

WPF

Greške, tabele, 2D grafika



Greške

- Greške se odnose na greške koje nastaju tokom razvoja ili izvođenja programa. Ako ih ne pronađete i ispravite, one uzrokuju da program proizvede pogrešne rezultat ili da ne radi onako kako se od njega očekuje.
- Izuzetak je svako stanje greške ili neočekivano ponašanje koje se desi tokom izvršavanja programa. Izuzeci mogu da se pojave iz više razloga, neki od razloga su:
 - Greške u vašem kodu ili u kodu koji pozivate (kao što su biblioteke),
 - Nedostupni resursi operativnog sistema,
 - Neočekivani uslovi sa kojima se susreće uobičajeno vreme rada (kao što je kod koji ne može da se testira).



Greške

- Sve klase koje predstavljaju izuzetke u .NET radnom okviru direktno ili indirektno nasleđuju klasu ***Exception***. Najbitnije klase izuzetaka koje potiču od ove klase su:
 - *ApplicationException* class – podržava izuzetke koje generišu programi. Kada programer želi da definiše izuzetak, onda treba da nasledi ovu klasu.
 - *SystemException* class – osnovna klasa za sve unapred definisane izuzetke sistema izvršavanja.
 - Novo kreirani izuzeci mogu da naslede ili klasu *Exception* ili klasu *ApplicationException*. Na slici ispod je dat primer kako se kreira novi izuzetak.

```
public class EmployeeListNotFoundException : Exception
{
    public EmployeeListNotFoundException()
    {
    }

    public EmployeeListNotFoundException(string message)
        : base(message)
    {
    }

    public EmployeeListNotFoundException(string message, Exception inner)
        : base(message, inner)
    {
    }
}
```

Greške

- Izuzeci imaju mogućnost prenosa toka programa iz jednog dela u drugi. U .NET okviru, za rukovanje izuzecima se koriste:
 - ***Try***
 - U ovom bloku program identifikuje određeni uslov koji stvara izuzetak.
 - ***Catch***
 - Ključna reč *catch* označava hvatanje izuzetaka.
 - *Try* blok prati jedan ili više *catch* blokova u zavisnosti od broja izuzetaka koje može da izazove *try* blok.
 - ***Finally***
 - *Finally* blok se koristi za izvršavanje datog skupa izjava, bilo da se desio izuzetak ili ne
 - ***Throw***
 - Pomoću ključne reči *throw* program izbacuje izuzetak kada se pojavi problem.



Greške – obrada izuzetaka

- Na desnoj slici je dat primer kako se obrađuje neki izuzetak. U *try* bloku se nalazi kod koji može da izazove izuzetak. Pomoću *catch* bloka obrađujemo određeni izuzetak, tako što kao parametar catch funkciji prosledimo klasu izuzetka koji obrađujemo. U ovom slučaju smo prosledili *Exception* klasu, što znači da ćemo da obradimo bilo koji izuzetak. I na kraju, imamo i *finally* blok koji će uvek da se izvrši.

```
void ReadFile(int index) {  
    string path = @"D:\Test.txt";  
    StreamReader file = new StreamReader(path);  
    char[] buffer = new char[80];  
  
    try {  
        file.ReadBlock(buffer, index, buffer.Length);  
        string str = new string(buffer);  
        str.Trim();  
        textBox.Text = str;  
    }  
  
    catch (Exception e) {  
        MessageBox.Show("Error reading from " + path + "\nMessage = " + e.Message);  
    }  
  
    finally {  
        if (file != null) {  
            file.Close();  
        }  
    }  
}
```



Validacija podataka

- Većina aplikacija koje od korisnika zahtevaju unos podataka mora imati logiku validacije da bi osigurale da je korisnik uneo očekivane informacije. Provere validnosti mogu da se odnose na tip, raspon, format ili druge zahteve specifične za aplikaciju.
- Validaciju podatak možemo da odradimo na 2 načina:
 - Pomoću ručno pisanih validatora
 - Pomoću ugrađenih validatora



Validacija podataka – ugrađena validacija

- WPF prilikom Binding mehanizma podrazumevano radi i validaciju tipova
 - Proverava da li se tip unešenog podatka poklapa sa tipom atributa koji je povezan sa elementom grafičkog interfejsa
 - Ukoliko tipovi nisu jednaki dogodiće se greška prilikom provere validnosti i korisniku će se prikazati vizuelna povratna informacija koja ukazuje na grešku.
 - Podrazumevano će se prikazati crveni okvir oko elementa korisničkog interfejsa
 - Podešavanjem atributa *Validation.ErrorTemplate* možete da napravite i vlastiti šablon za prikaz greške
 - Npr. možete da ispišete poruku o grešci
 - Na sledećem slajdu, je prikazan primer ugrađene validacije sa podrazumevanim ponašanjem i ugrađene validacije sa korisnički definisanim šablonom.



Validacija podataka – ugrađena validacija

Za ovaj element će se iskoristiti ugrađena validacija za proveru tipova. Ako validacija bude neuspešna iskoristiće se podrazumevani šablon za prikaz greške.

```
<TextBox Grid.Column="1" Grid.Row="0" Text="{Binding Path=Test1,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="1" Margin="0,0,220,0" Text="{Binding Path=Test2,UpdateSourceTrigger=PropertyChanged}">
  <Validation.ErrorTemplate>
    <ControlTemplate>
      <Grid>
        <Grid.RowDefinitions>
          <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition />
          <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
        <AdornedElementPlaceholder Grid.Column="0" Grid.Row="0"/>
        <TextBlock Grid.Column="1" Grid.Row="0" Text="{Binding [0].ErrorContent}" Foreground="Red"/>
      </Grid>
    </ControlTemplate>
  </Validation.ErrorTemplate>
</TextBox>
```

TextBox za koji definišemo naš šablon za prikaz greške o neuspešnoj validaciji

Ovaj element ćemo da iskoristimo da bismo ostavili mesta za prikaz poruke. Naša poruka će se prikazati u TextBlock-u koji će se nalaziti ispod TextBox-a.

ErrorContent atribut čuva poruku koja opisuje grešku

Atribut pomoću kojeg definišemo naš šablon. On će se uvek prikazati u ukrasnom sloju, a elementi unutar tog sloja se uvek nalaze iznad ostatka vizuelnih elemenata i neće se uzeti u obzir prilikom rasporeda elemenata.



Validacija podataka – ručno pisana validacija

- Ne morate uvek da koristite ugrađene validacije, možete da napravite i svoje validacije. Da biste napravili svoju validaciju potrebno je da:
 - Kreirate klasu koja će da nasledi *ValidationRule* klasu i da implementira njenu *Validate* metodu
 - Kreiranu validaciju dodate u atribut *Binding.ValidationRules* elementa sa kojim povezujete validaciju. Prilikom dodavanja validacije potrebno je da za nju definišete i atribut *ValidationStep*, koji služi za definisanje kada će da se izvrši proces validacije. Može da ima jednu od sledećih vrednosti:
 - *RawProposedValue* – validacija će da se izvrši pre konverzije podataka. Ovo je podrazumevana vrednost, ukoliko se nenavede ovaj atribut.
 - *ConvertedProposedValue* – validacija će da se izvrši posle konverzije podataka
 - *CommittedValue* – validacija će da se izvrši nakon dodele vrednosti izvoru
 - *UpdatedValue* – validacija će da se izvrši nakon ažuriranja izvora
- Na sledećim slajdovima je dat primer kako da napravite svoju validaciju.



Validacija podataka – ručno pisana validacija

Naziv klase koja predstavlja kreiranu validaciju i ona nasleđuje klasu *ValidationRule*

```
public class StringToDoubleValidationRule : ValidationRule
{
    1 reference
    public override ValidationResult Validate(object value, System.Globalization.CultureInfo cultureInfo)
    {
        try
        {
            var s = value as string;
            double r;
            if(double.TryParse(s, out r))
            {
                return new ValidationResult(true, null);
            }
            return new ValidationResult(false, "Please enter a valid double value.");
        }
        catch
        {
            return new ValidationResult(false, "Unknown error occurred.");
        }
    }
}
```

Implementacija metode *Validate* iz klase *ValidationRule*

Ovo su poruke kojima pristupamo pomoću atributa *ErrorMessage* ukoliko je validacija neuspešna



Validacija podataka – ručno pisana validacija

Specificiramo jednu konkretnu validaciju navođenjem imena klase i atributa *ValidationStep*. Za ovu validaciju, ovaj atribut ima vrednost *RawProposedValue*, što znači da će ova validacija da se izvrši pre konverzije podataka.

```
<TextBox.Text>
  <Binding Path="Test3" UpdateSourceTrigger="PropertyChanged">
    <Binding.ValidationRules>
      <val:StringToDoubleValidationRule ValidationStep="RawProposedValue"/>
      <val:MinMaxValidationRule ValidationStep="ConvertedProposedValue" Min="10" Max="350"/>
    </Binding.ValidationRules>
  </Binding>
</TextBox.Text>
```

Definišemo atribut *ValidationRules* i u njega dodajemo naše validacije

Za neke validacije možemo da definišemo da imaju i određene attribute, kao što smo uradili za validaciju *MinMaxValidationRule*. Ova validacija ima dva atributa i to su *Min* i *Max*. Za ovu validaciju *ValidationStep* ima vrednost *ConvertedProposedValue*, što znači da će ona da se obavi posle konverzija podataka, ako ima konvertora.



Validacija podataka - postupak

1. Proces za vezivanje podataka proverava da li postoji napravljena validacija, koja je objekat klase *ValidationRule* čiji je *ValidationStep* atribut postavljen na *RawProposedValue*, i u tome slučaju poziva *Validate* iz svakog *ValidationRule* objekta sve dok jedna od njih ne izazove grešku ili dok se sve ne izvrše.
2. Proverava se da li postoje konvertovi podataka, i ukoliko postoji vrši se konverzija.
3. Ako je uspešno završena konverzija podataka, proces onda proverava da li postoje *ValidationRule* objekti čija je vrednost atributa *ValidationStep* postavljena na *ConvertedProposedValue*, a zatim se pozivaju njihove *Validate* metode sve dok jedna od njih ne izazove grešku ili dok se sve ne izvrše.
4. Postavlja se izvorno svojstvo.
5. Zatim se proverava da li postoje *ValidationRule* objekti čija je vrednost atributa *ValidationStep* *UpdateValue* i opet se poziva *Validate* metode za svaki objekat. Ukoliko je *DataErrorValidationRule* objekat vezan za objekat koji se vezuje onda se proverava i on. Takođe, proverava se i svako povezivanje za koje je *ValidatesOnDataErrors* postavljeno na *True*.
6. Na kraju, se proverava da li postoje *ValidationRule* objekti čija je vrednost atributa *ValidationStep* postavljena na *CommittedValue* i ako postoji, pozivaju se *Validate* metode za te objekte.



Validacija podataka

- Ukoliko *ValidationRule* objekat ne prođe kroz ceo proces, proces povezivanja kreira objekat *ValidationError* i dodaje ga u kolekciju grešaka.
- Ako kolekcija grešaka nije prazna onda atribut *HasError* ima vrednost *True*, a ako je atribut *NotifyOnValidationError* postavljen na *True*, onda se kreira i događaj *Validation.Error*



Tabele i mreže

TABELA

- dizajnirana za upotrebu unutar FlowDocument elementa (sadržaj sa naprednim funkcijama, kao što su straničenje i kolone)
- podržava straničenje sadržaja, preusmeravanje kolona i odabir sadržaja
- ne podržava slojevitost

GRID

- najbolje se koriste unutar oblika
- dodaje elemente na osnovu indeksa reda i kolone
- omogućuje da u jednoj ćeliji postoji više elemenata
- zahteva manje resursa od tabele



DataGrid

- Kontrola *DataGrid* vam omogućava fleksibilan način za prikazivanje i uređivanje kolekcije podataka u redovima i kolonama. Uključuje i ugrađene tipove kolona. Kolone mogu da prikazuju tekst, kontrole kao što su ComboBox ili bilo koji drugi WPF sadržaj, poput slika, dugmadi ili bilo kojeg sadržaja koji se nalazi u šablonima.
- Ona omogućava:
 - Ručno definisanje kolona
 - Automatsko definisanje kolona
 - Selekciju
 - Grupisanje
 - Sortiranje po kolonama, premeštanje kolona i promenu veličine kolone
 - Detalje o redovima
 - Zamrzavanje kolona
 - Vidljivost zaglavlja
 - Naizmeničnu promenu pozadine



DataGrid – automatsko definisanje kolona

- Da bi ste prikazali podatke koristeći *DataGrid* kontrolu, dovoljno je samo da kreirate *DataGrid* kontrolu i vežete njen *ItemsSource* atribut za kolekciju podatka.
- *DataGrid* pruža funkciju pod nazivom *AutoGenerateColumn* koja automatski generiše kolone u skladu sa javnim atributima vaših podataka. Može da kreira sledeće vrste kolona:
 - *TextBox* kolone za string vrednosti
 - *CheckBox* kolone za boolean vrednosti
 - *ComboBox* kolone za nabrojive vrednosti
 - *Hyperlink* kolone za *Uri* vrednosti

- Primer:

```
<DataGrid ItemsSource="{Binding Customers}" />
```



FirstName	LastName	Gender	WebSite	ReceiveNewsletter
Christian	Moser	Male	http://www.wpftutorial.net	<input checked="" type="checkbox"/>
Peter	Meyer	Male	http://www.petermeyer.com	<input type="checkbox"/>
Lisa	Simpson	Female	http://www.thesimpsons.com	<input type="checkbox"/>
Betty	Bossy	Female	http://www.bettybossy.ch	<input type="checkbox"/>
				<input type="checkbox"/>



DataGrid – ručno definisanje kolona

- Kolone možemo ručno da definišemo postavljanjem atributa `AutoGenerateColumn` na `False`. U ovom slučaju morate definisati kolone u kolekciju `Column` mreže podataka. Na raspolaganju su vam sledeće vrste kolona:
 - *`DataGridCheckBoxColumn`* – za boolean vrednosti
 - *`DataGridComboBoxColumn`* – za nabrojive tipove podataka
 - *`DataGridHyperlinkColumn`* – za *Uri* vrednosti
 - *`DataGridTemplateColumn`* – za prikazivanje bilo koje vrste podataka definsane vlastitim šablonom ćelije
 - *`DataGridTextColumn`* – za prikaz tekstualne vrednosti



DataGrid – selekcija podataka

- DataGrid uključuje različite načine selekcije. Konfiguriraju se pomoću atributa:

- *SelectionMode*

- Pomoću ovog atributa definišemo koliko se jedinica istovremeno može odabrati
- Vrednost ovog atributa može da bude Single ili Extended

- *SelectionUnit*

- Definiše opseg jedne selekcijske jedinice
- Može da se podesi na Cell, CellAndRowHeader i FullRow

- Primer:

- U ovom primeru smo definisali da možemo da istovremeno izabremo više ćelija.

```
<DataGrid ItemsSource="{Binding Customers}"  
  SelectionMode="Extended" SelectionUnit="Cell" />
```



Selection Mode: Extended		Selection Unit: Cell	
FirstName	LastName	Gender	WebSite
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



DataGrid – sortiranje po kolonama, premeštanje kolona i promena veličine kolone

- DataGrid pruža funkcije za sortiranje, zamenu mesta kolonama i promenu veličine kolone. Ove funkcionalnosti se podešavaju upotrebom sledećih atributa:
 - *CanUserReorderColumn* – omogućuje ili onemogućuje ponovno premeštanje kolona
 - *CanUserResizeColumn* – omogućava ili onemogućava promenu veličine kolone
 - *CanUserResizeRow* – omogućava ili onemogućava promenu veličine reda
 - *CanUserSortColumn* – omogućava ili onemogućava sortiranje kolona
- Primer:
 - U ovom primeru smo definisali da možemo da menjamo mesta kolonama, menjamo veličinu kolona i da možemo da sortiramo po kolonama.

```
<DataGrid ItemsSource="{Binding Customers}"  
  CanUserReorderColumns="True" CanUserResizeColumns="True"  
  CanUserResizeRows="False" CanUserSortColumns="True"/>
```



<input checked="" type="checkbox"/> Allow Reorder <input checked="" type="checkbox"/> Allow Resizing <input checked="" type="checkbox"/> Allow Sorting			
FirstName	LastName	Gender	WebSite
Betty	Bossy	Female	http://www.bettybossy.ch
Peter	Meyer	Male	http://www.petermeyer.com
Christian	Moser	Male	http://www.wpftutorial.net
Lisa	Simpson	Female	http://www.thesimpsons.com



DataGrid – grupisanje

- DataGrid podržava i grupisanje podataka. Da biste omogućili grupisanje potrebno je da:
 - definišete *CollectionView* koji sadrži najmanje jedan *GroupDescription*, koji definiše kriterijume za grupisanje.
 - Definišete obrazac kako grupe treba da izgledaju. To možete da uradite pomoću atributa *GroupStyle*, a kako se to radi možete pogledati na sledećem slajdu.
- Primer:
 - U ovom primeru smo napravili jedan *CollectionView* i za njega definisali kriterijum grupisanja, a u našem slučaju to je atribut Gender.

```
Customers = new ListCollectionView(_customers);  
Customers.GroupDescriptions.Add(new PropertyGroupDescription("Gender"));
```



DataGrid - grupisanje

```
<DataGrid ItemsSource="{Binding GrouppedCustomers}">
  <DataGrid.GroupStyle>
    <GroupStyle>
      <GroupStyle.HeaderTemplate>
        <DataTemplate>
          <StackPanel>
            <TextBlock Text="{Binding Path=Name}" />
          </StackPanel>
        </DataTemplate>
      </GroupStyle.HeaderTemplate>
      <GroupStyle.ContainerStyle>
        <Style TargetType="{x:Type GroupItem}">
          <Setter Property="Template">
            <Setter.Value>
              <ControlTemplate TargetType="{x:Type GroupItem}">
                <Expander>
                  <Expander.Header>
                    <StackPanel Orientation="Horizontal">
                      <TextBlock Text="{Binding Path=Name}" />
                      <TextBlock Text="{Binding Path=ItemCount}" />
                      <TextBlock Text="Items" />
                    </StackPanel>
                  </Expander.Header>
                  <ItemsPresenter />
                </Expander>
              </ControlTemplate>
            </Setter.Value>
          </Setter>
        </Style>
      </GroupStyle.ContainerStyle>
    </GroupStyle>
  </DataGrid.GroupStyle>
</DataGrid>
```



FirstName	LastName	Gender	WebSite
Male 2 Items			
Christian	Moser	Male	http://www.wpftutorial.net
Peter	Meyer	Male	http://www.petermeyer.com
Female 2 Items			
Lisa	Simpson	Female	http://www.thesimpsons.com
Betty	Bossy	Female	http://www.bettybossy.ch




DataGrid – detalji o redovima

- DataGrid pruža funkciju koja prikazuje ploču sa detaljima o odabranom redu. To se postiže postavljanjem *DataTemplate* elementa u atribut *RowDetailsTemplate*. *DataTemplate* dobija objekat koji je vezan za red koji prolazi kroz *DataContext* i može se povezati na njega.
- Primer:
 - U ovom primeru smo definisali da kada se klikne na neki red da se prikaže slika korisnika.

```
<DataGrid ItemsSource="{Binding Customers}">
  <DataGrid.Columns>
    <DataGridTextColumn Header="First Name" Binding="{Binding FirstName}" />
  </DataGrid.Columns>
  <DataGrid.RowDetailsTemplate>
    <DataTemplate>
      <Image Height="100" Source="{Binding Image}" />
    </DataTemplate>
  </DataGrid.RowDetailsTemplate>
</DataGrid>
```



First Name	FirstName	LastName	Gender	Web	ReceiveNewsletter	Image
Christian	Christian	Moser	Male	http/	<input checked="" type="checkbox"/>	Images/christian.jpg
Peter	Peter	Meyer	Male	http/	<input type="checkbox"/>	Images/peter.jpg
Lisa	Lisa	Simpson	Female	http/	<input type="checkbox"/>	Images/lisa.jpg
						
Betty	Betty	Bossy	Female	http/	<input type="checkbox"/>	Images/betty.jpg
					<input type="checkbox"/>	



DataGrid – zamrzavanje kolona

- DataGrid omogućava i funkcionalnost zamrzavanja kolona. To znači da kolone ostaju vidljive dok horizontalno pomeramo dati prikaz. Ovo je korisna funkcionalnost za zadržavanje referencijske kolone (ID ili ime), koja će uvek da bude vidljiva kako bi ste zadržali orijentaciju tokom pomeranja.
- Pomoću atributa *FrozenColumnCount* možete da definišete broj kolona koje će da budu zamrznute.
- Primer:
 - U sledećem primeru smo zamrznuli 2 kolone i to su kolone FirstName i LastName.

```
<DataGrid ItemsSource="{Binding Customers}" FrozenColumnCount="2" />
```



FirstName	LastName		Rec	Image
Peter	Meyer	meyer.com	<input type="checkbox"/>	Images/peter.j
Lisa	Simpson	mpsons.com	<input type="checkbox"/>	Images/lisa.jpg
Christian	Moser	torial.net	<input checked="" type="checkbox"/>	Images/christia
Betty	Bossy	bossy.ch	<input type="checkbox"/>	Images/betty.j
			<input type="checkbox"/>	



DataGrid – vidljivost zaglavlja

- DataGrid vam omogućava da možete da kontrolišete vidljivost zaglavlja redova i kolona postavljanjem atributa *HeadersVisibility* na neku od sledećih vrednosti:
 - *None*
 - *Row*
 - *Column*
 - *All*
- Primer:
 - U sledećem primeru smo definisali da nam se zaglavlje ne prikazuje, tako što smo za vrednost atributa HeadersVisibility postavili None.

```
<DataGrid ItemsSource="{Binding Customers}" HeadersVisibility="None" />
```



None ▾			
Betty	Bossy	Female	http://www.bettybossy.ch
Christian	Moser	Male	http://www.yoftutorial.net
Lisa	Simpson	Female	http://www.thesimpsons.com
Peter	Meyer	Male	http://www.petermeyer.com



DataGrid – naizmenična promena pozadine

- DataGrid omogućava i da definišete pozadinu redova. Pomoću atributa *AlternatingRowBackground* ćete definisati pozadinu koja će se primenjivati na svaki parni red. Dodatno ako želite da obojite svaki n-ti red podataka, onda možete da podesiti i atribut *AlternationCount*.
- Primer:
 - U sledećem primeru smo definisali da će nam svaki drugi red biti obojen navedenom bojom.

```
<DataGrid ItemsSource="{Binding Customers}"  
  AlternatingRowBackground="Gainsboro" AlternationCount="2"/>
```



FirstName	LastName	Gender	WebSite
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



2D Grafika

- WPF nudi integrisanu podršku za multimediju, vektorsku grafiku, animaciju i kompoziciju sadržaja, što olakšava izradu zanimljivih korisničkih interfejsa i sadržaja. Takođe, nudi i širok spektar 2D grafike i funkcionalnosti vezanih za slike koje se mogu optimizovati za potrebe vaše aplikacije.
- WPF vam nudi i klase *Shape* i *Drawing* koje služe za oblikovanje i crtanje grafičkog sadržaja i klasu *Image* koja služi za rad sa slikama.



2D Grafika

- *Shape* i *Drawing* klase
 - *Shape* vam omogućava da crtate grafički oblike na ekranu. *Shape* objekti se mogu koristiti unutar panela i većine kontrola.
 - WPF pruža i objekte osnovnog oblika koji su izvedeni iz klase *Shape* kao što su *Ellipse*, *Line*, *Path*, *Polygon*, *Polyline* i *Rectangle*
 - *Drawing* objekti omogućavaju lakšu implementaciju za prikazivanje oblika, slika i teksta; jednostavniji su u odnosu na *Shape* objekte. Takođe, *Drawing* objekti imaju i bolje performanse.



2D Grafika – Drawing objekti

- Postoje 4 tipa *Drawing* objekata:
 - *GeometryDrawing* objekti
 - Koriste se za prikaz geometrijskih figura.
 - *ImageDrawing* objekti
 - Ovi objekti pružaju manje funkcija od *Image* objekata za prikazivanje slika. Međutim, pružaju bolje performanse što ih čini idealnim za opisivanje pozadine, grafičkih slika i za crtanje na niskom nivou pomoću vizuelnih objekata.
 - *GlyphRunDrawing* objekti
 - Predstavljaju niz glifova sa jednog lica jednog fonta u jednoj veličini i sa jednim stilom prikazivanja.
 - *DrawingGroup* objekti
 - Koriste se za crtanje kompozitnih crteža, tj crteža koji kombinuju više crteža u jedan.



2D Grafika – Image

- *Image* klasa omogućava vam da učitajte slike različitih tipova podataka, kao što su .bmp, .gif, .ico, .jpg, .png i drugi.
- Kada se prikaže slika koja sadrži više frame-ova, onda se prikazuje samo prvi kadar. Ova kontrola ne podržava animaciju koja se sastoji od slika sa više frame-ova.
- Dok se sadržaj slike ne učitava, stvarna širina i visina kontrole će biti 0, jer se sadržaj slike koristi za određivanje veličine i lokacije kontrole. Za kontrolu fiksne veličine se mogu podesiti širina i visina. Međutim, da biste sačuvali omjer slike medija, podesite samo jedan od ovih atributa.
- Klasa *BitmapSource* je važna klasa koja se koristi za kodiranje i dekodiranje slike. Ona je osnovni gradivni blok *Image* toka, koji konceptualno predstavlja jedan konstantni skup piksela u određenoj veličini i rezoluciji. *BitmapSource* može da bude jedan frame u datoteci slike koji pruža dekodirer ili može da bude rezultat transformacije koja se radi na sopstvenom *BitmapSource*-u. Ova klasa se ne koristi za prikaz slike sa više frame-ova ili animacije. Za dekodiranje bitmape, *BitmapSource* koristi automatsko otkrivanje koda, zasnovano na instaliranim kodovima na korisničkom sistemu.



2D Grafika – Image

- Primer prikaza slike upotrebom klase *Image*

Učitavanje i
priprema slike za
prikaz

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    Uri myUri = new Uri("map.jpg", UriKind.RelativeOrAbsolute);
    JpegBitmapDecoder decoder2 = new JpegBitmapDecoder(myUri, BitmapCreateOptions.PreservePixelFormat, BitmapCacheOption.Default);
    BitmapSource bitmapSource2 = decoder2.Frames[0];

    // Draw the Image
    myImage2.Source = bitmapSource2;
    myImage2.Stretch = Stretch.Uniform;
    myImage2.Margin = new Thickness(20);
}
```

Element za prikaz
slike. Naziv ovog
elementa ćemo da
iskoristimo da
postavimo sliku koja
će se prikazivati

```
<Window x:Class="PrimerCas4._2DG.Prikaz"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Prikaz" Height="361" Width="649">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Image x:Name="myImage2" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>
        <Button Grid.Column="0" Grid.Row="1" HorizontalAlignment="Right" Margin="10" Padding="10" Click="Button_Click">Draw</Button>
    </Grid>
</Window>
```

2D Grafika

- Većina ugrađenih klasa koje smo videli na prethodnom slajdu potiču iz klase *FrameworkElement*. To je klasa koja proširuje *UIElement* klasu sledećim mogućnostima:
 - Definiše sistem rasporeda
 - Logičko stablo
 - Podrška za izražavanje stabla elemenata kao logičko stabla i prateća podrška za definisanje tog stabla u markiranju
 - Definiše događaje vezane za životni vek objekata
 - Pruža podršku za povezivanje podataka i reference dinamičkih resursa
 - Definiše stilove
 - Definiše dodatnu podršku za animacije
- Ako vam ne odgovaraju prikazi koji su dati u WPF-u, onda možete da napravite i svoje prikaze. Da biste ih napravili potrebno je da nasledite klasu *FrameworkElement* i redefinišete njenu metodu *OnRender*, a zatim da na mestu gde hoćete da vam se taj prikaz pojavi napravite objekat te klase.

