საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, ეკატერინე თურქია,
ირაკლი ბულია

Web-აპლიკაციების დამუშავება მონაცემთა ბაზების საფუძველზე (ADO.NET, ASP.NET, C#)

რეგისტრირებულია
სტუ-ს სარედაქციო-
საგამომცემლო
საბჭოს მიერ

თბილისი
2009

უაკ 681.3.06

სახელმძღვანელოში გადმოცემულია მაიკროსოფთის ფირმის ახალი პროგრამული პლატფორმის ბაზაზე შექმნილი Web-აპლიკაციების აგების ინსტრუმენტული საშუალებანი და სერვის-ფუნქციები ASP.NET პაკეტის საფუძველზე. შემოთავაზებულია .NET პლატფორმის კონცეფცია

განკუთვნილია პირველ რიგში, მართვის საინფორმაციო სისტემების (Management Information Systems) სპეციალობის მაგისტრანტებისათვის, აგრეთვე თეორიული და პრაქტიკული ინფორმატიკისა და სხვა დარგების სპეციალისტებისათვის, დოქტორანტების და სტუდენტებისათვის, რომლებიც მართვის ავტომატიზებული სისტემების დაპროექტებას და კვლევას აწარმოებენ.

რეცენზენტი: ტ.მ.დ.,პროფ. **რ. სამხარაძე**

პროფ. გ. სურგულაძის რედაქციით

ISBN 978-9941-14-289-5

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

Georgian Technical University

GIA SURGULADZE, EKATERINE TURKIA
IRAKLI BULIA

DEVELOPMENT A WEB-APPLICATIONS ON THE BASIS OF DATABASES

(ADO.NET, ASP.NET, C#)

Supported by DAAD (Germany)

ISBN 978-9941-14-289-5

გია სურგულაძე

gmx@gmx.net

ინფორმატიკის ფაკულტეტის „მართვის ავტომატიზებული სისტემების“ მიმართულების ხელმძღვანელი, სრული პროფესორი, ტექნიკის მეცნიერებათა დოქტორი. 50 წიგნის და 150-ზე მეტი სამეცნიერო ნაშრომის ავტორი ინფორმაციული ტექნოლოგიების სფეროში. გერმანიის ჰუმბოლდტისა და ნიურნბერგ-ერლანგენის უნივრსიტეტების მიწვეული პროფესორი.

ეკატერინე თურქია

ekaterinet7@gmail.com

სტუ-ს ინფორმატიკის ფაკულტეტის „მართვის ავტომატიზებული სისტემების“ მიმართულების ასოც. პროფესორი, ტექნიკის მეცნიერებათა კანდიდატი. 30-ზე მეტი სამეცნიერო ნაშრომის და წიგნის ავტორი ბიზნეს-პროცესების მოდელირებისა და დაპროექტების სფეროში. გერმანიის DAAD არაერთგზის გრანტის მფლობელი.

ირაკლი ბულია

სტუ-ს „მართვის ავტომატიზებული სისტემების“ კათედრის დოქტორანტი, ყოფილი კურსდამთავრებული და მაგისტრი, „რესპუბლიკა“ ბანკის web-დეველოპერი. აქვს დიდი პრაქტიკული გამოცდილება დაპროგრამების ობიექტ-ორიენტირებული და ვიზუალური C#, C++, VB ენების გამოყენების საქმეში, სხვადასხვა კომპიუტერული პროექტების რეალიზაციაში მონაცემთა ბაზებით NET-პლატფორმაზე.

ს ა რ ჩ ე ვ ი

შესავალი

კომპიუტერული ინდუსტრია და საინფორმაციო ტექნოლოგიები განვითარების მაღალი ტემპებით ხასიათდება. მსოფლიო ბაზარზე გამოჩნდა არა ერთი ახალი აპარატურული (Hardware) და პროგრამული (Software, Groupware) სისტემები.

წინამდებარე სახელმძღვანელოში გადმოცემულია მაიკროსოფთის ფირმის ახალი პროგრამული NET პლატფორმის ბაზაზე შექმნილი ჭებ-აპლიკაციების აგების ინსტრუმენტული საშუალებანი და სერვის-ფუნქციები ASP.NET პაკეტის საფუძველზე. შემოთავაზებულია ამ პლატფორმაზე C#.NET ენის ინტეგრირებული გარემო და მის საფუძველზე ASP.NET და ADO.NET პროგრამული კოდების მართვა.

პირველ თავში მოცემულია .NET პლატფორმის ზოგადი კონცეფცია და მუშაობის პრინციპები Visual Studio .NET Framework გარემოში. აგრეთვე ობიექტ-ორიენტირებული დაპროგრამების ძირითადი ცნებები.

მეორე თავში გადმოცემულია C#.NET ენის საფუძვლები და კომპონენტურ-ვიზუალური დაპროგრამების ელემენტები.

მესამე თავი ეხება ASP.NET პაკეტის საფუძველზე ვებ-გვერდების აგების მეთოდებისა და ინსტრუმენტების გაცნობას. შემოთავაზებულია პრაქტიკული მაგალითები aspx და C# კოდირებისათვის Visual Studio .NET გარემოში.

მეოთხე თავში გადმოცემულია ADO.NET პაკეტის ობიექტების, კოდების და მისი ფუნქციების აღწერის მაგალითები, SQL Server მონაცემთა ბაზასთან კავშირებისა და C#.NET ენის საფუძველზე ინტერფეისების დაპროექტების და აგების ილუსტრაციები.

წიგნი განკუთვნილია მართვის საინფორმაციო სისტემების (Management Information Systems) სპეციალობის მაგისტრანტებისათვის, აგრეთვე თეორიული და პრაქტიკული ინფორმატიკისა და სხვა დარგების სპეციალისტებისათვის, დოქტორანდების და სტუდენტებისათვის, რომლებიც მართვის ავტომატიზებული სისტემების დაპროექტებას და კვლევას აწარმოებენ.

I თავი

.NET ტექნოლოგიის არსი და ობიექტ-ორიენტირებული დაპროგრამების ძირითადი ცნებები

1.1. დაპროგრამების თანამედროვე პლატფორმები და ენები

ობიექტ-ორიენტირებული, კომპონენტურ-ვიზუალური დაპროგრამე-ბის თანამედროვე ინფორმაციული ტექნოლოგიებიდან განსაკუთრებული აქტუალობით ხასიათდება ფირმის „მაიკროსოფთი“ პროგრამული პლატფორმა Visual Studio .NET (პროგრამებით C#, VB, C++, J++). ასეთი ინტეგრირებული პროგრამული პაკეტების გამოყენების მიზანია რთული სისტემების მოდელირება, კონსტრუირება და რეალიზაცია უნიფიცირებული პროგრამირების კონცეფციის გამოყენებით.

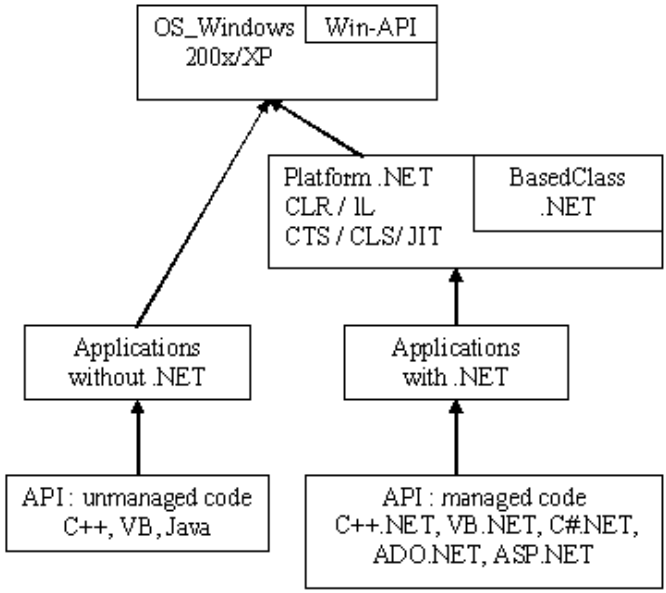
წიგნის მოცემულ თავში მოკლედ განიხილება ამ ორი პროგრამული სისტემის ძირითადი არსი, დანიშნულება, სტრუქტურა და გამოყენების სამუშაო გარემო.

1.1.1. პლატფორმა .NET

მაიკროსოფტის უახლესი პროგრამული ტექნოლოგია .NET (დოტ-ნეთ) პლატფორმის სახით სულ უფრო ფართოდ იკიდებს ფეხს მსოფლიოს მოწინავე ქვეყნების საუნივერსიტეტო-სამეცნიერო და საწარმოო ფირმების ბიზნესის სფეროებში. იგი გამოიყენება Windows, Unix და Linux ოპერაციული სისტემებისათვის.

.NET პლატფორმა შეიქმნა სპეციალურად განაწილებული გამოყენებითი სისტემების ასაგებად დიდი მოცულობის ინფორმაციის დასამუშავებლად კლიენტ-სერვერ არქიტექტურის ბაზაზე. იგი არის გამოყენებითი პროგრამული დანართების (API) სამუშაო გარემო, რომელიც

შეთანხმებულად ფუნქციონირებს Windows-ოპერაციულ სისტემასთან (იხ. ნახ.1.1)



ნახ.1.1

ნახაზიდან ჩანს, რომ Windows-სისტემა უშუალოდ მუშაობს C++, VB, Java და სხვა ენებზე დაწერილ პროგრამულ API-დანართებთან (Application Programming Interface-გამოყენებითი დანართების დაპროგრამების ინტერფეისი), რომლებიც რეალიზებულია როგორც უმართავი კოდები (unmanaged code). ამასთანავე იგი მუშაობს C#.NET, C++.NET, VB.NET და ა.შ., ზოგადად .NET-პლატფორმის მიერ მართვად (managed code) პროგრამულ დანართებთან.

მართვაში იგულისხმება ის, რომ ეს კოდები ამუშავდება უშუალოდ .NET-ის მიერ, იმართება მათი პროცესებისა და მონაცემთა ნაკადები, მიეწოდება შესასრულებლად საჭირო დამხმარე რესურსები და ა.შ.

პრინციპში, NET-პლატფორმა ასრულებს „ოპერაციული სისტემის“ გარკვეულ ფუნქციებს და მოქნილად ფუნქციონირებს Windows-თან.

ამავე ნახაზზე საყურადღებოა თვით NET-პლატფორმის ბლოკი. რომელშიც ძირითადი ქვებლოკი Based Class.NET არის ამ პლატფორმის საბაზო კლასების ბიბლიოთეკა (უმრავლესობა დაწერილია C#-ენაზე). იგი სრულად ობიექტ-ორიენტირებულია, შედგება ობიექტთა ერთობლიობისგან, რომელთაგანაც თითოეულში რეალიზებულია განსაზღვრულ მეთოდთა ჯგუფები. მაგალითად, ფანჯრებისა და ფორმების ასახვა (Windows GUI), მონაცემთა ფაილებთან ურთიერთობა (ADO.NET), ვებ-გვერდების ორგანიზება და ინტერნეტთან კავშირი (ASP.NET) და სხვ.

ამავე ბლოკში ნაჩვენებია .NET-runtime - პლატფორმის სამუშაო გარემო (რომელშიც სრულდება პროგრამა), ანუ CLR(Common Language Runtime) და მას შესრულების საერთო გარემოსაც უწოდებენ. ესაა პროგრამული უზრუნველყოფა მომხმარებელთა გამოყენებითი პროგრამების შესასრულებლად.

CTS საერთო ტიპების სისტემაა (Common Type System), რომლის საფუძველზეც NET-პლატფორმა უზრუნველყოფს დაპროგრამების სხვადასხვა ენის თავსებადობას. ამასთანავე CTS აღწერს მომხმარებელთა კლასების განსაზღვრის წესებსაც.

IL შუალედური გარდაქმნის ენაა (Intermediate Language). პროგრამები, რომელთა საწყისი კოდები დაწერილია, მაგალითად C#, C++, J++ ან VB ენებზე .NET-ში, კომპილატორი ამ მართვად კოდებს გადაიყვანს შუალედურ IL-ენაზე, რომელთაც შემდეგ CTS სწრაფად აკომპილირებს მანქანურ კოდში. ამგვარად, ობიექტური კოდები IL-ენის საშუალებით ისე მიიღება, რომ მათში არაა დაფიქსირებული, თუ რომელ ენაზეა დაწერილი საწყისი კოდი.

CLS ენის საერთო სპეციფიკაციაა (Common Language Specification), ანუ იმ სტანდარტების მინიმალური ერთობლიობა, რომელიც უზრუნველყოფს კოდებთან მიმართვას .NET-ის

ნებისმიერი ენიდან. ამ ენების ყველა კომპილატორს გააჩნია CLS მხარდაჭერა.

JIT (Just-In-Time) ესაა შუალედური კოდის კომპილაციის ფაზა მანქანურ კოდში. სახელწოდება მიუთითებს იმაზე, რომ კოდის მხოლოდ

იმ ცალკეული ნაწილების კომპილაცია ხდება, რომლებიც საჭიროა პროგრამის შესასრულებლად დროის მოცემულ მომენტში.

1.1.2. საერთო ტიპების სისტემა (CTS)

მაიკროსოფთის NET-პლატფორმისა და IL-შუალედური ენის არსებობის კონცეფციის საფუძველია ძირითადად ენის ობიექტ-ორიენტულობა და საერთო ტიპების სისტემის არსებობა, რომელთაც კომპილატორები ფლობს.

C# („სი შარფ“) ენა ობიექტ-ორიენტირებული ენების ერთ-ერთი ახალი და მძლავრი წარმომადგენელია, რომელიც შეიქმნა სპეციალურად NET-პლატფორმისათვის და თავსებადია Windows-ის თანამედროვე ვერსიებთან და ინტერნეტთან. ამ ენაზეა რეალიზებული NET-პლატფორმის უმრავლესი საბაზო კლასები.

როგორც ცნობილია, C++ ენა კომპილირდება ასემბლერულ კოდში, C# ენა კი - შუალედურ IL-ენაში.

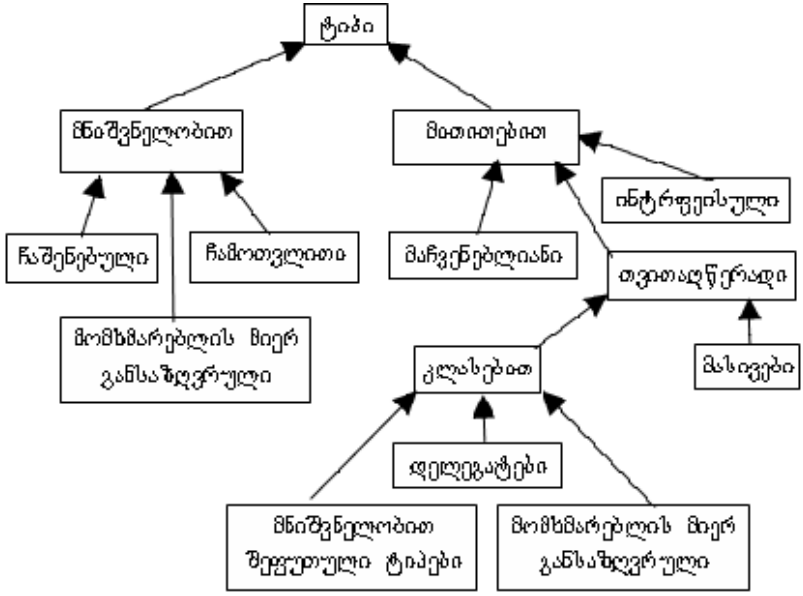
IL-ენის დანიშნულებაა პლატფორმული და ენობრივი დამოუკიდებლობის განხორციელება ობიექტ-ორიენტირებულ გარემოში. Java-ენაც უზრუნველყოფს პლატფორმულ (Windows, Unix, Linux) დამოუკიდებლობას, მაგრამ მისი ბაიტ-კოდის შესრულების ეტაპზე იგი ინტერპრეტირდება (IL -კი კომპილირდება).

NET-პლატფორმისათვის ენობრივი თავსებადობა ხორციელდება IL ენაში არსებული ტიპების დიდი რაოდენობით, რომლებიც ორგანიზებულია ტიპთა იერარქიის ობიექტ-ორიენტირებული პრინციპებით. 1.2 ნახაზზე ილუსტრირებულია ტიპთა ასეთი იერარქია მემკვიდრეობითობის კავშირის გამოყენებით.

მოვიტანოთ ზოგიერთი კომენტარი, რომელიც ახსნის ნახაზს:

- **ტიპი** არის საბაზო კლასი, რომელიც ზოგადად ასახავს ნებისმიერ ტიპს;

- **ტიპი მნიშვნელობით** არის საბაზო კლასი, რომელიც ზოგადად ასახავს ნებისმიერ ტიპს მნიშვნელობით;



ნახ.1.2. ტიპების ზოგადი სისტემა

- **ჩაშენებული ტიპები მნიშვნელობით** არის სტანდარტული საბაზო ტიპები, რომლებიც აღწერს რიცხვებს, სიმბოლოებსა და ლოგიკურ მნიშვნელობებს;

- **ჩამოთვლით ტიპი** არის ჩამონათვალთა ერთობლიობა, რომელშიც თითოეულ მნიშვნელობას შეესაბამება რიცხვითი მნიშვნელობა (0,1,... და ა. შ.) მისი მდებარეობის მიხედვით;

- **მომხმარებლის მიერ განსაზღვრული ტიპი** არის საწყის კოდში (მომხმარებლის პროგრამაში) აღწერილი ტიპები,

რომლებიც ინახება მნიშნელობებით (ესაა მაგალითად, სტრუქტურები).

- **ტიპი მითითებით** არის მონაცემთა ნებისმიერი ტიპები, რომელთანაც მიმართვა ხორციელდება მიმთითებლებით და ინახება ნაკადში;

- **თვითაღწერადი ტიპები** არის ტიპები, რომლებიც ასახავს ინფორმაციას თავიანთ შესახებ;

- **თვითაღწერადი ტიპები** არის ტიპები, რომლებიც ასახავს ინფორმაციას თავიანთ შესახებ;

- **მასივები** არის ნებისმიერი ტიპი, რომელიც შეიცავს ობიექტების მასივს;

- **ტიპები კლასებით** არის თვითაღწერადი ტიპები, რომლებიც არაა მასივები;

- **დელეგატები** - ტიპებია, რომლებიც დამუშავდა მიმთითებელთა შესანახად კლასის მეთოდებისათვის;

- **მნიშვნელობით შეფუთული ტიპები** არის ტიპები მნიშვნელობით, რომლებიც დროებით დაიყვანება ტიპებამდე მიმთითებლით, რათა შენახულ იქნას ნაკადში;

- **მომხმარებლის მიერ განსაზღვრული ტიპები მითითებით** არის ტიპები, განსაზღვრული საწყის კოდში, როგორც ტიპები მითითებით. პროგრამაში ესაა მაგალითად, ნებისმიერი კლასი.

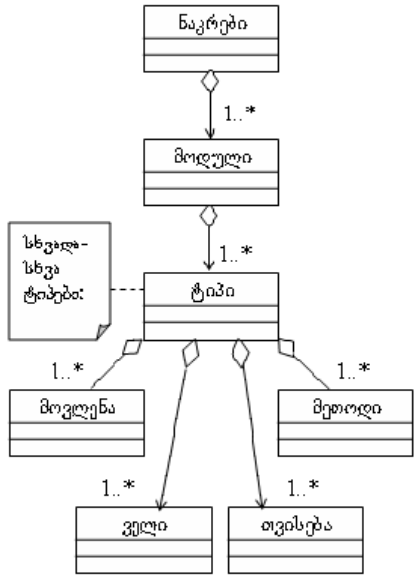
1.1.3. .NET-პლატფორმის კომპონენტები და მისი დანართის სტრუქტურა

როგორც აღვნიშნეთ, .NET პლატფორმის საბაზო კლასების დიდი ნაწილი დაწერილია C# ენის გამოყენებით, ამიტომაც საილუსტრაციო მაგალითებს ამ ენაზე გავუკეთებთ კომენტარს..

.NET პლატფორმის კომპონენტებიდან ერთ-ერთი მთავარი ბლოკია Assembly (ანაწყობი, ნაკრები), რომელიც ლოგიკურად აერთიანებს კოდს, რესურსებს და მეტამონაცემებს. იგი ლოგიკური და არა ფიზიკური ერთეულია, რადგან შეუძლია

მოთავსდეს რამდენიმე ფაილში. ასეთ შემთხვევაში არსებობს ერთი მთავარი ფაილი, რომელშიც ინახება ინფორმაცია დანარჩენებზე.

1.3 ნახაზზე ნაჩვენებია პროგრამული დანართის (Application) შესაბამისი ნაკრების ზოგადი იერარქიული სტრუქტურა აგრეგაციის კავშირის გამოყენებით.



ნახ.1.3. ნაკრების ზოგადი სტრუქტურა

ნახაზიდან ჩანს, რომ ნაკრები 1 ან რამდენიმე (1..*) მოდულისგან (module) შედგება. სწორედ მოდულში ინახება დანართის ან ბიბლიოთეკის კოდი, მისი მეტამონაცემებით. მოდულები შეიცავს ტიპებს. ესაა კოდის შაბლონები (კლასები), რომლებშიც ინკაფსულირებულია გარკვეული მონაცემები და მეთოდები. როგორც წინა პარაგრაფში ვახსენეთ, ტიპები ორი სახისაა: მიმთითებლებით (ანუ კლასები) და მნიშვნელობებით (ანუ სტრუქტურები).

ტიპებს აქვთ ველები, თვისებები და მეთოდები. ველი გამოყოფს მეხსიერების ადგილს შესაბამის მონაცემთა ტიპისათვის. თვისებები ველების მსგავსია, ოღონდაც მათი

დანიშნულებაა შესაბამის მონაცემთა საწყისი მნიშვნელობების განსაზღვრა და კონტროლი.

მეთოდები განსაზღვრავს მონაცემთა დასამუშავებლად კლასის ქცევას, ანუ რეაქციას გარედან შემოსულ შეტყობინებაზე (მოთხოვნაზე). შეტყობინება ინაფორმაციაა, რომელიც ამა თუ იმ მოვლენის შედეგად ფორმირდება.

პროგრამული ნაკრები შეიძლება იყოს ორი ტიპის: კერძო და საერთო გამოყენების. პირველ შემთხვევაში ნაკრები ინსტალირდება კერძო მომხმარებლს კატალოგში და მასთან სხვა მიმართვები გამორიცხულია.

საერთო გამოყენების ნაკრები შეიცავს პროგრამულ ბიბლიოთეკებს, რომელთაც იყენებს სხვადასხვა დანართი. აქ საჭიროა სპეციალური დაცვის მექანიზმების გამოყენება (სახელების კოლიზიისა და ნაკრებთა ვერსიების კონტროლის თვალსაზრისით).

კლასებს შორის სახელთა კოლიზიის აღმოფხვრის მიზნით .NET პლატფორმა იყენებს „სახელთა სივრცეს“.

სახელთა სივრცე (namespace): ესაა მონაცემთა ტიპების უბრალო დაჯგუფება. ყველა მონაცემთა ტიპის სახელს მოცემულ სახელთა სივრცეში ავტომატურად ემატება პრეფიქსი, რომელიც შედგენილია სახელთა სივრცის დასახელებისგან. ასევე შესაძლებელია ჩადგმული სახელთა სივრცეების შექმნა.

მაგალითად, საბაზო კლასების უმრავლესობისათვის, რომლებიც ზოგადი გამოყენებისთვისაა დანიშნული, მოთავსებულია სახელთა სივრცეში System, ვებ-გვერდებისათვის - System.Web და ა.შ.

C#-ის პროგრამის ტექსტის მაგალითზე შეიძლება შემდეგი კომენტარის გაკეთება:

```
namespace Magazia.Web  // აქ მითითებულია სახელი Magazia.Web
    {
      public class Checkout : PageBase
      {
        // და ა.შ.
```

დანართთა არეები (application area) არის .NET პლატფორმის მნიშვნელოვანი ელემენტი. მათი დანიშულებაა ერთდროულად და ერთმანეთთან მომუშავე დანართების იზოლაცია, რათა არ მოხდეს მონაცემთა არასასურველი დამუშავება.

პროგრამული დანართების იზოლაციისათვის Windows გამოიყენებს **„პროცესის“** ცნებას, რომელიც მისამართების სივრცეს ეხება. ყოველ პროცესს გამოეყოფა 4 გიგაბაიტი ვირტუალური მეხსიერება. ისინი დისკზე სხვადასხვა ფიზიკური მისამართებითაა და არ გადაიკვეთება.

პროცესებს აქვს მინიჭებული განსაზღვრული პრივილეგიები და ოპერაციული სისტემა აკონტროლებს მათ, თუ რომელ ოპერაციას რომელი პროცესის გამოყენება შეუძლია.

დანართთა არეების გამოყენების იდეა მდგომარეობს იმაში, რომ პროცესებს შორის მოხერხდეს მონაცემთა გაცვლა. ამიტომაც პროცესი იყოფა რამდენიმე დანართის არედ. თითოეულ დანართის არეში თავსდება ერთი დანართის კოდი.

.NET პლატფორმის მნიშვნელოვანი საშუალებაა JIT (Just-In-Time) კომპილატორი. იგი ახორციელებს პროგრამული კოდის ცალკეული ნაწილის დროულად კომპილირებას (საჭიროების შემთხვევაში).

Visual Studio.NET არის პროგრამული სისტემების დამუშავების ინტეგრირებული გარემო, რომელშიც შესაძლებელია კოდების დაწერა, კომპილირება და გამართვა VB.NET, C++.NET, C#.NET, ASP.NET, ADO.NETO და სხვა ტექნოლოგიებით.

ახლა მოვიტანოთ C++ და C# მარტივი კოდები, რათა დავინახოთ მსგავსება-განსხვავება ამ ენების სინტაკსებს შორის:

a) C++ -ის კოდის ფრაგმენტი:

```
#include <iostream.h> // C++ ----------
#include <Windows.h>
int main(int argc, char *argv)
{
 cout << "Hello, my friend ! ";
MessageBox(NULL, "By-By !", "", MB_OK);
return 0;
}
```

b) C#-ის კოდის ფრაგმენტი:

```
using System; // C# ----------
using System.Windows.Forms;
namespace Console1;
{
  class Class1
  {
    static int Main(string[] args)
      {
        Console.WriteLine("Hello, my friend !");
          MessageBox.Show("By-By !");
         return 0;
     }
  }
}
```

1.2. ობიექტ-ორიენტირებული დაპროგრამება და პოლიმორფიზმი

1.2.1. შესავალი ობიექტ-ორიენტირებულ დაპროგრამებაში

.NET გარემოში დაპროგრამება დაფუძნებულია ობიექტებზე. ობიექტი (object) – არის პროგრამული კონსტრუქცია, რომელშიც ინკაფსულირებულია ლოგიკურად დაკავშირებული მონაცემებისა და მეთოდების ერთობლიობა. ობიექტები ავტონომიური ელემენტებია, ისინი იქმნება შაბლონით (template), რომელსაც ობიექტ-ორიენტირებულ დაპროგრამების თეორიაში კლასს (class) უწოდებენ. .NET-ის საბაზო კლასების ბიბლიოთეკა არის კლასთა ერთობლიობა, რომელიც გამოიყენება ობიექტების შესაქმნელად მომხმარებლის კერძო აპლიკაციაში. ამასთანავე პროგრამული პაკეტი Visual Studio საშუალებას იძლევა შეიქმნას საკუთარი კლასები კერძო პროგრამებისთვის.

- **ობიექტები, წევრები და აბსტრაგირება:**

ობიექტი – პროგრამული კონსტრუქციაა, რომელიც ასახავს განსაზღვრულ არსს (Entity). ესაა რეალურ სამყაროში არსებული ობიექტები, მაგალითად, ადამიანები, მანქანები, ფირმები, ცხოველები, ფრინველები, მცენარეები, ინსტიტუტები, კომპიუტერები და ა.შ. ყოველ ობიექტს აქვს განსაზღვრული, მისთვის დამახასიათებელი ფუნქციონალობა და თვისებები.

პროგრამულ აპლიკაციაში ობიექტი შეიძლება იყოს ფორმა, მართვის ელემენტი, მაგ., ღილაკი, კომბობოქსი, ბაზასთან შეერთება და სხვ.

ამგვარად, ობიექტი არის დასრულებული ფუნქციონალური ერთეული, რომელიც შეიცავს ყველა მონაცემს და ყველა ფუნქციას, რომლებიც აუცილებელია იმ ამოცანის გადასაწყვეტად, რომლისთვისაც ეს ობიექტი გამოიყენება.

რეალური სამყაროს ობიექტების წარმოდგენას პროგრამული ობიექტებით უწოდებენ აბსტრაგირებას (abstraction).

- **კლასები, როგორც ობიექტთა შაბლონები:**

კლასი არის ერთგვაროვან ობიექტთა ერთობლიობა, რომელიც ამ ერთობლიობის სტრუქტურას ასახავს. კლასი განსაზღვრავს ობიექტთა წევრებს და მათ ყოფაქცევას, აგრეთვე საწყის მონაცემთა მნიშვნელობებს, საჭიროების შემთხვევაში.

კლასის ეგზემპლარის შექმნისას კომპიუტერის მეხსიერებაში შეიქმნება ამ კლასის ასლი. ასეთი სახით შექმნილ კლასის ეგზემპლარს უწოდებენ ობიექტს. დაპროგრამების ენაში მისი შექმნა ხდება ოპერატორით new. მაგალითად:

```
// გამოცხადდეს ცვლადი MyDataForm ტიპით DataForm
DataForm MyDataForm;
// შეიქმნას ობიექტს ეგზემპლარი DataForm და
// ჩაიწეროს იგი ცვლადში MyDataForm
MyDataForm = new DataForm( );
```

- **ობიექტები და წევრები:**

ობიექტები შედგება წევრებისგან, რომელთაც ეკუთვნის თვისებები, ველები, მეთოდები და მოვლენები. ისინი განსაზღვრავს ობიექტის მონაცემებს და ფუნქციონალობას.

ველები და თვისებები შეიცავს ობიექტის მონაცემებს, რომლებიც მის მდგომარეობას ასახავს. მეთოდები განსაზღვრავს მოქმედებებს, რომელთა შესრულება შეუძლია ობიექტს. მოვლენები კი წარმოადგენს შეტყობინებებს, რომელთაც მიიღებს ობიექტი ან გადასცემს სხვა ობიექტებს. მოვლენის დანიშნულებაა გააქტიუროს ობიექტის ესა თუ ის მეთოდი, რომელიც გადაამუშავებს მის მონაცემებს.

მაგალითად, ობიექტისთვის „ავტომობილი“ თვისებები და ველებია: მოდელი, ფერი, ასაკი, საწვავის–ხარჯი და ა.შ. ეს მონაცემები ასახავს ობიექტის მდგომარეობას. მეთოდის მაგალითებია: სიჩქარის–გადართვა, მუხრუჭის–დაჭერა, რულის–მობრუნება და სხვ., ისინი ობიექტის ქცევას

განსაზღვრავს. ავტომობილისთვის მოვლენები მიიღება შეტყობინების სახით გარედან (მაგ., ინფორმაცია გზის მოსახვევის შესახებ, ან სიჩქარის შეზღუდვის შესახებ) ან მანქანის მდგომარეობის ამსახველი ნათურებიდან (მაგ., წყლის გადახურება, საწვავის დამთავრება და ა.შ.).

- **ობიექტური მოდელები:**

მარტივ ობიექტებს აქვს რამდენიმე თვისება და მეთოდი, ასევე ერთი-ორი მოვლენა. რთულ ობიექტებს გააჩნია მეტი რაოდენობა თვისებების, მეთოდებისა და მოვლენების, აგრეთვე მათ შეიძლება ჰქონდეს „შვილი“ ობიექტებიც, რომლებზეც შეუძლია პირდაპირი მიმართვის განხორციელება. მაგალითად, მართვის ელემენტს TextBox, აქვს თვისება Font, რომელიც არის Font-ტიპის ობიექტი. ამ თვალსაზრისით, ნებისმიერი Form კლასის ეგზემპლარი მოიცავს მასზე განთავსებულ მართვის ელემენტებს (Controls ერთობლიობა).

„მშობელი-შვილი“ ჩადგმული ობიექტების იერარქია, რომელიც ქმნის ობიექტის სტრუქტურას, არის ობიექტური მოდელი (object model).

მაგალითად, ობიექტი *ავტომობილი* შედგება რამდენიმე „შვილი“ ობიექტისგან: *ძრავი, ბორბლები, ძარა* და ა.შ. ობიექტის *ავტომობილი* ყოფაქცევა, რომელთაც „შვილი“ ობიექტისთვის *ძრავი* თვისებაში *ცილინდრების რაოდენობა* აქვს 4 და 8, იქნება განსხვავებული.

„შვილი“ ობიექტი შეიძლება შედგებოდეს თავის საკუთარი „შვილი“ ობიექტებისგან და ა.შ. მაგალითად, *ავტომობილი -> ძრავი -> სანთლები.*

- **ინკაფსულაცია:**

ობიექტ-ორიენტირებული დაპროგრამების (ოოდ) ერთ-ერთი საბაზო პრინციპია ინკაფსულაცია, რომლის არსი მდგომარეობს ობიექტის რეალიზაციის გამოყოფაში მისი ინტერფეისისგან. ანუ პროგრამული დანართი (აპლიკაცია) ურთიერთქმედებს ობიექტთან

მისი ინტერფეისის საშუალებით, რომელიც შედგება ღია თვისებებისა და მეთოდებისაგან.

ობიექტები ერთმანეთთან ურთიერთმოქმედებს თავიანთი ღია თვისებებით და მეთოდებით, ამიტომაც ობიექტს უნდა ჰქონდეს ყველა აუცილებელი მონაცემი და მეთოდების ერთობლიობა, ამ მონაცემთა დასამუშავებლად.

ინტერფეისმა არ უნდა მისცეს ობიექტის შიგა მონაცემებთან წვდომის უფლება, რაც გამორიცხავს ამ შემთხვევაში ობიექტის შიგა მონაცემებისთვის public-მოდიფიკატორის გამოყენებას (გამოიყენება privat).

- **პოლიმორფიზმი:**

პოლიმორფიზმი მრავალფორმიანობას ნიშნავს და მას ობიექტ-ორიენტირებულ დაპროგრამებაში განსაკუთრებული როლი აქვს. პოლიმორფიზმის საშუალებით შესაძლებელია ერთიდაიგივე გახსნილი ინტერფეისის სხვადასხვაგვარი რეალიზაცია სხვადასხვა კლასში. ანუ პოლიმორფიზმის საშუალებით შესაძლებელია ობიექტის მეთოდებისა და თვისებების გამოძახება მათი რეალიზაციის მიუხედავად.

მაგალითად, ობიექტი *მძღოლი* ურთიერთქმედებს ობიექტთან *ავტომობილი* იმავე სახელის მქონე ღია ინტერფეისის საშუალებით. თუ სხვა ობიექტი, მაგალითად, *სატვირთო* ან *სპორტული–მანქანა* ფლობს ამ ღია ინტერფეისის მხარდაჭერას, მაშინ ობიექტს *მძღოლი* შეუძლია მათთანაც ურთიერთქმედება, მიუხედავად ინტერფეისის განსხვავებული რეალიზაციისა

პოლიმორფიზმის რეალიზაციის ორი ძირითადი მიდგომა არსებობს: ინტერფეისების და მემკვიდრეობითობის გამოყენებით. განვიხილოთ თვითოეული ცალ-ცალკე.

ინტერფეისი (interface) – ესაა შეთანხმება, რომელიც ობიექტის ყოფაქცევას განსაზღვრავს. იგი ადგენს კლასის წევრთა სიას, მაგრამ არაფერს ამბობს მათი რეალიზაციის შესახებ. ობიექტში დასაშვებია რამდენიმე ინტერფეისის რეალიზაცია, ხოლო ერთიდაიგივე ინტერფეისი შეიძლება რეალიზებულ იქნას სხვადასხვა კლასში.

ნებისმიერ ობიექტებს, რომლებშიც რეალიზებულია რომელიმე ინტერფეისი, შეუძლია ერთმანეთთან ურთიერთმოქმედება მისი საშუალებით. მაგალითად, ობიექტში *ავტომობილი*, რომელზეც ჩვენ ვსაუბრობთ, შეიძლება *IDrivable*-ინტერფეისის რეალიზაცია (ინტერფეისთა სახელები მიღებულია დაიწყოს „I“ ასოთი), მეთოდებით: *მოძრაობა–წინ, მოძრაობა–უკან, გაჩერება*. იგივე ინტერფეისი შეიძლება რეალიზდეს სხვა კლასებშიც, როგორიცაა, მაგალითად, *სატვირთო–მანქანა* ან *კატერი*. შედეგად, ამ ობიექტებს შეუძლია ურთიერთმოქმედება ობიექტთან *მძღოლი*. ეს უკანასკნელი მთლიანად უხილავია ინტერფეისის რეალიზაციისთვის, რომელთანაც იგი მოქმედებს, მისთვის ცნობილია მხოლოდ ინტერფეისი.

მემკვიდრეობითობა იძლევა ახალი კლასების შექმნის საშუალებას არსებულის ბაზაზე. ამასთანავე ახალ კლასებში ნებადართულია ძველი კლასების სრული ფუნქციონალობის ჩართვა და, საჭიროების შემთხვევაში, შესაძლებელია მათი წევრების მოდიფიცირებაც.

კლასი, რომელიც გამოცხადებულია სხვა კლასის საფუძველზე, უწოდებენ წარმოებულ კლასს (derived class). ყოველ კლასს შეიძლება ჰქონდეს მხოლოდ ერთი პირდაპირი წინაპარი – მისი საბაზო კლასი (base class). წარმოებულ კლასს აქვს საბაზო კლასის წევრების ერთობლიობა. ასევე შესაძლებელია ახალი წევრების დამატება და ძველი წევრების (მემკვიდრეობით მიღებული) რეალიზაციის ცვლილებაც. წარმოებული კლასები ინარჩუნებს თავისი საბაზო კლასის ყველა მახასიათებელს და უნარი აქვს სხვა ობიექტებთან ისეთი ურთიერთმოქმედებისათვის, როგორც საბაზო კლასის ეგზემპლარებს.

მაგალითად, საბაზო კლასიდან *ავტომობილი* შეიძლება წარმოებული კლასის *სპორტული–ავტომობილი* გამოცხადება, რომელიც თავის მხრივ, როგორც საბაზო კლასი, შეძლებს ახალი წარმოებული კლასის *სპორტული–კაბრიოლეტი* გამოცხადებას. ყოველ წარმოებულ კლასში დასაშვებია ახალი წევრების (თვისებები, მეთოდები, მოვლენები) შემოტანა, ამსთანავე ისინი ინარჩუნებს საწყისი საბაზო კლასის *ავტომობილი* ფუნქციონალობას უცვლელი ფორმით.

1.2.2. წევრების გადატვირთვა

გადატვირთვის შედეგად შესაძლებელია კლასის რამდენიმე წევრის შექმნა ერთიდაიმავე სახელით, ოღონდ განსხვავებული სიგნატურით (მოქმედებით). გადატვირთვას უფრო ხშირად იყენებენ მეთოდებისათვის, ხოლო რეალიზებილა ასევე ოპერატორების გადატვირთვაც. ამ პარაგრაფში ჩვენი მიზანია კლასის გადატვირთული წევრების შექმნა.

მაგალითად, თუ კლასს სჭირდება ისეთი წევრი, რომელსაც შეუძლია სხვადასხვა პარამეტრების მიღება, ექნება შემდეგი სახე:

```
public void Display(int DisplayValue)
 {
  // რეალიზაცია გამოტოვილია
 }
```

ეს მეთოდი იფუნქციონირებს ნორმალურად, თუ მის ცვლადს მიეწოდა მთელი ტიპის მნიშვნელობა. სხვა ტიპის (მაგ., სტრიქონული ცვლადი), ან ერთზე მეტი პარამეტრის შემთხვევაში ის ვეღარ შეასრულებს თავის საქმეს.

ამიტომაც იყენებენ გადატვირთვას, ანუ რამდენიმე მეთოდის შექმნას ერთი სახელით.

გადატვირთულ მეთოდებს შეუძლია ერთნაირი ტიპის მნიშვნელობების დაბრუნება, მათი გამოცხადება ერთნაირი წვდომის მოდიფიკატორებით, მაგრამ გადატვირთული მეთოდების სიგნატურები უნდა იყოს განსხვავებული.

გადატვირთული მეთოდის გამოძახებისას შესრულების გარემო ამოწმებს გადაცემულ არგუმენტების ტიპებს, ადარებს არგუმენტების სიას არსებულ გადატვირთულ მეთოდთა სიგნატურებთან და იძახებს იმას, რომლის სიგნატურაც დაემთხვევა არგუმენტთა სიას. თუ არც ერთი არ დაემთხვა, მაშინ ფორმირდება შეცდომა.

- **გადატვირთული მეთოდების შექმნა:**

გადატვირთული მეთოდები იქმნება ისევე, როგორც ყველა სხვა მეთოდი. უპირველესად ყოვლისა საჭიროა მეთოდის გამოცხადება რაღაც სახელით, წვდომის მოდიფიკატორით, დასაბრუნებელი მნიშვნელობის ტიპით და არგუმენტების სიით.

გადატვირთულ მეთოდს უნდა ჰქონდეს იგივე სახელი, რაც აქვს არსებულ მეთოდს, ოღონდ მას ექნება სხვა სიგნატურა. მაგალითად:

```
// გადატვირთული მეთოდის მაგალითი
public void DisplayMessage(int I)
{
 MessageBox.Show(I.ToString());
}
// შემდეგ მეთოდს აქვს იგივე სახელი და სხვა სიგნატურა
public void DisplayMessage(string S)
{
 MessageBox.Show(S);
}
```

აქ განსაზღვრულია ორი მეთოდი ერთნაირი სახელით, ოღონდ სხვადასხვა სიგნატურით და რეალიზაციით. **DisplayMessage** მეთოდის გამოძახებისას შესრულების გარემო შეამოწმებს გადაცემულ არგუმენტის ტიპს. თუ ესაა **String**, მაშინ ის გამოიძახებს მე-2 მეთოდს, ხოლო თუ **int**, - 1-ელს.

ამგვარად, გადატვირთული მეთოდის შესაქმნელად საჭიროა ახალი მეთოდის გამოცხადება ძველი სახელით, ოღონდ სიგნატურა არ უნდა ემთხვეოდეს არცერთ ძველს. წვდომის მოდიფიკატორისა და დასაბრუნებელი ტიპისთვის ეს შეზღუდვები მოხსნილია. შემდეგ ახალი მეთოდისთვის იწერება რეალიზაცია.

- **ოპერატორების გადატვირთვა:**

მომხმარებლის მონაცემთა ტიპებთან მუშაობისას მოსახერხებელია არითმეტიკული და ლოგიკური ოპერატორების

განსაზღვრა, აგრეთვე შედარების ოპერატორებისა, რომლებიც მუშაობს აღნიშნულ ტიპებთან. განვიხილოთ შემდეგი სტრუქტურა:

```
public struct HoursWorked
{
  float RegularHours;
  float OvertimeHours;
}
```

ასეთი მარტივი სტრუქტურის გამოყენება შესაძლოა ბუღალტრულ აპლიკაციაში სამუშაო საათებისა და ზედმეტი საათების (ზეგანაკვეთური სამუშაოს) აღრიცხვის მიზნით. მაგრამ ასეთი სტრუქტურის რამდენიმე ეგზემპლართან მუშაობისას შესაძლოა სირთულეების აღმოცენება. მაგალითად, დავუშვათ, რომ საჭიროა ასეთი სტრუქტურის ორი ეგზემპლარის მნიშვნელობათა შეჯამება. ასეთ შემთხვევაში აუცილებელია ახალი მეთოდის დაწერა, რომელსაც შეეძლება მისი გადაჭრა. თუ საჭიროა სამი ან მეტი ეგზემპლარის მნიშვნელობათა შეჯამება, მაშინ ეს ახალი მეთოდი უნდა გამოვიძახოთ რამდენჯერმე, რაც არაეფექტურია.

C# საშუალებას იძლევა განისაზღვროს ოპერატორის ქცევა მომხმარებელთა მონაცემეთა ტიპების დასამუშავებლად. მაგალითად, თუ საჭიროა სტრუქტურის შეჯამება ჩვენი მაგალითისთვის, შესაძლებელია „+“ ოპერატორის გადატვირთული ვერსიის შექმნა, რაც მოითხოვს ამ ოპერატორში აუცილებელი ფუნქციონალობის ჩამატებას.

გადატვირთული ოპერატორები იქმნება გასაღებური სიტყვით operator, რომელსაც შემდეგი სინტაკსი აქვს:

```
public static type operator op (Argiment1[,  Argument2])
{
  // რეალიზაცია
}
```

სადაც ***type*** ტიპია, რომელთანაც მუშაობს ოპერატორი. ის ამავდროულად არის ოპერატორის მიერ დაბრუნებული

მნიშვნელობის ტიპი; **Argiment1** და **Argiment2** ოპერატორის არგუმენტებია.

უნარული ოპერატორი ამუშავებს ერთ არგუმენტს ***type*** ტიპით. ბინარული ოპერატორები თხოულობს ორ არგუმენტს, რომელთაგან ერთი მაინც ***type*** ტიპისაა.

op - ეს თვით ოპერატორია, მაგალითად, { + , - , > , < , ! = და ა.შ.}.

გადატვირთული ოპერატორები უნდა გამოცხადდეს public და static მოდიფიკატორებით, რათა სხვა მომხმარებლებმა შეძლონ მათი გამოყენება.

გადატვირთული ოპერატორი უნდა გამოცხადდეს მომხმარებლის მონაცემთა ტიპის განსაზღვრის შიგნით, რომელთანაც უნდა იმუშაოს ოპერატორმა.

ჩვენ განვიხილეთ გადატვირთული ოპერატორის მაგალითი სტრუქტურასთან, მაგრამ იგი გამოიყენება ასევე კლასებთან. ახლა განვიხილოთ გადატვირთული ოპერატორი „+“, რომელიც შეიქმნა ზემოთ **HoursWorked** სტრუქტურასთან სამუშაოდ:

```
public struct HoursWorked
{
 float RegularHours;
 float OvertimeHours;
 // გადატვირთული ოპერატორი გამოცხადებულ უნდა იქნას
კლასის განსაზღვრებაში,
// რომელთანაც მან უნდა იმუშაოს
public static HoursWorked operator + (HoursWorked  a,
HoursWorked  b)
  {
    HoursWorked Result = new HoursWorked( );
    Result RegularHours = a.RegularHours + b.RegularHours;
    Result  OvertimeHours = a.OvertimeHours +
b.OvertimeHours;
    return Result;
  }
}
```

გადატვირთულ ოპერატორს პროგრამაში იყენებენ ისევე, როგორც სხვა ჩვეულებრივ ოპერატორს. ქვემოთ ნაჩვენებია კოდის მაგალითი, რომელიც ასრულებს **HoursWorked** სტრუქტურის ორი ეგზემპლარის შეკრებას.

```
// გათვალისწინებულია, რომ ცვლადები Sunday და Monday შეიცავს HoursWorked
// სტრუქტურის ეგზემპლარებს, რომლრბიც ინიცირებულია შესაბამისი მნიშვნელობებით.
HoursWorked  total = new HoursWorked();
total = Sunday + Monday;
```

გადატვირთული მეთოდების მსგავსად შესაძლებელია გადატვირთული ოპერატორების განსხვავებული რეალიზაციების შექმნა, იმ პირობით, რომ მათი სიგნატურები განსხვავებულია.

დავუშვათ, რომ **HoursWorked** სტრუქტურისათვის დაგვჭირდა int ტიპის მნიშვნელობის დამატება ველისათვის **NormalHours** „+“ ოპერატორის საშუალებით. ასეთი ამოცანის გადაჭრა შეიძლება კიდევ ერთი გადატვირთული „+“ ოპერატორის ვერსიის შექმნით.

```
public struct HoursWorked
{
 float RegularHours;
 float OvertimeHours;
 // გადატვირთული ოპერატორის ძველი ვერსია (იხ. წინა მაგალითი)
public static HoursWorked operator + (HoursWorked  a,
HoursWorked  b)
  {
    HoursWorked Result = new HoursWorked( );
    Result RegularHours = a.RegularHours + b.RegularHours;
    Result  OvertimeHours = a.OvertimeHours +
b.OvertimeHours;
    return Result;
  }
// გადატვირთული ოპერატორის ახალი რეალიზაციის ვერსია
```

```
public static HoursWorked operator + (HoursWorked  a,  int  b)
 {
   HoursWorked Result = new HoursWorked( );
   Result RegularHours = a. RegularHours + b;
   Result  OvertimeHours = a. OvertimeHours + B.
OvertimeHours;
   return Result;
 }
}
```

ვიზუალ C#–ით გადატვირთული ოპერატორის შესაქმნელად საჭიროა შემდეგი მოქმედებები:

1. გამოცხადდეს გადატვირთული ოპერატორი კლასის ან სტრუქტურის განსაზღვრების შიგნით, რომელთანაც უნდა იმუშაოს მან, გასაღებური სიტყვით operator. გადატვირთული ოპერატორი აუცილებლად უნდა გამოცხადდეს public და static მოდიფიკატორებით. თუ ოპერატორი ბინარულია, მაშინ მის ერთ არგუმენტს მაინც უნდა ჰქონდეს იგივე ტიპი, რაც აქვს დასაბრუნებელ მნიშვნელობას.

2. დაიწეროს ოპერატორის რეალიზაცია.

- **რეზიუმე:**

გადატვირთვის საშუალებით ხერხდება რამდენიმე მეთოდის შექმნა ერთიდაიმევე სახელით, ოღონდ სხვადასხვა რეალიზაციით. გადატვირთული მეთოდები განსხვავდება ერთმენეთისგან სიგნატურებით, მაგრამ შეიძლება ჰქონდეს წვდომის ერთნაირი დონეები და დასაბრუნებელ მნიშვნელობათა ტიპები. ისინი ცხადდება ჩვეულებრივი მეთოდების მსგავსად.

C# პროგრამებში შეიძლება არასტანდარტული ვერსიის ოპერატორების განსაზღვრა, რომელთა დანიშნულებაა მომხმარებელთა ტიპებთან მუშაობა. გადატვირთული ოპერატორები აუცილებლად უნდა გამოცხადდეს public და static მოდიფიკატორებით და გასაღებური სიტყვით operator.

1.2.3. პოლიმორფიზმის რეალიზაცია ინტერფეისებით

ინტერფეისები განსაზღვრავს კლასის ყოფაქცევას. ერთი და იგივე ინტერფეისი შეიძლება რეალიზებულ იქნას სხვადასხვა კლასებში, რაც უზრუნველყოფს მათ ურთიერთმოქმედების შესაძლებლობას, ამგვარად პოლიმორფიზმის კონცეფციის განხორციელებას.

ამ პარაგრაფში განვიხილავთ ინტერფეისების გამოცხადებისა და რეალიზების საკითხებს, აგრეთვე თუ როგორ ურთიერთმოქმედებს ობიექტები ინტერფეისების საშუალებით.

ჩვენს კონტექსტში ინტერფეისი არის გარკვეული შეთანხმება. ნებისმიერი ობიექტი, რომელშიც რეალიზებულია მოცემული ინტერფეისი, გარანტირებულად შეიცავს რეალიზაციას წევრებისათვის, რომლებიც გამოცხადებულია ამ ინტერფეისში. თუ ობიექტს აქვს რომელიმე ინტერფეისი, მაშინ ნებისმიერი სხვა ობიექტი, რომელშიც რეალიზებულია ეს ინტერფეისი, ფლობს შესაძლებლობას იმოქმედოს მასთან.

ინტერფეისი განსაზღვრავს მხოლოდ წევრებს, რომლებიც ობიექტშია რეალიზებული. თვით ინტერფეისის განსაზღვრებაში არაფერია ნათქვამი მისი წევრების რეალიზაციაზე. იგი შეიცავს მხოლოდ პარამეტრებისა და დასაბრუნებელ მნიშვნელობათა ტიპების აღწერას. ინტერფეისში გამოცხადებული წევრების რეალიზაცია მთლიანად და სრულად ეკისრება კლასს, რომელშიც თვით ეს ინტერფეისია რეალიზებული.

ამგვარად, სხვადასხვა ობიექტებში ინტერფეისის ერთიდაიგივე წევრები რეალიზებულია სრულიად განსხვავებულად. განვიხილოთ, მაგალითად, ინტერფეისი IShape, რომელიც განსაზღვრავს ერთადერთ მეთოდს – CalculateArea. ამასთანავე კლასი Circle, რომელშიც რეალიზებულია ეს ინტერფეისი, გაითვლის ფიგურის ფართობს სრულიად განსხვავებულად, ვიდრე კლასი Square, რომელშიც ასევე რეალიზებულია ეს ინტერფეისი. მიუხედავად ამისა, ობიექტს, რომელსაც სჭირდება ურთიერთქმედება IShape–ს გამოყენებით, შეუძლია მეთოდის CalculateArea გამოძახება Circle ან Square კლასიდან და მიიღოს კორექტული შედეგი.

- **ინტერფეისების განსაზღვრა:**

ინტერფეისი განისაზღვრება გასაღებური სიტყვით interface. მაგალითად:

```
public interface IDrivable
{
}
```

აქ გამოცხადებულია **IDrivable** ინტერფეისი წევრების გარეშე.

ინტერფეისის წევრი-მეთოდები განისაზღვრება მეთოდის ჩვეულებრივი სიგნატურით, ოღონდ წვდომის მოდიფიკატორების (public, private, . . .) გარეშე. მოდიფიკატორი, რომელიც ინტერფეისისთვისაა მოცემული, ვრცელდება მის ყველა წევრზე. განვიხილოთ ინტერფეისის წევრი-მეთოდების გამოცხადების მაგალითი.

```
public interface IDrivable
{
  void GoForward(int Speed);
  void Halt();
   int DistanceTraveled( );
}
```

მეთოდების მსგავსად ინტერფეისში განისაზღვრება წევრი-თვისებებიც. ამისათვის გამოიყენება სპეციალური მეთოდები: მიმღები-მეთოდი (getter) და დამყენებელი-მეთოდი (setter). თვისებათა განსაზღვრა მთავრდება მნიშვნელობის ტიპის მითითებით, რომელსაც თვისება აბრუნებს. მაგალითად:

```
public interface IDrivable
{
  // სხვა წევრების განსაზღვრა აქ გამოტოვილია
  int FuelLevel
  {
    get;
```

```
    // ეს თვისება რომ იყოს მისაწვდომი მხოლოდ წასაკითხად,
უნდა გამოცხადდეს აქ მეთოდი set
  }
}
```

თვისებებისგან განსხვავებით, ველები არ შეიძლება იყოს ინტერფეისის წევრები. ესაა გარანტია იმისა, რომ კლასები, რომლებიც ინტერფეისების საშუალებით ურთიერთმოქმედებს, ვერ განახორციელებს მიმართვას ობიექტის შიგა მონაცემებთან.

ინტერფეისები განსაზღვრავს აგრეთვე მოვლენებს, რომლებსაც ის ობიექტები აგენერირებს, რომლებშიც ეს ინტერფეისია რეალიზებული. ნებისმიერი კლასი, რომელიც ინტერფეისს უკეთებს რეალიზაციას, ვალდებულია მისცეს რეალიზაციის საშუალება ამ ინტერფეისის ყველა წევრ-მოვლენას. მაგრამ ობიექტებისთვის, რომლებიც ამ ინტერფეისის საშუალებით ურთიერთმოქმედებს, არასავალდებულოა მასში გამოცხადებული ყველა მოვლენის დამუშავება. C# მოითხოვს მოვლენებისთვის ტიპი-დელეგატების დანიშვნას. მაგალითად:

```
public interface IDrivable
{
  // სხვა წევრების განსაზღვრა აქ გამოტოვილია
  event System.EventHandler OutOfFuel;
}
```

ინტერფეისის განსაზღვრა ხდება გასაღებური სიტყვით interface და ემატება მისი წევრების: მეთოდების, თვისებების და მოვლენების სიგნატურები.

- **ინტერფეისები - პოლიმორფიზმის მიღწევის საშუალება:**

ნებისმიერი ობიექტი, რომელშიც რეალიზაბულია რომელიმე ინტერფეისი, შეუძლია ურთიერთქმედება სხვა ობიექტებთან, რომელთაც სჭირდებათ ეს ინტერფეისი. მაგალითად:

```
public void GoSomewhere(IDrivable v)
{
  // რეალიზაცია გამოტოვილია
}
```

აქ გამოცხადებულ მეთოდისთვის აუცილებელია, რომ მის არგუმენტში იყოს რეალიზებული **IDrivable** ინტერფეისი. შედეგად, ამ მეთოდს შეიძლება გადაეცეს ნებისმიერი ობიექტი, რომელშიც რეალიზებულია ეს ინტერფეისი, ამავდროულად ეს ეს ობიექტი არაცხადად გარდაიქმნება ამ ინტერფეისის ტიპად. ობიექტებისთვის, რომლებიც ურთიერთქმედებს ერთმენეთთან მათში რეალიზებული ინტერფეისის საშუალებით, ხელმისაწვდომია მხოლოდ ამ ინტერფეისის წევრები.

გარდა ამისა, ნებადართულია ობიექტების ცხადი სახით გარდაქმნა ინტერფეისის ტიპად, რომლებშიც რეალიზებულია რომელიმე ინტერფეისი. მაგალითში ნაჩვენებია Truck ობიექტის გარდაქმნა **IDrivable** ინტერფეისის ტიპად. იმისათვის რომ ეს პროცედურა შესრულდეს, საჭიროა Truck ობიექტში რეალიზებულ იყოს **IDrivable** ინტერფეისი.

```
Truck myTruck = new Truck();
Idivable myVehicle;
// Truck ობიექტის გარდაქმნა IDrivable ინტერფეისის ტიპად
myVehicle = (IDivable)myTruck;
```

- **ინტერფეისების რეალიზაცია:**

C#-ის პროგრამაში ინტერფეისის სარეალიზაციოდ გამოიყენება „ : “. მაგალითად,

```
public class Truck : IDrivable
{
  // რეალიზაცია გამოტოვილია
}
```

კლასში შესაძლებელია რამდენიმე ინტერფეისის რეალიზაცია. ასეთი კლასის გამოცხადების დროს საჭიროა ინტერფეისების მითითება მძიმის საშუალებით:

```
public class Truck : IDrivable, IFuelBurning, ICargoCarrying
{
```

```
  // რეალიზაცია გამოტოვილია
}
```

თუ კლასში ან სტრუქტურაში რეალიზებულია რომელიმე ინტერფეისი, აუცილებელია მიეცეს რეალიზაციის შესაძლებლობა მის ყველა წევრს. ეს წესი ვრცელდება მაშინაც, თუ რელიზებულია რამდენიმე ინტერფეისი.

- **ინტერფეისის წევრების რეალიზაცია:**

C#-ის კლასებსა და სტრუქტურებში გამოცხადებული ინტერფეისის წევრების რეალიზაცია ხდება შემდეგნაირად. ამისათვის განსაზღვრავენ წევრს სახელით, რომელიც ემთხვევა ინტერფეისის წევრის სახელს. ამავდროულად, კლასის ეს წევრი ცხადდება წვდომის იმავე დონით, რომლითაც გამოცხადებულია ინტერფეისი. მომდევნო მაგალითი ასახავს მეთოდის რეალიზაციას, როგორც ინტერფეისის წევრისა:

```
public interface IDrivable
{
  void GoForward(int Speed);
}

public class Truck : IDrivable
{
   public  void GoForward(int Speed)
     {
       // რეალიზაცია გამოტოვილია
     }
}
```

ასეთი სახით რეალიზებული ინტერფეისის წევრები მისაწვდომია როგორც ინტერფეისის, ასევე კლასისთვის. შედეგად, ამ წევრების გამოძახება შესაძლებელია ობიექტის გარდაქმნის შემდეგ (რომელშიც ინტერფეისია რეალიზებული) მის საკუთარ ტიპში ან ამ ინტერფეისის ტიპში.

გარდა ამისა, ინტერფეისის რეალიზაცია კლასში შეიძლება ცხადად. ასეთი სახით რეალიზებული წევრები მისაწვდომია მხოლოდ ობიექტის გარდაქმნის შემდეგ ინტერფეისის ტიპში. ინტერფეისის წევრის ცხადი სახით რეალიზაციისთვის საჭიროა კლასში გამოცხადდეს იმავე სახელის წევრი, ინტერფეისის სრული სახელის მითითებით. ქვემოთ ნაჩვენებია ცხადად რეალიზებული GoForward მეთოდი, რომელიც არის IDrivable ინტერფეისის წევრი.

```
public class Truck : IDrivable
{
  void  IDrivable.GoForward(int Speed)
    {
     // რეალიზაცია გამოტოვილია
    }
}
```

როგორც ვხედავთ, გამოცხადებულ წევრს არა აქვს წვდომის მოდიფიკატორი. ვინაიდან ეს წევრი რეალიზებულია ცხადად, მისი წვდომის დონე განისაზღვრება თვით ინტერფეისის მოდიფიკატორით.

ამგვარად, **C#**-ის პროგრამაში ინტერფეისის რეალიზაციისთვის საჭიროა:

1. გამოცხადდეს კლასი საჭირო ინტერფეის(ებ)ით „ : “ -ის გამოყენებით.

2. დაიწეროს რეალიზაცია ინტერფეისის ყველა წევრისათვის:

- იმისათვის, რომ წევრი იყოს მისაწვდომი კლასისა და ინტერფეისისთვის, იგი გამოცხადდეს იგივე სახელით, წვდომის დონით და სიგნატურით, როგორიც აქვს ინტერფეისის შესაბამის წევრებს;

- იმისათვის, რომ წევრი გავხადოთ მისაწვდომი მხოლოდ ინტერფეისიდან, იგი უნდა გამოცხადდეს ინტერფეისის სრული სახელით (მოდიფიკატორი არაა სავალდებულო).

1.2.4. პოლიმორფიზმის რეალიზაცია მემკვიდრეობითობით

მემკვიდრეობითობა საშუალებას იძლევა გამოცხადდეს ახალი კლასი არსებული კლასის საფუძველზე და გადასცეს ახალ კლასს მისი ყველა წევრი და ფუნქციონალობა. ამგვარად შესაძლებელია კლასების შექმნა, რომლებშიც რეალიზებულ იქნება საბაზო შესაძლებლობების ერთობლიობა, ხოლო შემდეგ გამოყენებულ იქნას ისინი წარმოებული კლასების გამოსაცხადებლად, სხვადასხვა, მაგრამ დაკავშირებული ფუნქციების შესასრულებლად.

წინამდებარე პარაგრაფში გავეცნობით მემკვიდრეობითობის გამოყენებას წარმოებული კლასების შესაქმნელად საბაზო კლასების საფუძველზე, ახალი რეალიზაციის შექმნას საბაზო კლასების წევრებისთვის, აგრეთვე აბსტრაქტული საბაზო კლასების გამოცხადებას.

- **მემკვიდრეობითობა:**

მემკვიდრეობითობა გვაძლევს საშუალებას შეიქმნას კლასები, რომლებიც რეალიზებას უკეთებს საერთო (ზოგად) შესაძლებლობათა ერთობლიობას. ამასთანავე, სპეციალიზებული კლასები, რომელთაც წარმოებულ კლასებსაც უწოდებენ, შთამომავლებია ერთი ზოგადი კლასის, რომელიც საბაზო კლასია. ამბობენ, რომ წარმოებული კლასები აფართოებენ საბაზო კლასის შესაძლებლობებს. საბაზო კლასში ინკაფსულირებულია ზოგადი ფუნქციონალობა, რომელიც გააჩნია ყველა მისგან წარმოებულ კლასს. ამასთანავე, წარმოებულ კლასებს აქვს დამატებითი შესაძლებლობები, რომლებიც უნიკალურია თვითოეული წარმოებული კლასისთვის.

მაგალითად, კლასი *ავტომობილი* ზოგადია და იგი ფლობს საბაზო კლასის ყველა ფუნქციონალობას (ობიექტებს, მეთოდებს, მოვლენებს). მის საფუძველზე შეიძლება შევქმნათ წარმოებული კლასი *სატვირთო ავტომობილი*, რომელიც მემკვიდრეობით შეიძენს საბაზოსგან მთლიან ფუნქციონალობას. დამატებითი შესაძლებლობების სახით მასში რეალიზებულ იქნება ისეთი

თვისებები, რომლებიც მხოლოდ მისთვისაა დამახასიათებელი, მაგალითად, *ტვირთის–საჯე, ტვირთამწეობა* და ა.შ.

- **პოლიმორფიზმი და წარმოებული კლასები:**

წარმოებული კლასი შეიძლება გავაიგივოთ მის საბაზო კლასთან. მაგალითად, კლასის *სატვირთო ავტომობილი* ყველა ობიექტი არის ამავდროულად ობიექტი კლასისა *ავტომობილი.*

პოლიმორფიზმი მემკვიდრეობითობის დროს მდგომარეობს იმაში, რომ წარმოებული კლასის ნებისმიერი ეგზემპლარი მზადაა შეასრულოს მისი საბაზო კლასის ფუნქციები. ნებისმიერი წარმოებული კლასი შეიძლება არაცხადად გარდაიქმნას თავისი საბაზო კლასის ობიექტში, ამასთანავე ყველა წევრი, რეალიზებული წარმოებულ კლასში მიუწვდომელი იქნება. მისაწვდომი იქნება მხოლოდ საბაზო კლასის წევრები.

- **წარმოებული კლასების შექმნა:**

წარმოებულ კლასებს აცხადებენ „ : “ სიმბოლოთი. მაგალითად:

```
public class PickupTruck : Truck
{
 // რეალიზაცია გამოტოვებულია
}
```

წარმოებულ კლასს მხოლოდ ერთი წინაპარი ჰყავს (საბაზო კლასი), მაგრამ მასში დამატებით დასაშვებია ერთი ან რამდენიმე ინტერფეისის რეალიზება. თუ ვაცხადებთ წარმოებულ კლასს, რომელშიც რამდენიმე ინტერფეისია რეალიზებული, მაშინ ისინი უნდა ჩამოითვალოს მიმდევრობით („ , “-ებით) „ : საბაზო–კლასის–სახელი“-ის შემდეგ. მაგალითად:

```
public class FourwheeIDrivePickupTruck : PickupTruck,
IFourwheeIDrive
{
 // რეალიზაცია გამოტოვებულია
}
```

წარმოებული კლასის გამოცხადების შემდეგ შეიძლება დაემატოს მას ახალი წევრები, რომლებიც არასტანდარტულ შესაძლებლობებს არეალიზებს.

- **ჩაბეჭდილი კლასების შექმნა:**

ზოგჯერ საჭიროა ისეთი კლასები, რომლებიც ვერ შესასრულებს საბაზო კლასის როლს. მაგალითად, სპეციალიზებული კლასები, რომლებიც საჭიროა უშუალოდ მომხმარებლის აპლიკაციის კომპონენტების სახით გამოსაყენებლად. ისინი გამოუსადეგარი იქნება სხვა პროგრამისტებისთვის, ვინაიდან მათ არ შეუძლია გადასცეს თავის შთამომავლებს რაიმე სასარგებლო ფუნქციონალობა. ასეთი კლასები ცხადდება გასაღებური სიტყვით sealed. მაგალითად:

```
public sealed class AClass
{
 // რეალიზაცია გამოტოვებულია
}
```

- **მემკვიდრეობით გადაცემული წევრები:**

როგორც აღნიშნული იყო, წარმოებული კლასი ფლობს მისი საბაზო კლასის მთელ ფუნქციონალობას. წარმოებულ კლასში შეიძლება არა მხოლოდ ახალი წევრების ჩამატება, არამედ საბაზოდან მემკვიდრეობით მიღებული წევრების რეალიზაციის შეცვლა დასმული ამოცანების გადასაწყვეტად.

საბაზო კლასიდან მემკვიდრეობით მიღებული წევრების განსაზღვრების შეცვლით (override), ხერხდება მათი რეალიზაციის შეცვლა.

C# - ში შესაძლებელია წევრების დამალვა (hide), რომლებიც მემკვიდრეობითაა მიღებული საბაზო კლასიდან. სამაგიეროდ რეალიზდება ახალი წევრი იმავე სახელით და სიგნატურით, ოღონდაც სხვა მახასიათებლებით განსხვავებული.

განსაზღვრების შეცვლა შეიძლება მხოლოდ თვისებებისა და მეთოდებისათვის, რომლებიც მიღებულია საბაზო კლასიდან. ცვლადებისა და მოვლენებისათვის განსაზღვრების შეცვლა დაუშვებელია.

წევრის ახალ რეალიზაციაში სიგნატურა, დასაბრუნებელი მნიშვნელობის ტიპი და წვდომის დონე უნდა ემთხვეოდეს განსაზღვრებაშეცვლილ წევრის იმავე მნიშვნელობებს. მაგალითად:

```
// საბაზო კლასს აქვს GoForward მეთოდი
public class SportsCar : Car
{
  public override void GoForward(int Speed)
   {
     // რეალიზაცია გამოტოვებულია
   }
}
```

განსაზღვრებაშეცვლილი წევრის გამოძახებისას იმ წევრის მაგივრად, რომელიც საბაზო კლასიდანაა მემკვიდრეობით მიღებული, გამოიძახება ამ წევრის ახალი რეალიზაცია, გამოძახების კონტექსტისგან დამოუკიდებლად. მაგალითად, თუ წარმოებული კლასის ეგზემპლარის გარდაქმნის შემდეგ მისი საბაზო კლასის ტიპში გამოძახებულ იქნება განსაზღვრბაშეცვლილი მეთოდი, მაინც შესრულდება ამ მეთოდის ახალი რეალიზაცია. გამოძახებული წევრი განისაზღვრება ობიექტით და არა ცვლადის ტიპით.

საბაზო კლასის წევრის განსაზღვრების შესაცვლელად იგი უნდა მოინიშნოს გასაღებური სიტყვით virtual, წინააღმდეგ შემთხვევაში წევრები ითვლება ფიქსირებულად და მათი განსაზღვრების შეცვლა არ შეიძლება. მაგალითად:

```
public virtual void OverrideMe()
{
  // რეალიზაცია გამოტოვებულია
}
```

საბაზო კლასის წევრების დამალვა C#-ში.

როგორც აღვნიშნეთ, C#-ის პროგრამაში შესაძლებელია საბაზო კლასიდან მემკვიდრეობით მიღებული წევრის შეცვლა სხვა წევრით, რომელსაც სრულიად განსხვავებული რეალიზაცია ექნება. ამ ხერხს უწოდებენ „დამალვას“ (hiding).

ახალი წევრის ტიპი და სიგნატურა უნდა იყოს იგივე, როგორიც ჩასანაცვლებელ დამალულ წევრს აქვს, მაგრამ წვდომის დონე, დასაბრუნებელ მნიშვნელობის ტიპი და რეალიზაცია შეიძლება განსხვავდებოდეს. თუ ახალი მეთოდის სახელი ემთხვევა არსებულიდან ერთ-ერთს, მაგრამ მათი სიგნატურები განსხვავდება, მაშინ ახალი მეთოდი განიხილება როგორც არსებული მეთოდის გადატვირთული ვერსია, ამასთანავე ეს უკანასკნელი რჩება მისაწვდომი. იმისათვის, რომ დავმალოთ საბაზო კლასიდან მემკვიდრეობითმიღებული წევრი, აუცილებელია გასაღებური სიტყვის new გამოყენება. მაგალითად:

```
// საბაზო კლასი
public class MyBaseClass
{
  public string MyMethod(int I)
    {
      // რეალიზაცია გამოტოვებულია
    }
}
// წარმოებული კლასი
public class MyInheritedClass : MyBaseClass
{
  // ეს ფუნქცია ჩაანაცვლებს MyMethod მეთოდს საბაზო კლასიდან.
  // მას იგივე სიგნატურა, მაგრამ სხვა წვდომის დონე და დასაბრუნებელი მნიშვნელობის ტიპი აქვს
  internal new int MyMethod(int  I)
    {
      // რეალიზაცია გამოტოვებულია
    }
}
```

- **თავსებადობის განზორციელება დამალულ წევრებზე:**

საბაზო კლასის დამალული წევრის რეალიზაცია ხდება მიუწვდომელი, მას ცვლის ახალი რეალიზაცია, შესაძლოა სულ სხვა მახასიათებლებით, რაც ზოგჯერ მნიშვნელოვან გავლენას ახდენს სხვა ობიექტებთან ურთიერთქმედების დროს.

ობიექტი, რომელიც იძახებს ფუნქციას MyMethod (იხ. წინა მაგალითი), ელოდება მნიშვნელობას string ტიპით, მაგრამ თუ ფუნქცია დააბრუნებს int–ს, მაშინ მოსალოდნელია შეცდომა. ამიტომაც, ძალზე საფრთხილოა დამალული წევრების გამოყენება და მხოლოდ მაშინ უნდა მივმართოთ მას, როცა უეჭველი გარანტიაა, რომ თავსებადობა არ დაირღვევა.

რიგ შემთხვევებში საბაზო კლასის დამალულ წევრების რეალიზაციაზე მიმართვა მაინც ხერხდება. საიდან იქნება წევრი გამოძახებული – საბაზოდან თუ წარმოებულიდან, დამოკიდებულია ცვლადის ტიპზე და არა ობიექტზე, როგორც ამას ადგილი ჰქონდა განსაზღვრებაშეცვლილი წევრის შემთხვევაში. მაგალითად:

```
// გამოიყენება კლასები MyBaseClass და MyInheritedClass (იხ. წინა კოდი)
MyInheritedClass X = new MyInheritedClass( );
MyBaseClass Y;
// ცვლადები X და Y მიუთითებს ახლა ერთიდაიმავე ობიექტს, მაგრამ მათი ტიპები განსხვავებულია.
Y = X;
// X -ის ცვლადის ტიპია MyInheritedClass, ამიტომ მოიმდეგნო სტიქონით
// გამოიძახება რეალიზაცია წარმოებული კლასიდან.
Y.MyMethod(42);
// Y -ის ცვლადის ტიპია MyBaseClass, ამიტომ მოიმდეგნო სტიქონით
// გამოიძახება რეალიზაცია საბაზი კლასიდან.
Y.MyMethod(42);
```

როგორც ვხედავთ, უშუალოდ ცვლადის ტიპი განსაზღვრავს, თუ რომელი წევრი იქნება გამოძახებული, დამალული –

წარმოებულიდან, თუ საწყისი – საბაზოდან. ეს უზრუნველყოფს წევრის რეალიზაციის მოდიფიცირებას პოლიმორფიზმის პრინციპის დაურღვევლად.

წარმოებული კლასების გამოცხადებისას დამალული წევრების ახალი რეალიზაცია არ გადაიცემა მემკვიდრეობით, წარმოებულ კლასს გადაეცემა საბაზო კლასის შესაბამისი წევრების რეალიზაცია.

- **საბაზო კლასის წევრებზე წვდომის მიღება:**

განსაზღვრებაშეცვლილი ან დამალული წევრების გამოცხადებისას ზოგჯერ საჭიროა წვდომა საბაზო კლასის ამ წევრის რეალიზაციაზე. ამისათვის იყენებენ გასაღებურ სიტყვას base. მაგალითად:

```
// აქ დემონსტრირებულია საბაზო კლასის რეალიზაციის გამოძახება
// განსაზღვრებაშეცვლილი მეთოდიდან MyMethod( ).
public override void MyMethod( )
  {
    base.MyMethod( );
    // რეალიზაცია გამოტოვებულია
  }
```

- **დაცული წევრები:**

კლასების წევრებზე წვდომა, როგორც ადრე აღვნიშნეთ, ხორციელდება წვდომის დონეებით: public – შესაძლებელია მიმართვა აპლიკაციის ნებისმიერი ადგილიდან, აგრეთვე გარე კლასებიდანაც; internal – წვდომა ხორციელდება მხოლოდ წევრებზე ლოკალური ნაკრებიდან; private – წვდომა შესაძლებელია მხოლოდ კლასის შიგნით. შესძლებელია კიდევ წვდომის ორი დონე: protected და protected internal – რომლებიც არ განხილულა.

კლასების წევრთა ხილვადობის არეები, რომლებიც გამოცხადებულია გასაღებური სიტყვებით protected და private –

მსგავსია, ერთი განსხვავებით, რომ პირველ შემთხვევაში წევრები მისაწვდომია წარმოებული კლასებისთვისაც. მაგალითად:

```
// საბაზო კლასში განსაზღვრულია ორი მეთოდი:
// წვდომის დონეებით protected და private
public class BaseClass
   {
     // მეთოდი private-წვდომის დონით არ შეიძლება გამოვიძახოთ წარმოებული კლასებიდან
      private void  Method1( )
        {
          // რეალიზაცია გამოტოვებულია
        }

     // მეთოდი protected-წვდომის დონით შეიძლება გამოვიძახოთ წარმოებული კლასებიდან
       protected void  Method2( )
        {
          // რეალიზაცია გამოტოვებულია
        }
   }
    // კლასი, წარმოებული BaseClass -იდან
   public class InheritedClass : BaseClass
   {
    public void  Demo( )
     {
       // ასეთი გამოძახება დასაშვებია, რადგან protected არ კრძალავს BaseClass–ის წევრებზე მიმართვას.
        this.Method2( );
        // ასეთი გამოძახება იწვევს კომპილაციის შეცდომას, ვინაიდან გამოყენებულია private წვდომის დონე.
         this.Method1( );
     }
   }
```

წვდომის დონე protected internal არის მოდიფიკატორების protected და internal ჰიბრიდი. ამიტომაც კლასის წევრი, რომელიც გამოცხადებულია protected internal –ით, მისაწვდომია ამ კლასის

და სხვა კლასის წევრებისთვისაც, რომლებიც ამავე ნაკრებშია ამ კლასთან, აგრეთვე მისი წარმოებული კლასების წევრებისთვისაც.

- **აბსტრაქტული კლასები და წევრები:**

კომპონენტების შექმნის დროს ზოგჯერ საჭიროა კლასები, რომლებიც გვთავაზობს განსაზღვრულ შესაძლებლობათა ერთობლიობას, უცვლელი სახით გამოსაყენებლად. ხოლო ამ კლასის სხვა წევრების რეალიზაციაზე პასუხს აგებენ მათი წარმოებული კლასები. ასეთ შესაძლებლობას ფლობს აბსტრაქტული (abstract) კლასები, რომელთაც შეუძლია მხოლოდ საბაზო კლასების როლის შესრულება.

აბსტრაქტული კლასები ჰგავს ინტერფეისებს, მაგრამ მათ ბევრი საერთო აქვს ჩვეულებრივ კლასებთან.

აბსტრაქტული კლასის ეგზემპლარის შექმნა არ შეიძლება, იგი გამოსადეგია მხოლოდ წარმოებული კლასების გამოცხადებისთვის. აბსტრაქტული კლასი იძლევა კლასის სრულ რეალიზაციას, ან მის ნაწილს ან საერთოდ არავითარ რეალიზაციას.

ინტერფეისებიც და აბსტრაქტული კლასებიც შეიცავს წევრების აღწერას, რომლებიც რეალიზებულ უნდა იქნას წარმოებულ კლასებში, ოღონდ ინტერფეისისგან განსხვავებით, მხოლოდ ერთ აბსტრაქტულ კლასს შეუძლია იყოს წარმოებული კლასის წინაპარი. აბსტრაქტული კლასები იძლევა მხოლოდ სრულად რეალიზებულ წევრებს, აგრეთვე იმ წევრებს, რომელთა რეალიზაციაზე პასუხისმგებელია წარმოებული კლასები.

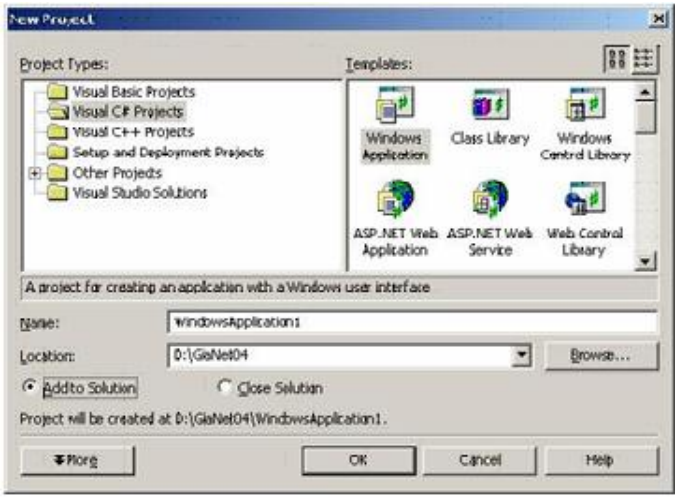
II თავი

C#.NET ენა

2.1. C# ენის სამუშაო გარემო კონსოლის რეჟიმში

.NET Frameworks SDK პლატფორმაზე C# (‘სი შარფ‘) ენის პროგრამების შესაქმნელად, კომპილაციისა და ტესტირებისათვის ძირითადად გამოიყენება ორი რეჟიმი: კონსოლის (ნახ.2.1), რომელიც MsDos და Unix ოპერაციული სისტემების გარემოს მოგვაგონებს და კომპონენტურ-ვიზუალური (ნახ.2.2), რომელსაც Windows, Linux და სხვ.

ნახ.2.1. Console mode



ნახ.2.2. Windows Application

თანამედროვე სისტემების მსგავსად გრაფიკული ინტერფეისები გააჩნია. ორივე რეჟიმი თავისთავად საინტერესო და მნიშვნელოვანია. პირველში შესაძლებელია C# პროგრამული მოდულების შექმნა და ტესტირება, ანალიზი და შეცდომების დიაგნოსტიკა, სტანდარტული მოდულების ბიბლიოთეკის შექმნა მისი შემდგომი მრავალჯერადი გამოყენებისათვის.

მეორე რეჟიმში იქმნება დიდი პროექტები (Applications), რომელთა სწრაფად დამუშავებას და ტესტირებას ძალზე მდიდარი ვიზუალურ კომპონენტთა ბიბლიოთეკის არსებობა განაპირობებს.

ჩვენ მოკლედ განვიხილავთ ორივე რეჟიმს და წარმოვადგენთ იმ ძირითად საკითხებს, რომლებიც აუცილებელია C#.NET ენაზე პროგრამული მოდულებისა და პროექტების ასაგებად.

პირველ რიგში საჭიროა კარგად გავერკვეთ C#-კოდის შედგენილობასა და სტრუქტურაში, ცვლადებისა და კონსტანტების ტიპებსა და მათი მნიშვნელობების დიაპაზონებში, ოპერაციებსა და ფუნქციებში, ობიექტ-ორიენტირებული და კომპონენტურ-ვიზუალური მეთოდების კონცეფციებსა და მათ რეალიზაციაში.

სწორედ ამ საკითხებს განვიხილავთ წინამდებარე თავის პარაგრაფებში ილუსტრაციებითა და პროგრამული ლისტინგებით.

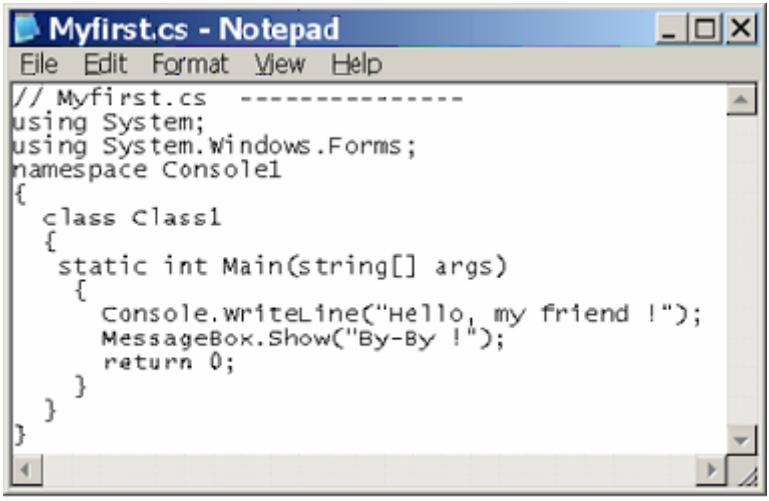
2.2. C# -კოდის აგება და ტესტირება კონსოლის რეჟიმში

C# - პროგრამის საწყისი ტექსტი უნდა მომზადდეს რომელიმე ტექსტურ რედაქტორში, მაგ., NotePad-ში (ნახ.2.3).

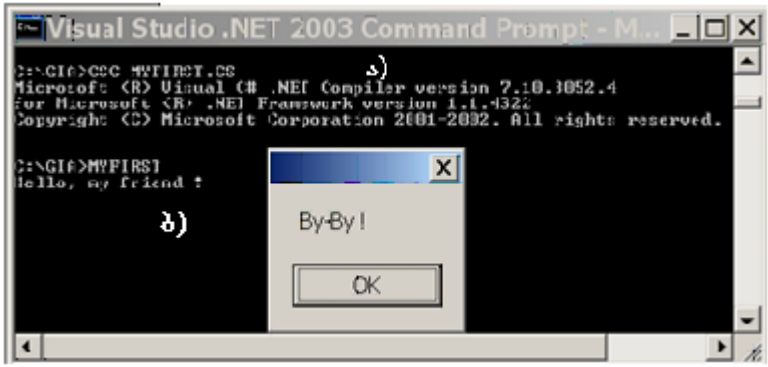
პროგრამის ტექსტს გაფართოვება (ფაილის ტიპი) დაუყენდება **sc**.

C# - ის კომპილატორია **csc,** რომელიც კონსოლის რეჟიმში გამოიძახება და არგუმენტად მიეთითება ტექსტური ფაილის სახელი, ანუ ჩვენს შემთხვევაში **Myfirst.cs** (ნახ.2.4-ა).

კომპილაციის შემდეგ, თუ კოდში არაა სინტაკსური შეცდომები, მიიღება გამომავალი – კომპილირებული კოდი, რომელსაც შესრულებად ფაილს (**exe**) უწოდებენ.



ნახ.2.2.3



ნახ.2.2.4. ა-კომპილაციის და ბ - ტესტირების ეტაპები

ამ მუშა ფაილის ამუშავებით (კონსოლის რეჟიმში გამოიძახება იგი სახელით, მაგ., Myfirst), მიიღება საბოლოო შედეგები (ნახ.2.2.4-ბ). OK–ღილაკზე დაწკაპუნებით დასრულდება პროგრამის შესრულება.

თუ პროგრამაში სინტაკსური შეცდომებია, მაშინ საჭიროა დაბრუნება ისევ ტექსტურ რედაქტორში, ამ შეცდომების პოვნა და

ჩასწორება. შემდეგ ტესტირების პროცესი განმეორებით ჩატარდება, სასურველი შედეგების მიღებამდე.

სტრიქონი MessageBox.Show(«By-By !»); ემსახურება "ინფორმაცია"- შეტყობინების გამოტანას Windows–სტილში.

2.3. C# ენის გასაღებური სიტყვები

1-ელ ცხრილში მოცემულია C# ენის ძირითადი გასაღებური სიტყვების ნუსხა, რომელთა გამოყენება აქტიურად ხდება პროგრამულ მოდულებში.

როგორც ვხედავთ, მრავალი მათგანი მსგავსია C, C++ და Java ენათა ოპერატორებისა. ახალია, მაგალითად, namespace, რომელსაც განსაკუთრებული დანიშნულება აქვს და მას ცალკე პარაგრაფი მიეძღვნება. რეკომენდებულია, რომ არ გამოიყენოთ უპირობო გადასვლის goto-ოპერატორი, იგი არღვევს სტრუქტურული დაპროგრამების პრინციპებს. მის სანაცვლოდ პროგრამაში სასურველია if..elese…if - განშტოებათა, for, while, do…while - ციკლებისა და switch - გადამრთველის გამოყენება.

ენის 'დაჯავშნულ" სიტყვებში განსაკუთრებული ადგილი უჭირავს მონაცემთა ტიპების ოპერატორებს, რომელთაც მომდევნო პარაგრაფში განვიხილავთ.

C# Keywords

ცხრ.1

abstract	event	new	struct
as	explicit	null	switch
base	extern	object	this
bool	false	operator	throw
break	finally	out	true
byte	fixed	override	try
case	float	params	typeof
catch	for	private	uint
char	foreach	protected	ulong
checked	goto (?)	public	unchecked
class	if	readonly	unsafe
const	implicit	ref	ushort
continue	in	return	using
decimal	int	sbyte	virtual
default	interface	sealed	volatile
delegate	internal	short	void
do	is	sizeof	while
double	lock	stackalloc	
else	long	static	
enum	namespace (!)	string	

2.4. C# ენის მონაცემთა ძირითადი ტიპები

მონაცემთა ტიპების აღწერით იწყება ყველა პროგრამა, თუკი მასში გამოიყენება სიმბოლურ-სტრიქონული, მთელრიცხვა ან ნამდვილრიცხვა მონაცემები. მე-2 ცხრილში მოტანილია C# ენაში მონაცემთა გამოყენებული ტიპების ნუსხა.

C# ენის ცვლადებისა და კონსტანტების ტიპები ცხრ.2

ცვლადის ტიპში	მოთავსება bit	დიაპაზონი
bool	1	true, false
byte	8	0 .. 255
char	16	0 .. 65535
decimal	128	1.0E-28 .. 7.9E28
double	64	5.0E-324 .. 1.9E308
enum	ჩამოთვლითი ტიპი	იღებს: byte, int, short,long ტიპების მნიშვნელობებს
float	32	1.5E-45 .. 3.4E38
int	32	-2 147 483 648 .. 2 147 483 648
long	64	- 9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807
sbyte	8	-128 .. 127
short	16	-32 768 .. 32 767
struct	ტიპების კონტეინერი	შეიძლება ერთდროულად შეიცავდეს სხვადასხვა ტიპებს
uint	32	0 .. 4 294 967 295
ulong	64	0.. 18 446 744 073 709 551 615
ushort	32	0 .. 65 535

საჭიროა მცირე კომენტარის გაკეთება.

- ჩამოთვლითი ტიპი (enum):

მაგ., enum friends {nino, gio, dito, mito, sico}; ამ ელემენტთა სიმრავლეში პირველი იქნება 0, მეორე 1 და ა.შ.

- decimal ტიპი (double-სგან განსხვავებით) გამოიყენება მეტი სიზუსტისათვის ფულად-საფინანსო ანგარიშებში და რიცხვს მიეწერება სუფიქსი M ან m. მაგ., decimal d = 37.15m;

C#-ენაში შემოტანილია ტიპი **string** , რომელიც Unicode-სიმბოლოთა სტრიქონია.

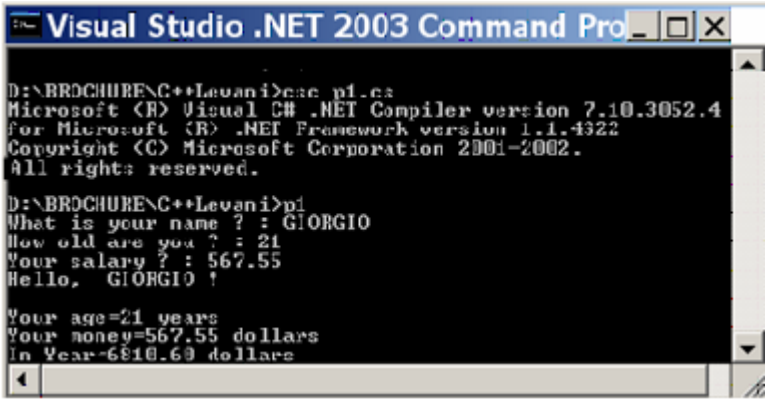
მაგალითად, string a="Hello", b="My friend"; Console.WriteLine(a+b);
//შედეგი: "Hello My friend".

განვიხილოთ ინტერაქტიული პროგრამის მაგალითი, რომელშიც ხდება მონაცემების შეტანა კონსოლის ბრძანების სტრიქონიდან და გამოტანა ეკრანზე (ნახ.2.5):

```
/*P1.cpp    Input-Output  */
using System;
class WhoIsWho
{
  static void Main()
  {
   string Name;
   int Age=0; decimal Money=0.0m;
   Console.Write("\aWhat is your name ? : ");
        Name = Console.ReadLine();
   Console.Write("\aHow old are you ? : ");
        Age = Convert.ToInt16(Console.ReadLine());
   Console.Write("\aYour salary ? : ");
     Money = Convert.ToDecimal(Console.ReadLine());
 // Output results ————————-
  Console.WriteLine("Hello,  {0} !\n", Name);
```

```
  Console.Write("Your age={0} years \n", Age);
  Console.Write("Your money={0} dollars \n",Money);
  Console.Write("In Year={0} dollars\n", Money*12);
 } // პროგრამაში გამოყენებულია ტიპების გარდაქმნის მეთოდები:
} //Convert.ToInt16(),Convert.ToDecimal().
```

კონსოლზე გამოიტანილია პროგრამის კომპილაციისა და შედეგე-ბის მაგალითი.



ნახ.2.5

2.5. C# ენის ტიპების გარდაქმნის ძირითადი ფუნქციები (მეთოდები)

არსებობს მონაცემთა ტიპების გარდაქმნის არაცხადი (implicit) და ცხადი (explicit) საშუალებანი. პირველის შემთხვევაში მონაცემთა გარდაქმნა ხდება პროგრამისტის ჩარევის გარეშე. ცხრილში ნაჩვენებია ეს შესაძლებლობანი.

ტიპების არაცხადი გარდაქმნის ცხრილი

საწყისი ტიპი	გარდაქმნის ტიპი
sbyte	short, int, long, float, double, ან decimal
byte	short, ushort, int, uint, long, ulong, float, double, ან decimal
short	int, long, float, double, ან decimal
ushort	int, uint, long, ulong, float, double, ან decimal
int	long, float, double, ან decimal
uint	long, ulong, float, double, ან decimal
long	float, double, ან decimal
char	ushort, int, uint, long, ulong, float, double, ან decimal
float	double
ulong	float, double, ან decimal

ტიპების ცხადი გარდაქმნის ცხრილი

საწყისი ტიპი	გარდაქმნის ტიპი
sbyte	byte, ushort, uint, ulong, ან char
byte	sbyte or char
short	sbyte, byte, ushort, uint, ulong, ან char
ushort	sbyte, byte, short, ან char
int	sbyte, byte, short, ushort, uint, ulong, ან char
uint	sbyte, byte, short, ushort, int, ან char
long	sbyte, byte, short, ushort, int, uint, ulong, ან char
ulong	sbyte, byte, short, ushort, int, uint, long, ან char
char	sbyte, byte, ან short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char, ან decimal
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, ან decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, ან double

C#-ენაში განსაკუთრებით საყურადღებოა სტრიქონული ცვლადების ტიპის string გამოყენება. როგორც წინა პარაგრაფის პროგრამულ მაგალითში განვიხილეთ, მონაცემების შეტანისას Console.ReadLine( ) მეთოდით, იგი მთელ და ფულად რიცხვებს ღებულობს სიმბოლოების სტრიქონის სახით. მონაცემთა ტიპის ცხადი გარდაქმნის მეთოდით, მაგალითად

Convert.ToDecimal(Console.ReadLine());

მიიღება რიცხვითი ტიპები და შესაძლებელი ხდება მათი მათემატიკურ ოპერაციებში გამოყენება. ასეთივეა

Convert.ToInt16(), Convert.ToFloat() და ა.შ. მეთოდები.

2.6. C# ენის მასივები (კოლექციები)

მასივები შეესაბამება კლასთა კოლექციებს, რომლებიც ჩადგმულია C#–ენაში და კონსტრუქციულად ემსახურება ერთგვაროვან (ერთი ტიპის) მონაცემთა მოხერხებული ორგანიზების საკითხებს. მასივები შეიძლება იყოს ერთ-, ორ- ან მრავალ-განზომილებიანი, ან თავისუფალი (jagged).

ერთგანზომილებიანი მასივის (ვექტორის) გამოცხადების ზოგადი ფორმატი ასეთია:

Type[ ] Array_Name [memory];

სადაც ტიპის შემდეგ მიეთიება მასივის სახელი და ამ მასივისთვის საჭირო მეხსიერება. ეს უკანასკნელი ყოველთვის არაა საჭირო. მაგალითად, ქვემოთ ნაჩვენებია სამსტრიქონიანი მასივის გამოცხადება მნიშვნელობების მინიჭებით:

string[ ] Vector = new string[3] {„black“, „red“, „green“};

n-განზომილებიანი მასივებისათვის გამოიყენება კვადრატულ ფრჩხილებში თითოეული განზომილების მძიმით გამოყოფა. მაგალითად:

int[,] Matrica =new int [5,5];

თავისუფალი მასივები მრავალგანზომილებიანია, რომელთაგან თითოეულში ელემენტთა განზომილება განსხვავებულია. ასეთი საშუალება მოსახერხებელია მეხსიერების ეკონომიურად გამოსაყენებლად. მაგალითად:

```
int[ ][ ] myJaggedArray = new int [ ][ ]
                { new int[ ] {1,3,5,7,9},
                  new int[ ] {0,2,4,6},
                  new int[ ] {11,22}    };
```

2.7. C# ენის ოპერაციები

ენაში გამოიყენება სხვადასხვა ტიპის ოპერაციები, კერძოდ არითმეტიკული, ლოგიკური, სტრიქონების გადაბმის, ინკრემენტ-დეკრემენტის, წაძვრის, რელაციური, ობიექტების შექმნის და სხვ. ქვემოთ ცხრილში მოცემულია ეს ოპერაციები, ხოლო მომდევნო პარაგრაფებში მოკლედ განვიხილავთ მათ დანიშნულებას.

C# Operators

Operator category	Operators
Arithmetic	+ - * / %
Logical (boolean and bitwise)	& | ^ ! ~ && || true false
String concatenation	+
Increment, decrement	++ --
Shift	<< >>
Relational	== != < > <= >=
Assignment	= += -= *= /= %= &= |= ^= <<= >>=
Member access	.
Indexing	[ ]
Cast	( )
Conditional	?:
Delegate concatenation and removal	+ -
Object creation	new
Type information	as is sizeof typeof
Overflow exception control	checked unchecked
Indirection and Address	* -> [ ] &

2.8. C# ენაში ბრძანებათა ნაკადების მართვა

ენებში განსაკუთრებული დანიშნულება აქვს სტრუქტურული დაპროგრამების ელემენტებს, რომლებიც კოდის ბრძანებების (ოპერატორების, პროცედურებისა და მეთოდების შესრულების) მიმდევრობას განსაზღვრავს. ასეთი ოპერატორებს მიეკუთვნება:

1. განშტოების (პირობითი გადასვლის ოპერატორები):

if, if..else, if..else..if

2. გადამრთველი: switch( ).

3. ციკლების :

while( ), do..while( ), for( ), foreach( ).

ბოლო ოპერატორი არ ყოფილა C, C++ და Java ენებში, ამიტომ მას აქ განვიხილავთ დეტალურად.

სინტაკსი შემდეგია:

```
foreach (type identifier in expression)
  {
  // ოპერატორები;
  }
```

სადაც expression არის კოლექციის (მასივის) დასახელება, რომლის შიგნითაც ხორციელდება ციკლური მოძრაობა. მაგალითად:

```
foreach (string Name in ListOfNames)
   {
     Console.WriteLine("\t{0}", Name);
   }
```

4. უპირობო გადასვლის ოპერატრი (goto), რომლის გამოყენებაც არაა რეკომენდებული სტრუქტურული დაპროგრამების თვალსაზრისით;

5. ციკლებიდან ან გადამრთველის case-ბლოკებიდან გამოსვლის ოპერატორი (break);

6. ციკლებში ლოგიკური პირობის შესამოწმებლად უშუალო გადასვლა (continue) სხვა არასაჭირო ოპერატორების გამოტოვებით;

7. მთავარი პროგრამიდან ან მეთოდებიდან გამოსვლის ოპერატორი (return). იგი ხშირად გამოიყენება კოდში. მისი ერთი შემთხვევა ნაჩვენებია ქვემოთ:

```
public static int Main(string[ ] args)
{
// პროგრამის ოპერატორები;
return 0;
}
```

Main-პროგრამა აბრუნებს მთელ (int) მნიშვნელობას, ამიტომაც მისი ნორმალურად დამთავრების შემთხვევაში კონსოლს return–ით გადაეცემა 0.

2.9. C# კოდის მაგალითი

განიხილება მარტივი C#-პროგრამის ტექსტი, რომელიც შესასვლელზე თხოულობს ორი მთელი რიცხვის (x და y) შეტანას და შემდეგ ანგარიშობს მათ ჯამს, ნამრავლს, განსაზღვრავს მაქსიმალურ და მინიმალურს მათ შორის.

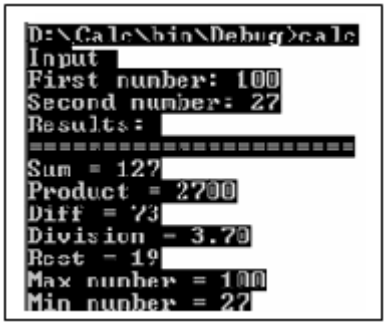
პირველი ორი ოპერაცია რეალიზებულია ჩვენს პფოგრამაში Sum და Product ქვეპროგრამების სახით. მაქსიმალურ და მინიმალურ მნიშვნელობებს კი იგი ღებულობას პროგრამულ ბიბლიოთეკაში არსებული სტანდარტული ფუნქციებით (მეთოდებით).

```
using System;
namespace Calcul
{
 public class SimpleCalculator
  {
   public static void Main()
    {
      int x, y;
      Console.Write("Input \n");
```

```
      Console.Write("First number: ");
      x=Convert.ToInt32(Console.ReadLine() );
      Console.Write("Second number: ");
      y=Convert.ToInt32(Console.ReadLine() );
      Console.Write("Result's: \n");
      Console.WriteLine("Sum = " + Sum(x,y));
      Console.WriteLine("Product = " +
                                    Product(x,y));
      Console.WriteLine(«Diff = « + Dif(x,y));
      Console.WriteLine(«Division = « +
                    Divis(x,y).ToString(«F2»));
      Console.WriteLine(«Rest = « + Module(x,y));
      Console.WriteLine(«Max number = « +
                                    Math.Max(x,y));
      Console.WriteLine(«Min number = « +
                                    Math.Min(x,y));
    }
public static int Sum(int a, int b)
 {
  int sumTotal;
  sumTotal=a+b;
  return sumTotal;
 }
public static int Product(int a, int b)
 {
  int productTotal;
  productTotal=a * b;
  return productTotal;
 }
public static int Dif(int a, int b)
 {
  int difTotal;
  difTotal=a-b;
  return difTotal;
 }
public static double Divis(int a, int b)
 {
  double diviseTotal;
  diviseTotal=a / Convert.ToDouble(b);
  return diviseTotal;
 }
public static int Module(int a, int b)
```

```
  {
   int moduleRest;
   moduleRest=a % b;
   return moduleRest;
  }
 }
}
```

შედეგები მოცემულია 2.6 ნახაზზე.



ნახ.2.6

2.10. C# ენაში ობიექტური და კომპონენტური დაპროგრამების კონცეფცია

C#-ენა, როგორც სხვა თანამევდროვე პროგრამული პაკეტები, არის ობიექტ-ორიენტირებული და კომპონენტური სტილის მქონე ინსტრუმენტი. განვიხილოთ ეს ორი კონცეფცია, რაც შემდგომში უფრო გასაგებს გახდის კომპონენტურ-ვიზუალური დაპროგრამების მეთოდის გამოყენების ეფექტურობას.

არა-ობიექტორიენტირებული დაპროგრამების სტილი ცნობილია **პროცედურულ-ორიენტირებული** დაპროგრამების სახით (მაგალითად, Basic, Pascal, C და სხვ.), ფართოდ გამოიყენებოდა ავტომატიზებული სისტემების აგების ადრინდელ ეტაპზე (90-იან წლებამდე). მათთვის დამახასიათებელი იყო პროცედურების წერა ოპერატორების დონეზე, ხოლო მონაცემები ინახებოდა მათგან

დამოუკიდებლად. დაპროგრამების ასეთი სტილი გამოსადეგია მცირე ზომის პროექტების ასაგებად.

C-ენა, როგორც Unix-ქსელური ოპერაციული სისტემის „დედა ენა“, კარგი სტრუქტურული და სისტემური (ასემბლერული) თვისებების მქონე ინსტრუმენტი, გაფართოვდა ახალი, ობიექტ-ორიენტირებული თვისებებით (ინკაფსულაცია, მემკვიდრეობითობა, პოლიმორფიზმი, აბსტრაქცია) და იქცა თანამედროვე პროგრამისტ-პროფესიონალების C++ მძლავრ ინსტრუმენტად. ამ ენის ძირითადი ელემენტებია კლასები, ობიექტები და მეთოდები, რომელთა საფუძველზეც აღიწერება სისტემის მდგომარეობა (სტატიკა) და მისი ქცევა (დინამიკა).

კომპონენტური დაპროგრამება ყურადღებას ამახვილებს ვიზუალურ-მოდულურ ელემენტებზე, როგორებიცაა მაგალითად, თვისებები (Properties), ინტერფეისები (Interfaces) და მოვლენები (Events). ინტერფეისების დანიშნულებაა კლასებს შორის მეთოდების ინიციალიზება და გადაცემა. თვისებები განსაზღვრავს კლასის კომპონენტებისათვის ამა თუ იმ ინტერფეისთან მიმართვის უფლებას. მოვლენები გამოიყენება გარკვეული სიტუაციების აღსაწერად, რომელთა შედეგად უნდა ამუშავდეს შესაბამისი მეთოდები. ახლა განვიხილოთ ეს საკითხები უფრო დეტალურად.

2.10.1. C# ენის ობიექტები და კლასები

C#-ენაში ობიექტების წარმოდგენა ხდება ისეთი ტიპით, როგორიცაა **კლასი**. კლასის ატრიბუტები იგივეა რაც ველები, ხოლო კლასის ფუნქციები – მეთოდებია, რომელთა საშუალებითაც C#-ში აღიწერება კლასის ქცევა. მაგალითად,

```
class Lector
{
  int LecID;
  string Gvari_S;
  string Status;
  float Xelfasi;
  char Tel[14];
```

```
  void Lekciis_chatareba( )
  {
    // ქცევის რეალიზაცია
  }

  void Xelfasis_ageba( )
  {
    // ქცევის რეალიზაცია
  }
}
```

ობიექტები, თავიანთი ტიპებისა და განზოგადების მიხედვით ჯგუფდება კლასებად. თუ კლასთა შორის არსებობს გარკვეული იერარქიული კავშირი, იგი შეიძლება იყოს მემკვიდრეობითი. მაგალითად, კლასები Lector და Student შეიცავს მსგავს ატრიბუტებს, როგორებიცაა Gvari, Saxeli, Dabad_Tarigi, Misamarti, Tel და ა.შ.

განსხვავებაა ის, რომ სტუდენტს არ აქვს თანამდებობა (Status), ხელფასი და სხვ., აგრეთვე მას არ გააჩნია მეთოდები ლექციის–კითხვა და ხელფასის აღება. თავის მხრივ, კლასს Student ექნება სტუდბილეთის–ნომერი, ჯგუფი, კურსი, საშუალო–რეიტინ-გული–ბალი და ა.შ. ასევე ჩვენ შეგვიძლია აღვწეროთ კლასები მაგისტრანტი (Magister), დოქტორანტი (Doctorand), თანამშრომელი (Employe) და სხვ. ყველას ექნება როგორც საერთო დამახასიათებელი თვისებები (ატრიბუტები), ასევე განსხვავებული, მხოლოდ მისთვის დამახასიათებელი.

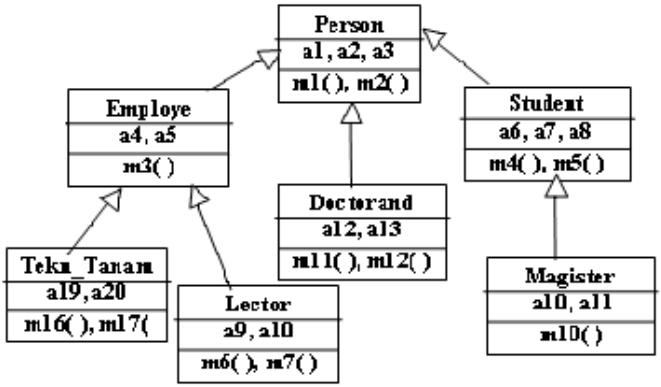
თუ ყველა კლასის საერთო თვისებებს გავიტანთ ცალკე, ხოლო კერძოს დავტოვებთ ამ კლასის ობიექტებთან, მივიღებთ 2.7 ნახაზზე მოცემულ იერარქიულ მოდელს.

ასეთი მოდელი ასახავს მემკვიდრეობით კავშირებს (სამკუთხათავიანი ისრები) კლასებს შორის.

აქ კლასი Person არის აბსტრაქტული და შედგება სამი ატრიბუტისა და ორი მეთოდისგან. იგი განზოგადებაა კლასებისა, ობიექტებით ,,ყველა ადამიანი'' (და არა ცხოველი, ფრინველი), რომელთა შორის ზოგი ლექტორია, ზოგი სტუდენტი, ზოგიც მაგისტრანტი და დოქტორანტი.

ეს უკანასკნელები შეესაბამება კონკრეტულ კლასებს და მათ ყველას აქვს ის ატრიბუტები და მეთოდები, რომლებიც

აბსტრაქტულ კლასში იყო. ასევე, როგორც აღვნიშნეთ, მათ აქვს კიდევ დამატებული თავიანთი კონკრეტული ატრიბუტები და მეთოდები, რაც არ აქვს თავის იერარქიულად ზემდგომ (მშობელ) კლასს.



ნახ.2.7

2.10.2. კომპონენტ-ორიენტირებული დაპროგრამება

კომპონენტ-ორიენტირებული დაპროგრამება არის ობიექტ-ორიენტირებული მეთოდი, იგი ეფუძნება კოდების (პროგრამების) განმეორებით გამოყენების კონცეფციას.

როგორც აღვნიშნეთ, ძირითად ელემენტს კლასი წარმოადგენს. იგი რეალიზებულია კოდის საშუალებით. ამგვარად, კოდების განმეორებად გამოყენებასთან უშუალო კავშირი სწორედ კლასებსა და კლასთაშორის დამოკიდებულებებს აქვს. ამ თვალსაზრისით შეიძლება განვიხილოთ ისეთი ცნებები, როგორიცაა:

- კლასთა ურთიერთქმედება (ობიექტების დონეზეც კი);
- კლასთა კატეგორიები (მაგალითად, მათემატიკური, გრაფიკული და ა.შ.);
- კლასების ბიბლიოთეკა, რომელიც აერთიანებს ერთი კატეგორიის კლასებს;
- კლასის რესურსები, რომლებიც არაა პროგრამები (აუდიო-ვიზუალური კომპონენტები);

- კლასის ფაილები, ესაა კლასების ან მათი ბიბლიოთეკების კომპიუტერში ფიზიკური რეალიზაციის შედეგად მიღებული კომპონენტები.

.NET ტექნოლოგიაში C#-კოდების განმეორებადი გამოყენების ძირითადი ელემენტია ნაკრები (assembly), ამიტომაც მას კომპონენტსაც უწოდებენ.

როგორც აღვნიშნეთ (ნახ.1.3), ნაკრები არის ლოგიკური პაკეტი, რომელიც შედგება MsIL-კოდის, მეტამონაცემებისა და რესურსებისაგან (მაგალითად, გამოსახულებები).

2.11. C# კოდში შეცდომებისა და გამოსარიცხ მოვლენათა დამუშავება

თუ პროგრამის მუშაობის პროცესში იქმნება საგანგებო (კოდის შესრულების არანორმალური) სიტუაცია და კომპიუტერი უნდა ‚ჩამოეკიდოს“, ანუ ადამიანის ჩაურევლად იგი ვერ გადაწყვეტს – რა ქნას, მაშინ ამ დროს აუცილებელია ‚ასეთი გამოსარიცხი შემთხვევის“ აღმოფხვრა.

შესაძლებელია წინასწარ იქნას განსაზღვრული არასასურველ სიტუაციათა ნუსხა და მათი თავიდან აცილების ვარიანტები.

მაგალითად, ნულზე გაყოფა, ტიპების არასწორი გარდაქმნა, მიმართვა არარსებულ ფაილთან და ა.შ. ასეთი შეცდომების ანალიზის მექანიზმი გამოიყენება თითქმის ყვალა თანამედრივე ენაში და მათ შორის C#–იც.

შეცდომის აღმოჩენისა და სათანადო აღდგენითი ფუნქციის შესრულების პროცედურას ‚პროგრამული წყვეტის“ დამუშავებასაც უწოდებენ.

.NET-ტექნოლოგიაში ასეთი ფუნქციები მიეკუთვნება System.Exception კლასს. ამ კონსტრუქციის ზოგადი სინტაკსი C#–ში try, catch და finaly ოპერატორებს იყენებს:

```
try
{
  // კოდის ბლოკი სპეცპირობების დასამუშავებლად
}

catch( 1-წყვეტის–კლასი, წყვეტის–ობიექტის–იდენტ.)
```

```
{
  // 1-წყვეტის–კლასის დასამუშავებელი კოდი
}

catch( 2-წყვეტის–კლასი, წყვეტის–ობიექტის–იდენტ.)
{
  // 2-წყვეტის–კლასის დასამუშავებელი კოდი
}
. . .

finaly
{
  // ყველა შემთხვევაში შესასრულებელი კოდი
}
```

ამგვარად, try ბლოკს შეიძლება ჰქონდეს:

- ერთი ან რამდენიმე catch ბლოკი და არცერთი ან ერთი finaly ბლოკი;
- ერთი finaly ბლოკი და არცერთი catch ბლოკი.

2.12. C# კოდის ორგანიზება სახელსივრცის დახმარებით

სახელსივრცე (namespace) მეტად მნიშვნელოვანი ელემენტია C#–ენაში. იგი ხელს უწყობს კოდის ლოგიკურად სწორ ორგანიზაციას და პროგრამული კონფლიქტების შემცირებას სხვადასხვა იდენტიფიკატორებს შორის.

C#–ენაში ყველა პროგრამა იყენებს **using System** ოპერატორს, რაც მიუთითებს System-სახელთა სივრცეზე. მის ქვეშ მოთავსებულია კლასები, რომლებიც ქმნის სახელთა ქვესივრცეს, მაგალითად: System.Console, System.Data, System.Exception და ა.შ.

საბაზო კლასთა მეთოდებზე მიმართვა ხდება სახელსივრცის გამოყენებით. მაგალითად:

System.Console.WriteLine( ).

ახლა განვიხილოთ სახელთა სივრცის დირექტივები. ესაა C#–ენის ისეთი ელემენტები, რომლებიც იდენტიფიცირებას უკეთებს სახელთა სივრცეს.

აქ გამოიყენება ორი დირექტივა: **using (**გამოიყენებს) და **alias (**ფსევდონიმი**).** მაგალითად:

```
using System;
class ABC
{
  static void Main( )
   {
     // ეკრანზე გამოტანა
     Console.WriteLine("Hello, Penguin !");
   }
}
```

ფსევდონიმის დირექტივის საშუალებით შესაძლებელია კოდში მოცემული სახელთა სივრცის სხვა სახელით გამოყენება. მაგალითად:

```
using System;
using MyAlias = MyCompany.Proj.Nested;  // alias-ის მაგალითი
namespace MyCompany.Proj
{
  public class MyClass
  {
    public static void DoNothing( )
    {
    }
  }
  namespace Nested   // a nested namespace
  {
    public class ClassInNestedNameSpace
    {
      public static void SayHello()
      {
        System.Console.WriteLine("Hello, Penguin !");
      }
    }
  }
}
```

```
public class UnNestedClass
{
  public static void Main( )
  {
    MyAlias.ClassInNestedNameSpace.SayHello();   // using alias
  }
}
 // შედეგი
Hello, Penguin !
```

2.13. მოქმედების და ხილვადობის არეები

პროგრამაში სპეც-ბლოკების ორგანიზებით შესაძლებელია მონაცემთა (ცვლადების) ლოკალური მოქმედების არეებისა და მათი ხილვადობის (კოდის სხვა ნაწილიდან) არეების გამოყენება.

ეს მეტად საჭირო საკითხია მეხსიერების ეფექტური მართვისათვის. მაგალითად, მონაცემები კლასის, ინტერფეისის და სტრუქტურის ბლოკებში გამოცხადდება კოდის ერთ ადგილას, მაგრამ მათი გამოყენება შეიძლება სხვა ადგილას ასეთი ბლოკების დახმარებით.

მეთოდების, თვისებებისა და ინდექსატორებისთვის კი აუცილებელია ამ მონაცემთა გამოცხადება უშუალოდ მათი ამუშავების წინ. ე.ი. ისინი დამალულია კოდის სხვა ნაწილისთვის. მაგრამ შეიძლება მათი ხელახლა გამოცხადება this-ოპერაციით, რაც მეტად ეფექტურია. მაგალითად:

```
using System;
using aMyAlias=System.Security.Permissions.FileIOPermissionAccess;
public class AliasMag
{
 aMyAlias fileAccess;
 public AliasMag()
  {
    fileAccess=aMyAlias.AllAccess;
  }
 public static int Main(string[] args)
  {
    AliasMag myAlias=new AliasMag();
    myAlias.printMyAlias();
    return 0;
  }
```

```
 public void printMyAlias()
  {
    string fileAccess = this.fileAccess.ToString();

    Console.WriteLine("IO Access:{0}", fileAccess);
  }
}
```

აქ განიხილება კლასის წევრების (მაგ., fileAccess, რომლის ტიპი აღწერილ იქნა ფსევდონიმით aMyAlias-ით) ნებადართული და აკრძალული ხელახალი განსაზღვრების შემთხვევები.

`printMyAlias()` მეთოდის მოქმედების არის შიგნით aMyAlias-ის წევრის fileAccess-ის ხილვადობა დამალულია ამ მეთოდში შემოტანილი (გამოცხადებული) სტრიქონული ლოკალური ცვლადის სახელით `string fileAccess`.

ასეთ შემთხვევაში `printMyAlias()` მეთოდის შიგნით მიმართვა fileAccess-ზე ნიშნავს ამ ლოკალური ცვლადის გამოყენებას.

მაგრამ თუ საჭიროა არა ეს ლოკალური, არამედ `AliasMag` კლასის `fileAccess`-ის გამოყენება, მაშინ აუცილებელია `this`-ოპერაციის ჩართვა სტრიქონით:

```
  string fileAccess = this.fileAccess.ToString();
```

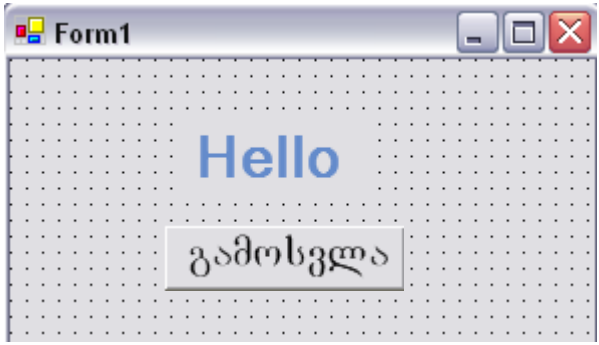
დასასრულ შეიძლება აღვნიშნოთ, რომ სახელთა სივრცე, მოქმედებისა და ხილვადობის არეები, ეს C#–ენის ის კომპონენტებია, რომელთა სწორი გამოყენებით შეიძლება ეფექტური პროგრამების შექმნა.

ერთმანეთში ჩადგმული სახელთა სივრცეების, მონაცემთა მოქმედებისა და ხილვადობის დიაპაზონების მართვის საკითხები მეტად მნიშვნელოვანია დიდი პროექტების ორგანიზების პროცესში. მათი დახმარებით მიიღწევა კოდების ეფექტური გამოყენების დამატებითი შესაძლებლობები.

2.14. VS C# ენის ვიზუალური კომპონენტები

Label – ტექსტის ასახვა ფორმაზე

კომპონენტი გამოიყენება ფორმაზე ტექსტის დასაწერად. ტექსტის შერჩევა ხდება **Text** თვისების საშუალებით, ხოლო ფონტისა და ტექსტის ზომის შერჩევა **Font** თვისებით. ასევე შეიძლება ტექსტის ფერის შეცვლა **ForeColor** თვისებით.



ნახ. 2.8

Button – ღილაკი წარწერით

კომპონენტს აქვს ღილაკის დანიშნულება. მას შეიძლება მიენიჭოს სხვადასხვა ბრძანებები. მაგალითად ღილაკზე თაგუს დაწკაპუნებით დაიხუროს ის ფორმა რომელზედაც მოთავსებულია ეს ღილაკი (ნახ.1.14).

```
private void button1_Click(object sender,
System.EventArgs e)
 {
   Close();
 }
```

TextBox – **სტრიქონის შეტანის ველი ფორმაზე**

კომპონენტი გამოიყენება უშუალოდ ფორმაზე ტექსტის შესატანად. **Text** თვისების საშუალებით TextBox-ში შეიძლება შევიტანოთ ტექსტი, რომელიც ფორმის გააქტიურების შემდეგ გამოჩნდება ან დავტოვოთ ცარიელი.

განვიხილოთ მაგალითი. ფორმაზე მოთავსებულია კომპონენტები: TextBox1, Label1 და Button1. ღილაკზე დაჭერით Label- ში გამოჩნდება ის ტექსტი რომელსაც შევიტანთ TextBox –ის საშუალებით (ნახ.2.9).

ნახ.2.9

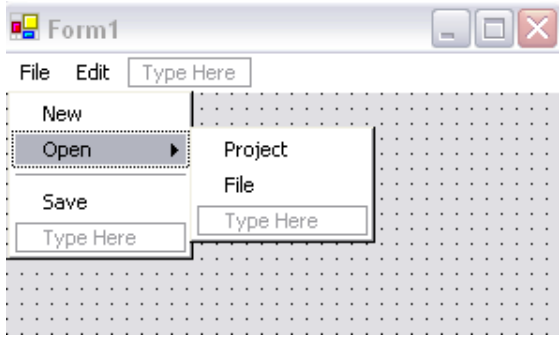
ხოლო ღილაკის ბრძანებას ექნება შემდეგი სახე:

```
private void button1_Click(object sender,
                                System.EventArgs e)
  {
    label1.Text=textBox1.Text;
  }
```

MainMenu – **მთავარი მენიუ**

კომპონენტის საშუალებით ხდება მთავარი მენიუს აგება. გადმოგვაქვს კომპონენტი ფორმაზე და **Text** თვისების საშუალებით

ვადგენთ მენიუს, ხოლო შემდეგ ვააქტიურებთ ფორმას და **Menu** თვისებაში ვაბავთ კომპონენტს (ნახ.2.10).



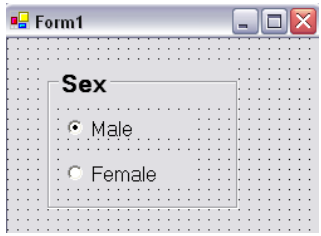
ნახ. 2.10

CheckBox – **მართვის ელემენტი ორი მდგომარეობით**

RadioButton – **რადიო ღილაკი ორი მდგომარეობით**

GroupBox – **ქმნის ფორმაზე ლოგიკურად დაკავშირებულ კომპონენტების კონტეინერს**

კომპონენტში შეიძლება იკოს გაერთიანებული როგორც RadioButton-ები ასევე CheckBox-ები (ნახ.2.11).



ნახ.2.11

PictureBox – **გრაფიკული გამოსახულება**

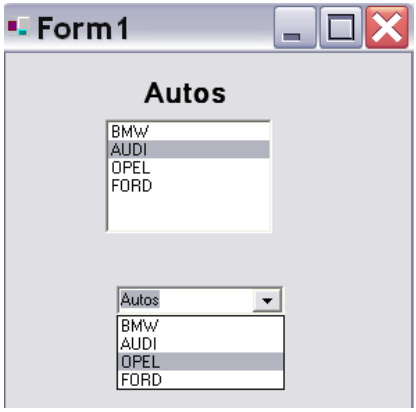
კომპონენტი გამოიყენება ფორმაზე (ან კომპონენტზე) გრაფიკული გამოსახულების გამოსატანად. ფაილის მიბმა ხდება კომპონენტის **Image** თვისებით.

ListBox – **ტექსტური სტრიქონების სიის არე**

კომპონენტში ველების ჩამატება ხდება **Items** თვისებით.

ComboBox –**ქმნის ტექსტური სტრიქონების დინამიკურ სიას და რედაქტირების არეს**

მონაცემებს ვამატებთ **Items** თვისებით. კომპონენტში, ჩამოშლამდე შეგვიძლია გამოვაჩინოთ საწყისი მონაცემი **Text** თვისებით ან დავტოვოთ ცარიელი (ნახ.2.12).



ნახ.2.12

TabControl – **მრავალგვერდიანი დიალოგის ორგანიზება ურთიერთგადაფარვის პრინციპით.**

კომპონენტზე გვერდის დამატება ხდება **AddTab** თვისებით, ხოლო გვერდის სახელის დასარქმევად უნდა მოინიშნოს გვერდი, რომელსაც სახელს უცვლით და ჩავწეროთ **Text** თვისებაში. თითეულ გვერდზე შეგვიძლია დავამათოთ კომპონენტი (ნახ.2.13).

ნახ.2.13

–ქმნის თარიღისა და დროის რედაქტირების არეს

– ქმნის თვის კალენდარს

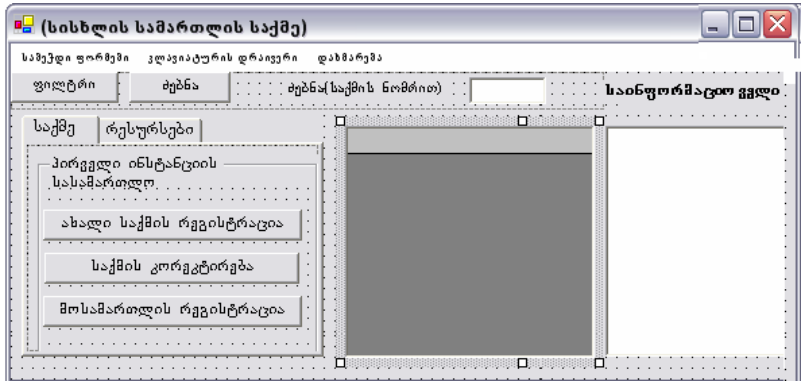
ნახ.2.14

2.15. C# ენის ვიზუალური კომპონენტებით ფორმების აგების მაგალითები

კონკრეტული საპრობლემო სფეროს მართვის ავტომატიზებული სისტემის აგების პროცესში განსაკუთრებული ადგილი უჭირავს ამ სფეროში არსებული დოკუმენტების საფუძველზე მომხმარებელთა დიალოგური ინტერფეისებისა და საბეჭდი ფორმების ატომატიზაციის ამოცანის გადაწყვეტას.

ვიზუალური კომპონენტები NET-ის ინტეგრირებულ C# პაკეტში იძლევა ფართო შესაძლებლობებს და ძალზე მოქნილია.

ამ ინსტრუმენტით ავტორების მიერ აგებულ იქნა, მაგალითად, სასამართლო საქმეთა წარმოების ავტომატიზებული სისტემის დიალოგური ფორმები (ნახ.2.15-2.23).

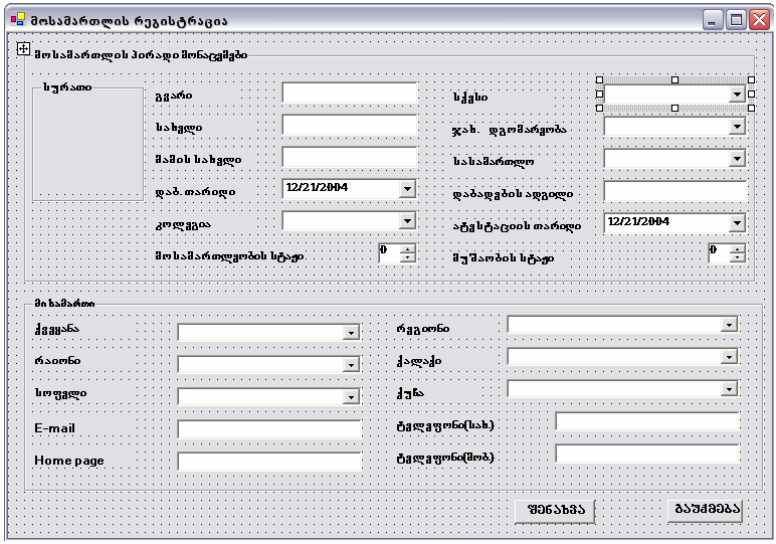


ნახ.2.15

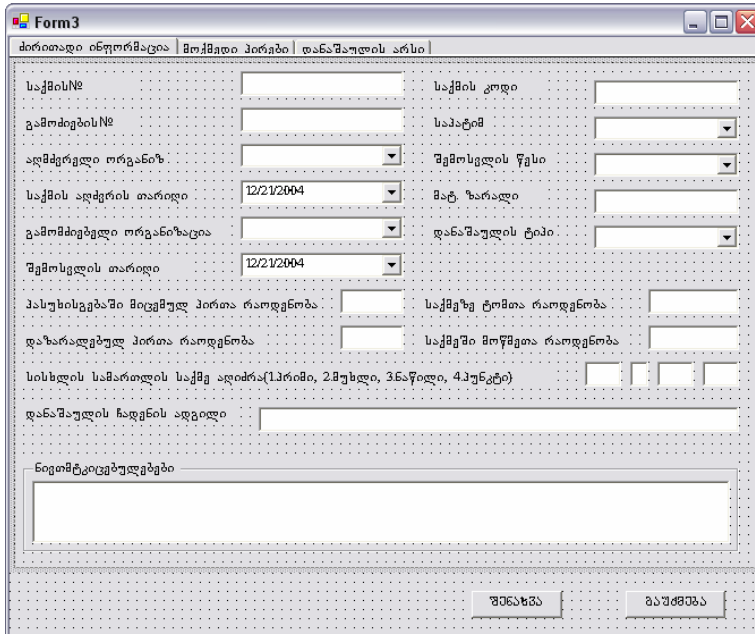
„მოსამართლის რეგისტრაციის“ ღილაკით გამოიტანება ახალი ფანჯარა, რომელიც 2.16 ნახაზზეა ნაჩვენები და მასში შესაძლებელია ინფორმაციის წაკითხვა, შეტანა ან კორექტირება.

„ახალი საქმის რეგისტრაციის“ ღილაკით კი შეიძლება 2.17 ნახაზზე ნაჩვენები ფორმით ვისარგებლოდ.

მონაცემები, რომლებიც შეიტანება ან კორექტირდება ამ და სხვა ფორმებზე, თავსდება სისტემის მონაცემთა ბაზაში და შესაძლებელია მათი შემდგომი გამოყენება. ფორმის შესაბამისი Form1.cs - კოდის ფრაგმენტი ნაჩვენებია პარაგრაფის ბოლოს.



ნახ.2.16



ნახ.2.17

```
//---------------------- Form1.CS ----------------------//
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace WindowsApplication1
{/// <summary>
 /// Summary description for Form1.
 /// </summary>
 public class Form1 : System.Windows.Forms.Form
 {this.tabPage1 = new System.Windows.Forms.TabPage();
  this.tabPage2 = new System.Windows.Forms.TabPage();
  this.groupBox1 = new System.Windows.Forms.GroupBox();
       this.button1 = new System.Windows.Forms.Button();
       this.button2 = new System.Windows.Forms.Button();
       this.button3 = new System.Windows.Forms.Button();
       this.panel1 = new System.Windows.Forms.Panel();
       this.button4 = new System.Windows.Forms.Button();
       this.button5 = new System.Windows.Forms.Button();
this.treeView1 = new System.Windows.Forms.TreeView();
this.richTextBox1=new System.Windows.Forms.RichTextBox();
this.label1 = new System.Windows.Forms.Label();
this.textBox1 = new System.Windows.Forms.TextBox();
this.label2 = new System.Windows.Forms.Label();
this.dataGrid1 = new System.Windows.Forms.DataGrid();
this.tabControl1.SuspendLayout();
this.tabPage1.SuspendLayout();
this.tabPage2.SuspendLayout();
this.groupBox1.SuspendLayout();
this.panel1.SuspendLayout();
((System.ComponentModel.ISupportInitialize)(this.dataGri
d1)).BeginInit();
       this.SuspendLayout();
       // mainMenu1
       this.mainMenu1.MenuItems.AddRange(new
System.Windows.Forms.MenuItem[ ]
    { this.menuItem1, this.menuItem2,this.menuItem3});
       // menuItem1
       this.menuItem1.Index = 0;
       this.menuItem1.Text = "საბეჭდი ფორმები";
this.menuItem1.Click += new System.EventHandler
                                (this.menuItem1_Click);
       // menuItem2
       this.menuItem2.Index = 1;
       this.menuItem2.Text = "კლავიატურის დრაივერი";
```

```
        // menuItem3
        this.menuItem3.Index = 2;
        this.menuItem3.Text = "დახმარება";
        // tabControl1
this.tabControl1.Anchor =
((System.Windows.Forms.AnchorStyles)(((System.Windows.Fo
rms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
System.Windows.Forms.AnchorStyles.Left)));
        this.tabControl1.Controls.Add(this.tabPage1);
        this.tabControl1.Controls.Add(this.tabPage2);
        this.tabControl1.Font = new
System.Drawing.Font("AcadNusx", 9.749999F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
        this.tabControl1.Location = new
System.Drawing.Point(8, 32);
        this.tabControl1.Name = "tabControl1";
        this.tabControl1.SelectedIndex = 0;
        this.tabControl1.Size = new
System.Drawing.Size(224, 400);
        this.tabControl1.TabIndex = 0;
        // tabPage1
        this.tabPage1.Controls.Add(this.groupBox1);
        this.tabPage1.Font = new
System.Drawing.Font("AcadNusx", 12F,
System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
        this.tabPage1.Location = new
System.Drawing.Point(4, 22);
        this.tabPage1.Name = "tabPage1";
this.tabPage1.Size = new System.Drawing.Size(216, 374);
this.tabPage1.TabIndex = 0;
this.tabPage1.Text = "saqme";
// tabPage2
this.tabPage2.Controls.Add(this.treeView1);
this.tabPage2.Location = new System.Drawing.Point(4,25);
this.tabPage2.Name = "tabPage2";
// button4
        this.button4.Font = new
System.Drawing.Font("AcadNusx", 8.25F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
        this.button4.Location = new
System.Drawing.Point(0, 0);
        this.button4.Name = "button4";
this.button4.Size = new System.Drawing.Size(80, 23);
```

```
        this.button4.TabIndex = 0;
        this.button4.Text = "filtri ";
        // button5
        this.button5.Font = new
System.Drawing.Font("AcadNusx", 8.25F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
        this.button5.Location = new
System.Drawing.Point(88, 0);
        this.button5.Name = "button5";
        this.button5.TabIndex = 1;
        this.button5.Text = "Zebna";
        // treeView1
        this.treeView1.Font = new
System.Drawing.Font("AcadNusx", 8.25F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
        this.treeView1.ImageIndex = -1;
        this.treeView1.Location = new
System.Drawing.Point(0, 0);
        this.treeView1.Name = "treeView1";
        this.treeView1.Nodes.AddRange(new
System.Windows.Forms.TreeNode[]
    { new System.Windows.Forms.TreeNode("saqmesTan
dakavSirebuli moqmedi pirebi", new
System.Windows.Forms.TreeNode[] {
        new System.Windows.Forms.TreeNode("advokati")})});
        this.treeView1.SelectedImageIndex = -1;
this.treeView1.Size = new System.Drawing.Size(240, 368);
this.treeView1.TabIndex = 0;
        // richTextBox1
this.richTextBox1.Anchor =
((System.Windows.Forms.AnchorStyles)(((System.Windows.Fo
rms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
System.Windows.Forms.AnchorStyles.Right)));
```

C#-ის პროგრამის ტექსტისათვის, ზოგადად 2.18 ნახაზზე წარმოდგენილია მისი შედგენილობის კონსტრუქციული სქემა. აქ მითითებულია თითოეული using-დირექტივის დანიშნულება.

```
using System;                          // ძირითადი კლასები საერთო გამოყენების ტიპებისთვის
using System.Drawing;                  // სასაზო გრაფიკული ბიბლიოთეკა
using System.Collections;              // ინტერფეისები და კლასები ობიექტთა კოლექციისთვის
using System.ComponentModel;           // კომპონენტთა ქცევის მართვის საშუალებანი
using System.Windows.Forms;              შესრულებისა და დამუშავებისას
using System.Data;                     // Windows-დაწართის ფორმების შექმნის საშუალებანი
                                       // ADO.NET-არქიტექტურის კლასები

namespace WindowsApplication1
{
    /**/
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button button1;
        /**/
        private System.ComponentModel.Container components = null;
        public Form1()...
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )...
        Windows Form Designer generated code

        /**/ [...]
        static void Main() ...
        private void button1_Click(object sender, System.EventArgs e)
        {
            DataForm2 abc= new DataForm2();
            abc.Show();
        }
    }
}
```

ნახ.2.18

საერთოდ, .NET პლატფორმის მნიშვნელოვანი ელემენტია „დანართთა არეები“. მათი დანიშულებაა ერთდროულად და ერთმანეთთან მომუშავე დანართების იზოლაცია, რათა არ მოხდეს მონაცემთა არასასურველი დამუშავება.

პროგრამული დანართების იზოლაციისათვის Windows გამოიყენებს „პროცესის“ ცნებას, რომელიც მისამართების სივრცეს ეხება. ყოველ პროცესს გამოეყოფა 4 გიგაბაიტი ვირტუალური მეხსიერება. ისინი დისკზე სხვადასხვა ფიზიკური მისამართებითაა და არ გადაიკვეთება. პროცესებს აქვს მინიჭებული განსაზღვრული პრივილეგიები და ოპერაციული სისტემა აკონტროლებს მათ, თუ რომელ ოპერაციას რომელი პროცესის გამოყენება შეუძლია.

III თავი

VISUAL STUDIO .NET FRAMEWORK-ის WEB-გვერდების აგების ინსტრუმენტი ASP.NET

3.1. შესავალი ASP.NET სისტემაში

ASP.NET (Active Server Pages – აქტიური სერვერული გვერდები) – არის NET-პლატფორმის ნაწილი და ტექნოლოგია, რომელიც დინამიკურად ქმნის დოკუმენტებს Web-სერვერზე, როცა ისინი მოითხოვება HTTP-ს საშუალებით.

ASP.NET ტექნოლოგია ანალოგიურია PHP, ColdFusion და სხვ. ტექნოლოგიების, მაგრამ მათ შორის მნიშვნელოვანი განსხვავებაცაა. ASP.NET, როგორც მისი დასახელება გვიჩვენებს, შეიქმნა სპეციალურად NET პლატფორმასთან სრული ინტეგრაციის მიზნით, რომლის ნაწილიც ითვალისწინებს C# ენის მხარდაჭერას.

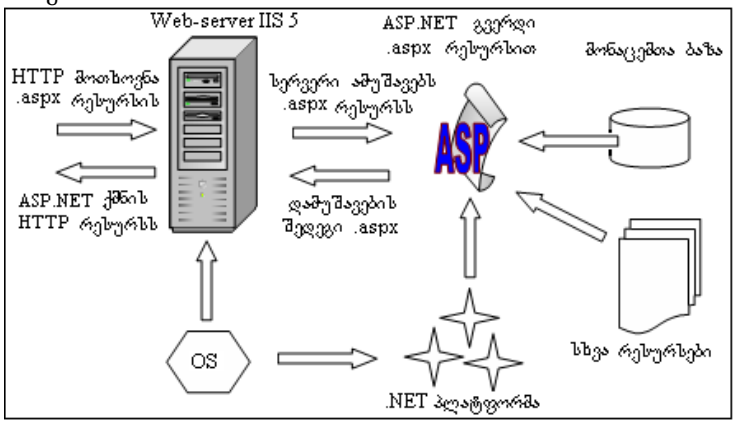
როგორც ცნობილია, Web-გვერდების დასაპროგრამებლად გამოიყენება ისეთი სცენარების ენები, როგორიცაა VBScript ან JScript. ეს სკრიპტული ენები მუშაობდა, მაგრამ ხშირად გარკვეულ პრობლემებს უქმნიდა დაპროგრამების ,,ნამდვილი" ენების პროგრამისტებს სხვადასხვა ადმინისტრატორული დავალებათა შესრულებისას, რაც საბოლოო ჯამში აისახებოდა სისტემის მწარმოებლურობის დაქვეითებაში.

მაღალგანვითარებული ენების გამოყენების შემთხვევაში, მაგალითად, შესაძლებელია მუშაობის პროცესის უზრუნველყოფა სრული სერვერული ობიექტური მოდელით. ASP.NET ახორციელებს მიმართვას გვერდის ყველა მმართველ ელემენტთან, როგორც ზოგადად გარემოს ობიექტებთან. სერვერის მხარესაც კი ხორციელდება წვდომა .NET–ის ყველა საჭირო კლასთან.

გვერდების მართვის ელემენტები ფუნქციონალურია და ფაქტობრივად, შესაძლებელია ყველაფრის გაკეთება, რასაც Windows-ის ფორმის კლასებთან ვაკეთებდით, რაც უფრო მოქნილს ხდის სისტემას. ამის გამო ASP.NET-ის გვერდებს, რომლებიც ქმნის HTML შედგენილობას, ხშირად უწოდებენ Web-ფორმებს.

3.2. ASP.NET–ის საბაზო არქიტექტურა

ASP.NET გამოიყენებს ინტერნეტის ინფორმაციულ სერვერს (IIS – Internet Information Server) HTTP მოთხოვნებზე საპასუხო შინაარსის მისაწოდებლად. ASP.NET გვერდები მოთავსებულია ფაილებში .aspx გაფართოებით. მისი საბაზო არქიტექტურა იხ. 3.1 ნახაზზე.



ნახ.3.1. საბაზო არქიტექტურა

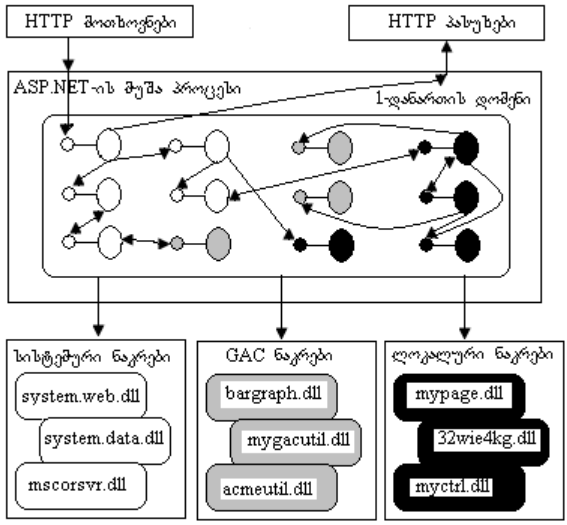
ASP.NET-ის დამუშავების დროს მისაწვდომია .NET-ის ყველა კლასი, სპეციალური კომპონენტები, შექმნილი C# ან სხვა ენებზე, მონაცემთა ბაზები და ა.შ. ფაქტობრივად, სახეზეა ყველა ის შესაძლებლობა, რომელსაც იყენებს C# დანართის აგებისას. ე.ი., C#-ის გამოყენება ASP.NET-ში ეფექტურს ხდის დანართის შესრულებას.

ASP.NET ფაილი შეიძლება შეიცავდეს ნებისმიერ ელემენტს შემდეგი სიიდან:

- ინსტრუქციის დამუშავება სერვერისთვის;
- კოდები ენებზე: C#, VB.NET, Jscript.NET ან სხვ., რომელთა მხარდაჭერა ხდება .NET პლატფორმით;
- შინაარსი ნებისმიერი ფორმით, რომელიც გენერირდება რესურსის სახით HTML–ით;
- ASP.NET -ის ჩადგმული სერვერული მართვის ელემენტები.

ამგვარად, შესაძლებელია ASP.NET ფაილის არსებობა, რომელიც მხოლოდ ერთი სტრიქონისგან შედგება, მაგალითად, Hello: , ყოველგვარი სხვა კოდის ან ინსტრუქციის გარეშე. ამის საფუძველზე იქმნება დასაბრუნებელი HTML გვერდი, რომელშიც მოთავსებულია აღნიშნული ტექსტი.

ამგვარად, ASP.NET-ის ბირთვია .NET-ის კლასების ერთობლიობა, რომელიც ემსახურება HTTP მოთხოვნების დამუშავებას. ზოგიერთი მათგანი ამ კლასებიდან განსაზღვრულია სისტემურ ნაკრებში (Assembly), როგორც .NET-ის საბაზო კლასების ბიბლიოთეკის ნაწილი. ზოგი შეიძლება მოთავსებული იყოს გლობალური კეშის ნაკრებში (GAC – Global Assembly Cache), ზოგიც შეიძლება ჩაიტვირთოს ლოკალური ნაკრებიდან, რომელიც განთავსებულია დანართის ვირტუალურ კატალოგში. ყველა კლასი, რომლებიც ემსახურება HTTP მოთხოვნებს, ჩაიტვირთება დანართის დომენში (დანართის არე – Application area) ASP.NET-ის მუშა პროცესში, და ურთიერთქმედებს ერთმანეთთან, აფორმირებს მოთხოვნებზე პასუხებს. 3.2 ნახაზი ასახავს ASP.NET-ის საბაზო არქიტექტურას დონეების მიხედვით. იგი 3.1 ნახაზის დეტალურად ასახსნელადაა შემოტანილი.



ნახ.3.2. საბაზო არქიტექტურა დონეების მიხედვით

ASP.NET-ის ძირითადი სიახლე მის წინამორბედებთან შედარებით ისაა, რომ აქ ყველა არსი აისახება კლასის საშუალებით, რომელიც ნაკრებიდან ჩაიტვირთება.

დანართების აგება ASP.NET-ში ხდება კლასების კონსტრუირებით, რომლებიც ურთიერთმოქმედებს სხვა კლასებთან. ზოგი კლასი იწარმოება პლატფორმის საბაზო კლასებიდან, ზოგს შეუძლია ინტერფეისების რეალიზება ამ პლატფორმაში, ზოგიც ურთიერთქმედებს პლატფორმის საბაზო კლასებთან მათი მეთოდების გამოძახების გზით.

ASP.NET-ის კლასიკური სინტაკსი ისევ შენარჩუნებულია, ოღონდაც მისი კოდები, რომლებიც ინახება ნაკრების ფაილებში სერვერის მხარეს, გარდაქმნილია კლასების განსაზღვრების სახით.

3.3. ASP გვერდის კოდის მაგალითები

ქვემოთ, 3.1 ლისტინგში მოცემულია ტრადიციული ASP გვერდის მარტივი მაგალითი, რომელშიც JavaScript -ის სერვერული კოდი შერწყმულია HTML–ის სტატიკურ კოდთან. ამ მაგალითში ნაჩვენებია ერთდროულად რამდენიმე კოდირების მეთოდიკა სერვერის მხარეს. კოდში მოთავსებულია სცენარის ბლოკი, რომელიც runat=server ატრიბუტითაა წარმოდგენილი და რომელიც შეიცავს Add( ) ფუნქციას.

ლისტინგი 3.1: ASP გვერდის მაგალითი

```
< ! --  faili test.asp  -- >
<%@  language =’JScript’  %>

<script  language=’JScript’  runat=server>
function Add(x, y)
{
 return x + y;
}
</script>
<html> <body>
 <h1> ASP-is testirebis gverdi </h1>

 <h2> 2+2=<%=Add(2,2) %> </h2>
 <table border=2>
 <%
```

```
   for (var i=0; i<10; i++)  {
 %>
    <tr><td> striqoni <%=i%> sveti-0 </td>
         <td> striqoni <%=i%> sveti-1 </td> </tr>
 <%
      }
 %>
 </table>

 <%
  Response.Write("<h2> Cawerilia uSualod Response-Si </h2>" );
%>
</body> </html>
```

სერვერის მხარეზე სინტაკსი "<%=" გამოიყენება Add() მეთოდის გამოსაძახებლად და შედეგის გამოსატანად h2 ელემენტში, ხოლო სცენარის სინტაკსი სერვერის მხარეზე "<% " კი ცხრილის 10 სტრიქონის პროგრამულად გენერირებისათვის. ჩადგმული ობიექტი Response გამოიყენება ამ სტრიქონების ბოლოში h2 ელემენტის პროგრამულად დასამატებლად.

ASP.NET-ში ფაილი.asp მიიღებს ფაილი.aspx ტიპს, ხოლო კოდის ტექსტი და შედეგები იქნება მსგავსი. ASP.NET- ში JScript-ის ენასთან ერთად გამოიყენება სხვა ენებიც, მაგალითად, C#.NET, VB.NET. 3.2 ლისტინგში ნაჩვენებია C#-ის ვარიანტი:

ლისტინგი 3.2: aspx გვერდის მაგალითი C#-ის და Page დირექტივის გამოყენებით

```
< ! --  faili test.aspx   -- >
<%@  Page language ='C# '  %>

<script  runat=server>
function Add(int x, int y)
{
  return x + y;
}
</script>
<html> <body>
<h1> ASP.NET-is testirebis gverdi </h1>

<h2> 2+2=<%=Add(2,2) %> </h2>
<table border=2>
```

```
<%
  for (var i=0; i<10; i++)  {
%>
   <tr><td> striqoni <%=i%> sveti-0 </td>
        <td> striqoni <%=i%> sveti-1 </td> </tr>
<%
     }
%>
</table>

<%
   Response.Write("<h2> Cawerilia uSualod Response-Si </h2>" );
%>
</body>
</html>
```

დირექტივაში Page უფრო ხშირად ათავსებენ გვერდის დონის ატრიბუტებს, რომლებითაც იმართება ASP.NET გვერდების გამოტანა.

ტრადიციული ASP (ლისტინგი 3.1 .asp) და ახალი ASP.NET (ლისტინგი 3.2 .aspx) შედარებისას განსხვავება მათი მუშაობის მწარმოებლურობაშია. ASP არის ინტერპრეტატორი და ყოველი ახალი გაშვებისას მისი JScript და HTML პროგრამები ინტერპრეტირდება მუშა ფაილებში, რაც მოითხოვს დროს. ASP.NET კი კომპილატორია, იგი ერთხელ (პირველი შესრულების დროს) ქმნის მუშა ფაილებს და შემდეგ ინახავს მას კომპილირებული სახით .NET–ის კლასებში. კომპილირებული კლასი შეიცავს როგორც სერვერული სცენარების კოდებს, ასევე სტატიკურ HTML კოდებს. შემდეგი გაშვებისას ეს კოდები იტვირთება მუშა პროგრამების სახით, რითაც უზრუნველყოფილია მაღალმწარმოებლურობა.

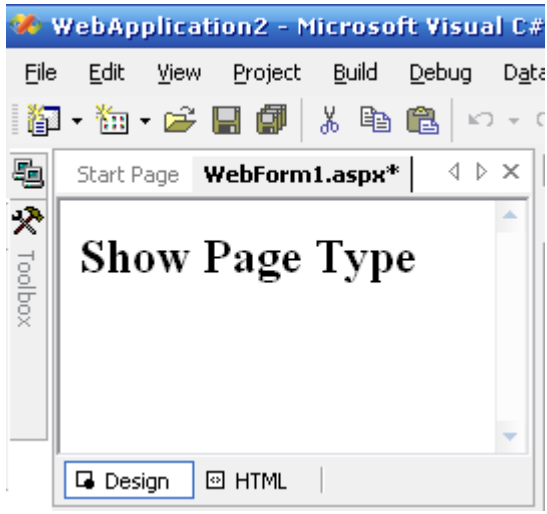
ამგვარად, შეიძლება ვთქვათ, რომ ყოველთვის, როდესაც იქმნება ASP.NET გვერდი, მაშინ იქმნება ახალი კლასი.

3.4. Web.UI.Page კლასი

როგორც ავღნიშნეთ, ყოველი გვერდი კომპილირდება კლასის განსაზღვრებაში. ამიტომაც საინტერესოა, თუ როგორ იქმნება კლასი და როგორ იმართება ეს პროცესი.

ჩვენი პირველი მაგალითისთვის გამოვიტანოთ გვერდის ტიპი და მისი საბაზო ტიპი. 3.3 ნახაზზე ნაჩვენებია .aspx ფაილი და მისი გამოტანა ბროუზერით. გვერდის ტიპი და მისი საბაზო ტიპი დამოტანილია GetType( ) მეთოდით და BaseType თვისებით.

```
< ! --  faili ShowPageType.aspx   -- >
<%@  Page language =’C# ’  %>
<html> <body>

<h2> Show Page Type  </h2>

<%
   Response.Output.Write(“<p>Page type {0} </p>”, this.GetType());
   Response.Output.Write(“<p>Page base type {0} </p>”,
this.GetType().BaseType);
%>
</body>
</html>
```



ნახ.3.3. .aspx-ფაილი და შედეგი VS-ბროუზერში

3.5. ASP.NET-ში მდგომარეობათა მართვა

ASP.NET-ის გვერდების დამახასიათებელი თვისებაა ის, რომ მათ არ გააჩნია მდგმარეობა. ჩვეულებრივად არავითარი ინფორმაცია არ ინახება სერვერზე მომხმარებელთა მოთხოვნებს შორის (თუმცა არსებობს მეთოდები, რომელთაც საჭიროების შემთხვევაში ამის გაკეთება შეუძლია).

ეს საკითხი, თავიდან უცნაურად ჩანს, რადგან მდგომარეობების მართვა ინტერაქტიული სეანსებისთვის მეტად მნიშვნელოვანია მომხმარებლისთვის. ამ თვალსაზრისით ASP.NET გვთავაზობს მეტად მოხერხებულ საშუალებას ასეთი პრობლემის გვერდის ასავლელად და სეანსების მართვის თითქმის გამჭვირვალე პროცესის განსახორციელებლად.

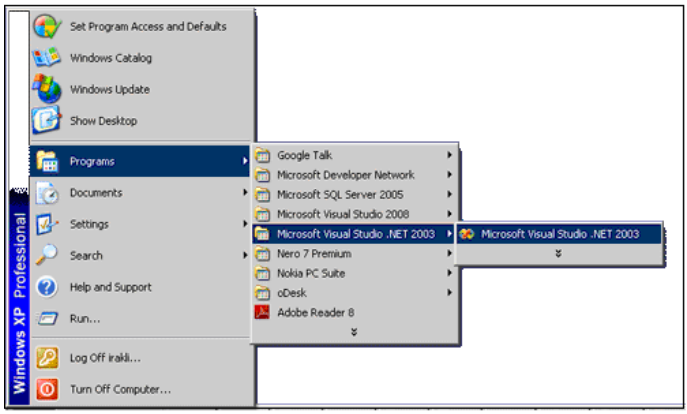
არსებითად, ინფორმაცია Web-ფორმაზე მართვის ელემენტების მდგომარეობათა შესახებ (მონაცემები, შეტანილი ტექსტურ ველებში, ამონარჩევი კომბობოქსებიდან და ა.შ.) ინახება მდგომარეობათა ასახვის დამალულ ველებში (viewstate), რომლებიც გვერდის შემადგენელი ნაწილია. იგი გენერირდება სერვერის მიერ და გადაეცემა მომხმარებელს.

თუ შემდგომში საჭირო იქნება ფორმის მონაცემთა გადაგზავნის ტიპის სერვერული დამუშავება, მაშინ ხდება ამ ინფორმაციის დაბრუნება (postback) სერვერზე. აქ ეს ინფორმაცია გამოიყენება გვერდის ობიექტური მოდელის ხელმეორედ შესავსებად, ლოკალური ცვლილებების ხელით განხორციელების შესაძლებლობით (შემდგომში ამ საკითხს პრაქტიკულად შევეხებით).

3.6. ASP.NET აპლიკაციის შექმნის ეტაპები

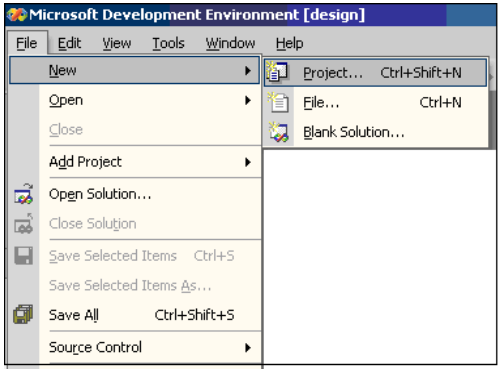
ASP.NET აპლიკაციის შესაქმნელად საჭიროა შემდეგი საფეხურების შესრულება:

1. პროგრამების პანელიდან ავირჩიოთ:
   Start->Programs->Microsoft Visual Studio .NET 2003



ნახ.3.4

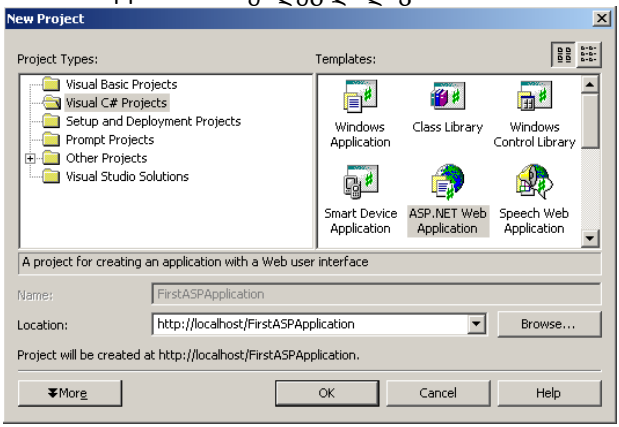
2. პროგრამის გაშვების შემდეგ აირჩიეთ მენიუს პუნქტი:
   File -> New -> Project.



ნახ.3.5

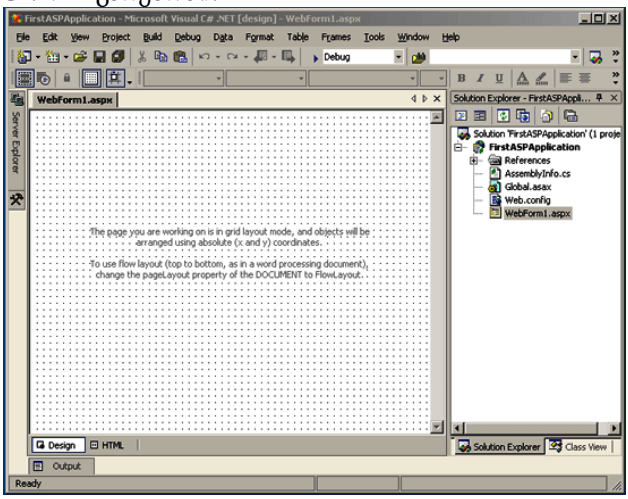
3. გაიხსნება ახალი პროექტების ტიპის არჩევის ფანჯარა. Project Types ფანჯარაში აირჩიეთ Visual C# Project, ხოლო Templates ფანჯარაში ASP.NET Web Application. Location ველში http://localhost/

შემდეგ აკრიფეთ აპლიკაციის სახელი, ამ შემთხვევაში აპლიკაციის სახელია FirstASPApplication. შემდეგ ღილაკი OK.



ნახ.3.6

4. Visual Studio შექმნის აპლიკაციას და გახსნის Microsoft Visual C#.NET გარემოში.



ნახ.3.7

შექმნილ პროექტს შექმისთანავე შეიცავს რამდენიმე ფაილს:

- AssemblyInfo.cs
- Global.asax
- Web.config

- WebForm1.aspx და WebForm1.aspx.cs

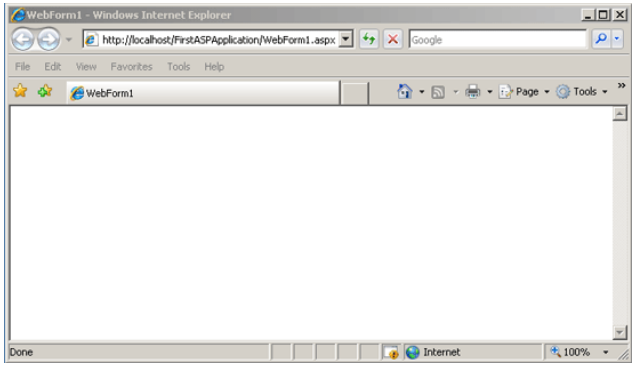
AssemblyInfo.cs არის შეიცავს ინფორმაციას აპლიკაციის შესახებ, როგორიცაა აპლიკაციის სახელი, ავტორი, ვერსია და სხვა.

Global.asax ფაილი ემსახურება აპლიკაციის დონის მოვლენების დამუშავებას, როგორიცაა შეცდომების დაჭერა, ახალი სესიის შექმნა, სესიის დასრულება და სხვა.

Web.config არის XML ფაილი რომელიც შეიცავს აპლიკაციის კონფიგურაციის მონაცემებს: სესიის პარამეტრები, მონაცემთა ბაზასთან კავშირის პარამეტრები, მომხმარებლების ავტორიზაციასა და აუთენთიფიკაციის კონფიგურაციის პარამეტრები.

WebForm1.aspx და WebForm1.aspx.cs ქმნიან ერთ ვებ-გვერდს. WebForm1.aspx ფაილი შეიცავს ვიზუალურ ელემენტებს, ხოლო WebForm1.aspx.cs ვებ-ფორმის კლასის მოვლენების დამუშავების მეთოდებს და ბიზნეს ლოგიკას.

5. შექმნილი ვებ-გვერდის ნახვა შესაძლებელია CTRL+F5 დაჭერით ან მენიუს პუნქტი Debug -> Start Without Debugging არჩევით:



ნახ.3.8

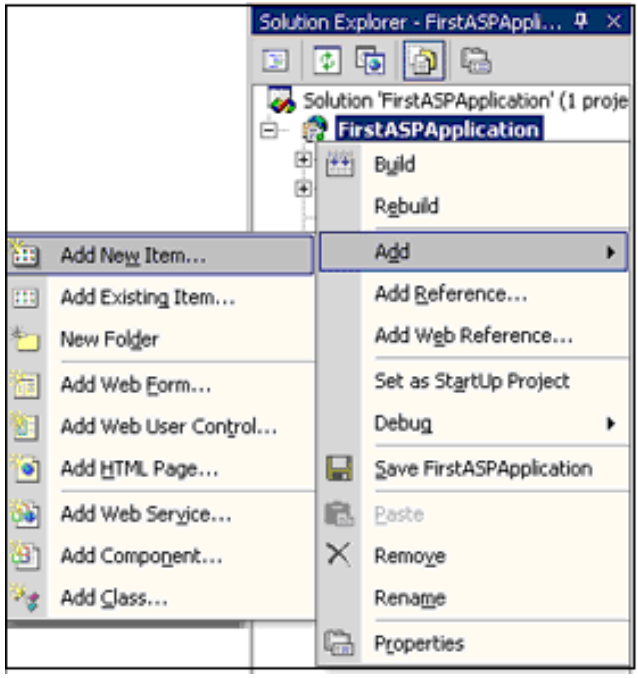
გახსნილი ვებ-გვერდი ცარიელია, რადგან არ შეიცავს ვიზუალურ ელემენტებს.

3.7. ახალი ვებ-გვერდის შექმნა

შექმნილ აპლიკაციას დავამატოთ ახალი ASPX ვებ-გვერდი.

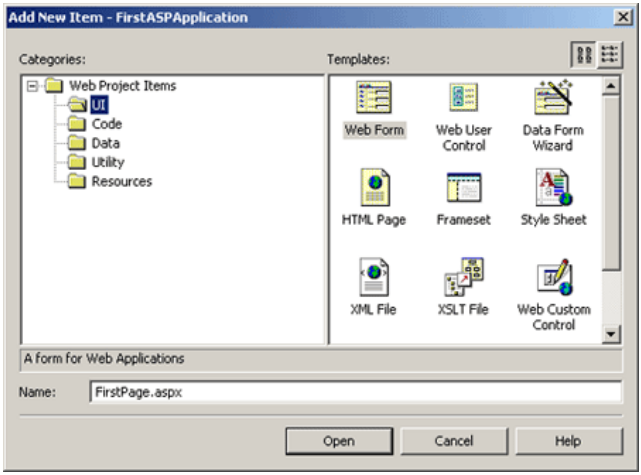
ახალი ფაილის დამატება შესაძლებელია კლავიატურის ღილაკების კომბინაცის CTRL+SHIFT+A საშუალებით, ან მონიშნეთ პროექტი Solution Explorer ფანჯარაში, თაგუს მარჯვენა ღილაკზე დაჭერით გამოიძახეთ კონტექსტური მენიუ და აირჩიეთ:

Add->Add New Item



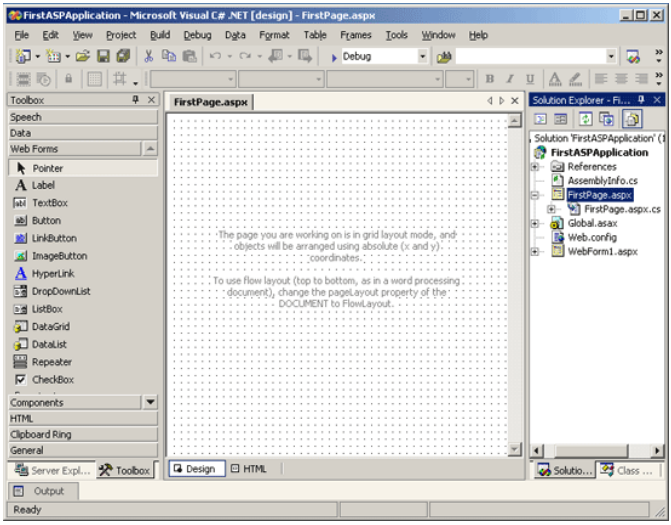
ნახ.3.9

გაიხსნება ახალი ფაილის არჩევის ფანჯარა, მარცხენა ფანჯარაში შესაძებელია ფაილების კატეგორიის არჩევა, ხოლო მარცხენაში ფანჯარაში წინასწარ განსაზღვრეული შაბლონებიდან შესაბამისი ტიპის ფაილების არჩევა. კატეგორიების ფანჯარაში აირჩიეთ UI ხოლო შაბლონების ფანჯარაში Web Form. Name ველში შეიყვანეთ ფაილის სასურველი სახელი:



ნახ.3.10

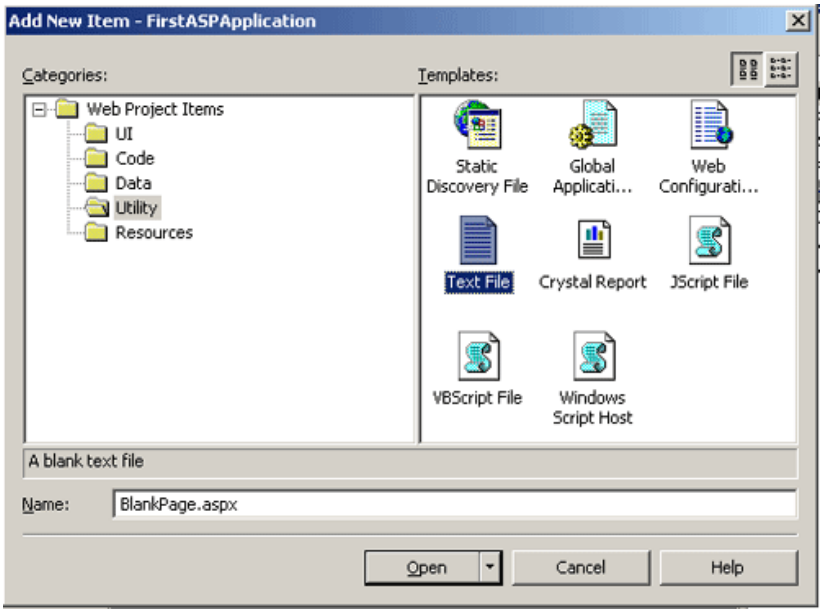
Open ღილაკზე დაჭერით პროექტს დაემატება ახალი ფაილი FirstPage.aspx და FirstPage.aspx.cs. გვერდი ავტომატურად გაიხსნება დიზაინის რეჟიმში.



ნახ.3.11

3.8. ახალი გვერდის დამატება შაბლონის გამოყენების გარეშე

შესძლებელია პროექტს დაემატოს ახალი ვებ-გვერდი შაბლონების გამოყენების გარეშე, ამისათვის გახსენით ახალი ფაილის დამატების ფანჯარა, მონიშნეთ Utility კატეგორია, შაბლონების ფანჯარაში აირჩიეთ Text file. Name ველში აკრიფეთ ფაილის სახელი და გაფართოება მიუთითეთ .aspx, მაგალითად აკრიფეთ BlankPage.aspx.



ნახ.3.12

პროექტს დაემატება შესაბამის ფაილი რომელიც არის ცარიელი (არ შეიცავს HTML და C# კოდს).

3.9. ფუნქციონალური ვებ-გვერდის შექმნა

შევქმნათ ვებ-გვერდი რომელიც ვებ-ბრაუზერის ეკრანზე გამოტანს მიმდინარე თარიღსა და დროს:

1. შექმენით ASP.NET პროექტი და დაამატეთ ცარიელი ASPX გვერდი სახელით CurrentDate.aspx.

2. გახსენით ფაილი და აკრიფეთ შემდეგი ტექსტი:

```
<%@ Page Language="C#" %>
<html>
      <head>
      </head>
      <body>
            <p><%= System.DateTime.Now %></p>
      </body>
</html>
```

ნახ.3.13

გაუშვით პროექტი CTRL+F5 ღილაკების კომბინაციის აკრეფით, ან კონტექსტური მენიუში აირჩიეთ პუნქტი View in Browser. ვებ-ბრაუზერში დაინახავთ მიმდინარე თარიღს და დროს. მაგ.: 20.09.2008 14:19:34.

გვერდის დასაწყისში მოთავსებული დირექტივა-<%@ Page Language="C#" %> მიუთითებს, რომ გვერდი უნდა დაკომპილირდეს C# ენის კომპილატორის საშუალებით. შესაძლებელია C# მაგივრად იყოს VB, რაც ნიშნავს, რომ გვერდის კოდი შექმნილია Visual Basic .Net ენით. ვებ-ბრაუზერში გვერდის გამოძახებისას, დირექტივაში მითითებული ენის კომპილატორი წაიკითხავს და შეასრულებს გვერდზე მოთავსებულ კოდს.

<% %> ტეგებს შორის შესაძლებელია კოდის მოთავსება, რომელიც შესრულდება გვერდის გამოძახების პროცესში სერვერის მხარეს, ხოლო გამოსახულების შედეგი გადაეცემა ვებ-ბრაუზერს. გამოსახულება <%= System.DateTime.Now %> გვერდის გამოძახებისას დააბრუნებს მიმდინარე თარიღის და დროის მნიშვნელობას.

3.10. სერვერული კონტროლების გამოყენება

ჩვეულებრივი HTML კონტროლების გარდაქმნა სერვერულ კონტროლებად შესაძლებელია მისთვის 2 ატრიბუტის დამატებით: Runat და ID. Runat ატრიბუტს მიენიჭება ყოველთვის მიენიჭება მნიშვნელობა Server, ხოლო ID ატრიბუტი წარმოადგენს

კონტროლის იდენტიფიკატორს-სახელს, რომლის საშუალებითაც შესაძლებელია მივმართოთ კონტროლს სერვერილი სკრიპტის კოდში. იგი უნდა იყოს უნიკალური და არ უნდა ემთხვეოდეს სხვა სერვერული კონტროლების იდენტიფიკატორებს.

წინა მაგალითი შესაძლებელია შესრულებული იყოს სერვერული კონტროლის გამოყენებით:

```
<%@ Page Language="C#" %>
<%
      CurrentDate.InnerText =
System.DateTime.Now.ToString();
%>
<html>
      <head>
      </head>
      <body>
            <p id="CurrentDate"
runat="server"></p>
      </body>
</html>
```

ნახ.3.14

<p> ტეგისთვის runat="server" ატრიბუტის მნიშვნებით იგი გარდაიქმნა სერვერულ კონტროლად და შესაძლებელი გახდა გვერდის კოდში მისი მიმართვა.

3.11. Web-კონტროლების გამოყენება

ვებ-კონტროლები სერვერული კონტროლების ნაირსახეობაა. ისინი HTML კონტროლების მსგავსია, ოგონდ უფრო მეტი, რთული თვისებები და მეთოდები გააჩნია. კლიენტის ვებ-გვერდზე ისინი გამოისახება როგორც ერთი ან რამდენიმე HTML კონტროლის სახით.

გადავაკეთოთ წინა მაგალითი ვებ-კონტროლის გამოყენებით:

```
<%@ Page Language="C#" %>
<%
      CurrentDate.Text =
System.DateTime.Now.ToString();
%>
```

```
<html>
 <head>
 </head>
 <body>
  <asp:Label Runat="server"
ID="CurrentDate"></asp:Label>
 </body>
</html>
```

ნახ.3.15

asp:Label არის სერვერული კონტროლი რომელსაც გამოაქვს ტექსტი, რომელიც მას მიენიჭება Text ატრიბუტის საშუალებით.

3.12. Response ობიექტის გამოყენება

Response ობიექტის მეთოდების საშუალებით შესაძლებელია კლიენტს ანუ ვებ-ბრაუზერს გაუგზავნოს მონაცემები.

```
<%@ Page Language="C#" %>
<html>
  <head>
  </head>
  <body>
      <% Response.Write(System.DateTime.Now); %>
  </body>
</html>
```

ნახ.3.16

3.13. სერვერული ფუნქციის გამოყენება

ვებ-გვერდზე სერვერული მეთოდები და ფუნქციები აღიწერება <script> ტეგის საშუალებით.

```
<%@ Page Language="C#" %>
<script runat="server">
 string GetCurrentDateTime(){
  return System.DateTime.Now.ToString();
 }
```

```
</script>
<html>
 <head>
 </head>
 <body>
  <p><%=GetCurrentDateTime()%></p>
 </body>
</html>
```

ნახ.3.17

ამ მაგალითში აღწერილია ფუნქცია GetCurrentDateTime(), რომელიც აბრუნებს მიმდინარე დროის მნიშველობას. ფუნქციის გამოძახება ხდება სერვერული სკრიპტის ტეგებს შორის.

3.14. სერვერული კონტროლების გამოყენება web-გვერდის მოვლენების დამუშავების პროცედურაში

ASPX გვერდის გამოძახებისას ვებ-გვერდის ეკრანზე გამოტანამდე სრულდება რამდენიმე ეტაპი. გვერდზე სხვადასხვა მონაცემების გამოსატანად ძირითადად გამოიყენება გვერდის ჩატვირთვის ეტაპი: Page_Load. გვერდზე სერვერული მხარეს შესასრულებელი კოდი თავსდება <script> ტეგის საშუალებით, რომელსაც ენიჭება ატრიბუტი runat="server".

```
<%@ Page Language="C#" %>
<html>
 <head>
  <script runat="server">
 void Page_Load(Object sender, EventArgs e)
 {
  CurrentDate.Text = System.DateTime.Now.ToString();
 }
  </script>
</head>
  <body>
    <asp:Label Runat="server"ID="CurrentDate"></asp:Label>
  </body>
</html>
```

ნახ.3.18

3.15. Web-გვერდის ვიზუალური და პროგრამული ნაწილების განცალკევება

ASP.NET გვერდები შესაძლოა შედგებოდეს 2 ფაილისგან: 1. ვიზუალური გვერდისგან, სადაც განთავსებულია HTML კოდი და კომპონენტები; 2. კოდის ფაილისგან, სადაც მუშავდება ვებ-ბრაუზერის მომხმარებლის მიერ ინიცირებული სხვადასხვა შეტყობინებები და ბიზნეს ლოგიკის პროცედურები და ფუნქციები. ეს მეთოდი ცნობილია Codebehind-ის სახელწოდებით. განვიხილოთ მაგალითი, ამ მეთოდის გამოყენებით:

1. შექმენით ახალი ვებ-აპლიკაცია ან გახსენით უკვე არსებული;
2. დაამატეთ ახალი ვებ-გვერდი VisualPage.aspx (ცარიელი ASPX ფაილი);
3. დაამატეთ ახალი C# ფაილი ProgramPage.cs (ცარიელი C# ფაილი);
4. VisualPage.aspx შეავსეთ HTML კოდით და სერვერული კონტროლებით:

```
<%@ Page Language="C#" %>
<html>
 <head>
 </head>
 <body>

 </body>
</html>
```

ნახ.3.19

5. ProgramPage.cs ფაილში აკრიფეთ შემდეგი ტექსტი:

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

public class ProgramPage : System.Web.UI.Page
{  private void Page_Load(object sender,
     System.EventArgs e) {   }
```

```
override protected void OnInit(EventArgs e)
  {
   InitializeComponent();
   base.OnInit(e);
  }

  private void InitializeComponent()
  {
   this.Load += new
System.EventHandler(this.Page_Load);
  }
 }
```

6. VisualPage.aspx ფაილში <body></body> ტეგებს შორის დაამატეთ სერვერული კონტროლი:

```
<body>
      <asp:Label Runat="server"
                 ID="CurrentDate"></asp:Label>
</body>
```

ნახ.3.20

7. VisualPage.aspx გვერდის დასაწყისში დაამატეთ დირექტივა:

```
<%@ Page language="c#"
Codebehind="ProgramPage.cs"
AutoEventWireup="false"
Inherits="ProgramPage" %>
```

ეს დირექტივა მიუთითებს, რომ ამ გვერდის მოვლენები დამუშავდება ფაილში ProgramPage.cs და გვერდის შესაბამისი კლასის სახელია ProgramPage.

8. VisualPage.aspx მოთავსებულია სერვერულ კონტროლზე მიმართვისთვის პროგრამის კოდში. საჭიროა ეს კონტროლი აღიწეროს ProgramPage.cs კლასში, ამიტომ მას დაამატეთ ველი:

```
public class ProgramPage : System.Web.UI.Page
 {
      protected System.Web.UI.WebControls.Label
      CurrentDate;
...
```

ნახ.3.21

System.Web.UI.WebControls.Label მიუთითებს რომ ამ ველის ტიპია სერვერული კონტროლი Label-ის კლასი.

9. მიმდინარე თარიღის და დროის გამოსატანად სერვერული კონტროლის საშუალებით გვერდის ჩატვირთვის მოვლენის დამუშავების პროცედურაში-Page_Load მეთოდში კონტროლის მიმართვა ხდება მისი იდენტიფიკატორის მიხედვით CurrentDate და მის ატრიბუტს Text მიენიჭება მიმდინარე თარიღი და დროის მნიშვნლეობა:

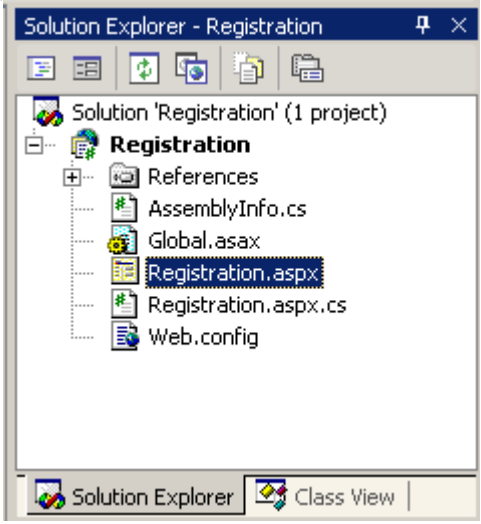
```
   private void Page_Load(object sender,
System.EventArgs e)
  {
   CurrentDate.Text =
System.DateTime.Now.ToString();
  }
```

ნახ.3.22

3.16. ინტერაქტიული Web-გვერდის შექმნა

შევქმნათ ვებ-გვერდი რომელზეც მომხმარებელი შეიყვანს საკუთარ მონაცემებს და გადააგზავნის სერვერზე.

შექმენით ახალი ASP.NET აპლიკაცია სახელით Registration. დაამატეთ ფაილები Registration.aspx და Registration.aspx.cs.



ნახ.3.23

ვებ-გვერდის მოთავსებულია სერვერული კონტროლები: form, asp:TextBox, asp:TextBox, asp:DropDownList, asp:CheckBoxList, asp:Button, asp:Label. გვერდის ჩატვირთვისადსს ღილაკზე დაჭერისას გამოიძახება onclick მოვლენაზე მიბმული მეთოდი Register_Click.

გახსენით Registration.aspx და შეიყვანეთ შემდეგი კოდი:

```
<%@ Page language="c#" Codebehind="Registration.aspx.cs"
  AutoEventWireup="false"
Inherits="FirstASPApplication.Registration" %>
<HTML>
 <HEAD>
  <title>რეგისტრაციის ფორმა</title>
 </HEAD>
 <body>
  <form method="post" runat="server" id="registration">
   შეიყვანეთ საკუთარი მონაცემები:
   <table border="1">
    <tr>
     <td>სახელი:</td>
     <td>
      <asp:TextBox id="FirstName"
runat="server"></asp:TextBox></td>
    </tr>
    <tr>
     <td>გვარი:</td>
     <td>
      <asp:TextBox id="LastName"
runat="server"></asp:TextBox></td>
    </tr>
    <tr>
     <td>სქესი:</td>
     <td><asp:RadioButtonList id="Sex" runat="server"
                    RepeatDirection="Horizontal">
       <asp:ListItem Value="მდედრობითი"></asp:ListItem>
       <asp:ListItem Value="მამრობითი"></asp:ListItem>
      </asp:RadioButtonList></td>
    </tr>
    <tr>
     <td>ქალაქი</td>
     <td><asp:DropDownList id="City" runat="server">
       <asp:ListItem Value="თბილისი"></asp:ListItem>
```

```
        <asp:ListItem Value="ქუთაისი"></asp:ListItem>
        <asp:ListItem Value="რუსთავი"></asp:ListItem>
        <asp:ListItem Value="გორი"></asp:ListItem>
        <asp:ListItem Value="ზათუმი"></asp:ListItem>
        <asp:ListItem Value="თელავი"></asp:ListItem>
       </asp:DropDownList></td>
    </tr>
    <tr>
     <td>ინტერესების სფერო:</td>
     <td>
      <asp:CheckBoxList id="Interests" runat="server">
       <asp:ListItem Value="საინფორმაციო
ტექნოლოგიები"></asp:ListItem>
       <asp:ListItem
Value="სამართალმცოდნეობა"></asp:ListItem>
       <asp:ListItem Value="ეკონომიკა და
მენეჯმენტი"></asp:ListItem>
       <asp:ListItem Value="სამშენებლო
სფერო"></asp:ListItem>
      </asp:CheckBoxList>æ/td>
    </tr>
   </table>
   <asp:Button id="Register" runat="server"
Text="რეგისტრაცია"></asp:Button>
   <br />
   <asp:Label id="Message" runat="server"></asp:Label>
  </form>
 </body>
</HTML>
```

ნახ.3.24

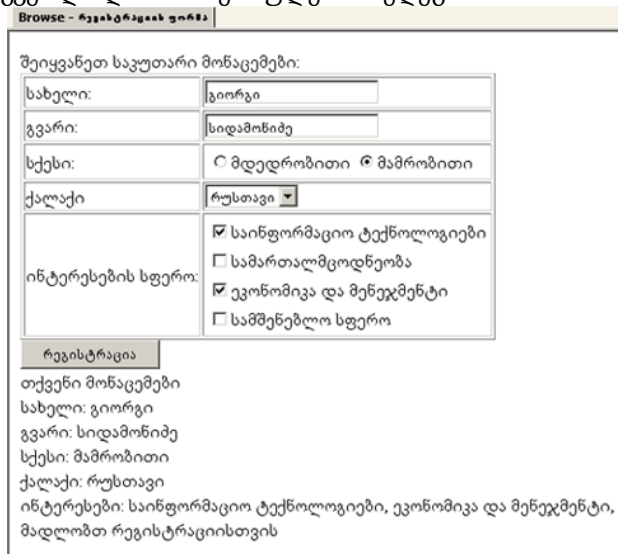
გახსენით Registration.aspx.cs და შეიყვანეთ შემდეგი კოდი:

```
using System;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
// გაგრძელება შემდეგ გვერდზე
```

```
namespace FirstASPApplication
{
 public class Registration : System.Web.UI.Page
  {
   protected System.Web.UI.WebControls.TextBox FirstName;
   protected System.Web.UI.WebControls.TextBox LastName;
   protected System.Web.UI.WebControls.RadioButtonList Sex;
   protected System.Web.UI.WebControls.DropDownList City;
   protected System.Web.UI.WebControls.CheckBoxList
                                                Interests;
   protected System.Web.UI.WebControls.Label Message;
   protected System.Web.UI.WebControls.Button Register;

    private void Register_Click(object sender,
                                System.EventArgs e)
   {
       StringBuilder sb = new StringBuilder();
       sb.Append("თქვენი მონაცემები<br>");
       sb.AppendFormat("სახელი: {0}<br>", FirstName.Text);
       sb.AppendFormat("გვარი: {0}<br>", LastName.Text);
       sb.AppendFormat("სქესი: {0}<br>", Sex.SelectedValue);
    sb.AppendFormat("ქალაქი: {0}<br>", City.SelectedValue);
       sb.Append("ინტერესები: ");
    foreach(ListItem item in Interests.Items)
     {
       if(item.Selected)
         sb.AppendFormat("{0}, ", item.Value);
     }
    sb.Append("<br>მადლობთ რეგისტრაციისთვის");
    Message.Text = sb.ToString();
  }

   override protected void OnInit(EventArgs e)
  {
    InitializeComponent();
    base.OnInit(e);
   }
  private void InitializeComponent()
   {
    this.Register.Click += new
       System.EventHandler(this.Register_Click);
   }
 }
}
```

ნახ. 3.25

ვებ-გვერდი და მისი შესრულების შედეგი ჩანს 3.26 ნახაზზე:



ნახ.3.26

3.17. კონფიგურაციის ფაილები და მათი იერარქია.

ვებ-აპლიკაციათა სერვერების ერთ-ერთი მთავარი მოთხოვნაა კონფიგურაციის მდიდარი და მოქნილი საშუალებების ქონა. ეს მექანიზმი საშუალებას იძლევა, რომ აპლიკაციის კოდში რომელიმე პარამეტრის მნიშვნელობა არ იყოს სტატიკურად აღწერილი და იყოს შესაძლებელი მისი ცვლილება აპლიკაციის კოდში შესწორებების შეტანის და რეკომპილაციის გარეშე. აგრეთვე ვებ სერვერების და აპლიკაციების ადმინისტრატორებს საშუალებას აძლევს ადვილად ცვალონ პარამეტრები აპლიკაციის გაშვების შემდეგ.

კონფიგურაციის ფაილები არის XML ფორმატის, რომელიც შეიცავს აპლიკაციის კონფიგურაციის მონაცემებს: სესიის პარამეტრები, მონაცემთა ბაზასთან კავშირის პარამეტრები, მომხმარებლების ავტორიზაციასა და აუთენთიფიკაციის კონფიგურაციის პარამეტრები და სხვა.

კონფიგურაციის ფაილებში ცვლილებები ავტომატურად აღიქმება სისტემის მიერ და არ საჭიროებს სერვერის გადატვირთვას.

უმეტეს შემთხვევაში კონფიგურაციის ფაილი მოთავსებულია აპლიკაციის ფესვურ დირექტორიაში. web.config არის სპეციალური ფაილი რომელშიც აღწერილი პარამეტრები გამოიყენება დირექტორიაში არსებული ფაილების და კლასების მიერ.

კონფიგურაციის პარამეტრის მნიშვნელობის დადგენა ხდება კონგიგურაციის მთელი იერარქიის დონეების გავლით. თუ პარამეტრის მნიშვნელობა განსაზღვრულია სხვადასხვა დონეებზე, მაშინ მისი იერარქიაში ყველაზე ზედა დონეზე მინიჭებული მნიშვნელობა

ASP.NET -ში არის კონფიგურაციის იერარქიული სტრუქტურა: სერვერის(მანქანური), ვებ-საიტის, აპლიკცაიის, აპლიკაციის ქვეკატალოგების დონეებზე.

კონფიგურაციის ფაილების დონეები და შესაძლო ფიზიკური მისამართები:

- მანქანური: `C:\WinNT\Microsoft.NET\Framework\v.1.00\config\machine.config`
- Web-სერვერის: `C:\inetpub\wwwroot\web.config`
- Web-საიტის: `D:\MyApplication\web.config`
- Web-საიტის ქვედირექტორიის: `D:\MyApplication\MyDir\web.config`

`machine.config` და `web.config` ფაილების XML ფაილებია და ფესვური ელემენტია: `<configuration>`.

კონფიგურაციის ფაილის მაგალითი:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <appSettings>
  <add key="mySetting" value="mySettingValue"></add>
 </appSettings>
 <system.web>
  <compilation defaultLanguage="c#" debug="true" />
  <customErrors mode="RemoteOnly" />
  <authentication mode="Windows" />
  <authorization>
   <allow users="*" />
```

```
  </authorization>
  <trace enabled="false" requestLimit="10"
pageOutput="false" traceMode="SortByTime"
localOnly="true" />
  <sessionState mode="InProc" cookieless="false"
timeout="20" />
  <globalization requestEncoding="utf-8"
responseEncoding="utf-8" />
 </system.web>
</configuration>
```

კონფიგურაციის ფაილების საშუალებით შესაძლებელია განისაზღვროს ცალკეული ვებ-გვერდების და ქვედირექტორიების პარამეტრები `<location>` ტეგის საშუალებით:

```
<configuration>
 <location path="EnglishPages">
<appSettings>
  <add key="BgColor" value="Red"></add>
 </appSettings>
 </location>
<location path="EnglishPages/OneJapanesePage.aspx">
<appSettings>
  <add key="BgColor" value="Green"></add>
 </appSettings>
 </location>
</configuration>
```

ამ შემთხვევაში `EnglishPages` ქვეკატალოგში არსებულ გვერდებზე BgColor პარამეტრს ენიჭება `Red` მნიშვნელობა. მხოლოდ `OneJapanesePage.aspx` გვერდზე ექნება BgColor პარამეტრს განსხვავებული მნიშვნელობა.

შემდეგ მაგალითში კონფიგურაციის ფაილის საშუელაბით განსაძრვრულია რომ აპლიკაციის სესიის დასრულების დრო არის 20 წუთი. ანუ მომხმარებლის სესია დასრულდება მისი ბოლო აქტიურობიდან 20 წუთის შემდეგა და სესიაში დამახსოვრებული მონაცემები დაიკარგება.

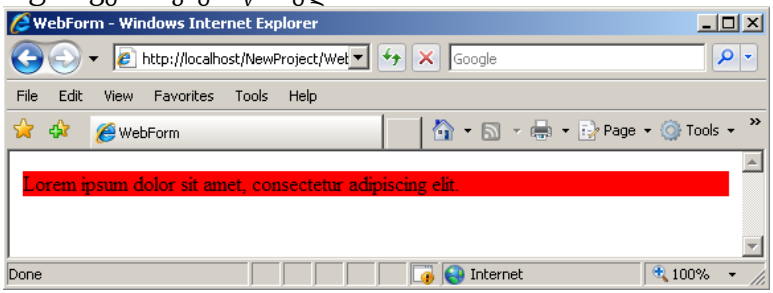
```
<configuration>
  <system.web>
  <sessionState timeout="20" />
</system.web>
</configuration>
```

შემდეგ მაალითში მოყვანილია თუ როგორ არის შესაძლებელი პროგრამის კოდში კონფიგურაციის პარამტრების მნიშვნელობის გაგება. კონფიგურაციის პარამეტრების წვდომისთვის გამოიყენება `ConfigurationSettings` კლასის სტატიკური ფუნქცია `AppSettings`. ამ შემთხვევაში კონფიგურაციის ფაილიდან ხდება მომხმარებლის მიერ განსაზღვრული პარამეტრის BgColor მნიშვნელობის დადგენა და მისი მინიჭება ვებ გვერდის BODY ტეგის BgColor ატრიბუტისთვის.

// ფაილი web.config:
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<appSettings>
 <add key="BgColor" value="Red"></add>
 </appSettings>
</configuration>
```

// ფაილი `WebForm6.aspx.cs`:
```
using System;
using System.Web;
using System.Configuration;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;


namespace NewProject
{
  public class WebForm1 : System.Web.UI.Page
  {
  protected System.Web.UI.WebControls.Panel panel1;

  private void Page_Load(object sender, System.EventArgs
e)
  {
   String settingValue =
ConfigurationSettings.AppSettings["BgColor"];
   panel1.BackColor =
System.Drawing.Color.FromName(settingValue);
  }
 }
}
```

შედეგად გვერდის ჩატვირთვისას `panel1` ელემენტის ფონური ფერი იქნება წითელი

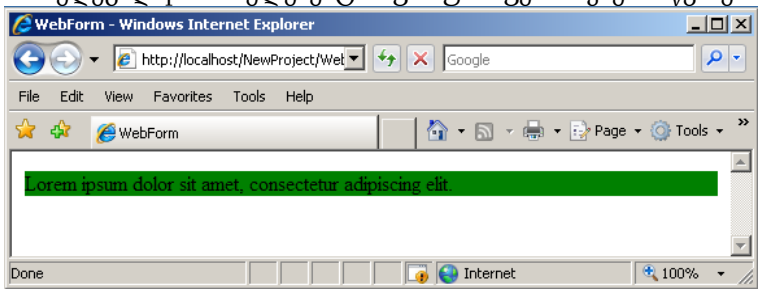


ნახ.3.27

ხოლო თუ web.cofig ფაილში შევცვლით BgColor-ის მნიშვნელობას სხვა ფერის დასახელებით, მაგალითად მწვანე ფერით:

```
<add key="BgColor" value="Green"></add>
```

შედეგად panel1 ელემენტის ფონური ფერი იქნება მწვანე:



ნახ.3.28

3.18. HTTP კონვეიერი. მისი შინაგანი სტრუქტურა. კლასები, მოვლენები, სპეციალიზებული დამმუშავებლები და მოდულები

ASP.NET - ის პლატოფორმა აგებულია მოთხოვნების დამუშავების გაფართოებად არქიტექტურაზე, რომელსაც HTTP კონვეიერი ეწოდება. ყოველი გვერდის გამოძახებისას სრულდება კონვეიერში არსებული სხვადასხვა კლასების გამოძახება, რომლებიც ამ მოთხოვნის დამუშვებას ემსახურება. კონვეიერის გაფართოება სხვა კლასების დამატებით შესაძლებელია სამი გზით:

სპეციალიზებული აპლიკაციით, სპეციუალური დამმუშავებლით და სპეციუალური მოდულით.

სპეციალური აპლიკაცია იქმნება ახალი კლასის შექმნით HttpApplication საბაზო კლასით. სპეციალური აპლიკაცია გამოიყენება აპლიკაციის დონის დავალებების შესასრულებლად სხვადასხვა მოვლენების დაჭერით და მათი დამუშავების გზით.

ვებ პროექტში Global.asax ფაილის დამატებისას აპლიკაციას ემატება კლასი რომელიც წარმოადგენს სპეციალურ აპლიკაციას. ამ კლასის საშუალებით შესაძლებელია სხვადასხვა მოვლენების დაჭერა. მაგალითად, თითოეული მოთხვნის დაწყება და დასასრული, სესიის დაწყება და დასასრული, შეცდომის მოვლენა და სხვ. მასში არსებული ფუნქციონალობის და რესურსებზე წვდომა შესაძლებელია Page და HttpContext კლასების ApplicationInstance თვისების საშაუალებით.

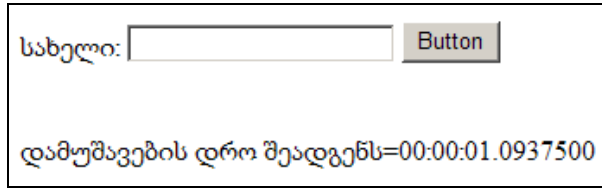
თუ აპლიაკცაის დავამატებთ ფაილს Globa.asax რომელშიც გადატვირთულია მეთოდები `Application_BeginRequest`, `Application_EndRequest` ამ აპლიკაციაში ყველა გვერდის გამოძახებისას გვერიდის ბოლოში გამოჩნდება გვერდის დამუშავების დრო:

// ფაილი Globa.asax:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Web;
using System.IO;
using System.Data;
using System.Web.SessionState;

namespace NewProject
{
 public class Global : System.Web.HttpApplication
 {
  protected void Application_BeginRequest(Object sender,
EventArgs e)
  {
   this.Context.Items["startTime"] = DateTime.Now;
  }
  protected void Application_EndRequest(Object sender,
EventArgs e)
  {
```

```
   DateTime start =
(DateTime)this.Context.Items["startTime"];
   TimeSpan ts = DateTime.Now - start;
   this.Context.Respsonse.Output.Write("<br />დამუშავების
დრო შეადგენს={0}",ts);
  }
 }
}
```

ყოველი გვერდის ბოლოში იქნება შეტყობინება, რომელიც გვერდის გამოძახების დაწყებიდან მის გამოძახების დასრულებამდე გავიდა:



ნახ.3.29

სპეციალური დამმუშავებელი არის კლასი რომლის ბაზური ინტერფეისია IhttpHandler. რომ შევძლოთ ამ დამმუშვებლის გამოძახება აგრეთვე საჭიროა კონფიგურაციის ფაილში ინფორმაციის დამატება, რომელიც მიუთითებს, თუ როდის უნდა მოხდეს ამ კლასის გამოძახება. ასეთი დამმუშავებლის შექმნის მარტივი ხერხია აპლიკაციაში ashx გაფართოების ფაილის დამატება, რაც კონფიგურაციაში დამატებითი პარამეტრების რეგისტრირებას აღარ საჭიროებს.

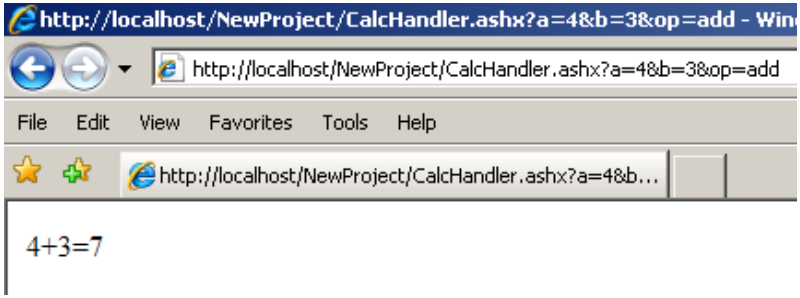
შემდეგ სპეციალურ დამმუშავებელში რეალიზებულია კალკულაციის რამდენიმე ფუნქცია.

// ფაილი CalcHandler:

```
<%@ WebHandler Language="C#" Class="CalcHandler" %>
using System;
using System.Web;
 public class CalcHandler : IHttpHandler
 {
  public void ProcessRequest(HttpContext context)
  {
   int a = Int32.Parse(context.Request["a"]);
   int b = Int32.Parse(context.Request["b"]);
   switch(context.Request["op"])
```

```
    {
    case "add":

context.Response.Output.Write("{0}+{1}={2}",a,b,a+b);
       break;
    case "substract":
       context.Response.Output.Write("{0}-
{1}={2}",a,b,a-b);
       break;
    case "multiply":

context.Response.Output.Write("{0}*{1}={2}",a,b,a*b);
        break;
    default:
       context.Response.Output.Write("ოპერაციის ტიპი
უცნობია");
       break;
   }
  }

  public bool IsReusable{get{return true;}}
 }
```

ამ დამმუშავებლის გამოყენება შესაძლებელია თუ გამოვიძახებთ ამ ფაილს და გადავცემთ მას საჭირო პარამეტრებს. მაგალითად:
http://localhost/NewProject/CalcHandler.ashx?a=4&b=3&op=add:



ნახ.3.30

3.19. შეცდომების დიაგნოსტიკა. ტრასირება და მონიტორის მწარმოებლურობის მთვლელები. გამართვის პროცესი

ASP.NET იძლევა აპლიკაციის მონიტორინგის და სხვადასხვა პრობლემების დიაგნოზის საშუალებას, როგორც შექმნის პროცესში ასევე მისი გაშვების შემდეგაც.

ტრასირება

ტრასირება საშუალებას იძლევა პროგრამის მუშაობის პროცესში შემოწმდეს სხვადასხვა ცვლადების მნიშვნელობები, გამოვიდეს შეტყობინებები, სესიის, აპლიკაციის, Cookies-ის მონაცემები და სხვა.

ცალკეული ვებ-გვერდისთვის ტრასირების ჩასართავად გამოიყენება დირექტივა:

```
<%@ Page Trace="true" %>
```

// ფაილი WebForm1.aspx“

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false"
Inherits="WebApplication1.WebForm1" Trace="true" %>
<HTML>
 <HEAD>
  <title>WebForm1</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <TABLE cellSpacing="0" cellPadding="0" border="0">
    <TR>
     <TD>სახელი:</TD>
     <TD>
      <asp:TextBox id="txtFirstname"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD>გვარი:</TD>
     <TD>
      <asp:TextBox id="txtLastname"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD>ასაკი:</TD>
     <TD>
      <asp:TextBox id="txtAge"
runat="server"></asp:TextBox></TD>
```

```
    </TR>
    <TR>
     <TD></TD>
     <TD>
       <asp:Button id="btnSave" runat="server"
            Text="დამახსოვრება"></asp:Button></TD>
    </TR>
   </TABLE>
  </form>
 </body>
</HTML>
```

ღილაკზე დაჭერის შემთხვევაში მოხდება ტრასირების ინფორმაციაში მომხმარებლის მიერ საკუთარი მონაცემების დამატება, მაგალითად:

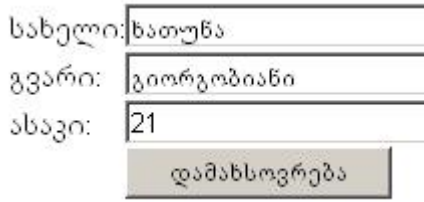
```
Trace.Write("UserDefined",
String.Format("Session[\"Age\"] =
                      {0}", Session["Age"]));
```

`Trace.Warn()` მეთოდით გამოტანილი შეტყობინებები წითელი ფერისაა.

ფაილი WebForm1.aspx-ის ფრაგმენტი:

```
private void btnSave_Click(object sender,
System.EventArgs e)
{
    Session["FirstName"] = txtFirstname.Text;
    Session["LastName"] = txtLastname.Text;
    Session["Age"] = Int32.Parse(txtAge.Text);
    Trace.Warn("სესიაში შენახული მონაცემები:");
    Trace.Write("UserDefined",
           String.Format("Session[\"FirstName\"] = {0}",
           Session["FirstName"]));
    Trace.Write("UserDefined",
           String.Format("Session[\"LastName\"] = {0}",
           Session["LastName"]));
    Trace.Write("UserDefined",
           String.Format("Session[\"Age\"] = {0}",
           Session["Age"]));
}
```

ამ გვერდის ბრაუზერში ნახვისას გამოვა შემდეგი სახის ინფორმაცია (ნახ.3.30-ა,ბ):



ნახ.3.30-ა

Request Details

Session Id:	np14nff2srnoqe45u5six445	**Request Type:**	POST
Time of Request:	12/22/2008 3:42:13 PM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

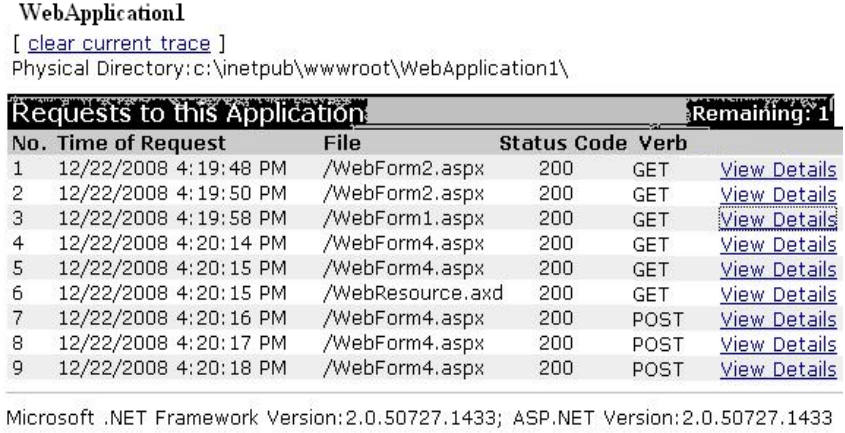
Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	1.57892978078774E-05	0.000016
aspx.page	Begin Init	3.36387230440207E-05	0.000018
aspx.page	End Init	5.27663295312778E-05	0.000019
aspx.page	Begin InitComplete	6.52674434147211E-05	0.000013
aspx.page	End InitComplete	7.78387319550882E-05	0.000013
aspx.page	Begin LoadState	8.98786980930316E-05	0.000012
aspx.page	End LoadState	0.000155722576189628	0.000066
aspx.page	Begin ProcessPostData	0.000172073271252896	0.000016
aspx.page	End ProcessPostData	0.000232909686330422	0.000061
aspx.page	Begin PreLoad	0.000248999732667974	0.000016
aspx.page	End PreLoad	0.000261791570130102	0.000013
aspx.page	Begin Load	0.000273976898057387	0.000012
aspx.page	End Load	0.000287435394760292	0.000013
aspx.page	Begin ProcessPostData Second Try	0.000299550548030654	0.000012
aspx.page	End ProcessPostData Second Try	0.000311229615932989	0.000012
aspx.page	Begin Raise ChangedEvents	0.000323274594546427	0.000012
aspx.page	End Raise ChangedEvents	0.000370091115665657	0.000047
aspx.page	Begin Raise PostBackEvent	0.000384988192835502	0.000015
	სესიაში შენახული მონაცემები:	0.000406957872928177	0.000022
UserDefined	Session["FirstName"] = ხათუნა	0.000425098021743005	0.000018
UserDefined	Session["LastName"] = გიორგობიანი	0.00044005023614329	0.000015
UserDefined	Session["Age"] = 21	0.000459403404027803	0.000019
aspx.page	End Raise PostBackEvent	0.000474415768134022	0.000015

ნახ.3.30-ბ

შესაძლებელია აგრეთვე ტრასირების ჩართვა აპლიკაციის დონეზე:

```
<configuration>
 <system.web>
  <trace enabled="true" enabled="true"
       traceMode="SortByCategory"
       requestLimit="40"
       pageOutput="false"
       localOnly="true"
 />
 </system.web>
</configuration>
```

ამ შემთხვევაში ტრასირება ყველა გვერდისთვის არის ჩართული და მისი გამოტანა შესძლებელია trace.axd ფაილის გამოძახების საშუალებით, რომელიც უნდა მიეთითოს ვებ აპლიკაციის დასახელების შემდეგ. ამ დროს ინახება requestLimit პარამეტრით განსაზღვრული ვებ-გვერდების გამოძახების რაოდენობა.



ნახ.3.31

View Details დაჭერით შესაძლებელია თითოელ ვებ გვერდის გამოძახებასთან დაკავშირებული ტრასირების ინფორმაციის მიღება.

შეცდომების დამუშვება კონფიგურაციის ვებ-საიტის ფაილის დონეზე:

```
<configuration>
 <system.web>
  <customErrors mode="RemoteOnly"
defaultRedirect="/genericerror.htm">
 <error statusCode="500"
redirect="/error/callsupport.htm" />
 <error statusCode="404" redirect="/error/notfound.aspx"
/>
 <error statusCode="403" redirect="/error/noaccess.aspx"
/>
  </customErrors>
 </system.web>
</configuration>
```

defaultRedirect მიუთითებს, რომ თუ შეცდომების კოდების ჩამონათვალში გათვალისწინებული შეცდომისგან განსხვავებული შეცდომა მოხდება საიტი გადამისამართდეს მითითებულ გვერდზე.

ვებ-გვერდის შეცდომის დამუშავება შესაძლებელია ვებ-გვერდის კლასის მეთოდში. ეს მეთოდი ავტომატურად გამოიძახება ამ გვერდზე შეცდომის შემთხვევაში:

```
private void Page_Error(object sender, EventArgs e)
{
 Response.Write(Server.GetLastError().Message);
 Server.ClearError();
}
```

აპლიკაციის დონეზე შეცდომების დაჭერა და დამუშვება შესაძლებელია Global.asax ფაილში. იგი შეიცავს მეთოდს, რომელიც შეცდომის დროს გამოიძახება ავტომატურად. შემდეგ მაგალითში მოცემულია შეცდომის დამუშვება, შეცდომის შესახებ ინფორმაციის ჩაწერა Windows -ის მოვლენათა რეესტრში (EventLog):

```
void Application_Error(Object sender, EventArgs e)
 {
  String Message = "\n\nURL:\n http://localhost/" +
Request.Path
    + "\n\nMESSAGE:\n " + Server.GetLastError().Message
    + "\n\nSTACK TRACE:\n" +
Server.GetLastError().StackTrace;

   String LogName = "Application";
   if (!EventLog.SourceExists(LogName))
   {
```

```
    EventLog.CreateEventSource(LogName, LogName);
   }

   EventLog Log = new EventLog();
   Log.Source = LogName;
   Log.WriteEntry(Message, EventLogEntryType.Error);
 }
```

3.20. შემოწმებათა სისტემა კლიენტისა და სერვერის მხარეს

ASP.NET-ში სტანდარტულ კონტროლებს შორის არის აგრეთვე სერვერული კონტროლები, რომლებიც საშუალებას იძლევა შემოწმდეს მომხმარებლის მიერ ვებ-გვერდზე შეტანილი მონაცემები. არსებობს სხვადასხვა ტიპის შემოწმების კონტროლები, როგორიცაა მნიშვნელობის რაიმე განსაზღვრულ დიაპაზონში შემოწმება, განსაზღვრული შაბლონის მსგავსებაზე შემოწმება, აგრეთვე შემოწმება კონტროლში შეტანილია თუ არა მნიშვნელობა.

შემოწმების კონტროლებია:

- RequiredFieldValidator - ამოწმებს შევსუებული თუ არა კონტროლი
- CompareValidator - ადარებს ორი კონტროლის მნიშნველობებს
- RangeValidator - ამოწმებს შეტანილი მნიშვნელობა არის თუ არა განსაზღვრულ საზღვრებში.
- RegularExpressionValidator - ამოიწმებს ეთანხმება თუ არა შეტანილი მნიშვნელობა განსაზღვრულ შაბლონურ გამოსახულებას.
- CustomValidator - საშუალებას იძლევა მომხმარებლის მიერ განისაზღვროს შემოწმების ლოგიკა.
- ValidationSummary - გამოაქვს შეტყობინება შეცდომების შესახებ, რომლებსაც აბრუნებს ვალიდაციის კონტროლები.

შემოწმება, როგორც წესი ხდება სერვერის მხარეს, თუმცა თუ ვებ-ბროუზერს უზრუნველყოფს DHTML სტანდარტს, შესაძლებელია შესრულდეს შემოწმება კლიენტის მხარეს. ამ შემთხვევაში არ ხდება გვერდის გადაგზავნა სერვერზე, სანამ

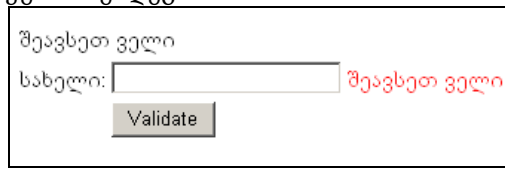
ყველა შემოწმების კონტროლი არ იქნება დასაშვები. როცა კონტროლი არავალიდურია, მაშინ ეკრანზე გამოდის კონტროლის Text ატრიბუტის მნიშვნელობა.

შემდეგ მაგალითში ნაჩვენებია RequiredFieldValidator შემოწმების კონტროლის გამოყენების ხერხი. ეს კონტროლი ამოწმებს შეტანილი არის თუ არა კონტროლში რაიმე მნიშვნელობა. თუ მომხმარებელი შეიტანს რაიმე მნიშვნელობას, მაშინ ის შემოწმების კონტროლი არის ვალიდური. თუ ყველა შემოწმების კონტროლი ვალიდურია, მაშინ ვებ გვერდიც ვალიდურია:

```
<html>
 <head>
  <script language="C#" runat="server">
    void ValidateBtn_Click(Object Sender, EventArgs E)
     { if (Page.IsValid == true)
         {
           lblOutput.Text = "გვერდი ვალიდურია!";
         }
       else {
            lblOutput.Text = "ველის მნიშვნელობა
ცარიელია";
           }
     }
  </script>
 </head>

 <body>
  <form runat="server" ID="Form1">
   <table>
    <tr>
     <td colspan="3">
      <asp:Label ID="lblOutput" Text="შეავსეთ ველი"
runat="server" /><br>
     </td>
    </tr>
    <tr>
     <td align="right">
      სახელი:
     </td>
     <td>
      <ASP:TextBox id="TextBox1" runat="server" />
     </td>
     <td>
      <asp:RequiredFieldValidator
id="RequiredFieldValidator2"
```

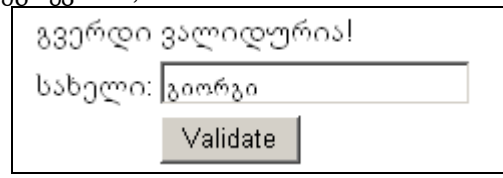
```
ControlToValidate="TextBox1" Display="Static"
Width="100%"
 runat="server" ErrorMessage="შეავსეთ ველი">
</asp:RequiredFieldValidator>
     </td>
    </tr>
    <tr>
     <td></td>
     <td>
      <ASP:Button id="Button1" text="Validate"
OnClick="ValidateBtn_Click" runat="server" />
     </td>
     <td></td>
    </tr>
   </table>
  </form>
 </body>
</html>
```

თუ მომხმარებელი ველს დატოვებს შეუვსებელს, მაშინ ღილაკზე დაჭერის შემდეგ:



ნახ.3.32-ა

ველის შევსების შემთხვევაში ღილაკზე დაჭერის შემდეგ გამოვა შემდეგი ეკრანი;



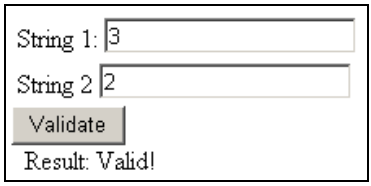
ნახ.3.32-ბ

განვიხილოთ CompareValidator (შედარების) კონტროლი. ეს კონტროლი ამოწმებს ორი სხვადასხვა ველის მნიშვნელობას. ამ კონტროლის ატრიბუტში `ControlToValidate` მიეთითება კონტროლის იდენტიფიკატორი, რომლის მნიშვნელობაც უნდა

შემოწმდეს, ხოლო ატრიბუტით `ControlToCompare` კონტროლის იდენტიფიკატორი, რომლის მნიშვნელობასთანაც უნდა შემოწმდეს. თუ ამ კონტროლების მნიშვნელობები აკმაყოფილებს შედარების პირობებს, მაშინ `CompareValidator` –ის მნიშვნელობა ვალიდურია.

```
<%@ Page clienttarget=downlevel %>
<HTML>
 <HEAD>
  <script language="C#" runat="server">

   void Button1_OnSubmit(Object sender, EventArgs e) {

            if (comp1.IsValid) {
               lblOutput.Text = "Result: Valid!";
            }
            else {
               lblOutput.Text = "Result: Not valid!";
            }
        }
  </script>
 </HEAD>
 <body>
  <form runat="server" ID="Form1">
   <table>
    <tr>
     <td>
      String 1:
      <asp:TextBox id="txtComp"
runat="server"></asp:TextBox>
     </td>
     <td>
     String 2
    <asp:TextBox id="txtCompTo"
runat="server"></asp:TextBox>
     </td>
    </tr>
   </table>
   <asp:Button runat="server" Text="Validate"
ID="Button1" onclick="Button1_OnSubmit" />
   <br>
   <asp:CompareValidator id="comp1"
ControlToValidate="txtComp" ControlToCompare="txtCompTo"
Type="String"
    runat="server" Operator="GreaterThan" />
```

```
   <br>
   <asp:Label ID="lblOutput" runat="server" />
  </form>
 </body>
</HTML>
```



ნახ.3.33

- CustomValidator კონტროლის გამოყენება:

CustomValidator კონტროლი ძირითადად გამოიყენება, მაშინ როცა ვალიდაციის სხვა სტანდარტული კონტროლების მიერ რაიმე პირობების შემოწმება ვერ ხერხდება. ამისთვის ეს კონტროლი იძახებს მომხარებლის მიერ განსაზღვრულ ფუნქციას, რომელშიც შესაძლებელია სხვადასხვა პირობის შემოწმება. შესაძლებელია განისაზღვროს როგორც კლიენტის მხარეზე შემოწმების ფუნქცია, ასევე სერვერულ მხარეზეც. ClientValidationFunction ატრიბუტით განისაზღვრება კლიენტური მხარის ფუნქცია, ხოლო OnServerValidate ატრიბუტით სერვერული მხარის.

შემდეგ მაგალითში CustomValidator კონტროლის გამოყენებით მოწმდება შეყვანილი ციფრი ლუწია თუ კენტი. შემოწმება სრულდება როგორც კლიენტის ასე სერვერის მხარეს:

```
<html>
 <head>
  <script language="C#" runat="server">

 void ValidateBtn_OnClick(object sender, EventArgs e)
  {
    if (Page.IsValid)
       {
        lblOutput.Text = "Page is valid!";
       }
     else
       {
        lblOutput.Text = "Page is not valid! :-(";
       }
   }
```

```
void ServerValidate(object source, ServerValidateEventArgs value)
  { // even number?
    try {
          int num = Int32.Parse(value.Value);
          if (num%2 == 0) {
                  value.IsValid = true;
                   return;
               }
         }
    catch (Exception) {}
    value.IsValid = false;
  }
  </script>
 </head>
 <body>
  <h3><font face="Verdana">CustomValidator
Example</font></h3>
  <p>
   <form runat="server" ID="Form1">
    <asp:Label id="lblOutput" runat="server" Text="Enter
     an even number:" /><br>
    <p>
     <asp:TextBox id="Text1" runat="server" />
     <asp:RequiredFieldValidator
       id="RequiredFieldValidator1" runat="server"
       ControlToValidate="Text1" ErrorMessage="Please
       enter a number"
      Display="Dynamic" ></asp:RequiredFieldValidator>
     <asp:CustomValidator id="CustomValidator1"
       runat="server" ControlToValidate="Text1"
       ClientValidationFunction="ClientValidate"
      OnServerValidate="ServerValidate"
Display="Static">
           Not an even number!
    </asp:CustomValidator>
    <p>
     <asp:Button text="Validate"
      onclick="ValidateBtn_OnClick" runat="server"
      ID="Button1" />
     <script language="javascript">
       function ClientValidate(source, arguments)
       {
          // even number?
          if (arguments.Value%2 == 0)
            arguments.IsValid = true;
```

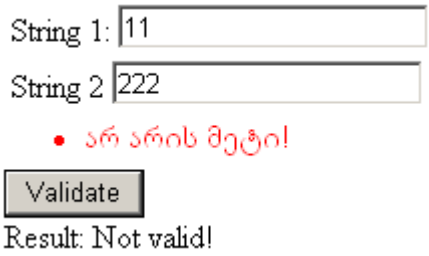
```
          else
            arguments.IsValid = false;
        }
      </script>
    </form>
  </body>
</html>
```

- ValidationSummary კონტროლის გამოყენება:

ამ კონტროლის საშუალებით შესაძლებელია ვალიდაციის კონტროლების შეტყობინებების გამოტანა ერთ სიაში ვებ გვერდის ერთ კონტრეტულ ადგილზე. სხვა ვალიდაციის კონტროლებს აქვთ განსაზღვრული შეცდომის შესახებ შეტყობინება, ატრიბუტით ErrorMessage. ამ ატრიბუტის მნიშვნელობა გამოჩნდება ValidationSummary კონტრლოში. ამ კონტრლის გამოყენებით კონტროლების მნიშვნელობების შედარების მაგალითი შესაძლებელია გადაკეთდეს შემდეგნაირად:

```
<%@ Page clienttarget=downlevel %>
<HTML>
 <HEAD>
  <script language="C#" runat="server">
  void Button1_OnSubmit(Object sender, EventArgs e)
   {
      if (comp1.IsValid)
        {
          lblOutput.Text = "Result: Valid!";
        }
       else {
         lblOutput.Text = "Result: Not valid!";
      }
  }
   </script>
 </HEAD>
 <body>
  <form id="Form1" runat="server">
   <table>
    <tr>
     <td>String 1:
      <asp:textbox id="txtComp"
runat="server"></asp:textbox></td>
    </tr>
```

```
    <tr>
     <td>String 2
      <asp:textbox id="txtCompTo"
runat="server"></asp:textbox></td>
    </tr>
   </table>
     <asp:ValidationSummary id="ValidationSummary1"
      runat="server"></asp:ValidationSummary><asp:button
      id="Button1" onclick="Button1_OnSubmit"
      runat="server" Text="Validate"></asp:button>
     <br>
     <asp:comparevalidator id="comp1" runat="server"
      Type="String" ControlToCompare="txtCompTo"
      ControlToValidate="txtComp"
      Operator="GreaterThan" ErrorMessage="არ არის მეტი!"
      Display="None"></asp:comparevalidator>
    <asp:label id="lblOutput"
runat="server"></asp:label></form>
 </body>
</HTML>
```



ნახ.3.34

3.21. ASP.NET პაკეტში მონაცემებთან მუშაობა

ASP.NET-ში მონაცემებთან მუშაობა საკმაოდ მარტივია. სტანდარტულ ელემენტებს შორის არის მონაცემების გამოტანის და დამუშვების ელემენტებიც. ამ ელემენტების თვისება DataSource და მეთოდი DataBind უზრუნველყოფენ მონაცემთა მიბმას კონტორლებზე. DataSource თვისებას ენიჭება მონაცემთა კოლექცია, მაგალითად მონაცემთა მასივი (კლასის ობიექტი, რომელშიც რეალიზებულია IDataReader ინტერფეისი), ან ობიექტი კლასისა DataSet. როდესაც მზადაა მონაცემების წყარო, რომ მოხდეს მისი წაკითხვა, იძახებენ მეთოდს DataBind(). ამის შემდეგ ელემენტი კითხულობს მონაცემებს და გამოაქვს ეკრანზე გამოტანისთვის საჭირო ფორმატით.

მონაცემების მიბმას უზრულველყოფს სხვადასხვა ელემენტები, ისეთი მარტივი როგორიცაა, მაგალითად ListBox, CheckBoxList, RadioButtonList, DropDownList და სპეციალური ელემენტები DataGrid, DataList DataRepeater. ამ ელემენტებზე შესაძლებელია .NET-ში აღწერილი კოლექციის კლასების ობიექტების მიბმა.

შემდეგ მაგალითშია ნაჩვენები, მონაცემების ელემენტებზე მიბმის ნიმუში. მონაცემები ინახება ArrayList ტიპის ობიექტში, რომელიც ებმება ელემენტების DataSource თვისებას და შემდეგ გმოიძახება ვებ-გვერდის DataBind() მეთოდი, რომლის საშუალებითაც სრულდება ამ ვებ გვერდზე არსებული ყველა ელემენტისთვის DataSource პარამეტრით მინიჭებული მონაცემების გამოტანა ვებ-ბრაუზერში.

ფაილი `WebForm12.aspx.cs`:

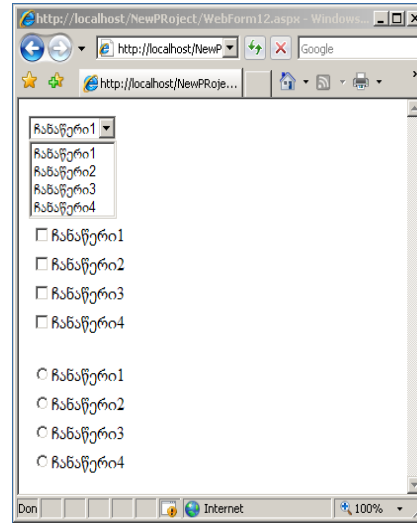
```
<%@ Page language="c#" Codebehind="WebForm12.aspx.cs"
AutoEventWireup="false" Inherits="NewPRoject.WebForm12"
%>
<HTML>
 <HEAD>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:dropdownlist id="ddl1"
runat="server"></asp:dropdownlist><br>
   <asp:listbox id="lb1"
runat="server"></asp:listbox><br>
   <asp:CheckBoxList id="cbl1"
runat="server"></asp:CheckBoxList><br>
```

```
   <asp:RadioButtonList id="rbl1"
runat="server"></asp:RadioButtonList>
    </form>
 </body>
</HTML>
```

ფაილი WebForm12.aspx.cs (ფრაგმენტი):

```
public class WebForm12 : System.Web.UI.Page
{
  protected System.Web.UI.WebControls.DropDownList ddl1;
  protected System.Web.UI.WebControls.ListBox lb1;
  protected System.Web.UI.WebControls.CheckBoxList cbl1;
  protected System.Web.UI.WebControls.RadioButtonList
rbl1;

 private void Page_Load(object sender, System.EventArgs
e)
      {
   if(!Page.IsPostBack)
   {
    ArrayList vals = new ArrayList();
    vals.Add("ჩანაწერი1");
    vals.Add("ჩანაწერი2");
    vals.Add("ჩანაწერი3");
    vals.Add("ჩანაწერი4");

    ddl1.DataSource = vals;
    lb1.DataSource = vals;
    cbl1.DataSource = vals;
    rbl1.DataSource = vals;
    Page.DataBind();
   }
  }
}
```

ბრაუზერში გამოვა შედეგი:



ნახ.3.35

ვებ-გვერდიდან მონაცემთა ბაზასთან დაკავშირება სრულდება ADO.NET ტექნოლოგიის გამოყენებით. არსებობს

მონაცემების ბაზიდან ამოღების 2 მეთოდი: IDataReader ინტერფეისის ნაკადის და DataSet -ის კლასისი ობიექტის საშუალებით.

ყველაზე ეფექტური მეთოდი მონაცემების ამოღების არის IDataReader ინტერფეისის ნაკადის საშუალებით. ამ დროს არ ხდება ამოღებული მონაცემების შენახვა (caching).

შემდეგ მაგალითში შესრულდება ბაზიდან მონაცემების წაკითხვა და რეზულტატის DropDownList ელემენტში გამოტანა. DropDownList ელემენტს აქვს 2 თვისება DataTextField და DataValueField, პირველი მათგანი განსაზღვრავს მონაცემთა ცხრილში ველის დასახელებას, რომლის მნიშვნელობები გამოჩნდება ელემენტის ჩანაწერებში, ხოლო მეორე ელემენტის ჩანაწერთა მნიშვლეობების სვეტის დასახელებას. არსებულ მონაცემთა ბაზაში განსაზღვრულია ცხრილი Authors. ამ ცხრილიდან გამოგვაქვს ავტორის სახელები და მათი იდენტიფიკატორები. სიაში რომელიმე ჩანაწერის არჩევის შემდეგ, ღილაკზე დაჭერის შემთხვევაში ეკრანზე გამოგვავს ავტორის იდენტიფიკატორი.

ფაილი Authors.aspx:

```
<%@ Page language="c#" Codebehind="Authors.aspx.cs"
AutoEventWireup="false" Inherits="NewProject.Authors" %>
<HTML>
 <HEAD>
  <title>Authors</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:DropDownList id="ddlAuthors"
runat="server"></asp:DropDownList>
   <asp:Button id="btnSelect" runat="server"
Text="Button"></asp:Button><BR>
   <asp:Label id="selValue" runat="server"></asp:Label>
  </form>
 </body>
</HTML>
```

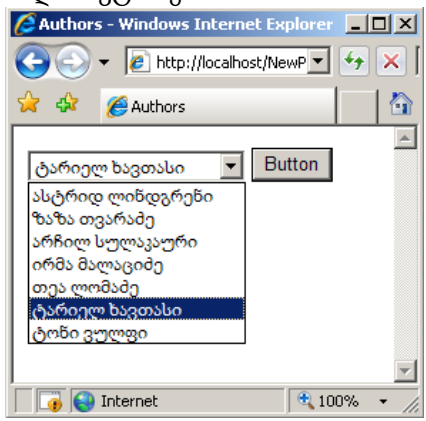
ფაილი Authors.aspx.cs :

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web;
```

```
using System.Web.UI.WebControls;
namespace NewProject
{
public class Authors : System.Web.UI.Page
{
  protected System.Web.UI.WebControls.Label selValue;
  protected System.Web.UI.WebControls.Button btnSelect;
  protected System.Web.UI.WebControls.DropDownList
ddlAuthors;

    private void Page_Load(object sender,
        System.EventArgs e)
         {
          if(!IsPostBack)
          {
           SqlConnection cn = new
               SqlConnection("server=(local);uid=sa;pwd=
               1;database=Books");
           SqlCommand cmd = cn.CreateCommand();
           cmd.CommandText = "SELECT * FROM Authors";

           try
           {
            cn.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            ddlAuthors.DataSource = reader;
            ddlAuthors.DataTextField = "Name";
            ddlAuthors.DataValueField = "ID";
            ddlAuthors.DataBind();
           }
           finally
           {
            cn.Dispose();
           }
          }
       }
       private void btnSelect_Click(object sender,
System.EventArgs e)
       {
          selValue.Text = "თქვენ აირჩიეთ ავტორი: " +
ddlAuthors.SelectedValue;
       }
       override protected void OnInit(EventArgs e)
       {
              InitializeComponent();
             base.OnInit(e);
       }
```

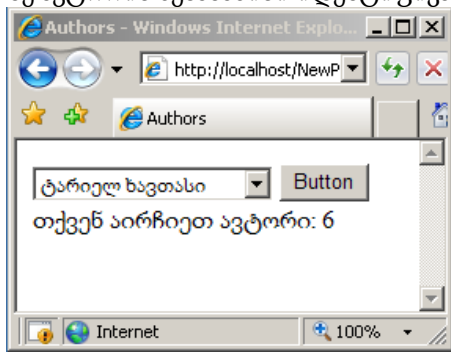
```
        private void InitializeComponent()
        {
                this.Load += new
        System.EventHandler(this.Page_Load);
                btnSelect.Click +=new
        EventHandler(btnSelect_Click);
        }
    }
}
```

ვებ-გვერდზე გამოდის ავტორების სია:



ნახ.3.36

რომელიმე ჩანაწერის არჩევის და ღიალკზე დაჭერის შემდეგ გამოდის ეკრანზე ავტორის შესაბამისი იდენტიფიკატორი:



ნახ.3.37

განვიხილოთ DataSet-ის ობიექტის საშუალებით მონაცემთა გამოტანის მეთოდი. DataSet - ის ობიექტი შესაძლებელია შეიცავდეს

მონაცემების რამდენიმე ცხრილს, ამიტომ საჭიროა მიუთითოთ თუ რომელი ცხრილთან გვსურს კონკრეტულ შემთხვევაში მუშაობა. როდესაც DataSet მიებმევა რომელიმე ელემენტს, ამ ელემენტის DataMember ველის საშუალებით განისაზღვრება რომელი ცხრილის მიბმა არის საჭირო. თუ არ არის მითითებული, მაშინ აიღება პირველი ცხრილი. DataSet-ში არსებული მონაცემების ფილტრაციისა და სორტირებისთვის გამოიყენება DataView ობიექტი.

შემდეგ მაგალითში ნაჩვენებია DataSet ობიექტის გამოყენება, მისი საშუალებით მონაცემების სორტირება და ფილტრაცია.

ფაილი Employees.aspx:

```
<%@ Page language="c#" Codebehind="Employees.aspx.cs"
AutoEventWireup="false" Inherits="NewProject.Employees"
%>
<HTML>
 <HEAD>
  <title>Employees</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:TextBox id="txtFilter"
runat="server"></asp:TextBox>
   <asp:Button id="btnFilter" runat="server"
Text="ფილტრი"></asp:Button><BR>
   <asp:DataGrid id="dgEmployees"
        runat="server"></asp:DataGrid>სორტირება: 
   <asp:Button id="btnSortByID" runat="server"
Text="ID"></asp:Button> ან
   <asp:Button id="btnSortByLName" runat="server"
        Text="LastName"></asp:Button>
  </form>
 </body>
</HTML>
```

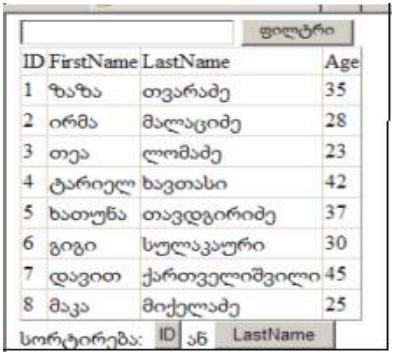
ფაილი Employees.aspx.cs:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.UI.WebControls;
namespace NewProject
{
```

```
 public class Employees : System.Web.UI.Page
 {
  protected System.Web.UI.WebControls.DataGrid
dgEmployees;
  protected System.Web.UI.WebControls.TextBox txtFilter;
  protected System.Web.UI.WebControls.Button btnFilter;
  protected System.Web.UI.WebControls.Button
btnSortByLName;
  protected System.Web.UI.WebControls.Button
btnSortByID;
  protected DataSet _ds;

   private void Page_Load(object sender,
      System.EventArgs e)
       {
        GetData();
        if(!IsPostBack)
        {
           BindGrid();
        }
       }
      private void GetData()
      {
        SqlConnection cn = new
SqlConnection("server=(local);uid=sa;pwd=1;
                                              database=HR");
  SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM
                                               Employees", cn);
   _ds = new DataSet();
   da.Fill(_ds, "Employees");
  }
private void BindGrid()
  {
   dgEmployees.DataSource = _ds;
   dgEmployees.DataBind();
  }
override protected void OnInit(EventArgs e)
  {
   InitializeComponent();
   base.OnInit(e);
  }
private void InitializeComponent()
  {
   this.btnFilter.Click += new
         System.EventHandler(this.btnFilter_Click);
   this.btnSortByLName.Click += new
        System.EventHandler(this.btnSortByLName_Click);
```

```
    this.btnSortByID.Click += new
System.EventHandler(this.btnSortByID_Click);
    this.Load += new System.EventHandler(this.Page_Load);
   }
private void btnFilter_Click(object sender, System.EventArgs e)
   {
    DataView dv = new DataView(_ds.Tables["Employees"]);
    dv.RowFilter = String.Format("LastName like '{0}%'",
         txtFilter.Text.Trim());
    dgEmployees.DataSource = dv;
    dgEmployees.DataBind();
   }
private void btnSortByID_Click(object sender, System.EventArgs e)
   {
    DataView dv = new DataView(_ds.Tables["Employees"]);
    dv.Sort = "ID ASC";
    dgEmployees.DataSource = dv;
    dgEmployees.DataBind();
   }
private void btnSortByLName_Click(object sender,
System.EventArgs e)
   {
    DataView dv = new DataView(_ds.Tables["Employees"]);
    dv.Sort = "LastName ASC";
    dgEmployees.DataSource = dv;
    dgEmployees.DataBind();
   }
  }
}
```

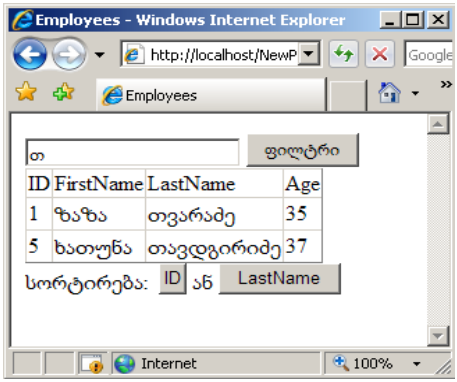
გვერდის პირველი ჩატვირთვისას გამოდის ვებ-ბრაუზერში შემდეგი მონაცემები:



ნახ.3.38

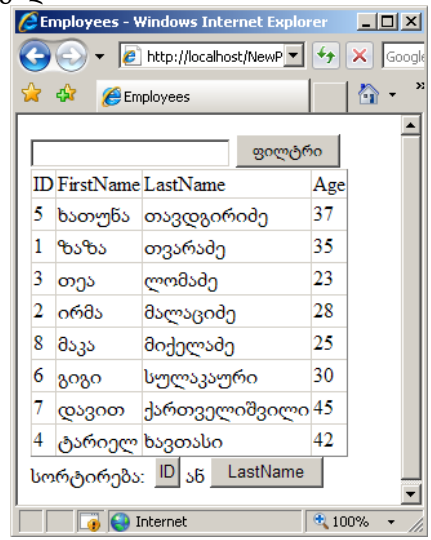
თუ გვინდა გვარის

მიხედვით გაფილტვრა უჯრაში შევიტანთ გვარის პირველ ასოს და დავაწვებით ღილაკს ”ფილტრი”. შემდეგ შემთხვევაში გაფილტრულია ასო ”თ”-თი:



ნახ.3.39

სორტირება გვარის ან იდენტიფიკატორის საშუალებით სრულდება შესაბამის ღილაკებზე დაჭერით. გვარის მიხედვით სორტირების მაგალითი:



ნახ.3.40

3.22. მონაცემთა ბადეები (ცხრილები), სორტირება და რედაქტირება. შაბლონები და მათი ელემენტები

შემდეგ მაგალითში ნაჩვენებია DataGrid ელემენტის საშაულებით მონაცემების გამოტანის და ცლილებების მაგალითი. ცხრილის მონაცემების რედაქტირებისთვის ელემენტს ემატება EditCommandColumn, მისი საშუალებით გამოიძახება ელემენტის სხვადსხვა მეთოდები მოვლენების შემთხევაში, როგორიცაა ჩანაწერის ცვლილება, ცვლიელების გაუქმება, ცვლილების დამახსოვრება.

წაშლისთვის ემატება ღილაკი ButtonColumn რომელიც გადასცემს ბრძანებას Delete და იძახება შესაბამისი მეთოდი. ჩანაწერის განახლების ან წაშლისთვის იქმნება SQL ბრძანება-`SqlCommand`, და მისთვის საჭირო პარამეტერების გადაცემის შემდეგ გამოიძახება მისი მეთოდი `ExecuteNonQuery`, რომელიც მონაცემთა ბაზაში შეასრულებს ოპერაციას.

ფაილი EmployeesEdit.aspx:

```
<%@ Page language="c#"
Codebehind="EmployeesEdit.aspx.cs"
AutoEventWireup="false"
Inherits="NewProject.EmployeesEdit" %>
<HTML>
 <HEAD>
  <title>Employees</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:DataGrid id="dgEmployees" DataKeyField="ID"
runat="server" AutoGenerateColumns="False">
    <Columns>
     <asp:EditCommandColumn ButtonType="LinkButton"
UpdateText="დამახსოვრება" CancelText="უარფოფა"
EditText="შეცვლა"></asp:EditCommandColumn>
     <asp:BoundColumn ReadOnly="True" DataField="ID"
HeaderText="ID"></asp:BoundColumn>
     <asp:BoundColumn DataField="FirstName"
HeaderText="სახელი"></asp:BoundColumn>
     <asp:BoundColumn DataField="LastName"
HeaderText="გვარი"></asp:BoundColumn>
     <asp:BoundColumn DataField="Age"
HeaderText="ასაკი"></asp:BoundColumn>
```

```
     <asp:ButtonColumn Text="წაშლა"
CommandName="Delete"></asp:ButtonColumn>
    </Columns>
   </asp:DataGrid>
  </form>
 </body>
</HTML>
```

ფაილი EmployeesEdit.aspx.cs:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.UI.WebControls;
namespace NewProject
{
 public class EmployeesEdit : System.Web.UI.Page
 {
  protected System.Web.UI.WebControls.DataGrid
dgEmployees;
  protected DataSet _ds;

 private void Page_Load(object sender,
   System.EventArgs e)
   {
   if(!IsPostBack)
   {
    BindGrid();
   }
  }
private void BindGrid()
  {
   SqlConnection cn = new
   SqlConnection("server=(local);uid=sa;pwd=1;
                                    database=HR");
   SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM
                                        Employees", cn);
   _ds = new DataSet();
   da.Fill(_ds, "Employees");
   dgEmployees.DataSource = _ds;
   dgEmployees.DataBind();
  }
override protected void OnInit(EventArgs e)
  {
   InitializeComponent();
   base.OnInit(e);
  }
```

```
private void InitializeComponent()
 {
  this.dgEmployees.CancelCommand += new
   System.Web.UI.WebControls.DataGridCommandEventHandler
      (this.dgEmployees_CancelCommand);
   this.dgEmployees.EditCommand += new
   System.Web.UI.WebControls.DataGridCommandEventHandler
      (this.dgEmployees_EditCommand);
   this.dgEmployees.UpdateCommand += new
   System.Web.UI.WebControls.DataGridCommandEventHandler
      (this.dgEmployees_UpdateCommand);
   this.dgEmployees.DeleteCommand += new
   System.Web.UI.WebControls.DataGridCommandEventHandler
      (this.dgEmployees_DeleteCommand);
   this.Load += new System.EventHandler(this.Page_Load);
  }
private void dgEmployees_EditCommand(object source,
   System.Web.UI.WebControls.DataGridCommandEventArgs e)
  {
   dgEmployees.EditItemIndex = e.Item.ItemIndex;
   BindGrid();
  }
private void dgEmployees_CancelCommand(object source,
   System.Web.UI.WebControls.DataGridCommandEventArgs e)
  {
   dgEmployees.EditItemIndex = -1;
   BindGrid();
  }
private void dgEmployees_UpdateCommand(object source,
   System.Web.UI.WebControls.DataGridCommandEventArgs e)
 {
   string updateCmd = "UPDATE Employees SET FirstName =
      @fName," + " LastName = @lName, Age = @age "+
                                          " WHERE
ID=@id";
  SqlConnection cn = new
   SqlConnection("server=(local);uid=sa;pwd=1;database=Books");
   SqlCommand cmd =  new SqlCommand(updateCmd, cn);

   cmd.Parameters.Add("@fName",
           ((TextBox)e.Item.Cells[2].Controls[0]).Text);
   cmd.Parameters.Add("@lName",
           ((TextBox)e.Item.Cells[3].Controls[0]).Text);
   cmd.Parameters.Add("@age",
           ((TextBox)e.Item.Cells[4].Controls[0]).Text);
```

```
    cmd.Parameters.Add("@id",
        dgEmployees.DataKeys[dgEmployees.EditItemIndex]);

    try
    { cn.Open();
     cmd.ExecuteNonQuery();
     dgEmployees.EditItemIndex = -1;
    }
    finally
    { cn.Dispose();
    }
    BindGrid();
   }
  private void dgEmployees_DeleteCommand(object source,
    System.Web.UI.WebControls.DataGridCommandEventArgs e)
  {
    string deleteCmd = "DELETE FROM Employees WHERE ID=@id";
    SqlConnection cn = new
SqlConnection("server=(local);uid=sa;pwd=1;database=HR");
    SqlCommand cmd =  new SqlCommand(deleteCmd, cn);
cmd.Parameters.Add("@id",dgEmployees.DataKeys[e.Item.Ite
mIndex]);

    try
    { cn.Open();
     cmd.ExecuteNonQuery();
    }
    finally
    { cn.Dispose();
    }
    BindGrid();
   }
 }
}
```

გვერდის ჩატვირთვის შემდეგ DataGrid ელემენტი (ნახ.3.41):

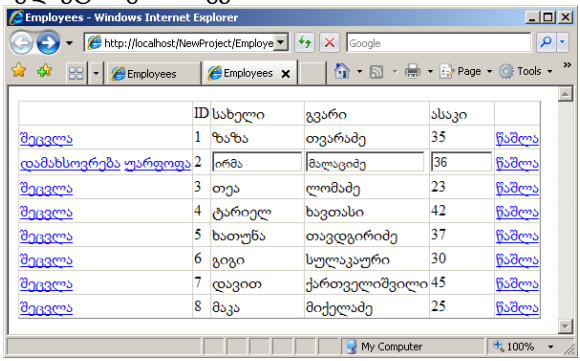
	ID	სახელი	გვარი	ასაკი	
შეცვლა	1	ზაზა	თვარაძე	35	წაშლა
შეცვლა	2	ირმა	მალაციძე	36	წაშლა
შეცვლა	3	თეა	ლომაძე	23	წაშლა
შეცვლა	4	ტარიელ	ხავთასი	42	წაშლა
შეცვლა	5	ხათუნა	თავდგირიძე	37	წაშლა
შეცვლა	6	გიგი	სულაკაური	30	წაშლა
შეცვლა	7	დავით	ქართველიშვილი	45	წაშლა
შეცვლა	8	მაკა	მიქელაძე	25	წაშლა

ნახ.3.41

ღილაკზე ”შეცვლა” დაჭერის შემდეგ გამოიძახება მეთოდი, რომელიც განსაზღვრულია DataGrid კომპონენისთვის ”Edit” ბრძანების მიღები შემთხვევაში:

```
private void dgEmployees_EditCommand(object source,
  System.Web.UI.WebControls.DataGridCommandEventArgs e)
  {
   dgEmployees.EditItemIndex = e.Item.ItemIndex;
   BindGrid();
  }
```

ამ მეთოდის გამოძახების შემდეგ შესაბამისი სსტრიქონი გადადის რედაქტირების რეჟიმში:



ნახ.3.42

”დამახსოვრება” ღილაკზე დაჭერის შემდეგ გამოიძახება მეთოდი `dgEmployees_UpdateCommand`. ამ მეთოდში მყარდება კავშირი მონაცემთა ბაზასთან `SqlConnection` ობიექტის საშუალებით, იქმნება `SqlCommand` რომელსაც გადაეცემა Sql ბრძანება და საჭირო პარამეტრები. `cmd.ExecuteNonQuery()` მეთოდის გამოძახების საშუალებით მონაცემთა ბაზაში შესაბამისი ჩანაწერი განახლდება.

თუ არ არის საჭირო შეცვლილი მონაცემების ცვლილება ”უარყოფა” ღილაკზე დაჭერის შემდეგ რედაქტირებადი სტრიქონი დაბრუნდება ჩვულებრივ რეჟიმში.

”წაშლა” ღილაკზე დაჭერის შემთხვევაში გამოიძახება მეთოდი `dgEmployees_DeleteCommand`. ისევე როგორც ჩანაწერის განახლების შემთხვევაში აქაც იქმნება `SqlCommand` ობიექტი, გადეცემა შესაბამის Sql ბრძანება და `cmd.ExecuteNonQuery()` მეთოდის გამოძახებით შესრულდება ჩანაწერის წაშლა ბაზაში.

შაბლონის საშუალებით მონაცემთა გამოტანას უზრუნველყოფს ელემენტები: DataList, Repeater.

ჩანაწერების გამოტანისთვის წინასწარ განისაზღვრება შაბლონები (ასევე საკმარისია განისაზღვროს მხოლოდ ItemTemplate შაბლონი):

```
ItemTemplate
AlternatingItemTemplate
SeparatorTemplate
HeaderTemplate
FooterTemplate
EditItemTemplate
SelectedItemTemplate
```

DataList-ს აქვს საშუალება ჩანაწერები გამოიტანოს ჰორიზონტალური მიმდევრობით RepeatLayout პარამეტრისთვის Flow მნიშვნელობის მინიჭებით, ან ჩვეულებრივად Table მნიშვნელობის მინიჭებით. წინა მაგალითებში გამოყენებული მონაცემები გამოვიტანოთ DataList –ის საშუალებით:

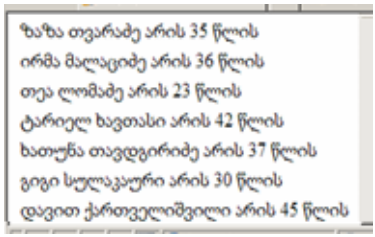
// ფაილი EmployeesView.aspx:

```
<%@ Page language="c#"
Codebehind="EmployeesView.aspx.cs"
AutoEventWireup="false" Inherits="NewProject.Employees"
%>
<HTML>
 <HEAD>
  <title>Employees</title>
 </HEAD>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:datalist id="dlEmployees" runat="server">
    <ItemTemplate>
     <%#DataBinder.Eval(Container.DataItem,
"FirstName")%>
     <%#DataBinder.Eval(Container.DataItem,
"LastName")%>
     არის
     <%#DataBinder.Eval(Container.DataItem, "Age")%>
     წლის
    </ItemTemplate>
   </asp:datalist></form>
 </body>
</HTML>
```

```
// ფაილი EmployeesView.aspx.cs:
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.UI.WebControls;

namespace NewProject
{
 public class EmployeesView : System.Web.UI.Page
 {
  protected System.Web.UI.WebControls.DataGrid
dgEmployees;
  protected System.Web.UI.WebControls.DataList
dlEmployees;
  protected DataSet _ds;

private void Page_Load(object sender, System.EventArgs
e)
  {
   if(!IsPostBack)
   {
    BindGrid();
   }
  }
  private void BindGrid()
  {
   SqlConnection cn = new
SqlConnection("server=(local);uid=sa;pwd=a;database=Book
s");
   SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM
Employees", cn);
   SqlCommandBuilder bldr = new SqlCommandBuilder(da);
   _ds = new DataSet();
   da.Fill(_ds, "Employees");
   dlEmployees.DataSource = _ds;
   dlEmployees.DataBind();
  }
  override protected void OnInit(EventArgs e)
  {
   InitializeComponent();
   base.OnInit(e);
  }
  private void InitializeComponent()
  {
   this.Load += new System.EventHandler(this.Page_Load);
```

```
    }
  }
}
```

ვებ-ბრაუზერში გამოვა შედეგი:



ნახ.3.43

3.23. ASP.NET პაკეტის მართვის სპეციალიზებული ელემენტები. კლასები System.Web.UI.Control და HtmlTextWriter. კლიენტის სცენარის გენერაცია

ASP.NET-ში შესაძლებელია მოხმარებელმა თავად შექმნას სერვერული კონტროლები. ეს მას საშუალებას აძლევს მოახდინოს მომხმარებლის ინტერფეისის და სხვა ფუნქციონალობის ინკაპსულაცია კონტროლში, რომელის გამოყენება აპლიკაციაში მრავალჯერადად არის შესაძლებელი.

მომხმარებლის კონტროლის შექმნა შესაძლებელია `System.Web.UI.Control` ბაზური კლასის გამოყენებით. ჩვეულებრივ უნდა გადაიტვირთოს მისი მეთოდი Render რათა შევძლოთ HTML კოდი გენერაცია. Render მეთოდს გადაეცემა `HtmlTextWriter` ობიექტი, რომლის საშუალებითაც ხდება HTML კოდის გენერაცია.

მომხრებლის კონტროლებს კლასებს შესაძლებელია დაემატოს თვისებები, რომელთა საშალებითაც შესრულდება კონტროლის რომელიმე ველისთვის მნიშვნელობის მინიჭება ან ამ მნიშვნელობის გაგება.

შემდეგ მომხმარებლის კონტროლს დამატებული აქვს 2 თვისება `Message` და `Iterations`.

ამ თვისებებისთვის მნიშვნელობის მინიჭება ხდება ვებ გვერდის კოდში `Message="გამარჯობათ"`, `Iterations="3"`. კონტროლი გამოიტანს მითითებულ ტექსტს (`Message`) მითითებული რაოდენობით (`Iterations`):

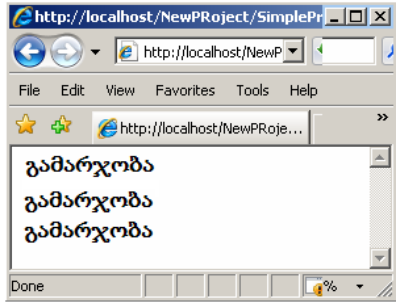
ფაილი SimplePropertyPage.aspx:

```
<%@ Register TagPrefix="SimpleControlSample"
Namespace="NewProject" Assembly="NewProject" %>
<html>
 <body>
  <form method="POST" runat="server" ID="Form1">
   <SimpleControlSample:SimpleProperty Message="Hello
There" Iterations="3" runat="server"
ID="Simpleproperty1" />
  </form>
 </body>
</html>
```

// ფაილი SimpleProperty.cs:

```
using System;
using System.Web;
using System.Web.UI;
namespace NewProject
{
 public class SimpleProperty : Control
 {
  private String _message;
  private int _iterations = 1;

  public String Message
  {
   get { return _message;  }
   set { _message = value; }
  }

  public int Iterations
  {
   get { return _iterations; }
   set { _iterations = value; }
  }

  protected override void Render(HtmlTextWriter output)
  {
   for (int i=0; i<_iterations; i++)
   {
    output.Write("<h1>" + _message + "</h1>");
   }
  }
}
```

}ეკრანზე მიიღება შედეგი:



ნახ.3.44

HtmlTextWriter ობიექტს გააჩნია ასევე სხვა მეთოდები HTML კონტროლების გენერაციისთვის: RenderBeginTag(), AddStyleAttribute(), AddAttribute(). ამ მეთოდების გამოყენება უფრო აადვილებს HTML ბლოკების კონსტრუქტორების კოდის წერას.

ფაილი SimpleTablePage.aspx:

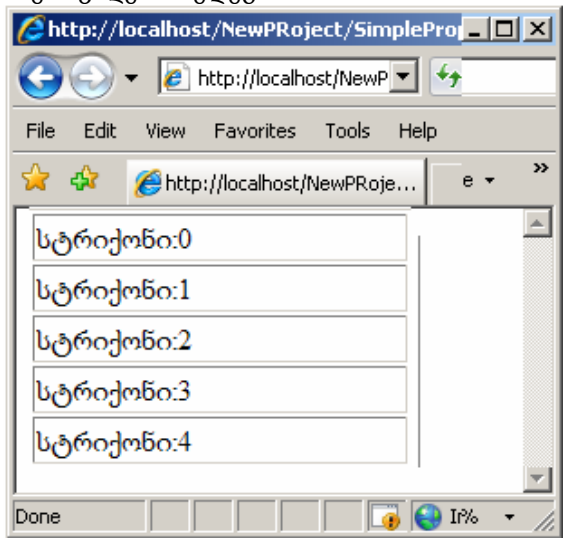
```
<%@ Register TagPrefix="SampleTableSample"
Namespace="NewProject" Assembly="NewProject" %>
<html>
 <body>
  <form method="POST" runat="server" ID="Form1">
       <SampleTableSample:SampleTable Rows="5"
       runat="server" ID="SampleTable1" />
  </form>
 </body>
</html>
```

ფაილი SampleTable.cs:

```
using System;
using System.Web;
using System.Web.UI;

namespace NewProject
{
 public class SampleTable : Control
 {
  private int _rows = 1;
  public int Rows
  {
   set {_rows = value;}
  }
```

```
  protected override void Render(HtmlTextWriter output)
  {
   output.AddAttribute("width", "50%");
   output.AddAttribute("border", "1");
   output.RenderBeginTag(HtmlTextWriterTag.Table);
   for (int i=0; i<_rows; i++)
   {
    output.RenderBeginTag(HtmlTextWriterTag.Tr);
    output.RenderBeginTag(HtmlTextWriterTag.Td);
    output.Write(String.Format("სტრიქონი:{0}",i));
    output.RenderEndTag();
    output.RenderEndTag();
   }
   output.RenderEndTag();
  }
 }
}
```

ეკრანზე მივიღებთ შედეგს:



ნახ.3.45

3.24. მართვის შედგენილი და მომხმარებელთა ელემენტები

თუ ელემენტი შეიცავს სხვა ელემენეტებსაც, ამ შემთხვევაში მათ შედგენილ ელემენტებს უწოდებენ. შედგენილი ელემენტების შექმნა შესაძლებელია ქველემენტების შექმნით და მათი დამატებით მშობელი ელემენტის Controls კოლექციაში. უნდა შესრულდეს Control კლასის CreateChildControls მეთოდის გადატვირთვა, რომელიც გამოიძახება ქველემენტების გენერაციისთვის. თუ კონტროლის გამოყენება მოხდება ერთ გვერდზე მრავალჯერადად, ამ შემთხვევაში საჭიროა, რომ ამ კონტროლში იყოს System.Web.UI.INamingContainer ინტერფეისი რეალიზებული. ეს უზრუნველყოფს ამ კონტროლის ეგზემპლარებისთის უნიკალური იდენტიფიკატორების უზრუნველყოფას.

შემდეგ მაგალითში CreateChildControls მეთოდში ხდება კონტროლების გენერაცია.

ფაილი Composition1.aspx:

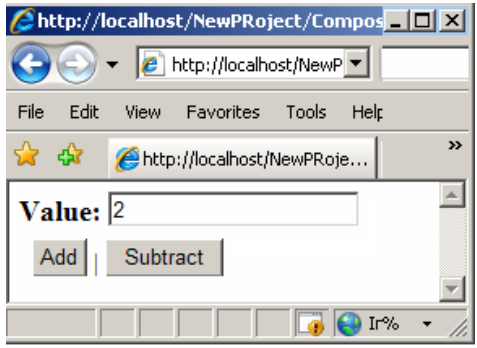
```
<%@ Register TagPrefix="CompositionSample"
Namespace="NewProject" Assembly="NewProject" %>
<html>
 <script language="C#" runat="server">
      private void AddBtn_Click(Object sender, EventArgs
e) {
          MyControl.Value++;
      }
      private void SubtractBtn_Click(Object sender,
EventArgs e) {
          MyControl.Value--;
      }
 </script>
 <body>
  <form method="POST" action="Composition1.aspx"
runat="server" ID="Form1">
   <CompositionSample:Composition1 id="MyControl"
runat="server" />
   <br>
   <asp:button text="Add" OnClick="AddBtn_Click"
     runat="server" ID="Button1" NAME="Button1" />
   |
```

```
   <asp:button text="Subtract"
     OnClick="SubtractBtn_Click" runat="server"
     ID="Button2" NAME="Button2" />
  </form>
 </body>
```

ფაილი Composition1.cs:

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace NewProject
{
 public class Composition1 : Control, INamingContainer
 {
  public int Value
  {
   get
   {
    this.EnsureChildControls();
    return Int32.Parse(((TextBox)Controls[1]).Text);
   }
   set
   {
    this.EnsureChildControls();
    ((TextBox)Controls[1]).Text = value.ToString();
   }
  }

  protected override void CreateChildControls()
  {
   this.Controls.Add(new LiteralControl("<h3>" + "Value:
"));
   TextBox box = new TextBox();
   box.Text = "0";
   this.Controls.Add(box);
   this.Controls.Add(new LiteralControl("</h3>"));
  }
 }
}
```

ეკრანზე გამოდის ფორმა სადაც არის ტექსტური ველი და ორი ღილაკი: Add და Substract. Add ღილაკზე დაჭერით ტექსტურ ველში არსებული რიცხვი გაიზრდება 1-ით, ხოლო Substract ღილაკზე დაჭერით შემცირდება 1-ით.



ნახ.3.46

აქამდე განვიხილეთ შედგენილი ელემენტების შექმნის ხერხი მხოლოდ პროგრამული კოდის საშუალებით, თუმცა ასევე შესაძლებელია მათი შექმნა სპეციალური გვერდების საშუალებით, რომლებსაც მომხმარებელთა ელემენტებს უწოდებენ (user controls). ამ გვერდების გაფართოებაა .ascx, მათზე ძესაძლებელია სხვადასხვა HTML და სერვერული კონტროლების განთავსება. მომხმარებლის ელემენტების გამოყენება შესაძლებელია aspx გვერდებზე გვერდის @Register დირექტივის Src ატრიბუტის საშუალებით. თუ საჭიროა კონტროლის ჩატვირთვა პროგრამულად მაშინ გამოიყენება გვერდის მეთოდი LoadControl(), რომელსაც გადაეცემა მომხარებლის ელემენტის ფაილის მისამართი.

ქვემოთ მოცემულია მომხმარებლის ელემენტის მაგალითი, რომლეშიც ხდება Label კონტროლისთვის ტექსტის ცვლილება:

ფაილი WebUserControl1.ascx:

```
<%@ Control Language="c#" AutoEventWireup="false"
Codebehind="WebUserControl1.ascx.cs"
Inherits="NewProject.WebUserControl1"
TargetSchema="http://schemas.microsoft.com/intellisense/
ie5"%>
<asp:Label ID="lblMessage" Runat="server"></asp:Label>
```

ფაილი WebUserControl1.ascx.cs:

```
namespace NewProject
{
 using System;
 public class WebUserControl1 : System.Web.UI.UserControl
```

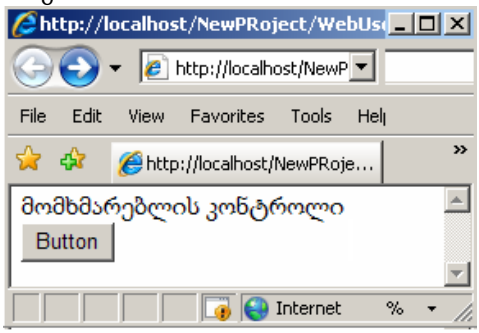
```
 {
  protected System.Web.UI.WebControls.Label lblMessage;

  public string Text
  {
   set{lblMessage.Text = value;}
  }

  private void Page_Load(object sender, System.EventArgs
e) { }

  override protected void OnInit(EventArgs e)
  {
       this.Load += new
System.EventHandler(this.Page_Load);
       base.OnInit(e);
  }
 }
}
```

ამ კონტროლის გამოყენების მაგალითი aspx ვებ-გვერდზე:
ფაილი WebUserControlPage.aspx:

```
<%@ Page language="c#" %>
<%@ Register TagPrefix="control"
TagName="WebUserControl1" Src="WebUserControl1.ascx" %>
<HTML>
 <script language="C#" runat="server">

      void btnSubmit_Click(Object sender, EventArgs E) {
          MyWebUserControl.Text = "მომხმარებლის კონტროლი
შეიცვალა!";
       }

  </script>
 <HEAD>
 </HEAD>
 <body >
  <form id="Form1" runat="server">
   <control:WebUserControl1 id="MyWebUserControl"
     Text="მომხმარებლის კონტროლი"
     runat="server"></control:WebUserControl1>
   <br>
   <asp:Button id="btnSubmit"  runat="server"
OnClick="btnSubmit_Click"
    Text="Button"></asp:Button>
```

```
  </form>
 </body>
</HTML>
```

გვერდის ჩატვირთვისას ეკრანზე ჩანს (ნახ.3.13) კონტროლის საწყისი მდგომარეობა:



ნახ.3.47

კომპოზიტური კონტროლის მაგალითი: `CreateChildControls` მეთოდში ხდება კონტროლების გენერაცია.

```
<%@ Register TagPrefix="CompositionSampleControls"
Namespace="CompositionSampleControls"
Assembly="CompositionSampleControls" %>
<html>
 <script language="C#" runat="server">
  private void AddBtn_Click(Object sender, EventArgs e)
  {
      MyControl.Value++;
  }
private void SubtractBtn_Click(Object sender, EventArgs
e) {
          MyControl.Value--;
      }
 </script>
 <body>
  <form method="POST" action="Composition1.aspx"
runat="server" ID="Form1">
   <CompositionSampleControls:Composition1
id="MyControl" runat="server" />
   <br>
   <asp:button text="Add" OnClick="AddBtn_Click"
runat="server" ID="Button1" NAME="Button1" />
   |
```

```
   <asp:button text="Subtract"
OnClick="SubtractBtn_Click" runat="server" ID="Button2"
NAME="Button2" />
  </form>
 </body>
</html>

using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CompositionSampleControls
{
 public class Composition1 : Control, INamingContainer
 {
  public int Value
  {
   get
   {
    this.EnsureChildControls();
    return Int32.Parse(((TextBox)Controls[1]).Text); }
   set
   {
    this.EnsureChildControls();
    ((TextBox)Controls[1]).Text = value.ToString(); }
  }

  protected override void CreateChildControls()
  {
   this.Controls.Add(new LiteralControl("<h3>" + "Value:
"));
   TextBox box = new TextBox();
   box.Text = "0";
   this.Controls.Add(box);
   this.Controls.Add(new LiteralControl("</h3>"));
  }
 }
}
```

3.25. ASP.NET პაკეტში მდგომარეობათა ტიპები. დანართებისა და სეანსების მდგომარეობები

HTTP პროტოკოლი არ იძლევა ვებ გვერდების მდგომარებობის შენახვის საშუალებას, რადგან გვერდების ყოველ ახალ მოთხოვნაზე სრულდება მონაცემების ახლიდან დამუშავება და გადაგზავნა, ამის შემდეგ სერვერზე არსებული ყველა მონაცემი იკარგება. მაგრამ არსებობს საჭიროება, რომ ხდებოდეს მდგომარეობის შენახვა გვერდების გამომახებებს შორის.

ASP.NET-ის ვებ-აპლიკაციებში არსებობს მდგომარეობების შენახვის სხვადასხვა ხერხი.

- **აპლიკაციის მდგომარეობა**

განვიხილოთ აპლიკაციის მდგომარეობა (Application State). აპლიკაციის მდგომარეობაში ინახება გლობალური პარამეტერების მნიშვნელობები. ძირითადად აპლიკაციის პარამეტერებისთვის მნიშვნელობების მინიჭება ხდება Application_OnStart მეთოდში Global.asax ფაილში, ხოლო მათი შემდგომი ცვლილება ხდება ცალკეულ ASP.NET გვერდებზე. აპლიკაციის ცვლადების სიცოცხლის ხანგრძლოივობა ტოლია აპლიკაციის სიცოცხლის ხანგრძლივობისა.

// ფაილი Global.asax:

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.IO" %>
<script language="C#" runat="server">
    void Application_Start(Object sender, EventArgs e)
     {
        DataSet ds = new DataSet();
        FileStream fs = new
           FileStream(Server.MapPath("schemadata.xml"),
                   FileMode.Open,FileAccess.Read);
        StreamReader reader = new StreamReader(fs);
        ds.ReadXml(reader);
        fs.Close();
        DataView view = new DataView(ds.Tables[0]);
        Application["Source"] = view;
     }
</script>
```

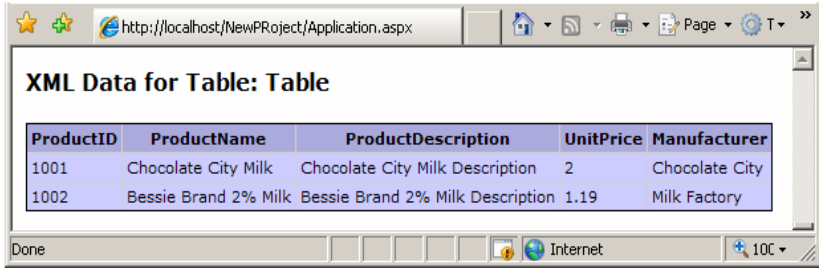
//ფაილი Application.aspx:

```
<%@ Import Namespace="System.Data" %>
<html>
  <script language="C#" runat="server">
    void Page_Load(Object Src, EventArgs E )
      {
       DataView Source =
(DataView)(Application["Source"]);
       MySpan.Controls.Add(new

LiteralControl(Source.Table.TableName));
       MyGridView.DataSource = Source;
       MyGridView.DataBind();
      }
  </script>
<body>
<form runat="server">
  <h3><font face="Verdana">Reading Data in
                        Application_OnStart</font></h3>
  <h4><font face="Verdana">XML Data for Table:
<asp:PlaceHolder runat="server"
id="MySpan"/></font></h4>
<asp:GridView id="MyGridView" runat="server"
    Width="900"
    BackColor="#ccccff"
    BorderColor="black"
    ShowFooter="false"
    CellPadding=3
    CellSpacing="0"
    Font-Name="Verdana"
    Font-Size="8pt"
    HeaderStyle-BackColor="#aaaadd"
    EnableViewState="false"
  />

</form>
</body>
</html>
```

//ფაილი SchemaData.xml:

```
<NewDataSet>
  <Table>
    <ProductID>1001</ProductID>
    <CategoryID>1</CategoryID>
    <ProductName>Chocolate City Milk</ProductName>
    <ProductDescription>Chocolate City Milk
                        Description</ProductDescription>
    <UnitPrice>2</UnitPrice>
```

```
    <ImagePath>images/milk5.gif</ImagePath>
    <Manufacturer>Chocolate City</Manufacturer>
  </Table>
  <Table>
    <ProductID>1002</ProductID>
    <CategoryID>1</CategoryID>
    <ProductName>Bessie Brand 2% Milk</ProductName>
    <ProductDescription>Bessie Brand 2% Milk
                        Description</ProductDescription>
    <UnitPrice>1.19</UnitPrice>
    <ImagePath>images/milk1.gif</ImagePath>
    <Manufacturer>Milk Factory</Manufacturer>
  </Table>
</NewDataSet>
```



ნახ.3.48

- **სესიის მდგომარეობა:**

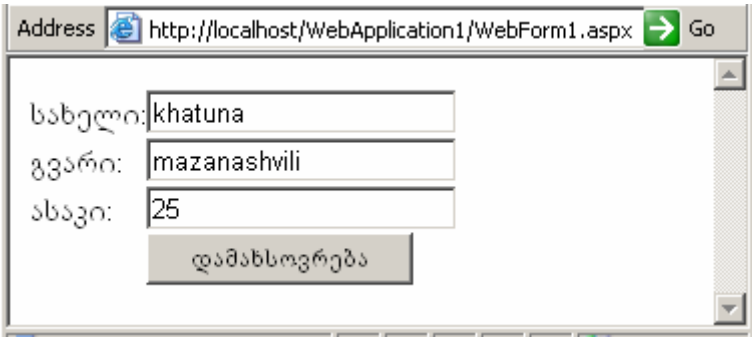
მომხმარებლის სესიის განმავლობაში მნიშვნელობების შენახვისათვის გამოიყენება სესიის ცვლადებში. ეს ცვლადები არის ყველა მომხმარებლისთვის უნიკალური. აღსანიშნავია რომ სესიის ცვლადების მნიშვნელობები იკარგება სესიის დასრულებისთანავე.

სესიის ცვლადებისთვი მნიშვნელობების მინიჭება აგრეთვე შესაძლებელია სესიის ინიციალიზების დროს Global.asax ფაილში.

```
protected void Session_Start(Object sender, EventArgs e)
{
       Session["FirstName"] = "khatuna";
       Session["LastName"] = "mazanashvili";
       Session["Age"] = 25;
}
```

ASP.NET გვერდზე შესაძლებელია სესიის ცვლადების გამოტანა `Page_Load` მეთოდში:

```
private void Page_Load(object sender, System.EventArgs
e)
{
       if(!IsPostBack)
       {
              txtFirstname.Text =
(string)Session["FirstName"];
              txtLastname.Text =
(string)Session["LastName"];
              txtAge.Text = Session["Age"].ToString();
       }
}
```



ნახ.3.49

შესაძლებელია სესიის მდგომარეობაში დამახსოვრება:
ფრაგმენტი WebForm1.aspx.cs ფაილიდან:

```
private void btnSave_Click(object sender,
System.EventArgs e)
  {    Session["FirstName"] = txtFirstname.Text;
       Session["LastName"] = txtLastname.Text;
       Session["Age"] = Int32.Parse(txtAge.Text);   }
```

// ფაილი WebForm1.aspx:

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false"
Inherits="WebApplication1.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
 <HEAD>
  <title>WebForm1</title>
 </HEAD>
```

```
 <body>
  <form id="Form1" method="post" runat="server">
   <TABLE cellSpacing="0" cellPadding="0" border="0">
    <TR>
     <TD>სახელი:</TD>
     <TD>
      <asp:TextBox id="txtFirstname"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD>გვარი:</TD>
     <TD>
      <asp:TextBox id="txtLastname"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD>ასაკი:</TD>
     <TD>
      <asp:TextBox id="txtAge"
runat="server"></asp:TextBox></TD>
    </TR>
    <TR>
     <TD></TD>
     <TD>
      <asp:Button id="btnSave" runat="server"
Text="დამახსოვრება"></asp:Button></TD>
    </TR>
   </TABLE>
  </form>
 </body>
</HTML>
```

- **Cookies გამოყენება**

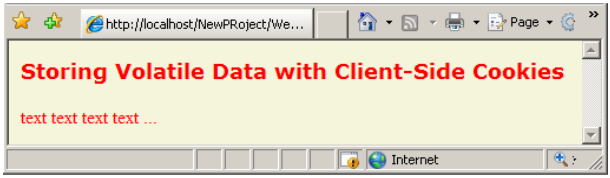
Cookies არის ფაილი რომელიც იქმნება კლიენტის კომპიუტერზე და გადაეცემა ვებ სერვერს სხვადსხვა გვერდების გამოძახებისას. მასში შესაძლებელია სხვადსხვა ინფორმაციის შენახვა, თუმცა მისი მაქსიმალური ზომა შეზღუდულია 4096 კილობაიტამდე.

შემდეგ მაგალითში გვერდის ჩატვირთვისას მოწმდება გადმოეცა თუ არა კლიენტისგან ქუქი სახელით preferences1. თუ არ არის გადმოცემულიმაშინ იქმნება ახალი ქუქი და გადაეგზავნება კლიენტის კომპიუტერს. იგივე გვერდზე გამოიყენება GetStyle

ფუნქცია, რომელიც აბრუნებს ქუქიში დამახსოვრებულ ცალკეულ მნიშვნელობებს. ეს ფუნქცია გამოიყენება ვებ გვერდზე ფონტის სახელის, ფონის ფერის და ტექსტის ფერის დასაყენებლად.

```
<html>
 <script language="C#" runat="server">
   void Page_Load(Object sender, EventArgs E)
    { if (Request.Cookies["preferences1"] == null)
       { HttpCookie cookie = new
HttpCookie("preferences1");
         cookie.Values.Add("ForeColor","red");
         cookie.Values.Add("BackColor","beige");
         cookie.Values.Add("FontName","Verdana");
         Response.Cookies.Add(cookie);
        }
      }
    protected String GetStyle(String key)
     {
       HttpCookie cookie =
Request.Cookies["preferences1"];
       if (cookie != null)
        { switch (key)
           { case "ForeColor" :
                            return
cookie.Values["ForeColor"];
                            break;
            case "BackColor" :
                            return
cookie.Values["BackColor"];
                            break;
             case "FontName" :
                            return
cookie.Values["FontName"];
                            break;
          }
        }
        return "";
      }
 </script>
 <style>
    body {
          font: <%=GetStyle("FontName")%>;
          background-color: <%=GetStyle("BackColor")%>;
         }
    </style>
 <body style="color:<%=GetStyle("ForeColor")%>">
```

```
  <h3><font face="Verdana">Storing Volatile Data with
Client-
                            Side Cookies</font></h3>
   text text text text ...<br>
 </body>
</html>
```



ნახ.3.50

- **ViewState-ის გამოყენება**

ASP.NET-ში შესაძლებელია რომ თითოელმა კონტროლმა შეინახოს მისი შიგა მდგომარეობა გვერდების გამოძახებებს შორის ViewState-ის გამოყენებით. ViewState-ის საშუალებით ხდება ვებ-გვერდის გამოძახებებს შორის მონაცემების დამახსოვრება.

შემდეგ მაგალითში, როდესაც მომხარებელი დააჭერს ღილაკს, ვებგვერდის ViewState-ში შეინახება მნიშვნელობა ”Hello!”.

```
<html>
 <head>
  <script runat="server" language="C#">

 void Button1_Click(object sender, System.EventArgs e)
 {  ViewState["Text"] = "Hello!";  }
 void Page_Load(object sender, System.EventArgs e)
 { if(ViewState["Text"]!=null)
     lblText.Text = ViewState["Text"].ToString();  }

  </script>
 </head>
 <body>
  <form id="Form1" method="post" runat="server">
   <asp:Label Runat="server" ID="lblText"></asp:Label>
   <asp:Button id="Button1" runat="server"
OnClick="Button1_Click" Text="Button"></asp:Button>
  </form>
 </body>
</html>
```

3.26. Web-ის უსაფრთხოება ASP.NET-ში. სერვერის, კლიენტების, ფორმების და როლების აუთენტიფიკაცია. პაროლების შენახვა

ვებ აპლიკაციებში ხშირად საჭიროა მომხმარებელთა იდენტიფიკაცია და მათთვის სხვადასხვა რესურსებზე წვდომის უფლების განსაზღვრა. ASP.NET-ში არსებობს აუთენტიფიკაციის რამდენიმე მეთოდი: Windows, Forms, Passport. ეს მოთოდები განისაზღვრება კონფიგურაციის ფაილში `authentication` ტეგის `mode` ატრიბუტის საშუალებით:

```
<configuration>
       <system.web>
              <authentication/>
       </system.web>
</configuration>
```

მეთოდები:

```
<authentication mode="Windows" />
<authentication mode="Forms" />
<authentication mode="Passport" />
<authentication mode="None" />
```

ვებ–აპლიკაციაზე მომხმარებლის წვდომის უფლებას განსაზღვრავს `authentication` ელემენტი, ხოლო აპლიკაციის გარკვეულ ნაწილებზე განისაზღვრება `authorization` ელემეტით: deny და allow ქველემენტების საშუალებით:

*** – განსაზღვრავს ყველა მომხმარებელს,**

? – ანონიმურ მომხმარებლებს.

მაგალითად, ანონიმურ მომხმარებელთათვის საიტზე წვდომის უფლების გასათიშად ვებ-საიტის კონფიგურაციის ფაილში ჩაიწერება:

```
<configuration>
 <system.web>
  <authorization>
   <deny users="?" />
  </authorization>
 </system.web>
</configuration>
```

შესაძებელია ცალკეულ მომხმარებელზე და მომხმარებელთა ჯგუფებზე წვდომის უფლების მინიჭება. შემდეგი ჩანაწერი

ნიშნავს, რომ წვდომის უფლება აქვთ someone და Admins ჯგუფში შემავალ მომხმარებლებს:

```
<authorization>
    <allow users="someone" />
    <allow roles="Admins" />
    <deny users="*" />
</authorization>
```

შესაძლებელია აგრეთვე ცალკეულ ვებ-გვერდებზე წვდომის უფლებების დაყენებაც:

```
<location path="webpage.aspx">
 <authorization>
  <allow roles="managers" />
  <deny users="?" />
 </authorization>
</location>
```

წინა ფრაგმენტი გვიჩვენებს, რომ webpage.aspx წვდომის უფლება აქვთ მხოლოდ managers ჯგუფის მომხმარებლებს.

Forms მეთოდით მომხმარებელთა ავტორიზაციის დროს საჭიროა მომხმარებლის სარეგისტრაციო გვერდის მითითება:

```
<configuration>
 <system.web>
  <authentication mode="Forms">
   <forms name=".ASPXCOOKIE" loginUrl="login.aspx"
protection="All"
            timeout="30" path="/"></forms>
  </authentication>
 </system.web>
</configuration>
```

ვებ საიტზე მომხმარებლების ავტორიზაციის მაგალითი:

// ფაილი Web.config:

```
<configuration>
  <system.web>
      <authentication mode="Forms">
        <forms name=".ASPXUSERDEMO"
loginUrl="login.aspx" protection="All" timeout="60" />
      </authentication>
      <authorization>
        <deny users="?" />
      </authorization>
    <globalization requestEncoding="UTF-8"
responseEncoding="UTF-8" />
  </system.web>
</configuration>
```

// ფაილი Default.aspx:

```
<%@ Import Namespace="System.Web.Security " %>
<html>
 <script language="C#" runat="server">

   void Page_Load(Object Src, EventArgs E )
     { Welcome.Text = "Hello, " + User.Identity.Name;
     }
    void Signout_Click(Object sender, EventArgs E)
      { FormsAuthentication.SignOut();
        Response.Redirect("login.aspx");
      }

 </script>
 <body>
  <h3><font face="Verdana">Using Cookie
Authentication</font></h3>
  <form runat="server" ID="Form1">
   <h3><asp:label id="Welcome" runat="server" /></h3>
   <asp:button text="Signout" OnClick="Signout_Click"
runat="server" ID="Button1" NAME="Button1" />
  </form>
 </body>
</html>
```

// ფაილი Login.aspx:

```
<%@ Import Namespace="System.Web.Security " %>
<html>
 <script language="C#" runat="server">

    void Login_Click(Object sender, EventArgs E) {
//authenticate user: this samples accepts only one user
with a
//name of someone@www.contoso.com and a password of
'password'
if((UserEmail.Value=="someone")&&(UserPass.Value=="passwo
rd"))
  {

FormsAuthentication.RedirectFromLoginPage(UserEmail.Valu
e,
                            PersistCookie.Checked);
  }
else
  { Msg.Text = "Invalid Credentials: Please try again";
}
 }
 </script>
```

```
 <body>
    <form runat="server" ID="Form1">
   <h3><font face="Verdana">Login Page</font></h3>
   <table>
    <tr>
      <td>Email:</td>
      <td><input id="UserEmail" type="text"
runat="server"
             NAME="UserEmail" /></td>
      <td
      </td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><input id="UserPass" type="password"
runat="server"
             NAME="UserPass" /></td>
      <td>
      </td>
    </tr>
    <tr>
     <td>Persistent Cookie:</td>
     <td><ASP:CheckBox id="PersistCookie" runat="server"
/>
     </td>
     <td></td>
    </tr>
   </table>
   <asp:button text="Login" OnClick="Login_Click"
           runat="server" ID="Button1" NAME="Button1" />
   <asp:Label id="Msg" ForeColor="red" Font-
Name="Verdana"
           Font-Size="10" runat="server" />
  </form>
 </body>
</html>
```

ნახ.3.51

IV თავი

ADO.NET-ის საშუალებით მონაცემებთან მიმართვა

მომხმარებელთა პროგრამულ აპლიკაციებს (დანართებს) აუცილებლად ესაჭიროებათ ურთიერთქმედება მონაცემთა ცენტრალიზებულ ბაზებთან, მონაცემთა საცავებთან XML–ფორმატით, ან მონაცემთა ლოკალურ ბაზებთან, როდესაც ისინი მუშაობენ კლიენტის მანქანებზე.

ADO.NET-ტექნოლოგია გვთავაზობს მონაცემებთან მიმართვისათვის გამოსაყენებლად მარტივ, მაგრამ მეტად მძლავრ საშუალებებს, რომლებიც უზრუნველყოფს სისტემის რესურსების მაქსიმალურად სრულ ურთიერთქმედებას.

აქ გავეცნობით:

- ADO.NET-ის მონაცემებთან მიმართვის ძირითადი კომპონენტებს;
- თითოეული კომპონენტის როლს;
- ADO.NET-ის მონაცემებთან მიმართვის ორგანიზაციის აღწერას.

სხვადასხვა დანართები მონაცემებთან მიმართვის ორგანიზაციისათვის აყენებს სხვადასხვა მოთხოვნებს. მნიშვნელობა არა აქვს იმას, თუ რას აკეთებს დანართი: ასახავს ცხრილების შინაარსს, თუ გადაამუშავებს და განაახლებს მონაცემებს ცენტრალურ SQL-სერვერზე. ADO.NET აძლევს მომხმარებელს მონაცემებთან მიმართვის მარტივ და ეფექტურ საშუალებებს რეალიზაციის სხვადასხვა სცენარით.

4.1. გამოყოფილ მონაცემებთან მიმართვა

მონაცემებთან მიმართვის ტრადიციული ტექნოლოგიები ჩვეულებრივად ახორციელებდა მონაცემების წვდომას წყაროსთან მუდმივი მიერთების გზით. ასეთი მოდელის გამოყენებისას აპლიკაცია გახსნის მონაცემთა ბაზასთან მიერთებას და არ დახურავს მას მუშაობის დამთავრებამდე. დანართის სირთულის

ზრდასთან ერთად იზრდება მონაცემთა ბაზის კლიენტების რაოდენობაც, რაც არაეფექტურს ხდის ბაზასთან მუდმივი მიერთების ტექნოლოგიას, კერძოდ:

- სისტემური რესურსები გამოიყენება არაეფექტურად. რაც უფრო მეტი მუდმივად მიერთებული (გახსნილი) ფაილია, მით უფრო დაბალია სისტემის მწარმოებლურობა;

- დანართები, რომლებიც იყენებს მონაცემებთან მიმართვას მუდმივი მიერთების საშუალებით, ცუდად მასშტაბირებადია. ასეთი დანართები კარგად ემსახურება ორ მიერთებულ კლიენტს, 10-თან უკვე უჭირს მუშაობა და 100-თან საერთოდ ვერ ფუნქციონირებს.

ADO.NET სისტემაში ეს პრობლემები წყდება მონაცემებთან მიმართვის ისეთი მოდელის გამოყენებით, როგორიცაა გამოყოფილი მონაცემები. ასეთი მოდელის შემთხვევაში მონაცემთა წყაროსთან მიერთება გახსნილია მხოლოდ გარკვეული პროცედურების შესასრულებლად. მაგალითად, თუ აპლიკაციას დასჭირდა მონაცემები ბაზიდან, იგი მიუერთდება მას ამ მონაცემების გადმოტვირთვამდე, შემდეგ კი მიერთება დაიხურება. ასევე, როდესაც ხორციელდება მონაცემთა განახლება ბაზაში, მიერთება წყაროსთან განხორციელდება UPDATE-ბრძანების შესრულების დამთავრებამდე, შემდეგ იგი დაიხურება. ამგვარად, მონაცემებთან მიერთების დროის (გახსნა-დახურვის პერიოდი) შემცირებით, ADO.NET უზრუნველყოფს სისტემური რესურსების ეკონომიურ გამოყენებას და მონაცემთა წვდომის ინფრასტრუქტურის მასშტაბირებას, მწარმოებლურობის შემცირების გარეშე.

4.2. ADO.NET მონაცემთა არქიტექტურა

მონაცემებთან მიმართვა ADO.NET-ში ხორციელდება ორი კომპონენტით:

- მონაცემთა ერთობლიობით (ობიექტით DataSet), რომელშიც მონაცემები ინახება ლოკალურ კომპიუტერში;

- მონაცემთა მიმწოდებლით (პროვაიდერით, Data Provider), რომელიც ასრულებს შუამავლის ფუნქციას პროგრამასა და მონაცემთა ბაზას შორის.

ობიექტი DataSet: ესაა მონაცემთა წარმოდგენა კომპიუტერის მეხსიერებაში მონაცემთა წყაროსგან იზოლირებულად. ეს ობიექტი შეიძლება განვიხილოთ, როგორც მონაცემთა ბაზის ფრაგმენტის ლოკალური ასლი (კოპიო).

DataSet-ში მონაცემთა ჩატვირთვა შესაძლებელია ნებისმიერი დასაშვები წყაროდან, მაგალითად, SQL Server, Ms Access ბაზებიდან ან XML–ფაილიდან. დასაშვებია მეხსიერებაში ამ მონაცემებით მანიპულირება, აგრეთვე მათი განახლება მთავარი წყაროსაგან დამოუკიდებლად.

ობიექტი DataSet შედგება DataTable ობიექტთა ერთობლიობისგან (ის შეიძლება ცარიელიც იყოს, ანუ არ შეიცავდეს არც ერთ DataTable-ს).

ყოველი DataTable ობიექტი კომპიუტერის მეხსიერებაში ასახავს ერთ ცხრილს. მისი სტრუქტურა შეიცავს ორ ერთობლიობას: DataColumns, რომელშიც თავსდება ცხრილის სვეტები, და ცხრილის შეზღუდვათა ერთობლიობა. ეს ორი ერთობლიობა ქმნის ცხრილის სქემას.

DataTable ობიექტი შეიცავს აგრეთვე DataRows ერთობლიობას, რომელშიც ინახება DataSet-ობიექტის მონაცემები.

გარდა ამისა, DataSet ობიექტი შეიცავს DataRelations ერთობლიობას, რომელიც უზრუნველყოფს კავშირების შექმნას სხვადასხვა ცხრილის სტრიქონებს შორის. DataRelations შეიცავს DataRelation - ობიექტთა ერთობლიობას, რომლებიც განსაზღვრავს ცხრილთაშორის კავშირებს (მაგალითად, 1:M კავშირის სარეალიზაციოდ).

და ბოლოს, DataSet ობიექტი შეიცავს ExtendedProperties ერთობლიობას, რომელშიც შეინახება დამატებითი მონაცემები.

მონაცემთა პროვაიდერი: ესაა ურთიერთდაკავშირებულ კომპონენტთა ერთობლიობა, რომელიც უზრუნველყოფს ეფექტურ მაღალმწარმოებლურ კავშირს მონაცემთა ბაზასთან.

.NET Framework-ს აქვს ორი პროვაიდერი: SQL Server .NET Data Provider, რომელიც შექმნილია SQL Server 7.0 ან უფრო მაღალ ვერსიებთან სამუშაოდ, და OleDb .NET Data Provider - სხვა ტიპის მონაცემთა ბაზებთან დასაკავშირებლად.

მონაცემთა ნებისმიერი პროვაიდერი შედგება მსგავსი უნივერსალური კლასების კომპონენტებისგან:

- Connection, რომელიც უზრუნველყოფს მონაცემტა ბაზასთან მიერთებას;
- Command, რომელიც გამოიყენება მონაცემთა წყაროს სამართავად. იგი გამოიყენებს ბრძანებებს, რომლებიც არ აბრუნებს მონაცემებს, მაგალითად, INSERT, UPDATE და DELETE, ან ბრძანებებს, რომლებიც აბრუნებს DataReader ობიექტს (მაგალითად, SELECT) ;
- DataReader გამოიყენება მხოლოდ ჩანაწერთა ერთობლიობის წასაკითხად მიერთებული მონაცემთა წყაროდან ;
- DataAdapter შეავსებს გამოყოფილ ობიექტს DataSet ან DataTable და განაახლებს მათ შედგენილობას.

მონაცემებთან მიმართვა ხორციელდება შემდეგნაირად: ობიექტი Connection აყენებს დანართის (აპლიკაციის) მონაცემთა ბაზასთან მიერთებას, რომელიც პირდაპირ მისაწვდომია Command და DataAdapter ობიექტებისთვის. Command ობიექტი უზრუნველყოფს ბრძანებათა შესრულებას უშუალოდ მონაცემთა ბაზაში. თუ შესასრულებელი ბრძანება აბრუნებს რამდენიმე მნიშვნელობას, მაშინ Command ხსნის მათთან მიმართვას DataReader ობიექტის საშუალებით. მიღებული შედეგები შესაძლებელია დამუშავდეს უშუალოდ დანართის კოდით, ან DataSet ობიექტით, რომელიც შეივსება DataAdapter ობიექტის დახმარებით. მონაცემთა ბაზის განახლებისთვის ასევე გამოიყენება Command და DataAdapter ობიექტები.

ობიექტი Connection გვთავაზობს მიერთებას მონაცემთა ბაზასთან. Visual Studio .NET –ს აქვს Connection-ის ორი კლასი: SqlConnection, შესაერთებლად SQL Server 7.0 ან უფრო მაღალ ვერსიებთან, და OleDbConnection - სხვა ტიპის მონაცემთა ბაზებთან დასაკავშირებლად. მონაცემთა ბაზასთან კავშირის არხის გასახნელი აუცილებელი მონაცემები ინახება Connection ობიექტის ConnectionString თვისებაში. ეს ობიექტი ინახავს აგრეთვე რიგ

მეთოდებს, რომლებიც საჭიროა მონაცემთა დასამუშავებლად ტრანზაქციების გამოყენებით.

ობიექტს Command აქვს ორი კოლასი: SqlCommand და OleDbCommand. იგი უზრუნველყოფს ბრძანებათა გამოყენებას მონაცემთა ბაზაზე, რომელთანაც დამყარებულია კავშირი (მიერთება). აქ შეიძლება გამოყენებულ იქნას შენახვადი პროცედურები, SQL-ის ბრძანებები, აგრეთვე ოპერატორები მთლიანი ცხრილების მისაღებად. ობიექტს Command აქვს სამი მეთოდი :

- Execute Non Query. იყენებს ბრძანებებს, რომლებიც არ აბრუნებს მონაცემებს, მაგალითად, INSERT, UPDATE და DELETE;
- Execute Scalar. იყენებს მოთხოვნებს მონაცემთა ბაზისადმი, რომლებიც აბრუნებს მხოლოდ ერთ მნიშვნელობას;
- Execute Reader. აბრუნებს საშედეგო ერთობლიობას Data Reader ობიექტის საშუალებით.

ობიექტი DataReader გვთავაზობს ნაკადს მონაცემთა ბაზის ჩანაწერების ერთობლიობით, ოღონდ მხოლოდ ერთი მიმართულებით წასაკითხად. მონაცემთა პროვაიდერის სხვა კომპონენტებისგან განსხვავებით DataReader-ის ეგზემპლარების შექმნა პირდაპირ არაა დასაშვები. მისი მიღება შეიძლება Command ობიექტის Execute Reader მეთოდებით: SqlCommand.ExecuteReader მეთოდი აბრუნებს SqlDataReader ობიექტს, ხოლო მეთოდი OleDbCommand.ExecuteReader კი - OleDbDataReader ობიექტს. თუ DataReader ობიექტის შემცველი მონაცემების ჩაწერა დისკზე არაა საჭირო, მაშინ ეს სტრიქონები შეიძლება პირდაპირ გადაეგზავნოს დანართს. ვინაიდან დროის ნებისმიერ მომენტში მეხსიერებაში იმყოფება მხოლოდ ერთი სტრიქონი, DataReader ობიექტის გამოყენება თითქმის არ ამცირებს სისტემის მწარმოებლურობას, ოღონდ მოითხოვს მონოპოლურ მიმართვას გახსნილ Connection-ობიექტზე DataReader ობიექტის სასიცოცხლო დროისგანმავლობაში.

ობიექტი DataAdapter არის ADO.NET–ის ძირითადი კლასი, რომელიც უზრუნველყოფს გამოყოფილ მონაცემებთან მიმართვას. არსებითად, იგი ასრულებს შუამავლის ფუნქციებს მონაცემთა ბაზისა და DataSet-ობიექტის ურთიერთქმედებისთვის.

Fill მეთოდის გამოძახებისას DataAdapter ობიექტი შეავსებს მონაცემებით DataTable-ს ან DataSet-ს მონაცემთა ბაზიდან. მონაცემების დამუშავების შემდეგ, რომლებიც ჩატვირთულია მეხსიერებაში, შესაძლებელია მოდიფიცირებული ჩანაწერების მოთავსება მონაცემთა ბაზაში, DataAdapter ობიექტის Update მეთოდის გამოძახებით. DataAdapter-ს აქვს ოთხი თვისება, რომლებიც წარმოადგენს მონაცემთა ბაზის ბრძანებებს:

- SelectCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს მონაცემთა ბაზიდან ამორჩევას (მაგალითად, მეთოდი Fill);
- InsertCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს სტრიქონის ჩასმას ცხრილში;
- DeleteCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს სტრიქონის წაშლას ცხრილიდან;
- UpdateCommand შეიცავს ტექსტს ან ბრძანების ობიექტს, რომელიც ახორციელებს მნიშვნელობათა განახლებას მონაცემთა ბაზაში;

Update მეთოდის გამოძახებისას ყველა შეცვლილი მონაცემი კოპირდება DataSet-ობიექტიდან მონაცემთა ბაზაში, შესაბამისი ბრძანებების InsertCommand, DeleteCommand ან UpdateCommand გამოყენებით.

4.3. ADO.NET პროგრამული პაკეტი მონაცემთა ბაზებთან სამუშაოდ

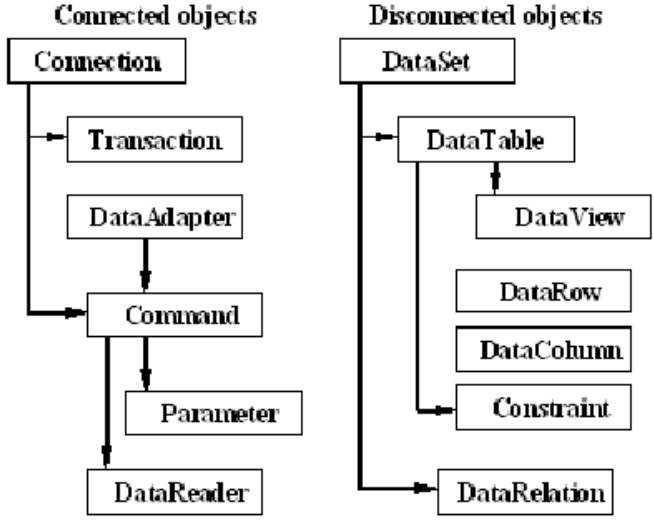
MS ADO.NET (Microsoft ActiveX Data Objects) არის ფირმა მაიკროსოფთის უახლესი ინტეგრირებული პროგრამული ტექნოლოგია (კლასების ბიბლიოთეკა Visual Studio.NET პაკეტში), რომელიც გამოიყენება ინტერნეტში სტრუქტურული მონაცემების გასაცვლელად კლიენტ-სერვერ არქიტექტურისათვის, მონაცემთა ბაზებთან (მაგალითად, Oracle, SQL Server და სხვ.) სამუშაოდ და მომხმარებელთა ინტერფეისების ასაგებად [5].

Visual Studio.NET პაკეტში ორიგინალურად გამოიყენება ობიექტ-ორიენტირებული დაპროგრამების ენები C#, Visual Basic, Java, Viasual C++, აგრეთვე XML ენა Web აპლიკაციების ასაგებად.

SQL Server და ADO.NET პროგრამული პაკეტები, როგორც მონაცემთა ბაზებისა და ბაზებთან მუშაობის მომხმარებელთა ინტერფეისების დამუშავების კლასიკური ინსტრუმენტული საშუალებანი, ფლობს აგრეთვე ჯგუფურ CASE-ტექნოლოგიებს და UML-ენას. მისი საშუალებით ხორციელდება ეფექტური მრავალდონიანი გამოყენებითი სისტემების დამუშავება მონაცემთა ბაზებთან სამუშაოდ ინტრაქსელსა და ინტერნეტში.

4.4. ADO.NET-ის ობიექტური მოდელის სტრუქტურა

სისტემის ობიექტური მოდელი ორი ნაწილისგან შედგება: მარცხენა - მიერთებადი ობიექტები (connected objects) და მარჯვენა - განცალკევებადი ობიექტები (disconnected obects). 4.1 ნახაზზე ნაჩვენებია ADO.NET ობიექტური მოდელის შემადგენელი კლასები, რომელთა დანიშნულებასაც მოკლედ შევეხებით ამ პარაგრაფში.



ნახ.4.1. ADO.NET–ის ობიექტების იერარქია

მიერთებადი ობიექტები უშუალოდ უკავშირდება მონაცემთა ბაზას, ახორციელებს მასში ტრანზაქციებს, მონაცემთა ძებნას, განახლებას და გადაცემას. განცალკევებადი ობიექტების დანიშნულებაა მონაცემთა ბაზებიდან მიღებული ობიექტების (ცხრილები, სტრიქონები, სვეტების და ა.შ.) ავტონომიურად დამუშავება.

ერთ-ერთი მთავარი ტერმინი „მონაცემთა მიმწოდებელია“ (DataProvider), რომლიდანაც სისტემა ღებულობს მონაცემებს. ამგვარად, .NET-ის მონაცემთა მიმწოდებელი არის კლასთა ერთობლიობა, რომელიც უზრუნველყოფს მონაცემთა საცავებთან (Repository) ურთიერთქმედებას. ამ სისტემაში მონაცემთა ორი მიმწოდებელია გათვალისწინებული: SQL Client .NET Data Provider, რომელიც SQL Server მონაცემთა ბაზასთან მუშაობს, და OLE DB .NET Data Provider, რომელიც ურთიერთქმედებს მონაცემთა სხვადასხვა ბაზებთან OLE DB მიმწოდებლის საშუალებით.

მონაცემთა ყველა მიმწოდებელი ერთიდაიმავე საბაზო კლასებს იყენებს: Connection, Command, DataProvider, Parameter, Transaction, ოღონდაც თავიანთ კონტექსტში. მაგალითად, თუ ის SQL Server-ია, მაშინ გვექნება SqlConnection, თუ OLE DB , მაშინ OleDbConnection.

- ობიექტი Connection გამოიყნება მონაცემთა ბაზასთან ფიზიკური კავშირის დასამყარებლად (ან ამ კავშირის გასაწყვეტად). ობიექტის თვისებებში მიეთითება მონაცემთა წყაროს ტიპი, მისი ადგილმდებარეობა და სხვა პარამეტრები.

- Command ობიექტები გამოიყნება მონაცემთა ბაზასთან SQL-მოთხოვნების მისაწოდებლად, შენახვადი პროცედურების (Stored Procedures) გამოსაძახებლად, ცხრილებისა (Tables) და წარმოდგენების (Views) მნიშვნელობების მისაღებად, შესაცვლელად და დასაბრუნებლად. მაგალითად, SqlCommand

ობიექტს აქვს ExecuteXmlReader მეთოდი, რომელიც მოთხოვნების შედეგებს აბრუნებს XML-ფორმატში.

- ობიექტი DataReader გამოიყენება მოთხოვნების საფუძველზე მონაცემთა ბაზიდან ჩანაწერების სწრაფად ამოსარჩევად და დასათვალიერებლად. მისი საშუალებით ჩანაწერებში მონაცემთა ცვლილება არ ხდება.

- ობიექტი Transaction გამოიყნება მონაცემთა ბაზაში ერთდროულად რამდენიმე მიმდევრობითი ცვლილების განსახორციელებლად. ეს რამდენიმე ცვლილება ჯგუფდება როგორც ერთი მთლიანი პროცედურა, რომელსაც ტრანზაქციას უწოდებენ. ობიექტს Connection აქვს მეთოდი BeginTransaction, რომელიც ქმნის Transaction ობიექტს. მისი დანიშნულებაა ჯგუფში შემავალი ცვლილებების განხორციელების დამოწმება ან უარყოფა. თუ, მაგალითად, ჯგუფში არ შესრულდა ყველა დაგეგმილი ცვლილება, მაშინ ტრანზაქცია მონაცემთა ბაზაში არ მოახდენს ნაწილობრივ შეცვლილი ჩანაწერების დაფიქსირებას.

- ობიექტი Parameter გამოიყენება SQL-მოთხოვნის ფორმირების პროცესში Where-კონსტრუქციაში „ ? “ - მარკერის ჩასმით (მაგალითად, Where Client_Id = ?). ე.ი. შესაძლებელი იქნება ყველა კლიენტისთვის ამ პარამეტრის გამოყენება.

- ობიექტი DataAdapter არის „ხიდი“ მონაცემთა ბაზასა და ADO.NET მოდელის განცალკევებულ ობიექტებს შორის. მაგალითად, მეთოდი DataAdapterFill ახორციელებს მონაცემთა ეფექტურ ამორჩევას ბაზაში შესაბამისი მოთხოვნის საფუძველზე, შემდეგ კი ეს მონაცემები მუშავდება ავტონომიურად DataSet ან DataTable ობიექტებით. ბოლოს, შეცვლილი და გადამუშავებული მონაცემები DataSet-ის ობიექტებიდან DataAdapter-ის საშუალებით ჩაიწერება მონაცემთა ბაზაში.

DataAdapter-ს აქვს რამდენიმე თვისება, რომლებიც Command ობიექტებია. მაგალითად, SelectCommand, არის მოთხოვნა DataSet ობიექტის შესავსებად. აგრეთვე InsertCommand, Update-Command და DeleteCommand, რომლებიც შეესაბამება მონაცემთა

ბაზაში ახალ, შეცვლილ ან წასაშლელ ჩანაწერთა Command-ობიექტებს.

ცხრილების შევსებისა და მონაცემთა ცვლილებების დაფიქსირების პროცესებისათვის ბაზაში DataAdapter-ს გააჩნია მონიტორინგის თვისებები. მაგალითად, TableMapping - მონაცემთა ბაზის ცხრილების შედარების მეთვალყურეობის თვისება, ColumnMapping - კი იგივე სვეტებისათვის.

როგორც აღვნიშნეთ, მონაცემთა მისაღებად ბაზიდან აუცილებელია ზემოაღწერილი მიერთებადი ობიექტების გამოყენება. მათ დასამუშავებლად (დალაგება, შეცვლა და ა.შ.) კი აუცილებელია გადავიდეთ ავტონომიურ რეჟიმში, ანუ გამოვიყენოთ ADO.NET-ის განცალკევებადი ობიექტები. შევეხოთ მათ მოკლედ.

- ობიექტი DataSet შეიცავს მონაცემთა ერთობლიობას. იგი ცხრილების კონტეინერია DataTable ობიექტებისათვის. მაგალითად, მრავალდონიან სისტემაში სერვერის მონაცემთა ბაზასთან ერთხელ მიმართვის დროს შესაძლებელია ამოვირჩიოთ ერთდროულად რამდენიმე ცხრილი (კონტეინერში) და გადმოვიტანოთ შედეგები. რამდენჯერმე მიმართვა აღარ იქნება საჭირო. DataSet მონაცემებს ამუშავებს განცალკევებულ რეჟიმში, DataRow ობიექტით კეშ-მეხსიერებაში. ამგვარად, თუ მოხდა მათი ნაწილის შეცვლა და საჭიროა ცხრილების უკან დაბრუნება სერვერში, მაშინ ყველა ცხრილის გადაგზავნა უკან არაეფექტურია და სისტემა გვთავაზობს მხოლოდ შეცვლილი ნაწილის დაბრუნებას, რომელიც ხორციელდება GetChanges მეთოდით. აქვე განიხილება Merge მეთოდი, რომელსაც შეუძლია ორი DataSet ობიექტის შემცველი მონაცემების გაერთიანება ერთ ობიექტად. ეს ძალზედ მნიშვნელოვანი მომენტია მრავალდონიან კლიენტ-სერვერ არქიტექტურიან სისტემებისთვის. DataSet ობიექტი ინახავს მონაცემებს XML-დოკუმენტის სახით და სწრაფადაც გარდაქმნის მას.

- ობიექტი DataTable მონაცემებს ამუშავებს სვეტებისა და ჩანაწერების (სტრიქონების) სახით. მეთოდი DataAdapterFill

გამოიყენება მოთხოვნის საფუძველზე მიღებული მონაცემების მოსათავსებლად DataTable ობიექტში.

- ობიექტი DataColumn არის ცხრილის სვეტი. მაგრამ მასში ინახება არა რეალური ცხრილის სვეტის მონაცემები, არამედ ამ სვეტის შესაბამისი სტრუქტურის ინფორმაცია (მეტამონაცემები). მისი თვისებებია Type, ReadOnly, AllowDBNull, Unique, Default, AutoIncrement. გამოიყენება აგრეთვე თვისება Expression, რომელიც გაანგარიშებადი სვეტებისთვისაა განკუთვნილი და უზრუნველყოფს ფორმულის ჩაწერას.

- ობიექტი Constraint არის DataTable-ობიექტის ფარგლებში ერთი ან რამდენიმე სვეტის შეზღუდვიების მონაცემები. მაგალითად, ველის განსაზღვრა უნიკალურობაზე.

- ობიექტი DataRow ცხრილის რეალურ მონაცემებთან სამუშაოდ გამოიყენება. არჩეული ცხრილის საჭირო ველის მნიშვნელობის მისაღებად გამოიყენება Item თვიება. მონაცემთა განახლების პროცედურებისათვის გამოიყენება მეთოდი DataRowBeginEdit, შემდეგ Item თვიებით მოხდება ცვლილება და ბოლოს, მეთოდით EndEdit ცვლილებები შეინახება. თუ არაა საჭირო ცვლილებების შენახვა, მაშინ გამოიყენება მეთოდი CancelEdit.

- ობიექტი DataRelation ახორციელებს ცხრილთაშორის კავშირებს. მისი საშუალებით შესაძლებელია კასკადური კავშირების აღწერა „დედა-შვილ“ ცხრილებს შორის და, საბოლოო ჯამში, მონაცემთა მთლიანობის უზრუნველყოფა. ყოველი ცვლილება „დედა“-ცხრილში ასახული იქნება ავტომატურად „შვილში“.

- ობიექტი DataView გამოიყენება მოთხოვნის საფუძველზე მიღებული რეალური ცხრილის მონაცემების წარმოსადგენად (დასათვალიერებლად) სხვადასხვა ფორმით. მაგალითად, ცხრილის სტრიქონების დალაგება რომელიმე ველის მოწესრიგებით (Sort-თვისება), ან ჩანაწერების ამორჩევა რაიმე კრიტერიუმით (Filter-თვისება). რაც მთავარია, ეს მონაცემები რეალურად მეხსიერებაში არ თავსდება შესანახად.

4.5. მონაცემებთან მიმართვა

Visual Studio .NET სისტემას აქვს სტანდარტული ოსტატი პროგრამებისა და დიზაინერების სიმრავლე, რომელთა საშუალებითაც ადვილად და ეფექტურად ხორციელდება მონაცემებთან წვდომის არქიტექტურა დანართების დამუშავების პროცესში. ამასთანავე ADO.NET ობიექტური მოდელის ყველა შესაძლებლობა მისაწვდომია პროგრამულად, რაც უზრუნველყოფს არასტანდარტული ფუნქციების რეალიზაციის ან დანართების აგების შესაძლებლობას, რომლებიც მომხმარებელთა მოთხოვნილებებზეა ორიენტირებული.

ამ ლექციაზე ჩვენ გავეცნობით, თუ როგორ დავუკავშირდეთ მონაცემთა ბაზას ADO.NET–ის გამოყენებით, როგორ ამოვიღოთ საჭირო მონაცემები და გადავცეთ ისინი დანართს. ეს საკითხები შეიძლება შესრულდეს Visual Studio .NET-ის გრაფიკული ინსტრუმენტებითაც და პროგრამულადაც.

4.6. მონაცემთა ბაზასთან მიერთება

აპლიკაციაში არსებობს მონაცემთა ბაზასთან მიერთების რამდენიმე ხერხი. ყველაზე მარტივია ამის განხორციელება Visual Studio .NET-ის გრაფიკული ინსტრუმენტით.

მონაცემთა წყაროსთან მიერთებისა და მისი მართვისათვის გამოიყენება ფანჯარა Server Explorer. იგი

4.7. ინტერფეისის დამუშავების სადემონსტრაციო მაგალითი

ძირითადი ამოცანა, რომელსაც ჩვენ აქ განვიხილავთ, არის ADO.NET პროგრამული პაკეტის გამოყენებით მომხმარებელთა სამუშაო ინტერფეისის დამუშავების სადემონსტრაციო მაგალითის აგება. ამასთანავე, მონაცემთა ბაზების სახით უნდა გამოვიყენოთ SQL Server პაკეტით დამუშავებული ცხრილები.

4.2 ნახაზზე ნაჩვენებია Start | Microsoft Visual Studio NET-ით სამუშაო სისტემაში შესვლის ფანჯარა.

პირველ რიგში შევამოწმოთ მონაცემთა ბაზასთან კავშირი. ამისათვის მენიუდან View | Server Explorer-ით გამოვიტანოთ ფანჯარა (ნახ.4.3) და ავირჩიოთ HOMESERVER.BusinProc.dbo. გამოჩნდება SQL Server-ის ბაზა, ცხრილები და ველები. DIAGRAM1-ის არჩევით მივიღებთ ცხრილებს კავშირებით, რომელიც იდენტურია SQL Server-ში ჩვენს მიერ შექმნილი ბაზისა. ამგვარად თავსებადობა კარგადაა რეალიზებული. 4.4 ნახაზზე ნაჩვენებია ცხრილთა კავშირების ფრაგმენტი, მხოლოდ მათი სათაურების ჩვენებით.

ახლა დავიწყოთ მომხმარებლის ინტერფეისის (ერთი სამუშაო ფორმის) დამუშავება. მენიუდან ავირჩიოთ

File | New | Project

კომბინაცია და მივიღებთ 4.5 ნახაზზე ნაჩვენებ ფანჯარას. აქ შევარჩევთ სახელს (Name), ვინჩესტერზე შესანახ გზას და პაპკის სახელს (Location). Template-ში ავირჩევთ Windows Application.

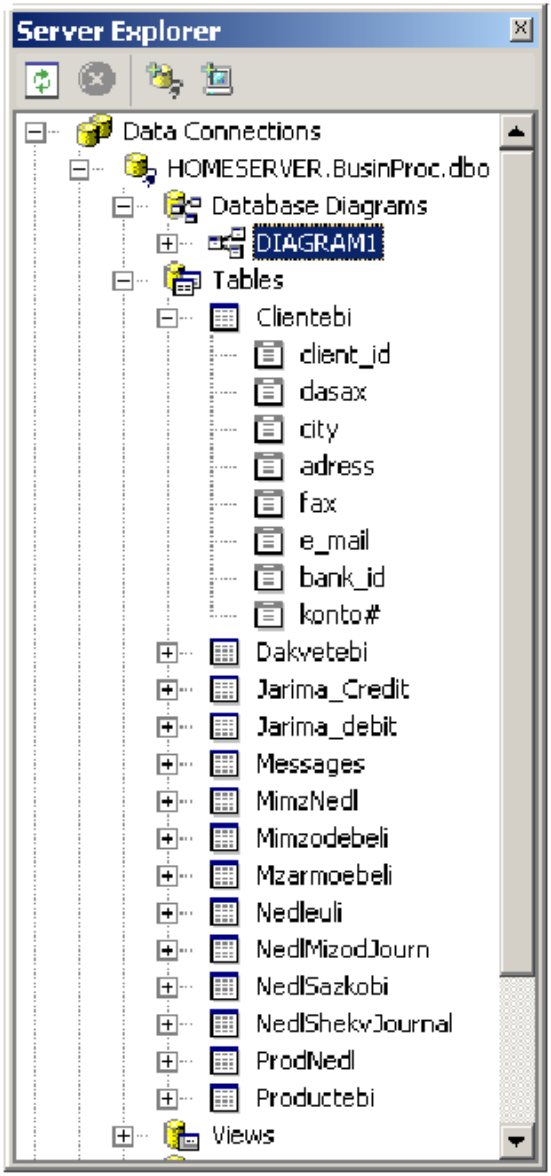
შემდეგ მენიუდან ვირჩევთ:

File | New Item

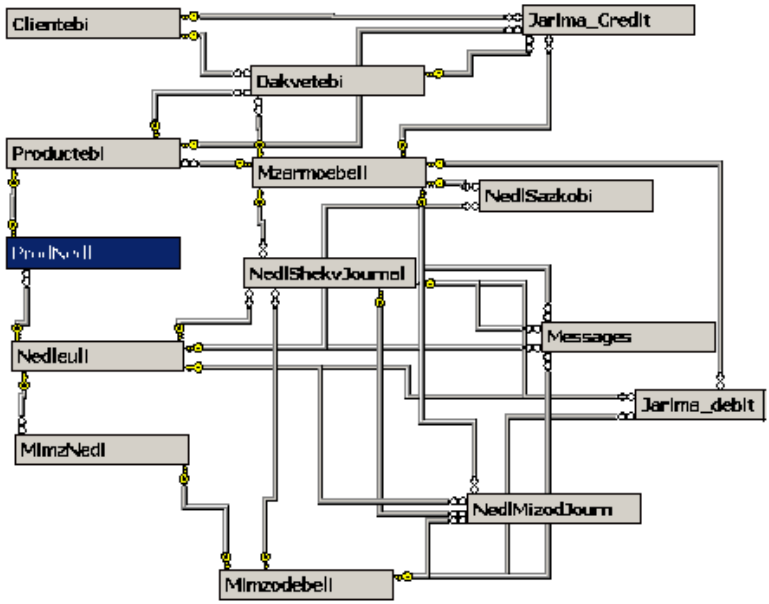
და მივიღებთ 4.6 ნახაზზე მოცემულ ფანჯარას.

ავირჩიოთ Data Form Wizard და ეკრანზე გამოჩნდება ახალი ფორმა (ნახ.4.7). აქ ღილაკით Next გადავალთ შემდეგ ბიჯზე (ნახ.4.8).

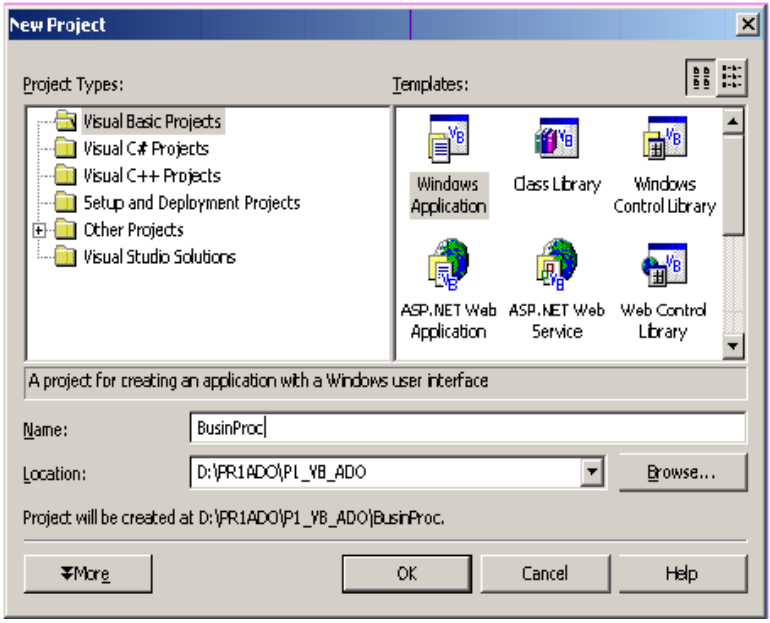
ნახ.4.2. ADO.NET სასტარტო ფანჯარა



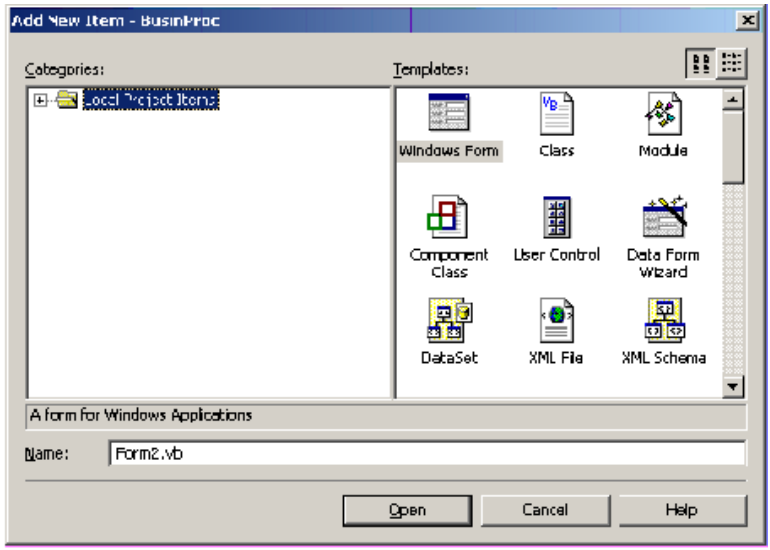
ნახ.4.3. მონაცემთა ბაზასთან დაკავშირება



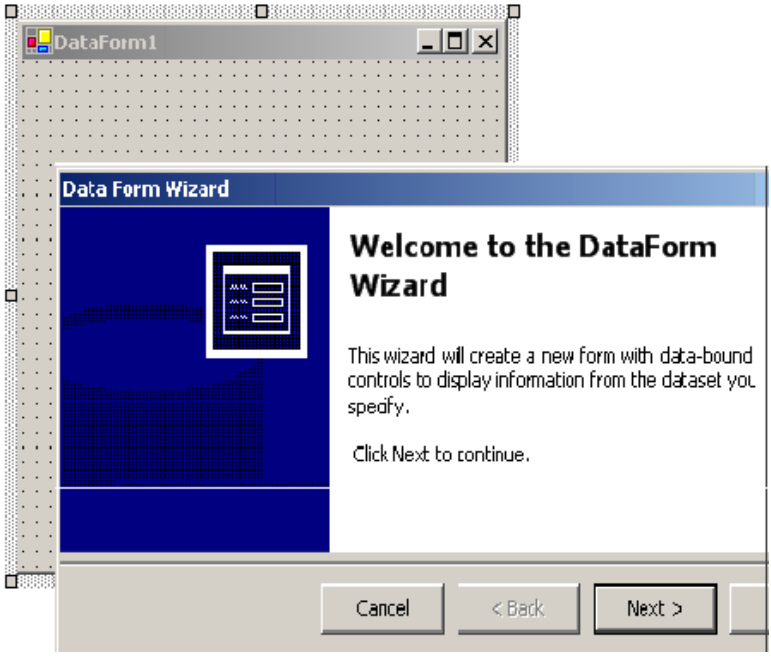
ნახ.4.4. SQL Server მონაცემთა ბაზის ასახვა ADO.NET-ში


New Project
Project Types:
Templates:
Visual Basic Projects
Visual C# Projects
Visual C++ Projects
Setup and Deployment Projects
Other Projects
Visual Studio Solutions
Windows Application
Class Library
Windows Control Library
ASP.NET Web Application
ASP.NET Web Service
Web Control Library
A project for creating an application with a Windows user interface
Name:
BusinProc
Location:
D:\PR1ADO\P1_VB_ADO
Browse...
Project will be created at D:\PR1ADO\P1_VB_ADO\BusinProc.
More
OK
Cancel
Help

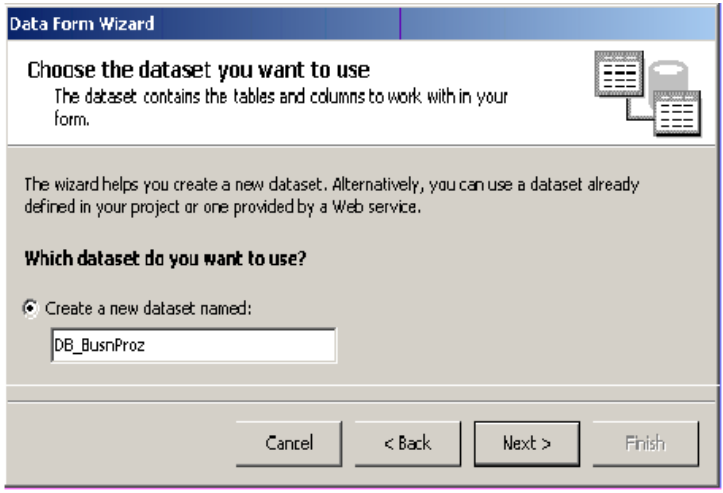
ნახ.4.5.


Add New Item - BusinProc
Categories:
Templates:
Windows Form
Class
Module
Component Class
User Control
Data Form Wizard
DataSet
XML File
XML Schema
A form for Windows Applications
Name:
Form2.vb
Open
Cancel
Help

ნახ.4.6.



ნახ.4.7.

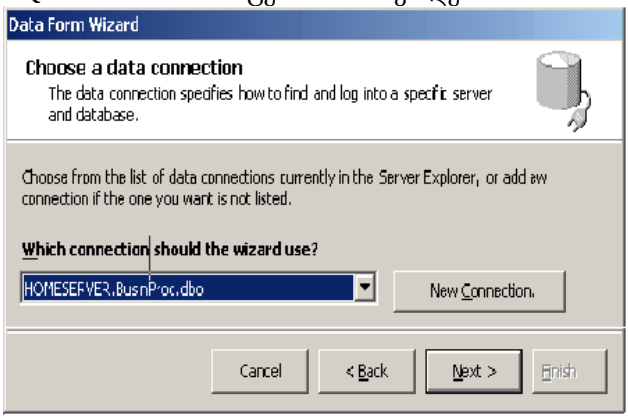


ნახ.4.8.

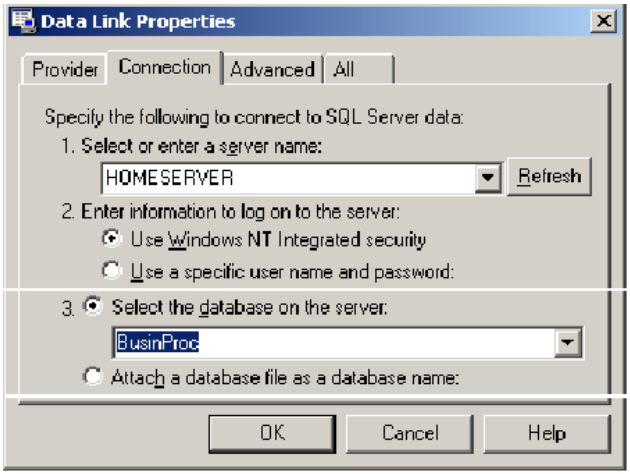
შევარჩიოთ მონაცემთა ბაზის სახელი და Next.

მონაცემთა ბაზასთან კავშირის დასამყარებლად 4.9 ნახაზის შესაბამისი ფანჯრიდან New Connection-ში შევარჩევთ სახელს.

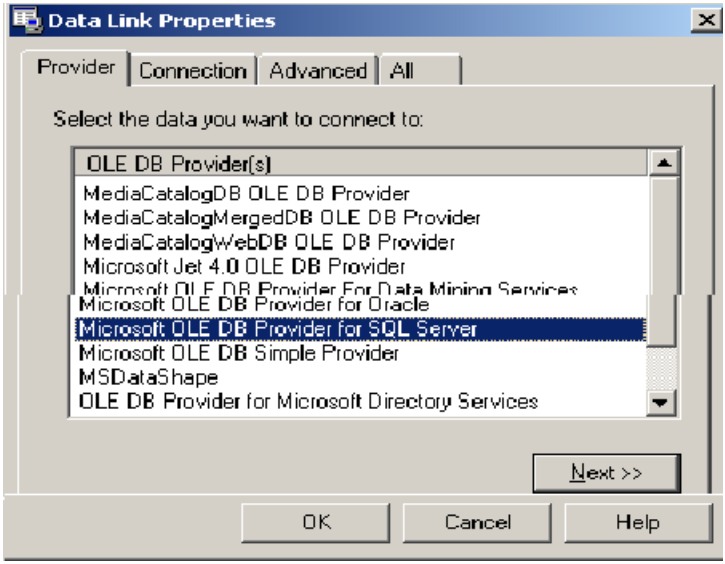
ამისათვის უნდა განვიხილოთ მომდევნო სამი ფანჯარა, შესაბამისად ნახ.4.10-ა,ბ,გ. როგორც ვხედავთ, ჩვენ შევარჩიეთ SQL Server-ის მონაცემთა პროვაიდერი.



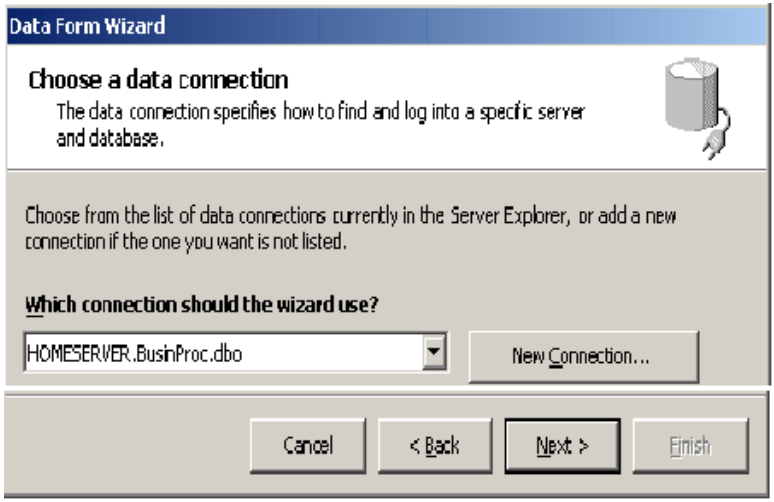
ნახ.4.9



ნახ.4.10-ა



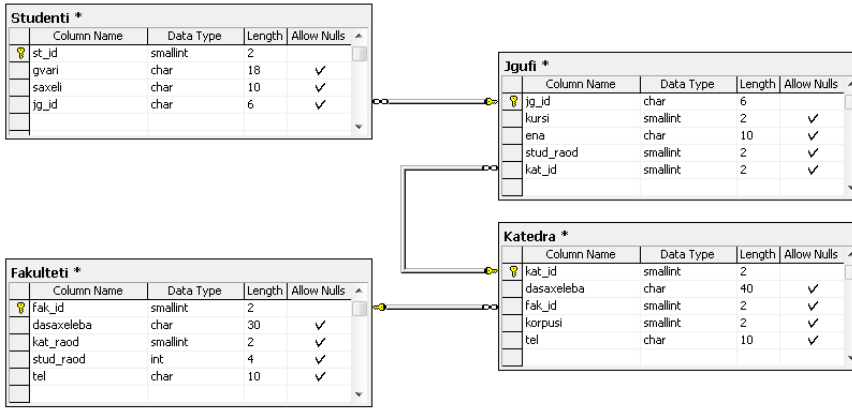
ნახ.4.10-ბ



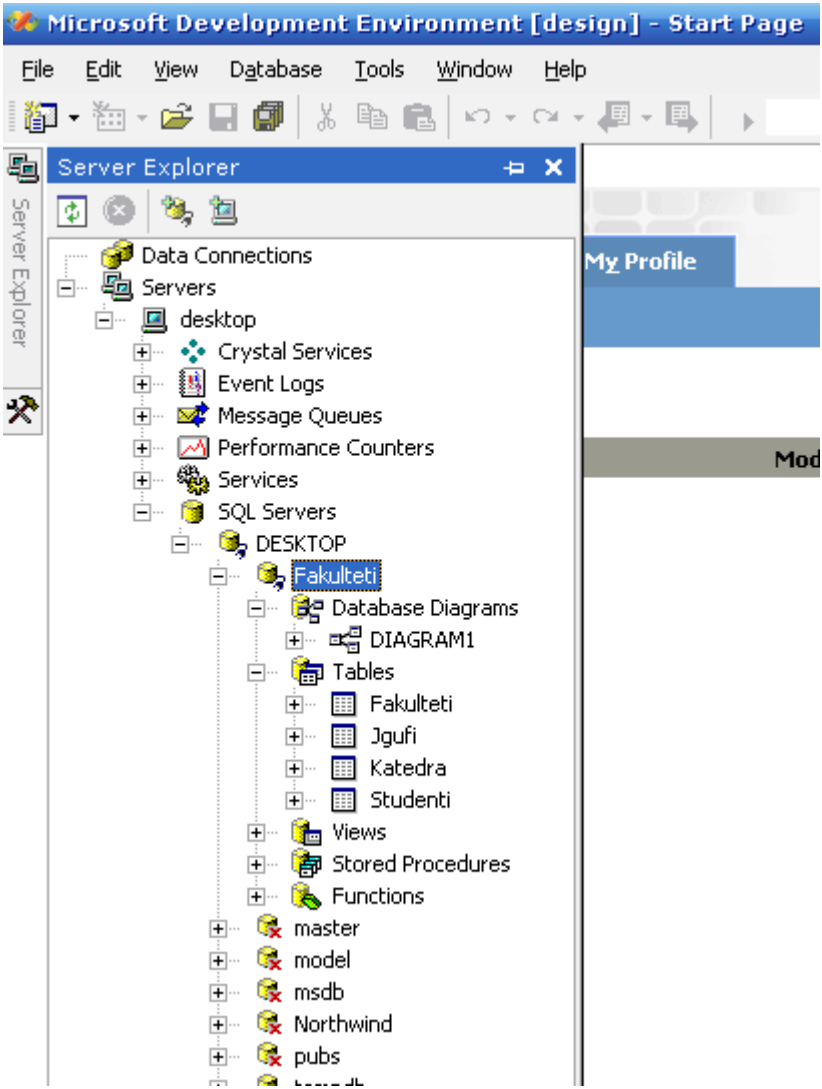
ნახ.4.10-გ

4.8. სადემონსტრაციო მაგალითი MsSQL Server ბაზის, ADO.NET და C# ის გამოყენებით

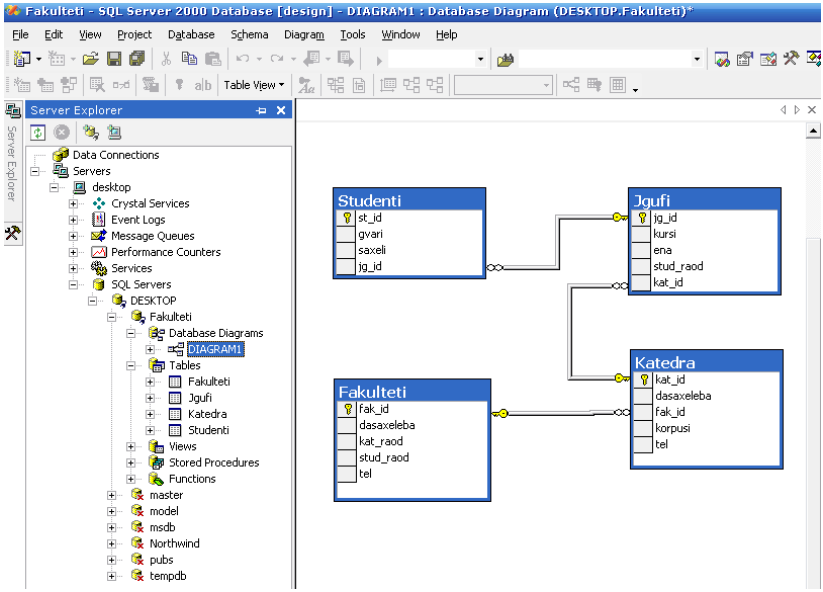
განვიხილოთ მაგალითი SQL Server-ის მონაცემთა ბაზისათვის ,,ფაკულტეტი“ (ნახ.4.11-4.16) ცხრილებით ,,სტუდენტები“, ,,ჯგუფები“, ,,ლექტორები“ და ,,კათედრები“.



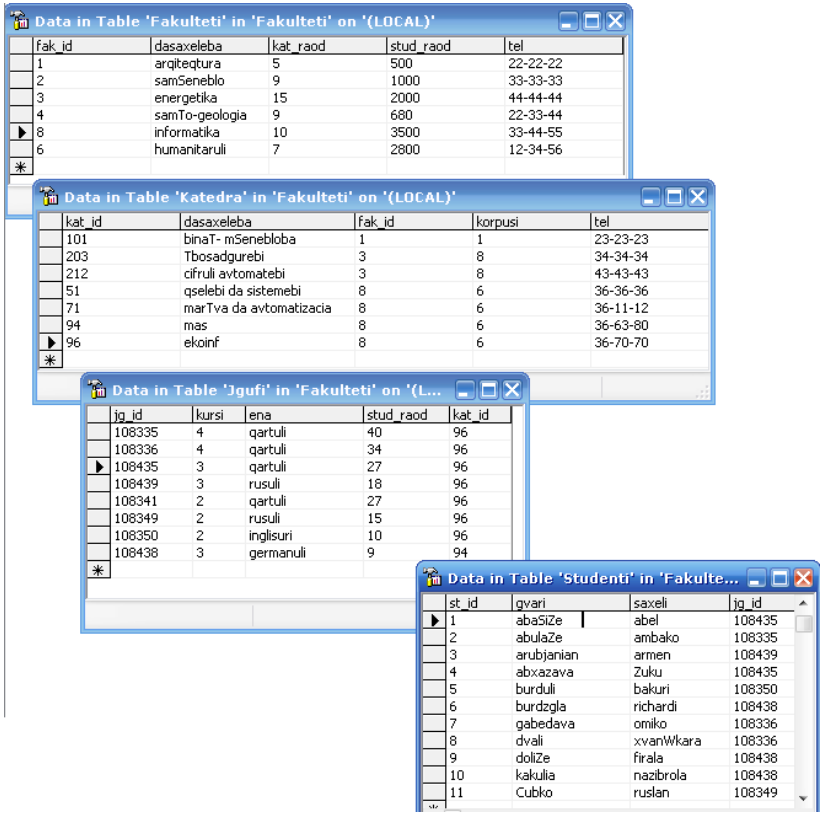
ნახ.4.11. MsSQL Server ბაზის სტრუქტურა



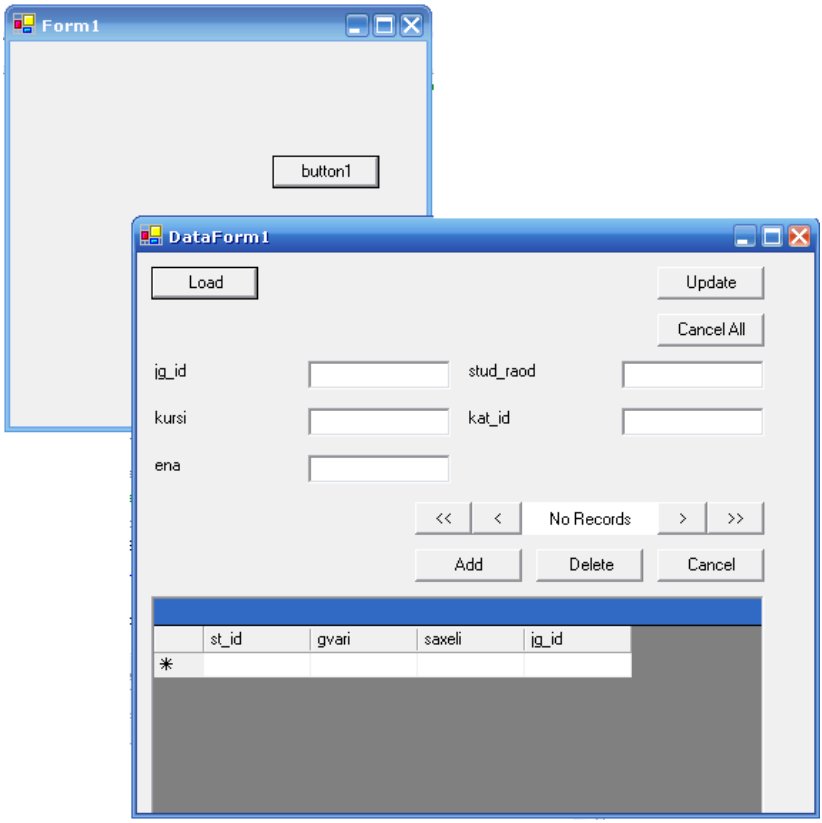
ნახ.4.12. MsVisual Studio ბაზასთან მიერთება



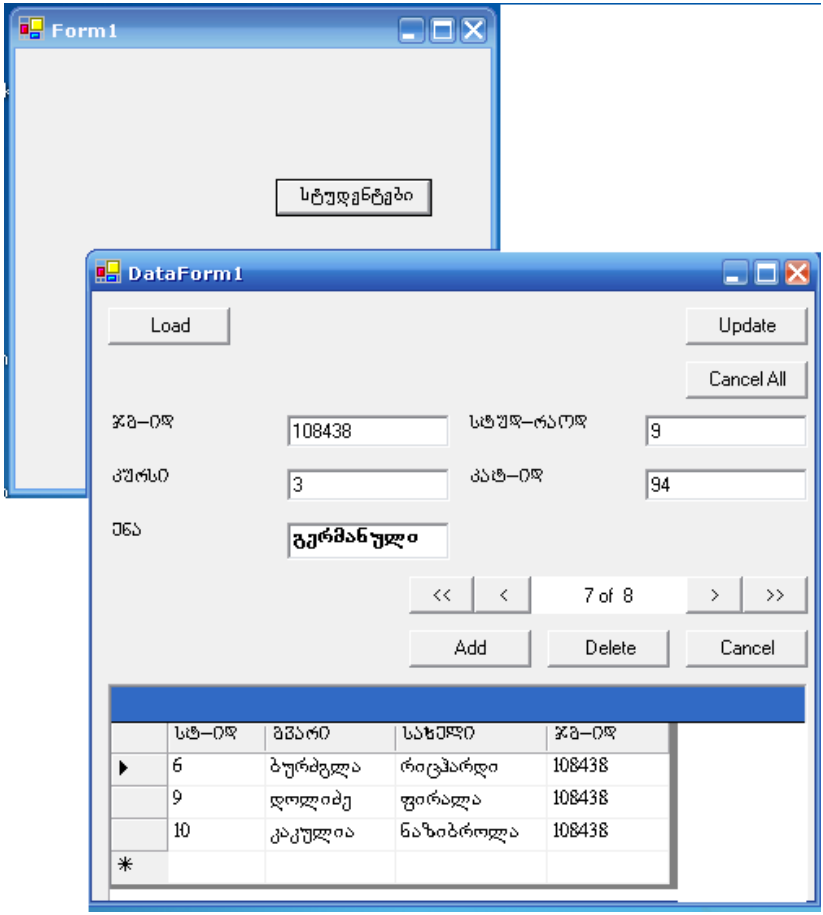
ნახ.4.13. ADO.NET ბაზის დიაგრამა



ნახ.4.14. ADO.NET ბაზის ცხრილები



ნახ.4.15. C# -ის ინტერფეისი



ნახ.4.16. Visual Studio-C# -ის შედეგები

შედეგად მიიღება C#.NET–ის ორი ფანჯარა, რომლის ცხრილებიც დაკავშირებულია SQL Server-ის მონაცემთა ბაზასთან ADO.NET პროვაიდერის საშუალებით.

4.9. ტესტირების ნიმუში

მოდული: დაპროგრამების ტექნოლოგიები
სასწავლო კურსი: **განაწილებული სისტემების დაპროგრამება C#.NET და VB.NET პლატფორმაზე**

1. ზოგადი ნაწილი .Net Framework პლატფორმა:

1. **.Net Framework** პლატფორმის არსი და მისი მუშაობა **Windows** გარემოში.
2. ჩამოთვალეთ **.Net Framework** პლატფორმის მთავარი კომპონენტები და აღწერეთ მათი დანიშნულება.
3. რას წარმოადგენს საერთო ტიპების სისტემა. როგორია მითითებითი და მნიშვნელობითი ტიპების სტრუქტურა ?
4. როგორ ხდება **.Net Framework-**ის საბაზო კლასების წევრებზე მიმართვა მათი შემოკლებული სახელებით.

2. C#.NET –ის საწყისები:

5. **C#** -ის რომელ გასაღებურ სიტყვებს იცნობთ. ჩამოთვალეთ და მოკლედ დაახასიათეთ მათი დანიშნულება.
6. რომელი ძირითადი ტიპები გამოიყენება **C#** -ში ? დაწერეთ პროგრამა მონაცემთა შეტანა-გამოტანის მაგალითისათვის კონსოლის რეჟიმში.
7. **C#** -ის ტიპების გარდაქმნის ძირითადი ფუნქციები.

3. C#.NET –ის ვიზუალური კომპონენტები:

8. ტექსტის შეტანისა და გამოტანის კომპონენტები.
9. ღილაკის და მთავარი მენიუს კომპონენტები.
10. გადამრთველი და რადიო-ბუტონები.
11. ტექსტური სია (ListBox) და დინამიკური სია (ComboBox). ტექსტის ჩაწერა.
12. მრავალგვერდიანი დიალოგი (TabControl).
13. თვის კალენდრის, თარიღისა და დროის რედაქტირების კომპონენტები.

4. ASP.NET–ის ვიზუალური კომპონენტები:

14. ASP.NET–ის საბაზო არქიტექტურა და მუშაობის პრინციპები.
15. როგორ იქმნება ვებ-გვერდი ASP.NET–ით ?
16. ASP.NET–ით ინტერაქტიული გვერდის შექმნა.
17. ASP.NET–ში მონაცემთა ბაზასთან მუშაობა.
18. Visual C#.NET და ASP.NET–პაკეტებით ინტეგრირებული მუშაობის არსი

5. C#.NET –ის გამოყენებით მომხმარებლის ინტერფეისის აგების ამოცანა ADO.NET გარემოში:

1. ვიზუალური კომპონენტების გამოყენებით ააგეთ აპლიკაცია ობიექტისათვის **სუპერმარკეტი**, ობიექტებით: **პროდუქციის–კატეგორია**, **პროდუქცია**, **ფირმა** (მ-ბაზა:MsSQLServer-ში).

2. ააგეთ სუპერმარკეტის მომხმარებლის (მაგ., მენეჯერის) ინტერფეისი მენიუთი და ფორმებით. პირველი სამი ფორმა მონაცემთა შეტანა-კორექტირების პროცედურებს ემსახურება.

3. ააგეთ მე-4 ფორმა, რომელზაც ასახული იქნება პროდუქციის სია (ცხრილი-1)კატეგორიების (ცხრილი-2) და ფირმების (ცხრილი-3) მიხედვით.

ლიტერატურა

1. სურგულაძე გ., დოლიძე თ., ყვავაძე ლ. კომპონენტურ-ვიზუალური დაპროგრამება: ინტერფეისების აგება C# და C++ ენებზე მონაცემთა განაწილებული ბაზებისათვის. სტუ, თბილისი, 2006.

2. Майо Дж. C#: Искусство программирования. Энциклопедия программиста. Пер.с англ., "DiaSoft", СПб., 2002.

3. Robinson S., Cornes O., Glynn J., Harvey B., McQueen C., Moemeka J., Nagel C., Skinner M., Watson K. Professional C#. Bimingham, WroxPress, 2001.

4. Sceppa D. Microsoft ADO.NET. Пер. с англ., М., «Русская Редакция», 2003.

5. Microsoft Corporation. Разработка Windows-приложений на Ms VB и Ms VC#.NET. Пер. с англ., М., «Русская Редакция», 2003.

6. სურგულაძე გ., შონია ო., ყვავაძე ლ. მონაცემთა განაწილებული ბაზების მართვის სისტემები (MsSQL Server, Access, InterBase, JDBC, Oracle). სტუ, თბილისი, 2004.

იბეჭდება ავტორთა მიერ
წარმოდგენილი სახით

გადაეცა წარმოებას 30.01.2009 წ. ხელმოწერილია დასაბეჭდად 11.02.2009 წ. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 9. ტირაჟი 100 ეგზ.

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“
თბილისი, მ. კოსტავას 77

ი.მ. „გოჩა დალაქიშვილი“,
ქ. თბილისი, ვარკეთილი 3, კორპ.333, ბ.38