Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások

---

9. előadás

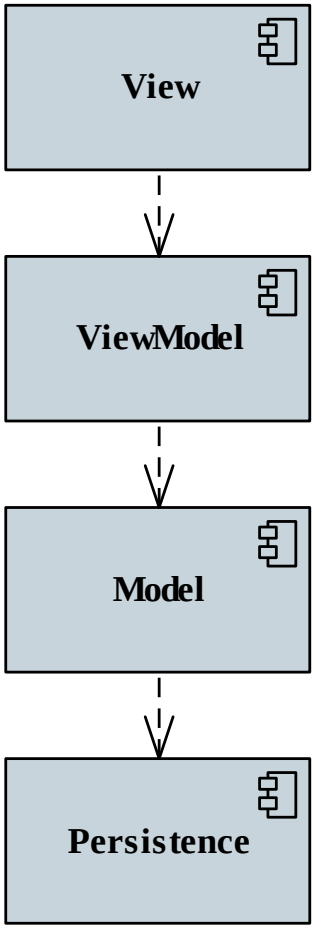
Összetett WPF alkalmazások és erőforrások kezelése

Cserép Máté
mcserep@inf.elte.hu
https://mcserep.web.elte.hu

Összetett WPF alkalmazások

Az MVVM architektúra

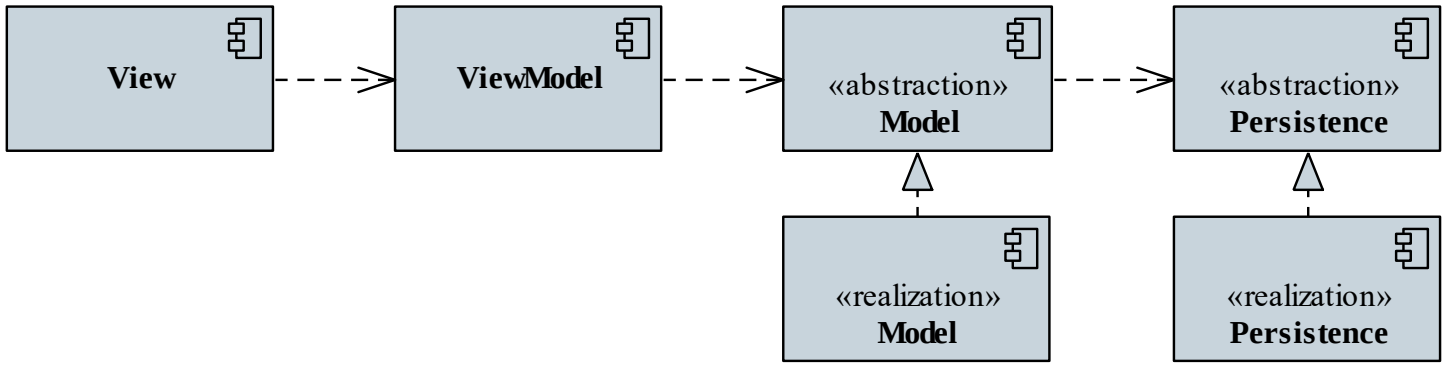
- Az MVVM architektúrában
  - a *nézet* tartalmazza a grafikus felületet és annak erőforrásait
  - a *nézetmodell* egy közvetítő réteg, lehetőséget ad a modell változásainak követésére és tevékenységek végrehajtására
  - a *modell* tartalmazza az alkalmazás logikáját
  - a *perzisztencia* a hosszú távú adattárolást és adatelérést biztosítja



Összetett WPF alkalmazások

Függőség kezelés MVVM architektúrában

- Az architektúra akkor megfelelő, ha az egyes rétegek között minél kisebb a függőség (*loose coupling*)
  - egyik réteg sem függhet a másik konkrét megvalósításától, és nem avatkozhat be a másik működésébe
  - ennek eléréséhez függőség befecskendezést (*dependency injection*) használunk



Összetett WPF alkalmazások

Függőség kezelés MVVM architektúrában

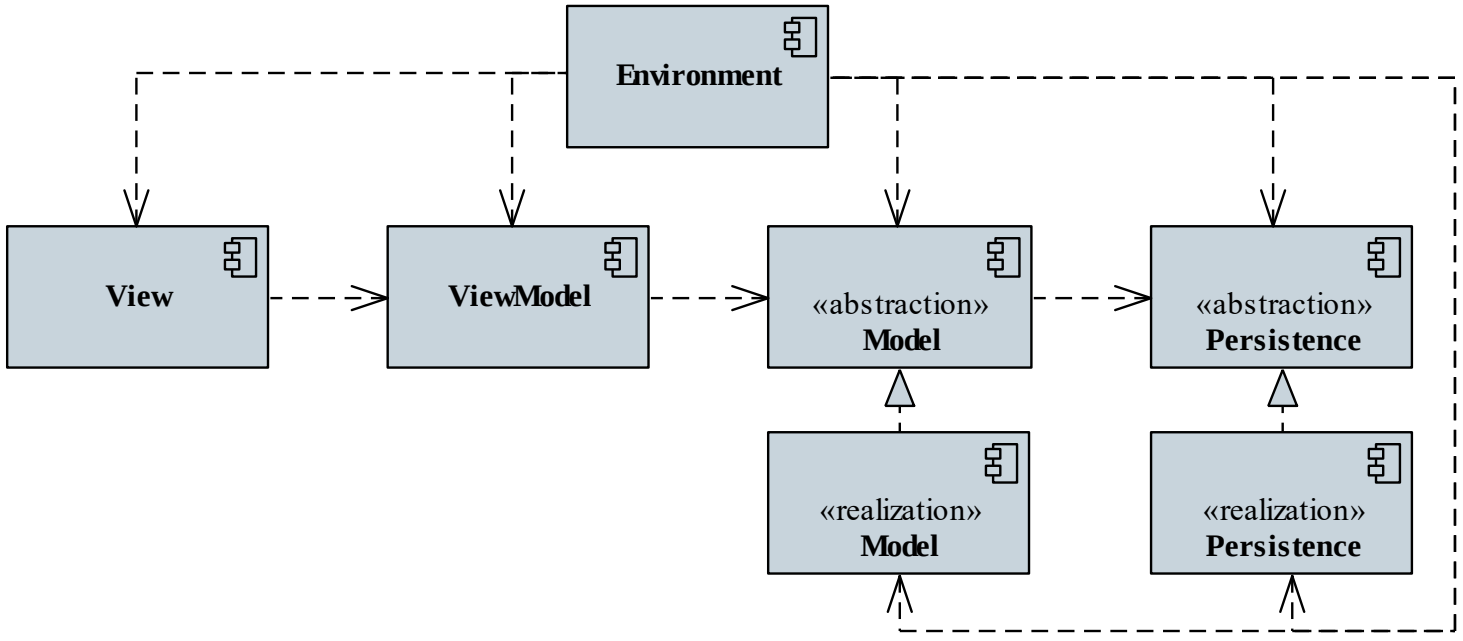
- a nézetmodellt a nézetbe egy tulajdonságon keresztül fecskendezzük be (*setter injection*)
- a modellt a nézetmodellbe, a perzisztenciát a modellbe konstruktoron keresztül helyezhetjük (*constructor injection*)

- A programegységek példányosítását és befecskendezését az *alkalmazás környezete* (*application environment*) végzi
  - ismeri és kezeli az alkalmazás összes programegységét (absztrakciót és megvalósítást is)
  - nem az adott komponens, hanem a környezet dönti el, hogy a függőségek mely megvalósításai kerülnek alkalmazásra (*Inversion of Control*, *IoC*)

Összetett WPF alkalmazások

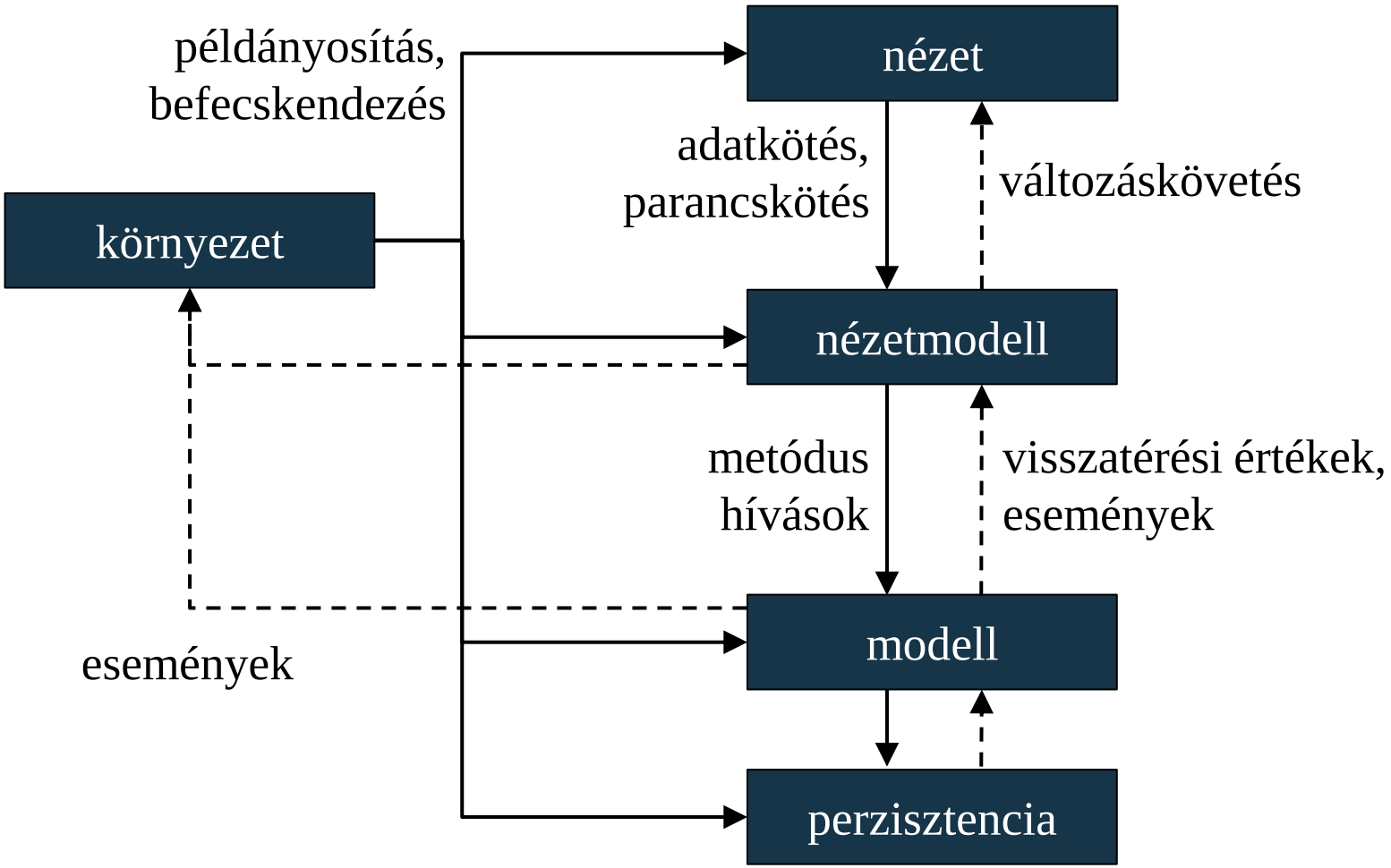
A környezet tevékenysége

- a környezetet egyszerű esetben megadhatja az alkalmazás (**`App`**), de használhatunk külön komponenst is
- a környezet hatásköre kibővíthető a globális, teljes alkalmazást befolyásoló tevékenységekkel (pl. időzítés)



Összetett WPF alkalmazások

A környezet tevékenysége



Összetett WPF alkalmazások

Időzítés

- Időzítésre használhatjuk
  - a **`System.Timers.Timer`** időzítőt, amely független a felülettől, így nem szinkronizál (a modellben)
  - a **`DispatcherTimer`** felületi időzítőt, amely szinkronizál a felülettel (környezetben, vagy nézetmodellben)
- A tevékenységek szálbiztos végrehajtása (pl. modellbeli időzítő esetén) elvégezhető a **`Dispatcher.BeginInvoke(…)`** metódussal (az alkalmazásból), pl.

```
Application.Current.Dispatcher.
    BeginInvoke(new Action(() => {
        textBox.Text = "Hello World!";
    }));
```

Összetett WPF alkalmazások

Példa

Feladat: Készítsünk egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- a modell (**`ExamGeneratorModel`**) valósítja meg a generálást, tétel elfogadást/eldobást, valamint a történet tárolását, a modellre egy interfészen keresztül (**`IExamGenerator`**) hivatkozunk
- két nézetet hozunk létre, egyik a főablak (**`MainWindow`**), a másik a beállítások ablak (**`SettingWindow`**)
- a két nézetet ugyanaz a nézetmodell (**`ExamGeneratorViewModel`**) szolgálja ki, amelybe befecskendezzük a modellt

Összetett WPF alkalmazások

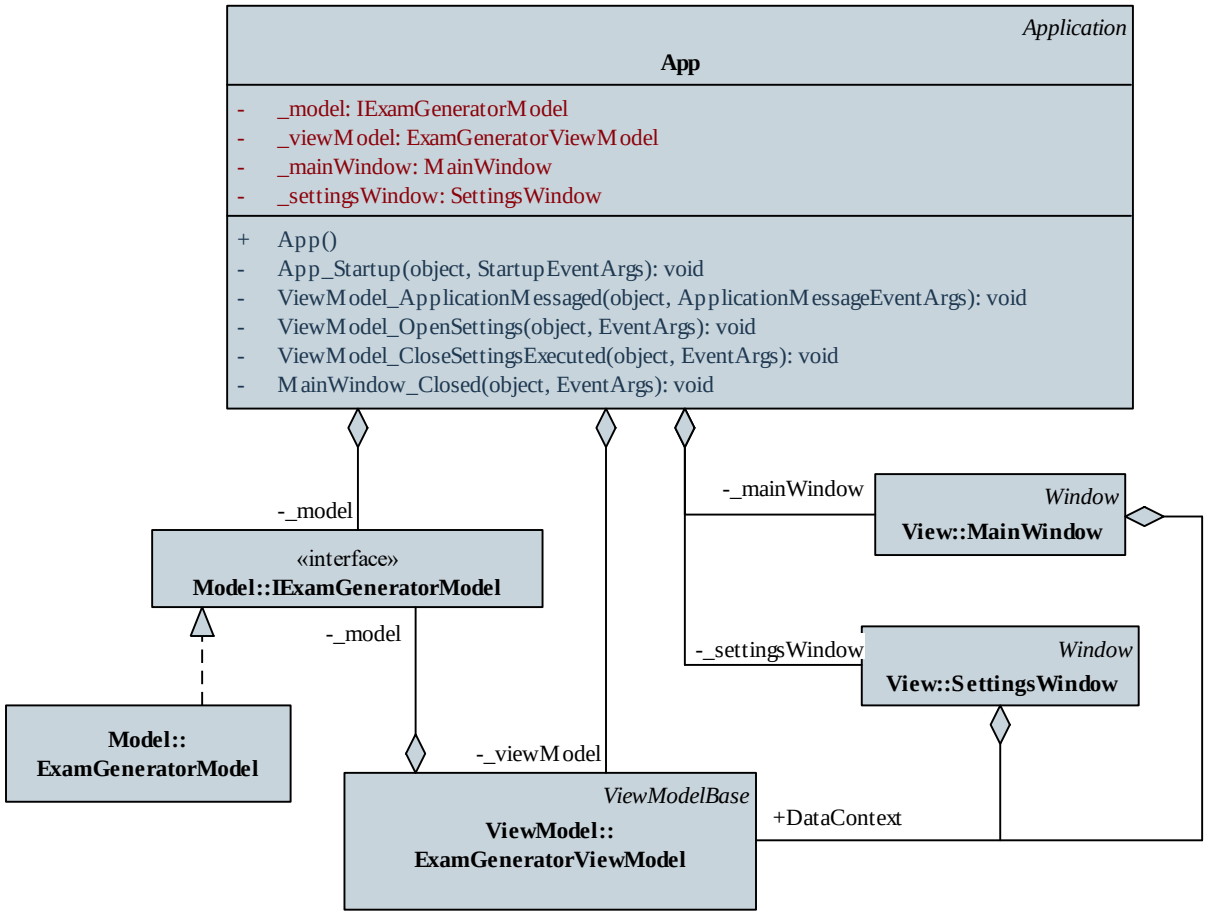
Példa

- a nézetmodell tárolja a start/stop funkcióért, beállítások megnyitásáért és bezárásáért felelős utasításokat
- a nézetmodell kezeli a modell eseményét (**`NumberGenerated`**), és frissíti a megjelenített számot
- a nézetmodell egy listában tárolja a kihúzott tételeket (**`History`**), ehhez létrehozunk egy segédtípus (**`HistoryItem`**), amely tárolja az elem sorszámát, illetve az állapotát (kiadható, vagy sem), ezeket a tulajdonságokat kötjük a nézetre
- az alkalmazás (**`App`**) felel az egyes rétegek példányosításáért, valamint a nézetmodell események kezeléséért

Összetett WPF alkalmazások

Példa

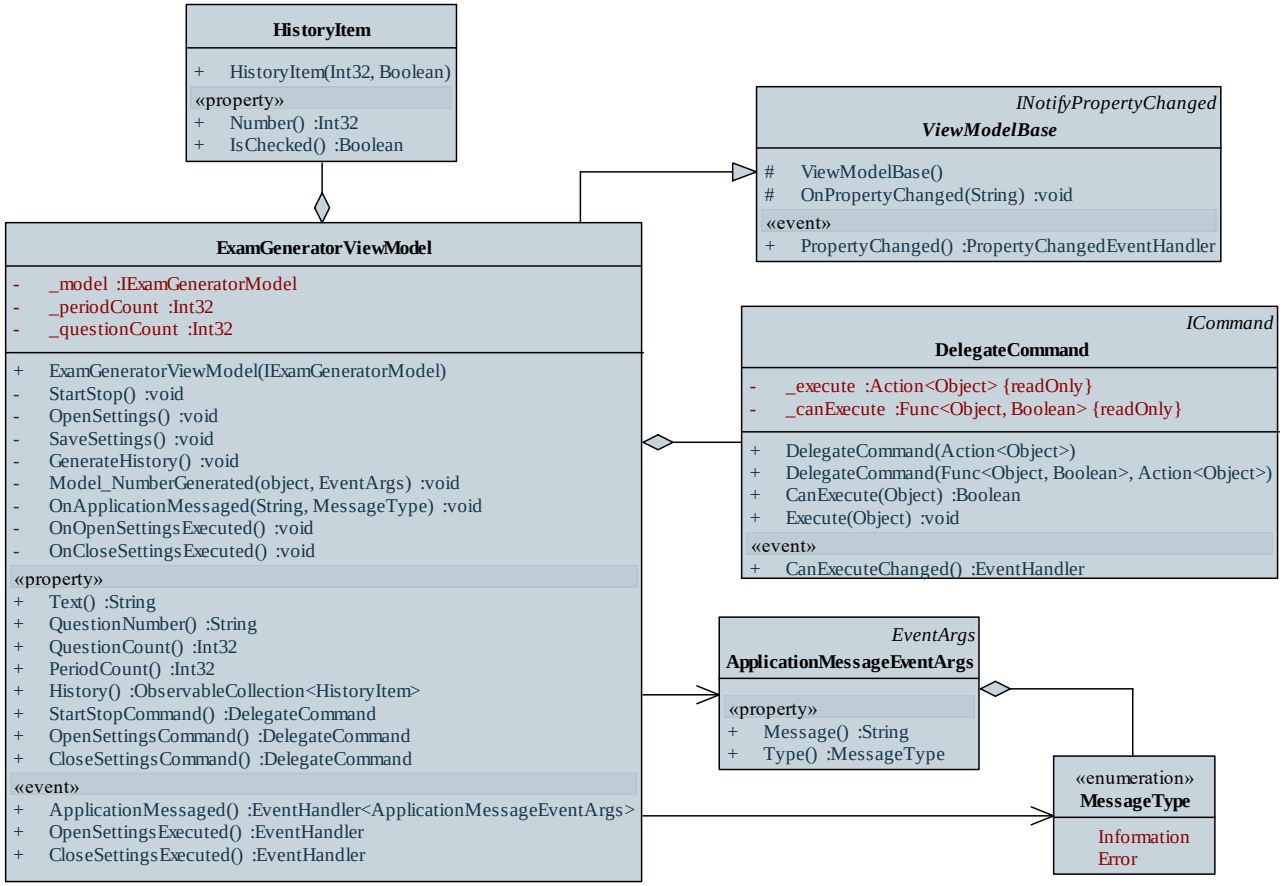
Tervezés:



Összetett WPF alkalmazások

Példa

Tervezés:



Összetett WPF alkalmazások

Példa

*Megvalósítás (**App.xaml.cs**):*

```
private void App_Startup(…)
{
    _model = new ExamGeneratorModel(10, 0);
    _viewModel =
        new ExamGeneratorViewModel(_model);
        // a nézetmodell két nézetet is kiszolgál
    …
    _viewModel.OpenSettingsExecuted +=
        new EventHandler(ViewModel_OpenSettings);
    …
    _mainWindow = new MainWindow();
    _mainWindow.DataContext = _viewModel;
}
```

Összetett WPF alkalmazások

Példa

*Megvalósítás (**App.xaml.cs**):*

```
…
private void ViewModel_OpenSettings(…) {
    if (_settingsWindow == null) {
        // ha már egyszer létrehoztuk az ablakot,
        // nem kell újra
        _settingsWindow = new SettingsWindow();
        _settingsWindow.DataContext = _ViewModel;
          // a beállításoknak is átadjuk a
          // nézetmodellt
    }
    _settingsWindow.ShowDialog();
        // megjelenítjük dialógusként
}
```

Összetett WPF alkalmazások

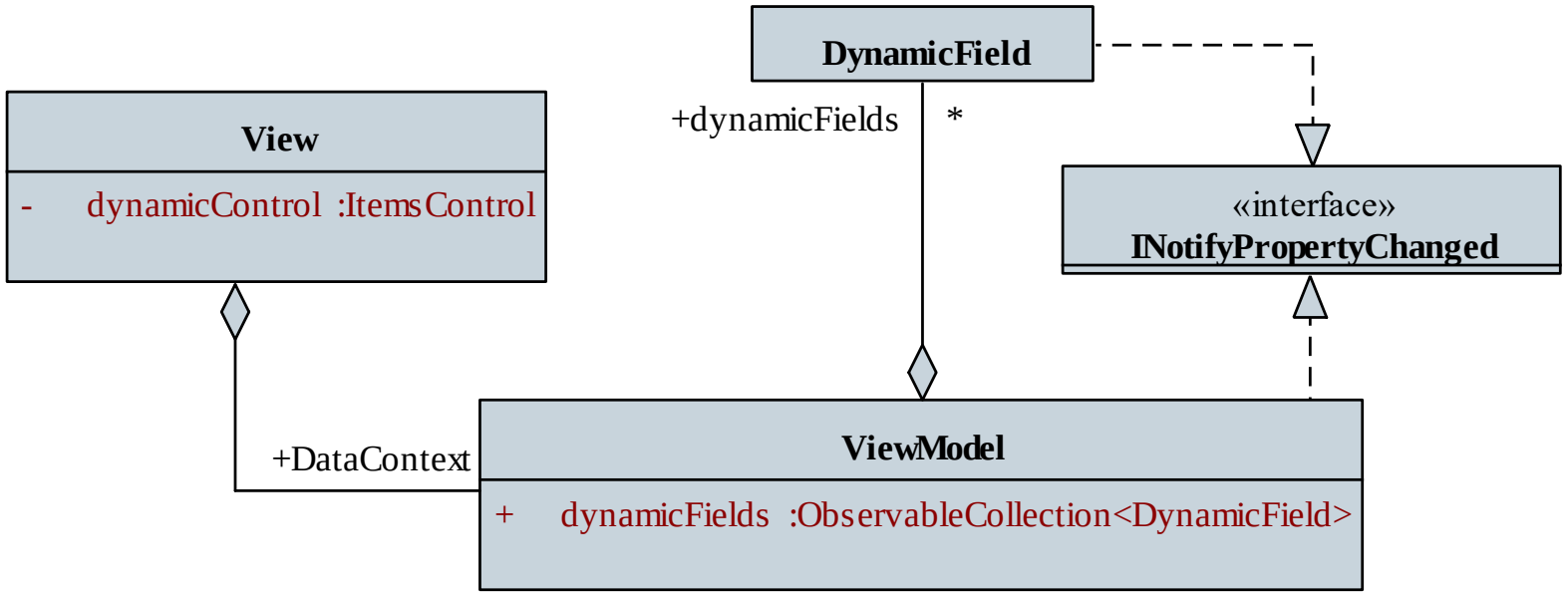
Dinamikus felhasználói felület

- Bár a WPF is lehetőséget ad vezérlők dinamikus létrehozására, az MVVM architektúra miatt speciális megközelítést igényel
  - a nézetmodellben nem hozhatunk létre vezérlőket, mivel a vezérlők megadása a nézet feladata
  - a nézetben adjuk meg a generálandó vezérlőket egy gyűjteményben
    - a gyűjteményt az **`ItemsControl`** vezérlő biztosítja, amely a megadott típusú elemeket (**`Item`**) tetszőleges tartalmazó vezérlőbe (**`ItemsPanel`**) helyezi el megadott módon (**`ItemContainer`**)
    - az elemek típusát is a nézetben adjuk meg (pl. gomb, kép, de lehet egyedi osztály is)

Összetett WPF alkalmazások

Dinamikus felhasználói felület

- a nézetmodellben a generált vezérlőhöz tartozó függőségeket helyezzük egy típusba (amennyiben szükséges), majd ezeket egy felügyelt gyűjteménybe (**`ObservableCollection`**) csoportosítjuk



Összetett WPF alkalmazások

Dinamikus mezők

- A nézetmodellbeli osztály feladata egy vezérlő összes köthető tulajdonságának (pl. parancs, tartalom) egy helyen történő kezelése
  - pl.:

```
class DynamicField {
    // a dinamikus vezérlő megjelenése a
    // nézetmodellben
    public ICommand FieldCommand { get; set; }
    public String FieldText { get; set; }
    public Int32 X { get; set; }
    public Int32 Y { get; set; }
    … // megadjuk a köthető tulajdonságokat
}
```

Összetett WPF alkalmazások

Dinamikus felhasználói felület

- Az **`ItemsControl`** egy olyan vezérlő, amelyben tetszőleges sok, azonos típusú vezérlő helyezhető el
  - az elemek sorrendje alapesetben oszlopfolytonos, azaz egymás alatt helyezkednek el (mint a **`WrapPanel`**-ben)
  - a tartalmazott vezérlőre sablont adunk az **`ItemTemplate`** tulajdonsággal, vagyis megadjuk, milyen vezérlő jelenjen meg
    - itt egy **`DataTemplate`**-t adunk meg, és abban a konkrét vezérlőt (pl. **`Button`**, **`TextBlock`**, **`Rectangle`**, …)
  - az adatforrást az **`ItemsSource`** tulajdonságon keresztül köthetjük, az elemek száma az adatforrás darabszáma lesz

Összetett WPF alkalmazások

Dinamikus felhasználói felület

- Pl.:

```
<ItemsControl ItemsSource="{Binding Fields}">
   <!-- megadjuk az adatforrást -->
   <ItemsControl.ItemTemplate>
      <DataTemplate>
         <!- megadjuk az elemek megjelenésének
             módját -->
         <Button Command="{Binding FieldCommand}"
                 Content="{Binding FieldText}" …/>
         <!-- gombokat helyezünk fel a rácsra,
              amelyek tartamát szintén kötjük -->
      </DataTemplate>
   </ItemsControl.ItemTemplate>
</ItemsControl>
```

Összetett WPF alkalmazások

Dinamikus felhasználói felület

- Az **`ItemsControl`** elrendezését felüldefiniálhatjuk az **`ItemsPanel`** tulajdonságban
  - bármilyen panel megadható (pl. **`Grid`**, **`UniformGrid`**, **`Canvas`**, **`StackPanel`**, …)

- Pl.:

```
<ItemsControl ItemsSource="{Binding Fields}">
   <ItemsControl.ItemsPanel>
      <ItemsPanelTemplate>
         <!-- tartalmazó vezérlő megadása -->
         <StackPanel Orientation="Horizontal" />
         <!-- vízszintes tájolású elrendezés -->
      </ItemsPanelTemplate>
   </ItemsControl.ItemsPanel> …
```

Összetett WPF alkalmazások

Példa

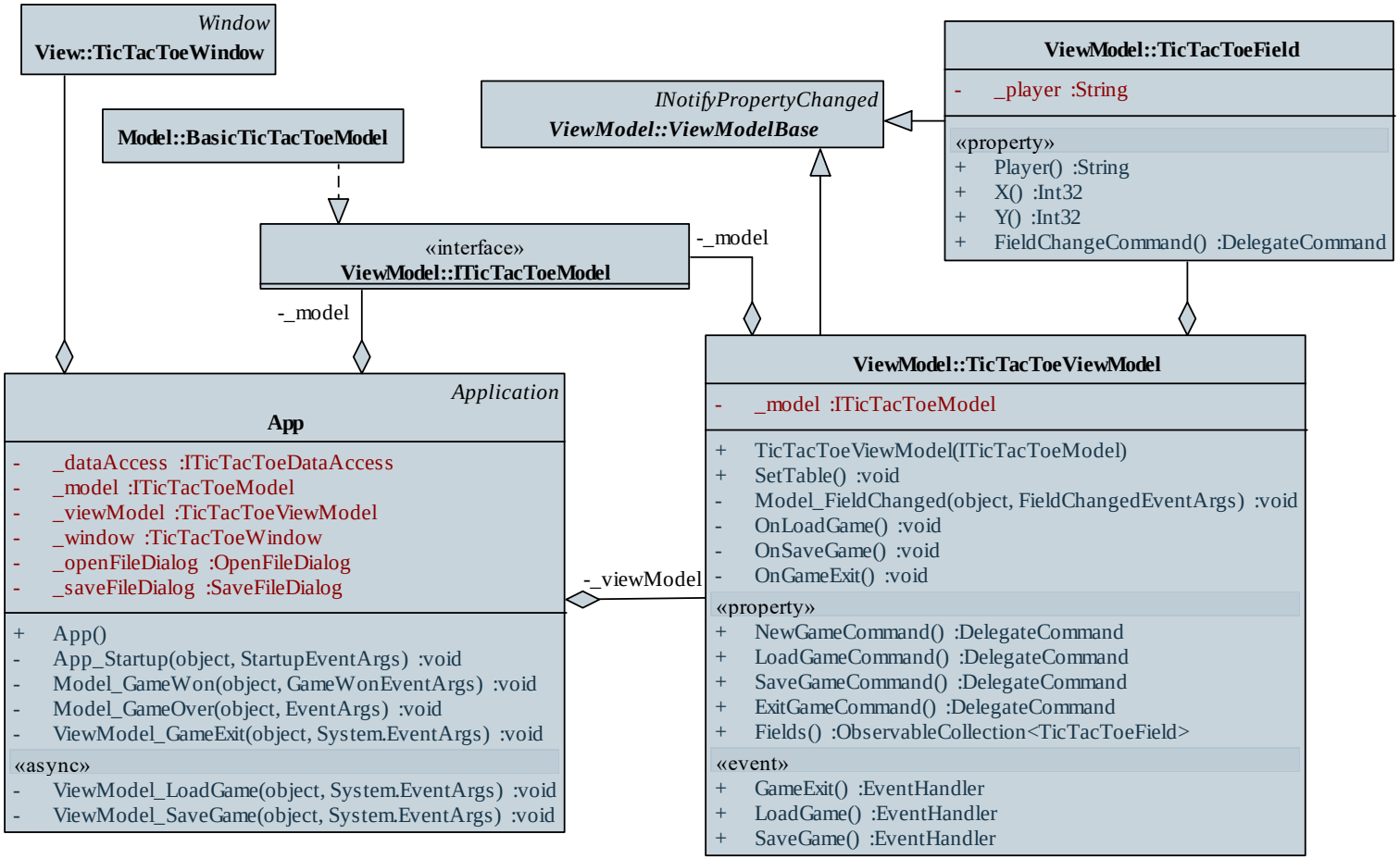
Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- MVVM architektúrát használunk, külön projektet hozunk létre a nézetmodellnek (**`TicTacToeGame.ViewModel`**), valamint a nézetnek (**`TicTacToeGame.View`**)
- a mező típusában (**`TicTacToeField`**) megadjuk az elhelyezkedést, a parancsot, valamint a mező jelét karakterként
- a felületen gombokat (**`Button`**) helyezünk el egy fix méretű **`WrapPanel`** elrendezésben, a gombok feliratát módosítjuk
- a dinamikus felületet egy **`Viewbox`**-ba helyezzük, hogy a tartalom alkalmazkodjon az ablak méretéhez

Összetett WPF alkalmazások

Példa

Tervezés:



Összetett WPF alkalmazások

Példa

Megvalósítás (`TicTacToeField.cs`*):*

```
public class TicTacToeField : ViewModelBase {
   private String _player;

   public String Player {
      get { return _player; }
      set {
         if (_player != value) {
            _player = value;
            OnPropertyChanged();
         }
      }
   }
   …
```

Összetett WPF alkalmazások

Példa

Megvalósítás (`TicTacToeViewModel.cs`):

```
…
Fields.Add(new TicTacToeField { …
    FieldChangeCommand = new DelegateCommand(
        param => {
            try {
                _model.StepGame(
                    (param as TicTacToeField).X,
                    (param as TicTacToeField).Y);
                // ha mezőre lépünk, akkor lépünk a
                // játékban
            } catch { }
        })
});
```

Összetett WPF alkalmazások

Példa

Megvalósítás (`TicTacToeViewModel.cs`*):*

```
…
private void Model_FieldChanged(object sender,
                                FieldChangedEventArgs e) {
   Fields.FirstOrDefault(
      field =>
         field.X == e.X &&
         field.Y == e.Y).
      Player =
         (e.Player == Player.PlayerX) ? 'X' : 'O';
      // lineáris keresés a megadott sorra,
      // oszlopra, majd a játékos átírása
}
```

WPF erőforrások kezelése

Erőforrások

- A Windows Presentation Foundation általánosítja az *erőforrás* fogalmát
  - a Windows Forms erőforrások azok a képek, hangok, stb. amelyeket csatolunk az egyes felületi osztályokhoz
  - a WPF-ben erőforrás lehet bármely külső fájl, sőt bármely osztály példánya, elsősorban:
    - *stílusok* (**`Style`**): a felületi elemek egységes megjelenését definiálják
    - *sablonok* (**`Template`**): a vezérlők felépülését és adatkötéseit definiálják
    - *forgatókönyvek* (**`StoryBoard`**): animációk végrehajtását biztosítják

WPF erőforrások kezelése

Erőforrások a felületi kódban

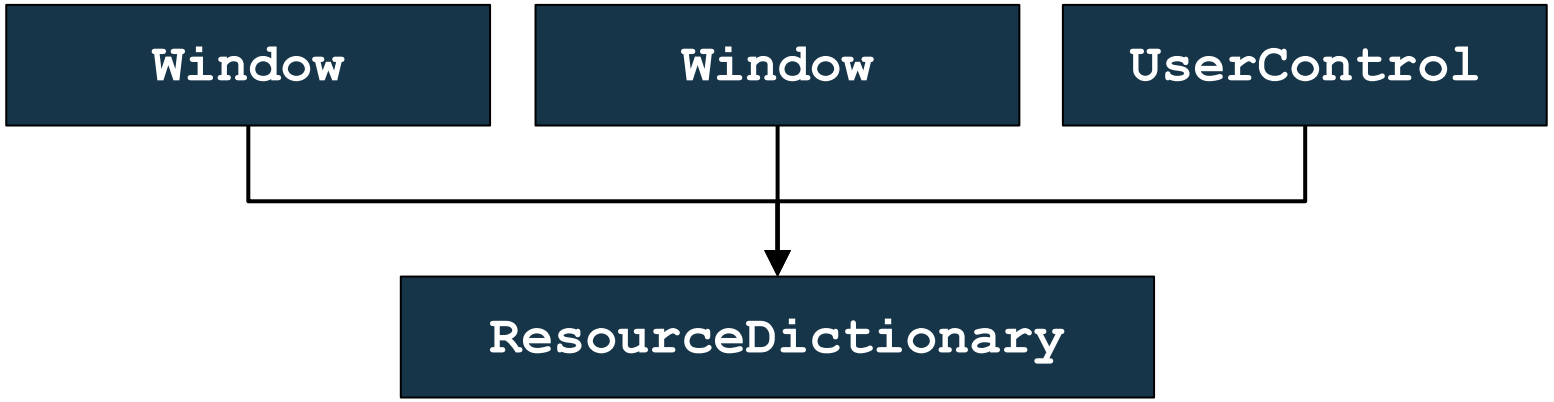
- Bármely felületi elem (**`UIElement`**) tartalmazhat erőforrásokat a **`Resources`** tulajdonság segítségével, pl.:

```
<Window … >
    <Window.Resources>
        … <!– erőforrások az egész ablakra -->
    </Window.Resources>
    <Grid Name="LayoutRoot">
        <Grid.Resources>
            … <!-- rácson belüli erőforrások -->
        </Grid.Resources>
        …
    </Grid>
</Window>
```

WPF erőforrások kezelése

Erőforrásfájlok

- Amennyiben több ablak, vagy vezérlő számára biztosítani akarjuk ugyanazt a stílus-, animáció- és sablonkészletet, akkor használhatunk *erőforrásfájl*okat (*Resource Dictionary*)



  - csak XAML erőforrásokat tartalmazó fájlok
  - használatba vehetőek bármely ablakban és egyedi vezérlőben, vagy akár a teljes alkalmazásban (az **`App`** osztályon keresztül)

WPF erőforrások kezelése

Erőforrásfájlok

- Pl.:
  erőforrásfájl (**StyleDict.xaml**):

```
<ResourceDictionary … >
    <Style x:Key=… > <!-- stíluselem -->
    …
</ResourceDictionary>
```

  felhasználása egy ablakban (**MainWindow.xaml**):

```
…
<Window.Resources>
    <ResourceDictionary Source="styleDict.xaml" />
    <!-- erőforrásfájl betöltése -->
</Window.Resources>
```

WPF erőforrások kezelése

Erőforrások használata

- Az erőforrás kulccsal (**`x:Key`**) rendelkezik, amely alapján lekérdezhetjük a **`StaticResource`** hivatkozással, pl.:

```
<Grid Name="grid">
    <Grid.Resources>
        <Style x:Key="buttonStyle"> … </Style>
        <!-- megadtuk az erőforrás célját -->
    </Grid.Resources>
    …
    <Button Style="{StaticResource buttonStyle}">
```

- Maga a **`Resources`** tulajdonság egy asszociatív tömb, amely a kulcsok szerint indexelt, pl.:

```
Style myButtonStyle =
    (grid.Resources["buttonStyle"] as Style);
```

WPF erőforrások kezelése

Vezérlők megjelenése

- A vezérlők megjelenése sokféleképpen befolyásolható, a függőségi tulajdonságok állításával, pl.:

```
<Label Content="Hello World" FontSize="20">
   <Label.Background> <!-- háttér -->
      <LinearGradientBrush> <!-- átmenetes -->
         <GradientStop Color="Green" Offset="0"/>
         <GradientStop Color="Red" Offset="1"/>
      </LinearGradientBrush>
   </Label.Background>
   <Label.Effect> <!-- speciális hatások -->
      <DropShadowEffect BlurRadius="40"
         Direction="50" Opacity="1"/>
         <!-- árnyék -->
   …
```

WPF erőforrások kezelése

Stílusok

- A stílusok (**`Style`**) olyan megjelenési beállítás gyűjtemények, amellyel egyszerre számos elem kinézetét vezérelhetjük
  - a **`FrameworkElement`** leszármazottaira használhatóak a **`Style`** függőségi tulajdonságon keresztül
  - lehetővé teszik, hogy vezérlők kinézetét egyszerre kezeljük, teljesen függetlenül az operációs rendszer beállításaitól
  - megadhatóak elemenként, pl.:

```
<Button Content="Blue Button">
  <Button.Style>
     <Setter Target="Foreground" Value="Blue" />
  </Button.Style>
</Button>
```

WPF erőforrások kezelése

Stílusok

- megadhatóak erőforrásként, pl.:

```
<Style x:Key="buttonStyle" TargetType="Button">
    <!-- megadható a céltípus is -->
    <Setter Target="Foreground" Value="Blue" />
</Style>
...
<Button Style="{StaticResource buttonStyle}" />
```

- a stílusoknak két típusát tartjuk nyilván:
  - *implicit*: mennyiben nem adunk meg kulcsot, úgy a stílus az összes megadott típusú elemre érvényes lesz, nem szükséges a **`StaticResource`** hivatkozás
  - *explicit*: a kulcs megadásával és a **`Style`** tulajdonság használatával definiáljuk a vezérlő stílusát

WPF erőforrások kezelése

Stílusok

- a stílusokban a **`Setter`** elem segítségével függőségi tulajdonságokra (**`Property`**) adunk a típusnak megfelelő értéket (**`Value`**), pl.:

```
<Style x:Key="buttonStyle" TargetType="Button">
   <Setter Property="Width" Value="400"/>
      <!-- egyszerű érték -->
   <Setter Property="Canvas.Left" Value="200" />
   <Setter Property="RenderTransform">
      <Setter.Value> <!-- összetett érték -->
         <TranslateTransform X="100" Y="50" />
      </Setter.Value>
   </Setter>
</Style>
```

WPF erőforrások kezelése

Stílusok dinamikus felületű alkalmazásokban

- Dinamikus felhasználói felületet **`ItemsControl`** vezérlő segítségével tudunk megjeleníteni
  - a megjelenítőt és az elemeket sablonok (**`ItemsPanel`**, **`ItemTemplate`**) segítségével adjuk meg
  - az elemek tárolókba kerülnek (**`ItemContainer`**)
- Amennyiben speciális megjelenítőt használunk, az elemekre függőségi tulajdonságokat alkalmazhatunk az elhelyezésre vonatkozóan
  - pl. **`UniformGrid`** esetén a **`Grid.Row`** és **`Grid.Column`** tulajdonságokkal szabályozhatjuk az elhelyezést
- Az **`ItemsControl`** az elemeket **`ContentPresenter`**-be csomagolja

WPF erőforrások kezelése

Stílusok dinamikus felületű alkalmazásokban

- a függőségi tulajdonságot nem a dinamikus vezérlőn, hanem a tárolóban kell megadnunk, stílus használatával, erre szolgál az **`ItemContainerStyle`** tulajdonság

- pl.:

```
<ItemsControl ItemsSource="{Binding Fields}">
   <ItemsControl.ItemsPanel>
      <ItemsPanelTemplate>
         <UniformGrid Rows="5" Columns="5" />
         <!-- egy 5x5-ös rácsot használunk -->
      </ItemsPanelTemplate>
   </ItemsControl.ItemsPanel>
   <ItemsControl.ItemTemplate>
      … <!-- a megjelenített elem -->
   </ItemsControl.ItemTemplate>
```

WPF erőforrások kezelése

Stílusok dinamikus felületű alkalmazásokban

```
    <ItemsControl.ItemContainerStyle>
       <!-- az elemek megjelenítési stílusa -->
       <Style>
          <!-- az elemek elhelyezését stílus
               keretében adjuk meg -->
          <Setter Property="Grid.Row"
                  Value="{Binding X}" />
          <Setter Property="Grid.Column"
                  Value="{Binding Y}" />
       </Style>
    </ItemsControl.ItemContainerStyle>
</ItemsControl>
```

WPF erőforrások kezelése

Példa

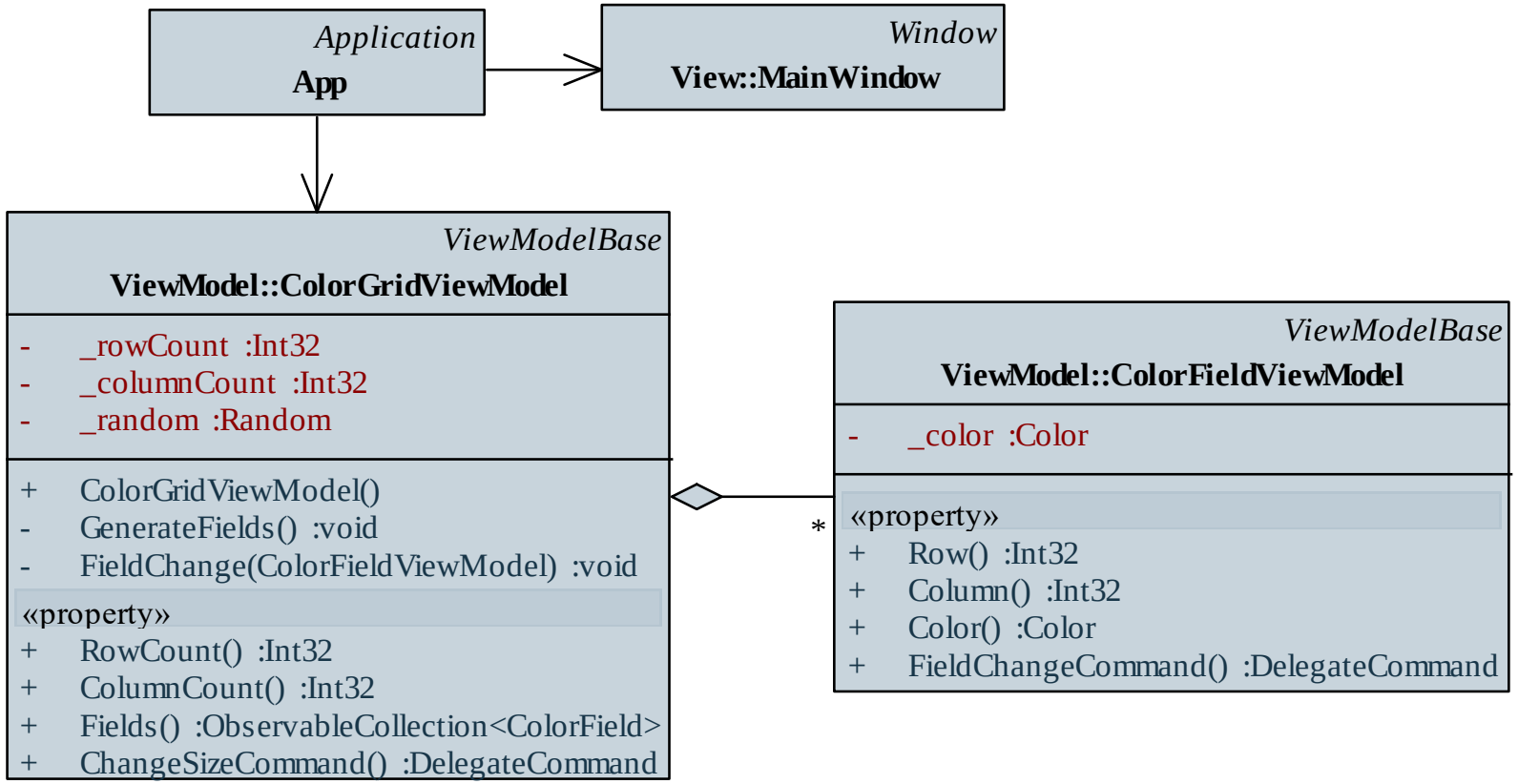
Feladat: Készítsünk egy dinamikus méretezhető táblát, amely véletlenszerű színre állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- a felületen egy **`ItemsControl`** vezérlőben helyezzük el az elemeket, amely egy **`UniformGrid`** segítségével jelenít meg gombokat (**`Button`**)
- a nézetmodell megadja a mező típusát (**`ColorFieldViewModel`**), amely tárolja a sor (**`Row`**), oszlop (**`Column`**), szín (**`Color`**) értékeket, valamint a végrehajtandó utasítást (**`FieldChangeCommand`**), amely paraméterben az egész mezőt megkapja, így a nézetmodell könnyen tudja módosítani a megfelelő elemeket

WPF erőforrások kezelése

Példa

Tervezés:



WPF erőforrások kezelése

Példa

*Megvalósítás (**MainWindow.xaml**):*

```
…
<GroupBox Margin="2" Header="Méret:" …>
   <StackPanel Orientation="Horizontal">
      <TextBlock Text="Sorok:" Margin="5" />
      <TextBox Text="{Binding RowCount}" … />
      <TextBlock Text="Oszlopok:" Margin="5" />
      <TextBox Text="{Binding ColumnCount}" … />
      <Button Name="_ChangeSizeButton"
         Command="{Binding ChangeSizeCommand}"
         Content="Méretváltás" Width="80" … />
      </StackPanel>
   </GroupBox>
…
```

WPF erőforrások kezelése

Példa

*Megvalósítás (**MainWindow.xaml**):*

```
…
<ItemsControl.ItemTemplate>
    <DataTemplate> <!-- megadjuk, milyenek legyenek
                        az elemek -->
        <Button CommandParameter="{Binding}"
            Command="{Binding FieldChangeCommand}">
            <Button.Background>
                <SolidColorBrush
                        Color="{Binding Color}" />
            </Button.Background>
        </Button>
    </DataTemplate>
…
```

WPF erőforrások kezelése

Animációk

- A WPF támogatja animációk végrehajtását, amely lényegében függőségi tulajdonságok adott időn keresztül történő folyamatos módosítását jelenti
  - az animáció típusa megadja a módosítani szánt érték típusát (pl. **`DoubleAnimation`**, **`ColorAnimation`**, **`ThicknessAnimation`**, …)
  - az animációnál definiálnunk kell a kezdőállapotot (**`From`**), a végállapotot (**`To`**), valamint az időt (**`Duration`**)
  - az animáció rendelkezhet tetszőlegesen sok köztes állapottal (**`KeyFrame`**), amelyekre egyéni kritériumok és időkorlátok szabhatóak, valamint megadható az animáció módja (lineáris, diszkrét, spline)

WPF erőforrások kezelése

Animációk

- Az animációkat forgatókönyvekbe (**`StoryBoard`**) szervezzük
  - a forgatókönyvvel megadható a célobjektum (**`Storyboard.Target`**, **`Storyboard.TargetName`**), illetve a céltulajdonság (**`Storyboard.TargetProperty`**)
  - a céltulajdonság tetszőlegesen összetett lehet, pl.:
    `Opacity`, **`Canvas.Left`**,
    `(Control.Foreground).(SolidColorBrush.Color)`,
    `(Control.RenderStransform).`
    `(TransformGroup.Children[0]).`
    `(ScaleTransform.ScaleX)`
  - a forgatókönyvvel szabályozhatjuk a végrehajtást (**`Start`**, **`Stop`**) az ismétlődést (**`RepeatBehavior`**), gyorsulási és lassulási mértéket, esetleg visszajátszást (**`AutoReverse`**)

WPF erőforrások kezelése

Animációk

- Pl.:

```
<Storyboard Storyboard.TargetName="myButton"
    Duration="0:00:04">
    <!-- forgatókönyv, amely 4 másodpercig fut a
         myButton vezérlőre -->
    <DoubleAnimation From="1" To="0"
        Storyboard.TargetProperty="Opacity" />
        <!-- áttetszővé tesszük -->
    <DoubleAnimation From="100" To="200"
        Storyboard.TargetProperty="Canvas.Left" />
        <!-- eltoljuk jobbra -->
    …
</Storyboard>
```

WPF erőforrások kezelése

Animációk végrehajtása

- Animációk végrehajthatóak kódban, valamint a felületen *trigger*ek segítségével
  - a *trigger* valamilyen esemény (**`EventTrigger`**), vagy értékváltozás (**`DataTrigger`**) hatására képes animációt futtatni (**`BeginAnimation`**), vagy tulajdonságot beállítani (**`Setter`**)
  - elhelyezhetőek stílusban, vezérlőben, sablonban, pl.:

```
<Button.Triggers>
    <EventTrigger RoutedEvent="MouseEnter">
        <!-- MouseEnter eseményre fut le -->
        <BeginStoryboard Storyboard="…" />
            <!-- animáció futtatása -->
    </EventTrigger> …
```

WPF erőforrások kezelése

Példa

Feladat: Készítsünk egy dinamikus méretezhető táblát, amely véletlenszerű színre állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- adjunk animációt a gombokhoz, amelyben az egér felülhúzására (**`MouseEnter`**) a gomb elhalványul és összemegy, majd visszaalakul eredeti formájára
- ehhez 3 animáció szükséges (áttetszőség és a két méret)
- a relatív méretezés érdekében a gomboknak a transzformációját (**`RenderTransform`**) animáljuk, így annak összetett elérési útvonala lesz (pl. (**`(Control.RenderTransform).(ScaleTransform.ScaleX)`**)

WPF erőforrások kezelése

Példa

Megvalósítás (`MainWindow.xaml`*):*

```
…
<Window.Resources>
    <Storyboard x:Key="fieldSizeStoryboard"
        Duration="0:0:2" AutoReverse="True">
        <!-- animáció a mezőkre -->
        <DoubleAnimation
            Storyboard.TargetProperty="Opacity"
            From="1" To="0"/>
        <DoubleAnimation Storyboard.TargetProperty="
            (Control.RenderTransform).
            (ScaleTransform.ScaleX)" From="1"
            To="0.5" />
…
```

WPF erőforrások kezelése

Példa

Megvalósítás (`MainWindow.xaml`):

```
…
<DataTemplate>
    <Button … >
        …
        <Button.Triggers>
            <!-- eseményre történő animálás -->
            <EventTrigger RoutedEvent="MouseEnter">
                <BeginStoryboard
                    Storyboard="{StaticResource
                                fieldSizeStoryboard}" />
            </EventTrigger>
        </Button.Triggers>
        …
```

WPF erőforrások kezelése

Megjelenítés befolyásolás

- A *triggerek* akkor is hasznosak, ha a megjelenítést akarjuk szabályozni a nézetmodell adatai alapján, pl.:

```
<Style TargetType="Button">
    <!-- stílus gombokra -->
    <Style.Triggers>
        <!-- a szín adatkötés hatására változik -->
        <DataTrigger Binding="{Binding FieldText}"
                     Value="">
            <!-- ha nincs szöveg megadva -->
            <Setter Property="Background"
                    Value="Gray" />
            <!-- a gomb szürke lesz -->
            </DataTrigger>
        </Style.Triggers>
```

WPF erőforrások kezelése

Példa

Feladat: Készítsünk egy dinamikus méretezhető táblát, amely három szín között (piros, fehér, zöld) állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- a színt a nézet adja meg, így a nézetmodell nem adhat vissza konkrét színt, csak egy sorszámot (0 és 2 között), amely alapján a szín állítható (**`ColorNumber`**)
- a színt trigger segítségével állítjuk a nézetben, a gomb stílusában, amely az érték függvényében színezi a gombot, (a gomb emellett animálódik, így **`DataTrigger`** és **`EventTrigger`** is hatni fog a vezérlőre)
- a triggereket az ablak erőforrásaként megadott stílusban hozzuk létre

WPF erőforrások kezelése

Példa

*Megvalósítás (**MainWindow.xaml**):*

```
…
<Style x:Key="buttonStyle" TargetType="Button">
   <Style.Triggers>
      <!-- a színezés a nézetmodellben lévő adat
           függvényében fog változni -->
      <DataTrigger Binding="{Binding ColorNumber}"
                   Value="0">
         <Setter Property="Background"
                 Value="Green" />
      </DataTrigger>
      …
</Style>
…
```

WPF erőforrások kezelése

Példa

Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- javítsuk a megjelenítést azáltal, hogy karakterek helyett grafikus alakzatokat (**`Line`**, **`Ellipse`**, **`Rectangle`**) jelenítünk meg a nézetben
  - a karakterek hatására változnak az elemek **`DataTrigger`** segítségével (amely a lehetséges **`Player`** értékeket figyeli)
- ugyanakkor továbbra is gombokat jelenítünk meg (amely kattintható), de felüldefiniáljuk a sablont (**`Template`**) egy egyedi felépítéssel (**`ControlTemplate`**), így a gomb megjelenése teljesen más lesz

WPF erőforrások kezelése

Példa

Megvalósítás (`TicTacToeWindow.xaml`*):*

```
<Style.Triggers>
   <DataTrigger Binding="{Binding Player}"
                Value="O">
   <Setter Property="Template">
      <!-- a gomb sablonját cserélgetjük -->
      <Setter.Value>
         <ControlTemplate>
            <Canvas Background="White">
               <Ellipse … />
            </Canvas>
         </ControlTemplate>
      </Setter.Value>
   …
```