

GUI u C# - Termin 3

Objektno orijentisantisane tehnologije

Sadržaj

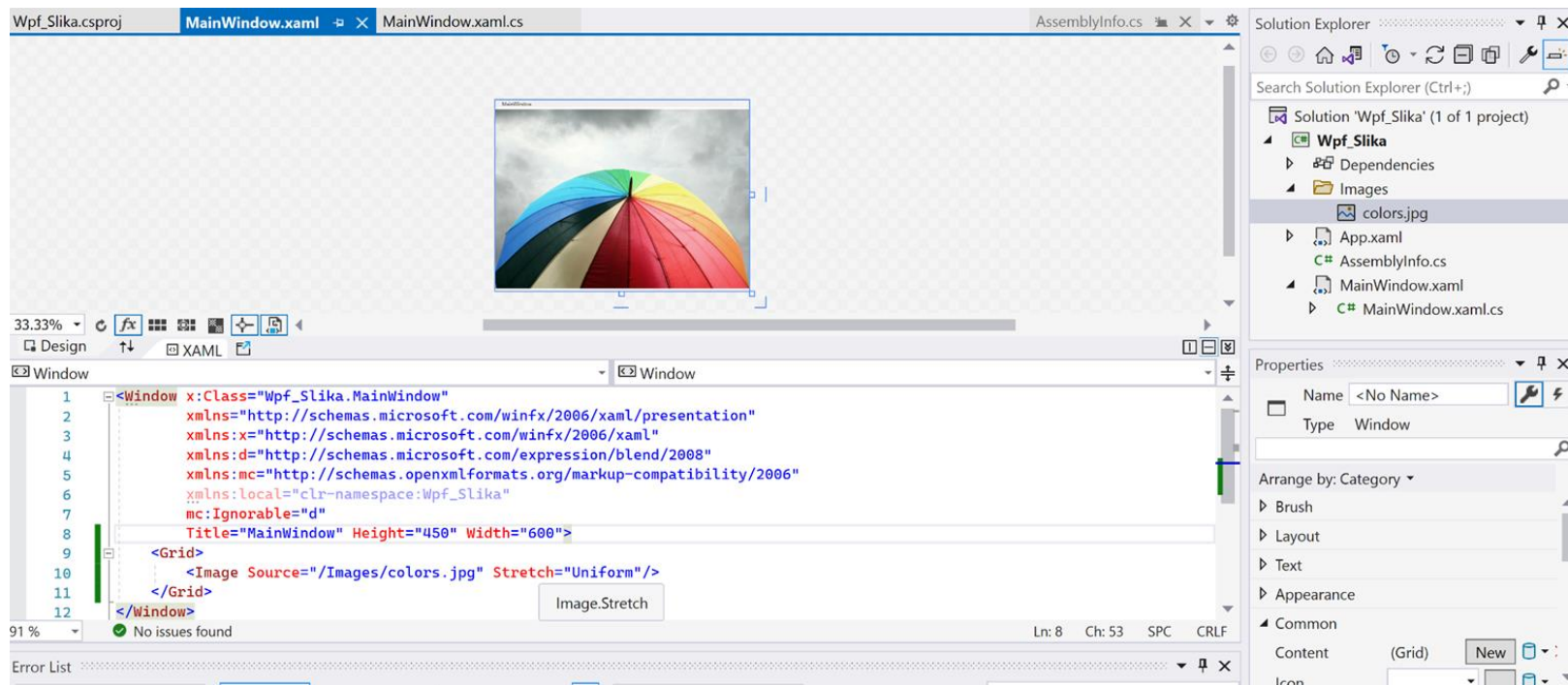
1. Image
2. Primer - Image
3. Meni
4. Primer - MeniKomande
5. Komande
6. Primer - MeniKomande
7. Primer - NovaKomanda
8. ListView
9. Primer korišćenja DataGrid-a kao tabele
10. TreeView
11. Slider
12. Zadaci

Kontrolni element za sliku - Image

- Postoji nekoliko načina da se slika koristi u aplikaciji
- Najjednostavniji način je da se u Source svojstvu u Image kontrolnom elementu navede putanja do slike (relativna ili apsolutna)

```
<Window x:Class="Wpf_Slika.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_Slika"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="600">
    <Grid>
        <Image Source="/Images/colors.jpg" Stretch="Uniform"/>
    </Grid>
</Window>
```

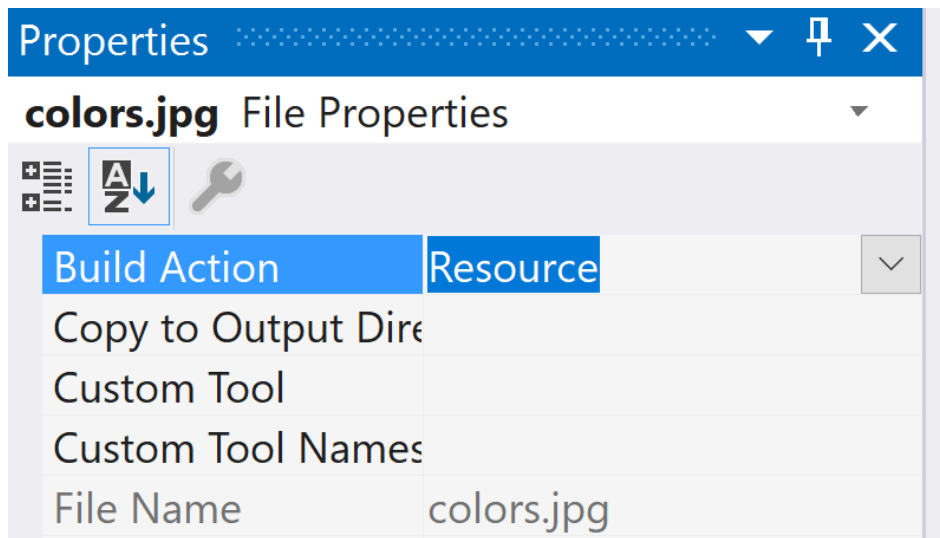
Kontrolni element za sliku - Image



Primer1_Slika

Kontrolni element za sliku - Image

- Napomena: Kako bi prikaz slike radio, kliknite desni klik na sliku, odaberite Properties i označite Build Action na Resource



Meni

- Jedan od najčešćih delova aplikacije je meni, koji se ponekad naziva i glavni meni jer samo jedan takav obično postoji u aplikaciji
- Meni je praktičan jer nudi mnogo opcija, koristeći samo vrlo malo prostora
- Kontrolni element za kreiranje menija se zove Meni
- Dodavanje stavki u meni se vrši dodavanjem elemenata Menuitem, a svaki Menuitem može imati niz podstavki, što nam omogućava da kreiramo hijerarhijske menije

Meni

```
<Window x:Class="Wpf_Meni.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_Meni"
  mc:Ignorable="d"
  Title="MainWindow" Height="450" Width="800">
  <DockPanel>
    <Menu DockPanel.Dock="Top">
      <MenuItem Header="Opcija 1">
        <MenuItem Header="Opcija 1.1" />
        <MenuItem Header="Opcija 1.2" />
      </MenuItem>
      <MenuItem Header="Opcija 2">
        <MenuItem Header="Opcija 2.1"/>
        <MenuItem Header="Opcija 2.2" IsCheckable="True" IsChecked="True"/>
      </MenuItem>
    </Menu>
    <TextBox AcceptsReturn="True" />
  </DockPanel>
</Window>
```

Meni

- obrađivanje događaja i dodavanje logike se vrši tako što se u Menu kontrolnom elementu doda događaj sa *EventHandlerom* koji će reagovati na njega

```
<MenuItem Header="Opcija 1" Click="MenuItem_Click">  
    <MenuItem Header="Opcija 1.1" Click="MenuItem_Click_1" />  
    <MenuItem Header="Opcija 1.2" />  
</MenuItem>
```

0 references

```
private void MenuItem_Click(object sender, RoutedEventArgs e)  
{  
    MessageBox.Show("Opcija 1");  
}
```

0 references

```
private void MenuItem_Click_1(object sender, RoutedEventArgs e)  
{  
    MessageBox.Show("Opcija 2");  
}
```


Meni

- Češći pristup rukovanjem događaja u meniju od onog ranije opisanog je pristup korišćenjem komandi
- Komande ćemo detaljno objasniti uskoro, za sada ćemo samo pokazati kako se upotrebljavaju u meniju
- Komande obezbeđuju da možemo imati istu radnju na meniju, Toolbar-u, kontekstnom meniju bez potrebe da implementiramo isti kod na više mesta
- Rukovanje prečica na tastaturi je mnogo lakše uz korišćenje komandi

Meni

- Uz korišćenje komandi WPF nam je automatski omogućio neke funkcionalnosti u zavisnosti od aktivne kontrole i njenog stanja (automatski je dodao funkcionalnost komandama i prečice)
- Pošto WPF zna kako da rukuje određenim komandama u kombinaciji sa određenim kontrolama, npr komandama Cut/Copy/Paste u kombinaciji sa kontrolom unosa teksta, ne moramo čak ni da rukujemo njihovim događajima – one rade automatski. Ipak, moramo to da uradimo za komandu New, pošto WPF nema načina da pogodi šta želimo da uradi kada je korisnik aktivira. Ovo se radi pomoću komandnih veza prozora, što ćemo videti kad budemo prelazili komande

Komande

- U modernom korisničkom interfejsu, tipično je da funkcija bude dostupna sa nekoliko mesta i da je pozivaju različite radnje korisnika
- Na primer, ako imamo interfejs sa glavnim menijem i Toolbar-om, radnja kao što je Novo ili Otvori može biti dostupna u meniju, u Toolbar-u, u kontekstnom meniju (npr. kada kliknemo desnim klikom u glavnoj oblasti aplikacije) i sa prečice na tastaturi kao što su Ctrl+N i Ctrl+O
- Svaka od ovih radnji treba da izvrši ono što je isti deo koda, tako da je nepraktično da definišemo događaj za svaku od njih, a zatim da pozovemo zajedničku funkciju. To bi dovelo do najmanje tri obrađivača događaja i koda za rukovanje prečicama na tastaturi

Komande

- Komande se sastoje od interfejsa *ICommand*, koji samo definiše događaj i dve metode: *Execute()* i *CanExecute()*. Prva je za izvođenje stvarne radnje, dok je druga za utvrđivanje da li je radnja trenutno dostupna
- Da bismo izvršili stvarnu radnju komande, potrebna nam je veza između komande i koda i tu se upotrebljava *CommandBinding*
- *CommandBinding* se obično definiše u prozoru ili korisničkoj kontroli i sadrži reference na komandu kojom rukuje, kao i stvarne rukovaoce događajima za rad sa *Execute()* i *CanExecute()* događajima komande

Komande

- Komande nam pomažu da odgovorimo na zajedničku akciju iz nekoliko različitih izvora, koristeći jedan obrađivač događaja. Takođe čine mnogo lakšim omogućavanje i onemogućavanje elemenata korisničkog interfejsa na osnovu trenutne dostupnosti i stanja
- WPF sadrži preko 100 često korišćenih ugrađenih komandi
- Podeljene su u 5 kategorija: `ApplicationCommands`, `NavigationCommands`, `MediaCommands`, `EditingCommands` i `ComponentCommands`
 - Npr `ApplicationCommands` sadrži komande za često korišćene radnje kao što su Otvori, Sačuvaj, Iseci, Kopiraj i Nalepi

Komande

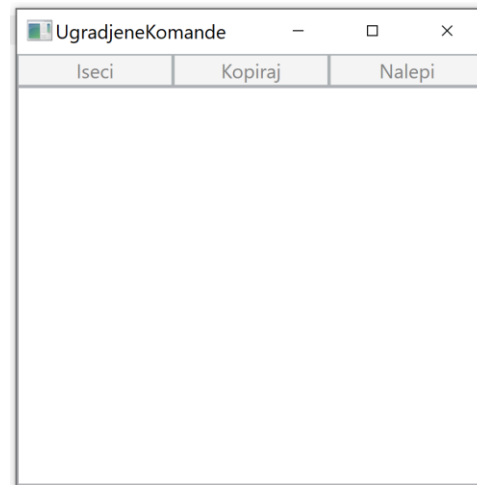
- Primer za komande Iseci, Kopiraj i Nalepi

```
<Window x:Class="Wpf_Meni.UgradjeneKomande"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:local="clr-namespace:Wpf_Meni"
    mc:Ignorable="d"
    Height="300" Width="300"
    Title="UgradjeneKomande">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Button Grid.Column="0" Grid.Row="0" Command="ApplicationCommands.Cut" CommandTarget="{Binding ElementName=txtMain}">Iseci</Button>
        <Button Grid.Column="1" Grid.Row="0" Command="ApplicationCommands.Copy" CommandTarget="{Binding ElementName=txtMain}">Kopiraj</Button>
        <Button Grid.Column="2" Grid.Row="0" Command="ApplicationCommands.Paste" CommandTarget="{Binding ElementName=txtMain}">Nalepi</Button>
        <TextBox Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="3" Name="txtMain" AcceptsReturn="True" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"></TextBox>
    </Grid>
</Window>
```

Komande

- U primeru gde koristimo ugrađene komande ne moramo u kodu koji se nalazi iza xaml fajla ništa da implementiramo, sve se radi u xaml fajlu
- Sve što je potrebno da uradimo jeste da u okviru dugmeta navedemo komandu i binding za tu komandu
- Npr za komandu Iseci:
 - `Command="ApplicationCommands.Cut"`
 - `CommandTarget="{Binding ElementName=txtMain}"`
- Dugmići se sami omogućavaju/onemogućavaju po potrebi



Komande

- Kada želimo da definišemo ponašanje za određenu komandu u XAML fajlu napišemo:

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

- a u fajlu sa kodom iza njega :

```
1 reference  
private void CommandBinding_CanExecute(object sender, CanExecuteRoutedEventArgs e)  
{  
    e.CanExecute = true;  
}
```

```
1 reference  
private void CommandBinding_Executed(object sender, ExecutedRoutedEventArgs e)  
{  
    txtEditor.Text = "";  
}
```


Komande - pravljenje nove komande

- Najlakši način da implementiramo sopstvene komande je da napravimo statičku klasu koja će ih sadržati
- Svaka komanda se zatim dodaje klasi kao statičko polje, što nam omogućava da ih koristimo u svojoj aplikaciji
- U primeru 3 se nalazi kod za novu komandu - Exit

ListView

- ListView kontrola izgleda slično kao ListBox, sve dok ne počnemo da mu dodajemo specijalizovane view-ove
- ListView nasleđuje ListBox kontrolu i delovi su mu ListViewItem
- Pošto ListViewItem potiče od klase ContentControl, možemo navesti drugu WPF kontrolu kao njen sadržaj
- Primer ListView-a u osnovnom obliku:

```
<ListView>  
  <ListViewItem>Element 2</ListViewItem>  
  <ListViewItem IsSelected="True">Element 2</ListViewItem>  
  <ListViewItem>Element 3</ListViewItem>  
</ListView>
```

ListView

- Ukoliko želimo da podatke u ListViewItem-e uvezemo iz koda, radimo slično kao sa ListBoxom. Imenujemo ga i pomoću svojstva ItemSource ga povežemo sa odgovarajućom kolekcijom podataka
- U okviru ListView.ItemTemplate-a možemo da definišemo DataTemplate u kome ćemo navesti izgled koji želimo za elemente
- U okviru ListView.View-a možemo da definišemo GridView koji sadrži kolone GridViewColumn i GridViewColumn.CellTemplate u kome možemo da definišemo DataTemplate u kome ćemo navesti izgled koji želimo za elemente

ListView

- U okviru ListView resursa možemo da ulepšamo izgled elemenata na razne načine
- Npr u ListView.Resources možemo navesti Style u okviru koga preko setera možemo da definišemo poravnanje elemenata u levo
- Korišćenjem stila, ciljanog na GridViewColumnHeader, koji je element koji se koristi za prikazivanje zaglavlja GridViewColumn, možemo promeniti svojstvo

HorizontalAlignment

```
<ListView.Resources>
    <Style TargetType="{x:Type GridViewColumnHeader}">
        <Setter Property="HorizontalAlignment" Value="Left" />
    </Style>
</ListView.Resources>
```

Zadatak 1 - primer korišćenja DataGrid-a kao tabele

- Napisati klasu Student koja implementira interfejs *INotifyPropertyChanged* i koja ima polja ime, prezime, broj indeksa, email (tipa string) i upisan (bool). Za datu klasu napisati konstruktor sa parametrima i svojstva za polja
- Napisati klasu Sluzba koja implementira interfejs *INotifyPropertyChanged* i čuva listu studenata `ObservableCollection<Student>`
- U klasi služba napisati i polja koja će se koristiti za menjanje izgleda aplikacije: polje *View* (tipa `ICollectionView`) koje će služiti za prikaz svih studenata i polje *GroupView* (tipa bool) koje će služiti za prikaz studenata u odnosu na vrednost bool polja upisan
- Potrebno je kreirati tabularni prikaz podataka. Tabelu simulirati korišćenjem DataGrid-a.
- Aplikacija treba da izgleda kao na slikama u nastavku

Zadatak 1 - primer korišćenja DataGrid-a kao tabele

MainWindow

Indeks	Ime	Prezime	
PR1/2020	Pera	Peric	
PR2/2020	Mara	Maric	
PR3/2020	Zoran	Zoric	
PR4/2020	Sara	Saric	

Indeks:

Ime:

Prezime:

Mail:

Upisan/a: ☐

☐ Grupisanje

Zadatak 1 - primer korišćenja DataGrid-a kao tabele

MainWindow

Indeks	Ime	Prezime
PR1/2020	Pera	Peric
PR2/2020	Mara	Maric
PR3/2020	Zoran	Zoric
PR4/2020	Sara	Saric

Indeks: PR4/2020
Ime: Sara
Prezime: Saric
Mail: sarasaric@uns.ac.sr
Upisan/a: ☒

☐ Grupisanje

MainWindow

Indeks	Ime	Prezime
Upisani		
PR1/2020	Pera	Peric
PR4/2020	Sara	Saric
Nisu upisani		
PR2/2020	Mara	Maric
PR3/2020	Zoran	Zoric

Indeks:
Ime:
Prezime:
Mail:
Upisan/a: ☐

☒ Grupisanje

TreeView

- TreeView kontrola nam omogućava da prikazemo hijerarhijske podatke, sa svakim delom podataka predstavljenim kao čvorom u stablu
- Svaki čvor tada može imati podređene čvorove, a podređeni čvorovi mogu imati podređene čvorove i tako dalje u dubinu
- Struktura stabla se često koristi za prikaz foldera i fajlova u fajl sistemu
- Kao i kod ListView kontrole, TreeView kontrola ima sopstveni tip stavke, `TreeViewItem`, koji možemo koristiti za popunjavanje TreeView-a
- Bolji način je da se TreeView poveže sa hijerarhijskom strukturom podataka i zatim koristi odgovarajući šablon za prikazivanje sadržaja

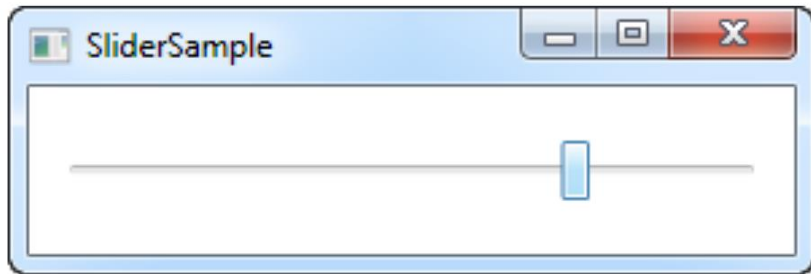
TreeView

- TreeView podržava data binding, kao i skoro sve druge WPF kontrole, ali pošto je TreeView po prirodi hijerarhijski, običan DataTemplate često neće biti dovoljan
- Umesto toga, koristimo HierarchicalDataTemplate, koji nam omogućava da šablonizujemo i sam čvor stabla, dok kontrolišemo koje svojstvo da koristimo kao izvor za podređene stavke čvora

Slider kontrola

- Slider kontrola nam omogućava da odaberemo numeričku vrednost prevlačenjem duž horizontalne ili vertikalne linije. Ovo će omogućiti krajnjem korisniku da izabere vrednost između 0 i 100 prevlačenjem dugmeta duž linije

```
<Slider Maximum="100"/>
```



Slider kontrola

- Slider ima mnogo atributa koji mu omogućavaju promenu izgleda i ponašanja
- Uobičajeni scenario u korišćenju Slider-a je da se kombinuje sa TextBox-om, što će omogućiti korisniku da vidi trenutno izabranu vrednost, kao i da je promeni unosom broja umesto prevlačenja klizača
- Obično bismo morali da se pratimo događaje i na klizaču i na TextBox-u, a zatim da ažuriramo u skladu sa tim, međutim korišćenjem binding-a možemo sve to automatski uraditi

Ostalo

- Postoji još dosta korisnih kontrolnih elemenata u WPF-u kao što su:
 - ProgressBar
 - WebBrowser
 - Calendar
 - DatePicker
 - ...

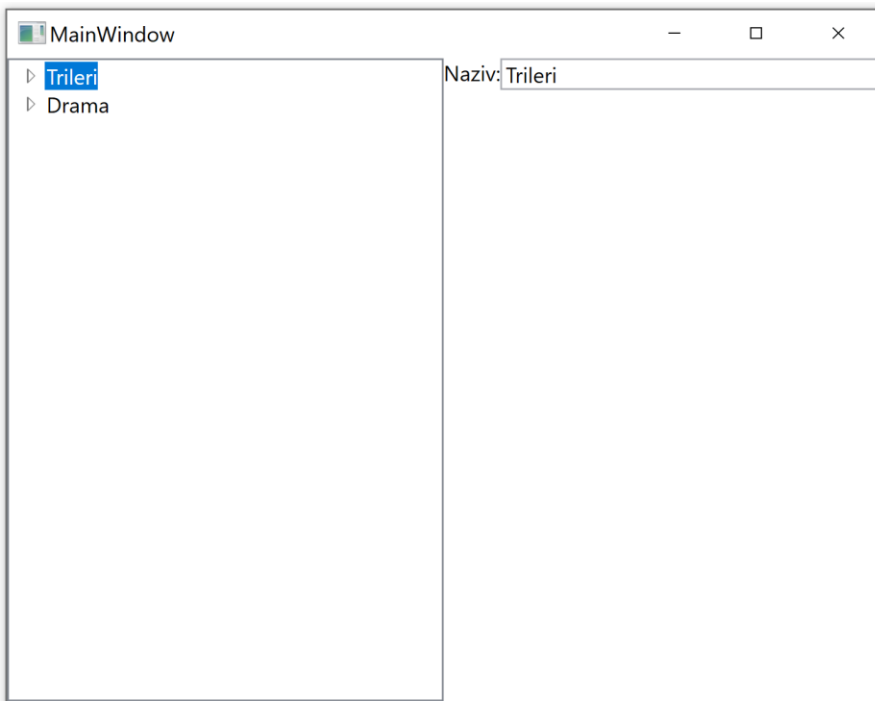
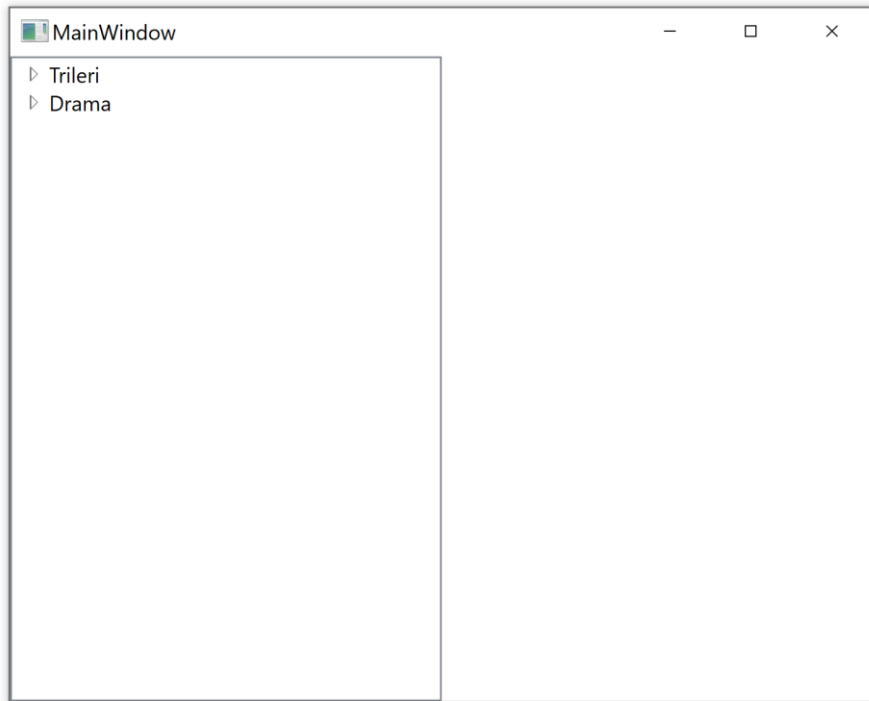
Zadatak 2

- Napisati klasu Knjiga koja implementira interfejs *INotifyPropertyChanged* i koja ima polja naslov, imeAutora, prezimeAutora (tipa string) i godinaIzdavanja (tipa int). Za datu klasu napisati svojstva za polja
- Napisati klasu Zanz koja implementira interfejs *INotifyPropertyChanged* i ima polje naziv i polje koje čuva listu knjiga `ObservableCollection<Knjiga>`. Za datu klasu napisati konstruktor bez parametara
- Napraviti izgled aplikacije u kome se sa leve strane nalazi prikaz svih žanrova i svih knjiga u okviru datih žanrova. Izgled treba da bude u formi stabla

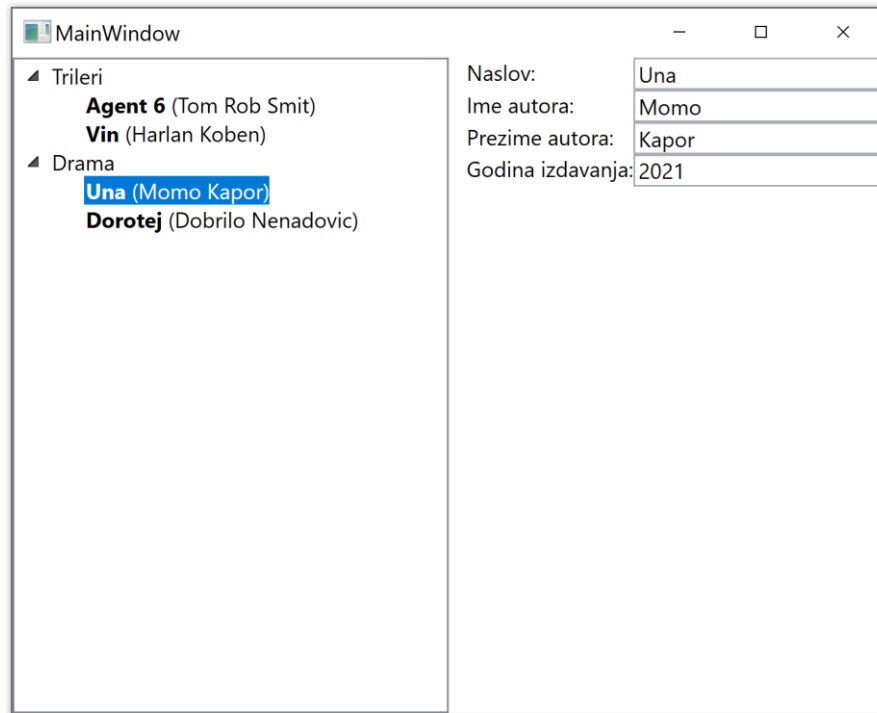
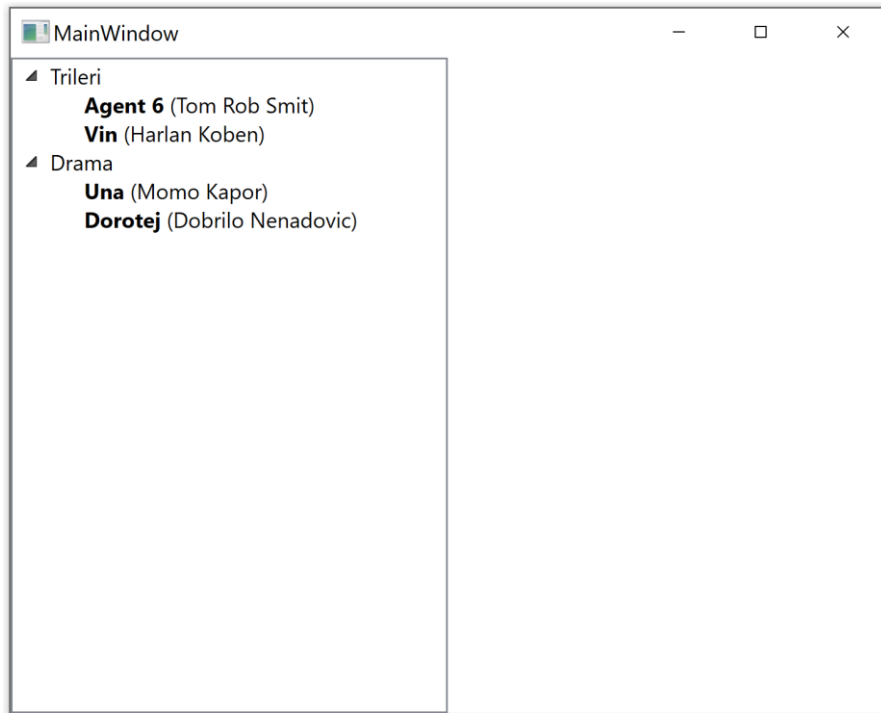
Zadatak 2

- Dodati kontekstni meni kada se desnim klikom klikne na žanr koji omogućava dodavanje nove knjige u okviru izabranog žanra
- Omogućiti izmenu selektovanog žanra i knjige
 - Kada se selektuje žanr ili knjiga potrebno je prikazati podatke o selektovanom sa desne strane i omogućiti menjanje

Zadatak 2



Zadatak 2



Zadatak 3

- Proširiti Zadatak 2 tako da se dodavanjem u kontekstnom meniju otvara novi prozor koji sadrži textbox-ove za sva polja za knjigu koja se dodaje
- Knjigu treba dodati u žanr samo ukoliko u listi ne postoji knjiga koja ima isti naslov i istu godinu izdanja (dodavanje je moguće ako je naslov isti, ali godina izdanja je drugačija)
- Potrebno je ispisati poruku o uspešnosti operacije u vidu MessageBox-a
 - Da bi dodavanje bilo moguće potrebno je napraviti klasu AddCommand koja implementira ICommand interfejs koja će opisivati ponašanje nove korisnički kreirane komande.
 - AddComand klasa treba da realizuje metode Execute i CanExecute i treba da ima žanr kao polje kako bi znali koji je odabrani žanr u okviru koga može da se izvrši dodavanje knjige

Zadatak 4

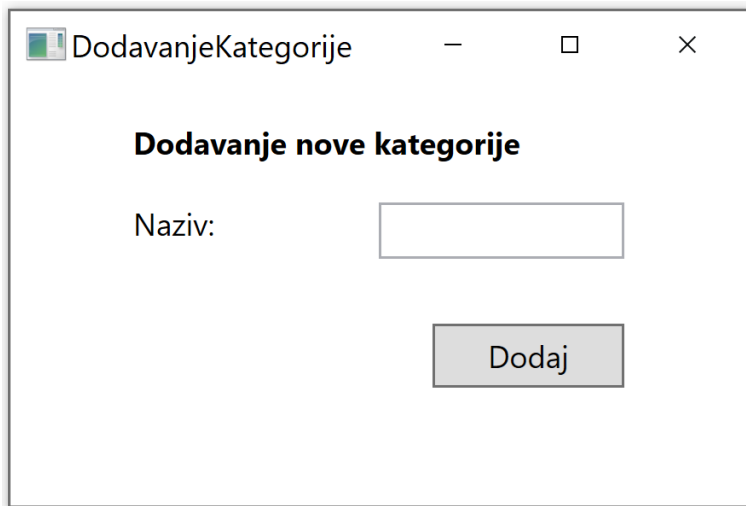
- Napisati klasu *Proizvod* koja implementira interfejs *INotifyPropertyChanged* i koja ima polja naziv (tipa string) i cena (tipa int). Za datu klasu napisati svojstva za polja
- Napisati klasu *Kategorija* koja implementira interfejs *INotifyPropertyChanged* i ima polje naziv i polje koje čuva listu proizvoda *ObservableCollection<Proizvod>*. Za datu klasu napisati konstruktor bez parametara i svojstva za polja. U klasi napisati i metodu *bool Dodaj(Proizvod p)* koja dodaje proizvod u listu ako u listi već ne postoji proizvod sa istim imenom i ako naziv proizvoda nije prazan string
- Napisati klasu *Supermarket* koji implementira interfejs *INotifyPropertyChanged* i ima polje naziv i polje koje čuva listu kategorija proizvoda *ObservableCollection<Kategorija>*. Za datu klasu napisati konstruktor bez parametara i svojstva za polja. U klasi napisati i metodu *bool Dodaj(Kategorija k)* koja dodaje kategoriju proizvoda u listu ako u listi već ne postoji kategorija sa istim imenom i ako naziv kategorije nije prazan string

Zadatak 4

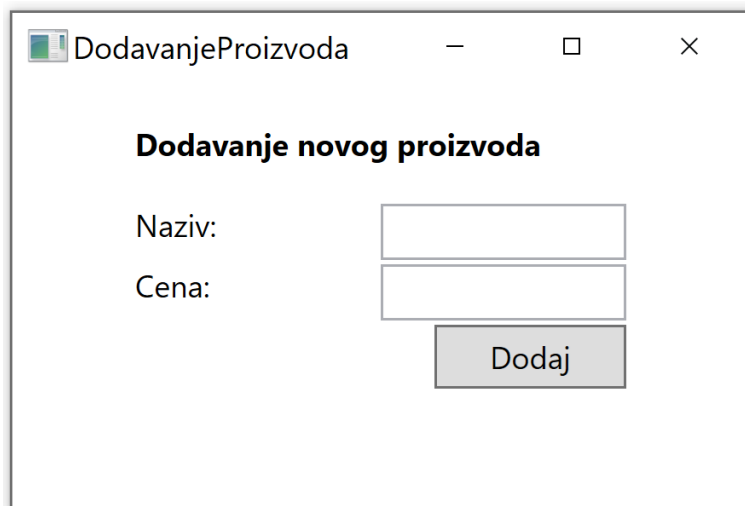
- Napraviti izgled aplikacije u kome se sa leve strane nalazi prikaz svih supermarketa, kategorija proizvoda i proizvoda. Izgled treba da bude u formi stabla
- Dodati kontekstni meni kada se desnim klikom klikne na supermarket koji omogućava dodavanje nove kategorije u okviru izabranog supermarketa i kada se desnim klikom klikne na kategoriju proizvoda omogućava dodavanje novog proizvoda u okviru odabrane kategorije. Da bi ovo bilo moguće potrebno je implementirati:
 - klase AddCommandProduct i AddCommand Category koja implementira ICommand interfejs koja će opisivati ponašanje nove korisnički kreirane komande
 - klase treba da realizuju metode Execute i CanExecute i treba da imaju kategoriju/supermarket kao polje kako bi znali koji je odabrani roditelj u okviru koga može da se izvrši dodavanje

Zadatak 4

- Napraviti i dva nova prozora za dodavanje nove kategorije i novog proizvoda
- Prozori treba da budu izgleda sličnog kao sa slika:



The screenshot shows a window titled "DodavanjeKategorije" with standard Windows window controls (minimize, maximize, close). The main heading is "Dodavanje nove kategorije". Below it, the label "Naziv:" is followed by a single text input field. At the bottom right, there is a grey button labeled "Dodaj".



The screenshot shows a window titled "DodavanjeProizvoda" with standard Windows window controls (minimize, maximize, close). The main heading is "Dodavanje novog proizvoda". Below it, the label "Naziv:" is followed by a text input field. Below that, the label "Cena:" is followed by another text input field. At the bottom right, there is a grey button labeled "Dodaj".