Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások

---

11. előadás

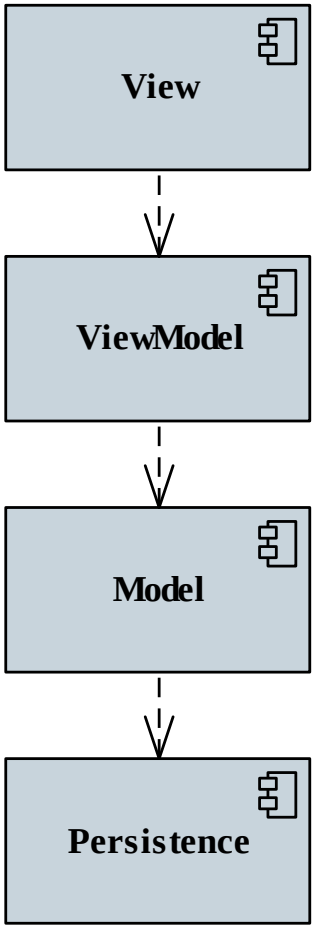
Összetett WPF alkalmazások

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

Összetett WPF alkalmazások

Az MVVM architektúra

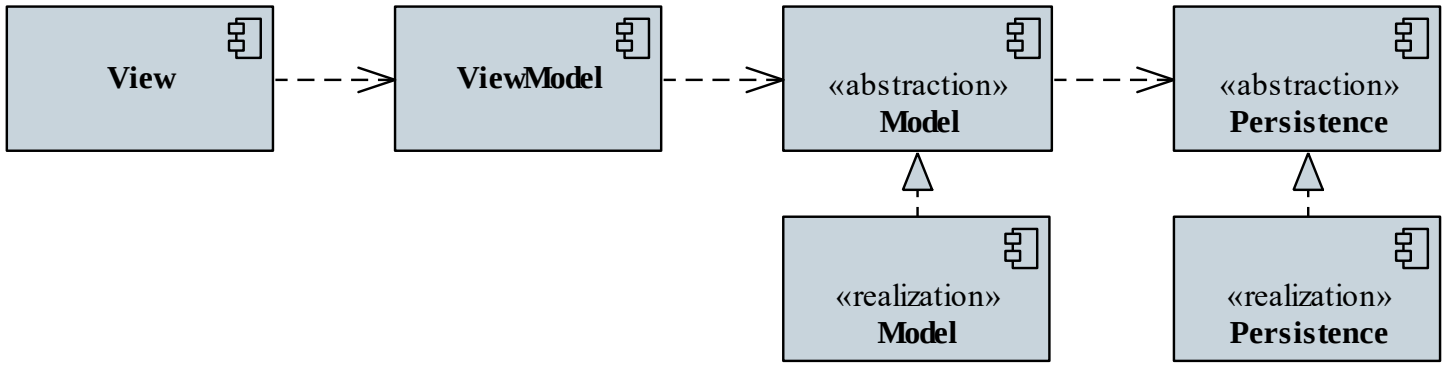
- Az MVVM architektúrában
  - a *nézet* tartalmazza a grafikus felületet és annak erőforrásait
  - a *nézetmodell* egy közvetítő réteg, lehetőséget ad a modell változásainak követésére és tevékenységek végrehajtására
  - a *modell* tartalmazza az alkalmazás logikáját
  - a *perzisztencia* a hosszú távú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

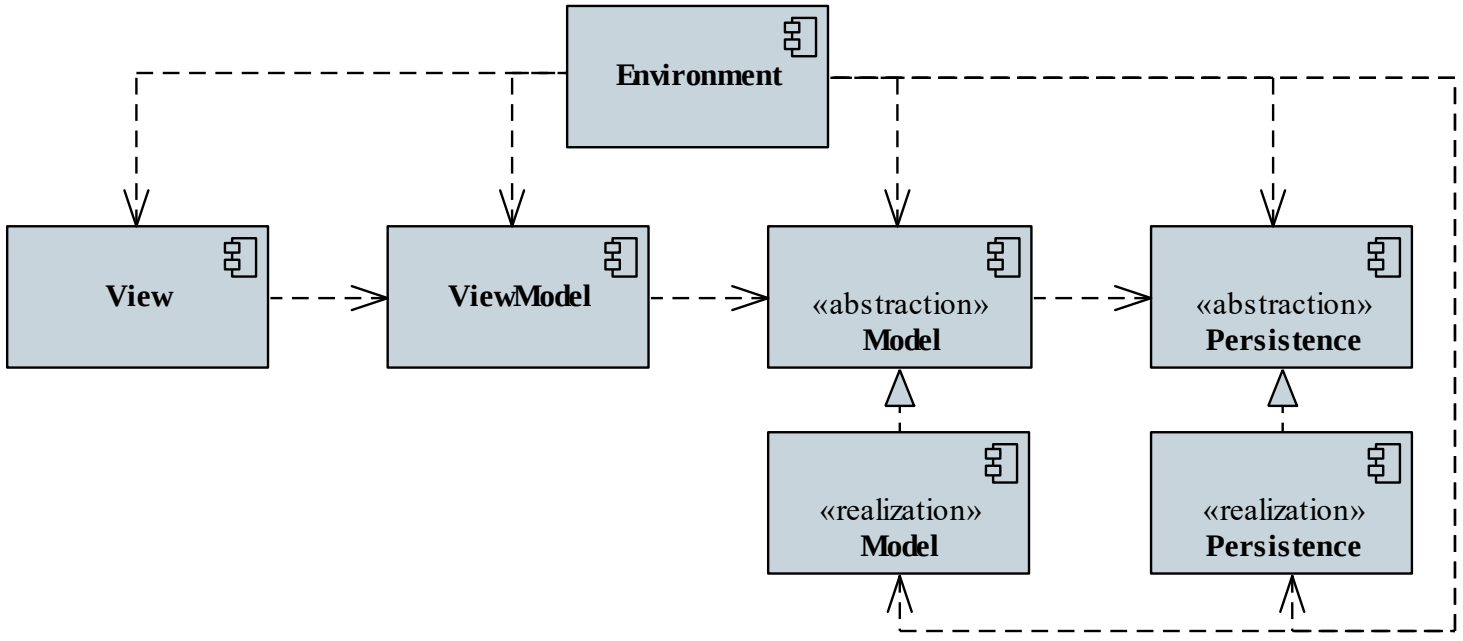
- a nézetmodellt a nézetbe egy tulajdonságon keresztül fecskendezzük be (*setter injection*)
- a modellt a nézetmodellbe, a perzisztenciát a modellbe konstruktoron keresztül helyezhetjük (*constructor injection*)

- 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

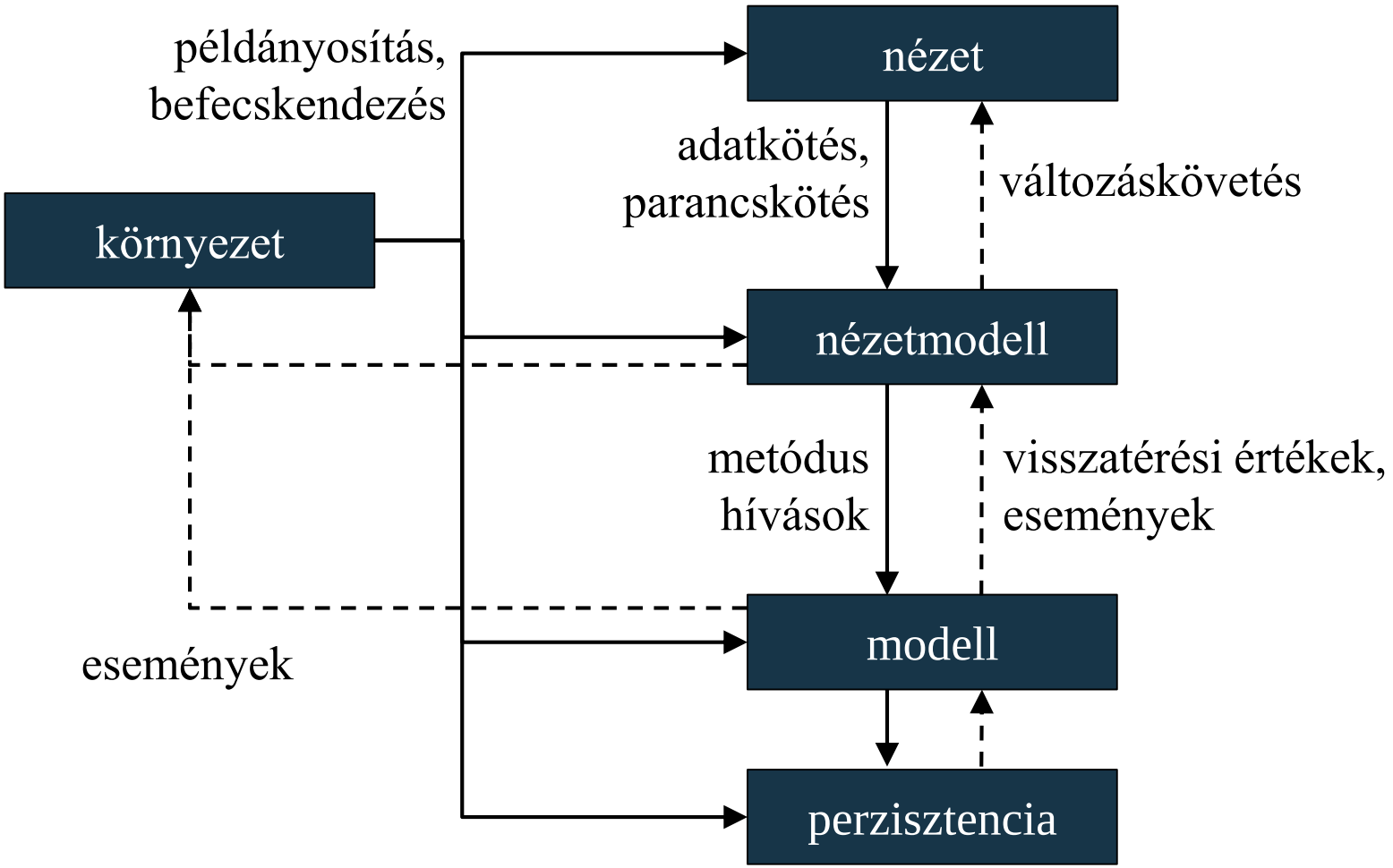
A környezet tevékenysége

- 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

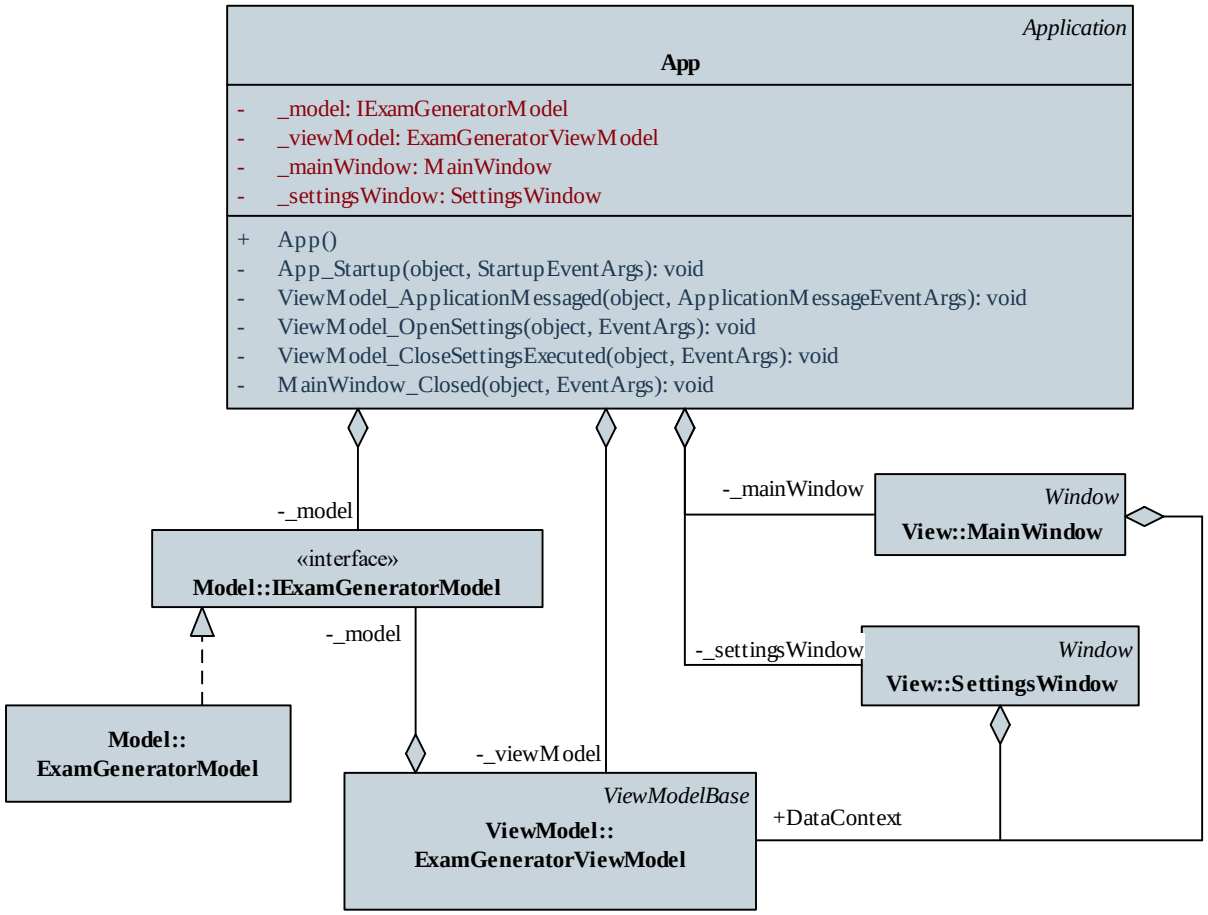
Példa

- 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

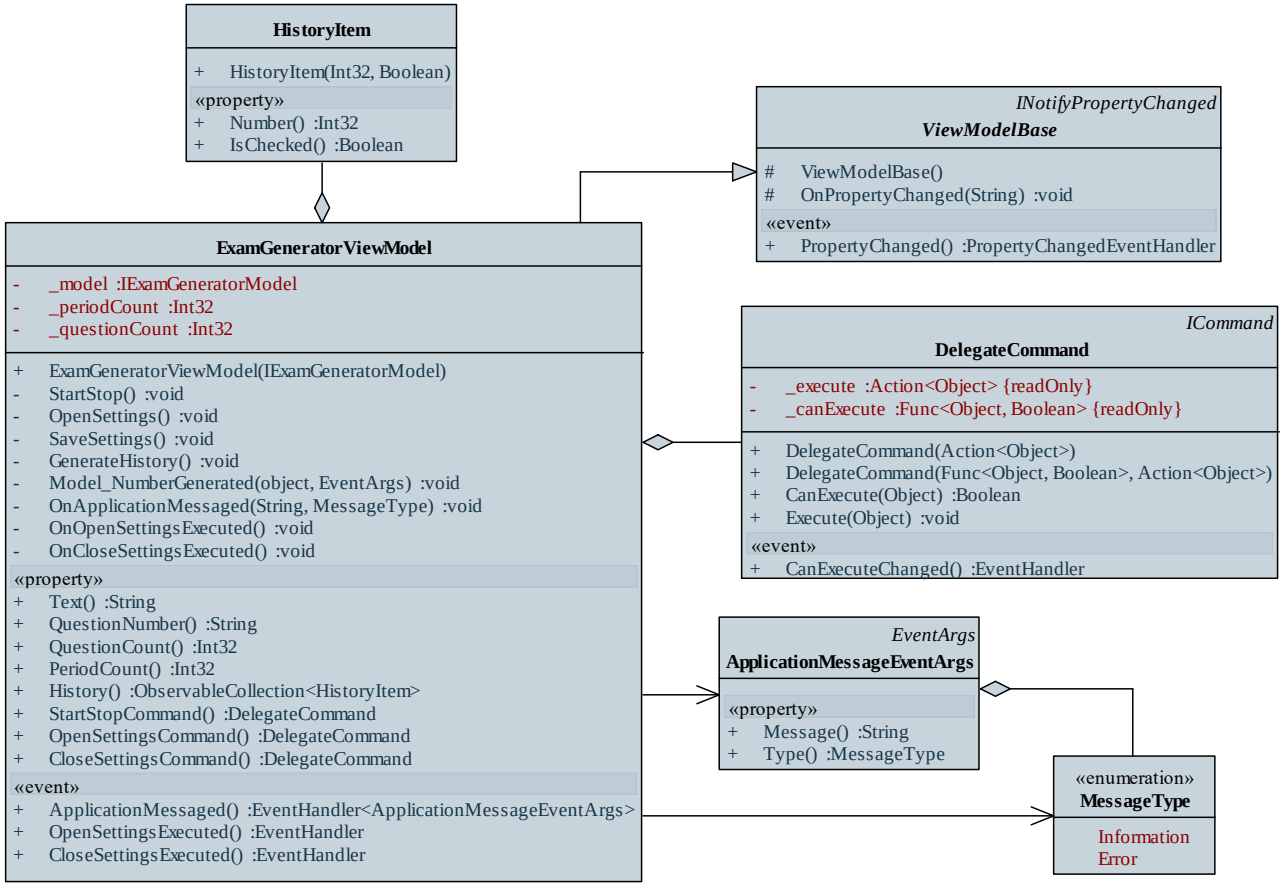
Tervezés:



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

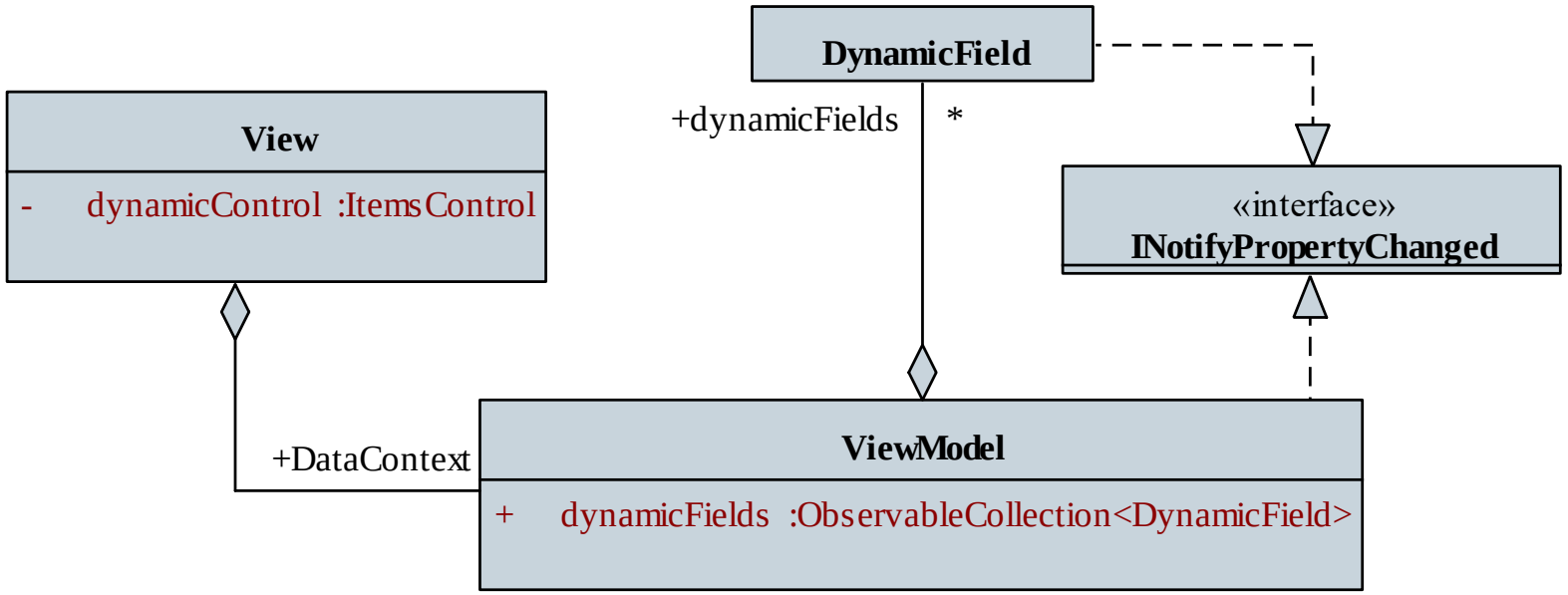
Dinamikus felhasználói felület

- Bár a WPF is lehetőséget ad vezérlők dinamikus létrehozására, az MVVM architektúra miatt speciális megközelítést igényel
  - a kódban 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 tulajdonságát (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

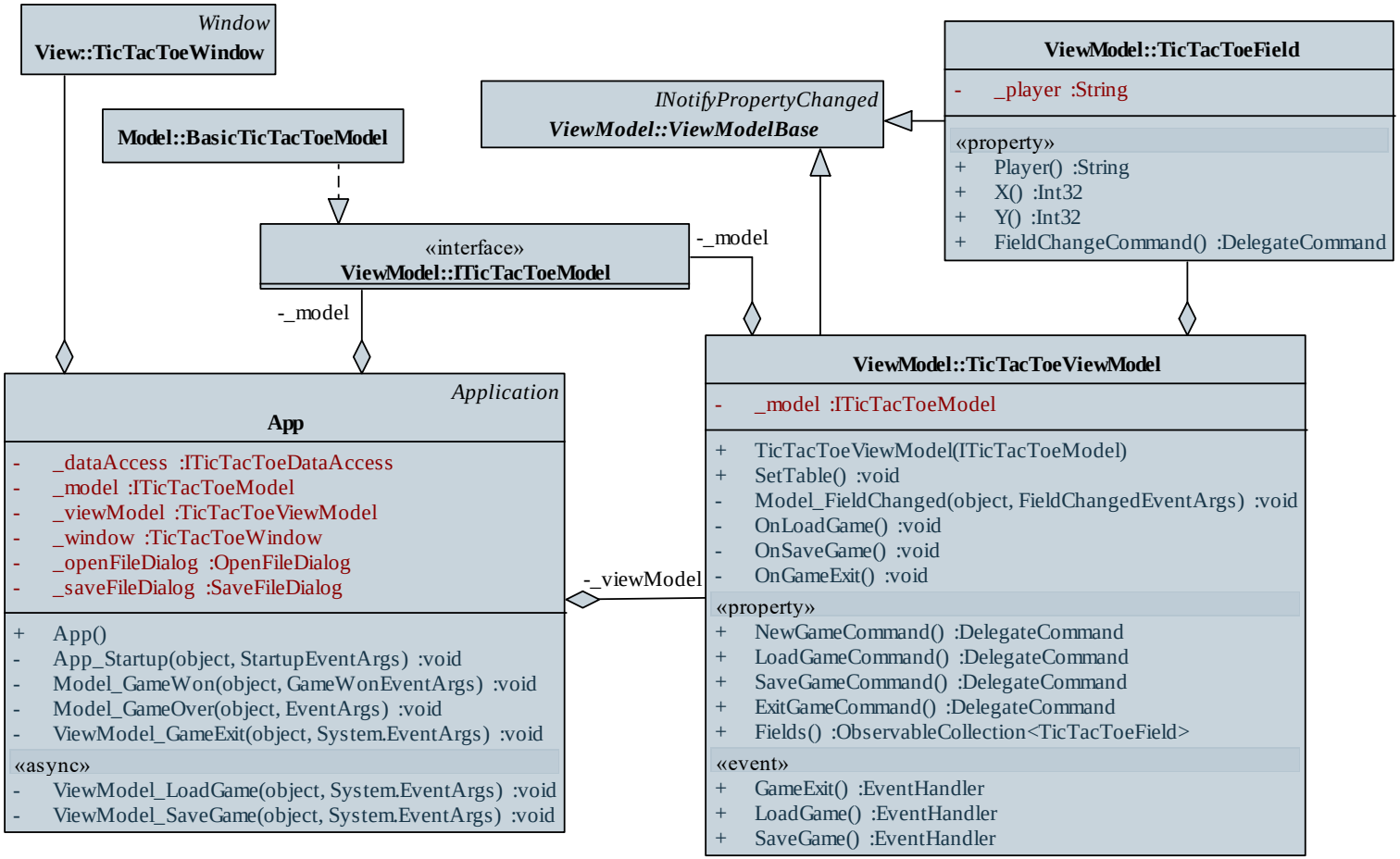
Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- 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
}
```