Direktna manipulacija - Drag & Drop tehnika

Drag & Drop funkcionalnost se ogleda u tome da se elementi interfejsa mogu „prevlačiti“ iz jednog dela aplikacije u drugi. Na osnovu toga, potrebno je da se odrede sami elementi i između kojih delova aplikacije će se „prevlačiti“.

Najjednostavnije je da to budu slike. Slika se određuje svojom putanjom, ali da bi se slike mogle da se „prevlače“, rešenje je napraviti klasu koja ima samo jedno polje, koje je tipa **string** i predstavlja putanju do slike (njenu lokaciju na hard-disku računara).

Da bi se slike hijerarhijski izlistale, može se iskoristiti kontrola **TreeView**. Slike će se grupisati prema ekstenziji, pa se kreira klasa koja modeluje listu slika prema ekstenziji. Da bi se unutar **TreeView**-a učitao sadržaj lista svih slika, ovakva lista mu se dodeljuje kao vrednost svojstva ***ItemsSource***. Kao resursi **TreeView**-u se dodaju način prikaza hijerarhijskih elemenata (onih koji se mogu „proširivati“ unutar kontrole **TreeView**), kao i elemenata koji su im podređeni. Hijerarhijski elementi se ispisuju kao ekstenzija slika koje su grupisane, a sami elementi unutar hijerarhije su predstavljeni imenom datoteke slike i samom slikom.

```
<TreeView Name="listaSlika" ItemsSource="{Binding SveSlike}"
          MouseLeftButtonUp="listaSlika_MouseLeftButtonUp"
          SelectedItemChanged="listaSlika_SelectedItemChanged">
     <TreeView.Resources>
            <HierarchicalDataTemplate DataType="{x:Type local:SlikePoEkstenziji}"
                                      ItemsSource="{Binding Slike}">
                <StackPanel Orientation="Horizontal">
                    <TextBlock Text="Slike ekstenzije: "/>
                    <TextBlock Text="{Binding Ekstenzija}"/>
                </StackPanel>
            </HierarchicalDataTemplate>
            <DataTemplate DataType="{x:Type local:Slika}">
                <StackPanel Orientation="Horizontal">
                    <Image Source="{Binding Putanja}" Width="50" Margin="0,0,5,0" />
                    <TextBlock Text="{Binding ImeDatoteke}"
                               VerticalAlignment="Center"/>
                </StackPanel>
            </DataTemplate>
      </TreeView.Resources>
</TreeView>
```

Listing 1. Prikaz XAML definicije sadržaja **TreeView**-a

Akcija *„Drag“* se dešava kada se promeni izabrani element (događaj *SelectionChanged*), a prekida se kada se pusti levi taster miša iznad **TreeView**-a (događaj *MouseLeftButtonUp*). Sa strane C# implementacije, biće potrebne tri promenljive: jedna tipa klase **Slika** koja označava koji se element konkretno „prevlači“, druga bi bila **bool** promenljiva koja označava da li proces „prevlačenja“ traje ili ne, a treća je pozicija objekta u konkretnoj listi odakle se uklanja nakon kompletirane *„Drag“* akcije.

Kada se reaguje na promenu izabranog elementa (selekcije) unutar **TreeView**-a, u metodi u kojoj je implementirana reakcija na događaj, se najpre mora proveriti da li korisnik već nije u procesu „prevlačenja“, jer ako jeste, mora prvo to da završi pre nego što pokuša ponovo. Ako nije bio, proverava se da li ne prevlači neki od hijerarhijskih elemenata (pošto oni služe samo za grupisanje) i ako to nije slučaj, onda u tom trenutku promene izabranog elementa započinje akciju „prevlačenja“. Potom je potrebno u kodu pristupiti korisnim informacijama o datom elementu i preuzeti iz objekta putanju (lokaciju) slike. Takođe, čuva se pozicija u konkretnoj listi objekta koji se „prevlači“.

Sa tom putanjom, formira se objekat koji se prevlači i nad njim se poziva metoda *DoDragDrop* klase **DragDrop**, kojom se instancira akcija. Kao parametri, navode se izvor odakle (klasa/prozor u kojoj se nalazi), šta se „prevlači" (objekat koji je formiran) i navode se efekti odnosno ponašanje objekta koji se „prevlači", gde se može navesti *Move* ili *Copy* (kao kada se datoteke „prevlače" između dva foldera). Ova implementacija je data u listingu koda ispod.

```
if (!dragging && listaSlika.SelectedItem.GetType() != typeof(SlikePoEkstenziji))
{
   dragging = true;
   draggedItem = (Slika)listaSlika.SelectedItem;
   selected = pronadjiElement(draggedItem);
   DragDrop.DoDragDrop(this, draggedItem, DragDropEffects.Copy |
                                          DragDropEffects.Move);
}
```

Listing 2. Implementacija unutar metode kojom se reaguje na promenu izabranog elementa (selekcije) u **TreeView**-u

Kao implementacija za reakciju na otpuštanje levog tastera miša, odnosno prekidanje *„Drag"* akcije, referenca na sam objekat koji se „prevlači" će dobiti vrednost *null*. Pored toga, **bool** promenljiva koja označava da li je proces „prevlačenja" u toku će dobiti vrednost *false*, kako bi kasnije postojao uslov za iniciranje nove *„Drag"* akcije. Ova implementacija je navedena u listingu koda ispod.

```
draggedItem = null;
dragging = false;
```

Listing 3. Implementacija unutar metode kojom se reaguje na puštanje levog tastera miša nad **TreeView** -om

Dizajn aplikacije mora imati i mesto gde se može kompletirati *„Drag & Drop"* funkcionalnost, odnosno gde se može „spustiti prevlačeni element". Jedno rešenje da se to postigne je upotrebom kontrole **Canvas**. Da bi na njenoj površini mogao da se „pusti" element, ona najpre mora biti obojena nekom bojom kako bi njena „površina" postojala. Dodatno, svojstvo ***AllowDrop*** se mora postaviti na vrednost *true*, čime se dopušta da se nad ovim elementom interfejsa izvrši *„Drop"* akcija.

Uz to, potrebno je definisati metode za reakciju na dva događaja (*DragOver* i *Drop*). Reakcija na događaj *DragOver* će predstavljati situaciju kada korisnik „prevuče" element koji želi na poziciju gde želi, ali ga još nije „pustio" na dato mesto. Reakcija je značajna i u slučaju da je potrebno da se korisnik spreči da kompletira celu *„Drag & Drop"* akciju na datom mestu, ako je pre toga tu već „spušten" određeni element, odnosno da je mesto „zauzeto".

Situacija da je mesto „zauzeto" će se modelovati pomoću postojanja resursa sa ključem *„taken"*, koji će imati vrednost *true*, zavisno od toga da li konkretan **Canvas** već ima u sebi „spušten" element (resurs se dodeljuje samom **Canvas**-u).

Pošto, radi provera o „zauzetosti" **Canvas**-a, treba da se izmeni izvršavanje samog događaja, nad baznom klasom (**Window**) se poziva metoda *OnDragOver* čiji su parametar argumenti događaja sa ciljem da se izmeni samo izvršavanje događaja. Tu se proverava da li je resurs sa ključem *„taken"* nad konkretnim **Canvas**-om ima vrednost.

Ovo se može implementirati kao univerzalna metoda, referenciranjem u odnosu na objekat sender, odnosno konkretan objekat tipa **Canvas** nad kojim se izvršava događaj. Ako resurs *„taken"* nema vrednost *null* (dodeljena mu je neka vrednost, što predstavlja slučaj da je **Canvas** zauzet), efekat će biti kao da je nemoguće kompletirati *„Drop"* akciju (*DragDropEffects.None*).

U slučaju da resurs *„taken“* ne postoji, odnosno ima vrednost *null*, efekat će biti kao da se *„Drop“* akcija može kompletirati kao kopiranje, odnosno da se napravi kompija elementa u toj površini (*DragDropEffects.Copy,* kao kada se datoteka kopira iz jednog foldera u drugi). Na kraju, pošto je izvršavanje događaja korisnički definisano, mora se i korisnički navesti kao da je događaj završen, definicijom svojstva ***Handled*** nad argumentima događaja koji tada dobija vrednost *true*. Ova implementacija je navedena u listingu koda ispod.

```
base.OnDragOver(e);
if (((Canvas)sender).Resources["taken"] != null)
{
    e.Effects = DragDropEffects.None;
}
else
{
    e.Effects = DragDropEffects.Copy;
}
e.Handled = true;
```

Listing 4. Implementacija unutar metode kojom se reaguje na *DragOver* događaj nad **Canvas**-om

Identično, implementacija za metodu kojom se reaguje na događaj *Drop*, se mora korisnički definisati, pa i ona započinje sa pozivom metode *OnDrop* čiji su parametar argumenti poziva događaja. Posle toga, bilo bi potrebno da se proveri da li je objekat koji se „prevlači“ različit od *null*. Ako je ovaj uslov zadovoljen i dati **Canvas** (sender, nad kojim se izvršava ova akcija, mora se kastovati u **Canvas** jer je tipa **object**) nema definisan resurs *„taken“* potrebno je učitati sliku na osnovu njene lokacije na disku računara.

Slika se najpre formira kao objekat klase **BitmapImage**, započinje svoju inicijalizaciju (metoda *BeginInit*), posle koje joj se dodeljuje konkretna putanja (svojstvo ***UriSource***) i završava se inicijalizacija (metoda *EndInit*). Konkretnom **Canvas**-u (sender-u) se potom za pozadinu (svojstvo ***Background***) postavlja učitana slika, odnosno objekat klase **BitmapImage**, kao parametar konstruktoru objekta **ImageBrush**, koji označava da se površina boji slikom.

Uz to, menja se tekst i njegova boja na datom **Canvas**-u. Ovo se postiže time što **Canvas** ima kao „dete“ objekat klase **TextBlock**, koji slično labeli, služi za prikaz teksta. Da bi se pristupilo „deci“ neke klase (kontrole), ona ima uz sebe svojstvo ***Children***, koji se ponaša kao kolekcija. Referencirajući se na neki objekat iz te kolekcije, dobija se objekat tipa **UIElement**, koji se mora kastovati u **TextBlock**, kako bi se moglo uticati na njegov sadržaj i boju teksta. Svojstvo koji označava sadržaj **TextBlock**-a je ***Text*** i dodeljuje mu se nova vrednost tipa **string**.

Slično kao i za ostale kontrole, za promenu boje teksta koristi se svojstvo ***Foreground***. Ako je potrebno neku boju konvertovati iz heksadecimalnog oblika definicije, koristiće se objekat klase **BrushConverter** i njegova *ConvertFrom* metoda koja pretvara string format boje u objekat klase **Brush**, koji se kastuje u objekat klase **SolidColorBrush**, kako bi se on mogao dodeliti svojstvu ***Foreground***. **Canvas**-u se, potom, dodeljuje resurs „taken“ čime se priprema za sve provere kako ne bi bilo moguće „spustiti „prevlačeni objekat“ u slučaju da je u datom kanvasu već „spušten“ neki objekat.

Na kraju, potrebno je poništiti *„Drag“* akciju, referenca na sam objekat koji se „prevlači“ će dobiti vrednost *null*. Pored toga, **bool** promenljiva koja ukazuje na to da li proces „prevlačenja“ i dalje traje se postavlja na vrednost *false*. Izabrani element unutar **TreeView**-a se uklanja jer je „prevučen“.

Pošto je i ovde izvršavanje događaja korisnički definisano, mora se i korisnički navesti kao da je događaj završen, definicijom svojstva ***Handled*** nad argumentima događaja koji tada dobija vrednost *true*. Ova implementacija je navedena u listingu koda ispod.

```
base.OnDrop(e);
if (draggedItem != null)
{
   if (((Canvas)sender).Resources["taken"] == null)
   {
      BitmapImage logo = new BitmapImage();
      logo.BeginInit();
      logo.UriSource = new Uri(draggedItem.Putanja, UriKind.Relative);
      logo.EndInit();
      ((Canvas)sender).Background = new ImageBrush(logo);
      ((TextBlock)((Canvas)sender).Children[0]).Foreground = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#FF1338BE"));
      ((TextBlock)((Canvas)sender).Children[0]).Text = draggedItem.ImeDatoteke;
      ((Canvas)sender).Resources.Add("taken", true);
   }
   ukloniElement(draggedItem);
   draggedItem = null;
   dragging = false;
}
e.Handled = true;
```

Listing 5. Implementacija unutar metode kojom se reaguje na Drop događaj nad Canvas-om

Da bi se objekti klase **Canvas** mogli osloboditi, odnosno, da se ukloni slika koja je u njih „spuštena“, to će biti implementirano tako da se u odnosu na **Button** kontrolu koja radi „oslobađanje“ pronađe koja **Canvas** kontrola joj odgovara. Ako je za nju definisan resurs *„taken“*, vraća joj se bela boja površine i tekst se vraća na prazan. Na kraju, uklanja se resurs *„taken“*, kako bi se **Canvas** oslobodio i da bi se mogla opet u njega „spustiti“ slika. Data slika se takođe vraća u **TreeView**. Ova implementacija je navedena u listingu koda ispod.

```
Canvas kanvasRoditelj = ((Canvas)((DockPanel)((Button)sender).Parent).Children[1]);
if (kanvasRoditelj.Resources["taken"] != null)
{
    vratiElement(kanvasRoditelj);
    kanvasRoditelj.Background = Brushes.White;
    ((TextBlock)kanvasRoditelj.Children[0]).Text = string.Empty;
    kanvasRoditelj.Resources.Remove("taken");
}
```

Listing 6. Implementacija unutar metode kojom se oslobađa Canvas

Da bi se realizovala manipulacija elementima **TreeView**-a, dodate su tri pomoće funkcije. Prva od njih će služiti da se pronađe element koji modeluje sliku koja se „prevlači“. Ona na osnovu ekstenzije (tipa) datoteke pronalazi kojoj grupi pripada i pretražuje listu dodeljenu toj grupi kako bi pronašla poziciju objekta i time olakšala njegovo uklanjanje. Implementacija ove funkcije je data u listingu koda ispod.

```
private int pronadjiElement(Slika draggedItem)
{
      int index = 0;
      if (draggedItem.ImeDatoteke.Split('.')[1].Equals("jpg"))
      {
           index = SveSlike[0].Slike.IndexOf(draggedItem);
      }
      else
      {
```

```
            index = SveSlike[1].Slike.IndexOf(draggedItem);
      }
      return index;
}
```

Listing 7. Funkcija koja traži i vraća indeks tekućeg objekta koji se prevlači

Druga pomoćna funkcija će iz **TreeView**-a ukloniti element koji modeluje sliku koja se „prevlači“, nakon što se on „spusti“ u konkretan **Canvas**. Element se uklanja iz grupe na osnovu indeksa (pozicije) koju je omogućila prethodna funkcija. **Listing 8** sadrži implementaciju ove funkcije.

```
private void ukloniElement(Slika draggedItem)
{
      if (draggedItem.ImeDatoteke.Split('.')[1].Equals("jpg"))
      {
            SveSlike[0].Slike.RemoveAt(selected);
      }
      else
      {
            SveSlike[1].Slike.RemoveAt(selected);
      }
}
```

Listing 8. Funkcija koja iz **TreeView**-a uklanja tekući objekat koji se prevlači

Treća pomoćna funkcija će, nakon „oslobađanja“ **Canvas**-a, vratiti element u **TreeView**, pod hijerarhijskim elementom grupe kojoj pripada, preuzimajući podatke o njemu i dodati objekat. Ova implementacija je sadržaj **Listinga 9**.

```
private void vratiElement(Canvas kanvasRoditelj)
{
     string datoteka = ((TextBlock)kanvasRoditelj.Children[0]).Text;
     if (datoteka.Split('.')[1].Equals("jpg"))
     {
          ImageBrush slika = (ImageBrush)kanvasRoditelj.Background;
          SveSlike[0].Slike.Add(new Slika() { ImeDatoteke = datoteka, Putanja =
slika.ImageSource.ToString() });
     }
     else
     {
          ImageBrush slika = (ImageBrush)kanvasRoditelj.Background;
          SveSlike[1].Slike.Add(new Slika() { ImeDatoteke = datoteka, Putanja =
slika.ImageSource.ToString() });
     }
}
```

Listing 9. Funkcija koja u **TreeView** vraća „oslobođeni“ objekat iz **Canvas**-a

Sve ove funkcionalnosti su iskorišćene da se kreira primer za petu nedelju vežbi koji sarži TreeView sa četiri slike grupisane po ekstenzijama i četiri Canvas kontrole u koje se te slike mogu „prevući“ Drag & Drop tehnikom, a potom osloboditi klikom na dugme ispod odgovarajućeg Canvas-a.