

GUI u C# - Termin 2

Objektno orijentisantisane tehnologije

Sadržaj

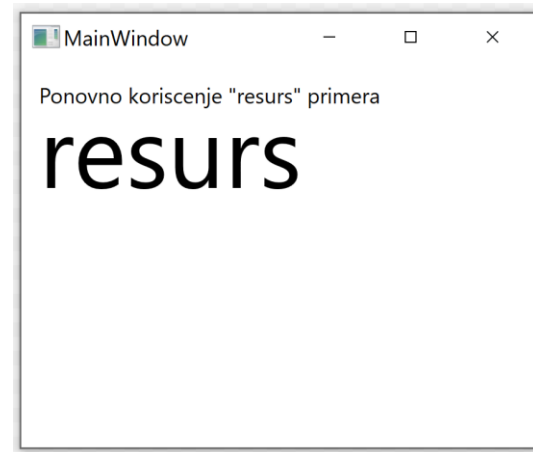
1. Resursi
2. Vrste resursa
3. Primer - Resursi
4. Data binding
5. Primer - Data binding
6. Data context
7. Primer - Data binding u klasi Student
8. TabControl
9. Primer - TabControl
10. ItemsControl i ListBoxControl
11. Primer - Prikaz liste
12. Zadaci

Resursi

- WPF pruža mogućnost čuvanja podataka u vidu resursa
- Resursi se mogu čuvati lokalno za neki kontrolni element, lokalno za neki prozor ili globalno za celu aplikaciju
- Podaci mogu biti konkretna informacija ili hijerarhija WPF kontrolnih elemenata
- Ovo omogućava čuvanje podataka na jednom mestu i korišćenje na nekoliko drugih mesta
- Ovaj koncept se koristi dosta za stilove i šablone koji utiču na izgled aplikacije

Resursi

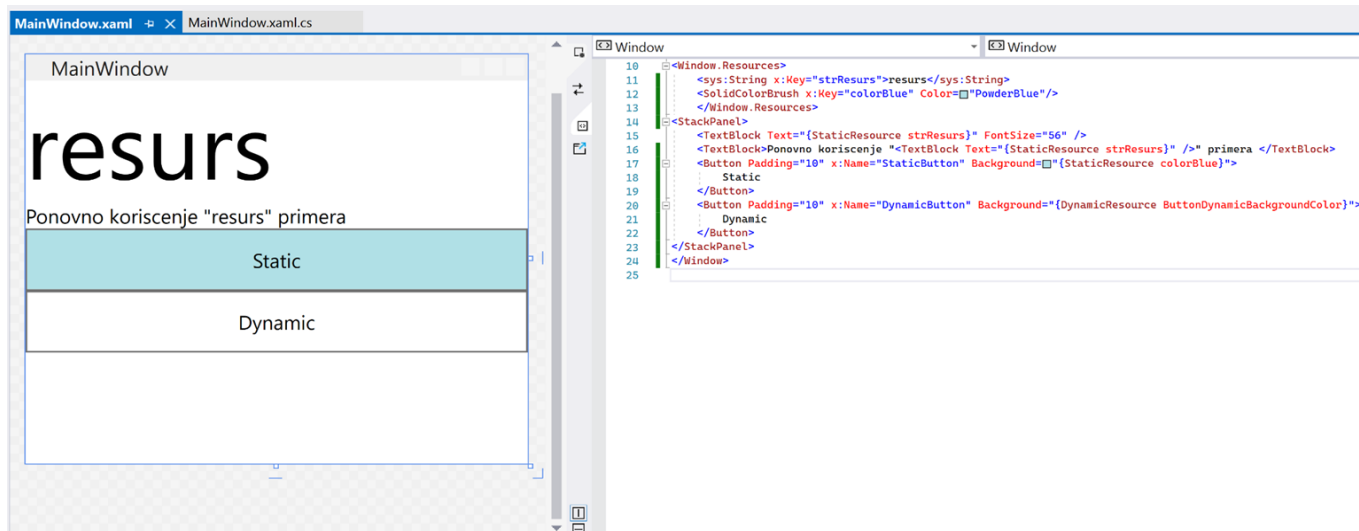
```
<Window x:Class="WpfResursi.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:sys="clr-namespace:System;assembly=mscorlib"
  xmlns:local="clr-namespace:WpfResursi"
  mc:Ignorable="d"
  Title="MainWindow" Height="250" Width="300">
  <Window.Resources>
    <sys:String x:Key="strResurs">resurs</sys:String>
  </Window.Resources>
  <Grid Margin="10 10 0 0">
    <TextBlock Text="{StaticResource strResurs}" FontSize="56" />
    <TextBlock>Ponovno koriscenje "<TextBlock Text="{StaticResource strResurs}" />" primera </TextBlock>
  </Grid>
</Window>
```



- Resursima se dodeljuje ključ pomoću atributa `x:Key`. Na taj način možemo da ih koristimo iz drugih delova aplikacije
- Resursu se pristupa tako što se napiše oznaka *StaticResource* i ključ resursa

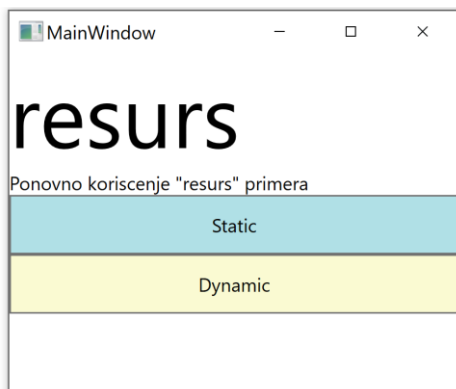
Vrste resursa

- Resursi mogu biti statički i dinamički
 - statički se alociraju samo u vremenu kompajliranja
 - dinamički se mogu menjati putem koda i u toku rada programa



Vrste resursa

```
namespace WpfResursi
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 references
    public partial class MainWindow : Window
    {
        0 references
        public MainWindow()
        {
            InitializeComponent();
            this.Resources["ButtonDynamicBackgroundColor"] = new SolidColorBrush(Colors.LightGoldenrodYellow);
        }
    }
}
```



Primer1_Resursi

Data binding

- Data binding u WPF-u omogućava jednostavan način da se u aplikaciji prikazuju i razmenjuju podaci
- Elementi se mogu vezati za podatke iz različitih izvora (C# objekata ili XAML objekata)
- Data binding uspostavlja vezu između korisničkog interfejsa aplikacije i podataka koji se prikazuju
 - Kada se podaci (u kodu) promene, promeniće se i prikaz (u aplikaciji)
 - Kada se podaci preko prikaza u aplikaciji promene, promeniće se i podaci u kodu
 - Na primer ako korisnik menja vrednost u TextBox elementu, podaci koji su povezani sa vrednošću se takođe automatski menjaju

Data binding

- Svako povezivanje (binding) ima sledeće delove: *target*, *target property*, *source object*, *source object* putanju ka vrednosti
- Na primer u slučaju da imamo klasu Student sa poljem broj indeksa i da želimo da povežemo broj indeksa sa tekstom u TextBox-u, delovi su:
 - target: TextBox
 - target property: Text
 - source object: Student
 - source object value path: BrojIndeksa

Data binding

- Ako želimo da povežemo svojstvo sa drugim svojstvom u kontekstu podataka, sintaksa za binding bi izgledala ovako:

```
{Binding Path=NameOfProperty}
```

- Putanja beleži svojstvo za koje želite da se povežete, međutim, pošto je putanja podrazumevano svojstvo vezivanja ona može da se izostavi

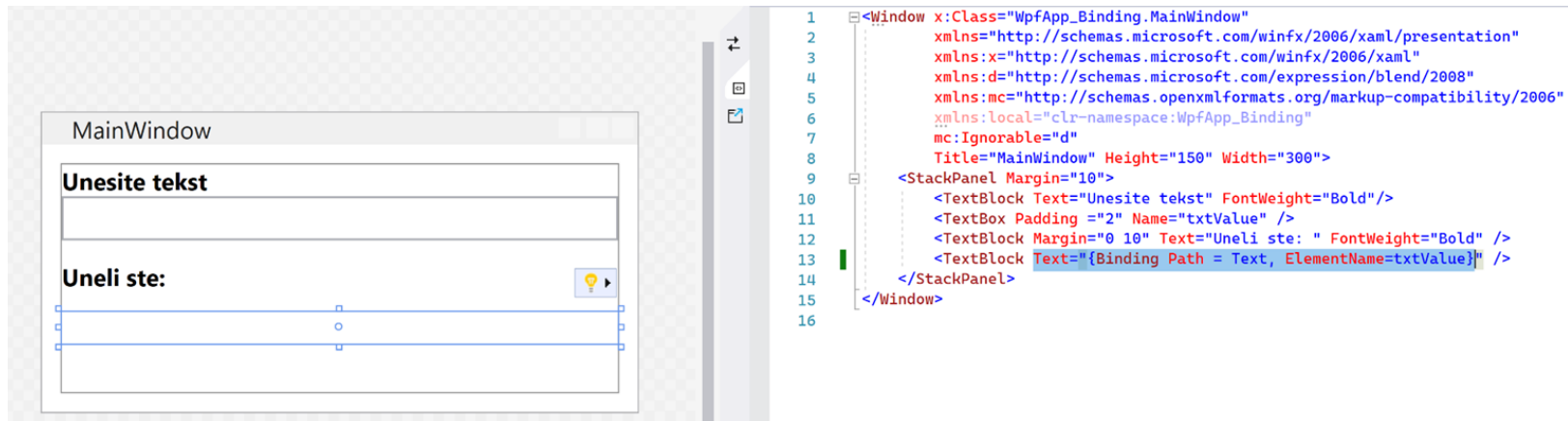
```
{Binding NameOfProperty}
```

- Binding ima mnoga druga svojstva, npr ElementName koje nam omogućava da se direktno povežemo sa drugim elementom kao izvorom. Svako svojstvo koje smo postavili u vezivanju je odvojeno zarezom:

```
{Binding Path =Text, ElementName=txtValue}
```

Data binding - jednostavan primer

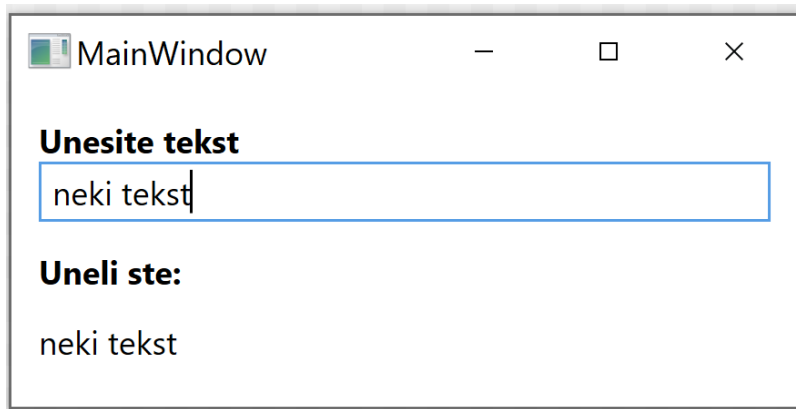
- U narednom primeru smo povezali tekst u TextBlock elementu sa sadržajem iz TextBox elementa



```
1 <Window x:Class="WpfApp_Binding.MainWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5       xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6       xmlns:local="clr-namespace:WpfApp_Binding"
7       mc:Ignorable="d"
8       Title="MainWindow" Height="150" Width="300">
9     <StackPanel Margin="10">
10       <TextBlock Text="Unesite tekst" FontWeight="Bold"/>
11       <TextBox Padding="2" Name="txtValue" />
12       <TextBlock Margin="0 10" Text="Uneli ste: " FontWeight="Bold" />
13       <TextBlock Text="{Binding Path = Text, ElementName=txtValue}" />
14     </StackPanel>
15 </Window>
```

Data binding - jednostavan primer

- TextBox se automatski ažurira kada unesemo tekst u TextBox
- Kada ne bismo koristili DataBinding, morali bi da slušamo događaj na TextBox-u, a onda ažuriramo TextBox svaki put kad se tekst promeni



Primer2_Binding

DataContext

- Kada je vezivanje podataka deklarirano na XAML elementima, oni ga rešavaju gledajući svoje neposredno svojstvo *DataContext*
- Kontekst podataka je obično izvorni objekat vezivanja za procenu putanje vrednosti izvora vezivanja. Možemo zaobići ovo ponašanje u vezivanju i postaviti određenu vrednost izvornog objekta vezivanja
- Ako *DataContext* za objekat u kome se nalazi vezivanje nije podešen, proverava se *DataContext* roditeljskog elementa, i tako dalje, sve do korena stabla XAML objekata. Ukratko, kontekst podataka koji se koristi za rešavanje vezivanja nasleđuje se od roditelja osim ako nije eksplicitno postavljen direktno na objektu

DataContext

- Dakle, izvori vezivanja su vezani za aktivni *DataContext* za element. Elementi automatski nasleđuju svoj *DataContext* ako ga nisu eksplicitno definisali
- Korišćenje svojstva *DataContext* je kao postavljanje osnove svih vezivanja kroz hijerarhiju kontrola
- Međutim, to ne znači da moramo da koristimo isti *DataContext* za sve kontrole unutar prozora. Pošto svaka kontrola ima svoj *DataContext*, možemo lako prekinuti lanac nasleđivanja i zameniti *DataContext* novom vrednošću. Tako se može postaviti globalni *DataContext* na window-u, a zatim lokalni i specifičniji *DataContext* na nekom panelu npr

Data binding kroz kod

- Definisanje bindinga može da se uradi i iz koda iza XML fajla
- U nastavku je prikazan primer od malo pre, ali je binding urađen iz koda

```
<Window x:Class="WpfApp_Binding.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApp_Binding"
        mc:Ignorable="d"
        Title="MainWindow" Height="150" Width="300">
    <StackPanel Margin="10">
        <TextBlock Text="Unesite tekst" FontWeight="Bold"/>
        <TextBox Padding="2" Name="txtValue" />
        <TextBlock Margin="0 10" Text="Uneli ste: " FontWeight="Bold" />
        <TextBlock Name="lblValue" />
    </StackPanel>
</Window>
```

```
namespace WpfApp_Binding
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 references
    public partial class MainWindow : Window
    {
        0 references
        public MainWindow()
        {
            InitializeComponent();
            Binding binding = new Binding("Text");
            binding.Source = txtValue;
            lblValue.SetBinding(TextBlock.TextProperty, binding);
        }
    }
}
```

Data binding kroz kod

- Najpre kreiramo instancu klase Binding tako što u konstruktoru prosledimo putanju do svojstva sa kojim želimo da se povežemo (u ovom slučaju to je Text)
- Zatim navodimo izvor koji vezujemo za kreirani objekat i koji je u ovom primeru TextBox (*txtValue*) - time smo naglasili da ćemo za izvornu kontrolu koristiti dati TextBox i njegovo svojstvo (property) Text
- Na kraju koristimo metod SetBinding da kombinujemo naš Binding objekat sa odredišnom kontrolom, u ovom slučaju TextBlock (*lbValue*). Metoda SetBinding() ima dva parametra, jedan koji sadrži svojstvo zavisnosti u odredišnoj kontroli, a drugi koji je objekat vezivanja koji želimo da koristimo

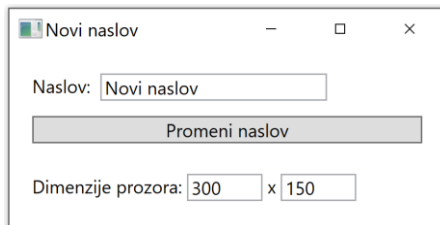
UpdateSourceTrigger svojstvo

- Promene u odredišnom kontrolnom elementu ne moraju odmah biti reflektovane na izvoru
- Ovo ponašanje kontroliše svojstvo u binding-u pod nazivom *UpdateSourceTrigger*. Podrazumevana je vrednost *Default*, što znači da se izvor ažurira na osnovu svojstva za koje se vezujete. U trenutku pisanja, sva svojstva osim svojstva *Text*, ažuriraju se čim se svojstvo promeni (*PropertyChanged*), dok se svojstvo *Text* ažurira kada se izgubi fokus na odredišnom elementu (*LostFocus*)
- Pored *Default*, *PropertyChanged* i *LostFocus*, postoji i opcija *Explicit* koja znači da ažuriranje mora da se progura ručno da bi se dogodilo, koristeći poziv *UpdateSource()* na binding objektu u kodu

UpdateSourceTrigger svojstvo

```
<Window x:Class="WpfApp_Binding.Binding"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApp_Binding"
mc:Ignorable="d"
Title="Binding" Height="150" Width="300">
<StackPanel Margin="15">
<WrapPanel>
<TextBlock Text="Naslov: " />
<TextBox Name="txtNaslov" Text="{Binding Title, UpdateSourceTrigger=Default}" Width="150" />
<!--TextBox Name="txtNaslov" Text="{Binding Title, UpdateSourceTrigger=Explicit}" Width="150" /-->
</WrapPanel>
<Button Name="btnPromeni" Click="btnPromeni_Click" Margin="0 10" Padding="5,0">Promeni naslov</Button>
<WrapPanel Margin="0,10,0,0">
<TextBlock Text="Dimenzije prozora: " />
<TextBox Text="{Binding Width, UpdateSourceTrigger=LostFocus}" Width="50" />
<TextBlock Text=" x " />
<TextBox Text="{Binding Height, UpdateSourceTrigger=PropertyChanged}" Width="50" />
</WrapPanel>
</StackPanel>
</Window>
```

```
namespace WpfApp_Binding
{
    /// <summary>
    /// Interaction logic for Binding.xaml
    /// </summary>
    2 references
    public partial class Binding : Window
    {
        0 references
        public Binding()
        {
            InitializeComponent();
            this.DataContext = this;
        }
        1 reference
        private void btnPromeni_Click(object sender, RoutedEventArgs e)
        {
            BindingExpression binding = txtNaslov.GetBindingExpression(TextBox.TextProperty);
            binding.UpdateSource();
        }
    }
}
```



Primer2_Binding

Data binding sa korisnički definisanim klasama

- Da bi korisnički definisane klase reagovala na promene na korisničkom interfejsu potrebno je koristiti ***ObservableCollection*** klasu i ***INotifyPropertyChanged*** interfejs
- Npr ako imamo listu objekata koju prikazujemo na korisničkom interfejsu i ako u njemu imamo dodavanje i brisanje iz liste koju prikazujemo, promene neće biti prikazane na korisničkom interfejsu, takođe izmene podatka neće biti sačuvane
 - Da bi promene u sastavu elemenata kolekcije bile prikazane potrebno je da umesto liste koristimo objekat *ObservableCollection* klase
 - Da bi promene u objektima (elementima) bile prikazane potrebno je da objekat koji čuvamo implementira interfejs *INotifyPropertyChanged*

Data binding sa korisnički definisanim klasama

- Objekat klase List (promene neće biti prikazane)

2 references

```
public partial class MainWindow : Window
{
    private List<Student> studenti = new List<Student>();

    0 references
    public MainWindow()
    {
        InitializeComponent();

        studenti.Add(new Student("PR 12/2010"));
        studenti.Add(new Student("PR 20/2010"));
        studenti.Add(new Student("PR 14/2010"));

        lbStudenti.ItemsSource = studenti;
    }
}
```

- Objekat klase ObservableCollection (promene će biti prikazane)

```
public partial class MainWindow : Window
{
```

```
    private ObservableCollection<Student> studenti =
        new ObservableCollection<Student>();
```

0 references

```
public MainWindow()
{
    InitializeComponent();

    studenti.Add(new Student("PR 12/2010"));
    studenti.Add(new Student("PR 20/2010"));
    studenti.Add(new Student("PR 14/2010"));

    lbStudenti.ItemsSource = studenti;
}
```

Data binding sa korisnički definisanim klasama

```
internal class Student: INotifyPropertyChanged
{
    private string brojIndeksa;

    public event PropertyChangedEventHandler PropertyChanged;

    2 references
    public string BrojIndeksa
    {
        get { return this.brojIndeksa; }
        set
        {
            if (this.brojIndeksa != value)
            {
                this.brojIndeksa = value;
                this.NotifyPropertyChanged("BrojIndeksa");
            }
        }
    }

    4 references
    public Student(string brojIndeksa)
    {
        this.brojIndeksa = brojIndeksa;
    }

    1 reference
    private void NotifyPropertyChanged(string v)
    {
        if (this.PropertyChanged != null)
            this.PropertyChanged(this, new PropertyChangedEventArgs(v));
    }
}
```

```
<Window x:Class="Wpf_BindingStudent.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Wpf_BindingStudent"
        mc:Ignorable="d"
        Title="MainWindow" Height="200" Width="350">
    <Grid Margin="10">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"></ColumnDefinition>
            <ColumnDefinition Width="*"></ColumnDefinition>
            <ColumnDefinition Width="*"></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Column="2">
            <Button Name="btnDodaj" Click="btnDodaj_Click">Dodaj</Button>
            <Button Name="btnIzmeni" Click="btnIzmeni_Click" Margin="0,5">Izmeni</Button>
            <Button Name="btnObrisi" Click="btnObrisi_Click">Obrisi</Button>
        </StackPanel>
        <ListBox Grid.Column="0" Name="lbStudenti" DisplayMemberPath="BrojIndeksa"
            MouseDoubleClick="lbStudenti_MouseDoubleClick"></ListBox>
        <TextBox Grid.Column="1" Name="txtStudenti"></TextBox>
    </Grid>
</Window>
```

TabControl kontrolni element

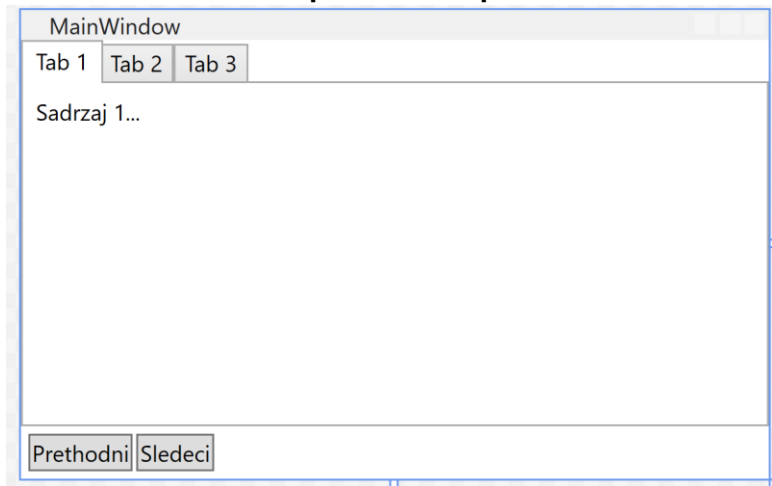
- WPF TabControl omogućava da korisnički interfejs podelimo na više delova kojima se može pristupiti klikom na zaglavlje kartice, koje se obično nalazi na vrhu
- Svaka kartica je predstavljena elementom TabItem, gde tekst prikazan na njoj kontroliše svojstvo *Header*
- Element TabItem dolazi iz klase ContentControl, što znači da možemo definisati jedan element unutar njega koji će biti prikazan ako je kartica aktivna
- WPF je veoma fleksibilan kada želimo da prilagodimo izgled kartica. Sadržaj se može prikazati kako god želimo, ali isto tako mogu i zaglavlja kartica - svojstvo Header se može popuniti čime god želimo (npr može se dodati slika kao ikonica...)

TabControl primer

```
<TabControl>
  <TabItem>
    <TabItem.Header>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="Blue" Foreground=■"Blue" />
        <!--Image-->
        <!--Drugi elementi-->
      </StackPanel>
    </TabItem.Header>
  </TabItem>
</TabControl>
```

TabControl - upravljanje

- Dugmići u donjem delu programa koriste svojstvo *SelectedIndex* da bi odredili gde se trenutno nalazimo, a zatim ili oduzimaju ili dodaju jedan toj vrednosti, pazeći da novi indeks ne padne ispod ili iznad broja dostupnih stavki



TabControl - upravljanje

```
<Window x:Class="Zadatak1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Zadatak1"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="400">
    <Grid>
        <DockPanel>
            <StackPanel Orientation="Horizontal" DockPanel.Dock="Bottom" Margin="3">
                <Button Name="btnPrethodni" Click="btnPrethodni_Click" Margin="1">Prethodni</Button>
                <Button Name="btnSledeci" Click="btnSledeci_Click" Margin="1">Sledeci</Button>
            </StackPanel>
            <TabControl Name="tabC">
                <TabItem Header="Tab 1">
                    <Label Content="Sadrzaj 1..." />
                </TabItem>
                <TabItem Header="Tab 2">
                    <Label Content="Sadrzaj 2..." />
                </TabItem>
                <TabItem Header="Tab 3">
                    <Label Content="Sadrzaj 3..." />
                </TabItem>
            </TabControl>
        </DockPanel>
    </Grid>
</Window>
```

```
public partial class MainWindow : Window
{
    0 references
    public MainWindow()
    {
        InitializeComponent();
    }

    1 reference
    private void btnPrethodni_Click(object sender, RoutedEventArgs e)
    {
        int newIndex = tabC.SelectedIndex - 1;
        if (newIndex < 0)
            newIndex = tabC.Items.Count - 1;
        tabC.SelectedIndex = newIndex;
    }

    1 reference
    private void btnSledeci_Click(object sender, RoutedEventArgs e)
    {
        int newIndex = tabC.SelectedIndex + 1;
        if (newIndex >= tabC.Items.Count)
            newIndex = 0;
        tabC.SelectedIndex = newIndex;
    }
}
```

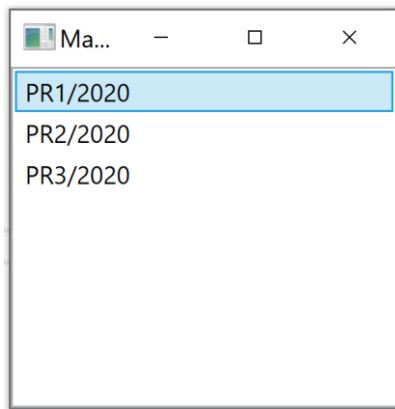

ItemsControl i ListBox

- ItemsControl je najjednostavniji način da se prikaže lista
- Nema mogućnost odabira prikazanih elemenata
- Ako želimo da koristimo Data Binding potrebno je da upotrebimo u okviru `ItemsControl.ItemsTemplate` `DataTemplate` tag
- Da bi povezali kod sa prikazom, potrebno je da pristupimo `ItemSource` svojstvu `ItemsControl`a u kodu i da njemu dodelimo listu elemenata
- `ListBox` je unapređena verzija `ItemControl`-a koja ima mogućnost selekcije elemenata

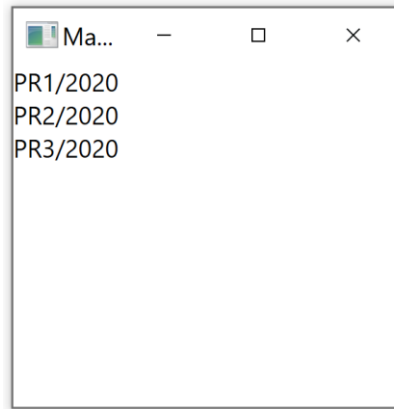
ListBox

ItemsControl

```
<ListBox Name="icList">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding BrojIndeksa}" />
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```



```
<ItemsControl Name="icList">
  <ItemsControl.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding BrojIndeksa}" />
    </DataTemplate>
  </ItemsControl.ItemTemplate>
</ItemsControl>
```



ListBox i ItemsControl

```
internal class Student
{
    private string brojIndeksa;

    0 references
    public string BrojIndeksa
    {
        get { return this.brojIndeksa; }
        set
        {
            if (this.brojIndeksa != value)
            {
                this.brojIndeksa = value;
            }
        }
    }

    3 references
    public Student(string brojIndeksa)
    {
        this.brojIndeksa = brojIndeksa;
    }
}
```

```
public MainWindow()
{
    InitializeComponent();

    List<Student> studenti = new List<Student>();
    studenti.Add(new Student("PR1/2020"));
    studenti.Add(new Student("PR2/2020"));
    studenti.Add(new Student("PR3/2020"));

    icList.ItemsSource = studenti;
}
```

ListBox - selekcija elemenata

- SelectedItem svojstvo koje se nalazi u ListBox-u nam omogućava da proverimo koji element je trenutno selektovan
- SelectedIndex svojstvo koje se nalazi u ListBox-u nam omogućava da proverimo koji indeks elementa je trenutno selektovan
- ListBox ima SelectionMode atribut koji brine o tome da li je moguće selektovati više elemenata istovremeno (npr može biti One, Extended...)
- ListBox ima SelectionChanged atribut koji može da se veže za događaj koji opisuje ponašanje programa ako korisnik promeni selektovan element

ListBox - selekcija elemenata

- Ako želimo da selektujemo sve elemente u listi, koristimo sledeći kod:

```
foreach(object o in icList.Items)
    icList.SelectedItems.Add(o);
```

- Ako želimo da reagujemo na promenu selekcije, koristimo kod:

```
if(icList.SelectedItem != null)
    this.BrojIndeksa = (icList.SelectedItem as Student).BrojIndeksa;
```

- Ako želimo da selektujemo sledeći element od trenutnog, pišemo:

```
int nextIndex = 0;
if((icList.SelectedIndex >= 0) && (icList.SelectedIndex < (icList.Items.Count - 1)))
    nextIndex = icList.SelectedIndex + 1;
icList.SelectedIndex = nextIndex;
```

Zadatak 1

- Napisati klasu Student koja implementira interfejs *INotifyPropertyChanged* i koja ima polja ime, prezime i broj indeksa (tipa string). Za datu klasu napisati konstruktor bez parametara, konstruktor sa parametrima i svojstvo za polja.
- U okviru klase MainWindow kreirati polje studenti (tipa ObservableCollection<Student>) koje predstavlja listu studenata sa kojom ćemo raditi.
- Napraviti GUI tako da ima 3 taba gde je
 - prvi tab “Pregledaj sve” tab koji prikazuje sve studente iz liste (koristiti DataGridView kontrolni element)
 - drugi tab “Dodaj novog” tab koji omogućava dodavanje novog studenta u listu
 - treći tab “Izmeni” tab koji omogućava prikaz liste brojeva indeksa (koja može da se selektuje) svih studenata u listi i brisanje i izmenu selektovanog elementa (koristiti ListBox za prikaz)

Zadatak 1

MainWindow

Pregledaj sve Dodaj novog Izmeni

Ime	Prezime	Broj indeksa
Pera	Peric	PR1/2020
Mira	Miric	PR2/2020
Zika	Zikic	PR3/2020

MainWindow

Pregledaj sve Dodaj novog Izmeni

Ime :

Prezime :

Broj indeksa :

Zadatak 1

MainWindow

Pregledaj sve Dodaj novog Izmeni

PR1/2020
PR2/2020
PR3/2020

Ime :

Prezime :

Broj indeksa :

Izmeni Obrisi

MainWindow

Pregledaj sve Dodaj novog Izmeni

PR1/2020
PR2/2020
PR3/2020

Ime :

Prezime :

Broj indeksa :

Izmeni Obrisi

Zadatak 2

- Napisati klasu Stavka koja implementira interfejs *INotifyPropertyChanged* i koja ima id (int), naziv (string), cena (double), opis (string) i vrsta (nabrojivog tipa Vrsta koji ima vrednosti HRANA, PICE, OSTALO). Za datu klasu napisati konstruktor bez parametara, konstruktor sa parametrima i proprietije za polja
- U okviru klase MainWindow kreirati polje jelovnik (tipa ObservableCollection<Stavka>) koje predstavlja listu stavki sa kojom ćemo raditi
- Napraviti GUI tako da ima 3 taba gde je:
- prvi tab “Pregledaj sve” tab koji prikazuje sve stavke iz liste (koristiti DataGridView kontrolni element) i koji ima implementiranu pretragu elemenata po nazivu i opisu

Zadatak 2

Pretragu treba uraditi tako da dok se kuca pojam u polje za pretragu u prikazu stavki vidimo rezultate koji počinju sa unesenim pojmom. Potrebno je razlikovati velika i mala slova.

Implementirati i dugme “Pretraga” koje prikazuje sve stavke koje sadrže ukucani pojam.

Velika i mala slova ne treba razlikovati

- drugi tab “Dodaj novog” tab koji omogućava dodavanje nove stavke u listu. Dodavanje je moguće samo ako u listi već ne postoji stavka sa istim id-em ili nazivom. Dodavanje nije moguće ukoliko se na mestima gde se očekuje broj prosledi nešto drugo ili ako su id, naziv i cena prazni. Potrebno je ispisati poruku o uspehu operacije u vidu MessageBox-a

Zadatak 2

- treći tab “Izmeni” tab koji omogućava prikaz svih stavki (prikazati id, naziv i cenu) iz liste i brisanje i izmenu selektovanog elementa (koristiti ListBox za prikaz). Izmena mora da zadovolji sve ono što je bilo potrebno i za dodavanje stavke, a id nije moguće izmeniti. Potrebno je ispisati poruku o uspehu operacije. Brisanje je moguće samo ukoliko je element selektovan

Zadatak 2

MainWindow

Pregledaj sve Dodaj Izmeni

Pretrazi po nazivu

Pretrazi

Id	Naziv	Cena	Vrsta	Opis
1	Giros	360	HRANA	Svinjski mali
2	Burito	460	HRANA	Vege
3	Coca Cola	120	PICE	Gazirano
4	Sok od pomorandze	300	PICE	Cedjeni

MainWindow

Pregledaj sve Dodaj Izmeni

1 Giros 360

2 Burito 460

3 Coca Cola 120

4 Sok od pomorandze 300

Id :

Naziv :

Cena :

Vrsta :

Opis :

Dodaj

Zadatak 2

MainWindow

Pregledaj sve Dodaj Izmeni

- 1 Giros 360
- 2 Burito 460
- 3 Coca Cola 120
- 4 Sok od pomorandze 300

Id :

Naziv :

Cena :

Vrsta :

Opis :

Izmeni Obrisi

MainWindow

Pregledaj sve Dodaj Izmeni

- 1 Giros 360
- 2 Burito 460
- 3 Coca Cola 120
- 4 Sok od pomorandze 300

Id :

Naziv :

Cena :

Vrsta :

Opis :

Izmeni Obrisi