Eseményvezérelt alkalmazások: 9. gyakorlat

A munkafüzet az MVVM architekúra alapú fejlesztésbe vezet be minket, WPF alapokon. Később, az így kidolgozott modell és nézetmodell rétegeket az Avalonia UI alapú asztali- és mobilalkalmazás fejlesztéshez is újra fel tudjuk majd használni.

Refaktoráljuk az előző gyakorlaton elkészített aszinkron képletöltő alkalmazást úgy, hogy továbbra is a WPF keretrendszert használjuk, de a kétrétegű modell-nézet architektúra helyett alkalmazzuk az MVVM (Model-View-ViewModel: modell-nézet-nézetmodell) architektúrát. Vegyük észre, hogy ehhez a modell réteget nem kell módosítanunk!

1 Nézetmodell ^{KM}

Kezdetben hozzunk létre egy új könyvtárat `ViewModel` néven. Ahhoz, hogy a nézetmodellünk mezőit hozzá tudjuk rendelni a felületi elemekhez és azok változásairól a felület értesüljön, implementálnunk kell az `INotifyPropertyChanged` interfészt. Annak érdekében, hogy ezt az implementációt ne kelljen minden nézetmodellben megtenni, készítsünk egy absztrakt `ViewModelBase` osztályt:

```
public abstract class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    public ViewModelBase() { }

    protected virtual void OnPropertyChanged(
      [CallerMemberName] string? propertyName = null)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Megjegyzés: A `[CallerMemberName]` annotáció azt jelöli, hogy a megjelölt `string` paramétert a fordító tölti ki fordítás közben a függvényt aktuálisan közvetlen meghívó függvény nevével annak alapértelmezett `null` értéke esetén.

Az előadáson ismertetett módon hozzuk létre a `DelegateCommand` osztályt, amely megvalósítja az `ICommand` interfészt, így nem kell majd minden parancshoz külön osztályt származtatnunk. Az osztály konstruktora paraméterként várja a szükséges `execute` és `canExecute` `Action` és `Predicate` típusú kifejezéseket. Az osztály legyen kiegészítve egy `RaiseCanExecuteChanged` metódussal, amely a `CanExecuteChanged` eseményt fogja kiváltani.

```
public class DelegateCommand : ICommand
{
  private readonly Action<object?> _execute;
  private readonly Predicate<object?>? _canExecute;
  public event EventHandler? CanExecuteChanged;

  public DelegateCommand(
    Action<object?> execute, Predicate<object?>? canExecute = null)
  {
    if (execute == null)
    {
      throw new ArgumentNullException(nameof(execute));
    }

    _execute = execute;
    _canExecute = canExecute;
  }
  public bool CanExecute(object? parameter)
  {
    return _canExecute == null ? true : _canExecute(parameter);
  }
  public void Execute(object? parameter)
  {
    if (!CanExecute(parameter))
    {
      throw new InvalidOperationException("Command execution is disabled");
    }

      _execute(parameter);
  }
  public void RaiseCanExecuteChanged()
  {
    CanExecuteChanged?.Invoke(this, EventArgs.Empty);
  }
}
```

1.1 MainViewModel

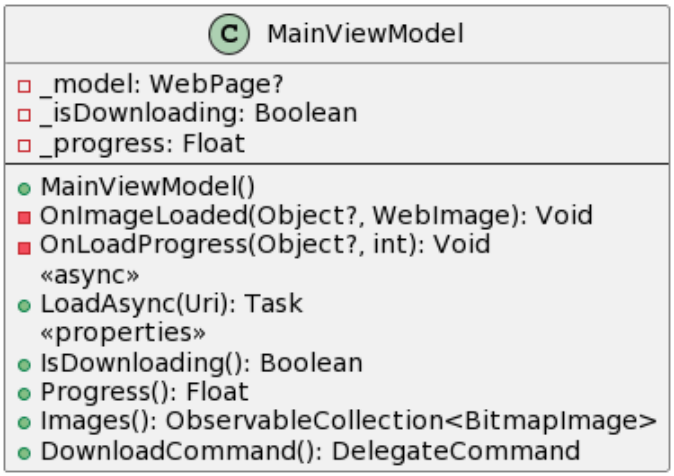

Figure 1: A MainViewModel UML-diagramja

A `MainViewModel` osztály fogja szolgáltatni az adatokat a nézet (`MainWindow`) számára, ezért ennek az oszálynak a következőkkel kell rendelkeznie:

- `IsDownloading` property (`bool`): megadja, hogy jelenleg folyamatban van-e letöltés.
- `Progress` property (`int`): százalékosan megadja, hogy jelenleg hol tart a letöltési folyamat.
- `Images` property (`ObservableCollection`): a letöltött képeket tartalmazó kollekció.
- `DownloadCommand` property (`DelegateCommand`): a letöltés parancsa.

A property-k úgy legyenek létrehozva, hogy a publikus *getter*-ük adja vissza az megfelelő privát adattag nevét. A *setter* legyen privát, amely az adattag értékének a beállítása után hívja meg az `OnPropertyChanged` metódust. (Az `IsDownloading` setter-e ezután még hívja meg a `DownloadCommand` `RaiseCanExecuteChanged` metódusát is, ugyanis majd később ezen fog múlni, hogy a gomb aktív lesz-e.)

Az `Images` property esetén az `ObservableCollection` típus biztosítja bizonyos események kiváltását a kollekció megváltozása esetén (`INotifyCollectionChanged` interfész).

> Fontos, hogy az adatkötést a nézet a publikus property-kre tudja elvégezni. Amennyiben a kötés kétirányú, akkor kell publikusan írhatónak is lennie a property-nek.

Például a `Progress` property:

```
public float Progress
{
    get => _progress;
    private set
    {
        _progress = value;
        OnPropertyChanged();
    }
}
```

1.1.1 Konstruktor

A `MainViewModel` konstruktura példányosítsa az `Images` property-t, majd hozza létre a `DownloadCommand`-ot. Utóbbi esetén a feltétel az legyen, hogy éppen nem zajlik letöltés (ekkor lesz majd aktív a gomb is), és a feltétel fennállása esetén lefutó akció pedig a következő lépésben implementálandó `LoadAsync` metódus meghívása és eredményének aszinkron megvárása (`await`) legyen.

1.1.2 LoadAsync

A `LoadAsync(Uri)` aszinkron metódus kezdetben állítsa az `IsDownloading`-ot `true`-ra (jelezvén, hogy elkezdődött a letöltés), majd törölje az `Images` elemeit.

Ezután példányosítson egy új `WebPage` modellt a `_model` adattagba, a paraméterként kapott url segítségével. Az új modell példány eseményeire való feliratkozás után hívjuk meg a modell `LoadImagesAsync` metódusát (és várjuk is meg a befejeződését).

Végül az `IsDownloading` értéke legyen `false`-ra állítva, jelezve, hogy a letöltés befejeződött.

1.1.3 OnImageLoaded

Az előző gyakorlaton implementált módon hozzunk létre a paraméterként kapott `WebImage` felhasználásával egy új `BitmapImage`-t, majd ezt adjuk hozzá az `Images` kollekció. (Ilyenkor nincs szükségünk manuálisan kiváltanunk a nézet értesítését célzó eseményt, erről az `ObservableCollection` gondoskodik helyettünk.)

1.1.4 OnLoadProgress

Az `OnLoadProgress` eseménykezelő a `Progress`-t állítsa be a paraméterként kapott értékre.

2 Nézet *EM*

A `MainWindow.xaml.cs` fájl eddigi tartalma törölhető, hogy csak az osztály és a konstruktora maradjon! Az ebben a fájlban leírt funkcionalitást (megjelenítés logika) a nézetmodell, a `MainViewModel` vett át.

A `MainWindow.xaml`-ben leírt felület struktúrája továbbra is megfelelő, ám pár módosítást ez is igényel:

- A legtöbb vezérlő esetén a `Name` tulajdonság törölhető, erre csak az `urlTextBox` esetén lesz szükség, ugyanis arra a nézeten belül a nyomógomb vezérlőből is hivatkozni akarunk majd.
- Töröljük a letöltést indító gomb `Click` eseményét, helyette a `Command`-ot állítsuk be. A parancsnak átadott paramétert a `CommandParameter` segítségével szabályozhatjuk: ezt a paramétert kössük az `urlTextBox` `Text` tulajdonságának értékéhez.

```
<Button DockPanel.Dock="Top"
        Content="Képek betöltése"
        Command="{Binding DownloadCommand}"
        CommandParameter="{Binding ElementName=urlTextBox, Path=Text}" />
```

- A letöltött képek számát megjelenítő felirat szövegét kössük az `Images.Count` értékéhez.
- A letöltés közben megjelenő `ProgressBar` értékét kössük a `Progress` property értékéhez. (A kötés módja `OneWay` legyen, mivel ezt az értéket csak olvasni szeretnénk és a `Progress` nem rendelkezik setterrel.)
- A `ProgressBar` láthatóságát valamilyen módon az `IsDownloading` értékéhez kellene kötnünk, de míg az `IsDownloading` `bool` típusú, a láthatóságot a `Visibility` enummal kell megadni. Az ilyen esetekben, amikor különböző típusú tulajdonságokhoz szeretnénk kötést létrehozni, alkalmazhatunk úgynevezett konvertereket. A leggyakrabban szükséges konverterek előre definiáltak a keretrendszerben, jelen esetben egy `BooleanToVisibilityConverter`-t vegyünk fel az ablak erőforrásai közé, amit utána használhatunk a folyamatjelzőnél:

```
<Window.Resources>
    <BooleanToVisibilityConverter x:Key="VisibilityConverter" />
</Window.Resources>
...
<ProgressBar Value="{Binding Progress, Mode=OneWay}"
             Visibility="{Binding IsDownloading,
                          Converter={StaticResource VisibilityConverter}}"
             Height="10" Width="200" />
```

- Az eddigi `WrapPanel` helyett használjunk `ItemsControl`-t, melynek `ImageSource`-át kössük a nézetmodell `Images` property-jéhez. Az `ItemsControl` használt panele (`ItemsPanel`) egy `WrapPanel` legyen (ebbe fognak felhelyezésre kerülni az elemek). Az egyes elemek használt sablon (`ItemTemplate`) pedig egy-egy `Image` legyen, amelynek `Source` tulajdonságát kötjük az aktuális elemhez. (Ezt a képet az előző óraihoz hasonló módon egy `Border`-rel körbe vehetjük.)

3 Vezérlés *EM*

Jelen állapotban az `App.xaml`-ben megadott `StartupUri` értékétől függően nyílik meg az alkalmazás elindításakor egy ablak. Ahhoz, hogy a későbbiekben nagyobb kontrollunk legyen az elindulás felett, ezt a sort töröljük ki.

Az `App.xaml.cs` fájlban található `App` osztálynak hozzunk létre egy konstruktort, amelyben a `Startup` eseményéhez kössünk egy `OnStartup` eseménykezelőt. Ebben példányosítsunk egy új `MainWindow` objektumot (amelynek `DataContext` tulajdonságát állítsuk be egy új `MainViewModel` példányra), majd ennek hívjuk meg a `Show` metódusát.

4 Képek megjelenítése és mentése ^{OP}

A `MainViewModel`-be vegyünk fel egy új eseményt és egy parancsot:

- Az `ImageSelected` eseményt (amelynek argumentuma maga a kiválasztott kép legyen), ez egy kép kiválasztását fogja jelezni a vezérlés felé.
- Az `ImageSelectedCommand` parancsot is, amelynek az akciója kiváltja az előbbi `ImageSelected` eseményt.

A `MainWindow.xaml`-ben módosítsuk az `ItemsControl` elem sablonját (`ItemTemplate`), hogy az `Image`-t mostantól ne egy `Border` tartalmazza, hanem egy `Button`, amelyhez az `ImageSelectedCommand` van kötve, ugyanis keretre és képre közvetlenül nem alkalmazható parancskötés.

Vegyük észre, hogy ha megpróbáljuk kötni a `Button` vezérlő `Command`-jához az `ImageSelectedCommand`-ot, azt tapasztalhatjuk, hogy a kötés kialakítása sikertelen. Ennek oka, hogy a gomb egy `ItemsControl` vezérlő `ItemTemplate` sablonjában helyezkedik el, előbbit pedig a nézetmodell `Images` gyűjteményére kötöttük (`BitmapImage` objektumok gyűjteménye). Az `ItemTemplate`-ben a parancsot alapértelmezetten így az éppen aktuális elem, azaz egy egy `BitmapImage` tulajdonságai között keresi. Egy lehetséges megoldás, ha a kötés `RelativeSource`-t beállítjuk, például arra, hogy `Window` típusú őst keressen. Ilyen esetben viszont a `DataContext.ImageSelectedCommand`-hoz kell kötni, mert a `Window` típusú vezérlőtől így érhető el a kívánt parancs:

```
<ItemsControl.ItemTemplate>
  <DataTemplate>
    <Button Command="{Binding DataContext.ImageSelectedCommand,
                      RelativeSource={RelativeSource AncestorType=Window}}"
            CommandParameter="{Binding}"
            Margin="2">
        <Image Source="{Binding}" Width="100" Height="100" />
    </Button>
  </DataTemplate>
</ItemsControl.ItemTemplate>
```

4.1 ImageWindow

Hozzunk létre a `ViewModelBase`-ből leszármazó `ImageWindowViewModel` osztályt. Ez az osztály fog az `ImageWindow` nézet nézetmodelljeként szolgálni. Az osztály rendelkezzen egy `Image` nevű, `BitmapImage` típusú property-vel, egy `SaveImageCommand` paranccsal, valamint egy `SaveImage` eseménnyel. A `SaveImageCommand` hatására váltódjon ki a `SaveImage` esemény.

Módosítsuk az `ImageWindow.xaml`-t, hogy a megjelenő gomb kattintására a `SaveImageCommand` fusson le, valamint a kép `Source` tulajdonságát az `Image` property adja. Az `ImageWindow.xaml.cs` tartalmát visszaállíthatjuk az eredetire (csak az osztály egy konstruktorral).

4.2 Vezérlés

Architekturálisan nem lenne helyes, ha a `MainViewModel`, mint nézetmodell új nézetet (`ImageWindow`) jelenítene meg. Ez a nézetmodell réteg egyszerű újrafelhasználhatóságát (pl. *Avalonia UI* keretrendszerben) sem tenné lehetővé.

Az új ablak megjelenítése ezért az alkalmazás környezeti/vezérlő komponens feladata lesz, amit az `App.xaml.cs` fájlban implementáltunk. Az `App` osztály `OnStartup` metódusában iratkozzunk fel egy eseménykezelővel a `MainViewModel` nézetmodell `ImageSelected` eseményére. Ebben hozzunk létre a paraméterként kapott kép alapján egy új `ImageWindowViewModel`-t, majd ennek segítségével egy új `ImageWindow`-t, amit jelenítsünk meg a `Show` metódussal.

Végül az előbb létrehozott `ImageWindowViewModel` nézetmodell `SaveImage` eseményére kell feliratkozni egy eseménykezelővel, amely tartalmazni fogja a mentés logikáját az előző gyakorlaton implementált módon.

5 Letöltés megszakíthatósága ^{OP}

Adjunk hozzá a `MainViewModel`-hez egy `CancelLoad` nevű metódust, amely meghívja a modell `CancelLoad` metódusát, ha éppen zajlik letöltés. Ezután módosítsuk a `DownloadCommand`-ot úgy, hogy folyamatban lévő letöltés esetén a `CancelLoad`-ot hívja, ellenkező esetben pedig az eddigi módon a `LoadAsync`-ot. (A parancs őrfeltétele ebben az esetben elhagyható, a gombnak mindig aktívnak kell lennie.)

Végezetül a gomb feliratát kell változtatni attól függően, hogy éppen van-e folyamatban letöltés. Erre vegyünk fel egy `DownloadButtonLabel` property-t, amely az `_isDownloading` értéke alapján a megfelelő `string`-et adja vissza, és ezt adatkötéssel állítsuk be a gomb feliratának. Az `IsDownloading` értéknek változása esetén hívjuk meg az `OnPropertyChanged`-et a *DownloadButtonLabel* stringgel, hogy a nézet értesüljön a felirat megváltozásáról. (A `DownloadCommand` parancs `RaiseCanExecuteChanged` eljárásnak meghívására pedig már nincs szükség.)

```
OnPropertyChanged(nameof(DownloadButtonLabel));
```

> *Tipp:* érdemes a bemutatott módon elvégezni a metódushívást, ugyanis ha később változik a property neve, fordítási hiba figyelmeztet a hibára, míg egy "beégetett" string esetén nem.