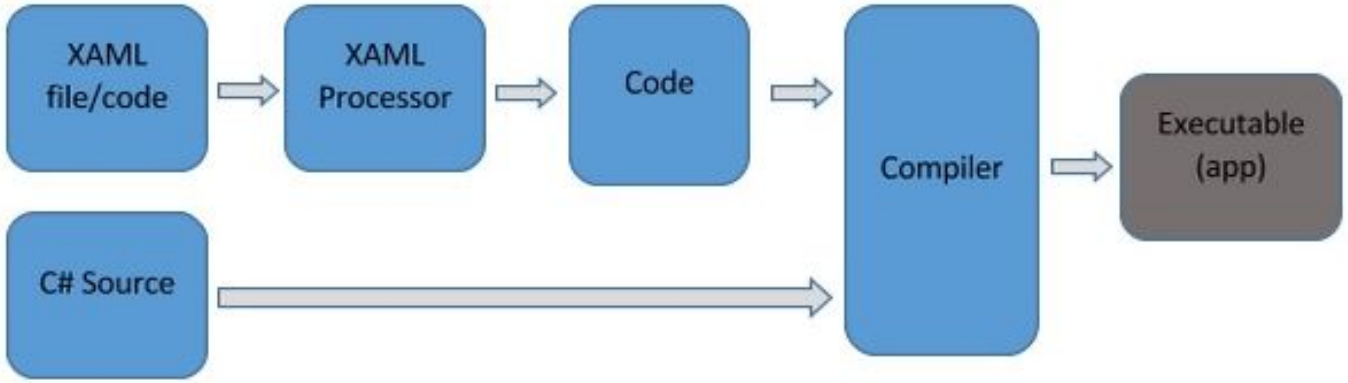
WPF

OSNOVNI ELEMENTI GUI-A

XAML

- Extensible Application Markup Language
- jednostavan i deklarativni jezik zasnovan na XML-u
  - omogućava vrlo lako kreiranje, incijalizaciju i postavljanje atributa objekata sa hijerahijskom strukturom;
  - koristi se za kreiranje dizajna GUI-a, ali može da se koristi i u druge svrhe, kao što je deklarisanje procesa rada.
- Za razliku od drugih markup jezika, XAML je u toku interpretiranja u direktnoj vezi sa sistemom za podršku.

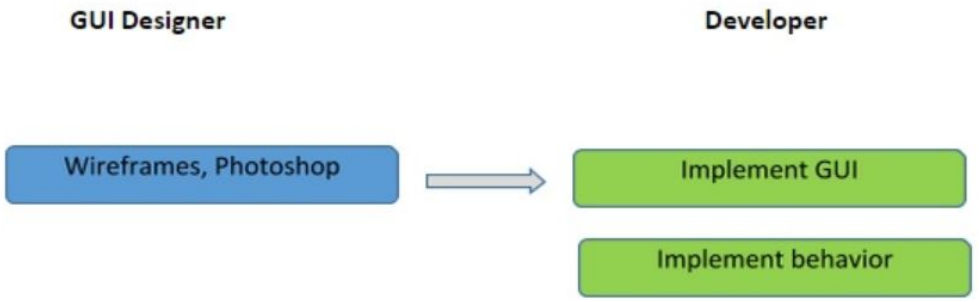
Kako radi XAML?



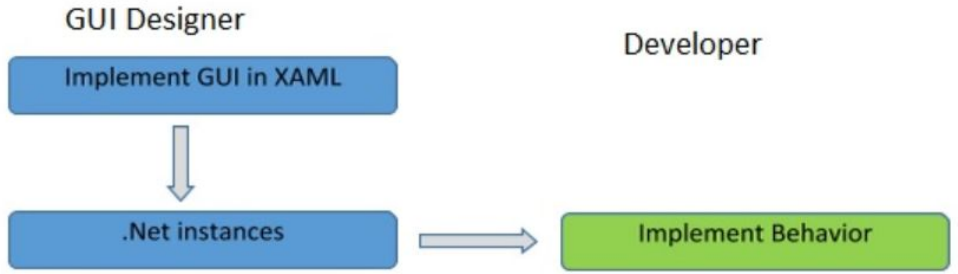
- XAML fajl se interpretira pomoću XAML procesora; rezultat interpretiranja je interni kod koji opisuje UI elemente; dobijeni interni kod i C# kod su povezani kroz definiciju parcijalnih klasa, a zatim .NET kompajler prevodi povezani kod i gradi aplikaciju.

XAML - prednosti

- XAML nam omogućava jednostavniji razvoj aplikacija i paralelan rad programera i dizajnera zato što je upotrebom XAML lako odvojiti dizajnerski kod korisničkog interfejsa od definicije njegovog ponašanja; unutar XAML fajla definišemo UI elemente, a ponašanje definišemo upotrebom C# koda.
- lak za čitanje i razumevanje



Bez XAML-a



Sa XAML-om

XAML struktura fajla

- XAML fajl se sastoji od elemenata; svaki element se preslikava u .NET tip, a atribute .NET tipa definišemo kao atribute elementa; referenciranjem CLR tipa u XAML dobijamo pristup i naslednicima tog tipa.
- Svaki XAML fajl ima jedan korenski element unutar kojeg se definišu svi ostali elementi; korenski elementi su obično elementi tipa Window ili Page; ovaj element sadrži atribut ***xmlns*** za definisanje prostora imena u kojima se nalaze definicije tipova koje ćemo referencirati iz XAML fajla; ovaj atribut je **NEOPHODAN NA KORENSKOM ELEMENTU** svake XAML datoteke i može da ih bude definisano više samo uz korišćenje različitih sufiksa (*npr. xmlns:x*); definicija *xmlns* se primenjuje na sve elemente u XAML fajlu i nema potrebe da je navodimo u svakom elementu.
- Kada kreiramo novu formu, uvek ćemo u XAML fajlu već imati specificirana dva prostora imena, a to su podrazumevani (*xmlns*) i *xmlns:x* prostor imena, tako da o tome nema potrebe da vodimo računa.

XAML prostori imena

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
</Page>
```

Podrazumevani prostor imena

Dodatni prostor imena koji podržava gradivne elemente XAML jezika

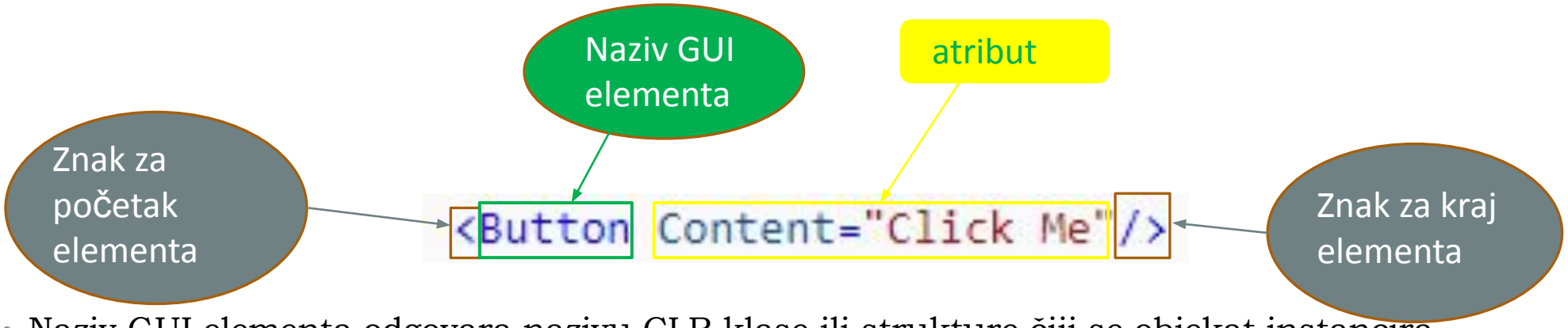
- Elementi koji se referenciraju iz podrazumevanog prostora imena ne moraju da budu navedeni uz upotrebu prefiksa, dok elementi iz drugih prostora imena zahtevaju strogu upotrebu prefiksa ispred naziva elementa koji se referencira (npr. ako hoćemo da referenciramo neki element iz prostora imena koji je naveden kao vrednost xmlns:x atributa, onda ćemo koristiti prefiks “x:” ispred naziva elementa).

XAML – gradivni elementi XAML jezika

- Prostor imena u kojima se ovi elementi nalaze specificiramo pomoću *xmlns:x* atributa korenskog elementa.
- Najčešći elementi koje ćemo koristiti:
  - *x:Key* – postavlja jedinstven ključ za svaki resurs ;
  - *x:Class* – navodi CLR prostor imena i ime klase u kojoj je definisano ponašanje kreiranog korisničkog interfejsa;
  - *x:Name* – ime objekta za vreme izvođenja za instancu koja postoji u kodu; pomoću ovog imena možemo da pristupimo GUI elementu iz koda u kojem definišemo ponašanje;
  - *x:Type* – predstavlja referencu na Type koja se zasniva na nazivu tipa; koristi se za navođenje atributa *Type* kao što je *Style.TargetType*;
  - *x:Static* – omogućava referencu koja vraća statičku vrednost koja nije kompatibilna sa XAML atributom.

XAML sintaksa elemenata

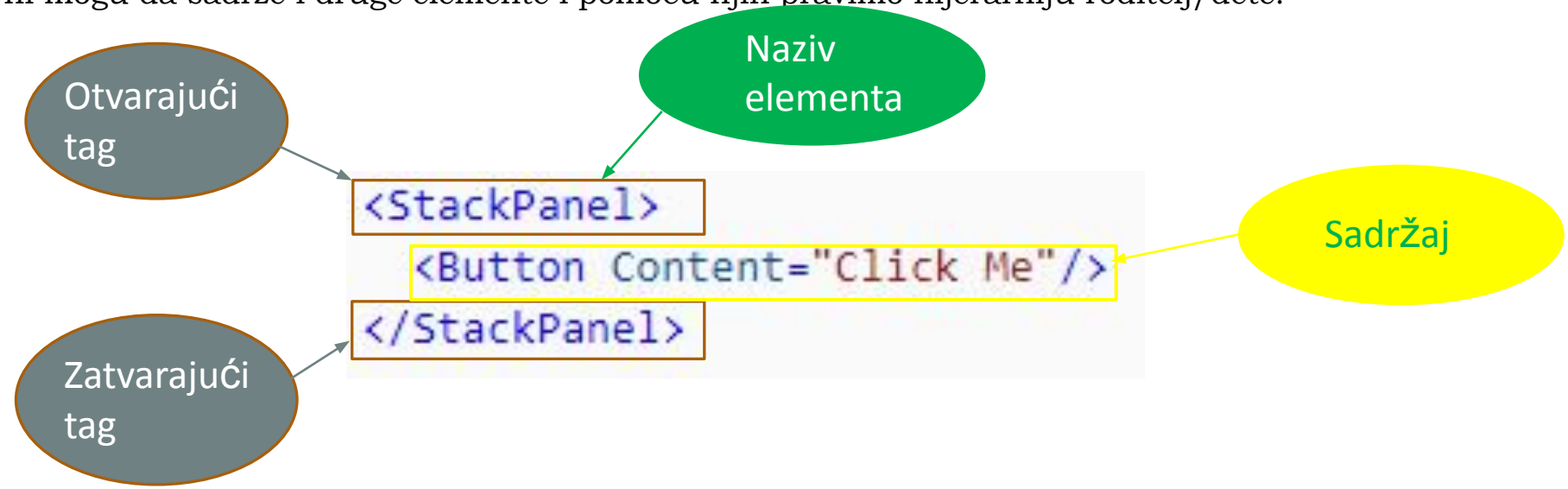
- XAML fajl se sastoji od elemenata, a njihova sintaksa je slična sintaksi elemenata u HTML-u. Postoje 2 vrste elemenata, a to su:
  - Prosti
    - oni u sklopu sebe ne mogu da sadrže druge elemente;
    - sastoje se samo od otvarajućeg taga.



  - Naziv GUI elementa odgovara nazivu CLR klase ili strukture čiji se objekat instancira

XAML sintaksa elemenata

- Složeni
  - Oni mogu da sadrže i druge elemente i pomoću njih pravimo hijerarhiju roditelj/dete.



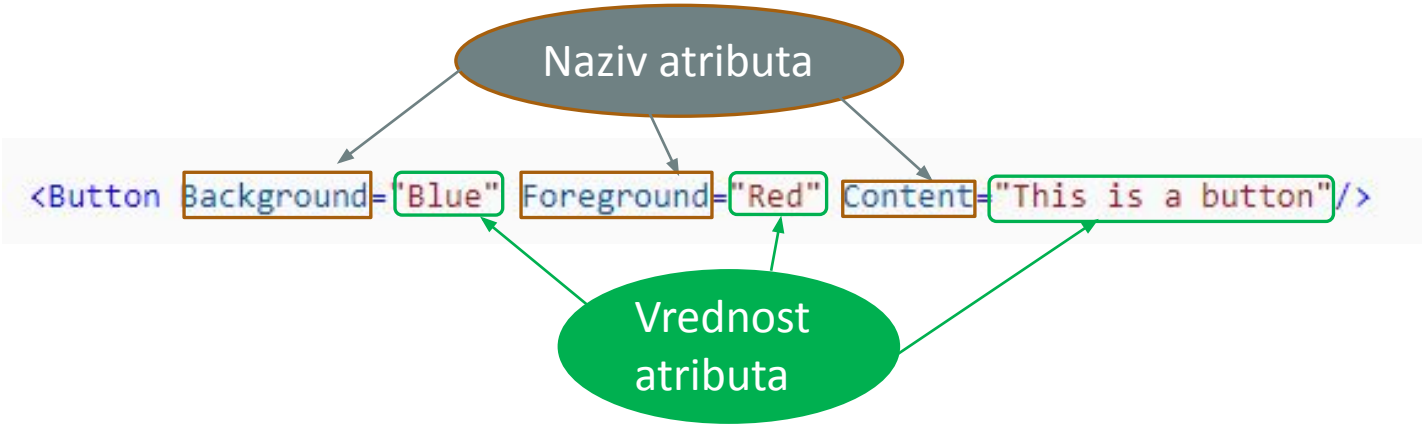
- **TAGOVI NE SMEJU DA SE PREKLAPAJU!**

XAML atributi elementa

- Definisanjem atributa objektima dodajemo karakteristike; početno stanje objekta zasniva se na konstruktoru bez parametara.
- Nekom elementu atribute možemo da dodelimo na 2 načina:
  - definisanjem atributa direktno u elementu (sintaksa atributa)
  - definisanjem atributa kao posebnog elementa (element kao atribut).

XAML atributi elemenata - atribut u elementu

- Naziv atributa mora da odgovara imenu CLR atributa klase koja podržva određeni element; takođe, da bismo mogli da postavimo vrednost atributu, atribut mora da bude javan i da nije readonly ili const.
- Vrednost atributa mora da bude tip vrednosti ili referentni tip koji XAML procesor može da instancira ili referencira prilikom pristupa relevantnom tipu sigurnosne kopije.
- Naziv atributa može da bude događaj, ali je potrebno da taj događaj bude javan i da ima javni delegat.



XAML atributi elementa

- Vrednost atributa je niz sadržan u navodnicima i nju obrađuje XAML procesor. Vrednost atributa mora da ispunjava jednu od sledećih stvari:
  - ako je vrednost napisana između vitičastih zagrada ili ako je objektni element koji proizilazi iz *MarkupExtension-a*, tada se prvo procenjuje referentno proširenje; niz se ne obrađuje, već se objekat koji je predstavljen pomoću MarkupExtension-a koristi kao vrednost; vraćen objekat u većini slučajeva predstavlja referencu na već postojeći objekat ili izraz koji odlaže evaluaciju do vremena izvršavanja, a nije novoinstancirani objekat;
  - ako je atribut ili tip vrednosti tog atributa deklarisano nekim konvertorom (*TypeConvertor)*, onda se vrednost string atributa prosleđuje konvertoru kao ulaz za pretvaranje, a konvertor vraća novu instancu objekta;
  - ako ne postoji konvertor (*TypeConvertor*), onda se pokušava direktno pretvaranje vrednosti u tip atributa.

XAML atributi elementa - atribut kao element

- Koristimo ga kada za neki atribut ne možemo adekvatno da izrazimo njegovu vrednost upotrebom navodnika ili zbog strogih ograničenja navođenja atributa.

PREPORUČUJE SE UPOTREBA SINTAKSE ATRIBUTA AKO JE TO MOGUĆE!

XAML atributi elementa

- Događaj kao atribut elementa
  - Sintaksu atributa možemo da iskoristimo i za članove koji su događaji.
  - Ime atributa je ime događaja, a vrednost je ime obrađivača događaja koji implementira delegat tog događaja.

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="ExampleNamespace.ExamplePage">
  <Button Click="Button_Click" >Click Me!</Button>
</Page>
```

  - Na slici iznad smo definisali događaj *Click* i za njegovu vrednost postavili *Button_Click*, što znači da u klasi *ExamplePage* imamo definisanu metodu *Button_Click*, koja je obrađivač događaja *klik na dugme (Click)*. Na sledećem slajdu ćemo videti kako izgleda ta metoda.

XAML atributi elementa

```
namespace ExampleNamespace
{
  public partial class ExamplePage
  {
    void Button_Click(object sender, RoutedEventArgs e)
    {
      Button b = e.Source as Button;
      b.Foreground = Brushes.Red;
    }
  }
}
```

- Na slici vidimo kako izgleda metoda *Button_Click* koja je obrađivač na događaj prikazan na prethodnom slajdu. *Sender* parametar ukazuje na onoga ko je okinuo događaj, a pomoću *RoutedEventArgs* prosleđujemo parametre događaja.

Kontrole

- Kontrole su drugi naziv za komponente korisničkog interfejsa. Pod kontrolom podrazumevamo bilo koji vidljivi objekat u aplikaciji. Sve kontrole u WPF su predstavljene klasom *Control*. Da bi nešto bilo vidljivo u aplikaciji, nije potrebno da nasledi klasu *Control*. Klase koje su naslednice klase *Control* sadrže *ControlTemplate*, što korisniku omogućava da radikalno promeni izgled kontrole bez potrebe za stvaranjem nove podklase.
- Kontrole su: *Button*, *TextBox*, *RadioButton*, *CheckBox*...

Kreiranje kontrola

- Kontrole možemo da kreiramo na 2 načina:
  - upotrebom XAML jezika – dodavanje kontrole u XAML fajl, možemo da otkucamo kod ili samo da željenu kontrolu prevučemo iz *Toolbox-a* na dizajn prozora;
  - pisanjem C# koda.

```
<Label>
  Enter your first name:
</Label>
<TextBox Grid.Row="0" Grid.Column="1"
         Name="firstName" Margin="0,5,10,5"/>

<Label Grid.Row="1" >
  Enter your last name:
</Label>
```

Dodavanje kontrole u XAML fajl

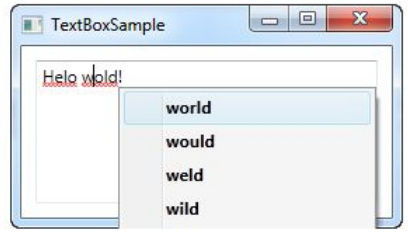
XAML fajl

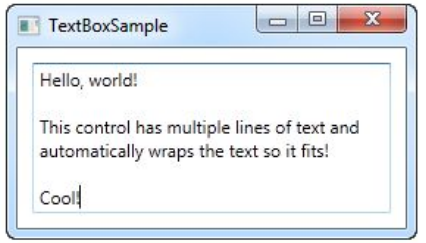
```
firstNameLabel = new Label();
firstNameLabel.Content = "Enter your first name:";
grid1.Children.Add(firstNameLabel);

firstName = new TextBox();
firstName.Margin = new Thickness(0, 5, 10, 5);
Grid.SetColumn(firstName, 1);
grid1.Children.Add(firstName);
```

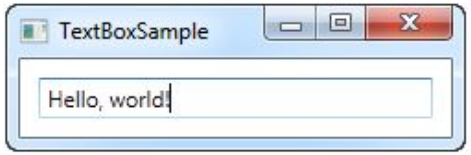
Dodavanje kontrole u kodu

C# kod

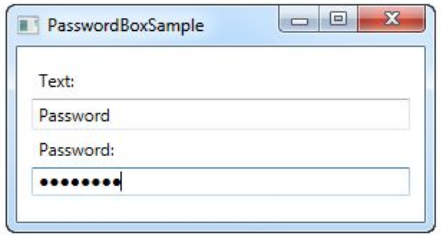
Kontrole i njihova upotreba







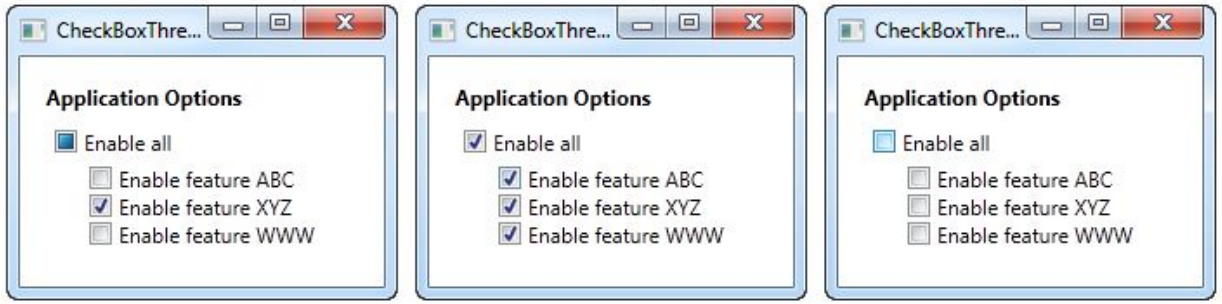
- Kontrole za unos sadržaja sa tastature
  - *TextBox*
    - služi za unos neformatiranog teksta;
    - ima ugrađenu podršku za višelinijski tekst i za automatski „*spellcheck*”.
  - *PasswordBox*
    - Služi za unos lozinke.



Obe kontrole, i *TextBox* i *PasswordBox*, imaju ugrađene obrađivače za događaje, *MouseUp* i *MouseDown*, tako da ne možete da napravite svoje obrađivače za ove dve vrste događaja. Ako treba to da uradite možete da iskoristite *PreviewMouseUp* i *PreviewMouseDown* događaje ili da registrujete obrađivač događaja sa argumentom *HandlerEventTo*.

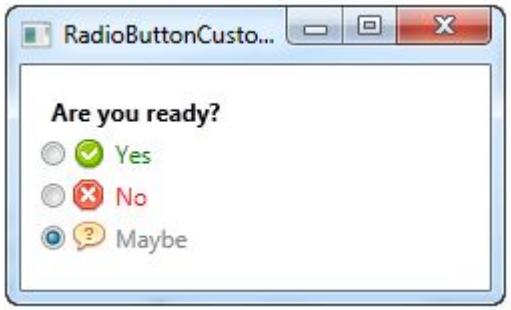
Kontrole i njihova upotreba

- Kontrole za selekciju
  - *CheckBox*
    - služi da korisniku omogućite da izabere nula ili više ponuđenih opcija;
    - svako dugme ima 3 stanja, a to su selektovano, neselektovano i neodređeno;
    - iako se obično nalaze u grupi, svako dugme radi pojedinačno, tako da korisnik može nezavisno da potvrdi ili poništi opciju;
    - obično se koristi da se prikažu ponuđeni odgovori na neka pitanja.



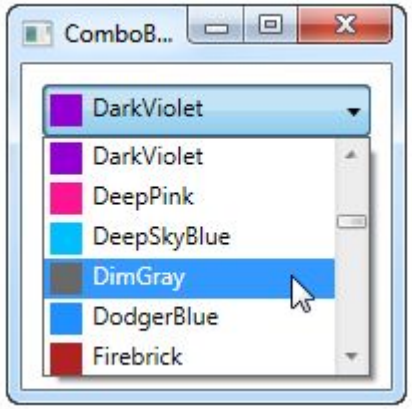
Kontrole i njihova upotreba

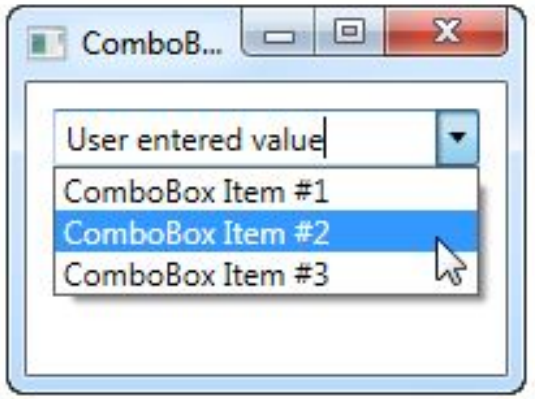
- *RadioButton*
  - kontrola koja služi da korisnika ograničimo da može da izabere samo jednu od ponuđenih opcija;
  - svako dugme može da bude u 2 stanja – selektovano ili ne;
  - kao i *CheckBox*, i ova kontrola se prikazuje u grupi, ali za razliku od *CheckBox*-a gde svako dugme radi pojedinačno, ovde se dugmići posmatraju kao jedna kontrola;
  - koristi se za neka određena podešavanja, prebacivanje između opcija ili isključivanje nečega;
  - ne treba je koristiti ako imamo više od 5 opcija.



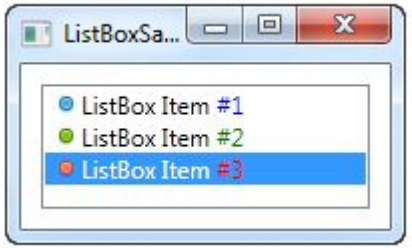
Kontrole i njihova upotreba

- *ComboBox*
  - Kontrola koja služi da korisnik odabere jednu od ponuđenih opcija iz padajuće liste ili da opciono unese novi tekst u tekstualno polje kontrole
  - Slična je *ListBox* kontroli, samo što zauzima manje prostora zato što se lista prikazuje samo kada je neophodna
  - Da bismo omogućili da korisnik može da unese novu opciju, potrebno je da atribut *IsEditable* postavimo na *True* (*IsEditable=True*)
  - Može da sadrži kolekciju bilo koje vrste
  - Poželjno koristiti ako imamo više ponuđenih opcija



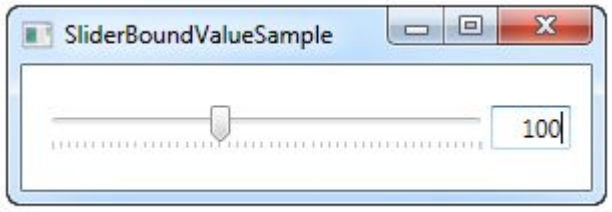


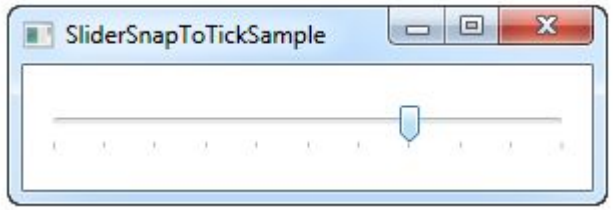
Kontrole i njihova upotreba



- *ListBox*
  - Kontrola koja korisniku omogućava da izabre jednu ili više stavki iz liste
  - Automatski daje vizuleni prikaz odabranih stavki
  - Uvek je prikazan cela lista, tako da je ne treba koristiti ukoliko imamo previše stavki u listi

- *Slider*
  - Omogućava korisniku da odabere vrednost iz nekog određenog raspona prevlačeći pokazivač
  - Za lakšu upotrebu možete da dodate da se ispod trake na kojoj se nalazi pokazivač prikazuju crtice koje označavaju vrednost ili da dodate da se sa strane prikazuje brojčana vrednost na koju pokazivač trenutno pokazuje





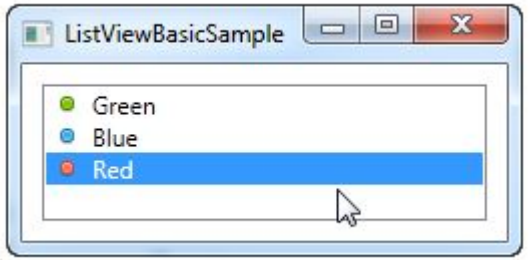
Kontrole i njihova upotreba

- Kontrole za prikaz podataka
  - *DataGrid*
    - Kontrola koja pruža fleksibilan način za prikaz kolekcije podataka u redovima i kolonama
    - Ima ugrađenu podršku za sortiranje, filtriranje, grupisanje...
    - Koristi se kada imate veliku količinu podataka

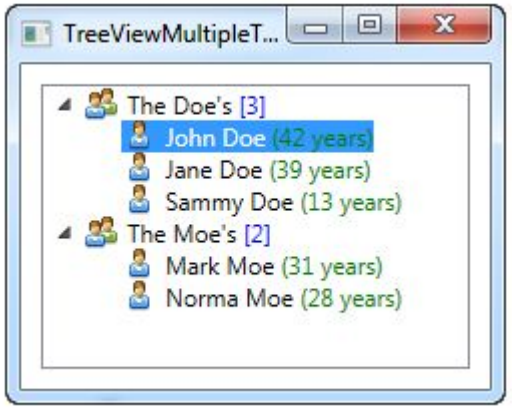
FirstName	LastName	Gender	WebSite	ReceiveNewsletter
Christian	Moser	Male	http://www.wpftutorial.net	☑
Peter	Meyer	Male	http://www.petermeyer.com	☐
Lisa	Simpson	Female	http://www.thesimpsons.com	☐
Betty	Bossy	Female	http://www.bettybossy.ch	☐
				☐

Kontrole i njihova upotreba

- *ListView*
  - Služi za jednostavan prikaz kolekcije podataka



- *TreeView*
  - Služi za hijerarhijski prikaz kolekcije podataka



Kontrole u kontrolama

- Kontrole u sebi mogu da sadrže i podređene elemente koji mogu biti druge kontrole
  - Kontrole koje prikazuju sadržaj smeju da imaju samo 1 podređeni element, zato što one nasleđuju klasu *ContentControl* koja može da sadrži samo jednu kontrolu u sebi
  - Kontrole za vizuelni prikaz (*Grid*, *WrapPanel*, *StackPanel*) mogu da sadrže više kontrola i one su naslednice klase *Panel*
  - Kontrola *Window* je kontrola sadržaja i može da sadrži samo jednu kontrolu, pa se zbog toga unutar kontrole *Window* stavlja neka od kontrola za vizuelni prikaz, a sve ostale kontrole se onda nalaze unutar kontrole za vizuelni prikaz

Kontrole u kontrolama

Na slici je prikazana kontrola *Label*, kontrola za prikaz sadržaja, i ona u sebi sadrži 1 kontrolu za vizuelni prikaz sadržaja, a to je kontrola *StackPanel*, koja u sebi sadrži 2 kontrole – *Image* i *AccessText*.

Layout

- Kontrola koja je jako bitna za upotrebljivost aplikacije zato što nam omogućava prilagodljivost dizajna ekranima različite rezolucije
- Koristi se za grupisanje i raspoređivanje GUI elemenata
- Najpopularnije kontrole su *GridPanel*, *StackPanel*, *DockPanel*, *WrapPanel i CanvasPanel*
- Praksa:
  - Izbegavati fiksiranje elemenata – koristiti atribut *Alignment* u kombinaciji sa atributom *Margine* za pozicioniranje elemenata na panel
  - Izbegavati fiksiranje veličine elemenata – atribute *Weight* i *Height* postaviti na vrednost *Auto* kada god je to moguće, ova vrednost označava da veličina nekog dugmeta zavisi od ostalih elemenata koji se prikazuju i od rezolucije ekrana
  - *CanvasPanel* koristiti samo za vektorsku grafiku, a ne i za postavljanje ostalih elemenata
  - *StackPanel* koristiti za postavljanje dugmića dijaloga
  - *GridPanel* koristiti za kreiranje statičkog obrasca za unos podataka. Napraviti kolonu promenljive veličine za labele i još jednu kolonu za prikaz prostora za unos podataka

Layout – stack panel

- Svoje elemente postavlja jedan ispod drugog ili jedan pored drugog u zavisnosti od njegove orijentacije koju podešavate pomoću atributa *Orientation*
- Sve WPF kontrole kao što su *ComboBox*, *ListBox ili Menu* ga koriste kao svoju unutrašnju ploču za izgled
- Pogodan je za pravljenje listi

```
<StackPanel>
  <TextBlock Margin="10" FontSize="20">How do you like your coffee?</TextBlock>
  <Button Margin="10">Black</Button>
  <Button Margin="10">With milk</Button>
  <Button Margin="10">Latte machiato</Button>
  <Button Margin="10">Chappuchino</Button>
</StackPanel>
```

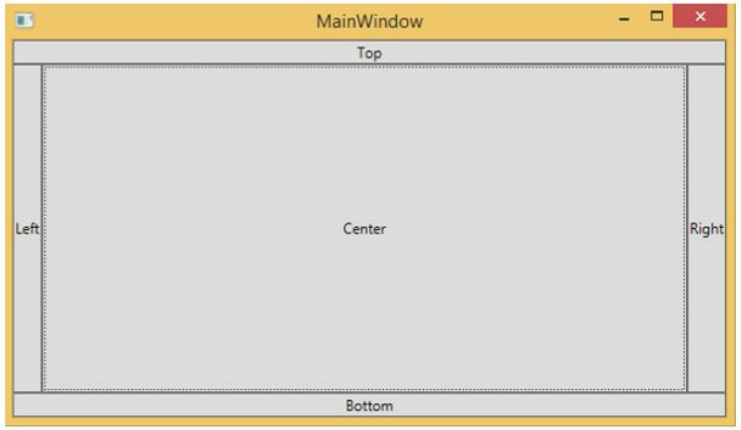
Kao što vidite na slici, za StackPanel je podrazumevana vertikalna orijentacija.

Primer *StackPanelExample*

Layout – dock panel

- Definiše prostor za raspoređivanje podređenih elemenata međusobno, vodoravno ili vertikalno
- Omogućava lako postavljanje elemenata levo, desno, na vrh, na dno ili u sredinu ploče upotrebom atributa *Dock*
- Element koji hoćete da postavite na sredinu ploče treba da bude poslednji u listi podređenih elemenata i on ispunjava slobodan prostor

```
<Grid>
   <DockPanel LastChildFill = "True">
      <Button Content = "Top" DockPanel.Dock = "Top" Click = "Click_Me" />
      <Button Content = "Bottom" DockPanel.Dock = "Bottom" Click = "Click_Me" />
      <Button Content = "Left" Click = "Click_Me" />
      <Button Content = "Right" DockPanel.Dock = "Right" Click = "Click_Me" />
      <Button Content = "Center" Click = "Click_Me" />
   </DockPanel>
</Grid>
```

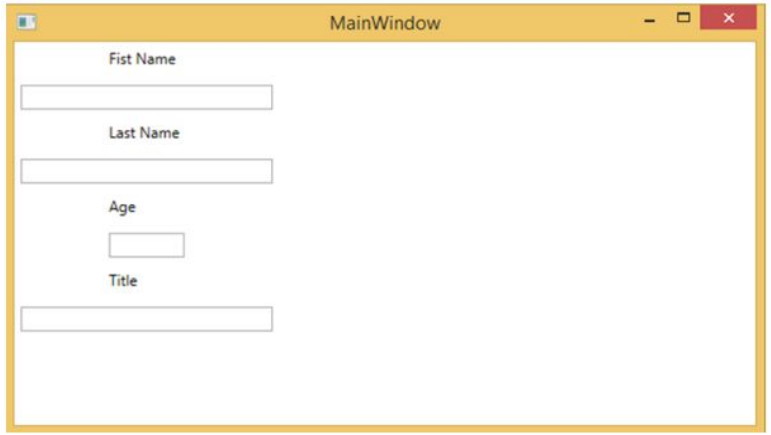


Primer *DockPanelExample*

Layout – wrap panel

- Kreira prostor u kojem se elementi raspoređuju sekvencijalno sa leva na desno ili od vrha ka dnu u zavisnosti od orijentacije koja može da bude horizontalna ili vertikalna i definiše se upotrebom atributa *Orientation*
- Za razliku od *StackPanel-a,* ne raspoređuje sve elemente u jednu liniju, nego ako nema prostora, on elemente prebaci u novu liniju
- Može da se koristi za kreiranje kartica ili stavki menija

```
<Grid>
   <WrapPanel Orientation = "Vertical">
      <TextBlock Text = "Fist Name" Width = "60" Height = "20" Margin = "5" />
      <TextBox  Width = "200" Height = "20" Margin = "5" />
      <TextBlock Text = "Last Name" Width = "60" Height = "20" Margin = "5" />
      <TextBox  Width = "200" Height = "20" Margin = "5"/>
      <TextBlock Text = "Age" Width = "60" Height = "20" Margin = "5" />
      <TextBox  Width = "60" Height = "20" Margin = "5" />
      <TextBlock Text = "Title" Width = "60" Height = "20" Margin = "5" />
      <TextBox  Width = "200" Height = "20" Margin = "5" />
   </WrapPanel>
</Grid>
```

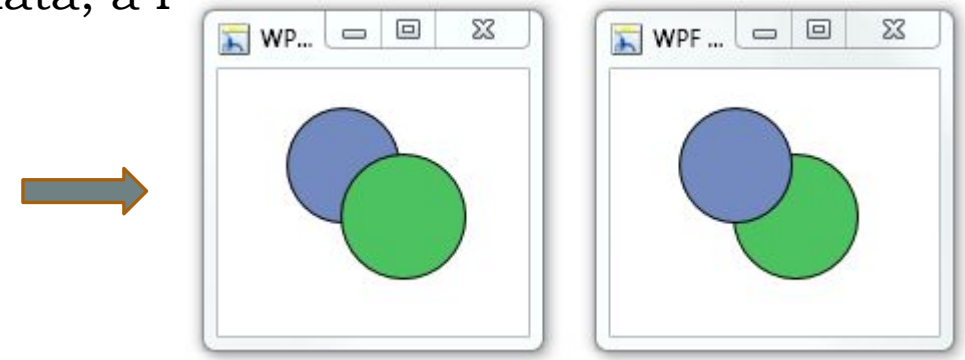


Primer *WrapPanelExample*

Layout – canvas panel

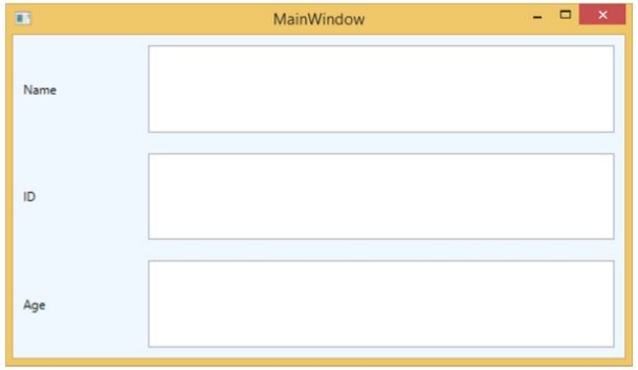
- Najosnovniji panel paketa u WPF-u podređene elemente raspoređuje na osnovu eksplicitno zadatih koordinata. Koordinate se mogu odrediti u odnosu na bilo koju stranu panela upotrebom atributa *Canvas.Left*, *Canvas.Right*, *Canvas.Top* i *Canvas.Bottom*.
- Obično se koristi za grupisanje 2D grafičkih elemenata, a r
elemenata korisničkog interfejsa.

```
<Canvas>
    <Ellipse Fill="Green" Width="60" Height="60" Canvas.Left="30" Canvas.Top="20"
             Canvas.ZIndex="1"/>
    <Ellipse Fill="Blue"  Width="60" Height="60" Canvas.Left="60" Canvas.Top="40"/>
</Canvas>
```



Z-osa koordinatnog sistema je obično određena redosledom navođenja elemenata unutar panela, ali to možete promeniti eksplicitnim navođenjem atributa *Canvas.ZIndex*

Layout – grid panel



- *GridPanel* je prostor koji svoje elemente raspoređuje u tabelarnu strukturu redova i kolona. Funkcionalnost mu je slična tabeli u HTML-u, ali je fleksibilniji. Jedna ćelija može da sadrži više kontrola, može da prelazi preko drugih ćelija ili čak da se preklapa sa drugim ćelijama.
- Podrazumevani *GridPanel* se sastoji od 1 reda i 1 kolone. Više redova i kolona možemo da definišemo pomoću atributa *RowDefinitions* i *ColumnDefinitions*
- Elemente možete da postavite u određenu ćeliju upotrebom atributa *Grid.Row* i *Grid.Column*
- Veličina kontrole se definiše atributima *HorizontalAlignment* i *VerticalAlignment*

Primer *GridExample*

Model–View–Presenter

- MVP je obrazac arhitekture koji se sastoji od 3 sloja, a to su:
  - Model: sadrži poslovnu logiku i podatke. Odgovara u skladu sa naredbama koje dobija od Presenter sloja, a nakon procesa rukovanja podacima šalje podatke Presenter sloju ili drugim modelima.
  - View: sloj za prikaz podataka korisniku.
  - Presenter: glavni deo sistema, povezuje View i Model, tako što prima zahteve od View sloja i prosleđuje ih u Model, a zatim dobijene podatke od Modela vraća u View. Omogućava korisniku slanje zahteva.
- Glavni cilj MVP je odvajanje poslovne logike od korisničkog interfejsa.
- Za razliku od MVC obrasca, MVP kontroliše stranicu.