საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე

.NET ტექნოლოგია

(შესავალი, C#.NET, ADO.NET)

დამტკიცებულია
სტუ-ს სარედაქციო-
საგამომცემლო
საბჭოს მიერ

თბილისი 2007

უაკ 681.3.06

გადმოცემულია კომპონენტურ-ვიზუალური დაპროგრამების თეორიული საფუძვლები *UML* ტექნოლოგიისა და *Microsift*-ფირმის *.NET*-პლატფორმისათვის.

განხილულია დაპროგრამების ვიზუალური ინსტრუმენტული საშუალება *Ms C#.NET*, მონაცემთა ბაზების დრაივერი *ADO.NET* პაკეტი და მათი ერთობლივი გამოყენების საკითხები.

სახელმძღვანელოში შემოთავაზებულია მომხმარებელთა ინტერფეისების დამუშავების ტექნოლოგია ვიზუალური კომპონენტებით, საილუსტრაციო მაგალითების ჩვენებით. ინტერფეისების აგება და გამოყენება ხორციელდება მონაცემთა განაწილებულ რელაციურ ბაზებთან სამუშაოდ კლიენტ-სერვერ არქიტექტურის გარემოში.

განკუთვნილია ინფორმატიკის სპეციალობის სტუდენტების, მაგისტრანტებისა და სპეციალისტებისათვის.

ISBN 99940-48-99-6

ს ა რ ჩ ე ვ ი

შესავალი

I თავი. .NET პლატფორმის არსი

1.1. დაპროგრამების თანამედროვე პლატფორმები და ენები

1.2. პლატფორმა .NET

1.3. საერეთო ტიპების სისტემა (CTS)

1.4. .NET-პლატფორმის კომპონენტები და მისი დანართის სტრუქტურა

II თავი. დაპროგრამების ინსტრუმენტი C#.NET

2.1. C# ენის სამუშაო გარემო კონსოლის რეჟიმში

2.2. C# კოდის აგება და ტესტირება

2.3. C# ენის გასაღებური სიტყვები

2.4. C# ენის მონაცემთა ძირითადი ტიპები

2.5. C# ენის ტიპების გარდაქმნის ძირითადი მეთოდები

2.6. C# ენის მასივები (კოლექციები)

2.7. C# ენის ოპერაციები

2.8. C# ენაში ბრძანებათა ნაკადების მართვა

2.9. C# კოდის მაგალითი

III თავი. C#.NET -ის ობიექტ-ორიენტირებული დაპროგრამების ელემენტები

3.1. C# ენაში ობიექტური და კომპონენტური დაპროგრამების კონცეფცია

3.2. C# ენის ობიექტები და კლასები

3.3. კომპონენტ-ორიენტირებული დაპროგრამება

3.4. C# კოდში შეცდომებისა და გამოსარიცხ მოვლენათა დამუშავება

3.5. C# კოდის ორგანიზება სახელთა სივრცის დახმარებით

IV თავი. C# -ის ვიზუალური კომპონენტები

4.1. .NET პლატფორმის სამუშაო გარემო

4.2. C# ენის ვიზუალური კომპონენტები

4.3. C# ენის ვიზუალური კომპონენტებით ფორმების აგების მაგალითები

V თავი. ADO.NET პაკეტი და ინტერფეისების აგება C#.NET ინსტრუმენტით

5.1. Ms ADO.NET პაკეტი მონაცემთა ბაზებთან სამუშაოდ

5.2. ADO.NET-ის ობიექტური მოდელის სტრუქტურა

5.3. ინტერფეისის დამუშავების სადემონსტრაციო მაგალითი

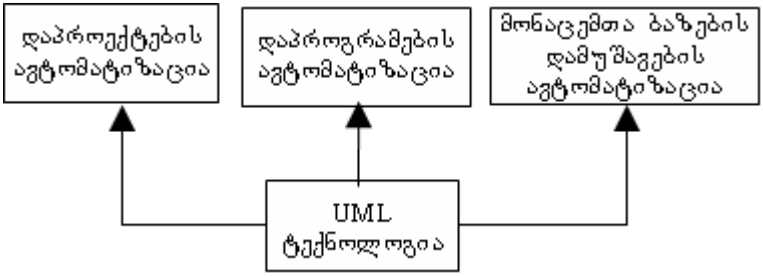
5.4. მონაცემთა ბაზის ცხრილების გამოტანა C#-ენისათვის

ლიტერატურა

შესავალი

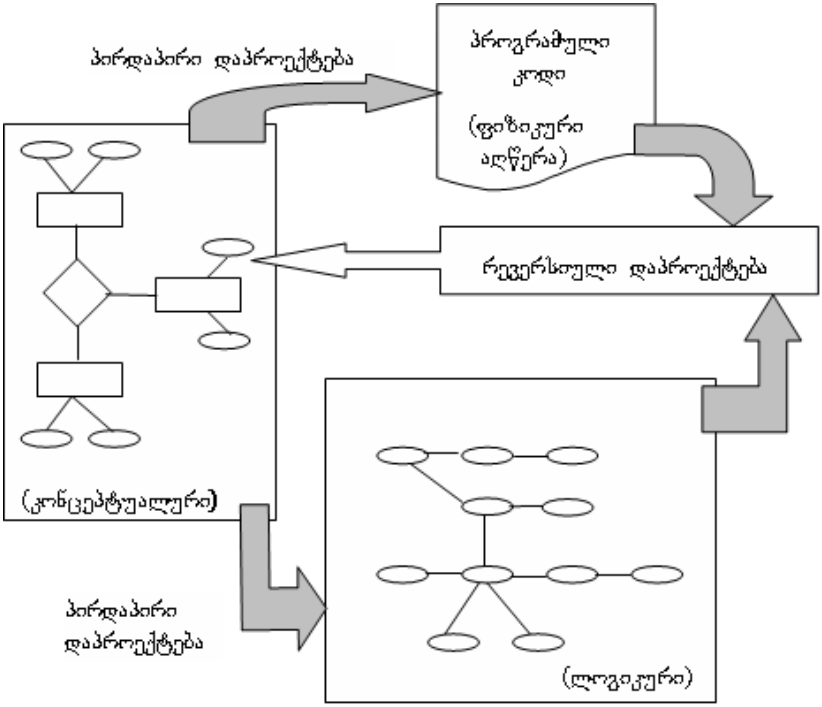
კომპიუტერული ინდუსტრია და საინფორმაციო ტექნოლოგიები განვითარების მაღალი ტემპებით ხასიათდება. მსოფლიო ბაზარზე გამოჩნდა არა ერთი ახალი აპარატურული (Hardware) და პროგრამული (Software, Groupware) სისტემები. მათ შორის საყურადღებოა ამერიკული კორპორაციის Rational Software მეცნიერ-კონსტრუქტორების უნიფიცირებული მოდელირების ენის - UML-ტექნოლოგია კომპიუტერული სისტემების პროგრამული უზრუნველყოფის ასაგებად. Unified Modeling Language, როგორც უახლესი სტანდარტი, საფუძვლად დაედო თანამედროვე საინფორმაციო ტექნოლოგიების სისტემებს [1,2].

UML-ტექნოლოგია ობიექტ-ორიენტირებული მოდელირებისა და სტრუქტურული დაპროგრამების იდეოლოგიის მატარებელია, რომელიც თეორიული და პრაქტიკული ინფორმატიკის სამი ძირითადი მიმართულების „გენეტიკური“ მემკვიდრეა (ნახ.1):



ნახ.1

მართვის კომპიუტერული სისტემების პროგრამული უზრუნველყოფის აგების პროცესების ასეთი სრულფასოვანი ავტომატიზაცია კომპონენტურ-ვიზუალური დაპროგრამების სახელწოდებით დამკვიდრდა და იგი მოდელების გრაფო-ანალიზურ წარმოდგენას ეყრდნობა. ასეთი ინსტრუმენტები ფლობს როგორც პირდაპირ (გრაფიკიდან პროგრამული კოდისაკენ), ასევე რევერსიულ (კოდიდან გრაფიკისაკენ) ტექნოლოგიას (ნახ.2).



ნახ.2

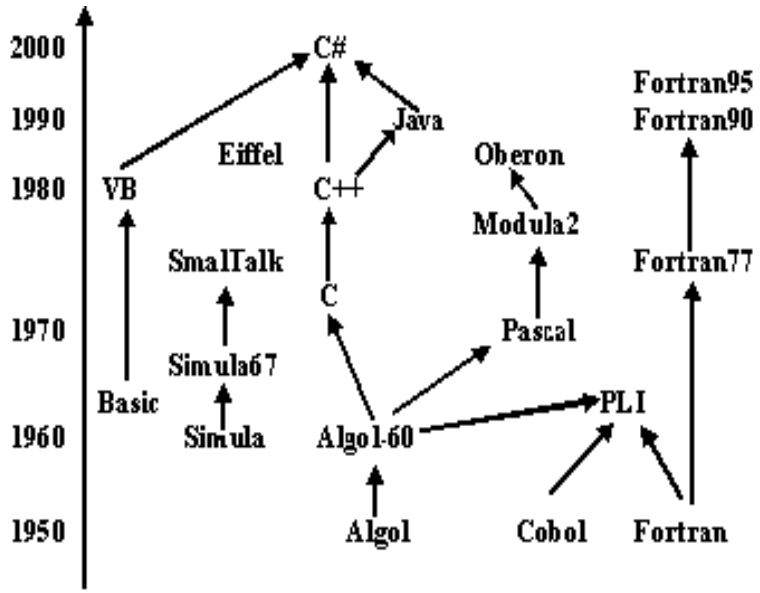
შეიძლება დავასკვნათ, რომ UML–ტექნოლოგია (ენა) დაპროგრამების ენებისა და მონაცემთა ბაზების მართვის სისტემების ერთგვარ სიმბიოზს, ინტეგრირებულ ინსტრუმენტს წარმოადგენს [1,12].

ფორმალურად იგი შეიძლება განვიხილოთ როგორც „წარმოებული კლასი“ დაპროგრამების ენისა და მონაცემთა ბაზების მართვის სისტემის „საბაზო კლასებიდან“, რომელსაც გააჩნია როგორც „მშობლების“ თვისებები, ასევე ახალი, მძლავრი ვიზუალური მახასიათებლები [4].

რამ განაპირობა ასეთი ტიპის ინტეგრირებული მძლავრი კომპიუტერული ინსტრუმენტის შექმნა ? განვიხილოთ მოკლედ კომპიუტერული სისიტემების აგების ტექნოლოგიების განვითარების გზაზე ინსტრუმენტების ძირითადი თვისებები და კონცეფციები,

რომლებმაც თავი მოიყარა („მემკვიდრული“ თვისებების სახით) UML-ტექნოლოგიის კომპონენტურ-ვიზუალურ დაპროგრამებაში.

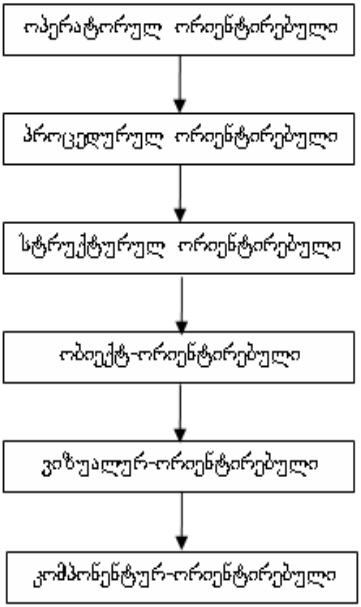
50-70-იან წლებში, როგორც ცნობილია, გავრცელებული იყო ოპერატორული და პროცედურულ-ორიენტირებული სტილის დაპროგრამების ენები (ნახ.3, 4). „ტანდემურ“ პრინციპზე მომუშავე ეს ტრადიციული ინსტრუმენტები პროგრამულ კოდებს და მათთვის საჭირო ინფორმაციულ მონაცემებს ერთად, ერთ ფაილში ათავსებდა.



ნახ.3

ეს პრინციპი დღეს ობიექტ-ორიენტირებული სტილის ენებშიც შენარჩუნებულია: კლასი არის მონაცემებისა და ამ მონაცემების დამუშავების ფუნქციების ინკაფსულირებით მიღებული კომპონენტი.

ასეთი მიდგომა აღნიშნულ ტრადიციულ ენებში, ერთის მხრივ სისტემის საიმედოობას ზრდიდა, მაგრამ, მეორეს მხრივ ასევე იზრდებოდა პროგრამული კოდისა და მონაცემთა მოვლის, მოდიფიკაციის, შედეგების მიღების დრო, მანქანური მეხსიერების რესურსების ხარჯი ინფორმაციის დუბლირების გამო და ა.შ.



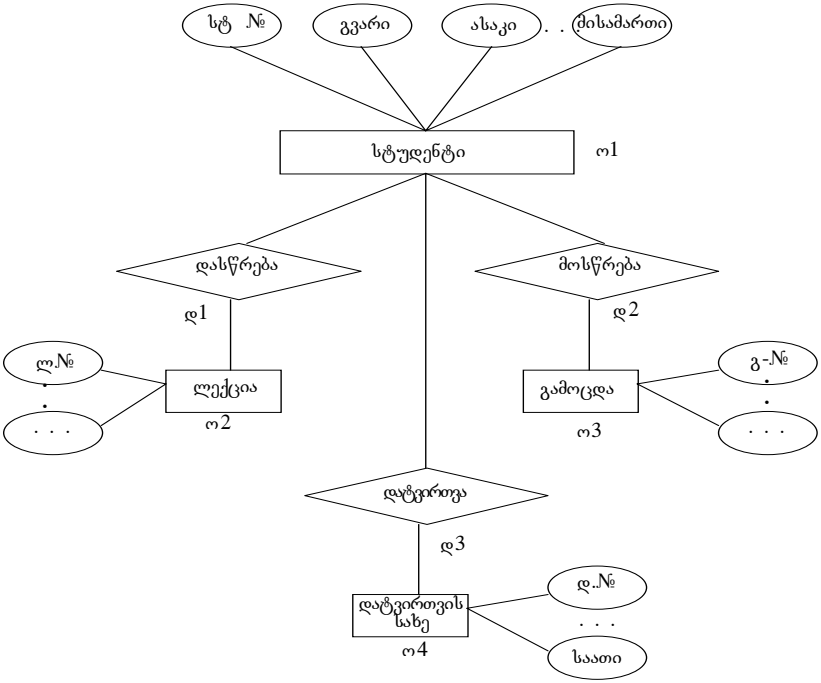
ნახ.4

70-90-იან წლებში განვითარდა მონაცემთა ბაზების მართვის სისტემების (მბმს) პროგრამული პაკეტები. აქ ძირითადი კონცეფცია, განსაკუთრებით რელაციურ ბაზებში, პროგრამული კოდისა და მონაცემების ცალ-ცალკე შენახვის იდეაში მდგომარეობდა.

რელაციურ მონაცემთა ბაზების მართვის სისტემები დღესაც აქტუალურ მიმართულებად ითვლება, მაგალითად Oracle, Access, SQLServer, InterBase, MySQL და სხვ. [5].

ამ სისტემებში ერთ-ერთი მნიშვნელოვანი კომპონენტია საპრობლემო სფეროს კონცეპტუალური სქემა (ნახ.5) ანუ ER-მოდელები (Entity-Relations Model), რომლის საფუძველზეც დაპროექტდება შემდგომში მონაცემთა ბაზის ლოგიკური სტრუქტურა.

დაპროგრამების ენებთან შედარებით მბმს იყო პირველი ცდა პროგრამულ კოდში მონაცემთა სტრუქტურების აგების ავტომატიზაციისა.

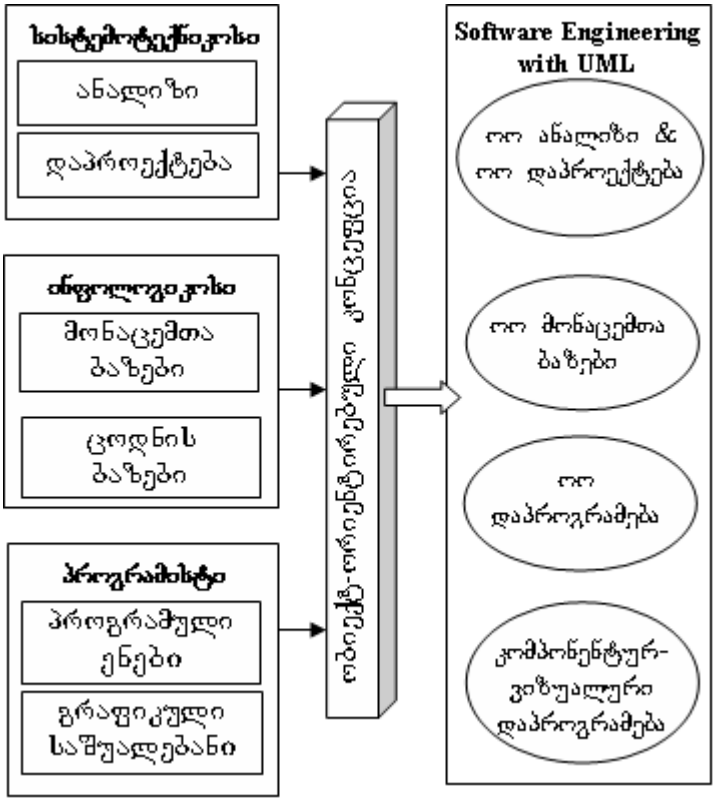


ნახ.5

ობიექტ-ორიენტირებულ დაპროგრამების ენებში და UML-ტექნოლოგიაშიც ეს კონცეპტუალური ER-მოდელები, „კლასების დიაგრამების“ სახით, მნიშვნელოვანი ვიზუალური კომპონენტია. მომხმარებელი Use Case (გამოყენებითი შემთხვევა) დიაგრამებიდან ააგებს კლასების, შემდეგ კი კომპონენტების დიაგრამებს, რომლებიც საბოლოოდ ფიზიკური განლაგების დიაგრამებში აისახება [4,12].

90-ანი წლებიდან მნიშვნელოვნად განვითარდა სტრუქტურული დაპროგრამებისა და ობიექტ-ორიენტირებული მოდელირების კონცეფცია (ნახ.6) ენები, მაგალითად C++, Java, Eiffel [2].

2000 წლიდან დღემდე ეს ენები ვიზუალური და კომპონენტური თვისებებით გამდიდრდა, დაიხვეწა დაპროგრამების ინსტრუმენტები და გაფართოვდა გრაფიკული ინტერფეისები [12].



ნახ.6

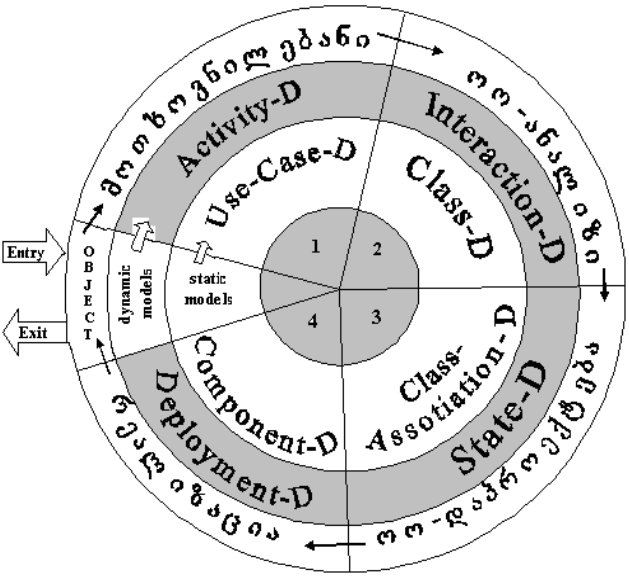
C#, XML, Visual-C++, Visual Basic, J++ და სხვ. კომპიუტერული დაპროგრამების ის ინტეგრირებული პაკეტებია რომლებიც დღეისათვის ყველაზე პოპულარული და აქტუალურია ამერიკისა და ევროპის თითქმის ყველა უნივერსიტეტსა და ბიზნესის მართვის სფეროში.

განსაკუთრებით საყურადღებოა ამ თვალსაზრისით უახლესი საინფორმაციო ტექნოლოგია, რომელიც dot-NET პლატფორმითაა ცნობილი. იგი აღჭურვილია ისეთი სპეციალური მედიატორული თვისებებით, რომლებიც უზრუნველყოფს ზემოჩამოთვლილ დაპროგრამების ენებს შორის სრულ თავსებადობას.

ეს საკითხები არის წარმოდგენილი სწორედ წინამდებარე წიგნის მომდევნო თავებში.

შესავლის დასკვნით ნაწილში გვინდა განვიხილოთ UML-ტექნოლოგიის განზოგადებული კონცეფცია და მისი, როგორც კომპიუტერული სისტემების ავტომატიზებული დაპროგრამების მეთოდოლოგიური საფუძლის კონკტრეტული რეალიზაციის ეტაპები.

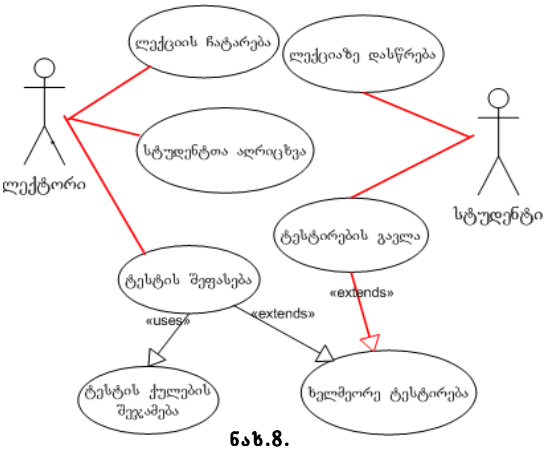
პროგრამული პაკეტების შექმნა UML-ტექნოლოგიის მიხედვით ოთხ ეტაპად ხორციელდება (ნახ.7).



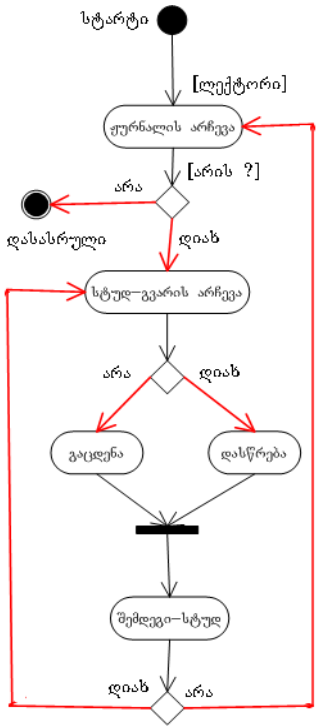
ნახ.7. UML-ეტაპები

საკვლევი ობიექტის ავტომატიზაციის მოთხოვნილებების დადგენა, მისი ობიექტ-ორიენტირებული (ოო) ანალიზი, ოო-დაპროექტება (დეტალური დონე) და რეალიზაცია (პროგრამული კოდი). ეს ეტაპები სტატიკური და დინამიკური დიაგრამებით (D) ხორციელდება.

UseCase-D დიაგრამა უჩვენებს როლებს (Actor) და მათ ფუნქციებს (Action), აგრეთვე მათ კავშირებს (ნახ.8); ყოველ UseCase-ფუნქციას (ოვალს) შეესაბამება ერთი დინამიკური მოდელი, რომელიც **Activity-D** დიაგრამის სახით ფორმირდება (ნახ.9).



ნახ.8.



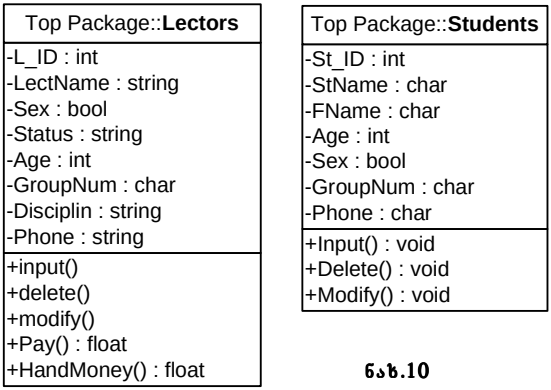
ნახ.9.

აქტიურობათა დიაგრამას ერთი დასაწყისი და რამდენიმე დასასრული შეიძლება ჰქონდეს. მასში მონაწილეობს რამდენიმე "როლის" შემსრულებელი (მაგ., ლექტორი, სტუდენტი, დეკანი და ა.შ.).

მრგვალკუთხედებში მოთავსებულია მათ მიერ შესასრულებელ პროცედურათა დასახელებები.

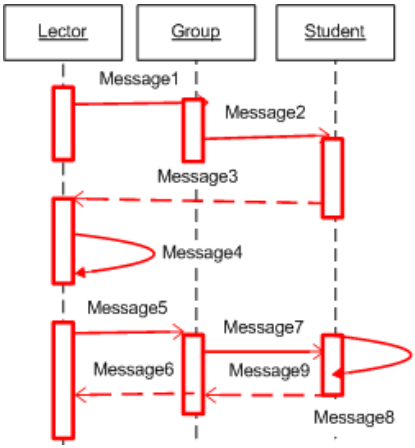
პროცედურები შეიძლება შესრულდეს მიმდევრობით ან პარალელურად.

კლასებისა (Class-D) და ინტარაქტიურობათა (Sequence-D, Collaboration-D) დიაგრამების ასაგებად საჭიროა ოო-მოდელირების საფუძველზე განისაზღვროს კლასთა დასახელებები (ნახ.10, 11 და 12), მათი ატრიბუტები (მონაცემები) და ფუნქციები (მეთოდები).



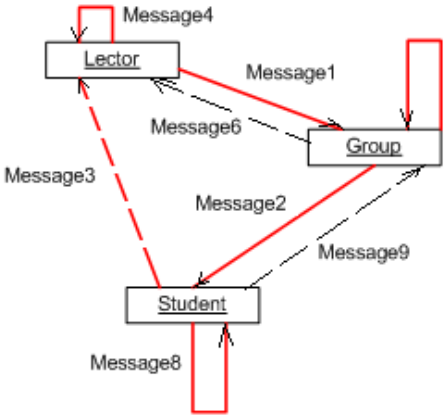
ნახ.10

მიმდევრობითობის დიაგრამაზე შეტყობინებები და ოპერაციები დალაგებულია მათი შესრულების მიმდევრობით, აქ მთავარი დროა.



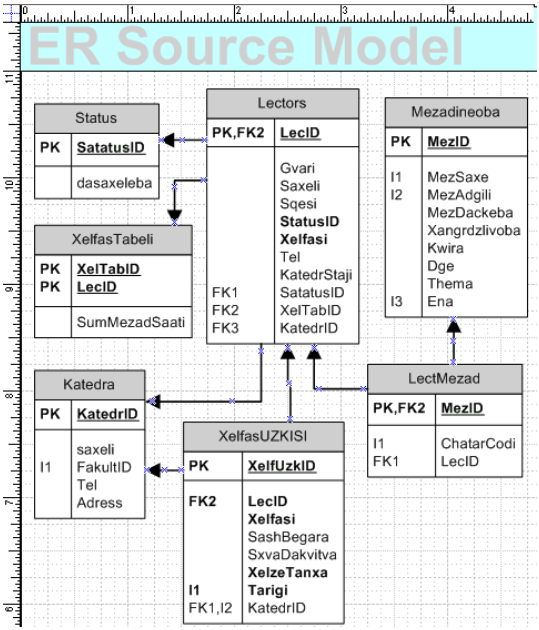
ნახ.11

თანამოქმედების დიაგრამაზე კარგად ჩანს კლასებს შორის ინფორმაციის გაცვლა შეტყობინებებისა და მეთოდების გამოყენების საფუძველზე.



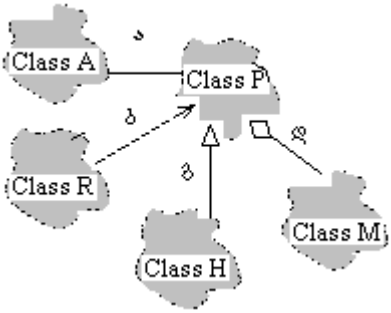
ნახ.12

ამ ეტაპზე სასარგებლოა "არსთა დამოკიდებულების მეთოდის" (Entity-Relationship-Model) გამოყენება, რომლის მაგალითი მე-5 ნახაზზეა წარმოდგენილი. მე-13 ნახაზზე ნაჩვენებია MsVisio-პაკეტით აგებული ER-მოდელი.



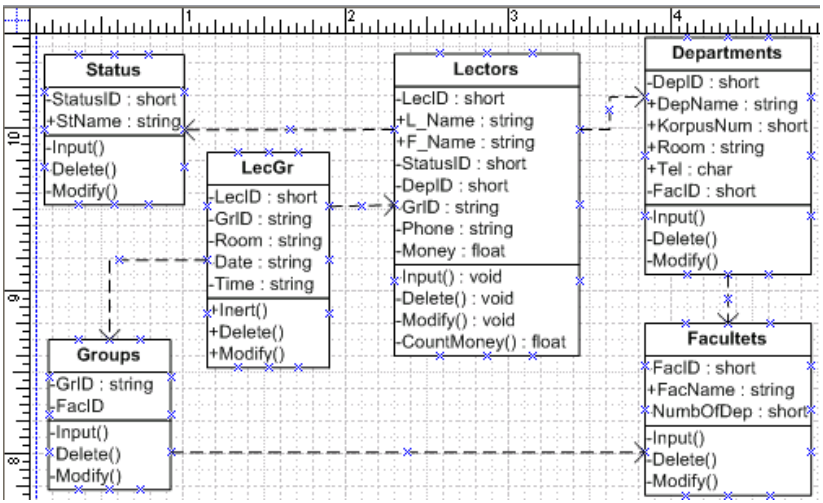
ნახ.13.
PK-პირველადი და FK-მეორადი გასაღებები, In -ინდექსური კავშირი

Class-D კლასების დიაგრამა გამოიყენება კლასებისა და მათ შორის კავშირების (Class-Assotiation-D) აღსაწერად. კავშირები კლასებს შორის ოთხი ტიპისაა: ასოციაციური-a, რელაციური-r, აგრეგატული-ag და მემკვიდრეობითი-c. მათი გრაფიკული აღნიშვნები მოცემულია მე-14 ნახაზზე.



ნახ.14.

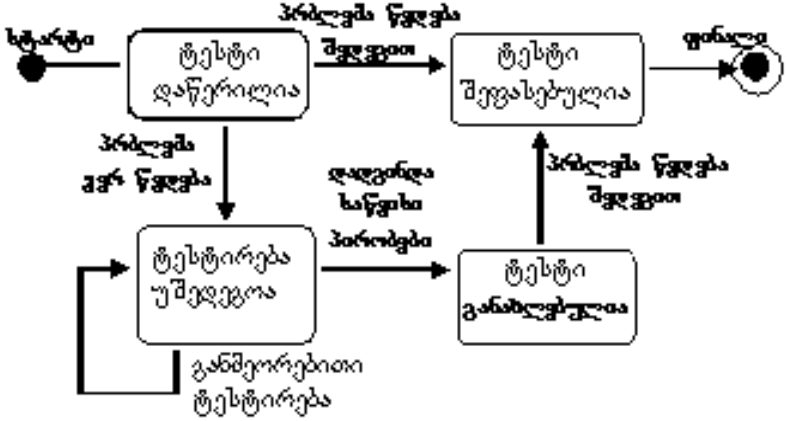
მე-15 ნახაზზე ნაჩვენებია MsVisio-ში აგებული კლასთა-კავშირების სების დიაგრამა.



ნახ.15.

ყოფაქცევის დიაგრამებიდან ჩვენ უკვე განვიხილეთ Activity-D და Interaction-D. არსებობს აგრეთვე კლასების მდგომარეობათა დიაგრამა ანუ State-D (ნახ.16). იგი აღწერს მოქმედებებს, ობიექტთა მდგომარეობებს, მდგომარეობათა გადასვლებს და მოვლენებს.

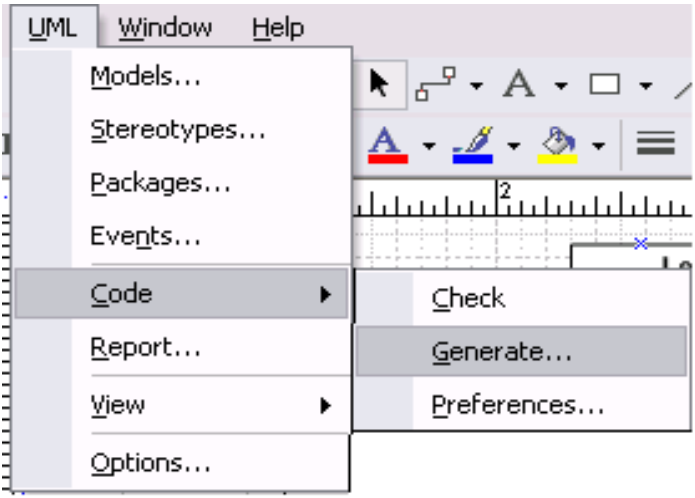
მისი გამოყენება ყველა კლასისთვის არაა საჭირო. აუცილებელია მაშინ, როდესაც კლასი შეიძლება იმყოფებოდეს რამდენიმე მდგომარეობაში და თითოეულ მათგანში იგი იქცევა სხვადასხვანაირად.



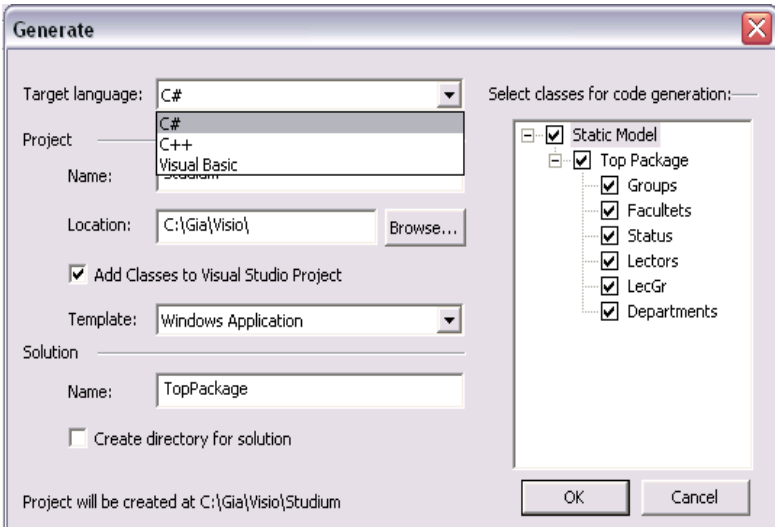
ნახ.16.

პროგრამული კოდის ავტომატური გენერაცია შესაძლებელია კლასთა კავშირების დიაგრამის აგების შემდეგ (ნახ.17). მენიუდან ავირჩევთ UML | Code | Generate, რის შემდეგაც გამოჩნდება ახალი კადრი (ნახ.18). სისტემა შემოგვთავაზებს დაპროგრამების ენის არჩევას (C# , C++, Visual Basic).

აქვე უნდა ავირჩიოთ პროექტის სახელი (Name), Browse-ს გამოყენებით პროგრამული მოდულების ჩასაწერი კატალოგი (Location), მაგ., : D:\Gia\Visio-1\ და მარჯვენა ნაწილში ამოვირჩიოთ კლასები, რომელთა დაპროგრამებასაც ვაპირებთ. ნახაზზე ყველა კლასია მონიშნული. დავამოწმოთ შედეგი ღილაკით Ok.


UML
Window
Help
Models...
Stereotypes...
Packages...
Events...
Code
Report...
View
Options...
Check
Generate...
Preferences...

ნახ.17

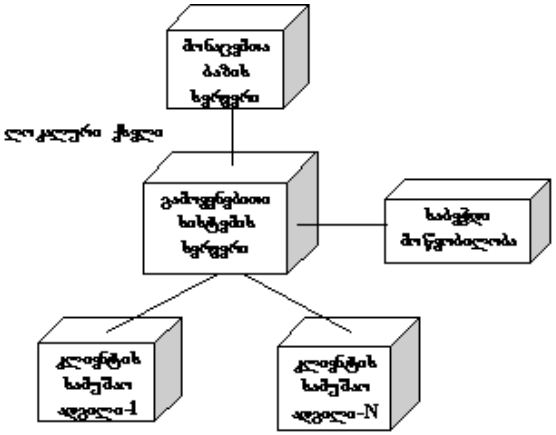

Generate
Target language:
C#
C#
C++
Visual Basic
Project
Name:
Location:
C:\Gia\Visio\
Browse...
Add Classes to Visual Studio Project
Template:
Windows Application
Solution
Name:
TopPackage
Create directory for solution
Project will be created at C:\Gia\Visio\Studium
Select classes for code generation:
Static Model
Top Package
Groups
Facultets
Status
Lectors
LecGr
Departments
OK
Cancel

ნახ.18

რეალიზაციის ეტაპზე აიგება კომპონენტების დიაგრამა (Component-D), რომელშიც იგულისხმება პროგრამული კოდების (CPP, H, DLL და ა.შ.) დამუშავება (ნახ.19).

ნახ.19. კომპონენტების დიაგრამა

პროექტის ბოლოს აიგება განთავსების დიაგრამა (Deployment-D), რომელიც აღწერს კომპონენტების განაწილებას "კლიენტ-სერვერ" ქსელში (ნახ.20).



ნახ.20. განთავსების დიაგრამა

I თავი

.NET - პლატფორმის არსი

1.1. დაპროგრამების თანამედროვე პლატფორმები და ენები

ობიექტ-ორიენტირებული, კომპონენტურ-ვიზუალური დაპროგრამების თანამედროვე საინფორმაციო ტექნოლოგიებიდან განსაკუთრებული აქტუალობით ხასიათდება ფირმის „მაიკროსოფთი“ პროგრამული პლატფორმა Visual Studio .NET (პროგრამებით C#, VB, C++ და სხვ.) და ფირმის „ბორლანდი“ ინტეგრირებული პაკეტი C++Builder [8,9,11].

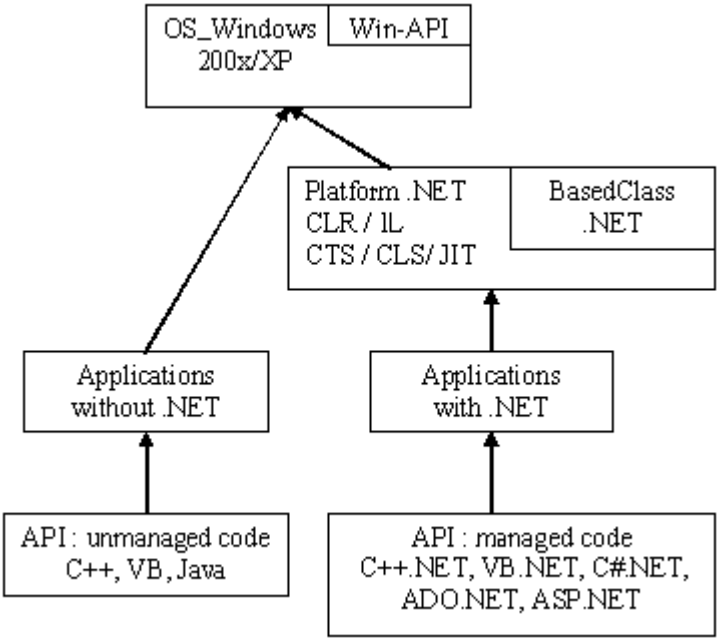
ორივე პროგრამული პაკეტის მიზანია დიდი, რთული პროგრამული სისტემების კონსტრუირება და რეალიზაცია (Software Engineering) უნიფიცირებული პროგრამირების კონცეფციის გამოყენებით.

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

1.2. 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.

1.3. 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.

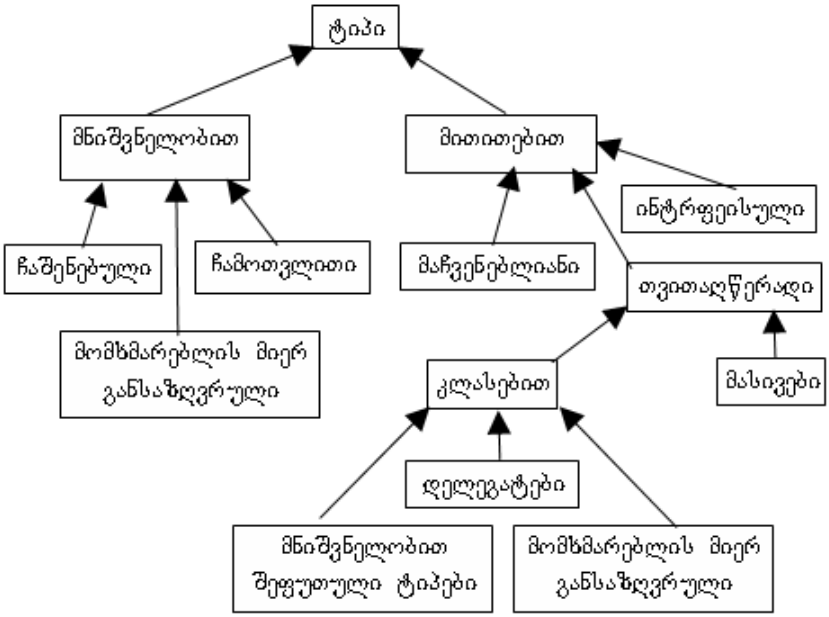
IL-ენის დანიშნულებაა პლატფორმული და ენობრივი დამოუკიდებლობის განხორციელება ობიექტ-ორიენტირებულ გარემოში. Java-ენაც უზრუნველყოფს პლატფორმულ (Windows, Unix, Linux) დამოუკიდებლობას, მაგრამ მისი ბაიტ-კოდის შესრულების ეტაპზე იგი ინტერპრეტირდება (IL -კი კომპილირდება).

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;

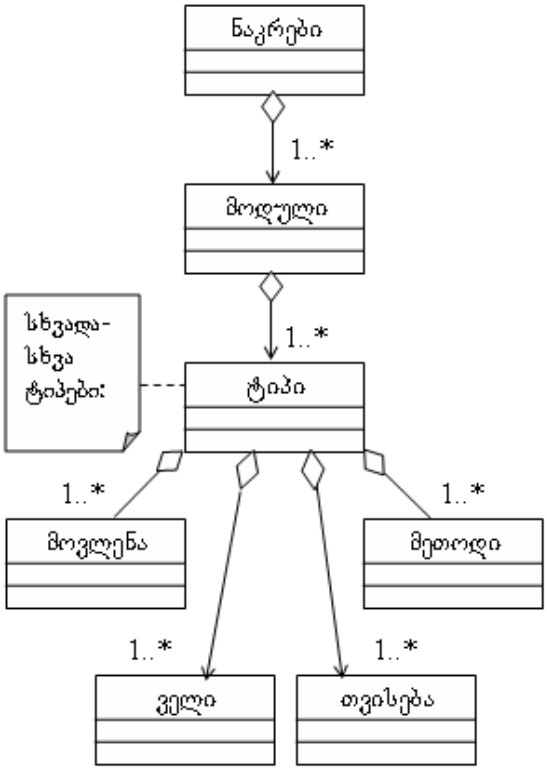
- **მომხმარებლის მიერ განსაზღვრული ტიპები მითითებით** არის ტიპები, განსაზღვრული საწყის კოდში, როგორც ტიპები მითითებით. პროგრამაში ესაა მაგალითად, ნებისმიერი კლასი.

1.4. .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 მნიძშვნელობების 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:

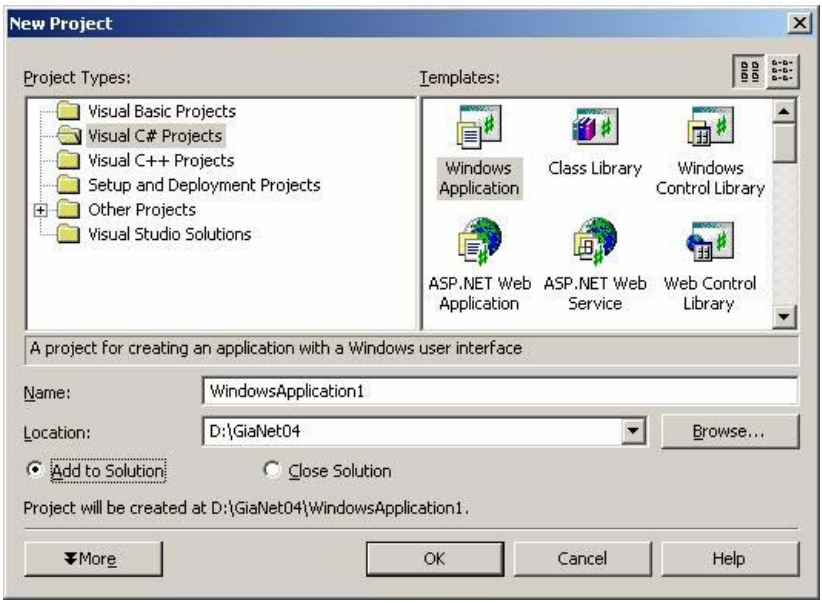
```
using System;  //  C# -------------
using System.Windows.Forms;
namespace Console1;
{
  class Class1
  {
    static int Main(string[] args)
      {
        Console.WriteLine("Hello, my friend !");
         MessageBox.Show("By-By !");
           return 0;
       }
  }
}
```

II თავი. დაპროგრამების ინსტრუმენტი C#.NET

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 განაპიროებებს.

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.

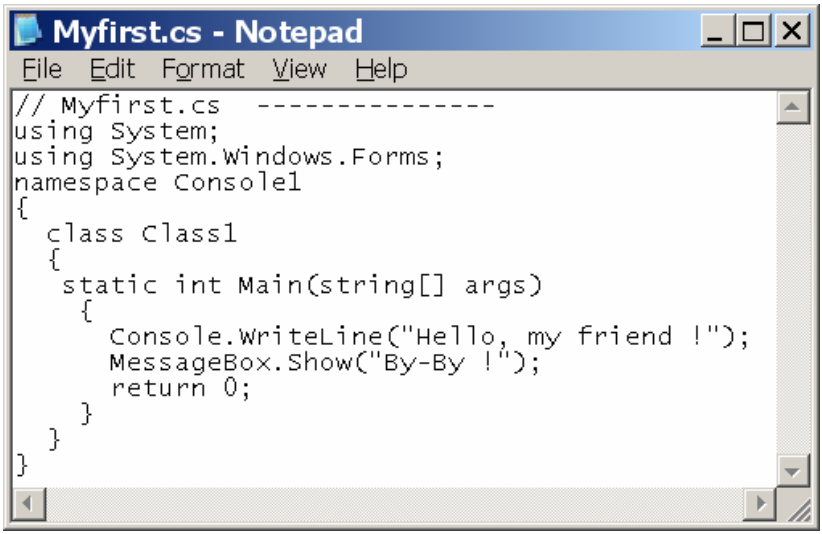
სწორედ ამ საკითხებს განვიხილავთ წინამდებარე თავის პარაგრაფებში ილუსტრაციებითა და პროგრამული ლისტინგებით.

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).



ნახ.2.3.

ნახ.2.4. ა-კომპილაციის და ბ - ტესტირების ეტაპები

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

ამ მუშა ფაილის ამუშავებით (კონსოლის რეჟიმში გამოიძახება იგი სახელით, მაგ., Myfirst), მიიღება საბოლოო

შედეგები (ნახ.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

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

C# Keywords

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	

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 „დავჯავშნულ“ სიტყვებში განსაკუთრებული ადგილი უჭირავს მონაცემთა ტიპების ოპერატორებს, რომელთაც მომდევნო პარაგრაფში განვიხილავთ.

2.4. C# ენის მონაცემთა ძირითადი ტიპები

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

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.

C#- ენის ცვლადებისა და კონსტანტების ტიპების ცხრილი

ცვლადთა ტიპები	**თანრიგი 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

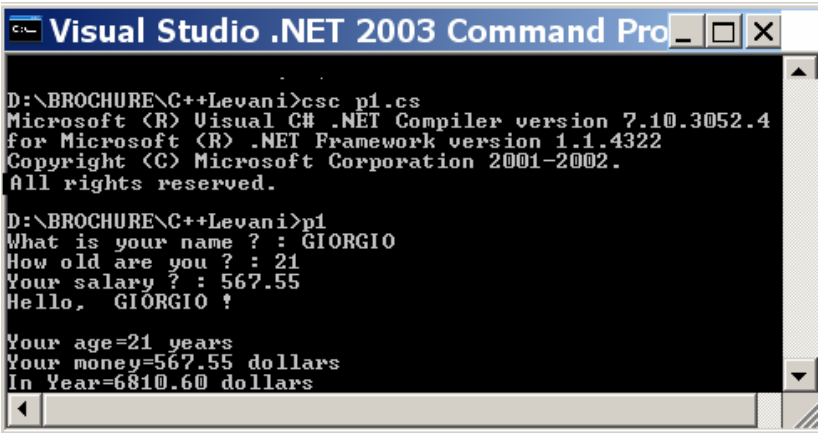
- decimal ტიპი (double-სგან განსხვავებით) გამოიყენება მეტი სიზუსტისათვის ფულად-საფინანსო ანგარიშებში და რიცხვს მიეწერება სუფიქსი M ან m. მაგ., decimal d = 37.15m;

C#-ენაში შემოტანილია ტიპი **string** , რომელიც Unicode-სიმბოლოთა სტრიქონია. მაგ., 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.1.8):

```
/*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.1.8.

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	გადაყვანის 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	გადაყვანის 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 აშ. 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

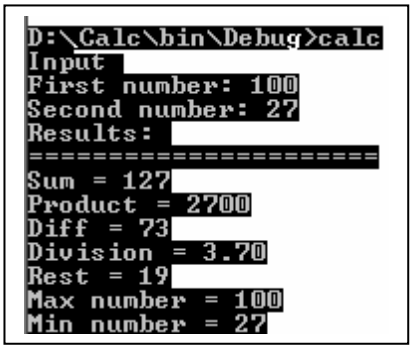
განიხილება მარტივი C#-პროგრამის ტექსტი, რომელიც შესასვლელზე თხოულობს ორი მთელი რიცხვის (x და y) შეტანას და შემდეგ ანგარიშობს მათ ჯამს, ნამრავლს, განსაზღვრავს მაქსიმალურ და მინიმალურს მათ შორის.

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 2.5 ნახაზზე.



ნახ.2.5.

III თავი. C#.NET -ის ობიექტ-ორიენტირებული დაპროგრამების ელემენტები

3.1. 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.

3.2. 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,

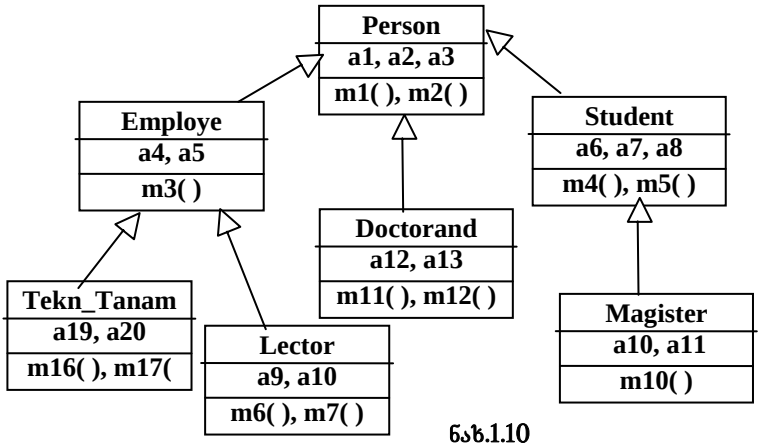
```
class Lector
{
  int LecID;
  string Gvari_S;
  string Status;
  float Xelfasi;
  char Tel[14];

  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 1.10 ნახაზზე მოცემულ იერარქიულ მოდელს.



ნახ.1.10

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 დოქტორანტი და ა.შ.

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.

3.3. 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).

3.4. 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 სპეკპირობების 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.

3.5. C# კოდის ორგანიზება სახელთა სივრცის დახმარებით

სახელთა სივრცე (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 საელთა ქვესივრცეს, მაგალითად: System.Console, System.Data, System.Exception და ა.შ.

საბაზო კლასთა მეთოდებზე მიმართვა ხდება სახელთა სივრცის 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 !
```

მოქმედების და ხილვადობის არეები. პროგრამაში სპეცბლოკების 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();
```

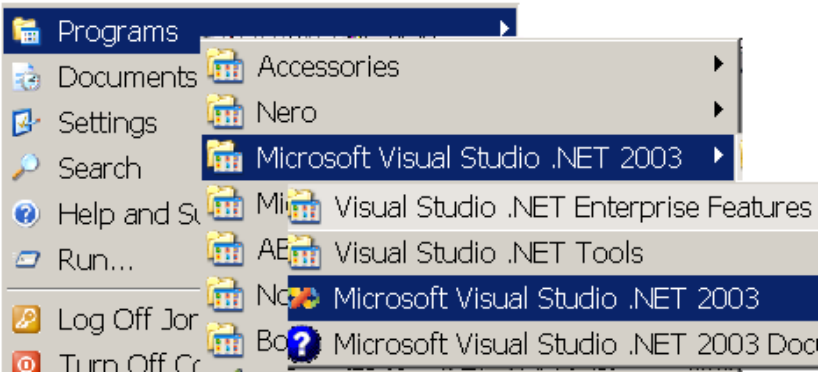
დასასრულ შეიძლება აღვნიშნოთ, რომ სახელთა სივრცე, მოქმედებისა და ხილვადობის არეები, ეს C#–ენის ის კომპონენტებია, რომელთა სწორი გამოყენებით შეიძლება ეფექტური პროგრამების შექმნა.

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.

IV თავი. C#-ის ვიზუალური კომპონენტები

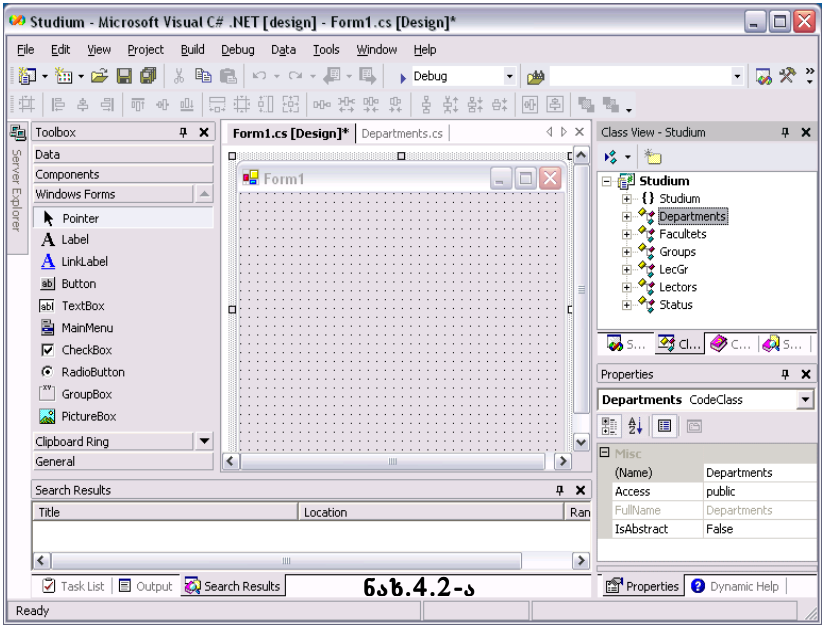
4.1. .NET-პლატფორმის სამუშაო გარემო

პროგრამული პაკეტი Visual Studio .NET გამოიძახება ვინდოუსის ოპერაციული სისტემის Start ღილაკიდან (ნახ.4.1).



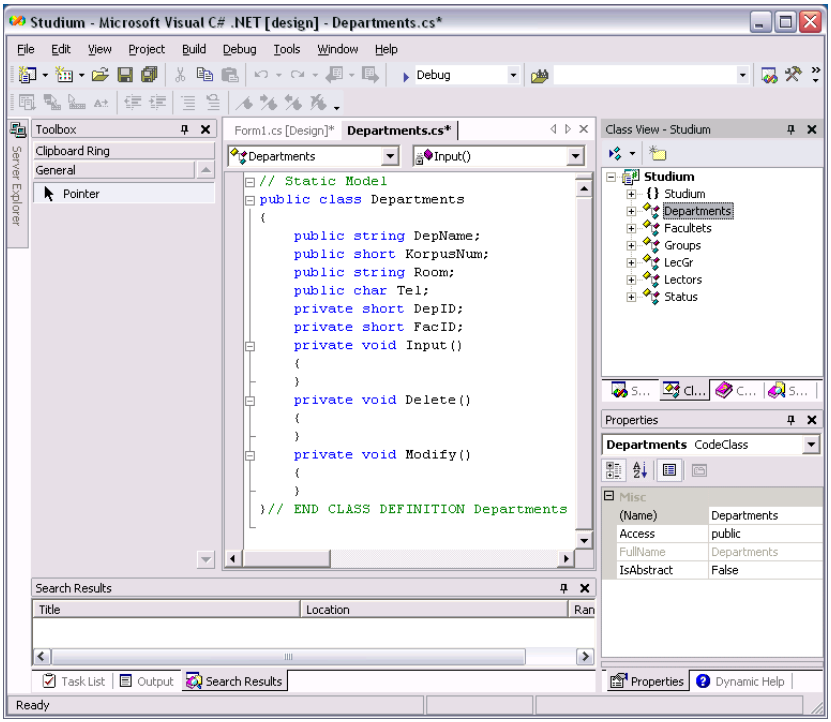
ნახ.4.1

4.2-ა ნახაზზე მოცემულია .NET პლატფორმის სამუშაო გარემო.



ნახ.4.2-ა

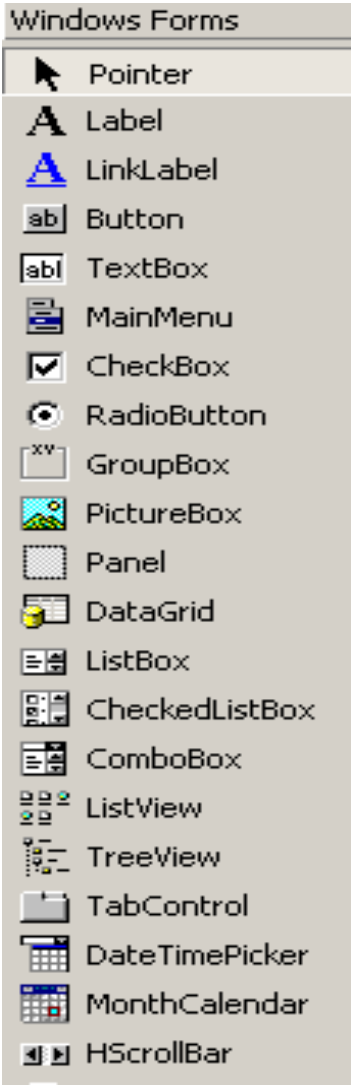
Solution Explorer განყოფილებაში Departments-კლასის არჩევით გამოვა ფორმაზე პროგრამის ტექსტი (ნახ.4.2-ბ).



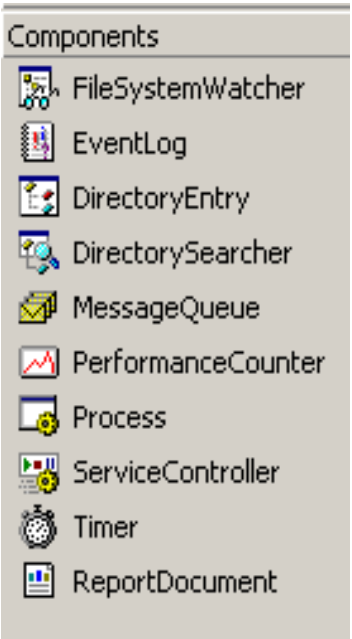
ნახ.4.2-ბ

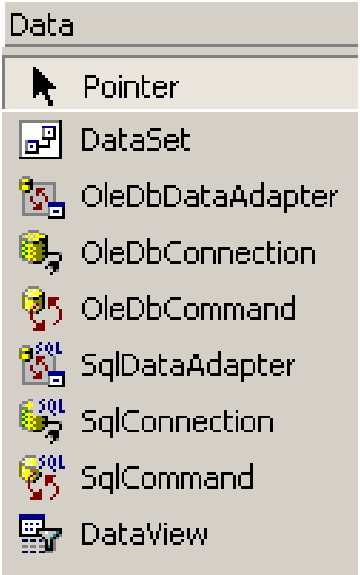
4.2-ა ნახაზზე ნაჩვენებ ფორმაზე ვიზუალური კომპონენტების გადმოსატანად და შესაბამისი ინტერფეისის ასაგებად იყენებენ 4.3-ა და ბ ნახაზზე ილუსტრირებულ ელემენტებს.

ამ კომპონენტების დეტალური აღწერა მოცემულია შემდეგ პარაგრაფებში.


Windows Forms
Pointer
Label
LinkLabel
Button
TextBox
MainMenu
CheckBox
RadioButton
GroupBox
PictureBox
Panel
DataGrid
ListBox
CheckedListBox
ComboBox
ListView
TreeView
TabControl
DateTimePicker
MonthCalendar
HScrollBar

ნახ.4.3-ა


Components
FileSystemWatcher
EventLog
DirectoryEntry
DirectorySearcher
MessageQueue
PerformanceCounter
Process
ServiceController
Timer
ReportDocument

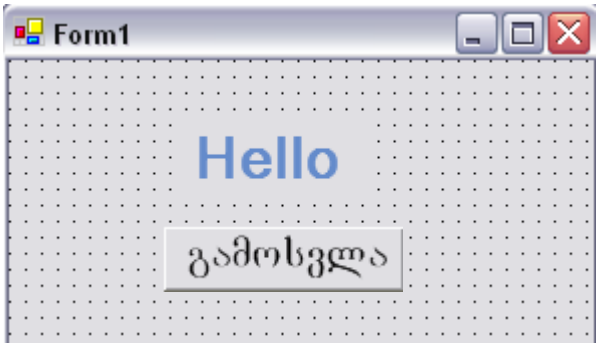

Data
Pointer
DataSet
OleDbDataAdapter
OleDbConnection
OleDbCommand
SqlDataAdapter
SqlConnection
SqlCommand
DataView

ნახ.4.3-ბ

4.2. VS C# ენის ვიზუალური კომპონენტები

A 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. 4.4

ab Button – **ღილაკი წარწერით**

კომპონენტს აქვს ღილაკის დანიშნულება. მას შეიძლება მიენიჭოს სხვადასხვა ბრძანებები. მაგალითად ღილაკზე თაგუს დაწკაპუნებით დაიხუროს ის ფორმა რომელზედაც მოთავსებულია ეს ღილაკი (ნახ.4.4).

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

– 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.4.5).

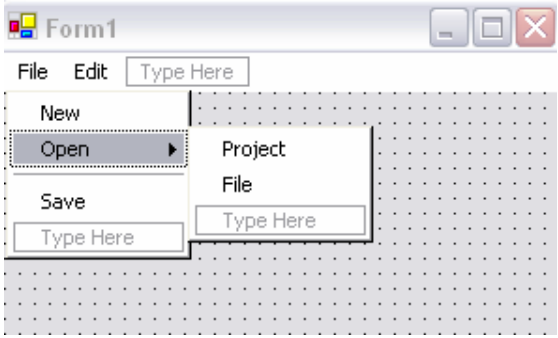
ნახ.4.5

ხოლო ღილაკის ბრძანებას ექნება შემდეგი სახე:

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

– 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.4.6).



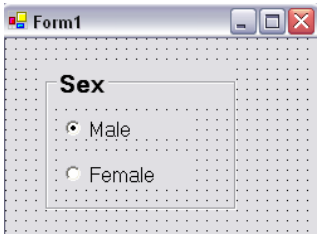
ნახ. 4.6

– მართვის ელემენტი ორი მდგომარეობით

– რადიო ღილაკი ორი მდგომარეობით

– ქმნის ფორმაზე ლოგიკურად დაკავშირებულ კომპონენტების კონტეინერს

კომპონენტში შეიძლება იკოს გაერთიანებული როგორც RadioButton-ები ასევე CheckBox-ები (ნახ.4.7).



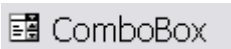
ნახ.4.7

– **გრაფიკული გამოსახულება**

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

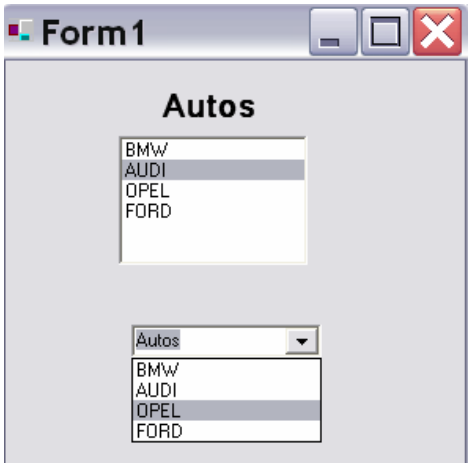
– **ტექსტური სტრიქონების სიის არე**

კომპონენტში ველების ჩამატება ხდება **Items** თვისებით.



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

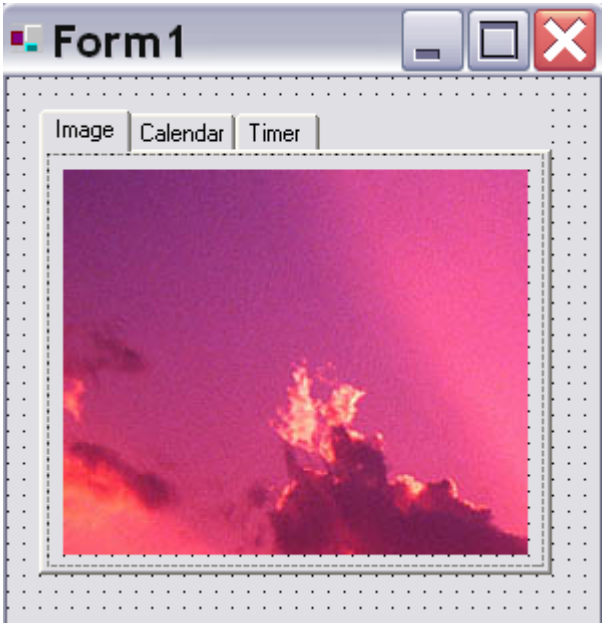
მონაცემებს ვამატებთ **Items** თვისებით. კომპონენტში, ჩამოშლამდე შეგვიძლია გამოვაჩინოთ საწყისი მონაცემი **Text** თვისებით ან დავტოვოთ ცარიელი (ნახ.4.8).



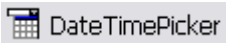
ნახ.4.8

– 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.4.9).

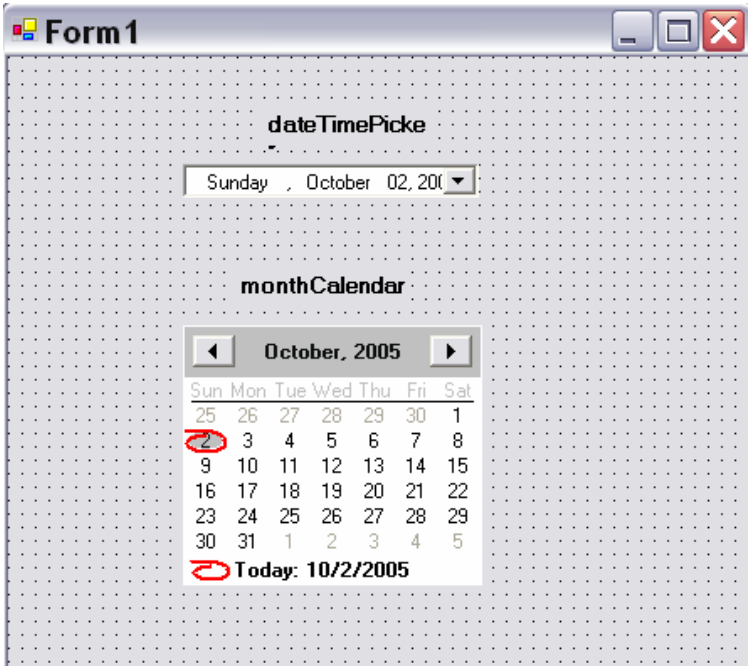


ნახ.4.9



–ქმნის თარიღისა და დროის რედაქტი-
რების არეს

– ქმნის თვის კალენდარს



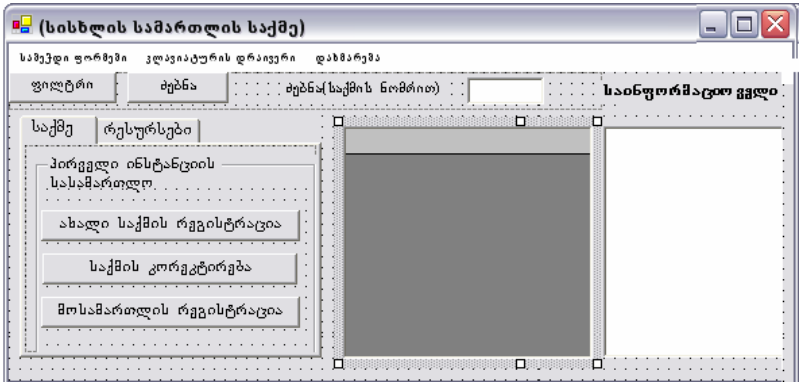
ნახ.4.10

4.3. C# ენის ვიზუალური კომპონენტებით ფორმების აგების მაგალითები

კონკრეტული საპრობლემო სფეროს მართვის ავტომატი-ზებული 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.4.11-4.13).

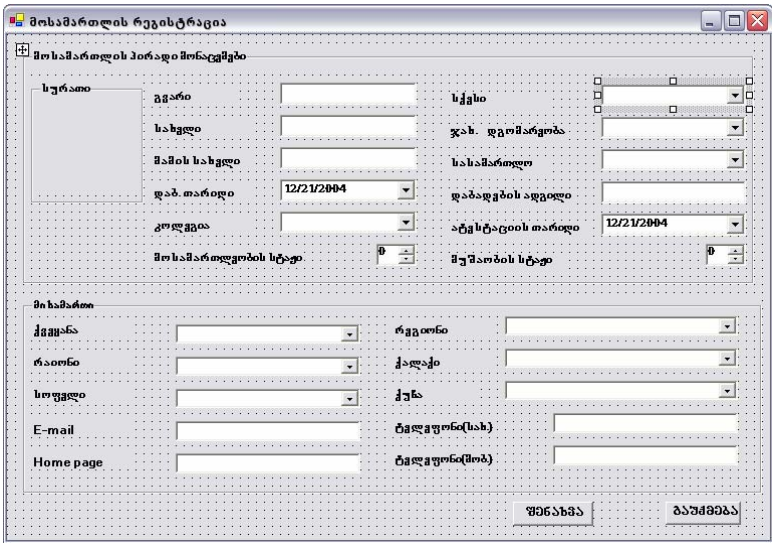


ნახ.4.11

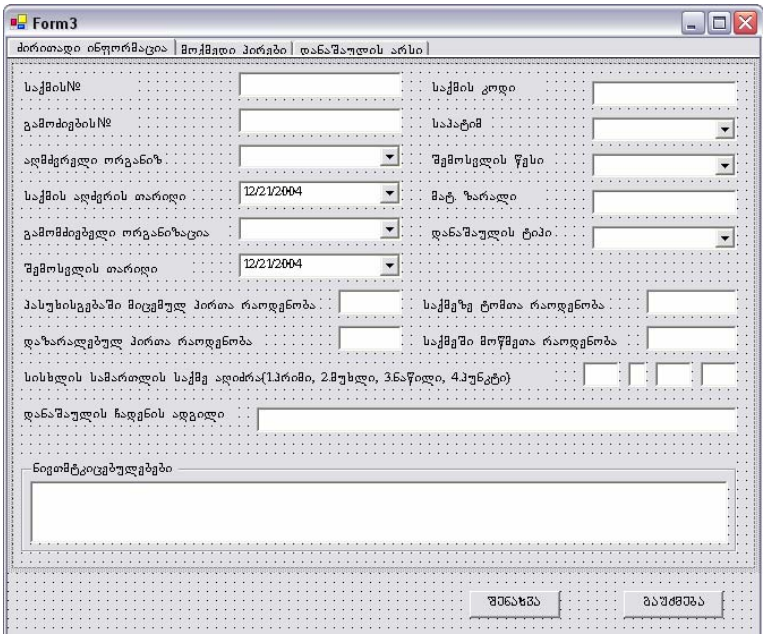
„მოსამართლის რეგისტრაციის“ ღილაკით გამოიტანება ახალი ფანჯარა, რომელიც 4.12 ნახაზზეა ნაჩვენები და მასში შესაძლებელია ინფორმაციის წაკითხვა, შეტანა ან კორექტირება.

„ახალი საქმის რეგისტრაციის“ ღილაკით კი შეიძლება 4.13 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.4.12



ნახ.4.13

```
//---------------------- 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 4.14 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
                                          შესრულებისა და დამუშავებისას
using System.Windows.Forms;            // Windows-დანართის ფორმების შექმნის საშუალებანი
using System.Data;                     // 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.4.14

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.

V თავი. ADO.NET პაკეტი და ინტერფეისების აგება C#.NET ინსტრუმენტით

5.1. MS 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].

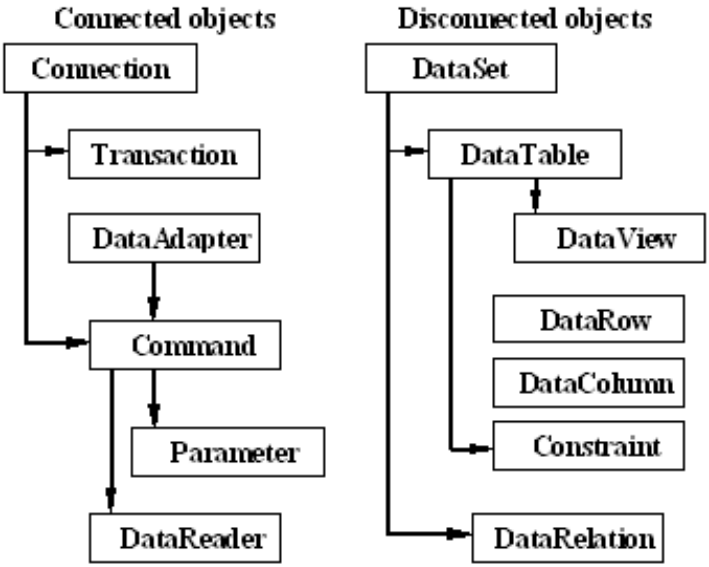
Visual Studio.NET პაკეტში ორიგინალურად გამოიყენება ობიექტ-ორიენტირებული დაპროგრამების ენები C#, Visual Basic, Java, Viasual C++, აგრეთვე XML ენა Web აპლიკაციების ასაგებად.

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.

5.2. ADO.NET-ის ობიექტური მოდელის სტრუქტურა

სისტემის ობიექტური მოდელი ორი ნაწილისგან შედგება: მარცხენა - მიერთებადი ობიექტები (connected objects) და მარჯვენა - განცალკევებადი ობიექტები (disconnected obects). 5.1 ნახაზზე ნაჩვენებია ADO.NET ობიექტური მოდელის შემადგენელი კლასები, რომელთა დანიშნულებასაც მოკლედ შევეხებით ამ პარაგრაფში.

მიერთებადი ობიექტები უშუალოდ უკავშირდება მონაცემთა ბაზას,



ნახ.4.1. ADO.NET-ის ობიექტების იერარქია

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

- Command ობიექტები გამოიყენება 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 გამოიყენება 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 ჩანაწერერბის (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“.

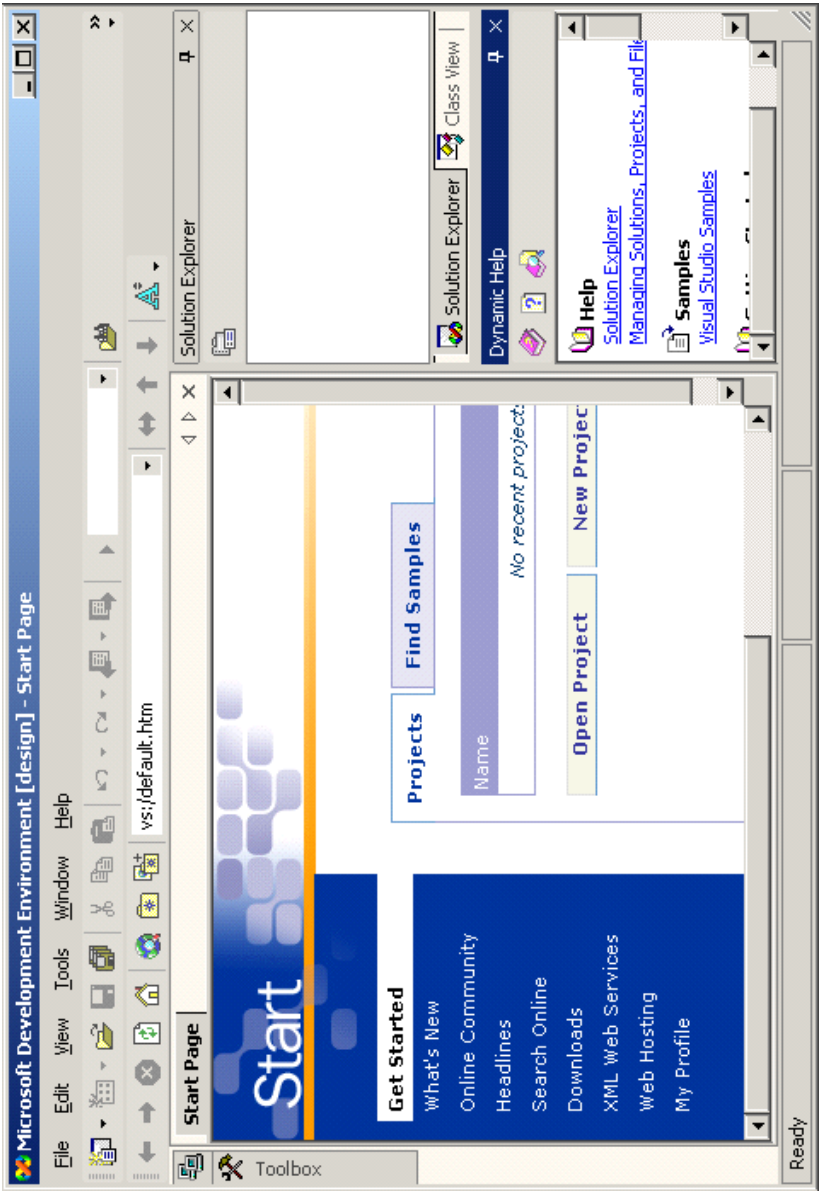
- ობიექტი DataView გამოიყენება მოთხოვნის საფუძველზე მიღებული რეალური ცხრილის მონაცემების წარმოსადგენად (დასათვალიერებლად) სხვადასხვა ფორმით. მაგალითად, ცხრილის სტრიქონების დალაგება რომელიმე ველის მოწესრიგებით (Sort-თვისება), ან ჩანაწერების ამორჩევა რაიმე კრიტერიუმით (Filter-თვისება). რაც მთავარია, ეს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

5.3. ინტერფეისის დამუშავების სადემონსტრაციო მაგალითი

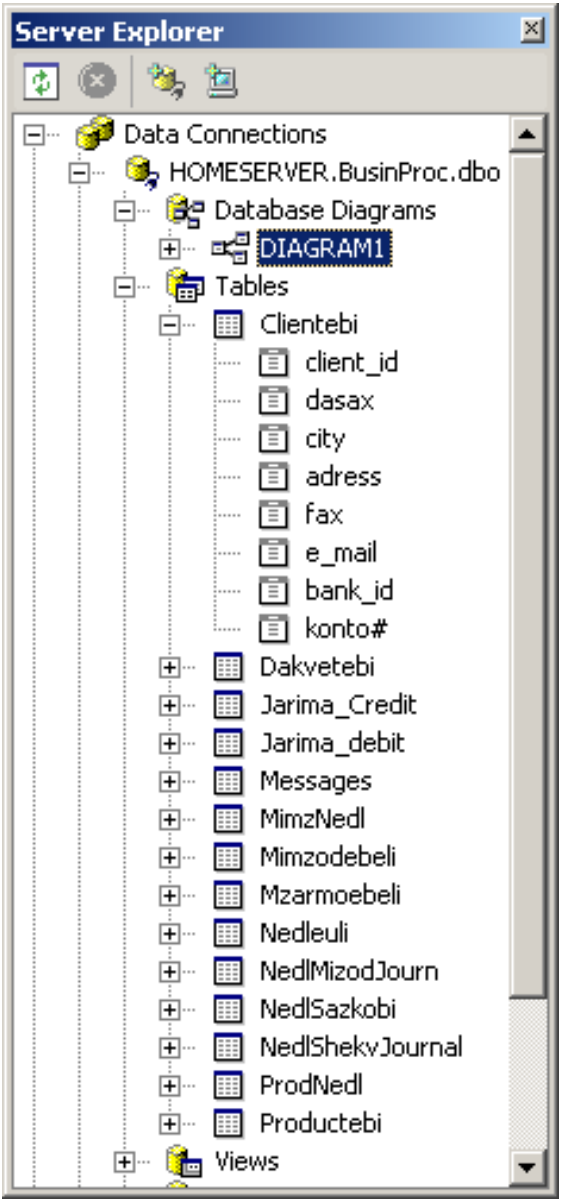
ძირითადი ამოცანა, რომელსაც ჩვენ აქ განვიხილავთ, არის ADO.NET პროგრამული პაკეტის გამოყენებით მომხმარებელთა სამუშაო ინტერფეისის დამუშავების სადემონსტრაციო მაგალითის აგება. ამასთანავე, მონაცემთა ბაზების სახით უნდა გამოვიყენოთ SQL Server პაკეტით დამუშავებული ცხრილები.

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

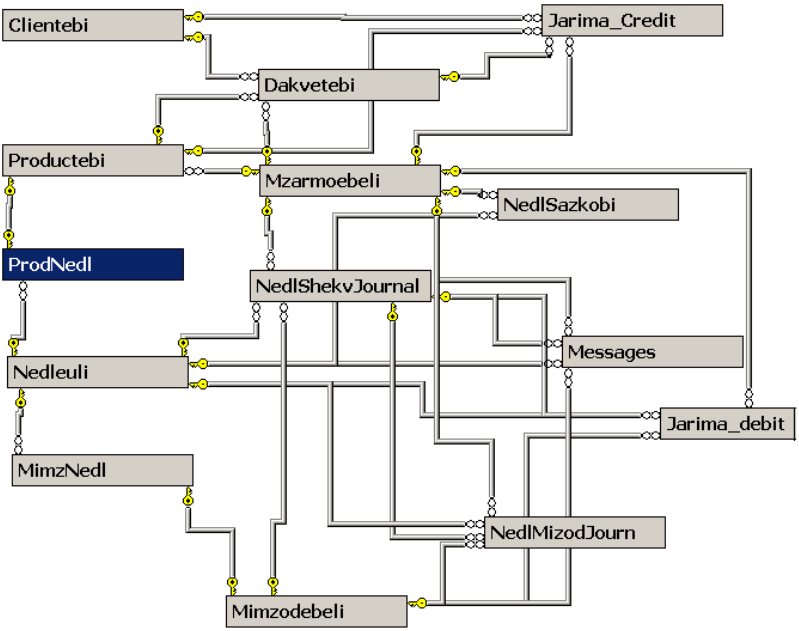
პირველ რიგში შევამოწმოთ მონაცემთა ბაზასთან კავშირი. ამისათვის მენიუდან View | Server Explorer-ით გამოვიტანოთ ფანჯარა (ნახ.5.3) და ავირჩიოთ HOMESERVER.BusinProc.dbo. გამოჩნდება SQL Server-ის ბაზა, ცხრილები და ველები. DIAGRAM1-ის არჩევით მივიღებთ ცხრილებს კავშირებით, რომელიც იდენტურია SQL Server-ში ჩვენს მიერ შექმნილი ბაზისა. ამგვარად თავსებადობა კარგადაა რეალიზებული. 5.4 ნახაზზე ნაჩვენებია ცხრილთა კავშირების ფრაგმენტი, მხოლოდ მათი სათაურების ჩვენებით.



ნახ. 5.2. ADO.NET სასტარტო ფანჯარა



ნახ.5.3. BusinProc.dbo მონცემთა ბაზასთან დაკავშირება



ნახ.5.4. SQL Server მონცემთა ბაზის ასახვა ADO.NET-ში

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

File | New | Project

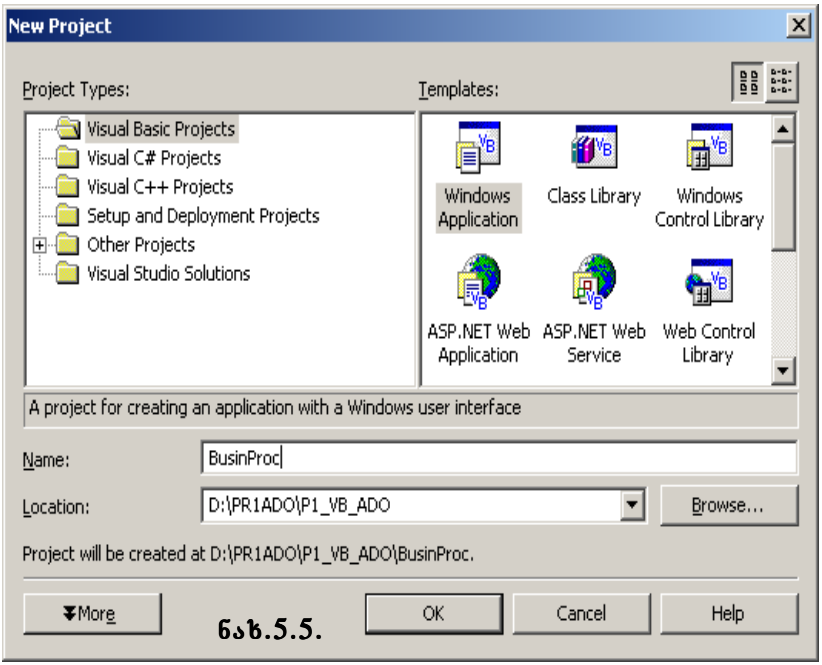
კომბინაცია და მივიღებთ 5.5 ნახაზზე ნაჩვენებ ფანჯარას. აქ შევარჩევთ სახელს (Name), ვინჩესტერზე შესანახ გზას და პაპკის სახელს (Location). Template-ში ავირჩევთ Windows Application.

შემდეგ მენიუდან ვირჩევთ:

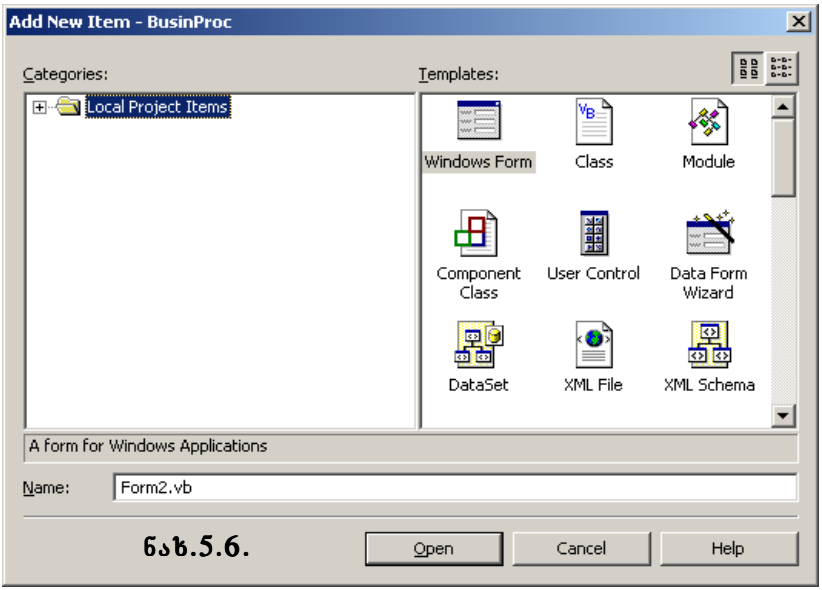
File | New Item

და მივიღებთ 5.6 ნახაზზე მოცემულ ფანჯარას.

ავირჩიოთ Data Form Wizard და ეკრანზე გამოჩნდება ახალი ფორმა (ნახ.5.7). აქ ღილაკით Next გადავალთ შემდეგ ბიჯზე (ნახ.5.8). შევარჩიოთ მონაცემთა ბაზის სახელი და Next.


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

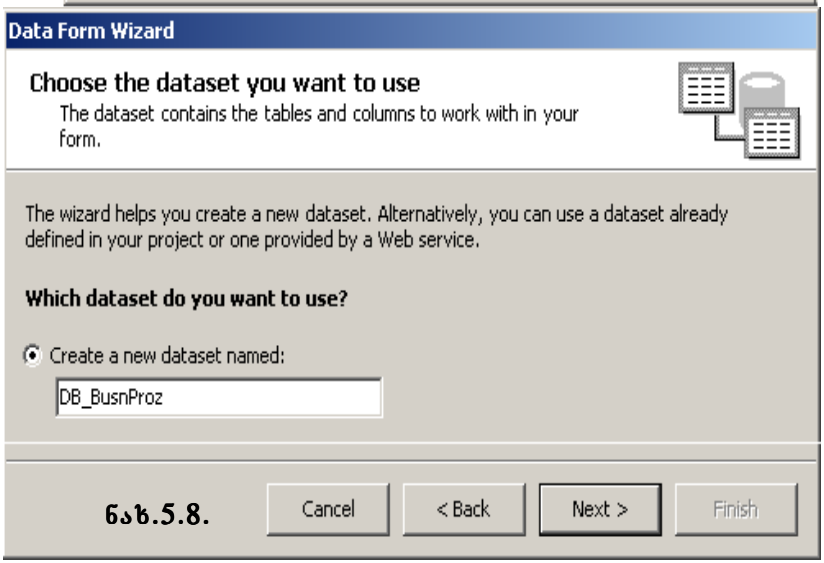
ნახ.5.5.


Add New Item - BusinProc
Categories:
Templates:
Local Project Items
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.5.6.

DataForm1
Data Form Wizard
Welcome to the DataForm Wizard
This wizard will create a new form with data-bound controls to display information from the dataset you specify.
Click Next to continue.
Cancel
< Back
Next >

Նկ.5.7.


Data Form Wizard
Choose the dataset you want to use
The dataset contains the tables and columns to work with in your form.
The wizard helps you create a new dataset. Alternatively, you can use a dataset already defined in your project or one provided by a Web service.
Which dataset do you want to use?
Create a new dataset named:
DB_BusnProz
Cancel
< Back
Next >
Finish

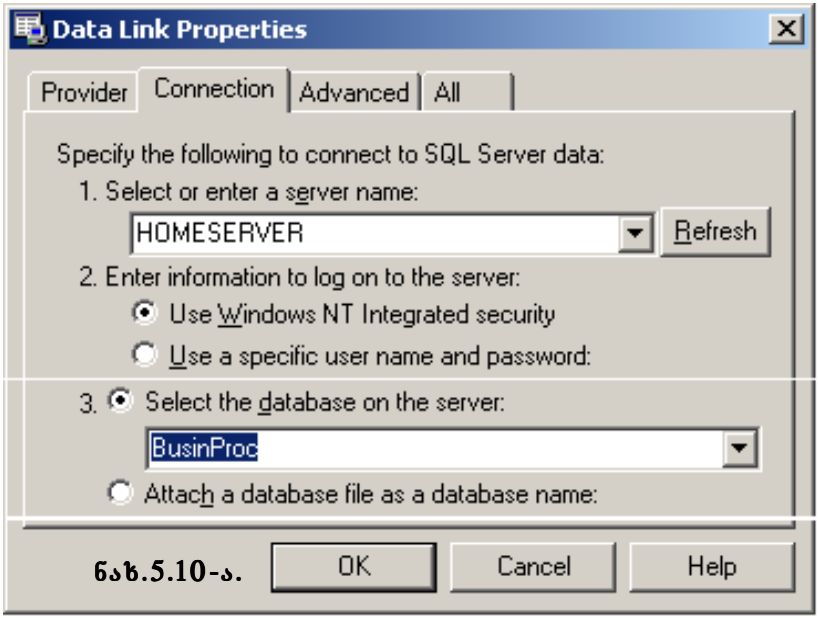
Նկ.5.8.

მონაცემთა ბაზასთან კავშირის დასამყარებლად 5.9 ნახაზის შესაბამისი ფანჯრიდან New Connection-ში შევარჩევთ სახელს.

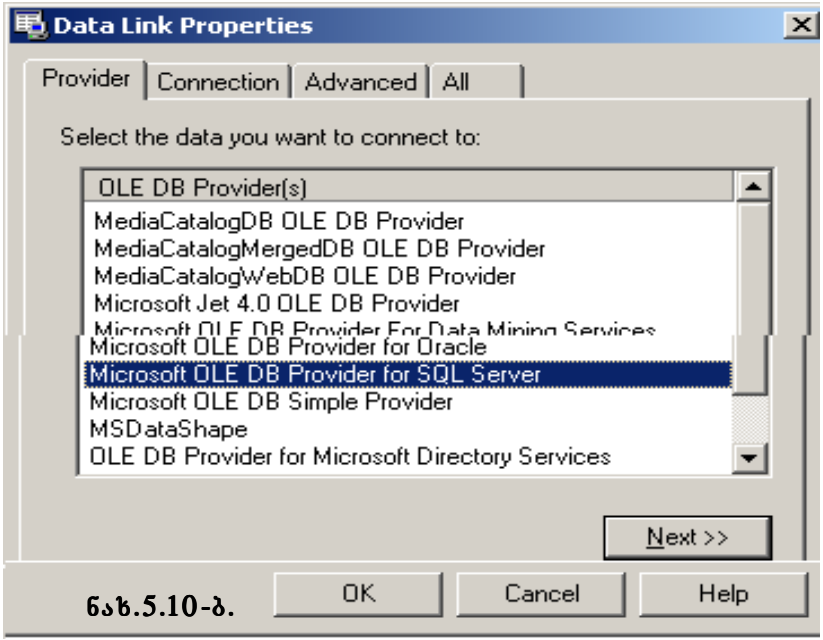
ამისათვის უნდა განვიხილოთ მომდევნო სამი ფანჯარა, შესაბამისად ნახ.5.10-ა,ბ,გ.

როგორც ვხედავთ, ჩვენ შევარჩიეთ SQL Server-ის მონაცემთა პროვაიდერი.

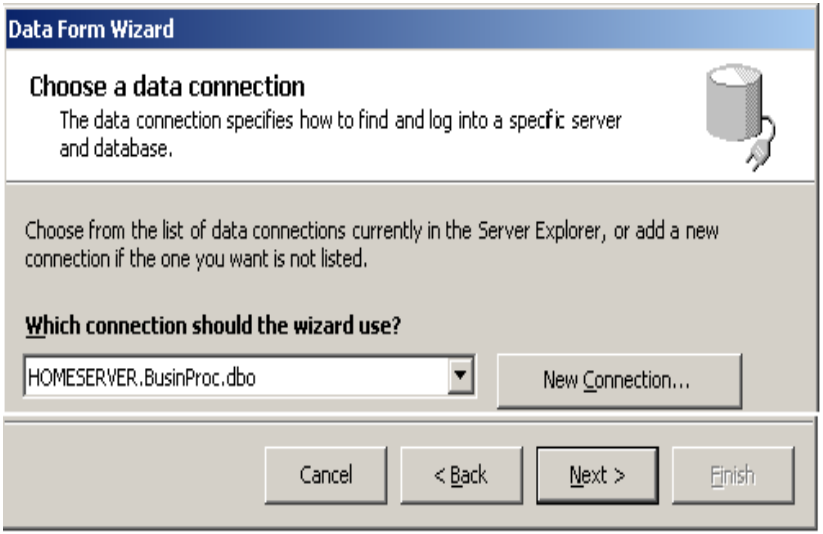
ნახ.5.9



ნახ.5.10-ა.



ნახ.5.10-ბ.


Data Form Wizard
Choose a data connection
The data connection specifies how to find and log into a specific server and database.
Choose from the list of data connections currently in the Server Explorer, or add a new connection if the one you want is not listed.
Which connection should the wizard use?
HOMESERVER.BusinProc.dbo
New Connection...
Cancel
< Back
Next >
Finish

ნახ.5.10-ბ.

5.4. მონაცემთა ბაზის ცხრილების გამოტანა C# - ენისათვის

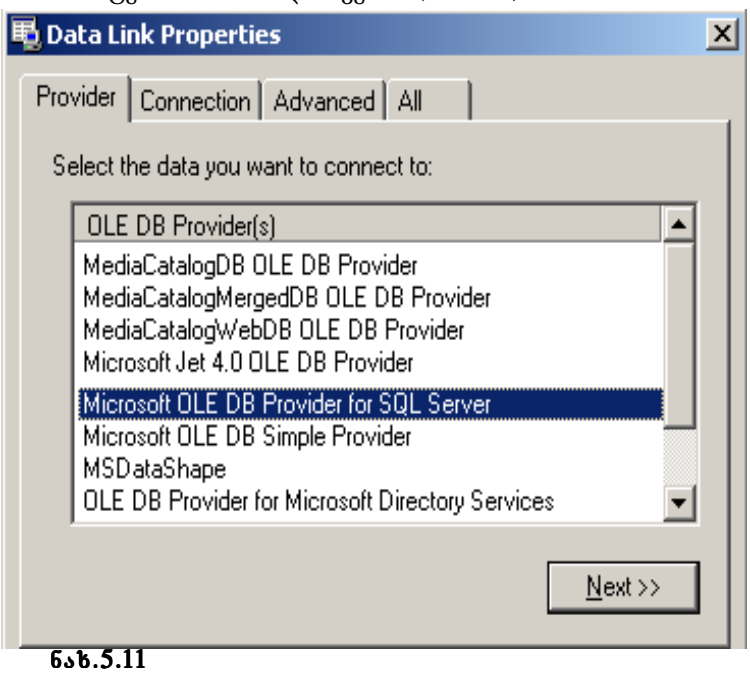
განვიხილოთ .NET პლატფორმაზე C# ენის სამუშაო გარემო, სადაც მომხმარებელთა ინტერფეისის აგების შემდეგ საჭირო ხდება მონაცემთა ბაზიდან ინფორმაციის გამოტანა.

დავუშვათ, რომ არსებობს რომელიმე პროგრამულ პაკეტში აწყობილი მონაცემთა ბაზა, მაგალიღთად, MsAccess, MsSQL_Server ან სხვ.

მოხმარებლის ინტერფეისის დასაპროგრამებლად და მონაცემთა ამოსაღებად მითითებული ბაზიდან, საჭიროა პროგრამული პაკეტის ADO.NET გამოყენება. ამისათვის .NET-ში მენიუდან ავირჩიოთ:

View | Server Explorer.

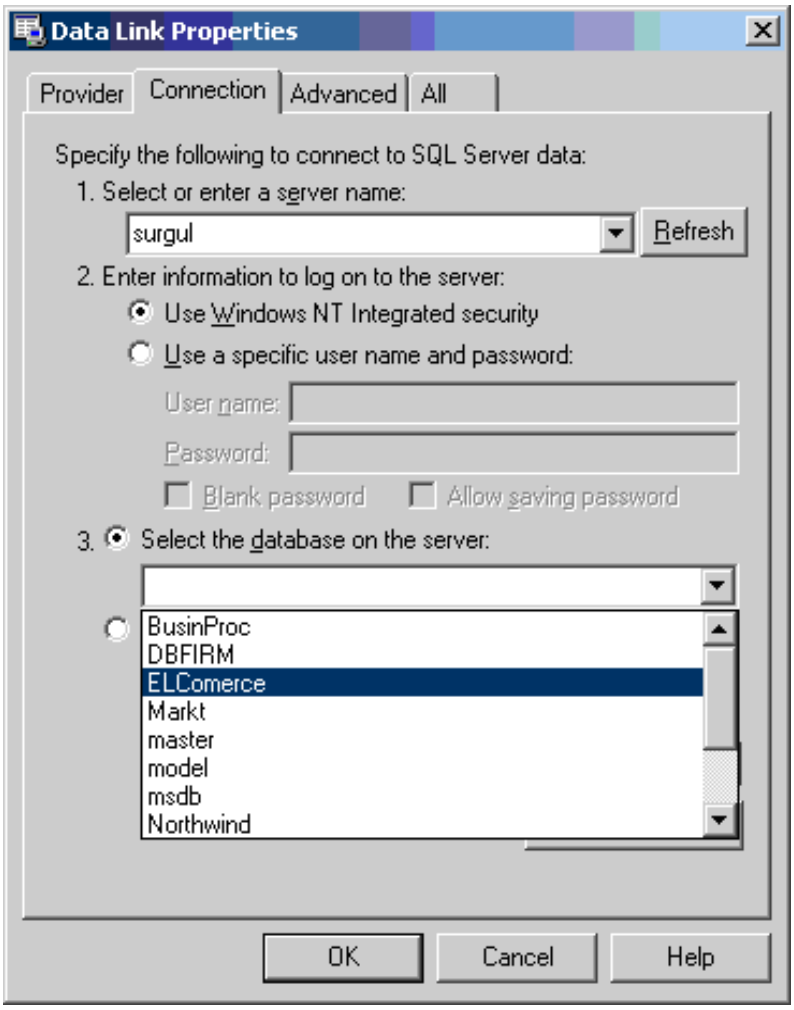
ეკრანის მარცხენა ზედა ნაწილში Data Connections-ზე მაოსის მარჯვენა ღილაკით ავირჩიოთ Add Connection. გამოჩნდება დამხმარე ფანჯარა Data Link Properties, რომელშიც უნდა შევირჩიოთ ჩვენთვის საჭირო მონაცემთა ბაზის დრაივერი (ნახ.5.11).



ნახ.5.11

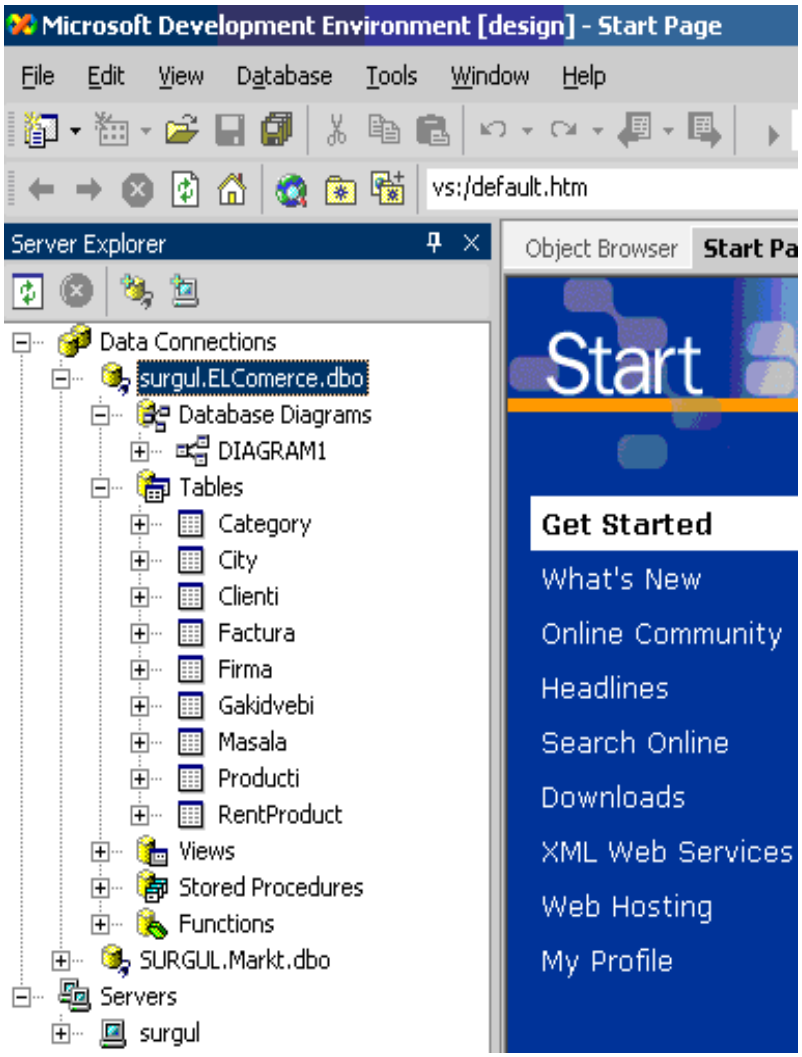
აქ ნაჩვენებია MsSQLServer-ის დრაივერის მაგალითი. თუ საჭიროა Ms Access, მაშინ ავირჩევთ Microsoft Jet 4.0 OLE DB-ს.

დრაივერის არჩევის შემდეგ Connection გვერდზე (ნახ.5.12) შევირჩევთ სერვერის სახელს (მაგ.,surgul),გადავრთავთ UseWindows NT security და ავირჩევთ მონაცემთა ბაზას (მაგალითად, ElComerce).

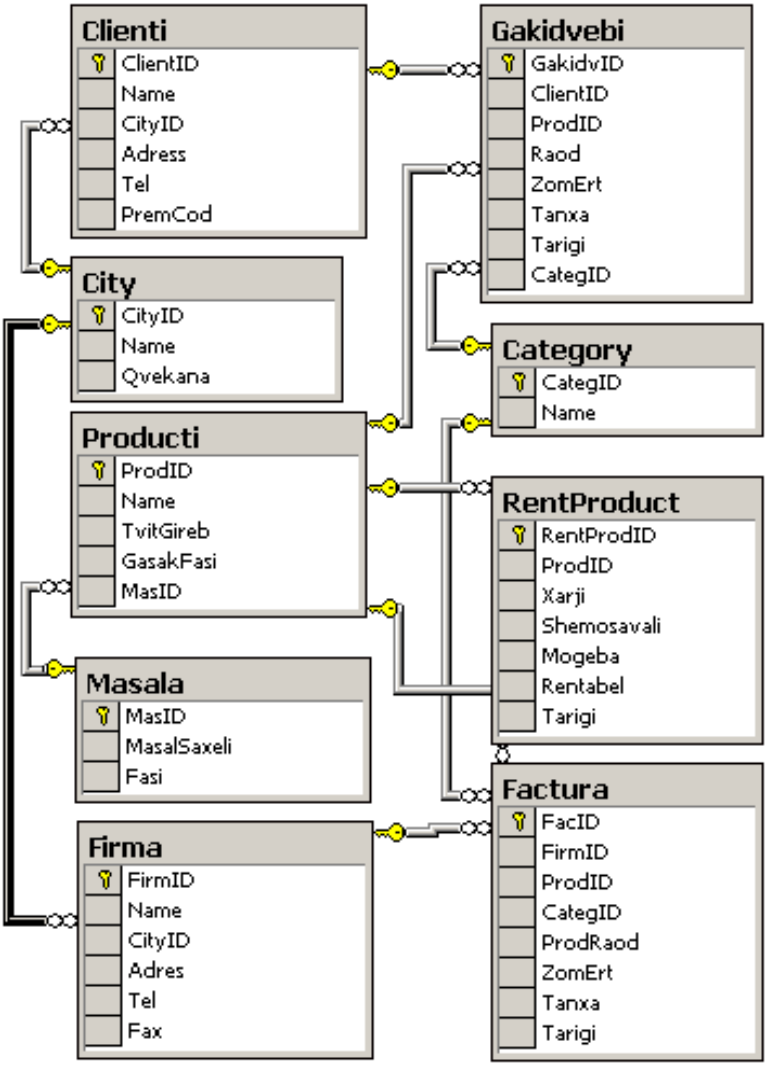


ნახ.5.12

მივიღებთ 5.13 ნახაზზე მოცემულ კადრს, რომელზედაც ჩანს MsSQLServer-ში აგებულ E_Comerce მონაცემთა ბაზის ცხრილები (Tables) და დიაგრამა (DIAGRAM1). ამგვარად, დაკავშირება ბაზასთან განხორციელდა.



ნახ.5.13


Clienti
ClientID
Name
CityID
Adress
Tel
PremCod
Gakidvebi
GakidvID
ClientID
ProdID
Raod
ZomErt
Tanxa
Tarigi
CategID
City
CityID
Name
Qvekana
Category
CategID
Name
Producti
ProdID
Name
TvitGireb
GasakFasi
MasID
RentProduct
RentProdID
ProdID
Xarji
Shemosavali
Mogeba
Rentabel
Tarigi
Masala
MasID
MasalSaxeli
Fasi
Factura
FacID
FirmID
ProdID
CategID
ProdRaod
ZomErt
Tanxa
Tarigi
Firma
FirmID
Name
CityID
Adres
Tel
Fax

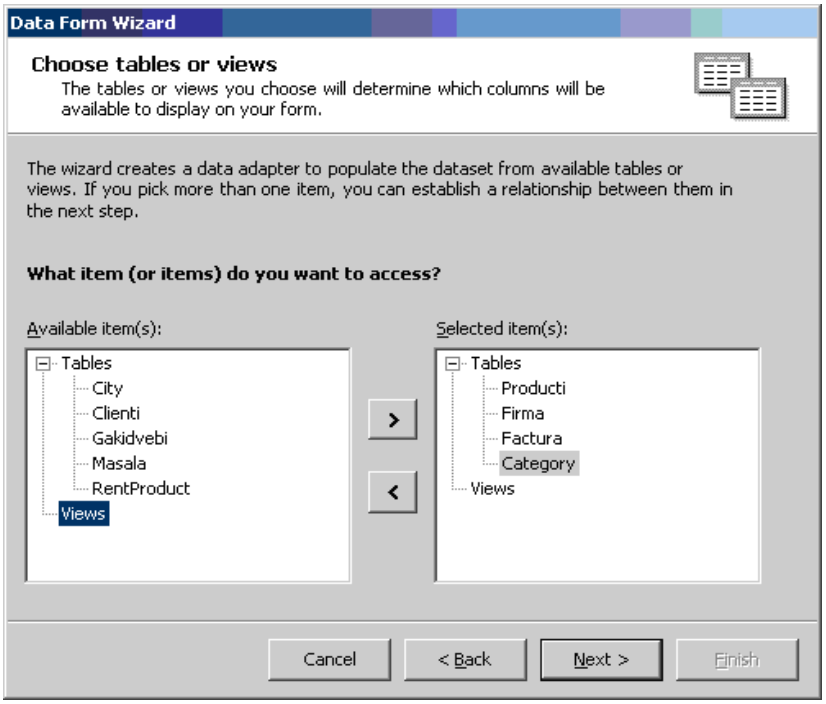
ნახ.5.14

თუ გავხსნით DIAGRAM1-ს, მაშინ გამოჩნდება ADO.NET-ში ასახული ElComerce-ბაზის ცხრილთაშორის კავშირების სქემა (ნახ.5.14).

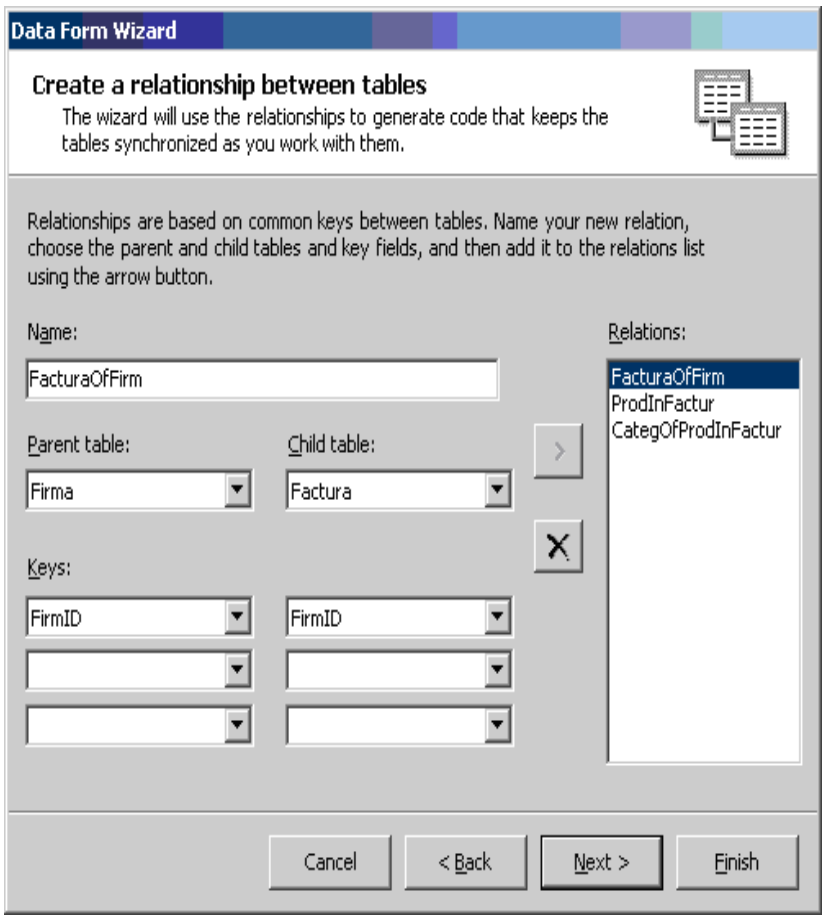
შემდეგ ეტაპზე მენიუდან ვირჩევთ:

File | Add New Item | Data Form Wizard.

ამით შესაძლებელია ფორმის აგება და მონაცემთა გამოტანა ოსტატი-პროგრამით (Wizard). ეს პროგრამა თვითონ წარმართავს დიალოგს, რაც აადვილებს მუშაობას. აქ მთავარია ჩვენ სწორად შევირჩიოთ ცხრილები (ნახ.5.15) და ცხრილთაშორისი კავშირები (ნახ.5.16).

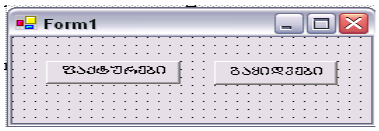


ნახ.5.15



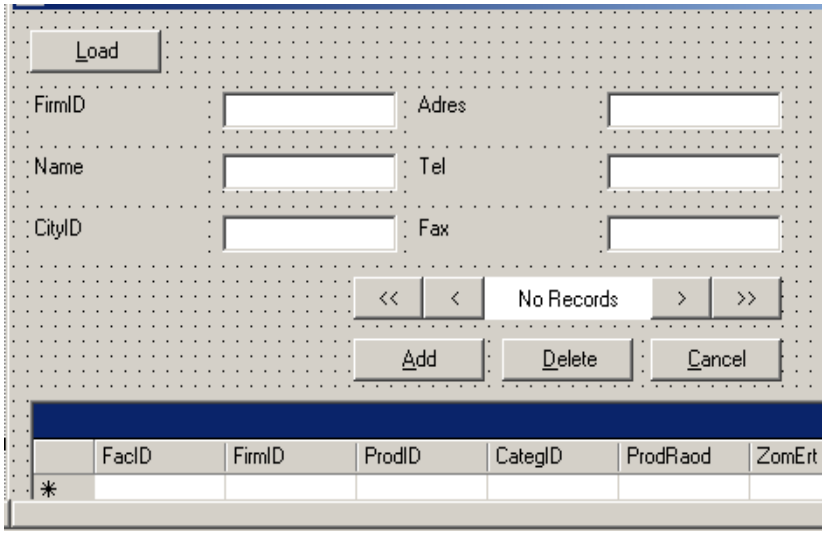
ნახ.5.16. ცხრილთაშორისი კავშირის აგება

ფორმაზე Form1 ინსტრუმენტების პანელიდან გადმოვიტანოთ ორი ღილაკი (button1, button2). მოვნიშნოთ პირველი ღილაკი და Properties-ში შევცვალოთ მისი თვისებები. მაგ., ქართული შრიფტი (Font|LitMtavrPS-ით) და წარწერა „ფაქტურები“ (Text). 5.17 ნახაზზე ეს შემთხვევაა ნაჩვენები.



ნახ.5.17

5.18 ნახაზზე მოცემულია მეორე ფორმა DataForm1, რომელიც ფირმისა და ფაქტურების ცარიელი ცხრილებია.

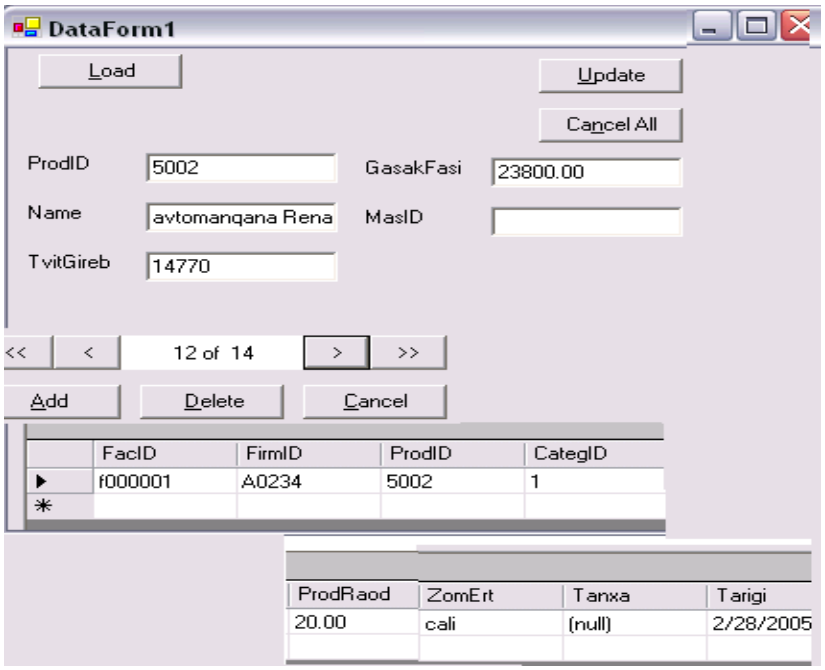


ნახ.5.18

პირველი ფორმიდან რომ გამოვიძახოთ მეორე ფორმა, საჭიროა ღილაკზე მაოსით 2-ჯერ დავაწკაპუნოთ და C# კოდის ტექსტში, სადაც შეჩერებულია კურსორი (button1_Click) ჩავწეროთ ხელით შემდეგი სტრიქონები:

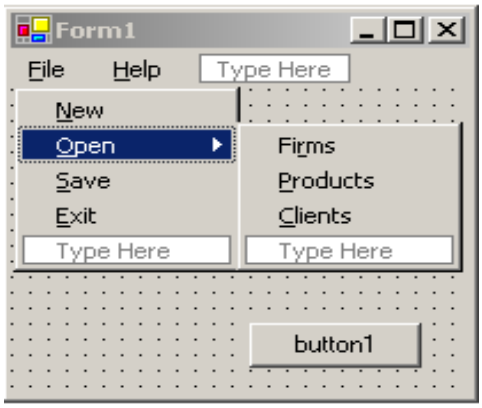
```
private void button1_Click(object sender, System.EventArgs e)
    { // შესატანი ტექსტის ორი სტრიქონი:
      DataForm1 ob=new DataForm1( );
       ob.Show( );
     }
```

ამის შემდეგ Standard-პანელის Start-ღილაკით შევასრულებთ კოდის კომპილირებას და ეკრანზე გამოვა 1-ელი ფორმა, რომლის „ფაქტურის“ ღილაკის არჩევით მივიღებთ მე-2 ფორმას (შეუვსებელი). აქ საჭიროა Load-ის არჩევა და მონაცემები განლაგდება ფორმაზე (ნახ.5.19).


DataForm1
Load
Update
Cancel All
ProdID
5002
GasakFasi
23800.00
Name
avtomanqana Rena
MasID
TvitGireb
14770
<<
<
12 of 14
>
>>
Add
Delete
Cancel
FacID
FirmID
ProdID
CategID
f000001
A0234
5002
1
ProdRaod
ZomErt
Tanxa
Tarigi
20.00
cali
(null)
2/28/2005

ნახ.5.19

ღილაკის მსგავსად ფორმაზე შესაძლებელია სხვა ვიზუალური კომპონენტების გადმოტანაც. მაგალითად, მთავარი მენიუს შექმნა (ნახ.5.20).



ნახ.5.20

ლიტერატურა

1. რეისიგი გ., სურგულაძე გ., გულუა დ. გიზუალური ობიექტ-ორიენტირებული დაპროგრამების მეთოდები. თბ., სტუ, 2002.

2. სურგულაძე გ., შონია ო., ყვავაძე ლ. მონაცემთა ბაზების მართვის სისტემები: Ms Access, SQL Server, InterBase, Oracle, Corba. თბ., სტუ, 2004.

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

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

5. სურგულაძე გ., დაპროგრამების გიზუალური მეთოდები და ინსტრუმენტები (UML, MsVisio, C++Builder). თბ.,სტუ, 2005.

6. სურგულაძე გ., დოლიძე თ., ყვავაძე ლ. კომპონენტურ-ვიზუალური დაპროგრამება. სტუ. თბ., 2006.

7. Bothe K., Surguladze G. OO-Modellierung mit UML, Tb., 2002.

8. სურგულაძე გ., პეტრიაშვილი ლ. მონაცემთა საცავის აგების ტექნოლოგია ინტერნეტული ბიზნესის სისტემებისათვის. თბ., სტუ, 2005.