ლაბ #12
ADO.NET კლასები. SQL მოთხოვნების უშუალო შესრულება

სკალარული მნიშვნელობის დაბრუნება. *ExecuteScalar()* მეთოდი

როცა საჭიროა SQL მოთხოვნისაგან ერთი შედეგის მიღება, მაგალითად პერსონალის საშუალო ასაკი, მაქსიმალური ხელფასი და ა.შ. მაშინ უმჯობესია SlqCommand ობიექტის ExecuteScalar() მეთოდის გამოყენება. ის გასცემს სკალარულ მნიშვნელობას. მოყვანილი პროგრამის შესრულებით გაიცემა ფირმაში მომუშავე თანამშრომლების რაოდენობა:

```
{
        SqlConnection conn_1 = new SqlConnection("server=ROMANI\\SQLL_2019_1;
database=Shekveta; uid=sa; pwd=1");
        conn_1.Open();
        SqlCommand comm_1 = conn_1.CreateCommand();
        comm_1.CommandText = "SELECT COUNT(*) FROM Personali";
        object countResult = comm_1.ExecuteScalar();
        label1.Text = countResult.ToString() + "\n";
        //
        comm_1.CommandText = "SELECT COUNT(*) FROM Personali WHERE qalaqi =
N'თბილისი'";
        object countResult_1 = comm_1.ExecuteScalar();
        label1.Text += countResult_1.ToString() + "\n";
        //
        comm_1.CommandText = "SELECT SUM(xelfasi) FROM Personali";
        Object sumResult = comm_1.ExecuteScalar();
        label8.Text += sumResult.ToString() + "\n";
        //
        comm_1.CommandText = "SELECT AVG(xelfasi) FROM Personali";
        Object avgResult = comm_1.ExecuteScalar();
        label9.Text += avgResult.ToString() + "\n";
        //
        comm_1.CommandText = "SELECT MAX(staji) FROM Personali WHERE
tarigi_dabadebis >= '12.22.1980'";
        Object maxResult = comm_1.ExecuteScalar();
        label10.Text += maxResult.ToString() + "\n";
        //
        comm_1.CommandText = "SELECT MIN(asaki) FROM Personali WHERE staji > 10";
        Object minResult = comm_1.ExecuteScalar();
        label11.Text += minResult.ToString();
//      შეგვიძლია პარამეტრების გამოყენებაც:
comm_1.CommandText = "SELECT COUNT(*) FROM Personali WHERE asaki >= @asaki";
comm_1.Parameters.Add("@asaki", SqlDbType.Int, 4);
comm_1.Parameters["@asaki"].Value = int.Parse(textBox1.Text);
Object minResult = comm_1.ExecuteScalar();
label11.Text += minResult.ToString();
```

```
//
        conn_1.Close();
}
```

INSERT მოთხოვნის უშუალო შესრულება

Personali ცხრილს შეგვიძლია დავუმატოთ ახალი სტრიქონი INSERT მოთხოვნის უშუალოდ შესრულების გზით. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
{
SqlConnection conn_1 = new SqlConnection("server=ROMANI\\SQLL_2019_1;
database=Shekveta; uid=sa; pwd=1");
        conn_1.Open();
        SqlCommand comm_1 = conn_1.CreateCommand();
        comm_1.CommandText =
          "INSERT INTO Personali (gvari, ganyofileba, xelfasi, asaki, tarigi_dabadebis) " +
          "VALUES (@gvari, @ganyofileba, @xelfasi, @asaki, @tarigi_dabadebis)";
        comm_1.Parameters.Add("@gvari", SqlDbType.NVarChar,20);
        comm_1.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 20);
        comm_1.Parameters.Add("@xelfasi", SqlDbType.Float, 8);
        comm_1.Parameters.Add("@asaki", SqlDbType.Int);
        comm_1.Parameters.Add("@tarigi_dabadebis", SqlDbType.DateTime, 8);

        comm_1.Parameters["@gvari"].Value = textBox1.Text;
        comm_1.Parameters["@ganyofileba"].Value = textBox2.Text;
        comm_1.Parameters["@xelfasi"].Value = double.Parse(textBox3.Text);
        comm_1.Parameters["@asaki"].Value = int.Parse(textBox4.Text);
        comm_1.Parameters["@tarigi_dabadebis"].Value = dateTimePicker1.Value;

        //int asaki = int.Parse(textBox4.Text);
        //comm_1.Parameters.AddWithValue("@asaki", asaki);

        int rows_1 = comm_1.ExecuteNonQuery();
        label1.Text = rows_1.ToString();
        conn_1.Close();
}
```

UPDATE მოთხოვნის უშუალო შესრულება. *ExecuteNonQuery()* მეთოდი

INSERT, UPDATE და DELETE ბრძანებები ახდენენ ცხრილში მონაცემების შეცვლას. მათ შესასრულებლად გამოიყენება SqlCommand ობიექტის ExecuteNonQuery() მეთოდი, რომელიც, აგრეთვე გასცემს ამ ბრძანებებით შეცვლილი სტრიქონების რაოდენობას. მოყვანილი პროგრამით ხდება ფირმის თანამშრომლების ხელფასის გაზრდა 10%-ით:

```
{
```

```
SqlConnection conn_1 = new SqlConnection("server=ROMANI\\SQLL_2019_1;
database=Shekveta; uid=sa; pwd=1");
conn_1.Open();
SqlCommand comm_1 = conn_1.CreateCommand();
//comm_1.CommandText = "UPDATE Personali SET xelfasi = xelfasi * 1.10";
//comm_1.CommandText = "UPDATE Personali SET xelfasi = xelfasi * 1.1 WHERE asaki > 35";
//comm_1.CommandText = "UPDATE Personali SET xelfasi = xelfasi * 1.1 WHERE
//tarigi_dabadebis >= '01.01.1985'";
comm_1.CommandText = "UPDATE Personali SET xelfasi = xelfasi * 1.1 WHERE ganyofileba =
N'სასპორტო'";
int rows_1 = comm_1.ExecuteNonQuery();
label1.Text = rows_1.ToString();
conn_1.Close();
}
```

ახლა იგივე პროგრამა დავწეროთ პარამეტრების გამოყენებით:

```
{
SqlConnection conn_1 = new SqlConnection("server=ROMANI\\SQLL_2019_1;
database=Shekveta; uid=sa; pwd=1");
conn_1.Open();
SqlCommand comm_1 = conn_1.CreateCommand();
comm_1.CommandText =
"UPDATE Personali SET xelfasi = xelfasi + @xelfasi WHERE ganyofileba = @ganyofileba";
comm_1.Parameters.Add("@xelfasi", SqlDbType.Float, 8);
comm_1.Parameters.Add("@ganyofileba", SqlDbType.NVarChar, 20);
comm_1.Parameters["@xelfasi"].Value = double.Parse(textBox3.Text);
comm_1.Parameters["@ganyofileba"].Value = textBox2.Text;
int rows_1 = comm_1.ExecuteNonQuery();
label1.Text = rows_1.ToString();
conn_1.Close();
}
```

მოყვანილ პროგრამაში ფილტრში მონაწილეობს DATETIME ტიპის სვეტი:

```
{
SqlConnection conn_Shekveta = new SqlConnection("server=ROMANI\\SQLL_2019_1;
database=Shekveta; uid=sa; pwd=paroli");
conn_Shekveta.Open();
SqlCommand comm_1 = conn_Shekveta.CreateCommand();
comm_1.CommandText = "UPDATE Personali SET xelfasi = xelfasi + @xelfasi WHERE
tarigi_dabadebis >= @tarigi_dabadebis";
comm_1.Parameters.Add("@xelfasi", SqlDbType.Float, 8);
comm_1.Parameters.Add("@tarigi_dabadebis", SqlDbType.DateTime, 8);
comm_1.Parameters["@xelfasi"].Value = double.Parse(textBox1.Text);
comm_1.Parameters["@tarigi_dabadebis"].Value = dateTimePicker1.Value.ToString();
int rows_1 = comm_1.ExecuteNonQuery();
label1.Text = rows_1.ToString();
conn_Shekveta.Close();
```

```
}
```

შეიძლება, ასევე ორი სვეტის შეცვლაც:

```
{
SqlConnection conn_1 = new SqlConnection("server=ROMANI\\SQLL_2019_1;
database=Shekveta; uid=sa; pwd=1");
        conn_1.Open();
        SqlCommand comm_1 = conn_1.CreateCommand();
        comm_1.CommandText =
                "UPDATE Personali SET xelfasi = xelfasi + @xelfasi, staji = staji + @staji";
        comm_1.Parameters.Add("@xelfasi", SqlDbType.Float, 8);
        comm_1.Parameters.Add("@staji", SqlDbType.Int, 4);
        comm_1.Parameters["@xelfasi"].Value = double.Parse(textBox3.Text);
        comm_1.Parameters["@staji"].Value = int.Parse(textBox4.Text);

        int rows_1 = comm_1.ExecuteNonQuery();
        label1.Text = rows_1.ToString();
        conn_1.Close();
}
```

DELETE მოთხოვნის უშუალო შესრულება

მოყვანილ პროგრამაში ხდება მონაცემების (სტრიქონების) წაშლა იმ თანამშრომლების შესახებ, რომლებიც დაბადებული არიან მითითებულ თარიღზე ადრე. თარიღის შესატანად შეგვიძლია dateTimePicker კომპონენტის გამოყენება:

```
{
SqlConnection conn_1 = new SqlConnection("server=ROMANI\\SQLL_2019_1;
database=Shekveta; uid=sa; pwd=1");
        conn_1.Open();
        SqlCommand comm_1 = conn_1.CreateCommand();
        comm_1.CommandText = "DELETE FROM Personali WHERE personaliID =
@personaliID";
        comm_1.Parameters.Add("@personaliID", SqlDbType.Int);
        comm_1.Parameters["@personaliID"].Value = int.Parse(textBox1.Text);
        //comm_1.CommandText = "DELETE FROM Personali WHERE (tarigi_dabadebis <
@tarigi_dabadebis)";

        int rowsAffected = comm_1.ExecuteNonQuery();
        label1.Text = rowsAffected.ToString();
        conn_1.Close();
}
```

სავარჯიშოები

1. შეადგინეთ პროგრამა, რომელიც გასცემს თბილისელი შემკვეთების რაოდენობას. (ExecuteScalar() მეთოდი).
2. შეადგინეთ პროგრამა, რომელიც გასცემს იმ ხელშეკრულებების რაოდენობას, სადაც გადასახდელი თანხა (ლარებში) აღემატება 3000 ლარს. (ExecuteScalar() მეთოდი).
3. შეადგინეთ პროგრამა, რომელიც გასცემს თანამშრომლების საშუალო ხელფასის მნიშვნელობას. (ExecuteScalar() მეთოდი).
4. შეადგინეთ პროგრამა, რომელიც თბილისელი თანამშრომლების ხელფასს გაზრდის 15%-ით UPDATE მოთხოვნის უშუალოდ შესრულების გზით. (ExecuteNonQuery() მეთოდი)
5. Shemkveti ცხრილს დაუმატეთ ახალი სტრიქონი INSERT მოთხოვნის უშუალოდ შესრულების გზით. (ExecuteNonQuery() მეთოდი).
6. Shemkveti ცხრილიდან სტრიქონები წაშალეთ იმ თანამშრომლების შესახებ, რომლებიც ქალაქ თბილისში ცხოვრობენ, DELETE მოთხოვნის უშუალოდ შესრულების გზით. (ExecuteNonQuery() მეთოდი).

ლაბ #14
განსაკუთრებული სიტუაციები

განსაკუთრებული სიტუაცია არის სიტუაცია, რომელიც გამოწვეულია პროგრამის შესრულების დროს აღძრული შეცდომის მიერ. ასეთი სიტუაციებია: ნულზე გაყოფა, გადავსება, მასივის ინდექსის გასვლა დიაპაზონის გარეთ, არარსებული ფაილის გახსნის მცდელობა და ა.შ. ეს შეცდომები იწვევენ პროგრამის შესრულების ავარიულად დამთავრებას. C# ენაში არსებობს განსაკუთრებული სიტუაციების დამუშავების საშუალებები, რომელთა გამოყენებით შესაძლებელია ასეთი შეცდომების დამუშავება ისე, რომ არ შეფერხდეს პროგრამის შესრულება.

C# ენაში განსაკუთრებული სიტუაციები წარმოდგენილია კლასებით. განსაკუთრებული სიტუაციების ყველა კლასი მემკვიდრეობით არის მიღებული განსაკუთრებული სიტუაციის Exception კლასისგან: Divide ByZeroException, IndexOutOfRangeException, OverflowException და ა.შ. Exception კლასის თვისებებია: Message (შეიცავს განსაკუთრებული სიტუაციის აღწერას), Source (შეიცავს განსაკუთრებული სიტუაციის გამომწვევი პროგრამის სახელს) და ა.შ. განსაკუთრებული სიტუაციების დამუშავება სრულდება ოთხი საკვანძო სიტყვის გამოყენებით: try, catch, throw და finally. ისინი, ხშირ შემთხვევაში, ერთობლივად გამოიყენება. try ბლოკში მოთავსებულია პროგრამის ის ოპერატორები, რომელთა შესრულებასაც თვალყური უნდა ვადევნოთ. გენერირებულ განსაკუთრებულ სიტუაციას იჭერს და ამუშავებს catch ბლოკი. throw სიტყვა იწვევს განსაკუთრებული სიტუაციის ხელოვნურად გენერირებას. finally ბლოკში მოთავსებულია კოდი, რომელიც ყოველთვის შესრულდება try ბლოკიდან გამოსვლისას.

try და catch ბლოკები

როგორც ცნობილია, როცა ხდება ინდექსთან მიმართვა, რომელიც გადის მასივის საზღვრებს გარეთ, აღიძვრება შეცდომა. ამ დროს გენერირდება განსაკუთრებული სიტუაცია. მოყვანილი პროგრამა ახდენს ამ განსაკუთრებული სიტუაციის გენერირებისა და დაჭერის დემონსტრირებას.

```
{
// პროგრამაში ხდება მასივის საზღვრებს გარეთ ინდექსის გასვლის შეცდომის
დამუშავება
int[] masivi = new int[5];
try
{
label1.Text = "ეს სტრიქონი გამოჩნდება განსაკუთრებული სიტუაციის გენერირებამდე";
masivi[6] = 20; // ინდექსის მნიშვნელობა გადის დიაპაზონის გარეთ
label2.Text = "ეს სტრიქონი არ გამოჩნდება";
}
// განსაკუთრებული სიტუაციის დაჭერა
catch (IndexOutOfRangeException)
{
label3.Text = "ინდექსი დიაპაზონის გარეთაა ";
}
label4.Text = "ეს ოპერატორი შესრულდება";
```

}

ამ პროგრამაში განსაკუთრებული სიტუაციის გენერირების შემთხვევაში მისი დაჭერა ხდება catch ოპერატორის მიერ. ამ მომენტიდან დაწყებული მუშაობას იწყებს catch ბლოკი, ხოლო try ბლოკი კი მუშაობას ამთავრებს. შედეგად,
label3.Text = "ეს სტრიქონი არ გამოჩნდება";

ოპერატორი, რომელიც მოსდევს ინდექსის არასწორად გამოყენების ოპერატორს, არ შესრულდება. catch ბლოკის შესრულების შემდეგ მართვა გადაეცემა ოპერატორს, რომელიც მოსდევს ამ catch ბლოკს. ამრიგად, მიუხედავად აღძრული შეცდომისა, პროგრამა აგრძელებს მუშაობას და ავარიულად არ მთავრდება.

ქვემოთ მოყვანილ პროგრამაში ერთი მასივის ელემენტები იყოფა მეორე მასივის ელემენტებზე. ნულზე გაყოფის შემთხვევაში გენერირდება DivideByZeroException განსაკუთრებული სიტუაცია. პროგრამა ახდენს მის დამუშავებას გასცემს რა შესაბამის შეტყობინებას. ამრიგად, ნულზე გაყოფის შეცდომა არ გამოიწვევს პროგრამის ავარიულ გაჩერებას. ამის ნაცვლად, label კომპონენტში გამოჩნდება შესაბამისი შეტყობინება.

```
{
// პროგრამაში ხდება ნულზე გაყოფის შეცდომის დაჭერა მასივების გაყოფის დროს
int[] masivi1 = { 1, 3, 5, 7, 9 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[5];
label1.Text = ""; label2.Text = "";
for ( int ind = 0; ind < masivi1.Length; ind++ )
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text += masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
catch ( DivideByZeroException )
{
label2.Text += "ადგილი აქვს ნულზე გაყოფას!" + '\n';
}
}
```

ამ პროგრამაში, try ბლოკი მოთავსებულია ციკლის ტანში, ამიტომ, შესაძლებელი ხდება გამეორებადი შეცდომების დამუშავება.

მოყვანილ პროგრამაში სრულდება ფაილის გახსნისა და ფაილიდან წაკითხვის ოპერაციების შემოწმება.

```
{
// პროგრამაში ხდება ფაილის გახსნისა და ფაილიდან წაკითხვის ოპერაციების
// მონიტორინგი
int ricxvi = 0;
FileStream file1;
// try ბლოკი ამოწმებს ფაილის გახსნის ოპერაციას
try
{
file1 = new FileStream("filetext.txt", FileMode.Open);
```

```
}
catch ( FileNotFoundException arg1 )
{
label1.Text = arg1.Message;
return;
}
// მონაცემების წაკითხვა ფაილიდან მანამ, სანამ არ შეგვხვდება EOF სიმბოლო
for ( ; ricxvi != -1; ) // თუ ricxvi = -1, მაშინ მიღწეულია ფაილის დასასრული
{ // try ბლოკი ამოწმებს ფაილიდან წაკითხვის ოპერაციას
try
{
ricxvi = file1.ReadByte();
}
catch ( IOException arg1 )
{
label2.Text = arg1.Message;
return;
}
if ( ricxvi != -1 ) label3.Text += ( char ) ricxvi;
}
file1.Close();
}
```

მოყვანილ პროგრამაში სრულდება ფაილის შექმნისა და ფაილში ჩაწერის ოპერაციების შემოწმება.

```
{
// პროგრამაში ხდება ფაილის შექმნისა და ფაილში ჩაწერის ოპერაციების მონიტორინგი
BinaryReader file_in;
BinaryWriter file_out;
int ricxvii1 = 10, ricxvii2;
double wiladi1 = 1001.47, wiladi2, wiladi3;
bool b1 = true, b2;
// try ბლოკი ამოწმებს ფაილის შექმნის ოპერაციას
try
{
file_out = new BinaryWriter(new FileStream("file1.dat", FileMode.Create));
}
catch ( IOException arg1 )
{
MessageBox.Show(arg1.Message + "\n შეუძლებელია ფაილის შექმნა");
return;
}
// try ბლოკი ამოწმებს ფაილში ჩაწერის ოპერაციებს
try
{
// ფაილში მთელი რიცხვის ჩაწერა
```

```
file_out.Write(ricxvii1);
// ფაილში წილადის ჩაწერა
file_out.Write(wiladi1);
// ფაილში ლოგიკური მონაცემის ჩაწერა
file_out.Write(b1);
// ფაილში იწერება 10.2 * 2.3 გამოსახულების გამოთვლის შედეგი
file_out.Write(10.2 * 2.3);
}
catch ( IOException arg1 )
{
MessageBox.Show(arg1.Message + "\n ჩაწერის შეცდომა");
return;
}
file_out.Close();
// ფაილიდან წაკითხვა
try
{
file_in = new BinaryReader(new FileStream("file1.dat",FileMode.Open));
}
catch ( IOException arg1 )
{
MessageBox.Show(arg1.Message + "\n შეუძლებელია ფაილის გახსნა");
return;
}
try
{
// მთელი რიცხვის წაკითხვა ფაილიდან
ricxvii2 = file_in.ReadInt32();
label1.Text = ricxvii2.ToString();
// წილადის წაკითხვა ფაილიდან
wiladi2 = file_in.ReadDouble();
label2.Text = wiladi2.ToString();
// ლოგიკური მონაცემის წაკითხვა ფაილიდან
b2 = file_in.ReadBoolean();
label3.Text = b2.ToString();
// წილადის წაკითხვა ფაილიდან
wiladi3 = file_in.ReadDouble();
label4.Text = wiladi3.ToString();
}
catch ( IOException arg1 )
{
MessageBox.Show(arg1.Message + "\n წაკითხვის შეცდომა");
return;
}
file_in.Close();
```

```
}
```

რამდენიმე catch ბლოკის გამოყენება

ერთ try ბლოკთან შეიძლება დაკავშირებული იყოს რამდენიმე catch ბლოკი. მაგრამ, თითოეული catch ბლოკი იჭერს კონკრეტული ტიპის განსაკუთრებული სიტუაციას. მოყვანილ პროგრამაში ხდება ინდექსის მნიშვნელობის დიაპაზონის გარეთ გასვლისა და ნულზე გაყოფის შეცდომების დაჭერა.

```
{
// პროგრამაში ხდება რამდენიმე catch ოპერატორის გამოყენების დემონსტრირება
label1.Text = "";
int[] masivi1 = { 1, 3, 5, 7, 9, 11, 13 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[7];
for ( int ind = 0; ind < masivi1.Length; ind++ )
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label1.Text += masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
// ეს catch ოპერატორი იჭერს ნულზე გაყოფის შეცდომას
catch ( DivideByZeroException )
{
label2.Text = "ადგილი აქვს ნულზე გაყოფას!";
}
// ეს catch ოპერატორი იჭერს მასივის ინდექსის დიაპაზონის გარეთ გასვლის შეცდომას
catch ( IndexOutOfRangeException )
{
label3.Text = "ინდექსი დიაპაზონის გარეთაა!";
}
}
```

ყველა განსაკუთრებული სიტუაციის დაჭერა

ზოგჯერ საჭიროა ყველა განსაკუთრებული სიტუაციის დაჭერა მათი ტიპის მიუხედავად. ასეთ შემთხვევაში, გამოიყენება catch ოპერატორი შეცდომის ტიპის მითითების გარეშე. ასეთი catch ოპერატორი იჭერს ყველა განსაკუთრებულ სიტუაციას. მოყვანილ პროგრამაში IndexOutOfRangeException და DivideByZeroException განსაკუთრებული სიტუაციის დაჭერა და დამუშავება სრულდება ერთი catch ოპერატორის მიერ.

```
{
label1.Text = ""; label2.Text = "";
int[] masivi1 = { 1, 3, 5, 7, 9, 11, 13 };
int[] masivi2 = { 0, 2, 4, 0, 3 };
int[] shedegi = new int[7];
for ( int ind = 0; ind < masivi1.Length; ind++ )
```

```
try
{
shedegi[ind] = masivi1[ind] / masivi2[ind];
label2.Text += masivi1[ind].ToString() + " / " +
masivi2[ind].ToString() + " = " + shedegi[ind].ToString() + '\n';
}
catch ( Exception arg )
{
label1.Text += arg.Message + '\n';
}
}
```

განსაკუთრებული სიტუაციის იძულებით გენერირება

განსაკუთრებული სიტუაციის გენერირება შეიძლება, აგრეთვე, throw ოპერატორის საშუალებით. მოყვანილ მაგალითში ხდება DivideByZeroException განსაკუთრებული სიტუაციის გენერირება throw ოპერატორის მიერ.

```
{
// პროგრამაში ხდება throw ოპერატორთან მუშაობის დემონსტრირება
int ricxvi1, ricxvi2, shedegi;
try
{
// ეს ოპერატორები შესრულდება throw ოპერატორის წინ
ricxvi1 = int.Parse(textBox1.Text);
ricxvi2 = int.Parse(textBox2.Text);
// განსაკუთრებული სიტუაციის გენერირება
if ( ricxvi2 == 0 ) throw new DivideByZeroException();
shedegi = ricxvi1 / ricxvi2;
label1.Text = shedegi.ToString();
}
catch ( DivideByZeroException )
{
label1.Text = "ადგილი აქვს ნულზე გაყოფას";
}
// try-catch ბლოკის დასასრული
}
```

finally ბლოკი

როგორც ვიცით, განსაკუთრებული სიტუაციის გენერირება იწვევს მიმდინარე მეთოდის შესრულების ავარიულ შეწყვეტას. მაგრამ, მეთოდმა შეიძლება გახსნას ფაილი ან ქსელური შეერთება, რომლებიც უნდა იყოს დახურული. ასეთ შემთხვევებში უნდა გამოვიყენოთ finally ბლოკი. ის ეთითება try/catch მიმდევრობის ბოლოში.

finally ბლოკი გამოიძახება იმისგან დამოუკიდებლად გენერირდება თუ არა განსაკუთრებული სიტუაცია. ამრიგად, არა აქვს მნიშვნელობა try ბლოკი როგორ დამთავრდება - ნორმალურად თუ აღიძვრება განსაკუთრებული სიტუაცია. finally

ბლოკის კოდი ყოველთვის შესრულდება. მოყვანილ პროგრამაში ხდება finally ბლოკის გამოყენების დემონსტრირება.

```
// პროგრამაში ხდება finally ბლოკის გამოყენების დემონსტრირება
class finally1
{
public static void gen_finally(int par1)
{
int ricxvi;
int[] masivi = new int[5];
MessageBox.Show(par1.ToString() + "-ის მიღება");
try
{
switch ( par1 )
{
case 0 : ricxvi = 10 / par1; // ნულზე გაყოფის შეცდომა
break;
case 1 : masivi[7] = 7; // დიაპაზონის გარეთ ინდექსის გასვლის შეცდომა
break;
case 2 : return; // try ბლოკიდან გამოსვლა
}
}
catch ( DivideByZeroException )
{
MessageBox.Show("ნულზე გაყოფის შეცდომა!");
return;
}
catch ( IndexOutOfRangeException )
{
MessageBox.Show(" დიაპაზონის გარეთ ინდექსის გასვლის შეცდომა!");
return;
}
finally
{
MessageBox.Show("try ბლოკიდან გამოსვლა");
}
}
}
private void button1_Click(object sender, System.EventArgs e)
{
for ( int indeqsi = 0; indeqsi < 3; indeqsi++ )
{
finally1.gen_finally(indeqsi);
}
}
```

სავარჯიშოები

1. შეადგინეთ პროგრამა, რომელიც შეკრებს ერთგანზომილებიანი მასივის ელემენტებს. ინდექსის არასწორი მნიშვნელობის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.

2. შეადგინეთ პროგრამა, რომელიც შეკრებს ორგანზომილებიანი მასივის ელემენტებს. რომელიმე ინდექსის არასწორი მნიშვნელობის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.

3. შეადგინეთ პროგრამა, რომელიც ერთი ორგანზომილებიანი მასივის ელემენტებს გაყოფს მეორე ორგანზომილებიანი მასივის შესაბამის ელემენტებზე. ნულზე გაყოფის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.

4. შეადგინეთ პროგრამა, რომელიც მთელრიცხვა ერთგანზომილებიანი მასივის ელემენტებს ფაილში ჩაწერს. შეტანა-გამოტანის შეცდომის აღძვრის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.

5. შეადგინეთ პროგრამა, რომელიც მთელრიცხვა ორგანზომილებიანი მასივის ელემენტებს ფაილიდან წაიკითხავს. შეტანა-გამოტანის შეცდომის აღძვრის შემთხვევაში გამოიტანეთ შესაბამისი შეტყობინება.

პრაქტიკული #13
ინტეგრირებული მოთხოვნების ენა LINQ

var საკვანძო სიტყვა

var საკვანძო სიტყვა გამოიყენება როგორც ტიპის ალტერნატივა. მისის სინტაქსია:

var *ცვლადის_სახელი = მნიშვნელობა;*

აქ *ცვლადის_სახელი* იღებს *მნიშვნელობის* ტიპს. მოყვანილი პროგრამით ხდება var საკვანძო სიტყვის გამოყენების დემონსტრირება:

```
{
var cvladi1 = 15;
int cvladi2 = 10;
var cvladi3 = cvladi2;
var cvladi4 = cvladi1 + cvladi2;
var cvladi5 = cvladi1 + 0.5;
label1.Text = cvladi1.ToString() + "    " + cvladi3.ToString() + "   " + cvladi2.ToString() + "   " +
cvladi4.ToString() + "   " + cvladi5.ToString();
var masivi1 = new[] { 1, 2, 3, 4, 5 };
}
```

პროგრამაში cvladi1 არის int ტიპის ცვლადი და არა var ტიპის. როცა ვიყენებთ var სიტყვას, ჩვენ არ ვქმნით უტიპო ცვლადს ან ტიპს, რომელიც შეიძლება შეიცვალოს. კომპილატორი თვითონ განსაზღვრავს ცვლადის ტიპს.

```
var cvladi1 = 15;
```

სტრიქონში cvladi1 ცვლადს ენიჭება int ტიპი, რადგან 15 მთელი რიცხვია.

```
int cvladi2 = 10;
var cvladi3 = cvladi2;
```

ამ მინიჭებების შედეგად, cvladi3 ცვლადს ენიჭება int ტიპი, რადგან cvladi2 ცვლადის ტიპია int. თუ cvladi2 ცვლადს არ მივანიჭებთ საწყის მნიშვნელობას, მაშინ კომპილატორი cvladi3 ცვლადს ვერ მიანიჭებს საწყის მნიშვნელობას და გასცემს შეტყობინებას შეცდომის შესახებ.

```
var cvladi5 = cvladi1 + 0.5;
```

სტრიქონში cvladi5 ცვლადს ენიჭება double ტიპი, რადგან cvladi1 + 0.5 გამოსახულების მნიშვნელობას წილადი ტიპი აქვს.

```
var masivi1 = new[] { 1, 2, 3, 4, 5 };
```

სტრიქონში masivi1 ცვლადს ენიჭება int[] ტიპი. ამ შემთხვევაში, მასივის ყველა ელემენტს ერთნაირი ტიპი უნდა ჰქონდეს, ან ჰქონდეს მიმართვითი ტიპი ან null, ან უნდა შეიძლებოდეს თითოეული მათგანის დაყვანა ერთ ტიპზე. მაგალითად, კომპილატორი შეცდომას გასცემს შემდეგი კოდის კომპილირებისას:

```
var masivi1 = new[] { 5, „რომან სამხარაძე”, 15 };          // „რომან სამხარაძე”-ის ნაცვლად უნდა
                                                           // იყოს მთელი რიცხვი
var masivi2 = new[] { 5, null, 15 };                       // null-ის ნაცვლად უნდა იყოს მთელი
                                                           // რიცხვი
```

არ არის რეკომენდებული var სიტყვის გამოყენება როგორც კლასის სახელი, რადგან ასეთ შემთხვევაში ის თავის სტანდარტულ ფუნქციას ვეღარ შეასრულებს.

LINQ მოთხოვნის სინტაქსი

ინტეგრირებული მოთხოვნების ენა (LINQ, Language-Integrated Query) არის C# ენის გაფართოება, რომელიც ინტეგრირებულია ამ ენაში. ის არის დიდი მოცულობის მონაცემებთან მუშაობის მოხერხებული, სწრაფი და ეფექტური მექანიზმი. LINQ გვათავისუფლებს მონაცემების გაფილტვრისა და დახარისხების რთული და გრძელი კოდების წერისგან. ის არის მოთხოვნების ენა, რომელიც საშუალებას გვაძლევს მიღებული შედეგები ადვილად დავახარისხოთ, გავფილტროთ და შევასრულოთ გამოთვლები. LINQ საშუალებას გვაძლევს ეფექტურად ვიმუშაოთ დიდი ზომის მონაცემთა ბაზებთან და კომპლექსურ XML დოკუმენტებთან, რომლებშიც მილიონზე მეტი ჩანაწერია.

LINQ ტექნოლოგია შეგვიძლია გამოვიყენოთ მონაცემთა Object, SQL და XML ტიპების მიმართ:

1. წარმოქმნის მოთხოვნებს, რომლებიც გამოიყენება მასივების, სიებისა და სხვა კოლექციების მიმართ.
2. წარმოქმნის მოთხოვნებს, რომლებიც გამოიყენება მონაცემთა რელაციური ბაზების მიმართ, რომლებიც იყენებენ სტანდარტულ SQL ენას. ასეთია Microsoft SQL Server, Oracle და ა.შ.
3. წარმოქმნის მოთხოვნებს, რომლებიც გამოიყენება XML ობიექტებთან სამუშაოდ.

LINQ მოთხოვნა ოთხი ნაწილისგან შედგება: var, from, where და select. მისი სინტაქსია:

var *შედეგი* =
from *ცვლადი* **in** *კოლექცია*
[**where** *პირობა*]
select *ცვლადი*;

var განყოფილებაში მითითებულია ცვლადი, რომელიც შეიცავს მოთხოვნის მიერ გაცემულ შედეგს. from განყოფილებაში ეთითება დასამუშავებელი კოლექცია ან მასივი. კოლექცია არ უნდა შეიცავდეს მხოლოდ ერთ ობიექტს ან ცვლადს. where განყოფილებაში ეთითება პირობა ჩვენი მოთხოვნისთვის. ამ განყოფილების მითითება აუცილებელი არ არის. select განყოფილებაში მითითებული ცვლადის მნიშვნელობა უნდა გამოჩნდეს შედეგობრივ ნაკრებში.

იმისთვის, რომ შევძლოთ LINQ მოთხოვნების შესრულება, using დირექტივების ბლოკში უნდა მოვათავსოთ **using System.Linq;** დირექტივა. ჩვეულებრივ, ამას Visual Studio ავტომატურად აკეთებს.

განვიხილოთ LINQ მოთხოვნის გამოყენების მარტივი მაგალითი. მოყვანილი პროგრამით ხდება სტრიქონების მასივიდან იმ სტრიქონების ამორჩევა, რომლებიც „ს“ ასოთი იწყება. მოცემული პროგრამა დაწერილია LINQ *მოთხოვნის სინტაქსის* გამოყენებით.

```
{
string[] saxelebi = { "რომანი", "ანა", "გიორგი", "არჩილი", "სანდრო", "დავითი", "ვახტანგი",
"ბექა", "ალექსანდრე", "საბა", "სიმონი", "ლია", "ლუკა" };
//      LINQ მოთხოვნის ფორმირება
var shedegi =
from cvladi in saxelebi
where cvladi.StartsWith("ს")
select cvladi;
```

```
//      შედეგების ეკრანზე გამოტანა
label1.Text = "სახელები, რომლებიც იწყება „ს“ ასოთი:\n";
foreach ( var elementi in shedegi )
        label1.Text += elementi.ToString() + "\n";
}
```

როგორც პროგრამიდან ჩანს, LINQ მოთხოვნის მიერ გაცემული სტრიქონები მოთავსთდება shedegi ცვლადში. saxelebi მასივიდან ამორჩეული თითოეული სტრიქონი შემოწმდება იწყება თუ არა ის „ს“ ასოთი. თუ კი, მაშინ ის ამოირჩევა და მოთავსთდება shedegi ცვლადში. შედეგად, label1 კომპონენტში გამოჩნდება ის სტრიქონები, რომლებიც „ს“ ასოთი იწყება.

LINQ მოთხოვნის მიერ გაცემული შედეგების დასახარისხებლად გამოიყენება orderby განყოფილება. მოყვანილი პროგრამით ხდება მოთხოვნის მიერ გაცემული შედეგების დახარისხება ზრდადობით.

```
{
string[] saxelebi = { "რომანი", "ანა", "გიორგი", "არჩილი", "სანდრო", "დავითი", "ვახტანგი",
"ბექა",
                        "ალექსანდრე", "საბა", "სიმონი", "ლია", "ლუკა" };
//      LINQ მოთხოვნის ფორმირება
var shedegi = from cvladi in saxelebi
where cvladi.StartsWith("ს")
orderby cvladi
select cvladi;
//      შედეგების ეკრანზე გამოტანა
label1.Text = "ზრდადობით დალაგებული სახელები, რომლებიც იწყება ‘ს’ ასოთი:\n";
foreach (var elementi in shedegi)
        label1.Text += elementi.ToString() + "\n";
}
```

იმისთვის, რომ კლებადობით დავახარისხოთ მოთხოვნის მიერ გაცემული შედეგები, LINQ მოთხოვნა ასე უნდა ჩავწეროთ:

```
var shedegi = from cvladi in saxelebi where cvladi.StartsWith("ს")
orderby cvladi descending select cvladi;
```

თუ გვინდა, რომ სახელები დავახარისხოთ უკანასკნელი ასოების მიხედვით, მაშინ LINQ მოთხოვნა ასე უნდა ჩავწეროთ:

```
var shedegi = from cvladi in saxelebi where cvladi.StartsWith("ს")
orderby cvladi.Substring(cvladi.Length - 1) select cvladi;
```

მონაცემთა დიდი ზომის ნაკრებთან მუშაობა

LINQ მოთხოვნა შეგვიძლია, აგრეთვე გამოვიყენოთ დიდი ზომის რიცხვითი მასივიდან 5500-ზე ნაკლები მნიშვნელობის მქონე რიცხვების ამოსარჩევად. მოყვანილი პროგრამით ხდება ამის დემონსტრირება.

```
{
label1.Text = "";
Random Shemtxveviti_Ricxvebi = new Random();
int[] masivi = new int[1000000];
for ( int i = 0; i < masivi.Length; i++ )
```

```
masivi[i] = Shemtxveviti_Ricxvebi.Next();
//        LINQ მოთხოვნის ფორმირება
var shedegi =
from cvladi in masivi
where cvladi < 5500
select cvladi;
//        შედეგების ეკრანზე გამოტანა
label1.Text = "5500-ზე ნაკლები რიცხვები:\n";
foreach (var elementi in shedegi)
        label1.Text += elementi.ToString() + "   ";
}
```

ძირითად პროგრამაში ხდება 1000000 შემთხვევითი რიცხვის წარმოიქმნა, რომლებიც ჩაიწერება masivi მასივში. LINQ მოთხოვნის გამოყენებით ამ მასივიდან ამოირჩევა ის რიცხვები, რომლებიც 5500-ზე ნაკლებია.

LINQ მოთხოვნის გამოყენებით შეიძლება მასივიდან კენტი რიცხვების ამორჩევა. ამის დემონსტრირება ხდება მოყვანილი პროგრამით.

```
{
label1.Text = "";
int[] numbers = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
//        LINQ მოთხოვნის ფორმირება
var numQuery =
from num in numbers
where ( num % 2 == 1 )
select num;
 //        შედეგების ეკრანზე გამოტანა
foreach (int num in numQuery)
        label1.Text += num.ToString() + "  ";
}
```

თუ გვინდა კენტი რიცხვების ამორჩევა 5-15 დიაპაზონში, მაშინ მოთხოვნა ასე უნდა ჩავწეროთ:

```
where ( (num % 2 == 1 ) && ( num >= 5 && num <= 15) )
```

აგრეგირების ოპერატორები

LINQ მოთხოვნას აქვს აგრეგირების ოპერატორები, რომლებიც იძლევიან შედეგის ანალიზის საშუალებას. ხშირად გამოყენებადი ოპერატორებია:

Average() – გასცემს შედეგში მოთავსებული რიცხვების საშუალო არითმეტიკულს;
Count() – გასცემს მონაცემების რაოდენობას შედეგში;
Max() – გასცემს შედეგში მოთავსებული მონაცემების მაქსიმალურ მნიშვნელობას;
Min() – გასცემს შედეგში მოთავსებული მონაცემების მინიმალურ მნიშვნელობას;
Sum() - გასცემს შედეგში მოთავსებული რიცხვების ჯამს.

ამ ოპერატორების მუშაობის დემონსტრირება ხდება მოყვანილი პროგრამით.

```
Random Shemtxveviti_Ricxvebi = new Random();
int[] masivi = new int[12345678];
for ( int i = 0; i < masivi.Length; i++ )
masivi[i] = Shemtxveviti_Ricxvebi.Next();
```

```
//        LINQ მოთხოვნის ფორმირება
var shedegi =
from cvladi in masivi
where cvladi > 9000
select cvladi;
//        შედეგების ეკრანზე გამოტანა
label1.Text = "იმ რიცხვების რაოდენობა, რომელთა მნიშვნელობები > 9000" + " - ";
label1.Text += shedegi.Count().ToString() + '\n';
label1.Text += "იმ რიცხვებს შორის მაქსიმალური, რომელთა მნიშვნელობები > 9000" + " -
";
label1.Text += shedegi.Max().ToString() + '\n';
label1.Text += "მინიმალური რიცხვი, რომელიც აღემატება 9000-ს" + " - ";
label1.Text += shedegi.Min().ToString() + '\n';
label1.Text += "იმ რიცხვების საშუალო არითმეტიკული, რომელთა მნიშვნელობები >
9000" + " - ";
label1.Text += shedegi.Average().ToString() + '\n';
label1.Text += "იმ რიცხვების ჯამი, რომელთა მნიშვნელობები > 9000" + " - ";
label1.Text += shedegi.Sum(cvladi => (long) cvladi);
}
```

რთულ ობიექტებთან მუშაობა

ამ განყოფილებაში ვნახავთ თუ როგორ შეიძლება LINQ მოთხოვნის გამოყენება რთული ობიექტების მიმართ. მაგალითისთვის შევქმნათ სტუდენტის კლასი.

```
class Studenti1
{
public string gvari;
public int asaki;
public double tanxa;
public int kursi;
public string fakulteti;
public Studenti1(string par1, int par2, double par3, int par4, string par5)
{
gvari = par1;
asaki = par2;
tanxa = par3;
kursi = par4;
fakulteti = par5;
}
}
private void button1_Click(object sender, EventArgs e)
{
Studenti1[] obj = new Studenti1[10];
obj[0] = new Studenti1("სამხარაძე", 21, 1200.50, 4, "ინფორმატიკის");
obj[1] = new Studenti1("კაპანაძე", 20, 1250.50, 4, "ენერგეტიკის");
obj[2] = new Studenti1("კირვალიძე", 18, 1300.50, 4, "ინფორმატიკის");
```

```
obj[3] = new Studenti1("ჭუმბურიძე", 19, 1100.50, 4, "ინფორმატიკის");
obj[4] = new Studenti1("ხუციშვილი", 20, 1400.50, 4, "სამშენებლო");
obj[5] = new Studenti1("კიკნაძე", 21, 1200.50, 4, "ინფორმატიკის");
obj[6] = new Studenti1("ნონიაშვილი", 20, 1250.50, 4, "ენერგეტიკის");
obj[7] = new Studenti1("ქევხიშვილი", 18, 1300.50, 4, "ინფორმატიკის");
obj[8] = new Studenti1("ხოშტარია", 19, 1100.50, 4, "ინფორმატიკის");
obj[9] = new Studenti1("ქარცივაძე", 20, 1400.50, 4, "სამშენებლო");
//        LINQ მოთხოვნის ფორმირება
var shedegi =
from cvladi in obj
where cvladi.fakulteti == "ინფორმატიკის"
select cvladi;
//        ეკრანზე შედეგების გამოტანა
label1.Text = "ინფორმატიკის ფაკულტეტის სტუდენტები:\n
                გვარი        ფაკულტეტი  ასაკი  კურსი  თანხა\n";
foreach (Studenti1 cvladi in shedegi)
   label1.Text += cvladi.gvari + "   " + cvladi.fakulteti + "   " + cvladi.asaki.ToString() + "   " +
                cvladi.kursi.ToString() + "   " + cvladi.tanxa.ToString() + "\n";
}
```

თითოეული ობიექტის შესაქმნელად კონსტრუქტორის გამოძახების ნაცვლად უფრო სწრაფი და მარტივია List<T> ტიპის გამოყენება. მისი საშუალებით შეგვიძლია შევქმნათ Studenti ტიპის ობიექტების კოლექცია.

```
class Studenti
{
public string gvari;
public int asaki;
public double tanxa;
public int kursi;
public string fakulteti;
}
private void button1_Click(object sender, EventArgs e)
{
List<Studenti> Studentebi = new List<Studenti>
{
new Studenti { gvari ="სამხარაძე", asaki = 21, tanxa = 1600, kursi = 4, fakulteti =
"ინფორმატიკის" },
new Studenti { gvari ="კაპანაძე", asaki = 18, tanxa = 1770.50, kursi = 3, fakulteti =
"ენერგეტიკის" },
new Studenti { gvari ="ქევხიშვილი", asaki = 20, tanxa = 1830.50, kursi = 1, fakulteti =
"სამშენებლო" },
new Studenti { gvari
="ყალაბეგიშვილი",asaki=19,tanxa=2150.00,kursi=2,fakulteti="ჰუმანიტარული"},
new Studenti { gvari ="ხოშტარია", asaki = 21, tanxa = 2050.50, kursi = 2, fakulteti =
"ინფორმატიკის" },
```

```
new Studenti { gvari ="მამაიაშვილი", asaki = 19, tanxa = 1950.50, kursi
=3,fakulteti="ინფორმატიკის"},
new Studenti { gvari ="ნონიაშვილი", asaki = 18, tanxa = 1900.00, kursi = 4, fakulteti =
"სამშენებლო" },
new Studenti { gvari ="ასანიძე", asaki = 20, tanxa = 1500.50, kursi = 4, fakulteti =
"ჰუმანიტარული" },
new Studenti { gvari ="ჭუმბურიძე", asaki = 19, tanxa = 2000.50, kursi = 4, fakulteti =
"ენერგეტიკის" },
new Studenti { gvari ="კირვალიძე", asaki = 18, tanxa =
1275.00,kursi=2,fakulteti="ინფორმატიკის"}
};
//        LINQ მოთხოვნის ფორმირება
var shedegi =
from cvladi in Studentebi
where cvladi.fakulteti == "ინფორმატიკის"
select cvladi;
//        ეკრანზე შედეგების გამოტანა
label1.Text = "ინფორმატიკის ფაკულტეტის სტუდენტები:\n
                    გვარი        ფაკულტეტი  ასაკი  კურსი  თანხა\n";
foreach (Studenti cvladi in shedegi)
   label1.Text += cvladi.gvari + "   " + cvladi.fakulteti + "   " + cvladi.asaki.ToString() + "   " +
                     cvladi.kursi.ToString() + "   " + cvladi.tanxa.ToString() + "\n";
}
```

მოთხოვნაში ახალი ობიექტის შექმნა

LINQ მოთხოვნაში ჩვენ Studentebi სიიდან შეგვიძლია ამოვირჩიოთ მხოლოდ ერთი, რომელიმე ველი, მაგალითად gvari. ამისთვის მოთხოვნა ასე უნდა ჩავწეროთ:

```
var shedegi =
from cvladi in Studentebi
where cvladi.fakulteti == "ინფორმატიკის"
select cvladi.gvari;
```

თუ გვინდა რამდენიმე ველის ამორჩევა, მაშინ select განყოფილებაში უნდა მივუთითოთ new საკვანძო სიტყვა. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
class Studenti
{
   public string gvari;
   public int asaki;
   public double tanxa;
   public int kursi;
   public string fakulteti;
}
private void button1_Click(object sender, EventArgs e)
{
label1.Text = "";
List<Studenti> Studentebi = new List<Studenti>
```

```
{
new Studenti { gvari ="სამხარაძე", asaki = 21, tanxa = 1600, kursi = 4, fakulteti =
"ინფორმატიკის" },
new Studenti { gvari ="კაპანაძე", asaki = 18, tanxa = 1770.50, kursi = 3, fakulteti =
"ენერგეტიკის" },
new Studenti { gvari ="ქევხიშვილი", asaki = 20, tanxa = 1830.50, kursi = 1, fakulteti =
"სამშენებლო" },
new Studenti { gvari
="ყალაბეგიშვილი",asaki=19,tanxa=2150.00,kursi=2,fakulteti="ჰუმანიტარული"},
new Studenti { gvari ="ხოშტარია", asaki = 21, tanxa = 2050.50, kursi = 2, fakulteti =
"ინფორმატიკის" },
new Studenti { gvari ="ქარცივაძე", asaki = 19, tanxa = 1950.50, kursi
=3,fakulteti="ინფორმატიკის"},
new Studenti { gvari ="ნონიაშვილი", asaki = 18, tanxa = 1900.00, kursi = 4, fakulteti =
"სამშენებლო" },
new Studenti { gvari ="გასიტაშვილი", asaki = 20, tanxa = 2150.50,
kursi=1,fakulteti="ინფორმატიკის"},
new Studenti { gvari ="კიკნაძე", asaki = 20, tanxa = 1500.50, kursi = 4, fakulteti =
"ჰუმანიტარული" },
new Studenti { gvari ="სიდამონიძე", asaki = 21, tanxa = 1700.00, kursi =
3,fakulteti="ინფორმატიკის"},
new Studenti { gvari ="ჭუმბურიძე", asaki = 19, tanxa = 2000.50, kursi = 4, fakulteti =
"ენერგეტიკის" },
new Studenti { gvari ="ბარბაქაძე", asaki = 19, tanxa = 2200.50, kursi = 3, fakulteti =
"ჰუმანიტარული"},
new Studenti { gvari ="ბერძენიშვილი", asaki = 18, tanxa =
1275.00,kursi=2,fakulteti="ინფორმატიკის"}
};
//        LINQ მოთხოვნის ფორმირება
var shedegi = from cvladi in Studentebi where cvladi.fakulteti == "ინფორმატიკის"
select new { cvladi.gvari, cvladi.tanxa, cvladi.asaki };
//        ეკრანზე შედეგების გამოტანა
foreach (var elementi in shedegi)
   label1.Text += elementi.ToString() + "\n";
}
```

ADO.NET და LINQ

LINQ მოთხოვნები შეგვიძლია, აგრეთვე გამოვიყენოთ ADO.NET-ის მიმართ. ასეთი მიდგომა სასარგებლოა მაშინ, როცა პროგრამას ჭირდება შეასრულოს გაფართოებული მანიპულაციები გამორთულ მონაცემთა ნაკრებთან. გარდა ამისა, LINQ უმატებს მოთხოვნების ოპერატორების სრულ ნაკრებს იმ მოთხოვნების შეზღუდულ შესაძლებლობებს, რომლებიც ჩადგმულია DataSet საწყის კლასში.

დავუშვათ, გვაინტერესებს იმ თანამშრომლების გვარები, რომელთა ასაკი აღემატება 30 წელს. პროგრამას ექნება სახე:

```
{
```

```
label1.Text = "";
string conn_string = "server=ROMANI\\SQL_2012;database=Shekveta;Integrated Security=true";
        SqlConnection conn_Shekveta = new SqlConnection(conn_string);
        SqlCommand comm_1 = conn_Shekveta.CreateCommand();
        comm_1.CommandText = "SELECT * FROM Personali";
        SqlDataAdapter adap_Personali = new SqlDataAdapter();
        adap_Personali.SelectCommand = comm_1;
        DataSet ds_1 = new DataSet();
        adap_Personali.Fill(ds_1, "Personali");
        conn_Shekveta.Close();
        dataGridView1.DataSource = ds_1;
        dataGridView1.DataMember = "Personali";
        //
        var Personalis = ds_1.Tables["Personali"].AsEnumerable();
        var result_Personali = from variable in Personalis
         where int.Parse(variable["asaki"].ToString()) > 30
         //where DateTime.Parse(variable["tarigi_dabadebis"].ToString()) >
                                DateTime.Parse("2003-01-01")  //+dateTimePicker1.Value
      //where double.Parse(variable["xelfasi"].ToString()) > 1500.43
      //where variable["department"].ToString() == "sport"
            orderby variable["gvari"] descending
            select variable;
        //
        foreach (var variable_1 in result_Personali)
           label1.Text += variable_1["gvari"] + "    " +
              variable_1["asaki"].ToString() + "    " +
              variable_1["xelfasi"].ToString() + "    " +
              variable_1["tarigi_dabadebis"].ToString() + "\n";
}
```

დავუშვათ, გვაინტერესებს ის თანამშრომლები, რომელთაც ერთზე მეტი შემკვეთი ჰყავთ. პროგრამას ექნება სახე:

```
{
label1.Text = "";
string conn_string = "server=ROMANI\\SQL_2012;database=Shekveta;Integrated Security=true";
        SqlConnection conn_Shekveta = new SqlConnection(conn_string);
        SqlCommand comm_1 = conn_Shekveta.CreateCommand();
SqlDataAdapter adap_Personali = new SqlDataAdapter("SELECT * FROM Personali", conn_Shekveta);
SqlDataAdapter adap_Shemkveti = new SqlDataAdapter("SELECT * FROM Shemkveti", conn_Shekveta);
        DataSet ds_1 = new DataSet();
        adap_Personali.Fill(ds_1, "Personali");
        adap_Shemkveti.Fill(ds_1, "Shemkveti");
        conn_Shekveta.Close();
        dataGridView1.DataSource = ds_1;
        dataGridView1.DataMember = "Personali";
        //
DataColumn parent_column = ds_1.Tables["Personali"].Columns["Personali_ID"];
DataColumn child_column = ds_1.Tables["Shemkveti"].Columns["Personali_ID"];
DataRelation Personali_Shemkveti = new DataRelation("Personali_Shemkveti", parent_column,
child_column);
```

```
        ds_1.Relations.Add(Personali_Shemkveti);
        //
        BindingSource bind_Personali = new BindingSource(ds_1, "Personali");
        BindingSource bind_Shemkveti = new BindingSource(ds_1, "Shemkveti");
        bind_Shemkveti.DataSource = bind_Personali;
        bind_Shemkveti.DataMember = "Personali_Shemkveti";
        dataGridView1.DataSource = bind_Personali;
        dataGridView2.DataSource = bind_Shemkveti;
        //
        var Parents = ds_1.Tables["Personali"].AsEnumerable();
        var Childs = ds_1.Tables["Shemkveti"].AsEnumerable();
        var result = from variable in Parents
                     where variable.GetChildRows("Personali_Shemkveti").Length > 1
                     orderby variable.GetChildRows("Personali_Shemkveti").Length
                     select variable;
        //
        foreach (var variable_1 in result)
        {
           label1.Text += variable_1["gvari"] + "   " +
           variable_1["asaki"].ToString() + "   " +
           variable_1["xelfasi"].ToString() + "   " +
           variable_1["tarigi_dabadebis"].ToString() + "\n";
           label1.Text += "Number of Shemkvetis = " +
              variable_1.GetChildRows("Personali_Shemkveti").Length.ToString() + "\n";
        }
}
```

Personali_Shemkveti ობიექტი გადაეცემა GetChildRows() მეთოდს, რომელიც გასცემს DataRowCollection ობიექტს, რომელიც შეიცავს მოცემულ თანამშრომელთან დაკავშირებულ Shemkveti ცხრილის ობიექტებს.

სავარჯიშოები

1. შეადგინეთ LINQ მოთხოვნა, რომელიც სტრიქონების მასივიდან იმ სტრიქონებს ამოირჩევს, რომლებიც „ბ“ ასოთი იწყება.
2. შეადგინეთ LINQ მოთხოვნა, რომელიც სტრიქონების მასივს დახარისხებს ზრდადობით.
3. შეადგინეთ LINQ მოთხოვნა, რომელიც სტრიქონების მასივს დახარისხებს კლებადობით.
4. შეადგინეთ LINQ მოთხოვნა, რომელიც დიდი ზომის ერთგანზომილებიანი მასივიდან ამოირჩევს 5-ის ჯერად რიცხვებს. მასივის ზომაა 1000000 ელემენტი. მასივი შემთხვევითი რიცხვებით შეავსეთ.
5. შეადგინეთ LINQ მოთხოვნა, რომელიც დიდი ზომის ერთგანზომილებიანი მასივიდან ამოირჩევს 1234-ზე მეტ რიცხვებს. მასივის ზომაა 1000000 ელემენტი. მასივი შემთხვევითი რიცხვებით შეავსეთ.
6. შეადგინეთ LINQ მოთხოვნა, რომელიც დიდი ზომის ერთგანზომილებიანი მასივში დათვლის 12340-ზე მეტი რიცხვების რაოდენობას. მასივის ზომაა 1000000 ელემენტი. მასივი შემთხვევითი რიცხვებით შეავსეთ.
7. შეადგინეთ LINQ მოთხოვნა, რომელიც დიდი ზომის ერთგანზომილებიანი მასივში იპოვის 12340-ზე მეტი რიცხვების ჯამს. მასივის ზომაა 1000000 ელემენტი. მასივი შემთხვევითი რიცხვებით შეავსეთ.
8. შეადგინეთ LINQ მოთხოვნა, რომელიც დიდი ზომის ერთგანზომილებიანი მასივში იპოვის 12340-ზე მეტი რიცხვების საშუალო არითმეტიკულს. მასივის ზომაა 1000000 ელემენტი. მასივი შემთხვევითი რიცხვებით შეავსეთ.
9. შეადგინეთ LINQ მოთხოვნა, რომელიც დიდი ზომის ერთგანზომილებიანი მასივში იპოვის მინიმალურს. მასივის ზომაა 1000000 ელემენტი. მასივი შემთხვევითი რიცხვებით შეავსეთ.
10. შეადგინეთ LINQ მოთხოვნა, რომელიც დიდი ზომის ერთგანზომილებიანი მასივში იპოვის მაქსიმალურს. მასივის ზომაა 1000000 ელემენტი. მასივი შემთხვევითი რიცხვებით შეავსეთ.
11. List<T> ტიპის გამოყენებით შექმენით Avtomanqana კლასის ობიექტების კოლექცია, რომელიც 10 ობიექტისგან შედგება. Avtomanqana კლასის ველებია: მანქანის მარკა, გამოშვების წელი, ფასი, მწარმოებელი ქვეყანა. შემდეგ, LINQ მოთხოვნის გამოყენებით ეკრანზე გამოიტანეთ შემდეგი მონაცემები Toyota მარკის მანქანების შესახებ: მანქანის მარკა და ფასი.
12. List<T> ტიპის გამოყენებით შექმენით Avtomanqana კლასის ობიექტების კოლექცია, რომელიც 10 ობიექტისგან შედგება. Avtomanqana კლასის ველებია: მანქანის მარკა, გამოშვების წელი, ფასი, მწარმოებელი ქვეყანა. შემდეგ, LINQ მოთხოვნის გამოყენებით მონაცემები დაახარისხეთ ზრდადობით მანქანის მარკისა და ფერის მიხედვით. ეკრანზე გამოიტანეთ მანქანის მარკა და გამოშვების წელი.
13. List<T> ტიპის გამოყენებით შექმენით Mascavlebeli და Moscavle კლასის ობიექტების კოლექციები. თითოეული კოლექცია 10-10 ობიექტისგან შედგება. Mascavlebeli კლასი შემდეგ ველებს შეიცავს: გვარი, საგანი, ხელფასი. Moscavle კლასი შემდეგ ველებს შეიცავს: გვარი, კლასის ნომერი, საგანი, ქულა. შემდეგ, Intersect(), Except() და Union() მეთოდების გამოყენებით იპოვეთ: ა) ერთნაირი გვარის მქონე მასწავლებლები და მოსწავლეები; ბ) ის მასწავლებლები, რომელთა გვარები არ ემთხვევა მოსწავლეების გვარებს; გ) განსხვავებული გვარის მქონე მასწავლებლების და მოსწავლეების საერთო სია.
14. List<T> ტიპის გამოყენებით შექმენით Avtomanqana კლასის ობიექტების კოლექცია, რომელიც 10 ობიექტისგან შედგება. Avtomanqana კლასის ველებია: მანქანის მარკა, გამოშვების წელი, ფასი, მწარმოებელი ქვეყანა. შემდეგ, Take() და Skip() მეთოდების გამოყენებით ეკრანზე გამოიტანეთ მონაცემები: ა) პირველი ხუთი მანქანის შესახებ; ბ)

ყველა მანქანის შესახებ პირველი ექვსი მანქანის გარდა.

15. List<T> ტიპის გამოყენებით შექმენით Avtomanqana კლასის ობიექტების კოლექცია, რომელიც 10 ობიექტისგან შედგება. Avtomanqana კლასის ველებია: მანქანის მარკა, გამოშვების წელი, ფასი, მწარმოებელი ქვეყანა. შემდეგ, Distinct() მეთოდის გამოყენებით ეკრანზე გამოიტანეთ არაგამეორებადი მნიშვნელობები მანქანის მარკის მიხედვით.
16. შეადგინეთ პროგრამა, რომელიც ეკრანზე გამოიტანს იმ თანამშრომლების გვარებს, რომელთა ასაკი ნაკლებია 30 წელზე. გამოიყენეთ LINQ-მოთხოვნა.
17. შეადგინეთ პროგრამა, რომელიც ეკრანზე გამოიტანს ბათუმელი შემკვეთების გვარებს. გამოიყენეთ LINQ-მოთხოვნა.
18. შეადგინეთ პროგრამა, რომელიც ეკრანზე გამოიტანს gadasaxdeli_l სვეტის მნიშვნელობებს იმ ხელშეკრულებებისთვის, სადაც vali_l სვეტის მნიშვნელობა აღემატება 500.00. გამოიყენეთ LINQ-მოთხოვნა.

ლაბ #11
ADO.NET კლასები. მონაცემების შეცვლა, დამატება და წაშლა. რელაცია ცხრილებს შორის

ცხრილში მონაცემების შეცვლა

მოყვანილი პროგრამით ხდება Personali ცხრილის მე-4 სტრიქონში (მისი ინდექსია 3) asaki სვეტის მნიშვნელობის შეცვლა. ახალი მნიშვნელობა TextBox ელემენტიდან შეიტანეთ. SELECT ბრძანებაში პირველადი გასაღების (personaliID სვეტი) მითითება აუცილებელია:

```
{
// შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
SqlConnection("server=ROMANI-PC; database=Shekveta; Integrated Security = True");
// შეერთების გახსნა
myConnection.Open();
// ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM Personali",
myConnection);
// SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
// მონაცემების გამოტანა ცვლილებამდე
label1.Text = myDataset.Tables["Personali"].Rows[3]["asaki"].ToString() + "\n";
// Personalis ცხრილის მე-3 სტრიქონში ასაკის შეცვლა
myDataset.Tables["Personali"].Rows[3]["asaki"] = Convert.ToInt32(textBox1.Text);
// Update ბრძანების გამოძახება ცვლილებების დაფიქსირებისათვის ცხრილში
myAdapter.Update(myDataset, "Personali");
// ეკრანზე ცვლილებების გამოტანა
label2.Text = myDataset.Tables["Personali"].Rows[3]["asaki"].ToString();
// Personali ცხრილის სტრიქონების გამოტანა ეკრანზე
dataGridView1.DataSource = myDataset;
dataGridView1.DataMember = "Personali";
// შეერთების დახურვა
myConnection.Close();
}
```

ცხრილში სტრიქონის ჩამატება

მოყვანილი კოდით ხდება Personali ცხრილში სტრიქონის ჩამატება:

```
{
// შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
```

```
SqlConnection("server=ROMANI-PC; database=Shekveta; Integrated Security = True");
// შეერთების გახსნა
myConnection.Open();
// ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM Personali",
myConnection);
// SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
// სტრიქონების რაოდენობა ცვლილებამდე
label1.Text = myDataset.Tables["Personali"].Rows.Count.ToString() + "\n";
// ახალი სტრიქონის შექმნა
DataRow myRow = myDataset.Tables["Personali"].NewRow();
// ახალი სტრიქონის სვეტებისთვის მნიშვნელობების მინიჭება
myRow["gvari"] = textBox1.Text;
myRow["staji"] = textBox2.Text;
myRow["xelfasi"] = textBox3.Text;
myRow["tarigi_dabadebis"] = dateTimePicker1.Value.ToString();
myDataset.Tables["Personali"].Rows.Add(myRow);
// სტრიქონების რაოდენობა ცვლილების შემდეგ
label1.Text += myDataset.Tables["Personali"].Rows.Count.ToString() + "\n";
// Update ბრძანების გამოძახება ცვლილებების დაფიქსირებისათვის ცხრილში
myAdapter.Update(myDataset, "Personali");
// Personali ცხრილის სტრიქონების გამოტანა ეკრანზე
dataGridView1.DataSource = myDataset;
dataGridView1.DataMember = "Personali";
// შეერთების დახურვა
myConnection.Close();
}
```

სტრიქონების წაშლა

ცხრილიდან სტრიქონის წასაშლელად გამოიყენება DataRow ობიექტის Delete() მეთოდი. მოყვანილ პროგრამაში, ჯერ იძებნება წასაშლელი სტრიქონი, შემდეგ კი ხდება მისი წაშლა. Delete() მეთოდის შემდეგ აუცილებელია Update() მეთოდის შესრულება. საქმე ის არის, რომ Delete() მეთოდი არ შლის სტრიქონს, არამედ მონიშნავს მას შემდგომი წაშლისთვის. Update() მეთოდი კი ახდენს ცვლილებების რეალურად ფიქსირებას, ჩვენს შემთხვევაში, წასაშლელად მონიშნული სტრიქონის წაშლას:

```
{
// find_key ცვლადში უნდა მოვათავსოთ საძებნი მნიშვნელობა
int find_key = Convert.ToInt32(textBox1.Text);
// შეერთების ობიექტის შექმნა
SqlConnection myConnection =
```

```
new SqlConnection("server=ROMANI-PC; database=Shekveta; Integrated Security = True");
// შეერთების გახსნა
myConnection.Open();
// ადაპტერის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM Personali",
myConnection);
// SqlCommandBuilder ობიექტის შექმნა SQL ბრძანების ასაგებად
SqlCommandBuilder myBuilder = new SqlCommandBuilder(myAdapter);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
// პირველადი გასაღების ფორმირება
DataColumn[] keys = new DataColumn[1];
keys[0] = myDataset.Tables["Personali"].Columns["PersonaliID"];
myDataset.Tables["Personali"].PrimaryKey = keys;
// findRow ობიექტს ენიჭება ნაპოვნი სტრიქონი
DataRow findRow = myDataset.Tables["Personali"].Rows.Find(find_key);
// მოწმდება სტრიქონის არსებობა
if ( findRow != null )
{
// სტრიქონის წაშლა
findRow.Delete();
myAdapter.Update(myDataset, "Personali");
label1.Text = "სტრიქონი წაიშალა";
}
else label1.Text = "წასაშლელი სტრიქონი ვერ მოიძებნა";
// შეერთების დახურვა
myConnection.Close();
}
```

ორ ცხრილს შორის რელაციის შექმნა

ორ ცხრილს შორის ბმის შესაქმნელად გამოიყენება DataRelation ობიექტი. ის კავშირს ამყარებს DataTable ობიექტებს შორის. რელაცია დავამყაროთ Personali და Shemkveti ცხრილებს შორის. Personali ცხრილში მოთავსებულია მონაცემები ფირმის თანამშრომლების შესახებ, Shemkveti ცხრილში კი - შემკვეთების შესახებ. თითოეულ თანამშრომელს შეიძლება ჰყავდეს ერთი ან მეტი შემკვეთი. მთავარია Personali ცხრილი. მისი პირველადი გასაღებია personaliID სვეტი. Shemkveti ცხრილი დამოკიდებულია. მისი გარე გასაღებია personaliID სვეტი. ამ ცხრილებს შორის კავშირის დასამყარებლად Personali ცხრილის პირველადი გასაღები უნდა დავუკავშიროთ Shemkveti ცხრილის გარე გასაღებს. DataRelation ობიექტის შესაქმნელად გამოიყენება Relations ობიექტის Add() მეთოდი. მოყვანილი პროგრამით ხდება რელაციის დემონსტრირება:

```
{
// შეერთების ობიექტის შექმნა
SqlConnection myConnection = new
```

```
SqlConnection("server=ROMANI-PC; database=Shekveta; Integrated Security = True");
// შეერთების გახსნა
myConnection.Open();
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// DataAdapter ობიექტების შექმნა თითოეული ცხრილისთვის და მათი შევსება
SqlDataAdapter personaliAdapter = new SqlDataAdapter("SELECT * FROM Personali",
myConnection);
SqlDataAdapter shemkvetiAdapter = new SqlDataAdapter("SELECT * FROM Shemkveti",
myConnection);
// myDataset ობიექტის შევსება Personali და Shemkveti ცხრილებით
personaliAdapter.Fill(myDataset, "Personali");
shemkvetiAdapter.Fill(myDataset, "Shemkveti");
// კავშირის დამყარება ცხრილებს შორის
DataRelation personaliShemkvetiRelation = myDataset.Relations.Add("PersonaliShemkveti",
myDataset.Tables["Personali"].Columns["PersonaliID"],
myDataset.Tables["Shemkveti"].Columns["PersonaliID"]);
// ეკრანზე სტრიქონების გამოტანა
//myDataset.Relations.Add(personaliShemkvetiRelation);
// შეერთების დახურვა
myConnection.Close();
BindingSource bindingSource1 = new BindingSource(myDataset, "Personali");
BindingSource bindingSource2 = new BindingSource(myDataset, "Shemkveti");
bindingSource2.DataSource = bindingSource1;
bindingSource2.DataMember = "PersonaliShemkveti";
dataGridView1.DataSource = bindingSource1;
dataGridView2.DataSource = bindingSource2;
}
```

სავარჯიშოები

1. Personali ცხრილის მე-6 სტრიქონში staji სვეტის მნიშვნელობა შეცვალეთ. ახალი მნიშვნელობა TextBox ელემენტიდან შეიტანეთ. ახალი მნიშვნელობაა 14.
2. Personali ცხრილში სტრიქონი ჩაუმატეთ. staji და asaki სვეტების ახალი მნიშვნელობებია 7 და 30. ახალი მნიშვნელობები TextBox ელემენტებიდან შეიტანეთ.
3. Personali ცხრილიდან წაშალეთ ერთი სტრიქონი, რომლის პირველადი გასაღებია 2. ეს მნიშვნელობა TextBox ელემენტიდან შეიტანეთ.
4. Personali და Xelshekruleba ცხრილებს შორის დაამყარეთ რელაცია.
5. Shemkveti ცხრილის მე-7 სტრიქონში qalaqi სვეტის მნიშვნელობა შეცვალეთ. ახალი მნიშვნელობა TextBox ელემენტიდან შეიტანეთ. ახალი მნიშვნელობაა ‘ონი’.
6. Shemkveti ცხრილში სტრიქონი ჩაუმატეთ. qalaqi და regioni სვეტების ახალი მნიშვნელობებია ‘ქობულეთი’ და ‘აჭარა’. ახალი მნიშვნელობები TextBox ელემენტებიდან შეიტანეთ.
7. Shemkveti ცხრილიდან წაშალეთ ერთი სტრიქონი, რომლის პირველადი გასაღებია 1. ეს მნიშვნელობა TextBox ელემენტიდან შეიტანეთ.
8. Shemkveti და Xelshekruleba ცხრილებს შორის დაამყარეთ რელაცია.

პრაქტიკული სამუშაო #10

ADO.NET კლასები. მონაცემების წაკითხვა, გაფილტვრა, დახარისხება და ძებნა, გამოთვლების შესრულება

ADO.NET არის კლასების სიმრავლე, რომელიც გამოიყენება C# და .NET Framework გარემოსთან ერთად სხვადასხვა ფორმატის მონაცემთა ბაზასთან სამუშაოდ. ADO.NET ტექნოლოგია ინტეგრირებულია .NET Framework გარემოსთან და ორიენტირებულია .NET-ის ნებისმიერ ენასთან, განსაკუთრებით კი - C#-თან სამუშაოდ. ის მოიცავს System.Data სახელების სივრცეს და მასში შემავალ ჩადგმულ სახელების სივრცეებს: System.Data.SqlClient, System.Data.Linq და ა.შ.

მონაცემების კითხვა

მოყვანილი პროგრამა სტრიქონებს კითხულობს Personali ცხრილიდან და ისინი DataGridView ელემენტში გამოაქვს.

```
//      პროგრამით ხდება Personali ცხრილის მოთავსება myDataset მონაცემთა ნაკრებში
//      შეერთების ობიექტის შექმნა
SqlConnection myConnection =
new SqlConnection("server=ROMANI\\SQL_2012; database=Shekveta; Integrated Security = True");
//      შეერთების გახსნა
myConnection.Open();
//      DataAdapter ობიექტის შექმნა
SqlDataAdapter myAdapter = new
SqlDataAdapter("SELECT  * FROM Personali", myConnection);
//      DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
//      myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset,"Personali");
//      Personali ცხრილის სტრიქონების გამოტანა ეკრანზე
dataGridView1.DataSource = myDataset;
dataGridView1.DataMember = "Personali";
//      შეერთების დახურვა
myConnection.Close();
}
```

მონაცემების გაფილტვრა

მონაცემების გასაფილტრად შეგვიძლია DataView ობიექტის გამოყენება. დავუშვათ, გვაინტერესებს ინფორმაცია სამედიცინო განყოფილებაში მომუშავე იმ თანამშრომლების შესახებ, რომელთა ასაკი აღემატება 30 წელს. მოცემული კოდით ხდება ამის დემონსტრირება:

```
{
// შეერთების ობიექტის შექმნა
SqlConnection myConnection =
new SqlConnection("server=ROMANI\\SQL_2012; database=Shekveta; Integrated Security = True");
```

```
// შეერთების გახსნა
myConnection.Open();
// DataAdapter ობიექტის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM Personali",
myConnection);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
// შეერთების დახურვა
myConnection.Close();
// dataView1 ობიექტის დაკავშირება Personali ცხრილთან
DataView dataView1 = new DataView(myDataset.Tables["Personali"]);
// სტრიქონების გაფილტვრა
dataView1.RowFilter = "ganyofileba = 'სავაჭრო' AND asaki > 30";
//dataView1.RowFilter = textBox1.Text; // ganyofileba = 'სავაჭრო' AND asaki > 30
// გაფილტრული სტრიქონების ასახვა ეკრანზე
dataGridView1.DataSource = dataView1;
}
```

მონაცემების დახარისხება

მონაცემების დასახარისხებლად შეგვიძლია DataView ობიექტის გამოყენება. ამისათვის გამოიყენება მისი Sort თვისება. მოყვანილი კოდით ხდება Personali ცხრილის სტრიქონების დახარისხება ganyofileba სვეტის მნიშვნელობების ზრდის მიხედვით და asaki სვეტის მნიშვნელობების კლების მიხედვით:

```
{
// შეერთების ობიექტის შექმნა
SqlConnection myConnection =
new SqlConnection("server=ROMANI\\SQL_2012; database=Shekveta; Integrated Security =
True");
// შეერთების გახსნა
myConnection.Open();
// DataAdapter ობიექტის შექმნა
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM Personali",
myConnection);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
// შეერთების დახურვა
myConnection.Close();
// dataView1 ობიექტის დაკავშირება Personali ცხრილთან
DataView dataView1 = new DataView(myDataset.Tables["Personali"]);
// სტრიქონების დახარისხება
dataView1.Sort = "ganyofileba, asaki DESC";
```

```
//dataView1.Sort = textBox1.Text;
// დახარისხებული სტრიქონების ასახვა ეკრანზე
dataGridView1.DataSource = dataView1;
}
```

გამოთვლების შესრულება

გამოთვლების შესასრულებლად შეგვიძლია გამოვიყენოთ DataSet კომპონენტის Compute მეთოდი. მისი პირველი პარამეტრია შესასრულებელი ფუნქცია, მეორე კი - ფილტრი. ის საშუალებას გვაძლევს მითითებული სვეტის მნიშვნელობებზე შემდეგი ფუნქციები შევასრულოთ: SUM, AVG, COUNT, MAX, MIN. მოყვანილ კოდში სრულდება Personali ცხრილის xelfasi სვეტის მნიშვნელობების შეკრება:

```
{
object sumObject = shekvetaDataSet.Personali.Compute("SUM(xelfasi)", "");
label1.Text = sumObject.ToString();
}
```

Compute მეთოდის პირველი პარამეტრი შეიცავს შესასრულებელ ფუნქციას, მეორე კი ფილტრს. მოყვანილ პროგრამაში გამოითვლება asaki სვეტის საშუალო არითმეტიკული იმ სტრიქონებისთვის, რომლებშიც xelfasi სვეტის მნიშვნელობა 700-ს აღემატება.

```
{
object avgObject = shekvetaDataSet.Personali.Compute("AVG(asaki)", "xelfasi > 700");
label1.Text = avgObject.ToString();
}
```

ანალოგიურად, შეგვიძლია დანარჩენი ფუნქციების მნიშვნელობების გამოთვლა:

```
{
// შეერთების ობიექტის შექმნა
        SqlConnection myConnection =
      new SqlConnection("server=ROMANI\\SQL_2012; database=Shekveta; Integrated Security
= True");
        // შეერთების გახსნა
        myConnection.Open();
        // DataAdapter ობიექტის შექმნა
        SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM Personali",
myConnection);
        // DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
        DataSet myDataset = new DataSet();
        // myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
        myAdapter.Fill(myDataset, "Personali");
        // შეერთების დახურვა
        myConnection.Close();
        dataGridView1.DataSource = myDataset;
        dataGridView1.DataMember = "Personali";
        //
object count_1 = ds_1.Tables["Personali"].Compute("Count(personaliID)","asaki > 30");
object max_1 = ds_1.Tables["Personali"].Compute("Max(asaki)","ganyofileba = 'სასპორტო'");
```

```
object min_1 = ds_1.Tables["Personali"].Compute("Min(staji)","xelfasi >= 1030.55");
object avg_1 = ds_1.Tables["Personali"].Compute("AVG(xelfasi)","tarigi_dabadebis >
'01.01.2000'");
object sum_1 = ds_1.Tables["Personali"].Compute("SUM(xelfasi)","");
        label1.Text = sum_1.ToString();
        label2.Text = avg_1.ToString();
        label3.Text = count_1.ToString();
        label4.Text = max_1.ToString();
        label5.Text = min_1.ToString();
}
```

მონაცემების ძებნა

ცხრილში სტრიქონის მოსაძებნად Find() მეთოდი გამოიყენება. ეს მეთოდი ითხოვს ცხრილში პირველადი გასაღების დაყენებას. საჭირო მნიშვნელობა იძებნება პირველად გასაღებში. პირველადი გასაღები შეიძლება შედგებოდეს ერთი ან მეტი სვეტისგან, რომელთა მნიშვნელობები ცალსახად განსაზღვრავენ ერთ სტრიქონს. ამიტომ, ძებნის შედეგი ყოველთვის იქნება ცხრილის ერთი სტრიქონი. მოყვანილი პროგრამით ხდება Personali ცხრილში სტრიქონის ძებნის დემონსტრირება:

```
{
// შეგვაქვს პირველადი გასაღების მნიშვნელობა
int find_key = Convert.ToInt32(textBox1.Text);
// შეერთების ობიექტის შექმნა
SqlConnection myConnection =
new SqlConnection("server=ROMANI\\SQL_2012; database=Shekveta; Integrated Security =
True");
// შეერთების გახსნა
myConnection.Open();
// ადაპტერის მომზადება სამუშაოდ
SqlDataAdapter myAdapter = new SqlDataAdapter("SELECT * FROM Personali",
myConnection);
// DataSet ობიექტის შექმნა ცხრილების, სტრიქონებისა და სვეტების შესანახად
DataSet myDataset = new DataSet();
// myDataset ობიექტის შევსება Personali ცხრილის სტრიქონებით
myAdapter.Fill(myDataset, "Personali");
// გასაღებების მასივის გამოცხადება პირველადი გასაღების განსაზღვრისათვის
DataColumn[] keys = new DataColumn[1];
keys[0] = myDataset.Tables["Personali"].Columns["PersonaliID"];
myDataset.Tables["Personali"].PrimaryKey = keys;
// findRow ობიექტს ენიჭება ნაპოვნი სტრიქონი
DataRow findRow = myDataset.Tables["Personali"].Rows.Find(find_key);
if ( findRow != null )
{
label1.Text = "სტრიქონი მოიძებნა: ";
label1.Text += findRow["gvari"] + " " + findRow["xelfasi"].ToString() + " ";
}
```

```
else label1.Text = "სტრიქონი ვერ მოიძებნა";
// შეერთების დახურვა
myConnection.Close();
}
```

სავარჯიშოები

1. მოთხოვნა შეასრულეთ ADO.NET კლასების გამოყენებით. Shemkveti ცხრილიდან წაიკითხეთ მონაცემები და ისინი DataGridView ელემენტში გამოიტანეთ.
2. ცხრილში მონაცემების გაფილტვრა ADO.NET კლასებისა და ობიექტების გამოყენებით. გაფილტრეთ Shemkveti ცხრილი. ფილტრში უნდა გამოჩნდეს მონაცემები ქობულეთელი იურიდიული პირების შესახებ.
3. ცხრილში მონაცემების დახარისხება ADO.NET კლასებისა და ობიექტების გამოყენებით. დაახარისხეთ Xelshekruleba ცხრილი კლებადობით tarigi_dawyebis სვეტის მიხედვით და ზრდადობით gadasaxdeli_l სვეტის მიხედვით.
4. ცხრილში მონაცემების ძებნა ADO.NET კლასებისა და ობიექტების გამოყენებით. ძებნა შეასრულეთ Personali ცხრილში. ეკრანზე გამოიტანეთ იმ ხელშეკრულების სტრიქონი, რომლის ნომერია = 4.
5. მოთხოვნების შესრულება ADO.NET კლასების გამოყენებით. Shemkveti ცხრილში დათვალეთ ქალი ფიზიკური პირის რაოდენობა.
6. ცხრილში მონაცემების გაფილტვრა ADO.NET კლასებისა და ობიექტების გამოყენებით. Personali ცხრილში იპოვეთ თანამშრომლების მაქსიმალური ხელფასი.
7. ცხრილში მონაცემების გაფილტვრა ADO.NET კლასებისა და ობიექტების გამოყენებით. Personali ცხრილში იპოვეთ თანამშრომლების მინიმალური ხელფასი.
8. ცხრილში მონაცემების გაფილტვრა ADO.NET კლასებისა და ობიექტების გამოყენებით. Personali ცხრილში იპოვეთ სავაჭრო განყოფილების თანამშრომლების საშუალო ხელფასი.

პრაქტიკული სამუშაო #8

დელეგატი. მოვლენა

დელეგატი. დელეგატების მრავალმისამართიანობა

დელეგატი - ესაა ობიექტი, რომელიც მეთოდს მიმართავს. დელეგატები გამოიყენება მეთოდების გამოძახებისთვის. ამასთან, გამოსაძახებელი მეთოდის არჩევა ხდება დინამიკურად, პროგრამის შესრულების დროს. ამიტომ, დელეგატები იმ შემთხვევაში გამოიყენება, როცა წინასწარ არ ვიცით, თუ რომელი მეთოდი უნდა გამოვიძახოთ. დელეგატების გამოყენების კიდევ ერთი უპირატესობა იმაში მდგომარეობს, რომ ისინი უზრუნველყოფენ მოვლენებს.

დელეგატი ინახავს გამოსაძახებელი მეთოდის სახელს, მის მიერ დაბრუნებული შედეგის ტიპსა და პარამეტრების სიას, ანუ მეთოდის სიგნატურას (ხელმოწერას). ამრიგად, მეთოდის სიგნატურა - ესაა მეთოდის სახელი, მის მიერ დაბრუნებული მნიშვნელობის ტიპი და პარამეტრების სია.

როგორც ცნობილია, მეთოდისთვის მეხსიერებაში გარკვეული ზომის უბანი გამოიყოფა. ამ უბნის მისამართი - ესაა მეთოდის შესვლის წერტილი. სწორედ ამ მისამართის ინიცირება ხდება მეთოდის გამოძახებისას. მეთოდის მისამართი ენიჭება დელეგატს. თუ დელეგატი მიმართავს მეთოდს, მაშინ მოცემული მეთოდი შეგვიძლია გამოვიძახოთ ამ დელეგატის საშუალებით. გარდა ამისა, ერთი და იგივე დელეგატი შეგვიძლია გამოვიყენოთ სხვადასხვა მეთოდის გამოძახებისთვის. ამისთვის, მას უნდა მივანიჭოთ ამ მეთოდებზე მიმართვა.

დელეგატის გამოცხადებისთვის უნდა გამოვიყენოთ delegate საკვანძო სიტყვა. დელეგატის გამოცხადების სინტაქსია:

delegate ***ტიპი დელეგატის_სახელი(პარამეტრების_სია)*****;**

აქ ტიპი არის იმ მნიშვნელობის ტიპი, რომელსაც ის მეთოდი აბრუნებს, რომელიც დელეგატის მიერ იქნება გამოძახებული.

როგორც უკვე აღვნიშნეთ, დელეგატს შეუძლია გამოიძახოს ის მეთოდი, რომლის სიგნატურა შეესაბამება დელეგატის სიგნატურას. ეს საშუალებას იძლევა პროგრამის შესრულების დროს ავირჩიოთ გამოსაძახებელი მეთოდი. ასეთი მეთოდი შეიძლება იყოს ობიექტის მეთოდი ან კლასში განსაზღვრული სტატიკური მეთოდი. მეთოდი მხოლოდ მაშინ შეიძლება გამოვიძახოთ, როცა მისი ხელმოწერა შეესაბამება დელეგატის ხელმოწერას.

ქვემოთ მოყვანილ პროგრამაში ხდება დელეგატთან მუშაობის დემონსტრირება:

```
//      პროგრამაში ხდება დელეგატისთვის ობიექტის მეთოდზე მიმართვის მინიჭება
delegate string Delegati(string striqoni1);
class ChemiKlasi
{
public string Shecvla(string striqoni2)
{
return striqoni2.Replace(' ', '-');
}
public string Washla(string striqoni2)
{
string temp = "";
int i;

for ( i = 0; i < striqoni2.Length; i++ )
        if (striqoni2[i] != ' ') temp += striqoni2[i];
return temp;
}
public string Shebruneba(string striqoni2)
```

```
{
string temp = "";
int i;

for ( i = striqoni2.Length - 1; i >= 0; i-- )
        temp += striqoni2[i];
return temp;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
ChemiKlasi obieqti = new ChemiKlasi();
//      დელეგატის ეგზემპლარის შექმნა
Delegati delegati_obieqti = new Delegati(obieqti.Shecvla);
string striqoni;
//      მეთოდის გამოძახება დელეგატის საშუალებით
striqoni = delegati_obieqti("მეთოდების მუშაობის შემოწმება");
label1.Text = striqoni;

delegati_obieqti = new Delegati(obieqti.Washla);
striqoni = delegati_obieqti("მეთოდების მუშაობის შემოწმება");
label2.Text = striqoni;

delegati_obieqti = new Delegati(obieqti.Shebruneba);
striqoni = delegati_obieqti("მეთოდების მუშაობის შემოწმება");
label3.Text = striqoni;
}
```

პროგრამის მუშაობის შედეგად ეკრანზე გამოიცემა იგივე სტრიქონები, რომლებიც წინა პროგრამაში. მაგრამ, ამ შემთხვევაში დელეგატი მეთოდებს მიმართავს ChemiKlasi კლასის ობიექტის გამოყენებით.

დელეგატების მრავალმისამართიანობა

დელეგატს შეუძლია შეინახოს რამდენიმე მეთოდის მისამართი. ამ მისამართების მიმდევრობით ინიციალიზების გზით დელეგატს შეუძლია ერთმანეთის მიყოლებით გამოიძახოს შესაბამისი მეთოდები. დელეგატის ამ თვისებას მრავალმისამართიანობა ეწოდება. გამოსაძახებელი მეთოდების მისამართების ასეთი მიმდევრობა, ანუ მეთოდებზე მიმართვების მწკრივი, შემდეგნაირად იქმნება. თავდაპირველად იქმნება დელეგატი (დელეგატი-ობიექტი), შემდეგ კი გამოიყენება += ოპერატორი მწკრივისთვის მეთოდების მისამართების დასამატებლად. შედეგად, დელეგატი მრავალმისამართიანი ხდება. მწკრივიდან მეთოდის მისამართების წასაშლელად გამოიყენება -= ოპერატორი. ერთადერთი შეზღუდვაა ის, რომ დელეგატს, რომელიც ინახავს რამდენიმე მიმართვას, უნდა ჰქონდეს დასაბრუნებელი მნიშვნელობის void ტიპი.

ქვემოთ მოყვანილია პროგრამა, რომელიც ახდენს მრავალმისამართიანი დელეგატის გამოყენების დემონსტრირებას. რადგან, მეთოდის მიერ დაბრუნებული შედეგის ტიპია void, ამიტომ გამომძახებელი პროგრამისთვის შეცვლილი სტრიქონის დასაბრუნებლად გამოიყენება ref მოდიფიკატორი.

```
//      პროგრამაში ხდება მრავალმისამართიან დელეგატთან მუშაობის დემონსტრირება
delegate void Delegati(ref string striqoni1);
```

```
class ChemiKlasi
{
public void Shecvla(ref string striqoni2)
{
striqoni2 = striqoni2.Replace(' ', '-');
}
public void Washla(ref string striqoni2)
{
string temp = "";
int i;

for ( i = 0; i < striqoni2.Length; i++ )
        if (striqoni2[i] != ' ' ) temp += striqoni2[i];
striqoni2 = temp;
}
public void Shebruneba(ref string striqoni2)
{
string temp = "";
int i;

for ( i = striqoni2.Length - 1; i >= 0; i-- )
        temp += striqoni2[i];
striqoni2 = temp;
}
}
private void button1_Click(object sender, System.EventArgs e)
{
        ChemiKlasi obieqti = new ChemiKlasi();
        Delegati delegati_mimartva;
        Delegati Shecvla_obieqti = new Delegati(obieqti.Shecvla);
        Delegati Washla_obieqti = new Delegati(obieqti.Washla);
        Delegati Shebruneba_obieqti = new Delegati(obieqti.Shebruneba);
        string striqoni = " მეთოდების მუშაობის შემოწმება";
        label1.Text = striqoni;
        //      დელეგატს ორი მიმართვა ენიჭება
        delegati_mimartva = Shecvla_obieqti;
        delegati_mimartva += Shebruneba_obieqti;        //      მიმართვების მწკრივის შექმნა
        //      მრავალმისამართიანი დელეგატის გამოძახება
        delegati_mimartva(ref striqoni);
        label2.Text = striqoni;
        //მიმართვების მწკრივიდან Shecvla მიმართვის წაშლა და Washla მიმართვის დამატება
        delegati_mimartva -= Shecvla_obieqti;
        delegati_mimartva += Washla_obieqti;        //      მიმართვების ახალი მწკრივის შექმნა
        striqoni = " მეთოდების მუშაობის შემოწმება";
        //      მრავალმისამართიანი დელეგატის გამოძახება
        delegati_mimartva(ref striqoni);
        label3.Text = striqoni;
}
```

პროგრამაში იქმნება ოთხი დელეგატი (დელეგატი-ობიექტი). delegati_obieqti დელეგატს აქვს null მნიშვნელობა, რადგან შექმნისას ის არ იყო ინიციალიზებული. დანარჩენი სამი დელეგატი მიმართავს შესაბამის მეთოდებს. შემდეგ, პროგრამაში იქმნება მრავალმისამართიანი დელეგატი, რომელიც იძახებს Shecvla() და Shebruneba() მეთოდებს. ეს ხორციელდება სამი ოპერატორის საშუალებით:

```
delegati_obieqti  = Shecvla_mimartva;
delegati_obieqti += Shebruneba_mimartva;
delegati_obieqti(ref striqoni);
```

პირველ ოპერატორში delegati_obieqti დელეგატს ენიჭება Shecvla_mimartva მიმართვა. შემდეგ += ოპერატორის საშუალებით მწკრივს ემატება Shebruneba_mimartva მიმართვა. delegati_obieqti დელეგატთან მიმართვისას ჯერ შესრულდება Shecvla() მეთოდი, შემდეგ კი Shebruneba() მეთოდი. delegati_obieqti -= Shecvla_mimartva; ოპერატორი მწკრივიდან წაშლის Shecvla_mimartva მიმართვას, ხოლო delegati_obieqti += Washla_mimartva; ოპერატორი მწკრივს დაუმატებს Washla_mimartva მიმართვას. შედეგად, delegati_obieqti დელეგატთან მიმართვისას ჯერ შესრულდება Shebruneba() მეთოდი, შემდეგ კი Washla() მეთოდი.

მოვლენა

მოვლენა არის ავტომატური უწყება რაიმე მომხდარ მოქმედებაზე. მოვლენის საშუალებით ერთი ობიექტი მეორე ობიექტს ატყობინებს, რომ რაღაც მოხდა. მოვლენის მაგალითებია: კლავიატურის კლავიშის დაჭერა, კლავიატურის კლავიშის აშვება, თაგვის მარცხენა კლავიშის დაჭერა, თაგვის მარჯვენა კლავიშის დაჭერა, თაგვის კლავიშის აშვება და ა.შ.

ობიექტისთვის, რომელიც მისი განსაზღვრის (კოდის) მიხედვით უნდა რეაგირებდეს რაიმე მოვლენაზე, რეგისტრირდება ამ მოვლენის დამამუშავებელი (მეთოდი). მოვლენების დამამუშავებლები იქმნება დელეგატების საფუძველზე.

მოვლენები წარმოადგენენ კლასის წევრებს და მათი გამოცხადება ხდება event საკვანძო სიტყვის გამოყენებით. მოვლენების გამოცხადების სინტაქსია:

event *დელეგატის_სახელი მოვლენის_სახელი;*

აქ დელეგატის_სახელი იმ დელეგატის სახელია, რომელიც გამოიყენება მოვლენის დასამუშავებლად, ხოლო მოვლენის_სახელი არის შესაქმნელი მოვლენის (მოვლენა-ობიექტის) სახელი.

მაგალითი. მოვლენის აღმძვრელ მეთოდს გადავცეთ რიცხვი და თუ ის დადებითია, მაშინ აღვძრათ მოვლენა. შესაბამისი შეტყობინება Label კომპონენტში გამოვიტანოთ. მოყვანილი პროგრამით ხდება ამის დემონსტრირება:

```
//      პროგრამას მოვლენის აღძვრისას შეტყობინება  Label კომპონენტში გამოაქვს
 delegate void ChemiDelegati(Label lab);
 //      კლასის გამოცხადება, რომელშიც ხდება მოვლენის ინიცირება
 class Klasi1
 {
  public event ChemiDelegati ChemiMovlena; //          მოვლენის გამოცხადება
  // ამ მეთოდში ინიცირდება მოვლენა
  public void Metodi1(int par1, Label lab)
  {
   //      პირობის განსაზღვრა მოვლენის ინიცირებისთვის
   if ( par1 >= 0 ) ChemiMovlena(lab);
  }
  public void MovlenisDamamushavebeli(Label lab)
  {
```

```
  lab.Text = "აღიძრა მოვლენა, რიცხვი დადებითია";
 }
}
//
private void button1_Click(object sender, EventArgs e)
{
 label1.Text = “”;
 int ricxvi = int.Parse(textBox1.Text);
 Klasi1 obieqti_movlena = new Klasi1();                    //        მოვლენის ეგზემპლარის შექმნა
 //
 ChemiDelegati deleg_1 = new ChemiDelegati(obieqti_movlena.MovlenisDamamushavebeli);
 obieqti_movlena.ChemiMovlena += deleg_1;
 //       მოვლენის ინიცირება
 obieqti_movlena.Metodi1(ricxvi, label1);
}
```

მოცემულ შემთხვევაში გამოიძახება მხოლოდ ერთი დამამუშავებელი, თუმცა ისინი შეიძლება რამდენიმე იყოს.

სავარჯიშოები

დელეგატი. დელეგატების მრავალმისამართიანობა

1. დელეგატს დაუმატეთ 2 მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით I მეთოდის მისამართი და დაუმატეთ III მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. I მეთოდს გადაეცემა ერთგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის კენტ ელემენტებს ამრავლებს 3-ზე. II მეთოდს გადაეცემა ერთგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის ლუწ ელემენტებს ამრავლებს 2-ზე. III მეთოდს გადაეცემა ერთგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის ელემენტებს ამრავლებს 10-ზე.
2. დელეგატს დაუმატეთ 2 მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით II მეთოდის მისამართი და დაუმატეთ III მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. I მეთოდს გადაეცემა ერთგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის კენტ ელემენტებს ზრდის 5-ჯერ. II მეთოდს გადაეცემა ერთგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის კენტ ელემენტებს ამცირებს 5-ით. III მეთოდს გადაეცემა ერთგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის ელემენტებს ამრავლებს 10-ზე.
3. დელეგატს დაუმატეთ 3 მეთოდის მისამართი. სამივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით I და III მეთოდების მისამართები და დაუმატეთ IV მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. I მეთოდს გადაეცემა ორგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის კენტ ელემენტებს ამრავლებს 3-ზე. II მეთოდს გადაეცემა ორგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის ლუწ ელემენტებს ამრავლებს 2-ზე. III მეთოდს გადაეცემა ორგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის ელემენტებს ამრავლებს 10-ზე. IV მეთოდს გადაეცემა ორგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის ელემენტებს ზრდის 5-ით.
4. დელეგატს დაუმატეთ 3 მეთოდის მისამართი. სამივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით I და III მეთოდების მისამართები და დაუმატეთ IV მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. I მეთოდს გადაეცემა ორგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის კენტ ელემენტებს ამრავლებს 7-ზე. II მეთოდს გადაეცემა ორგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის ლუწ ელემენტებს ამრავლებს 4-ზე. III მეთოდს გადაეცემა ორგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის ელემენტებს ამრავლებს 20-ზე. IV მეთოდს გადაეცემა ორგანზომილებიანი მთელრიცხვა მასივი. მეთოდი ამ მასივის ელემენტებს ამცირებს 5-ით.
5. დელეგატს დაუმატეთ 3 მეთოდის მისამართი. სამივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით II და III მეთოდების მისამართები და დაუმატეთ IV მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. I მეთოდს გადაეცემა სტრიქონი. მეთოდი ამ სტრიქონში ხმოვან სიმბოლოებს შეცვლის სიმბოლო-ციფრით ‘0’. II მეთოდს გადაეცემა სტრიქონი. მეთოდი ამ სტრიქონში ‘ბ’ სიმბოლოს შეცვლის სიმბოლო-ციფრით ‘1’. III მეთოდს გადაეცემა სტრიქონი. მეთოდი ამ სტრიქონში წაშლის ‘ა’ სიმბოლოს. IV მეთოდს გადაეცემა სტრიქონი. მეთოდი ამ სტრიქონში წაშლის სასვენ ნიშნებს.
6. დელეგატს დაუმატეთ 3 მეთოდის მისამართი. სამივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. შემდეგ, დელეგატს გამოაკელით II და III მეთოდების მისამართები და დაუმატეთ IV მეთოდის მისამართი. ორივე მეთოდი გამოიძახეთ დელეგატის საშუალებით. I მეთოდს გადაეცემა სტრიქონი. მეთოდი ამ სტრიქონში ხმოვან სიმბოლოებს შეცვლის სიმბოლო-ციფრით ‘0’. II მეთოდს გადაეცემა სტრიქონი. მეთოდი

ამ სტრიქონში სასვენ ნიშნებს შეცვლის სიმბოლო-ციფრით ‘1’. III მეთოდს გადაეცემა სტრიქონი. მეთოდი ამ სტრიქონში წაშლის ‘ე’ სიმბოლოს. IV მეთოდს გადაეცემა სტრიქონი. მეთოდი ამ სტრიქონში წაშლის ხმოვან სიმბოლოებს.

მოვლენა

1. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ, როცა პირველი რიცხვი მეორეზე მეტია. დამამუშავებელს გამოაქვს შესაბამისი შეტყობინება.
2. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ, როცა ორივე რიცხვი ლუწია. დამამუშავებელს გამოაქვს შესაბამისი შეტყობინება.
3. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ როცა ორი სტრიქონი ერთნაირი არ არის.
4. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ, როცა მასივის ყველა ელემენტი 50-ზე ნაკლებია.
5. შექმნენით მოვლენა, რომელიც აღიძვრება ნულზე გაყოფის შემთხვევაში.
6. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ როცა 5-ელემენტიან მთელრიცხვა მასივში მხოლოდ ნულებია.
7. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ როცა 25-სიმბოლოიან სტრიქონში მხოლოდ '*'-ებია.
8. შექმენით მოვლენა, რომელიც აღიძვრება მაშინ, როცა შეტანილია ლუწი რიცხვი. დამამუშავებელი გასცემს ლუწი რიცხვის კვადრატს.

პრაქტიკული #9
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.