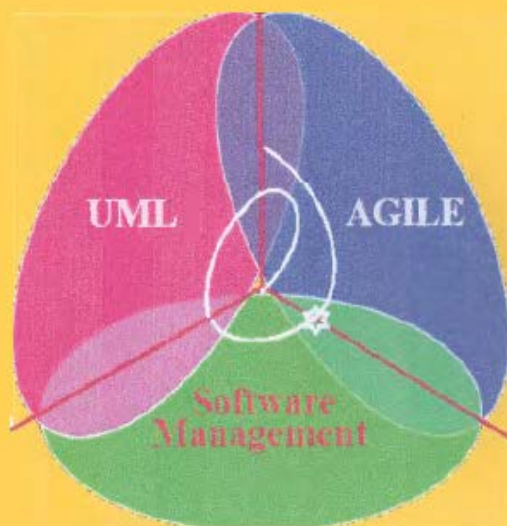


გია სურგულაძე, ეკატერინე თურქია

პროგრამული სისტემების მენეჯმენტი

(ლაბორატორიული პრაქტიკუმი)



„IT-კონსალტინგის ცენტრი“

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, ეკატერინე თურქია

პროგრამული სისტემების მენეჯმენტი

(ლაბორატორიული პრაქტიკუმი)



დამტკიცებულია:
სტუ-ს „IT-კონსალტინგის“
სამეცნიერო ცენტრის
სარედაქციო-საგამომცემლო
კოლეგიის მიერ

თბილისი
2016

უაკ 004.5

განიხილება მართვის საინფორმაციო სისტემების პროგრამული უზრუნველყოფის ობიექტორიენტირებული ანალიზის, დაპროექტების, დეველოპმენტის, ტესტირების, დანერგვისა და რეინჟინერინგის პრაქტიკული ამოცანების პროგრამული რეალიზაციის საკითხები.

დამხმარე სახელმძღვანელოში შემოთავაზებულია „პროგრამული სისტემების მენეჯმენტის“ დისციპლინის 15 ლაბორატორიული ამოცანა. გამოყენებულია MsVisual Studio.NET Framework 4.5 ინტეგრირებული სამუშაო გარემო.

წიგნი ორიენტირებულია ინფორმატიკის ფაკულტეტის მართვის საინფორმაციო სისტემების სპეციალობის ბაკალავრიატის და მაგისტრატურის სტუდენტებზე, აგრეთვე პროგრამული უზრუნველყოფის მენეჯმენტის საკითხებით დაინტერესებულ მკითხველზე.

რეცენზენტები:

- პროფ. გიორგი გოგიჩაიშვილი (საქ. მეცნიერებათა ეროვნული აკადემიის წევრ-კორესპოდენტი)
- პროფ. რომან სამხარაძე

პროფ. გ. სურგულაძის რედაქციით

© სტუ-ს „IT-კონსალტინგის სამეცნიერო ცენტრი“, 2016

ISBN 978-9941-0-8558-1

ყველა უფლება დაცულია, ამ წიგნის არც ერთი ნაწილი (იქნება ეს ტექსტი, ფოტო, ილუსტრაცია თუ სხვა) არანაირი ფორმით და საშუალებით (იქნება ეს ელექტრონული თუ მექანიკური), არ შეიძლება გამოყენებულ იქნას გამომცემლის წერილობითი ნებართვის გარეშე.

რედკოლეგია:

ა. ფრანგიშვილი (თავმჯდომარე), მ. ახოზაძე, ზ. ბაიაშვილი,
ზ. ბოსიკაშვილი, ზ. გასიტაშვილი, გ. გოგიჩაიშვილი, მ. თევდორაძე,
ე. თურქია, ლ. იმნაიშვილი, თ. კაიშაური, რ. კაკუბავა,
ჰ. მელაძე, თ. ლომინაძე, ნ. ლომინაძე, თ. ოზგაძე, რ. სამხარაძე,
გ. სურგულაძე, გ. ჩაჩანიძე, ა. ცინცაძე, გ. ძიძიგური, ზ. წვერაიძე

ნაშრომი ეძღვნება:

საქართველოს ტექნიკური უნივერსიტეტის
„მართვის ავტომატიზებული სისტემების“

კათედრის დაარსების 45-ე წლისთავს.

(20 მაისი: 1971-2016 წწ.)

ავტორთა შესახებ:

გია სურგულაძე - სტუ-ს პროფესორი, ტექნიკის მეცნიერებათა დოქტორი, გაეროსთან არსებული „ინფორმატიზაციის საერთაშორისო აკადემიის (IIA)“ ნამდვილი წევრი, სტუ-ს „IT-კონსალტინგის სამეცნიერო ცენტრის“ ხელმძღვანელი, გერმანიის DAAD-ის გრანტის მრავალგზის მფლობელი, ბერლინის ჰუმბოლდტის, ნიურნბერგ-ერლანგენის და სხვა უნივერსიტეტების მიწვეული პროფესორი 1991-2014 წწ. 300-ზე მეტი სამეცნიერო ნაშრომის ავტორი, მათ შორის 63 წიგნის და 47 ელ-სახელმძღვანელოსი მართვის საინფორმაციო სისტემების ინჟინერინგის სფეროში.

ეკატერინე თურქია - სტუ-ს პროფესორი, ტექნიკის მეცნიერებათა კანდიდატი, გერმანიის DAAD-ის გრანტის მრავალგზის მფლობელი, ნიურნბერგ-ერლანგენის, ბაიროითის, ზაარბრუკენის და სხვა უნივერსიტეტების მიწვეული პროფესორი 2002-2012 წწ. 60-ზე მეტი სამეცნიერო ნაშრომის ავტორიმ, მათ შორის 10 წიგნის. საქართველოს პრეზიდენტის გრანტის მფლობელი ახალგაზრდა მეცნიერთათვის, მონაწილეობდა საქართველოს სასამართლო სისტემის რეფორმის მსოფლიო ბანკის პროექტში. ფლობს „მაიკროსოფტის“ ვინდოუს- და ვებ-სისტემების დაპროგრამების უახლოეს ტექნოლოგიებს. არის ბიზნეს-ანალიტიკოსი და კონსულტანტი საბანკო რისკების პროცეს-ორიენტირებული მოდელირების და ავტომატიზაციის სფეროში.

სარჩევი

- შესავალი: პროგრამული სისტემების მენეჯმენტის მიზნები და ამოცანები, განვითარების პერსპექტივები	7
1. ლაბ.N1: აპლიკაციის აგება დაპროგრამების რამდენიმე ენის საფუძველზე .dll ფაილების შექმნით	15
2. ლაბ.N2: კლასების და ობიექტების დაპროგრამება მართვის საინფორმაციო სისტემების ამოცანებში	32
3. ლაბ.N3: მონაცემთა განახლების მეთოდები ListView-ში.....	47
4. ლაბ.N4: ცხრილებთან მუშაობა - DataGridView	53
5. ლაბ.N5: DataGridViewComboBoxColumn და DataGridViewTextBoxColumn კლასები ცხრილებთან სამუშაოდ.....	63
6. ლაბ.N6: C# აპლიკაციის მუშაობა Ms Access ბაზასთან	68
7. ლაბ.N7: C# აპლიკაციის მუშაობა MySQL ბაზასთან	85
8. ლაბ.N8: C# აპლიკაციის მუშაობა Ms SQL Server ბაზასთან ADO.NET დრაივერით და DataGridView კლასით	99
9. ლაბ.N9: რეფაქტორინგი: კოდის რეორგანიზაცია	114
10. ლაბ.N10: Visual Studio.NET -ის რევერსიული ინჟინერიის ინსტრუმენტული საშუალებები: „Model-Code-Model“.....	120
11. ლაბ.N11: პროგრამული აპლიკაციების ტესტირება	131
12. ლაბ.N12: პროგრამული კოდების ხარისხის შეფასება	142
13. ლაბ.N13: პროექტის ბიუჯეტის შედგენისთვის სამომხმარებლო აპლიკაციის მომზადება	146

14. ლაბ.N14: პროგრამული სისტემების შექმნის დავალებათა კალენდარული გეგმის ფორმირება	159
15. ლაბ.N15: პროგრამული აპლიკაციის დისტრიბუციული ფაილის (საინსტალაციო პაკეტის) შექმნა	172
- ლიტერატურა	179

შესავალი

მართვის საინფორმაციო სისტემების პროგრამული უზრუნველყოფის ობიექტორიენტირებული ანალიზის, დაპროექტების, დეველოპმენტის, ტესტირების, დანერგვისა და რეინჟინერინგის საკითხების ინტენსიური კვლევა და სწავლება განსაკუთრებით მნიშვნელოვანი და აქტუალურია თანამედროვე კომპიუტერული ინდუსტრიისა და ინფორმაციული ტექნოლოგიების უახლესი მიღწევების ფონზე.

წიგნში შემოთავაზებულია გამოყენებითი პროგრამული სისტემების დაპროექტების, რეალიზაციის, ტესტირებისა და რეინჟინერინგის საკითხები. გამახვილებულია ყურადღება სისტემების პროგრამული უზრუნველყოფის დამუშავების პროცესების მენეჯმენტის ამოცანებზე. გადმოცემულია პროგრამული სისტემების აგების 15 ლაბორატორიული ამოცანა, კერძოდ .NET პლატფორმაზე დაპროგრამების მრავალენიანი პროექტის აგება dll-ფაილებით, მომხმარებელთა ინტერფეისებისა და ბაზების გამოყენება, პროგრამების ტესტირება, დისტრიბუციული (საინსტალიაციო) პაკეტების მომზადება და სხვა. გამოყენებულია MsVisual Studio.NET Framework 4.5 ინტეგრირებული სამუსაო გარემო: C#, ADO.NET, MsAccess, MySQL, SQL Server პაკეტებით.

პროგრამული სისტემების (პროექტების) დამუშავება, ხშირ შემთხვევაში განიხილება როგორც სპეციალისტთა კოლექტიური ნაშრომი, რომელიც მიმართულია მომხმარებელთა მოთხოვნილებების დასაკმაყოფილებლად მათი საქმიანი პროცესების (ბიზნეს-პროცესების) ავტომატიზაციის მიზნით [1-5].

როგორც ნებისმიერი სხვა კოლექტიური შრომა, ესეც მოითხოვს გარკვეულ ორგანიზაციას, კერძოდ მენეჯმენტს (მართვა სოციალურ და ორგანიზაციულ სისტემებში). ეს ხანგრძლივი პროცესია, რომელიც აკავშირებს პროგრამული პაკეტების

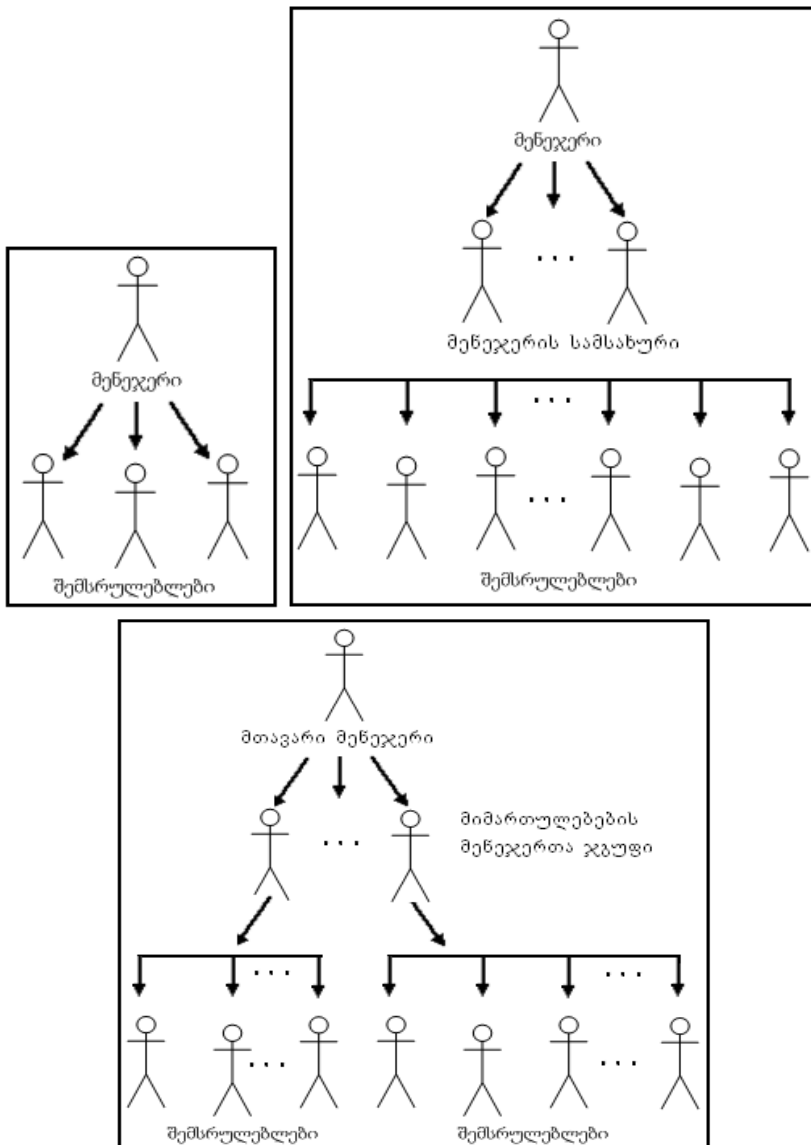
მწარმოებელ სპეციალისტებს საწარმოო და სხვა დამოკიდებულებებში ერთმანეთთან [6,7].

პროგრამული სისტემების დამუშავების აუცილებელი ელემენტია, ერთის მხრივ, მომხმარებელთა (მოთხოვნილებების) შესწავლა და, მეორეს მხრივ, მათთან უკუკავშირის უზრუნველყოფა, რომლითაც წარიმართება პროგრამების შექმნის საწარმოო პროცესები. ასეთი ამოცანების გადაწყვეტა ევალება ხელმძღვანელს, ანუ პროექტის მენეჯერს [3,4,6].

პროგრამული სისტემის ზომებიდან და სირთულიდან გამომდინარე, პროექტს შეიძლება ჰყავდეს ერთპიროვნული მენეჯერი (მცირე პროექტი), ან მენეჯერული სამსახური (დიდი პროექტი), ან მთავარი მენეჯერი და მიმართულებათა მენეჯერები (განსაკუთრებით დიდი პროექტი) ნახ.1.

პროგრამული სისტემების შექმნის პროექტების მენეჯმენტი, როგორც ეს ნახაზიდან ჩანს, შრომის განაწილების ორგანიზაციის თვალსაზრისით, მეტად მრავალფეროვანია და დამოკიდებულია როგორც კონკრეტული პროექტის მიზნებზე, მის სირთულეზე და შესრულების ვადებზე, ასევე შრომით კოლექტივში დავალებათა დელეგირების (შესასრულებელი ფუნქციების გადანაწილება) განსხვავებული მოდელების გამოყენებაზე.

მაგალითად, პროგრამისტთა მცირე ჯგუფებში (2-:10 კაცი), გამოიყენება პროგრამული სისტემების „სწრაფი დამუშავების“ (agile development) პრინციპები, რომელიც თანამედროვე პროგრამირების თეორიაში ცნობილია „ექსტრემალური დაპროგრამების“ მეთოდის სახელით (იხ. Google: ავტორი Kent Beck). ამ შემთხვევაში მთავარი მენეჯერის მოვალეობები გადანაწილდება პროგრამისტთა გუნდის წევრებზე, რომლებიც თვითონ განსაზღვრავენ დავალებათა დაგეგმვის, შესრულებისა და კონტროლის საკითხებს. ექსტრემალურ დაპროგრამების მეთოდს ჩვენ შემდგომში დავუბრუნდებით.



ნახ.1. პროგრამული პროექტის მენეჯმენტის ორგანიზაციის სქემები

პროგრამული სისტემების მენეჯმენტის ორგანიზაციის ყველა ვარიანტის ჩამოთვლა შეუძლებელია. ისინი ზოგჯერ აღმოცენდება სტიქიურად, ზოგჯერ კი იგეგმება პროგრამების წარმოების გარკვეული მეთოდოლოგიის გამოყენებით. ხშირად ასეთი სამუშაოები იგეგმება და წარიმართება არსებული კოლექტივის ტრადიციებიდან გამომდინარე.

პროგრამული პროექტების მენეჯერისთვის ყოველთვის საყურადღებოა ერთმანეთთან მჭიდროდ დაკავშირებული ორი ასპექტი:

- **პროექტის მართვა**, როგორც პროგრამული პროდუქტის წარმოების პროცესი გარკვეული მიზნის მისაღწევად;

- პროექტში მონაწილე ადამიანების **ხელმძღვანელობა**.

პროგრამული სისტემების დამუშავების მენეჯმენტის უმთავრესი და მუდმივი ამოცანაა პროექტის თანამიმდევრული რეალიზაცია-განვითარება დასმული მიზნის შედეგების მისაღწევად. გარდა იმ ხერხებისა და მეთოდებისა, რომლებითაც შესაძლებელია ასეთი ამოცანის გადაწყვეტა, საჭიროა ასევე პროექტის რესურსების ეფექტური განაწილებისა და გამოყენების კონტროლის საკითხის გადაწყვეტა. რესურსებში, ტრადიციულად მოიაზრება დრო, ფინანსები, ტექნიკური საშუალებები და კადრების საწარმოო პოტენციალი.

პროექტის მენეჯერმა ყოველთვის უნდა გაითვალისწინოს ისეთი ორგანიზაციული და საწარმოო კონტექსტები, როგორცაა პროექტის დამკვეთთა და შემსრულებელთა ინტერესების შეთანხმება, პროექტის შესასრულებლად საქმიანობის სფეროთა მრავალფეროვნება, აგერთვე სხვა კრიტერიუმების ერთობლიობა.

განასხვავებენ მომხმარებელთა და სისტემურ მოთხოვნილებებს:

- **მომხმარებელთა მოთხოვნილებები**, ესაა ბუნებრივ ენაზე აღწერილი ფუნქციები (ბიზნეს-პროცესები) და დიაგრამები,

რომლებიც სრულდება სისტემის მიერ, აგრეთვე მასზე დადებული შეზღუდვები (ბიზნეს-წესები).

- **სისტემური მოთხოვნები** არის სისტემური ფუნქციების და შეზღუდვების დეტალიზებული აღწერა, რომელსაც ზოგჯერ ფუნქციონალურ სპეციფიკაციასაც უწოდებენ. იგი კონტრაქტის გაფორმების საფუძველია სისტემის დამკვეთსა და პროგრამული სისტემის შემსრულებელს შორის.

ზოგჯერ განიხილავენ ასევე **საპროექტო სისტემურ სპეციფიკაციასაც**, რომელიც პროგრამული სისტემის სტრუქტურის განზოგადებული აღწერაა. იგი საფუძველია სისტემის უფრო დეტალიზებული დაპროექტებისა და მისი შემდგომი რეალიზაციისათვის.

ამგვარად, პროგრამული პროექტი შეიძლება განვიხილოთ როგორც სისტემის რეალიზაციის პროცესი, რომელიც აკმაყოფილებს სრულიად განსაზღვრულ შეზღუდვებს. ყველა შეზღუდვა არ ფორმირდება ერთბაშად, ხშირად ზოგიერთი მათგანი აღმოცენდება პროექტის შესრულების პროცესში, ან თვით პროგრამული სისტემის გამოყენების ეტაპზეც კი. ამიტომაც მენეჯერმა უნდა გაითვალისწინოს ის გარემოება, რომ მას პროექტზე მუშაობა მოუხდება დიდი განუსაზღვრელობის პირობებში საბოლოო შედეგთან მიმართებაში.

რა ამოცანების გადაწყვეტა უხდება მენეჯერს პროექტზე მუშაობისას, მისი მიზანმიმართულად და ეფექტიანად წარმართვის მიზნით? აქ განვიხილავთ ორ საკითხს:

- ვინ მონაწილეობს პროექტის დამუშავებაში;
- რა ეტაპებს გადის პროექტი მისი განხორციელებისას.

პირველი მათგანი გამოკვეთავს მენეჯერის ფუნქციებს სისტემის დამმუშავებლების გუნდში. აქ ორ ასპექტს ექცევა ყურადღება: მართვისას და ხელმძღვანელობისას.

მართვის თვალსაზრისით, პროექტის მონაწილეები - აბსტრაქტული მოქმედი აგენტებია, რომლებიც არსულებენ მათზე დაკისრებულ ფუნქციებს. იყენებენ გარკვეულ რესურსებს და იღებენ განსაზღვრულ შედეგებს. ასეთი ფუნქციონალური განხილვა შესაძლებელს ხდის პროექტის მონაწილეთა ურთიერთშენაცვლების შესაძლებლობას და მას მიყვავართ „როლის“ ცნებამდე.

ხელმძღვანელობის თვალსაზრისით, მთავარი ამოცანაა საქმიანი, შეთანხმებული და მეგობრული გუნდის ჩამოყალიბება, რომელსაც შეუძლია პროექტის შესრულება. გუნდის ძირითადი მოთხოვნა მისი წევრების კომპეტენტურობა და კვალიფიკაციაა, მაგრამ ეს არასაკმარისია. გარდა ამისა აუცილებელია გუნდური მუშაობის (ურთიერთდამოკიდებულება), ხელმძღვანელობასთან, დამკვეთებთან ურთიერთობის წესების ჩამოყალიბება.

ასეთი ურთიერთობების შექმნა და მისი მართვა პროექტის მენეჯერის ამოცანაა, როგორც გუნდის ხელმძღვანელისა. იგი ამყარებს გარე და შიგა კონტაქტებს (იგულისხმება ურთიერთობები პროგრამული სისტემის დამკვეთებთან და საკუთარი ორგანიზაციის ხელმძღვანელობასთან).

პროგრამული სისტემის შესრულების და განვითარების ეტაპების საკითხები პროექტის მენეჯერის ამოცანაა, როგორც ხელმძღვანელისა. ის ანაწილებს ფუნქციებს გუნდის წევრებს შორის დროში. ადგენს გადასაწყვეტ ამოცანათა სირთულეს და ლოგიკურ თანამიმდევრობას, მათი გადაწყვეტის კოლექტიურ ხასიათს. ახორციელებს სამუშაოების შესრულების მენეჯერულ კონტროლს.

მთელი ეს პროცესები კავშირშია **პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის** ცნებასთან.

- **პროგრამული უზრუნველყოფის სასიცოცხლო ციკლი** - ესაა პროგრამული უზრუნველყოფის კონსტრუირების პროცესი. პროგრამულ პროექტზე მოთხოვნილებანი განსაზღვრავს ამ პროცესის მიზანს და რეგლამენტს.

- არსებობს **მთავარი, საწარმოო და დამხმარე რესურსები.**
- **მთავარი რესურსებია: კადრები** (კუ-ს დამმუშავებლები, შემქმნელები), აგრეთვე პროგრამული პროექტის შესრულების **დრო** (ვადები) და **ფინანსები**, რომლებიც გამოიყოფა საპროექტო დავალების შესასრულებლად.
 - **საწარმოო რესურსებია:** ტექნიკური საშუალებები, აგრეთვე პროგრამები, რომლებიც გამოიყენება ინსტრუმენტული საშუალებების, პროტოტიპების ან ჩასასმელი კომპონენტების სახით.
 - **დამხმარე რესურსებია:** გათბობა, ელექტროენერგია, სამუშაო ოთახები, ტექნიკური და ორგანიზაციული საშუალებები, რომლების უშუალოდ პროექტის წარმოებასთან არაა კავშირში, მაგრამ ხელს უწყობს მის წარმატებით შესრულებას.
 - **ადამიანური** რესურსი კარგი პროექტისთვის სტაბილურია, არ იცვლება პროექტის შესრულების პროცესში. საქმის ცუდად ორგანიზებისას ადგილი აქვს კადრების დენადობას, რაც პროგრამული პროექტების საბოლოო ხარისხზე ახდენს უარყოფითად გავლენას.
 - **დრო** - შეუქცევადი რესურსია, ამიტომ მისი სწორად დაგეგმვა მენეჯერის მთავარი საზრუნავია.
 - **ფინანსები** - ძირითადი სახარჯო რესურსია, რომელიც ყოველთვის შეზღუდულია. კერძოდ, იგი გამოიყენება არა მხოლოდ ხელფასის სახით, არამედ ტექნიკური და ინსტრუმენტული საშუალებების შესაძენად.
 - პროგრამული პროექტების მენეჯმენტისთვის საჭიროა **რესურსების ხარჯვის ოპტიმალური ბალანსის შედგენისა და მისი შესრულების** ამოცანების გადაწყვეტა. ეს ამოცანები ეხება ყველა სახის პროექტის მენეჯერს. მასზე მოქმედებს პროექტის სპეციფიკა და მისი შესრულების პირობები.
 - არსებობს მენეჯმენტის მხარდამჭერი მიდგომების, მეთოდების, მეთოდიკების და ტექნოლოგიების სიმრავლე, მაგრამ

მათი გამოყენება არაა მარტივი და გარკვეულ ცოდნას და მომზადებას მოითხოვს. აგრეთვე პროგრამული პროექტები ხშირად ინდივიდუალურია და დიდი მნიშვნელობა ექცევა მენეჯერისა და შემსრულებლების პრაქტიკულ გამოცდილებას.

- ამგვარად, მენეჯერის ამოცანების განხილვისას პროგრამული პროდუქტების დამუშავების პროცესში გამოიკვეთება სამი ურთიერთდაკავშირებული მიმართულება:

1. ფუნქციები, რომლებიც აუცილებელია პროექტის წარმატებული განვითარებისთვის. აქ აუცილებელია გაირკვეს, თუ მოცემული პროექტისთვის თანამშრომელთა რომელი როლებია საჭირო;

2. პროექტის დაგეგმვა და მისი შესრულების კონტროლი შესაქმნელი პროგრამული სისტემის სასიცოცხლო ციკლის შესაბამისად.

3. პროექტის კოლექტივის ფორმირება (საკადრო უზრუნველყოფა).

დამხმარე სახელმძღვანელო ორიენტირებულია პროგრამული ინჟინერიისა და მართვის საინფორმაციო სისტემების სპეციალობების მაღალი კურსის სტუდენტებისა და მაგისტრანტებისთვის. აგრეთვე პროგრამული უზრუნველყოფის მენეჯმენტის საკითხებით დაინტერესებული მკითხველისთვის.

1. ლაბორატორიული სამუშაო N 1:

აპლიკაციის აგება დაპროგრამების რამდენიმე ენის საფუძველზე .dll ფაილების შექმნით

მიზანი: ობიექტ-ორიენტირებული დაპროგრამების ენების: C++, Visual Basic და C# გამოყენებით ერთი პროგრამული პროექტის აგების ტექნოლოგიის შესწავლა .NET პლატფორმაზე. ამ ენებზე .dll ფაილების შექმნის და მათი ერთად მუშაობის პროცესის გაცნობა.

1. თეორიული ნაწილი

ილუსტრირებულია .NET ტექნოლოგიის ერთ-ერთი ძირითადი პრინციპი, რომ პროგრამული აპლიკაციის დამუშავება შესაძლებელია დეველოპერების გუნდში სხვადასხვა პროგრამული ენის მცოდნე სპეციალისტების მიერ, კერძოდ C++, Visual Basic და C# ენების საფუძველზე. პროექტის აგებისას გათვალისწინებულ უნდა იქნას საერთო მოთხოვნები ღია ცვლადებზე, მეთოდებსა და თვისებებზე.

სამუშაო გეგმა:

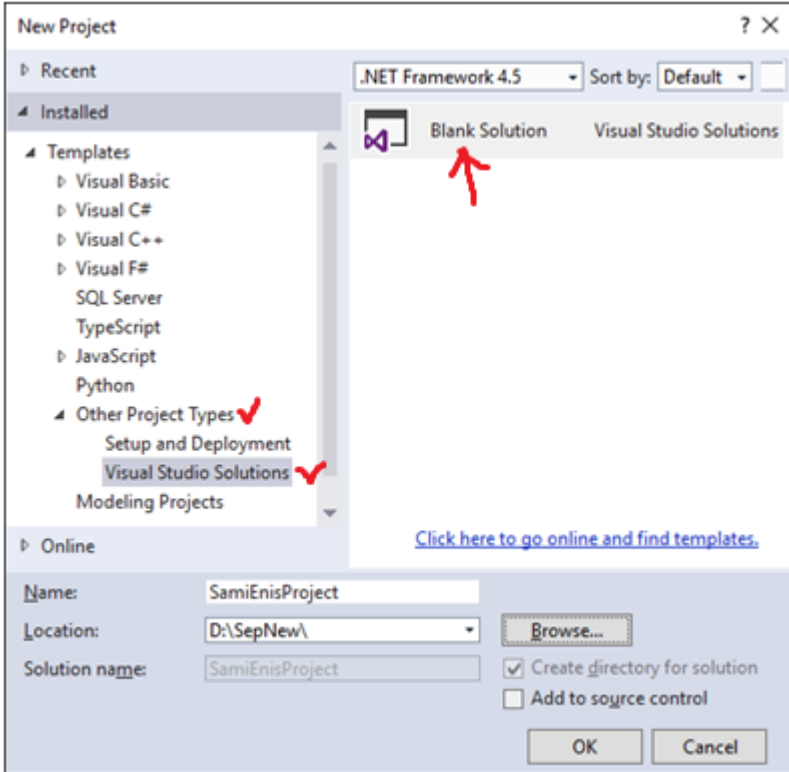
- შეიქმნას C++ -ის საბაზო კლასი;
- შეიქმნას Visual Basic.NET კლასი, რომელიც იქნება C++ - კლასის მემკვიდრე;
- შეიქმნას C# კლასი, რომელიც კონსოლის აპლიკაციაში შექმნის Visual Basic.NET კლასის ეგზემპლარს და გამოიძახებს მის მეთოდს.

2. პრაქტიკული ნაწილი

განვახორციელოთ პროექტის პროგრამული რეალიზაცია Visual Studio .NET Framework 4.0 სამუშაო გარემოში (შესაძლებელია სხვა ვერსიის გამოყენებაც, მაგალითად, 4.5).

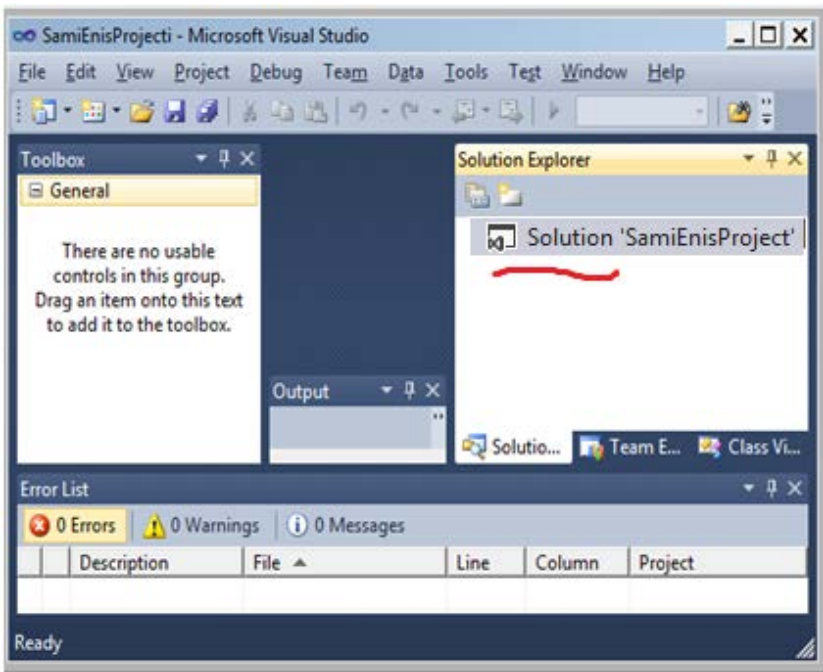
- შევარჩიოთ ახალი პროექტის განთავსების ადგილი
- მენიუდან გამოვიძახოთ ახალი ცარიელი პროექტის შექმნის პროგრამა:

(New->Project და Visual Studio Solution-> Blank Solution)



ნახ.1.1.

პროექტის სახელი (Name: SamiEnisProjecti) და მისი შენახვის ადგილი (Location) მივუთითოთ ჩვენი სურვილით. OK-ის შემდეგ Blank Solution შაბლონის დახმარებით შეიქმნება აპლიკაცია, რომელიც *ნეიტრალური* იქნება დაპროგრამების ენებისადმი. შედეგად მივიღებთ 1.2 ნახაზზე ნაჩვენებ ფანჯარას.



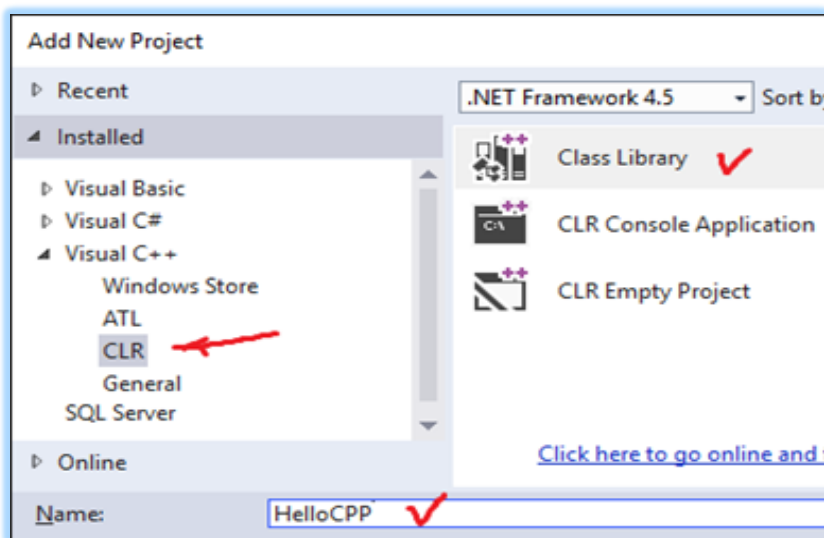
ნახ.1.2

- **კლასის შექმნა ახალ პროექტში C++.NET ენაზე**

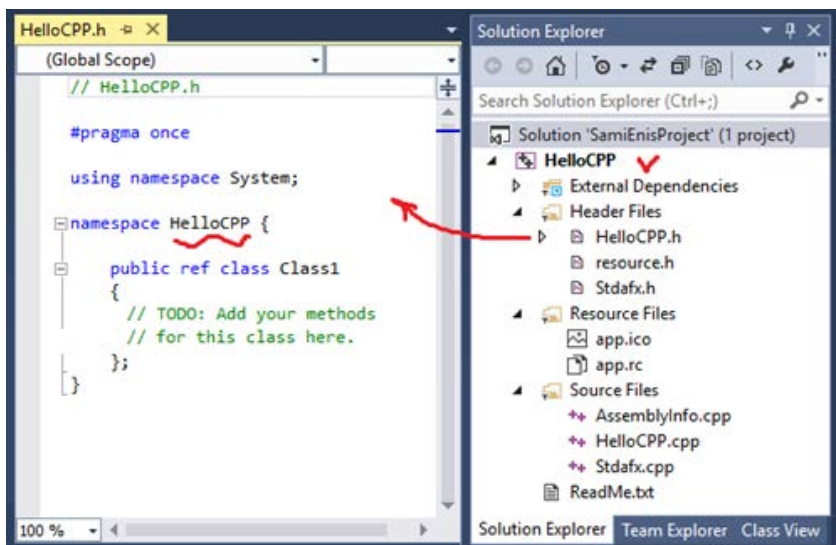
გარიელ პროექტს (გადაწყვეტა - Solution 'SamiEnisProject') დავამატოთ ახალი (ქვე)პროექტი მასზე მარჯვენა ღილაკის დაწკაპუნებით და შემდეგ Add -> New Project არჩევით. 1.3 ნახაზზე ნაჩვენებია თუ როგორ მიეთითება VisualC++ ენა, Class Library და ქვეპროექტის სახელი Name: HelloCPP, შემდეგ OK.

Solution Explorer -ში გამოჩნდება შედეგი (ნახ.1.4). გავხსნათ Hello.CPP.h ფაილი რედაქტირებისთვის. ახლადშექმნილი კლასის Class1 მაგივრად ჩავწეროთ სახელი HelloCPP.

HelloCPP.h კოდი მოცემულია 1-ელ ლისტინგში.



ნაბ.1.3



ნაბ.1.4

```
// ---- ლისტინგი_1 -- შეცვლილი HelloCPP.h ----
#pragma once
using namespace System;
namespace HelloCPP
{
    public ref class HelloCPP
    {
        // --- აქ ჩაერთება კლასის მეთოდი ----
    public:
        virtual void Hello()
        {
            Console::WriteLine("Mogesalmebit C++ enis
                                garemodan da !\n
                                viZaxeb Visual Basics !\n\n");
        }
    };
}
```

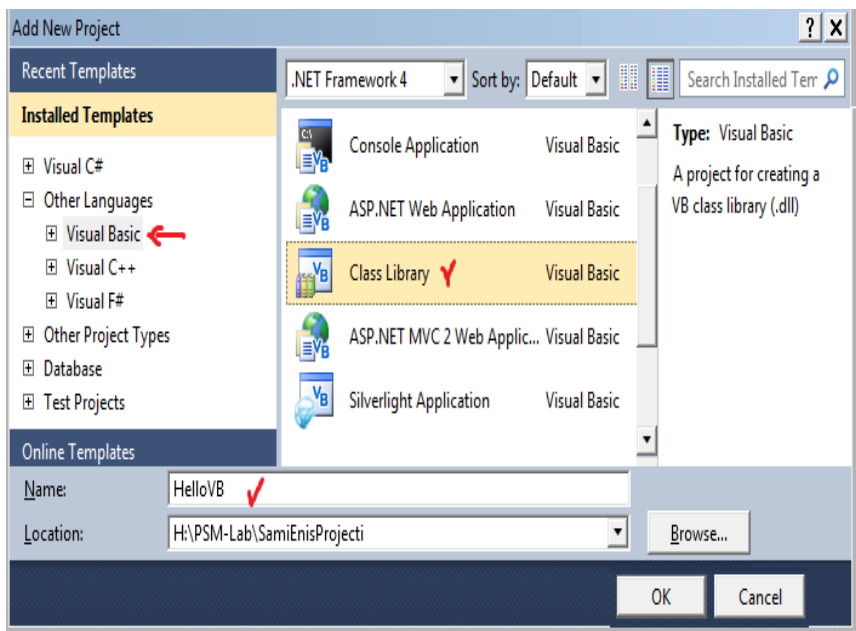
HelloCPP::Hello() მეთოდი გამოიყენებს .NET Framework კლასების ბიბლიოთეკიდან System::Console::WriteLine() ფუნქციას, რათა კონსოლზე გამოიტანოს მისაღმება C++ კოდიდან.

პროექტი გავუშვავთ სინტაქსური შეცდომების გასასწორებლად, თუ ასეთი იქნება. იგი ჯერ არ უნდა ავამუშავოთ შესრულებაზე.

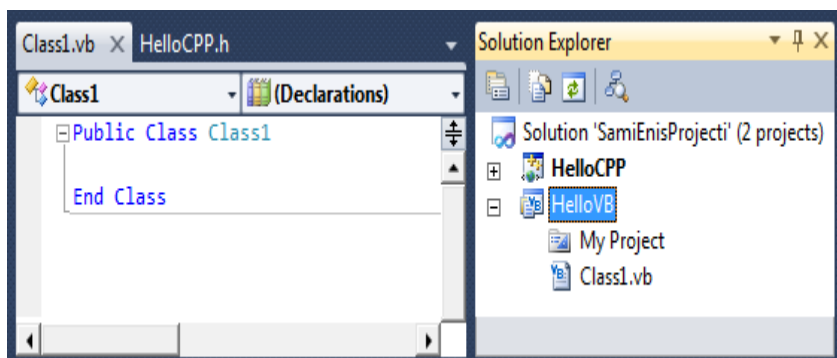
- **კლასის შექმნა ახალ პროექტში Visual Basic .NET ენაზე**

დავამატოთ Solution-ში ახალი პროექტი და მივუთითოთ, რომ იგი შეიქმნას Visual Basic .NET -ში სახელით Name: HelloVB (ნახ.1.5).

მივიღებთ ასეთ შედეგს (ნახ.1.6). ახალი პროექტი HelloVB დაემატება Solution Explorer პანელზე თავისი შემადგენელი ფაილებით.



ნახ.1.5



ნახ.1.6

ახლა უკვე გვაქვს HelloCPP და HelloVB ორი პროექტი.

მეორეში ავირჩიოთ Class1.vb ფაილი და შევუცვალოთ სახელი შინაარსის გათვალისწინებით, მაგალითად, HelloVB.vb, შემდეგ გავხსნათ იგი და ჩავწეროთ ჩვენთვის საჭირო ტექსტი (ლისტინგი_2).

```
'--ლისტინგი_2 --- HelloVB.vb -----
```

```
Public Class HelloVB
```

```
    Inherits HelloCPP.HelloCPP
```

```
    Public Overrides Sub Hello()
```

```
        MyBase.Hello()
```

```
        System.Console.WriteLine("Mec mogesalmebiT
```

```
            Visual Basic.NET-idan !!")
```

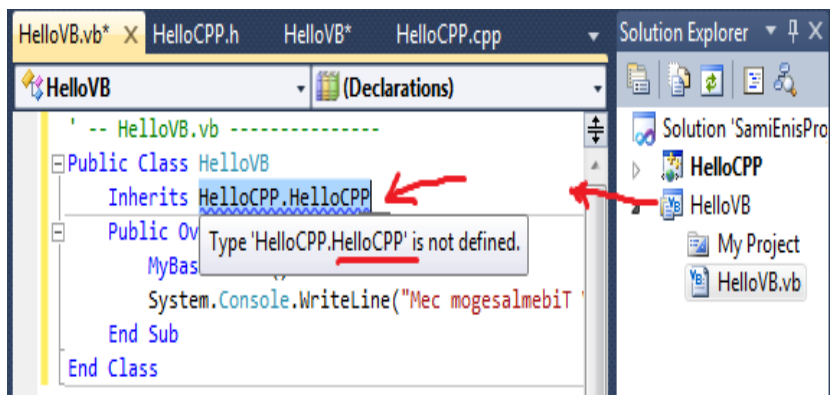
```
    End Sub
```

```
End Class
```

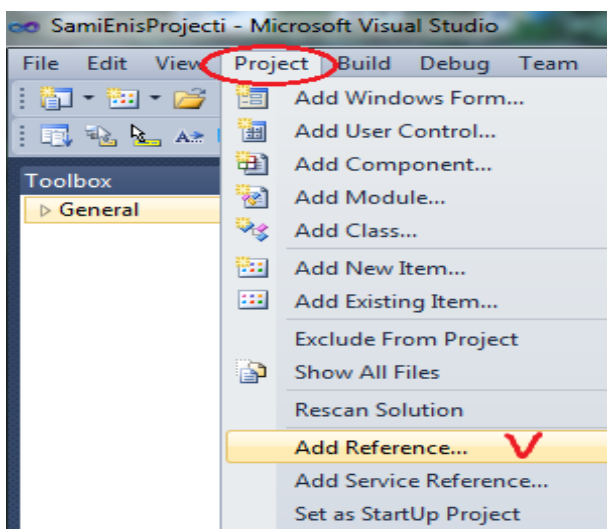
ეს კოდი განსაზღვრავს HelloVB კლასს, რომელიც მემკვიდრეობით იქნა მიღებული C++ ის მმართველი კლასიდან HelloCPP. ამგვარად, HelloVB კლასი ჩაანაცვლებს (Overrides) მშობელი HelloCPP კლასის ვირტუალურ Hello() მეთოდს, ოღონდ ჯერ გამოიძახებს Hello() მეთოდის ვერსიას მშობელი კლასიდან HelloCPP, შემდეგ კი გამოყავს ეკრანზე თავისი მისაღმება.

საყურადღებოა, რომ კოდის რედაქტორში საბაზო კლასის სახელი (HelloCPP.HelloCPP) და ჩასანაცვლებელი მეთოდის სახელი Hello() ტალღისებური ხაზით. ეს ნიშნავს, რომ კოდის რედაქტორი ამ სახელებს ვერ ხედავს და წინასწარ იძლევა სინტაქსურ შეცდომას. თუ კურსორს დავაყენებთ ამ ფრაგმენტზე, იგი მოგვცემს შეცდომის მნიშვნელობას (ნახ.1.7).

აუცილებელია Visual Basic-ის კომპილატორს მიეთითოს თუ სად იმყოფება ეს ნაკრები (assembly), რომელიც განსაზღვრავს მოცემულ ტიპს. ამისათვის მთავარი მენიუს Project-> Add Reference->Projects-დან (ნახ.1.8) ავირჩევთ HelloCPP-ს და OK.

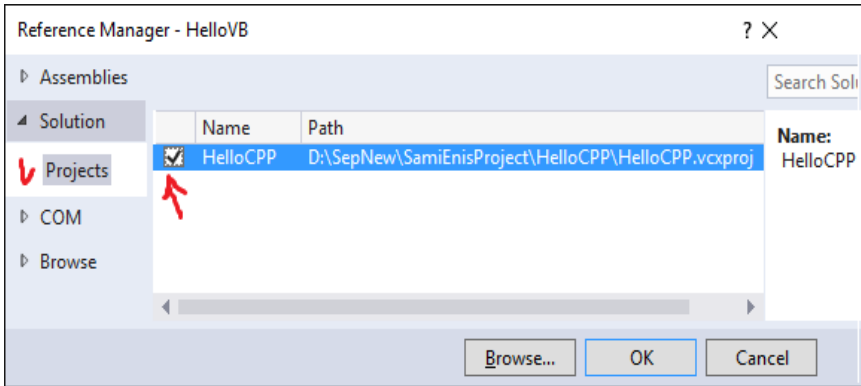


ნახ.1.7



ნახ.1.8.

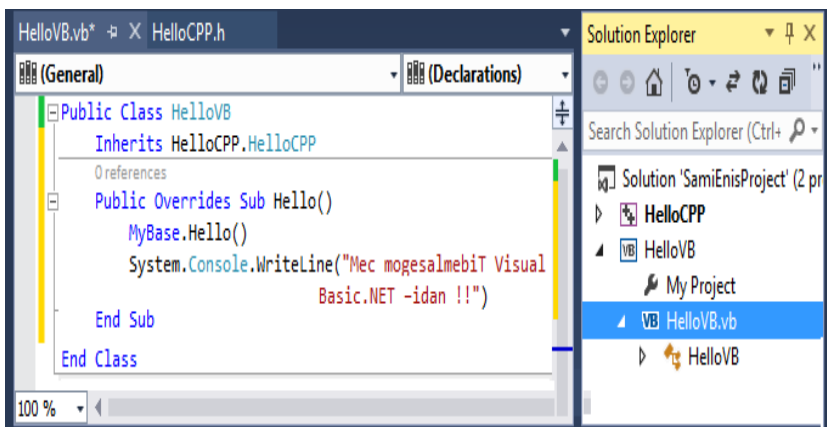
Reference Manager-ში Solution Projects-ზე ჩავრთოთ ჩვეულებრივი HelloCPP და Ok-ით დავადასტურებთ.



ნახ.1.9

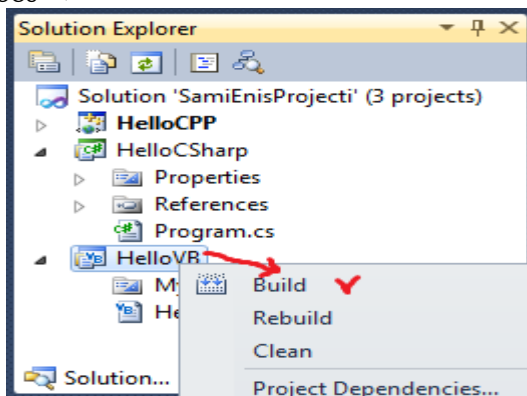
ამის შემდეგ გამოჩნდება Add Reference (ნახ.1.10) და OK ღილაკით ვასრულებთ CPP-კლასთან დაკავშირების პროცედურას.

ამგვარად, Hello.CPP პროექტი მიუერთდება გადაწვეტას (Solution-ში) და გამოჩნდება მომავალში HelloVB-ში, როგორც ზემოთ იქნა ნაჩვენები. ამასთანავე, 1.7 ნახაზზე ნაჩვენები „ქვეშეგახაზული“ შეცდომების აღნიშვნები HelloVB.vb პროგრამის ტექსტიდან გაქრა !



ნახ.1.10

ამის შემდეგ საჭიროა HelloVB პროექტზე მარჯვენა ღილაკით გამოვიტანოთ 1.11 კონტექსტური მენიუ და ავირჩიოთ Build (ეს პროექტი ტრანსლატორით ამუშავდება და სინტაქსური გამართვის შედეგს მოგვცემს).

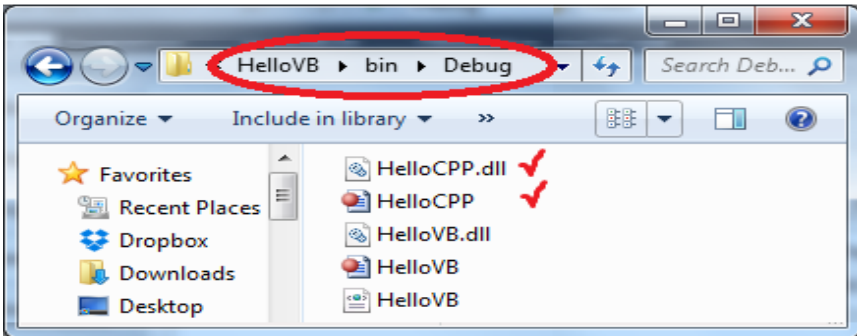


ნახ.1.11

როგორც წესი, თუ შეცდომები არაა HelloVB პროექტში, მაშინ მის შესაბამის ფოლდერის bin->Debug-ში უნდა გამოჩნდეს დაკავშირებული Hello.CPP.dll კლასის ფაილი (ნახ.1.12).

პროგრამის საბოლოო ტექსტი მოცემულია ლისტინგში:

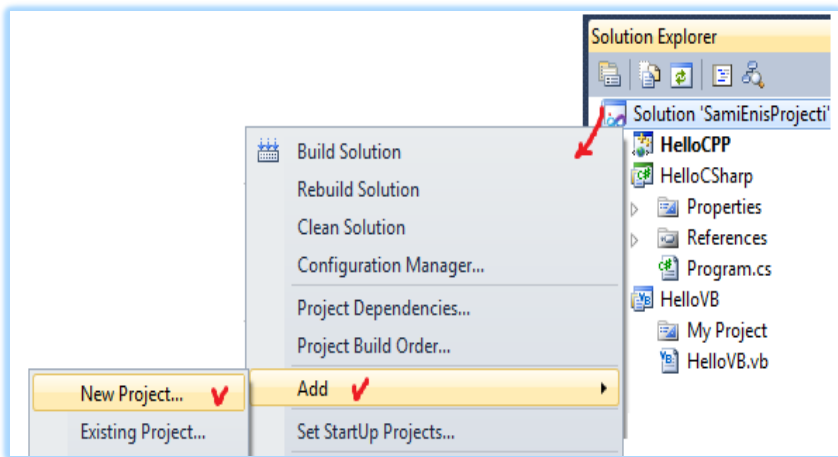
```
' -- HelloVB.vb -----  
Public Class HelloVB  
    Inherits HelloCPP.HelloCPP  
    Public Overrides Sub Hello()  
        MyBase.Hello()  
        System.Console.WriteLine("Mec mogesalmebiT Visual  
Basic.NET -idan !!!")  
    End Sub  
End Class
```



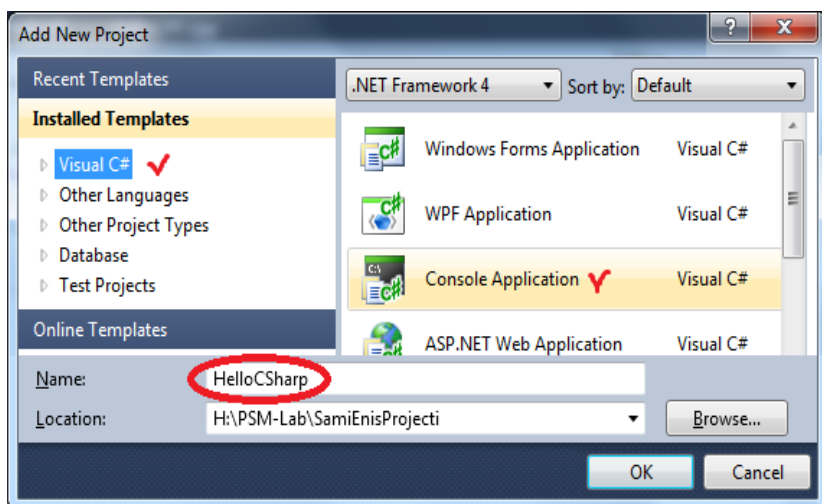
ნახ.1.12

- **სასტარტო პროექტის დამატება Solution-ში C# .NET ენაზე**

დასკვნით ფაზაში ჩვენი გადაწყვეტისათვის (Solution-ში) SamiEnisProjectი უნდა შევექმნათ მესამე, მთავარი სასტარტო პროექტი , რომელიც აგებული იქნება C#.NET ენაზე Console Application შაბლონის სახით. დავარქვათ მას HelloCSharp. ამგვარად, ვამატებთ პროექტს (ნახ.1.13-ა,ბ) HelloCSharp სახელით.

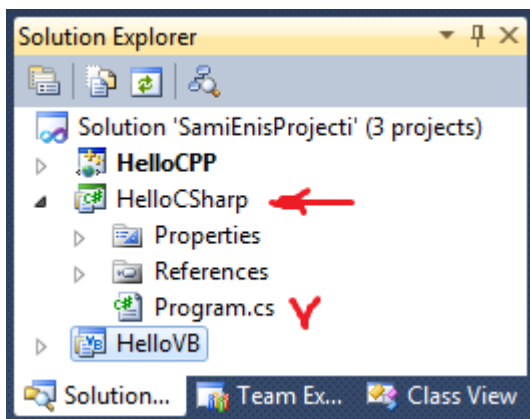


ნახ.1.13-ა



ნახ.1.13-ბ

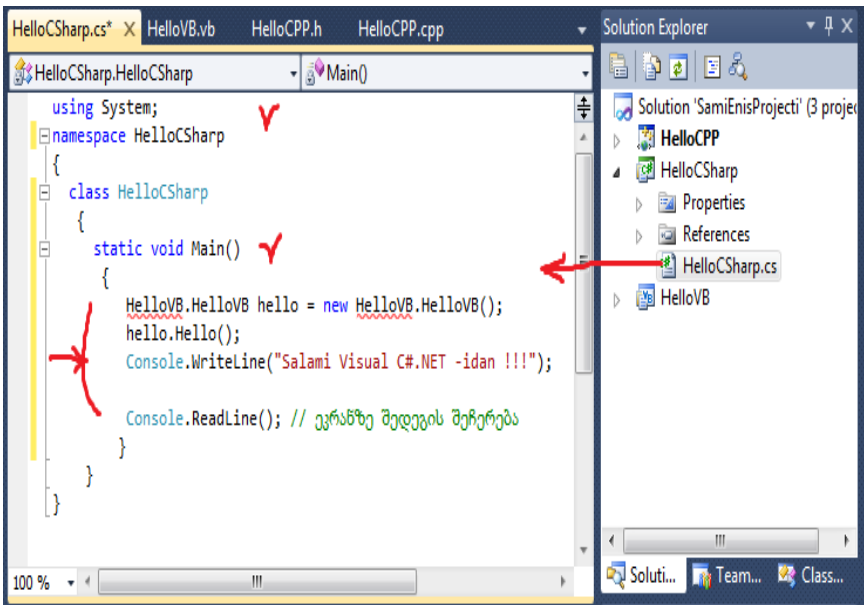
OK ღილაკის ამოკმელების შემდეგ მივიღებთ Solution-ში ახალ, HelloCSharp პროექტს თავისი შემადგენელი კომპონენტებით (ნახ.1.14).



ნახ.1.14

გადავარქვით სახელი Program.cs ფაილს ჩვენი პროექტის შინაარსის შესაბამისად: HelloCSharp.cs.

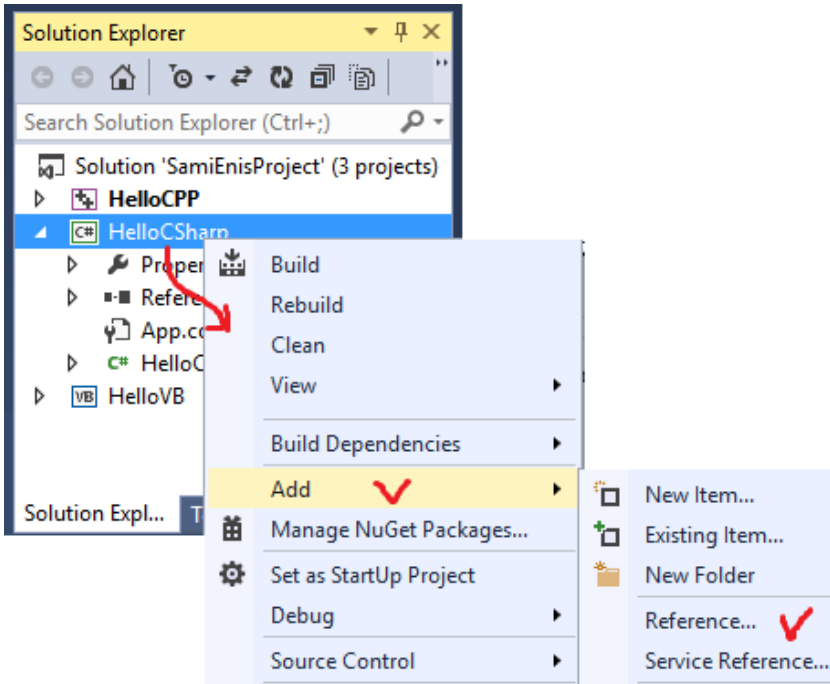
გამოვიტანოთ ეს ფაილი რედაქტირების ფანჯარაში და ჩავამატოთ Main() -ში შესაბამისი სტრიქონები. ასევე შეიძლება Main() -ის არგუმენტების წაშლა, აქ არ გვჭირდება. რედაქტირების შემდეგ ტექსტი მოცემულია 1.13. ნახაზზე.



ნახ.1.15

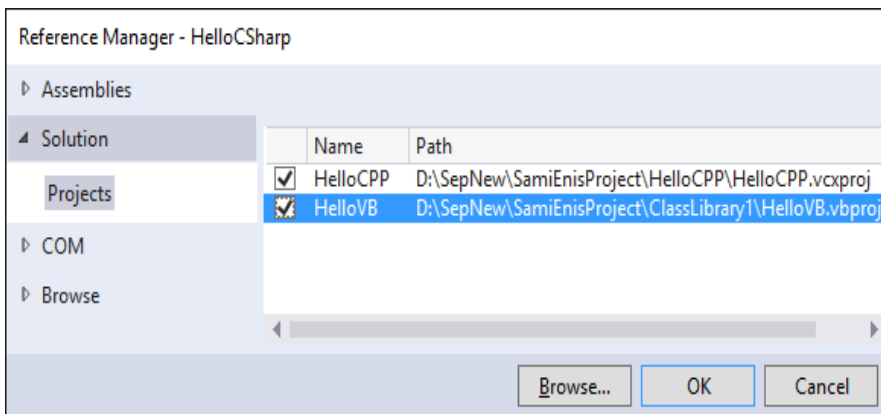
პროგრამაში Main() სტატიკური მეთოდი არის აპლიკაციაში შესვლის წერტილი. ეს მეთოდი ქმნის HelloVB კლასის ახალ ობიექტს hello და მისთვის იძახებს Hello() მეთოდს, რომელშიც ინახება ორი შეტყობინება. ერთი CPP.NET-იდან, მეორე VB.BET-დან, რომლებიც ადრე მოვამზადეთ. მესამე შეტყობინებას თვით C#.NET გამოიტანს, დამშვიდობებასთან ერთად.

ახლა საჭიროა HelloCSharp პროექტში ჩავამატოთ მიმთითებლები (კავშირები) HelloCPP და HelloVB პროექტებზე. მაშინ 1.15 ნახაზზე ნაჩვენები შეცდომები (ქვეშაზღვასმული HelloVB) გასწორდება. ამისათვის ვირჩევთ Add -> Reference...-ს (ნახ.1.16).

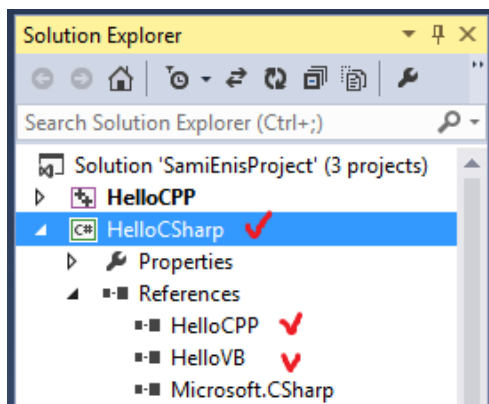


ნახ.1.16

Reference Manager ფანჯარაში გადავრთოთ Projects-ზე, მოვნიშნოთ ორივე, HelloCPP და HelloVB პროექტები და დავაწკაპოთ OK (ნახ.1.17). შედეგად გაქრება შეცდომები, ანუ 1.15 ნახაზზე „ხაზები“, რაც მიუთითებს იმაზე, რომ კავშირი პროექტებს შორის კარგად შესრულდა. ეს შედეგი აისახება Solution-ფანჯარაში HelloCSharp პროექტის References –ში (ნახ.1.18).

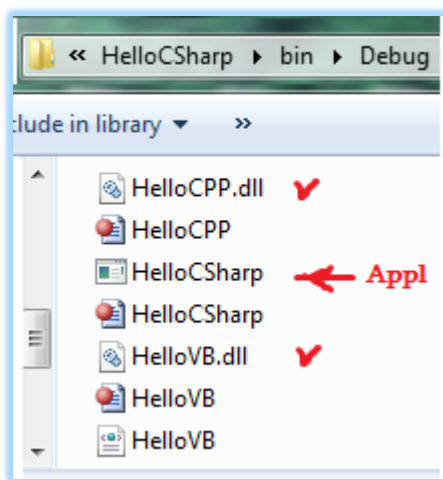


ნახ.1.17



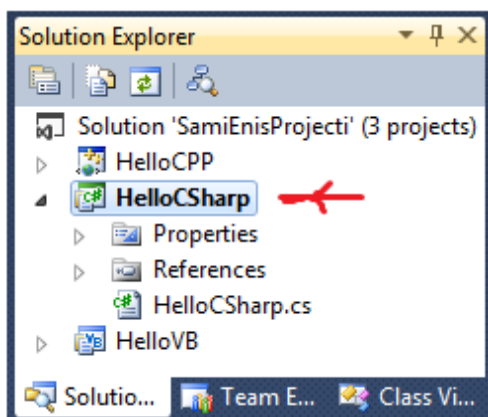
ნახ.1.18

ავამუშავოთ HelloCSharp პროექტი მარჯვენა ღილაკით და build-ით, რათა სინტაქსურად დამუშავდეს იგი. შეცდომების არარსებობის შემთხვევაში მოხდება HelloCPP.dll და HelloVB.dll ფაილების განთავსება HelloCSharp პროექტის bin->Debug ფოლდერში (ნახ.1.19). აქვეა HelloCSharp.exe აპლიკაციის ფაილიც.

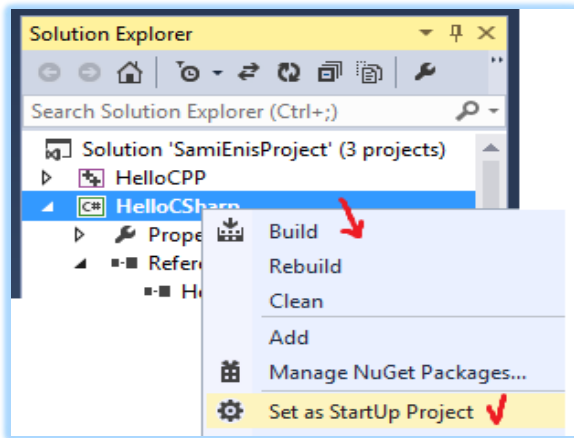


ნახ.1.19

Solution-ში HelloCSharp პროექტი გადავაკეთოთ სასტარტო ფაილად (ნახ.1.20-ა,ბ).

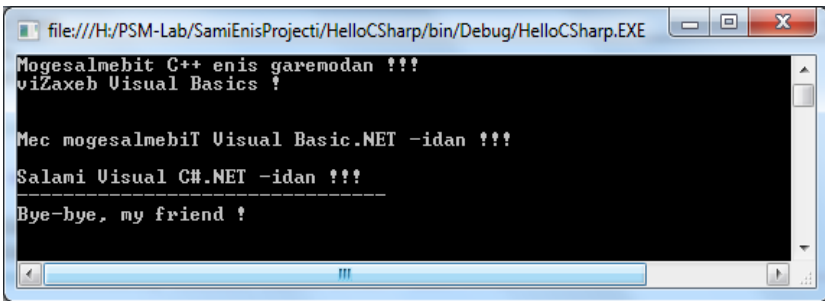


ნახ.1.20-ა



ნახ.1.20-ბ

ბოლოს, ავამუშავოთ აპლიკაცია და მივიღებთ შედეგს (ნახ.1.21).



ნახ.1.21

რეზიუმე: აპლიკაციის სხვადასხვა პროექტები დამუშავდა დაპროგრამების სხვადასხვა ენებზე, რომლებიც გამოიყენება .NET გარემოში. მრავალჯერადი გამოყენების ფაილები შემუშავებულ იქნა კლასების დახმარებით და რეალიზებულია დინამიკური .dll - ფაილების სახით.

დავალეზა: ააგეთ განხილული პროგრამული პროექტის კოდი.

2. ლაბორატორიული სამუშაო N2

კლასების და ობიექტების დაპროგრამება მართვის ამოცანებში

მიზანი: ობიექტ-ორიენტირებული დაპროგრამების ინკაფსულაციის თვისების დეტალიზებული შესწავლა. კლასის განსაზღვრა როგორც მონაცემებისა და მეთოდების ერთობლიობა და მათი პროგრამული რეალიზაციის განხორციელება. როგორ შევქმნათ კლასები, მათი ობიექტები, მეთოდები და კლასთაშორის კავშირები კონსოლისა და ვინდოუს_ფორმების პროექტებში ?

განვიხილოთ აღნიშნული საკითხები პრაქტიკული მართვის ამოცანის საფუძველზე, მათი შემდგომი განზოგადების მიზნით.

ამოცანა_2.1: ავაგოთ ობიექტ-ორიენტირებული პროგრამის კოდი (კლასების და მეთოდების გამოყენებით) მაღლივ შენობაში ლიფტის გადაადგილების მოდელირებისათვის.

მაგალითად, შენობა N სართულიანია. მას აქვს ლიფტი. პერსონა, რომელიც შედის ლიფტში (ხდება მგზავრი), ირჩევს მისთვის საჭირო სართულს და მიემგზავრება (ზევით ან ქვევით). საჭიროა ამ პროცესის მოდელირება და დაპროგრამება ისე, რომ დაფიქსირდეს ლიფტის საწყისი მდგომარეობა, ამუშავების შემდეგ მისი საბოლოო მდგომარეობა, გავლილი სართულების რაოდენობა (ერთი ამუშავებისას). საბოლოოდ გამოიცეს რეპორტი, თუ სულ რამდენი სართული გაიარა ლიფტმა ერთი სეანსის (გარკვეული პერიოდის) განმავლობაში.

2.1) კლასთა მოდელი და მათი აგების პირობები:

ინკაფსულაციის სქემა მოცემულია 2.1 ნახაზზე.

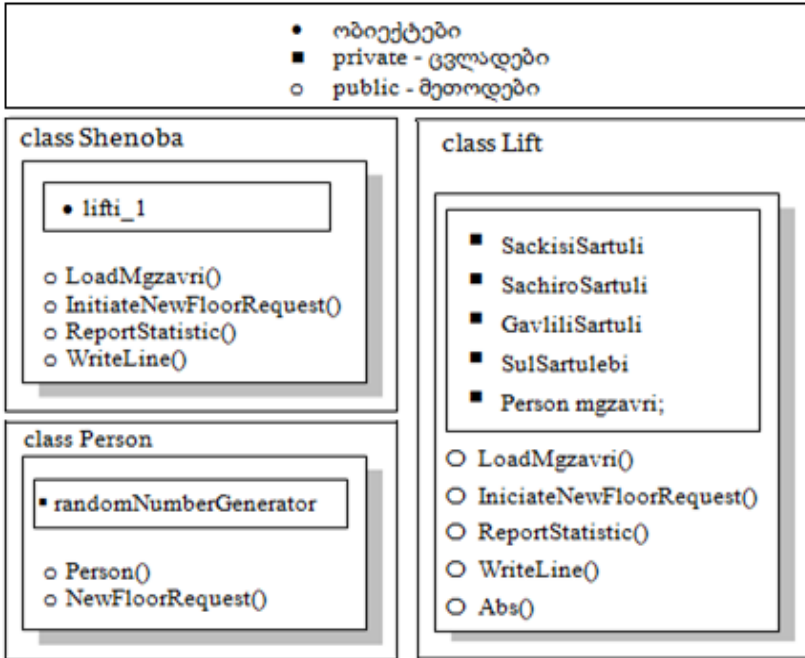
- პროგრამაში სამი კლასია: Shenoba, Lift და Person;

- Shenoba კლასს აქვს Lift კლასის ერთი ობიექტი, სახელით

lifti_1 ;

- Person კლასის ერთი ობიექტი იმყოფება mgzavri - ცვლადის შიგნით, რომელიც იყენებს lifti_1;

- Lift კლასის ობიექტს შეუძლია გადაადგილება ნებისმიერ სართულზე, ინტერვალში [1,N=60].



ნახ.2.1

- პროგრამაში სართულის არჩევა ხდება მგზავრის მიერ (გამოიყენება „შემთხვევით რიცხვთა გენერატორი“ Random-ფუნქციით);

- lifti_1 ლიფტი მოძრაობს საჭირო სართულისკენ, რაც აისახება კონსოლზე;

- მგზავრ(ებ)ის ლიფტით მოძრაობის სენსი შედგება რამდენიმე ეტაპისგან, რომლებზეც შეირჩევა ახალი მიზნობრივი სართულები;

- სენსის ბოლოს გამოიცემა ანგარიშის ტექსტი (რეპორტი), თუ ჯამში რამდენი სართული გაიარა ლიფტმა;

- შუალედური და საბოლოო შედეგები გამოიტანება ეკრანზე.

2.2) კლასთა კავშირების აღწერა UML ტექნოლოგიით:

მომხმარებლის სამი კლასი: Shenoba, Lift, Person და ერთი სისტემური კლასი System.Random ამოდელირებს ლიფტის მუშაობის პროცესს:

- Shenoba-კლასი შეიცავს Lift-ობიექტს და იძახებს მის მეთოდებს;

- Lift-ობიექტი იყენებს Person-ობიექტს, რათა მართოს ლიფტის მოძრაობა;

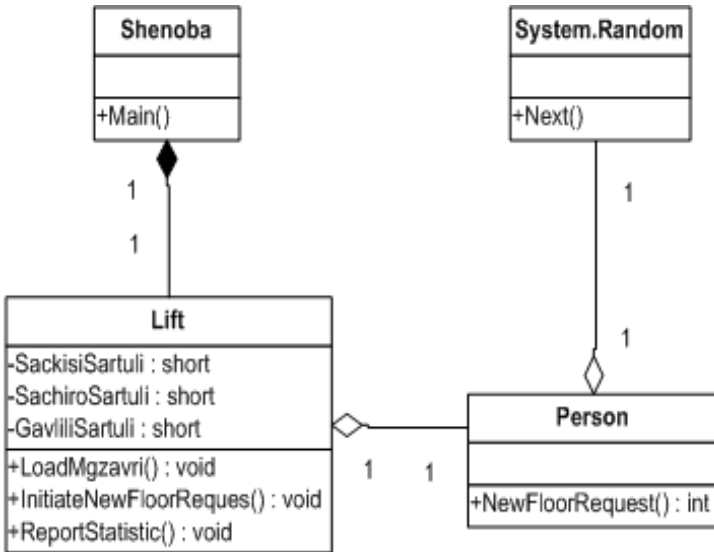
- Person-ობიექტი კი იყენებს System.Random-ობიექტს, რათა აირჩიოს საჭირო სართული შემდგომი გადაადგილებისთვის.

ობიექტ-ორიენტირებული პროგრამული სისტემების დაპროექტებისას კლასებმა მსგავსი ფუნქციები უნდა შეასრულოს. თითოეულს უნდა ჰქონდეს თავისი უნიკალური ფუნქციონალობა, და ყველა ერთად უნდა უზრუნველყოფდეს მთლიანი პროგრამის მუშაობას.

ნებისმიერი მაღლივი შენობა, როგორც „მთელი“ შედგება სხვადასხვა ნაწილებისგან: იატაკები, კედლები, ფანჯრები, ლიფტები და ა.შ. ამგვარად, შენობასა და ლიფტს შორის არსებობს დამოკიდებულება „მთელი-ნაწილი“ (აგრეგატული კავშირი UML ენაზე). ამიტომაცაა, რომ Lift-კლასის ობიექტის ცვლადი გამოცხადებულია Shenoba-კლასის შიგნით (ნახ.2.1, lifti_1). პროგრამულად იგი რეალიზებულია სტატიკური ცვლადის სახით (47-ე სტრუქტონი, ლისტინგი_2.1): `private static Lift lifti_1;`

იგი ინახავს Lift კლასის ობიექტს და შეუძლია მისი მეთოდების გამოძახება. გარდა ამისა, Shenoba-კლასს შეიძლება ჰქონდეს აგრეთვე სხვა ცვლადებიც, რომლებიც აგრეგატულად დაქვემდებარებული კლასების ობიექტების ცვლადები იქნება.

2.2 ნახაზზე მოცემულია ჩვენი მაგალითის კლასებს შორის კავშირების UML დიაგრამა, აგებული Ms Visio პაკეტის გარემოში.



ნახ.2.2

Shenoba და Lift კლასებს შორის აგრეგაციას უწოდებენ კომპოზიციას და იგი შავი რომბიკით გამოისახება. „1“-ები ნიშნავს, რომ 1 შენობაში არის 1 ლიფტი (ჩვენს შემთხვევაში). Lift და Person, აგრეთვე Person და System.Random კლასებს შორის აგრეგატული დამოკიდებულებაა, მაგრამ არა-კომპოზიციური. ის თეთრი რომბიკითაა ნაჩვენები. განსხვავება ისაა, რომ შენობას ლიფტი ყოველთვის აქვს. ლიფტში კი პერსონა შეიძლება იყოს, ან არ იყოს, ლიფტი ისეც მუშაობს.

2.3) პროგრამული რეალიზაცია:

ცხრილში მოცემულია განხილული კლასების პროგრამული რეალიზაციის ლისტინგი.

<ul style="list-style-type: none"> ■ private - ცვლადები ○ public - მეთოდები <p>class Lift</p> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <ul style="list-style-type: none"> ■ SackisiSartuli ■ SachiroSartuli ■ GavliliSartuli ■ SulSartulebi ■ Person mgzavri; </div> <ul style="list-style-type: none"> ○ LoadMgzavri() ○ IniciateNewFloorRequest() ○ ReportStatistic() ○ WriteLine() ○ Abs() 	<pre> 1 using System; 2 namespace ConsoleApp_Lift 3 { 4 class Lift 5 { 6 private int SackisiSartuli = 1; 7 private int SachiroSartuli = 0; 8 private int GavliliSartuli = 0; 9 private int SulSartulebi = 0; 10 public int i = 1; 11 private Person mgzavri; 12 public void LoadMgzavri() 13 { 14 mgzavri = new Person(); 15 } 16 public void InitiateNewFloorRequest() 17 { 18 SachiroSartuli = mgzavri.NewFloorRequest(); 19 GavliliSartuli = Math.Abs(SackisiSartuli - 20 SachiroSartuli); 21 Console.WriteLine("{0,2}. Sackisi: {1,2} 22 Sachiro: {2,2} Gavlili: {3,2} ", i.ToString(), 23 SackisiSartuli.ToString(), 24 SachiroSartuli.ToString(), 25 GavliliSartuli.ToString()); 26 // Math.Abs(SackisiSartuli - SachiroSartuli); 27 SulSartulebi += GavliliSartuli; 28 SackisiSartuli = SachiroSartuli; 29 i++; 30 } 31 public void ReportStatistic() 32 { 33 Console.WriteLine("\n====>>> Sul gavlili 34 sartulebi: " + SulSartulebi); 35 } 36 } 37 }</pre>
--	--

<p>class Person</p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <ul style="list-style-type: none"> • randomNumberGenerator; </div> <ul style="list-style-type: none"> ○ Person() ○ NewFloorRequest() 	<p>30 31 32 33 34 35 36 37 38 39 40 41 42 43 44</p>	<pre> } class Person { private System.Random randomNumberGenerator; public Person() // კონსტრუქტორი { randomNumberGenerator = new System.Random(); } public int NewFloorRequest() { // აბრუნებს არჩეულ შემთხვევით // რიცხვს 1-60 დიაპაზონში return randomNumberGenerator.Next(1,60); } } </pre>
<p>• ობიექტი</p> <p>class Shenoba</p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <ul style="list-style-type: none"> • lifti_1 </div> <ul style="list-style-type: none"> ○ LoadMgzavri() ○ InitiateNewFloorRequest() ○ ReportStatistic() ○ WriteLine() 	<p>45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60</p>	<pre> class Shenoba // კლასი შენობა { private static Lift lifti_1; private static void Main() { lifti_1 = new Lift(); lifti_1.LoadMgzavri(); Console.WriteLine(" samgzavro sartulebi \n =====\n"); lifti_1.InitiateNewFloorRequest(); lifti_1.InitiateNewFloorRequest(); lifti_1.InitiateNewFloorRequest(); lifti_1.InitiateNewFloorRequest(); lifti_1.InitiateNewFloorRequest(); lifti_1.ReportStatistic(); } } </pre>

ღოსტინგი_2.1: კონსოლის რეჟიმი

2.4) პროგრამის ლისტინგის ანალიზი

N	დანიშნულება
4	Lift - კლასის განსაზღვრების დასაწყისი
6	SackisiSartuli - ცვლადის გამოცხადება int -ტიპით, private-წვდომის სპეციფიკატორით და საწყისი მნიშვნელობით=1
11	Mgzavri - ცვლადის გამოცხადება, რომელიც შეიცავს Person- კლასის ობიექტს. Person- კლასი ასრულებს მგზავრის როლს Lift-კლასთან მიმართებაში
12	LoadMgzavri() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
14	Person-კლასის ახალი (new) ობიექტის შექმნა. ეს ობიექტი მიენიჭება ცვლადს - mgzavri
16	InitiateNewFloorRequest() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
18	mgzavri-ობიექტისთვის NewFloorRequest() მეთოდის გამოძახება. ამ მეთოდით დაბრუნებული მნიშვნელობა მიენიჭება ცვლადს SachiroSartuli
19	ერთი მგზავრობის შემდეგ იანგარიშება გავლილი სართულების რაოდენობა
20	ეკრანზე გამოიტანება: საწყისი_სართული, საჭირო_სართული, გავლილი_სართულების_რაოდენობა
21	იანგარიშება სულ გავლილი სართულების რაოდენობა ლიფტის მუშაობის მთელი სეანსის დროს
22	ლიფტის საწყისი სართულის მნიშვნელობას ენიჭება მისი ბოლო გაჩერების სართულის მნიშვნელობა
23	ლიფტის ამუშავების მომდევნო ბიჯის ინკრემენტი

„პროგრამული სისტემების მენეჯმენტი: ლაბორატორიული პრაქტიკუმი“

26	ReportStatistic() მეთოდის გამოძახებით ეკრანზე გამოიტანება სტატისტიკა SulSartulebi ცვლადით
31	Person კლასის განსაზღვრების დასაწყისი
33	randomNumberGenerator ცვლადის გამოცხადება System.Random კლასის ობიექტის შესანახად
34	სპეციალური მეთოდის (კონსტრუქტორის !) განსაზღვრების დასაწყისი, რომელიც გამოიძახება ავტომატურად Person კლასის ობიექტის შექმნის დროს
36	System.Random-კლასის ახალი ობიექტის შექმნა და მისი მინიჭება randomNumberGenerator - ცვლადზე
39	int ტიპის NewFloorRequest() მეთოდის განსაზღვრა. ის Person-კლასის ინტერფეისის ნაწილია
42	პერსონა (ვირტუალური მგზავრი) ლიფტში ირჩევს საჭირო სართულს (შემთხვევით რიცხვთა გენერატორი ასრულებს ამ ფუნქციას) დიაპაზონში [1-60]
45	Shenoba კლასის აღწერა
47	Shenoba კლასში გამოცხადებულია Lifti ტიპის ცვლადი Lifti_1. Shenoba კლასი კომპოზიციურ კავშირშია Lift კლასთან (ნახ.3.2)
48	Main() მეთოდის აღწერის დასაწყისი
49	Lifti კლასის ახალი ობიექტის შექმნა და მისი მინიჭება lifti_1 ცვლადზე
50	lifti_1 ობიექტისთვის LoadMgzavri() მეთოდის გამოძახება
51- 56	lifti_1 ობიექტისთვის .IniciateNewFloorRequest() მეთოდის გამოძახება 5-ჯერ
57	lifti_1 ობიექტისთვის . ReportStatistic() მეთოდის გამოძახება (შედეგების გამოსაცემად ეკრანზე)

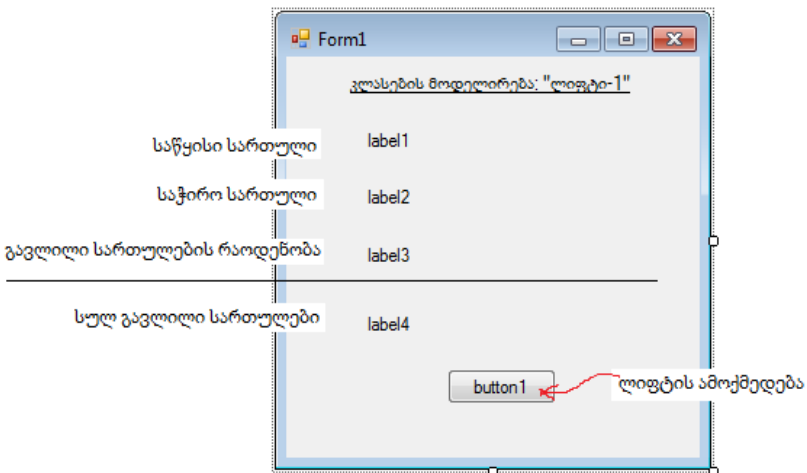
2.5) პროგრამის მუშაობის შედეგები:

2.3 ნახაზზე ნაჩვენებია განხილული პროგრამის შესრულების შედეგები კონსოლის რეჟიმში.

```
file:///C:/C#2010/ConsoleLift/ConsLift1/Con...
=====
savgzavro sartulebi
=====
1.! Sackisi: 1! Sachiro: 32! Gavlili: 31
2.! Sackisi: 32! Sachiro: 58! Gavlili: 26
3.! Sackisi: 58! Sachiro: 11! Gavlili: 47
4.! Sackisi: 11! Sachiro: 37! Gavlili: 26
5.! Sackisi: 37! Sachiro: 23! Gavlili: 14
===>>>          Sul gavlili sartulebi: 144
```

ნახ.2.3

ამოცანა_2.2: განხილული ამოცანისათვის ავგოთ პროგრამული კოდი კლასების საფუძველზე ვინდოუსის ფორმის რეჟიმში. 2.4 ნახაზზე ნაჩვენებია სამუშაო ფანჯარა, რომელიც Form კლასის Form1 ობიექტია. მასზე განთავსებულია ოთხი label (1,2,3,4) და ერთი button1, რომლითაც მოდელირდება ლიფტის ამოქმედება (ლიფტის გამომახება, როცა პერსონა გარეთაა ან სართულის არჩევა ლიფტის დილაკით, როცა პერსონა ლიფტშია, ანუ მგზავრია).

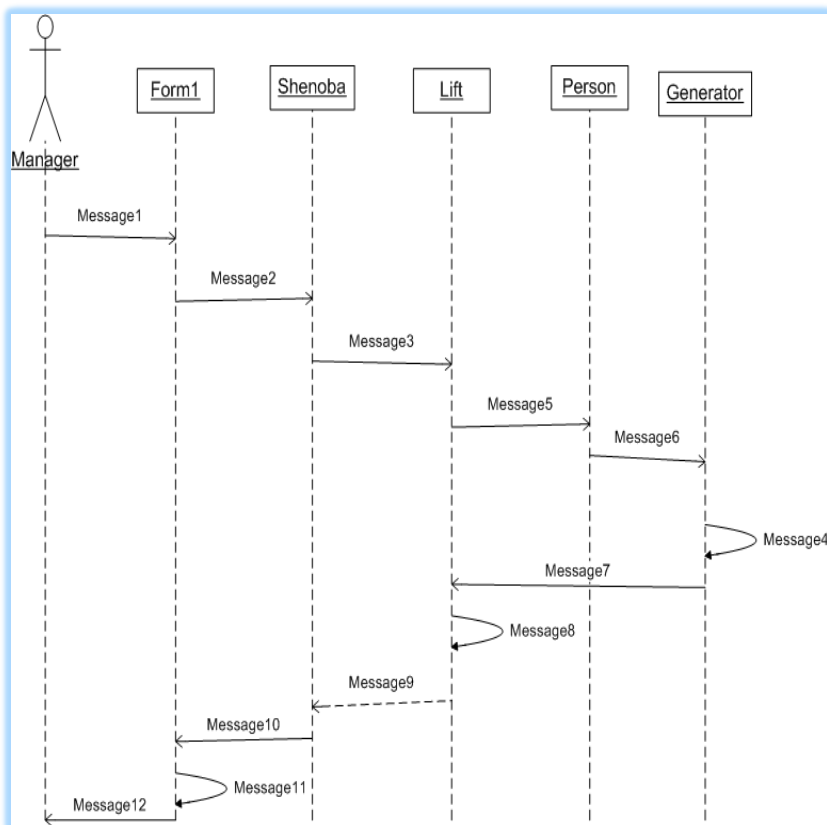


ნახ.2.4

```
// ლისტინგი 2.2--Form1 ელემენტებით და button1-ის კოდი ---
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormLift
{
    public partial class Form1 : Form
    {
        Shenoba shenoba_1; // კლასი Shenoba უნდა შეიქმნას
        public Form1() { InitializeComponent(); }
        private void button1_Click(object sender, EventArgs e)
        { // shenoba_1 ობიექტის შექმნა
            shenoba_1 = new Shenoba(label1, label2, label3, label4);
        }
    }
}
```

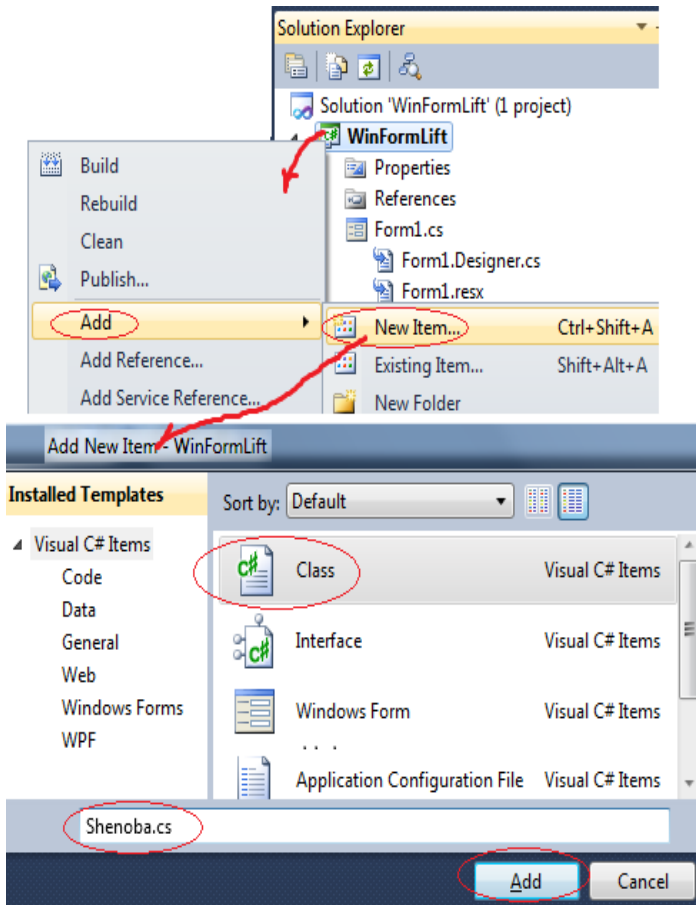
გარდა Form კლასისა (Form1 ობიექტით), რომელიც ასრულებს პროგრამის მომხმარებლის ინტერფეისის ფუნქციას, ლიფტის მუშაობის პროცესის ობიექტ-ორიენტირებული მოდელის ასაგებად საჭიროა კლასები: Shenoba, Lift და Person.

ამ კლასების თვისებები და ფუნქციონალობა ჩვენ ზემოთ აღვწერეთ. ახლა საჭიროა ავაგოთ სცენარი „ლიფტის მუშაობა“, რომლისთვისაც გამოვიყენებთ UML-ის მიმდევრობითობის (Sequence) დიაგრამას [8,9].



ნახ.2.5

ახლა შევექმნათ დანარჩენი კლასები და აღვწეროთ მათი ფუნქციონალობები. Solution Explorer-ში დავამატოთ (Add new Items) ახალი კლასები, როგორც ეს 2.6 ნახაზზეა ნაჩვენები.



ნახ.2.6

Shenoba კლასის კოდი მოცემულია 2.3_ ლისტინგში.

```
// ლისტინგი_2.3 --- კლასი Shenoba -----  
using System;  
using System.Windows.Forms;  
namespace WinFormLift
```

```
{
    public class Shenoba
    {
        static Lift lift_1 = new Lift();

        public Shenoba(Label label1, Label label2, Label
                                label3, Label label4)
        {
            lift_1.LoadMgzavri();
            lift_1.InitiateNewFloorRequest(label1,label2, label3);
            lift_1.ReportStatistic(label4);
        }
    }
}
```

Lift კლასის პროგრამული კოდი მოცემულია 2.4_ლისტინგში.

// ლისტინგი_2.4 --- კლასი Lift -----

```
using System;
using System.Windows.Forms;
namespace WinFormLift
{
    public class Lift
    {
        private int SackisiSartuli = 1;
        private int SachiroSartuli = 0;
        private int GavliSartulebi = 0;
        private int SulSartulebi = 0;
        public int i;
        private Person mgzavri;

        public void LoadMgzavri()
```

```
{
    mgzavri = new Person();
}
public void InitiateNewFloorRequest(Label label1,
                                    Label label2, Label label3)
{
    SachiroSartuli = mgzavri.NewFloorRequest();
    GavlilSartulebi = Math.Abs(SackisiSartuli - SachiroSartuli);
    label1.Text = "საწყისი სართული: " + SackisiSartuli.ToString();
    label2.Text = "საჭირო სართული: " + SachiroSartuli.ToString();
    label3.Text = "გავლილი სართულები: " + GavlilSartulebi.ToString();
    SulSartulebi += GavlilSartulebi;
    SackisiSartuli = SachiroSartuli;
    i++;
}
public void ReportStatistic(Label label4)
{
    label4.Text = "სულ გავლილი სართულები: " + SulSartulebi;
}}}
```

Person კლასის პროგრამა მოცემულია 3.5_ლისტინგში.

```
// ლისტინგი_3.5 --- კლასი Person -----
using System;
using System.Windows.Forms;
namespace WinFormLift
{
    public class Person
    {
        private System.Random randomNumberGenerator;
        public Person() // კონსტრუქტორი
        {
            randomNumberGenerator = new System.Random(); }
        public int NewFloorRequest()
```

```
{ return randomNumberGenerator.Next(1, 60);  
}}}
```

პროგრამის ამუშავების შემდეგ მიიღება 2.7 ნახაზზე ნაჩვენები შედეგები.

კლასების მოდელირება: "ლიფტი-1"

საწყისი სართული: 1

საჭირო სართული: 45

გავლილი სართულები: 44

სულ გავლილი სართულები: 44

button1

ნახ.2.7-ა. 1-ელი ბიჯი

კლასების მოდელირება: "ლიფტი-1"

საწყისი სართული: 45

საჭირო სართული: 32

გავლილი სართულები: 13

სულ გავლილი სართულები: 57

button1

ნახ.2.7-ბ. მე-2 ბიჯი

კლასების მოდელირება: "ლიფტი-1"

საწყისი სართული: 32

საჭირო სართული: 27

გავლილი სართულები: 5

სულ გავლილი სართულები: 62

button1

ნახ.2.7-გ. მე-3 ბიჯი

კლასების მოდელირება: "ლიფტი-1"

საწყისი სართული: 34

საჭირო სართული: 49

გავლილი სართულები: 15

სულ გავლილი სართულები: 202

button1

ნახ.2.7-დ. მე-10 ბიჯი,

და ა.შ.

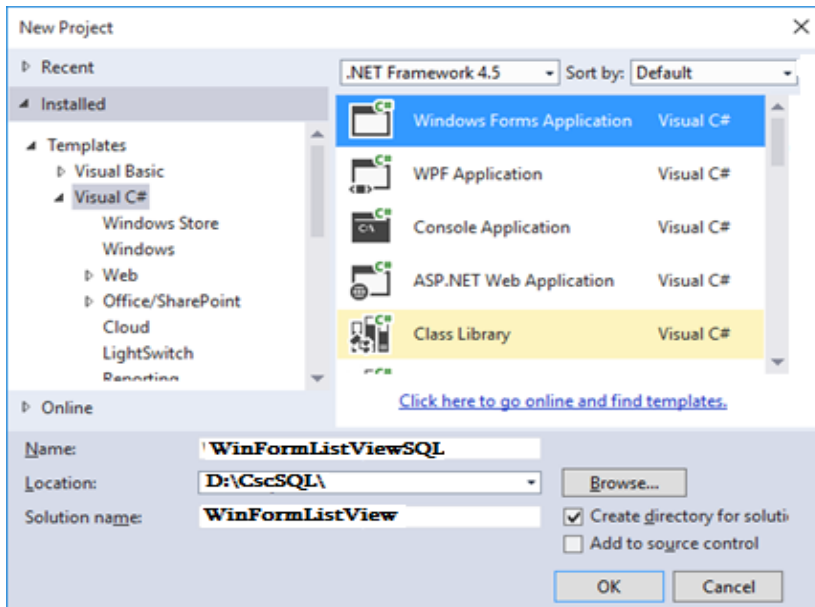
3. ლაბორატორიული სამუშაო N3

მონაცემთა განახლების მეთოდები ListView-ში

მიზანი: ვიზუალური დაპროგრამების C# ენის ListView კლასის საფუძველზე მონაცემთა მენეჯმენტის განხილვა, SQL Server ბაზის განახლების პროცედურების შესწავლა Insert, Update და Delete მეთოდებით.

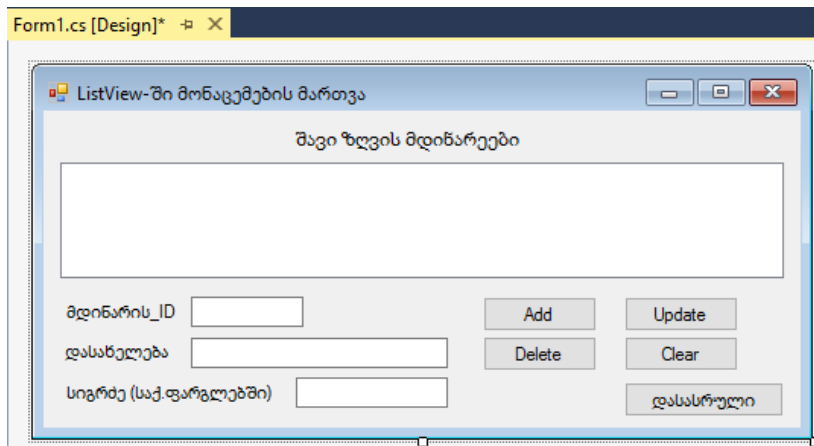
პროგრამული პროექტის აგების მაგალითისათვის განვიხილოთ ამოცანა საქართველოს აკვატორიაში შავი ზღვის მდინარეების მონაცემთა ბაზის შექმნის და მისი ცხრილების შევსების და განახლების ინტერფეისული პროგრამის დამუშავება.

Visual Studio.NET 2013/15 გარემოში შევქმნათ ახალი პროექტი სახელით: WinFormListViewSQL (ნახ,3,1).



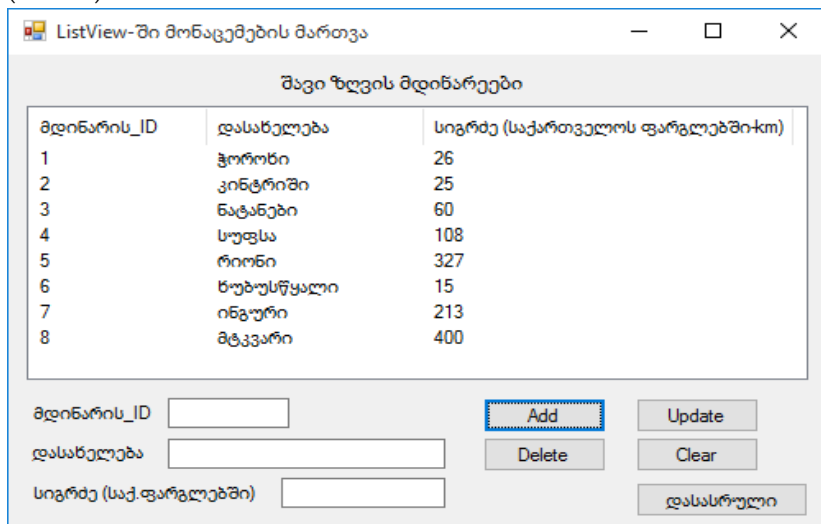
ნახ.3.1

ჩვენს მიერ დაპროექტებული ინტერფეისის ფორმა მოცემულია 3.2 ნახაზზე.



ნახ.3.2

მონაცემთა ბაზის ჩანაწერები გამოიტანება ListView სვეტებში (ნახ.3.3).



ნახ.3.3

მე-6 სტრიქონში შეცდომაა და საჭიროა ცვლილების განხორციელება Update მეთოდით. ვირჩევთ სტრიქონს (ნახ.3.4).

მდინარის_ID	დასახელება	სიგრძე (საქართველოს ფარგლებში-km)
1	ჭოროზი	26
2	კინტრიში	25
3	ნატანები	60
4	სუფსა	108
5	რიონი	327
6	ხუბუსწყალი	15
7	ინგური	213
8	მტკვარი	400

მდინარის_ID:

დასახელება:

სიგრძე (საქ.ფარგლებში):

Buttons: Add, Update, Delete, Clear, დასასრული

ნახ.3.4

„ხუბუსწყალი“ შეცვალეთ „ხობისწყალი“-თ (ნახ.3.5).

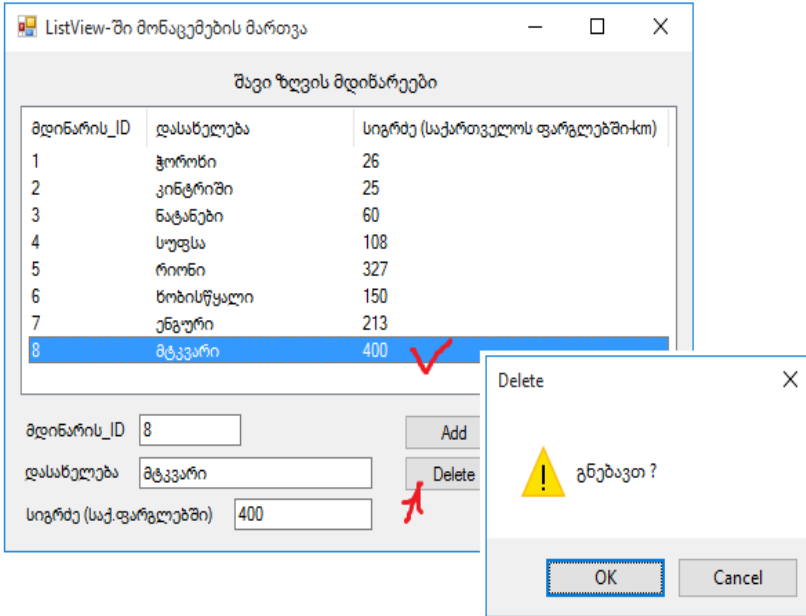
მდინარის_ID	დასახელება	სიგრძე (საქართველოს ფარგლებში-km)
1	ჭოროზი	26
2	კინტრიში	25
3	ნატანები	60
4	სუფსა	108
5	რიონი	327
6	ხობისწყალი ✓	150
7	ინგური	213
8	მტკვარი	400

მდინარის_ID:

Buttons: Add, Update

ნახ.3.5

„მტკვარი“ არაა შავი ზღვის მდინარე, ამიტომ ის უნდა წაიშალოს Delete მეთოდით (ნახ.3.6).



ნახ.3.6

3.1_ლისტინგში მოცემულია ListView-ში მონაცემების დამატების, ცვლილების და წაშლის მეთოდების კოდები.

// -- ლისტინგი_3.1 ---- ListView-ის Add, Update Delete მეთოდები --

```
using System;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WinFormListViewSeqD
{
```

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        // ListView-ის თვისებები -----
        listView1.View = View.Details;
        listView1.FullRowSelect = true;
        // ListView-ის ველების სიგანე-----
        listView1.Columns.Add("მდინარის_ID", 50);
        listView1.Columns.Add("დასახელება", 150);
        listView1.Columns.Add("სიგრძე (საქართველოს
            ფარგლებში-კმ)", 50);
    }
    // Add
    private void add(String id, String name, String length)
    {
        //row---
        String[] row = {id, name, length};
        ListViewItem item = new ListViewItem(row);
        listView1.Items.Add(item);
    }
    // update-----
    private void update()
    {
        listView1.SelectedItems[0].SubItems[0].Text =
            idTxt.Text;
        listView1.SelectedItems[0].SubItems[1].Text =
            nameTxt.Text;
        listView1.SelectedItems[0].SubItems[2].Text =
            lengthTxt.Text;
        // clear txt
        idTxt.Text = "";
        nameTxt.Text = "";
        lengthTxt.Text = "";
    }
    // delete -----
    private void delete()
    {
        if (MessageBox.Show("გნებავთ წაშლა ?", "DELETE",
            MessageBoxButtons.OKCancel,
            MessageBoxIcon.Warning) == DialogResult.OK)
    }
}
```

```
{
listView1.Items.RemoveAt(listView1.SelectedIndices[0]);
    // clear txt
    idTxt.Text = "";
    nameTxt.Text = "";
    lengthTxt.Text = "";
}
}
private void button1_Click(object sender, EventArgs e)
{
    add(idTxt.Text, nameTxt.Text, lengthTxt.Text);
    // clear txt
    idTxt.Text = "";
    nameTxt.Text = "";
    lengthTxt.Text = "";
}
private void button2_Click(object sender, EventArgs e)
{
    update();
}
private void button3_Click(object sender, EventArgs e)
{
    delete(); }
private void button4_Click(object sender, EventArgs e)
{
    listView1.Items.Clear();
    // clear txt
    idTxt.Text = "";
    nameTxt.Text = "";
    lengthTxt.Text = ""; }
private void listView1_MouseClick(object sender, MouseEventArgs e)
{
    idTxt.Text=listView1.SelectedItems[0].SubItems[0].Text;
    nameTxt.Text=listView1.SelectedItems[0].SubItems[1].Text;
    lengthTxt.Text=listView1.SelectedItems[0].SubItems[2].Text;
}
private void button5_Click(object sender, EventArgs e)
{
    Close(); }
}
}
```

4. ლაბორატორიული სამუშაო N4

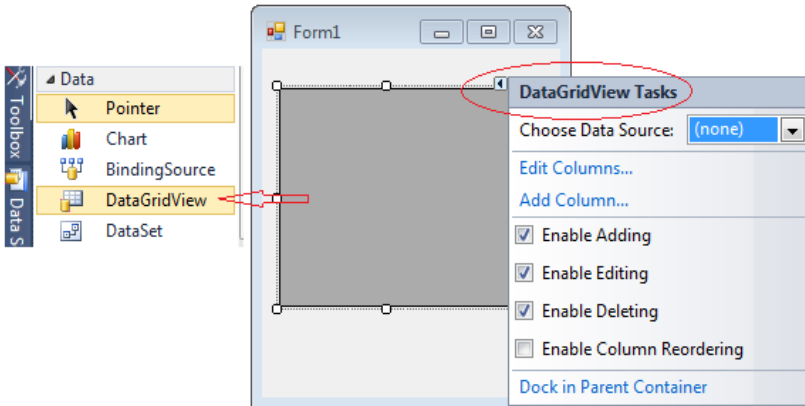
ცხრილებთან მუშაობა - DataGridView

მიზანი: DataGridView მართვის ელემენტის შესწავლა ცხრილებთან (ბრტყელ ფაილებთან) სამუშაოდ.

1. თეორიული ნაწილი

ცხრილები (Tables), რომლებიც სტრიქონებისა და სვეტებისგან შედგება, საუკეთესო საშუალებაა მონაცემთა ორგანოზომილებიანი ველების, მასივების წარმოსადგენად. C# ენაში ასეთი ობიექტების ასახვის მიზნით გამოიყენება მართვის ელემენტი, ტიპით DataGridView. ამ ელემენტს, პრაგმატული თვალსაზრისით, დიდი გამოყენება აქვს მონაცემთა ბაზების სისტემებში (ADO.NET), რასაც ჩვენ მომდევნო თავში დავუბრუნდებით.

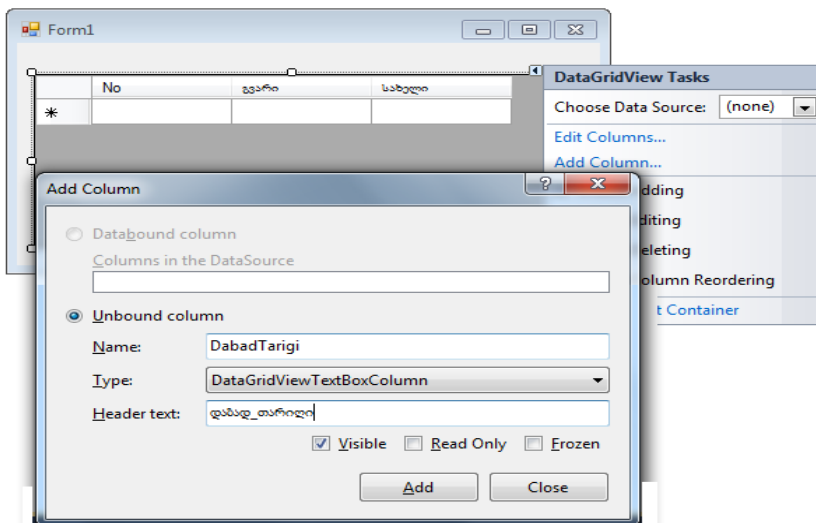
ახლა განვიხილოთ ცხრილებთან მუშაობის ვიზუალური საშუალებანი. 4.1 ნახაზზე მოცემულია ToolBox-დან Form1-ფორმაზე გადმოტანილი DataGridView ელემენტი და საწყისი სიტუაცია.



ნახ.4.1

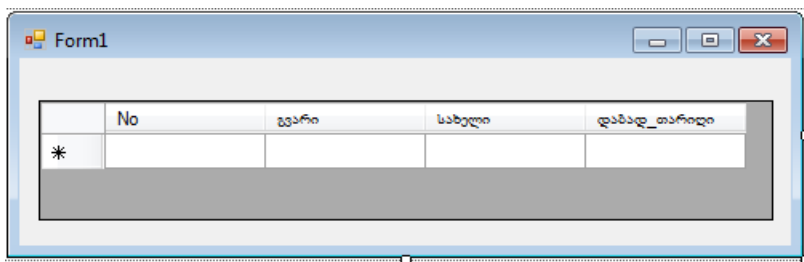
დიალოგურ რეჟიმში ცხრილის ველების (სვეტების) შესატანად ვირჩევთ Add Columns და გადავდივართ 4.2 ნახაზზე

მოცემულ ფანჯარაში. შვიტანოთ მიმდევრობით „სტუდენტები“-ს ატრიბუტები, მაგალითად, No, First_Name (გვარი), Last_Name (სახელი), Birth_data (დაბად_თარიღი) და ა.შ.



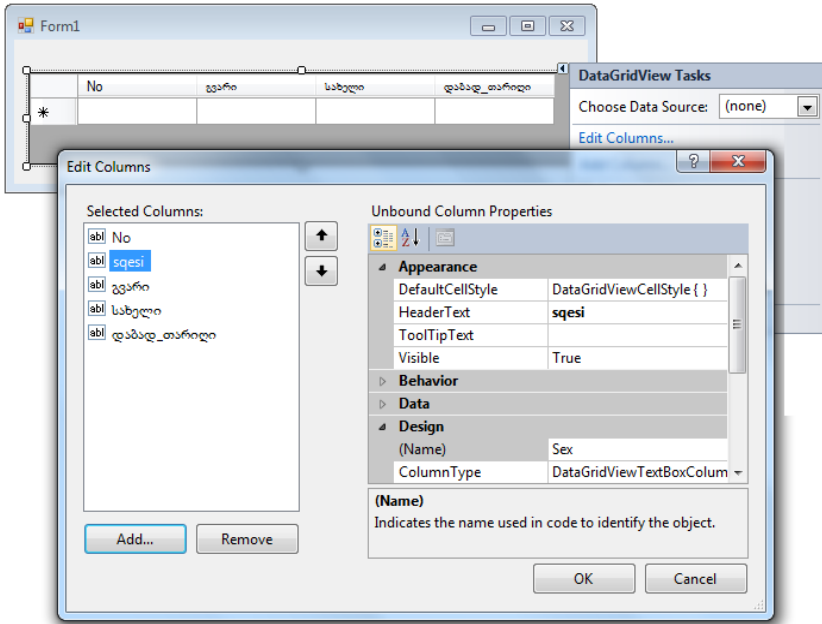
ნახ.4.2

პროგრამის ამუშავების შემდეგ მივიღებთ 4.3 ნახაზზე მოცემულ ცხრილს.

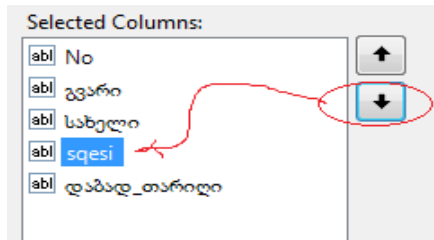


ნახ.4.3

შეტანილი ველების კორექტირებისათვის ვიყენებთ Edit Columns პუნქტს. ჩავამატოთ ველები Sex (სქესი) ან შევასწოროთ უკვე ჩაწერილი დასახელებები. მაგალითად, 4.3 ნახაზზე, ედიტორის ფანჯარაში გვინდა ველი sqesi შევცვალოთ ქართული შრიფტით და ამასთანავე იგი გადავიტანოთ ველი „სახელი“-ს შემდეგ. 4.4 და 4.5 ნახაზებზე ნაჩვენებია ეს შემთხვევები.

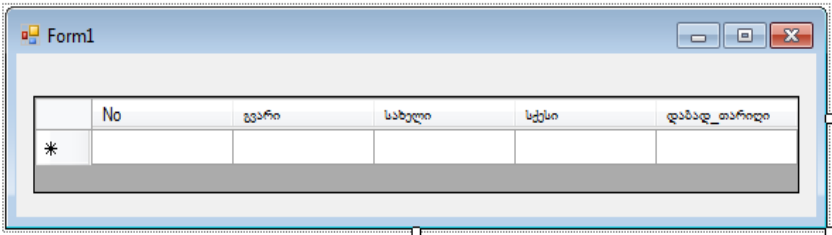


ნახ.4.4



ნახ.4.5

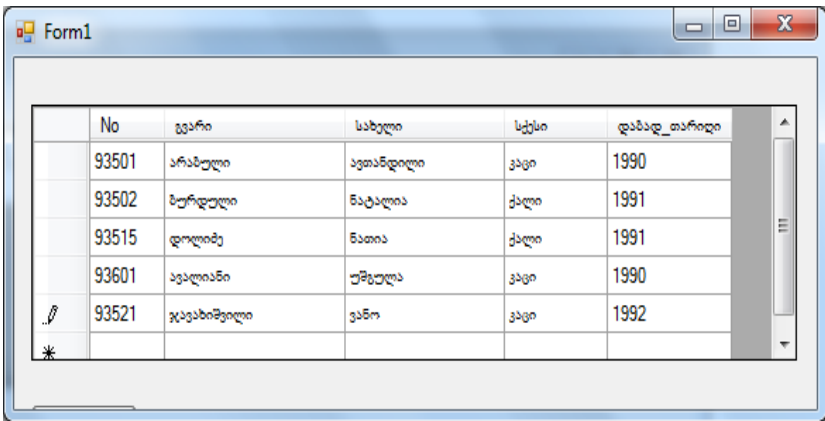
კოდის მუშაობის ახალი შედეგი მოცემულია 4.6 ნახაზზე.



No	გვარი	სახელი	სქესი	დაბად_თარიღი
*				

ნახ.4.6

აქვე შეიძლება მონაცემთა სტრიქონების (Rows) შეტანა ველების ქვეშ. მაგალითი ნაჩვენებია 4.7 ნახაზზე.



No	გვარი	სახელი	სქესი	დაბად_თარიღი
93501	არაზული	აფთაწდილი	კაცი	1990
93502	ხურდული	ნატალია	ქალი	1991
93515	დოლიძე	ნათია	ქალი	1991
93601	ავალიანი	უმბულა	კაცი	1990
93521	ჯაგახიშვილი	ვანო	კაცი	1992
*				

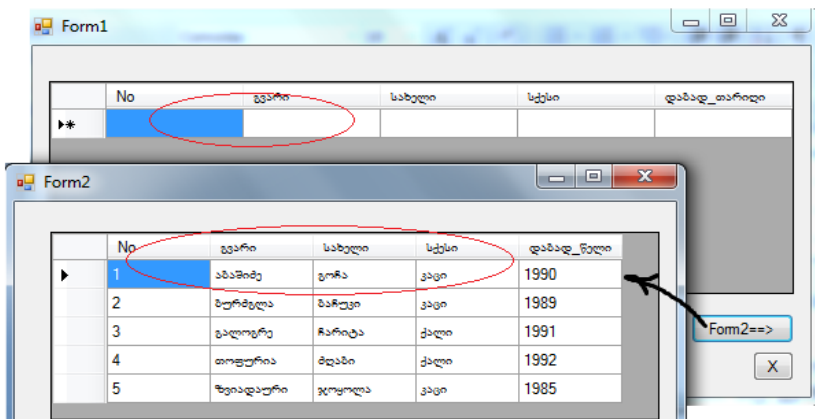
ნახ.4.7

პროგრამის დამთავრებისა და ხელახალი ამუშავების შემდეგ შეტანილი მონაცემები იკარგება, ანუ არაა შენახული მეხსიერებაში. თუ გვინდა, რომ პროექტის გაშვებისას მონაცემები ჩაიტვირთოს პროგრამულად, მაშინ ან უნდა გამოვიყენოთ მონაცემთა ბაზასთან კავშირი (იხ. მომდევნო თავი), ან კოდის Form2_Load მეთოდში უნდა ჩავწეროთ შემდეგი სტრიქონები (ლისტინგი 11_1).

```
// ლისტინგი_4.1 --- DataGridView -----
private void Form2_Load(object sender, EventArgs e)
{
    int i;
    // ველების (სვეტების) შევსება -----
    dataGridView1.Columns.Add("No", "No");
    dataGridView1.Columns.Add("FirstName", "გვარი");
    dataGridView1.Columns.Add("LastName", "სახელი");
    dataGridView1.Columns.Add("Sex", "სქესი");
    dataGridView1.Columns.Add("Dab_Celi", "დაბად_წელი");
    // ველების სიგანის დაყენება -----
    for (i = 0; i < dataGridView1.Columns.Count; i++)
        dataGridView1.Columns[i].Width = 75;
    // სტრიქონების შევსება -----
    dataGridView1.Rows.Add("1", "აბაშიძე", "გოჩა", "კაცი", "1990");
    dataGridView1.Rows.Add("2", "ბურძღლა", "ზაჩუცი", "კაცი", "1989");
    dataGridView1.Rows.Add("3", "გალოგრე", "ჩარიტა", "ქალი", "1991");
    dataGridView1.Rows.Add("4", "თოფურია", "მღაზი", "ქალი", "1992");
    dataGridView1.Rows.Add("5", "ზვიადაური", "ჯოყილა", "კაცი", "1985");
}

```

პროგრამის ამუშავებით მიიღება ასეთი შედეგი (ნახ.4.8).



ნახ.4.8

როგორც აღვნიშნეთ, კოდში ხისტადაა შეტანილი კონკრეტული მონაცემები სტუდენტების შესახებ. ნებისმიერი დამატება ან ცვლილება მოითხოვს პროგრამის გადაკეთებას, რაც არაა რეკომენდებული. ამის თავიდან აცილება შესაძლებელია მონაცემთა ბაზის გამოყენებით. შედეგიდან ჩანს, რომ Form1 ცარიელია, ხოლო Form2 შევსებულია საწყისი მონაცემებით.

ახლა განვიხილოთ ჩვენი კოდის მაგალითით DataGridView ცხრილში შეტანილი მონაცემების საფუძველზე მომხმარებლის მოთხოვნების პროგრამული დამუშავების შესაძლებლობანი.

ამოცანა_4.1: ვიპოვოთ სახელები და გვარები ყველა 1990 წელს დაბადებული მამაკაცი სტუდენტის.

მოთხოვნის ფორმალური მხარე მდგომარეობს „გვარი“ ველის მნიშვნელობების გამობეჭდვაში, ველი „სქესი“=“კაცი“ მნიშვნელობისთვის.

საჭიროა Form2-ზე დავდოთ ბუტონი და მივაბათ მას 4.2_ლისტინგის პროგრამული კოდი.

//--ლისტინგი_4.2--DataGridView-ში სტრიქონების ამორჩევა

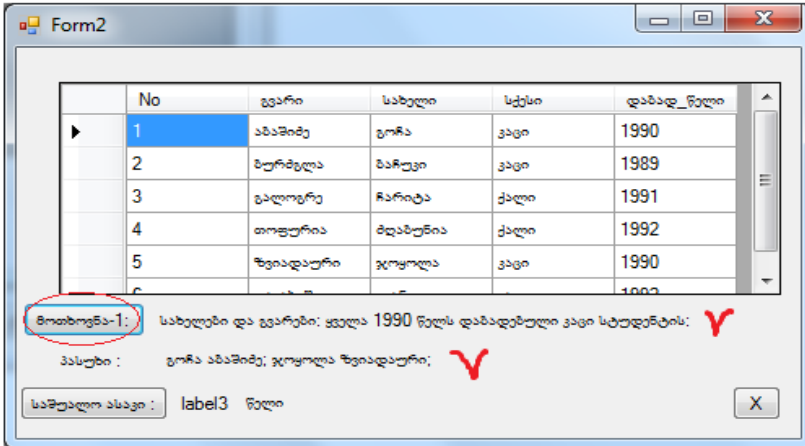
//-- პირობით ----

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = " ";
    label2.Text = "სახელები და გვარები: ყველა 1990 წელს
                    დაბადებული კაცი სტუდენტის:";
    for (int i = 0; i < dataGridView1.Rows.Count; i++)
    {
        if (dataGridView1.Rows[i].Cells[3].Value == "კაცი" &&
            dataGridView1.Rows[i].Cells[4].Value == "1990")
        {
```

```

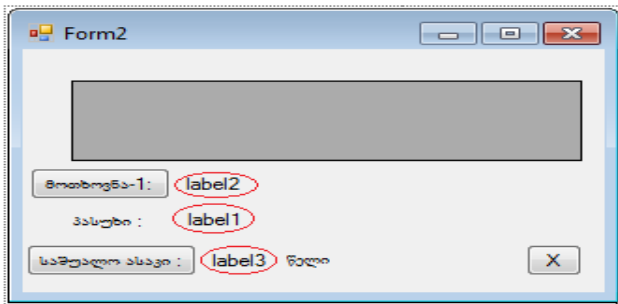
label1.Text += dataGridView1.Rows[i].Cells[2].Value
+ " " +dataGridView1.Rows[i].Cells[1].Value + "; ";
}
}
} // შედეგები გამოტანილია 4.9 ნახაზზე.

```



ნახ.4.9

ამოცანა_4.2: შევადგინოთ კოდი დილაკისთვის „საშუალო ასაკი“, რომელიც გამოიტანს label3-ში ყველა სტუდენტის საშუალო ასაკის მნიშვნელობას (ნახ. 4.10). 11_3 ლისტინგზე მოცემულია ეს კოდი.



ნახ.4.10

```
// ლისტინგი_4.3 --- DataGridView საშუალო ასაკის ანგარიში -----
private void button2_Click(object sender, EventArgs e)
{
    int Birth_Year, Averag_Age, Sum=0;
    DateTime now = DateTime.Now; // მიმდინარე თარიღი
    int age,a,b;
    for (int i = 0; i < dataGridView1.Rows.Count; i++)
    {
        Birth_Year=Convert.ToInt32(dataGridView1.Rows[i].Cells[4].Value);
        a = now.Year; b = Birth_Year; age = a - b;
        Sum += age;
    }
    Averag_Age = Sum / dataGridView1.Rows.Count;
    label3.Text = Averag_Age.ToString();
}
}
```

შედეგები ასახულია 4.11 ნახაზზე.

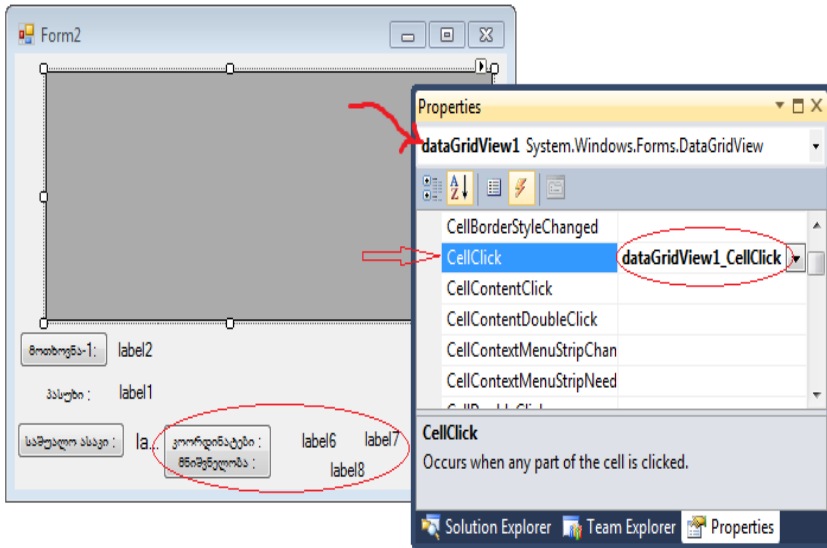
No	გვარი	სახელი	სქესი	დაბად_წელი
1	ანაშიძე	გოჩა	კაცი	1990
2	შერშელა	ბაჩუცი	კაცი	1989
3	გალოგოე	ჩარიტა	ქალი	1991
4	თოჭტურია	მღამუშია	ქალი	1992
5	წვიადაური	ჯოჯოლა	კაცი	1990
6	ჯგუბაშივილი	ივანე	კაცი	1992

მოთხოვნა-1: სახელები და გვარები: ყველა 1990 წელს დაბადებული კაცი სტუდენტის:

პასუხი : გოჩა ანაშიძე; ჯოჯოლა წვიადაური;

საშუალო ასაკი : 20 წელი ✓

ნახ.4.11



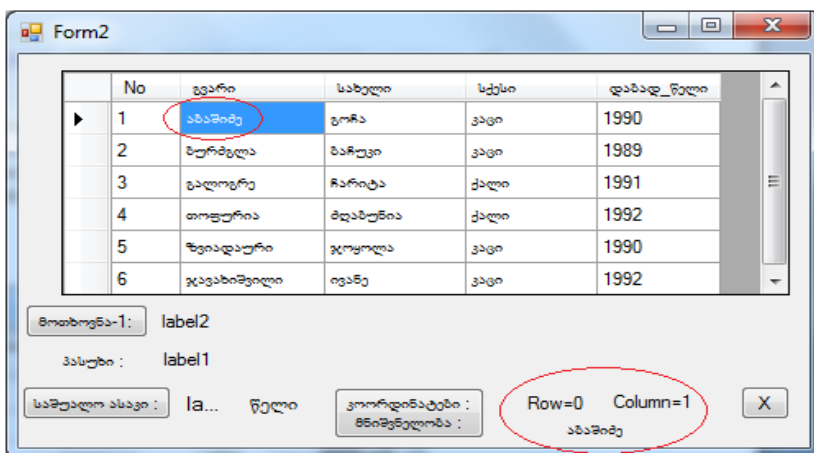
ნახ.4.12

// ლისტინგი_4.4 ---- ცხრილის კოორდინატები ----

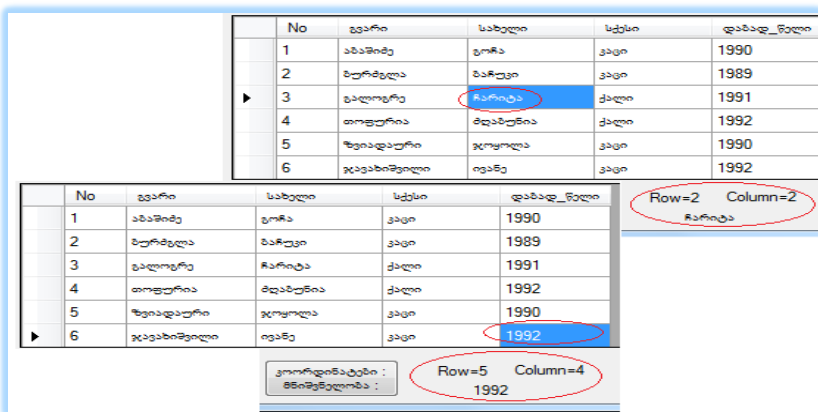
```
private void dataGridView1_CellClick(object sender,
                                     DataGridViewCellEventArgs e)
{
    label8.Text = "";
    label6.Text = "Row="+e.RowIndex.ToString();
    label7.Text = "Column="+e.ColumnIndex.ToString();
    if (e.RowIndex>=0 && e.ColumnIndex >=0)
        label8.Text += dataGridView1.Rows[e.RowIndex].
                               Cells[e.ColumnIndex].Value;
}
```

შდეგები ასახული 4.13-ა,ბ ნახაზზე.

„პროგრამული სისტემების მენეჯმენტი: ლაბორატორიული პრაქტიკუმი“



ნახ.4.13-ა



ნახ.4.13-ბ

დავალება:

ააგეთ ცხრილი „წიგნები“, რომელსაც ექნება შემდეგი ველები: შიფრი, დასახელება, ავტორი, გამოცემის_წელი, გამომცემლობა, ქალაქი, ჟანრი, ენა, გვერდების_რაოდ., ფასი. შეიტანეთ 10 სტრიქონი და განახორციელეთ მასში წიგნის ძებნა ავტორით, წიგნების ძებნა ჟანრით, დაალაგეთ წიგნები გამოცემის წლის კლებადობით.

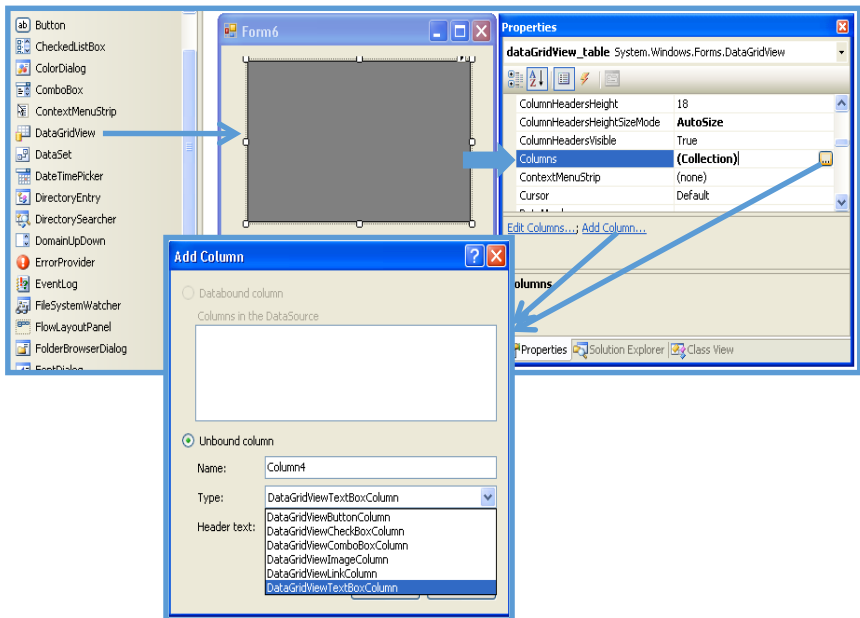
5. ლაბორატორიული სამუშაო N5

DataGridViewComboBoxColumn და DataGridViewTextBoxColumn კლასები ცხრილებთან სამუშაოდ

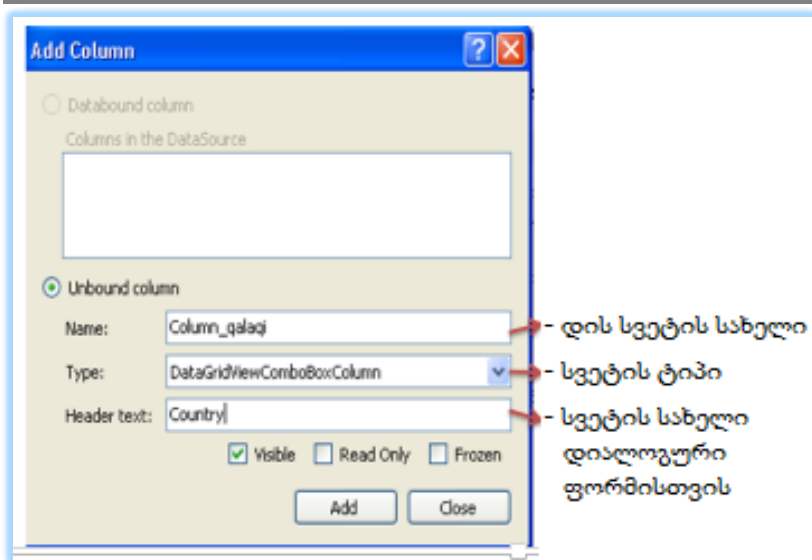
მიზანი: მონაცემთა ბაზის ცხრილებთან მუშაობის დროს DataGridViewComboBoxColumn და DataGridViewTextBoxColumn კლასების გამოყენების შესწავლა.

1. თეორიული ნაწილი

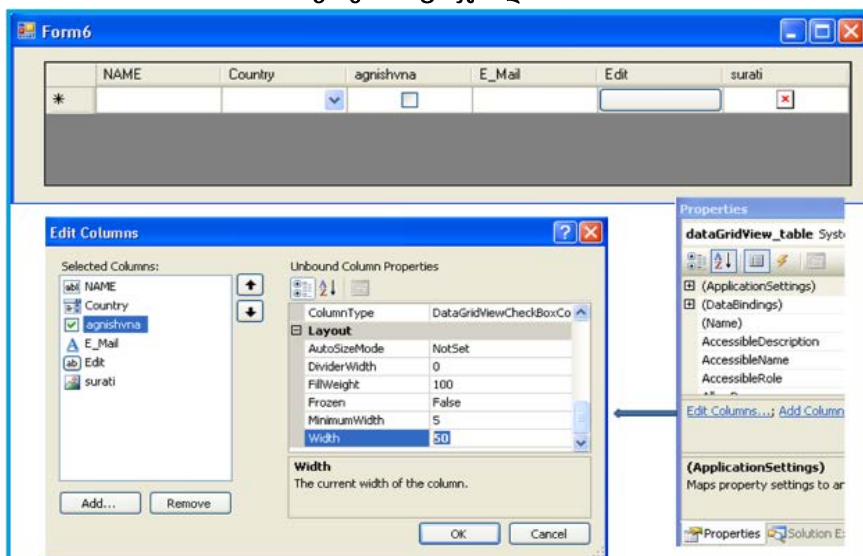
განიხილება შემდეგი საკითხები: DataGridView კომპონენტის სვეტების შექმნა სხვადასხვა ტიპის კომპონენტებით, DataGridView კომპონენტში DataGridViewComboBoxColumn და DataGridViewTextBoxColumn სვეტების გამოყენება, DataGridView-ს სხვადასხვა ტიპის ველების შევსება DataTable ვირტუალური ცხრილიდან.



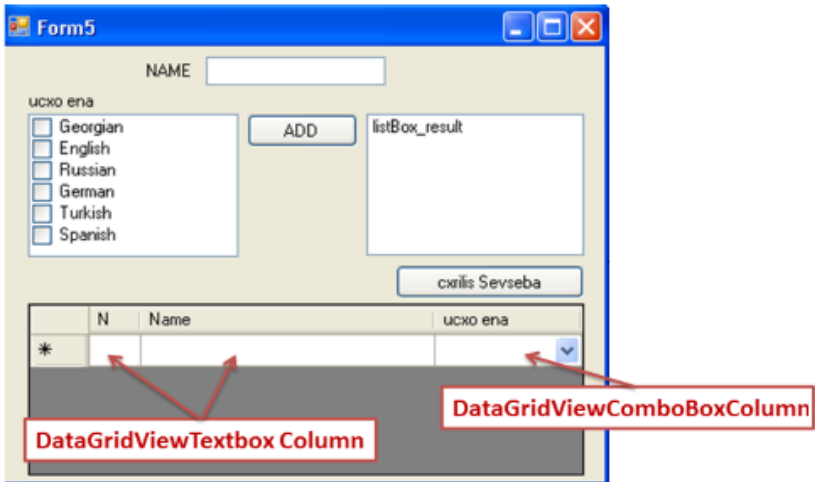
ნახ.5.1. DataGridView -ს სვეტების შექმნა სხვადასხვა ტიპის კომპონენტით



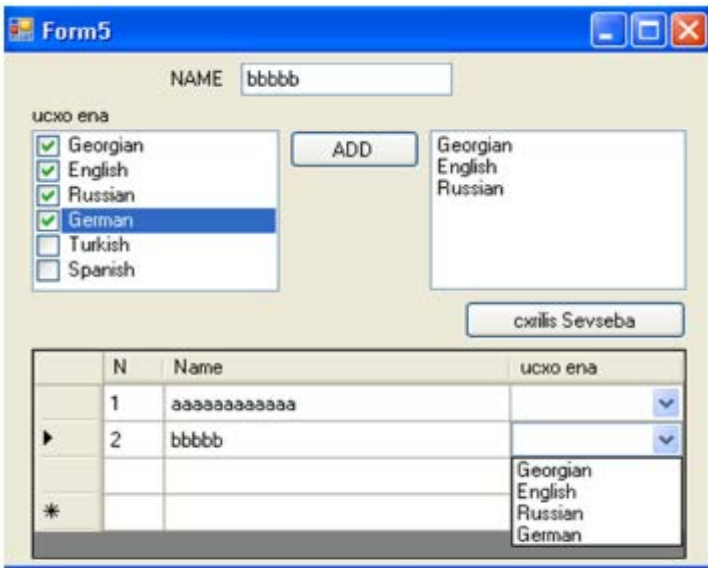
ნახ.5.2. DataGridView-ში სხვადასხვა ტიპის სვეტების დამატება თვისებათა ფანჯრიდან



ნახ.5.3. სვეტების დამატების შედეგი



ნახ.5.4. CheckedListbox და TextBox კომპონენტებიდან
DataGridView-ის ველების შევსება



ნახ.5.5. შედეგი

```

public partial class Form5 : Form
{
    DataTable table = new DataTable();
    public Form5()
    {
        InitializeComponent();
        basa();
    }
}
    
```

↑ მეთოდის გამოძახება

```

void basa()
{
    table = new DataTable();
    DataColumn col_N = new DataColumn("", typeof(int));
    DataColumn col_name = new DataColumn("", typeof(string));
    DataColumn col_combo = new DataColumn("", typeof(List<String>));
    table.Columns.AddRange(new DataColumn[] { col_N, col_name, col_combo });
}
    
```

ვირტუალური ცხრილის შექმნა

← სტრიქონული ტიპის მასივის სვეტი

ნახ.5.6. DataTable ვირტუალური ცხრილის შექმნა DataGridView-ს ველების შესავსებად

```

private void button_table_Click(object sender, EventArgs e)
{
    String name = textBox_name.Text;
    table = this.table;
    List<String> checked_list = new List<string>();
    int lists_count = checkedListBox_list_tipy.Items.Count;
    for (int i = 0; i < lists_count; ++i)
    {
        if (checkedListBox_list_tipy.GetItemCheckState(i) == CheckState.Checked)
        {
            String result = (String)checkedListBox_list_tipy.Items[i];
            checked_list.Add(result);
        }
    }
    int tableN = table.Rows.Count;
    table.Rows.Add(new object[] { tableN + 1, name, checked_list });
    fill_Table(table);
}
    
```

სტრიქონული ტიპის მასივის შექმნა

სტრიქონული ტიპის მასივში ელემენტების დამატება

dataGridView1 ცხრილში მონაცემების დამატების მეთოდის გამოძახება

ნახ.5.7. ვირტუალური ცხრილის შევსება CheckedListBox და TextBox კომპონენტებიდან

```
public void fill_Table(DataTable table)
{
    int tableN = table.Rows.Count;
    if (tableN != 0)
    {
        for (int i = 0; i < tableN; i++)
        {
            dataGridView1.Rows.Add();
            dataGridView1.Rows[i].Cells[0].Value = (int)table.Rows[i][0];
            dataGridView1.Rows[i].Cells[1].Value = (String)table.Rows[i][1];
            List<String> list_result = (List<String>)table.Rows[i][2];
            DataGridViewComboBoxCell cells = (DataGridViewComboBoxCell)dataGridView1.Rows[i].Cells[2];
            cells.Items.Clear();
            for (int j = 0; j < list_result.Count; j++)
            {
                cells.Items.Add(list_result[j]);
            }
        }
    }
}
```

**ნახ.5.8. fill_Table(table) მეთოდი dataGridView-ს ველების შესავსებად
DataTable ვირტუალური ცხრილიდან**

დავალება:

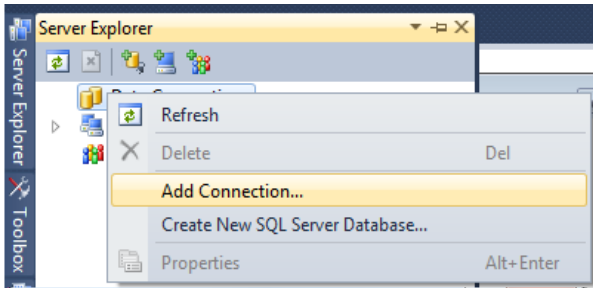
- ააგეთ პროგრამული პროექტი განხილული მაგალითისთვის და გამართეთ იგი;
- განახორციელეთ რომელიმე კონკრეტული საპრობლემო სფეროს მონაცემებისთვის ცხრილის აგება და DataGridViewComboBoxColumn და DataGridViewTextBoxColumn კლასების გამოყენება;

6. ლაბორატორიული სამუშაო N6

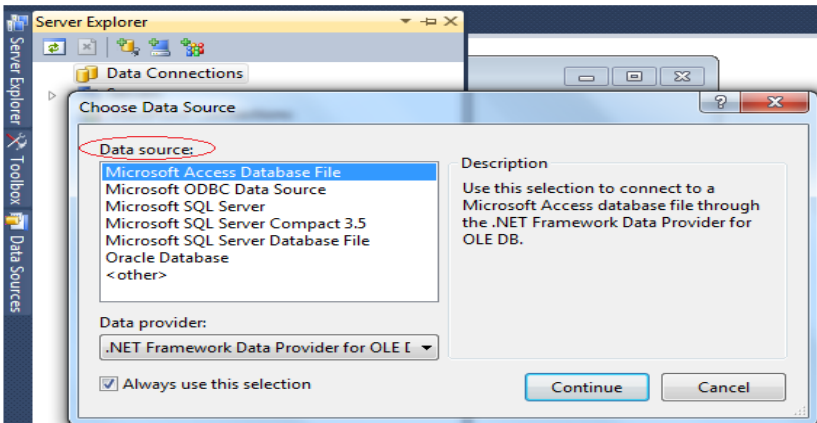
C# აპლიკაციის მუშაობა Ms Access ბაზასთან

მიზანი: მონაცემთა ბაზების მართვის სისტემების და მომხმარებელთა ინტერფეისების ერთობლივი გამოყენების პროექტების აგების პროცესების მენეჯმენტის შესწავლა Ms Access-პაკეტით.

სისტემის მენიუდან View | Server Explorer-ით გამოვიტანოთ ფანჯარა (ნახ.6.1) და ავირჩიოთ Add Connection.

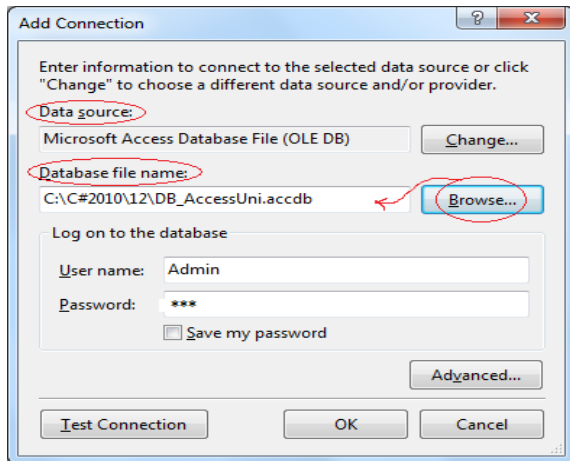


ნახ.6.1

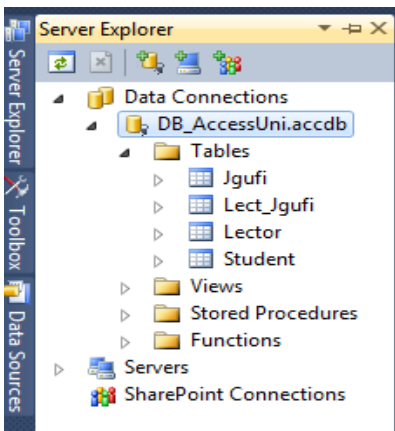


ნახ.6.2

6.2 ნახაზზე Data Source ველში ავირჩიეთ სტრიქონი Microsoft Access Database File და Continue. მივიღებთ 6.3 ფანჯარას, რომელშიც Browse ღილაკით გამოვიძახებთ კატალოგის მართვის დიალოგის ფანჯარას და მივუთითებთ ჩვენს მიერ წინასწარ მომზადებულ Ms Access-ის ბაზის ფაილს. მაგალითად, როგორც ეს Database file name ველშია ჩაწერილი.



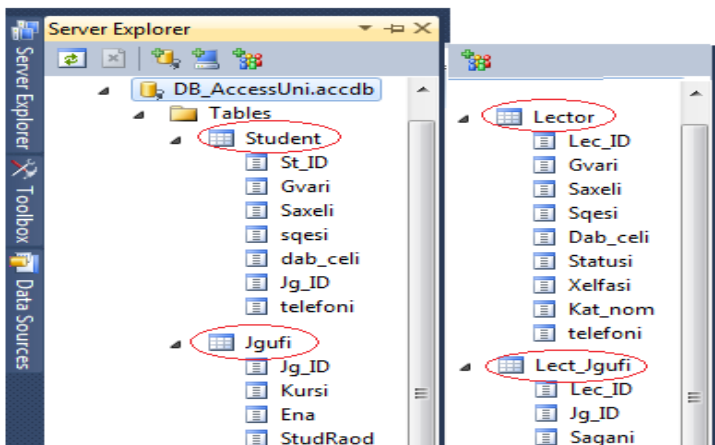
ნახ.6.3



ნახ.6.4

აქვე, საჭიროების შემთხვევაში, მიეთითება User name და Password. ამის შემდეგ Server Explorer-ში გამოჩნდება 6.4 ნახაზზე მოცემული სურათი.

როგორც ვხედავთ, Data Connection-ში უნივერსიტეტის მონაცემთა ბაზის ფაილი - **DB_AccessUni.accdb** გამოჩნდა, რომელიც შედგება ოთხი ცხრილისგან: Jgufi, Lect_Jgufi, Lector და Student. ცხრილები შედგება ველებისგან, რომელთაგან ერთ-ერთი გასაღებურია (ინდექსი): Lec_ID, St_ID, Jg_ID და ერთივე შედგენილი გასაღებია ორი ატრიბუტით: Lec_ID+Jg_ID (ნახ.6.5).

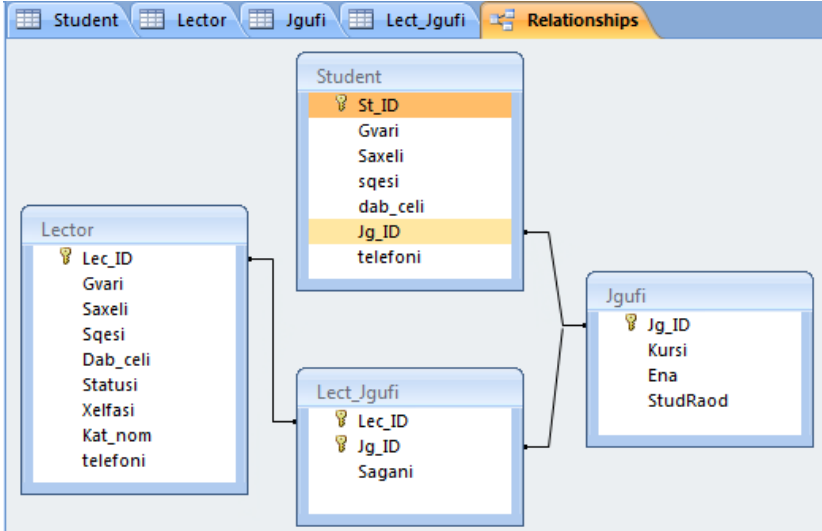


ნახ.6.5

რელაციური კავშირები მათ საფუძველზეა აგებული (ნახ.6.6).

ცხრილში **სტუდენტი** პირველადი გასაღებური ატრიბუტია St_ID ინდექსი, ხოლო მისი მეორადი გასაღებია Jg_ID, რომლითაც იგი უკავშირდება ცხრილს **ჯგუფი**, პირველადი ინდექსით Jg_ID. ესაა კავშირი 1:N, რომელიც ასახავს ბიზნეს-წესს (არსებულ კანონზომიერებას), რომ ერთი სტუდენტი შეიძლება იყოს მხოლოდ ერთ ჯგუფში და ერთ ჯგუფში შეიძლება იყოს რამდენიმე (N) სტუდენტი. ასევე, **ლექტორი** ასწავლის რამდენიმე (N) ჯგუფს, მაგრამ ჯგუფსაც ჰყავს რამდენიმე (M) ლექტორი. ესაა M:N კავშირი. მისი რეალიზაცია არაა შესაძლებელი **ლექტორი**-ს და **ჯგუფი**-ს პირდაპირი კავშირით (განმეორებადი ველების პრობლემა !). ამისათვის შემოტანილია დამატებითი ცხრილი (რელაცია)

ლექტორი_ჯგუფი. მასში შედგენილი ინდექსი იქმნება Lec_ID+Jg_ID, რომლებიც უკავშირდება ცალკ-ცალკე **ლექტორს** და **ჯგუფს** (ნახ.6.7).



ნახ.6.6

St_ID	Gvari	Saxeli	sqesi	dab_cel	Jg_ID	telefoni
8	აკოფოვი	რობერტინო	კაცი	1989	108936	297-11-11-11
1	ალავიძე	ალეკო	კაცი	1990	108935	222-22-20
2	ბურდული	ნინო	ქალი	1991	108935	137-33-33
3	ბურმგლა	დიტო	კაცი	1989	108935	599-10-20-20
4	გაბედავა	ვახტანგ	კაცი	1992	108935	333-67-89
5	გაბელია	ცუცა	ქალი	1992	108935	222-45-67
13	გალოგრე	ხვიჩა	კაცი	1989	108937	270-44-44
6	დანელია	მიმოზა	ქალი	1990	108935	577-44-44-45
9	დოლიძე	რიჩარდი	კაცი	1990	108936	597-34-56-78
14	დუნდუა	გოჩა	კაცი	1991	108937	599-22-33-44
15	ვასაძე	სოკრატე	კაცი	1985	108937	577-33-67-55
10	ზარანდია	მუმნი	კაცი	1980	108936	597-12-23-34
11	თოფურია	ძლავი	ქალი	1991	108936	577-10-10-10
12	კეკელია	კეკელა	ქალი	1992	108936	579-30-30-30
7	ხვიტია	ძუკუ	კაცი	1992	108935	593-45-67-89
16	ჯალაღონია	მაცი	კაცი	1992	108937	577-99-00-00

ნახ.6.7-ა

„პროგრამული სისტემების მენეჯმენტი: ლაბორატორიული პრაქტიკუმი“

Student	Lector	Jgufi	Lect_Jgufi	Relationships				
Lec_ID	Gvari	Saxeli	Sqesi	Dab_cel	Statusi	Xelfasi	Kat_norr	telefoni
6	ბარათელი	ანი	ქალი	1970	ასოც.პროფესორი	600	94 599-23-23-23	
7	კუცია	თეა	ქალი	1987	ლაბორანტი	280	94 577-12-13-14	
8	გაბედავა	ომიკო	კაცი	1950	სრ.პროფესორი	900	94 577-33-55-22	
9	დევალი	დავითი	კაცი	1950	სრ.პროფესორი	900	86 599-99-90-90	
10	ფიფია	კორი	კაცი	1960	ასოც.პროფესორი	650	86 233-33-46	
11	მეფარია	თვარისა	ქალი	1980	ას.პროფესორი	450	94 593-55-55-55	
12	ნიწიძე	ლია	ქალი	1980	ასოც.პროფესორი	600	94 577-78-89-90	
13	ოდინარია	ოდინა	კაცი	1956	ას.პროფესორი	450	86 236-37-38	
14	სამხარაძე	ვანშაშა	კაცი	1970	ასოც.პროფესორი	690	51 577-88-99-00	

ნახ.6.7-ბ

Student	Lector	Jgufi	Lect_Jgufi	Relatio
Jg_ID	Kursi	Ena	StudRaod	
108050	2	ქართული	29	
108051	2	ქართული	30	
108059	2	რუსული	12	
108835	4	ქართული	29	
108836	4	ქართული	25	
108935	3	ქართული	28	
108936	3	ქართული	30	
108937	3	ქართული	17	
108940	3	ინგლისური	20	

ნახ.6.7-გ

Student	Lector	Jgufi	Lect_Jgufi	Relation
Lec_ID	Jg_ID	Sagani		
5	5	კომპიუტერის არქიტექტურა		
8	6	კომპიუტერის არქიტექტურა		
8	7	სერვერული ტექნოლოგიები		
8	8	სერვერული ტექნოლოგიები		
8	9	კომპიუტერის არქიტექტურა		
9	7	მათემატიკა		
9	8	მათემატიკა		
10	12	მონაცემთა ბაზები		
10	13	მონაცემთა ბაზები		

ნახ.6.7-დ

ჩანაწერის წინ „+“ სიმბოლო ხსნის კავშირს მეორე, იერარქიულად დაქვემდებარებულ ცხრილთან (ნახ.6.8).

Student	Lector	Jgufi	Lect_Jgufi	Katedr
Lec_ID	Gvari	Saxeli	Sqesi	
6	ბარათელი	ანი	ქალი	
7	კუცია	თეა	ქალი	
8	გაბედავა	ომიკო	კაცი	
Jg_ID	Sagani			Add
5	კომპიუტერის არქიტექტურა			
6	კომპიუტერის არქიტექტურა			
7	სერვერული ტექნოლოგიები			
8	სერვერული ტექნოლოგიები			
9	კომპიუტერის არქიტექტურა			
*				
9	დვალი	დავითი	კაცი	
10	ფიფია	კოჩი	კაცი	

ნახ.6.8

ამგვარად, მონაცემთა ბაზა DB_AccessUni.acce მზადაა. ახლა განვიხილოთ C# პროგრამიდან მონაცემთა ბაზასთან წვდომის საკითხი. ეს პროცესი შედგება ოთხი ბიჯისგან:

- მონაცემთა ბაზასთან მიერთება (რაც ზემოთ განვიხილეთ Server Explorer->Data Connection-ში);

- SQL-ბრძანების გადაცემა მონაცემთა ბაზაზე;
- SQL-ბრძანების შეფასება (და შესრულება);
- მონაცემთა ბაზასთან კავშირის დახურვა.

SQL-ბრძანება წარმოადგენს სტრუქტურირებული მოთხოვნების ენაზე დაწერილ სკრიპტს, რომელიც გასაგებია მონაცემთა ბაზების მართვის სისტემისთვის და ასრულებს მას. ძირითადად, არსებობს ორი ტიპის მოთხოვნა:

- select : მონაცემთა ამორჩევის SQL-ბრძანება;
- insert, delete, update : მონაცემთა ბაზაში ცვლილებების განსახორციელებელი SQL-ბრძანებები.

C# პროგრამულ პროექტს მონაცემთა ბაზასთან სამუშაოდ სჭირდება სახელსივრცე OleDb, რომელიც using.System.Data.OleDb ბრძანებითაა რეალიზებული.

SQL-ბრძანებები Ms_Access ბაზაში გადაიგზავნება OleDb სახელსივრცის OleDbCommand კლასის ობიექტით. ამ კლასის ორი მნიშვნელოვანი თვისებაა: Connection (დავალემა ბაზასთან დასაკავშირებლად, საითაც გაიგზავნება SQL-მოთხოვნა) და CommandText (თვით SQL ბრძანების ტექსტი).

მოთხოვნის ტიპებისგან დამოკიდებულებით (არჩევითი, ცვლილებების), OleDbCommand კლასს გააჩნია შემდეგი მეთოდები:

ExecuteReader() - აქვს select მოთხოვნის გაგზავნის და შედეგების მიღების ფუნქცია;

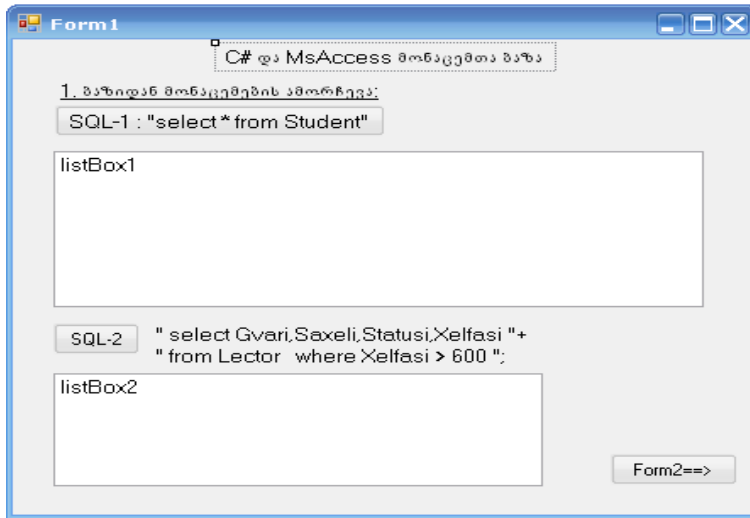
ExecuteNonQuery() - ემსახურება ცვლილების (insert, delete, update) აქციის ჩატარებას და რიცხვის მიღებას, რომელიც გვიჩვენებს, თუ მონაცემთა ბაზის რამდენ ჩანაწერს შეეხო ეს პროცედურა.

მონაცემთა ბაზიდან select-მოთხოვნით ამორჩეული ჩანაწერები (ველებით და მნიშვნელობებით) ინახება OleDb სახელსივრცის OleDbReader კლასის ობიექტში. განვიხილოთ კონკრეტული მაგალითი.

ამოცანა_6.1: DB_AccessUni.acce უნივერსიტეტის მონაცემთა ბაზაში: 1 - ვნახოთ ყველა სტუდენტის ყველა ატრიბუტის მნიშვნელობა (ანუ მთლიანი ინფორმაცია); 2 - ვიპოვოთ იმ ლექტორთა გვარი, სახელი, სტატუსი და ხელფასი, რომელთა ხელფასი მეტია 600 ლარზე.

6.9-ა ნახაზზე 1-ელ მოთხოვნას შეესაბამება SQL-1, ხოლო მეორეს კი - SQL-2 “select” კონსტრუქცია. ისინი ორ დილაკზეა მიმაგრებული. საილუსტრაციო მაგალითში შედეგები გამოიტანება listBox1 და listBox2 ველებში.

C#-პროგრამის კოდი ამ ორი დილაკისთვის მოცემულია 6.1_ლისტინგში.



ნახ.6.9-ა

// ლისტინგი 6.1 – C# + Ms Access -----

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
    OleDbConnection con = new OleDbConnection();
```

```
    OleDbCommand cmd = new OleDbCommand();
```

```
    OleDbDataReader reader;
```

```
con.ConnectionString =
```

```
    "Provider=Microsoft.ACE.OLEDB.7.0;" +
```

```
    "Data Source=C:\\C#2010\\WinADO\\DB_AccessUni.accdb";
```

```
cmd.Connection = con;
```

```
cmd.CommandText = "select * from Student";
```

```
try
```

```
{
```

```
    con.Open();
```

```
    reader = cmd.ExecuteReader();
```

```
    listBox1.Items.Clear();
```

```
while (reader.Read())
{
    listBox1.Items.Add(reader["St_ID"] + " : " +
        reader["Gvari"] + " : " +
        reader["Saxeli"] + " : " +
        reader["sqesi"] + " : " +
        reader["dab_celi"] + " : " +
        reader["Jg_ID"] + " : " +
        reader["telefoni"]);
}
reader.Close();
con.Close();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
// end button1_click -----

private void button2_Click(object sender, EventArgs e)
{
    OleDbConnection con = new OleDbConnection();
    OleDbCommand cmd = new OleDbCommand();
    OleDbDataReader reader;

    con.ConnectionString =
        "Provider=Microsoft.ACE.OLEDB.7.0;" +
        "Data Source=C:\\C#2010\\WinADO\\DB_AccessUni.accdb";
    cmd.Connection = con;
    cmd.CommandText =
        "select Gvari,Saxeli,Statusi,Xelfasi " +
        "from Lector " +
        "where Xelfasi > 600 ";
```

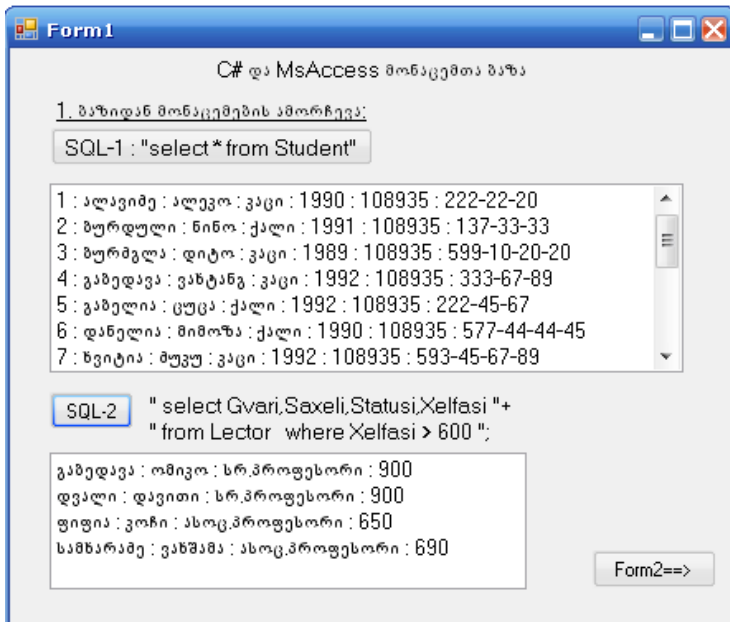
```
try
{
    con.Open();
    reader = cmd.ExecuteReader();
    listBox2.Items.Clear();
    while (reader.Read())
    {
        listBox2.Items.Add( reader["Gvari"] + " : " +
            reader["Saxeli"] + " : " +
            reader["Statusi"] + " : " +
            reader["Xelfasi"]);
    }
    reader.Close();
    con.Close();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
// end button2_click -----
```

შენიშვნა: პროგრამის თავში using-სტრიქონებში უნდა ჩაემატოს:

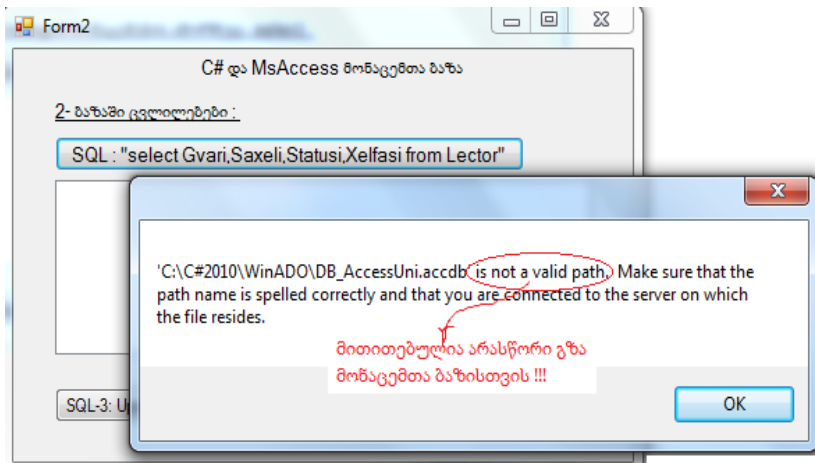
```
using System.Data.OleDb;
```

6.9-ბ ნახაზზე ნაჩვენებია აღნიშნული მოთხოვნების შესრულების შედეგები MsAccess ბაზის Student და Lector ცხრილებიდან. პირველში „ * ”- ნიშნავს „ყველა“ - ველს, ხოლო მეორეში აიღება მხოლოდ „გვეარი,სახელი, სტატუსი და ხელფასი“.

პროგრამაში **try...catch** ბლოკით ხდება მონაცემთა ბაზასთან წვდომის პროცესში შესაძლო შეცდომების აღმოჩენა, რაც აადვილებს პროგრამისტის მუშაობას. მაგალითად, ინფორმაციის მიღება, რომ მონაცემთა ბაზის ფაილი არაა მითითებულ patch-კატალოგში (ნახ.6.10), ან რომ SQL-მოთხოვნის სინტაქსში შეცდომაა და ა.შ.



ნახ.6.9-ბ



ნახ.6.10

Open() მეთოდით ხდება პროგრამის კავშირის გახსნა ბაზასთან. შემდეგ, ExecuteReader() მეთოდით მოთხოვნა გადაეგზავნება ბაზას. შედეგები ბრუნდება OleDbReader კლასით, რაც ხორციელდება reader მიმთითებლით (მაჩვენებლით).

ვინაიდან წინასწარ არაა ცნობილი თუ რამდენი სტრიქონი იქნება შედეგში, გამოიყენება ListBox, რომელიც წინასწარ სუფთავდება.

Read() მეთოდი გვაწვდის ბაზიდან ერთ ჩანაწერს (სტრიქონს) და ამავდროულად სპეც-მაჩვენებლით მიუთითებს მომდევნო ჩანაწერზე. თუ ჩანაწერი ბოლოა, მაშინ მაჩვენებლის მნიშვნელობა ხდება false. ეს მართვა ხორციელდება while ციკლით try-ბლოკში.

ჩანაწერის შიგნით ველების მნიშვნელობები შეესაბამება მათ ნომრებს ან დასახელებებს. შესაძლებელია ასევე ყველა ველის გამოტანა („ *“-ით), რომლებიც სპეც-გამყოფითაა (მაგ., „ :“) დაცილებული. ბოლოს, Reader ობიექტი და კავშირი უნდა დაიხუროს Close() მეთოდით.

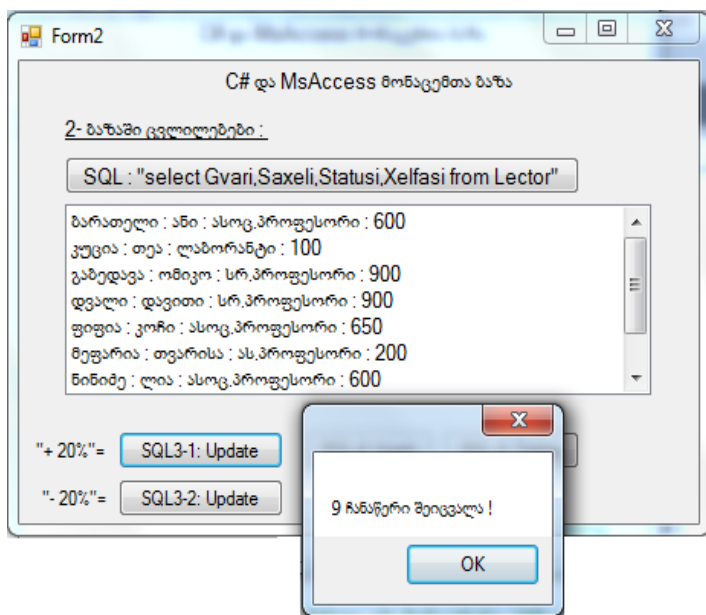
ამოცანა_6.2: მონაცემთა ბაზაში „უნივერსიტეტი“ DB_AccessUni.acce საჭიროა ცვლილებების განხორციელება insert(), delete() და update() მეთოდების გამოყენებით. Form2-ზე მოვათავსოთ შესაბამისი ელემენტები და განვახორციელოთ ტრანზაქციები:

ა) ყველა ლექტორის ხელფასი გაიზარდოს 20%-ით. (ამასთანავე შესაძლებელი უნდა იყოს საწყისი მონაცემების აღდგენა, ანუ შემცირდეს ხელფასები 14.67 %-ით);

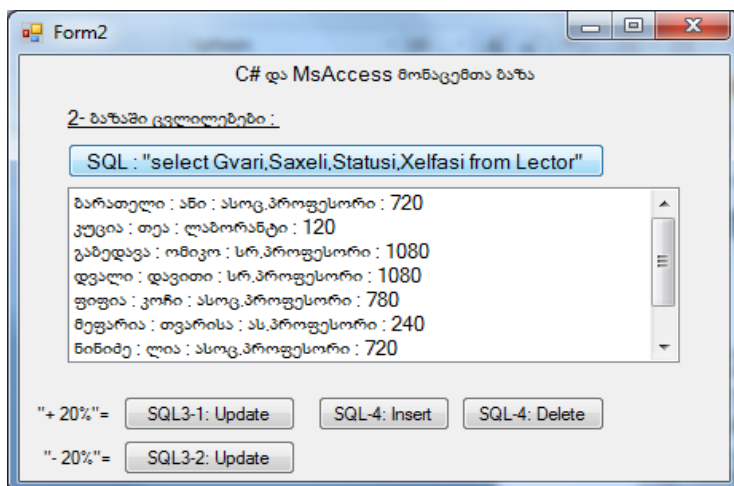
ბ) 108935 ჯგუფში დაემატოს ახალი სტუდენტი გვარით „ახალაძე“, სახელით „ნოუთბუკა“ და ა.შ.;

გ) ამოიშალოს ბაზიდან 108836 ჯგუფი, რომელმაც დაასრულა 4-წლიანი სწავლების კურსი.

ლექტორთა ხელფასების შეცვლილი შედეგები მოცემულია 6.12 ნახაზზე.



ნახ.6.11



ნახ.6.12

SQL3-2 დილაკით შემცირდება ხელფასის რაოდენობა 20%-ით, ანუ აღდგება პირველადი მონაცემები ბაზაში.

შესაბამისი update - კოდი მოცემულია 6.2_ლისტინგში.

```
// ლისტინგი_6.2 --- Ms Acees "update" -----
using System;
using System.Data.OleDb;
using System.Windows.Forms;
namespace WinADO
{
    public partial class Form2 : Form
    {
        public Form2() { InitializeComponent(); }
        private void button1_Click(object sender, EventArgs e)
        {
            OleDbConnection con = new OleDbConnection();
            OleDbCommand cmd = new OleDbCommand();
            int Raod;
            // ხელფასის ზრდა ან შემცირება 20%-ით ----
            con.ConnectionString =
                "Provider=Microsoft.ACE.OLEDB.7.0;" +
                "Data Source=C:\\C#2013\\12\\DB_AccessUni.accdb";
            cmd.Connection = con;
            if(ReferenceEquals(sender,button1))
                // op = "*" or "/" -----
                cmd.CommandText = "update Lector set Xelfasi=Xelfasi * 1.2";
            else
                cmd.CommandText = "update Lector set Xelfasi=Xelfasi / 1.2";
            try
            {
                con.Open();
                Raod = cmd.ExecuteNonQuery();
                MessageBox.Show(Raod + " ჩანაწერი შეიცვალა !");
                con.Close();
            }
        }
    }
}
```

```
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
private void button4_Click(object sender, EventArgs e)
{ OleDbConnection con = new OleDbConnection();
  OleDbCommand cmd = new OleDbCommand();
  OleDbDataReader reader;
  con.ConnectionString =
      "Provider=Microsoft.ACE.OLEDB.7.0;" +
      "Data Source=C:\\C#2013\\12\\DB_AccessUni.accdb";
  cmd.Connection = con;
  cmd.CommandText = "select Gvari,Saxeli,Statusi,
                                                              Xelfasi from Lector";
  try
  {
      con.Open();
      reader = cmd.ExecuteReader();
      listBox1.Items.Clear();
      while (reader.Read())
      {
          listBox1.Items.Add(reader["Gvari"] + " : " +
                              reader["Saxeli"] + " : " +
                              reader["Statusi"] + " : " +
                              reader["Xelfasi"]);
      }
      reader.Close();
      con.Close();
  }
  catch (Exception ex)
  {
      MessageBox.Show(ex.Message);
  }
}
```

კოდში update ტიპის ცვლილებისათვის DB_MsAccessUni.accedb ფაილში გამოყენებული კონსტრუქცია:

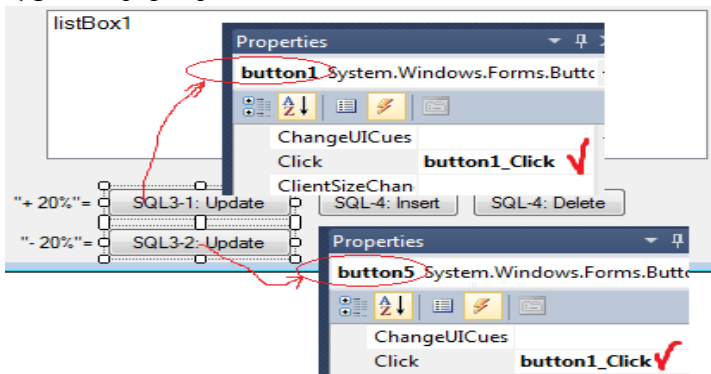
“update Lector set Xelfasi=Xelfasi * 1.2”

რაც ნიშნავს ცხრილის აქტუალიზაციას (update... set), სახელით Lector და ცხრილის ველის (ატრიბუტის) ცვლილების ალგორითმს ($Xelfasi=Xelfasi * 1.2$), რომელიც უნდა შესრულდეს ჩანაწერებზე;

try...catch ბლოკში იხსნება ბაზა (Open-ით) და გამოიძახება მეთოდი ExecuteNonQuery(), რომელიც დააბრუნებს ჩანაწერების რაოდენობას (Raod), რომელთაც შეეხო ცვლილება. ეს ინფორმაცია გამოიტანება MessageBox-ით.

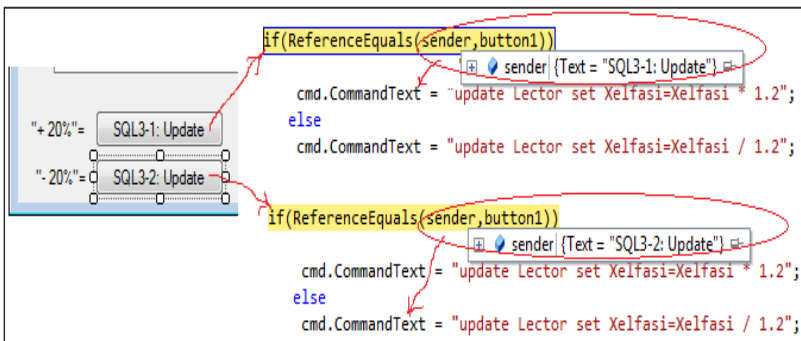
მონაცემთა ბაზის ცხრილში Lector ველისათვის Xelfasi უნდა მოხდეს მნიშვნელობათა გაზრდა 20%-ით (button1: SQL3-1) ან შემცირება (button5: SQL3-2), ნახ.6.11.

ამის განსახორციელებლად კოდში გამოყენებულია მეთოდი ReferenceEquals(sender object). ამ მეთოდით განისაზღვრება - აქვს თუ არა ორ ობიექტს ერთი მისამართი, ანუ ინახება თუ არა ისინი მემსიერების ერთიდაიმავე უჯრედებში. თუ „კი“, მაშინ „true“ და სრულდება გამრავლება (ხელფასის მომატება), ხოლო თუ სხვადასხვა ობიექტია, მაშინ გვექნება “false” და შესრულდება გაყოფა (ხელფასის შემცირება).



ნახ.6.13

კოდის `ReferenceEquals(sender, button1)` მეთოდში, იმისდა მიხედვით, თუ რომელი ბუტონი (SQL3-1 თუ SQL3-2) იქნება არჩეული, `sender`-ს მიენიჭება `button1` ან `button5` მიმთითებლის მნიშვნელობა (ნახ.6.14).



ნახ.6.14

ამგვარად, როდესაც `sender` და `button1` ერთიდაიმავე მისამართზეა, მაშინ ხდება ხელფასის მომატება. დასასრულ, პროგრამის `button4_Click` ლილაკით `listBox1`-ში გამოიტანება `MsAccess` ბაზის `Lector` ცხრილის განახლებული მონაცემები (მომატებული ან დაკლებული ხელფასებით).

დავალება:

- მოცემული მონაცემთა ბაზის მაგალითზე („უნივერსიტეტი“) ააგეთ C# პროგრამული კოდები და გამართეთ პაკეტი. გააკეთეთ დაპროგრამებული მეთოდების ანალიზი;
- ააგეთ რომელიმე საპრობლემო სფეროს მონაცემთა ბაზა ექსპერიმენტის ჩასატერებლად. გამოიყენეთ აგებული პროექტი პროტოტიპის სახით და შექმენით ახალი ინტერფეისული პროგრამა.

7. ლაბორატორიული სამუშაო N7

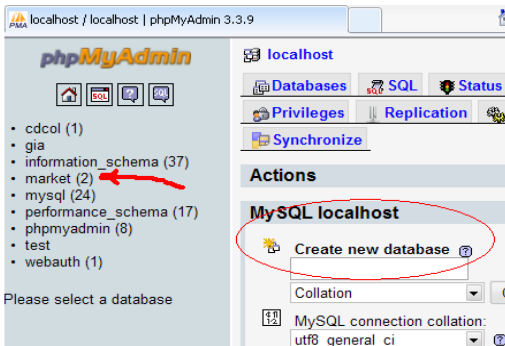
C# აპლიკაციის მუშაობა MySQL ბაზასთან

მიზანი: C# ენაზე აგებული აპლიკაციის მუშაობის შესწავლა MySQL მონაცემთა ბაზების მართვის სისტემასთან.

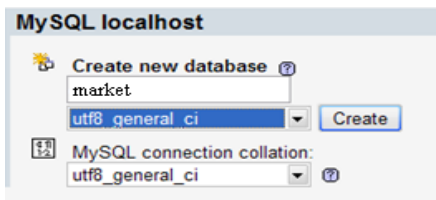
MySQL განაწილებული რელაციური ბაზა ერთ-ერთი აქტუალური და ფართოდ გამოყენებადი კლიენტ-სერვერული პაკეტია დღეისათვის, განსაკუთრებით php ვებ-ტექნოლოგიებში. ვგულისხმობთ, რომ MySQL სისტემა დაინსტალირებულია კომპიუტერზე (თუ არა, მაშინ ის ჩამოწერილ უნდა იქნეს <http://www.mysql.com> საიტიდან და დაინსტალირდეს).

როგორც წესი MySQL სისტემის გამომახება ხდება რომელიმე ბრაუზერში, მაგალითად Internet Explorer-ში url-მისამართში უნდა მიეთითოს: <http://localhost/phpmyadmin/> სტრიქონი. 7.1-ა ნახაზზე გამოტანილია შედეგი. აქ ავირჩევთ ჩვენთვის საჭირო ბაზას, მაგალითად, market, რომელსაც აქვს 2 ცხრილი. თუ საჭიროა ახალი ბაზის შექმნა, მაშინ გამოვიყენებთ ველს: create new database. აქ

ჩავწერთ ახალი ბაზის სახელს, Collation ველში ვირჩევთ სტრიქონს utf8_general_ci, რაც უნიკოდის გამოყენების საშუალებას იძლევა (ნახ 7.1-ბ).



ნახ.7.1-ა



ნახ.7.1-ბ

შემდეგ უნდა შევქმნათ ბაზის ცხრილები, რისთვისაც ველში “Create new table on database”: market - ჩავეწერთ ცხრილის სახელს, მაგალითად, product ან category და დავიწყოთ ცხრილის ველების (სვეტების) შექმნას (მაგალითად, ნახ.7.2-ა).

Field	Type	Length/Values ¹
pr_ID	INT	4
Name	VARCHAR	30
prize	DECIMAL	
cat_ID	INT	2

ნახ.7.2-ა

7.2-ბ ნახაზზე ნაჩვენებია ცხრილები: category და product.

The screenshot shows the phpMyAdmin interface for a database named 'market'. The left sidebar shows the database structure with 'category' and 'product' tables listed. The main panel displays a table view for '2 table(s)'. The table list includes:

Table	Action	Records ¹	Type	Colla
<input type="checkbox"/> category		8	InnoDB	latin1_sw
<input type="checkbox"/> product		23	InnoDB	latin1_sw
2 table(s) Sum		31	InnoDB	latin1_sw

At the bottom of the interface, there is a section titled "Create new table on database market" with input fields for "Name:" and "Number of fields:".

ნახ.7.2-ბ

ავირჩიოთ category ცხრილი და მენიუში Structure. 7.3-ა ნახაზზე ნაჩვენებია შედეგი.

	Field	Type	Collation	Attributes	Null	Default
<input type="checkbox"/>	cat_ID	int(2)			No	None
<input type="checkbox"/>	name	text	utf8_general_ci		No	None

ნახ.7.3-ა

cat_ID გასადებური ველი (პირველადი ინდექსი) აქ გახაზულია. ახლა უნდა შევავსოთ სტრიქონები კონკრეტული მნიშვნელობებით. ამისათვის ავამოქმედებთ Browse გადამრთველს და გამოჩნდება 7.3-ბ ნახაზზე მოცემული ფანჯარა.

Showing rows 0 - 7 (~8¹ total, Query took 0.0004 sec)

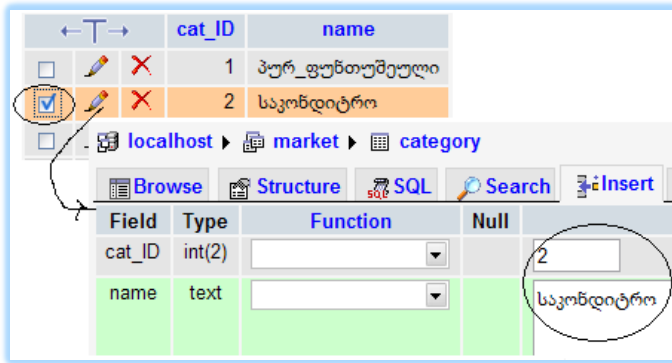
```
SELECT *
FROM `category`
LIMIT 0 , 30
```

+ Options

			cat_ID	name
<input type="checkbox"/>			1	პურ_ფუნთუშეული
<input type="checkbox"/>			2	საკონდიტრო
<input type="checkbox"/>			3	უაღკ_სასმელი
<input type="checkbox"/>			4	ალკ_სასმელი
<input type="checkbox"/>			5	ზორცეული
<input type="checkbox"/>			6	რძის_ნაწარმი
<input type="checkbox"/>			7	თევზეული
<input type="checkbox"/>			8	ხილი

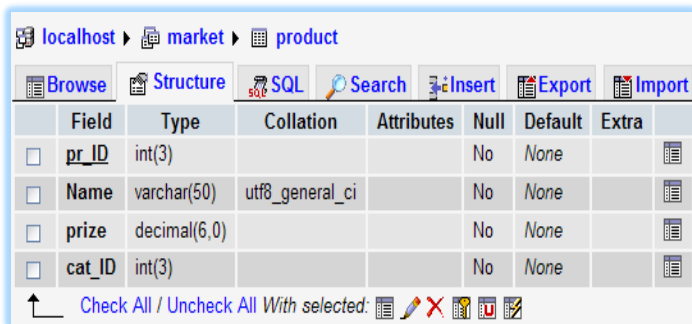
ნახ.7.3-ბ

”ჩეკბოქსის” ჩართვით და ფანქრის არჩევით შვეალთ სტრიქონის რედაქტირების რეჟიმში და შევასრულებთ ჩვენთვის საჭირო ფუნქციას (ნახ.7.4).

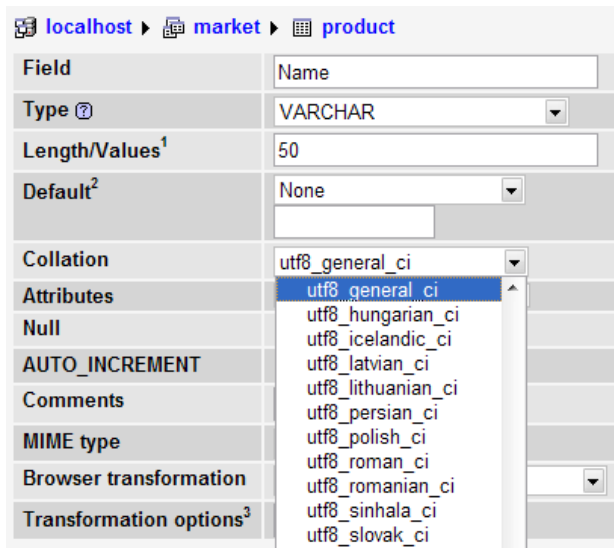


ნახ.7.4

ახლა გადავიდეთ product ცხრილზე. მისი ველების განსაზღვრით და სტრიქონების შეტანით შესაძლოა ქართული უნიკოდების მაგივრად '???? ????' სტრიქონის მიღება. ასეთი ველის სტრუქტურაში Collation სვეტში უნდა ჩავსვათ utf8_general_ci (ნახ.7.5-ა,ბ).

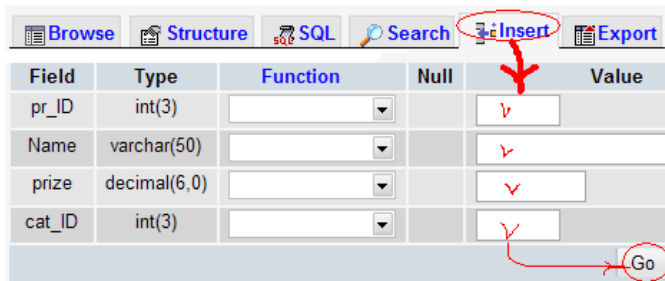


ნახ.9.45-ა



ნახ.7.5-ბ

შევავსოთ პროდუქციის ცხრილი მნიშვნელობებით Browse და Insert გადამრთველების გამოყენებით (ნახ.7.6-ა).



ნახ.7.6-ა

შევსებული ცხრილის ფრაგმენტი მოცემულია 7.6-ბ ნახაზზე.

← T →		pr_ID	Name	prize	cat_ID
<input type="checkbox"/>		1	არაქანი	4	6
<input type="checkbox"/>		2	პური	1	1
<input type="checkbox"/>		3	ბეხვი	11	5
<input type="checkbox"/>		4	ხაჭო	3	6
<input type="checkbox"/>		5	ფუნთუშა ქიმიშით	1	1
<input type="checkbox"/>		6	კოკა_კოლა	2	3
<input type="checkbox"/>		7	ფანტა	2	3
<input type="checkbox"/>		8	ღვინო ქინძმარაული	18	4
<input type="checkbox"/>		9	ღვინო ხვანჭყარა	23	4
<input type="checkbox"/>		10	ტირამუსი	7	2
<input type="checkbox"/>		11	საქონლის ხორცი	18	5
<input type="checkbox"/>		12	კონიაკი "ვარციხე"	20	4
<input type="checkbox"/>		13	არაყი "გომი"	13	4
<input type="checkbox"/>		14	ასატრინა	25	7
<input type="checkbox"/>		15	ვაშლი "გოლდენი"	3	8
<input type="checkbox"/>		16	კონსერვი "შპროტი"	5	7
<input type="checkbox"/>		17	ფორთოხალი	3	8
<input type="checkbox"/>		18	მინერალური "ბორჯომი"	2	3
<input type="checkbox"/>		19	მინერალური "ნაბელავი"	1	3
<input type="checkbox"/>		20	ორცხობილა "ნუმის"	4	2
<input type="checkbox"/>		21	ნაყინი "ვანილის"	6	2
<input type="checkbox"/>		22	მინერალური "ლიკანი"	2	3
<input type="checkbox"/>		23	ღორის სამწვადე	23	5

↑ Check All / Uncheck All With selected:

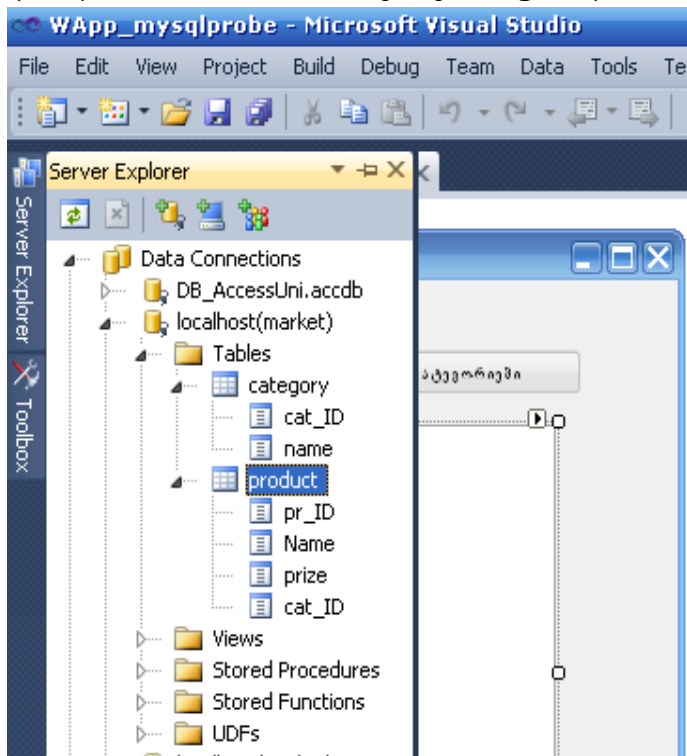
ნახ.7.6-ბ

ამგვარად, MySQL ბაზის ორი ცხრილი category და product მზადაა გამოსაყენებლად. დავხუროთ ეს ბაზა.

ახლა ავამუშავოთ Ms Visual Studio 2013 პაკეტი და C# ენის WindowsForm რეჟიმში Form1-ზე გამოვიტანოთ MySQL ბაზის მონაცემები, სხვადასხვა ჭრილში, მოთხოვნების შესაბამისად.

7.7 ნახაზზე ნაჩვენებია Server Explorer-ის ფანჯარა, სადაც localhost(market) მიერთებულ ბაზაში ჩანს Tables: category და product

თავიანთი ველებით (ატრიბუტებით). მთავარი მომენტია C# კოდიდან MySQL-ის market ბაზის მიერთება სამუშაოდ.



ნახ.7.7

პროგრამის კოდში უნდა მოთავსდეს MySQL ბაზის დასაკავშირებელი რამდენიმე სტრიქონი, რომელიც ქვემოთაა მოცემული.

```
MySql.Data.MySqlClient.MySqlConnection msqlConnection = null;  
msqlConnection = new MySql.Data.MySqlClient. MySqlConnection  
("server=localhost;"+  
"User Id=user@localhost; database=market;  
Persist Security Info=True");
```

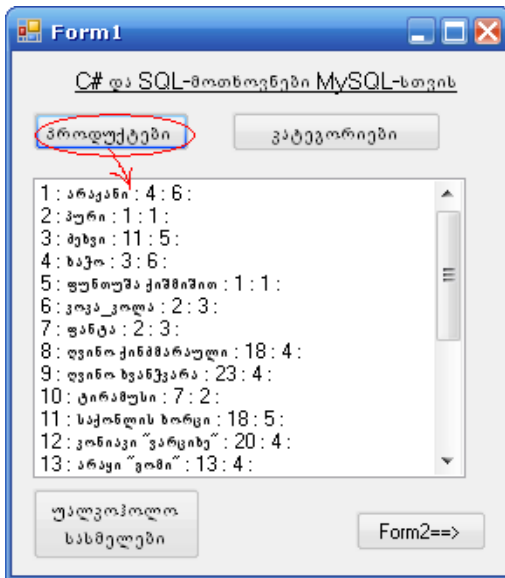
აქ განსაზღვრულია კლიენტის ბაზის მისაერთებლად აუცილებელი მონაცემები, როგორცაა localhost-სერვერი, მომხმარებლის User-იდენტიფიკატორი და ბაზის სახელი market.

პროგრამის მთლიან კოდს შემდეგ განვიხილავთ. ახლა Form1-ფორმაზე მოთავსებული რამდენიმე ღილაკი განვიხილოთ.

ღილაკით **”პროდუქტები”** ListBox1-ში გამოიტანება ყველა პროდუქტის სია, იდენტიფიკატორით, დასახელებით, ფასით და კატეგორიის ნომრით. მისი SQL-მოთხოვნის **”ამორჩევის”** კოდს ექნება ასეთი სახე:

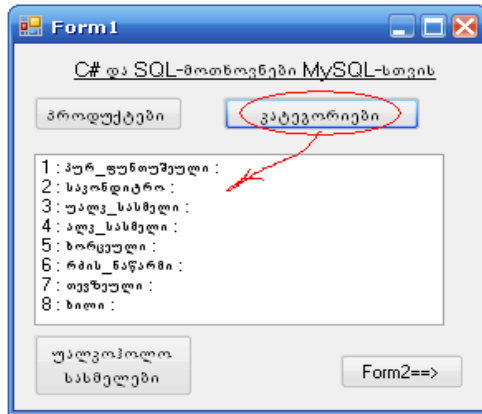
```
private void button1_Click(object sender, EventArgs e)
{
    Amorcheva("select * from product order by pr_ID, Name",
              "pr_ID", "Name", "prize", "cat_ID");
}
```

შედეგები ასახულია 7.8 ნახაზზე მოცემულ ფანჯარაში.



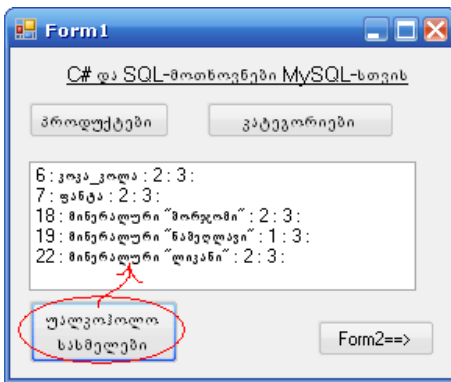
ნახ.7.8

ლილაკით ”კატეგორიები” ListBox1-ში გამოიტანება ბაზაში არსებული პროდუქციის ყველა კატეგორიის დასახელება. როგორც 7.9 ნახაზიდან ჩანს, სახეზეა 8 კატეგორია.



ნახ.7.9

ლილაკით ”უაღკოპოლო სასმელები” ListBox1-ში გამოიტანება იმ პროდუქტების სია, რომელთა ველში ”კატეგორიის იდენტიფიკატორი” შეესაბამება უაღკოფოლო სასმელებს, ანუ ჩვენ შემთხვევაში cat_ID=3. შედეგი ასახულია 7.10 ნახაზზე.



ნახ.7.10

პროგრამის კოდი მოცემულია 7.1 ლისტინგში.

```
// ლისტინგი_7.1-----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WApp_mysqlprobe
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }
        private void Amorcheva(string sqlbefehl,
                                params string[] velebi)
        {
            int i;
            string striqoni;
            MySql.Data.MySqlClient.MySqlConnection msqlConnection = null;
            msqlConnection = new
            MySql.Data.MySqlClient.MySqlConnection("server=localhost;" +
            "User Id=user@localhost;database=market;Persist
            Security Info=True");
            MySql.Data.MySqlClient.MySqlCommand msqlCommand = new
            MySql.Data.MySqlClient.MySqlCommand();
            msqlCommand.Connection = msqlConnection;
            msqlCommand.CommandText = sqlbefehl;
            try
            {
                msqlConnection.Open();
                MySql.Data.MySqlClient.MySqlDataReader msqlReader =
                msqlCommand.ExecuteReader();
                listBox1.Items.Clear();
                while (msqlReader.Read())
                {
                    striqoni = "";
                    for (i = 0; i < velebi.Length; i++)
                        striqoni += msqlReader[velebi[i]] + " : ";

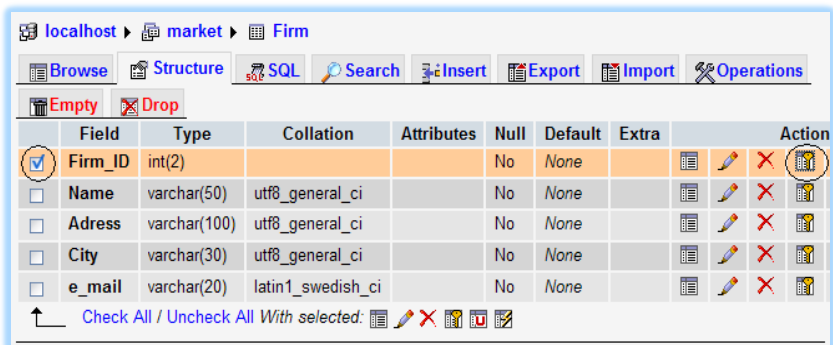
                    listBox1.Items.Add(striqoni);
                }
            }
        }
    }
}
```

```

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        msqlConnection.Close();
    }
}
private void button1_Click(object sender, EventArgs e)
{
    Amorceva("select * from product order by pr_ID, Name",
            "pr_ID", "Name", "prize", "cat_ID");
}
private void button2_Click(object sender, EventArgs e)
{
    Amorceva("select * from category order by cat_ID",
            "cat_ID", "Name");
}
private void button3_Click(object sender, EventArgs e)
{
    Amorceva("select * from product where cat_ID=3 order by
            pr_ID", "pr_ID", "Name", "prize", "cat_ID");
}
}
}

```

ამოცანა_7.7: განხილულ ბაზას დავამატოთ ახალი ცხრილი პროდუქციის მწარმოებელი ფირმებისათვის, სახელით Firm, რომელსაც ექნება ხუთი ველი: იდენტიფიკატორი, დასახელება, მისამართი, ქალაქი და ელ-ფოსტა (ნახ.7.11-ა).



ნახ.7.11-ა

Browse გადამრთველით გამოვიტანოთ ფორმების ცხრილი და შევავსოთ იგი (ნახ.7.11-ბ).














































+ Options					
← →	Firm_ID	Name	Address	City	e_mail
<input type="checkbox"/>	1	სანტე	წერეთლის 55	თბილისი	sante@gmail.ge
<input type="checkbox"/>	2	საკურთ	გორგასლის 111	თბილისი	kalanda@puri.ge
<input type="checkbox"/>	3	ნიკორა	ფუჩინაძის 200	თბილისი	nikora@gmail.ge
<input type="checkbox"/>	4	კოკა-კოლა	წერეთლის 30	თბილისი	cc@mail.ru
<input type="checkbox"/>	5	მიტანა	რიონის 177	ქუთაისი	mitana@mail.ru
<input type="checkbox"/>	6	ნატახტარი	რუსთაველის 3	მცხეთა	nataktari@gmail.com
<input type="checkbox"/>	7	თელიანი_ველი	ერეკლეს 55	თელავი	teliანი@org.com
<input type="checkbox"/>	8	ქუთათური	აღმაშენებლის 222	ქუთაისი	varcikhe@mail.ru
<input type="checkbox"/>	9	So&Co	კოსტავას 77	თბილისი	so&co@gmail.ge
<input type="checkbox"/>	10	გორიციხე	სტალინის 15	გორი	gorivashla@hotmail.c
<input type="checkbox"/>	11	ქობულეთა	რუსთაველის 555	ქობულეთი	Citron@achara.ge
<input type="checkbox"/>	12	ბორჯომულა	ლიკანის 20	ბორჯომი	borjomi@ckali.ge
<input type="checkbox"/>	13	ნახმარო	ნასაკირალის 1	ოზურგეთი	nabeglavi@hotmail.ge




↑ Check All / Uncheck All With selected:

ნახ.7.11-ბ

product ცხრილში საჭიროა დავამატოთ ველი Firm_ID, რომლითაც ის დაუკავშირდება ამ პროდუქციის მწარმოებელი ფორმის სტრიქონს. შევიტანოთ product ცხრილის Firm_ID ველის სვეტში შესაბამისი ფორმების იდენტიფიკატორები (ნახ. 7.12).

„პროგრამული სისტემების მენეჯმენტი: ლაბორატორიული პრაქტიკუმი“

← T →	pr_ID	Name	prize	cat_ID	Firm_ID
<input type="checkbox"/>  	1	არაჯანი	4	6	1
<input type="checkbox"/>  	2	პური	1	1	2
<input type="checkbox"/>  	3	ძეხვი	11	5	3
<input type="checkbox"/>  	4	ხაჭო	3	6	1
<input type="checkbox"/>  	5	ფუნთუშა ქიშიშით	1	1	3
<input type="checkbox"/>  	6	კოვასკოლა	2	3	4
<input type="checkbox"/>  	7	ფანტა	2	3	6
<input type="checkbox"/>  	8	ღვინო ქინძმარაული	18	4	7
<input type="checkbox"/>  	9	ღვინო ხვანჭყარა	23	4	7
<input type="checkbox"/>  	10	ტირამუსი	7	2	5
<input type="checkbox"/>  	11	საქონლის ხორცი	18	5	5
<input checked="" type="checkbox"/>  	12	კონიაკი "ვარციხე"	20	4	8
<input type="checkbox"/>  	13	არაყი "გომი"	13	4	8
<input type="checkbox"/>  	14	ასატრინა	25	7	9
<input type="checkbox"/>  	15	ვაშლი "გოლდენი"	3	8	10
<input type="checkbox"/>  	16	კონსერვი "შპროტი"	5	7	9
<input type="checkbox"/>  	17	ფორთოხალი	3	8	11
<input type="checkbox"/>  	18	მინერალური "ბორჯომი"	2	3	12
<input type="checkbox"/>  	19	მინერალური "ნაბეღლავი"	1	3	13
<input type="checkbox"/>  	20	ორცხოზილა "ნუშის"	4	2	5
<input type="checkbox"/>  	21	ნაყინი "ვანილის"	6	2	1
<input type="checkbox"/>  	22	მინერალური "ლიკანი"	2	3	12
<input type="checkbox"/>  	23	ღორის სამწვადე	23	5	5

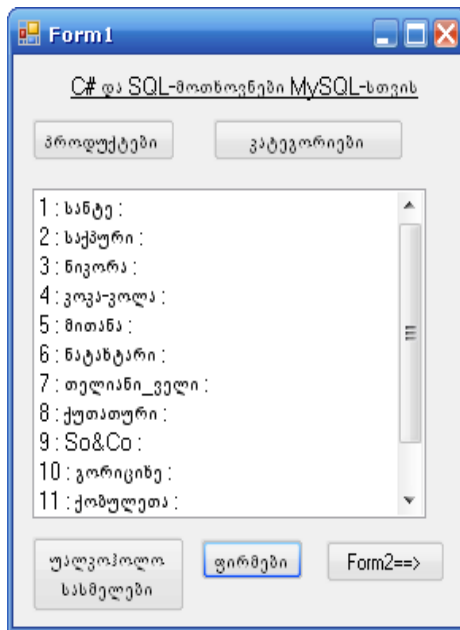
Check All / Uncheck All With selected:   

ნახ.7.12

დილაკით ”ფირმები” ListBox1-ში გამოვიტანოთ პროდუქციის მწარმოებელი ფირმების სია. C#კოდის ფრაგმენტს ექნება შემდეგი სახე:

```
private void button5_Click(object sender, EventArgs e)
{
    Amorceva("select * from Firm order by Firm_ID",
            "Firm_ID", "Name");
}
```

შედეგები მოცემულია 7.13 ნახაზზე.



ნახ.7.13

დავალება:

- ააგეთ MySQL საექსპერიმენტო მონაცემთა ბაზა და გამართეთ პროგრამული პაკეტი.

8. ლაბორატორიული სამუშაო N 8

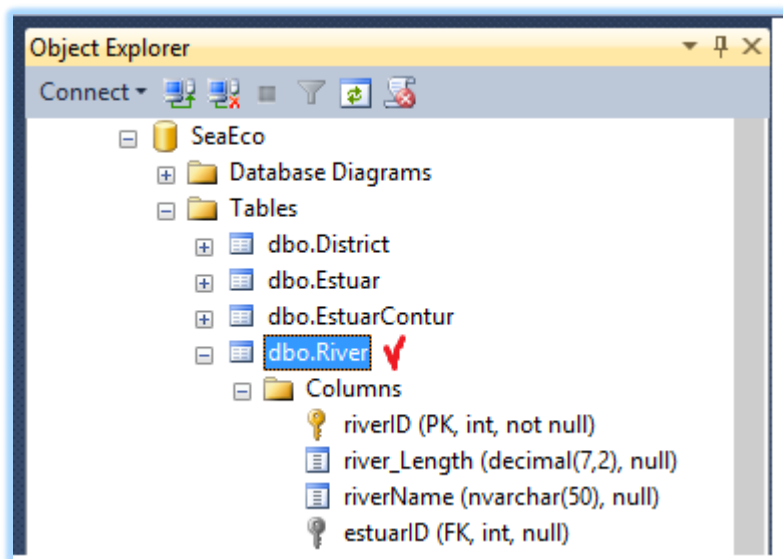
C# აპლიკაციის მუშაობა Ms SQL Server ბაზასთან ADO.NET დრაივერით და DataGridView კლასით

მიზანი: ვიზუალური დაპროგრამების C# ენის, Ms SQL Server-მონაცემთა ბაზების პაკეტის და ADO.NET-დრაივერის ერთობლივი გამოყენებით პროგრამული აპლიკაციების აგების შესწავლა.

განვიხილოთ პროგრამული აპლიკაციის აგება ADO.NET-ის ობიექტების გამოყენებით, რომელთა საშუალებითაც ხორციელდება კავშირი C# ენაზე დაწერილ ინტერფეისსა და მონაცემთა ბაზას შორის.

ამოცანა 8.1. მოცემულია Ms SQL Server მონაცემთა ბაზა (მაგალითად, შავი ზღვის ეკოლოგიური სისტემა), რომლის ერთ-ერთი ცხრილი (Table) არის River.dbo (მდინარეები). საჭიროა ავავოთ C# პროექტი (მომხმარებლის ინტერფეისი), რომელიც მონაცემთა ბაზიდან ამოიღებს მდინარეების მონაცემებს DataGridView ცხრილში, შეძლებს Insert, Update და Delete ოპერაციების განხორციელებას. გამოყენებულ უნდა იქნას ADO.NET დრაივერის საშუალებები.

8.1 ნახაზზე ნაჩვენებია საწყისი მონაცემთა ბაზა, რომელიც Ms SQL Server 2012 ვერსიაშია რეალიზებული. (ა) შეესაბამება ბაზის ცხრილების იერარქიას და აქვე ჩანს River ცხრილის სტრუქტურაც, მონაცემთა შესაბამისი ტიპებით. (ბ) -ზე მოცემულია ჩანაწერები, რომელთა შეტანა მოხდა წინასწარ (თუმცა ჩვენი პროგრამისთვის არაა აუცილებელი მისი არსებობა. შესაძლებელია იგი შეივსოს ინტერფეისიდან).

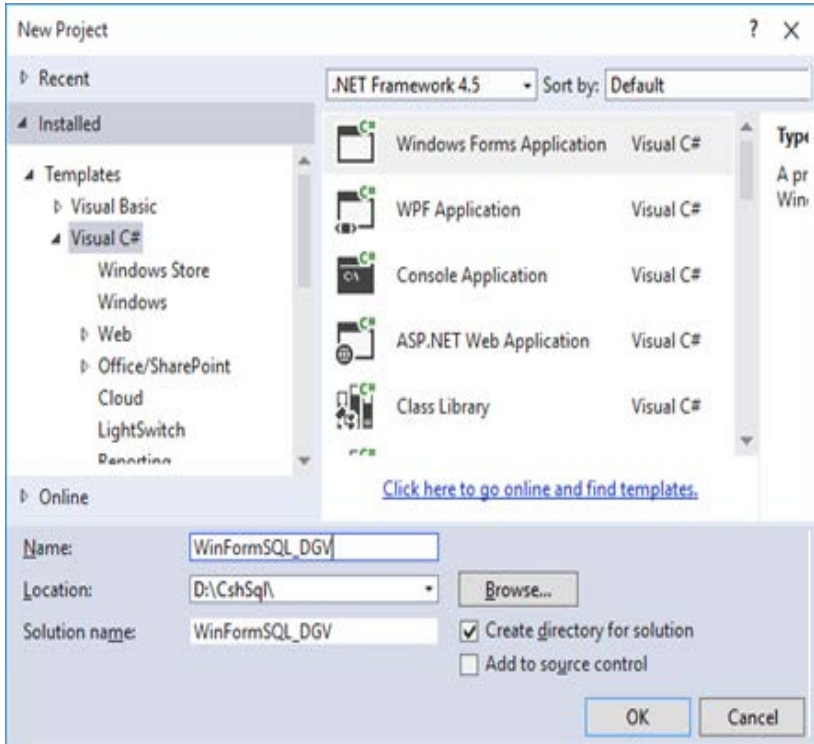


ნახ.8.1-ა. მონაცემთა ბაზა Ms SQL Server-ში

riverID	river_Length	riverName	estuarID
1	26,00	ჭოროხი (საქართველოს საზღვრებში)	1
2	25,20	კინტრიში	2
3	60,00	ნატანები	3
4	108,00	სუფსა	4
5	327,00	რიონი (სამხრეთ განშტოება)	5
6	327,00	რიონი (ჩრდილოეთ განშტოება)	6
7	150,00	ზობისწყალი	7
8	213,00	ენგური	8
*	NULL	NULL	NULL

ნახ.8.1-ბ. River (მდინარეების) საწყისი ცხრილი Ms SQL Server-ში

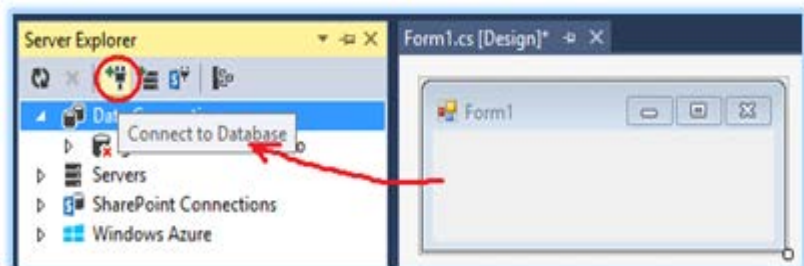
დავიწყით ახალი პროექტის აგება Ms Visual Studio.NET 2013 სამუშაო გარემოში. 8.2 ნახაზზე ნაჩვენებია პროექტის შექმნის პროცედურა.



ნახ.8.2. WinFormSQL_DGV პროექტის შექმნა

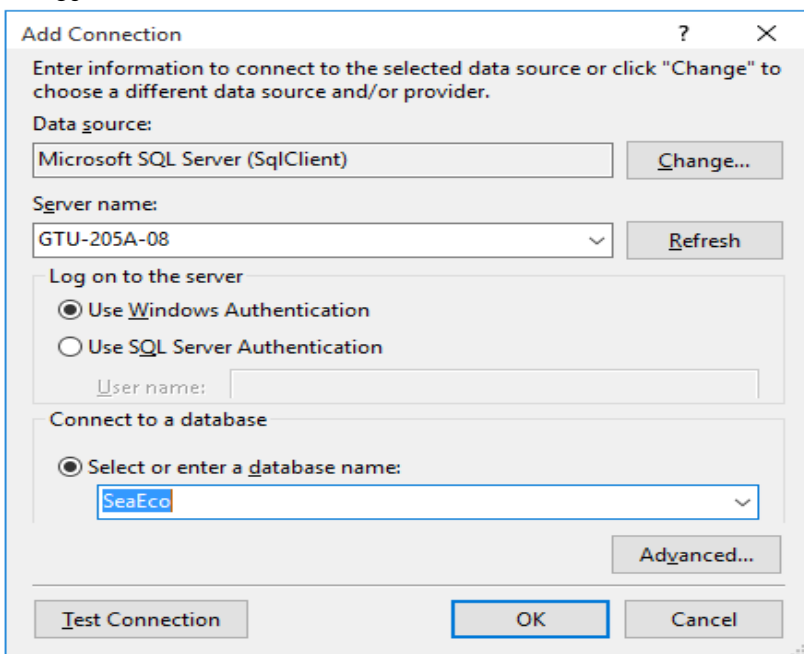
ვირჩევთ პროექტის სახელს (Name), მისი შენახვის ადგილს (Browse-ს დახმარებით) და Solutionname-ს. შემდეგ OK.

საჭიროა პროექტისთვის განვახორციელოთ მონაცემა ბაზის მიერთება (Connect to Database), რაც 8.3 ნახაზზეა ნაჩვენები.



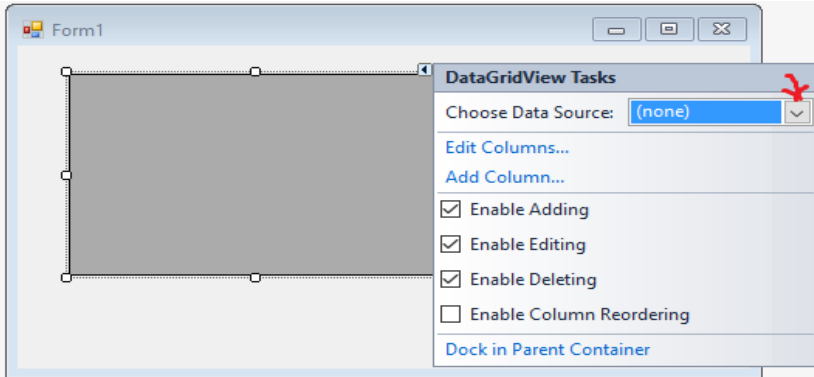
ნახ.8.3. Connect Database ამოქმედება

გამოიტანება ახალი ფანჯარა (ნახ.8.4), რომელშიც უნდა შეირჩეს შესაბამისი წყარო (Data Source), სერვერი (Server name) და მონაცემთა ბაზა (Database name).



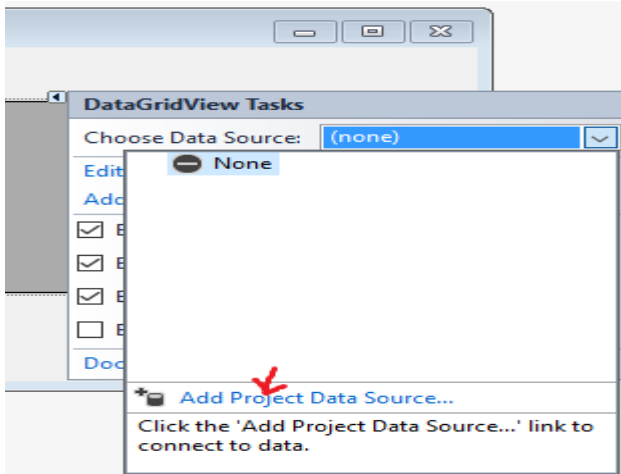
ნახ.8.4. მონაცემთა წყაროს, სერვერის და ბაზის არჩევა

ინსტრუმენტების პანელიდან ფორმაზე გადავიტანოთ DataGridView ელემენტი და ზედა მარჯვენა კუთხე პატარა ისრით ავამოქმედოთ. მივიღებთ 8.5 ნახაზს.



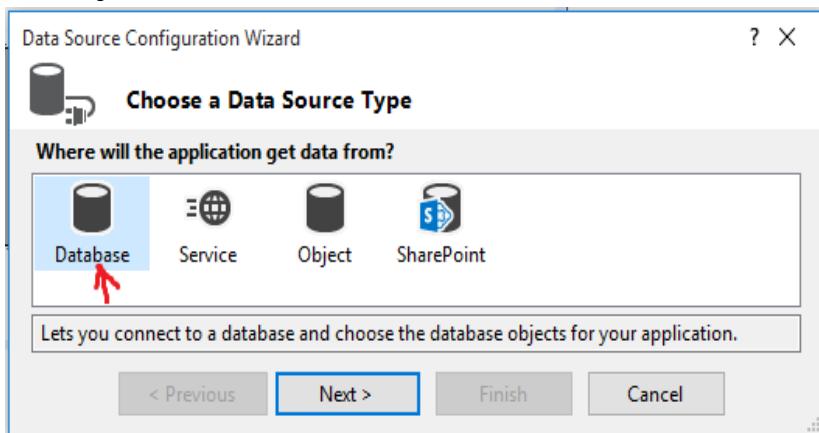
ნახ.8.5. პარამეტრების განსაზღვრა

ნახაზზე ჩანს, რომ ჩამატების, რედაქტირების და წაშლის ოპერაციები ნებადართულია (ჩეკბოქსები მონიშნულია). ავირჩიოთ Choose Data Source კომბობოქსის დილაკი, მივიღებთ 8.6 ნახაზს.



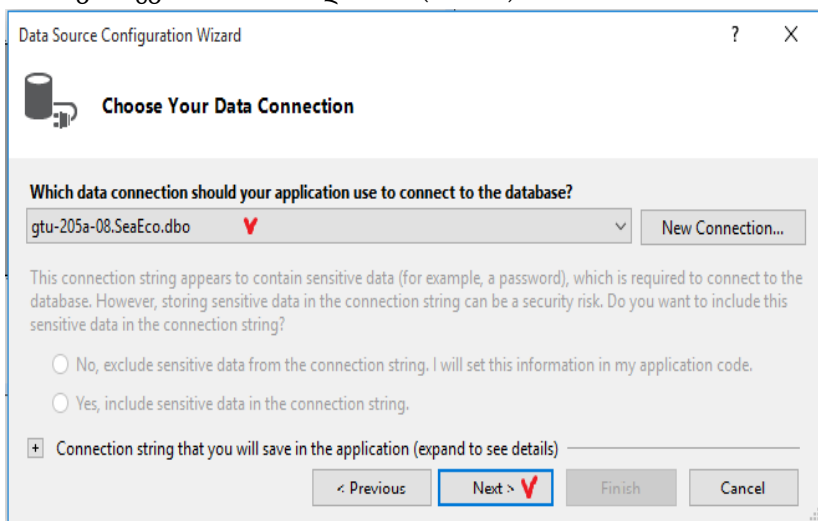
ნახ.8.6. მონაცემთა წყაროს პროექტის დამატება

ავამოქმედოთ Add Project Data Source და გადავიდეთ 8.7 ნახაზზე.



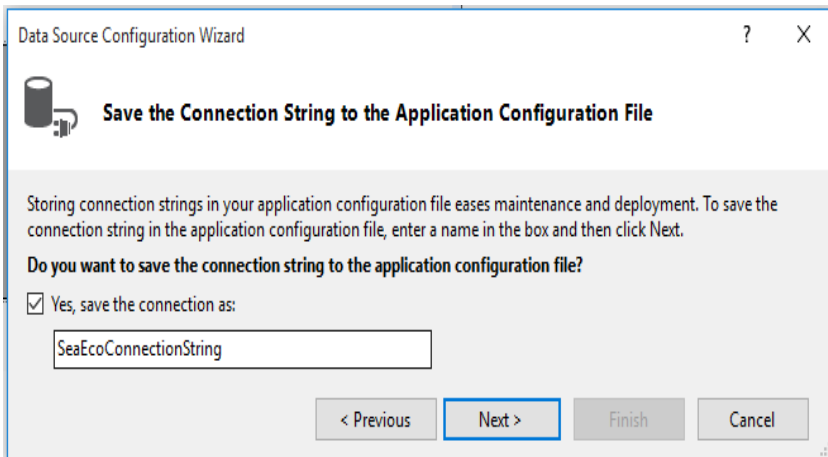
ნახ.8.7. მონაცემთა წყაროს ტიპის არჩევა

ვირჩევთ Database-ს და Next (ნახ.8.8).



ნახ.8.8. მონაცემთა Connection (მიერთების) შერჩევა

Connection პარამეტრი ყველა კომპიუტერს ექნება თავისი. მისი განსაზღვრა შესაძლებელია Server Explorer-იდან (ჩვენ შემთხვევაში იგი არის: GTU-205a-08.SeaEco.dbo). ზოლოს Next და გადავალთ 8.9 ნახაზზე.



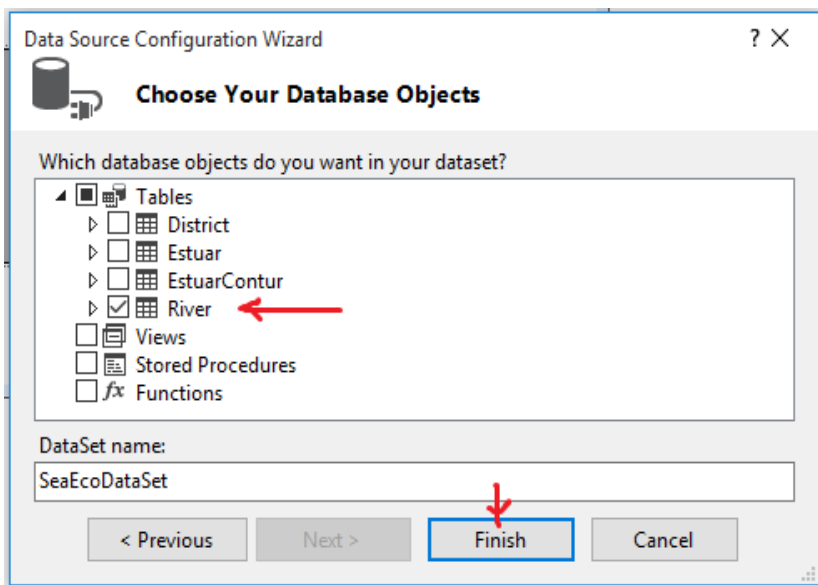
ნახ.8.9. Connection String-ის შენახვა აპლიკაციის კონფიგურაციის ფაილში

შემდეგ გამოიძახება მონაცემთა ბაზის ობიექტების არჩევის ფანჯარა (ნახ.8.10).

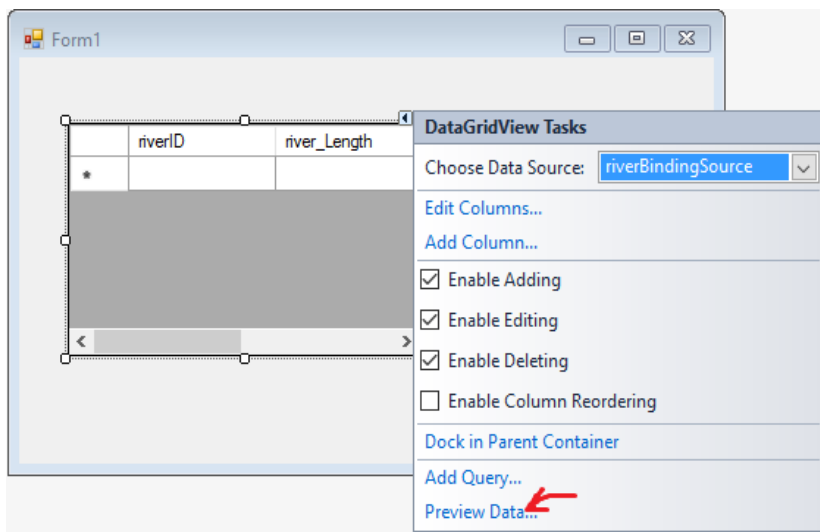
8.11 ნახაზზე ნაჩვენებია მონაცემთა წყაროს (Data Source) განსაზღვრის შედეგის მნიშვნელობა, ჩვენ შემთხვევაში იგი არის:

riverBindingSource.

აქვე შეიძლება გამოვიყენოთ Preview Data ლინკი და დავათვალიეროთ წინასწარ მიერთებული ბაზის ცხრილის ჩანაწერები (ნახ.8.12).



ნახ.8.8. ობიექტების მონიშვნა (მაგალითად, River)



ნახ.8.9. Data Source შედეგი: "riverBindingSource"

Preview Data

Select an object to preview:
 SeaEcoDataSet.River.Fill,GetData ()

Target DataSet:

Parameters:

Name	Type
No parameters are de	

Preview

Results:

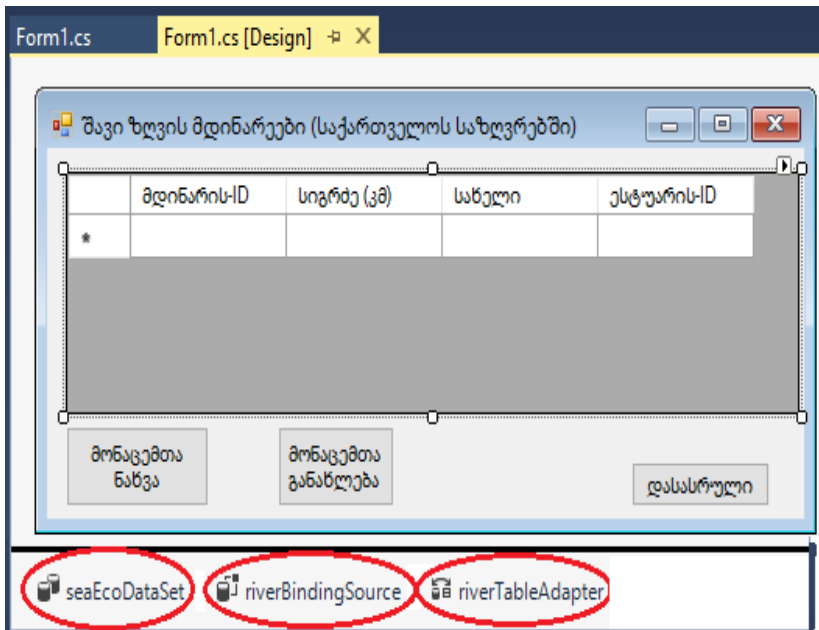
riverID	river_Length	riverName	estuarID
1	26,00	ჭოროხი (საქა...	1
2	25,20	კინტრიში	2
3	60,00	ნატანები	3
4	108,00	სუფსა	4
5	327,00	რიონი (სამხრ...	5
6	327,00	რიონი (ჩრდი...	6
7	150,00	ნობისწყალი	7
8	213,00	ენგური	8

Columns: 4 Rows: 8

Close

ნახ.8.10. Preview Data ცხრილი

ბოლოს, Form1-ს Properties-ში შევუცვალეთ სახელი „შავი ზღვის მდინარეები (საქართველოს საზღვრებში)“, ინსტრუმენტების პანელიდან გადმოვიტანეთ სამი ღილაკი (Button1,2,3), დავარქვათ სახელები. მიიღება 8.13 ნახაზი.



ნახ.8.11. პროექტის ძირითადი ინტერფეისი

ახლა გადავიდეთ ღილაკების ფუნქციების დაპროგრამებაზე, ანუ უნდა განხორციელდეს SQL Server -შესაბამისი ბაზის ცხრილის ჩანაწერების დავალიერება, ჩამატება, შეცვლა და წაშლა.

თავდაპირველად ჩავამატოთ სახელსივრცის სტრიქონი:
`using System.Data.SqlClient;`

შემდეგ გამოვაცხადოთ გლობალური ცვლადები:
`SqlDataAdapter sda;`
`SqlCommandBuilder scb;`
`DataTable dt;`

„მონაცემთა ნახვის“ ღილაკისთვის (button1) გვექნება შემდეგი კოდი:

```
private void button1_Click(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection("Data Source=
        GTU-205A-08;Initial Catalog=SeaEco;
        Integrated Security=True");
    sda = new SqlDataAdapter(@"SELECT riverID,
        river_Length, riverName,
        estuarID from River", con);
    dt = new DataTable();
    sda.Fill(dt);
    dataGridView1.DataSource = dt;
}
```

პროგრამის ამუშავებით, თუ მასში არაა შეცდომები, მიიღება 8.14 ნახაზი.

	მდინარის-ID	სიგრძე (კმ)	სახელი	ესტუარის-ID
▶	1	26,00	ჭოროში (საქარ...	1
	2	25,20	კინტრიში	2
	3	60,00	ნატანები	3
	4	108,00	სუფსა	4
	5	327,00	რიონი (სამხრე...	5
	6	327,00	რიონი (ჩრდი...	6
	7	150,00	ნობისწყალი	7
	8	213,00	ენგური	8
*				

ნახ.8.14. „მონაცემთა ნახვის“ (DataShow) დილაგის ამოქმედებით მიღებული შედეგი

„მონაცემთა განახლების“ ღილაკის კოდი (Insert, Update, Delete) ნაჩვენებია ქვემოთ:

```
private void button2_Click(object sender, EventArgs e)
{
    scb = new SqlCommandBuilder(sda);
    sda.Update(dt);
}
```

მთლიანი პროგრამის კოდი მოცემულია 8.1 ლისტინგში.

```
// --- ლისტინგი_8.1 --- Insert, Update, Delete for DataGridView_SQL----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WinFormSQL_DGV
{
    public partial class Form1 : Form
    {
        SqlDataAdapter sda;
        SqlCommandBuilder scb;
        DataTable dt;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // მონაცემთა ნახვა
            private void button1_Click(object sender, EventArgs e)
```

```
{
    SqlConnection con = new SqlConnection("Data
        Source=GTU-205A-08;Initial Catalog=SeaEco;
        Integrated Security=True");
    sda = new SqlDataAdapter(@"SELECT riverID,
        river_Length, riverName, estuarID
        from River", con);
    dt = new DataTable();
    sda.Fill(dt);
    dataGridView1.DataSource = dt;
}

// განახლება
private void button2_Click(object sender, EventArgs e)
{
    scb = new SqlCommandBuilder(sda);
    sda.Update(dt);
}

private void button3_Click(object sender, EventArgs e)
{
    Close();
}
}
```

8.15 ნახაზზე მოცემულია არსებულ ცხრილში ორი მნიშვნელობის (ესტუარისID) შეცვლა (ახალი რიცხვები ნაჩვენებია წრეში), შემდეგ 1-ელი და მე-2 ღილაკების ამოქმედებით ბაზაში ჩაიწერება ეს შეცვლილი მნიშვნელობები (შეიძლება დავხუროთ პროგრამა და თავიდან ავამუშავოთ).

შავი ზღვის მდინარეები (საქართველოს საზღვრებში)

	მდინარის-ID	სიგრძე (კმ)	სახელი	ესტუარის-ID
	1	26,00	ჭოროზი (საქარ...	5
	2	25,20	კინტრიში	2
	3	60,00	ნატანები	3
	4	108,00	სუფსა	4
▶	5	327,00	რიონი (სამრე...	1
	6	327,00	რიონი (ჩრდი...	6
	7	150,00	ნობისწყალი	7
	8	213,00	ენგური	8
*				

მონაცემთა ნახვა
მონაცემთა განაწლება
დასასრული

ნახ.8.13. Update ცვილების განხორციელება

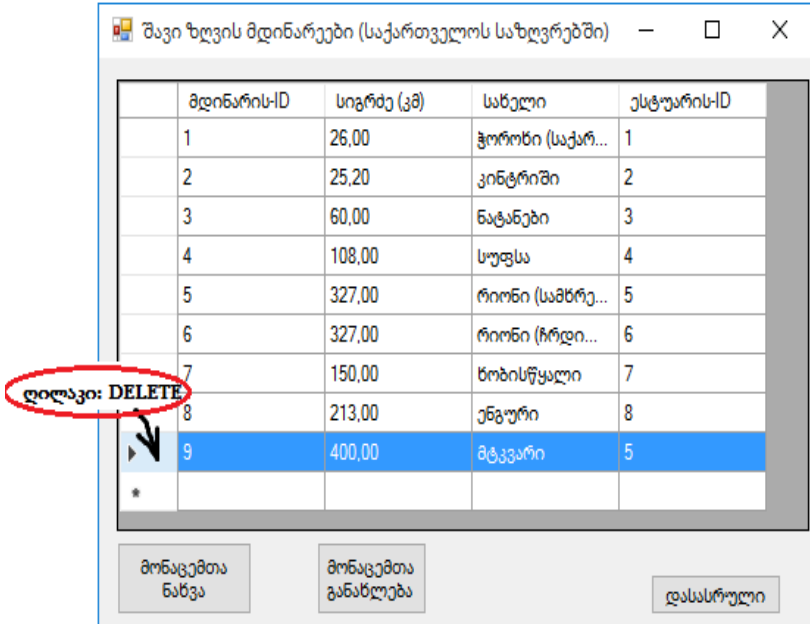
შავი ზღვის მდინარეები (საქართველოს საზღვრებში)

	მდინარის-ID	სიგრძე (კმ)	სახელი	ესტუარის-ID
▶	1	26,00	ჭოროზი (საქარ...	1
	2	25,20	კინტრიში	2
	3	60,00	ნატანები	3
	4	108,00	სუფსა	4
	5	327,00	რიონი (სამრე...	5
	6	327,00	რიონი (ჩრდი...	6
	7	150,00	ნობისწყალი	7
	8	213,00	ენგური	8
	9	400,00	მტკვარი	5
*				

მონაცემთა ნახვა
მონაცემთა განაწლება
დასასრული

ნახ.8.14. Insert ოპერაციის განხორციელება

ბოლოს ნაჩვენებია სტრიქონის წაშლის (Delete) ოპერაცია. იგი ხორციელდება მაგალითად, „მდინარის-ID“ შესაბამისი სტრიქონის მონიშვნით და შემდეგ კომპიუტერუს კლავიატურის Delete-ლილაკის ამოქმედებით (ნახ.8.17).



ნახ.8.15. Delete ოპერაციის განხორციელება

ამით დავასრულებთ ეს ლაბორატორიული სამუშაო.

დავალება: SQL Server მონაცემთა ბაზის ცხრილის ჩანაწერების დასამუშავებლად გამოიყენეთ BindingNavigator ინსტრუმენტი, გამოიკვლიეთ მისი ფუნქციები.

ააგეთ მომხმარებლის ინტერფეისი რომელიმე საპრობლემო სფეროსთვის (ამოცანა შეათანხმეთ მასწავლებელთან).

9. ლაბორატორიული სამუშაო N9

რეფაქტორინგი: კოდის დამუშავება და მისი რეორგანიზაცია

მიზანი: C# პროგრამული კოდის დამუშავების და მოდიფიკაციის პროცედურების შესწავლა, „რეფაქტორინგის“ არსის გაცნობა და მისი გამოყენება არსებული კოდის ვერსიის განახლების მიზნით.

პროგრამული კოდის დამუშავება ხდება არჩეული პროექტის ტიპის და კოდის შაბლონის მიხედვით, რომელსაც გვთავაზობს Visual Studio.NET 2013 სამუშაო გარემო. ამავე დროს ხორციელდება საწყისი კოდის სინტაქსის სისწორის შემოწმება, კოდის გენერირების ავტომატური დასრულება, აგრეთვე კოდში ნავიგაციის პროცესის გამოყენება (მაგალითად, კონტექსტური მენიუდან Go to definition - ით).

ყოველივე ეს მნიშვნელოვნად ზრდის პროგრამისტ-დეველოპერების მუშაობის მწარმოებლურობას !

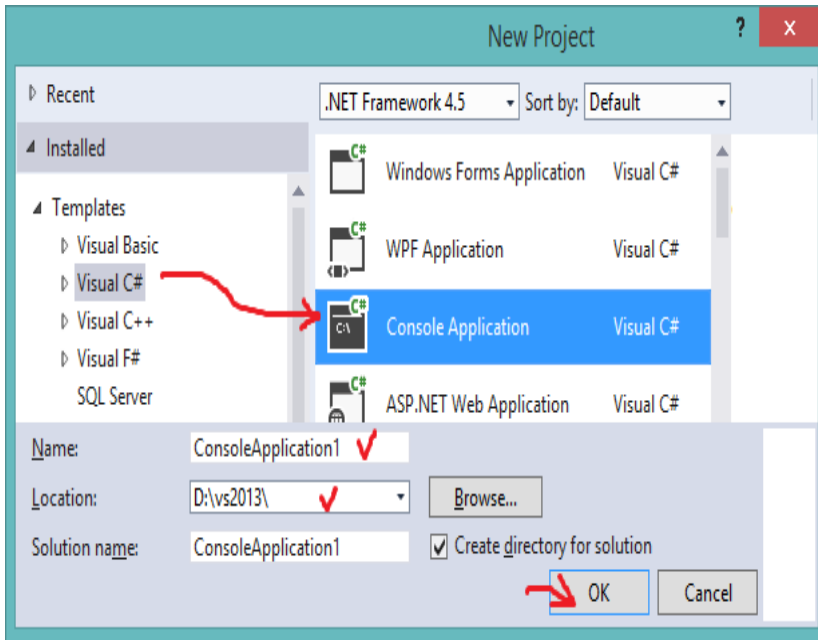
„რეფაქტორინგი“ არის არსებული პროგრამული კოდის სისტემატური მოდიფიკაცია და სრულყოფა, მისი ფუნქციონირების სემანტიკის ძირფესვიანი ცვლილების გარეშე. კოდში ცვლილებები ხორციელდება ავტომატიზებული გარდაქმნებით, რომლებსაც .NET სამუშაო გარემო გვთავაზობს.

ამის ტიპური მაგალითია - *მეთოდის სახელის შეცვლა*. ამოცანა მდგომარეობს რომელიმე კონკრეტული მეთოდისთვის *დასახელების* და მისი *განსაზღვრების* ცვლილების განხორციელებაში, ამასთანავე *მისი გამოყენების ყველა ადგილზე* !

თუ პროექტი საკმაოდ დიდია, მაშინ ხელით ასეთი ცვლილებების ჩატარების ამოცანის გადაწყვეტა საკმაოდ შრომატევადი და მოუხერხებელია. არაა გამორიცხული შემთხვევა, რომ რომელიმე მოდულში დაგვავიწყდეს ამ ცვლილების

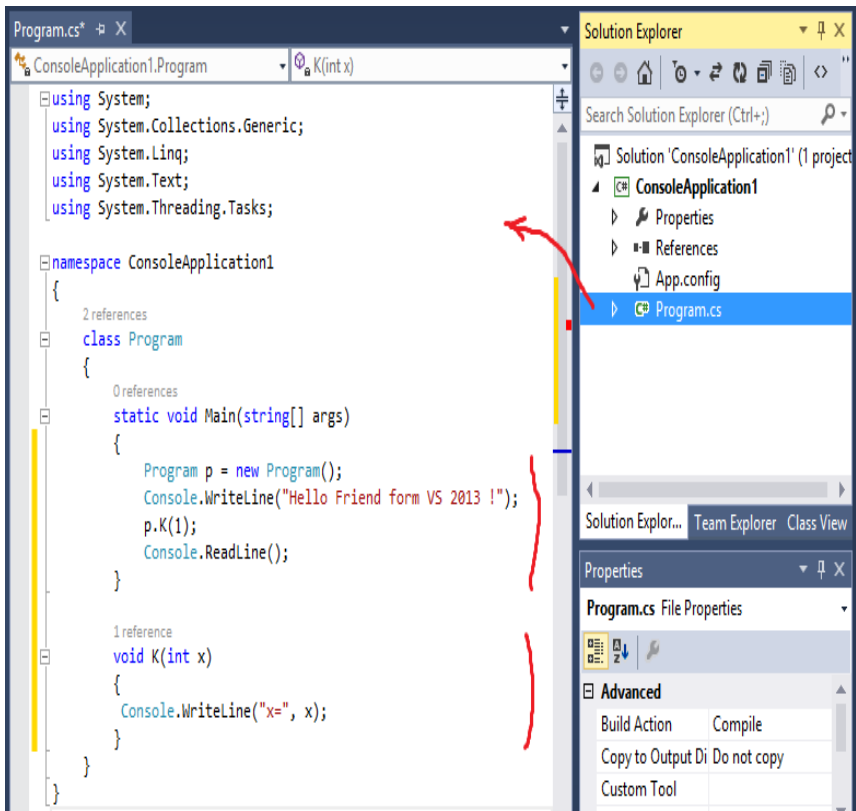
განხორციელება, რაც შემდგომში, პროგრამის ამუშავებისას, აუცილებლად გამოჩნდება შეცდომის სახით და მოგვიწევს პროექტის ხელახალი შესწორება.

განვიხილოთ რეფაქტორინგის ამოცანა მარტივი კონსოლური აპლიკაციისთვის (ნახ.9.1).



ნახ.9.1. კონსოლური აპლიკაციის შექმნა

მიღებული პროგრამის საწყის ტექსტს ჩავამატებთ რამდენიმე სტრიქონი, რომელსაც აქვს 9.2 ნახაზზე ნაჩვენები სახე.

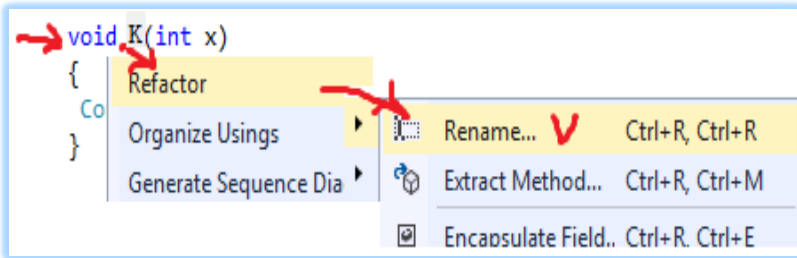


ნახ.9.2

პროგრამაში განსაზღვრულია Program კლასი, სტატიკური მეთოდი Main და K- ეგზემპლარის მეთოდი x არგუმენტით.

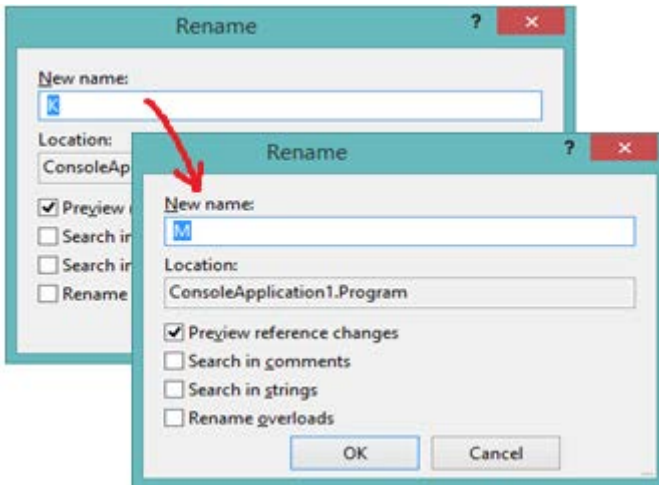
დავუშვათ, რომ საჭიროა შეიცვალას K მეთოდის სახელი M-ით. ცვლილება უნდა განხორციელდეს არა მხოლოდ ამ მეთოდის განსაზღვრებაში, არამედ მისი გამოყენების ადგილებზეც !

კოდის რედაქტირების გარემოში მაუსის კურსორს ვაყენებთ K მეთოდის პირველ სტრიქონზე, მოვნიშნავთ შესაცვლელ სიტყვას და კონტექსტურ მენიუდან ვირჩევთ პუნქტს Refactor / Rename (ნახ.9.3).

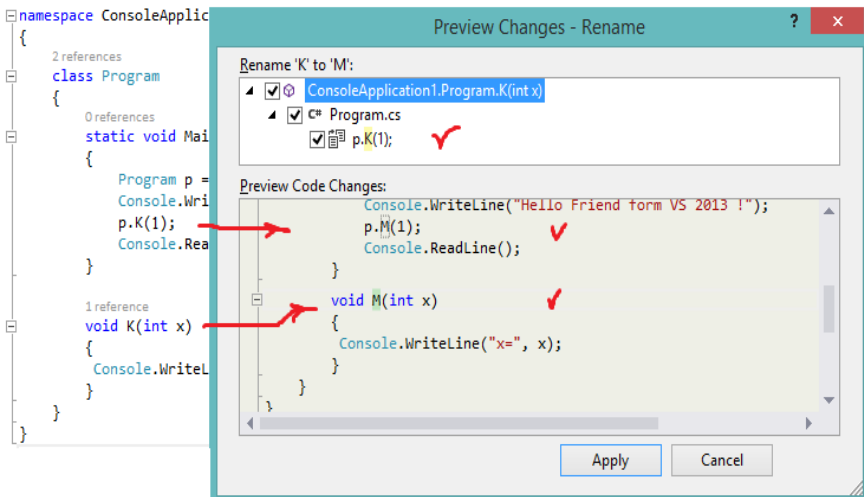


ნახ.9.3. K მეთოდისთვის რეფაქტორინგის ქმედების არჩევა

ნახ.9.4. K-ივლება M-ით

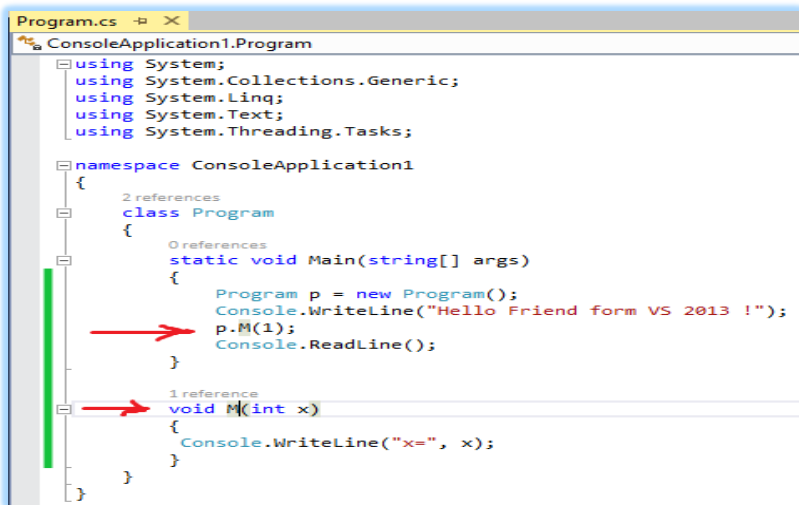


მიიღება 9.5 სურათი.



ნახ.9.5. დაგეგმილი ცვლილებების წინასწარ ნახვა (Preview Changes)

თუ ყველაფერი წესრიგშია, მაშინ ვირჩევთ Apply-ს და ვღებულობთ რეფაქტორინგის შედეგს (ნახ.9.6).



ნახ.9.6. რეფაქტორინგის შედეგი

VS 2013 სამუშაო გარემოს აქვს რეფაქტორინგის შემდეგი შესაძლებლობები:

- **Rename** - სახელის შეცვლა;
- **Extract Method** - მეთოდის ამოღება: კოდის მონიშნული ფრაგმენტის მოდიფიკაცია მეთოდში მითითებული სახელით;
- **Encapsulate Field** - ველის ინკაფსულირება, მისი პრივატად გაკეთებით, ოღონდ პაბლიკ-თვისების დამატებით მისი წვდომის მიზნით;
- **Extract Interface** - ინტერფეისის ამღება: კლასის ტექსტის მონიშვნა მისთვის ავტომატურად შესაბამისი ინტერფეისის ფორმირება (თუ ეს შესაძლებელია);
- **Remove parameters** - მეთოდის პარამეტრების ნაწილის წაშლა;
- **Reorder parameters** - მეთოდის პარამეტრების რიგითობის შეცვლა.

დავალება:

- ააგეთ მარტივი კონსოლის აპლიკაციის პროექტი Visual Studio.NET 2013 სამუშაო გარემოში (მაგალითად: უნივერსიტეტი, სუპერმარკეტი, აფთიაქი, პროდუქციის_წარმოება_მიწოდება ან სხვა).

- ჩაატარეთ რეფაქტორინგის პროცედურები და გამოიკვლიეთ მისი ზემოქანმოთვლილი სახეების შესაძლებლობები.

10. ლაბორატორიული სამუშაო N10

Visual Studio.NET -ის რევერსიული ინჟინერის ინსტრუმენტული საშუალებები: „Model-Code-Model...“

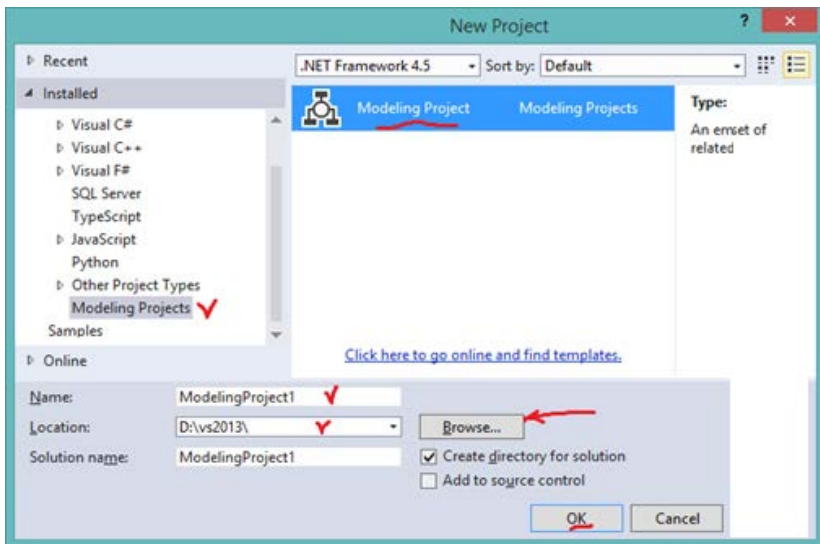
მიზანი: C# პროგრამული კოდის ავტომატური გენერაციის პროცედურის შესწავლა UML-ის კლასთა დიაგრამის საფუძველზე.

განვიხილოთ მარტივი UML-მოდელის გენერაცია ორი ურთიერთდაკავშირებული კლასის საფუძველზე, შემდეგ ამ მოდელის ვიზუალიზაცია და კოდის გენერაცია C# ენაზე.

UML-მოდელის ასაგებად გამოიყენება სპეციალური სახის პროექტი **Modeling Project**.

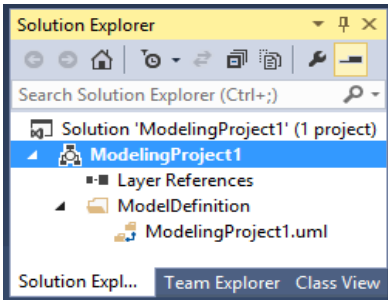
შევქმნათ პროექტი მოდელირების მიზნით. VS 2013 გარემოს მთავარ მენიუში ავირჩიოთ (ნახ.10.1):

File / NewProject და პროექტის ტიპი: Modeling Project

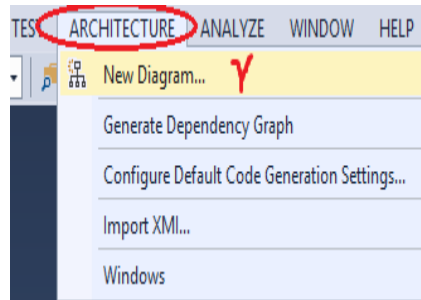


ნახ.10.1. პროექტის შექმნა UML-მოდელის ასაგებად

მიიღება 10.2 ნახაზზე ნაჩვენები Solution Explorer.

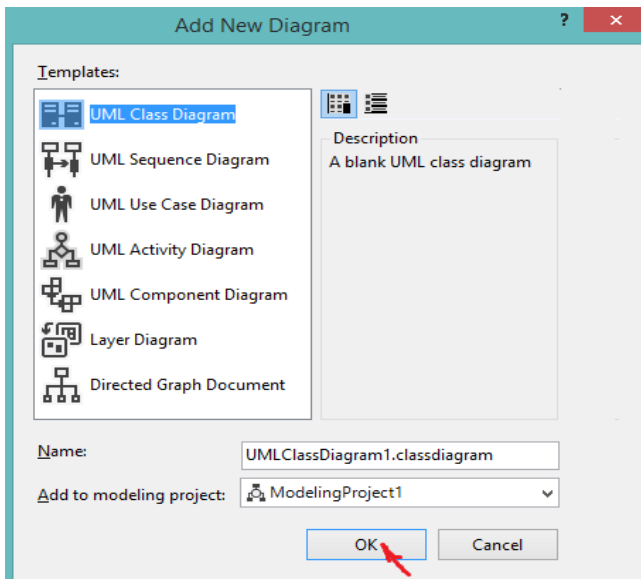


ნახ.10.2



ნახ.10.3

პროექტში ახალი UML-დიაგრამის დასამატებლად ავირჩიოთ მენიუს პუნქტი Architecture, რომელიც სპეციალურად გამოიყენება პროგრამის არქიტექტურის წარმოსადგენად UML-მოდელის სახით და მის შიგნით პუნქტი Add new Diagram (ნახ.10.3). მიიღება ფანჯარა (ნახ.10.4).

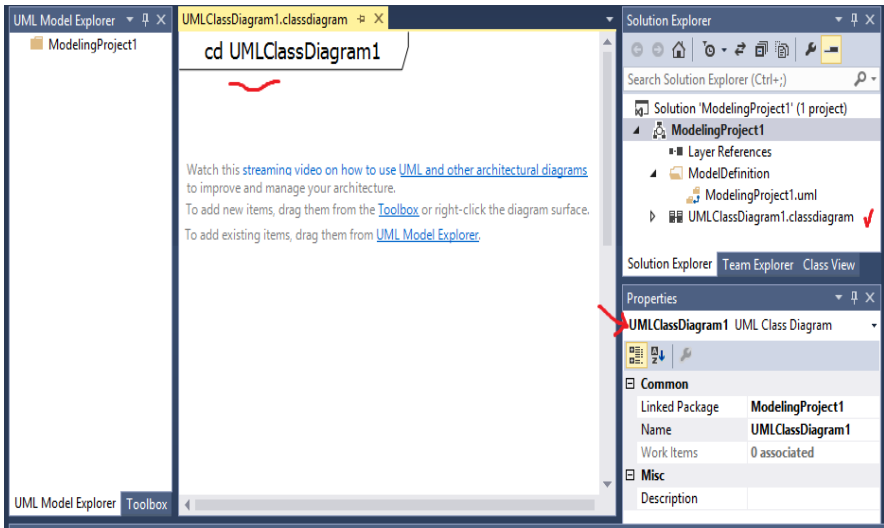


ნახ.10.4. ახალი UML-დიაგრამის არჩევა

ავტომატურად აირჩევა კლასების დიაგრამა (**UML Class Diagram**), როგორც ხშირად გამოყენებადი. მაგრამ UML-ენის ახალი ვერსიის შესაბამისად, აქ შესაძლებელია სხვა დიაგრამების შექმნაც, როგორებიცაა:

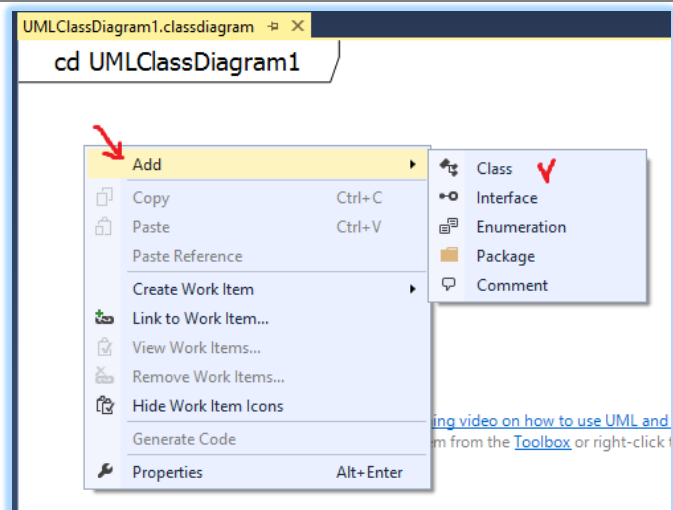
- UML Sequence Diagram - მიმდევრობითობის დიაგრამა;
- UML Use Case Diagram - გამოყენებით შემთხვევათა დიაგრამა;
- UML Activity Diagram - აქტიურობის დიაგრამა;
- UML Component Diagram - კომპონენტების დიაგრამა;
- Layer Diagram - დონეების დიაგრამა;
- Directed Graph Document - დიაგრამა, რომელიც ასახავს დოკუმენტს ორიენტირებული გრაფის სახით.

შევქმნათ ახალი კლასთა დიაგრამა, თავიდან ცარიელი (ნახ.10.5).



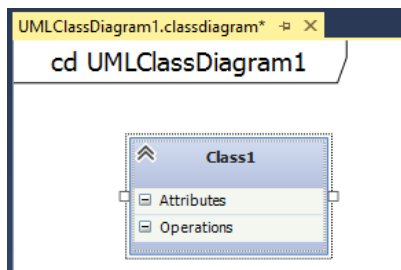
ნახ.10.5. კლასების (ცარიელი) დიაგრამის შექმნა

კლასის დასამატებლად დიაგრამაზე ვირჩევთ მენიუს პუნქტს Add / class (ნახ.10.6).



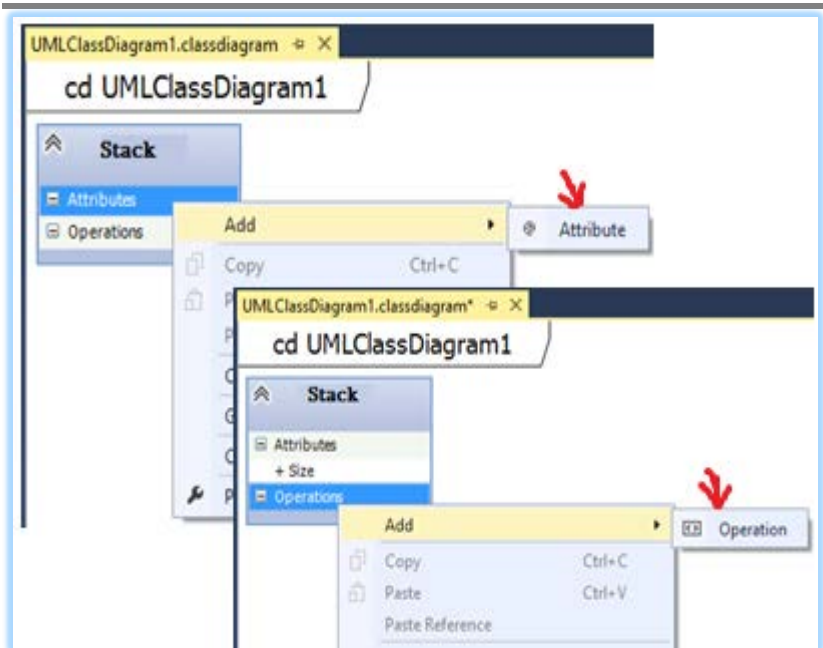
ნახ.10.6. დიაგრამაზე ახალი კლასის დამატება

ახალი კლასი ემატება ავტომატურად სახელით Class1. იგი შედგება ატრიბუტებისა და ოპერაციებისგან (ნახ.10.7).

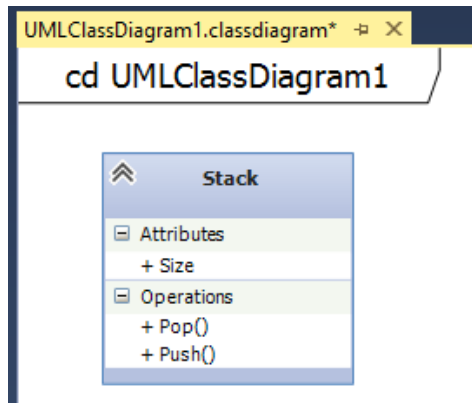


ნახ.10.7. კლასი ატრიბუტებით და ოპერაციებით

შეცვალეთ მოდელში კლასის სახელი **Stack**-ით, ატრიბუტის სახით დავამატოთ **Size**, ხოლო ოპერაციის სახით **Push** და **Pop** (ნახ.10.8). დასამატებლად გამოიყენება კონტექსტური მენიუ (მაუსის მარჯვენა ღილაკით) და პუნქტი Add. შედეგი მოცემულია 10.9 ნახაზზე.

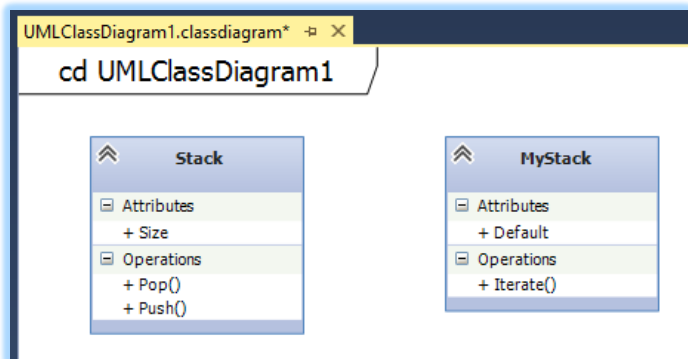


ნახ.10.8. ატრიბუტის და ოპერაციის დამატება კლასში



ნახ.10.9. Stack კლასის შედეგი

ახლა დავამატოთ მეორე კლასი - **MyStack**, ატრიბუტით **Default** და ოპერაციით **Iterate** (ნახ.10.10).



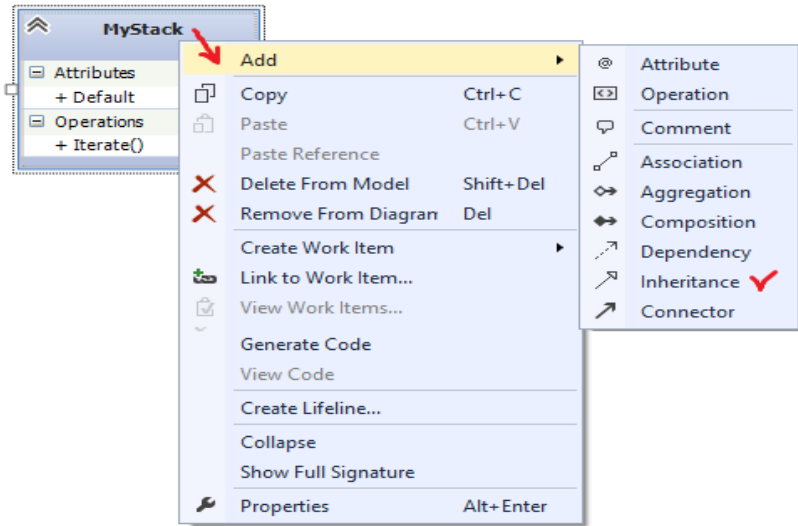
ნახ.10.8. **MyStack** კლასის დამატება **Default** ატრიბუტით და **Iterate** ოპერაციით

ახლა დიაგრამაზე უნდა ავსახოთ ინფორმაცია იმის შესახებ, რომ **MyStack** კლასი არის **Stack** კლასის მემკვიდრე. ვდგებით **MyStack** კლასზე და კონტექსტურ მენიუში ვირჩევთ **Add**-ს.

აქ უნდა ამოვირჩიოთ კავშირის ელემენტი და დავამატოთ დიაგრამაზე [9]:

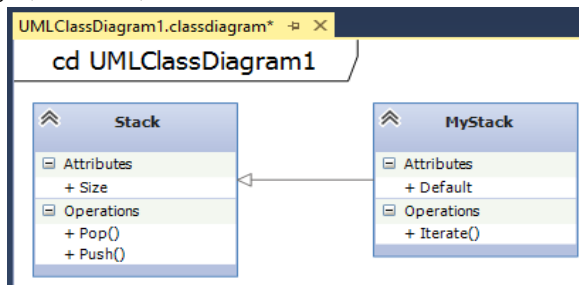
- Attribute** - ახალი ატრიბუტი,
- Operation** - ახალი ოპერაცია,
- Comment** - კომენტარი;
- Association** - ასოციაცია,
- Aggregation** - აგრეგაცია,
- Composition** - კომპოზიცია,
- Dependency** - დამოკიდებულება,
- Inheritance** - მემკვიდრეობითობა,
- Connector**- კონექტორი.

ჩვენ გვინტერესებს **Inheritance** (ნახ.10.11).



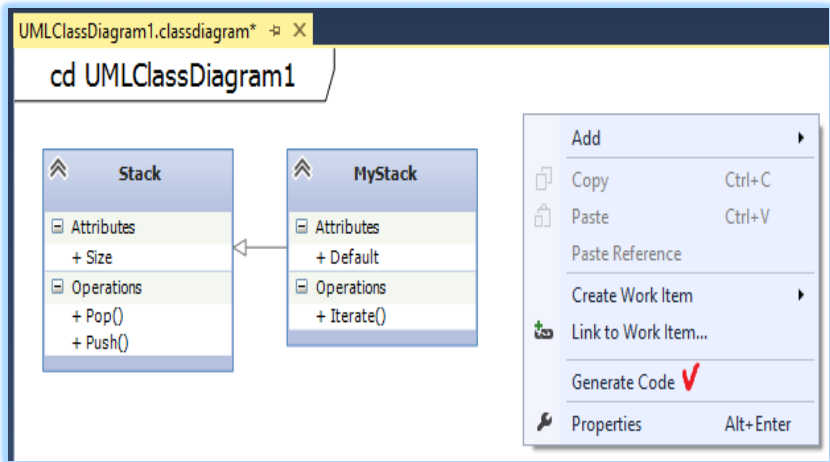
ნახ.10.9. ორ კლასს შორის Inheritance-კავშირის არჩევა

Add / Inheritance - არჩევით დიაგრამაზე დაემატა მემკვიდრეობითობის კავშირი ისრით მიმართული **MyStack**-იდან **Stack**-ისკენ (ნახ.10.12).



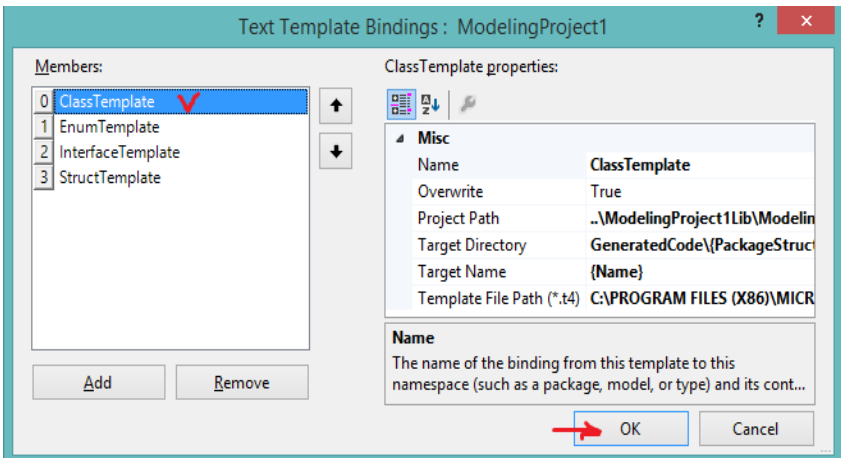
ნახ.10.10. კლასთა კავშირი Inheritance

ახლა შეიძლება კოდის გენერირება, რომელსაც შემდომში გამოვიყენებთ. კონტექსტურ მენიუში ვირჩევთ პუნქტს - Generate code (ნახ.10.13).



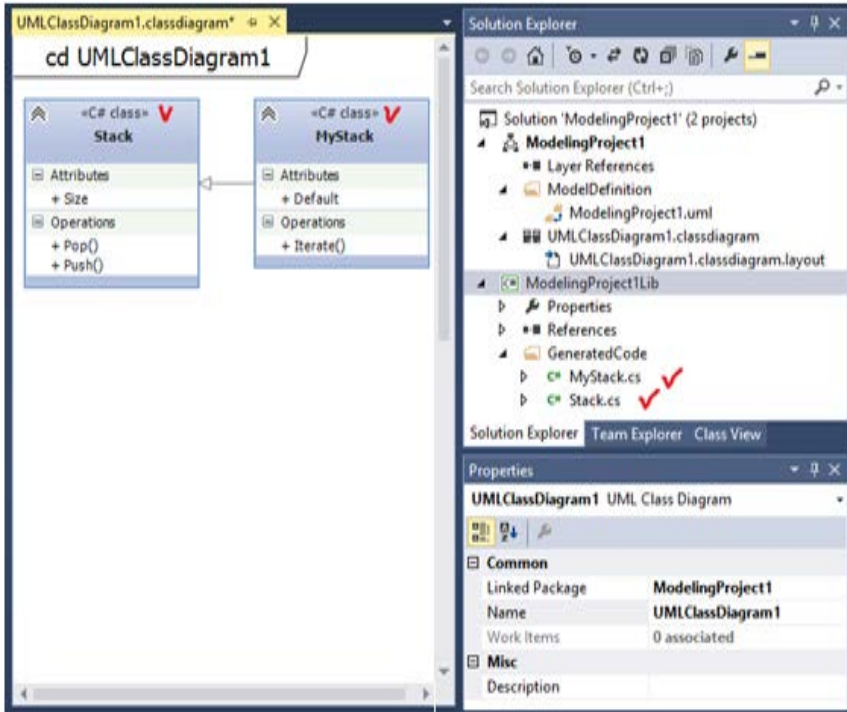
ნახ.10.11. კოდის გენერაცია UML-დიაგრამაზე

კოდის გენერატორი გვთხოვს დავაზუსტოთ თუ რომელი შაბლონით (Template) მოხდება გენერაცია. ვირჩევთ შაბლონს კლასისთვის (ნახ.10.14).



ნახ.10.14. ClassTemplate - შაბლონის არჩევა კოდის გენერაციისთვის

კოდის გენერაციის შემდეგ Solution Explorer-ში გამოჩნდება ორი ახალი სტრიქონი, C#-ის ფაილებისთვის: Stack.cs და MyStack.cs (ნახ.10.15).



ნახ.10.13. Stack.cs და MyStack.cs კოდების გენერაცია UML-დიაგრამიდან

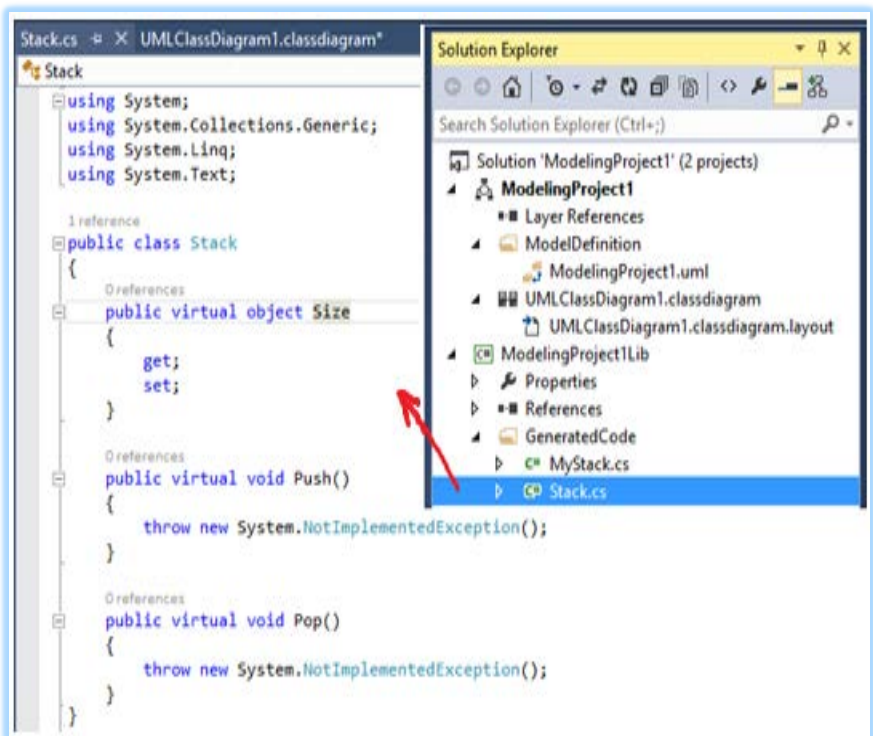
გავხსნათ ახალი Stack.cs და MyStack.cs ფაილები. ტესტები მოცემულია 10.16 და 10.17 ნახაზებზე.

ატრიბუტი **Size** რეალიზებულია კლასისი თვისების (property) სახით **get** და **set** მეთოდებით. ხოლო მეთოდების სახშობები წარმოდგენილია ვირტუალური მეთოდების სახით და რეალიზებულია როგორც გამოწვევის გენერაცია. მაგალითად,

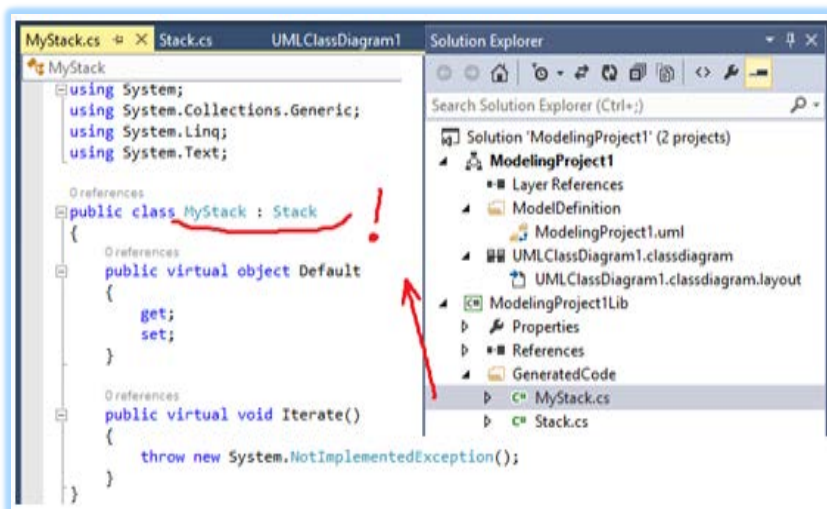
```
public virtual void Push()  
{  
    throw new NotImplementedException();  
}
```

10.17 ნახაზზე წარმოდგენილი კოდიდან ჩანს კლასების მემკვიდრეობითობა:

Public class MyStack : Stack



ნახ.10.14. გენერირებული Stack.cs ფაილის კოდი



ნახ.10.15. გენერირებული MyStack.cs ფაილის კოდი

ამის შემდეგ გენერირებული ფაილები შეიძლება გამოყენებულ იქნას დამუშავების მომდევნო ეტაპებზე. აგებული მოდელი თამაშობს მნიშვნელოვან როლს: ესაა მოდელირების და პროექტირების წინასწარი ეტაპების შედეგების ასახვა პროექტში. პროექტის შეცვლის აუცილებლობის შემთხვევაში, შეიძლება ცვლილებები ჩატარდეს UML-დიაგრამებში, საიდანაც კოდები ავტომატურად იქნება გენერირებული. შესაძლებელია პირიქითაც (**reverse engineering**-ის გამოყენება), რეალიზებული კლასებიდან მოხდეს UML-დიაგრამების გენერირება.

დავალება:

- ააგეთ ახალი პროექტი Modeling Project-ით. დააპროექტეთ დიაგრამა სამი კლასით: „მშობელი“-Person, „შვილები“ Student და Lector. შეიტანეთ ატრიბუტები და ოპერაციები, გამოიყენეთ მემკვიდრეობითობის კავშირი „მშობელ-შვილების“ კლასებს შორის.

- განახორციელეთ კლასების კოდის ავტომატური გენერირება C# ენაზე.

11. ლაბორატორიული სამუშაო N11

პროგრამული აპლიკაციების ტესტირება

მიზანი: პროგრამული კოდების ტესტირების პროცესის შესწავლა Visual Studio.NET ინტეგრირებულ გარემოში.

1. თეორიული ნაწილი

Unit testing და Coded UI არის Microsoft-ის ტესტირების ინსტრუმენტები, რომლებიც სრულდება Visual Studio.NET-გარემოში.

Unit testing - ანუ მოდულური ტესტირება დაპროგრამების პროცესია, რომლის საშუალებითაც მოწმდება საწყისი კოდის ცალკეული მოდულების კორექტულობა. ასეთი ტესტირების იდეა მდგომარეობს იმაში, რომ ყოველი არატრივიალური ფუნქციის ან მეთოდისთვის დაიწეროს ტესტი. ეს უზრუნველყოფს კოდის სწრაფად შემოწმებას, ხომ არ მიიყვანა კოდის ბოლო ცვლილებამ პროგრამა რეგრესიამდე, ანუ შეცდომების გაჩენამდე პროგრამის უკვე ტესტირებულ ნაწილებში [10].

Coded UI ტესტი კი ავტომატურად იწერს, შესრულებაზე უშვებს და ამოწმებს ტესტ-ქეისებს.

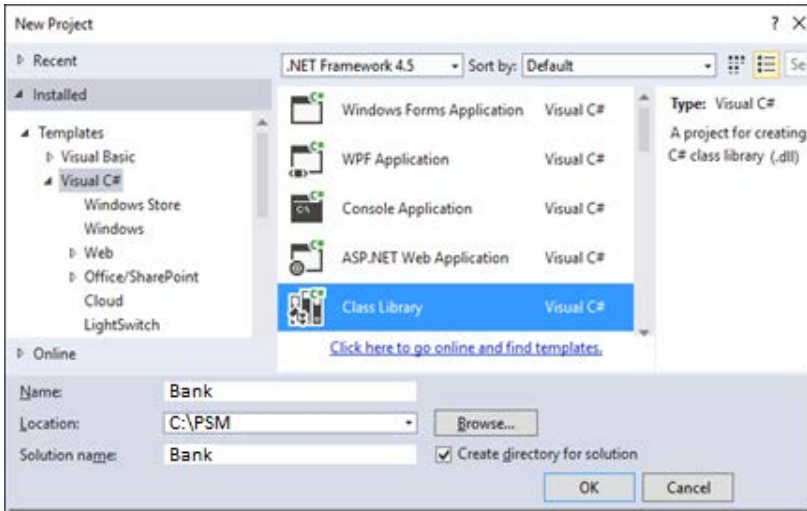
ასეთი ტესტების წერა შესაძლებელია C# ან Visual Basic-ზე Visual Studio გარემოში. Unit testing ტესტირების ტექნოლოგია განვიხილოთ ვირტუალური ობიექტის, მაგალითად, ფინანსური ობიექტის, ბანკის მაგალითზე.

- დასატესტი პროგრამის პროექტის შექმნა: Visual Studio-ში საჭიროა ავირჩიოთ: **File => New => Project**.

შედეგად გამოჩნდება დიალოგური ფანჯარა (ნახ.11.1). სადაც ავირჩევთ:

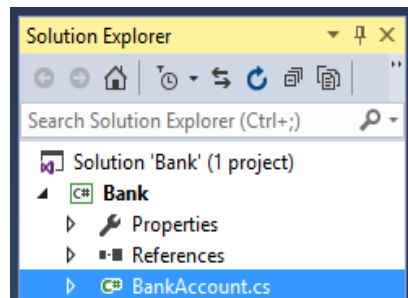
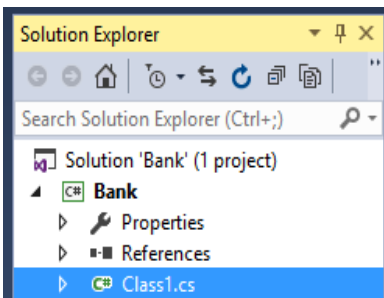
Visual C# => ClassLibrary

პროექტი სახელით Bank.



ნახ.11.1. დასატესტი კლასის პროექტის შექმნა

მიიღება 11.2 ნახაზზე ნაჩვენები Solution Explorer ფანჯარა. აქ Class1.cs სახელი შეცვალეთ BankAccount.cs -ით.



ნახ.11.2. Class1 - > BankAccount

შემდეგ BankAccount.cs -ის ტექსტი რედაქტორის არეში შევცვალეთ ჩვენი დასატესტი პროგრამის კოდით.

ეს საწყისი ტექსტი, მაგალითად, მოცემულია 11.1 ლისტინგში.

```
//-- ლისტინგი_11.1 --- BankAccount.cs -----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

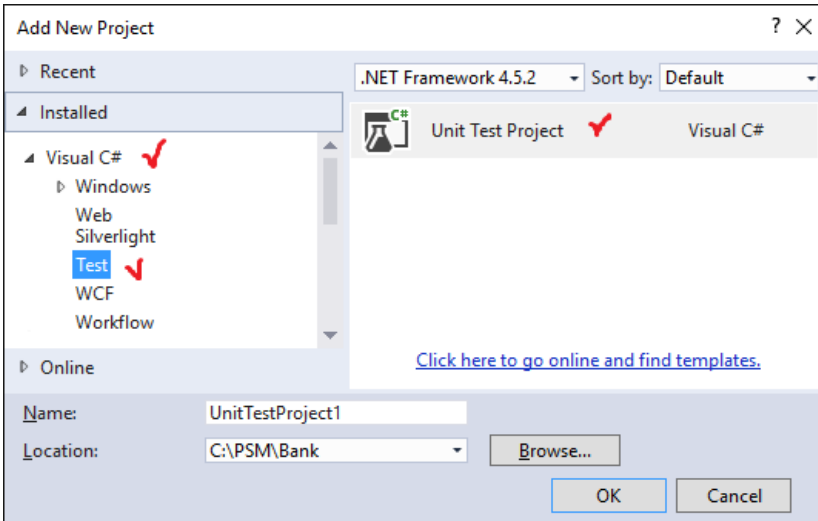
namespace BankAccountNS
{
    public class BankAccount
    {
        private string m_customerName;
        private double m_balance;
        private bool m_frozen = false;
        private BankAccount()
        {
        }

        public BankAccount(string customerName, double balance)
        {
            m_customerName = customerName;
            m_balance = balance;
        }
        public string CustomerName
        {
            get { return m_customerName; }
        }
        public double Balance
        {
            get { return m_balance; }
        }
        public void Debit(double amount)
        {
            if (m_frozen)
            {
                throw new Exception("Account frozen");
            }
            if (amount > m_balance)
            {

```

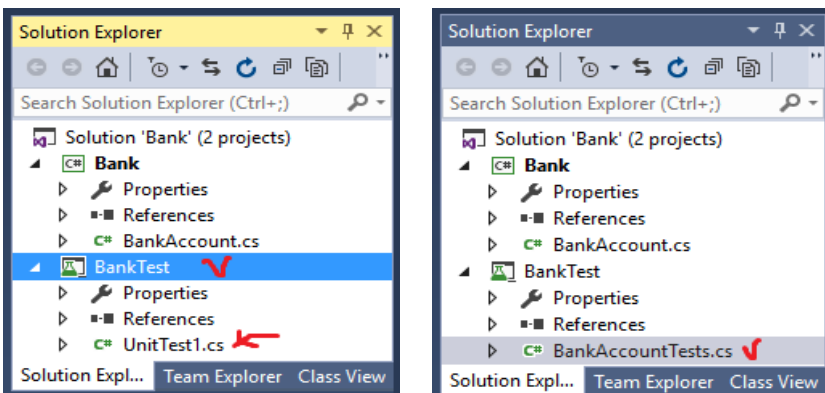
```
        throw new ArgumentOutOfRangeException("amount");
    }
    if (amount < 0)
    {
        throw new ArgumentOutOfRangeException("amount");
    }
    m_balance += amount; // განზრახ არასწორი კოდი
    // m_balance -= amount; // გასწორებული
}
public void Credit(double amount)
{
    if (m_frozen)
    {
        throw new Exception("Account frozen");
    }
    if (amount < 0)
    {
        throw new ArgumentOutOfRangeException("amount");
    }
    m_balance += amount;
}
private void FreezeAccount()
{
    m_frozen = true;
}
private void UnfreezeAccount()
{
    m_frozen = false;
}
public static void Main()
{
    BankAccount ba = new BankAccount("Mr.Bryan Walton",
                                     9.99);
    ba.Credit(5.77); ba.Debit(9.22);
    Console.WriteLine("Current balance is ${0}",
                      ba.Balance);
}
}
```

- ტესტ-ფაილის პროექტის აგება (Unit Test Project):



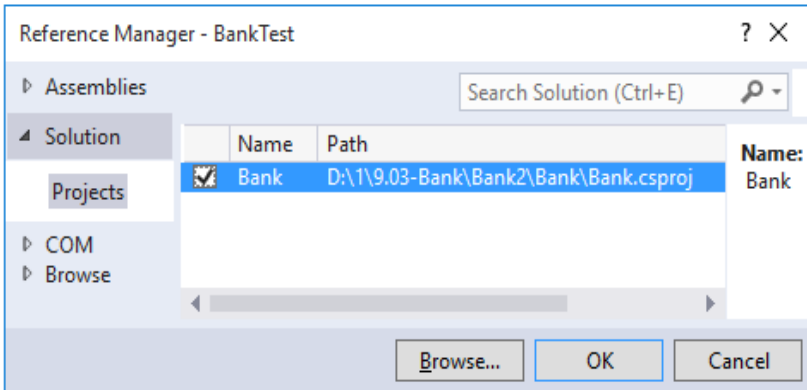
ნახ.11.3. Unit Test პროექტის შექმნა

მივიღებთ 14.4. ნახაზზე ნაჩვენებ სურათს BankTest პროექტით. მარჯვენა სურათზე შეცვლილია UnitTest კლასის სახელი BankAccountTests-ით.



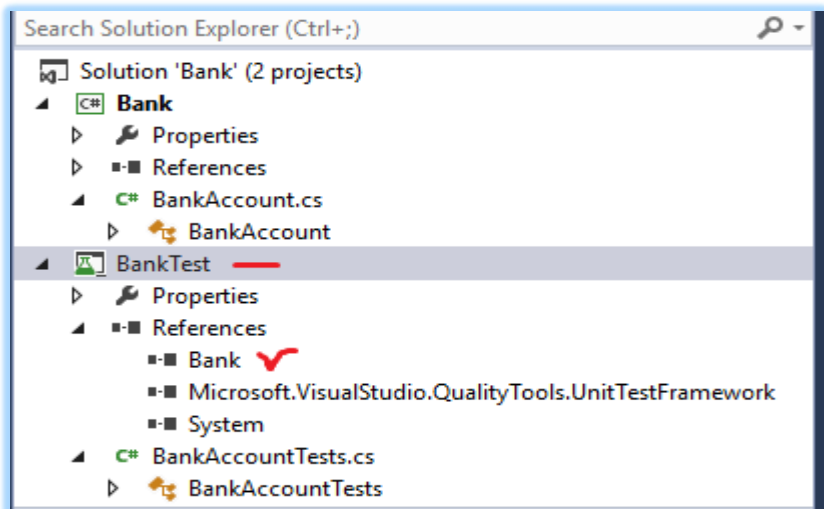
ნახ.11.4

BankTests პროექტში დავამატოთ reference **Bank** solution-იდან. ამისათვის **BankTests**-ზე მაუსის მარჯვენა ღილაკით ავირჩიოთ **Add Reference** და მივიღებთ 11.5 ნახაზზე ნაჩვენებ ფანჯარას. აქ Solution სტრიქონში ვირჩევთ Projects და Bank-ის ჩეკბოქსს მოვნიშნავთ.



ნახ.11.5

მივიღებთ 14.6 ნახაზზე ნაჩვენებ შედეგს.



ნახ.11.6

ჩვენამდომამდე BankAccountTests პროგრამაში სახელსივრცე Bank-ის პროექტიდან: `using BankAccountNS;`

ამგვარად, BankAccountTests.cs ფაილს ექნება 11.2 ლისტინგზე ნაჩვენები სახე.

```
//-- ლისტინგი_11.2 ----- BankAccountTests.cs -----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
    [TestClass]
    public class BankAccountTests
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

ახლა შევქმნათ პირველი ტესტ-მეთოდი. ამ პროცედურაში, დაიწერება უნიტ-ტესტის მეთოდები BankAccount class-ის Debit მეთოდის ქცევის ვერიფიკაციისათვის. ეს მეთოდები ზემოთაა ჩამოთვლილი.

დასატესტი მეთოდების ანალიზის გზით გაირკვა, რომ საჭიროა მინიმუმ სამი ქცევის შემოწმება:

1. მეთოდი ქმნის `ArgumentOutOfRangeException` - გამონაკლისს, თუ კრედიტის ჯამი გადააჭარბებს ბალანსს;
2. იგი ქმნის `ArgumentOutOfRangeException` - გამონაკლისს მაშინაც, როცა კრედიტის ზომა უარყოფითია;
3. თუ 1 და 2 პუნქტები წარმატებით დასრულდა, მაშინ მეთოდი ითვლის ჯამს ბალანსის ანგარიშიდან.

პირველ ტესტში შევამოწმოთ, რომ კრედიტის დასაშვები მნიშვნელობისთვის (როცა დადებითი მნიშვნელობისაა და ბალანსის ანგარიშზე ნაკლებია) ანგარიშიდან მოიხსნება საჭირო თანხა.

1. დავამატოთ BankAccountTests კლასს შემდეგი მეთოდი:

```
// unit test code ----  
[TestMethod]  
public void Debit_WithValidAmount_UpdatesBalance()  
{  
    // arrange  
    double beginningBalance = 9.99;  
    double debitAmount = 4.55;  
    double expected = 7.44;  
    BankAccount account = new BankAccount("Mr. Dito",  
        beginningBalance);  
    // act  
    account.Debit(debitAmount);  
    // assert  
    double actual = account.Balance;  
    Assert.AreEqual(expected, actual, 0.001, "Account not debited  
        correctly");  
}
```

მეთოდი საკმაოდ მარტივია. ჩვენ ვქმნით ახალ BankAccount ობიექტს საწყისი ბალანსით და შემდეგ ვაკლებთ სწორ ოდენობას. ჩვენ ვიყენებთ Microsoft-ის unit-ტესტის ფრეიმვორკს მართვადი კოდის AreEqual მეთოდისთვის, რათა მოხდეს საბოლოო ბალანსის ვერიფიკაცია - არის ის, რასაც ჩვენ ველით.

ტესტ-მეთოდის მოთხოვნები ასეთია:

1. მეთოდი მონიშნული უნდა იყოს [TestMethod] ატრიბუტით;
2. მეთოდმა უნდა დააბრუნოს void;
3. მეთოდს არ შეიძლება ჰქონდეს პარამეტრები.

ტესტის აგება და ამუშავება:

მოლანი ტესტის კოდი მოცემულია 11.3 ლისტინგში.

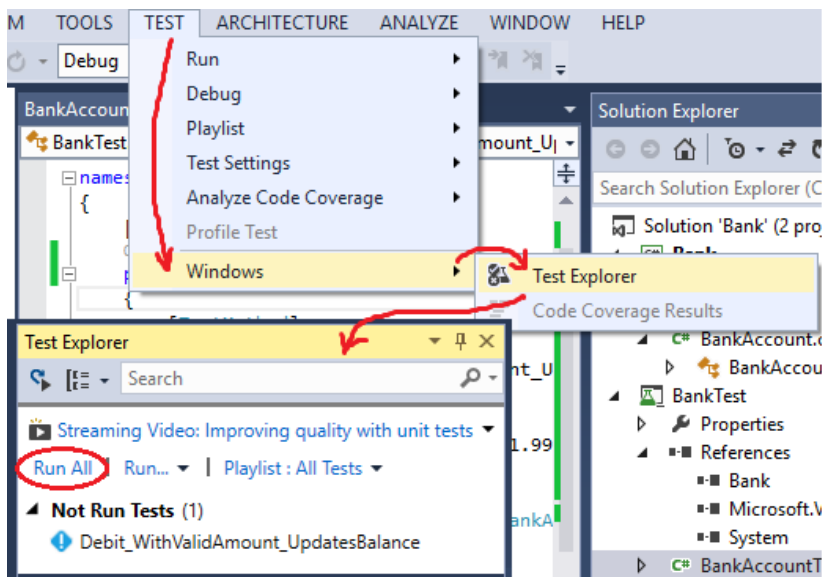
```
//-- ლისტინგი_11.3 ----- BankAccountTests.cs ----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
    [TestClass]
    public class BankAccountTests
    {
        [TestMethod]
        public void Debit_WithValidAmount_UpdatesBalance()
        {
            // arrange
            double beginningBalance = 9.99;
            double debitAmount = 4.55;
            double expected = 7.44;
            BankAccount account = new BankAccount("Mr. Dito",
                beginningBalance);

            // act
            account.Debit(debitAmount);

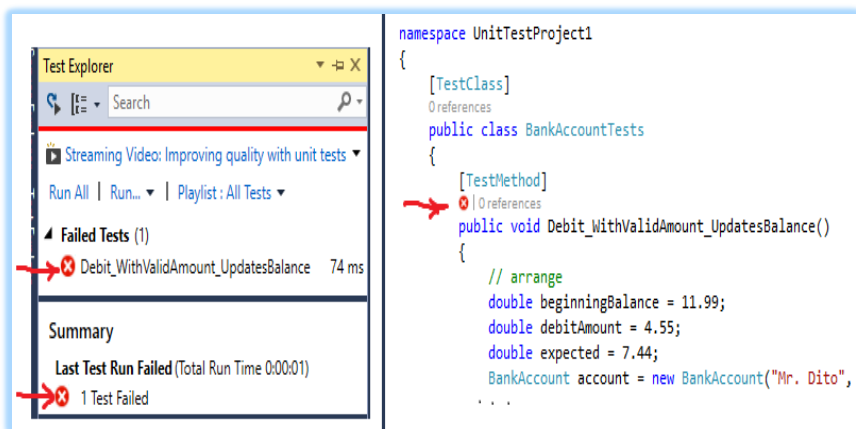
            // assert
            double actual = account.Balance;
            Assert.AreEqual(expected, actual, 0.001, "Account
                not debited correctly");
        }
    }
}
```

- BUILD მენიუდან ვირჩევთ Build Solution;
 - TEST მენიუდან ვირჩევთ Windows და Test Explorer პუნქტებს.
- იხსნება Test Explorer ფანჯარა (ნახ.11.7).



ნახ.11.7.

აქ ვირჩევთ Run All - ს და ვვლტებულობთ შედეგს (ნახ.14.8).



ნახ.11.8

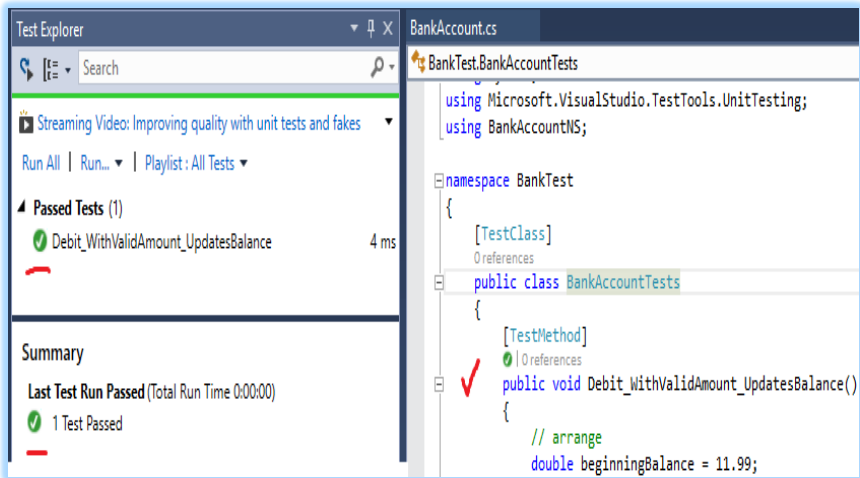
ნახაზზე მითითებული „x“-სიმბოლოები წითელ წრეშია მოთავსებული, ე.ი. ტესტირებამ აღმოაჩინა შეცდომები და კოდი ვერ შესრულდა წარმატებით.

თუ მეთოდი წარმატებით ჩაივლიდა, მაშინ მივიღებდით მწვანე ფერის x-სიმბოლოებს.

შემდეგი ეტაპი კოდის გასწორება და ხელახალი ტესტირებაა. დასატესტ პროგრამაში შევცვალოთ სტრიქონში „+“, ნიშანი „-“, -ით.

```
// m_balance += amount; // intentionally incorrect code  
m_balance -= amount; // intentionally correct code
```

ტესტის თავიდან ამუშავებით ვღებულობთ წარმატებულ შედეგს, ანუ მიიღება მწვანე ფერის სიმბოლოები (ნახ.11.9).



ნახ.11.9. სწორი შედეგი

დავალება: ააგეთ განხილული მაგალითის ანალოგიურად C# პროგრამული კოდი და ჩაატარეთ ტესტირების ექსპერიმენტი.

12. ლაბორატორიული სამუშაო N12

პროგრამული კოდების ხარისხის შეფასება

მიზანი: პროგრამული პროექტების შემუშავების პროცესში პროგრამული მოდულების ხარისხის შეფასების მეთოდების შესწავლა.

პროგრამული უზრუნველყოფის ხარისხი მოცემული დავალების ფარგლებში დადგენილი მოთხოვნილებების სრულყოფილად შესაბამისობაა რეალიზებულ პროგრამულ პროდუქტთან. პროგრამული უზრუნველყოფის ხარისხის საერთაშორისო სტანდარტებია - ISO/IEC 25000:2014, IEEE Std 68.12-1990 [11,12].

პროგრამული უზრუნველყოფის ხარისხის მახასიათებლები არაფუნქციონალური მოთხოვნების ნაწილია, რომელიც ძირითადად მოიცავს შემდეგ კრიტერიუმებს:

გასაგები - პროგრამული უზრუნველყოფის დანიშნულება გასაგები უნდა იყოს როგორც სისტემის მუშაობიდან, ისე მისი დოკუმენტაციიდან;

სრული - პროგრამული უზრუნველყოფის ყველა აუცილებელი კომპონენტი უნდა იყოს სრულად წარმოდგენილი და რეალიზებული;

ლაკონური - ზედმეტი და დუბლირებული ინფორმაცია უნდა იყოს შეზღუდული. კოდის განმეორებადი ნაწილებისთვის დაცული უნდა იყოს პოლიმორფიზმის პრინციპი;

პორტირების შესაძლებლობა - პროგრამული უზრუნველყოფა ადვილად უნდა ადაპტირდეს ახალ გარემოში (მაგალითად, არქიტექტურის, პლატფორმის, ვერსიის და ა.შ. შეცვლისას.)

შეთანხმებული - პროგრამული უზრუნველყოფის ნებისმიერ კომპონენტში (პროგრამული კოდი, ტექნიკური დავალება,

ტექნიკური პროექტი, დოკუმენტაცია და ა.შ.) გამოყენებულ უნდა იქნას ერთიანი, შეთანხმებული ტერმინოლოგია და აღნიშვნები.

არსებობს კოდის შემოწმებისა და ანალიზის შემდეგი მიდგომები:

1. **Maintainability Index** (მხარდაჭერის ინდექსი) - კოდის ხარისხის კომპლექსური მაჩვენებელია. იგი განისაზღვრება ფორმულით [40]:

$$MI = \text{MAX}(0, (171 - 5.2 * \ln(HV) - 0.23 * CC - 14.2 * \ln(\text{LoC})) * 100 / 171),$$

სადაც,

- HV – Halstead Volume, გამოთვლითი სირთულე. მეტრიკის სიდიდე პირდაპირპროპორციულად იზრდება გამოყენებული ოპერატორების სიმრავლის შესაბამისად;

- CC – Cyclomatic Complexity. კოდის სტრუქტურული სირთულე ანუ კოდში სხვადასხვა განშტოებების რაოდენობა. რაც უფრო მაღალია მაჩვენებელი მითი უფრო მეტი ტესტირებებია გასაწერი;

- LoC – კოდის სტრიქონების რაოდენობა.

ამ მეტრიკის ფარგლებში, კოდის სირთულის მაჩვენებელი ვარიირებს 0-დან 100-მდე. რაც უფრო მაღალია მნიშვნელობა, მით უფრო მოქნილია კოდის მხარდაჭერა.

Visual Studio პაკეტში თვალსაჩინოებისთვის შემუშავებულია ფერებად რანჟირებული დაყოფა - მწვანე ფერი შეესაბამება 20-100 დიაპაზონის მნიშვნელობას, ყვითელი ფერი შეესაბამება 10-20 დიაპაზონის მნიშვნელობას, ხოლო 10-ზე ნაკლების დიაპაზონის მნიშვნელობის ფერია წითელი.

2. **Depth of Inheritance** – მემკვიდრეობითობის სიღრმე. თითოეული კლასისთვის აჩვენებს მემკვიდრეობის ჯაჭვის იერარქიას. მაგალითად, პირველი კლასის მემკვიდრეა მეორე კლასი, ხოლო მესამე კლასი მეორე კლასის მემკვიდრეა. შესაბამისად, პირველი კლასის მაჩვენებელია 1, მეორესი 2, მესამესი 3.

3. **Class Coupling** – კლასების ურთიერთდამოკიდებულების და დაკავშირების მაჩვენებელი.

კარგი პრაქტიკაა კლასებს შორის დამოკიდებულება არ იყოს „ჩახლართული“ და არ იყოს გამოყენებული დიდი რაოდენობის ქვეკავშირები (გამოთვლაში მონაწილეობს - პარამეტრული კლასები, ლოკალური ცვლადები, მეთოდით დაბრუნებული ტიპები, საბაზო კლასები, ატრიბუტები და სხვ.)

4. **Lines of Code** – კოდის სტრიქონების რაოდენობა (არ გაითვალისწინება კოდში ცარიელი სტრიქონები და კომენტარები).

Visual Studio პაკეტში პროგრამული კოდის მეტრიკის შედეგების მიღება პროექტისთვის ხდება შემდეგი ბრძანებით:

ძირითად მენიუმში ღილაკით Analyze - Calculate Code Metrics - for Solution.

ასევე, შესაძლებელია ნაწილში Solution Explorer, solution-Calculate Code Metrics გამოძახება თავუნას მარჯვენა ღილაკით.

აღნიშნული ბრძანება აჩვენებს როგორც ზოგად, ისე კლასების დონეზე ჩაშლილ შედეგს (Code Metrics Results) პარამეტრებით - Maintainability Index, Cyclomatic Complexity, Depth of Inheritance, Class Coupling, Lines of Code (ნახ.12.1).

კოდში კომპილატორის შეცდომების ან გაფრთხილების ფანჯარა Error List იხსნება ძირითადი მენიუს ღილაკით View-Error List, ასევე ძირითად მენიუმში ღილაკით Analyze – Run code Analysis and suppress active issues.

კომპილატორის გაფრთხილება (Compiler warnings) - პროგრამულ კოდში საექვო ადგილების არსებობაა, რომელიც პროგრამული ენის თვალსაზრისით არ არის შეცდომა და არ იწვევს პროგრამული კოდის კომპილირების პროცესის შეწყვეტას, თუმცა არის პროგრამული შეცდომა.

„პროგრამული სისტემების მენეჯმენტი: ლაბორატორიული პრაქტიკუმი“

Hierarchy	Maintainability Index	Cyclomatic Comple...	Depth of Inheritance	Class Coupling ▲	Lines of Code
ProjectBudjet (Debug)	61	50	7	51	439
ProjectBudjet	61	50	7	51	439
Program	81	1	1	3	3
gamot/lebi	69	7	1	7	16
Form_SubXarj	50	9	7	25	118
button1_Click(object)	92	1	3	3	1
Dispose(bool): void	80	3	5	3	3
Form_SubXarj(Form_	62	1	5	5	10
dataGridView_SubXarj	67	3	5	5	6
InitializeComponent()	32	1	19	19	98
Form_Xarj	58	15	7	30	103
Form3	45	18	7	42	199

ნახ.12.1

კომპილიატორის გაფრთხილება შესაძლებელია მნიშვნელოვანი იყოს საინფორმაციო სისტემების რისკების მართვის პროცესისთვის, რაც იმის მანიშნებელია რომ კოდს შესაძლოა გააჩნდეს სისუსტეები, ღია ადგილები ან გადატვირთული გამოუყენებელი ელემენტები. ასეთი ტიპის შევდომებმა შესაძლოა გამოიწვიოს კოდის შესრულების შენელებაც.

ნაწილში (Warnings) ველი suppress (გაბათილებადა) მიანიშნებს თუ კოდის რომელი ელემენტია გამოცხადებული თუმცა გამოუყენებელი.

13. ლაბორატორიული სამუშაო N13

პროექტის ბიუჯეტის შედგენისთვის სამომხმარებლო აპლიკაციის მომზადება

მიზანი: პროექტის ბიუჯეტისა და ხარჯის ელემენტების გაცნობა. ცხრილის ჩანაწერებსა და ტესტურ ველებში მონაცემების მიმოცვლა. ორ დიალოგურ ფორმას შორის (კლასებს შორის) მონაცემების ურთიერთგაცვლა. პროექტის ბიუჯეტის ბაზაზე ხარჯებისა და ქვებარჯების აღრიცხვა.

პროექტების მართვაში ერთ-ერთი მთავარი დატვირთვა აქვს პროექტის ბიუჯეტის მართვის საკითხს. პროექტების მართვის სტანდარტის მიხედვით, პროექტების მართვის მოდელია ე.წ. სამმაგი შეზღუდვა ან რკინის სამკუთხედი, რომლის პარამეტრებია მიზანი, ბიუჯეტი/ღირებულება და გეგმა.

პროექტების მართვისას ფასთან მიმართებაში განიხილება ორი ვარიანტი: 1. პროექტის ბიუჯეტი ცნობილია და უნდა გადანაწილდეს თანხა რესურსებისა და დასახარჯი დროის მიხედვით (აღმავალი მიდგომა); 2. პროექტის ღირებულება უნდა შედგეს რესურსებზე თანხის გადანაწილებით გონივრულ ფარგლებში (დაღმავალი მიდგომა).

მიღებულია პროექტის ბიუჯეტის შედგენისას შემდეგი ხარჯების გათვალისწინება:

- პირდაპირი ხარჯები: მაგალითად, ადამიანური რესურსები, საკონსულტაციო მომსახურება, პროგრამული უზრუნველყოფის შესყიდვა/ლიცენზიები, მივლინება, აპარატურა/მომსახურების ხარჯი
- ირიბი ხარჯი: მაგალითად, ზედნადები ხარჯები სატელე-ფონო/საკომუნიკაციო ხარჯი, დაზღვევა (მაგალითად, მივლინების დროს), საკანცელარიო ხარჯი, სხვა.

განვიხილოთ ბიუჯეტის ფორმირების ამოცანა. ამოცანის გამარტივებისთვის გავაერთიანოთ პირდაპირი და ირიბი ხარჯები - ხარჯების კატეგორიად.

ვარიანტი 1. აღმავალი მიდგომა - ცნობილია პროექტის ბიუჯეტი.

სცენარი:

1. პროექტის ბიუჯეტის ფორმირების დიალოგური ფორმა (Form_Xarji), სადაც:

2. ველში პროექტის ბიუჯეტი (Textbox_budjet) მიეთითება ბიუჯეტისთვის განკუთვნილი თანხა

3. ხარჯების ცხრილში (DataGrid_Xarji) მიეთითება - ხარჯების კატეგორია (მაგ., ადამიანური რესურსები) და ხარჯი

4. ცხრილი - DataGrid_Xarji შედგება 3 სვეტისგან (Column-col_kategoria, col_xarji, col_button). აქედან, col_button სვეტის ტიპია DataGridViewButtonColumn (ლილაკის ტიპი, რომლის საშუალებითაც შესაძლებელია ოპერაციების გამოძახება/შესრულება);

5. ხარჯების კატეგორია შესაძლებელია შეიცავდეს ქვე-ხარჯებს (მაგალითად, ადამიანური რესურსებია ბიზნეს-ანალიტიკოსი, პროგრამისტი და სხვ. განსხვავებული სახელფასო თანხით). col_button ლილაკი გამოიყენება ხარჯების ქვეკატეგორიის თანხების ჯამის შესავსებად და გამოსათვლელად შემდეგი პრინციპით:

6. ხარჯების ქვეკატეგორიის ფორმირების დიალოგური ფორმა (Form_SubXarji):

6.1. თუ ხარჯების კატეგორია განშტოვდება ქვე-ხარჯებად, მაშინ იხსნება დიალოგური ფორმა - Form_SubXarji, იგივე მართვის ობიექტებით, სადაც ცხრილში (DataGrid_SubXarji) მოხდება შესაბამის ხარჯის ქვე-კატეგორიის შევსება.

6.2. ცხრილი - DataGrid_SubXarji შედგება 2 ველისგან ხარჯების ქვეკატეგორია (Col_SubKategoria) და თანხა (Col_SubTanxa)

6.3. დიალოგურ ფორმაში Form_SubXarji, ცხრილის - DataGrid_SubXarji, ველის Col_SubTanxa - ჯამი იკრიბება და გადმოცემა დიალოგურ ფორმას Form_Xarji. ქვეკატეგორიის ხარჯების ჯამი ჯდება DataGrid_Xarji-ის შესაბამის ველში, საიდანაც გამომახებულ იქნა დამატებითი დიალოგური ფორმა.

7. ძირითად ფორმაზე - Form_Xarji, ველში Textbox_Xarji, ჩაიწერება საერთო ხარჯების ჯამი დინამიურად ცხრილის DataGrid_Xarji ველიდან Col_Tanxa, რაც ასევე დინამიურად დააკლდება ველს Textbox_budjet.

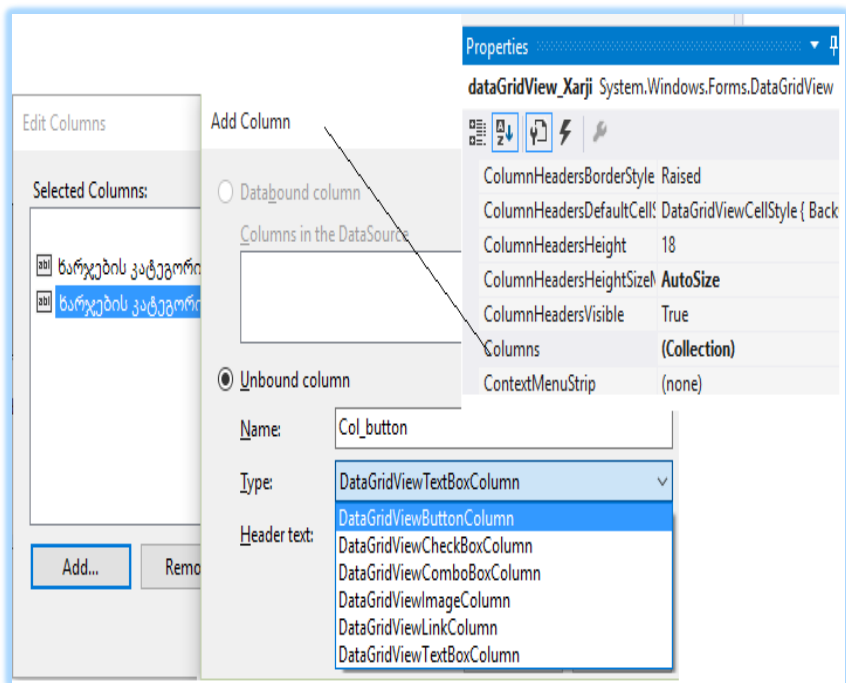
DataGridView ცხრილში DataGridViewButtonColumn სვეტის შექმნის მაგალითი წარმოდგენილია 13.1 ნახაზზე.

დიალოგური ფორმების სტრუქტურა, პროგრამულ კოდში გამოყენებული მართვის ობიექტების დასახელებებით ნაჩვენებია 13.2 ნახაზზე.

ამოცანის სარეალიზაციოდ ვხსნით 3 კლასს.

1. ძირითადი დიალოგური ფორმა - Form_Xarji
2. დამატებითი დიალოგური ფორმა - Form_SubXarji
3. დამხმარე კლასი gamotvlebi.cs, სადაც ვწერთ ერთ საერთო მეთოდს ორივე დიალოგურ ფორმაში გამოსაყენებლად

ობიექტ-ორიენტირებული დაპროგრამების პრინციპებიდან გამომდინარე შესაძლებელია ერთი კლასის ობიექტებზე მანიპულაცია მეორე კლასიდან. მოცემულ მაგალითში გამოიყენება ერთი დიალოგიდან, მეორე დიალოგის გამოძახება, პირველი დიალოგის მონაცემის გადაცემა მეორე დიალოგში, მეორე დიალოგში მონაცემის დამუშავება და უკან დაბრუნება პირველ დიალოგში.



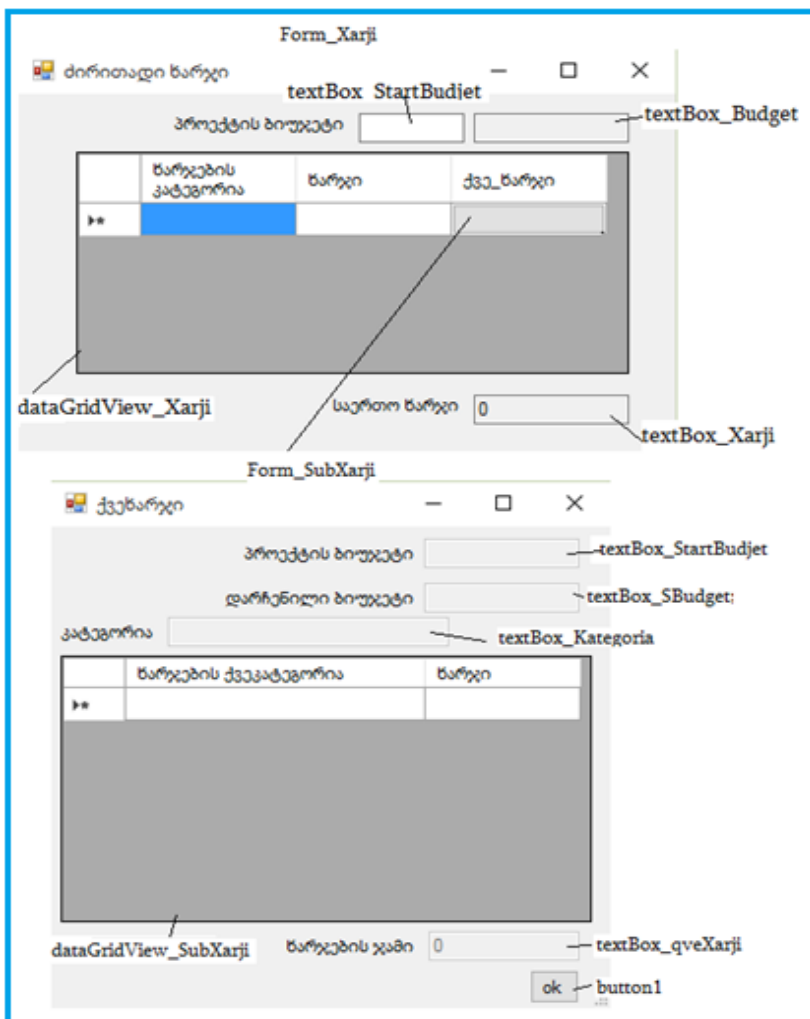
ნახ.13.1. DataGridView ცხრილში DataGridViewButtonColumn სვეტის შექმნის მაგალითი

ამისათვის ხდება შემდეგი ოპერაციების ჩატარება:

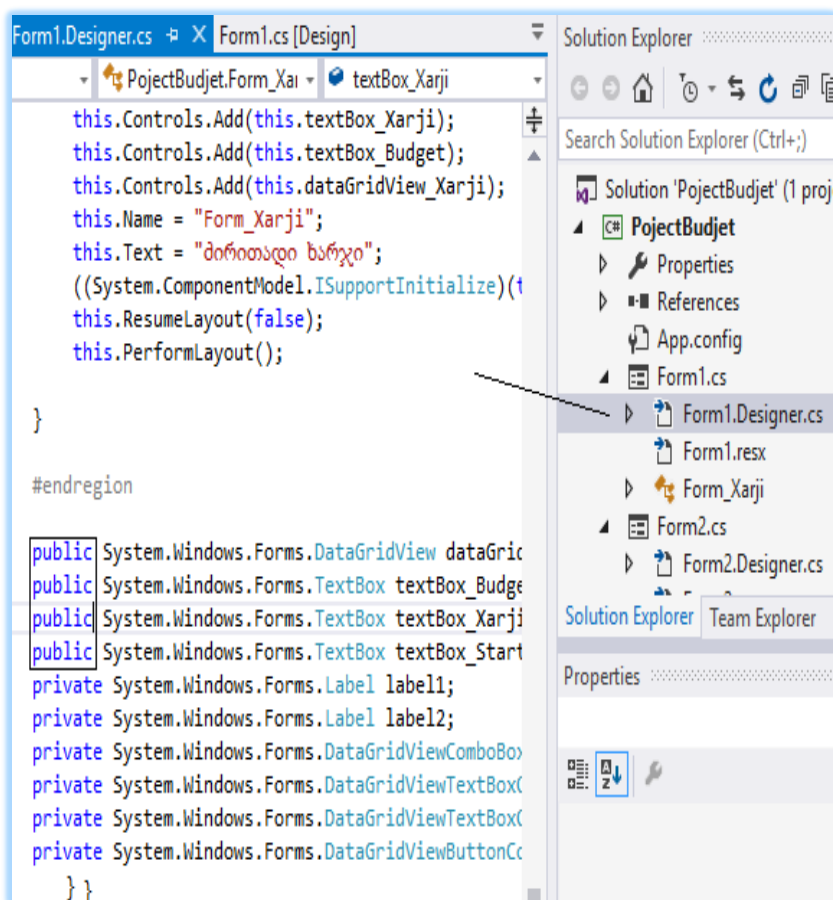
1. იმ ობიექტს, რომელიც უნდა მანიპულირდეს მეორე კლასიდან, პირველ კლასში მიენიჭება ღია (Public) სტატუსი. დიალოგური ფორმის კლასის შემთხვევაში ობიექტზე Public სტატუსის მინიჭება ხდება -designers.rs ნაწილიდან, რომელიც ნებისმიერი დიალოგური ფორმის კომპლექტში შედის, ფორმის მართვის ელემენტების აღწერისთვის (ნახ.13.3);

2. მეორე კლასს, რომლის გამოძახებაც ხდება პირველი კლასიდან პარამეტრად გადაეცემა პირველი კლასის ფორმის ობიექტი (შესაძლებელია ჩაიწეროს - this). მეორე კლასი მოითხოვს ამისათვის პარამეტრს პირველი კლასის პარამეტრის მითითებას (ნახ.13.4).

3. აღწერილი ოპერაციების წარმატებით განხორციელებისას, მეორე კლასი, ხედავს პირველი კლასის ყველა ღია ობიექტს.



ნახ.13.2. დიალოგური ფორმების სტრუქტურა, პროგრამულ კოდში გამოყენებული მართვის ობიექტების დასახელებებით



ნახ. 13.3. დიალოგური ფორმის ობიექტზე Public სტატუსის მინიჭება


```
private void dataGridView_Xarji_CellClick(object sender, DataGridViewCellEventArgs e)
{
    if (e.ColumnIndex == 3) {
        String kategoria = dataGridView_Xarji.Rows[e.RowIndex].Cells[1].Value.ToString();
        Form_SubXarji subxarji = new Form_SubXarji(this, kategoria, e.RowIndex);
        subxarji.ShowDialog();
    }
}

```

პირველი კლასი

```
public partial class Form_SubXarji : Form
{
    gamotvlebi gamotvla = new gamotvlebi();
    public int row_indexs;
    public string darchenili;
    private Form_Xarji ziritadia;
}

```

მეორე კლასი

```
public Form_SubXarji(Form_Xarji ziritadi, string kategoria, int row_index)
{
    InitializeComponent();
    textBox_Kategoria.Text = kategoria;
    this.row_indexs = row_index;
    ziritadia = ziritadi;
    this.darchenili = ziritadia.textBox_Budget.Text;
    textBox_StartBudjet.Text = ziritadia.textBox_StartBudjet.Text;
    textBox_SBudget.Text = ziritadia.textBox_Budget.Text;
}

```

ნახ.13.4. კლასის ობიექტის პარამეტრად გადაცემა სხვა კლასში და მისი გამოყენება

კოდის ლისტინგი:

1. ძირითადი დიალოგური ფორმა - Form_Xarji:

```
namespace PojectBudjet
{
    public partial class Form_Xarji : Form
    {
        public string tanxa_sub = ""; // Form_SubXarji - ფორმიდან
        // ქვეხარჯის მონაცემის მიღებისთვის
        public String kategoria = ""; // Form_SubXarji - ფორმაზე
        // მონაცემის გადაცემისთვის
        public int row_indexs = 0; // Form_SubXarji - ფორმაზე ცხრილის
        // იმ ჩანაწერის ნომრის გადაცემისთვის, სადაც უნდა
        // ჩაჯდეს დაბრუნებული მონაცემი
        public Form_Xarji()
        {
            InitializeComponent();
        }
        private void sumBudjet() //გამოთვლის მეთოდი. ამ მეთოდით ხდება
        // სასტარტო ბიუჯეტისა და ხარჯის დინამიური გამოკლება
        {
            try // გამორიცხვის ოპერაცია
            {
                if (textBox_StartBudjet.Text != "")
                {
                    //ტექსტური ველის რიცხობრივ ველად გადაკეთება ანგარიშისთვის
                    Double startBudjet = Double.Parse(textBox_StartBudjet.Text);
                    Double xarji = Double.Parse(textBox_Xarji.Text);
                    textBox_Budget.Text = (startBudjet - xarji).ToString();
                }
            }
        }
    }
}
```

```
catch (FormatException) // გამორიცხვის ოპერაცია, რაც
    // გამოსათვლელ ველებში რიცხვისა და სიმბოლოს
    // მითითებას აკონტროლებს
{
    MessageBox.Show("only nummber");
}
}
```

```
private void dataGridView_Xarji_CellValueChanged(object sender,
    DataGridViewCellEventArgs e)
    // ჯდება ავტომატურად ამ ფორმის დიზაინში ცხილის მოვლენებში
    // CellValueChange 2-ჯერ მაუსის დაწკაპუნებით
{
    // დამხმარე კლასის ობიექტის შექმნა
    gamotvlebi gamotvla = new gamotvlebi();

    if (e.ColumnIndex == 2) // მე-2 ველი არის ხარჯის ველი ცხრილში
    {
        // დამხმარე კლასის მეთოდს გადაეცემა 2 პარამეტრი - ცხრილის
        // ობიექტი და სვეტის ნომერი
        textBox_Xarji.Text =
            gamotvla.CellSum(dataGridView_Xarji,2).ToString();
        sumBudjet(); //გამოთვლის მეთოდის გამოძახება
    }
}

public void cxrilis_shevseba(string tanxa_sub, int rowid,
    ComboBox fill_subcategory)
{
    // ეს მეთოდი ავსებს სხვა ქვეხარჯების ფორმიდან მონაცემებს
    // ჩამატებას ცხრილის ხარჯების ველში
    dataGridView_Xarji.Rows[rowid].Cells[2].Value = tanxa_sub;
}
}
```

// ცხრილში ღილაკის ველზე დაჭერის ოპერაცია

```
private void dataGridView_Xarji_CellClick(object sender,
                                         DataGridViewCellEventArgs e)
{ // ჯდება ავტომატურად ამ ფორმის დიზაინში ცხრილის
  // მოვლენებში CellClick 2-ჯერ მაუსის დაწკაპუნებით
  if (e.ColumnIndex == 3) // მე-3 ველი არის ღილაკის ველი ცხრილში
  {
    try // გამორიცხვის ოპერაცია
    {
      kategoria = dataGridView_Xarji.Rows[e.RowIndex].
        Cells[1].Value.ToString();
      row_indexs = e.RowIndex;
      // ქვეკატეგორიის დიალოგური ფორმის ობიექტის შექმნა
      Form_SubXarji subxarji = new Form_SubXarji(this);
      // ქვეკატეგორიის დიალოგური ფორმის გამოძახება
      subxarji.ShowDialog();
    }
    catch // გამორიცხვის ოპერაცია, აკონტროლებს ცხრილში
      // ხარჯვის კატეგორიის ველის შევსებას
    { MessageBox.Show("შეავსეთ ცხრილში ხარჯვის კატეგორიის ველი"); }
  }
}

private void textBox_StartBudget_TextChanged(object sender,
EventArgs e)
{
  sumBudget();//გამოთვლის მეთოდის გამოძახება
}
}
```

2. დამატებითი დიალოგური ფორმა - Form_SubXarji

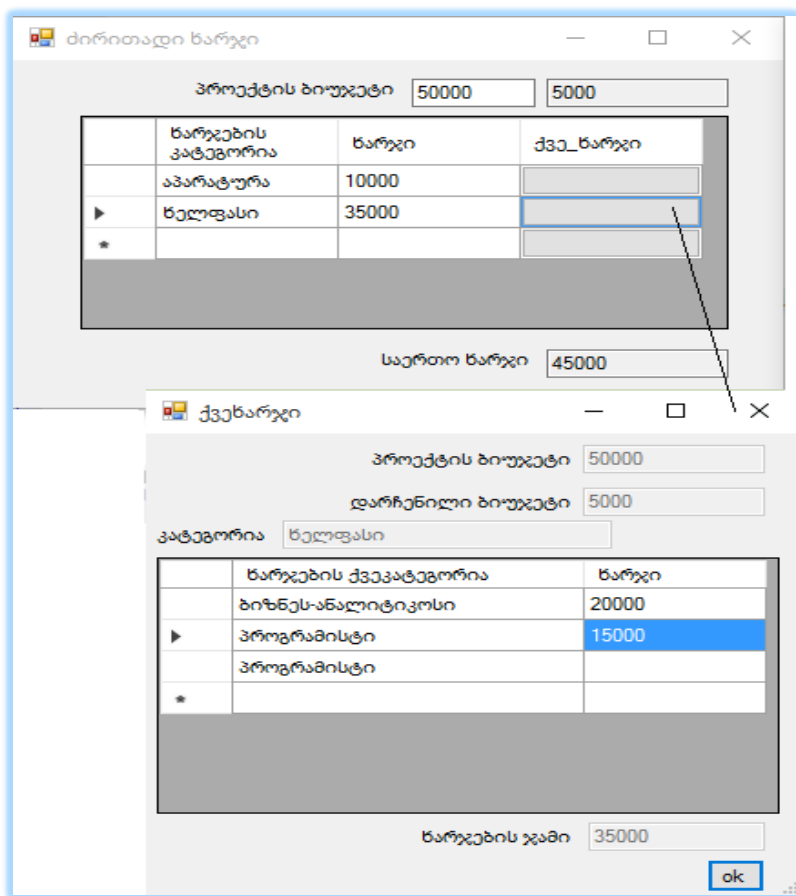
```
namespace ProjectBudjet
{
    public partial class Form_SubXarji : Form
    {
        gamotvlebi gamotvla = new gamotvlebi();
        // პირველი კლასიდან გადმოცემული ცვლადისთვის
        public int row_indexes;
        public string darchenili;
        private Form_Xarji ziritadia; // პირველი კლასის ობიექტის შექმნა
        public Form_SubXarji(Form_Xarji ziritadi)
        {
            InitializeComponent();
            // პირველი კლასის ობიექტზე გადმოცემული კლასის ობიექტის მინიჭება
            ziritadia = ziritadi;
            // პირველი კლასიდან გადმოცემული მონაცემის მინიჭება მეორე
            // კლასის ობიექტზე-----
            textBox_Kategoria.Text = ziritadia.kategoria;
            this.row_indexes = ziritadia.row_indexes;
            this.darchenili = ziritadia.textBox_Budjet.Text;
            textBox_StartBudjet.Text = ziritadia.textBox_StartBudjet.Text;
            textBox_SBudjet.Text = ziritadia.textBox_Budjet.Text;
        }
        private void dataGridView_SubXarji_CellValueChanged(object sender,
            DataGridViewCellEventArgs e)
        {
            if (e.ColumnIndex == 1) // 1 ველი არის ხარჯის ველი ცხრილში
            {
                textBox_qveXarji.Text =
                    gamotvla.CellSum(dataGridView_SubXarji, 1).ToString();
                if (textBox_SBudjet.Text != "")
            }
        }
    }
}
```

```
{ Double darchBudget = Double.Parse(textBox_SBudget.Text);
  Double subxarji = Double.Parse(textBox_qveXarji.Text);
  textBox_SBudget.Text = (Double.Parse(darchenili) -
    subxarji).ToString();
}} }
private void button1_Click(object sender, EventArgs e)
{
  ziritadia.cxrilis_shevseba(textBox_qveXarji.Text, row_indexes);
  // პირველი კლასის მეთოდის გამოძახება. გადმოცემული ცხრილის
  // ჩანაწერის ნომრის მიხედვით ქვებარჯის გადაცემა პირველ კლასზე
}}}
```

3. დამხმარე კლასი gamotvlebi.cs

```
using System.Windows.Forms; // ვამატებთ დიალოგური ფორმის
// მართვის ელემენტების გამოსაჩენად
namespace ProjectBudget
{ public class gamotvlebi
  { public double CellSum(DataGridView dataGridView_, int cel_n)
    { double sum = 0;
      try
      { for (int i = 0; i < dataGridView_.Rows.Count; ++i)
        { double dxarji = 0;
          dxarji = Double.Parse(dataGridView_.Rows[i].Cells[cel_n]
            .Value.ToString());
          sum += dxarji; }
        }
      catch { }
      return sum;
    }
  }
```

შედეგი მოცემულია 13.5 ნახაზზე.



სურ.13.5. პროგრამის მუშაობის შედეგი

დავალება:

ააგეთ მოცემული მაგალითის მიხედვით პროექტის ბიუჯეტის ფორმირების პროგრამული კოდი , ბიუჯეტის ფინანსური მართვის მე-2 ვარიანტისთვის - პროექტის ღირებულება არაა ცნობილი. პროექტის ღირებულება შედგეს რესურსებზე ხარჯის დათვლით (დაღმავალი მიდგომა).

14. ლაბორატორიული სამუშაო N 14

პროგრამული სისტემების შექმნის დავალებათა კალენდარული გეგმის ფორმირება

მიზანი: dataGridView ცხრილისა და dateTimePicker თარიღის მართვის ელემენტებთან მუშაობა; DateTime და dataGridView კლასის სხვადასხვა მეთოდების გამოყენების შესწავლა.

ამოცანა: პროექტების მართვის სისტემისთვის დავალებათა კალენდარული გეგმის ფორმირება Gantt Chart მოდელის ბაზაზე:

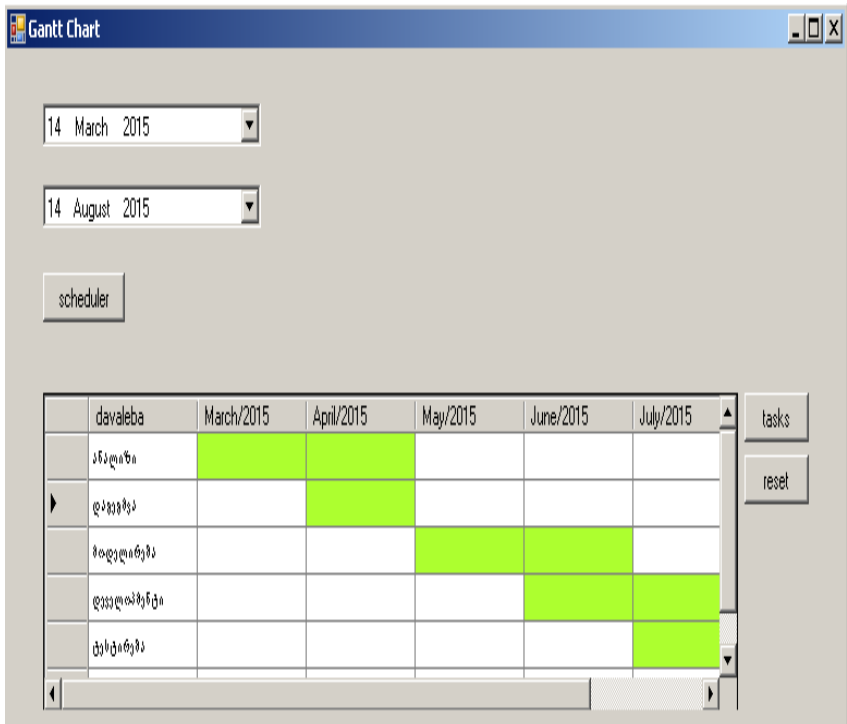
1. კალენდარული გეგმისთვის ცხრილის სვეტების ფორმირება
2. პროექტის დავალებათა გადანაწილება ცხრილში

ორ dateTimePicker (საწისი და საბოლოო თარიღი) - ობიექტიდან არჩეული თარიღების მიხედვით dataGridView ცხრილზე თარიღების გადანაწილებული ასახვა სვეტების სახით:

მაგალითად, დავთვალთ ორ არჩეულ თარიღს შორის არსებული თვეები, რომელთა რაოდენობის მიხედვით dataGridView ცხრილზე დავამატებთ FOR ციკლით სვეტებს დინამიურად. ასევე, დინამიურად მივიანიჭოთ სვეტებს თვეების დასახელება მიმდევრობით და გადავანაწილოთ დავალებები პერიოდების მიხედვით.

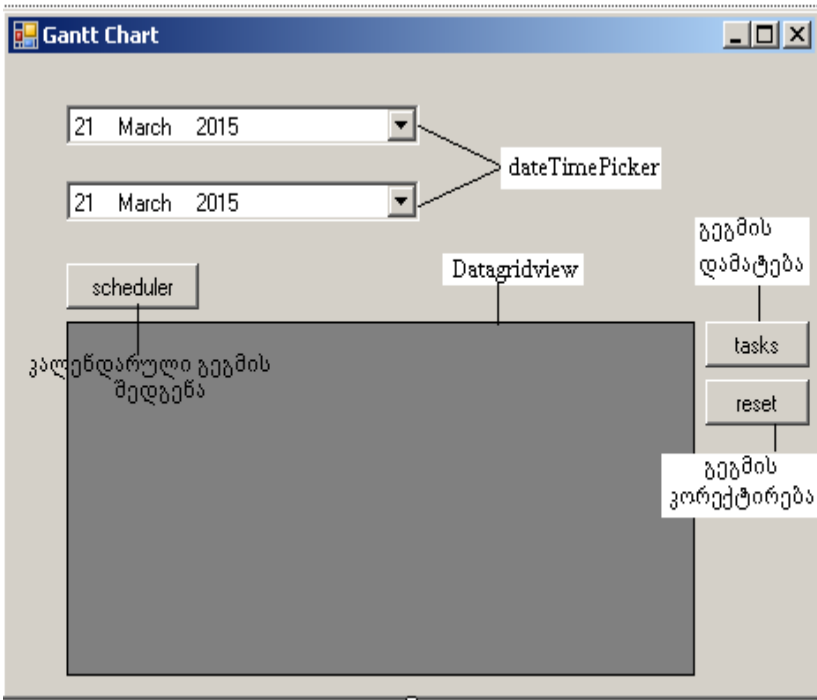
დავუშვათ მოცემულია 6 თვიანი პროექტი 5 საფეხურიანი ფაზით, რომელიც სრულდება მარტიდან-აგვისტოს ჩათვლით: ანალიზი, დაგეგმვა, მოდელირება, დეველოპმენტი და ტესტირება. ფაზების დამუშავების პერიოდის გეგმა: ანალიზი -2 თვე, დაგეგმვა შემდგომი ეტაპი 1 თვე, ანალიზის დამუშავების შუა ეტაპიდან, მოდელირება 2 თვე დაგეგმვის დასრულებიდან, დეველოპმენტისთვის საჭიროა 3 თვე და ამასთან ერთად დავალებაში გარკვევისა და მისი დახვეწისთვის საჭიროა 1 თვე ანალიტიკოსებისა და პროგრამისტების ერთობლივი მუშაობისათვის.

დავალების შესრულების შედეგი ნაჩვენებია 14.1 ნახაზზე.



ნახ.14. 1

დიალოგურ ფორმაზე მმართველი ობიექტები შემდგენაირია (ნახ.14. 2):



ნახ.14. 2

განვიხილოთ ობიექტებზე მუშაობა:

dataGridView ცხრილის მართვის ელემენტზე DataGridViewTextBoxColumn სვეტის პროგრამულად დამატება:

```
//ცხრილში TextBox სვეტის ობიექტის შექმნა  
DataGridViewTextBoxColumn column_name =  
    new DataGridViewTextBoxColumn();
```

```
//ცხრილის უჯრედის სტილის ობიექტის შექმნა  
DataGridViewCellStyle dataGridViewCellStyle_p =  
    new DataGridViewCellStyle();
```

```
// ცხრილის უჯრედის სიგანის განსაზღვრა
column_column_name.Width = 100;
// ცხრილის სვეტის სახელის დარქმევა
column_name.HeaderText = „tarigi“;
// ცხრილის სვეტის კოდური სახელის დარქმევა
column_name.Name = „tarigi“;
// ცხრილის სვეტის დამატება ცხრილზე
dataGridView1.Columns.Add(column_name.);
```

➤ **DateTime კლასით თარიღების განსაზღვრა:**

```
DateTime ad_days = dateTimePicker1.Value.AddMonths(1);
//dateTimePicker1 ობიექტიდან DateTime კლასის ობიექტზე
// „თვეების“ რაოდენობის გადაცემა
String twe_S = Thread.CurrentThread.CurrentCulture.DateTimeFormat
                .MonthNames[ad_days.Month - 1];
// თვეების სახელების გაფორმება - საჭიროა using
// System.Threading; კლასის გამოყენება
```

```
DateTime startDate = (DateTime) dateTimePicker1.Value;
DateTime endDate = (DateTime)enddata.Value;
// DateTime კლასზე dateTimePicker ობიექტის დაყვანა
```

```
TimeSpan ts = endDate.Subtract(startDate);
// TimeSpan- დროის ინტერვალი, Subtract- ორ თარიღს შორის
// სხვაობის გამოთვლა
```

```
int days = ts.Days;
// TimeSpan ობიექტით დღეების განსაზღვრის მეთოდი
int Week = days / 7;
// ორ დროის ინტერვალს შორის კვირის რაოდენობის დათვლა
```

მაგალითები:

1. ორ თარიღს შორის თვეების რაოდენობის განსაზღვრის მეთოდი

```
public int monthDifference(DateTimePicker startDate,
                          DateTimePicker endDate)
{
    int months_raod = 12 * (startDate.Value.Year - endDate.Value.Year) +
        startDate.Value.Month - endDate.Value.Month;
    // აბრუნებს აბსოლუტურ მნიშვნელობას
    return Math.Abs(months_raod);
}
```

2. ორ თარიღს შორის წლის რაოდენობის განსაზღვრის მეთოდი

```
public static int yearDifference(DateTimePicker startDate,
                                 DateTimePicker endDate)
{
    int months_raod = 12 * (startDate.Value.Year -
endDate.Value.Year) +
        startDate.Value.Month - endDate.Value.Month;
    int years = months_raod / 12;
    return Math.Abs(years); // აბრუნებს აბსოლუტურ
მნიშვნელობას
}
```

3. საწყისი და საბოლოო თარიღების მიხედვით თვეების რაოდენობის განსაზღვრა და ცხრილზე თარიღების გადანაწილებული ასახვა სვეტების სახით:

```
DataGridViewCellStyle dataGridViewCellStyle_p;
DataGridViewTextBoxColumn column_period_all;
private void button_scheduler_Click(object sender, EventArgs e)
{
    dataGridView1.Columns.Clear();
}
```

```
int month_caunt = statics.monthDifference(dateTimePicker1,
    dateTimePicker2);
for (int i = 0; i < month_caunt + 1; i++)
{
    dataGridViewCellStyle_p = new DataGridViewCellStyle();
    column_period_all = new DataGridViewTextBoxColumn();
    column_period_all.Width = 100;
    dataGridViewCellStyle_p.Format = "N2";
    column_period_all.ReadOnly = true;
    DateTime ad_days = dateTimePicker1.Value.AddMonths(i);
    String[] days_data_list =token_data(ad_days.ToString());
    String days_data = days_data_list[0];
    int year = ad_days.Year;
    String twe_S = Thread.CurrentThread.CurrentCulture
        .DateTimeFormat.MonthNames[ad_days.Month - 1];
    String tarigi = twe_S + "/" + year;
    column_period_all.HeaderText = tarigi;
    column_period_all.Name = tarigi;
    dataGridView1.Columns.Add(column_period_all);
}
}
public String[] token_data(String text)
{ char[] seps = { ' ' };
  String[] values = text.Split(seps);

  return values;
}
public int monthDifference(DateTimePicker startDate,
    DateTimePicker endDate)
{
```

```
int monthsApart = 12 * (startDate.Value.Year -
    endDate.Value.Year) +
    startDate.Value.Month - endDate.Value.Month;
return Math.Abs(monthsApart);
}
public static int yearDifference(DateTimePicker startDate,
    DateTimePicker endDate)
{
    int monthsApart = 12 * (startDate.Value.Year -
        endDate.Value.Year) +
        startDate.Value.Month - endDate.Value.Month;
    int years = monthsApart / 12;
    return Math.Abs(years);
}
```

კოდის ლისტინგი:

1. დამხმარე კლასი მეთოდებთან სამუშაოდ

```
// -- listingi 15.1----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication_IT_Project
{
// სიმბოლოების დაშლის მეთოდი
class Class_statistic
{
    public String[] token_data(String text)
    {
        char[] seps = { ' ' };
        String[] values = text.Split(seps);
        return values;
    }
}
```

```
    }  
// სიმბოლოების დაშლის მეთოდი  
public int monthDifference(DateTimePicker  
    startDate, DateTimePicker endDate)  
{  
    int monthsApart = 12 * (startDate.Value.Year -  
        endDate.Value.Year) +  
        startDate.Value.Month - endDate.Value.Month;  
  
    return Math.Abs(monthsApart);  
}  
  
public static int yearDifference(DateTimePicker  
    startDate, DateTimePicker endDate)  
{  
    int monthsApart = 12 * (startDate.Value.Year -  
        endDate.Value.Year) +  
        startDate.Value.Month - endDate.Value.Month;  
    int years = monthsApart / 12;  
    return Math.Abs(years);  
}  
public String result_data_string(DateTimePicker  
    startdata, DateTimePicker enddata)  
{  
    String result_data = "";  
    int year = enddata.Value.Year -  
        startdata.Value.Year;  
    int month = monthDifference(startdata, enddata);  
    int years = yearDifference(startdata, enddata);  
  
    DateTime startDate = (DateTime)startdata.Value;  
    DateTime endDate = (DateTime)enddata.Value;  
    TimeSpan ts = endDate.Subtract(startDate);  
    String aaas = ts.ToString();  
    String ddd = ts.Days.ToString();
```

```
int days = ts.Days;
int Week = days / 7;
String monat = "";
int monat_all = Math.Abs(years * 12 - month);
if (year != 0)
{
    if (month > 11)
    {
        result_data = "";
        result_data = years + "წელი";
    }
    if (startdate.Value.Month !=
        enddata.Value.Month)
    {
        result_data = "";
        monat = (enddata.Value.Month -
            startdata.Value.Month).ToString();
        result_data = years + " წელი, " +
            monat_all + " თვე";
    }
    if (startdate.Value.Day !=
        enddata.Value.Day)
    {
        result_data = "";
        String Day = (enddata.Value.Day -
            startdata.Value.Day).ToString();
        result_data = years + " წელი, " +
            monat_all + " თვე, " + Day + " დღე";
    }
}
if (year == 0 & month != 0)
{
    result_data = "";
```



```
monat = (enddata.Value.Month -
        startdata.Value.Month).ToString();
result_data = monat_all + " თვე";
if (startdata.Value.Day !=
    enddata.Value.Day)
{
    result_data = "";
    String Day = (enddata.Value.Day -
        startdata.Value.Day).ToString();
    result_data = monat_all + " თვე, " + Day
        + " დღე";
}
}
if (year == 0 & month == 0 & days != 0)
{
    result_data = "";
    String Day = (enddata.Value.Day -
        startdata.Value.Day).ToString();
    result_data = Day + " დღე";
}
return result_data;
}
}
}
```

2. დიალოგური ფორმის კლასი

```
// --- ლისტინგი_15.2 -----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Windows.Forms;
using System.Threading;

namespace WindowsFormsApplication_IT_Project
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        DataGridViewCellStyle dataGridViewCellStyle_p;
        DataGridViewTextBoxColumn column_period_all;
        DataGridViewTextBoxColumn tasks;
        Class_statistic statics = new Class_statistic();

//დამხმარე კლასის ობიექტის შექმნა
// =====კალენდარული გეგმის შედგენა=====
private void button_scheduler_Click(object sender,
    EventArgs e)
    {
        DateTime startDate =
            (DateTime)dateTimePicker1.Value;
        DateTime endDate =
            (DateTime)dateTimePicker2.Value;
        TimeSpan ts = endDate.Subtract(startDate);
        String aaas = ts.ToString();
        String ddd = ts.Days.ToString();
        int days = ts.Days;
        int Week = days / 7;

        string monat = (dateTimePicker2.Value.Month -
            dateTimePicker1.Value.Month).ToString();
        dataGridView1.Columns.Clear();
```

```
tasks = new DataGridViewTextBoxColumn();
tasks.Width = 100;
tasks.HeaderText = "davaleba";
tasks.Name = "davaleba";
dataGridView1.Columns.Add(tasks);
int month_caunt =
    statics.monthDifference(dateTimePicker1,
        dateTimePicker2);
for (int i = 0; i < month_caunt + 1; i++)
{
    dataGridViewCellStyle_p = new
        DataGridViewCellStyle();
    column_period_all = new
        DataGridViewTextBoxColumn();
    column_period_all.Width = 100;

    dataGridViewCellStyle_p.Format = "N2";

    column_period_all.ReadOnly = true;
    DateTime ad_days =
        dateTimePicker1.Value.AddMonths(i);
    String[] days_data_list =
        statics.token_data(ad_days.ToString());
    String days_data = days_data_list[0];
    int year = ad_days.Year;
    String twe_S =
        Thread.CurrentThread.CurrentCulture.
            DateTimeFormat.MonthNames[ad_days.Month - 1];
    String tarigi = twe_S + "/" + year;
    column_period_all.HeaderText = tarigi;
    column_period_all.Name = tarigi;

    dataGridView1.Columns.Add(column_period_all);
}
}
```

```
// =====გეგმის დამატება=====
private void button_tasks_Click(object sender,
    EventArgs e)
{
    foreach (DataGridViewCell cell in
        dataGridView1.SelectedCells)
    {
        dataGridView1.Rows[cell.RowIndex].
            Cells[cell.ColumnIndex].Style.BackColor =
                System.Drawing.Color.GreenYellow;
    }
}

// =====გეგმის კორექტირება=====
private void button_reset_Click(object sender,
    EventArgs e)
{
    foreach (DataGridViewCell cell in
        dataGridView1.SelectedCells)
    {
        dataGridView1.Rows[cell.RowIndex].
            Cells[cell.ColumnIndex].Style.BackColor =
                System.Drawing.Color.White;
    }
}
```

დავალება:

ორ dateTimePicker (საწისი და საბოლოო თარიღი) - ობიექტიდან არჩეული თარიღების მიხედვით dataGridView ცხრილზე გადანაწილებულად ასახეთ სვეტები წლებით 3 წლიანი პროექტისთვის; კვირებით - 4 კვირიანი პროექტისთვის; დრეებით 20 დღიანი პროექტისთვის.

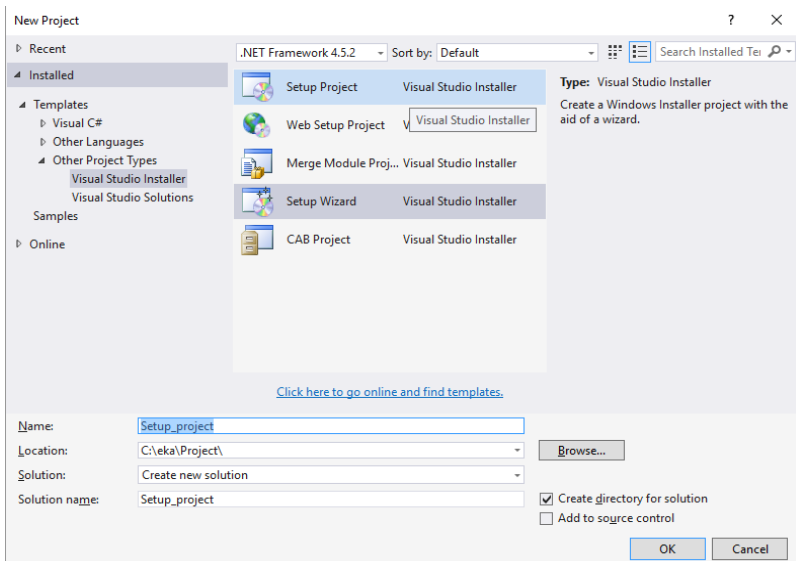
15. ლაბორატორიული სამუშაო N 15

პროგრამული აპლიკაციის დისტრიბუციული ფაილის (საინსტალაციო პაკეტის) შექმნა

მიზანი: Setup და Deployment პროექტში რეალიზებული პროგრამული პაკეტის ინტეგრირება. რეალიზებული პროგრამული პაკეტის გამშვები (exe) ფაილის შექმნის შესწავლა.

ამოცანა: პროგრამული პაკეტის გამშვები ფაილის მომზადება

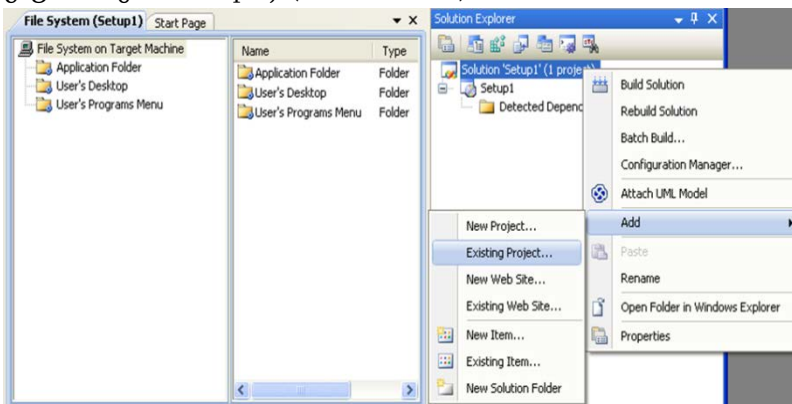
Visual Studio სისტემაში ახალი პროექტის შექმნის მენიუში Other Project Type-Visual Studio Installer პროგრამული პაკეტის გამშვები (exe) ფაილის შექმნისთვის (ნახ.15.1) არის ორი არჩევანი Setup Project და Setup Wizard. მიუხედავად ამისა, მათი ფუნქციონირება მცირედით განსხვავდება.



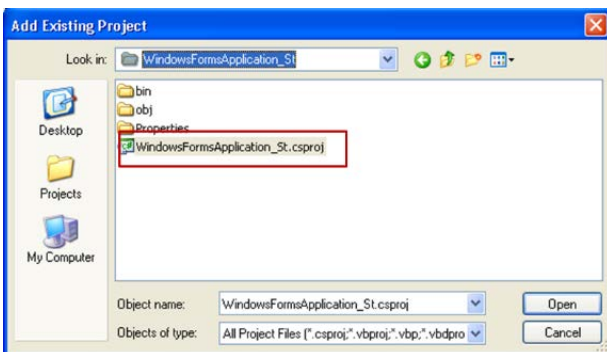
ნახ.15.1

Setup Wizard საშუალებას იძლევა ბიჯური რეჟიმით შეიქმნას Web ან Windows ფორმის საინსტალაციო გარემო და ინტეგრირდეს სისტემისთვის თანმდევი ფაილები. თუმცა, გამწვები ფაილის მომზადების ბირთვის ავტომატიზებულ მექანიზმებს Setup Wizard საშუალება არ შეიცავს.

Setup Project ან Setup Wizard საშუალებებით პროექტის შექმნის შემდეგ, პროექტში უნდა ინტეგრირდეს რეალიზებული პროგრამული პაკეტი ფუნქციით Add – Existing Project (მაუსის მარჯვენა ღილაკი Solution Explorer ფანჯარაში. მიეთითება ფაილი გაფართოებით - .csproj (ნახ. 15.2, 15.3).

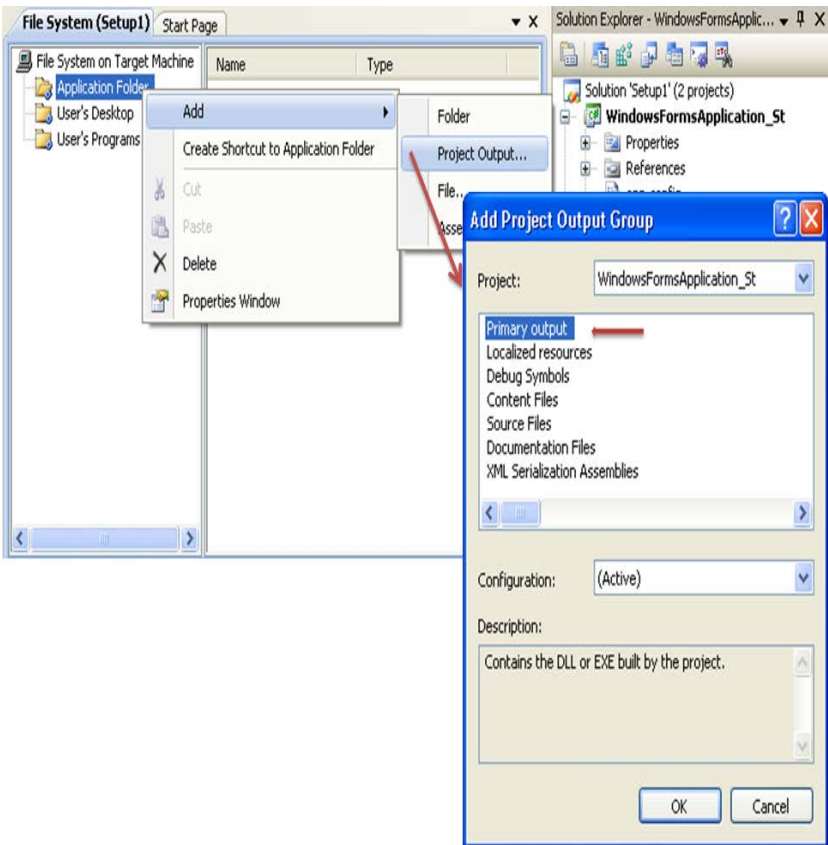


ნახ.15.2

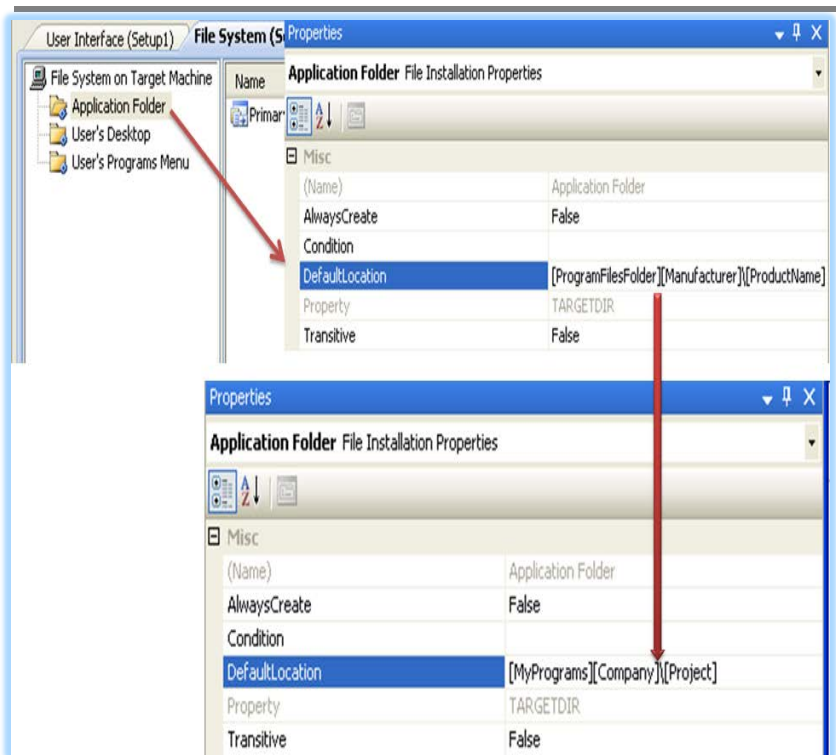


ნახ.15.3

პროექტის ნაწილში File System საქაღალდე Application Folder განკუთვნილია საინსტალაციო პაკეტისთვის. ამ საქაღალდეზე ფუნქციით Add (მაუსის მარჯვენა ღილაკი) იხსნება პროექტის შედეგების დასაპროექტებელი ფანჯარა, სადაც რეკომენდებულია ბაზური შედეგის - Primer output მითითება (ნახ.15.4). Application Folder საქაღალდეს თვისებების ფანჯარაში შესაძლებელია დასაინსტალირებელი პაკეტის სახელის მითითებაც (ნახ.15.5)



ნახ.15.4

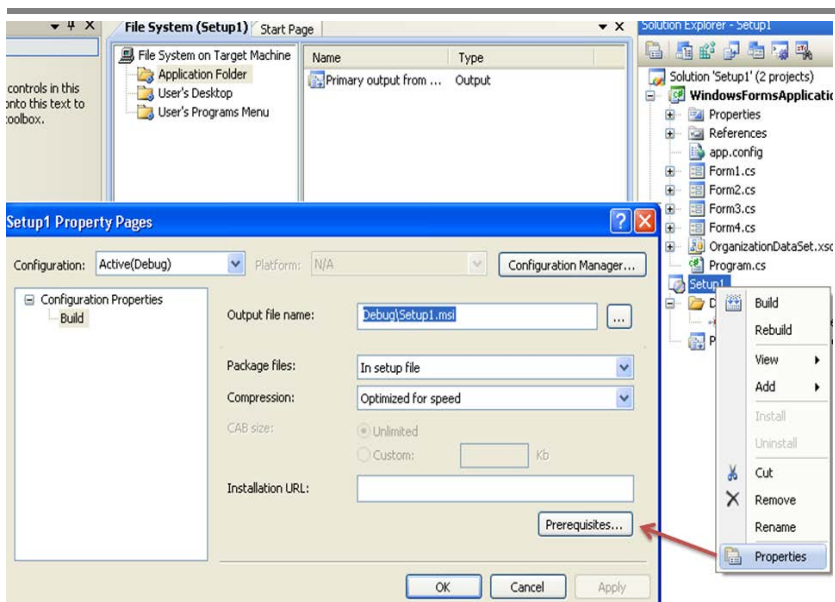


ნახ.15.5

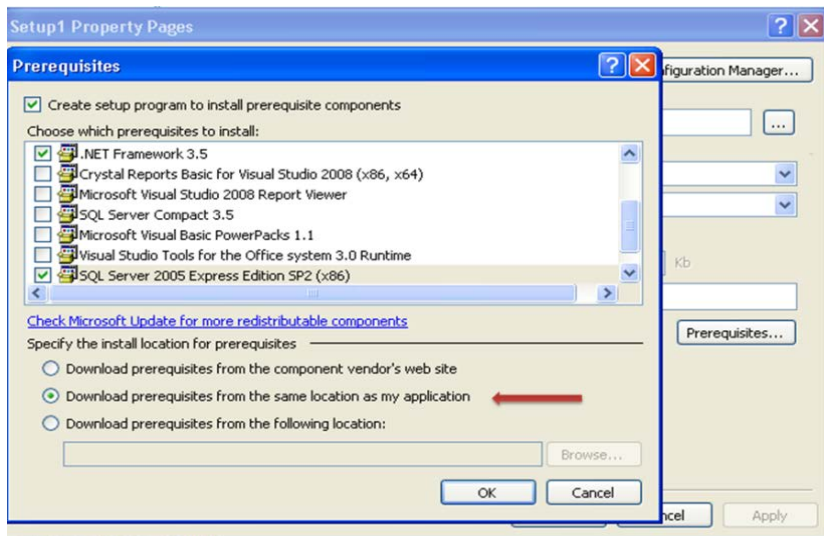
სისტემის ინსტალირების დროს, ხშირად საჭიროა დამხმარე პროგრამების, კომპონენტების ან ბიბლიოთეკების დაყენებაც (მაგ., Windows Installer, .NET Framework, SQL Server Express და სხვ.).

პროექტის Solution Explorer-ის დიალოგურ ფანჯარაში რეალიზებული პროგრამული პაკეტის დამატების შემდეგ ჩნდება ორი პროექტი 1. მიმდინარე ანუ Setup პროექტი და 2. მიერთებული ანუ რაც უნდა დაინსტალირდეს. Setup პროექტზე თავუნას მარჯვენა ღილაკის მეშვეობით ფუნქციაზე Property ღილაკით ხდება თვისებების დიალოგური ფორმის გამოძახება, სადაც ღილაკით Prerequisites იხსნება წინაპირობების მისათითებელი დიალოგური ფანჯარა (ნახ.15.6 და 15.7).

„პროგრამული სისტემების მენეჯმენტი: ლაბორატორიული პრაქტიკუმი“



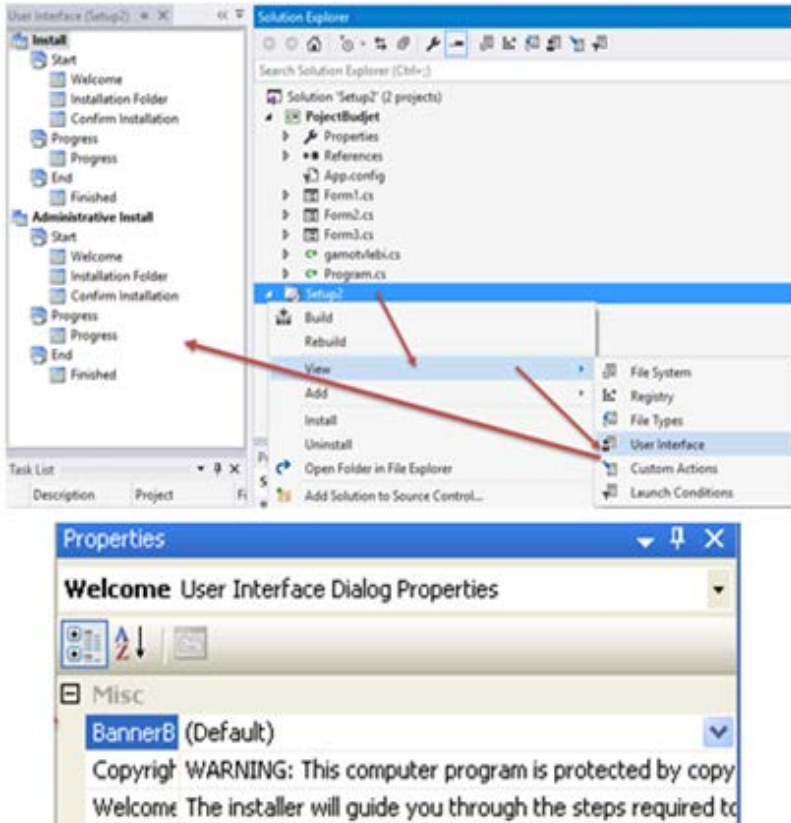
ნახ.15.6



ნახ.15.7

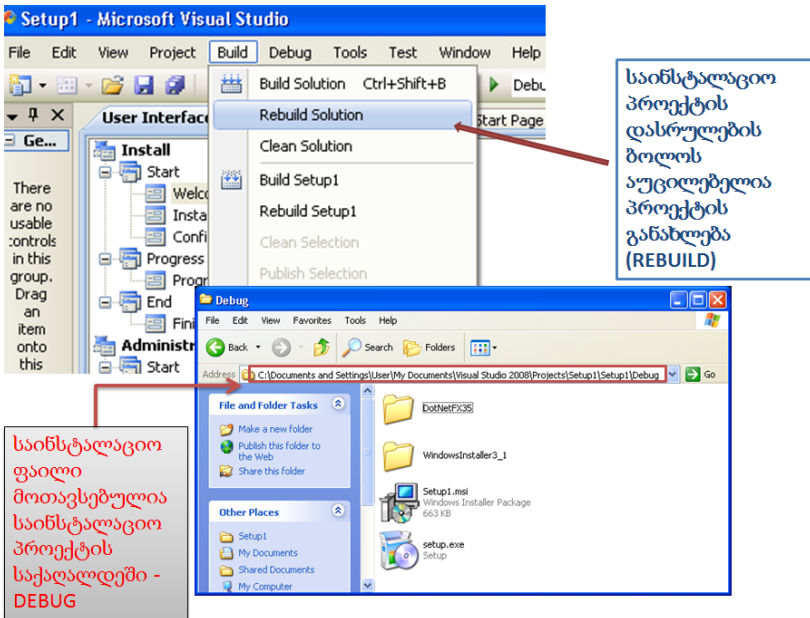
პროექტის ნაწილში File System მოთავსებულია ასევე ორი დამატებითი საქალაქე User’s Desktop (რა განთავსდეს მომხმარებლის სამუშაო მაგიდის საქალაქეში) და User’s Programs Menu (რა განთავსდეს პროგრამული ჩამონათვალის საქალაქეში).

შესაძლებელია ინსტალაციის პროგრამის დიზაინის ცვლილება (ძირითადად, სურათებით/იკონებით გაფორმება) Setup პროექტზე თავუნას მარჯვენა ღილაკის მეშვეობით View-User Interface ფუნქციის გამოძახება (ნახ.15.8).



ნახ.15.8

საბოლოოდ, აუცილებელია Setup პროექტის განახლება ფუნქციით Rebuild (ნახ.15.9). Setup.exe გამშვები ფაილი მოთავსებულია Setup პროექტის საქაღალდის ქვესაქაღალდეში - Debug.



ნახ.15.9

დავალება.

- ლაბორატორიული სამუშაოს ინსტრუქციის და პროგრამული კოდების გამოყენებით დაამუშავეთ აპლიკაციის პროექტი, გამართეთ იგი, რომ იყოს მუშაობისუნარიანი;
- ჩატარეთ ექსპერიმენტი აგებული პროგრამული აპლიკაციის დეხმარებით. საწყისი მონაცემები და საბოლოო შედეგები გაანალიზეთ.

ლიტერატურა:

1. სურგულაძე გ. თურქია ე. (2003). ბიზნესპროცესების მართვის ავტომატიზებული სისტემების დაპროექტება. მონოგრ., ISBN 99940-14-81-1. სტუ, თბილისი. „ტექნიკური უნივერსიტეტი“.
2. თურქია ე. (2010). ბიზნეს-პროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. სტუ. თბილისი. „ტექნიკური უნივერსიტეტი“.
3. ფრანგიშვილი ა., სურგულაძე გ., ვაჭარაძე ი. (2009). ბიზნეს-პროგრამების ექსპერტულ შეფასებებში გადაწყვეტილებათა მიღების მხარდამჭერი მეთოდები და მოდელები. სტუ. მონოგრ., ISBN 978-9941-14-450-9. თბილისი. „ტექნიკური უნივერსიტეტი“.
4. Скопин И. (2004). Основы менеджмента программных проектов. Новосибирский Государственный Университет. INTUIT.
5. Бадави Х., Изуно А., Левики П., Свитинбенк П., Хи Дж., Шварцер Х., Юсуф Л. Создание бизнес-процесса с помощью инструментов Rational и WebSphere. <http://www.intuit.ru/studies/courses/1152/258/info>
6. სურგულაძე გ., ფხაკაძე ც., კვეენაძე ა. (2016). ორგანიზაციული მართვის ბიზნესპროცესების მოდელირება და დაპროექტება. მონოგრ., ISBN 978-9941-0-8259-7. სტუ, თბილისი. „IT კონსალტინგის ცენტრი“.
7. სურგულაძე გ., ქრისტესიაშვილი ხ., სურგულაძე გ. (2015). საწარმოო რესურსების მენეჯმენტის ბიზნესპროცესების მოდელირება და კვლევა. მონოგრ., ISBN 978-9941-20-557-6. სტუ. თბილისი. „ტექნიკური უნივერსიტეტი“.

8. Booch G., Jacobson I., rambaugh J. (1996). Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara.

9. გოგიჩაიშვილი გ., ბოლხი გ., სურგულაძე გ., პეტრიაშვილი ლ. (2013). მართვის ავტომატიზებული სისტემების ობიექტ-ორიენტირებული დაპროექტების და მოდელირების ინსტრუმენტები (MsVisio, WinPepsy, PetNet, CPN). სახელმძღვ., ISBN 99940-56-77-8. სტუ.. თბილისი. „ტექნიკური უნივერსიტეტი“.

10. სურგულაძე გ., გულიტაშვილი მ., კვიციანი ნ. Web-სისტემების ტესტირება, ვალიდაცია და ვერიფიკაცია. (2015). მონოგრ. ISBN 9789941-0-7682-4. სტუ. „IT-კონსალტინგის ცენტრი“. თბილისი.

11. Carnegie Mellon Software Engineering Institute.
<http://www.sei.cmu.edu/>

12. Software quality assurance. https://en.wikipedia.org/wiki/Software_quality_assurance



სტუ-ს „IT-კონსალტინგის ცენტრი“

(თბილისი, მ.კოსტავას 77)