_Click(object sender, System.EventArgs e)
        {
        Saxelebis_Sivrce.Chadgmuli_Sivrce1.ChemiKlasi obieqti1 =
                                new Saxelebis_Sivrce. Chadgmuli_Sivrce1.ChemiKlasi();
        Saxelebis_Sivrce.Chadgmuli_Sivrce2.ChemiKlasi obieqti2 =
                                        new Saxelebis_Sivrce.Chadgmuli_Sivrce2.ChemiKlasi();
        label1.Text = obieqti1.Metodi();
        label2.Text = obieqti2.Metodi();
        }
```

პროგრამაში, Saxelebis_Sivrce სახელების სივრცეში ჩადგმულია Chadgmuli_Sivrce1

და Chadgmuli_Sivrce2 სახელების სივრცეები. სახელების ჩადგმული სივრცე შეგვიძლია, აგრეთვე გამოვაცხადოთ ერთ სტრიქონში. ასეა გამოცხადებული Chadgmuli_Sivrce2 სახელების სივრცე.

ChemiKlasi კლასთან მიმართვისას სრულად უნდა მივუთითოთ მისი ადგილმდებარეობა. ამისათვის, ჯერ უნდა მივუთითოთ ზედა დონის სახელების სივრცის სახელი, შემდეგ წერტილი და ბოლოს, ქვედა დონის სახელების სივრცის სახელი, მაგალითად, Saxelebis_Sivrce. Chadgmuli_Sivrce1.ChemiKlasi.

თუ გვინდა, რომ Chadgmuli_Sivrce1 სახელების სივრცეში გამოცხადებული ChemiKlasi კლასი მისაწვდომი გახდეს მისი ადგილმდებარეობის სრული მითითების გარეშე, using დირექტივა უნდა ჩავწეროთ შემდეგნაირად: using Saxelebis_Sivrce.Chadgmuli_Sivrce1; თუ using დირექტივას ასე ჩავწერთ: using Saxelebis_Sivrce; , მაშინ Chadgmuli_Sivrce1 სახელების სივრცეში გამოცხადებული ChemiKlasi კლასი არ იქნება მისაწვდომი მისი ადგილმდებარეობის სრული მითითების გარეშე.

სავარჯიშოები

კონსტრუქტორი

1. შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ბაკის ტევადობა და მანძილი, რომელსაც გაიფრენს თვითმფრინავი 1 ლიტრი საწვავით; ღია მეთოდებს: პირველი მეთოდია კონსტრუქტორი, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები; მესამე მეთოდი გამოთვლის და აბრუნებს თვითმფრინავის მიერ სავსე ბაკით გავლილ მანძილს. მთავარ პროგრამაში კონსტრუქტორის გამოყენებით შექმენით ობიექტი და გამოიძახეთ მეორე და მესამე მეთოდები. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, მონაცემების გამოსატანად კი - Label ვიზუალური ელემენტი.
2. შექმენით სამკუთხედის კლასი, რომელიც შეიცავს ღია ცვლადებს: სამკუთხედის სამივე გვერდის ზომებს; პრივატულ ცვლადებს: სამკუთხედის პერიმეტრსა და ფართობს; ღია მეთოდებს: პირველი მეთოდია კონსტრუქტორი, რომელიც ღია და პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები. მთავარ პროგრამაში კონსტრუქტორის გამოყენებით შექმენით ობიექტი და გამოიძახეთ მეორე მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, მონაცემების გამოსატანად კი - Label ვიზუალური ელემენტი.
3. შექმენით ავტომანქანის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: მანქანის ფერი და კარებების რაოდენობა; ღია ცვლადებს: მფლობელის გვარი და გამომშვები ფირმა; ღია მეთოდებს: პირველი მეთოდია კონსტრუქტორი, რომელიც ღიას და პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები. მთავარ პროგრამაში კონსტრუქტორის გამოყენებით შექმენით ობიექტი და გამოიძახეთ მეორე მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, მონაცემების გამოსატანად კი - Label ვიზუალური ელემენტი.

this სიტყვა

1. შექმენით კლასი, რომლის ცვლადებსა და მეთოდის პარამეტრებს ერთნაირი სახელები აქვს. პარამეტრების მნიშვნელობები მიანიჭეთ კლასის ამავე სახელის მქონე ცვლადებს. მეთოდი აბრუნებს კლასის ცვლადების ჯამს. მთავარ პროგრამაში შექმენით ობიექტი ნაგულისხმევი კონსტრუქტორის გამოყენებით და გამოიძახეთ მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, მონაცემების გამოსატანად კი - Label ვიზუალური ელემენტი.
2. შექმენით კლასი, რომლის ცვლადებსა და მეთოდის პარამეტრებს ერთნაირი სახელები აქვს. პარამეტრების მნიშვნელობები მიანიჭეთ კლასის ამავე სახელის მქონე ცვლადებს. მეთოდი აბრუნებს კლასის ცვლადების ნამრავლს. მთავარ პროგრამაში შექმენით ობიექტი ნაგულისხმევი კონსტრუქტორის გამოყენებით და გამოიძახეთ მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, მონაცემების გამოსატანად კი - Label ვიზუალური ელემენტი.
3. შექმენით კლასი, რომელშიც განსაზღვრულია წილადი ტიპის ორგანზომილებიან მასივზე მიმართვა და მეთოდი, რომლის პარამეტრია წილადი ტიპის ორგანზომილებიანი მასივი. მასივს და პარამეტრს ერთნაირი სახელები აქვს.

პარამეტრის მნიშვნელობა მიანიჭეთ კლასის ამავე სახელის მქონე მასივს. მეთოდი აბრუნებს მასივის პირველ უარყოფით ელემენტს. მთავარ პროგრამაში შექმენით ობიექტი ნაგულისხმევი კონსტრუქტორის გამოყენებით და გამოიძახეთ მეთოდი. მონაცემების გამოსატანად გამოიყენეთ Label ვიზუალური ელემენტი.

4. შექმენით კლასი, რომელშიც განსაზღვრულ მასივსა და მეთოდის პარამეტრს ერთნაირი სახელები აქვს. პარამეტრის მნიშვნელობა მიანიჭეთ კლასის ამავე სახელის მქონე მასივს. მეთოდი აბრუნებს მასივის ელემენტებს შორის მინიმალურს. მთავარ პროგრამაში შექმენით ობიექტი ნაგულისხმევი კონსტრუქტორის გამოყენებით და გამოიძახეთ მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, მონაცემების გამოსატანად კი - Label ვიზუალური ელემენტი.

სახელების სივრცე

1. შექმენით სახელების 2 სივრცე - Sivrce_1 და Sivrce_2 რომლებიც შეიცავენ Samkutxedi კლასს. Sivrce_1 სივრცის Samkutxedi კლასი შეიცავს Fartobi მეთოდს, რომელსაც ორი პარამეტრი აქვს და მართკუთხა სამკუთხედის ფართობს აბრუნებს, Sivrce_2 სივრცის Samkutxedi კლასი კი შეიცავს Perimetri მეთოდს, რომელსაც სამი პარამეტრი აქვს და სამკუთხედის პერიმეტრს აბრუნებს. მთავარ პროგრამაში შექმენით ორი ობიექტი თითოეულ სივრცეში მოთავსებული კლასის გამოყენებით. გამოიძახეთ Fartobi და Perimetri მეთოდები. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, მონაცემების გამოსატანად კი - Label ვიზუალური ელემენტი.
2. შექმენით სახელების 2 სივრცე - Sivrce_1 და Sivrce_2 რომლებიც შეიცავენ Martkutxedi კლასს. Sivrce_1 სივრცის Martkutxedi კლასი შეიცავს Fartobi მეთოდს, რომელსაც ორი პარამეტრი აქვს და მართხკუთხედის ფართობს აბრუნებს, Sivrce_2 სივრცის Martkutxedi კლასი კი შეიცავს Perimetri მეთოდს, რომელსაც ორი პარამეტრი აქვს და მართხკუთხედის პერიმეტრს აბრუნებს. მთავარ პროგრამაში შექმენით ორი ობიექტი თითოეულ სივრცეში მოთავსებული კლასის გამოყენებით. გამოიძახეთ Fartobi და Perimetri მეთოდები. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, მონაცემების გამოსატანად კი - Label ვიზუალური ელემენტი.
3. შექმენით სახელების 2 სივრცე - Sivrce_1 და Sivrce_2 რომლებიც შეიცავენ Klasi_1 კლასს. Sivrce_1 სივრცის Klasi_1 კლასი შეიცავს Metodi1 მეთოდს, რომლის პარამეტრია ერთგანზომილებიანი მთელრიცხვა მასივი და, გამოთვლის და აბრუნებს ამ მასივის დადებითი ელემენტების ჯამს. Sivrce_2 სივრცის Klasi_1 კლასი შეიცავს Metodi2 მეთოდს, რომლის პარამეტრია ერთგანზომილებიანი მთელრიცხვა მასივი და, გამოთვლის და აბრუნებს ამ მასივის უარყოფითი ელემენტების ნამრავლს. მთავარ პროგრამაში შექმენით ორი ობიექტი თითოეულ სივრცეში მოთავსებული კლასის გამოყენებით. გამოიძახეთ Metodi1 და Metodi2 მეთოდები. მონაცემების გამოსატანად გამოიყენეთ Label ვიზუალური ელემენტი.

პრაქტიკული #8
ინტერფეისები

C# ენის ერთ-ერთი მნიშვნელოვანი ელემენტია ინტერფეისი. ის განსაზღვრავს მეთოდების, თვისებების, ინდექსატორებისა და მოვლენების ნაკრებს, რომელთა რეალიზება შესაძლებელია კლასის საშუალებით. პროგრამირების დროს ზოგჯერ წამოიჭრება კლასის მიერ შესრულებული მოქმედებების განსაზღვრის აუცილებლობა, მათი რეალიზების გზის მითითების გარეშე. C# ენაში შეიძლება მთლიანად განვაცალკევოთ კლასის ინტერფეისი ამ კლასის რეალიზებისგან. ინტერფეისის განსაზღვრის შემდეგ მისი რეალიზება შესაძლებელია ერთი ან მეტი კლასის საშუალებით. ამასთან, ასეთი კლასი დამატებით შეიძლება შეიცავდეს საკუთარ ელემენტებს.

ინტერფეისის გამოცხადების სინტაქსია:

interface ***ინტერფეისის_სახელი***
{
ინტერფეისის ტანი
}

ინტერფეისის ტანი შეიძლება შეიცავდეს მეთოდებს, თვისებებს, ინდექსატორებსა და მოვლენებს. ინტერფეისი არ შეიძლება შეიცავდეს ცვლადებს (ველებს), სტატიკურ წევრებს, კონსტრუქტორებს, დესტრუქტორებსა და ოპერატორულ მეთოდებს. ინტერფეისში შეიძლება გამოცხადებული იყოს მეთოდების, თვისებების, ინდექსატორებისა და მოვლენების სიგნატურები. მეთოდისთვის დამატებით ეთითება დასაბრუნებელი მნიშვნელობის ტიპი. გარდა ამისა, მეთოდის სიგნატურა ზუსტად უნდა შეესაბამებოდეს interface განსაზღვრაში მითითებულ სიგნატურას.

ინტერფეისების რეალიზება

როგორც კი ინტერფეისი განისაზღვრება, ის შეიძლება რეალიზებული იყოს ერთი ან მეტი კლასის მიერ. ინტერფეისის რეალიზაციისთვის კლასის სახელის შემდეგ უნდა მივუთითოთ ორი წერტილი და ინტერფეისის სახელი.

თუ კლასი ახდენს ინტერფეისის რეალიზებას, ის ვალდებულია მისი რეალიზება მოახდინოს მთლიანად.

კლასებს, რომლებიც ახდენს ინტერფეისების რეალიზებას, შეუძლია განსაზღვროს, აგრეთვე, დამატებითი წევრები. ქვემოთ მოყვანილი პროგრამით ხდება ინტერფეისის რეალიზება, რომელშიც ორი მეთოდია გამოცხადებული.

```
//      პროგრამაში რეალიზებულია ინტერფეისი, რომელშიც გამოცხადებულია მეთოდები
public interface ISamkutxedi
{
        int Perimetri();
        double Fartobi();
}
class Samkutxedi : ISamkutxedi
{
  int gverdi1;
  int gverdi2;
  int gverdi3;
  int perimetri;
  double fartobi;
  public Samkutxedi(int par1, int par2, int par3)
{
```

```
    gverdi1 = par1;
    gverdi2 = par2;
    gverdi3 = par3;
}
   public int Perimetri()
{
    perimetri = gverdi1 + gverdi2 + gverdi3;
    return perimetri;
}
   public double Fartobi()
{
    fartobi = ( gverdi1 * gverdi2 ) / 2.0;
    return fartobi;
}
}
private void button1_Click(object sender, EventArgs e)
  {
   int gverdi1 = int.Parse(textBox1.Text);
   int gverdi2 = int.Parse(textBox2.Text);
   int gverdi3 = int.Parse(textBox3.Text);
   Samkutxedi Obieqti = new Samkutxedi(gverdi1, gverdi2, gverdi3);

   int perimetri = Obieqti.Perimetri();
   double fartobi = Obieqti.Fartobi();
   label2.Text = "სამკუთხედის პერიმეტრი = " + perimetri.ToString();
   label3.Text = "სამკუთხედის ფართობი = " + fartobi.ToString();
  }
```

ერთ კლასს შეუძლია რამდენიმე ინტერფეისის რეალიზება. ამ შემთხვევაში, კლასში უნდა შესრულდეს ყველა ინტერფეისის წევრების რეალიზება. მოყვანილი პროგრამით ხდება ერთი კლასის მიერ ორი ინტერფეისის რეალიზება.

```
//          კლასი ახდენს ორი ინტერფეისის რეალიზებას
  public interface Interpeisi1
  {
   int Perimetri();
  }
  public interface Interpeisi2
  {
   double Fartobi();
  }
  class Samkutxedi : Interpeisi1, Interpeisi2
  {
   int gverdi1;
   int gverdi2;
   int gverdi3;
   int perimetri;
   double fartobi;
   public Samkutxedi(int par1, int par2, int par3)
```

```
  {
   gverdi1 = par1;
   gverdi2 = par2;
   gverdi3 = par3;
  }
  public int Perimetri()
  {
   perimetri = gverdi1 + gverdi2 + gverdi3;
   return perimetri;
  }
  public double Fartobi()
  {
   fartobi = ( gverdi1 * gverdi2 ) / 2.0;
   return fartobi;
  }
 }
 private void button1_Click(object sender, EventArgs e)
 {
  int gverdi1 = int.Parse(textBox1.Text);
  int gverdi2 = int.Parse(textBox2.Text);
  int gverdi3 = int.Parse(textBox3.Text);
  Samkutxedi obieqti = new Samkutxedi(gverdi1, gverdi2, gverdi3);
  int Perimetri = obieqti.Perimetri();
  double Fartobi = obieqti.Fartobi();
  label1.Text = "სამკუთხედის პერიმეტრია  " + Perimetri.ToString();
  label2.Text = "სამკუთხედის ფართობია  " + Fartobi.ToString();
}
```

კლასების მემკვიდრეობითობა და ინტერფეისის რეალიზება

კლასებს შეუძლიათ ერთზე მეტი ინტერფეისის რეალიზება. ამ შემთხვევაში, ინტერფეისების სახელები ერთმანეთისგან მძიმეებით უნდა გამოიყოს. კლასი შეიძლება იყოს, აგრეთვე, საბაზო კლასის მემკვიდრე და ახდენდეს ერთი ან მეტი ინტერფეისის რეალიზებას. ამ შემთხვევაში სიაში, წინაპარი კლასის სახელი უნდა იყოს პირველი. მოყვანილი პროგრამით ხდება ყოველივე ამის დემონსტრირება.

```
//      პროგრამაში ხდება ინტერფეისის რეალიზება მემკვიდრეობითობის შემთხვევაში
//      Interpeisi_A ინტერფეისის გამოცხადება
//      Interpeisi_A ინტერფეისის გამოცხადება
public interface Interpeisi_A
 {
  //      მეთოდების გამოცხადება
  int Perimetri();
 }
 //      Interpeisi_B ინტერფეისის გამოცხადება
 public interface Interpeisi_B
 {
  //      მეთოდების გამოცხადება
  double Fartobi();
```

```
 }
 public class Sabazo
 {
  public int gverdi1;
  public int gverdi2;
  public int gverdi3;
  protected int perimetri;
  protected double fartobi;
  public Sabazo(int par1, int par2, int par3)
  {
   gverdi1 = par1;
   gverdi2 = par2;
   gverdi3 = par3;
  }
 }
 public class Samkutxedi : Sabazo, Interpeisi_A, Interpeisi_B
 {
  //      საბაზო კლასის კონსტრუქტორის გამოძახება
  public Samkutxedi(int par4, int par5, int par6) : base(par4, par5, par6)
  {
   //      ცარიელი ტანი
  }
  //      Interpeisi_A ინტერფეისის Perimetri() მეთოდის რეალიზება
  public int Perimetri()
  {
   perimetri = gverdi1 + gverdi2 + gverdi3;
   return perimetri;
  }
  //      Interpeisi_B ინტერფეისის Fartobi() მეთოდის რეალიზება
  public double Fartobi()
  {
   fartobi = ( gverdi1 * gverdi2 ) / 2.0;
   return fartobi;
  }
 }
 private void button1_Click(object sender, EventArgs e)
 {
  int gverdi1 = int.Parse(textBox1.Text);
  int gverdi2 = int.Parse(textBox2.Text);
  int gverdi3 = int.Parse(textBox3.Text);
  Samkutxedi obj = new Samkutxedi(gverdi1, gverdi2, gverdi3);
  int perimetri = obj.Perimetri();
  double fartobi = obj.Fartobi();
  label1.Text = "სამკუთხედის პერიმეტრი = " + perimetri.ToString();
  label2.Text = "სამკუთხედის ფართობი = " + fartobi.ToString();
}
```

წარმოებული ინტერფეისები

კლასების მსგავსად, ინტერფეისებიც შეიძლება იყოს წარმოებული (მიღებული) მემკვიდრეობით. კლასებისგან განსხვავებით, ინტერფეისი შეიძლება მიღებული იყოს ერთზე მეტი ინტერფეისისგან. განვიხილოთ ორი მაგალითი. პირველ მაგალითში Interfeisi_B ინტერფეისი მიღებულია Interfeisi_A ინტერფეისისგან, მეორე მაგალითში კი Interfeisi_C ინტერფეისი მიღებულია Interfeisi_A და Interfeisi_B ინტერფეისებისგან.

```
//          პროგრამაში Interfeisi_B ინტერფეისი მიიღება Interfeisi_A ინტერფეისისგან
public interface Interfeisi_A
      {
         int Perimetri();
      }
      public interface Interfeisi_B : Interfeisi_A
      {
         double Fartobi();
      }
      class Samkutxedi : Interfeisi_B
      {
       private int gverdi1;
       private int gverdi2;
       private int gverdi3;
       private int perimetri;
       private double fartobi;
       public Samkutxedi(int par1, int par2, int par3)
       {
        gverdi1 = par1;
        gverdi2 = par2;
        gverdi3 = par3;
       }
       public int Perimetri()
       {
        perimetri = gverdi1 + gverdi2 + gverdi3;
        return perimetri;
       }
       public double Fartobi()
       {
        fartobi = ( gverdi1 + gverdi2 ) / 2.0;
        return fartobi;
       }
      }
 private void button1_Click(object sender, EventArgs e)
 {
         int gverdi1 = int.Parse(textBox1.Text);
         int gverdi2 = int.Parse(textBox2.Text);
         int gverdi3 = int.Parse(textBox3.Text);
         Samkutxedi obieqti = new Samkutxedi(gverdi1, gverdi2, gverdi3);
         int perimetri = obieqti.Perimetri();
         double fartobi = obieqti.Fartobi();
```

```
            label1.Text = perimetri.ToString();
            label2.Text = fartobi.ToString();
}
```

მეორე მაგალითი.

```
//          პროგრამაში Interfeisi_C ინტერფეისი მიიღება Interfeisi_A და Interfeisi_B
//          ინტერფეისებისგან
public interface Interfeisi_A
  {
   int Perimetri();
  }
  public interface Interfeisi_B
  {
   double Fartobi();
  }
  public interface Interfeisi_C : Interfeisi_A, Interfeisi_B
  {
   void Naxva(Label lab1);
  }
  class Samkutxedi : Interfeisi_C
  {
   private int gverdi1;
   private int gverdi2;
   private int gverdi3;
   private int perimetri;
   private double fartobi;
   public Samkutxedi(int par1, int par2, int par3)
   {
    gverdi1 = par1;
    gverdi2 = par2;
    gverdi3 = par3;
   }
   public int Perimetri()
   {
    perimetri = gverdi1 + gverdi2 + gverdi3;
    return perimetri;
   }
   public double Fartobi()
   {
    fartobi = ( gverdi1 * gverdi2 ) / 2.0;
    return fartobi;
   }
   public void Naxva(Label lab1)
   {
    lab1.Text = gverdi1.ToString() + "  " + gverdi2.ToString() + "  " + gverdi3.ToString();
   }
```

```
 }
 private void button1_Click(object sender, EventArgs e)
 {
  int gverdi1 = int.Parse(textBox1.Text);
  int gverdi2 = int.Parse(textBox2.Text);
  int gverdi3 = int.Parse(textBox3.Text);
  Samkutxedi obieqti = new Samkutxedi(gverdi1, gverdi2, gverdi3);
  obieqti.Naxva(label3);
  int shedegi1 = obieqti.Perimetri();
  label1.Text = shedegi1.ToString();
  double shedegi2 = obieqti.Fartobi();
  label2.Text = shedegi2.ToString();
}
```

ინტერფეისული მიმართვები

ჩვენ შეგვიძლია გამოვაცხადოთ მიმართვითი ცვლადი, რომელსაც აქვს ინტერფეისული ტიპი, ანუ შეგვიძლია შევქმნათ ინტერფეისული მიმართვითი ცვლადი. ასეთი ცვლადი შეიძლება მიმართავდეს ნებისმიერ ობიექტს, რომელიც ახდენს მისი ინტერფეისის რეალიზებას. ინტერფეისული მიმართვის საშუალებით ობიექტის მეთოდის გამოძახებისას გამოიძახება ის მეთოდი, რომლის რეალიზებას მოცემული ობიექტი ახდენს.

მოყვანილ პროგრამაში ნაჩვენებია ინტერფეისული მიმართვის გამოყენება. აქ გამოცხადებულია obieqti ინტერფეისული ცვლადი Samkutxedi და Otxkutxedi ობიექტების მეთოდების გამოსაძახებლად.

```
 public interface ChemiInterpeisi
 {
  int Perimetri();
  double Fartobi();
 }
 class Samkutxedi : ChemiInterpeisi
 {
  int gverdi1;
  int gverdi2;
  int gverdi3;
  public Samkutxedi(int par1, int par2, int par3)
  {
   gverdi1 = par1;
   gverdi2 = par2;
   gverdi3 = par3;
  }
  public int Perimetri()
  {
   return gverdi1 + gverdi2 + gverdi3;
  }
  public double Fartobi()
  {
   return ( gverdi1 * gverdi2 ) / 2;
  }
```

```
 }
 class Otxkutxedi : ChemiInterpeisi
 {
  int gverdi1;
  int gverdi2;
  public Otxkutxedi(int par1, int par2)
  {
   gverdi1 = par1;
   gverdi2 = par2;
  }
  public int Perimetri()
  {
   return ( gverdi1 + gverdi2 ) * 2;
  }
  public double Fartobi()
  {
   return gverdi1 * gverdi2;
  }
 }
 private void button1_Click(object sender, EventArgs e)
 {
  int gverdi1 = int.Parse(textBox1.Text);
  int gverdi2 = int.Parse(textBox2.Text);
  int gverdi3 = int.Parse(textBox3.Text);
  Samkutxedi Samkutxedi = new Samkutxedi(gverdi1, gverdi2, gverdi3);
  Otxkutxedi Otxkutxedi = new Otxkutxedi(gverdi1, gverdi2);
  ChemiInterpeisi obieqti;

  obieqti = Samkutxedi;
  label1.Text = "სამკუთხედის პერიმეტრი = " + obieqti.Perimetri().ToString() +
                "\nსამკუთხედის ფართობი = " + obieqti.Fartobi();

  obieqti = Otxkutxedi;
  label2.Text = "ოთხკუთხედის პერიმეტრი = " + obieqti.Perimetri().ToString() +
                "\nოთხკუთხედის ფართობი = " + obieqti.Fartobi();
}
```

პროგრამაში obieqti ობიექტი გამოცხადებულია როგორც მიმართვა ChemiInterpeisi ინტერფეისზე. ეს იმას ნიშნავს, რომ ის შეიძლება გამოყენებულ იქნას ნებისმიერ იმ ობიექტზე მიმართვების შენახვის მიზნით, რომელიც ახდენს ChemiInterpeisi ინტერფეისის რეალიზებას. მოცემულ შემთხვევაში ის გამოიყენება Samkutxedi და Otxkutxedi ობიექტებზე მიმართვის შენახვისთვის, რომლებიც წარმოადგენენ Samkutxedi და Otxkutxedi ტიპის ობიექტებს შესაბამისად. ორივე ეს ობიექტი ახდენს ChemiInterpeisi ინტერფეისის რეალიზებას. obieqti ინტერფეისულმა მიმართვამ იცის მხოლოდ იმ მეთოდების შესახებ, რომლებიც გამოცხადებულია ინტერფეისში.

ინტერფეისული თვისებები

ისევე, როგორც მეთოდების შემთხვევაში, ინტერფეისში თვისებების განსაზღვრისას არ

ეთითება ტანი. ინტერფეისის თვისების სინტაქსია:

ტიპი თვისების_სახელი

{

get;

set;

}

ქვემოთ მოყვანილია ChemiInterpeisi ინტერფეისი, აგრეთვე, ChemiKlasi კლასის კოდი, რომელიც თვისებას იყენებს ნაკრების მომდევნო ელემენტის გაცემისას და შეცვლისთვის.

```
//      პროგრამაში ხდება ინტერფეისის თვისებასთან მუშაობის დემონსტრირება
public interface ChemiInterpeisi
{
//      ინტერფეისის თვისება
int shemdegi
{
get;                    //      ნაკრების მომდევნო რიცხვის მიღება
set;                    //      მომდევნო რიცხვის დაყენება
}
}
class ChemiKlasi : ChemiInterpeisi
{
int cvladi;
public ChemiKlasi()
{
cvladi = 0;
}
//      მნიშვნელობის მიღება და დაყენება
public int shemdegi
{
get
{
cvladi += 5;
return cvladi;
}
set
{
cvladi = value;
}
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      ინტერფეისის თვისებების დემონსტრირება
ChemiKlasi obieqti = new ChemiKlasi();

for ( int i = 0; i < 5; i++ )
        label1.Text += "მომდევნო მნიშვნელობა - " +
                        obieqti.shemdegi.ToString() + '\n';
```

```
obieqti .shemdegi = 30;
for ( int i = 0; i < 5; i++ )
        label2.Text += " მომდევნო მნიშვნელობა - " + obieqti.shemdegi.ToString() + '\n';
}
```

ცხადი რეალიზება

ცხადი რეალიზება არის შემთხვევა, როცა რეალიზაციის დროს ინტერფეისის ელემენტის წინ ეთითება ინტერფეისის სახელი. კლასს შეუძლია ორი ინტერფეისის რეალიზება, რომლებშიც შეიძლება გამოცხადებული იყოს ერთნაირი სახელის ელემენტები. წამოიჭრება პრობლემა - როგორ განვასხვავოთ ასეთი ელემენტები. ეს პრობლემა წყდება ცხადი რეალიზაციის გამოყენებით.

ქვემოთ მოყვანილ პროგრამაში გამოცხადებულია ორი ინტერფეისი, რომლებიც შეიცავენ metodi() მეთოდს. იმისთვის, რომ განვასხვავოთ ერთნაირი სახელის მქონე ეს მეთოდი უნდა გამოვიყენოთ ცხადი რეალიზება:

```
//      პროგრამაში ხდება ცხადი რეალიზების გამოყენების დემონსტრირება
interface Interpeisi_A
{
int Metodi(int x);
}
interface Interpeisi_B
{
int Metodi(int x);
}
//      ChemiKlasi კლასში ხდება ორივე ინტერფეისის რეალიზება
class ChemiKlasi : Interpeisi_A, Interpeisi_B
{
Interpeisi_A a_obieqti;
Interpeisi_B b_obieqti;
//      ორივე metodi() მეთოდის აშკარა რეალიზება
int Interpeisi_A.metodi(int x)
{
return x + x;
}
int Interpeisi_B.metodi(int x)
{
return x * x;
}
//      metodi() მეთოდის გამოძახება ინტერფეისული მიმართვის საშუალებით
public int Metodi_A(int x)
{
a_obieqti = this;                   //      a_obieqti ობიექტს ენიჭება გამომძახებელ ობიექტზე მიმართვა
return a_obieqti.Metodi(x);         //        Interpeisi_A მეთოდის გამოძახება
}
public int Metodi_B(int x)
{
b_obieqti = this;                   //      b_obieqti ობიექტს ენიჭება გამომძახებელ ობიექტზე მიმართვა
return b_obieqti.Metodi(x);         //        Interpeisi_B მეთოდის გამოძახება
```

```
}
}
private void button1_Click(object sender, System.EventArgs e)
{
//      აშკარა რეალიზაციის გამოყენება არაერთგვაროვნობის აღმოსაფხვრელად
ChemiKlasi obieqti = new ChemiKlasi();
int ricxvi = int.Parse(textBox1.Text);

//      Interpeisi_A.Metodi() მეთოდის გამოძახება
label1.Text = obieqti.Metodi_A(ricxvi).ToString();
//      Interpeisi_B.Metodi() მეთოდის გამოძახება
label2.Text = obieqti.Metodi_B(ricxvi).ToString();
}
```

ყურადღება მივაქციოთ იმას, რომ Metodi() მეთოდს აქვს ერთი და იგივე სიგნატურა Interpeisi_A და Interpeisi_B ინტერფეისებში. ამიტომ, იმ შემთხვევაში თუ ChemiKlasi ახდენს ორივე ამ ინტერფეისის რეალიზებას, მაშინ მან აშკარად უნდა მოახდინოს თითოეული მათგანის ცალ-ცალკე რეალიზება, მათი სახელების სრული განსაზღვრით. რადგან ერთადერთი საშუალება, რომლის საშუალებითაც შეიძლება გამოვიძახოთ აშკარად რეალიზებული მეთოდი არის ინტერფეისული მიმართვა, ამიტომ ChemiKlasi კლასში გამოცხადებულია ორი მიმართვა, რომელთაგან ერთი განკუთვნილია Interpeisi_A, მეორე კი - Interpeisi_B ინტერფეისისთვის. შემდეგ სრულდება კლასის ორი მეთოდი გამოიძახება ინტერფეისის მეთოდების საშუალებით. რადგან, Metodi() მეთოდი ცხადად არის რეალიზებული, ამიტომ ის მიუწვდომელია ChemiKlasi კლასის გარეთ. ამ კლასის შიგნით Metodi() მეთოდთან მიმართვა შესაძლებელია მხოლოდ ინტერფეისული a_obieqti და b_obieqti ობიექტების საშუალებით.

ინტერფეისის წევრების დამალვა

კლასების ანალოგიურად, მემკვიდრეობითობის დროს, ინტერფეისების შემთხვევაშიც ადგილი აქვს წევრების დამალვას. დავუშვათ, Interfeisi_A ინტერფეისი არის საბაზო და შეიცავს Minicheba() მეთოდს, ხოლო Interfeisi_B ინტერფეისი არის მისგან მიღებული და შეიცავს Minicheba() მეთოდს. თუ გამოვაცხადებთ Klasi1 კლასს, რომელიც მოახდენს Interfeisi_B ინტერფეისის რეალიზებას, მაშინ ამ კლასში ერთ-ერთი Minicheba() მეთოდი უნდა გამოვაცხადოთ Interfeisi_A ან Interfeisi_B ინტერფეისის სახელის ცხადი მითითების გზით. მოყვანილი პროგრამით ხდება ამის დემონსტრირება.

```
//      პროგრამაში ხდება ინტერფეისის წევრების დამალვის დემონსტრირება
public interface Interfeisi_A
{
int Minicheba(int x);
}
public interface Interfeisi_B : Interfeisi_A
{
new int Minicheba(int x);
}
class Klasi1 : Interfeisi_B
{
private int ricxvi;
public int Minicheba(int par1)
{
```

```
ricxvi = par1;
return ricxvi * ricxvi;
}
int Interfeisi_B.Minicheba(int par1)
{
ricxvi = par1;
return ricxvi + ricxvi;
}
}
private void button1_Click(object sender, EventArgs e)
{
int cvladi = int.Parse(textBox1.Text);
Klasi1 obieqti = new Klasi1();
Interfeisi_B interfeisi_obj = ( Interfeisi_B ) obieqti;

int shedegi1 = obieqti.Minicheba(cvladi);
label1.Text = shedegi1.ToString();
int shedegi2 = interfeisi_obj.Minicheba(cvladi);
label2.Text = shedegi2.ToString();
}
```

სავარჯიშოები

ინტერფეისები. რეალიზება

1.1. შექმენით ინტერფეისი, რომელიც ორ მეთოდს შეიცავს. I მეთოდი გამოთვლის და აბრუნებს პარამეტრის კვადრატს, II კი - პარამეტრის კუბს. მოახდინეთ ამ ინტერფეისის რეალიზება.

1.2. შექმენით ინტერფეისი, რომელიც ერთ თვისებას შეიცავს. თვისება პრივატულ ცვლადს ლუწ მნიშვნელობებს ანიჭებს. მოახდინეთ ამ ინტერფეისის რეალიზება.

1.3. შექმენით ინტერფეისი, რომელიც ერთ ინდექსატორს შეიცავს. ინდექსატორი 5 მთელ რიცხვთან მუშაობს, ანიჭებს მათ კენტ მნიშვნელობებს და გასცემს მათ მნიშვნელობებს. მოახდინეთ ამ ინტერფეისის რეალიზება.

1.4. შექმენით ინტერფეისი, რომელიც ერთ მოვლენას შეიცავს. მოვლენა აღიძვრება მაშინ, როცა ერთგანზომილებიანი მასივის მაქსიმალური ელემენტი 25-ს გადააჭარბებს. მოახდინეთ ამ ინტერფეისის რეალიზება.

1.5. შექმენით ინტერფეისი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი გამოთვლის და აბრუნებს პარამეტრის კუბს; შეიცავს ერთ თვისებას, რომელიც პრივატულ ცვლადს დადებით მნიშვნელობებს ანიჭებს; შეიცავს ერთ ინდექსატორს, რომელიც 4 მთელ რიცხვთან მუშაობს და ანიჭებს მათ 7-ის ჯერად მნიშვნელობებს; შეიცავს ერთ მოვლენას, რომელიც აღიძვრება მაშინ, როცა ერთგანზომილებიანი მასივის მინიმალური ელემენტი იქნება 10-ზე ნაკლები.

ინტერფეისები. მემკვიდრეობითობა

2.1. შექმენით საბაზო კლასი. შექმენით ინტერფეისი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი გამოთვლის სამკუთხედის პერიმეტრს. მოახდინეთ ამ ინტერფეისის რეალიზება იმ კლასში, რომელიც საბაზო კლასის მემკვიდრეა. საბაზო კლასი შეიცავს მეთოდს, რომელიც ახდენს სამკუთხედის ფართობის გამოთვლას.

2.2. შექმენით საბაზო კლასი. შექმენით ინტერფეისი, რომელიც ერთ მეთოდს შეიცავს. მეთოდი პოულობს მასივის მინიმალურ ელემენტს. მოახდინეთ ამ ინტერფეისის რეალიზება იმ კლასში, რომელიც საბაზო კლასის მემკვიდრეა. საბაზო კლასი შეიცავს მეთოდს, რომელიც მეთოდი პოულობს მასივის მაქსიმალურ ელემენტს.

ინტერფეისები. ცხადი რეალიზება

3.1. შექმენით ორი ინტერფეისი, რომლებსაც ერთნაირი სახელის მქონე მეთოდი აქვთ. პირველი ინტერფეისის მეთოდი გამოთვლის და აბრუნებს სამკუთხედის პერიმეტრს. მეორე ინტერფეისის მეთოდი გამოთვლის და აბრუნებს სამკუთხედის ფართობს. შეასრულეთ ამ ინტერფეისების რეალიზება.

3.2. შექმენით ორი ინტერფეისი, რომლებსაც ერთნაირი სახელის მქონე თვისება აქვთ. პირველი ინტერფეისის თვისება პრივატულ ცვლადს დადებით მნიშვნელობებს ანიჭებს და აბრუნებს პრივატული ცვლადის მნიშვნელობას. მეორე ინტერფეისის თვისება პრივატულ ცვლადს უარყოფით მნიშვნელობებს ანიჭებს და აბრუნებს პრივატული ცვლადის მნიშვნელობას. შეასრულეთ ამ ინტერფეისების რეალიზება.

პრაქტიკული მეცადინეობა #1
კლასის შექმნა. ინკაფსულაცია. მეთოდები
კლასის შექმნა. ინკაფსულაცია

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

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

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

კლასის წევრებთან მიმართვა ხორციელდება ოთხი მოდიფიკატორის გამოყენებით: public, private, protected და internal. თუ მოდიფიკატორი არ არის მითითებული, მაშინ იგულისხმება private.შეადგინეთ სამკუთხედის კლასი, რომელშიც გამოცხადებულია სამკუთხედის გვერდები, პერიმეტრი და ფართობი.

```
class Samkutxedi
{
public int gverdi1;          //     სამკუთხედის პირველი გვერდი
public int gverdi2;          //     სამკუთხედის მეორე გვერდი
public int gverdi3;          //     სამკუთხედის მესამე გვერდი
int perimetri;
private int fartobi;
}
private void button1_Click(object sender, System.EventArgs e)
{
Samkutxedi Sam1 = new Samkutxedi();
int perimetri;

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

}

მეთოდები

მეთოდები - ესაა პროგრამები, რომლებიც ასრულებენ გარკვეულ ფუნქციას, მუშაობენ კლასში განსაზღვრულ მონაცემებთან და უზრუნველყოფენ ამ მონაცემებთან მიმართვას. მეთოდის გამოძახება ბევრჯერ შეიძლება. მეთოდი შეიძლება შეიცავდეს ერთ ან მეტ ოპერატორს და უნდა წყვეტდეს მხოლოდ ერთ კონკრეტულ ამოცანას. მეთოდის სახელად შეგვიძლია გამოვიყენოთ ნებისმიერი იდენტიფიკატორი. საკვანძო სიტყვების გამოყენება მეთოდების სახელებად არ შეიძლება.ქვემოთ მოყვანილია Samkutxedi კლასის მომდევნო ვერსია. მის Perimetri() მეთოდს label1 კომპონენტში გამოაქვს სამკუთხედის პერიმეტრის მნიშვნელობა.

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

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

მეთოდიდან მართვის დაბრუნება

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

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

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

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

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

//      tolgverda სამკუთხედისთვის პერიმეტრი გამოითვლება
tolgverda_perimetri = tolgverda.Perimetri();
label1.Text = tolgverda_perimetri.ToString();
}
```

პროგრამაში Perimetri() მეთოდის მიერ გაცემული მნიშვნელობა მიენიჭება tolgverda_perimetri ცვლადს.

პარამეტრების გამოყენება

გამოძახებისას მეთოდს შეგვიძლია გადავცეთ ერთი ან მეტი პარამეტრი. მეთოდისთვის გადასაცემ მნიშვნელობას არგუმენტი ეწოდება. ცვლადს მეთოდის შიგნით, რომელიც იღებს არგუმენტის მნიშვნელობას, პარამეტრი ეწოდება. პარამეტრების გამოცხადება ხდება მრგვალი ფრჩხილების შიგნით, რომლებიც მეთოდის სახელს მოსდევს. პარამეტრის გამოცხადების სინტაქსი ცვლადების გამოცხადების სინტაქსის ანალოგიურია. პარამეტრი იმყოფება თავისი მეთოდის ხილვადობის უბანში, ანუ ხილულია მხოლოდ ამ მეთოდის შიგნით, და მოქმედებს როგორც ლოკალური ცვლადი.

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

კი - false მნიშვნელობას.

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

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

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

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

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

```
        else label2.Text = ricxvi1.ToString() + " არ არის " + ricxvi2.ToString() + "-ის გამყოფი";
    }
```

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

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

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

სავარჯიშოები

კლასის შექმნა. ინკაფსულაცია

1. შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ბაკის ტევადობა და მანძილი, რომელსაც თვითმფრინავი 1 ლიტრი საწვავით გაიფრენს; ღია ცვლადებს: მგზავრების რაოდენობა და გაყიდული ბილეთების რაოდენობა. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი. მის ღია ცვლადებს მიანიჭეთ მნიშვნელობები TextBox ვიზუალური ელემენტიდან. ეს მნიშვნელობები გამოიტანეთ Label ვიზუალურ ელემენტში.
2. შექმენით სტუდენტის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: გვარი, სახელი და ასაკი; ღია ცვლადებს: უნივერსიტეტის დასახელება და კურსი. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი. მის ღია ცვლადებს მიანიჭეთ მნიშვნელობები TextBox ვიზუალური ელემენტიდან. ეს მნიშვნელობები გამოიტანეთ Label ვიზუალურ ელემენტში. გამოიყენეთ ნაგულისხმევი კონსტრუქტორი.
3. შექმენით მატარებლის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ვაგონების რაოდენობასა და მგზავრების რაოდენობას 1 ვაგონში; ღია ცვლადებს: ბილეთების ფასს და გაყიდული ბილეთების რაოდენობას. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი. მის ღია ცვლადებს მიანიჭეთ მნიშვნელობები TextBox ვიზუალური ელემენტიდან. ეს მნიშვნელობები გამოიტანეთ Label ვიზუალურ ელემენტში.
4. შექმენით სამკუთხედის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: პერიმეტრი და ფართობი; ღია ცვლადებს: სამკუთხედის გვერდებს. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი. მის ღია ცვლადებს მიანიჭეთ მნიშვნელობები TextBox ვიზუალური ელემენტიდან. ეს მნიშვნელობები გამოიტანეთ Label ვიზუალურ ელემენტში.
5. შექმენით ავტომანქანის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: მანქანის ფერი და კარებების რაოდენობა; ღია ცვლადებს: მფლობელის გვარი და გამომშვები ფირმა. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი. მის ღია ცვლადებს მიანიჭეთ მნიშვნელობები TextBox ვიზუალური ელემენტიდან. ეს მნიშვნელობები გამოიტანეთ Label ვიზუალურ ელემენტში.

მეთოდები

1. შექმენით მატარებლის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ვაგონების რაოდენობასა და მგზავრების რაოდენობას 1 ვაგონში; ღია ცვლადებს: ბილეთების ფასს და გაყიდული ბილეთების რაოდენობას. შეიცავს ღია მეთოდებს: პირველი მეთოდი პრივატულ და ღია ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს პრივატული და ღია ცვლადების მნიშვნელობები; მესამე მეთოდი გამოთვლის და აბრუნებს ბილეთების გაყიდვით მიღებულ თანხას. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი და გამოიძახეთ სამივე მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, გამოსატანად კი Label ვიზუალურ ელემენტი.

2. შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს ღია ცვლადებს: ბაკის ტევადობა და მანძილი, რომელსაც გაიფრენს თვითმფრინავი 1 ლიტრი საწვავით; ღია მეთოდებს: პირველი მეთოდი ღია ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს გამოაქვს ღია ცვლადების მნიშვნელობები; მესამე მეთოდი გამოთვლის და აბრუნებს თვითმფრინავის მიერ სავსე ბაკით გავლილ მანძილს. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი და გამოიძახეთ სამივე მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, გამოსატანად კი Label ვიზუალურ ელემენტი.
3. შექმენით სტუდენტის კლასი, რომელიც შეიცავს ღია მეთოდს. მეთოდს გადაეცემა სტუდენტის ნიშნები, რომლებიც წარმოადგენენ 10-ელემენტიან მთელრიცხვა მასივს, მეთოდი გამოთვლის და აბრუნებს ნიშნების საშუალო არითმეტიკულს. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი და გამოიძახეთ ეს მეთოდი. მასივი და შედეგი გამოიტანეთ Label ვიზუალურ ელემენტში.
4. შექმენით კვადრატის კლასი, რომელიც შეიცავს ღია ცვლადს: კვადრატის გვერდის ზომას; პრივატულ ცვლადს: კვადრატის ფართობს; ღია მეთოდებს: პირველი მეთოდი ღია ცვლადს მნიშვნელობას ანიჭებს; მეორე მეთოდს გამოაქვს ღია ცვლადის მნიშვნელობა; მესამე მეთოდი გამოთვლის და აბრუნებს კვადრატის ფართობს. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი და გამოიძახეთ სამივე მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, გამოსატანად კი Label ვიზუალურ ელემენტი.
5. შექმენით სტუდენტის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: გვარი, სახელი და ასაკი; ღია მეთოდებს: პირველი მეთოდი პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე მეთოდს ეკრანზე გამოაქვს პრივატული ცვლადების მნიშვნელობები. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი და გამოიძახეთ ორივე მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, გამოსატანად კი Label ვიზუალურ ელემენტი.
6. შექმენით სტუდენტის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: გვარი, სახელი და ასაკი; პრივატულ მეთოდს, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; ღია მეთოდებს: პირველი მეთოდი იძახებს პრივატულ მეთოდს და გადასცემს მნიშვნელობებს პრივატული ცვლადებისათვის მისანიჭებლად; მეორე მეთოდს გამოაქვს პრივატული ცვლადების მნიშვნელობები. მთავარ პროგრამაში ნაგულისხმევი კონსტრუქტორის გამოყენებით შექმენით შესაბამისი ობიექტი და გამოიძახეთ ორივე მეთოდი. მონაცემების შესატანად გამოიყენეთ TextBox ვიზუალური ელემენტი, გამოსატანად კი Label ვიზუალურ ელემენტი.

პრაქტიკული სამუშაო #7
ინფორმაციის შეტანა-გამოტანა

C# ენაში არსებობს ბაიტების, სიმბოლოებისა და ორობითი ნაკადების კლასები. ნაკადების კლასები განსაზღვრულია System.IO სახელების სივრცეში. ამიტომ, ნებისმიერი პროგრამის დასაწყისში, რომელიც იყენებს ნაკადების კლასებს, უნდა ჩავრთოთ ინსტრუქცია using System.IO.

ფაილში ბაიტების ჩაწერა-წაკითხვა
ფაილში ბაიტების ჩაწერა

FileStream კლასში განსაზღვრულია ორი მეთოდი, რომლებიც ფაილში წერენ ბაიტებს: WriteByte() და Write(). WriteByte() მეთოდი გამოიყენება ფაილში ბაიტების ჩასაწერად. მისი სინტაქსია:

void WriteByte(byte ცვლადის_სახელი)

Write() მეთოდი გამოიყენება ფაილში ბაიტების ბლოკის (მასივის) ჩასაწერად. მისი სინტაქსია:

int Write(byte[ ] მასივი, int პოზიცია, int ბაიტების_რაოდენობა)

Write() მეთოდი ფაილში ჩაწერს მითითებული რაოდენობის ბაიტს, დაწყებული მასივის მითითებული პოზიციიდან. გაიცემა რეალურად ჩაწერილი ბაიტების რაოდენობა.

ფაილში ჩაწერის დროს ხშირად არ ხდება მონაცემების მაშინვე ჩაწერა ფიზიკურ მოწყობილობაზე. ამის ნაცვლად, ოპერაციული სისტემა ახდენს გამოტანის ბუფერიზებას, ანუ ჩასაწერი ბაიტების ბუფერში დაგროვებას. ბუფერი არის მეხსიერების გარკვეული უბანი. ბუფერიზება სრულდება მანამ, სანამ არ დაგროვდება მონაცემების საკმარისი რაოდენობა. ბუფერის შევსებისთანავე მონაცემები ჩაიწერება დისკზე. შედეგად, იზრდება ოპერაციული სისტემის მწარმოებლურობა. საქმე ის არის, რომ დისკზე მონაცემების ჩაწერა ხდება სექტორებად, რომელთა ზომები იცვლება დაწყებული 128 ბაიტიდან ზევით. გამოტანის შედეგების ბუფერიზება ხდება მანამ, სანამ დისკზე შესაძლებელი არ გახდება მთელი სექტორის ჩაწერა.

რიგ შემთხვევებში საჭიროა მონაცემების დისკზე ჩაწერა მიუხედავად იმისა ბუფერი შევსებულია თუ არა. ასეთ შემთხვევაში უნდა გამოვიყენოთ Flush() მეთოდი. მისი სინტაქსია:

void Flush()

ფაილთან მუშაობის დამთავრებისას ის უნდა დაიხუროს Close() მეთოდის გამოყენებით. ამ დროს ბუფერში მოთავსებული მონაცემები, დისკზე აუცილებლად ჩაიწერება. ამიტომ, Close() მეთოდის გამოძახების წინ არაა საჭირო Flush() მეთოდის გამოძახება.

მოყვანილ პროგრამაში სრულდება ფაილში ბაიტების ჩაწერა.

```
{
        // პროგრამა ასრულებს ფაილში 10 ბაიტის ჩაწერას
        label1.Text = "";
        int i;
        FileStream file_out;
        // მიმღები ფაილის შექმნა
        file_out = new FileStream("d_file.txt", FileMode.Create);
        // ბაიტების ჩაწერა file_out ფაილში
        for (i = 33; i <= 42; i++)
        {
           file_out.WriteByte((byte) i);
           label1.Text += ((char) i).ToString() + " ";
        }
        // ფაილის დახურვა
        file_out.Close();
}
```

მოყვანილ პროგრამაში Write() მეთოდის გამოიყენებით სრულდება filetext.txt ფაილში ბაიტების მასივის ჩაწერა.

```
{
// პროგრამაში ხდება მასივიდან ფაილში ბაიტების ჩაწერა
byte[] masivi = new byte[] { 43, 44, 45, 46, 47, 48, 49, 50, 51, 52 };
FileStream file_out;
file_out = new FileStream("d_file.txt", FileMode.Create);
// ბაიტების ჩაწერა d_file.txt ფაილში
file_out.Write(masivi, 2, 4);
file_out.Close();
}
```

პროგრამის შესრულების შედეგად filetext.txt ფაილში ჩაიწერება masivi მასივის 4 ელემენტი დაწყებული მესამე ელემენტიდან (masivi[2]).

ფაილიდან ბაიტების წაკითხვა

FileStream კლასი გამოიყენება ფაილებში ბაიტ-ორიენტირებული შეტანა-გამოტანის შესასრულებლად, ანუ ფაილში ბაიტების ჩაწერა-წაკითხვის ოპერაციების შესასრულებლად. FileStream კლასში განსაზღვრულია ორი მეთოდი, რომლებიც ფაილიდან ბაიტებს კითხულობენ: ReadByte() და Read(). ReadByte() მეთოდის სინტაქსია:

int ReadByte()

ამ მეთოდის გამოძახებისას ხდება ერთი ბაიტის წაკითხვა ფაილიდან და გაიცემა ბაიტის კოდი (მთელრიცხვა მნიშვნელობა). თუ წაკითხული იქნა ფაილის დასასრულის სიმბოლო (EOF, End Of File), მაშინ გაიცემა -1.

ბაიტების ბლოკის წასაკითხად უნდა გამოვიყენოთ Read() მეთოდი. მისი სინტაქსია:

int Read(byte[ ] მასივი, int პოზიცია, int ბაიტების_რაოდენობა)

Read() მეთოდი ცდილობს ფაილიდან წაიკითხოს მითითებული რაოდენობის ბაიტი და მოათავსოს ისინი მასივში დაწყებული მითითებული პოზიციიდან. გაიცემა რეალურად წაკითხული ბაიტების რაოდენობა.

მოყვანილ პროგრამაში სრულდება ბაიტების წაკითხვა ReadByte() მეთოდის გამოიყენებით და მათი ეკრანზე გამოტანა. დავუშვათ filetext.txt ფაილში Notepad პროგრამის გამოყენებით ჩაწერილია ციფრები: 1234567890 .

```
{
// პროგრამაში ხდება ბაიტების წაკითხვის დემონსტრირება
// ReadByte() მეთოდის გამოყენებით
int ricxvi;
FileStream file1 = new FileStream("filetext.txt", FileMode.Open);
// მონაცემების წაკითხვა ფაილიდან მანამ, სანამ არ შეგვხვდება EOF სიმბოლო
for ( ; ; )
{
ricxvi = file1.ReadByte();
if ( ricxvi != -1 ) label1.Text += (( char ) ricxvi).ToString() + “  “;
else break;
}
file1.Close();
}
```

მოყვანილ პროგრამაში სრულდება ბაიტების წაკითხვა Read() მეთოდის გამოიყენებით და მათი ეკრანზე გამოტანა. დავუშვათ filetext.txt ფაილში Notepad პროგრამის გამოყენებით ჩაწერილია ციფრები: 1234567890 .

```
{
```

```
// პროგრამაში ხდება ბაიტების წაკითხვის დემონსტრირება Read() მეთოდის გამოყენებით
int wakitxuli_baitebis_raodenoba;
int pozicia = int.Parse(textBox1.Text),
raodenoba = int.Parse(textBox2.Text);
byte[] masivi = new byte[10];
FileStream file1 = new FileStream("filetext.txt", FileMode.Open);
label1.Text = "";
// masivi მასივში ჩაიწერება 4 ბაიტი დაწყებული მე-3 ელემენტიდან
wakitxuli_baitebis_raodenoba = file1.Read(masivi, pozicia, raodenoba);
file1.Close();
for (int indexi = 0; indexi < masivi.Length; indexi++)
label1.Text += masivi[indexi].ToString() + " ";
label2.Text = wakitxuli_baitebis_raodenoba.ToString();
}
```

როგორც ვიცით Read() მეთოდს სამი არგუმენტი აქვს: მასივი, პოზიცია და ბაიტების რაოდენობა. შესაბამისად, პროგრამაში გამოცხადებულია masivi მასივი, pozicia და raodenoba ცვლადები. დავუშვათ, pozicia ცვლადს მივანიჭეთ 2, raodenoba ცვლადს კი - 4. შედეგად, filetext.txt ფაილიდან შესრულდება 4 ბაიტის წაკითხვა და masivi მასივში მესამე პოზიციიდან (masivi[2]) ჩაწერა. masivi მასივი მიიღებს სახეს:

0 0 49 50 51 52 0 0 0 0

Read() მეთოდი, აგრეთვე, გასცემს რეალურად წაკითხული ბაიტების რაოდენობას, რომელსაც მოვათავსებთ wakitxuli_baitebis_raodenoba ცვლადში.

მოყვანილ პროგრამაში სრულდება ერთი ფაილის მეორეში გადაწერა.

```
{
// პროგრამა ასრულებს ერთი ფაილის მეორეში გადაწერას
label1.Text = "";
int ricxvi;
FileStream file_in;
FileStream file_out;
// საწყისი ფაილის გახსნა
file_in = new FileStream("s_file.txt", FileMode.Open);
// მიმღები ფაილის გახსნა
file_out = new FileStream("d_file.txt", FileMode.Create);
// ბაიტების გადაწერა file_in ფაილიდან file_out ფაილში
for ( ; ; )
{
// ბაიტების წაკითხვა file_in ფაილიდან
ricxvi = file_in.ReadByte();
// ბაიტების ჩაწერა file_out ფაილში
if ( ricxvi != -1 )
{
file_out.WriteByte(( byte ) ricxvi);
label1.Text += (( char ) ricxvi).ToString() + " ";
}
else break;
}
// ფაილების დახურვა
file_in.Close();
file_out.Close();
```

}

ფაილში სიმბოლოების შეტანა-გამოტანა

ფაილში სიმბოლოების ჩასაწერად და წასაკითხად სიმბოლოების ნაკადები გამოიყენება. ისინი Unicode სიმბოლოებზე ოპერირებენ. სიმბოლურ ფაილებთან მუშაობისას FileStream ობიექტი უნდა ჩავრთოთ StreamReader ან StreamWrtiter კლასის შემადგენლობაში. ეს კლასები უზრუნველყოფენ ბაიტების ნაკადების სიმბოლოების ნაკადებად ავტომატურ გარდაქმნას და პირიქით.

ფაილში სიმბოლოების ჩაწერა

StreamWriter კლასი გამოიყენება ფაილში სიმბოლოების ჩასაწერად. ამისათვის, FileStream ობიექტი უნდა ჩავრთოთ StreamWriter კლასის შემადგენლობაში. ამ კლასში განსაზღვრულია რამდენიმე კონსტრუქტორი. ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

StreamWriter(Stream ნაკადის_სახელი)

ობიექტის შექმნისთანავე StreamWriter კლასი ავტომატურად გარდაქმნის სიმბოლოებს ბაიტებად.

მოყვანილი პროგრამა ახდენს textBox კომპონენტში შეტანილი ტექსტის ჩაწერას ფაილში.

```
{
// პროგრამაში ხდება ფაილში ჩაწერის დემონსტრირება StreamWriter კლასის გამოყენებით
string striqoni = textBox1.Text;
FileStream file_out = new FileStream("text.txt", FileMode.Create);
//
StreamWriter file_stream_out = new StreamWriter(file_out);
// textBox კომპონენტში შეტანილი ტექსტის მინიჭება striqoni ცვლადისათვის
file_stream_out.Write(striqoni);          // text.txt ფაილში striqoni სტრიქონის ჩაწერა
file_stream_out.Close();
}
```

ფაილიდან სიმბოლოების წაკითხვა

სიმბოლოების შესასვლელი ნაკადი გამოიყენება სიმბოლოების ნაკადიდან მონაცემების წასაკითხად. ამისათვის, FileStream ობიექტი უნდა ჩავრთოთ StreamReader კლასის შემადგენლობაში. ამ კლასში განსაზღვრულია რამდენიმე კონსტრუქტორი. ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

StreamReader(Stream ნაკადის_სახელი)

StreamReader ობიექტის შექმნისთანავე ავტომატურად სრულდება ბაიტების გარდაქმნა სიმბოლოებად.

მოყვანილი პროგრამა კითხულობს ტექსტურ ფაილს და მისი შემცველობა ეკრანზე გამოაქვს. Notepad პროგრამის გამოყენებით ფაილში ჩავწეროთ რამდენიმე სტრიქონი და შემდეგ შევასრულოთ მოყვანილი პროგრამა.

```
{
// პროგრამაში ხდება ტექსტური ფაილიდან წაკითხვის დემონსტრირება
// StreamReader კლასის გამოყენებით
label1.Text = "";
FileStream file_in;
string striqoni;
file_in = new FileStream("file1.txt", FileMode.Open );
StreamReader file_stream_in = new StreamReader(file_in);
//
for ( ; ( striqoni = file_stream_in.ReadLine() ) != null; )
```

```
label1.Text += striqoni + '\n';
file_stream_in.Close();
}
```

პროგრამაში ფაილიდან სტრიქონების წაკითხვა მთავრდება მაშინ, როცა ReadLine() მეთოდის მიერ გაცემული სტრიქონი მიიღებს ნულოვან მნიშვნელობას. სწორედ ეს მიუთითებს ფაილის დასასრულზე.

ფაილში ორობითი მონაცემების ჩაწერა და წაკითხვა

ფაილში ბაიტებისა და სიმბოლოების წაკითხვისა და ჩაწერის გარდა შესაძლებელია სხვა ტიპის (ორობითი) მონაცემების წაკითხვა და ჩაწერა, როგორიცაა int, double და ა.შ. მათი წაკითხვისა და ჩაწერისათვის გამოიყენება BinaryReader და BinaryWriter ნაკადები. ასეთი მონაცემების წაკითხვა და ჩაწერა ხდება შიდა ორობით ფორმატში და არა ტექსტური ფორმით.

ფაილში ორობითი მონაცემების ჩაწერა

BinaryWriter ნაკადი გამოიყენება ფაილში ორობითი მონაცემების ჩასაწერად. ამ კლასის ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

BinaryWriter(Stream გამოსასვლელი_ნაკადი)

აქ გამოსასვლელი_ნაკადი პარამეტრი განსაზღვრავს იმ ნაკადს, რომელშიც სრულდება მონაცემების ჩაწერა. ის არის ობიექტი, რომელსაც ქმნის FileStream კონსტრუქტორი.

BynaryWriter კლასში განსაზღვრულია მეთოდები, რომლებიც უზრუნველყოფენ C# ენის ყველა ჩადგმული ტიპის მონაცემების ჩაწერას. ამავე კლასში განსაზღვრულია, აგრეთვე, Close() და Flush() მეთოდები.

ფაილიდან ორობითი მონაცემების წაკითხვა

BinaryReader ნაკადი გამოიყენება ფაილიდან ორობითი მონაცემების წასაკითხად. ამ კლასის ხშირად გამოყენებადი კონსტრუქტორის სინტაქსია:

BinaryReader(Stream შესასვლელი_ნაკადი)

აქ შესასვლელი_ნაკადი პარამეტრი განსაზღვრავს იმ ნაკადს, საიდანაც სრულდება მონაცემების წაკითხვა. ის არის ობიექტი, რომელსაც ქმნის FileStream კონსტრუქტორი.

BinaryReader კლასში განსაზღვრულია მეთოდები, რომლებიც ახდენენ C# ენის ყველა მარტივი ტიპის მონაცემების წაკითხვას.

ქვემოთ მოყვანილ პროგრამაში გამოყენებულია BinaryReader და BinaryWriter ნაკადები. პროგრამაში ჯერ სრულდება ფაილში სხვადასხვა ტიპის მონაცემების ჩაწერა, შემდეგ კი წაკითხვა.

```
{
// პროგრამაში ხდება ფაილში ორობითი მონაცემების ჩაწერა-წაკითხვის დემონსტრირება
BinaryReader file_in;
BinaryWriter file_out;
int mteli1 = 10, mteli2;
double wiladi1 = 1001.47, wiladi2, wiladi3;
bool b1 = true, b2;
file_out = new BinaryWriter(new FileStream("file1.dat", FileMode.Create));
// ფაილში მთელი რიცხვის ჩაწერა
file_out.Write(mteli1);
// ფაილში წილადის ჩაწერა
file_out.Write(wiladi1);
// ფაილში ლოგიკური მონაცემის ჩაწერა
file_out.Write(b1);
// ფაილში იწერება 10.2 * 2.3 გამოსახულების გამოთვლის შედეგი
```

```
file_out.Write(10.2 * 2.3);
file_out.Close();
// ფაილიდან წაკითხვა
file_in = new BinaryReader(new FileStream("file1.dat",FileMode.Open));
// მთელი რიცხვის წაკითხვა ფაილიდან
mteli2= file_in.ReadInt32();
label1.Text = mteli2.ToString();
// წილადის წაკითხვა ფაილიდან
wiladi2 = file_in.ReadDouble();
label2.Text = wiladi2.ToString();
// ლოგიკური მონაცემის წაკითხვა ფაილიდან
b2 = file_in.ReadBoolean();
label3.Text = b2.ToString();
// წილადის წაკითხვა ფაილიდან
wiladi3 = file_in.ReadDouble();
label4.Text = wiladi3.ToString();
file_in.Close();
}
```

ფაილებთან პირდაპირი მიმართვა

აქამდე ვიყენებდით ფაილებთან მიმდევრობით მიმართვას. ამ დროს მიმდევრობით სრულდება ფაილში მონაცემების ჩაწერა და ფაილიდან მონაცემების წაკითხვა. ფაილთან შესაძლებელია, აგრეთვე, პირდაპირი (ნებისმიერი) მიმართვაც. ამისათვის, გამოიყენება საფაილო მიმთითებელი. ის არის ფაილის მიმდინარე ელემენტის პოზიციის ნომერი. საფაილო მიმთითებლის მნიშვნელობის შესაცვლელად გამოიყენება Seek() მეთოდი. მისი სინტაქსია:

long Seek(long პოზიციის_ნომერი, SeekOrigin გადათვლის_დასაწყისი)

პოზიციის_ნომერი პარამეტრი მიუთითებს საფაილო მიმთითებლის ახალ პოზიციას (ბაიტებში) გადათვლილს გადათვლის_დასაწყისი პარამეტრით განსაზღვრული ადგილიდან. გადათვლის_დასაწყისი პარამეტრი იღებს SeekOrigin ჩამონათვალში განსაზღვრული მნიშვნელობებიდან ერთ-ერთს:

გადათვლის_დასაწყისი პარამეტრის მნიშვნელობები. მნიშვნელობა	აღწერა
Begin	გადათვლა ფაილის დასაწყისიდან
Current	გადათვლა მიმდინარე პოზიციიდან
End	გადათვლა ფაილის დასასრულიდან

Seek() მეთოდის გამოძახების შემდეგ წაკითხვის ან ჩაწერის ოპერაციები სრულდება საფაილო მიმთითებლის მიერ განსაზღვრული ახალი პოზიციიდან.

ქვემოთ მოყვანილ პროგრამაში, ხდება შეტანა-გამოტანის ოპერაციის დემონსტრირება ფაილთან ნებისმიერი მიმართვით. ფაილში ჯერ ჩაიწერება ანბანის ასოები, შემდეგ კი ხდება მათი სხვადასხვა მიმდევრობით წაკითხვა.

```
{
// პროგრამაში ხდება ფაილში ჩაწერა-წაკითხვის ოპერაციების შესრულება
// პირდაპირი მიმართვით
FileStream file;
file = new FileStream("file.txt",FileMode.Create);
char simbolo;
// ასოების ჩაწერა ანბანის მიხედვით
for ( int i = 0; i < 26; i++ )
```

```
{
file.WriteByte( ( byte ) ( 'a' + i ) );
}
// ასოების წაკითხვა ფაილიდან
file.Seek(0, SeekOrigin.Begin); // პირველი ბაიტის არჩევა
simbolo = (char) file.ReadByte(); // პირველი ბაიტის წაკითხვა
label1.Text = "პირველი მნიშვნელობა " + simbolo + '\n';
file.Seek(4, SeekOrigin.Begin); // მეხუთე ბაიტის არჩევა
simbolo = ( char ) file.ReadByte(); // მეხუთე ბაიტის წაკითხვა
label1.Text += "მეხუთე მნიშვნელობა " + simbolo + '\n';
file.Close();
}
```

File, FileInfo, Directory და DirectoryInfo კლასები

System.IO სახელების სივრცეში განსაზღვრულია File და FileInfo კლასები. File კლასში განსაზღვრულია სტატიკური მეთოდები, FileInfo კლასში კი - ჩვეულებრივი მეთოდები. ნებისმიერი პროგრამის დასაწყისში, რომელიც იყენებს ნაკადების კლასებს, უნდა ჩავრთოთ ინსტრუქცია using System.IO .

ფაილებთან მუშაობა

მოყვანილ პროგრამაში ხდება File კლასის AppendText მეთოდთან მუშაობის დემონსტრირება.

```
{
// პროგრამაში ხდება File კლასის AppendText() მეთოდთან მუშაობის დემონსტრირება
// ფაილისათვის ხდება სტრიქონების დამატება
string path = "file1.txt";
StreamWriter faili = File.AppendText(path);
faili.WriteLine("რომან");
faili.WriteLine("სამხარაძე");
faili.WriteLine("C# დაპროგრამების ენაა");
faili.Close();
}
```

აქ using არის ოპერატორი, რომელიც განსაზღვრავს მოქმედების უბანს, რომლის ბოლოსაც faili ობიექტი წაიშლება.

მოყვანილ პროგრამაში ხდება File კლასის OpenText() მეთოდთან მუშაობის დემონსტრირება.

```
{
// პროგრამაში ხდება File კლასის OpenText() მეთოდთან მუშაობის დემონსტრირება
// ფაილის გახსნა წაკითხვის მიზნით
label1.Text = "";
StreamReader faili = File.OpenText("file1.txt");
string striqoni = "";
//
for ( ; ( striqoni = faili.ReadLine() ) != null; )
        label1.Text += striqoni + "\n";
}
```

მოყვანილ პროგრამაში ხდება File კლასის CreateText() მეთოდთან მუშაობის დემონსტრირება.

```
{
```

```
// პროგრამაში ხდება File კლასის CreateText() მეთოდთან მუშაობის დემონსტრირება
string path = "file1.txt";
// ფაილის შექმნა მასში სტრიქონების ჩაწერის მიზნით
using ( StreamWriter faili = File.CreateText(path) )
{
faili.WriteLine("ანა ");
faili.WriteLine("და ");
faili.WriteLine("საბა ");
faili.WriteLine("სამხარაძეები");
faili.Close();
}
}
```

მოყვანილ პროგრამაში ხდება File კლასის Delete() და Copy() მეთოდებთან მუშაობის დემონსტრირება.

```
{
// პროგრამაში ხდება File კლასის Delete() და Copy() მეთოდთან მუშაობის დემონსტრირება
string path1 = "file1.txt";          //ეს ფაილი წინასწარ უნდა შევქმნათ და შევავსოთ სტრიქონებით
string path2 = "file2.tmp";
// file2.tmp ფაილის შექმნა
FileStream faili_obj = File.Create(path2);
faili_obj.Close();
// file1.tmp ფაილის წაშლა
File.Delete(path2);
// file1.txt ფაილის გადაწერა file2.tmp ფაილში. file2.tmp ფაილი არ უნდა არსებობდეს.
File.Copy(path1, path2);
label1.Text = path1 + " ფაილი გადაიწერა " + path2 + "ფაილში";
}
```

მოყვანილ პროგრამაში ხდება File კლასის GetAttributes() მეთოდთან მუშაობის დემონსტრირება.

```
{
FileAttributes faili_obj = File.GetAttributes("file1.txt");
label1.Text = faili_obj.ToString();
}
```

მოყვანილ პროგრამაში ხდება FileInfo კლასის თვისებებთან მუშაობის დემონსტრირება.

```
{
// პროგრამაში ხდება FileInfo კლასის თვისებებთან მუშაობის დემონსტრირება
System.IO.FileInfo obieqti = new System.IO.FileInfo("file1.txt");
label1.Text += obieqti.Directory.ToString() + '\n';
label1.Text += obieqti.DirectoryName + '\n';
label1.Text += obieqti.FullName + '\n';
label1.Text += obieqti.Name + '\n';
label1.Text += obieqti.Extension + '\n';
label1.Text += obieqti.Length.ToString() + '\n';
label1.Text += obieqti.Exists.ToString() + '\n';
label1.Text += obieqti.CreationTime.ToString() + '\n';
label1.Text += obieqti.LastAccessTime.ToString() + '\n';
label1.Text += obieqti.LastWriteTime.ToString() + '\n';
}
```

კატალოგებთან მუშაობა

System.IO სახელების სივრცეში განსაზღვრულია, აგრეთვე, Directory და DirectoryInfo კლასები. File კლასის ანალოგიურად Directory კლასში განსაზღვრულია სტატიკური მეთოდები, DirectoryInfo კლასში კი - ჩვეულებრივი მეთოდები. რადგან Directory კლასის მეთოდები სტატიკურია, ამიტომ მათთან სამუშაოდ არ არის საჭირო ობიექტის შექმნა, ხოლო DirectoryInfo კლასის მეთოდებთან სამუშაოდ კი საჭიროა ობიექტის შექმნა.

ქვემოთ მოყვანილ პროგრამაში ხდება მითითებული კატალოგის სახელის, მასში მოთავსებული ფაილებისა და კატალოგების სახელების ეკრანზე გამოტანა, კატალოგის გახსნა, ეკრანზე დისკების სახელების ეკრანზე გამოტანა. პროგრამაში კატალოგის სახელის შეტანა ხდება textBox1.Text კომპონენტში, მაგალითად, D:\\Directory. ამ დროს ბრჭყალების გამოყენება საჭირო არ არის.

```
// მიმდინარე კატალოგის სახელის გაცემა
label1.Text = Directory.GetCurrentDirectory() + '\n';
// მშობელი კატალოგის სახელის გაცემა
label1.Text += Directory.GetParent(textBox1.Text).ToString()+ '\n';
// მითითებული კატალოგის გახსნა
Directory.SetCurrentDirectory(textBox1.Text);
// ეკრანზე ფაილების სახელების გამოტანა
string[] striqonebis_masivi_1 = Directory.GetFiles(textBox1.Text);
foreach ( string s in striqonebis_masivi_1 )
label2.Text += s + '\n';
// ეკრანზე კატალოგების სახელების გამოტანა
string[] striqonebis_masivi_2 = Directory.GetDirectories(textBox1.Text);
foreach ( string s in striqonebis_masivi_2 )
label1.Text += s + '\n';
// ეკრანზე დისკების სახელების გამოტანა
string[] striqonebis_masivi_3 = Directory.GetLogicalDrives();
foreach ( string s in striqonebis_masivi_3 )
label1.Text += s + '\n';
}
```

ქვემოთ მოყვანილ პროგრამაში სრულდება კატალოგის შექმნის, წაშლისა და გადატანის დემონსტრირება. Move() მეთოდში ჯერ ეთითება საწყისი კატალოგის სახელი, შემდეგ კი - საბოლოო კატალოგის სახელი. ორივე კატალოგი ერთ დისკზე უნდა იყოს მოთავსებული.

```
{
// პროგრამაში ხდება კატალოგის შექმნა, წაშლისა და გადატანის დემონსტრირება
label2.Text = "";
// კატალოგის შექმნა
Directory.CreateDirectory(textBox1.Text);
// კატალოგის წაშლა. წასაშლელი კატალოგი ცარიელი უნდა იყოს
Directory.Delete(textBox2.Text);
// კატალოგის გადატანა. Move მეთოდში ჯერ ეთითება საწყისი კატალოგი,
// შემდეგ კი - მიმღები.
// მიმღები კატალოგი არ უნდა არსებობდეს
Directory.Move(textBox3.Text, "D:\\Directory2");
}
```

MoveTo() მეთოდი ახდენს Katalogi3 კატალოგში მოთავსებული ფაილებისა და კატალოგების გადატანას Directory2 კატალოგში. გადატანის შემდეგ Katalogi3 კატალოგი იშლება.

```
{
// პროგრამაში ხდება ეკრანზე მითითებული კატალოგის სახელისა და ატრიბუტების გამოტანა
```

```
label1.Text = "";
DirectoryInfo obj1 = new DirectoryInfo("D:\\Zprint");
label1.Text += obj1.Name + '\n';
label1.Text += obj1.Attributes.ToString() + '\n';
}
```

სავარჯიშოები

შეადგინეთ პროგრამა, რომელიც:

1. ერთი ფაილიდან წაიკითხავს byte ტიპის 5 მთელ რიცხვს, მოათავსებს მათ ერთგანზომილებიან მასივში და ამ მასივის ელემენტებს შორის მინიმალურს დაუმატებს ამავე ფაილს.
2. ერთი ფაილიდან წაიკითხავს byte ტიპის 5 მთელ რიცხვს, მოათავსებს მათ ერთგანზომილებიან მასივში და ამ მასივის ელემენტებსა და მათ შორის მაქსიმალურს ჩაწერს მეორე ფაილში.
3. ერთი ფაილიდან მეორეში გადაწერს byte ტიპის 10 მთელ რიცხვს.
4. ერთი ფაილიდან წაიკითხავს 10 სიმბოლოს, მოათავსებს მათ სიმბოლოების მასივში და ამ მასივის ელემენტებს ჩაწერს მეორე ფაილში.
5. ერთი ფაილიდან წაიკითხავს 10 სიმბოლოს, მოათავსებს მათ სიმბოლოების მასივში და ამ მასივის ელემენტებს დაუმატებს ამავე ფაილს.
6. ერთი ფაილიდან წაიკითხავს სტრიქონს და ჩაწერს მას მეორე ფაილში.
7. ფაილში ჩაწერს რამდენიმე სტრიქონს.
8. ფაილიდან წაიკითხავს რამდენიმე სტრიქონს.
9. ერთი ფაილიდან წაიკითხავს რამდენიმე სტრიქონს და ჩაწერს მათ მეორე ფაილში.
10. ფაილიდან წაიკითხავს 10 მთელ რიცხვს და ჩაწერს მათ მეორე ფაილში.
11. ფაილიდან წაიკითხავს 10 წილად რიცხვს და ჩაწერს მათ მეორე ფაილში.
12. ფაილიდან წაიკითხავს 20 სიმბოლოიან სტრიქონს და ჩაწერს მათ მეორე ფაილში.
13. ფაილიდან წაიკითხავს 10 მთელ რიცხვს. ამ რიცხვებს და მათ ჯამს დაუმატებს ამავე ფაილს.
14. ფაილიდან წაიკითხავს 10 წილად რიცხვს. ამ რიცხვებს და მათ ჯამს დაუმატებს ამავე ფაილს.
15. ფაილიდან წაიკითხავს 15 მთელ რიცხვს, მოათავსებს მათ ერთგანზომილებიან მასივში და ამ რიცხვეწბს ჩაწერს მეორე ფაილში.
16. ფაილის მე-2 პოზიციიდან წაიკითხავს 10 ბაიტს და მოათავსებს მათ ერთგანზომილებიან მასივში.
17. ფაილის მე-5 პოზიციიდან ჩაწერს 10 ბაიტს ერთგანზომილებიანი მასივიდან.

File, FileInfo, Directory და DirectoryInfo კლასები

შეადგინეთ პროგრამა, რომელიც:

1. ტექსტურ ფაილს ოთხ სტრიქონს დაუმატებს.
2. ტექსტური ფაილიდან წაიკითხავს ყველა სტრიქონს. ფაილში სტრიქონები წინასწარ ჩაწერეთ Notepad პროგრამის გამოყენებით (ხუთი სტრიქონი).
3. შექმნის ტექსტურ ფაილს და მასში ოთხ სტრიქონს ჩაწერს.
4. ჯერ წაშლის file2.txt ფაილს, შემდეგ კი - file1.txt ფაილს გადაწერს file2.txt ფაილს. ორივე ფაილი მოთავსებულია D:\ დისკზე.
5. გასცემს ფაილის ატრიბუტებს.
6. გასცემს ფაილის: სახელს, გაფართოებას, სრულ სახელს, კატალოგის სახელს, რომელშიც ის არის მოთავსებული, სიგრძეს, შექმნის დროს, მიმართვის დროს, ჩაწერის დროს. ფაილის სახელი TextBox ვიზუალური ელემენტიდან შეიტანეთ.
7. ეკრანზე გამოიტანს მიმდინარე კატალოგის სახელს;
8. ეკრანზე გამოიტანს მშობელი კატალოგის სახელს.
9. გახსნის მითითებულ კატალოგს. კატალოგის სახელი TextBox ვიზუალური ელემენტიდან შეიტანეთ.
10. ეკრანზე გამოიტანს მითითებულ კატალოგში მოთავსებული ფაილების სახელებს. კატალოგის სახელი TextBox ვიზუალური ელემენტიდან შეიტანეთ.

11. ეკრანზე გამოიტანს მითითებულ კატალოგში მოთავსებული კატალოგების სახელებს. კატალოგის სახელი TextBox ვიზუალური ელემენტიდან შეიტანეთ.
12. ეკრანზე გამოიტანს ლოგიკური დისკების სახელებს.
13. შექმნის კატალოგს. კატალოგის სახელის TextBox ვიზუალური ელემენტიდან შეიტანეთ.
14. წაშლის კატალოგს. კატალოგის სახელის TextBox ვიზუალური ელემენტიდან შეიტანეთ.
15. ერთ კატალოგს მეორეში გადაიტანს. კატალოგების სახელები TextBox ვიზუალური ელემენტებიდან შეიტანეთ.
16. ეკრანზე გამოიტანს მითითებული კატალოგის ატრიბუტებს. კატალოგის სახელის TextBox ვიზუალური ელემენტიდან შეიტანეთ.