გია სურგულაძე

კორპორაციული მენეჯმენტის სისტემების Windows დანართის დეველოპმენტი: WPF ტექნოლოგია

„IT-კონსალტინგის ცენტრი“

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე

კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: WPF ტექნოლოგია

(ლაბორატორიული პრაქტიკუმი, ნაწ.1)

დამტკიცებულია:
სტუ-ის სარედაქციო-
საგამომცემლო
საბჭოს მიერ

თბილისი
2014

უაკ 004.5

განიხილება კორპორაციული მენეჯმენტის პროცესების ავტომატიზაციის საფუძვლები ახალი ინფორმაციული ტქნოლოგიების ბაზაზე. კერძოდ მოცემულია MsVisual Studio.NET Framework 4.0/5 ინტეგრირებულ გარემოში ჰიბრიდული კომპიუტერული სისტემების (Windows- და Web-აპლიკაციების) დაპროგრამების ინსტრუმენტული საშუალებები WPF (Windows Presentation Foundation) ტექნოლოგიის ბაზაზე. იგი ეფუძნება XAML (სისტემის დიზაინის ნაწილი) და C# (სიტემის ლოგიკური ნაწილი) ენების კომპლექსურ გამოყენებას. წარმოდგენილია ლაბორატორიული პრაკტიკუმის ამოცანები და მეთოდური ინსტრუქციები ასეთი სისტემების კომპონენტების დასაპროგრამებლად.

დამხმარე სახელმძღვანელო ლაბორატორიული პრაქტიკუმის სახით განკუთვნილია ინფორმატიკისა და მართვის საინფორმაციო სისტემების სპეციალობის ბაკალავრიატის მაღალი კურსის სტუდენტებისა და მაგისტრანტებისთვის.

რეცენზენტები:

- სრ. პროფ. ე. თურქია
- სრ. პროფ. გ. ღვინეფაძე

პროფ. გ. სურგულაძის რედაქციით

ISBN 978-9941- 0-7103-4 (ყველა ნაწილის)
ISBN 978-9941-0-7104-1 (პირველი ნაწილის)

შინაარსი

I თავი. თეორიული ნაწილი: ჰიბრიდული აპლიკაციების აგების WPF-ტექნოლოგია **5**
1.1. Windows Presentation Foundation ტექნოლოგია ---------- **5**
1.1.1. WPF ტექნოლოგიის შესაძლებლობები ---------- **7**
1.1.2. WPF -ის შესაძლებლობები დიზაინერებისთვის ---------- **7**
1.1.3. WPF -ის შესაძლებლობები C# - დეველოპერებისთვის ---------- **9**
1.2. WPF-ის არქიტექტურა და კლასები ---------- **9**
1.3. მომხმარებლის ინტერფეისის დაკომპლექტება ---------- **14**
1.4. მართვის ელემენტები და შიგთავსები ---------- **17**
1.5. ტექსტური მართვის ელემენტები ---------- **23**
1.6. სიების მართვის ელემენტები ---------- **23**
1.7. სპეციალიზებული მართვის ელემენტები ---------- **24**
1.8. ბრძანებები ---------- **25**
1.9. რესურსები ---------- **27**
1.10. სტილები ---------- **31**
1.11. შაბლონები ---------- **33**
1.11.1. მართვის ელემენტთა შაბლონები ---------- **35**
1.11.2. მონაცემთა შაბლონები ---------- **38**
1.12. XAML ენის საფუძვლები ---------- **41**
1.13. WPF- აპლიკაციების შექმნა ---------- **50**
1.14. WPF აპლიკაცია მონაცემთა ბაზებით ---------- **51**
1.15. მართვის ელემენტების მიბმა მონაცემთა წყაროსთან ---------- **65**
1.16. მარშრუტიზირებადი მოვლენები ---------- **68**

II თავი. პრაქტიკული ნაწილი: WPF-ტექნოლოგიის ვიზუალური ელემენტები და მათი გამოყენება აპლიკაციების ასაგებად . . . **73**
2.1. Windows Presentation Foundation ტექნოლოგის სამუშაო გარემოს გაცნობა. WPF-ის ინსტრუმენტების პანელი (ლაბ.1) ---------- **73**
2.2. მულტიმედიალური შესაძლებლობები: მარტივი ღილაკის დაპროგრამება ანიმაციის ელემენტებით (ლაბ.2) ---------- **75**
2.3. WPF-ის ფანჯრების მართვის ელემენტები (ლაბ.3) ---------- **85**
2.4. მრავალფანჯრიანი პროექტი WPF-ში (ლაბ.4) ---------- **92**

2.5. WPF-ის მომხმარებელთა მართვის ელემენტები (ლაბ.5) ---------- 99
2.6. WPF-ში Web-გვერდების აპლიკაციების შექმნა (ლაბ.6) ---------- 104
2.7. WPF-ში ფუნჯის სახეები და ფერების შერჩევა (ლაბ.7) ---------- 117
2.8. WPF-ის მართვის ელემენტები: Menu, ToolBar, TabControl და ToolTip (ლაბ.8) ---------- 136
2.9. WPF-ის მართვის ელემენტები: Thumb, Border და Popup (ლაბ.9) — 144
2.10. WPF-ის მართვის ელემენტები: ScrollViewer, Viewbox და StackPanel (ლაბ.10) ---------- 151
2.11. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: Canvas (ლაბ.11) ---------- 158
2.12. ინტერფეისული ელემენტების განლაგების მენეჯერი: StackPanel და DockPanel (ლაბ.12) ---------- 164
2.13. ინტერფეისული ელემენტების განლაგების მენეჯერი: WrapPanel и UniformGrid (ლაბ.13) ---------- 170
2.14. ინტერფეისული ელემენტების განლაგების მენეჯერი: Grid პანელი (ლაბ.14) ---------- 179
2.15. WPF-ის საკუთარი განლაგების მენეჯერის დამუშავება: MyPanel (ლაბ.15) ---------- 190
ლიტერატურა ---------- 200
ავტორის გვერდი ---------- 201

I თავი. თეორიული ნაწილი: ჰიბრიდული აპლიკაციების აგების WPF-ტექნოლოგია

1.1. Windows Presentation Foundation ტექნოლოგია

აპლიკაციების (დანართების) ორი ნაირსახეობაა ცნობილი: ვინდოუს სისტემები, რომელთაც ასევე სამაგიდო (Desktop) აპლიკაციებს უწოდებენ და ვებ-აპლიკაციები, რომელთა გამოყენებაც ინტერნეტ ბრაუზერებიდანაა შესაძლებელი [1- 3].

ეს დანართები იქმნება .NET Framework -ის ორი სხვადასხვა პაკეტით. პირველი - Windows Forms კომპონენტებით და მეორე ASP.NET -ის საშუალებით. ორივეს აქვს თავისი უპირატესობები და ნაკლოვანებანი. კერძოდ, სამაგიდო დანართები ძალზე მოქნილი და რეაქციულია, ხოლო Web-დანართები ინტერნეტის საშუალებით იძლევა დისტანციური წვდომის საშუალებას ერთდროულად მრავალი მომხმარებლისთვის.

მაგრამ თანამედროვე კომპიუტერული ტექნოლოგიების სამყაროში ამ ორი სახის აპლიკაციებს შორის საზღვრები სულ უფრო და უფრო იშლება [4].

Web-სამსახურების და WCF (Windows Communication Foundation) სერვის-ორიენტირებული არქიტექტურის აგების საშუალებების გაჩენამ განაპირობა სამაგიდო- და ვებ-დანართების ფუნქციონირების შესაძლებლობა ერთიან განაწილებულ გარემოში, სადაც მონაცემთა გაცვლა ხორციელდება როგორც ლოკალურ, ასევე გლობალურ ქსელებში. 1.1-ა ნახაზზე ნაჩვენებია ეს გამაერთიანებელი პროცესი.

ნახ.1.1-ა

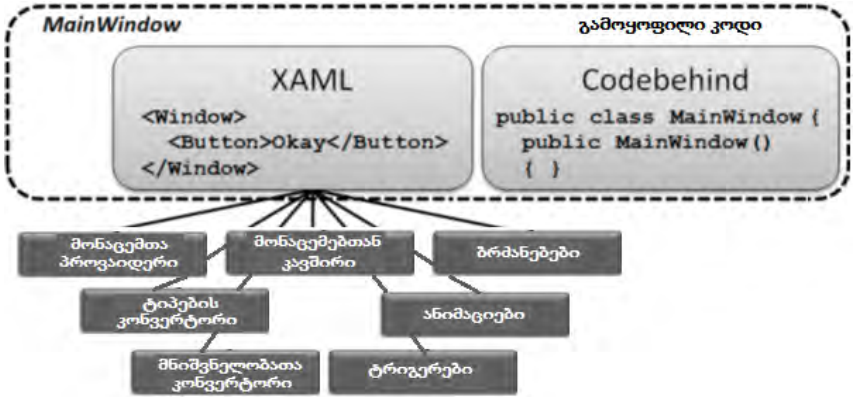
WPF – Windows Presentation Foundation არის ერთ-ერთი ასეთი გამაერთიანებელი ტექ–ნოლოგია. იგი საშუალებას იძლევა აიგოს ისეთი დანართი, რომელშიც გამორიცხულია დაპირისპირება სამაგიდო აპლიკაციას და ინტერნეტს შორის.

WPF-დანართს, როგორც ეს ნაჩვენები იქნება ქვემოთ, შეუძლია ფუნქციონირება როგორც სამაგიდო აპლიკაციას, ასევე როგორც ვებ-აპლიკაციას ბრაუზერის შიგნით. არსებობს ასევე WPF-ის შეზღუდული ვერსია, სახელით Silverlight, რომლითაც შესაძლებელია ვებ-დანართში დინამიკური მდგენელის დამატება.

წიგნში განიხილება დაპროგრამების საფუძვლები WPF ტექნოლოგიით, WPF-ის საბაზო სტრუქტურა, მისი ძირითადი ცნებები და შედგენილობა, პროექტების აგება Ms Visual Studio .NET Framework 4.0/4.5 გარემოში WPF/C# -ის საფუძველზე [1,5,8].

პრაქტიკული სამუშაოების შესასრულებლად Windows Presentation Foundation სისტემაში საჭიროა შემდეგი რესურსები:

- **ოპერაციული სისტემა** - .Net Framework-ის 3.0 (და SP1) ვერსიას აქვს Windows XP, Windows Vista და Windows Server 2003–ის მხარდაჭერა. .Net Framework 4.0/4.5-ის გამოჩენის შემდეგ იგი ბევრად სრულყოფილი გახდა. ახალი ფუნქციონალობის გამოსაყენებლად სასურველია კონფიგურაციის გაუმჯობესება:
  - Windows 7 ან Windows 8 (ან Windows Server 2008 R2);
  - გრაფიკული კარტა DirectX-ის მე-9 ვერსიისთვის;
- **ინტეგრირებული სამუშაო გარემო:** .Net Framework 4.0/4.5. 1.1-ბ ნახაზზე ნაჩვენებია ორი ძირითადი პროგრამული ტანდემი (XAML-თავისი მეთოდებით და კოდი).



ნახ.1.1-ბ

1.1.1. WPF ტექნოლოგიის შესაძლებლობები

WPF (ადრე ცნობილი იყო სახელით Avalon) არის ტექნოლოგია, რომელიც საშუალებას იძლევა დაიწეროს პლატფორმაზე დამოუკიდებელი აპლიკაციები, დიზაინისა და ფუნქციონალური შესაძლებლობების ცხადი დაყოფით. იგი ეფუძნება ადრე არსებულ ისეთ ტექნოლოგიათა გაფართოებულ ცნებებს და კლასებს, როგორიცაა Windows Forms, ASP.NET, XML, GDI+ და ა.შ. ასეთი მსგავსება კარგად ჩანს ვებ-დანართის აწყობისას .NET Framework გარემოში [1,2].

გარდა ამისა WPF-ში მრავლადაა ახალი ფუნქციონალური საშუალებები პროგრამისტებისა და მომხმარებლებისთვის, რაც მისი, როგორც ახალი ტექნოლოგიის განხილვის უფლებას იძლევა. მაგალითად, Silverlight ტექნოლოგიას შეუძლია პროგრამები პირდაპირ ბრაუზერში შეასრულოს. ამ პლაგინის ინსტალაცია აუცილებელია. არაა დიდი მოცულობის, კომპაქტურია. ისე როგორც WPF-ში, აქაც გამოიყენება XAML და C#. მაგრამ WPF პროგრამები უშუალოდ ბრაუზერში ვერ გაიშვება.

WPF მუშაობს ვექტორულ გრაფიკაზე ბაზირებულად და არა პიქსელებზე. მართვის ელემენტები, გრაფიკები, აგრეთვე ნახაზები არ იხაზება პიქსელებით, არამედ აღიწერება მრუდებით და წრფეებით. ამის გამო აღარაა ძლიერი დამოკიდებულება მონიტორებისგან.

1.1.2. WPF -ის შესაძლებლობები დიზაინერებისთვის

მომხმარებელთა ინტერფეისების დასაპროექტებლად WPF-ში გამოიყენება დანართების ფორმატირების გაფართოებული ენა XAML (Extensible Application Markup Language) [3]. იგი მსგავსია ASP.NET-ის ფორმატირების ენის, რომელიც იყენებს XML-სინტაკსს და შეუძლია მომხმარებლის ინტერფეისს დაუმატოს მართვის ელემენტები დეკლარაციული, იერარქიული სახით. ამასთანავე მას შეუძლია მერთვის ელემენტების შექმნა, რომლებიც შეიცავს სხვა მართვის ელემენტებს, რაც მნიშვნელოვანია როგორც ინტერფეისის კონსტრუირებისთვის, ისე დანართის ფუნქციონალური შესაძლებლობებისთვის.

XAML-ენა ბევრად მძლავრია, ვიდრე ASP.NET და არაა შეზღუდული HTML ენის შესაძლებლობებით მონაცემთა ვიზუალიზაციის დროს ინტერფეისებში [4]. ის დამუშავდა სპეციალურად დღეისათვის არსებული მძლავრი გრაფიკული კარტების გათვალისწინებით DirectX-ის საფუძველზე [5]. მაგალითად:

- ვექტორული გრაფიკა და კოორდინატები მცოცავი მძიმით - უზრუნველყოფს ობიექტების მასშტაბირებას, ბრუნვას და ტრანსფორმირებას ხარისხის დაუკარგავად;
- ორ- და სამგანზომილებიანი შესაძლებლობები ვიზუალიზაციის სრულყოფისთვის;
- შრიფტების დამუშავების და ვიზუალიზაციის სრულყოფილი საშუალებები;
- გრადიენტული, უწყვეტი და ტექსტური შევსება გამჭვირვალობის არააუცილებელი ეფექტებით მომხმარებლის ინტერფეისის ობიექტებისთვის;
- ანიმაციის კადრების დაშლის ტექნოლოგია, რომელიც გამოიყენება ყველა სიტუაციაში, მათ შორის მომხმარებლის მიერ გენერირებულ მოვლენებში, მაგალითად, ღილაკის დაჭერის იმიტაცია;
- მრავალჯერადი მოხმარების რესურსები, რომელთა გამოყენება შეიძლება მართვის ელემენტების დინამიკური სტილიზაციისთვის.

დიზაინერებისთვის განსაკუთრებული ინტერესი აქვს მაიკროსიფთის ინსტრუმენტს Expression Blend (EB), რომლითაც შესაძლებელია XAML-ფაილების შექმნა და შემდგომ მათი გადატანა დანართებში. EB-ში შესაძლებელია პროექტების შექმნა და რედაქტირება ისევე, როგორც VisualStudio-ს Solution Explorer-ში. ამგვარად, Expression Blend სასურველი, მაგრამ არააუცილებელი კომპონენტია WPF-დანართების ასაგებად. ის VS სტანდარტულ ვერსიას არ მოჰყვება, მაგრამ შეიძლება მისი დემო-ვერსიის ჩამოტვირთვა:

microsoft.com/expression/products/overview.aspx?key=blend

1.1.3. WPF -ის შესაძლებლობები C# - დეველოპერებისთვის

აპლიკაციების დეველოპერები პროექტებს ქმნიან VC (Visual Studio) და VCE (Visual C# Express - არის VC-ს შეზღუდული უფასო ვერსია) გარემოში სამუშაოდ.

WPF-ში გამოიყენება გამყოფილი კოდის მოდელი, როგორც ASP.NET-ში. მაგალითად, Button მართვის ელემნტისთვის მოვლენის დამმუშავებლის (პროგრამის) მიმაგრება შეიძლება მისი XML ელემენტისთვის Click ატრიბუტის დამატებით. ეს ატრიბუტი მიუთითებს მოვლენის დამმუშავებლის სახელზე შესაბამისი XAML-გვერდის გამოყოფილი კოდის ფაილში, რომელიც შეიძლება დაწერილი იყოს C#-ზე.

WPF დანართებში მართვის ელემენტების მანიპულირება შეიძლება Windows Forms მსგავსად, რომლის დროსაც გამოიყენება დაპროგრამების ხერხები მომხმარებლის ინტერფეისების ასაგებად. გამოყოფილი კოდის საშუალებით შეიძლება შეიქმნას მართვის ელემენტის ეგზემპლარი, დაყენდეს თვისებები, მიებას მოვლენათა დამმუშავებლები და დაემატოს ეს მართვის ელემენტი ფორმაზე. ამ პროცესს შეუძლია მთლიანად გამორიცხოს XAML-კოდი.

ასეთ შემთხვევაში გამოყოფილი კოდის ზომა იქნება გაცილებით დიდი, ვიდრე შესაბამისი დეკლარაციული XAML-ის კოდი და ამასთანავე იკარგება დიზაინსა და ფუნქციონალურ შესაძლებლობებს შორის ცხადი საზღვარი, რაც არასახარბიელოა. ამიტომ სასურველია მომხმარებლის ინტერფეისზე მართვის ელემენტების განლაგება განხორციელდეს XAML-ით.

1.2. WPF-ის არქიტექტურა და კლასები

Windows Presentation Foundation (WPF) ეფუძნება ვიზუალიზაციის ვექტორულ სისტემას და ორიენტირებულია კლიენტების ვებ-აპლიკაციების (დანართების) დამუშავებაზე Microsoft.NET პლატფორმისთვის. ამ თავში განიხილება WPF-ის არქიტექტურა, ძირითად კლასთა იერარქია, მომხმარებლის ინტერფეისის აგების საკითხები, კონტროლის ელემენტების გამოყენების თავისებურებანი, შედგენილობის მართვის ძირითადი

ელემენტები, მათი თვისებები და დეკლარაციული აღწერა. წარმოდგენილ ინფორმაციას - რესურსების, სტილის და შაბლონების შესახებ აქვს შესავალი ხასიათი WPF-ის საკმაოდ ეფექტურ და მრავალფეროვან კონსტრუქციებისთვის.

Windows Presentation Foundation (WPF) - ესაა კლიენტების Windows-დანართების აგების სისტემა Microsoft.NET ტექნოლოგიისთვის ვიზუალური შესაძლებლობებით. WPF-ით შეიძლება აიგოს ფართო სპექტრი როგორც ავტონომიური დანართების, ასევე ვებ-ბრაუზერში განთავსებული აპლიკაციებისა.

WPF-ის საფუძველია ვიზუალიზაციის ვექტორული სისტემა, რომელიც გათვლილია თანამედროვე გრაფიკულ საშუალებებზე. ვიზუალური ინტერფეისის ასაგებად გამოიყენება XAML ენა, მართვის ელემენტები, მონაცემებთან მიბმა, მაკეტები, 2- და 3-განზომილებიანი გრაფიკა, ანიმაცია, სტილები და შაბლონები, დოკუმენტები და ტექსტები, მულტიმედია და გაფორმებები.

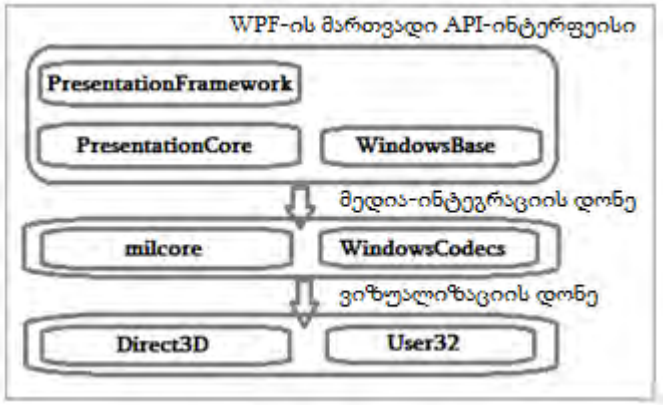
WPF-ის გრაფიკულ ტექნოლოგიას საფუძვლად უდევს DirectX. განსხვავებით Windows Forms-სგან, სადაც გამოიყენება GDI/GDI+. WPF-ის მწარმოებლურობა უფრო მაღალია, ვიდრე GDI+-ის, გრაფიკის აპარატურული დამაჩქარებლის გამოყენების გამო DirectX -ში.

WPF უზრუნველყოფს მომხმარებლის მაღალი დონის ინტერფეისს და იძლევა შემდეგ შესაძლებლობებს:

- ვებ-ისმაგვარი მოდელის აწყობა, რომელიც უზრუნველყოფს მართვის ელემენტების განლაგებას და მოწესრიგებას შინაარსის მიხედვით;
- ხატვის მრავალფუნქციურ მოდელს გრაფიკული კომპონენტების საფუძველზე (საბაზო ფორმები, ტექსტური ბლოკები, გრაფიკული ინგრედიენტები);
- მოდელი ფორმატირებული ტექსტით, რომელიც უზრუნველყოფს ფორმატირებული სტილიზებული ტექსტის ასახვას მომხმარებლის ინტერფეისის ნებისმიერ ნაწილში, ტექსტის კომბინირებას სიებთან, ნახატებთან და სხვა ინტერფეისულ ელემენტებთან;

- ანიმაციის მოცემა დეკლარაციული დესკრიპტორებით;
- აუდიო-ვიზუალური გარემოს მხარდაჭერა ნებისმიერი აუდიო- და ვიდეოფაილების შესასრულებლად;
- სტილები და შაბლონები, რომლებიც უზრუნველყოფს ფორმატირების და (მართვის ელემენტების) ვიზუალიზების მართვის სტანდარტიზებას, აგრეთვე ამ შედეგების მრავალჯერ გამოყენება პროექტის სხვადასხვა ადგილას;
- ბრძანებები, რომლებითაც ისინი განისაზღვრება ერთ ადგილას, მაგრამ მრავალჯერადად უკავშირდება დანართის სხვა ელემენტებს;
- მომხმარებლის დეკლარაციული ინტერფეისი, რომელიც ფანჯრების ან გვერდების შინაარსს აღწერს XAML ენის საშუალებით.

WPF არქიტექტურის ძირითადი კომპონენტები მოცემულია 1.2-ა ნახაზზე.

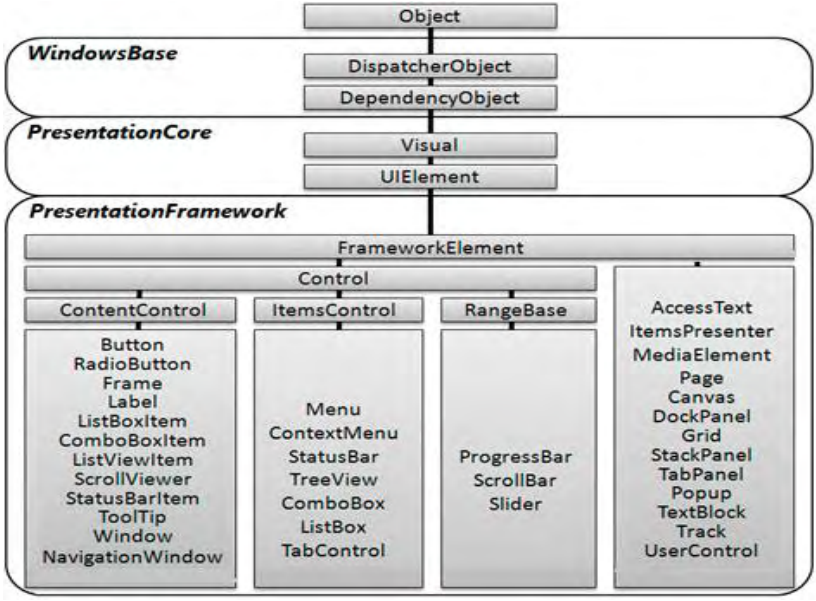


ნახ.1.2-ა. WPF –ის არქიტექტურა

- PresentationFramework კომპონენტი შეიცავს WPF–ის ზედა დონის ტიპებს, როგორიცაა ფანჯრების, პანელების და სხვა ელემენტების წარმოდგენა;
- PresentationCore შეიცავს საბაზო ტიპებს, როგორიცაა UIElement და Visual, რომელთაგანაც იწარმოება მართვის ყველა ფორმა და ელემენტები;

- WindowsBase შეიცავს განსხვავებულ ტიპებს, რომელთა გამოყენება შეიძლება WPF–ის გარეთ. მაგალითად, DispatchObject და DependencyObject კომპონენტები;
- milcore კომპონენტი არის WPF ვიზუალიზაციის ბირთვი;
- WindowsCodecs ქვედა დონის API-ინტერფეისია გამოსახულებათა აგების მხარდასაჭერად;
- Direct 3D არის ასევე ქვედა დონის API-ინტერფეისი, რომლითაც ხორციელდება WPF-ის გრაფიკის ვიზუალიზაცია;
- User32 გამოიყენება იმის დასადგენად, თუ რომელ პროგრამას ეკრანის რომელი უბანი აქვს მიცემული.

1.2-ბ ნახაზზე ნაჩვენებია WPF-ის მართვის ელემენტები კლასთა იერარქიის მიხედვით [8].



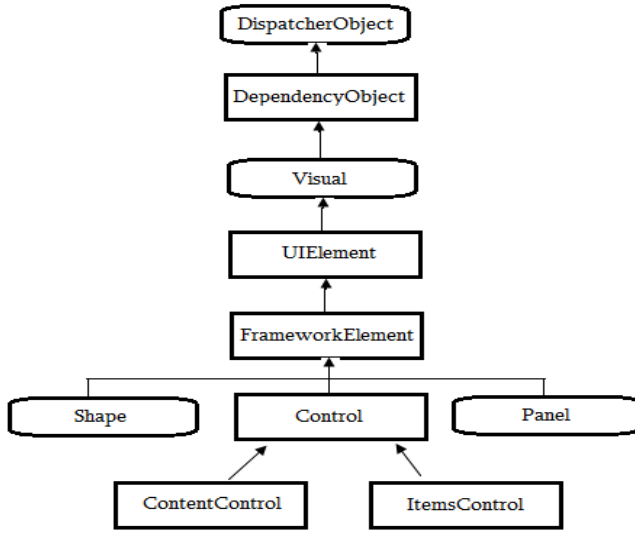
ნახ.1.2-ბ

WPF-ში მართვის ელემენტის საფუძველი არის ყოველთვის FrameworkElement კლასი, რომელიც წარმოიდგინება როგორც ყველაზე საფუძვლიანი საბაზი კლასი.

WPF–ის არქიტექტურა განსაზღვრავს ძირითად სახელსივრცეებს კლასთა იერარქიისთვის. მართვის ელემენტების საბაზო ერთობლიობა განსაზღვრავს სისტემის კლასთა საკვანძო იერარქიებს. 1.3 ნახაზზე აბსტრაქტული კლასები ნაჩვენებია ოვალებით, ხოლო კონკრეტული კლასები – მართკუთხედებით.

ობიექტთა უმრავლეოსობა WPF–ში იწარმოება DispatcherObject აბსტრაქტული კლასიდან. WPF ეფუძნება შეტყობინებათა გაცვლის სისტემას, რომლებიც მომხმარებლის ინტერფეისისთვის ფორმირდება ერთ ნაკადში, რომელიც იმართება და კონტროლდება დისპეტჩერის მიერ.

DispatcherObject კლასი უზრუნველყოფს დანართის ინტერფეისის ყოველი ელემენტისთვის ნაკადში შესრულების შემოწმებას და წვდომას დისპეტჩერისკენ.



ნახ.1.3. WPF ფუნდამენტური კლასები:
აბსტრაქტული (მრგვალკუთხა) და კონკრეტული (მართკუთხა)

WPF კლასები ღებულობს მხარდაჭერას დამოკიდებულების თვისებებისგან, რომელიც წარმოიშვება DependencyObject კლასისგან.

Visual კლასი ერთეულოვანი ობიექტია, რომელიც აინკაფსულირებს ხატვის ინსტრუქციებს და დეტალებს, აგრეთვე საბაზო ფუნქციებს. WPF–ის ინტერფეისული ელემენტები წარმოქმნილ უნდა იქნას Visual კლასისგან.

მომხმარებელთა ყველა მართვის ელემენტი შთამომავალია UIElement ან FrameworkElement კლასების.

UIElement კლასი უზრუნველყოფს ისეთ ფუნქციონალობას, როგორიცაა დაკომპლექტება, შეტანა, ფოკუსი და მოვლენები.

FrameworkElement-კლასი UIElement-ის ფუნქციონალობას უმატებს ველების განსაზღვრას, გასწორებას, მონაცემთა დაკავშირების მხარდაჭერას, ანიმაციას და სტილებს.

Shape კლასი საბაზოა ისეთი გეომეტრიული ფიგურების ასაგებად, როგორიცაა მართკუთხედი, ელიფსი, მრავალკუთხედი, წრფე და გზა.

Control კლასი განსაზღვრავს მართვის ელემენტებს, რომელთაც შეუძლია მომხმარებელთან ურთიერთობა. ესაა ღილაკები, სიები, ტექსტური ელემენტები.

ContentControl და ItemsControl კლასები საბაზოა მართვის ელემენტებისთვის, რომელთაც შეიძლება ჰქონდეს ერთადერთი მნიშვნელობა ან კოლექცია მნიშვნელობებისა, შესაბამისად.

Panel კლასი საბაზოა ყველა კონტეინერული ელემენტთა შემადგენლობისთვის, რომლებიც შეიცავს ერთ ან მეტ შვილობილ ელემენტს.

1.3. ინტერფეისის დაკომპლექტება

აპლიკაციის მომხმარებლის ინტერფეისის დაპროექტების დროს აუცილებელია ფანჯარაში (ფორმაზე) ან გვერდზე საჭირო მართვის ელემენტების ფორმირება და შესაბამისი თვისებების განსაზღვრა, ანუ შინაარსის ორგანიზების ჩატარება. ამ პროცესს უწოდებენ დაკომპლექტებას (შედგენას).

WPF-ში დაკომპლექტება ხორციელდება სხვადასხვა კონტეინერით. ყოველ მათგანს თავისი საკუთარი

დაკომპლექტების ლოგიკა აქვს. ზოგი ალაგებს ელემენტებს მიმდევრობით სტრიქონში, ზოგი ალაგებს მათ უხილავი უჯრების ბადეში.

ფანჯარას და გვერდს WPF-ში შეუძლია შეიცავდეს მხოლოდ ერთ ელემენტს - კონტეინერს. კონტეინერში შეიძლება განთავსდეს მომხმარებლის ინტერფეისის განსხვავებული ელემენტები და სხვა კონტეინერები. WPF-ში განლაგება განისაზღვრება გამოყენებული კონტეინერის ტიპით. ეს კონტეინერებია; პანელები, წარომებული System.Windows.Controls.Panel აბსტრაქტული კლასიდან.

აპლიკაციებში გამოიყენება შემდეგი კლასები:

- Grid და UniformGrid – განალაგებს ელემენტებს უხილავი ცხრილის სტრიქონებსა და სვეტებში, შესაბამისად;
- StackPanel – განალაგებს ელემენტებს ჰორიზონტალურ ან ვერტიკალურ სვეტში.
- WrapPanel – განალაგებს ელემენტებს ხელმისაწვდომ სივრცეში, ერთ სტრიქონად ან სვეტად;
- DockPanel – განალაგებს ელემენტებს ერთ-ერთი სასაზღვრო გვერდის შეფარდებით;
- Frame – ანალოგიურია StackPanel-ის, მაგრამ უფრო მოსახერხებელია გვერდების გადასასვლელბის ორგანიზებისთვის.

Grid არის WPF-ის ყველაზე მძლავრი კონტეინერი. ის, რასაც სხვა კონტეინერები ასრულებს ცალკ-ცალკე, შეიძლება Grid-ში შესრულდეს. იგი იდეალური ინსტრუმენტია ფანჯრის (გვერდის) დასაყოფად შედარებით მცირე ზომის არეებად, რომელთა მართვა განხორციელდება სხვა პანელებით. Grid ანაწილებს ელემენტებს უხილავი ბადის სტრიქონებსა და სვეტებში. ბადის ერთ უჯრაში მიზანშეწონილია ერთი ელემენტის მოთავსება, რომელიც, საჭიროების შემთხვევაში, თვითონ შეიძლება იყოს სხვა კონტეინერი, რომელშიც განლაგდება საკუთარი მართვის ელემენტთა ჯგუფი.

StackPanel - ერთ-ერთი უმარტივესი კონტეინერია. იგი ალაგებს თავის შვილობილ ელემენტებს ერთ სტრიქონში ან სვეტში.

UniformGrid კონტეინერი, Grid-ისგან განსხვავებით, მოითხოვს მხოლოდ სტრიქონების და სვეტების რაოდენობის მითითებას, და აფორმირებს ერთი ზომის უჯრებს, რომელთაც დაკავებული აქვს ფანჯრის (გვერდის) ან ჩადგმული კონტეინერის ელემენტის მთელი ხელმისაწვდომი არე.

WrapPanel აწესრიგებს ელემენტების განლაგებას პანელზე Orientation თვისების შესაბამისად, ჰორიზონტალურად (Horizontal) ან ვერტიკალურად (Vertical). სტრიქონის ან სვეტის შევსების შემდეგ მომდევნო ელემენტი გადადის ახალ სტრიქონზე ან სვეტზე.

DockPanel უახლოვებს მართვის ელემენტებს მის რომელიმე მხარეს, Dock თვისების შესაბამისად, რომელიც იქნება Left, Right, Top ან Bottom. ელემენტის სიმაღლე განისაზღვრება MaxHeight პარამეტრით.

Frame მართვის ელემენტია შიგთავსისთვის, რომელიც იძლევა შესაძლებლობას შიგთავსზე ან მის ასახვაზე გადასასვლელად. Frame შეიძლება მოთავსდეს სხვა შიგთავსის შიგნით, როგორც სხვა ელემენტები. შიგთავსი შეიძლება იყოს .NET Framework-ის და HTML-ფაილების ნებისმიერი ტიპი. ჩვეულებისამებრ, Frame გამოიყენება შიგთავსის ჩასაწყობად, რომელიც განსაზღვრავს გადასასვლელებს გვერდებზე.

დაკომპლექტების თვისებები განისაზღვრება კონტეინერით, მაგრამ შვილობილი ელემენტებიც ახდენს მასზე გალენას. დაკომპლექტების პანელები მუშაობს შვილობილ ელემენტებთან შეთანხმებით, შემდეგი თვისებების საუძველზე:

- HorizontalAlignment და VerticalAlignment - განსაზღვრავს, თუ როგორ პოზიციონირებს შვილობილი ელემენტი კომპლექტის შიგნით, როცა არსებობს დამატებითი ჰორიზონტალური/ვერტი-კალური სივრცე;
- Margin - ამატებს ცარიელ სივრცეს ელემენტის ირგვლივ;
- MinWidth / MaxWidth აყენებს ელემენტის მაქსიმალურ ზომებს;
- Width და Height – ცხადად აყენებს ელემენტის ზომებს.

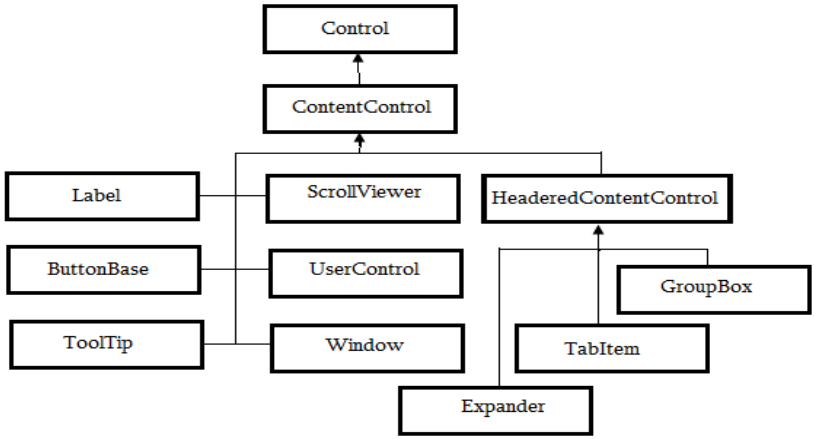
WPF-ში არსებობს კონტეინერები, რომლებსაც აქვს მართვის ელემენტები მოცემული კოორდინატების შესაბამისად, რომლებიც

მომხმარებლის ინტერფეისიზომებითაა მოცემული. ეს კონტეინერებია Canvas და InkCanvas. ისინი ძირითადად გამოიყენება გრაფიკული პრიმიტივების და ფიგურების ვიზუალიზაციისთვის.

1.4. მართვის ელემენტები და შიგთავსები

მართვის ელემენტების დანიშნულებაა მომხმარებელის ინტერაქტიული კავშირის მხარდაჭერა. მათ შეუძლია ფოკუსის და შემავალი მონაცემების მიღება კლავიატურიდან ან მაუსიდან.

შიგთავსის მართვის ელემენტები სპეციალიზებული ტიპის მართვის ელემენტებია, რომლებიც ინახავს განსაზღვრულ შიგთავსს - ერთ ან რამდენიმე ელემენტს. შიგთავსის ყველა მართვის ელემენტი მემკვიდრეა ContentControl კლასის (ნახ.1.4).



ნახ. 1.4. შიგთავსის მართვის ელემენტების იერარქია

ContentControl კლასი მემკვიდრეა System.Windows.Control კლასის, რომელიც მას და მის შვილობილ კლასებს ანიჭებს საბაზო მახასიათებლებს, რომლებიც:

- საშუალებას იძლევა განისაზღვროს შიგთავსი მართვის ელემენტის შიგნით;
- საშუალებას იძლევა განისაზღვროს გადასვლების მიმდევრობა Tab-კლავიშით;

- ხელს უწყობს ფონის, წინა პლანის და ჩარჩოს ხატვას;
- ხელს უწყობს ტექსტური შინაარსის ზომის და შრიფტის ფორმატირებას.

მართვის ელემენტებს აქვს ფონი (ელემენტის ზედაპირი) და ტექსტი (ფონზე მოთავსებული). მათი ფერები WPF-ში განისაზღვრება Background და Foreground თვისებებით. ისინი იყენებს ფუნჯს - ობიექტი Brush. იგი უზრუნველყოფს ფონის და ტექსტის შევსებას მთლიანი ფერით (ფუნჯის კლასი SolidColorBrush), ან გრადიენტულით, მაგალითად, LinearGradientBrush ფუნჯის კლასის დახმარებით.

მაგალითად, ღილაკის ფონის მისაცემად Brush ობიექტით აუცილებელია SolidColorBrush ფუნჯის Color თვისებას მიენიჭოს ფერის მნიშვნელობა, მაგალითად, Blue - ლურჯი.

```
<Button> ღილაკი A
  <Button.Background>
    <SolidColorBrush Color="Blue"></SolidColorBrush>
  </Button.Background>
</Button>
```

WPF-ში შესაძლებელია მოკლე ფორმის გამოყენებაც. მაგალითად, იგივეს ექნება ასეთი სახე:

```
<Button Background="Black"
      Foreground="Red">ღილაკი A</Button>
```

ამ დროს WPF-ის სინტაქსური ანალიზატორი ავტომატურად შექმნის SolidColorBrush ობიექტებს მითითებული ფერებით და გამოიყენებს ამ ობიექტებს ფონისა და ტექსტისათვის.

თუ საჭიროა გრადიენტული ტიპის LinearGradientBrush ფუნჯის გამოყენება, მაშინ ფუნჯის ობიექტი უნდა შეიქმნას დამოუკიდებლად:

```
<Button>ღილაკი A
  <Button.Background>
    <LinearGradientBrush>
    <LinearGradientBrush.GradientStops>
      <GradientStop Offset="0.0" Color="Blue"></GradientStop>
      <GradientStop Offset="1.0" Color="Red"></GradientStop>
```

```
  </LinearGradientBrush.GradientStops>
  </LinearGradientBrush>
 </Button.Background>
</Button>
```

ფუნჯის დახმარებით იხატება ჩარჩო მართვის ელემენტის ირგვლივ. ეს ხორციელდება BorderBrush და BorderThickness თვისებებით. ამ დროს BorderBrush თვისება იღებს არჩეულ ფუნჯს, ხოლო BorderThickness – ჩარჩოს სიგანეს:

```
<Button Width="80" Height="30" Background="Red"
BorderBrush="Blue" BorderThickness="5">ღილაკი A</Button>
```

WPF ტექნოლოგია უზრუნველყოფს გამჭვირვალობას. თუ ფორმაზე ან გვერდზე განლაგებულია რამდენიმე ელემენტი ერთიმეორეზე, და მათთვის განსაზღვრულია სხვადსხვა ხარისხის გამჭვირვალობა, მაშინ ქვედა ელემენტები (წარწერებით) გამოჩნდება ზედა ელემენტების შიგნით. ეს საშუალებას იძლევა შეიქმნას მრავალშრიანი ანომაციური ობიექტები. გამჭვირვალობა შეიძლება მოიცეს ორი ხერხით:

- თვისებით Opacity (არაგამჭვირვალე), რომელიც იღებს მნიშვნელობას საზღვრებში 0-1. აქ 1 არის სრულიად არაგამჭვირვალე, ხოლო 0 - სრულად გამჭვირვალე;
- ნახევრადგამჭვირვალე ფერის გამოყენებით, ალფა-არხის მნიშვნელობის მიწოდებით. თუ ალფა-არხის მნიშვნელობა 255-ზე ნაკლები, მაშინ ის ნახევრადგამჭვირვალეა.

Control კლასი განსაზღვრავს შრიფტების თვისებათა ერთობლიობას:

- FontFamily – შრიფტის სახელი მართვის ელემენტში;
- FontSize – შრიფტის ზომა (1/96 დიუმი);
- FontStyle – ტექსტის დახრის მითითება;
- FontWeight – ტექსტის წონა;
- FontStretch – სიდიდე, რომლის მიხედვითაც გაიწელება ან შეიკუმშება ტექსტი.

მართვის ელემენტისთვის ტექტსის შრიფტის არჩევისას უნდა მიეთითოს შრიფტების ოჯახის სრული სახელი:

```
<Button FontFamily="Sylfaen" FontSize="18">ღილაკი A</Button>
```

ნახევრადსქელი, დახრილი შრიფტის ასარჩევად უნდა მიეცეს შემდეგი თვისებები:

```
<Button FontFamily="Sylfaen"
FontSize="18" FontStyle="Italic" FontWeight="Bold"> ღილაკი A</Button>
```

WPF-ის მრავალი მართვის ელემენტი არის შიგთავსის მართვის ელემენტები. ესაა: Label, Button, CheckBox და RadioButton.

Label - ჭდე: არის შიგთავსის უმარტივესი მართვის ელემენტი. ამ ფუნქციისთვის გამოიყენება Target თვისება, რომელსაც ენიჭება მისაბმელი გამოსახულება. აქ უნდა მიეთითოს სხვა მართვის ელემენტი, რომელზეც გადავა ფოკუსი სწრაფი წვდომის ღილაკის დაჭერისას.

```
<Label Target="{Binding ElementName = txtA}">არჩევა_A</Label>
<TextBox Name="txtA">ტექსტის არჩევა</TextBox>
```

ჭდის ტექსტში რომელიმე ასოზე ხაზგასმა მიუთითებს სწრაფი წვდომის ღილაკზე. ასეთ მნემონიკურ ბრძანებებში ერთდროულად მუშაობს <Alt+A>, სადაც **A** ხაზგასმული ასოა. ჩვენი მაგალითისათვის ფოკუსი გადაეცემა მართვის ელემენტს TextBox, რომლის სახელია txtA.

WPF-ში განსაზღვრულია სამი კლასი: Button, CheckBox და RadioButton, რომლებიც ButtonBase კლასის მემკვიდრეებია. ButtonBase კლასი განსაზღვრავს Click მოვლენას და უმატებს იმ ბრძანებათა მხარდაჭერას, რომლებიც უზრუნველყოფს ღილაკების მიერთებას დანართის ამოცანებისათვის. აქვეა ClickMode თვისება, რომელიც განსაზღვრავს თუ როდის აგენერირებს ღილაკი Click-მოვლენას მაუსის მოქმედების საპასუხოდ. გამოუცხადებლად ესაა ClickMode.Release მნიშვნელობა, რომელიც ნიშნავს მოვლენის გენერირებას მაუსის ღილაკის დაჭერის ან აშვების დროს. ClickMode.Press - მხოლოდ დაჭერის დროს, ClickModeHover - როცა მაუსის კურსორი ღილაკზე დადგება და შეჩერდება.

Button კლასი ამატებს ორ თვისებას: IsCancel და IsDefault. როცა IsCancel = true, მაშინ ღილაკი იმუშავებს როგორც ფანჯრის შეცვლა, <Esc>-ის მსგავსად. თუ IsDefault= true, მაშინ ღილაკი ითვლება აქტიურად გამოუცხადებლად.

CheckBox და RadioButton კლასები არის ToggleButton კლასის მემკვიდრეები, რომელიც ასახავს ღილაკს ორი მდგომარეობით: „დაჭერილია" და „აშვებულია". ამ კლასში არის მოვლენები: Checked, Unchecked და Intermediate, რომლებიც გენერირდება ღილაკის ჩართვის, ამორთვის ან განუსაზღვრელ მდგომარეობაში გადასვლისას.

CheckBox ღილაკისთვის ჩართვა ნიშნავს „ალმის" („ v ") დაყენებას. IsChecked თვისებას, რომელიც ToggleButton-იდანაა ნაანდერძევი, შეუძლია სამი მნიშვნელობის მიღება: true (ჩართული), false (გამორთული), null (განუსაზღვრელი, რომელიც ჩანს როგორც ჩამუქებული ფანჯარა, და გამოიყენება როგორც შუალედური მდგომარეობა). XAML-აღწერა CheckBox-ის ამ სამი მდგომარეობისა ნაჩვენებია ქვემოთ:

```
<CheckBox Height="16" Name="checkBox1"
    Width="120" IsChecked="False"
            ClickMode="Release">არჩევა A</CheckBox>
<CheckBox Height="16" Name="checkBox2"
    Width="120" IsChecked="True"
            ClickMode="Press">არჩევა B</CheckBox>
<CheckBox Height="16" Name="checkBox3"
    Width="120" IsChecked="{x:Null}"
            ClickMode="Hover">არჩევა C</CheckBox>
```

RadioButton ღილაკისთვის დამატებულია GroupName თვისება, რომელიც უზრუნველყოფს გადამრთველების განლაგების მართვას ჯგუფში. ჯგუფიდან შეიძლება მხოლოდ ერთი გადამრთველის არჩევა.

```
<GroupBox Header="რადიოღილაკების ჯგუფი" Height="100"
                                        Name="groupBox1" Width="200">
  <StackPanel>
```

```
    <RadioButton Height="16" Name="radioButton2"
                 Width="120">არჩევა D</RadioButton>
    <RadioButton Height="16" Name="radioButton1"
                 Width="120">არჩევა E</RadioButton>
    <RadioButton Height="16" Name="radioButton3"
                 Width="120">არჩევა F</RadioButton>
  </StackPanel>
</GroupBox>
```

კონტექსტური ფანჯრის (მაუსის კურსორის მიტანისას მართვის ელემენტზე გამოჩნდება პატარა ფანჯარა ტექსტით, ხოლო კურსორის მოცილებისას - გაქრება) გამოტანა შეიძლება ToolTip კლასის ან თვით მართვის ელემენტის ToolTip თვისების გამოყენებით.

```
<Button ToolTip="ღილაკი A კონტექსტური ტექსტით "></Button>
```

ScrollViewer მართვის ელემენტი უზრუნველყოფს შიგთავსის დათვალიერებას ტექსტის ეკრანზე გადახვევით. ამ ელემენტით იქმნება ტექსტების გადახვევის შესაძლებლობის მქონე პანელები.

UserControl ელემენტის დანიშნულებაა მომხმარებლის კონტროლის ელემენტების შექმნა, რომლებშიც ერთიანდება რამდენიმე სხვა ელემენტი.

Windows ელემენტი შიგთავსის მართვის ელემენტია და გამოიყენება დანართის ყველა ფანჯრის შესაქმნელად.

HeaderedContentControl ელემენტი ContentControl კლასის მემკვიდრეა და მშობელია იმ ელემენტებისა, რომელთაც შიგთავსის გარდა აქვს სათაურის არე.

GroupBox ელემენტი არის ფანჯარა სათაურით და გამოიყენება დაკავშირებული ელემენტების დაჯგუფებისათვის, მაგალითად, რადიოღილაკები.

TabItem მართვის ელემენტი ასახავს გვერდს TabControl კლასისთვის. ეს ელემენტი ასახავს იმ ჩადგმულ გვერდს TabControl პანელში, რომელიც აქტიურია ამ წუთში.

Expander მართვის ელემენტი უზრუნველყოფს შიგთავსის განსაზღვრული უბნის გამოჩენას ან დამალვას.

1.5. ტექსტური მართვის ელემენტები

WPF-ში განსაზღვრულია შემდეგი ტექსტური მართვის ელემენტები: TextBlock, TextBox, RichTextBox და PasswordBox.

PasswordBox ელემენტი მემკვიდრეა Control კლასის, ხოლო ელემენტები TextBox და RichTextBox - მემკვიდრეა TextBase კლასის.

TextBlock ელემენტი გამოიყენება მცირე ზომის ტექსტისთვის.

TextBox ელემენტი ინახავს ტექსტის სტრიქონს.

RichTextBox ელემენტს შეუძლია FlowDocument-ის შიგთავსის შენახვა, რომელშიც შეიძლება იყოს ელემენტთა რთული კომბინაცია.

PasswordBox ელემენტი შეიცავს ტექსტის სტრიქონს, იყენებს SecureString ელემენტს დასაცავად ზოგიერთი სახის თავდასხმებისგან.

TextBox ელემენტი, ჩვეულებრივად ინახავს ერთ სტრიქონს: <TextBox Name="txtA">ტექსტის არჩევა</TextBox>.

თუ საჭიროა მრავალსტრიქონიანი წარმოდგენის შექმნა, მაშინ თვისებას TextWrapping მიენიჭება Wrap მნიშვნელობა. ასეთი TextBox-ისთვის შეიძლება მინიმალური და მაქსიმალური სტიქონების რაოდენობის მითითება, MinLines და MaxLines თვისაებათა გამოყენებით.

1.6. სიების მართვის ელემენტები

ListBox და ComboBox - სიების მართვის ელემენტებია. ისინი ItemsControl კლასის მემკვიდრეებია Selector კლასის გავლით.

ItemsControl კლასი საბაზოა სიების მართვის ყველა ელემენტისათვის და ითვალისწინებს სიის ელემენტების შევსების ორ ხერხს. პირველში ხდება ელემენტის დამატება უშუალოდ Items კოლექციაში, რომლის დროსაც გამოიყენება კოდი ან XAML. მეორე ხერხით ხდება მონაცემთა მიბმა ItemsSource თვისებით (მონაცემთა წყარო).

Selector კლასს აქვს თვისებები, რომლებიც თვალყურს ადევნებს მოცემული დროის მომენტში გამოყოფილი სიის ელემენტს (SelectedItem) ან მის პოზიციას (SelectedIndex).

ელემენტთა დასამატებლად ListBox-ში შეიძლება ჩაიდოს ListBoxItem-ის ელემენტები ListBox-ში, როგორც ეს ქვემოთაა ნაჩვენები ფერთა სიის შესადგენად (მწვანე, ცისფერი, ყვითელი, წითელი):

```
<ListBox>
    <ListBoxItem>მწვანე</ListBoxItem>
    <ListBoxItem>ცისფერი</ListBoxItem>
    <ListBoxItem>ყვითელი</ListBoxItem>
    <ListBoxItem>წითელი</ListBoxItem>
</ListBox>
```

მართვის ელემენტი ListBox ინახავს თავის კოლექციაში ყოველ ჩადგმულ ობიექტს. ამავდროულად, ListBoxItem-ს შეუძლია არა მხოლოდ სტრიქონების შენახვა, არამედ ნებისმიერი თავისუფალი ელემენტის.

მართვის ელემენტი ComboBox მსგავსია ListBox-ის, ოღონდ ვიზუალიზაციისთვის გამოიყენებს ჩამოშლად სიას, რომლიდანაც მომხმარებელი ირჩევს მხოლოდ ერთ ელემენტს.

1.7. სპეციალიზებული მართვის ელემენტები

WPF-ში არის მართის ელემენტები, რომლებიც იყენებს მნიშვნელობათა დიაპაზონებს: ScrollBar, ProgressBar და Slider. მართვის ეს ელემენტები მემკვიდრეებია RangeBase კლასის, რომელშიც განსაზღვრულია ისეთი თვისებები, როგორიცაა Value (ელემენტის მიმდინარე მნიშვნელობა), Minimum და Maximum დასაშვები მნიშვნელობები.

ScrollBar მართვის ელემენტი იძლევა გადახვევის ზოლს გადაადგილებადი ელემენტით, რომლის პოზიცია შეესაბამება განსაზღვრულ მნიშვნელობას.

ProgressBar მართვის ელემენტი უჩვენებს ხანგრძლივი ამოცანის შესრულების მსვლელობას.

Slider მართვის ელემენტი გამოიყენება რიცხვითი მნიშვნელობის მოსაცემად კურსორის გადაადგილების შესაბამისად გადახვევის სახაზავზე.

1.8. ბრძანებები

WPF-ის ბრძანებათა მოდელი იძლევა შემდეგ შესაძლებლობებს:

- მოვლენათა დელეგირება შესაფერის ბრძანებებზე;
- მართვის ელემენტის ჩართული მდგომარეობის შენარჩუნება სინქრონიზებული სახით შესაბამისი ბრძანების მდგომარეობის დახმარებით.

ბრძანებათა მოდელი შეიცავს შემდეგ კომპონენტებს:

- ბრძანებები, რომლებიც წარმოადგენს დანართის ამოცანას, მათი წვდომის თვალყურის დევნის მექანიზმით პროგრამის შესრულების დროს;
- ბრძანებათა მიბმა, რომელიც გულისხმობს ბრძანების მიერთებას დანართის ლოგიკასთან. იგი პასუხისმგებელია მომხმარებლის ინტერფეისის გარკვეული უბნის მომსახურებაზე;
- ბრძანების წყარო, რომელსაც იგი აინიცირებს;
- ბრძანების მიზნობრივი ობიექტები - დანართის ელემენტები, რომლებისთვისაც არის გამიზნული ეს ბრძანება.

WPF-ის ბრძანებათა კლასები ICommand ინტერფეისს უნდა უჭერდეს მხარს.

```
public interface ICommand
  {
     void Execute(object par);
     bool CanExecute(object par);
     event EventHandler CanExecuteChanged;
  }
```

მეთოდი Execute() განსაზღვრავს ბრძანების რეალიზაციას დანართში. ეს შეიძლება იყოს დანართის ლოგიკის კოდი ან მოვლენის ამოქმედება, რომელიც მუშავდება დანართის სხვა კლასში. მეთოდი CanExecute() აბრუნებს ინფორმაციას ბრძანების წვდომის მდგომარეობაზე. Execute() და CanExecute() მეთოდები შეიძლება გამოძახებულ იქნას **par** პარამეტრით, რომელიც გამოიყენება დამატებითი ინფორმაციის გადასაცემად.

CanExecuteChanged მოვლენა გამოიძახება ბრძანების წვდომის მდგომარეობის შეცვლისას. ამ შემთხვევისთვის

გამოიძახება CanExecute() მეთოდი და მოწმდება ბრძანების მდგომარეობა.

WPF-ში არსებობს საბაზო ბრძანებათა ბიბლიოთეკა, რომელიც ხელმისაწვდომია შემდეგი სტატიკური კლასების სტატიკური თვისებებიდან:

- ApplicationCommands – ზოგადი ბრძანებები, მაგალითად, Create, Open, Colpy, Store და სხვა;
- Navigation Commands – ბრძანებები ნავიგაციისთვის;
- Editing Commands – ბრძანებები დოკუმენტაციის რედაქტირებისთვის;
- Media Commands – მულტიმედიასთან სამუშაო ბრძანებები.

ბრძანების ინიციალიზაციისთვის საჭიროა ბრძანების წყაროს გამოყენება, ხოლო მასზე საპასუხოდ - ბრძანების მიბმა მისი შესრულების გადამისამართებასთან ჩვეულებრივ მოვლენის დამმუშავებელზე.

დავუშვათ, რომ **შექმნა** ბრძანების წყარო არის ღილაკი New. Command თვისების დახმარებით ღილაკს ვაბამთ New ბრძანებას.

```
<Button Name="New" Content="შექმნა" Width="200"
                Command="New"/>
```

შემდეგ ბიჯზე აუცილებელია ბრძანების მიბმა Window ფანჯარასთან.

```
<Window.CommandBindings>
    <CommandBinding Command="New"
            Executed="CommandBinding_Executed"
            CanExecute="CommandBinding_CanExecute"/>
</Window.CommandBindings>
```

მიბმის დროს CommandBinding ობიექტისთვის, გარდა New ბრძანებისა, საჭიროა დამმუშავებლების მითითება Execute() და CanExecute() მეთოდებისთვის, შესაბამისად Executed და CanExecute თვისებებით. დამმუშავებელი CommandBinding_Executed რეალიზებას გაუკეთებს **შექმნა**-ბრძანების ბიზნეს-ლოგიკას, ხოლო დამმუშავებელი CommandBinding_CanExecute კი - ბრძანების წვდომის შემოწმებას.

მომხმარებლის ბრძანების შექმნისას მიზანშეწონილია RoutedCommand კლასის გამოყენება, რომელიც რეალიზაციას უკეთებს ICommand ინტერფეისს.

საილუსტრაციოდ განვიხილოთ **რედაქტირება**-ბრძანების კონსტრუირება. შევქმნათ DataCommands კლასი.

```
public class DataCommands
{
    private static RoutedCommand edit;
    public static RoutedCommand Edit
   {
      get { return DataCommands.edit; }
   }
  static DataCommands()
  {
    InputGestureCollection inputs = new InputGestureCollection();
    inputs.Add(new KeyGesture(Key.E, ModifierKeys.Control, "Ctrl+E"));
    Edit = new RoutedCommand("Edit", typeof(DataCommands), inputs);
  }
}
```

DataCommands კლასში გამოცხადებულია RoutedCommand კლასის ეგზემპლარის სტატიკური ველი (edit) და თვისება (Edit). სტატიკური კონსტრუქტორი ქმნის **შექმნა**-ბრძანების (Edit) ეგზემპლარს, იყენებს რა InputGestureCollection-კლასის (inputs)-ეგზემპლარს. ეს კლასი არის KeyGesture ობიექტების მოწესრიგებული კოლექცია, რომლებიც განსაზღვრავს „ცხელ“ ღილაკს ბრძანების გამოსაძახებლად. მაგალითში ესაა „Ctrl+E”. შემდეგ აუცილებელია ბრძანებისთვის დაყენდეს წყარო და მიებას ბრძანება, როგორც ეს ადრე იყო განხილული.

1.9. რესურსები

ობიექტური რესურსი - ესაა .NET-ობიექტი, რომელიც განისაზღვრება ერთ ადგილას და გამოიყენება რამდენიმე სხვა ადგილას დანართში. ეს რესურსი განისაზღვრება XAML

ფორმატირების ენაში [4,8]. ობიექტურ რესურსებს აქვს რიგი მნიშვნელოვანი მახასიათებლები:

- ეფექტურობა კოდის მრავალჯერადი გამოყენების პრინციპის რეალიზაციის გამო;
- მოხერხებული თანხლება ფორმატირების დაბალი დონის დეტალების გადატანის გამო ერთ ცენტრალურ ადგილას;
- ადაპტაციის უნარი ინფორმაციის დინამიკურად ცვლილების შესაძლებლობის გამო, რომელიც კოდისგან გამოყოფილია.

ყოველი ელემენტი, რომელიც მემკვიდრეა FrameworkElement კლასის (ნახ.1.2) შეიცავს Resources თვისებას, რომელშიც ინახება რესურსების Resourcesdictionary-ლექსიკონური კოლექცია. აქ შეიძლება ინახებოდეს ნებისმიერი ტიპის ობიექტი. ყოველ ელემენტს აქვს წვდომა როგორც საკუთარ რესურების კოლექციასთან, ასევე თავისი მშობლების რესურსების კოლექციებთან. ხშირად რესურსები ინახება ფანჯრის ან გვერდის დონეზე მათი ერთობლივად გამოყენების უზრუნველსაყოფად ყველა მართვის ელემენტის მიერ.

რესურსები განისაზღვრება Resource სექციაში შესაბამისი ელემენტისთვის, მაგალითად, ფანჯრისთვის:

```
<Window x:Class="MyFirstWpfProject.MainWindow" ... >
        <Window.Resources>
                ....
        </Window.Resources>
....
</Window>
```

ყოველი რესურსი ჩაიწერება ასეთი სახით:

```
<TypeResource  x:Key="რესურსის გასაღები" >
        ...
</ TypeResource >
```

(TypeResource)- ობიექტის თვისებების მისაცემად შეიძლება გამოყენებულ იქნას ატრიბუტთა სინტაქსი ან ელემენტთა თვისებათა სინტაქსი. XAML ელემენტები მიმართავს რესურსებს

გასაღებით (KeyResource), რომელიც განისაზღვრება ატრიბუტით x:Key. რესურსის გასაღები მოიცემა StaticResource-ს გამოყენებით.

მაგალითად, დანართში შექმნილია გრადიენტული ფუნჯი ღილაკის ფერით შესავსებად, რომელიც გამოყენებულ უნდა იქნას ინტერფეისის სხვა ელემენტებში. მაშინ გრადიენტული ფუნჯი განსაზღვრული უნდა იყოს როგორც რესურსი:

```
<Window.Resources>
    <LinearGradientBrush x:Key="BrushButton"
                                    EndPoint="0,1" StartPoint="0,0">
        <GradientStop Color="#FFD2E995"  Offset="0.9"/>
        <GradientStop Color="#FF74EDB3"  Offset="0.55"/>
        <GradientStop Color="#FF72D8A7"  Offset="0.75"/>
        <GradientStop Color="#FF233CC6"  Offset="0.25"/>
    </LinearGradientBrush>
</Window.Resources>
<StackPanel>
     <Button x:Name="New" Content="შექმნა" Width="200"
             Command="New" Margin="154.5,0"
              Background="{StaticResource BrushButton}"/>
</StackPanel>
```

გრადიენტულ ფუნჯს აქვს LinearGradientBrush ტიპი და მისთვის მოცემულია გასაღები BrushButton. ღილაკის XAML-აღწერაში ატრიბუტი Background განისაზღვრება გაფართოებული ფორმატით {StaticResource BrushButton}.

ჩვენ მაგალითში გამოყენებულია სტატიკური რესურსი. იგი ამოიღება რესურსების კოლექციიდან მხოლოდ ერთხელ. ამასთანავე, ნებისმიერი ცვლილებები რესურსის ობიექტში შეიმჩნევა მყისიერად. თუ დანართის ფუნქციონირების პროცესში შექმნილი რესურსი შეიძლება შეიცვალოს, მაშინ გამოყენებულ უნდა იქნას დინამიკური რესურსი - DynamicResource. იგი განისაზღვრება რესურსების კოლექციაში ყოველთვის, როცა ამის საჭიროება აღმოცენდება.

რესურსები შეიძლება განისაზღვროს ელემენტის, კონტეინერის, ფანჯრის ან გვერდის, დანართის, სისტემის ან ნაკრების დონეზე.

რესურსების ხელმეორედ გამოსაყენებლად იქმნება რესურსების ლექსიკონები. ესაა XAML-ფაილი რესურსების შესანახად. თუ მაგალითად, ჩვენ მიერ შექმნილი გრადიენტული ფუნჯის რესურსს გადავიტანთ ლექსიკონში, მაშინ რესურსების ლექსიკონის MyDictionary.xaml ფაილს ექნება შემდეგი სახე:

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
      <LinearGradientBrush x:Key="BrushButton"
                                EndPoint="0,1" StartPoint="0,0">
        <GradientStop Color="#FFD2E995" Offset="0.9"/>
        <GradientStop Color="#FF74EDB3" Offset="0.55"/>
        <GradientStop Color="#FF72D8A7" Offset="0.75"/>
        <GradientStop Color="#FF233CC6" Offset="0.25"/>
    </LinearGradientBrush>
</ResourceDictionary>
```

რესურსების ლექსიკონის გამოსაყენებლად იგი გაერთიანებულ უნდა იქნას რესურსების კოლექციასთან, მაგალითად ფანჯრის:

```
<Window.Resources>
   <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
         <ResourceDictionary
                              Source="MyDictionary.xaml" />
      </ResourceDictionary.MergedDictionaries>
   </ResourceDictionary>
</Window.Resources>
```

რესურსების ლექსიკონის გაერთიანება სხვა ობიექტის რესურსეთან წარმოებს ResourceDictionary.MergedDictionaries თვისების მოცემით. MergedDictionaries კოლექცია - ესაა ResourceDictionary ობიექტების კოლექცია, რომლებიც უნდა დაემატოს რესურსებუს კოლექციას. დასამატებელი რესურსების ლექსიკონები მოიცემა როგორც Source ატრიბუტის მნიშვნელობები. განხილულ მაგალითში ესაა რესურსების ლექსიკონის ფაილი MyDictionary.xaml.

1.10. სტილები

სტილები - ესაა თვისებათა მნიშვნელობების კოლექციები, რომლებიც გამოიყენება ელემენტისთვის. ისინი საშუალებას იძლევა განისაზღვროს ფორმატირების მახასიათებლების ზოგადი ერთობლიობა და გამოყენებულ იქნას მთელი დანართის ფარგლებში შეთანხმებადობის უზრუნველსაყოფად. WPF-ში სტილებს შეუძლია დამოკიდებულებათა ნებისმიერი თვისების დაყენება. მათი გამოყენება შეიძლება რომელიმე ელემენტის ვიზუალური ქცევის სტანდარტიზაციისთვის. WPF-ის სტილებს აქვს ტრიგერები, რომლებიც იძლევა საშუალებას ელემენტის სტილის შესაცვლელად სხვა თვისებების შეცვლის დროს. სტილები იყენებს შაბლონებს მართვის ელემენტების სტანდარტული ვიზუალური წარმოდგენის ხელახლა განსაზღვრისთვის.

სტილი იქმნება Style კლასის ბაზაზე, რომლის თვისებები მოცემულია 1.1 ცხრილში.

Style კლასის თვისებები **ცხრ.1.1**

სახელი	აღწერა
BasedOn	აბრუნებს ან იძლევა განსაზღვრულ სტილს, რომელიც საბაზოა მიმდინარე სტილისთვის
Dispatcher	აბრუნებს Dispatcher ობიექტს, რომელთანაც კავშირშია ეს DispatcherObject ობიექტი
IsSealed	აბრუნებს მნიშვნელობას, რომელიც მიუთითებს, არის თუა არა სტილი მხოლოდ წაკითხვის
Resources	აბრუნებს ან იძლევა რესურსების კოლექციას, რომლებიც გამოყენებულ იქნება მოცემული სტილის ხილვადობის არეში
Setters	აბრუნებს Setter და EventSetter ობიექტების კოლექციას
TargetType	აბრუნებს ან იძლევა ტიპს, რომლისთვისაც დანიშნულია ეს სტილი
Triggers	აბრუნებს TriggerBase ობიექტთა კოლექციას, რომლებიც იყენებს თვისებების მნიშვნელობებს მოცემული პირობრბის საფუძველზე

მაგალითად, დანართში საჭიროა მრავალჯერადი გამოყენების ღილაკები, რომელთაც აქვს ღია-ცისფერი ფონი და ცისფერი ჩარჩო.

```
<Style x:Key="ButtonStyle" TargetType="Button">
    <Setter Property="Background"  Value="LightBlue"/>
    <Setter Property="BorderBrush" Value="Blue" />
</Style>
```

სახელი მიეცემა ატრიბუტით x:Key (ButtonStyle). ატრიბუტი TargetType განსაზღვრავს ტიპს, რომლისთვისაც გამოყენებულ იქნება სტილი (Button). ყოველი Setter-ობიექტი აყენებს ელემენტში ერთ თვისებას, რომელიც აუცილებლად უნდა იყოს დამოკიდებულების თვისება.

თვისებების დაყენება წარმოებს Property ატრიბუტის დახმარებით, რომელიც განსაზღვრავს თვისების სახელს და Value - მის მნიშვნელობას.

სტილში შეიძლება EventSetters ობიექტის დამატება, რომელსაც მიებმება მოვლენა გარკვეული დამმუშავებლისთვის.

ტრიგერები იძლევა სტილის შეცვლის უფლებას განსაზღვრული პირობების შესრულებისას.

დავამატოთ ButtonStyle ღილაკის სტილში ტრიგერი, რომელიც დააფორმირებს წითელ ფონს, როცა მაუსის კურსორი დადგება ღილაკზე.

```
<Style x:Key="ButtonStyle" TargetType="Button">
  <Setter Property="Background"
                                Value="LightBlue"/>
  <Setter Property="BorderBrush" Value="Blue" />
  <Style.Triggers>
    <Trigger Property="IsMouseOver" Value="true">
      <Setter Property="Background" Value="Red" />
    </Trigger>
  </Style.Triggers>
</Style>
```

ტრიგერში მითითებული უნდა იყოს მაიდენტიფიცირებელი თვისება (ჩვენ მაგალითში IsMouseOver), რომელიც უნდა

ვაკონტროლოთ, და მნიშვნელობა, რომელსაც უნდა ველოდოთ (მაგალითად, true). როცა გამოჩნდება აუცილებელი მნიშვნელობა, ყენდება თვისება, რომელიც განისაზღვრება Setter ობიექტით.

ჩვენ მაგალითში, როცა IsMouseOver = true, ანუ მაუსის კურსორი მიყვანილია ღილაკზე, მაშინ Background თვისებას მიენიჭება მნიშვნელობა Red, ანუ ღილაკის ფონი ხდება წითელი. როცა კურსორი გამოდის ღილაკის ზონიდან, მაშინ ტრიგერის ამუშავების პირობა ირღვევა და ღილაკის ფონი გადადის საწყის მდგომარეობაში.

თვისების ცვლილების მომლოდინე ტრიგერის გარდა არსებობს მოვლენათა ტრიგერები (EventTrigger), რომლებიც ელოდება განსაზღვრულ მოვლენათა აღმოცენებას.

1.11. შაბლონები

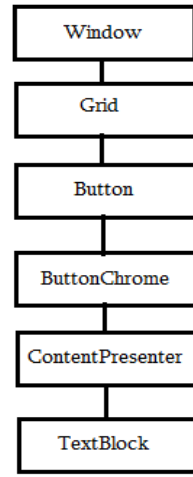
WPF-აპლიკაციების დაპროექტების დროს ფანჯრები ან გვერდები არის კონტეინერები, რომლებშიც განლაგებულია სხვა კონტეინერები და ინტერფეისის სხვადასხვა ელემენტები (ჭდეები, ტექსტური ბლოკები, ღილაკები, სიები და სხვა მართვის ელემენტები). მაგალითად, ფანჯარა ერთი ღილაკით (ნახ.1.5).

ნახ.1.5

XAML-დოკუმენტი, შექმნილი ფანჯრისთვის შემდეგი სახისაა.

```
<Window x:Class="WpfApplication1.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/
                                        xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
    Title="MainWindow" Height="200" Width="300">
    <Grid>
      <Button Content="ღილაკი" Width="100"
                    Height="50"/>
    </Grid>
</Window>
```

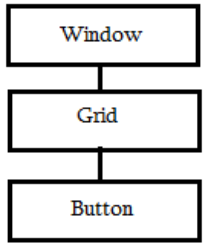


ფანჯრის XAML-დოკუმენტში მოცემულია სამი ობიექტი: Window, Grid და Button. ეს ელემენტები ქმნის ფანჯრის ლოგიკურ ხეს (ნახ.1.6).

ნახ.1.6. ფანჯრის ლოგიკური ხე

WPF-ში ლოგიკური ხე დეტალიზირდება *ვიზუალირი ხის* დახმარებით, რომელიც ასახავს ლოგიკური ხის ელემენტებს უფრო მცირე ფრაგმენტების სახით (ნახ.1.7).

მაგალითად, ღილაკი, ვიზუალური ხის დონეზე, ინკაპსულირებულია მართკუთხედის სახით, რომელიც შეიცავს საზღვარს (ButtonChrome კლასი), შინაარსს (ContentPresenter კლასი) და ბლოკს ტექსტით (TextBlock კლასი).



საერთოდ, არსებობს არაერთი ხერხი ლოგიკური ხის გაფართოებისა ვიზუალურ ხემდე. მაგალითად, ელემენტი Button, რომელიც შიგთავსის ელემენტია, შეუძლია შეიცავდეს ნებისმიერ სხვა ელემენტს, რაც აისახება მის ვიზუალურ ხეში.

ნახ.1.7. ფანჯრის ვიზუალური ხე

ვიზუალური ხე საშუალებას იძლევა შეიცვალოს მისი ელემენტები სტილების დახმარებით და შეიქმნას ახალი შაბლონები მართვის ელემენტებისათვის.

WPF-ში არსებობს მართვის ელემენტების შაბლონები, მონაცეთა შაბლონები და პანელის სპეციალური შაბლონები.

ControlTemplate - მართვის ელემენტების შაბლონი გამოიყენება ამ ელემენტების გამოსახვის (წარმოდგენის) და მათი ვიზუალური ქცევის მოსაცემად.

DataTemplate - მონაცემთა შაბლონი გამოიყენება მონაცემთა ამოსაღებად ობიექტიდან და მათი ასახვისთვის შიგთავსის მართვის ელემენტში.

Hierarchical DataTemplate - პანელების შაბლონები გამოიყენება ელემენთა კომპლექტების მართვისათვის სიის ტიპის მართვის ელემენტებში.

1.11.1. მართვის ელემენტთა შაბლონები

მართვის ყოველ ელემენტს აქვს Template-თვისება, რომელიც განსაზღვრავს შაბლონს მისი ვიზუალიზაციისთვის. თუ ეს თვისება ცხადად არაა მოცემული, მაშინ გამოიყენება მართვის ელემენტის სტანდარტული შაბლონი, რომელიც WPF-შია განსაზღვრული. მართვის ელემენტის მომხმარებლის შაბლონის შესაქმნელად აუცილებელია განისაზღვროს ControlTemplate ობიექტი.

```
<ControlTemplate x:Key="შაბლონის_გასაღები"
                 TargetType="ელემენტის_ტიპი">
    ...
</ControlTemplate>
```

ატრიბუტი x:Key განსაზღვრავს გასაღებს, რომლითაც მიმართავენ შაბლონს, ხოლო TargetType განსაზღვრავს ელემენტის ტიპს, რომლისთვისაც იქმნება შაბლონი.

ღილაკის უმარტივესი შაბლონის მაგალითი მოცემულია ქვემოთ:.

```
<ControlTemplate x:Key="ButtonTemplate"
                 TargetType="Button">
    <Border BorderBrush="Blue" BorderThickness="3"
```

```
                    CornerRadius="25"
    Background="Azure" TextBlock.Foreground="Green">
        <ContentPresenter
            HorizontalAlignment="Center"
            VerticalAlignment="Center"/>
    </Border>
</ControlTemplate>
```

ContentPresenter ელემენტი განსაზღვრავს მართვის ელემენტის შიგთავსის ჩასმის ადგილს კონტეინერში (მაგალითად, კონტეინერი არის ჩარჩო – Border).

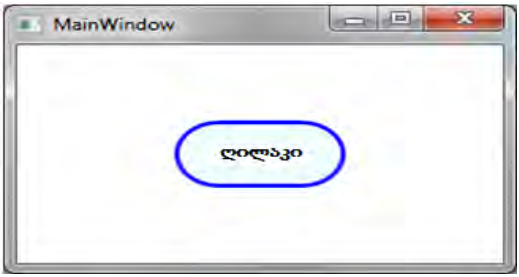
მართვის ელემენტის შაბლონი შეიძლება განთავსდეს ფანჯრის რესურსების კოლექციაში.

```
<Window.Resources>
        <ControlTemplate x:Key="ButtonTemplate"
                                    TargetType="Button">
        ...
        </ControlTemplate>
</Window.Resources>
```

შაბლონის მიბმა მართვის ელემენტთან ხორციელდება ამ ელემენტის Template თვისების მიცემით.

```
<Grid>
    <Button Content="ღილაკი" Width="100" Height="50"
                Template="{StaticResource
                                ButtonTemplate}"/>
</Grid>
```

Template თვისება განისაზღვრება განლაგების (layout) გაფართოებით სტატიკურ რესურსზე (StaticResource) მიმართვისთვის რესურსის გასაღებით (ButtonTemplate). პროექტის ამუშავებით ღილაკს ექნება ასეთი სახე (ნახ.1.8).



ნახ.1.8. ღილაკის წარმოდგენა შაბლონის გამოყენებით

განხილული შაბლონი განსაზღვრავს ღილაკის სტატიკურ ვიზუალურ წარმოდგენას. დინამიკური ვიზუალური ქცევის დასამატებლად შეიძლება შაბლონთა ტრიგერების გამოყენება. მაგალითად, დავამატოთ შაბლონში ტრიგერი, რომელიც შეცვლის ღილაკის ფონის შევსებას წითელი ფერით, როცა მასზე მაუსის კურსორი დადგება, ამასთანავე შეცვლის წარწერის ფერს თეთრით.

```
<ControlTemplate x:Key="ButtonTemplate" TargetType="Button">
   <Border Name="Border" BorderBrush="Blue"  BorderThickness="3"
      CornerRadius="25" Background="Azure"
          TextBlock.Foreground="Green">
     <ContentPresenter
               HorizontalAlignment="Center"
               VerticalAlignment="Center"/>
   </Border>
   <ControlTemplate.Triggers>
     <Trigger Property="IsMouseOver" Value="true">
       <Setter TargetName="Border"
                  Property="Background" Value="Red" />
       <Setter TargetName="Border"
                  Property="TextBlock.Foreground" Value="White" />
     </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>
```

როცა ამოქმედდება მოვლენა MouseOver თვისება IsMouseOver გახდება true, შედეგად იმუშავებს ტრიგერი და შეიცვლება ღილაკის თვისებები Background და TextBlock.Foreground შესაბამისად Red და White-ით.

Background და TextBlock.Foreground თვისებების დაყენებისას TargetName-ელემენტში მიეთითება Border სახელი, რომელიც მინიჭებული აქვს Border-ელემენტს შაბლონში.

1.9 ნახაზზე ნაჩვენებია ღილაკის წარმოდგენა, როცა მასზე მაუსის კურსორია მოთავსებული.

ნახ.1.9

ტრიგერების გამოყენებით შეიძლება განისაზღვროს მართვის ელემენტების ვიზუალური ქცევის რეაქცია აუცილებელ მოვლენებზე.

1.11.2. მონაცემთა შაბლონები

მონაცეთა შაბლონი - ესაა XAML-ფორმატირების ნაწილი, რომელიც განსაზღვრავს, თუ როგორ უნდა აისახოს მონაცემების მიმაგრებული ობიექტი. მონაცემთა შაბლონს შეიძლება ჰქონდეს ელემენტების ნებისმიერი კომბინაცია და უნდა შეიცავდეს ერთ ან მეტ მიბმის გამოსახულებას.

მონაცემთა შაბლონებს აქვს შემდეგი ელემენტები:

- შიგთავსის მართვის ელემენტები ContentTemplate თვისებით, რომელიც გამოიყენება ნებისმიერი Content-ში მოთავსებული შიგთავსის ვიზუალიზებისთვის;

- სიისებური მართვის ელემენტები ItemTemplate თვისებით, კოლექციის ელემენტების ვიზუალიზაციისთვის, რომელიც მითითებულია როგორც მონაცემთა წყარო.

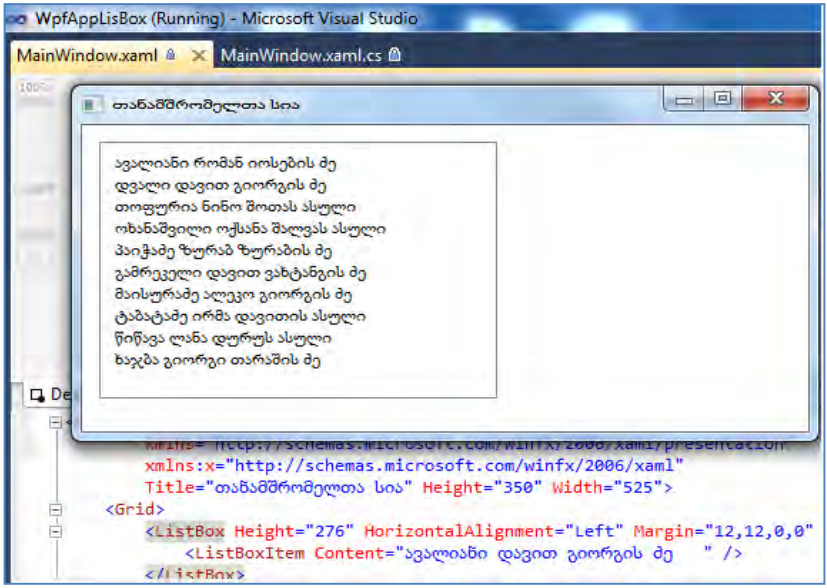
მაგალითად, დავამუშავოთ მონაცემთა შაბლონი ListBox სიისთვის. სიაში უნდა აისახოს მონაცემები თანამშრომლის შესახებ: გვარი, სახელი და მამის სახელი. მონაცემთა წყაროდ listBoxEmployees სიისთვის განიხილება კოლექცია Employees, რომელიც შეიცავს Employee კლასს. Employee კლასის თვისებებია:

- Surname –გვარი;
- Name - სახელი;
- Patronymic – მამის სახელი.

მონაცემთა შაბლონს listBoxEmployees სიისთვის აქვს შემდეგი სახე:

```
<ListBox Name="listBoxEmployees" >
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
          <TextBlock Text="{Binding Path=Surname}" />
          <TextBlock Text="{Binding Path=Name}" />
          <TextBlock Text="{Binding Path=Patronymic}" />
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

პროგრამის ამუშავებით ფანჯარაში გამოიტანება სია (ნახ.1.10).



ნახ.1.1.10. სიის წარმოდგენა მომხმარებლური მონაცემთა შაბლონით

მონაცემთა შაბლონის მრავალჯერადი გამოყენების მიზნით ის უნდა მოიცეს ფანჯრის ან დანართის რესურსის სახით, შაბლონის გასაღების მითითებით (ListBoxEmployee).

```
<Application.Resources>
        <DataTemplate x:Key="ListBoxEmployee" >
        ...
        </DataTemplate>
</Application.Resources>
```

ამავე დროს სიის XAML-აღწერაში ItemTemplate თვისებისთვის მოიცემა ფორმატის გაფართოება სტატიკურ რესურსზე.

```
<ListBox Grid.Row="1" Name="listBoxEmployees"
```

```
        ItemTemplate="{StaticResource
ListBoxEmployee}" />
```

მონაცემთა შაბლონი არის ეფექტური ინსტრუმენტი მართვის ელემენტთა ვიზუალური ასახვის ცვლილებისთვის.

1.12. XAML ენის საფუძვლები

მომხმარებელთა ინტერფეისების აგება WPF- და Silverlight-დანართებისთვის (აპლიკაციებისთვის) ხორციელდება XAML (Extensible Application Markup Language - აპლიკაციების გაფართოებადი ფორმატირების ენა) ენის გამოყენებით. XAML-დოკუმენტი შეიცავს ფორმატს, რომელიც აღწერს დანართის ფანჯრის (ან გვერდის) გარეგან სახეს და ქცევას, ხოლო მასთან კავშირში მყოფი C# კოდის ფაილები კი - დანართის ლოგიკას. XAML-ენა უზრუნველყოფს დანართის დიზაინის პროცესის (გრაფიკული ნაწილი) გამოყოფას ბიზნეს-ლოგიკის (პროგრამული კოდი) დამუშავების პროცესისგან, დიზაინერებსა და დეველოპერებს შორის [4,5].

WPF-ის XAML არის XML-ენის ქვესიმრავლე, გაფართოებული დამატებითი ფუნქციებით. იგი უზრუნველყოფს WPF-შიგთავსის აღწერას ისეთი ელემენტებით, როგორიცაა ვექტორული გრაფიკა, მართვის ელემენტები და დოკუმენტები.

XAML-ის საფუძველია XML და მისი სინტაკსი განისაზღვრება შემდეგი წესებით:

- XAML-დოკუმენტის ყოველი ელემენტი აისახება .NET კლასის რომელიღაც ეგზემპლარში. ასეთი ელემენტის სახელი ზუსტად შეესაბამება კლასის სახელს. მაგალითად, <Button> ელემენტი ემსახურება WPF-ინსტრუქციას Button-კლასის ობიექტის აგების მიზნით;
- XAML-ის ელემენტები შეიძლება ერთმანეთში ჩალაგდეს. ელემენტების ჩალაგების ფორმატი ასახავს ინტერფეისის ელემენტების ჩალაგებას;

- კლასის თვისებები განისაზღვრება ატრიბუტებით ან ჩალაგებული დესკრიპტორების დახმარებით, სპეცსინტაკსით.

XAML-ენა ხასიათდება თვითაღწერადობით. XAML-დოკუმენტში ყოველი ელემენტი არის ტიპის სახელი (მაგალითად, Button, Window ან Page) მოცემული სახელსივრცის ჩარჩოებში.

ელემენტთა ატრიბუტები გამოიყენება შესაბამისი ობიექტების თვისებების (მაგალითად, Name, Height, Width და ა.შ.) და მოვლენების (Click, Load და ა.შ) მოსაცემად.

WPF-დანართის MyFirstWpfProject შექმნის დროს VisualStudio აგენერირებს შემდეგ XAML-დოკუმენტს.

```
<Window x:Class="MyFirstWpfProject.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/
                                        2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
  <Grid>
  . . .
  </Grid>
</Window>
```

WPF-დანართის XAML-დოკუმენტი MyFirstWpfProject იწყება დესკრიპტორით < Window...>.

XAML-დოკუმენტის ყველა დესკრიპტორი იწყება „<“ - სიმბოლოთი და მთავრდება „>“ - სიმბოლოთი. ნებისმიერი XAML-დოკუმენტი შედგება XAML-ელემენტებისგან. ყოველი XAML-დოკუმენტი (XAML-ელემენტი) იწყება გახსნის დესკრიპტორით (მაგალითად, < Window >), რომელსაც მოჰყვება დოკუმენტის შიგთავსი (მაგალითად, ტექსტური სტრიქონი ან სხვა XAML-ელემენტები). გახსნის დესკრიპტორში შეიძლება იყოს მოთავსებული ატრიბუტების აღწერა (მაგალითად, Class, xmlns, Title, Height, Width და სხვ.). XAML-დოკუმენტი (XAML-ელემენტი)

უნდა დასრულდეს დახურვის დესკრიპტორით (მაგალითად, „/>“ ან „</Window>“).

XAML-დოკუმენტის ტექსტი უნდა შეიცავდეს ერთ ფესვურ ელემენტს - ჩალაგების უმაღლესი დონის ელემენტი. WPF-დანართის MyFirstWpfProject XAML-დოკუმენტში ასეთი ელემენტია <Window>. ფესვურ ელემენტში შეიძლება დაემატოს XAML-ის სხვა ელემენტებიც. მაგალითში ასეთი ელემენტია <Grid>.

WPF-დანართის XAML-დოკუმენტის კომპილაციის პროცესში სინტაკსურ ანალიზატორს გადაჰყავს XAML ფაილები აპლიკაციის ორობითი ფორმატირების ფაილებში BAML (Binary Application Markup Language), რომლებიც შემდეგ ჩაშენდება პროექტის ნაკრებში რესურსების სახით. WPF-დანართის კლასების ასაგებად სინტაკსური ანალიზატორი გამოიყენებს სახელსივრცეს, რომელიც განსაზღვრულია XAML-დოკუმენტის ფესვურ დესკრიპტორში.

XAML-დოკუმენტში სახელსივრცე მოიცემა xmlns ატრიბუტის საშუალებით. ზემოაღწერილ დოკუმენტში გამოცხადებულია ორი საბაზო სახელსივრცე:

- xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation - ესაა WPF-ის საბაზო სახელსივრცე, რომელიც მოიცავს WPF-ის ყველა კლასს, მართვის ელემენტების ჩათვლით. ისინი გამოიყენება მომხმარებლის ინტერფეისის ასაგებად. ვინაიდან სახელსივრცე ცხადდება პრეფიქსის გარეშე, იგი ვრცელდება მთელი XAML-დოკუმენტისთვის;
- xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" - ესაა XAML-ის სახელსივრცე. იგი შეიცავს XAML უტილიტების სხვადასხვა თვისებებს, რომლებიც გავლენას ახდენს იმაზე, თუ XAML-დოკუმენტი როგორ ინტერპრეტირდება. მოცემული სახელსივრცე აისახება x პრეფიქსზე. ეს პრეფიქსი შეიძლება მოთავსდეს ელემენტის სახელის წინ (მაგ., x:ელემენტის_სახელი).

მეორე სახელსივრცე გამოიყენება XAML-ის სპეციფიური ლექსემების („საკვანძო სიტყვები“) ჩასასმელად. 1.1 ცხრილში მოცემულია შედარებით ხშირად გამოყენებადი ასეთი სიტყვები.

XAML-ის საკვანძო სიტყვები ცხრ.1.1		
N	**საკვანძო სიტყვა**	**დანიშნულება**
1.	x:Array	წარმოადგენს .NET-ის მასივის ტიპს XAML-ზე
2.	x:Class	XAML-ფაილის კლასის სახელი
3.	x:ClassModifier	უზრუნველყოფს კლასის ტიპის ხილვადობის (internal ან public) განსაზღვრას, რომელიც Class საკვანძო სიტყვითაა აღნიშნული
4.	x:Code	პროგრამული კოდი შეიძლება უშუალოდ ჩაიდოს XAML-კოდში
5.	x:FieldModifier	უზრუნველყოფს ტიპის წევრის ხილვადობის (internal, public, private ან protected) განსაზღვრას ფესვის ნებისმიერი სახელმინიჭებული ელემენტისთვის (საკვანძო სიტყვით Name)
6.	x:Key	უზრუნველყოფს გასაღების მნიშვნელობის დაყენებას XAML ელემენტეისთვის, რომელიც უნდა მოთავსდეს ლექსიკონის ელემენტში
7.	x:Name	უზრუნველყოფს C#-ით გენერირებული სახელის მითითებას მოცემული XAML ელემენტისთვის
8.	x:Null	წარმოადგენს null-მიმთითებელს
9.	x:Shared	შესაძლებელია მხოლოდ იმ რესურსებისთვის, რომლებიც ერთხელ იძლევა ეგზემპლარს. ჩვეულებრივად, ყოველი წვდომისას იწარმოება რესურსის ახალი ეგზემპლარი
10.	x:Static	უზრუნველყოფს ტიპის სტატიკურ წევრზე მიმართვას
11.	x:Subclass	ეს კონსტრუქცია ქმნის წარმოებულ კლასს და გამოდგება დაპროგრამების მხოლოდ იმ ენებისთვის, რომელთაც არ აქვს დანაწევრებული (partielle) კლასების მხარდაჭერა

12.	x:Type	XAML-ეკვივალენტია C#-ია typeof ოპერაციის (იძახებს System.Type მითითებული სახელის საფუძველზე)
13.	x:TypeArgument	უზრუნველყოფს ელემენტის დაყენებას, როგორც განზოგადებული ტიპისას განსაზღვრული პარამეტრებით
14.	x:Uid	x:Name –ს პარალელურად შეუძლია მართვის ელემენტს ამ ატრიბუტით ცალსახა სახელი მიიღოს, რომელიც შემდგომიმთარგმანისთვის იქნება გამოყენებული
15	x:XData	ქმნის „მონაცემთა კუნძულს“ XAML-ის შიგნით, შეუძლია მარტივი XML-მონაცემთა კონსტრუქციით წარმოება
შენიშვნა: 2, 4, 9, 11, 14, 15		VS-2012 ვერსიაში დამატებული საკვანძო სიტყვები

WPF-დანართებში საბაზო სახელსივრცეთა გარდა იყენებენ ასევე სპეციალურს, რომლებიც არააუცილებელია:

- http://schemas.openxmlformats.org/markup-compatibility/2006 - XAML-ის სახელსივრცეა, დაკავშირებული ფორმატირების თავსებადობის პრობლემასთან სამუშაო გარემოსთან. ეს სახელსივრცე გამოიყენება XAML-ის სინტაკსური ანალიზატორის ინფორმირებისათვის იმის შესახებ, თუ რომელი ინფორმაციაა დასამუშავებელი და რომელი საიგნორირო;
- http://schemas.microsoft.com/expression/blend/2008 - XAML-ის სახელსივრცეა, რომელსაც აქვს მხარდაჭერა Expression Blend და Visual Studio პროგრამებიდან. გამოიყენება გვერდის გრაფიკული პანელის ზომების დასაყენებლად.

Window ობიექტის ატრიბუტებში შეიძლება დაემატოს შემდეგი XAML-აღწერები:

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="600"
```

მოცემული XAML-აღწერა აცხადებს არასავალდებულო სახელსივრცეებს პრეფიქსებით mc და d. თვისებები DesignHeight და DesignWidth იმყოფება სახელსივრცეში, რომელსაც აქვს პრეფიქსი d. ეს თვისებები განსაზღვრავს, რომ დანართის პროექტის დამუშავებისას Visual Studio დიზაინერში ფანჯარას უნდა ჰქონდეს ზომები 300x600. თვისება Ignorable მდებარეობს სახელსივრცეში, რომელიც აღნიშნულია პრეფიქსით mc და ის სინტაკსურ ანალიზატორს აინფორმირებს, რომ მან იგნორირება გაუკეთოს XAML-დოკუმენტის ნაწილს, რომელიც აღნიშნულია d პრეფიქსით.

WPF-დანართის XAML-დოკუმენტში ხშირად საჭიროა განხორციელდეს წვდომა პროექტის სხვა რომელიმე სახელსივრცესთან. ამ დროს აუცილებელია ახალი პრეფიქსის განსაზღვრა და მიეცეს სახელსივრცე. თუ პროექტში არის სახელსივრცე MyFirstWpfProject.Commands, მაშინ მის მიერთებას WPF -დანართის XAML-დოკუმენტთან ექნება შემდეგი სახე (command - გამოიყენება პრეფიქსის სახით).

```
xmlns:command="clr-namespace:
MyFirstWpfProject.Commands"
```

პრეფიქსი (command) გამოიყენება მიმართვისთვის სახელსივრცეზე XAML-დოკუმენტში. clr-namespace ლექსემს ენიჭება სახელსივრცის დასახელება .NET ნაკრებში.

XAML-დოკუმენტში კლასის აღსაწერად გამოიყენება ატრიბუტი Class. XAML-დოკუმენტის სტრიქონი

```
<Window x:Class="MyFirstWpfProject.MainWindow" ...>
```

ითვალისწინებს MyFirstWpfProject.MainWindow კლასის შექმნას Window კლასის ბაზაზე. Class ატრიბუტის x პრეფიქსი განსაზღვრავს იმას, რომ ეს ატრიბუტი თავსდება XAML-ის სახელსივრცეში.

MainWindow კლასი გენერირდება ავტომატურად კომპილაციის დროს. კლასის ნაწილისთვის ავტომატურად გენერირდება კოდი (ნაწილობრივი (partial) კლასი):

```
namespace MyFirstWpfProject
{
  // <summary>
  // ურთიერთქმედების ლოგიკა MainWindow.xaml - ისთვის
  // </summary>
  public partial class MainWindow : Window
   {
    public MainWindow()
     {
       InitializeComponent();
     }
   }
}
```

როდესაც სრულდება დანართის კომპილაცია, XAML-ფაილი, რომელიც განსაზღვრავს მომხმარებლის ინტერფეისს (MainWindow.xaml), ტრანსლირდება CLR ტიპის გამოცხადებაში, რომელიც ერთიანდება დანართის ლოგიკასთან გამოყოფილი კოდის კლასის ფაილიდან (MainWindow.xaml.cs).

InitializeComponent() მეთოდი გენერირდება დანართის კომპილაციის დროს და საწყის კოდში არ თავსდება.

XAML-დოკუმენტში აღწერილი მართვის ელემენტების პროგრამულად სამართავად, აუცილებელია მართვის ელემენტს მიეცეს XAML ატრიბუტი Name. მაგალითად, Grid ელემენტისთვის ჩაიწერება ასე:

```
<Grid Name="grid">

</Grid>
```

მარტივი თვისებები XAML-დოკუმენტში მოიცემა შემდეგი სინტაკსის შესაბამისად:

თვისების_სახელი = "მნიშვნელობა"

მაგალითად, **Name = "grid1"**

თვისების მისაცემად, რომელიც არის სრულფასოვანი ობიექტი, გამოიყენება *რთული თვისებები* „თვისება-ელემენტი“ სინტაკსის შესაბამისად:

მშობელი.თვისების_სახელი

მაგალითად, StackPanel კონტეინერისთვის აუცილებელია მიეცეს გრადიენტული ფუნჯი პანელის შესავსებად, რაც განისაზღვრება Background ატრიბუტით. იგი რეალიზდება დესკრიპტორებით:

<StackPanel.Background> . . . </StackPanel.Background>.

თვისების მნიშვნელობის მისაცემად გამოყოფილი კლასიდან გამოიყენება ფორმატირების გაფართოება, რომელიც უზრუნველყოფს XAML გრამატიკის გაფართოებას ახალი ფუნქციონალობით. ფორმატირების გაფართოება შეიძლება გამოყენებულ იქნას ჩალაგებულ დესკრიპტორებში ან XAML-ატრიბუტებში. როცა იყენებენ ატრიბუტებს, მაშინ აუცილებელია ფიგურული ფრჩხილების {...} გამოყენება.

ფორმატირების გაფართოებები იყენებს შემდეგ სინტაკსს:

{ფორმატირების_გაფართოების_კლასი არგუმენტი}

ფორმატირების გაფართოებები რეალიზდება კლასებით, რომლებიც შვილობილია System.Windows.Markup.MarkupExtention კლასის. MarkupExtention საბაზო კლასს აქვს ProvideValue() მეთოდი, რომელიც იძლევა ატრიბუტისთვის საჭირო მნიშვნელობას. მაგალითად, იმისათვის, რომ Button-ობიექტის Foreground ატრიბუტს მიეცეს სტატიკური თვისება, რომელიც სხვა კლასშია განსაზღვრული, აუცილებელია შემდეგი XAML-აღწერის შექმნა:

```
<Button Foreground="{x:Static
SystemColors.ActiveCaptionBrush}"    />
```

კომპილაციის დროს სინტაკსური ანალიზატორი შექმნის Static Extention კლასის ეგზემპლიარს, შემდეგ გამოიძახებს ProvideValue() მეთოდს, რომელიც ამოიღებს საჭირო მნიშვნელობას და დააყენებს მას Foreground თვისებისთვის.

ფორმატირების გაფართოებები შეიძლება გამოყენებულ იქნას როგორც ჩალაგებული თვისებები.

მიერთებული თვისებები აღწერს თვისებებს, რომელთა გამოყენება შეიძლება რამდენიმე მართვის ელემენტთან, ოღონდ რომლებიც განსაზღვრულია სხვა კლასში. WPF-დანართებში

მიერთებული თვისებები ხშირად გამოიყენება ინტერფეისის ელემენტების დაკომპლექტების სამართავად. მიერთებული თვისებების სინტაკსი შემდეგია:

განმსაზღვრელი_ტიპი.თვისების_სახელი

მაგალითად, თუ საჭიროა ღილაკის მოთავსება ბადის 0-ოვან სტრიქონში, მაშინ აუცილებელია შემდეგი XAML აღწერის შექმნა:

```
<Button ... Grid.Row="0" >
....
</Button>
```

აქ მიერთებული თვისებაა Grid.Row, ანუ Grid-ელემენტის Row-თვისება, რომელიც არაა Button ობიექტის თვისება. თვისება Row მიეკუთვნება Button ობიექტის თვისებებს, ვინაიდან ეს ობიექტი განთავსებულია Grid კონტეინერში.

ობიექტის ატრიბუტები შეიძლება გამოყენებულ იქნას მოვლენათა დამმუშავებლების მისაერთებლად, შემდეგი სინტაკსის გამოყენებით:

მოვლენის_სახელი = "მოვლენის_დამმუშავებლის_მეთოდის_სახელი"

მაგ., ღილაკის Click მოვლენისთვის (მისი დაჭერისას), შეიძლება დაყენდეს მოვლენის დამმუშავებელი Exit_Click.

```
<Button Name="Exit" Content="გამოსვლა"
        Click="Exit_Click" />
```

XAML-აღწერაში Exit_Click დამმუშავებლის განსაზღვრისას, აუცილებელია კლასის კოდში გვქონდეს მეთოდი კორექტული სიგნატურით. ქვემოთ მოყვანილია კოდი, რომელიც გენერირდება ავტომატურად მოვლენის დამმუშავებლის აღწერის შექმნისას XAML-დოკუმენტში.

```
private void Exit_Click(object sender,RoutedEventArgs e)
        {
         . . .
        }
```

1.13. WPF აპლიკაციის შექმნა

კორპორაციული აპლიკაცია (დანართი) პროგრამაა, რომელიც რეალიზაციას უკეთებს განსაზღვრულ ბიზნესამოცანას (ბიზნესფუნქციას). დანართი უნდა მუშაობდეს მონაცემებთან, რომლებიც ინახება საინფორმაციო სისტემის მონაცემთა ბაზაში.

დანართის არქიტექტურა მოიცავს **წარმოდგენის შრეს, ბიზნესლოგიკის შრეს** და **მონაცემთა შრეს.** დანართის თითოეული შრის ფუნქციონალობა ბევრადაა დამოკიდებული საინფორმაციო სისტემის საგნობრივ სფეროზე, თუმცა არსებობს აგრეთვე ზოგადი, ფუძემდებლური ფუნქციები, რომლებიც ახასიათებს პრაქტიკულად ნებისმიერ კორპორაციულ დანართს.

ამგვარად, აპლიკაციაში უნდა დამუშავდეს წარმოდგენის შრე, რომელიც უზრუნველყოფს მომხმარებლის ინტერფეისს სისტემასთან. ინტერფეისი შეიძლება შეიქმნას Windows-ფანჯრებით და WPF გვერდებით, რომლებიც შეივსება მართვის სხვადასხვა ვიზუალური ელემენტებით [5].

მართვის ელემენტები უნდა უზრუნველყოფდეს სისტემის ფუნქციონალობის ვიზუალურ წარმოდგენას მომხმარებლისათვის, აწარმოებდეს შესატანი მონაცემების ვერიფიკაციას და ურთიერთქმედებდეს ბიზნესკლასებთან.

ბიზნესლოგიკის შრე უნდა უზრუნველყოფდეს დანართის ძირითად ფუნქციონალობას: დააფორმიროს ბიზნესკლასები, რეალიზება გაუკეთოს მონაცემთა დამუშავების ალგორითმებს, უზრუნველყოს მონაცემებთან მიერთება და მათი კეშირება. ამ შრის რეალიზაცია შეიძლება განხორციელდეს კლასების საფუძველზე, რომლებიც ბიზნესლოგიკას არეალიზებენ ინტერფეისული ელემენტების კლასთა მეთოდებით, ან მონაცემთა მოდელის კლასთა მეთოდებით.

მონაცემთა შრე უნდა უზრუნველყოფდეს დანართის ურთიერთქმედებას ბაზის მონაცემებთან. კორპორაციულ აპლიკაციებში ამისათვის ყველაზე მიზანშეწონილია გამოყენებულ იქნას პლატფორმა ADO.NET Entity Framework და მოდელი EDM

(Entity Data Model). EDM მოდელი აღწერს მონაცემთა სტრუქტურას ფიზიკური შენახვის ფორმისგან დამოუკიდებლად.

კორპორაციული დანართების დაპროექტების საკითხების შესასწავლად განვიხილოთ ძირითადი მიდგომები ინფორმაციული სისტემის ცალკეული ფუნქციების ასაგებად, რომელიც უზრუნველყოფს კომპანიის თანამშრომელთა მონაცემების დამუშავებას.

მაგალითისთვის აქ გამოვიყენებთ მონაცემთა ბაზას TitlePresonal, ცხრილებისა და ველების მცირე რაოდენობით, ხოლო აპლიკაციის ფუნქციონალობა ითვალისწინებს კომპანიის თანამშრომელთა მონაცემების შეტანას, კორექტირებას და წაშლას. ასაგები დანართი უნდა უზრუნველყოფდეს შემდეგი მონაცემების შენახვას და გადამუშავებას: *გვარი, სახელი, სქესი, დაბადების_თარიღი, თანამდებობა, ტელეფონი, ელ_ფოსტა.*

აპლიკაციის ფუნქციებია:

- თანამშრომელთა მონაცემების დათვალიერება;
- ახალი თანამშრომლის მონაცემთა შეტანა;
- თანამშრომლის მონაცემთა რედაქტირება;
- თანამშრომლის მონაცემების წაშლა;
- მონაცემთა მოძებნა თანამშრომლის შესახებ.

შევქმნათ .NET Framework 4 გარემოში ახალი დანართის WPF-პროექტი სახელით WpfApplProject.

1.14. WPF აპლიკაცია მონაცემთა ბაზებით

განიხილება „არსთა-დამოკიდებულების“ მოდელის ძირითადი დებულებები, მისი საბაზო კომპონენტები: არსები (ობიექტები), ასოციაციები (კავშირები) და თვისებები (ატრიბუტები). სასწავლო მაგალითზე ნაჩვენებია მოდელის აგების პროცესი არსებული ბაზის გამოყენებით. აგებული PageEmployee დანართის გვერდისა და მონაცემთა მოდელისთვის ხორციელდება მონაცემთა მიბმა ინტერფეისის ელემენტებთან: ტექსტბოქსის, კომბობოქსის, ლისტბოქსის და თარიღის. განიხილება დანართის ურთიერთქმედების ოპერაციების დაპროექტების საკითხები

მონაცემთა ბაზასთან: მონაცეთა რედაქტირება, დამატება და წაშლა. აღიწერება მონაცემთა შემოწმების შესაძლებლობა მათი შეტანისას, მომხმარებლელთა შემოწმების წესების გამოყენებით.

Entity Data Model (EDM) - არსთა მონაცემთა მოდელი („არსთა-კავშირების" მოდელი). **EDM** მოდელი წარმოადგენს ძირითად ცნებათა ერთობლიობას, რომელიც აღწერს მონაცემთა სტრუქტურას მისი კომპიუტერის მეხსიერებაში ფიზიკურად შენახვის ფორმისგან დამოუკიდებლად. EDM მოდელში აღწერილ მონაცემებს შეიძლება ჰქონდეს განსხვავებული სტრუქტურები: რელაციური, ტექსტური ფაილების, XML-ის, ელექტრონული ცხრილების და რეპორტების. EDM მოდელი აღწერს მონაცემთა სტრუქტურას არსებისა და კავშირების საფუძველზე, რომლებიც დამოუკიდებელია შენახვის სქემებისგან. ასეთი მიდგომის საფუძველზე მონაცემთა ფიზიკური დამახსოვრება განცალკევებულია დანართისაგან და არ მოქმედებს მის დამუშავებაზე. ამის უზრუნველყოფა ხდება იმის გამო, რომ არსები და კავშირები აღწერს მონაცემთა სტრუქტურებს ისე, როგორც ეს სჭირდება დანართს. EDM მოდელი კონცეპტუალური მოდელია, რომელიც აღწერს მონაცემთა სტრუქტურებს არსების და კავშირების სახით.

EDM მოდელი იყენებს სამ ძირითად ცნებას მონაცემთა სტრუქტურის აღსაწერად:

- არსის ტიპი;
- ასოციაციის ტიპი;
- თვისება.

არსის ტიპი გამოიყენება მონაცემთა სტრუქტურის აღსაწერად EDM მოდელის დახმარებით. კონცეპტუალურ მოდელში არსთა ტიპები აიგება თვისებებისგან და აღწერს ზედა დონის ძირითად კონცეპტუალურ ელემენტთა სტრუქტურას, როგორიცაა მაგალითად, თანამშრომლები და როლები. არსის ტიპი არის შაბლონი არსებისთვის. არსი არის განსაზღვრული ობიექტი (მაგალითად, რომელიმე თანამშრომელი და მისი როლი ბიზნეს-პროცესში). ყოველ არსს უნდა ჰქონდეს უნიკალური გასაღები არსთა ერთობლიობის შიგნით. არსთა ერთობლიობა არის

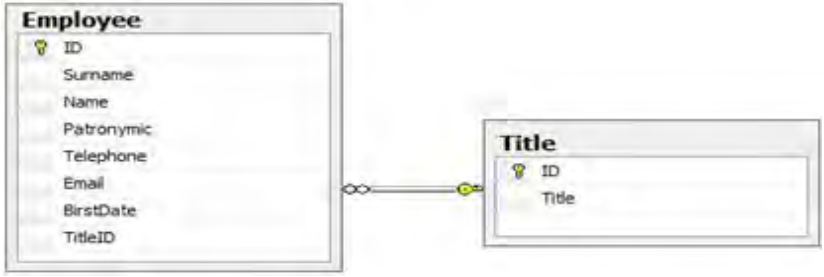
განსაზღვრული ტიპის არსის ეგზემპლართა კოლექცია. არსთა ერთობლიობები (და ასოციაციათა ერთობლიობები) ლოგიკურად დაჯგუფებულია არსთა კონტეინერებში.

ასოციაციის ტიპი გამოიყენება კავშირთა აღწერის ასაგებად EDM მოდელში. კონცეპტუალურ მოდელში ასოციაცია ასახავს კავშირს ორი ტიპის არსებს შორის. ყოველ ასოციაციას აქვს ორი ბოლო წერტილი, რომლებიც განსაზღვრავს არსთა ტიპებს. ისინი მონაწილეობს ასოციაციაში. ყოველი ბოლო წერტილი განსაზღვრავს მის ჯერადობას, რაც მიუთითებს არსების შესაძლო რაოდენობაზე ასოციაციის ამ ბოლო წერტილში.

არსთა ტიპები შეიცავს თვისებებს, რომლებიც განსაზღვრავს მათ სტრუქტურას და მახასიათებლებს. მაგალითად, არსის ტიპს „თანამშრომელი“ შეიძლება ჰქონდეს თვისებები: **ID, გვარი, სახელი, სქესი, დაბ_თარიღი, თანამდებობა, ტელეფონი, ელ_მისამართი** და ა.შ.

თვისებები კონცეპტუალურ მოდელში ანალოგიურია პროგრამული აპლიკაციის კლასებისა ან მონაცემთა ბაზების ატრიბუტების. თვისებები შეიძლება შეიცავდეს პრიმიტიულ მონაცემებს (სტრიქონი, მთელი რიცხვი, თარიღი, ლოგიკური მნიშვნელობა) ან სტრუქტურირებულ მონაცემებს (რთული ტიპი).

კონცეპტუალური მოდელი არის მონაცემთა სტრუქტურის სპეციფიკური ასახვა არსებისა და კავშირების სახით. 1.11 ნახაზზე ნაჩვენებია კონცეპტუალური მოდელის გამოსახვა სქემით, ბაზისათვის TitlePersonal – „თანამშრომელი“, ორი ტიპის არსით Employee – თანამშრომელი და Title –როლი/თანამდებობა), და ასოციაციური კავშირით 1:* (ერთი:მრავალთან).



ნახ.1.11. მონაცემთა ბაზის კონცეპტუალური მოდელი

Employee ცხრილი შეიცავს თანამშრომლის მონაცემებს. მისი ატრიბუტებია:

- ID – თანამშრომლის იდენტიფიკატორი;
- Surname – გვარი;
- Name – სახელი;
- Sex –სქესი;
- BirstDate – დაბადების თარიღი;
- Telephone – ტელეფონი;
- Email – ელ_მისამართი;
- TitleID – გარე გასაღები Title ცხრილთან დასაკავშირებლად.

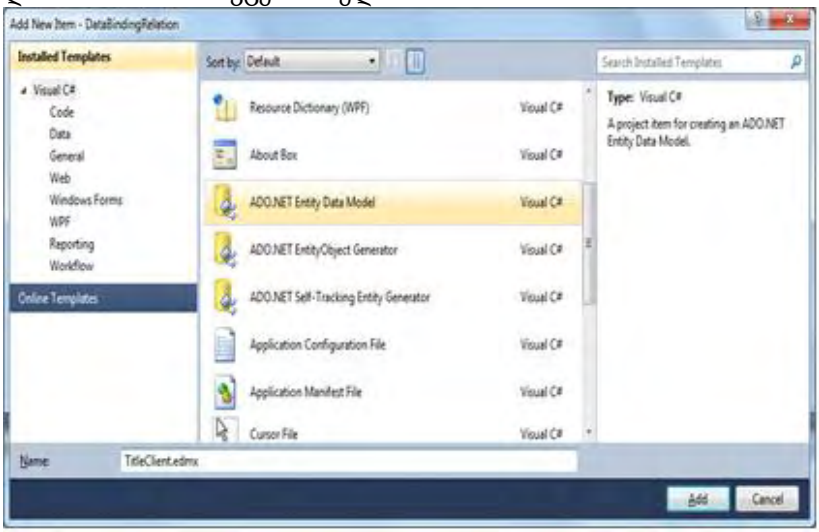
Title ცხრილი თანამდებობათა ცნობარია, რომელიც ამ ორგანიზაციას აქვს. იგი შეიცავს შემდეგ ატრიბუტებს:

- ID – თანამდებობის კოდი;
- Title – თანამდებობის დასახელება.

EDM-მოდელის შესაქმნელად პროექტში Solution Explorer-დან ჩავამატოთ ახალი ელემენტი (ნახ.1.12):

Add -> NewItem -> ADO.NET EDM

და File Name-ში მივცეთ სახელი: TitleClient.

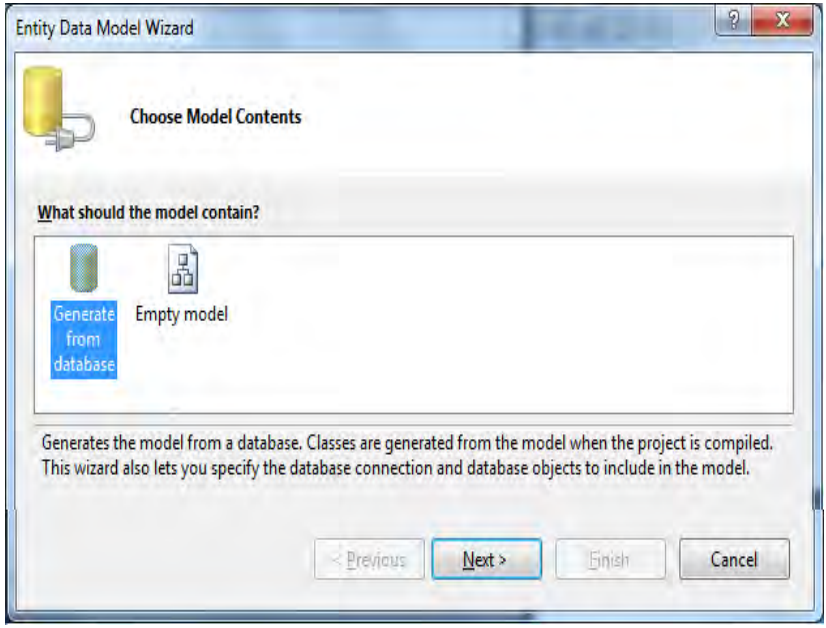


ნახ.1.12. პროექტში EDM მოდელის დამატება

EDM მოდელის შექმნისას 1.13 ნახაზზე ნაჩვენებ Wizard-ში მივუთითოთ „შექმნა მონაცემთა ბაზიდან“, რის შემდეგაც გამოვა 1.14 ნახაზზე ნაჩვენები ფანჯარა. აქ, მონაცემთა ბაზასთან მისაერთებლად, უნდა მივცეთ:

სერვერის_სახელი.მონაცემთა_ბაზის_სახელი

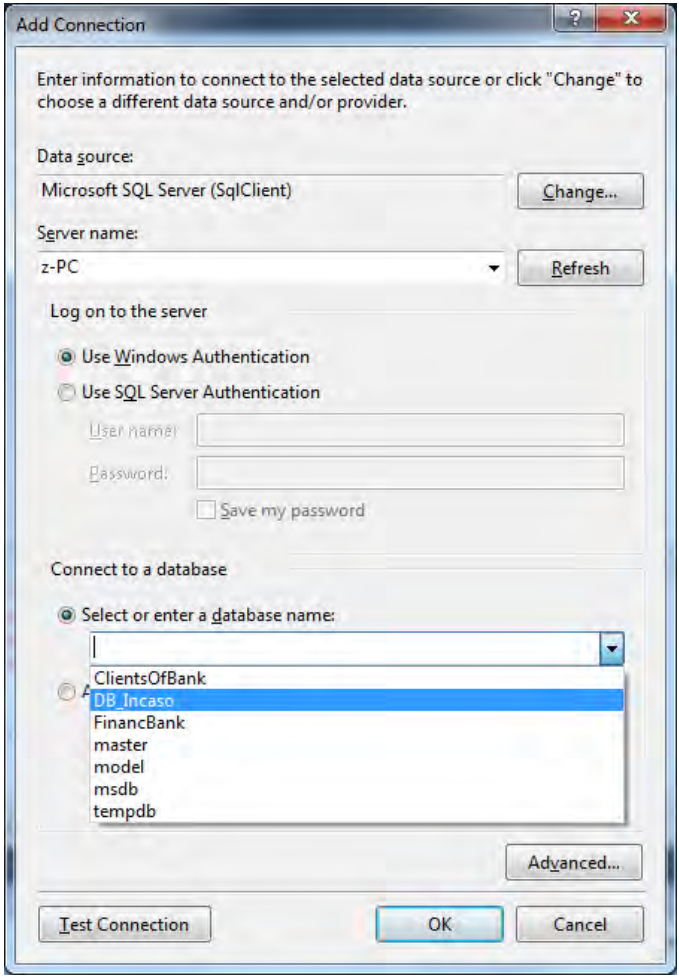
და მონაცემთა მოდელის სახელი - TitlePersonEntities (მიერთების პარამეტრების შენახვა Config-ში).



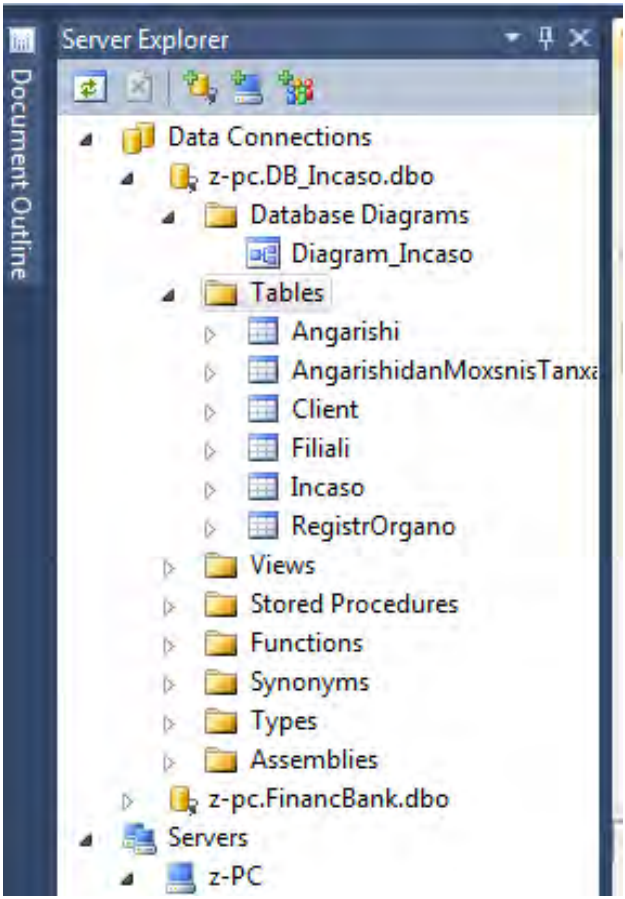
ნახ.1.13. EDM მოდელის შექმნა მონაცემთა ბაზიდან

Next ღილაკით გამოიტანება 1.15 ფანჯარა, რომლის ჩეკბოქსშიც ვირჩევთ მონაცემთა ბაზის ობიექტებს (მაგალითად,

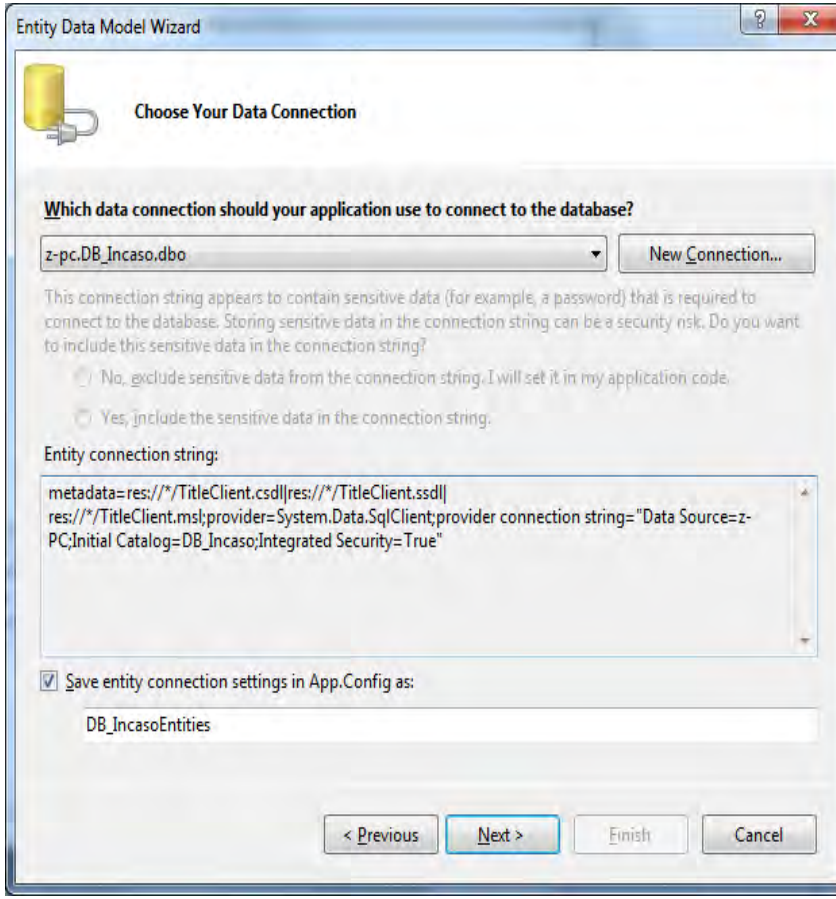
Employee და Title), მივუთითებთ ობიექტების ფორმირებას მხოლობით ან მრავლობით რიცხვში.



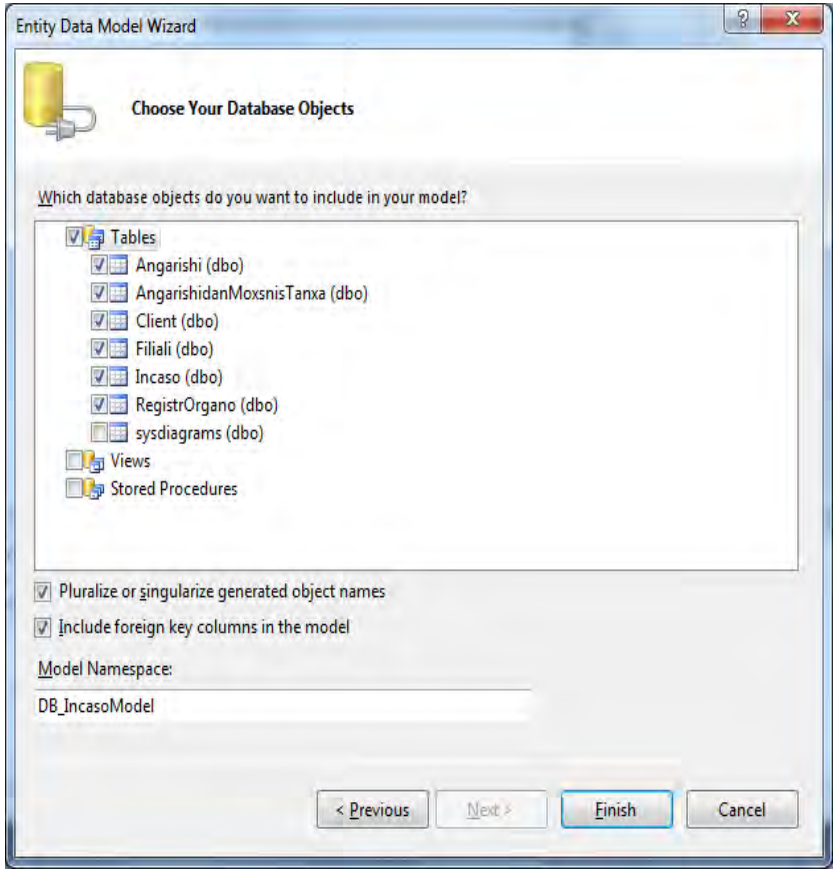
ნახ.1.14-ა. მონაცემთა ბაზასთა მიერთება: ეტაპი_1



ნახ.1.14-ბ. მონაცემთა ბაზასთა მიერთება: ეტაპი_ 2

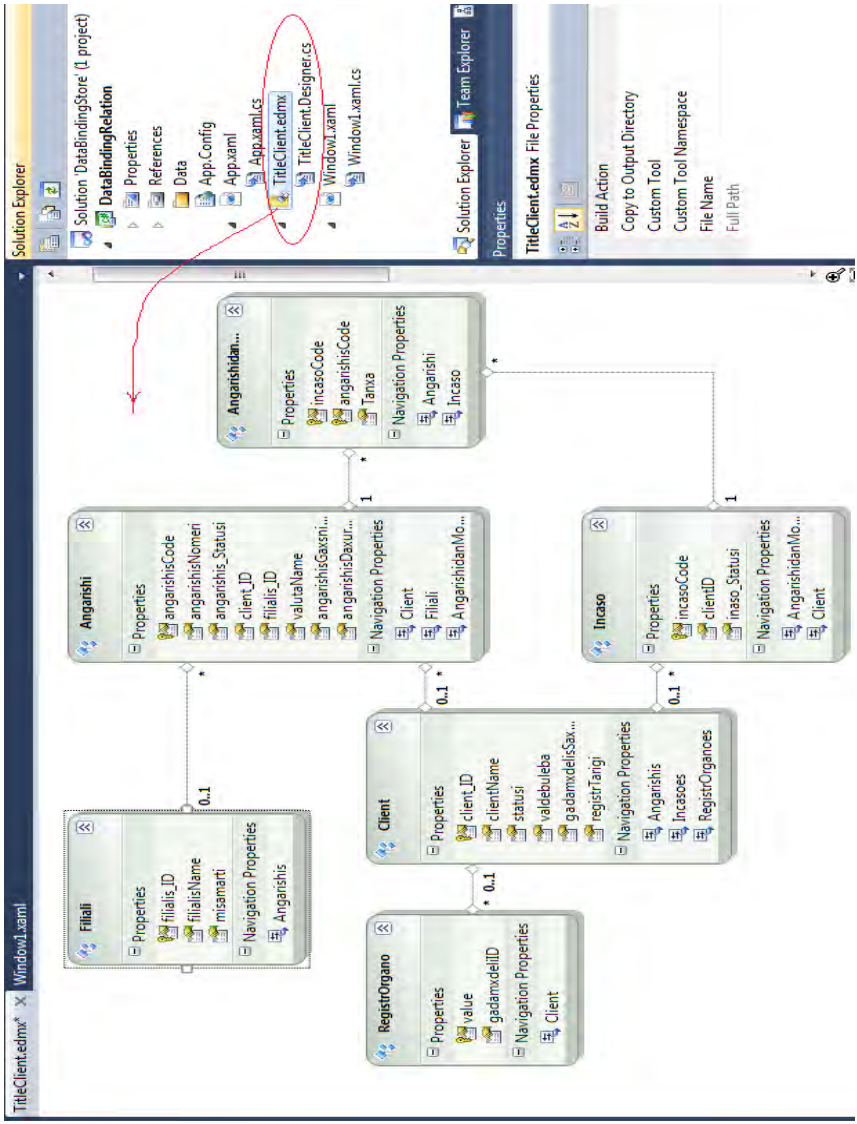


ნახ.1.14-გ. მონაცემთა ბაზასთა მიერთება: ეტაპი_3



ნახ.1.15. მონაცემთა ბაზის ცხრილების არჩევა

EDM მოდელის შექმნის შედეგად პროექტში დამატებული იქნება TitleEmployee.edmx ფაილი (ნახ.1.16). მიმართებები საჭირო ბიბლიოთეკებზე და კონფიგურაციის ფაილი.



ნახ.1.1.16. პროექტი EDM მოდელით

ავტომატურად გენერირებული კლასი `DB_IncasoEntities`, რომელიც მემკვიდრეა ObjectContext კლასის, ასახავს `TitleClient` მონაცემთა ბაზის არსებს, შეიცავს თვისებებს, რომლებიც ამოდელირებს Tanamshromeli და Tanamdeboba ცხრილებს, კავშირებს მათ შორის.

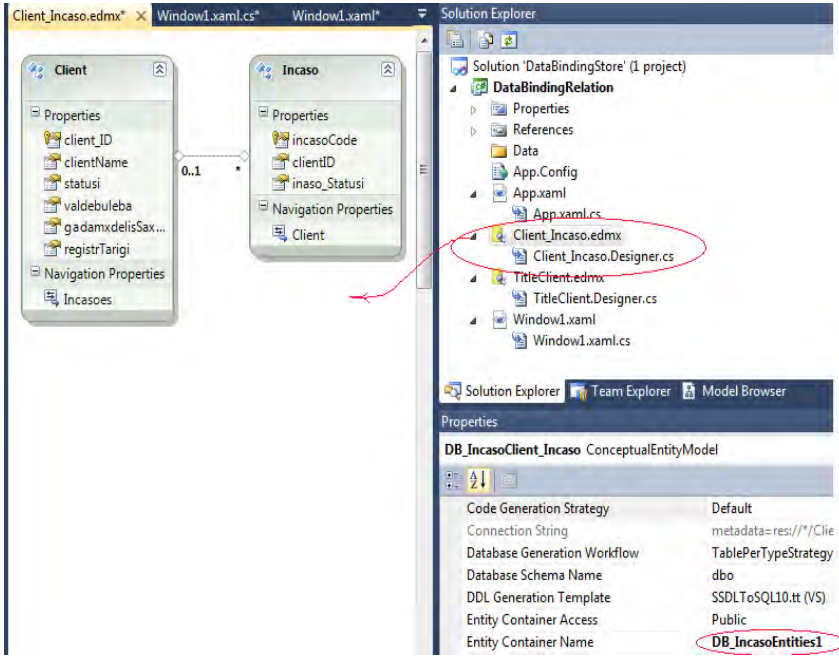
მონაცემთა მოდელის შექმნისას პროექტში ავტომატურად მოხდა კონფიგურაციის App.Config ფაილის შექმნა, რომელიც შეიცავს მონაცემთა ბაზასთან მიერთების სტრიქონს.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="DB_IncasoEntities"
      connectionString="metadata=&amp;quot;res://*/
      TitleClient.csdl|&amp;#xD;&amp;#xA;
      res://*/TitleClient.ssdl|res://*/
      TitleClient.msl&amp;quot;
      provider=System.Data.SqlClient; provider connection
      string=&amp;quot;Data Source=z-PC;Initial
      Catalog=DB_Incaso;&amp;#xD;&amp;#xA;
      Integrated
      Security=True;MultipleActiveResultSets=True&amp;quot;"
      providerName="System.Data.EntityClient" />
     <add name="DB_IncasoEntities1"
      connectionString="metadata=res://*/
      Client_Incaso.csdl|res://*/Client_Incaso.ssdl|res://*/
      Client_Incaso.msl;provider=System.Data.SqlClient;
      provider connection string=&amp;quot;
      Data Source=z-PC;Initial Catalog=DB_Incaso;Integrated
      Security=True;MultipleActiveResultSets=True&amp;quot;"
      providerName="System.Data.EntityClient" />
  </connectionStrings>
</configuration>
```

Solution Explorer–ში Edm მოდელის C#-ის TitleClient.Designer.cs -კოდის ფრაგმენტი ასეთი სახისაა:

```
// —— ლისტინგი_EDM მოდელი ————
using System;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using System.Data.EntityClient;
using System.ComponentModel;
using System.Xml.Serialization;
using System.Runtime.Serialization;

[assembly: EdmSchemaAttribute()]
#region EDM Relationship Metadata

[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_Angarishi_Client",
   "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
   typeof(DataBindingRelation.Client), "Angarishi",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.Angarishi), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_Angarishi_Filiali",
   "Filiali",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
   typeof(DataBindingRelation.Filiali), "Angarishi",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.Angarishi), true)]
```

```
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_AngarishidanMoxsnisTanxa_Angarishi",
   "Angarishi", System.Data.Metadata.Edm.RelationshipMultiplicity.One,
   typeof(DataBindingRelation.Angarishi), "AngarishidanMoxsnisTanxa",
System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.AngarishidanMoxsnisTanxa), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_AngarishidanMoxsnisTanxa_Incaso",
   "Incaso", System.Data.Metadata.Edm.RelationshipMultiplicity.One,
   typeof(DataBindingRelation.Incaso), "AngarishidanMoxsnisTanxa",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.AngarishidanMoxsnisTanxa), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_Incaso_Client",
   "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
   typeof(DataBindingRelation.Client), "Incaso",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.Incaso), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_RegistrOrgano_Client",
   "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
   typeof(DataBindingRelation.Client), "RegistrOrgano",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.RegistrOrgano), true)]
#endregion
namespace DataBindingRelation
{
  . . . }
```



ნახ.1.17. EDM მოდელი: Client_Incaso

```
// Client_Incaso.Designer.cs ---------------------
using System;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using System.Data.EntityClient;
using System.ComponentModel;
using System.Xml.Serialization;
using System.Runtime.Serialization;

[assembly: EdmSchemaAttribute()]
#region EDM Relationship Metadata
```

```
[assembly: EdmRelationshipAttribute("DB_IncasoClient_Incaso",
"FK_Incaso_Client",
    "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
      typeof(DataBindingRelation.Client),
    "Incaso", System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
      typeof(DataBindingRelation.Incaso), true)]
#endregion

namespace DataBindingRelation
{
...
}
```

1.15. მართვის ელემენტების მიბმა მონაცემთა წყაროსთან

აპლიკაციის სამუშაოდ მონაცემთა ბაზასთან აუცილებელია Window1 კლასის კოდში გამოცხადდეს შექმნილი EDM მოდელის DataEntitiesEmployee მონაცემთა კონტექსტის სტატიკური თვისება. დანართის დაპროექტების ამ ეტაპზე ეს თვისება შეიძლება იყოს მხოლოდ ყველასთვის მისაწვდომი, ანუ - public. შემდგომში ამ თვისებას გამოიყენებს სხვა კლასები, ოღონდ PageEmployee კლასის ეგზემპლარის შექმნის გარეშე, ამიტომაც იგი უნდა იყოს სტატიკური.

```
public static TitlePresonalEntities DataEntitiesEmployee
      { get; set; }
public static DB_IncasoEntities1 DataEntitiesClient
     { get; set; }
```

ასევე აუცილებელია გამოცხადდეს ListEmployee კოლექცია დანართის სამუშაოდ Employee ობიექტთა კოლექციასთან.

```
public PageEmployee()
{
  InitializeComponent();
DataEntitiesEmployee = new TitlePersonalEntities();
ListEmployee = new ObservableCollection<Employee>();
}
```

```
public Window1()
{
    InitializeComponent();
    DataEntitiesClient = new DB_IncasoEntities1();
    ListClient = new ObservableCollection<Client>();
    public static DB_IncasoEntities1 DataEntitiesClient
      { get; set; }
}
```

დანართისთვის მონაცემთა ფორმირება ბაზიდან მოხდება PageEmployee გვერდის ჩატვირთვის დროს. ამისათვის XAML-დოკუმენტში Page დავამატოთ თვისება Loaded:

Loaded="Page_Loaded"

PageEmployee კლასის კოდში ჩავრთოთ დამმუშავებელი Page_Loaded.

```
private void Page_Loaded(object sender,
RoutedEventArgs e)
{
  ObjectQuery<Employee> employees =
                DataEntitiesEmployee.Employees;
  var queryEmployee = from employee in employees
                      orderby employee.Surname
                      select employee;
  foreach (Employee emp in queryEmployee)
    {
      ListEmployee.Add(emp);
    }
  DataGridEmployee.ItemsSource = ListEmployee;
}

private void Page_Loaded(object sender,
RoutedEventArgs e)
{
  ObjectQuery<Client> clients =
                DataEntitiesClient.Clients;
  var queryClient = from Client in clients
                      orderby client.clientName
                      select client;
  foreach (Client cln in queryClient)
```

```
    {
        ListClient.Add(cln);
    }
  DataGridClient.ItemsSource = ListClient;
}
```

clients ველს აქვს ObjectQuery<Client> ტიპი. ObjectQuery<T> კლასი არის მოხოვნა, რომელიც აბრუნებს ტიპიზირებულ არსთა კოლექციას ელემენტთა ნებისმიერი რაოდენობით. ავაგოთ მოთხოვნა Linq-ტექნოლოგიის საშუალებით:

```
var queryClient = from client in clients
            orderby client.clientName
            select client;
```

მოთხოვნის შედეგები მოვაწესრიგოთ „გვარებით“ (orderby client.clientName). შემდეგ ვაფორმირებთ ListClient კლიენტთა კოლექციას და მონაცემთა წყაროს DataGridClient ბადისთვის.

```
foreach (Client cln in queryClient)
    {
        ListClient.Add(cln);
    }
DataGridClient.ItemsSource = ListClient;
```

პროექტირების შედეგად დანართში ფორმირებულია კოლექცია მონაცემებით TitleClient ბაზის Client ცხრილიდან. შემდგომი ამოცანაა DataGridClient ბადის აწყობა მონაცემთა კორექტული ასახვისათვის. თავიდან მოდიფიცირება გავუკეთოთ DataGrid-ის ზოგად აღწერას.

```
<DataGrid Name="DataGridClient"
ItemsSource="{Binding}"
AutoGenerateColumns="False"
HorizontalAlignment="Left"
MaxWidth="1000" MaxHeight="295"
RowBackground="#FFE6D3EF"
AlternatingRowBackground="#FC96CFD4"
BorderBrush="#FF1F33EB"
BorderThickness="3"
IsReadOnly="True"
RowHeight="25"
Cursor="Hand">
```

- განვსაზღვროთ მიბმა მონაცემთა წყაროსთვის:
  `ItemsSource="{Binding}"`
- გავაუქმოთ სვეტების ავტომატური გენერაცია:
  `AutoGenerateColumns="False"`
- დავადგინოთ მიბმა გვერდის მარცხენა საზღვართან:
  `HorizontalAlignment="Left"`
- განვსაზღვროთ ბადის მაქსიმალური ზომები:
  `MaxWidth="1000" MaxHeight="295"`
- დავადგინოთ ბადის შევსების ძირითადი და ალტერნატიული ფერები:
  `RowBackground="#FFE6D3EF"`
  `AlternatingRowBackground="#FC96CFD4"`
- დავადგინოთ ბადის ჩარჩოსთვის ფერი და ხაზის სისქე:
  `BorderBrush="#FF1F33EB"`
  `BorderThickness="3"`
- დავადგინოთ ბადის სტრიქონთა სიმაღლე:
  `RowHeight="25"`
- შევცვალოთ კურსორის ფორმა მაუსის ისრის მიტანისას DataGridEmployee ცხრილზე: Cursor="Hand"

1.16. მარშრუტიზირებადი მოვლენები

WPF-აპლიკაციები იერერქიული ბუნებისაა. მათ აქვს მართვის ელემენტები, რომლებიც თვითონ შეიცავს სხვა მართვის ელემენტებს, რომელთაც ასევე აქვს სხვა მართვის ელემენტები და ა.შ. [3,5]. მარშრუტიზირებადი მოვლენა არის ისეთი მექანიზმი, რომელიც საშუალებას იძლევა, რომ მოვლენა, რომელსაც აქვს გავლენა იერარქიის ერთ ელემენტზე, ჰქონდეს ასევე გავლენა ამავე იერარქიის სხვა ელემენტებზე, და ამასთანავე არ იყოს საჭირო რთული კოდის დაწერა.

ამის საუკეთესო მაგალითია სიტუაცია, როდესაც მომხმარებლებს ეძლევათ საშუალება დანართში იმუშაონ მაუსის დახმარებით, რაც ძალზე ხშირია. როდესაც მომხმარებელი კლიკავს ღილაკს დანართში, ჩვეულებისამებრ საჭიროა, რომ დანართმა როგორღაც იმოქმედოს ამ დაკლიკვის მოვლენაზე. ერთ-ერთი ვარიანტი, რომელიც ცნობილია Windows Form და ASP.NET დანართებიდან, არის ამ ღილაკისთვის მოვლენის დამმუშავებლის კოდის დაწერა (Button_Click(…) {..}), რომელშიც მითითებულია ის ქმედებები, რომლებიც უნდა შესრულდეს ამ ღილაკის დაკლიკვის საპასუხოდ.

ასეთი მიდგომა ერთგვარად ზღუდავს დამმუშავებლის შესაძლებლობებს და ხშირად Windows Form-ში ქმნის საკმაოდ რთულ, გაურკვეველ კოდს. მარტივი მაგალითისთვის დავუშვათ, რომ ფორმაზე ღილაკია. ასეთ დროს რომელმა მართვის ელემენტმა უნდა იმოქმედოს დაკლიკვაზე და შექმნას მოვლენა - ფანჯარამ თუ ღილაკმა, თუ ორივემ ერთად ? თუ არსებობს მოთხოვნა, რომ მოვლენა შექმნას ორივემ და ამასთანავე გარკვეული თანამიმდევრობით, მაშინ Windows Form დანართში უნდა დაიწეროს სპეციალური, საკმაოდ რთული კოდი.

WPF-ში მაუსის ღილაკის დაკლიკვა მართვის ელემენტებისთვის (მათ შორის Button და Window) რეალიზდება მარშრუტიზირებადი მოვლენის სახით, რაც სრულად გამორიცხავს ზემოხსენებულ პრობლემას. მარშრუტიზირებადი მოვლენები გენერირდება ყველა ობიექტის მიერ განსაზღვრული მიმდევრობით იერარქიაში. ამავე დროს დამმუშავებელს ეძლევა სრული კონტროლის საშუალება იმაზე, თუ როგორ იმოქმედოს მათზე.

მაგალითად, განვიხილოთ მართვის ელემენტი Window, რომელიც შეიცავს Grid ელემენტს, რომელიც თავის მხრივ შეიცავს Rectangle ელემენტს (ნახ.1.18) [4].

ნახ.1.18

თუ შესრულდა დაკლიკვა Rectangle მართვის ელემენტზე, მაშინ ადგილი ექნება შემდეგი მიმდევრობის მოვლენებს:

1. მაუსის ღილაკის დაკლიკვის მოვლენის გენერაცია Window-ში.
2. მაუსის ღილაკის დაკლიკვის მოვლენის გენერაცია Grid-ში.
3. მაუსის ღილაკის დაკლიკვის მოვლენის გენერაცია Rectangle-ში.

ამით პროცესი არ მთავრდება და შემდეგ მოხდება:

4. მაუსის ღილაკის დაკლიკვის კიდევ ერთი მოვლენის გენერაცია Rectangle-ში.
5. მაუსის ღილაკის დაკლიკვის კიდევ ერთი მოვლენის გენერაცია Grid-ში.
6. მაუსის ღილაკის დაკლიკვის კიდევ ერთი მოვლენის გენერაცია Window-ში.

დამმუშავებელს შეუძლია მოვლენაზე რეაგირება აღნიშნული მიმდევრობის ნებისმიერ წერტილში, ანუ უბრალოდ დაამატოს მოვლენის დამუშავების შესაბამისი მეთოდი.

მას შეუძლია ასევე ამ მიმდევრობის შეწყვეტა მოვლენის დამუშავების ჩარჩოს ნებისმიერ წერტილში.

XAML-კოდს აქვს ასეთი სახე:

```
<Window x:Class="Ch34Ex02.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Routed Events" Height="400" Width="800"
```

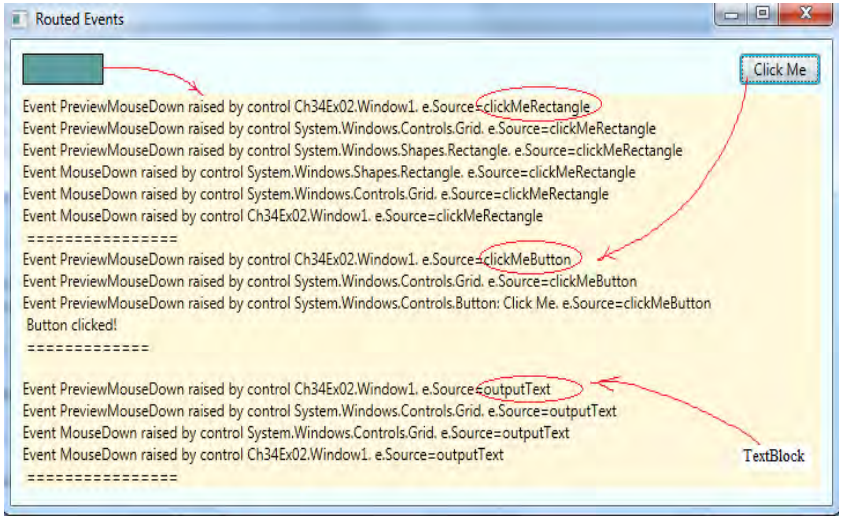
```
      MouseDown="Generic_MouseDown"
      PreviewMouseDown="Generic_MouseDown"
      MouseUp="Window_MouseUp"   >

<Grid Name="contentGrid" MouseDown="Generic_MouseDown"
   PreviewMouseDown="Generic_MouseDown" Background="Azure">
  <Grid.RowDefinitions>
    <RowDefinition Height="332*" />
    <RowDefinition Height="29*" />
  </Grid.RowDefinitions>
    <Rectangle Name="clickMeRectangle"   Margin="10,10,0,0" Height="23"
           HorizontalAlignment="Left" VerticalAlignment="Top"
              Width="70" Stroke="Black"
           MouseDown= "Generic_MouseDown" PreviewMouseDown=
                    "Generic_MouseDown"
           Fill="CadetBlue" />
  <Button Name="clickMeButton" Margin="0,10,10,0"  Height="23"
          HorizontalAlignment="Right" VerticalAlignment="Top"
             Width="70"
          MouseDown="Generic_MouseDown"
          PreviewMouseDown="Generic_MouseDown"
          Click="clickMeButton_Click">Click Me</Button>
      <TextBlock  Name="outputText" Margin="10,40,10,10"
                  Background="Cornsilk" />
  </Grid>
</Window>
```

3. შევცვალოთ კოდი Window1.xaml.cs ფაილში შემდეგი სახით:

```
//— ლისტინგი_ Window1.xaml.cs ————————
using System; using System.Collections.Generic;
using System.Linq; using System.Text;
using System.Windows; using System.Windows.Controls;
using System.Windows.Data; using System.Windows.Documents;
using System.Windows.Input; using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation; using System.Windows.Shapes;
using System.Windows.Media.Animation;   // დაემატა
namespace Ch34Ex02
{
```

```
public partial class Window1 : Window
{
 private void Generic_MouseDown(object sender, MouseButtonEventArgs e)
    {
      outputText.Text = string.Format("{0}Event {1} raised by control {2}.
e.Source={3}\n",
         outputText.Text, e.RoutedEvent.Name, sender.ToString(),
((FrameworkElement)e.Source).Name);
     }
 private void Window_MouseUp(object sender,
                                          MouseButtonEventArgs e)
 {
    outputText.Text = string.Format("{0} =======\n", outputText.Text);
 }
 private void clickMeButton_Click(object sender, RoutedEventArgs e)
  {
  outputText.Text = string.Format("{0} Button clicked!\n==n\n", outputText.Text);
  }
} }
```

4. ავამუშავოთ დანართი, შედეგი ნაჩვენებია 1.19 ნახაზზე.:



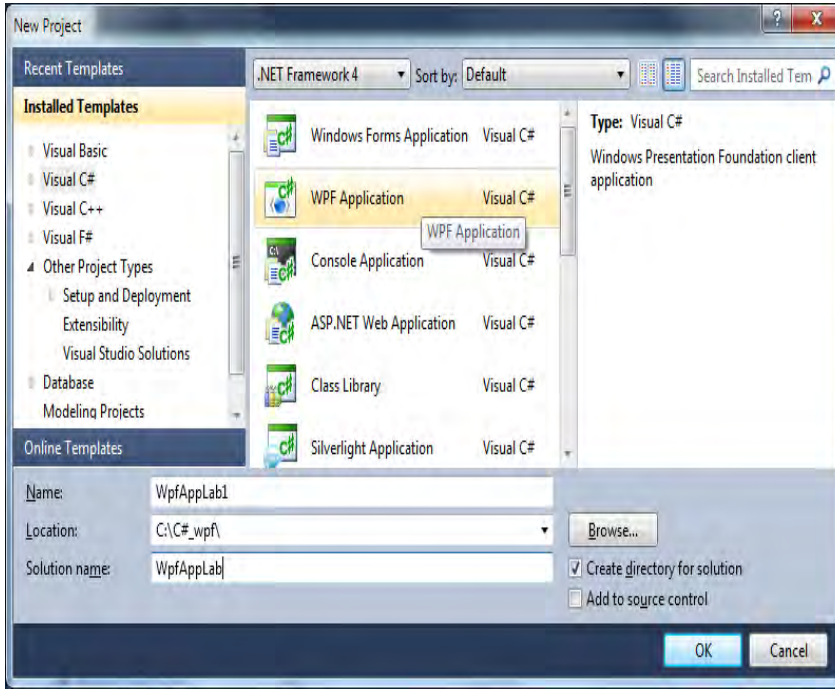
ნახ.1.19

II თავი. პრაქტიკული ნაწილი: WPF-ტექნოლოგიის ვიზუალური ელემენტები და მათი გამოყენება აპლიკაციების ასაგებად

2.1. Windows Presentation Foundation ტექნოლოგის სამუშაო გარემოს გაცნობა. WPF-ის ინსტრუმენტების პანელი

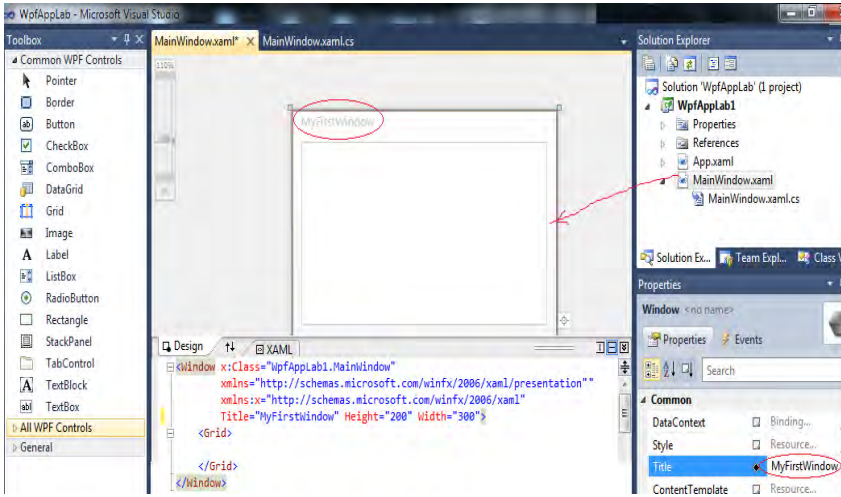
(ლაბორატორიული სამუშაო N 1)

ახალი WPF პროექტის შექმნის ფანჯარა:



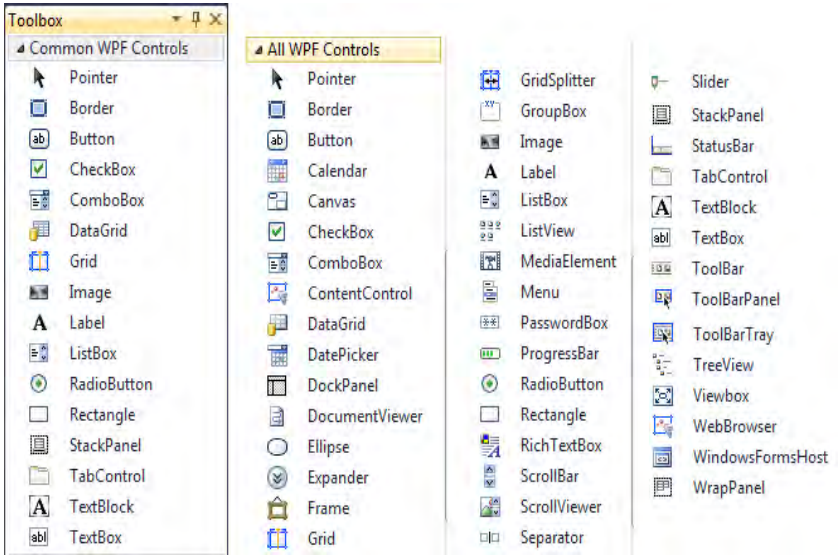
ნახ.2.1

მიიღება 2.2 ნახაზზე ნაჩვენები ფანჯარა:



ნახ.2.2

WPF–ის ინსტრუმენტების პანელი მოცემულია 2.3 ნახაზზე.

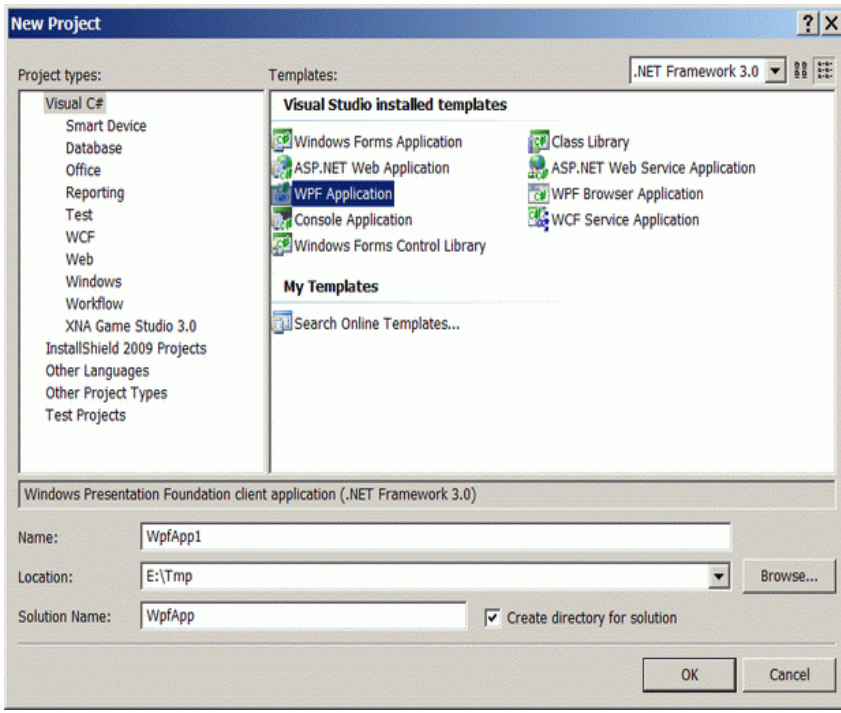


ნახ.2.3

2.2. მარტივი ღილაკის დაპროგრამება ანიმაციის ელემენტებით
(ლაბორატორიული სამუშაო N 2)

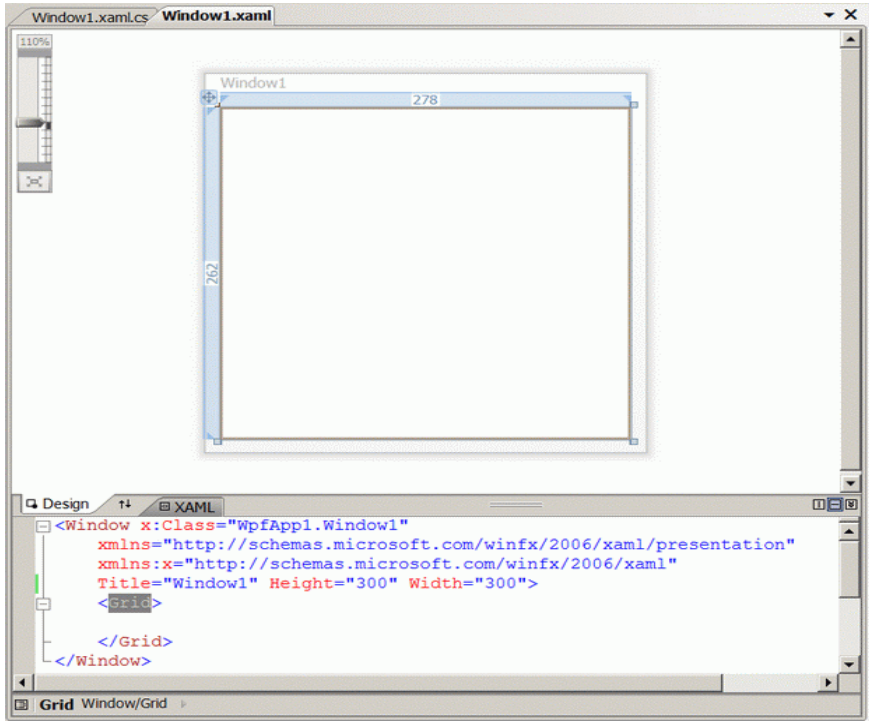
მიზანი: WPF-ის მულტიმედიალური შესაძლებლობების დემონსტრირება

1. შექმენით პროექტი სახელით: WpfApp1 და Solution-ის სახელით: WpfApp.



ნახ.2.4

შედეგი მოცემულია 2.5 ნახაზზე:



ნახ.2.5

2. შეავსეთ XAML კოდი:

```
<!----WpfApp1----- Window1.xaml ----- >
<!-- მთლიანად ფანჯრის აწყობა: მომხმარებლის ინტერფეისი -->
<Window x:Class="WpfApp1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="300" Height="300"
        Title="მაგალითი_1"
        x:Name="Window"
        Background="Aqua">

    <!-- აპლიკაციის ფანჯრის რესურსების შესანახი სექცია: -->
    <Window.Resources>
```

```
    <!-- იქმნება შაბლონი დასახელებით ClassButton, ღილაკის ტიპის
          ელემენტებისთვის-->
     <ControlTemplate x:Key="ClassButton"
            TargetType="{x:Type Button}">
       <!-- ფანჯრის რესურსების შესანახი სექცია ღილაკისთვის  -->
       <ControlTemplate.Resources>
        <!-- სექციაში Storyboard აღიწერება ანიმაციურო ეფექტი, მაგ.,
                      ღილაკის დაჭერა-->
    <Storyboard x:Key="Timeline1">
         <!-- მითითებულ დროში, მაგ., 0.3 წამი, ღილაკი ხდება
                                                  გაუმჭვირვალე  -->
      <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
        Storyboard.TargetName="glow"
        Storyboard.TargetProperty="(UIElement.Opacity)">
          <!-- ანიმაციის ბოლო წერტილი  -->
        <SplineDoubleKeyFrame KeyTime="00:00:0.3" Value="1" />
      </DoubleAnimationUsingKeyFrames>
    </Storyboard>

            <!-- სექცია Storyboard  ღილაკის ჩასაქრობად  -->
    <Storyboard x:Key="Timeline2">
<!- მითითებულ დროში, მაგ., 0.3 წამში ღილაკი ხდება გამჭვირვალე-->
      <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
        Storyboard.TargetName="glow"
        Storyboard.TargetProperty="(UIElement.Opacity)">
          <!-- ანიაციის ბოლო წერტილი  -->
       <SplineDoubleKeyFrame KeyTime="00:00:0.3" Value="0" />
      </DoubleAnimationUsingKeyFrames>
  </Storyboard>
</ControlTemplate.Resources>
       <!-- ღილაკის აღწერის სექცია  -->
         <!-- გარე საზღვარი - თეთრი -->
 <Border BorderBrush="#FFFFFFFF" BorderThickness="1,1,1,1"
                                        CornerRadius="4,4,4,4">
         <!-- შიგა საზღვარი - შავი  -->
 <Border x:Name="border" Background="#7F000000"
BorderBrush="#FF000000" BorderThickness="1,1,1,1"
CornerRadius="4,4,4,4">
      <Grid>
        <Grid.RowDefinitions>
          <!-- ღილაკის ზედა ნახევარი  -->
```

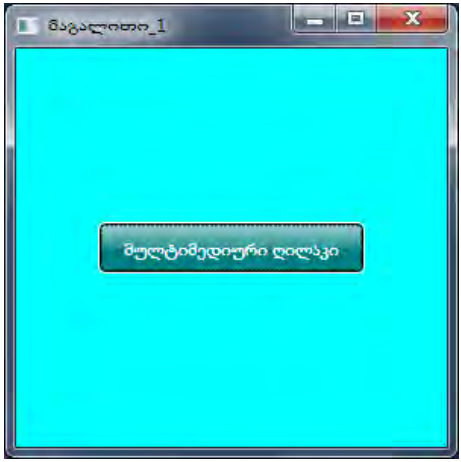
```
            <RowDefinition Height="0.5*" />
              <!-- ღილაკის ქვედა ნახევარი -->
                <RowDefinition Height="0.5*" />
        </Grid.RowDefinitions>

        <!-- იხატება ღილაკის განათება  -->
    <Border Opacity="0" HorizontalAlignment="Stretch"
      x:Name="glow" Width="Auto"
            Grid.RowSpan="2" CornerRadius="4,4,4,4">
        <Border.Background>
         <!-- მიეწოდება რადიალური გრადიენტი წანაცვლებით  -->
              <RadialGradientBrush>
                <RadialGradientBrush.RelativeTransform>
                   <TransformGroup>
              <ScaleTransform ScaleX="1.702" ScaleY="2.243" />
                      <SkewTransform AngleX="0" AngleY="0" />
                      <RotateTransform Angle="0" />
                 <TranslateTransform X="-0.368" Y="-0.152" />
                   </TransformGroup>
                </RadialGradientBrush.RelativeTransform>
                <!-- გრადიენტის ფერები ARGB  ფორმატში-->
                <GradientStop Color="#B28DBDFF" Offset="0" />
                <GradientStop Color="#008DBDFF" Offset="1" />
                </RadialGradientBrush>
            </Border.Background>
          </Border>
     <!-- ათინათის  დახატვა -->
     <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center" Width="Auto" Grid.RowSpan="2" />
    <Border HorizontalAlignment="Stretch" x:Name="shine"
       Width="Auto" CornerRadius="4,4,0,0">
         <Border.Background>
           <LinearGradientBrush StartPoint="0.494,0.028"
              EndPoint="0.494,0.889">
                <GradientStop Color="#99FFFFFF" Offset="0" />
                <GradientStop Color="#33FFFFFF" Offset="1" />
              </LinearGradientBrush>
         </Border.Background>
      </Border>
     </Grid>
    </Border>
  </Border>
```

```
<!--ტრიგერული მოვლენების დაყენება, მაუსის დაჭერის რეაქციაზე -->
    <ControlTemplate.Triggers>
      <!-- მაუსის ღილაკი დაჭერილია -->
    <Trigger Property="IsPressed" Value="True">
 <Setter Property="Opacity" TargetName="shine" Value="0.4" />
  <Setter Property="Background" TargetName="border"
                                        Value="#CC000000" />
  <Setter Property="Visibility" TargetName="glow"
                                        Value="Hidden" />
   </Trigger>
     <!-- მაუსის კურსორი ობიექტზეა -->
       <Trigger Property="IsMouseOver" Value="True">
    <!-- ობიექტზე შესვლა - ანიმაციის გამოძახება Timeline1 -->
         <Trigger.EnterActions>
  <BeginStoryboard Storyboard="{StaticResource Timeline1}" />
         </Trigger.EnterActions>
  <!-- ობიექტიდან გამოსვლა - ანიმაციის გამოძახება Timeline2 -->
         <Trigger.ExitActions>
  <BeginStoryboard Storyboard="{StaticResource Timeline2}" />
         </Trigger.ExitActions>
     </Trigger>
           </ControlTemplate.Triggers>
       </ControlTemplate>
    </Window.Resources>
    <Grid>
        <!-- იქმნება ღილაკის ეგზემპლარი ფანჯრის შუაში -->
        <Button x:Name="Btn1" HorizontalAlignment="Center"
     VerticalAlignment="Center"  Width="176" Height="34"
    Content="მულტიმედიალური ღილაკი" Foreground="#FFFFFFFF"
     Template="{DynamicResource ClassButton}" />
    </Grid>
</Window>
```

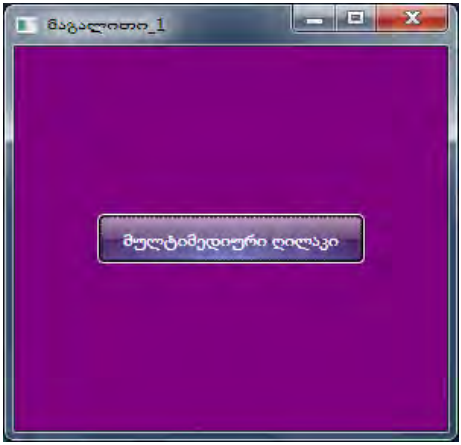
3. ავამუშავოთ პროგრამა, მივიღებთ 2.6 ნახაზზე ნაჩვენებ შედეგს, სადაც შაბიამნისფერის ფონია. ღილაკზე დაჭერით იცვლება ფონი იასამნისფერით. შემდგომ ფერების შეცვლა მონაცვლეობით ხდება მაუსის დაკლიკვით.

4. დავაპროგრამოთ ლოგიკა C#ენაზე:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp1
{
  // Interaction logic for Window1.xaml
  public partial class Window1 : Window
  {
    System.Windows.Media.Brush color;
    bool colorFlag = true;
   public Window1()
    {
       //InitializeComponent();
     Application.LoadComponent(this, new Uri("Window1.xaml",
                       UriKind.Relative));

     Btn1.Click += new RoutedEventHandler(Btn1_Click);
     color = this.Window.Background;
    }

     void Btn1_Click(object sender, RoutedEventArgs e)
       {
         // ფინის ფერის შეცვლა
        if (colorFlag)
          this.Window.Background =
               System.Windows.Media.Brushes.Purple;
        else
          this.Window.Background = color;

        colorFlag = !colorFlag;
       }
   }
}
```



ნახ.2.6



ნახ.2.7

5. Application - ობიექტი

ნებისმიერი დანართი (აპლიკაცია) იყენებს Application კლასს, რომელიც Run() მეთოდის საშუალებით ამ აპლიკაციას მიუერთებს ოპერაციული სისტემის მოვლენების მოდელს ასამუშავებლად.

Application - ობიექტი მართავს დანართის სიცოცხლის დროს, აკვირდება ხილვად ფანჯრებს, ათავისუფლებს რესურსებს და

აკონტროლებს აპლიკაციის გლობალურ მდგომარეობას. Run() მეთოდი აამუშავებს შესრულების გარემოს დისპეტჩერს, რომელიც დაიწყებს მოვლენების და შეტყობინებების გადაგზავნას დანართის კომპონენტებთან.

დროის მოცემულ მომენტში აქტიური შეიძლება იყოს მხოლოდ ერთი Application - ობიექტი, და ის მუშაობს მანამ, სანამ დანართი არ დასრულდება. შესრულებად Application - ობიექტზე მიმართვა შეიძლება დანართის ნებისმიერი ადგილიდან Application.Current სტატიკური თვისების საშუალებით.

WPF დანართის (და Application ობიექტის) სასიცოცხლო დრო შედგება შემდეგი ეტაპებისგან:

1. კონსტრუირდება Application ობიექტი;
2. გამოიძახება მისი Run() მეთოდი;
3. ინიცირდება მოვლენა Application.Startup;
4. მომხმარებლის კოდი აკონსტრუირებს ერთ ან რამდენიმე Window (ან Page) ობიექტს და დანართი მუშაობს;
5. გამოიძახება მეთოდი Application.Shutdown();
6. გამოიძახება მეთოდი Application.Exit().

ჩვენი WPF-დანართის აგებისას სისტემამ თვითონ შექმნა Solution Explorer-ში ორი ფაილი Application-ობიექტთან დაკავშირებული ტიპიური სახელებით: App.xaml და App.xaml.cs. მათი ნახვა შესაძლებელია. აქ არ ჩანს ცხადად არც Application და Window ობიექტების შექმნა და არც Run() მეთოდის გამოძახება (!)

WPF-პლატფორმის შემქმნელებმა გადაწყვიტეს, რომ, ვინაიდან ეს სტანდარტული ოპერაციები გამოიყენება ყველა დანართისათვის, არაა საჭირო მისი მომხმარებელზე გადაცემა და თვით კომპილატორი (სისტემა) ავტომატურად გამოიყენებს მათ (!), თვითონ შექმნის Application ობიექტს WPF-პროექტის კომპილაციის დროს, შექმნის ასევე Window (ან Page) ობიექტებს და გადასცემს მათ Application.Run() მეთოდს.

სხვა სიტყვებით რომ ვთქვათ, ჩვენ „ვერ ვხედავთ ცხადად" კომპილატორის მიერ ასეთი კოდის შესრულების ტექსტს:
// ეს ტექსტი მოტანილია საილუსტრაციოდ ---------

```
public partial class App : System.Windows.Application
{
  public App()
    {
       System.Windows.Window win = new System.Windows.Window();
       win.Title = "Hello World";
       win.Show();
    }
      // Application entry point
    [System.STAThreadAttribute()]
    [System.Diagnostics.DebuggerNonUserCodeAttribute()]
    public static void Main()
    {
       App app = new App();
       app.Run();
    }
}
```

ჩვენი დანართის მაგალითისთვის App.xaml.cs ფაილი ასე გადავაკეთოთ:

```
// ----- App.xaml.cs  -------------------------
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Windows;

namespace WpfApp1
{
  public partial class App : Application
    {
      public App()
        {
         // გაშვებულია Application ობიექტი
          this.Startup += new StartupEventHandler(App_Startup);
            // მოვლენა დაუმუშავებელი გამოსარიცხი სიტუაციისთვის
    this.DispatcherUnhandledException += App_DispatcherUnhandledException;
        }
```

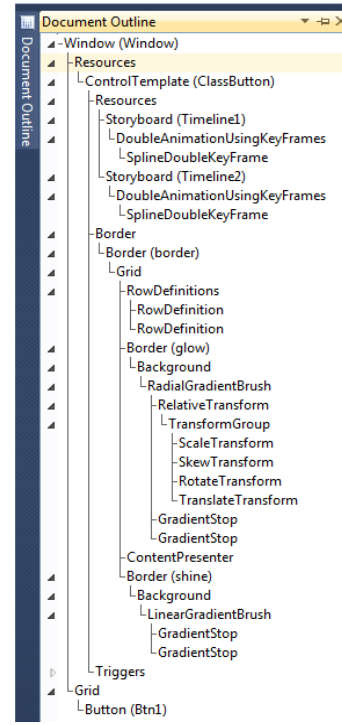
```
    void App_DispatcherUnhandledException(object sender,
        System.Windows.Threading.DispatcherUnhandledExceptionEventArgs e)
    {
        e.Handled = true;// შესრულების გაგრძელება
    }

    void App_Startup(object sender, StartupEventArgs e)
    {
        // იქმნება სასტარტო ფანჯრის ობიექტი -----
        Window1 win = new Window1();
        // ფანჯრის ობიექტის აწყობა -----
        win.Title = "ფანჯრის  ახალი სათაური ";
        // ფანჯრის ნახვა ---------
        win.Show();
    }
  }
}
```

ავამუშავოთ დანართი და ვნახოთ შედეგი.

6. XAML-პროგრამის სტრუქტურის სანახავად Document Outline პანელის გამოტანა:

View/Other Windows/Document Outline

მიიღება შემდეგი იერარქიული სურათი:



ნახ.2.8

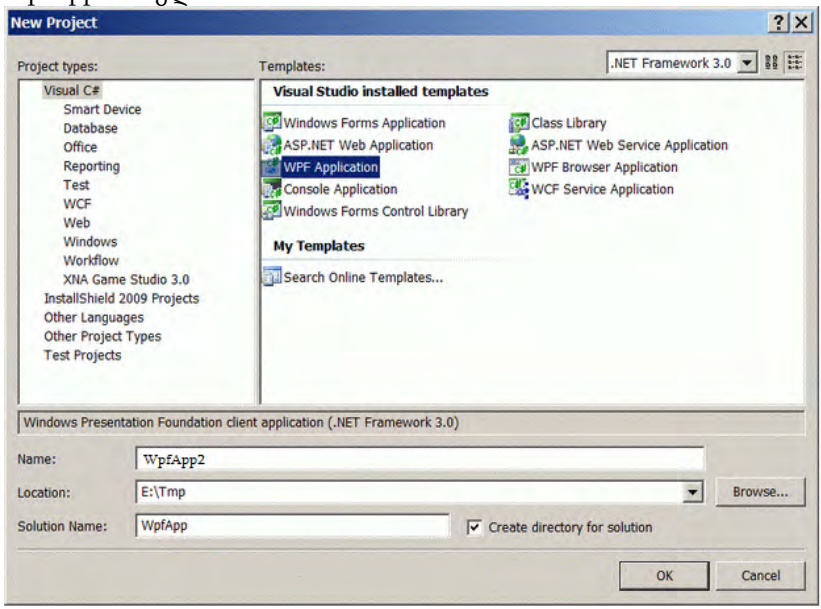
2.3. WPF-ის ფანჯრების მართვის ელემენტები
(ლაბორატორიული სამუშაო N 3)

მიზანი: WPF-ის ფანჯრების მართვის ელემენტების შესწავლა.

ფანჯრების შექმნა და მათი გამოყენება მოხერხებული საშუალებაა შესაბამისი ფუნქციონალობის და მონაცემების ინკაფსულაციის სარეალიზაციოდ. ფანჯარა (Window) ყველაზე მაღალი დონეა მომხმარებელთან ურთიერთობისათვის, ის არ შეიძლება იყოს სხვა ფანჯრის ნაწილი.

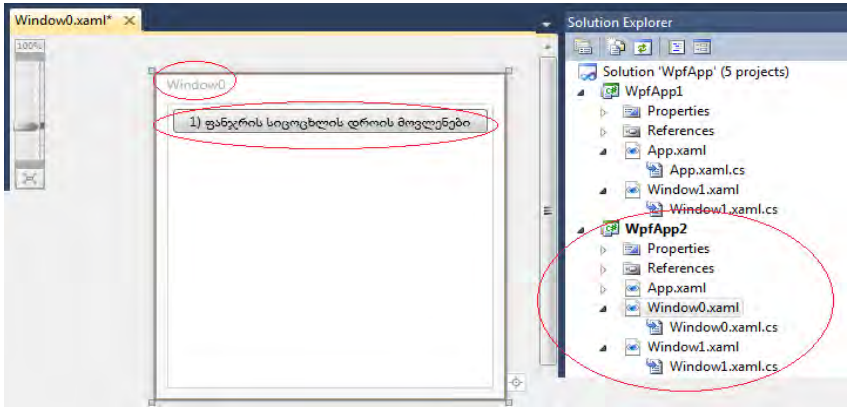
ფანჯარა შეიძლება იყოს მოდალური და არამოდალური. მოდალური ფანჯარა აბლოკირებს ამ დანართის სხვა ფანჯრებს მის დახურვამდე ძდა თვითონ მუშაობს ოპერაციულ სისტემასთან.

1. დავამატოთ WpfApp -ს Solution-იდან ახალი პროექტი WpfApp2 სახელით:.



ნახ.2.9

შედეგი:



ნახ.2.10

2. შეავსეთ XAML კოდი:

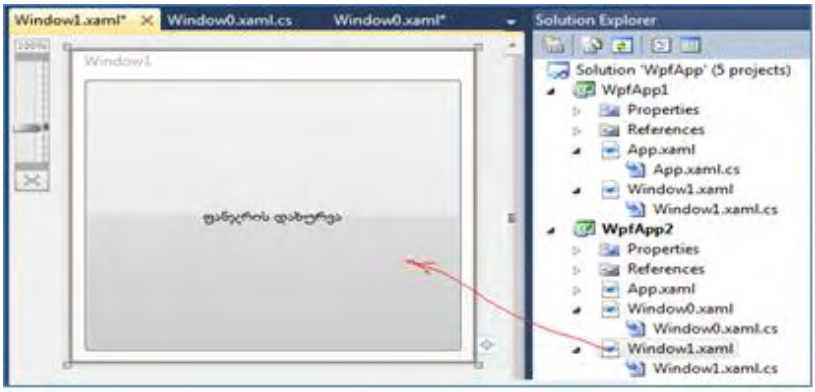
```
<Window x:Class="WpfApp2.Window0"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentati
on"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window0"
    Height="300" Width="300"
    SizeToContent="WidthAndHeight"
    MinWidth="300"
    MaxWidth="400"
    MinHeight="100"
    WindowStartupLocation="CenterScreen"
        >
    <StackPanel>
        <Button Margin="5" Click="LifeEvents">1) ფანჯრის
                        სიცოცხლის დროის მოვლენები</Button>
    </StackPanel>
</Window>
```

3. C# კოდის ნაწილი Window0-თვის:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
```

```
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApp2
{
   public partial class Window0 : Window
    {
        public Window0()
        {
            InitializeComponent();
        }

   private void LifeEvents(object sender, RoutedEventArgs e)
     {
       Window1 wnd1 = new Window1();
       wnd1.ShowInTaskbar = false; // არ ჩანს ამოცანების პანელზე
       wnd1.Show();                // გამოჩნდება მოდალურ რეჟიმში
      }
    }
}
```

4. დავამატოთ ფორმა Window1() :



ნახ.2.11

XAML –კოდი:

```
<Window x:Class="WpfApp2.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="Window1" Height="284" Width="300"

      Initialized="Window_Initialized"
      Activated="Window_Activated"
      Deactivated="Window_Deactivated"
      Loaded="Window_Loaded"
      ContentRendered="Window_ContentRendered"
      Closing="Window_Closing"
      Unloaded="Window_Unloaded"
      Closed="Window_Closed"
          >
      <Grid>
          <Button Click="Button_Click" Content="ფანჯრის
  დახურვა"></Button>
      </Grid>
  </Window>
```

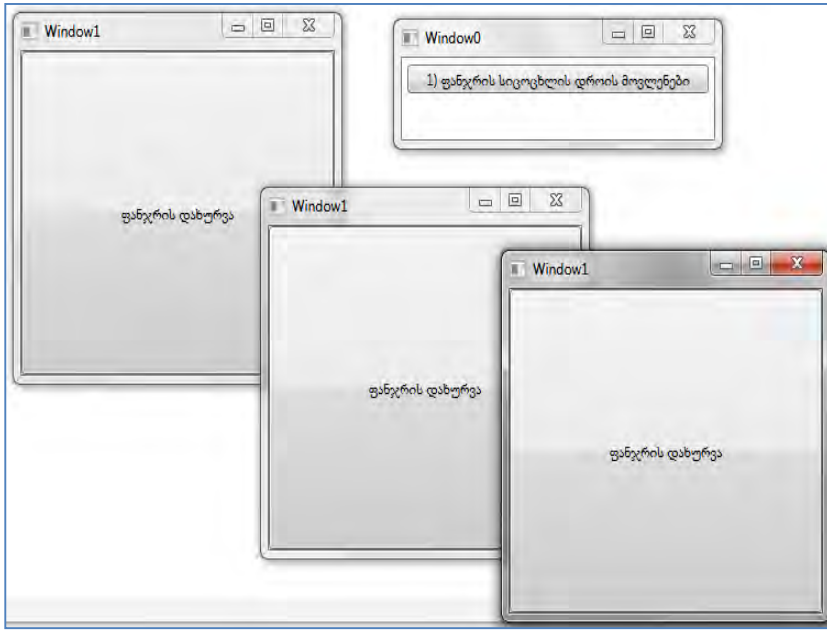
C# - კოდი :

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace WpfApp2
{
   public partial class Window1 : Window
    {
       public Window1()
        {
```

```
            System.Diagnostics.Debug.WriteLine("ამუშავდა ფანჯრის
                                                    კონსტრუქტორი");
             InitializeComponent();
           }
   private void Window_Initialized(object sender, EventArgs e)
           {
             System.Diagnostics.Debug.WriteLine("ამოქმედდა
                                    ფანჯრის მოვლენა  Initialized");
           }
   private void Window_Activated(object sender, EventArgs e)
           {
             System.Diagnostics.Debug.WriteLine("ამოქმედდა
                                      ფანჯრის მოვლენა Activated");
           }
   private void Window_Deactivated(object sender, EventArgs e)
           {
             System.Diagnostics.Debug.WriteLine("("ამოქმედდა
                                   ფანჯრის მოვლენა  Deactivated ");
           }
   private void Window_Loaded(object sender, RoutedEventArgs e)
           {
             System.Diagnostics.Debug.WriteLine("("ამოქმედდა
                                      ფანჯრის მოვლენა  Loaded ");
           }
private void Window_ContentRendered(object sender, EventArgs e)
           {
             System.Diagnostics.Debug.WriteLine("("ამოქმედდა
                                 ფანჯრის მოვლენა  ContentRendered");
           }
  private void Window_Closing(object sender,
                        System.ComponentModel.CancelEventArgs e)
           {
             System.Diagnostics.Debug.WriteLine("ამოქმედდა
                                      ფანჯრის მოვლენა  Closing");
      if (MessageBox.Show("ნამდვილად გნებავთ ფანჯრის დახურვა ?",
         "კი", MessageBoxButton.YesNo) == MessageBoxResult.No)
                   e.Cancel = true; // არ დაიხუროს
           }
  private void Window_Unloaded(object sender, RoutedEventArgs e)
           {
```

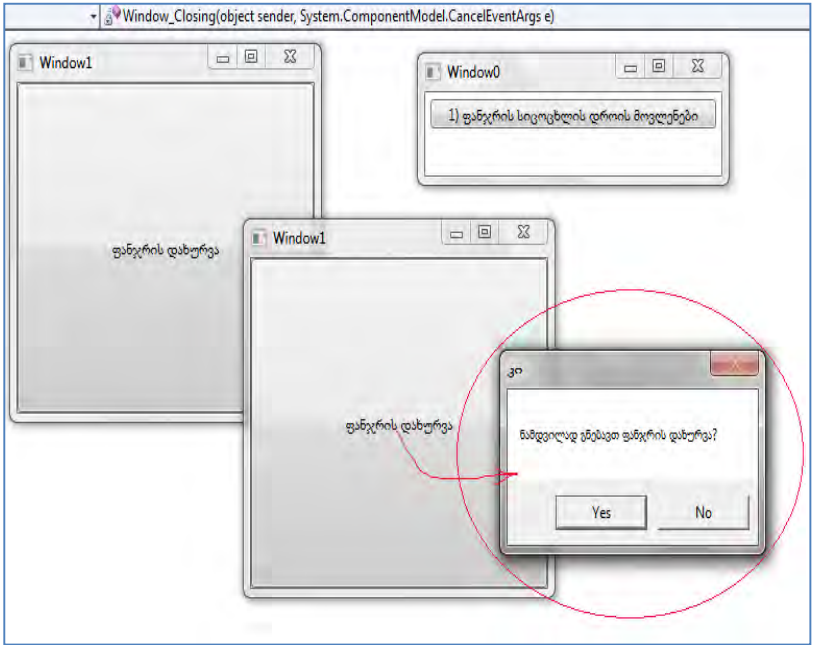
```
            System.Diagnostics.Debug.WriteLine("ამოქმედდა
                                        ფანჯრის მოვლენა  Unloaded");
        }
 private void Window_Closed(object sender, EventArgs e)
        {
            System.Diagnostics.Debug.WriteLine("ამოქმედდა
                                        ფანჯრის მოვლენა  Closed");
        }
 private void Button_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
    }
}
```

5. აპლიკაციის ამუშავებით მიიღება შმდეგი სურათი (ნახ.2.12):



ნახ.2.12

ფანჯარაზე მაუსით დაკლიკვის შემთხვევაში გამოვა 2.13 ნახაზზე ნაჩვენები შეტყობინების ფანჯარა.



ნახ.2.13

6. დავალება:

აამუშავეთ პროგრამა და გამოიკვლიეთ ფანჯრების მუშაობის ფუნქციონალობა.

2.4. მრავალფანჯრიანი პროექტი WPF-ში
(ლაბორატორიული სამუშაო N 4)

მიზანი: WPF-ის პროექტის აგება მრავალფანჯრიანი შემთხვევისთვის.

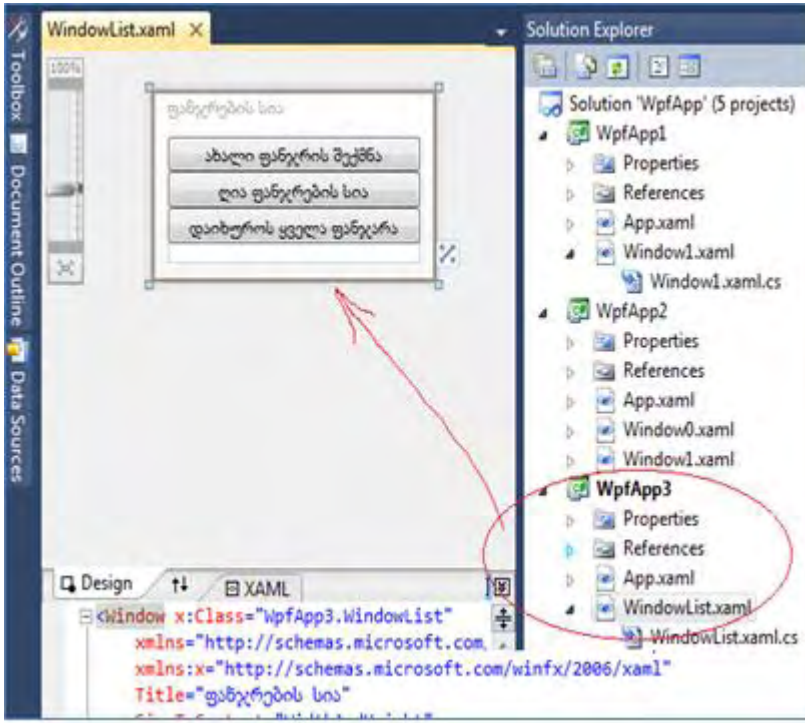
ერთ დანართში შეიძლება რამდენიმე ფანჯრის შექმნა, რომლებიც შეიძლება ერთდროულად იქნას გახსნილი. მათი დახურვის პროცესი მოითხოვს გარკვეული მართვის განხორციელებას. მაგალითად, თუ დაიხურაერთი რომელიმე, სხვა რჩება გახსნილი.

საჭიროა ისეთი პროცესის აგება, როდესაც ყველა ფანჯრის დახურვა იქნება შესაძლებელი ერთდროულად. განვიხილოთ ასეთი პროექტი.

1. დავამატოთ WpfApp -ს Solution-იდან ახალი პროექტი WpfApp3 სახელით:.

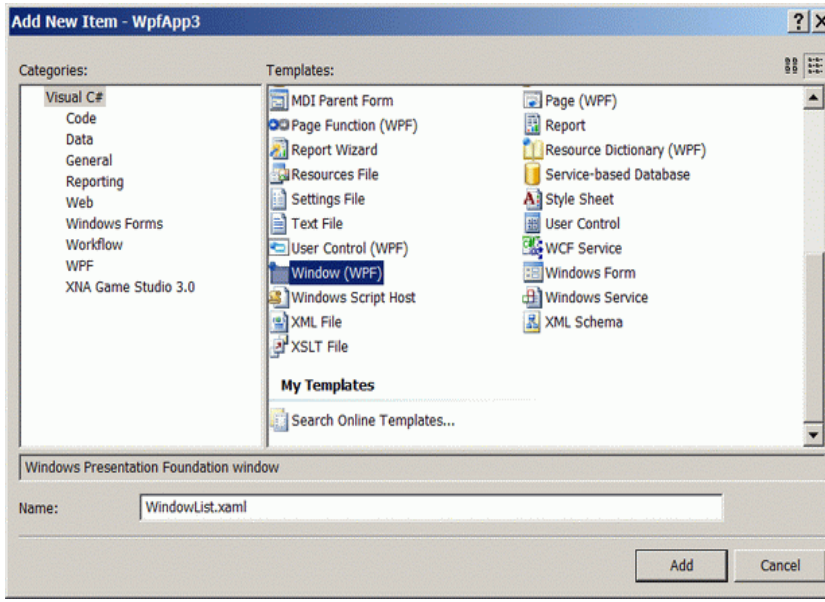
ნახ.2.14

შედეგი:



ნახ.2.15

2. Solution Explorer-ში წავშალოთ `Window1.xaml` ფაილი და დავამატოთ ახალი ფაილი სახელით `WindowList.xaml` (ეს შეცვლის ყველგან ამ სახელს).



ნახ.2.16

3. გახსენით App.xaml ფაილი და შეცვალეთ Application-ობიექტის პარამეტრი StartupUri ახალი სასტარტო ფანჯრის ფაილის სახელით.

```
<Application x:Class="WpfApp3.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="WindowList.xaml"
             >
</Application>
```

4. შეავსეთ XAML კოდი WindowList.xaml ფაილისთვის:

```
<Window x:Class="WpfApp3.WindowList"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ფანჯრების სია"
    SizeToContent="WidthAndHeight"
    MinWidth="200" mc:Ignorable="d"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

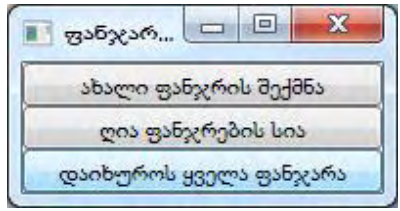
```
xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006" d:DesignHeight="116" d:DesignWidth="200">
    <StackPanel>
        <Button Click="NewWindowClicked">ახალი ფანჯრის
შექმნა</Button>
        <Button Click="ListOpenWindows">ღია ფანჯრების
სია</Button>
        <Button Click="AllCloseWindows">დაიხუროს ყველა
ფანჯარა</Button>
    </StackPanel>
</Window>
```

5. შეცვალეთ `WindowList.xaml.cs` C#-კოდი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace WpfApp3
{
 public partial class WindowList : Window
  {
   static int createCount;
   public WindowList()
     {
      InitializeComponent();
     // ფანჯრის სახელის განსაზღვრა და მთავარი ფანჯრის არჩევა
      if (createCount == 3)
        {
         Application.Current.ShutdownMode =
                       ShutdownMode.OnMainWindowClose;
         Application.Current.MainWindow = this;
         this.Title = "მთავარი ფანჯარა № " +
                           (createCount++).ToString();
        }
```

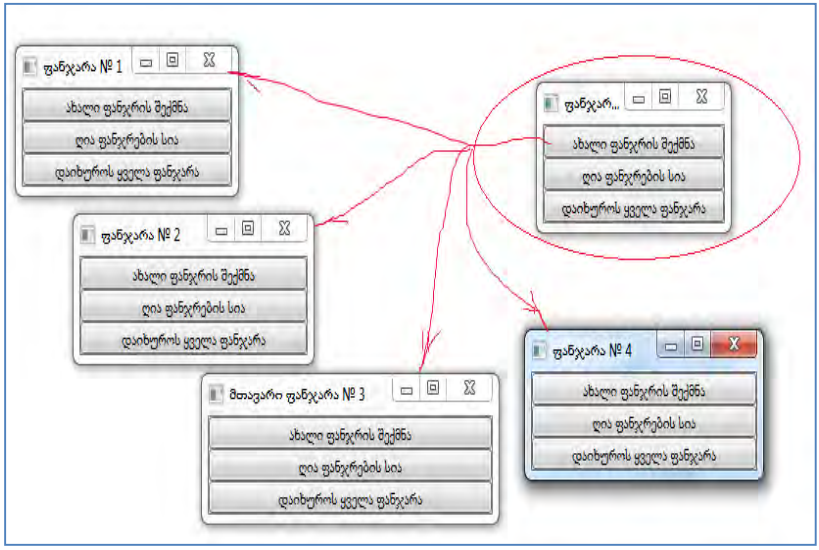
```
        else
          this.Title = "ფანჯარა № " + (createCount++).ToString();
        }
private void NewWindowClicked(object sender, RoutedEventArgs e)
        {
          new WindowList().Show();
        }
private void ListOpenWindows(object sender, RoutedEventArgs e)
        {
          StringBuilder sb = new StringBuilder();
          foreach (Window openWindow in
                        Application.Current.Windows)
            sb.AppendLine(openWindow.Title);

          MessageBox.Show(sb.ToString(), "დანართის გახსნილი
                                                ფანჯრები");
        }
private void AllCloseWindows(object sender, RoutedEventArgs e)
        {
         //Application.Current.ShutdownMode =
                              ShutdownMode.OnExplicitShutdown;
          Application.Current.Shutdown();// ხურავს ფანჯრებს
        }
    }
 }
```

6. ავამუშავოთ აპლიკაცია, მივიღებთ ასეთ შედეგს:



ნახ.2.17

7. პირველ ღილაკზე მაუსით დაკლიკვით იქმნება ახალი ფანჯარა ახალი რიგითი ნომრით (ნახ.2.18).

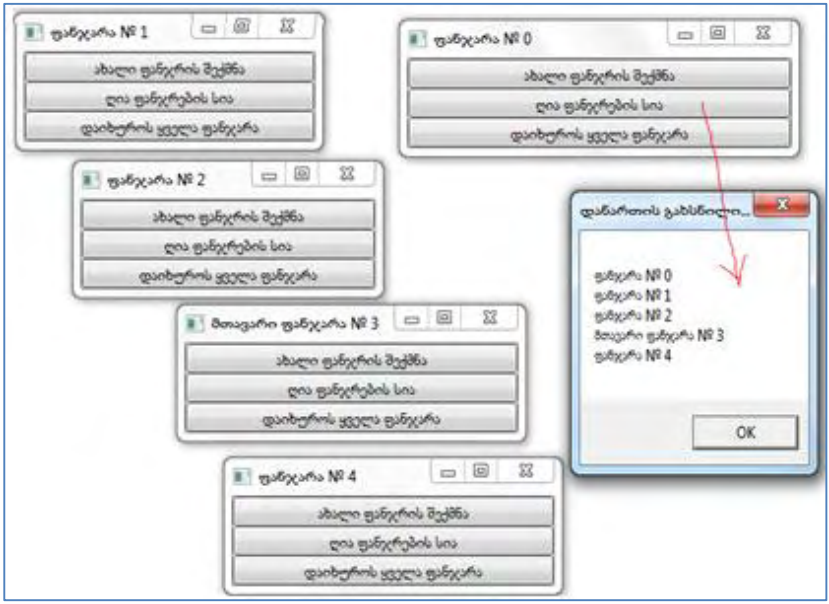


ნახ.2.18

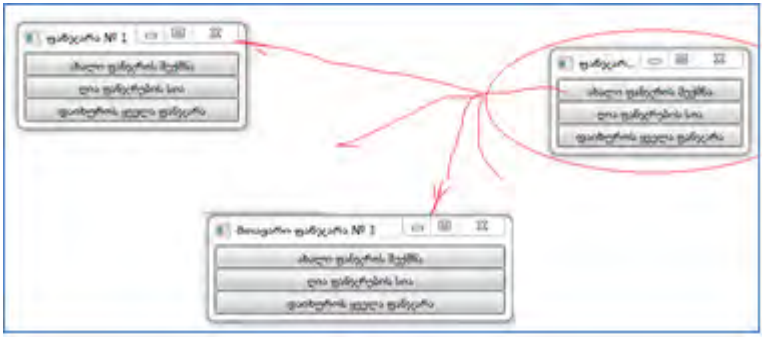
8. მეორე ღილაკზე „**ღია_ფანჯრების_სია**“ მაუსით დაკლიკვით იქმნება ერთი ახალი ფანჯარა „დანართის_გახსნილი_ფანჯრები“ (ნახ.2.19).

9. მესამე ღილაკზე „**დაიხუროს_ყველა_ფანჯარა**“ მაუსით დაკლიკვით ეკრანიდან გაქრება ეს კონკრეტული (მაგალითად, მე–2 და მე–4) ფანჯარა, რომელზეც მაუსით ვიმოქმედეთ (ნახ.2.20).

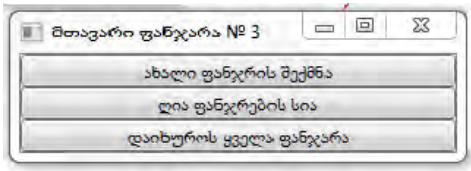
10. ვინაიდან ჩვენ მე-3 ფანჯარა გვქონდა გამოცხადებული, როგორც მთავარი, ამიტომ მისი დახურვის ღილაკის დაკლიკვით იხურება ყველა ფანჯარა ერთდროულად.



ნახ.2.19



ნახ.2.20

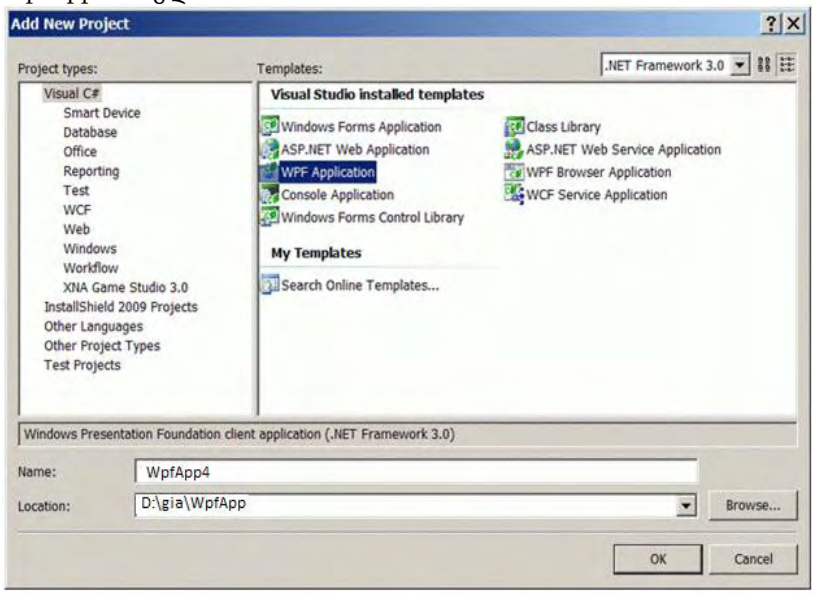


ნახ.2.21

2.5. WPF-ის მომხმარებელთა მართვის ელემენტები
(ლაბორატორიული სამუშაო N 5)

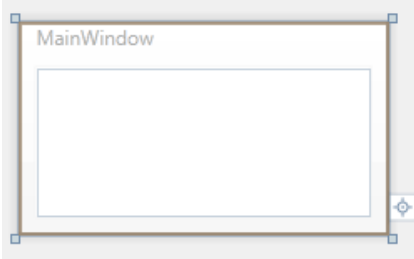
მიზანი: WPF-ის აპლიკაციის დამუშავება მომხმარებელთა ინტერფეისების ვიზუალური მართვის ელემენტებით.

1. დავამატოთ WpfApp -ს Solution-იდან ახალი პროექტი WpfApp4 სახელით:.



ნახ.2.22

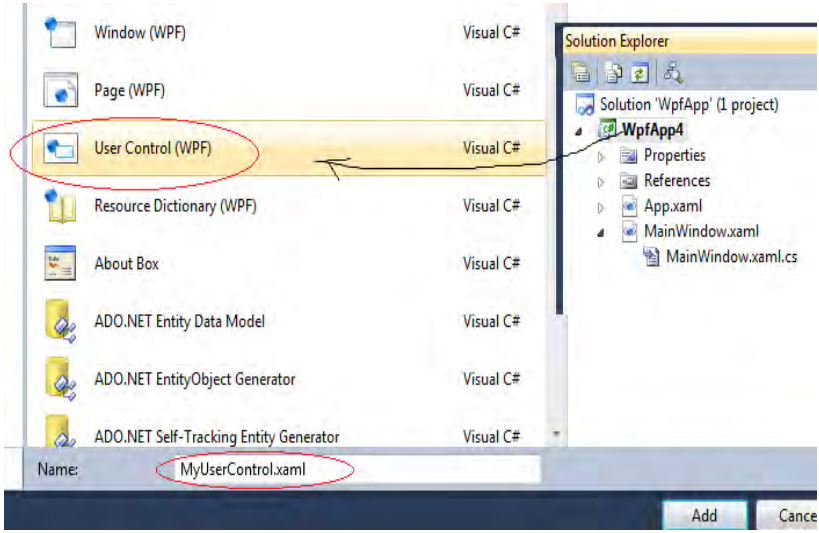
შედეგში ცარიელი ფორმაა:



ნახ.2.23

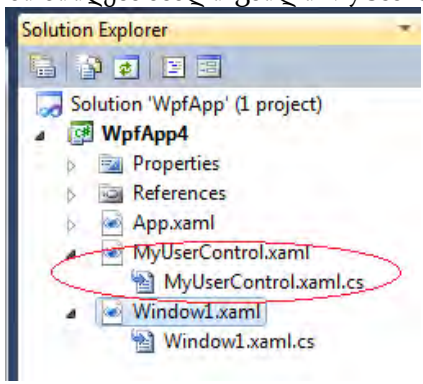
შევცვალოთ სახელი ყველგან Window1-ით.

2. Solution Explorer-ში WpfApp4-s დავამატოთ Add User Control ფაილი და დავამატოთ ახალი ფაილი სახელით `MyUserControl.xaml` ().



ნახ.2.24

Solution Explorer-ში მიიღება ახალი ფაილი MyUserControl.xaml.



ნახ.2.25

3. გახსენით MyUserControl.xaml ფაილი და შეცვალეთ ტექსტი შემდეგი სახით:.

```
<UserControl x:Class="WpfApp4.MyUserControl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pres
entation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >
        <Button Click="Button_Click">ჩემი ღილაკი</Button>
</UserControl>
```

4. შესავსეთ C# კოდი MyUserControl.xaml.cs ფაილისთვის:

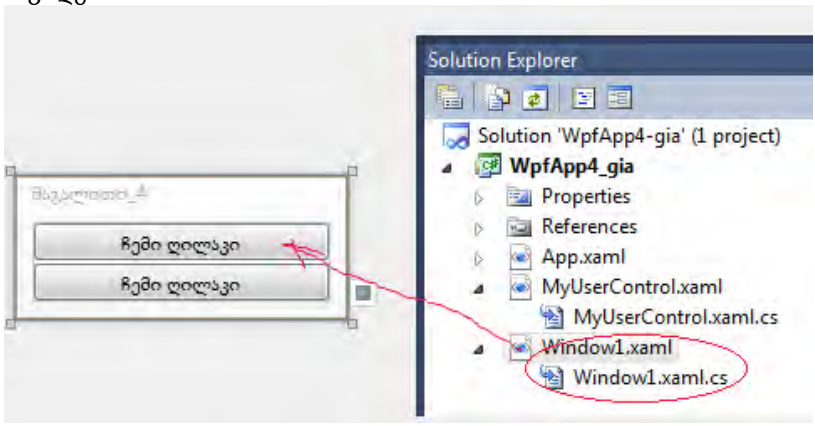
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApp4
{
    public partial class MyUserControl : UserControl
    {
        public MyUserControl()
        {
            InitializeComponent();
        }

     private void Button_Click(object sender,
                                  RoutedEventArgs e)
        {
          MessageBox.Show("მოგეწსალმებით !", "ელემენტი
                UserControl");
        }
    }
}
```

5. გაუშვით პროგრამა კომპილაციაზე, რათა შეიქმნას მომხმარებლის ერთი ღილაკი, რომელსაც შემდგომ გამოიყენებს სხვა ფანჯარაში რამდენჯერმე.

6. შეცვალეთ Window1.xaml კოდი:

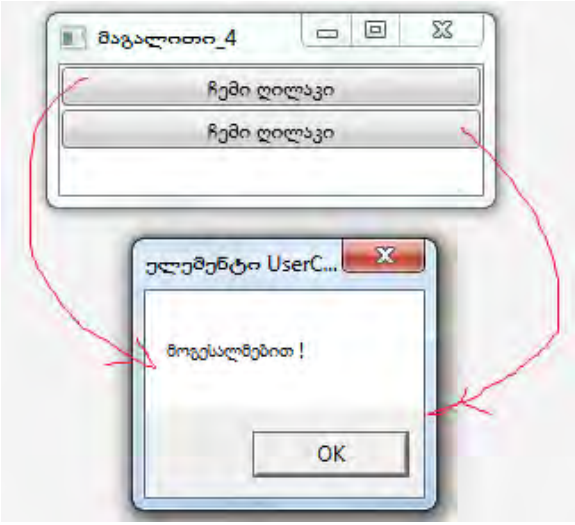
```
<Window x:Class="WpfApp4.Window1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pres
entation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:My="clr-namespace:WpfApp4"   (აქ My არის
ფსევდონიმი !)
    Title="მაგალითი_4"
    Width="200"
    SizeToContent="Height"
    WindowStartupLocation="CenterScreen"
        >
    <StackPanel>
        <My:MyUserControl Margin="1"/>
        <My:MyUserControl Margin="1"/>
    </StackPanel>
</Window>
```

მივიღებთ:



ნახ.2.26

7. ავამუშავოთ აპლიკაცია. შედეგი მოცემულია 2.27 ნახაზზე.



ნახ.2.27

დასკვნა:

შევქმენით მომხმარებლის ვიზუალური ელემენტი, მაგალითად, ღილაკი წარწერით „ჩემი ღილაკი“. შემდეგ ის გამოვიყენეთ რამდენჯერმე.

2.6. WPF-ში Web-გვერდების აპლიკაციების შექმნა
(ლაბორატორიული სამუშაო N 6)

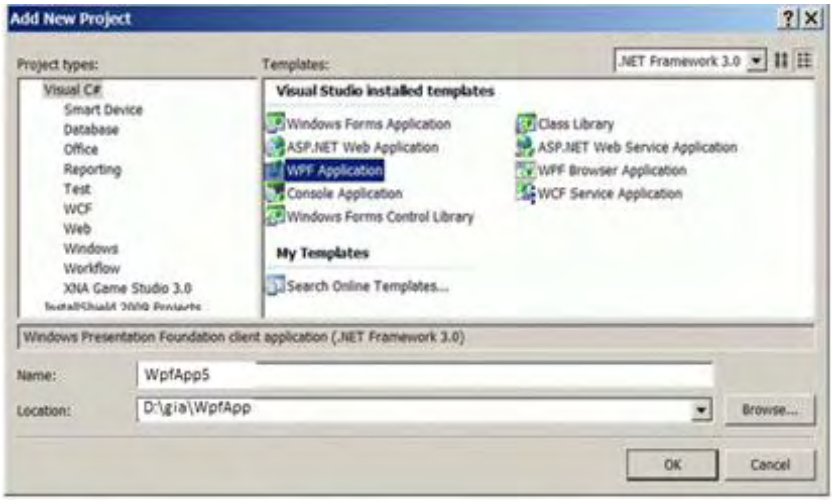
მიზანი: WPF-ის ვებ-აპლიკაციის დამუშავება მომხმარებელთა ინტერფეისების ვიზუალური მართვის ელემენტებით.

WPF-პლატფორმას აქვს ვებ-გვერდების შექმნის საშუალებები. ასეთი გვერდების გამოყენება ხშირად მომხმარებლისთვის უფრო მოსახერხებელია, ვიდრე ფანჯრული ორგანიზაცია.

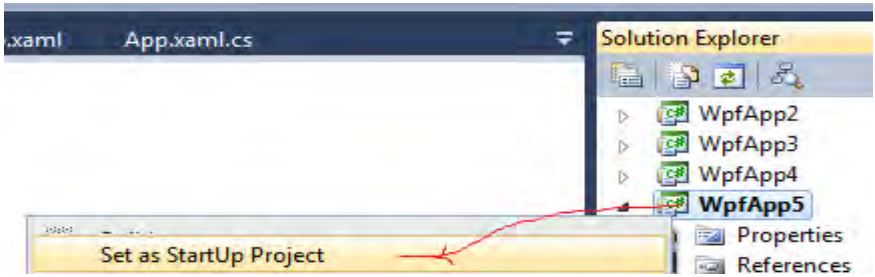
გვერდული დანართის (აპლიკაციის) შიგთავსი ჩაშენდება სპეციალურ ნავიგაციურ კარკასში, რომელსაც აქვს ნავიგაციური კავშირები და ნავიგაციური ჟურნალი.

მისი ძირითადი (ფესვური) კლასია NavigationWindow, რომელიც ამატებს აპლიკაციისთვის **ნავიგაციის სტანდარტულ ინტერფეისს** და მისთვის საჭირო ინფრასტრუქტურას. NavigationWindow კლასი წარმოებულია Window-კლასიდან და აქვს წვდომა დანართის იგივე საშუალებებზე.

1. დავამატოთ WpfApp -ს Solution-იდან ახალი პროექტი WpfApp5 სახელით (ნახ.2.28) და გავხადოთ „სასტარტო“ (ნახ.2.29).



ნახ.2.28



ნახ.2.29

2. წავშალოთ ავტომატურად შექმნილი Window1.xaml ფაილი SolutionExplorer-ში და ჩავამატოთ მის ნაცვლად Window(WPF)-შაბლონით ახალი ფაილი NavExample.xaml სახელით.

3. გავხსნათ App.xaml ფაილი და შევცვალოთ მასში ატრიბუტი StartupUri="NavExample.xaml"

4. ავამუშავოთ პროექტი WpfApp5. დავრწმუნდეთ, რომ არაა შეცდომები.

5. გავხსნათ ფაილი NavExample.xaml და შევასწოროთ მასში დესკრიპტორული კოდი:

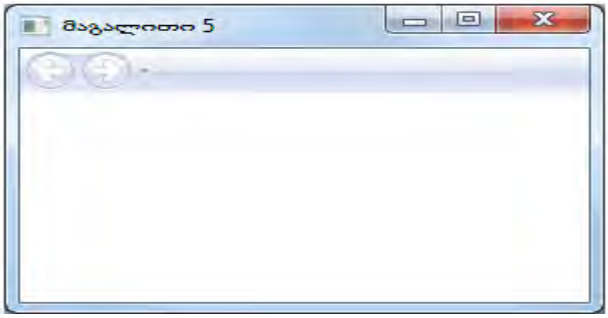
```
<NavigationWindow x:Class="WpfApp5.NavExample"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="მაგალითი 5"
    Height="300"
    Width="300"
    WindowStartupLocation="CenterScreen"
        >
</NavigationWindow>
```

6. გავხსნათ NavExample.xaml.cs ფაილი და შევასწოროთ C# კოდის ტექსტი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
```

```
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

using System.Windows.Navigation;

namespace WpfApp5
{
    public partial class NavExample : NavigationWindow
    {
        public NavExample()
        {
            InitializeComponent();

            //this.Navigate(new Page1());
        }
    }
}
```

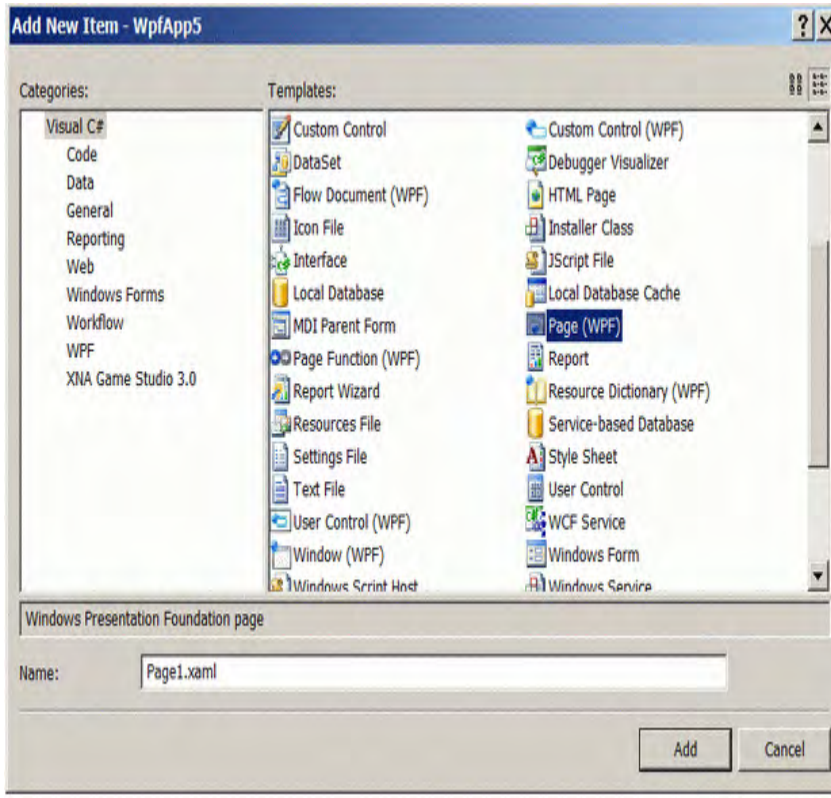
ავამუშავოთ პროექტი, შედეგი იქნება ნახ.2.30, ცარიელი გვერდი ნავიგაციური ელემენტით.



ნახ.2.30

7. ნავიგაციური კვანძის შიგთავსი წარმოდგენილი უნდა იყოს კლასით, რომელიც წარმოებულია ბიბლიოთეკის Page-კლასისგან. შევქმნათ სამი გვერდი და მივაბათ ნავიგაციის კვანძს Navigate() მეთოდის დახმარებით.

• დავამატოთ პროექტს ახალი Page ელემენტი Page1.xaml სახელით.



ნახ.2.31

8. შევასწოროთ Page1.xaml ფაილის ტექსტი:

```
<Page x:Class="WpfApp5.Page1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center"
FontSize="24">გვერდი 1</TextBlock>
```

```
        <TextBlock></TextBlock>
        <TextBlock>
      <Hyperlink Click="LinkClicked">მე-2 გვერდზე</Hyperlink>
        </TextBlock>
    </StackPanel>
</Page>
```

9. შევასწოროთ Page1.xaml.cs ფაილის კოდი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp5
{
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }

   private void LinkClicked(object sender, RoutedEventArgs e)
        {
            this.NavigationService.Navigate(page2);
        }
    }
}
```

9. დავამატოთ პროექტს ახალი Page ელემენტი Page2.xaml სახელით. შევასწოროთ xaml-კოდი:

```
<Page x:Class="WpfApp5.Page2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page2"
```

```
        >
    <StackPanel>
        <TextBlock TextAlignment="Center"
FontSize="24">გვერდი 2</TextBlock>
    </StackPanel>
</Page>
```

10. NavExample.xaml ფაილში დავამატოთ Source-ატრიბუტი, რომელიც მიუერთდება კარკასის გაშვებისას Page1.xaml საწყისი გვერდის შიგთავსს.

```
<NavigationWindow x:Class="WpfApp5.NavExample"
    ...
    WindowStartupLocation="CenterScreen"
    Source="Page1.xaml"
        >
</NavigationWindow>
```

11. დავალება: აამუშავეთ პროექტი. შეამოწმეთ ღილაკების მუშაობისუნარიანობა.

დასკვნა:

ამგვარად, ჩვენ შევქმენით კარკასი და ორი ცარიელი გვერდი, რომლებიც არაფერს არ აკეთებს. თითოეული გვერდი ავტონომიურია, შეიძლება მათი შევსება ტულბოქსის ელემენტებით.

გვერდებს შორის გადასვლისას საჭიროა ვიცოდეთ ინფორმაციის გადაცემა ერთი გვერდიდან მეორეში. უნდა არსებობდეს საერთო საფოსტო ყუთი, რომელიც არ იქნება დამოკიდებული გვერდებზე.

WPF-ში მონაცემების გადასაცემად გვერდებს შორის იყენებენ ლექსიკონს (წყვილების მასივი: „გასაღები-მნიშვნელობა“) Application.Current.Properties, ან ინფორმაციის „ჩაკერვას“ უშუალოდ ახალი გვერდის ობიექტში.

(მაგალითის გაგრძელება)

12. Page1-ზე დავამატოთ სახელმინიჭებული ტექსტური ველი შემდეგი სახით:

```
<Page x:Class="WpfApp5.Page1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page1"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center"
                FontSize="24">გვერდი 1</TextBlock>
        <TextBlock></TextBlock>
        <Label>შეიტანეთ თქვენი სახელი: </Label>
        <TextBox Name="nameBox" Width="200"></TextBox>
        <TextBlock></TextBlock>
        <TextBlock>
     <Hyperlink Click="LinkClicked">მე-2 გვერდზე</Hyperlink>
        </TextBlock>
    </StackPanel>
</Page>
```

TextBox-ობიექტს მივანიჭეთ სახელი, რათა შეიძლებოდეს მასზე მიმართვა კოდიდან.

13. შევცვალოთ Page1 გვერდის კოდი შემდეგი სახით;

```
using System;
...
namespace WpfApp5
{
  public partial class Page1 : Page
   {
    public Page1()
        {
            InitializeComponent();
        }
 private void LinkClicked(object sender, RoutedEventArgs e)
        {
          Page2 page2 = new Page2();
          page2.Message = nameBox.Text + " !!!"; // ინფორმაციის
                                                 //„ჩაკერვა“ ობიექტში
          this.NavigationService.Navigate(page2);
        }
    }
}
```

14. დავამატოთ Page2 გვერდზე სახელმინიჭებული ტექსტური ჭდე და ჰიპერლინკი Page3-ზე გადასასვლელად. აგრეთვე მომამზადეთ გვერდის მოვლენა Loaded და მოვლენა Click ჰიპერლინკისთვის.

```
<Page x:Class="WpfApp5.Page2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page2"
    Loaded="Page_Loaded"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center"
                FontSize="24">გვერდი 2</TextBlock>
        <TextBlock></TextBlock>
        <TextBlock>მოგესალმებით </TextBlock>
        <Label Name="nameLabel"></Label>
        <TextBlock Margin="0,10">   <!--Отступ сверху-->
 <Hyperlink Click="LinkClicked">მე-3 გვერდზე</Hyperlink>
        </TextBlock>
    </StackPanel>
</Page>
```

Label ობიექტს მივანიჭეთ სახელი, რათა კოდიდან შეიძლებოდეს მასზე მიმართვა.

15. დავამატოთ Page2-ის კოდს public თვისება, ტექსტურ ჭდეზე გადაცემული ტექსტის მისანიჭებელი კოდი Loaded-მოვლენაში და შემდეგ გვერდზე გადასვლის კოდი.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

```
namespace WpfApp5
{
    public partial class Page2 : Page
    {
        public Page2()
        {
            InitializeComponent();
        }

        string message;
        public string Message
        {
            set { message = value; }
        }
   private void Page_Loaded(object sender, RoutedEventArgs e)
        {
            nameLabel.Content = message;
        }
   private void LinkClicked(object sender, RoutedEventArgs e)
        {
            Page3 page3 = new Page3();
            this.NavigationService.Navigate(page3);
        }
    }
}
```

16. აამუშავეთ აპლიკაცია. ინფორმაცია გადაეცემა, ოღონდ ღილაკის ამუშავებით.

(!) ნავიგაციის ღილაკით ახალი ინფორმაცია ტექსტური ველიდან არ გადაიცემა. ინფორმაცია აიღება ისტორიის ჟურნალიდან.

17. Page3 გვერდისთვის შეავსეთ xaml-კოდი:

```
<Page x:Class="WpfApp5.Page3"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page3">
    <StackPanel>
        <!--გვერდის კონტექსტის სათაური -->
```

```
    <TextBlock TextAlignment="Center"
               FontSize="24">გვერდი 3
        <TextBlock.Margin>0,0,0,10</TextBlock.Margin>
    </TextBlock>
<!--პირველი ღილაკი-->
    <Button Content="Push Me!" FontSize="22" Width="175"
                        Height="50" Click="Button_Click">
        <Button.Effect>
            <DropShadowEffect />
        </Button.Effect>
    </Button>
<!--მეორე ღილაკი-->
    <Button FontSize="22" Height="50" Width="175"
                       Margin="0,10" Click="Button_Click">
        "დამკლიკე"
        <Button.Effect>
            <DropShadowEffect />
        </Button.Effect>
        <Button.Foreground>
      <LinearGradientBrush StartPoint="1,0" EndPoint="0,0">
              <GradientStop Color="Red" Offset="0" />
              <GradientStop Color="Orange" Offset=".17" />
              <GradientStop Color="Yellow" Offset=".33" />
              <GradientStop Color="Green" Offset=".5" />
              <GradientStop Color="CornflowerBlue"
                                          Offset=".67" />
             <GradientStop Color="Blue" Offset=".84" />
            <GradientStop Color="BlueViolet" Offset="1" />
            </LinearGradientBrush>
        </Button.Foreground>
        <Button.Background>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
              <GradientStop Color="Red" Offset="0" />
             <GradientStop Color="Orange" Offset=".17" />
             <GradientStop Color="Yellow" Offset=".33" />
              <GradientStop Color="Green" Offset=".5" />
```

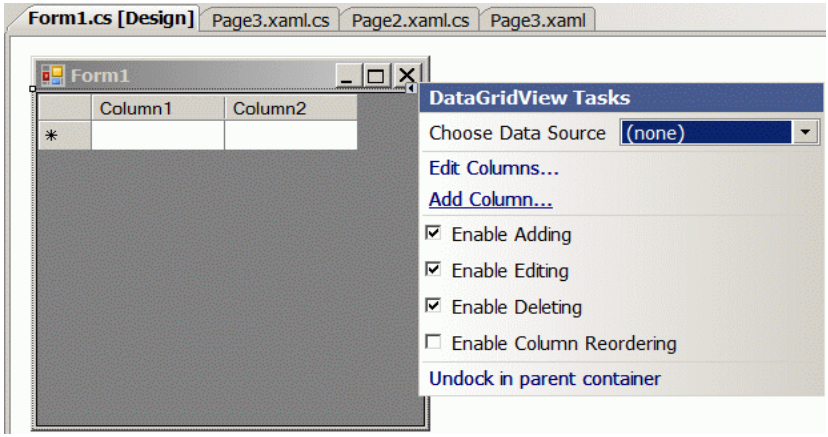
```
                <GradientStop Color="CornflowerBlue"
                                          Offset=".67" />
                <GradientStop Color="Blue" Offset=".84" />
              <GradientStop Color="BlueViolet" Offset="1" />
               </LinearGradientBrush>
            </Button.Background>
        </Button>
    </StackPanel>
</Page>
```

Click - მოვლენის დამმუშავებელი უნდა ჩაიწეროს ხელით, რათა ის შეიქმნას კოდის ნაწილში.

18. სანამ შევავსებთ Page3 გვერდის კოდის ნაწილს, საჭიროა პროექტს დაემატოს ახალი ფორმა. ამით შესაძლებელია WPF და Windows Forms ტექნოლოგიების ერთობლივად მუშაობის დემონსტრირება. ღილაკებზე დავდოთ ფორმის ამუშავების კოდი.

19. დავამატოთ WpfApp5-ში ახალი ფორმა Form1.cs

20. Form1 ფორმაზე ტულბოქსიდან დავდოთ DataGridView ელემენტი. დავაყენოთ მისი თვისება Dock=Fill და დავამატოთ ინტელექტუალური დესკრიპტორიდან (SmartTag) ორი სვეტი



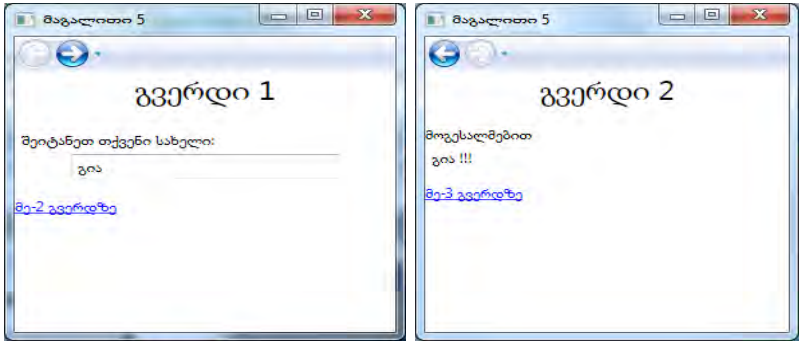
ნახ.2.32

21. ღილაკების საერთო დამმუშავებელი Page3 კოდის ნაწილში შევავსოთ ასე:

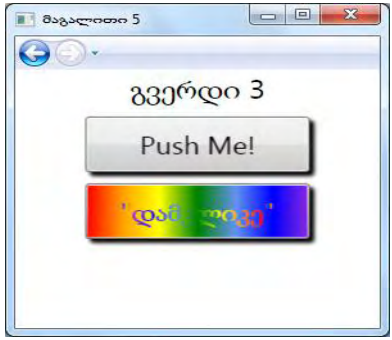
```
//--- ლისტინგი -------
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApp5
{
    public partial class Page3 : Page
    {
        public Page3()
        {
            InitializeComponent();
        }
 private void Button_Click(object sender, RoutedEventArgs e)
        {
            Form1 frm = new Form1();
            frm.ShowInTaskbar = false; // ფორმის ღილაკი არ
                                       //გამოჩნდეს ამოცანების პანელზე
            frm.Show();
        }
    }
}
```

22. ავამუშავოთ პროექტი. დავაკვირდეთ ახალი გვერდის დიზაინს. იხსნება Form1 -იც, რაც ადასტურებს WPF და Windows Form ტექნოლოგიების ერთობლივი მუშაობის შესაძლებლობას.

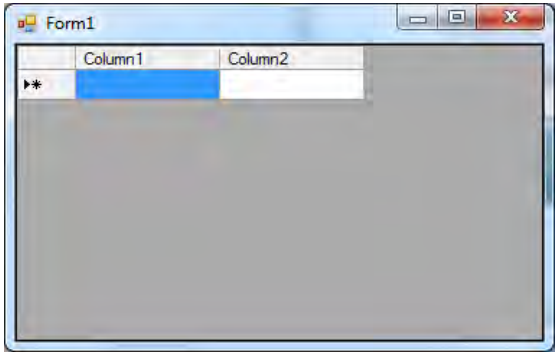
შედეგები (ნახ.2.32–ა,ბ,გ):



ნახ.2.32–ა



ნახ.2.32–ბ



ნახ.2.32–გ

2.7. WPF-ში ფუნჯის სახეები და ფერების შერჩევა
(ლაბორატორიული სამუშაო N 7)

მიზანი: WPF-აპლიკაციების დამუშავება მომხმარებელთა ინტერფეისების გრაფიკული ვიზუალური მართვის ელემენტებით, კერძოდ ფუნჯებისა და ფერთა პალიტრის გამოყენებით. კერძოდ განხილულ იქნება:

- გვერდი_1. SolidColorBrush - ერთგვაროვანი ფუნჯის გამოყენება
- გვერდი_2. Brushes კლასის ფერების გამოყენება
- გვერდი_3. LinearGradientBrush - წრფივი გრადიენტული ფუნჯის გამოყენება
- გვერდი_4. გრადიენტის მიმართულების პერიოდული ცვლა
- გვერდი_5. საფეხურებრივი შეფერადება გრადიენტებით
- გვერდი_6. RadialGradientBrush - შეფერადება რადიალური გრადიენტით

1. თეორიული ნაწილი: ფუნჯის სახეები WPF-ში

WPF–ში ობიექტების გარეგნული დიზაინის გასაფორმებლად არსებობს როგორც უშუალო ასაწყობი თვისებები, ასევე ფუნჯი–ინსტრუმენტები (Brush). თვისებები შემდეგი სახისაა:

- გამჭვირვალობა (Opacity)
- ხილვადობა (Visibility)
- შევსება (Fill)
- დაჩრდილვა (Stroke) – /შტრიხები/
- ფონი (Background)
- წინა პლანი (Foreground)
- საზღვარი (BorderBrush)
- გამჭვირვალობის ნიღაბი (Opacity masks).

ფუნჯები შემდეგი სახისაა:

- ერთფეროვანი (Solid color brush)
- წრფივი გრადიენტული (Linear gradient brush)
- რადიალური გრადიენტული (Radia l gradient brush)
- რასტრული გამოსახულების (Image brush)
- ვექტორული გამოსახულების (Drawing brush)
- ვიზუალური ეფექტების (Visual brush).

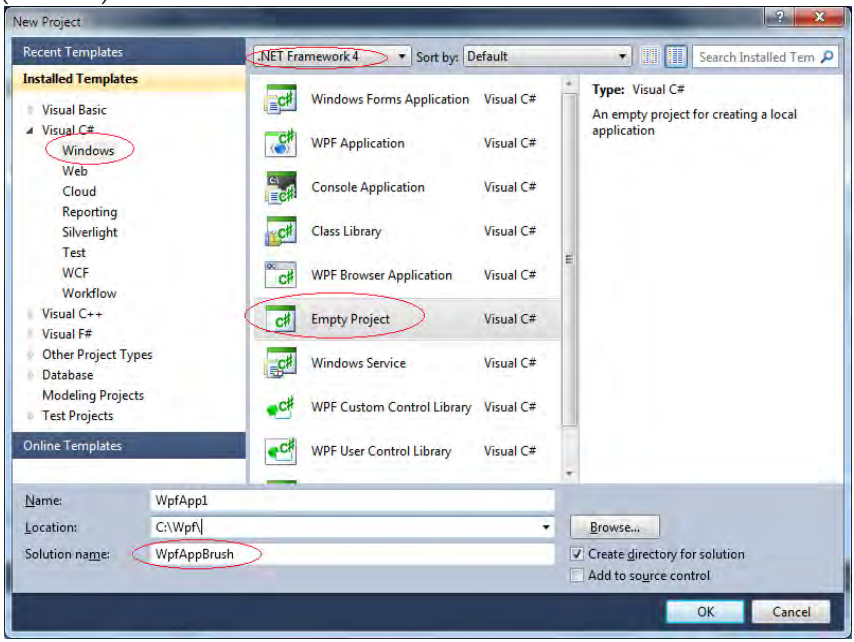
შესაძლებელია ფუნჯების კონვერტირება რესურსებად და შემდგომ მათი მრავალჯერადი გამოყენება სხვადასხვა ობიექტებისთვის.

WPF–ს აქვს სპეციალური რასტრული ეფექტების (Bitmap effects) მიღწევის საშუალებანი:

- არამკაფიო (Blur)
- გარე ნათების (Outer glow)
- ჩრდილი (Drop shadow)
- ზოლი (Bevel) – კანტი
- რელიეფური (Emboss) –ჭედური

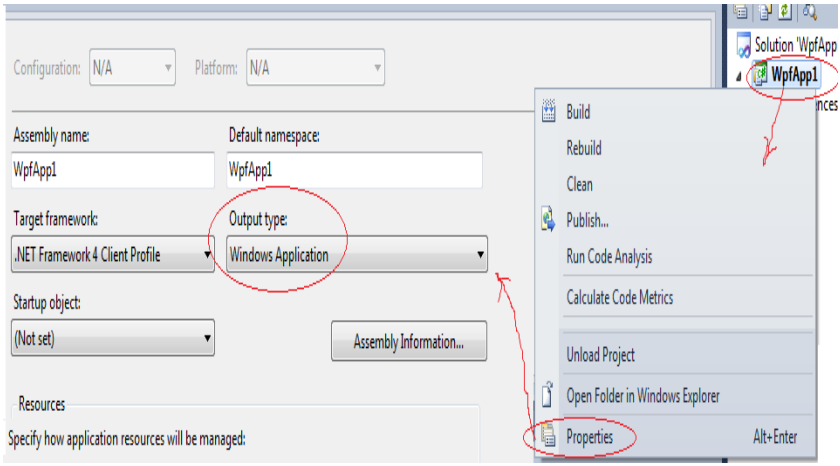
2. პრაქტიკული ნაწილი – აპლიკაციის აგება

1. Visual Studio-ს Solution Explorer-ში WpfAppBrush სახელით შევქმნათ ახალი ცარიელი პროექტი (**Empty Project**) WpfApp1 (ნახ.2.33) :



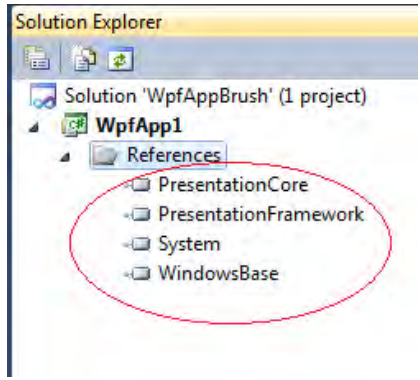
ნახ.2.33

2. WpfApp1-ისთვის გამოვიძახოთ კონტექსტური მენიუ და **Properties**–ის **Application**–ში Output type პარამეტრისთვის ავირჩიოთ მნიშვნელობა **Windows Application.**

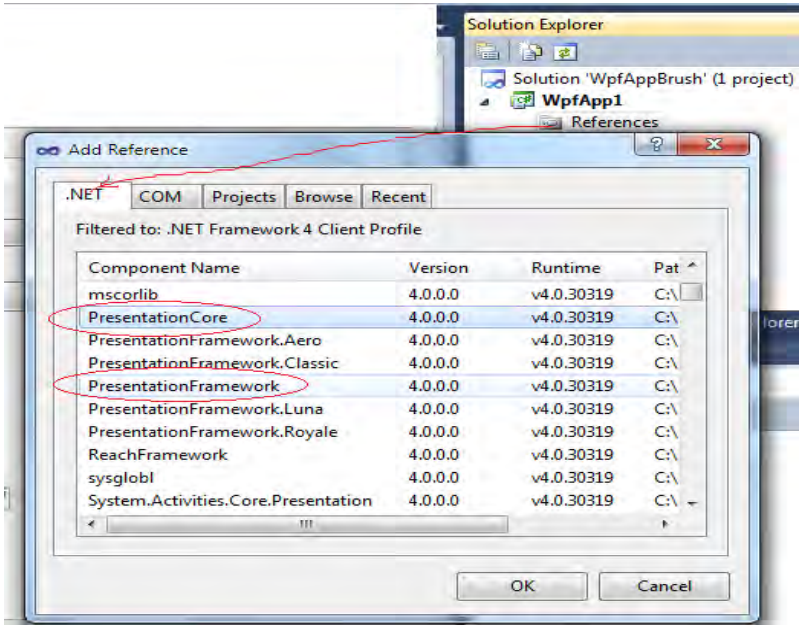


ნახ.2.34

3. პროექტის **References** კვანძს დავუმატოთ **.NET Framework**–ის WPF–პლატფორმის შემდეგი ბიბლიოთეკები: PresentationCore, PresentationFramework, WindowsBase და System (ესაა საერთო ბიბლიოთეკა აპლიკაციის ასამუშავებლად). 2.35–36 ნახაზებზე ნაჩვენებია ეს პროცედურები.

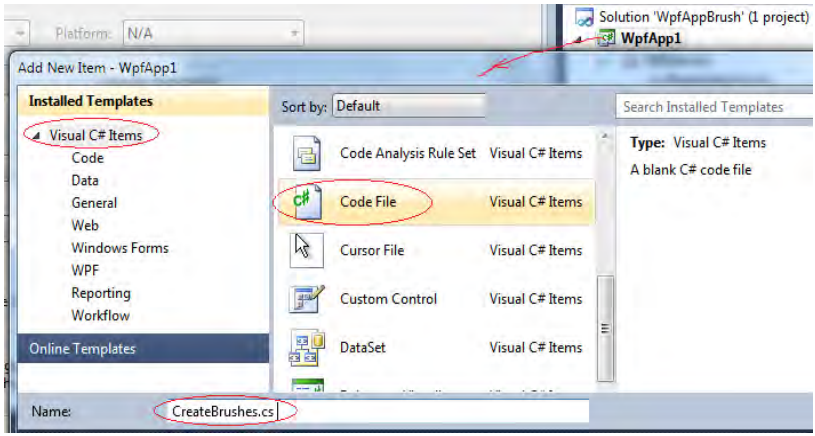


ნახ.2.35



ნახ.2.36

4. პროექტს დავუმატოთ ახალი ფაილი სახელით CreateBrushes.cs:



ნახ.2.37

5. შევავსოთ ფაილი შემდეგი კოდით:

```
// — ლისტინგი —
using System;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Navigation;
using System.Windows.Controls;
using System.Reflection;

namespace WpfApp1
{
    class CreateBrushes : NavigationWindow
    {
        // შესასვლელი წერტილი
        [STAThread] // ატრიბუტი ერთნაკადიანი აპლიკაციისთვის
        public static void Main()
        {
            Application app = new Application();
            app.Run(new CreateBrushes());
        }

        // კონსტრუქტორი
        public CreateBrushes()
        {
            this.Width = 300;
            this.Height = 300;
           this.Title = "1-ელი მაგალითის აპლიკაციის კარკასი";
            //this.ShowsNavigationUI = false;
            // ნავიგაციური ინტერფეისის დამალვა
        }
    }
}
```

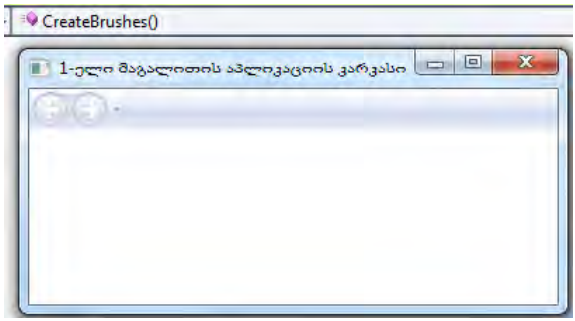
ამოცანა:

ავაგოთ მრავალგვერდიანი აპლიკაცია რამდენიმე მაგალითისთვის (პროექტებისთვის). წინასწარ გვერდებისთვის შევქმენით კარკასი შიგთავსით, რომელშიც ნავიგაციის კვანძია. იგი შეიძლება გაითიშოს ShowsNavigationUI – თვისებით.

კარკასი შექმნილია ნულიდან სუფთა **C#** კოდის გამოყენებით. არ ვიყენებთ XAML–ს. გვერდული კარკასი იქმნება

NavigationWindow კლასით, რომელიც წარმოებულია Window კლასიდან. ჩვენ ვაფართოვებთ NavigationWindow ბიბლიოთეკურ კლასს მომხმარებლის კლასით CreateBrushes. ამგვარად, მისთვის მემკვიდრეობით ხელმისაწვდომი იქნება Window საბაზო კლასის წევრები (მონაცემები და მეთოდები).

6. ავამუშავოთ აპლიკაცია – მივიღებთ „მკვდარ“ კარკასს (გამორთულია ისრები).

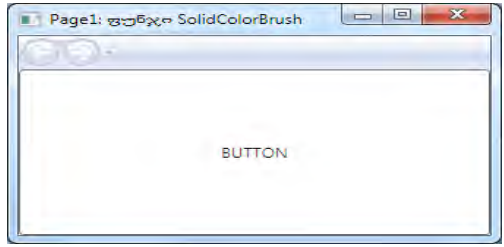


ნახ.2.38

შემდგომში ამ კარკასში უნდა მოვათავსოთ ახალ–ახალი გვერდები, რომლებიც იქმნება Page ბიბლიოთეკური კლასის გაფართოებით და გარკვეული თვისებების მინიჭებით. კარკასის ნავიგაციური მექანიზმი ამოქმედდება NavigationWindow საბაზო კლასის ხელშეწყობით.

2.7.1. გვერდი_1: ერთგვაროვანი ფუნჯი SolidColorBrush

ავაგოთ პირველი გვერდი, რომელზეც კლიენტის მთლიან არეში მოთავსებული იქნება ღილაკი “BUTON“. ეს ღილაკი უნდა იყოს ერთგვაროვანი, ნაცრისფერის გრადაციით (მაუსის კურსორის მოცილებისას, და მოცისფრო გრადაციით – მიახლოვებისას). გვერდი აიგება ხელით, როგორც გაფართოებული კლასი Page ბიბლიოთეკური კლასიდან.



ნახ.2.39

8. ჩავამატოთ CreateBrushes.cs კოდში Page1 გვერდის შექმნის კოდი.

```
namespace WpfApp1
{
 // კლასი - Page1 გვერდის გაფართოებისთვის
  class Page1 : Page
  {
    // ფუნჯის ველის შექმნა და ინიციალიზაცია თეთრი ფერით
   SolidColorBrush brush = new  SolidColorBrush(Colors.White);
   Button btnPage1 = new Button();

   public Page1()
    {
      this.WindowTitle = "Page1: ფუნჯი SolidColorBrush";
      // ღილაკის შექმნა
      btnPage1.Content = "BUTTON";
      btnPage1.Background = brush;
     // დამმუშავებლების რეგისტრაცია
     btnPage1.Click += new outedEventHandler(btnPage1_Click);
     btnPage1.MouseMove += new
                        MouseEventHandler(btnPage1_MouseMove);
     this.Content = btnPage1; // ღილაკის მოთავსება გვერდზე
    }

      // ღილაკის ფონის ფერის შეცვლა Page1 გვერდზე
    void btnPage1_MouseMove(object sender, MouseEventArgs e)
      {
       // კლიენტის სამუშაო არის სიგანე ფანჯრის ჩარჩოს გარეშე
         double width = btnPage1.ActualWidth;
```

```
            // კლიენტის სამუშაო არის სიმაღლე ფანჯრის ჩარჩოს და
                                                    // სათაურის გარეშე
            double height = btnPage1.ActualHeight;
              // კურსორის წერტილი ფანჯრის კლიენტის არეზე
            Point ptMouse = e.GetPosition(btnPage1);
              // კლიენტის არის ცენტრი
            Point ptCenter = new Point(width / 2, height / 2);
             // კურსორის გადახრა ცენტრიდან
            Vector vectMouse = ptMouse - ptCenter;
              // კურსორის მდებარეობის კუთხე
            double angle = Math.Atan2(vectMouse.Y, vectMouse.X);
             // ჩაწერილი ელიფსი
            Vector vectEllipse = new Vector(width / 2 *
                   Math.Cos(angle), height / 2 * Math.Sin(angle));
    // იზოკლინა (ღილაკის ცენტრში შავი) მრგვალდება ერთ ბაიტამდე
            Byte byLevel = (byte)(255 * (Math.Min(1,
                        vectMouse.Length / vectEllipse.Length)));
            // ველზე მიბმა
            Color color = brush.Color;
               // ფერები, იზოკლინის პროპორციულები
            color.R = color.G = color.B = byLevel;

                brush.Color = color;// იცვლება ფანჯრის ფონის
                            // ფერი თანაბარი შავი ფერის იზოკლინით
            }

            // გადასვლა მეორე გვერდზე
            Page2 page2;
            void btnPage1_Click(object sender, RoutedEventArgs e)
            {
                if (!this.NavigationService.CanGoForward)
                   page2 = new Page2();// იქმნება მხოლოდ ერთხელ
                this.NavigationService.Navigate(page2);
            }
        }
    }
```

9. კარკასის კოდის კონსტრუქტორში ჩავამატოთ საწყისი გვერდის ობიექტის შექმნის კოდი:

```
// კონსტრუქტორი
  public CreateBrushes()
  {
      ...
      // საწყისი გვერდის შექმნა
   Page1 page1 = new Page1();
   this.Content = page1;    // გვერდის მოთავსება კარკასში
  }
```

10. ავამუშავოთ პროგრამა, მივიღებთ 2.39 ნახაზს, კარკასში მოთავსდა გვერდი ღილაკით. ღილაკი ჯერ არ ფუნქციონირებს.

2.7.2. გვერდი_2: Brushes კლასის ფერების გამოყენება:

11. ავაგოთ მეორე გვერდი Page2, რომელზეც გადასვლა განხორციელდება BUTTON ღილაკით. კოდის ტექსტი იქნება ასეთი:

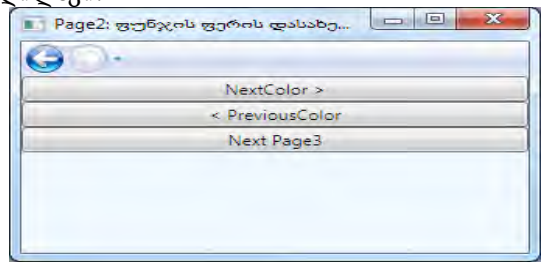
```
namespace WpfApp1
{
    // კლასი - Page2 გვერდის გაფართოება ----
    class Page2 : Page
    {
        int index = 0;// ფერის ნომერი
        PropertyInfo[] props;// თვისებების მასივი

        public Page2()
        {
 // რეფლექსიის გამოყენება Brushes კლასის თვისებების წასაკითხად
            props = typeof(Brushes).GetProperties(
                BindingFlags.Public | BindingFlags.Static);

            // პანელის შექმნა
            StackPanel stackPanel = new StackPanel();
            this.Content = stackPanel;// გვერდთან მიერთება

            Button btn = new Button();
            btn.Name = "ButtonNextColor";
            btn.Content = "NextColor >";
```

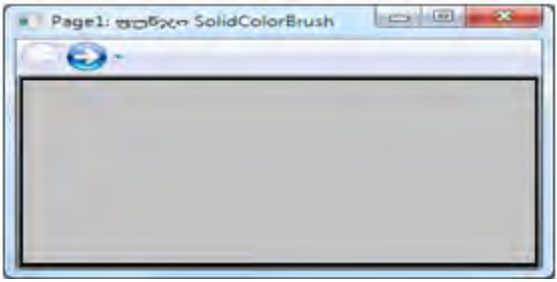
```
            btn.Click += new RoutedEventHandler(btn_Click);
            stackPanel.Children.Add(btn);// პანელზე დამატება

            btn = new Button();
            btn.Content = "< PreviousColor";
            btn.Click += new RoutedEventHandler(btn_Click);
            stackPanel.Children.Add(btn);

            btn = new Button();
            btn.Content = "Next Page3";
         btn.Click += new RoutedEventHandler(btnPage2_Click);
            stackPanel.Children.Add(btn);

            // აქტიურდება გვერდის ყოველი წარმოდგენისას
         this.Loaded += new RoutedEventHandler(Page2_Loaded);
        }
      void Page2_Loaded(object sender, RoutedEventArgs e)
        {
            // ვარიანტი
            //this.NavigationService.LoadCompleted +=
                        NavigationService_LoadCompleted;
            SetTitleAndBackground();
        }
      void NavigationService_LoadCompleted(object sender,
                                    NavigationEventArgs e)
        {
            // ვარიანტი
            //SetTitleAndBackground();
        }
// გვერდის სათაურის და ფერის შეცვლის ღილაკების დამმუშავებლები
      void btn_Click(object sender, RoutedEventArgs e)
        {
        // ღილაკის ამოცნობა სახელით და ინდექსის კორექტირება
            if (((Button)sender).Name == "ButtonNextColor")
                index += 1;
            else
                index += props.Length - 1;

            index %= props.Length;// მოდულის (%) ოპერაცია
            SetTitleAndBackground();
        }
    // გვერდისთვის სათაურის და ფონის ფერის დაყენება
```

```
    void SetTitleAndBackground()
     {
      this.WindowTitle = "Page2: ფუნჯის ფერის დასახელება-"+
                                              props[index].Name;
  this.Background = (Brush)props[index].GetValue(null, null);
     }
    // გადასვლა მესამე გვერდზე
       Page3 page3;
     void btnPage2_Click(object sender, RoutedEventArgs e)
       {
           if (!this.NavigationService.CanGoForward)
              page3 = new Page3();// იქმნება მხოლოდ ერთხელ
           this.NavigationService.Navigate(page3);
       }
   }
}
```

Page1–ის კოდის ბოლოში ნაჩვენები იყო Page2–ის გამოძახების ფრაგმენტი, ახალ გვერდზე გადასასვლელად.

12. ავამუშავოთ პროგრამა და კარკასის ნავიგაციის მარჯვენა ღილაკი:





ნახ.2.40

2.7.3. გვერდი_3: წრფივი გრადიენტული ფუნჯის – LinearGradientBrush გამოყენება

ერთგვაროვანი ფუნჯის ალტერნატიულია გრადიენტული ფუნჯი, რომლითაც შესაძლებელია ორი ან მეტი ფერის ერთმანეთში თანდათანობითი გადასვლის იმიტაცია. მარტივ შემთხვევაში წრფივი გრადიენტული ფუნჯის შესაქმნელად საკმარისია ორი ნაპირა წერტილის და მათ შორის ფერების განსაზღვრა. გრადიენტული ფერთა განაწილება მოხდება წერტილებსშორის შემაერთებელი წრფის პერპენდიკულარულად და ზედაპირი მართკუთხედის ფარგლებში შეივსება მითითებული ფერებით.

წრფივი გრადიენტის ფუნჯის ობიექტის კონსტრუქტორს აქვს რამდენიმე გადატვირთვა, რომელთაგან ერთში მიეთითება: - ორი წერტილი და ორი ფერი; - ორი ფერი, გრადიენტის ვექტორის მიბმის საწყისი წერტილი და შეფერადების მიმართულების დახრის კუთხე.

წერტილები იძლევა ფარდობით (გამოუცხადებლად) და აბსოლუტურ კოორდინატებს მართკუთხედის შიგნით, რომელიც განისაზღვრება MappingMode თვისებით და მისი მნიშვნელობით BrushMappingMode ჩამონათვალში.

13. შევქმნათ ახალი Page3 გვერდი CreateBrushes.cs ფაილში, რომელზეც ილუსტრირებული იქნება წრფივი გრადიენტული შეფერადება.

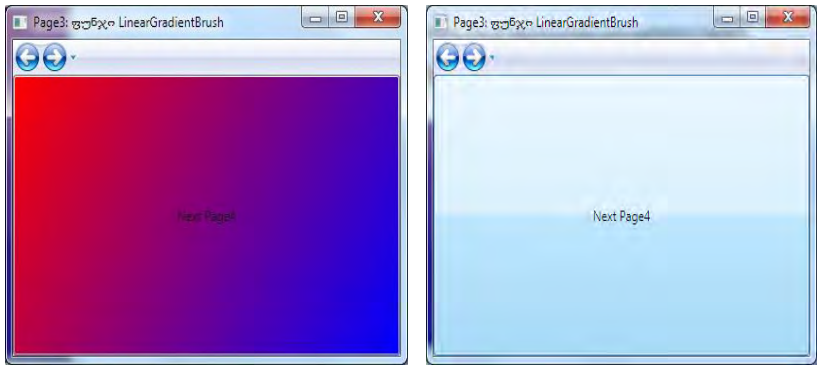
```
namespace WpfApp1
{
  class Page3 : Page
  {
    public Page3()
     {
      this.WindowTitle = "Page3: ფუნჯი LinearGradientBrush";
      Button btn = new Button();
      btn.Content="Next Page4";
      btn.Click += new RoutedEventHandler(btn3_Click);
      this.Content = btn;
     // გრადიენტის შექმნა და მიერთება
     LinearGradientBrush brush = new LinearGradientBrush(
  Colors.Red, Colors.Blue, new Point(0, 0), new Point(1, 1));
```

```
        btn.Background = brush;
    }
      // გადასვლა მეოთხე გვერდზე
      Page4 page4;
      void btn3_Click(object sender, RoutedEventArgs e)
       {
         if (!this.NavigationService.CanGoForward)
           page4 = new Page4();// იქმნება მხოლოდ ერთხელ

         this.NavigationService.Navigate(page4);
       }
    }
}
```

14. ჩავსვათ Page2 კლასის ღილაკში Page3 გვერდის ეგზემპლარის შექმნის კოდი:

```
  // გადასვლა მესამე გვერდზე
        Page3 page3;
        void btnPage2_Click(object sender, RoutedEventArgs e)
        {
            if (!this.NavigationService.CanGoForward)
               page3 = new Page3();// იქმნება მხოლოდ ერთხელ

            this.NavigationService.Navigate(page3);
        }
```

15. ავამუშავოთ აპლიკაცია, მივიღებთ (ნახ.2.41):



ნახ.2.41. მაუსის კურსორის მდებარეობით იცვლება ფერები

2.7.4. გვერდი_4: გრადიენტის მიმართულების პერიოდული ცვლა

თუ მონაკვეთი ნაკლებია შემზღუდველ მართკუთხედზე, მაშინ ნაპირა წერტილებს გარეთ ფერის გრადიენტი იცვლის მიმართულებას საწინააღმდეგოზე და ფერის შევსება გრძელდება ისეთივე ინტენსიურობით, როგორც ეს იყო ინტერბალს შიგნით. LinearGradientBrush ფუნჯის ასეთი ქცევა განისაზღვრება SpreatMethod-ის GradientBrush თვისებით და მისი მნიშვნელობით GradientSpreadMethod -ჩამონათლიდან.

16. შევქმენათ ახალი Page4 გვერდი CreateBrushes.cs -ში, რომელშიც ილუსტრირებული იქნება ფერების ტალღისებური გრადიენტული შევსება.

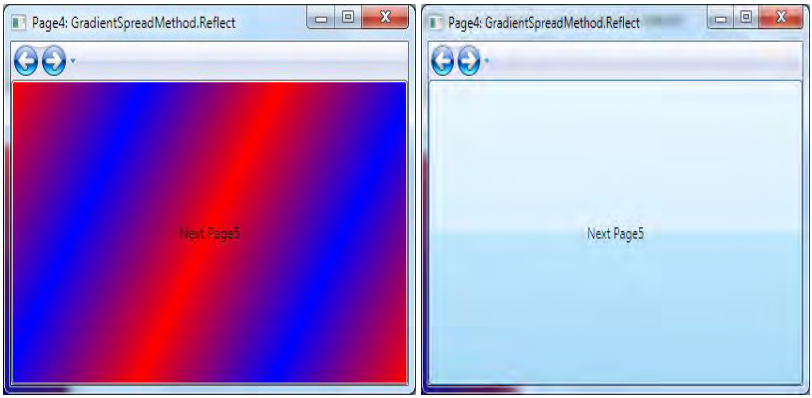
```
namespace WpfApp1
{
  class Page4 : Page
  {
   public Page4()
   {
    this.WindowTitle = "Page4: GradientSpreadMethod.Reflect";
    Button btn = new Button();
    btn.Content = "Next Page5";
    btn.Click += new RoutedEventHandler(btn4_Click);
    this.Content = btn;

     // გრადიენტის შექმნა და მიერთება
    LinearGradientBrush brush = new LinearGradientBrush(
       Colors.Red, Colors.Blue, new Point(0, 0), new
                                          Point(0.25, 0.25));
    brush.SpreadMethod = GradientSpreadMethod.Reflect;
    btn.Background = brush;
   }
     // გადასვლა მეხუთე გვერდზე
        Page5 page5;
      void btn4_Click(object sender, RoutedEventArgs e)
        {
            if (!this.NavigationService.CanGoForward)
               page5 = new Page5();// იქმნება მხოლოდ ერთხელ
```

```
            this.NavigationService.Navigate(page5);
        }
    }
}
```

17. Page3 კოდში ჩავსვათ Page4-ის ეგზემპლარის შექმნის კოდი:

```
// გადასვლა მეოთხე გვერდზე
Page4 page4;
void btn3_Click(object sender, RoutedEventArgs e)
{
    if (!this.NavigationService.CanGoForward)
       page4 = new Page4();// იქმნება მხოლოდ ერთხელ

    this.NavigationService.Navigate(page4);
}
```

18. ავამუშავოთ აპლიკაცია და გამოვცადოთ Page4 კლასის მუშაობა წრფივი გრადიენტის მიმართულების პერიოდული ცვლილების დროს (ნახ.2.42).



ნახ.2.42

2.7.5. გვერდი_5: საფეხურებრივი შევსება გრადიენტებით

წრფივ გრადიენტს აქვს GradientStops კოლექცია, რომელიც შეიცავს GradientStop ობიექტებს. თითოეული ობიექტი იძლევა გრადიენტის საწყის ფერს და მარჯვენა საზღვარს ამ ფერით წინა-არსებულის შესავსებად. ამ დროს დამატებით გამოიყენება შევსების სასაზღვრო წერტილები, რომლებიც განისაზღვრება StartPoint და EndPoint თვისებებით. ჩავატაროთ ექსპერიმენტი, შევქმნათ გვერდი, დავდოთ ღილაკი და გავაფერადოთ ეს ღილაკი ვერტიკალური გრადიენტებით.

19. შევქმნათ Page5 გვერდი CreateBrushes.cs ფაილში. დავდოთ ღილაკი. ჩავწეროთ კოდი:

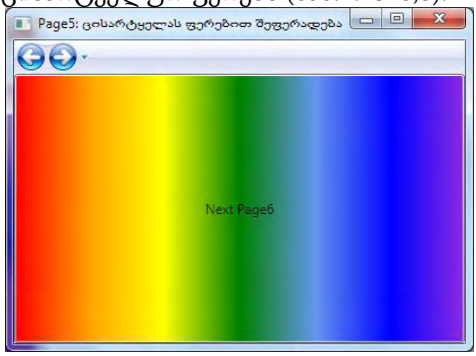
```
namespace WpfApp1
{
 class Page4 : Page
 {
   public Page4()
   {
    this.WindowTitle = "Page4: GradientSpreadMethod.Reflect";
    Button btn = new Button();
    btn.Content = "Next Page5";
    btn.Click += new RoutedEventHandler(btn4_Click);
    this.Content = btn;

    // გრადიენტის შექმნა და მიერთება
    LinearGradientBrush brush = new LinearGradientBrush(
        Colors.Red, Colors.Blue, new Point(0, 0), new
                                  Point(0.25, 0.25));
     brush.SpreadMethod = GradientSpreadMethod.Reflect;
     btn.Background = brush;
   }

        // გადასვლა მეხუთე გვერდზე
        Page5 page5;
        void btn4_Click(object sender, RoutedEventArgs e)
        {
            if (!this.NavigationService.CanGoForward)
               page5 = new Page5();// იქმნება მხოლოდ ერთხელ

            this.NavigationService.Navigate(page5);
```

```
        }
    }
}
```

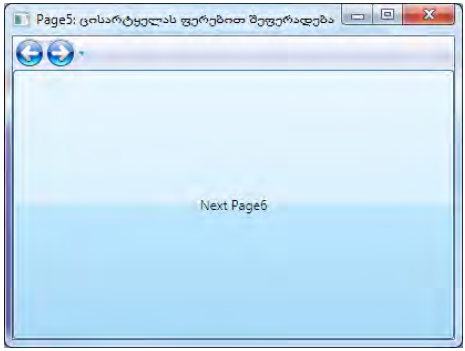
20. Page4 კოდში ჩავწეროთ Page5-ის ეგზემპლარის შექმნის კოდი:

```
// გადასვლა მეოთხე გვერდზე
        Page4 page4;
      void btn3_Click(object sender, RoutedEventArgs e)
        {
            if (!this.NavigationService.CanGoForward)
               page4 = new Page4();// იქმნება მხოლოდ ერთხელ

            this.NavigationService.Navigate(page4);
        }
```

21. ავამუშავოთ აპლიკაცია და Page5 გვერდზე დავაკვირდეთ ცისარტყელურ ფერებს (ნახ.2.43–ა,ბ):



ნახ.2.43–ა



ნახ.2.43–ბ

2.7.6. გვერდი_6: RadialGradientBrush რადიალური გრადიენტი

რადიალური გრადიენტი რეალიზდება RadialGradientBrush კლასით. განვიხილოთ მაგალითი.

22. შევქმნათ ახალი Page6 გვერდი CreateBrushes.cs ფაილში და შევაფერადოთ შემდეგი კოდით:

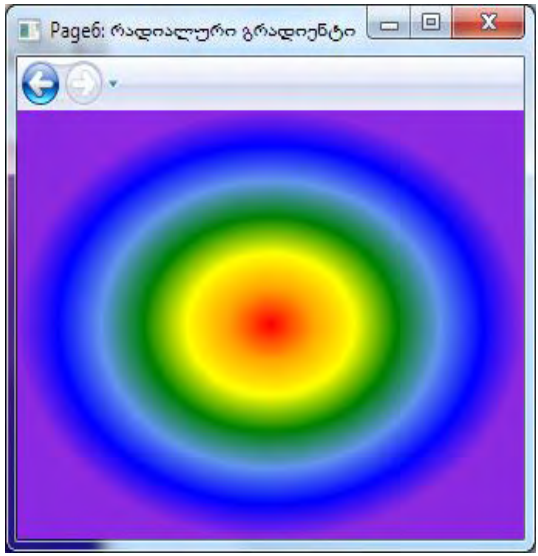
```
namespace WpfApp1
{
  class Page6 : Page
  {
    public Page6()
    {
      this.WindowTitle = "Page6: რადიალური გრადიენტი";

     // გრადიენტის შექმნა და მიერთება
      RadialGradientBrush brush = new RadialGradientBrush();
      this.Background = brush;

     // ცისარტყელას ფერები
brush.GradientStops.Add(new GradientStop(Colors.Red, 0));
brush.GradientStops.Add(new GradientStop(Colors.Orange, .17));
brush.GradientStops.Add(new GradientStop(Colors.Yellow, .33));
brush.GradientStops.Add(new GradientStop(Colors.Green, .5));
brush.GradientStops.Add(new GradientStop
                                (Colors.CornflowerBlue, .67));
brush.GradientStops.Add(new GradientStop(Colors.Blue, .84));
brush.GradientStops.Add(new GradientStop(Colors.BlueViolet, 1));
      }
   }
}
```

23. Page5 კოდში ჩავწეროთ Page6-ის ეგზემპლარის შექმნის კოდი:

```
// გადასვლა მეექვსე გვერდზე
        Page6 page6;
```

```
void btn5_Click(object sender, RoutedEventArgs e)
{
    if (!this.NavigationService.CanGoForward)
       page6 = new Page6();// იქმნება მხოლოდ ერთხელ
    this.NavigationService.Navigate(page6);
}
```

24. ავამუშავოთ აპლიკაცია და დავაკვირდეთ რადიალური გრადიენტის ფუნჯის მუშაობას.



ნახ.2.44

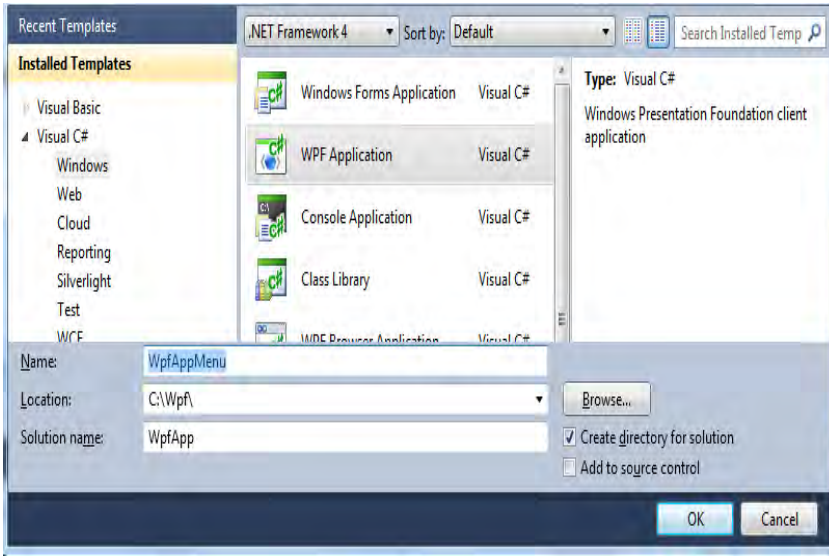
2.8. WPF-ის მართვის ელემენტები: Menu, ToolBar, TabControl და ToolTip

(ლაბორატორიული სამუშაო N 8)

მიზანი: WPF-აპლიკაციების ასაგებად Menu, ToolBar, TabControl და ToolTip მართვის ელემენტების გამოყენების შესწავლა. მომხმარებელთა ინტერფეისების ასაგებად WPF-ის ვიზუალური მართვის ელემენტების ათვისება მაიკროსოფთის ბიბლიოთეკის ელექტრონული ცნობარიდან:

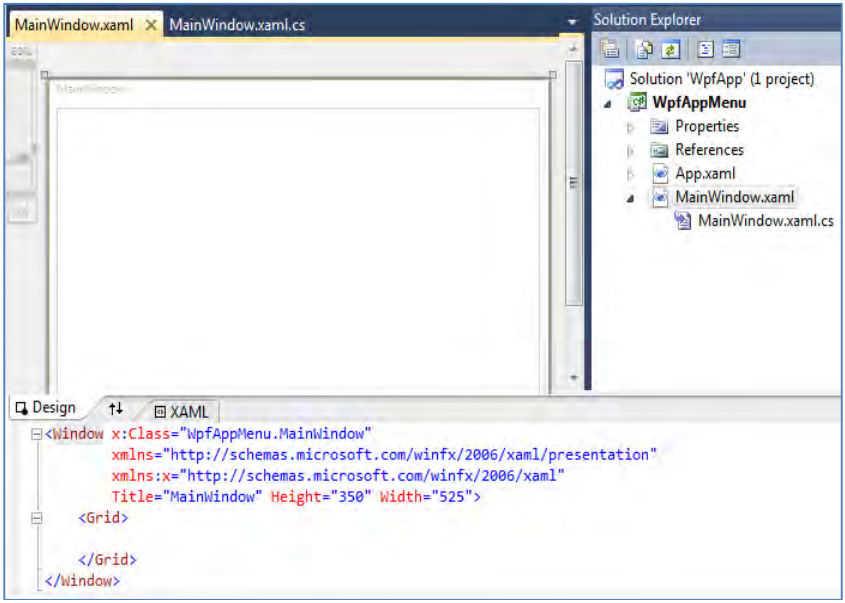
http://msdn.microsoft.com/en-en/library/ms752324.aspx

1. შევქმნათ ახალი პროექტი Visual Studio-ს Solution Explorer-ში WpfAppMenu სახელით (ნახ.2.45) :



ნახ.2.45

მიიღება პროექტის საწყისი მდგომარეობა (ნახ.2.46):



ნახ.2.46

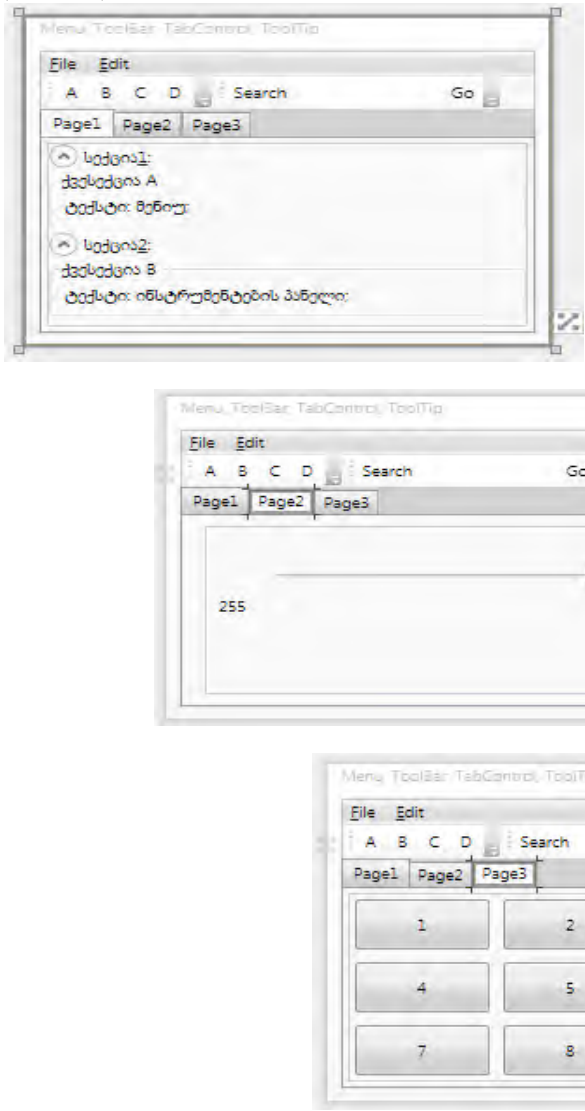
2. მომხმარებლის ინტერფეისის ფანჯრის დიზაინისთვის MainWindow.xaml კოდში ჩავწეროთ შემდეგი ტექსტი:

```
<Window x:Class="WpfApp2.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Menu, ToolBar, TabControl, ToolTip"
    SizeToContent="WidthAndHeight"

    ToolTipService.InitialShowDelay="0"
    ToolTipService.ShowDuration="500000" mc:Ignorable="d"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006" d:DesignHeight="270">

    <Window.ToolTip>
        <ToolTip x:Name="toolTip"
            Placement="RelativePoint"
```

```
            VerticalOffset="10"
            />
    </Window.ToolTip>

    <DockPanel Background="LightGray">
        <Menu DockPanel.Dock="Top">
            <MenuItem Header="_File">
              <MenuItem Header="E_xit" Click="ExitClicked" />
            </MenuItem>
            <MenuItem Header="_Edit">
                <MenuItem Header="_Cut" />
                <MenuItem Header="C_opy" />
                <MenuItem Header="_Paste" />
            </MenuItem>
        </Menu>

        <ToolBarTray DockPanel.Dock="Top">
            <ToolBar>
                <Button Width="23">A</Button>
                <Button Width="23">B</Button>
                <Button Width="23">C</Button>
                <Button Width="23">D</Button>
            </ToolBar>
            <ToolBar Header="Search">
                <TextBox Width="100" />
                <Button Width="23">Go</Button>
            </ToolBar>
        </ToolBarTray>

        <TabControl>
            <TabItem Header="Page1">
                <StackPanel>
                   <Expander Header="სექცია_1:"
                                    IsExpanded="True">
                      <GroupBox Header="ქვესექცია A">
                          <Label>ტექსტი: მენიუ; </Label>
                      </GroupBox>
```

```
                </Expander>
                <Expander Header="სექცია_2:"
                                IsExpanded="True">
                    <GroupBox Header="ქვესექცია B">
            <Label>ტექსტი: ინსტრუმენტების პანელი;</Label>
                    </GroupBox>
                </Expander>

            </StackPanel>
        </TabItem>
        <TabItem Header="Page2">
            <StackPanel Orientation="Horizontal"
                              Margin="5" Width="297">
              <TextBlock Name="value" Margin="10"
                    Text="255" Width="25" Height="23" />
               <Slider Name="slider" Width="241"
                     Minimum="0" Maximum="255" Value="255"
        ValueChanged="slider_ValueChanged" Height="77" />
            </StackPanel>
        </TabItem>
        <TabItem Header="Page3">
            <UniformGrid Rows="3" Columns="3">
         <Button ToolTip="ღილაკი 1" Margin="5">1</Button>
        <Button ToolTip="ღილაკი 2"  Margin="5">2</Button>
        <Button ToolTip="ღილაკი 3" Margin="5">3</Button>
        <Button ToolTip="ღილაკი 4" Margin="5">4</Button>
        <Button ToolTip="ღილაკი 5" Margin="5">5</Button>
        <Button ToolTip="ღილაკი 6" Margin="5">6</Button>
        <Button ToolTip="ღილაკი 7" Margin="5">7</Button>
        <Button ToolTip="ღილაკი 8" Margin="5">8</Button>
        <Button ToolTip="ღილაკი 9" Margin="5">9</Button>
      </UniformGrid>
    </TabItem>
  </TabControl>
 </DockPanel>
</Window>
```

შედეგში მიიღება სამი გვერდი Page1, Page2 და Page3 (ნახ.2.47).

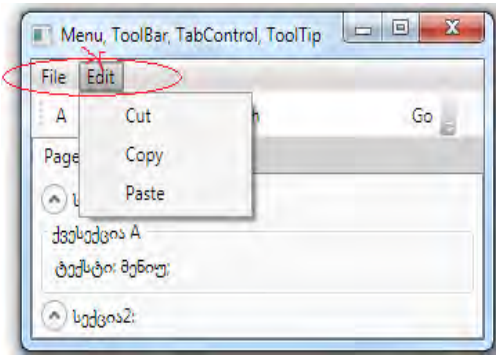


ნახ.2.47

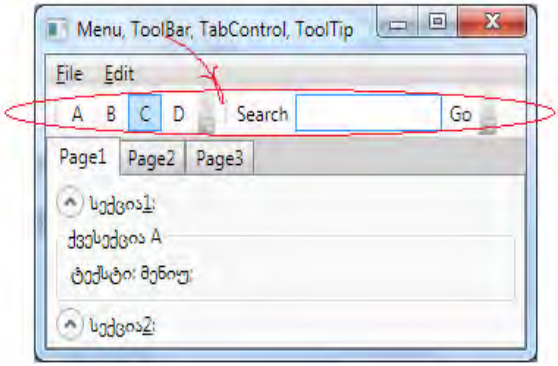
3. MainWindow.xaml.cs კოდისთვის შევიტანოთ შემდეგი ტექსტი:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfAppMenu
{
   public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

   private void ExitClicked(object sender, RoutedEventArgs e)
        {
            this.Close();
        }

   private void slider_ValueChanged(object sender,
                    RoutedPropertyChangedEventArgs<double> e)
        {
            value.Text = ((int)slider.Value).ToString();
        }
    }
}
```

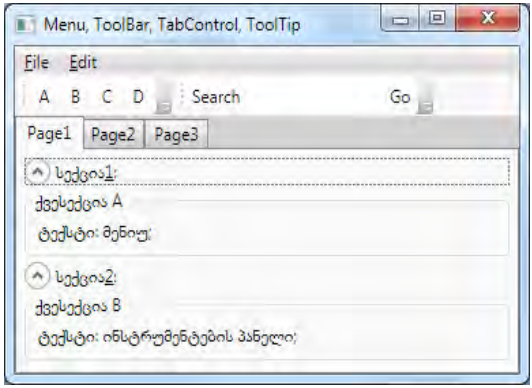
შედეგები ასახულია 2.48 (ა-ე) ნახაზზე.



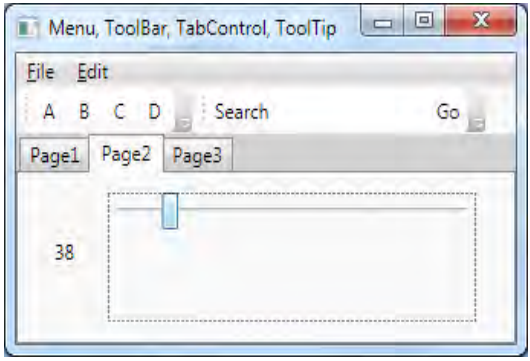
ნახ.2.48–ა



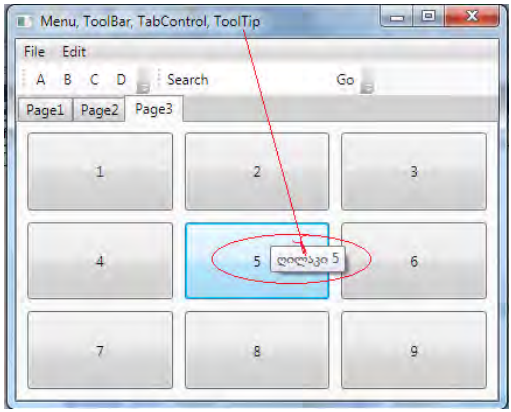
ნახ.2.48–ბ



ნახ.2.48–გ



ნახ.2.48–დ



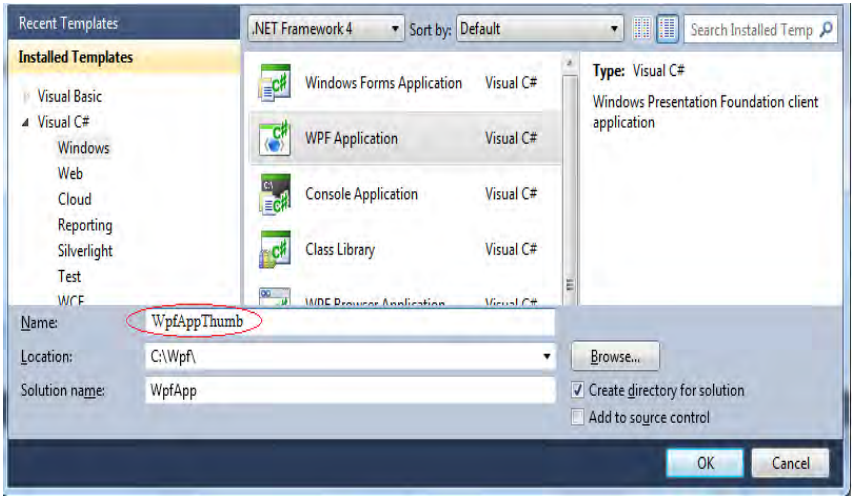
ნახ.2.48–ე

2.9. WPF-ის მართვის ელემენტები: Thumb, Border, Popup
(ლაბორატორიული სამუშაო N 9)

მიზანი: WPF-აპლიკაციების ასაგებად Thumb, Border და Popup მართვის ელემენტების გამოყენების შესწავლა.

მართვის ელემენტი **Thumb** (მანიპულატორი) აინკაპსულირებს არეს, რომლის გადაადგილებაც შესაძლებელია, იგი ასევე აგენერირებს ყველა აუცილებელ მოვლენას.

1. შევქმნათ ახალი პროექტი WpfAppThumb და მივანიჭოთ სასტარტო მნიშვნელობა.



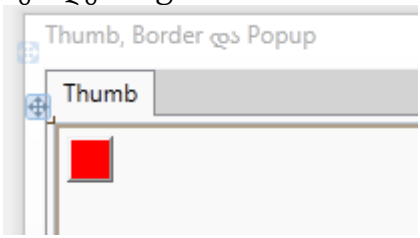
ნახ.2.49

2. Window1.xaml ფაილში ჩავწეროთ შემდეგი ტექსტი:

```
<Window x:Class="WpfAppThumb.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
         Title="Thumb, Border და Popup"
         Height="350"
         Width="525"
         Background="LightGray"
     >
```

```
    <TabControl>
        <!--ელემენტი, რომელიც აღწერს  გადაადგილების არეს-->
        <TabItem Header="Thumb">
            <Canvas>
                <Thumb
                        Name="thumb1"
                        Background="Red"
                        Canvas.Left="6"
                        Canvas.Top="6"
                        Width="23"
                        Height="23"
                        DragStarted="thumb1_DragStarted"
                        DragDelta="thumb1_DragDelta"
                        />
            </Canvas>
        </TabItem>
    <!-- მომავალი ტექსტების ჩასამატებელი ადგილი ----------------------
     </TabControl>
</Window>
```

მივიღებთ შემდეგ ფრაგმენტს წითელი კვადრატით, რომლის გადაადგილებაც შეიძლება მაუსით:



ნახ.2.50

3. Window1.xaml ფაილში ჩავამატოთ ტექსტი:

```
    <!-- Border ჩარჩოების გამოყენება -->
        <TabItem Header="Border">
            <StackPanel>
                <Border
                        BorderThickness="3"
                        CornerRadius="0"
                        BorderBrush="Black"
                        Padding="5"
```

```
            Margin="10"
            Background="Violet"
            >
        <TextBlock HorizontalAlignment="Center">
                მართკუთხა ღილაკი-1
        </TextBlock>
    </Border>
    <Border
            BorderThickness="3"
            CornerRadius="20"
            BorderBrush="Black"
            Padding="5"
            Margin="10"
            Background="Violet"
            >
        <TextBlock HorizontalAlignment="Center">
                მრგვალკუთხა ღილაკი-2
        </TextBlock>
    </Border>
    <Border
            BorderThickness="10,1,10,1"
            CornerRadius="150"
            BorderBrush="Black"
            Padding="5"
            Margin="10"
            Background="Violet"
            >
        <TextBlock HorizontalAlignment="Center">
                ელიფსური ღილაკი-3
        </TextBlock>
    </Border>
    <Border
            BorderThickness="5,10,9,15"
            CornerRadius="0,25,0,15"
            BorderBrush="Blue"
            Padding="5"
            Margin="10"
            Background="Violet"
            >
        <TextBlock HorizontalAlignment="Center">
                სქელჩარჩოიანი ღილაკი-4
        </TextBlock>
```

```
                </Border>
            </StackPanel>
        </TabItem>
    <!-- მომავალი ტექსტების ჩასამატებელი ადგილი ----------------
```

მივიღებთ:

ნახ.2.51

4. Window1.xaml ფაილში ჩავამატოთ ახალი ტექსტი Popup ელემენტით.

```
<!-- Popup ელემენტის გამოყენება -->
    <TabItem Header="Popup">
       <StackPanel>
         <Button Name="btn" Click="btn_Click">Toggle</Button>
              <Popup Name="popup"
                     PopupAnimation="Fade"
                     Placement="Mouse"
                   >
         <Button Background="Yellow" Foreground="Red">
                        ღილაკი შექმნილია Popup-ში
         </Button>
        </Popup>
      </StackPanel>
    </TabItem>
```

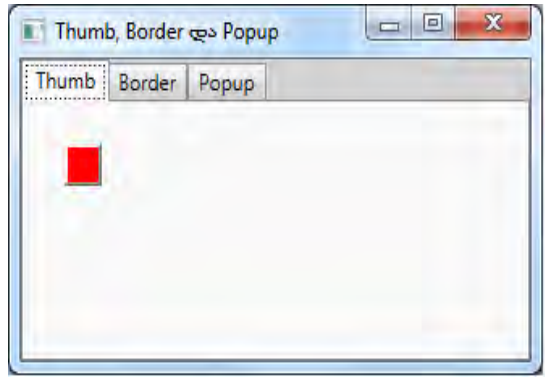
და ამავდროულად შევასრულოთ პ.5:

5. ჩავწეროთ Window1.xaml.cs ფაილში შემდეგი C# ტექსტი:

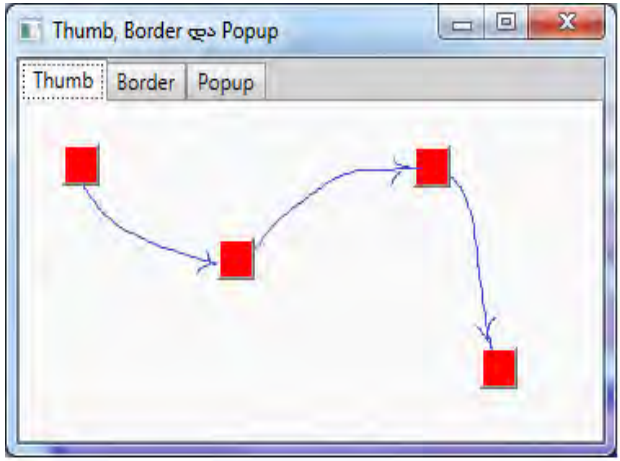
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Controls.Primitives; // ეს დაამატეთ !

namespace WpfAppThumb
{
   public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
        }

     protected override void OnMouseDoubleClick(
                                    MouseButtonEventArgs e)
        {
           base.OnMouseDoubleClick(e);

            // Thumb ბრუნდება საწყის მდგომარეობაში
            Canvas.SetLeft(thumb1, 5);
            Canvas.SetTop(thumb1, 5);
        }

        double originalLeft, originalTop;
     private void thumb1_DragStarted(object sender,
                                    DragStartedEventArgs e)
        {
            originalLeft = Canvas.GetLeft(thumb1);
            originalTop = Canvas.GetTop(thumb1);
        }
```

```
        private void thumb1_DragDelta(object sender,
                                      DragDeltaEventArgs e)
            {
                double left = originalLeft + e.HorizontalChange;
                double top = originalTop + e.VerticalChange;
                Canvas.SetLeft(thumb1, left);
                Canvas.SetTop(thumb1, top);
                originalLeft = left;
                originalTop = top;
            }

        private void btn_Click(object sender, RoutedEventArgs e)
            {
                popup.IsOpen = !popup.IsOpen;
            }
    }
}
```

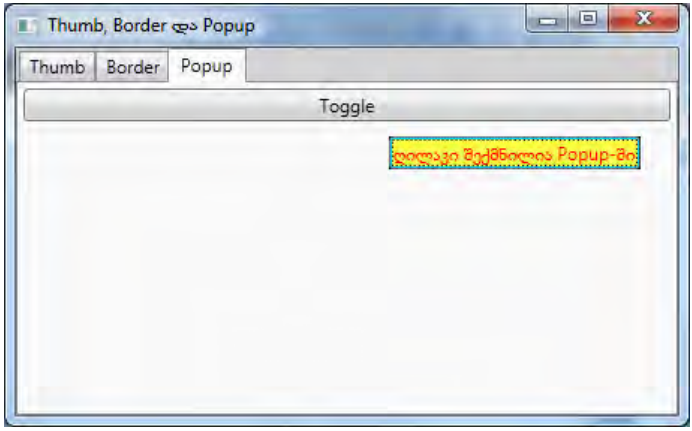
6. ავამუშავოთ აპლიკაცია და დავაკვირდეთ Thumb, Border, Popup ელემენტების მუშაობას:



ნახ.2.52–ა, საწყისი მდებარეობა



ნახ.2.52–ბ. მაუსით გადატანა



ნახ.2.52–გ. ბუტონის დაკლიკვისას ამოტივტივდება დამხმარე ტექსტი

2.10. WPF-ის მართვის ელემენტები: ScrollViewer, Viewbox და StackPanel

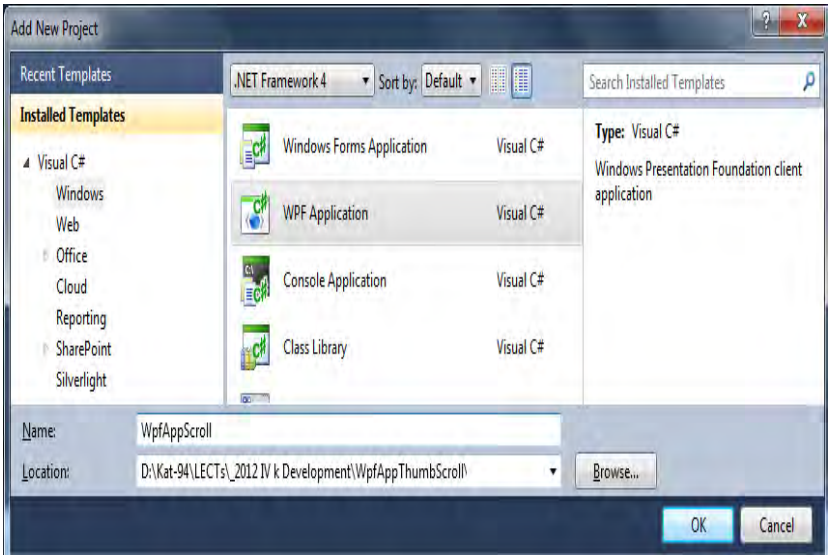
(ლაბორატორიული სამუშაო N 10)

მიზანი: WPF-აპლიკაციების ასაგებად ScrollViewer, Viewbox და StackPanel მართვის ელემენტების გამოყენების შესწავლა.

ScrollViewer ელემენტით ხდება ფანჯრის სივრცის გადახვევა, თუ შვილობილ-ელემენტიანი ინტერფეისი მასში ვერ ეტევა.

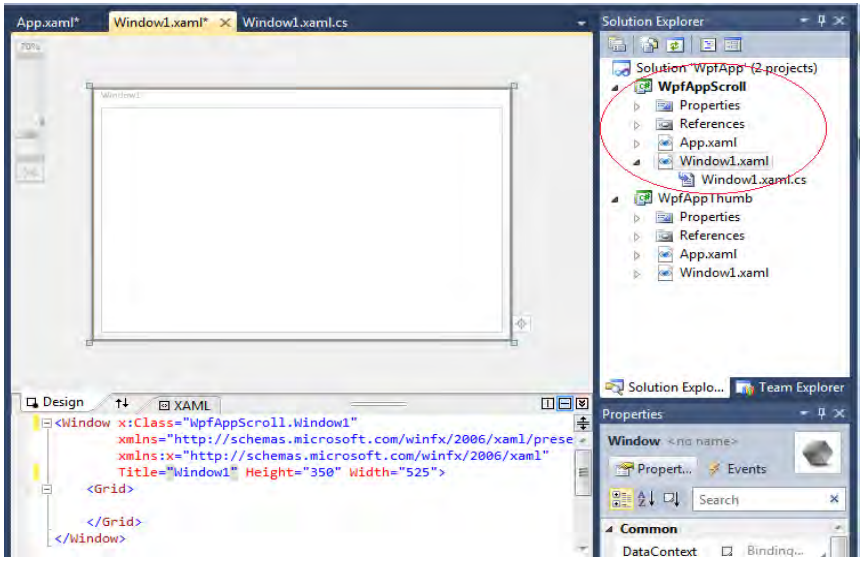
Viewbox ელემენტი ღებულობს მაგალითად, კონტეინერს და ახდენს მის მასშტაბირებას თავისი ზომების პროპორციულად.

1. დავამატოთ ახალი პროექტი WpfAppScroll სახელით და გავხადოთ სასტარტო.



ნახ.2.53

მივიღებთ საწყის მდგომარეობას:



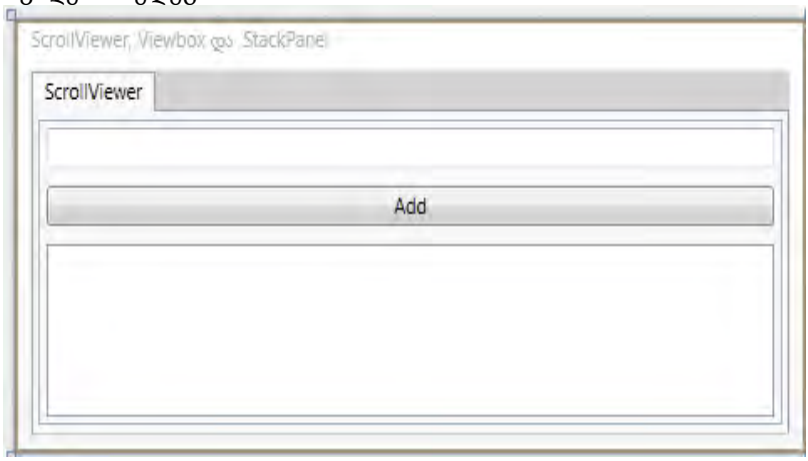
ნახ.2.54

2. ჩავწეროთ Window1.xaml კოდში შემდეგი ტექსტი:

```
<Window x:Class="WpfAppScroll.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="ScrollViewer, Viewbox და  StackPanel "
        Height="350"
        Width="525"
        Background="LightGray">
    <TabControl>
        <TabItem Header="ScrollViewer">
            <ScrollViewer
                HorizontalScrollBarVisibility="Auto"
                VerticalScrollBarVisibility="Auto"
                >
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="Auto" />
                        <RowDefinition />
```

```
            </Grid.RowDefinitions>
            <TextBox Name="textBox" Grid.Row="0" Margin="5" />
               <Button Grid.Row="1" Margin="5"
                            Click="Add_Click">Add</Button>
            <ListBox Name="listBox" Grid.Row="2" Margin="5" />
          </Grid>
        </ScrollViewer>
      </TabItem>
        <!-------------- ჩასამატებელი ადგილი ---------------- ->
    </TabControl>
</Window>
```

მივიღებთ შედეგს:



ნახ.2.55

3. დავამატოთ ახალი ტექსტი `Viewbox`-ისთვის:

```
<TabItem Header="Viewbox">
            <Viewbox>
                <TextBlock>Текст</TextBlock>
            </Viewbox>
        </TabItem>
    </TabControl>
</Window>
```

მივიღებთ შედეგს:

ნახ.2.56

4. დავამატოთ ტექსტი StackPanel -ისთვის:

```
<TabItem Header="StackPanel">
   <StackPanel Orientation="Vertical" Background="Aqua">
      <Button HorizontalAlignment="Center" Width="75"
                      Margin="5"> Center
      </Button>
      <Button HorizontalAlignment="Left" Width="75">
          <Button.LayoutTransform>
            <RotateTransform Angle="-45" />
          </Button.LayoutTransform> Left (-45)
      </Button>
      <Button HorizontalAlignment="Right" Width="75">
           <Button.LayoutTransform>
            <RotateTransform Angle="45" />
          </Button.LayoutTransform> Right (45)
      </Button>
      <Button HorizontalAlignment="Stretch"
                        Margin="5">Stretch</Button>
   </StackPanel>
</TabItem>
```

მივიღებთ შედეგს 2.57 ნახაზზე:

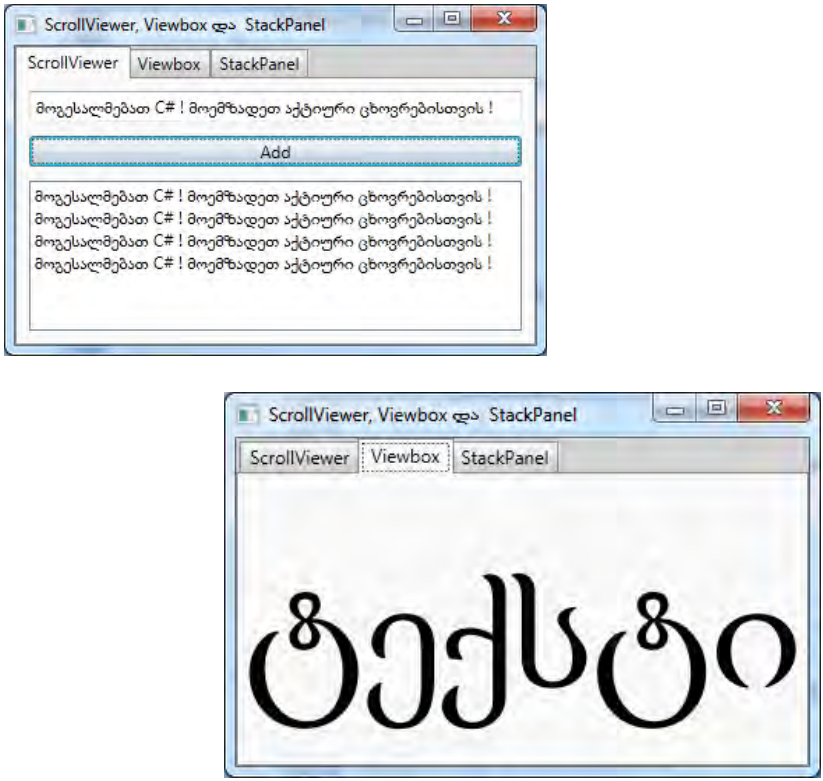
ნახ.2.57

5. შევავსოთ C# კოდი შემდეგი ტექსტით:

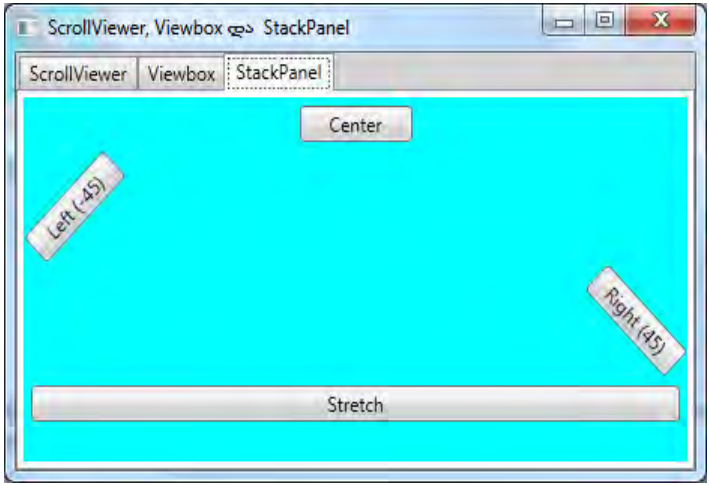
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfAppScroll
{
   public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
            textBox.Text = "მოგესალმებათ C# ! მოემზადეთ
                     აქტიური ცხოვრებისთვის !";
        }
```

```
        private void Add_Click(object sender, RoutedEventArgs e)
            {
                listBox.Items.Add(textBox.Text);
            }

    }
}
```

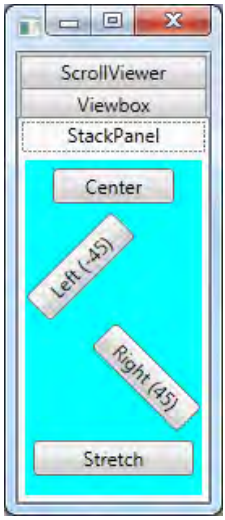
ავამუშავოთ აპლიკაცია, მივიღებთ (ნახ.2.58):



ნახ.2.58–ა



ნახ.2.58–ბ



ნახ.2.58–გ

მასშტაბი იცვლება და ელემენტების განთავსებაც შესაბამისად შეიცვლება.

2.11. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: Canvas

(ლაბორატორიული სამუშაო N 11)

მიზანი: WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების მენეჯერის Canvas ფუნქციონალობის შესწავლა.

პროგრამულ აპლიკაციებში ინტერფეისული ელემენტების დინამიკური განლაგებისთვის ხშირად გამოიყენება შემდეგი პანელები: **Canvas, StackPanel, DockPanel, UniformGrid და Grid.**

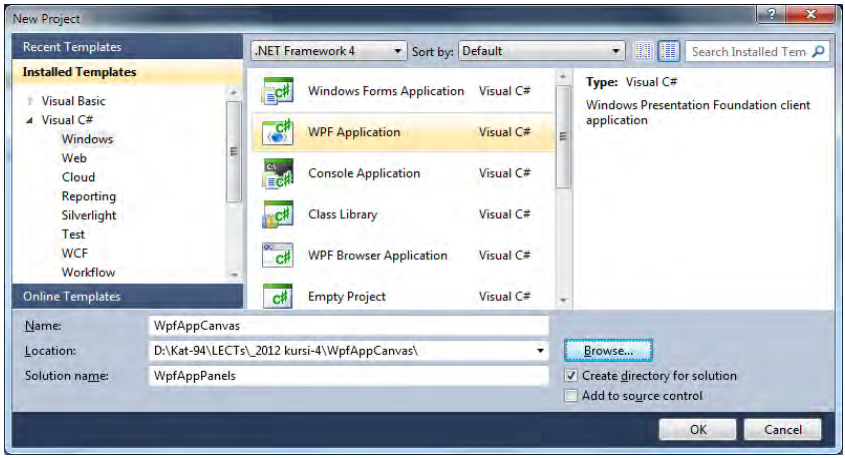
პანელები უზრუნველყოფს ფანჯარაში ან გვერდზე ელემენტების მოთავსებას სასურველ პოზიციებში. ყველა ზემონახსენები პანელი მემკვიდრეა Panel – აბსტრაქტული კლასის, რომელიც თავის მხრივ იწარმოება:

System.Windows.FrameworkElement

საბაზო კლასიდან.

თვით პანელები ეკრანზე არ ჩანს, თუ ცხადად არ მიეთითა პანელის ფერადი ფონი. მათი არსებობა იგრძნობა პანელზე მოთავსებული შვილობილი ელემენტების ქცევით.

1. გავხსნათ ახალი პროექტი name: WpfAppCanvas, რომლის Solution name: WpfAppPanels (შემდგომში მას დაემატება სხვა პანელების საილუსტრაციო პროექტები).



ნახ.2.59

2. MainWindow.xaml კოდში ჩავწეროთ შემდეგი ტექსტი:

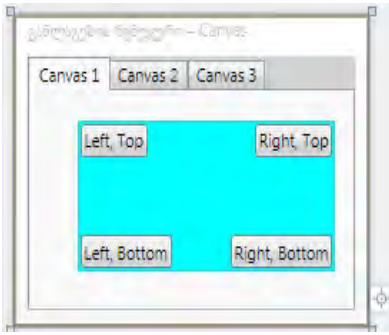
```
<Window x:Class="WpfAppCanvas.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="განლაგების მენეჯერი – Canvas" Height="300"
              Width="300"
    Background="LightGray">
    <TabControl>
      <TabItem Header="Canvas 1">
       <Canvas Width="200" Height="100" Background="Aqua">
   <Button Canvas.Left="2" Canvas.Top="2">Left, Top </Button>
  <Button Canvas.Right="2" Canvas.Top="2">Right, Top</Button>
<Button Canvas.Left="2" Canvas.Bottom="2">Left, Bottom</Button>
<Button Canvas.Right="2" Canvas.Bottom="2">Right, Bottom</Button>
       </Canvas>
    </TabItem>
    <TabItem Header="Canvas 2">
      <Canvas HorizontalAlignment="Center"
VerticalAlignment="Center" Background="Aqua">
  <Button Canvas.Left="2" Canvas.Top="2">Left, Top</Button>
  <Button Canvas.Right="2" Canvas.Top="2">Right, Top</Button>
<Button Canvas.Left="2" Canvas.Bottom="2">Left, Bottom</Button>
<Button Canvas.Right="2" Canvas.Bottom="2">Right, Bottom</Button>
      </Canvas>
    </TabItem>
        <TabItem Header="Canvas 3">
            <StackPanel>
                <Button>Button 1</Button>
                <Button>Button 2</Button>
                <Canvas HorizontalAlignment="Center"
                                      Background="Aqua">
   <Button Canvas.Left="-21" Canvas.Top="8">Canvas</Button>
           </Canvas>
         </StackPanel>
      </TabItem>
    </TabControl>
</Window>
```

3. C# კოდი ავტომატურად იქმნება და მას დამატებითი ცვლილებები ამჯერად არ სჭირდება.

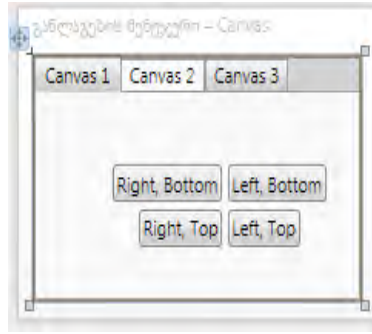
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfAppCanvas
{
   public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

4. შედეგი მიიღება სამი Canvas გვერდით, რომლებშიც ღილაკების განლაგება განსხვავებულია:

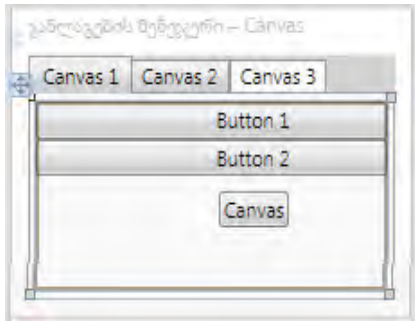
(1) Canvas–1: აქ განლაგების მენეჯერის სიგანე მოცემულია ცხადად, პანელის ხილვადობისთვის მოცემულია ფონი. ღილაკების განლაგებისთვის პანელის კუთხეებში გამოიყენება სპეციალური თვისებები – ატრიბუტები: Canvas.Left, Canvas.Right, Canvas.Top და Canvas.Bottom.



(1)



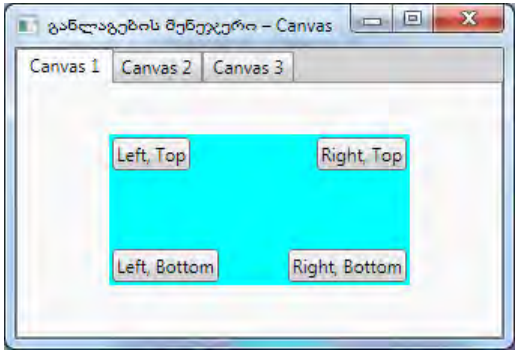
(2)



(3)

ნახ.2.60

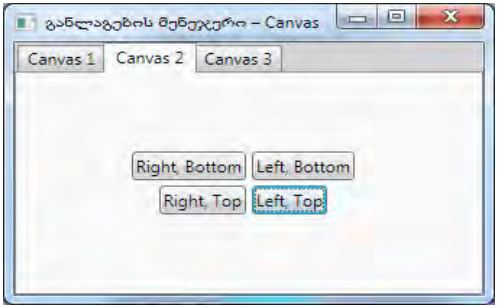
```
<TabItem Header="Canvas 1">
  <Canvas Width="200" Height="100" Background="Aqua">
    <Button Canvas.Left="2" Canvas.Top="2">Left,
                                        Top</Button>
    <Button Canvas.Right="2" Canvas.Top="2">Right,
                                        Top</Button>
    <Button Canvas.Left="2" Canvas.Bottom="2">Left,
                                      Bottom</Button>
    <Button Canvas.Right="2" Canvas.Bottom="2">Right,
                                       Bottom</Button>
  </Canvas>
</TabItem>
```



ნახ.2.61

(2) Canvas–2: აქ Canvas პანელის ზომები არაა ცხადად მოცემული, ამიტომაც პანელი შემცირდება მინიმუმამდე და ღილაკები ერთმანეთს მიედება, პანელი არ ჩანს. პანელისთვის აქ მოცემულია თვისებები: HorizontalAlignment და VerticalAlignment, რომელიც ასრულებს პანელის პოზიციონირებას კლიენტის სამუშაო გარემოს მიხედვით. ფანჯრის ზომების შეცვლისას პანელი ღილაკებით ინარჩუნებს ცენტრში მდებარეობას.

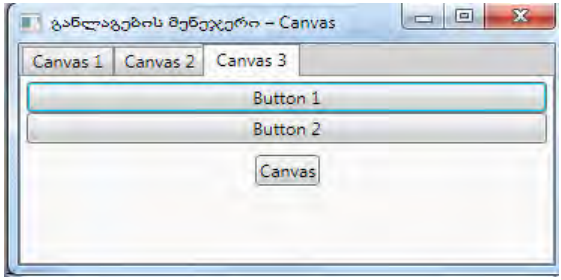
```
<TabItem Header="Canvas 2">
 <Canvas HorizontalAlignment="Center"
         VerticalAlignment="Center" Background="Aqua">

   <Button Canvas.Left="2" Canvas.Top="2">Left,Top
   </Button>
   <Button Canvas.Right="2" Canvas.Top="2">Right, Top
   </Button>
   <Button Canvas.Left="2" Canvas.Bottom="2">Left,Bottom
   </Button>
   <Button Canvas.Right="2" Canvas.Bottom="2">Right,Bottom
   </Button>
 </Canvas>
</TabItem>
```



ნახ.2.62

(3) Canvas–3: ამ შემთხვევაში StackPanel პანელი ახორციელებს ღილაკების განლაგებას ვერტიკალში. ბოლო ელემენტია წერტილოვანი Canvas-პანელი, პოზიციონირდება ცენტრში თვისებით HorizontalAlignment. Canvas პანელს მისი ღილაკი მიებმება x ღერძზე უარყოფითი წანაცვლებით.

```
<TabItem Header="Canvas 3">
     <StackPanel>
         <Button>Button 1</Button>
         <Button>Button 2</Button>
           <Canvas HorizontalAlignment="Center"
                                  Background="Aqua">
              <Button Canvas.Left="-21"
                   Canvas.Top="8">Canvas</Button>
           </Canvas>
        </StackPanel>
      </TabItem>
```



ნახ.2.63

<u>დავალება:</u> ააგეთ პროექტი და გამოიკვლიეთ Canvas პანელის ფუნქციონალური შესაძლებლობები.

2.12. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: StackPanel და DockPanel
(ლაბორატორიული სამუშაო N 12)

მიზანი: WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების მენეჯერის StackPanel და DockPanel ფუნქციონალობის შესწავლა.

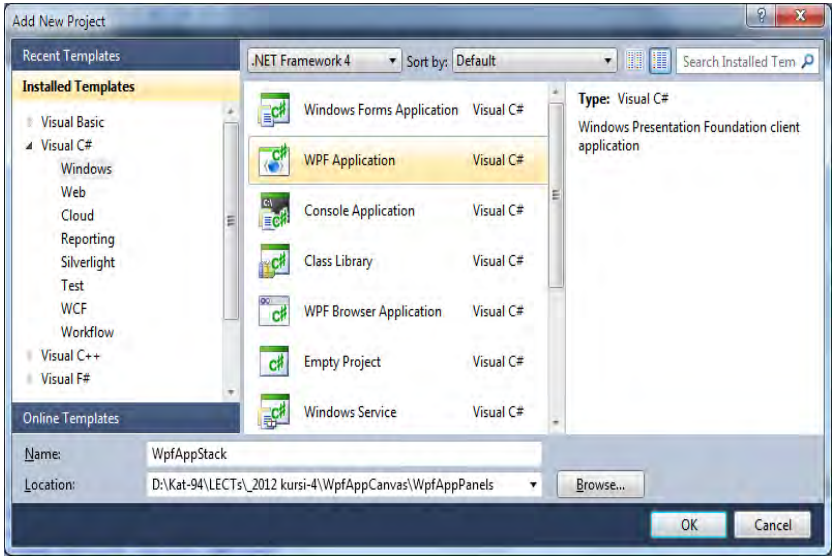
პროგრამულ აპლიკაციებში ინტერფეისული ელემენტების დინამიკური განლაგებისთვის ხშირად გამოიყენება შემდეგი პანელები:

1. Canvas (ლაბ.N11)
2. StackPanel
3. DockPanel
4. UniformGrid
5. Grid

StackPanel განათავსებს ფანჯარაზე მოთავსებულ ელემენტებს ვერტიკალში სტრიქონების სახით. Orientation თვისებით შეიძლება განლაგება ვერტიკალურად ან ჰორიზონტალურად განხორციელდეს. გამოუცხადებლად, StackPanel იკავებს კლიენტის სამუშაო არეს (მაგ., ფანჯარას) მთლიანად. მის ყოველ შვილობილ ელემენტს გამოუყოფს სლოტს (სასიცოცხლო არე), ხოლო თავის ზომას გაითვლის შვილობილებიან მაქსიმალურის მიხედვით.

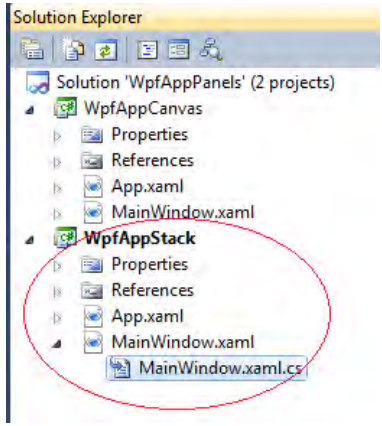
გამოყოფილი სლოტის მიხედვით შვილობილ ელემენტს შეუძლია პოზიციონირება თავისი შეხედულებისამებრ, Margin თვისების გამოყენებით. HorizontalAlignment და VerticalAlignment სლოტის შიგა ადგილი. ელემენტებსშორისი მანძილი მოიცემა Padding თვისებით.

1. დავამატოთ ახალი WpfAppStack პროექტი (იხ. ლაბ.13) ძველ Solution WpfAppPanels -ში და გავხადოთ იგი სასტარტო.



ნახ.2.64

მივიღებთ:

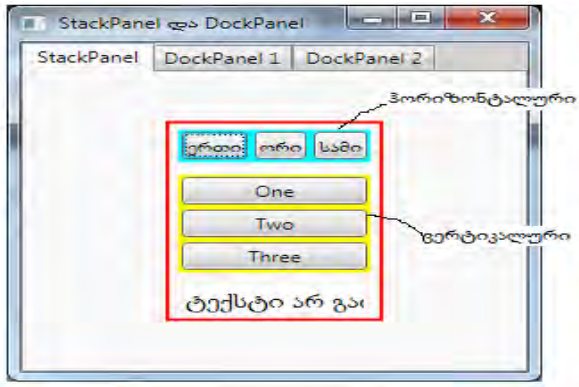


ნახ.2.65

2. დავამატოთ MainWindow.xaml კოდში შემდეგი ტექსტი (ჯერ `StackPanel`-ის გამოყენებისთვის):

```
<Window x:Class="WpfAppStack.MainWindow"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Title="StackPanel და DockPanel" Height="300" Width="300"
     Background="LightGray"
         >
     <!--  StackPanel-ის გამოყენება -->
     <TabControl>
         <TabItem Header="StackPanel">
             <Border BorderBrush="Red" BorderThickness="2"
                  HorizontalAlignment="Center"
                  VerticalAlignment="Center">
               <StackPanel Orientation="Vertical">
               <StackPanel Orientation="Horizontal" Margin="5"
                    Background="Aqua">
                 <Button Margin="2">ერთი</Button>
                 <Button Margin="2">ორი</Button>
                 <Button Margin="2">სამი</Button>
               </StackPanel>
               <StackPanel Orientation="Vertical" Margin="5"
                        Background="Yellow">
                 <Button Margin="2">One</Button>
                 <Button Margin="2">Two</Button>
                 <Button Margin="2">Three</Button>
               </StackPanel>
               <StackPanel Orientation="Horizontal" Margin="5"
                        Width="100" Background="White">
                  <TextBlock FontSize="12pt"
                    TextWrapping="Wrap">ტექსტი არ გადაიტანება,
                    რადგან ის ჩატვირთულია StackPanel-ში
                  </TextBlock>
                </StackPanel>
               </StackPanel>
             </Border>
         </TabItem>
     </TabControl>
 </Window>
```

3. ავამუშავოთ პროგრამა და მივიღებთ შედეგს:

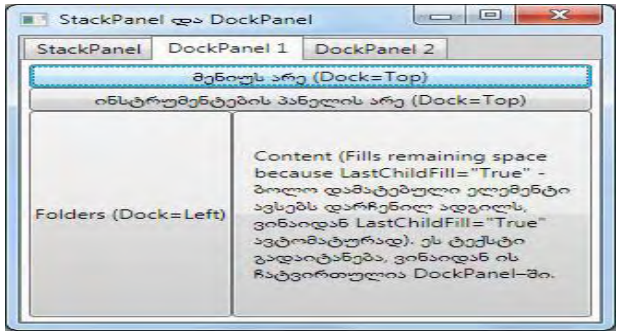


ნახ.2.66

4. ახლა დავამატოთ MainWindow.xaml ფაილში ახალი ტექსტი DockPanel–ის გამოსაყენებლად, რომელიც აისახება „DockPanel 1“ გვერდზე:

```
<!-- Windows Explorer სივრცის იმიტაცია DockPanel–ის
გამოყენებით -->
   <TabItem Header="DockPanel 1">
        <DockPanel>
   <Button DockPanel.Dock="Top">მენიუს არე (Dock=Top)</Button>
   <Button DockPanel.Dock="Top">ინსტრუმენტების პანელის არე
             (Dock=Top)</Button>
   <Button DockPanel.Dock="Left">Folders (Dock=Left)</Button>
   <Button>
     <TextBlock TextWrapping="Wrap" Padding="5pt">
        Content (Fills remaining space because
           LastChildFill="True" - ბოლო დამატებული ელემენტი
           ავსებს დარჩენილ ადგილს, ვინაიდან
                 LastChildFill="True" ავტომატურად).
                 ეს ტექსტი გადაიტანება, ვინაიდან ის
                 ჩატვირთულია DockPanel–ში.
    </TextBlock>
   </Button>
  </DockPanel>
 </TabItem>
```

5. პროგრამის ამუშავებით მიიღება ასეთი შედეგი:

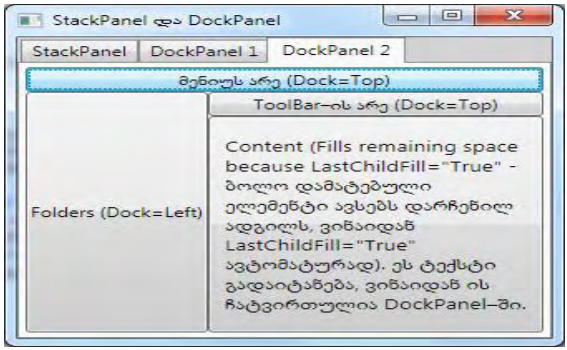


ნახ.2.67

6. MainWindow.xaml ფაილში დავამატოთ ტექსტი DockPanel 2 –თვის:

```
<TabItem Header="DockPanel 2">
  <DockPanel LastChildFill="True">
    <Button DockPanel.Dock="Top">მენიუს არე
             (Dock=Top)</Button>
    <Button DockPanel.Dock="Left">Folders
             (Dock=Left)</Button>
    <Button DockPanel.Dock="Top">ToolBar-ის არე
             (Dock=Top)</Button>
    <Button>
       <TextBlock TextWrapping="Wrap" FontSize="10pt"
            Padding="5pt">
          Content (Fills remaining space because
          LastChildFill="True"-ბოლო დამატებული ელემენტი
          ავსებს დარჩენილ ადგილს, ვინაიდან
          LastChildFill="True" ავტომატურად).
             ეს ტექსტი გადაიტანება, ვინაიდან ის
             ჩატვირთულია DockPanel-ში.
       </TextBlock>
     </Button>
   </DockPanel>
</TabItem>
```

7. პროგრამის ამუშავების შედეგად მიიღება:



ნახ.2.68

დასკვნა:

1. პირველ გვერდზე იხატება ჩარჩო მოცემული სიგანით და ფერით, რომელშიც თავსდება StackPanel, ჯერ ჰორიზონტალური (ერთი, ორი, სამი), შემდეგ ვერტიკალური ორიენტაციით (One, Two, Three). მათი ფონის ფერები განსხვავებულია. პანელში ქვემოთ მოცემულია ტექსტური ბლოკი, რომელიც სტრიქონს არ გადაიტანს (თუმცა ჩართულია თვისება TextWrapping="Wrap").

2. სტრიქონის გადატანა ტექსტურ ბლოკში შესაძლებელია მაშინ, როცა იგი ჩატვირთულია DockPanel პანელში.

3. DockPanel–ში განლაგების სტრუქტურა დამოკიდებულია ელემენტების დამატების მიმდევრობაზე ამ პანელში. LastChildFill თვისება უფლებას აძლევს ბოლო შვილობილ ელემენტს დაიკავოს მთელი თავისუფალი სივრცე. DockPanel–ის ელემენტებისთვის მოქმედებს მიბმის თვისება, ანუ თავისუფალი ადგილის რომელ მხარეს უნდა მიებას ახალი ელემენტი.

დავალება: ააგეთ აპლიკაცია და დააკვირდით განლაგების მენეჯერის მუშაობას ფანჯრის ზომების ცვლილებისას.

2.13. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: WrapPanel и UniformGrid
(ლაბორატორიული სამუშაო N 13)

მიზანი: WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების მენეჯერის WrapPanel და UniformGrid ფუნქციონალობის შესწავლა.

თეორიული ნაწილი:

პროგრამულ აპლიკაციებში ინტერფეისული ელემენტების დინამიკური განლაგებისთვის ხშირად გამოიყენება შემდეგი პანელები: **Canvas (ლაბ.N13), StackPanel (ლაბ.N14), DockPanel (ლაბ.N14), UniformGrid, Grid.**

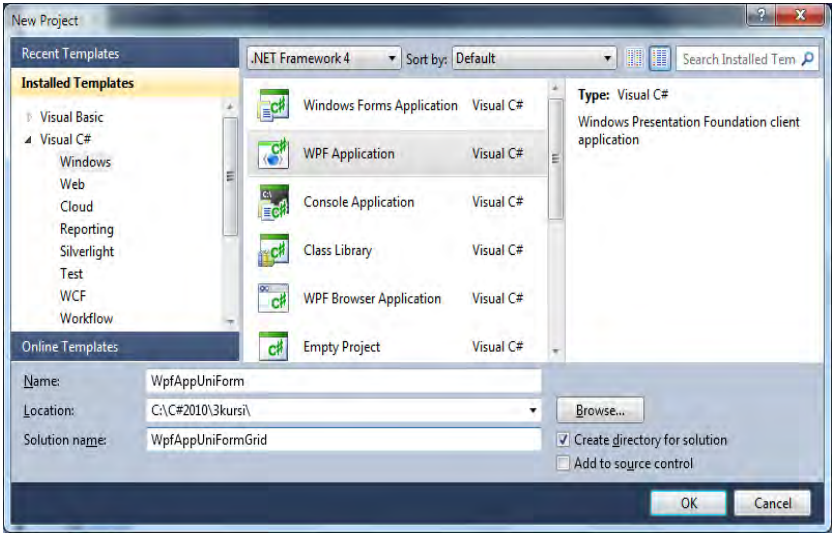
DockPanel კონტეინერში, როგორც ეს წინა ლაბორატორიული სამუშაოდან ვნახეთ, ელემენტები თავსდება (მიეკვრება) ერთ–ერთ თავისუფალ გვერდს მათი ფანჯარაში მოთავსების მიმდევრობის შესაბამისად. **WrapPanel** კი ელემენტს, რომელიც ვეღარ ეტევა, გადაიტანს ახალ სტრიქონზე.

WrapPanel–ის ვერტიკალური ორიენტაციისას, ელემენტების გადატანა არ ხდება, ოღონდ ელემენტების სიგანე ხდება მათ შორის ყველაზე განიერის. WrapPanel აყენებს თავის შვილების ზომებს ავტომატურად, ითვალისწინებს რა მათ შინაარსს. შესაძლებელია ასევე სხვა ზომების მიცემაც ItemWidth და ItemHeight თვისებებით.

UniformGrid პანელი (Uniform–ერთგვაროვანი, ერთნაირი, თანაბარი; Grid – ბადე) ანაწილებს (განათავსებს) შვილ–ელემენტებს თანაბარ ბადეში სტრიქონების და სვეტების მითითებული რაოდენობის მიხედვით. შეიძლება მხოლოდ სტრიქონების, ან მხოლოდ სვეტების რაოდენობის მითითება, მაშინ მეორე პარამეტრი გაითვლება შვილების საერთო რაოდენობიდან.

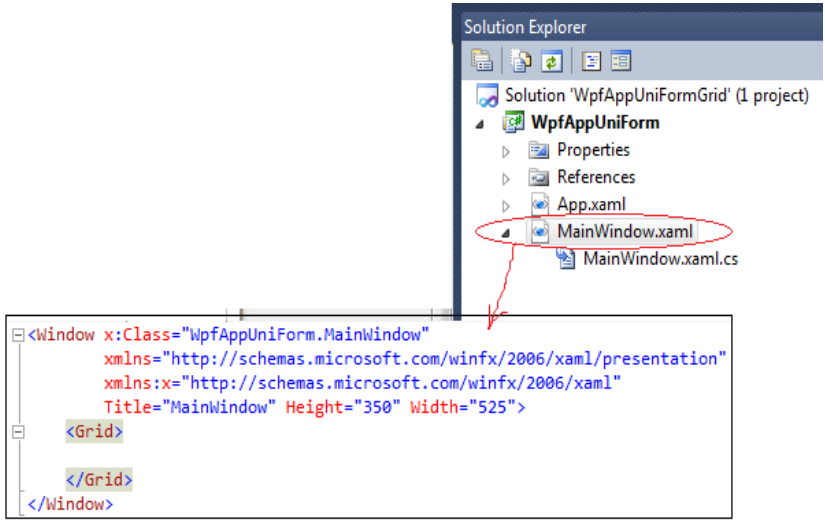
პრაქტიკული ნაწილი:

1. შევქმნათ SolutionExplorer-ში name=WpfAppUniFormGrid ახალი პროექტი name=WpfAppUniForm სახელით (ნახ.2.69).



ნახ.2.69

მივიღებთ საწყის მდგომარეობას (ნახ.2.70):



ნახ.2.70

2. MainWindow.xaml ფაილში შევცვალოთ Title:

```
Title="WrapPanel და UniformGrid"
    Height="300" Width="300"
     Background="LightGray"
```

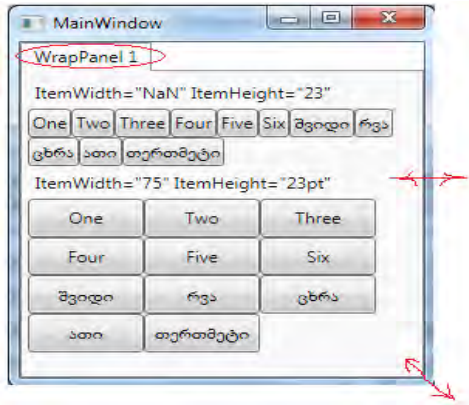
და წავშალოთ <Grid> … </Grid> სტრიქონები.

3. დავამატოთ შემდეგი კოდის ფრაგმენტი:

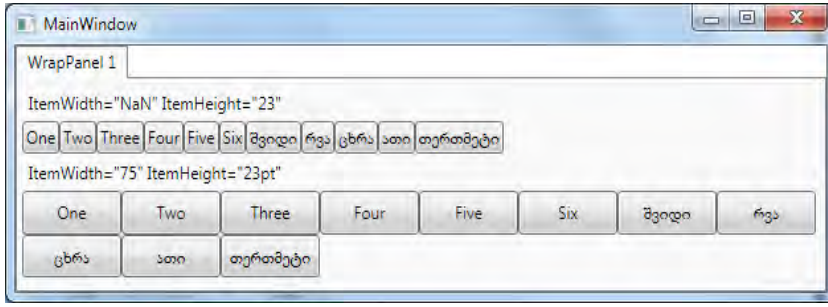
```
<TabControl>
        <TabItem Header="WrapPanel 1">
            <StackPanel>
                <TextBlock Margin="5">ItemWidth="NaN"
                     ItemHeight="23"</TextBlock>
                <WrapPanel ItemWidth="NaN" ItemHeight="23">
                    <Button>One</Button>
                    <Button>Two</Button>
                    <Button>Three</Button>
                    <Button>Four</Button>
                    <Button>Five</Button>
                    <Button>Six</Button>
                    <Button>შვიდი</Button>
                    <Button>რვა</Button>
                    <Button>ცხრა</Button>
                    <Button>ათი</Button>
                    <Button>თერთმეტი</Button>
                </WrapPanel>
                <TextBlock Margin="5">ItemWidth="75"
                           ItemHeight="23pt"</TextBlock>
                <WrapPanel ItemWidth="75" ItemHeight="23pt">
                    <Button>One</Button>
                    <Button>Two</Button>
                    <Button>Three</Button>
                    <Button>Four</Button>
                    <Button>Five</Button>
                    <Button>Six</Button>
                    <Button>შვიდი</Button>
                    <Button>რვა</Button>
                    <Button>ცხრა</Button>
                    <Button>ათი</Button>
```

```
                <Button>თერთმეტი</Button>
            </WrapPanel>
        </StackPanel>
    </TabItem>
</TabControl>
```

მივიღებთ (ნახ.2.71–ა,ბ.):



ნახ.2.71–ა

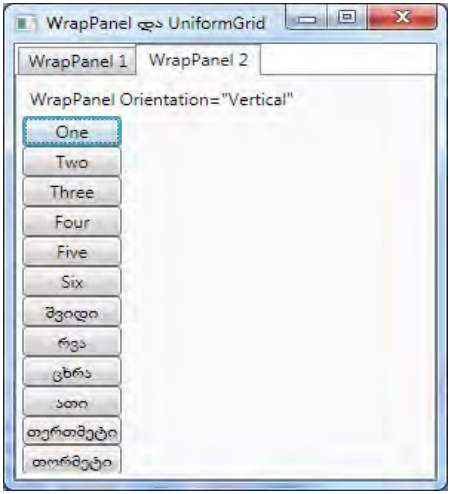


ნახ.2.71–ბ. ელემენტების განლაგების ცვლილება ფანჯრის ზომების მიხედვით

4. შევქმნათ WrapPanel 2 გვერდი `<TabControl>` – ის შიგნით, რომელიც ღილაკებს დაალაგებს ვერტიკალურად. დავამატოთ შემდეგი კოდი:

```
<TabItem Header="WrapPanel 2">
            <StackPanel>
                <TextBlock Margin="5">WrapPanel
                      Orientation="Vertical"</TextBlock>
                <WrapPanel Orientation="Vertical">
                    <Button>One</Button>
                    <Button>Two</Button>
                    <Button>Three</Button>
                    <Button>Four</Button>
                    <Button>Five</Button>
                    <Button>Six</Button>
                    <Button>შვიდი</Button>
                    <Button>რვა</Button>
                    <Button>ცხრა</Button>
                    <Button>ათი</Button>
                    <Button>თერთმეტი</Button>
                    <Button>თორმეტი</Button>
                </WrapPanel>
            </StackPanel>
        </TabItem>
```

მივიღებთ ვერტიკალში განლაგებულ ბუტონებს:



ნახ.2.72

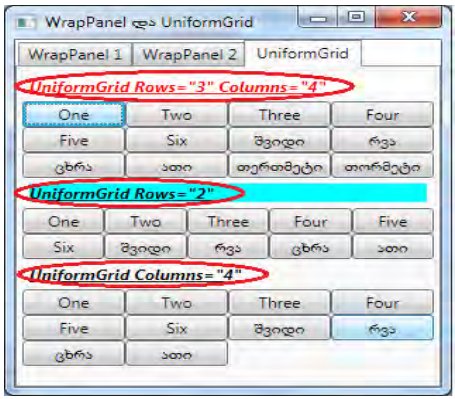
5. ახლა შევქმნათ UniformGrid გვერდი <TabControl> – ის შიგნით, რომელიც ღილაკებს განალაგებს თანაბრად, სტრიქონების (Rows) ან სვეტების (Columns) მითითებული რაოდენობით. დავამატოთ შემდეგი კოდი:

```
<TabItem Header="UniformGrid">
      <StackPanel>
          <TextBlock Margin="5" Background="White"
                                Foreground="Red"
                     FontWeight="Bold" FontStyle="Italic"
                     TextDecorations="Underline">
                    UniformGrid Rows="3" Columns="4"
            </TextBlock>
              <UniformGrid Rows="3" Columns="4">
                  <Button>One</Button>
                  <Button>Two</Button>
                  <Button>Three</Button>
                  <Button>Four</Button>
                  <Button>Five</Button>
                  <Button>Six</Button>
                  <Button>შვიდი</Button>
                  <Button>რვა</Button>
                  <Button>ცხრა</Button>
                  <Button>ათი</Button>
                  <Button>თერთმეტი</Button>
                  <Button>თორმეტი</Button>
              </UniformGrid>
            <TextBlock Margin="5" Background="Aqua">
                    <Bold>
                        <Italic>
                            <Underline>
                                UniformGrid Rows="2"
                            </Underline>
                        </Italic>
                    </Bold>
            </TextBlock>
              <UniformGrid Rows="2">
```

```
                <Button>One</Button>
                <Button>Two</Button>
                <Button>Three</Button>
                <Button>Four</Button>
                <Button>Five</Button>
                <Button>Six</Button>
                <Button>შვიდი</Button>
                <Button>რვა</Button>
                <Button>ცხრა</Button>
                <Button>ათი</Button>
            </UniformGrid>
        <TextBlock Margin="5">
            <Run FontWeight="Bold" FontStyle="Italic"
                                TextDecorations="Underline">
                    UniformGrid Columns="4"
                </Run>
        </TextBlock>
            <UniformGrid Columns="4">
                <Button>One</Button>
                <Button>Two</Button>
                <Button>Three</Button>
                <Button>Four</Button>
                <Button>Five</Button>
                <Button>Six</Button>
                <Button>შვიდი</Button>
                <Button>რვა</Button>
                <Button>ცხრა</Button>
                <Button>ათი</Button>
            </UniformGrid>
        </StackPanel>
    </TabItem>
```

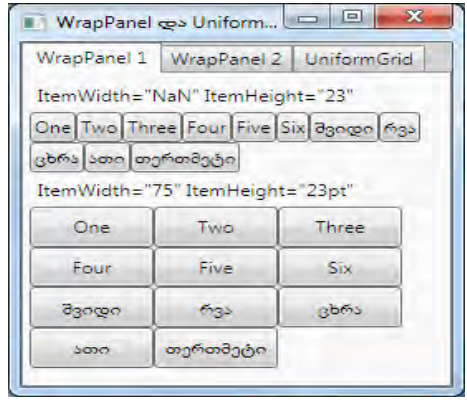
მივიღებთ, რომ ფანჯრის გაწელვით სვეტების რაოდენობა არ იცვლება, როგორც Wrap–ის დროს.

TextBlok–ელემენტი გამოიყენება ტექსტური სათაურების ჩასასმელად.

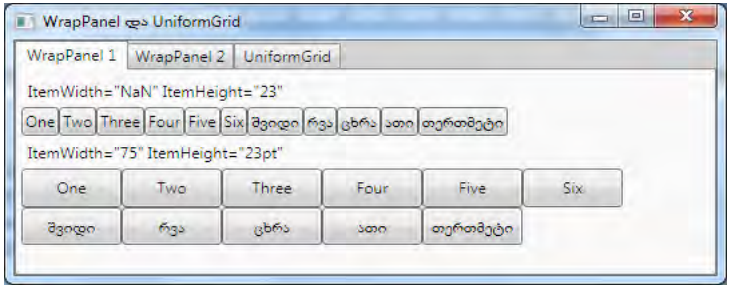


ნახ.2.73

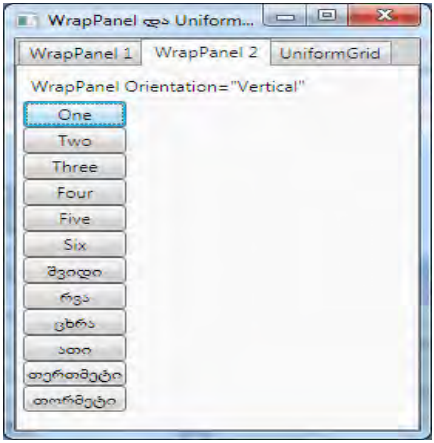
საბოლოო შედეგები მოცემულია 2.74 ა–დ ნახაზებზე:



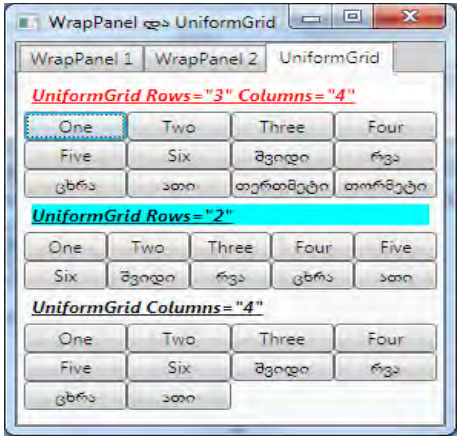
ნახ.2.74–ა



ნახ.2.74–ბ



ნახ.2.74–გ



ნახ.2.74–დ

დავალება:

1. ააგეთ ახალი პროექტი, აღწერილი თანამიმდევრობით, შეიტანეთ XAML-კოდის ტექსტები, გაამართეთ პროგრამა მუშა მდგომარეობამდე.
2. ჩაატარეთ ექსპერიმენტები და დააკვირდით პანელების და მასზე განლაგებული ელემენტების („შვილების“) ფუნქციონალობას.

2.14. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: Grid პანელი
(ლაბორატორიული სამუშაო N 14)

მიზანი: WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების მენეჯერის, კერძოდ Grid პანელის დამუშავება და მისი ფუნქციონალობის შესწავლა.

თეორიული ნაწილი:

პროგრამულ აპლიკაციებში ინტერფეისული ელემენტების დინამიკური განლაგებისთვის ხშირად გამოიყენება შემდეგი პანელები:

- Canvas (ლაბ.N11)
- StackPanel (ლაბ.N12)
- DockPanel (ლაბ.N12)
- UniformGrid (ლაბ.13)
- Grid

წინა ლაბორატორაში გავეცანით UniformGrid პანელს (Uniform–ერთგვაროვანი, ერთნაირი, თანაბარი; Grid – ბადე). აქ ყველა უჯრედს ჰქონდა თანაბარი ზომა. ხშირად ეს არაა საკმარისი და მოითხოვს მის გაფართოებას. ამისათვის კი გამოიყენება Grid პანელი. ასეთი პანელის მეზობელ სტრიქონებს შეიძლება ჰქონდეს განსხვავებული სიმაღლეები, ხოლო მეზობელ სვეტებს – კი განსხვავებული სიგანეები.

Grid–ელემენტი ყველაზე მოქნილი და უნივერსალურია ადრე განხილულ განთავსების მენეჯერებს შორის. იგი ფანჯრებისთვის ავსებს Table ელემენტის HTML ფუნქციონალობას. ამიტომაც, WPF-აპლიკაციის შემქნელი ოსტატი პროგრამა თავიდანვე ათავსებს მარკირებისაქს Grid ელემენტს.

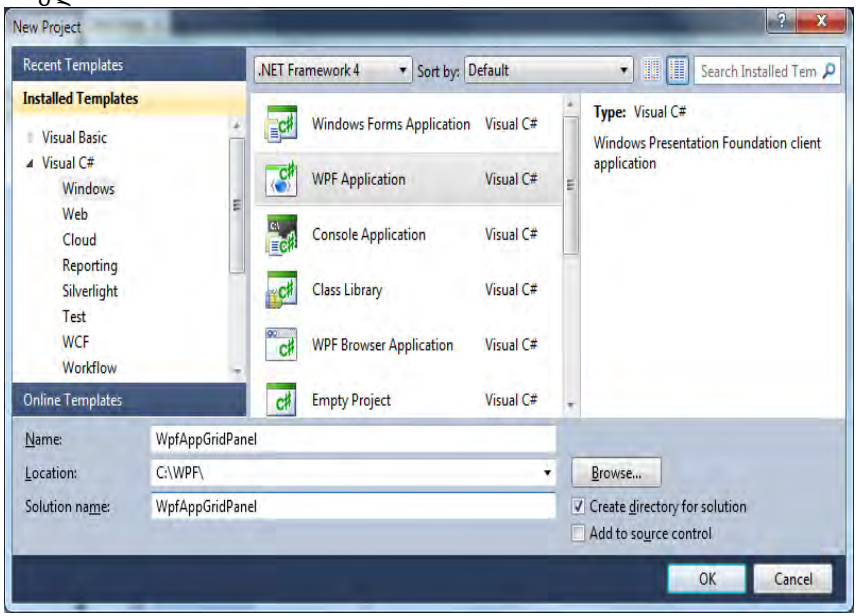
Grid ბადეში განლაგება შედგება ორი ეტაპისგან: ჯერ სრულდება სტრიქონების და სვეტების განსაზღვრა, ხოლო შემდეგ იწარმოება შვილობილი ელემენტების მოცემა და მათი განთავსება სლოტების მიხედვით. ყველაზე მარტივი ხერხი Grid–ის გამოსაყენებლად არის RowDefinitions და ColumnDefinitions თვისებების მიცემა. შემდეგ დაემატოს რამდენიმე შვილობილი

ელემენტი მათზე მიბმული თვისებებით Grid.Row და Grid.Column , განისაზღროს – რომელი ელემენტი რომელ სლოტში (გრიდის უჯრედში) მოთავსდეს.

არსებობს Grid–ის აწყობის უფრო ზუსტი და ფართო შესაძლებლობები, რომელთაც ქვემოთ განვიხილავთ პრაქტიკული ამოცანების დახმარებით.

პრაქტიკული ნაწილი:

1. შევქმნათ ახალი WPF აპლიკაცია WpfAppGrid პროექტის სახელით.



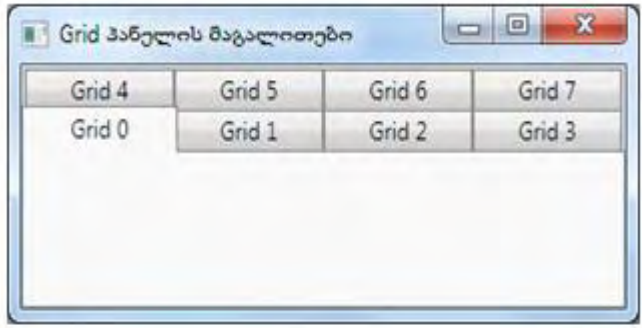
ნახ.2.75

2. MainWindow.xaml ფაილისთვის შევიტანოთ შემდეგი კოდის ფრაგმენტი:

```
<TabControl>

        <TabItem Header="Grid 0">
        </TabItem>
```

```
        <TabItem Header="Grid 1">
        </TabItem>

        <TabItem Header="Grid 2">
        </TabItem>

        <TabItem Header="Grid 3">
        </TabItem>

        <TabItem Header="Grid 4">
        </TabItem>

        <TabItem Header="Grid 5">
        </TabItem>

        <TabItem Header="Grid 6">
        </TabItem>

        <TabItem Header="Grid 7">
        </TabItem>
    </TabControl>
```

მიიღება შედეგი:



ნახ.2.76

ჩვენ უნდა განვიხილოთ Grid-პანელის შვიდი შემთხვევა (მაგალითებით).

3. მთლიანი ტექსტი:

```
<Window x:Class="WpfApp8.Window1"
```

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Grid პანელის მაგალითები"
Height="300" Width="300"
Background="LightGray"
  >
<TabControl>
  <TabItem Header="Grid 0">
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>
    <Button Grid.Row="0" Grid.Column="0"
                Background="LightPink">პირველი</Button>
    <Button Grid.Row="1" Grid.Column="0"
                Background="Lime">მეორე</Button>
    <Button Grid.Row="1" Grid.Column="1"
                Background="Aquamarine">მესამე (შიგთავსით)</Button>
    <Button Grid.Row="0" Grid.Column="1"
                Background="Yellow">მეოთხე</Button>
    </Grid>
  </TabItem>
  <TabItem Header="Grid 1">
    <Grid HorizontalAlignment="Center" VerticalAlignment="Center">
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
      </Grid.RowDefinitions>
```

```
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
        <Button Grid.Row="0" Grid.Column="0"
                        Background="LightPink">პირველი</Button>
        <Button Grid.Row="0" Grid.Column="1"
                        Background="Lime">მეორე</Button>
         <Button Grid.Row="1" Grid.Column="0"
                      Background="Aquamarine">მესამე (შიგთავსით)</Button>
         <Button Grid.Row="1" Grid.Column="1"
                      Background="Yellow">მეოთხე</Button>
      </Grid>
    </TabItem>
    <TabItem Header="Grid 2">
      <Grid>
          <Grid.RowDefinitions>
              <RowDefinition Height="50" />
              <RowDefinition Height="*" />
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
              <ColumnDefinition Width="50" />
              <ColumnDefinition Width="*" />
          </Grid.ColumnDefinitions>
   <Button Grid.Row="0" Grid.Column="0" MinWidth="0"
                          Background="LightPink">პირველი</Button>
   <Button Grid.Row="0" Grid.Column="1" MinWidth="0"
                          Background="Lime">მეორე</Button>
  <Button Grid.Row="1" Grid.Column="0" MinWidth="0"
                          Background="Aquamarine">მესამე</Button>
  <Button Grid.Row="1" Grid.Column="1" MinWidth="0"
                          Background="Yellow">მეოთხე</Button>
    </Grid>
```

```
    </TabItem>
    <TabItem Header="Grid 3">
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="2*" />
                <RowDefinition Height="1*" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="2*" />
                <ColumnDefinition Width="1*" />
             </Grid.ColumnDefinitions>
<Button Grid.Row="0" Grid.Column="0" MinWidth="0"
                        Background="LightPink">პირველი</Button>
<Button Grid.Row="0" Grid.Column="1" MinWidth="0"
                        Background="Lime">მეორე</Button>
<Button Grid.Row="1" Grid.Column="0" MinWidth="0"
                        Background="Aquamarine">მესამე</Button>
<Button Grid.Row="1" Grid.Column="1" MinWidth="0"
                       Background="Yellow">მეოთხე</Button>
        </Grid>
    </TabItem>
    <TabItem Header="Grid 4">
      <Grid HorizontalAlignment="Center" VerticalAlignment="Stretch">
            <Grid.RowDefinitions>
              <RowDefinition />
              <RowDefinition />
              <RowDefinition />
              <RowDefinition />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
<Button Grid.Row="0" Background="LightPink">პირველი</Button>
```

```
<Button Grid.Row="1" Background="Lime">მეორე</Button>
<Button Grid.Row="2" Background="Aquamarine">მესამე (ყველაზე განიერი
                    შიგთავსით) </Button>
<Button Grid.Row="3" Background="Yellow">მეოთხე</Button>
  </Grid>
</TabItem>
<TabItem Header="Grid 5">
   <Grid Grid.IsSharedSizeScope="True" HorizontalAlignment="Center">
        <Grid.RowDefinitions>
          <RowDefinition />
          <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto" SharedSizeGroup="myKey" />
          <ColumnDefinition Width="102" SharedSizeGroup="myKey" />
        </Grid.ColumnDefinitions>
<Button Grid.Row="0" Grid.Column="0" MinWidth="0"
                    Background="LightPink">პირველი</Button>
<Button Grid.Row="0" Grid.Column="1" MinWidth="0"
                    Background="Lime">მეორე</Button>
<Button Grid.Row="1" Grid.Column="0" MinWidth="0"
                    Background="Aquamarine">
                        ყველაზე განიერი</Button>
<Button Grid.Row="1" Grid.Column="1" MinWidth="0"
                    Background="Yellow">მეოთხე</Button>
    </Grid>
  </TabItem>
 <TabItem Header="Grid 6">
    <StackPanel Grid.IsSharedSizeScope="True">
      <TextBlock HorizontalAlignment="Center">
          ბადეები სვეტების განზოგადებული ზომებით
      </TextBlock>
    <StackPanel Margin="5" />
```

```
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="myKey1" />
            <ColumnDefinition Width="Auto" SharedSizeGroup="myKey2" />
          </Grid.ColumnDefinitions>
  <Button Grid.Row="0" Grid.Column="0"
                       Background="LightPink">პირველი</Button>
  <Button Grid.Row="0" Grid.Column="1"
                       Background="Lime">მეორე</Button>
  <Button Grid.Row="1" Grid.Column="0"
                       Background="Aquamarine">
                       ყველაზე განიერი 1–ელ სვეტში  </Button>
  <Button Grid.Row="1" Grid.Column="1"
                       Background="Yellow">მეოთხე</Button>
</Grid>
    <StackPanel Margin="5" />
      <Grid>
        <Grid.RowDefinitions>
          <RowDefinition />
          <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto" SharedSizeGroup="myKey1" />
          <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
    <Button Grid.Row="0" Grid.Column="0"
                         Background="LightPink">1</Button>
    <Button Grid.Row="0" Grid.Column="1"
                         Background="Lime">2</Button>
```

```
    <Button Grid.Row="1" Grid.Column="0"
                            Background="Aquamarine">3</Button>
    <Button Grid.Row="1" Grid.Column="1"
                            Background="Yellow">4</Button>
</Grid>
    <StackPanel Margin="5" />
      <Grid>
          <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="Auto" SharedSizeGroup="myKey2" />
          </Grid.ColumnDefinitions>
      <Button Grid.Row="0" Grid.Column="0"
                            Background="LightPink">1</Button>
      <Button Grid.Row="0" Grid.Column="1"
                            Background="Lime">2</Button>
      <Button Grid.Row="1" Grid.Column="0"
                            Background="Aquamarine">3</Button>
      <Button Grid.Row="1" Grid.Column="1"
                            Background="Yellow">4</Button>
    </Grid>
      <StackPanel Margin="5" />
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="myKey1" />
            <ColumnDefinition Width="Auto" SharedSizeGroup="myKey2" />
```

```
            </Grid.ColumnDefinitions>
        <Button Grid.Row="0" Grid.Column="0"
                                Background="LightPink">1</Button>
        <Button Grid.Row="0" Grid.Column="1"
                                Background="Lime">2</Button>
        <Button Grid.Row="1" Grid.Column="0"
                                Background="Aquamarine">3</Button>
        <Button Grid.Row="1" Grid.Column="1"
                                Background="Yellow">განიერი მე–2
                                                    სვეტისთვის</Button>
      </Grid>
    </StackPanel>
  </TabItem>
  <TabItem Header="Grid 7">
      <Grid>
        <Grid.RowDefinitions>
          <RowDefinition Height="Auto" />
          <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBlock Grid.Row="0" HorizontalAlignment="Center" Margin="5">
          სვეტების გაერთიანება და GridSplitter ელემენტი
        </TextBlock>
        <Grid Grid.Row="1">
          <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto"/>
          <RowDefinition Height="Auto" MinHeight="50" MaxHeight="150" />
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="Auto" MinWidth="40"
                                            MaxWidth="200" />
```

```
            </Grid.ColumnDefinitions>
        <Button Grid.Row="0" Grid.ColumnSpan="3"
                                Background="LightPink">პირველი და
                                                        მეორე</Button>
                <GridSplitter
                    Grid.Row="1"
                    Grid.ColumnSpan="3"
                    Height="2"
                    ResizeDirection="Rows"
                    ResizeBehavior="PreviousAndNext"
                    HorizontalAlignment="Stretch" />
      <Button Grid.Row="2" Grid.Column="0"
                            Background="Aquamarine">მესამე</Button>
                 <GridSplitter
                    Grid.Row="2"
                    Grid.Column="1"
                    Width="2"
                    ResizeDirection="Columns"
                    ResizeBehavior="PreviousAndNext"
                    VerticalAlignment="Stretch" />
      <Button Grid.Row="2" Grid.Column="2"
                            Background="Yellow">მეოთხე</Button>
              </Grid>
            </Grid>
          </TabItem>
       </TabControl>
    </Window>
```

2.15. WPF-ის საკუთარი განლაგების მენეჯერის დამუშავება: MyPanel
(ლაბორატორიული სამუშაო N 15)

მიზანი: WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების **საკუთარი** მენეჯერის MyPanel დამუშავების და ფუნქციონალობის შესწავლა.

თეორიული ნაწილი:

ამოცანა: ავაგოთ Wpf-აპლიკაცია, რომლის ინტერფეისის ფანჯარაში წრიულად განთავსდება რამდენიმე ელემენტი (მაგალითად, ღილაკები), თითოეული განსაზღვრული კუთხის მობრუნებით (ნახ.2.77).

ნახ.2.77

ესაა არასტანდარტული ამოცანა და მის გადასაწყვეტად განლაგების ტიპური მენეჯერები არ გამოდგება. საჭიროა მისთვის

სპეციალური ალგორითმის და კოდის შექმნა. განვიხილოთ ასეთი შესაძლებლობანი.

WPF–ის ყველა სტანდარტული პანელი (DockPanel, StackPanel, Canvas, UniformGrid, Grid) არის აბსტრაქტული საბაზო კლასის Panel მემკვიდრე. ჩვენ მიერ ასაგები ახალი ელემენტების განლაგების „კერძო მენეჯერი“–ც უნდა იყოს ამ ბიბლიოთეკური კლასის მემკვიდრე.

Panel კლასი, თავის მხრივ მემკვიდრეა UIElement კლასის, ამიტომ ჩვენი კლასი ყველა აუცილებელ გაფართოებას მიიღებს თავის წინაპრებიდან. Panel კლასი არის მემკვიდრეობის ჯაჭვის ისეთი მწვერვალი, რომელიც მოიცავს ყველა ნიმუშს (პატერნს: ყალიბი, „აგური“, „სამშენებლო ბლოკი“, საბაზო ფუნქციური ობიექტი), რომლებიც აუცილებელია განლაგების ნებისმიერი მენეჯერის ასაგებად. ჩვენ შემთხვევაში გაფართოების კლასში დაგვჭირდება რამდენიმე ვირტუალური მეთოდის გადაფარვა.

UIElement ობიექტური მოდელის ის ნაწილი, რომელიც მიეკუთვნება განლაგების მენეჯერს, საკმარისად მარტივია. მეთოდები Measure, MeasureCore, Arrange და ArrangeCore არეალიზებენ განლაგების ორ ეტაპს, ხოლო Visibility თვისება წყვეტს, უნდა აისახოს თუ არა შვილობილი და უნდა განთავსდეს თუ არა იგი.

Visibility თვისება იძლევა შვილობილი ელემენტის განთავსების სამ ხერხს. ავტომატურად ესაა Visible: ელემენტი აისახება და იკავებს ადგილს ეკრანზე. Hidden–ით ის ეკრანზე არ ჩანს, მაგრამ ადგილს იკავებს. Collapsed–ს დროს ელემენტი არც აისახება და არ ადგილს იკავებს.

შვილობილ ელემენტსა და განთავსების მენეჯერს შორის ურთიერთქმედების მექანიზმი შედგება ორი ნაწილისგან:

- კონტრაქტისგან, რომელიც აღწერს თუ როგორ იღებს მონაწილეობას ელემენტი განთავსებაში;

- ამ კონტრაქტის რეალიზაციების ერთობლიობისგან.

არაა არავითარი ჩაშენებული განთავსება, მისი ყველა კონკრეტული ხერხი აგებულია საბაზო კლასების გაფართოების პრინციპზე.

WPF–ის ყველა ელემენტი, მათ შორის განლაგების მენეჯერებიც, ადაპტირდება თავის შიგთავსზე. ეს კონცეფცია გამოიყენება მომხმარებლის ინტერფეისის აგების ყველა დონეზე: ფანჯრებს შეუძლია თავიანთი ზომების დაყვანა მათში არსებულ ელემენტებზე, ღილაკებს – მათზე არსებულ წარწერებზე, ტექსტური ველები – იცვლიან ზომებს ისე, რომ მოთავსდეს მასზე ყველა სიმბოლო.

მენეჯერები ახორციელებს თავის შვილობილი ელემენტების ორეტაპიან განთავსებას: გაზომვა და დაყენება. თავიდან მენეჯერი გამოკითხავს ყველა შვილობილ ელემენტს იმის შესახებ, თუ როგორი ფაქტობრივი ზომაა მისთვის მისაღები, რომ ეკრანზე მთლიანად დაატიოს შიგთავსი. ესაა გაზომვის ეტაპი. შემდეგ მენეჯერი ამ ინფორმაციას მიუსადაგებს თავის ანაწყობათა შესაძლებლობებს, რამდენი შეუძლია დაატიოს, და იწყებს მონტაჟს.

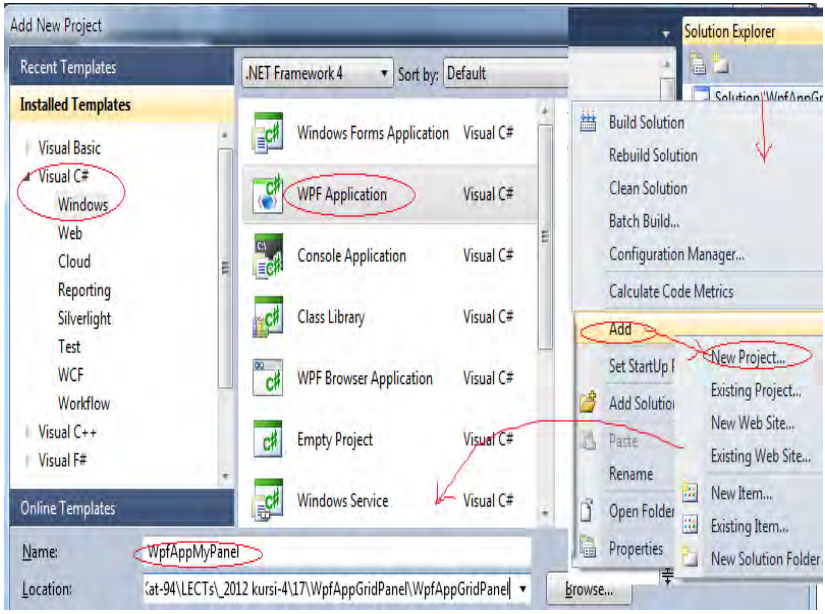
მონტაჟის დროს მშობელი ულაპარაკოდ თხოულობს თავის თითოეული შვილობილი ელემენტიდან, რომ მან შეარჩიოს განსაზღვრული ზომა და მდებარეობა. ეს მოდელი უზრუნველყოფს, რომ მშობელმა და შვილება მოილაპარაკონ ეკრანზე ადგილის შესახებ (ადგილზე მზის ქვეშ). ამ დროს სამი ზომა ფიგურირებს:

- შესაძლებელი ზომა (available size) – მაქსიმალური არე, რომელიც შეუძლია მშობელს თავიდან მისცეს შვილს;
- სასურველი ზომა (desired size) – ზომა, რომელსაც ისურვებს შვილი დასაყენებლად;
- ფაქტობრივი ზომა (actual size) – საბოლოო ზომა, რომელსაც მშობელი გამოუყოფს შვილს.

ელემენტის ზომებზე შიძლება შეზღუდვების დაყენება MinWidth, MaxWidth, MinHeight და MaxHeight, რომელთა შიგნითაც მოხდება ადაპტაცია შიგთავსის მიხედვით. თვისებები Width და Height ჩვეულებისამებრ არ მოიცემა, რაც უფლებას იძლევა ზომის ავტომატურად შერჩევისთვის. მაგრამ თუ ისინი მოცემულია ცხადად, მაშინ ადაპტაცია ზომებით გამოირთვება. თვისებები ActualWidth და ActualHeight ხდება განსაზღვრული მხოლოდ ორეტაპიანი ადაპტაციის დასრულების შემდეგ და ნიშნავს ელემენტის ფაქტობრივ ზომას, რომელიც დააყენა განლაგების მენეჯერმა.

პრაქტიკული ნაწილი:

1. დავამატოთ Solution Explorer-ში ახალი, WpfAppMyPanel სახელის პროექტი და გავხადოთ სასტარტო.



ნახ.2.78

2. შევიტანოთ C#-ის MainWindow.xaml.cs ფაილში შემდეგი კოდი:

```
using System;
```

```
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfAppMyPanel
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}

namespace WpfAppMyPanel
{
    class MyPanel : Panel
    {
      // პირველი ეტაპი: გაზომვა
      // შესასვლელზე: - რამდენი აქვს მშობელს მენეჯერისთვის
      // გამოსასვლელზე:- რამდენს ითხოვს მენეჯერი თავისთვის
  protected override Size MeasureOverride(Size availableSize)
      {
         double maxChildWidth = 0.0;
         double maxChildHeight = 0.0;

           // გაიზომოს ყოველი შვილის მოთხოვნა და
           //  დადგინდეს ყველაზე მაქსიმუმები
         foreach (UIElement child in this.InternalChildren)
           {
          child.Measure(availableSize);
          maxChildWidth =  Math.Max(child.DesiredSize.Width,
                                                 maxChildWidth);
```

```
            maxChildHeight = Math.Max(child.DesiredSize.Height,
                                                    maxChildHeight);
        }

        // მიახლოებითი არასრულყოფილი ალგორითმი
        // ყველა ელემენტის განსათავსებლად საჭირო წრის სიგრძე
    double idealCircumference = maxChildWidth *
                                this.InternalChildren.Count;
        // წრის საჭირო რადიუსი
    double idealRadius = (idealCircumference / (Math.PI * 2) +
                                                    maxChildHeight);
        // წრეზე შემოხაზული კვადრატის აუცილებელი ზომები
    Size ideal = new Size(idealRadius * 2, idealRadius * 2);
    Size desired = ideal; // მენეჯერს უნდა ამდენი თავისთვის
     // თუ მენეჯერისათვის გამოყოფილი ზომა არაა უსასრულო
            if (!double.IsInfinity(availableSize.Width))
            {
                // თუ გამოყოფილი ზომა მენეჯერს არ ყოფნის
                if (availableSize.Width < desired.Width)
                 {
              // მენეჯერის ზომის კორექტირება – სიგანისთვის
                    desired.Width = availableSize.Width;
                 }
            }
            // იგივე სიმაღლისთვის
            if (!double.IsInfinity(availableSize.Height))
            {
                if (availableSize.Height < desired.Height)
                {
                    desired.Height = availableSize.Height;
                }
            }
            return desired;
        }

        // მეორე ეტაპი – მონტაჟი (კონტრაქტის დადება)
        // მენეჯერისთვის გამოყოფილი ფართობის დაყოფა
    protected override Size ArrangeOverride(Size finalSize)
        {
```

```
            // ვათავსებთ კვადრატულ მენეჯერს მშობლის მიერ
            // გამოყოფილ უბნის ცენტრში
            Rect layoutRect;
            if (finalSize.Width > finalSize.Height)
             {
              layoutRect = new Rect((finalSize.Width -
                       finalSize.Height) / 2,0,
                       finalSize.Height, finalSize.Height);
              }
            else
              {
               layoutRect = new Rect(0,(finalSize.Height -
                       finalSize.Width) / 2,
                       finalSize.Width, finalSize.Width);
              }

        double angleInc = 360.0 / this.InternalChildren.Count;
        double angle = 0;

            // ვათავსებთ წრიულად ყველა შვილ ელემენტს
        foreach (UIElement child in this.InternalChildren)
         {
            // ვათავსებთ შემდეგ ელემენტს წრის ზედა წერტილში
                 Point childLocation = new Point(
                 layoutRect.Left + ((layoutRect.Width -
                             child.DesiredSize.Width) / 2),
                             layoutRect.Top);
            // გადაადგილება საათის ისრის მიხედვით და
            // შემობრუნება ღერძის გარშემო
             child.RenderTransform = new RotateTransform(angle,
                     child.DesiredSize.Width / 2,
                      finalSize.Height / 2 - layoutRect.Top);

    // დადგინდა შვილი-ელემენტის საბოლოო ზომა და მდებარეობა
   child.Arrange(new Rect(childLocation, child.DesiredSize));
                 angle += angleInc;
         }
        return base.ArrangeOverride(finalSize);
      }
    }
}
```

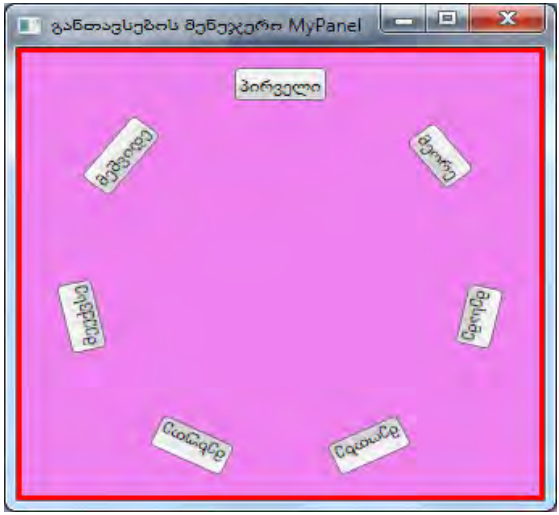
3. `Arrange` მეთოდის გამოძახება შვილი–ელემენტისთვის ასაბუთებს მასთან კონტრაქტის დადებას, რომლის გარეშეც ელემენტი ვერ გამოჩნდება ეკრანზე.

4. შევავსოთ MainWindow.xaml ფაილი შემდეგი XAML სკრიპტით:

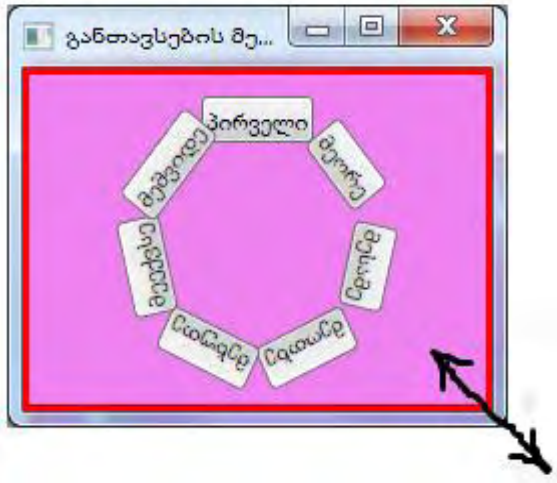
```
<Window x:Class="WpfAppMyPanel.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Height="300" Width="300"
        Title="განთავსების მენეჯერი MyPanel"
        xmlns:MyMy="clr-namespace:WpfAppMyPanel"
    >
    <Border
        BorderThickness="3"
        CornerRadius="0"
        BorderBrush="Red"
        Padding="10"
        Background="Violet"
        >
        <MyMy:MyPanel>
            <Button>პირველი</Button>
            <Button>მეორე</Button>
            <Button>მესამე</Button>
            <Button>მეოთხე</Button>
            <Button>მეხუთე</Button>
            <Button>მეექვსე</Button>
            <Button>მეშვიდე</Button>
        </MyMy:MyPanel>
    </Border>
</Window>
```

შენიშვნა: MyPanel კლასის ხილვადობისათვის დიზაინის ნაწილში ჩვენ მივუერთეთ მას CLR –შესრულების გარემოს სახელსივრცე, აღნიშნული ნებისმიერი სახელით (MyMy=”clr…).

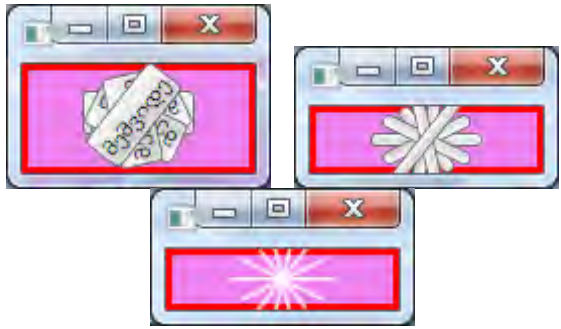
5. აპლიკაციის ამუშავებით მიიღება შედეგები:



ნახ.2.79



ნახ.2.80–ა



ნახ.2.80–ბ

*** დასასრული ***

სამუშაოთა ამ ციკლით ჩვენ შევისწავლეთ Windows-აპლიკაციათა პროექტებისთვის ინტერფეისული ელემენტების განლაგების პრინციპები, რომლებიც გამოიყენება მთლიან WPF ბიბლიოთეკაში.

წიგნის შემდეგი ნაწილები შეეხება WF (Workflow Foundation - ნაწ.2) და WCF (Windows Communication Foundation - ნაწ.3) ტექნოლოგიების შესწავლას.

გამოყენებული ლიტერატურა:

1. სურგულაძე გ. ვიზუალური დაპროგრამება C#_2010 ენის ბაზაზე. სტუ, თბ., 2011

2. Мак-Дональд М. WPF: Windows Presentation Fundation в .NET 3.5 с примерами на C# 2008 для профессионаюв. 2-е издание: Пер. с англ. - М. : ООО "И.Д. Вильяме". 2008

3. Petzold Ch. Applications=Code+Markup. A Guide to the MicroSoft Windows Presentation Foundation. St-Petersburg. 2008

4. Уотсон К.,Нейгел К., Педерсен Я., ХаммерР., Джон Д., Скиннер М., Уайт Э. Visual C# 2008: базовый курс. : Пер. с англ. - М. : ООО "И.Д. Вильяме", 2009

5. Долженко А.И. Разработка приложений на базе WPF и Silverlight. –М., 2011, www.intuit.ru/department/se/dawpfsl/

6. Снетков В.М. Практикум прикладного программирования на C# в среде VS.NET 2008. www.intuit.ru/department/se/prcsharp08/

7. Eberhardt C. WPF DataGrid Practical Examples. 2009. http://www.codeproject.com/Articles/30905/WPF-DataGrid-Practical-Examples.

8. Wegener J. WPF 4.5 und XAML. Grafische Benutzeroberflächen für Windows inkl. Entwicklung von Windows Store Apps. Herausgeber: Holger Schwichtenberg, www.IT-Visions.de, Essen. 2013 Carl Hanser Verlag München, *www.hanser-fachbuch.de*

• • •

გადაეცა წარმოებას 1.11.2014 წ. ხელმოწერილია დასაბეჭდად 25.11.2014 წ. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 12. ტირაჟი 100 ეგზ.

Verba volant, scripta manent

სტუ-ს „IT კონსალტინგის ცენტრი“
თბილისი, კოსტავას 77

gsurg@gmx.net

გია სურგულაძე, საქართველოს ტექნიკური უნივერსიტეტის „მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერია) კათედრის პროფესორი, ტექნიკის მეცნიერებათა დოქტორი, გაეროსთან არსებული ინფორმატიზაციის საერთაშორისო აკადემიის წამდვილი წევრი, IT-კონსალტინგის სამეცნიერო ცენტრის ხელმძღვანელი, სტუ-ს საერთაშორისო სამეცნიერო ჟურნალის „მართვის ავტომატიზებული სისტემები (ACS)“ რედაქტორი.

1974-75 წლებში იყო გერმანიის მაგდებურგის უნივერსიტეტის ასპირანტ-სტაჟიორი. პირველი დისერტაცია, მეცნიერებათა კანდიდატის ხარისხის მოსაპოვებლად დაიცვა 1980 წელს პეტერბურგის ელექტროტექნიკურ უნივერსიტეტში: „მონაცემთა რელაციური ბაზის სტრუქტურის დაპროექტების ავტომატიზაცია“, მეორე, ტექნიკის მეცნიერებათა დოქტორის ხარისხისა კი 1993 წელს სტუ-ში: „ავტომატიზებული სამუშაო ადგილების ქსელის დაპროექტების ტექნოლოგია საწარმოო გაერთიანებისათვის“.

აქვს 300-ზე მეტი სამეცნიერო ნაშრომი, მათ შორის 60 წიგნი, 40 ელ-სახელმძღვანელო საინფორმაციო სისტემების და მონაცემთა ბაზების დაპროექტების და აგების სფეროში, რამდენიმე მათგანი გერმანელ კოლეგებთან ერთად. არის გერმანიის DAAD-ის მრავალგზის გრანტის მფლობელი, ბერლინის ჰუმბოლდტის, პასაუს, ჰალეს და ნიურნბერგ-ერლანგენის უნივერსიტეტების მიწვეული პროფესორი 1980-2014 წლებში. 1997 წელს მიენიჭა ჟიული შარტავას სახელობის პრემია ტექნიკის დარგში. არის USAID-ის ექსპერტი, ბიზნეს-კონსულტანტი.