Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése II

11. előadás

Window Runtime specifikus alkalmazások megvalósítása

Giachetta Roberto

A jegyzet az ELTE Informatikai Karának
2014. évi Jegyzetpályázatának támogatásával készült

Windows Runtime specifikus alkalmazások megvalósítása

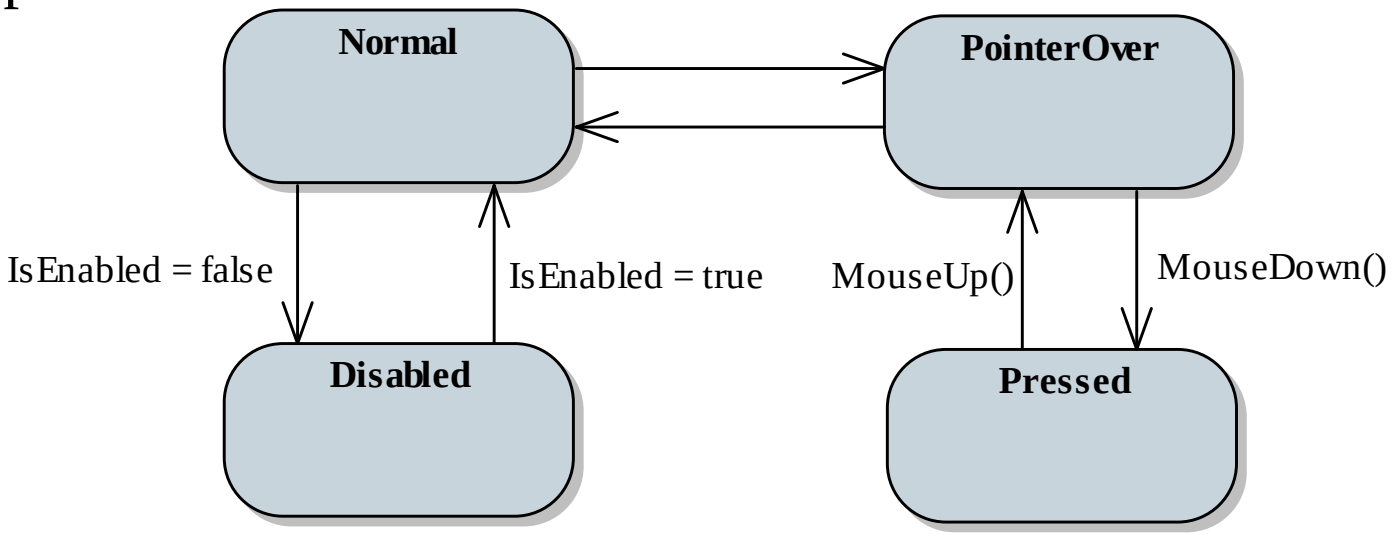
Alkalmazások megjelenése

- A Modern UI alapú alkalmazások megjelenése több speciális elemmel is rendelkezik
  - a felület *stílus*ának (színek, betűk) célszerű az egységes sémához alkalmazkodnia (*theme resource*)
  - a felületet jellemzően *animálódik*, a különböző átmenetekre szintén adott egy séma (*theme animation*)
  - a nézetnek kezelnie kell a különböző méreteket, tájolásokat, ezt egységes formában *vizuális állapot*ok (*visual state*) segítségével teheti meg
  - az egyes eseményekre, adatváltozásokra *viselkedés*ek (*behavior*) segítségével tudunk reagálni

Windows Runtime specifikus alkalmazások megvalósítása

Vizuális állapotok

- A *vizuális állapot*ok (**`VisualState`**) a vezérlők egyes állapotait tükrözik le, amelyek esemény hatására változnak
  - céljuk a vezérlők állapotkezelésének egyszerűsítése
  - állapotváltáskor lehetőségünk van animációk lejátszására
  - pl.:



Windows Runtime specifikus alkalmazások megvalósítása

Vizuális állapotok

- A vizuális állapotokat a vezérlőben a **`VisualStateManager`** osztály segítségével követhetjük
  - bármely vezérlőnek adhatunk állapotokat csoportokba foglalva (**`VisualStateGroups`**)
  - minden csoportnak és állapotnak egyedi névvel (**`x:Name`**) kell rendelkeznie, majd az állapotváltást a névre hivatkozva kezdeményezhetjük (**`GoToState`**)
  - a vezérlők eleve rendelkeznek állapotokkal, amelyeket felhasználhatunk
    - pl. a **`CommonStates`** csoport tartalmazza az általános állapotokat (**`Normal`**, **`Disabled`**, **`Pressed`**, …)

Windows Runtime specifikus alkalmazások megvalósítása

Vizuális állapotok

- az állapotok felhasználhatóak méretváltás kezelésére
- pl.:

```
<Grid …>
   <VisualStateManager.VisualStateGroups>
      <!-- vizuális állapotkezelés -->
      <VisualStateGroup x:Name="SizeStates">
         <VisualState x:Name="Normal" />
         <VisualState x:Name="Narrow">
            <Storyboard> … </Storyboard>
            <!– lefut egy animáció, amikor
                átváltunk az állapotba -->
         </VisualState>
      …
</Grid>
```

Windows Runtime specifikus alkalmazások megvalósítása

Vizuális állapotok

- pl.:

```
private void MainPage_SizeChanged(object sender,
                    SizeChangedEventArgs e) {
   if (e.NewSize.Width < 600) {
      VisualStateManager.GoToState(this,
         "Narrow", false);
      // ha lecsökken az oldalszélesség, a szűk
      // állapotra váltunk
   }
   else {
      VisualStateManager.GoToState(this,
         "Normal", false);
   }
}
```

Windows Runtime specifikus alkalmazások megvalósítása

Animációk

- Az alkalmazások megjelenését animációkkal gazdagíthatjuk
  - lehetőségünk van egyedi animációk készítésére (**`ColorAnimation`**, **`DoubleAnimation`**, ...)
  - az animációkat forgatókönyvbe (**`Storyboard`**) helyezzük
  - adott az animációknak egy kiinduló halmaza (*theme animation*), amely a Modern UI elvárásainak megfelelő animációkat biztosítja, pl.
    - elemek megjelenése (**`FadeInThemeAnimation`**, **`PopInThemeAnimation`**, **`AddDeleteThemeAnimation`**)
    - áthelyezés (**`RepositionThemeAnimation`**)
    - húzás (**`SwipeHintThemeAnimation`**)

Windows Runtime specifikus alkalmazások megvalósítása

Animációk

- pl.:

```
<Storyboard Storyboard.TargetName="myButton"
    <FadeInThemeAnimation />
    <!-- az animáció már tartalmazza a
         paramétereket -->
</Storyboard>
```

- lehetőségünk animációkat automatikusan futtatni átmenetek (*transition*) formájában, ekkor csupán az átmenetet kell megadnunk a vezérlőben, pl.:

```
<Button.Transitions>
    <FadeInTransition />
    <!-- automatikus animálódik -->
</Button.Transitions>
```

Windows Runtime specifikus alkalmazások megvalósítása

Példa

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

- felhasználjuk a vizuális állapotokat arra, hogy az alkalmazás kis szélesség esetén is jól látszódjon, ehhez
  - létrehozunk vizuális állapotokat mindkét ablakban (**`Normal`**, **`Narrow`**, **`VeryNarrow`**)
  - az állapotokban a címke méretét és a rács margóját állítjuk animációk segítségével
  - lekezeljük az oldalak méretváltozását (**`SizeChanged`**), és a változásnak megfelelően váltjuk az állapotot (**`GoToState`**)

Windows Runtime specifikus alkalmazások megvalósítása

Példa

Megvalósítás (`MainPage.xaml`*):*

```
<Grid Name="mainGrid"
      Style="{StaticResource NormalGridStyle}">
<!-- vizuális állapotokat használunk, hogy
     alkalmazkodjunk a mérethez -->
    <VisualStateManager.VisualStateGroups>
       <VisualStateGroup Name="SizeStates">
          <VisualState Name="Normal">
             <!-- normál állapot -->
                <Storyboard>…</Storyboard>
          </VisualState>
          <VisualState Name="Narrow">
          …
    </VisualStateManager.VisualStateGroups>
```

Windows Runtime specifikus alkalmazások megvalósítása

Viselkedések

- A nézet viselkedését jobban egyedivé szabhatjuk a *Behaviors SDK* segítségével
  - bármely elemhez hozzárendelhetünk viselkedéseket (**`Interaction.Behaviors`**)
  - a viselkedések kezdeményezhetőek esemény (**`EventTriggerBehavior`**), vagy adatváltozás (**`DataTriggerBehavior`**) hatására
  - a viselkedés lehet értékbeállítás (**`ChangePropertyAction`**), parancsfuttatás (**`InvokeCommandAction`**), állapotváltás (**`GoToStateAction`**), …
  - a használathoz két névtérre is szükségünk van

Windows Runtime specifikus alkalmazások megvalósítása

Viselkedések

- pl.:

```
xmlns:i="using:Microsoft.Xaml.Interactivity"
xmlns:icore="using:Microsoft.Xaml
                        .Interactions.Core"
…
<Button …> <!– viselkedést adunk a gombnak -->
  <i:Interaction.Behaviors>
    <icore:EventTriggerBehavior
          EventName="DoubleTapped">
      <icore:InvokeCommandAction Command="…" />
      <!-- parancs futtatása dupla
           kattintásra -->
  …
</Button
```

Windows Runtime specifikus alkalmazások megvalósítása

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 projektben meghivatkozzuk a *Behaviors SDK*-t (*Windows*/*Extensions*)
- a szín sorszámának változására egy **`DataTriggerBehavior`** reagál, amely egy **`ChangePropertyAction`** segítségével végzi el a módosítást

Windows Runtime specifikus alkalmazások megvalósítása

Példa

Megvalósítás (`MainPage.xaml`*):*

```
<Button Command="{Binding FieldChangeCommand}" …>
  <interactivity:Interaction.Behaviors>
    <interactions:DataTriggerBehavior
     Binding="{Binding ColorNumber}" Value="0">
      <!-- ha 0-ra váltott a ColorNumber
           tulajdonság, akkor zöldre váltjuk a
           színt -->
      <interactions:ChangePropertyAction
       PropertyName="Background">
        <interactions:ChangePropertyAction.Value>
          <SolidColorBrush Color="Green" />
        </interactions:ChangePropertyAction.Value>
      …
```

Windows Runtime specifikus alkalmazások megvalósítása

Beállítások megjelenítése

- Az alkalmazások beállításait a Windows 8 egységes módon jeleníti meg, külön beállítások képernyővel
  - minden beállítási lehetőséget célszerű áthelyezni oda, hogy egységesen kezelhetőek legyenek a felhasználók számára
  - a beállítások lapjait a beállítás-kezelő (**`SettingsPane`**) segítségével helyezhetjük ki
    - először fel kell vennünk a menüpontokat (**`SettingsCommand`**) a beállítások képernyőn (**`CommandsRequested`**)
    - a parancshoz kötött tevékenységben lehetőségünk van beállítás lapot (**`SettingFlyout`**) megjeleníteni
    - akár több lapot is használhatunk külön csoportoknak

Windows Runtime specifikus alkalmazások megvalósítása

Beállítások megjelenítése

- pl.:

```
SettingsPane.GetForCurrentView().
    CommandsRequested += …;
…
e.Request.ApplicationCommands.Add(
    new SettingsCommand("Settings",
                        "Alap beállítások",
        // megadjuk, hova helyezze a parancsot, és
        // milyen felirattal
        command => {
            settingsFlyout.Show();
            // beállítások lap megjelenítése
        }
));
```

Windows Runtime specifikus alkalmazások megvalósítása

Példa

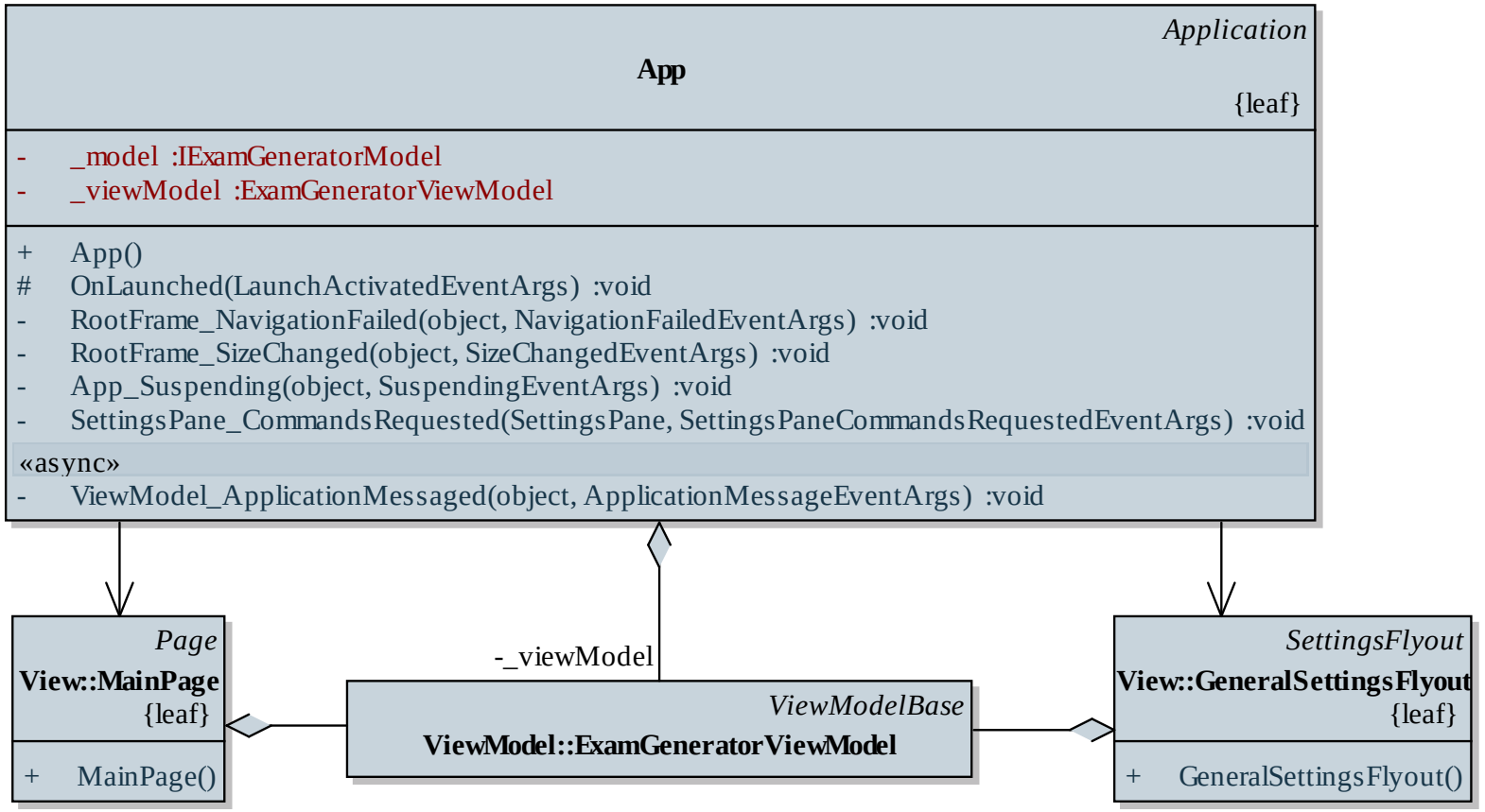
Feladat: Készítsünk egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- javítsunk a beállítások kezelésén, azáltal, hogy az alkalmazás beállításai közé helyezzük őket
- létrehozunk egy beállítás lapot (**`GeneralSettingsFlyout`**), amely tartalmazni fogja az összes beállítást
- az alkalmazásban (**`App`**) elvégezzük a megjelenítést, és az adattársítást
- a listának adunk egy új elemsablont annak érdekében, hogy ne változzon a háttérszín fókuszáláskor

Windows Runtime specifikus alkalmazások megvalósítása

Példa

Tervezés:



Windows Runtime specifikus alkalmazások megvalósítása

Példa

Megvalósítás (`App.xaml.cs`*):*

```
private void SettingsPane_CommandsRequested(…) {
   // felveszünk egy parancsot a beállításokhoz,
   // amely megjeleníti az ablakot
   e.Request.ApplicationCommands.Add(
     new SettingsCommand("Settings", "Általános",
       command => {
          GeneralSettingsFlyout settingsFlyout =
             new GeneralSettingsFlyout();
          settingsFlyout.DataContext = _viewModel;
          settingsFlyout.Show();
             // beállítások megjelenítése
       }));
}
```

Windows Runtime specifikus alkalmazások megvalósítása

Menü megvalósítása

- Az alkalmazások rendelkezhetnek *alkalmazássáv*val
  - két típusa adott, az általános (**`AppBar`**), amelyben bármilyen tartalmat elhelyezhetünk, és a parancssáv (**`CommandBar`**), amely szabványos utasításokat tud prezentálni
  - automatikusan megjelenik gesztus hatására, de programozottan is megjeleníthető (**`IsOpen`**, **`IsSticky`**)
  - elhelyezhető az ablak tetején, illetve alján (**`TopAppBar`**, **`BottomAppBar`**)
  - a sávra elsősorban gombokat helyezünk (**`AppBarButton`**, **`AppBarToggleButton`**), amely speciális megjelenéssel (**`Icon`**) rendelkeznek

Windows Runtime specifikus alkalmazások megvalósítása

Menü megvalósítása

- pl.:

```
<Page.BottomAppBar> <!-- alsó menü -->
  <CommandBar> <!-- gombok fogják alkotni -->
    <AppBarButton Label="Frissít" Icon="Refresh"
                  Command="…" />
    <AppBarButton Label="Segítség" Icon="Help"
                  Command="…" />
  </CommandBar>
</Page.BottomAppBar>
```

- lehetőségünk van speciális módon tagolni az elemeket (**`AppBarSeparator`**, **`SecondaryCommands`**)
- a gombokhoz rendelhetünk menüket is (**`MenuFlyout`**), amelyekre menüpontokat helyezhetünk (**`MenuFlyoutItem`**)

Windows Runtime specifikus alkalmazások megvalósítása

Adatkezelés

- Az alkalmazások adatait több formában is tárolhatjuk:
  - *lokálisan*: a helyi fájlrendszerben és regisztrációs adatbázisban (**`ApplicationData.Current.LocalSettings`**, **`ApplicationData.Current.LocalFolder`**)
  - *központilag* (roaming): a felhasználói fiókhoz társítva (**`ApplicationData.Current.RoamingSettings`**, **`ApplicationData.Current.RoamingFolder`**)
  - *átmenetileg*: futás közben elérhető tárhelyen (**`ApplicationData.Current.TemporaryFolder`**)
- Alkalmazás törlésekor a hozzá tartozó adatok is törlődnek

Windows Runtime specifikus alkalmazások megvalósítása

Adatkezelés

- Az alkalmazás beállításait kulcs/érték párok formájában tárolhatjuk az **`ApplicationDataContainer`** típuson keresztül
  - egyszerű adattípusok támogatottak, illetve több belső gyűjteményt is létrehozhatunk
  - pl.:

```
ApplicationDataContainer settings =
    ApplicationData.Current.LocalSettings;
    // helyi beállítások lekérdezése
settings.Values["myValue"] = 1000;
    // beállítás mentése
Int32 value =
    Convert.ToInt32(settings.Values["myValue"]);
    // beállítás lekérdezése
```

Windows Runtime specifikus alkalmazások megvalósítása

Adatkezelés

- Az alkalmazás fájljait a megadott könyvtáron keresztül (**`StorageFolder`**) kezelhetjük
  - létrehozhatunk (**`CreateFileAsync(…)`**), betölthetünk, törölhetünk fájlokat, illetve könyvtárakat
  - a fájlokat (**`StorageFile`**) olvashatjuk, illetve írhatjuk a **`FileIO`** osztály segítségével
    - szöveges, illetve bináris tartalmat tudunk kezelni (**`ReadTextAsync()`**, **`ReadLinesAsync()`**, **`ReadBufferAsync(…)`**, **`WriteTextAsync(…)`**)
  - a megszokott adatfolyam kezelés is elérhető (**`StreamReader`**, **`StreamWriter`**), azonban funkcionalitása korlátozott

Windows Runtime specifikus alkalmazások megvalósítása

Adatkezelés

- alapesetben az alkalmazás csak a lokális könyvtárát, a telepítés helyét, illetve a felhasználó letöltések könyvtárát láthatja, minden más engedélyhez kötött
- pl.:

```
StorageFolder folder =
    ApplicationData.Current.LocalFolder;
    // alkalmazás helyi könyvtárának lekérdezése
StorageFile file =
    await Folder.GetFileAsync("data.txt");
    // fájl (aszinkron) betöltése
IList<String> lines =
    await FileIO.ReadLinesAsync(file);
    // összes sor kiolvasása a fájlból
```

Windows Runtime specifikus alkalmazások megvalósítása

Adatkezelés

- Az adatkezelés fontos szerephez jut az állapot megőrzésében
  - mivel a felfüggesztett alkalmazást bármikor bezárhatja a rendszer, felfüggesztéskor (**`Suspending`** esemény) mentenünk kell az állapotot
  - aktiválásnál, amennyiben terminált állapotból (**`ApplicationExecutionState.Terminated`**) térünk vissza, be kell töltenünk a korábbi állapotot
    - ezt az **`OnLaunched(…)`** eseménykezelő argumentumában megkapjuk
  - állapot visszatöltést akkor is alkalmazhatunk, ha a felhasználó zárja be az alkalmazást (**`ApplicationExecutionState.ClosedByUser`**)

Windows Runtime specifikus alkalmazások megvalósítása

Adatkezelés

- Lehetőségünk van a teljes fájlrendszer elérésére is, amennyiben az alkalmazás erre kapott engedélyt (*Capabilities*)
  - betölthetünk könyvtárakat (**`FolderPicker`**), illetve fájlokat (**`FileOpenPicker`**), továbbá menthetünk fájlokat (**`FileSavePicker`**)
    - konfigurálható a megjelenítés módja (**`ViewMode`**), a kezdőkönyvtár (**`SuggestedStartLocation`**), illetve szűrhető a megjelenített tartalom (**`FileTypeFilter`**)
  - az engedély mellett (helyett) meghatározhatjuk a támogatott funkcionalitást és fájltípusokat is (*Declarations*)
    - nem minden könyvtár esetében szükséges

Windows Runtime specifikus alkalmazások megvalósítása

Adatkezelés

- pl.:

```
FileOpenPicker openPicker =
    new FileOpenPicker(); // fájl megnyitó lap
openPicker.SuggestedStartLocation =
    PickerLocationId.DocumentsLibrary;
    // a dokumentumok könyvtárban kezd
openPicker.FileTypeFilter.Add(".txt");
    // csak a txt kiterjesztésű fájlokat jeleníti
    // meg
StorageFile file =
    await openPicker.PickSingleFileAsync();
    // egy fájl megnyitása
```

Windows Runtime specifikus alkalmazások megvalósítása

Példa

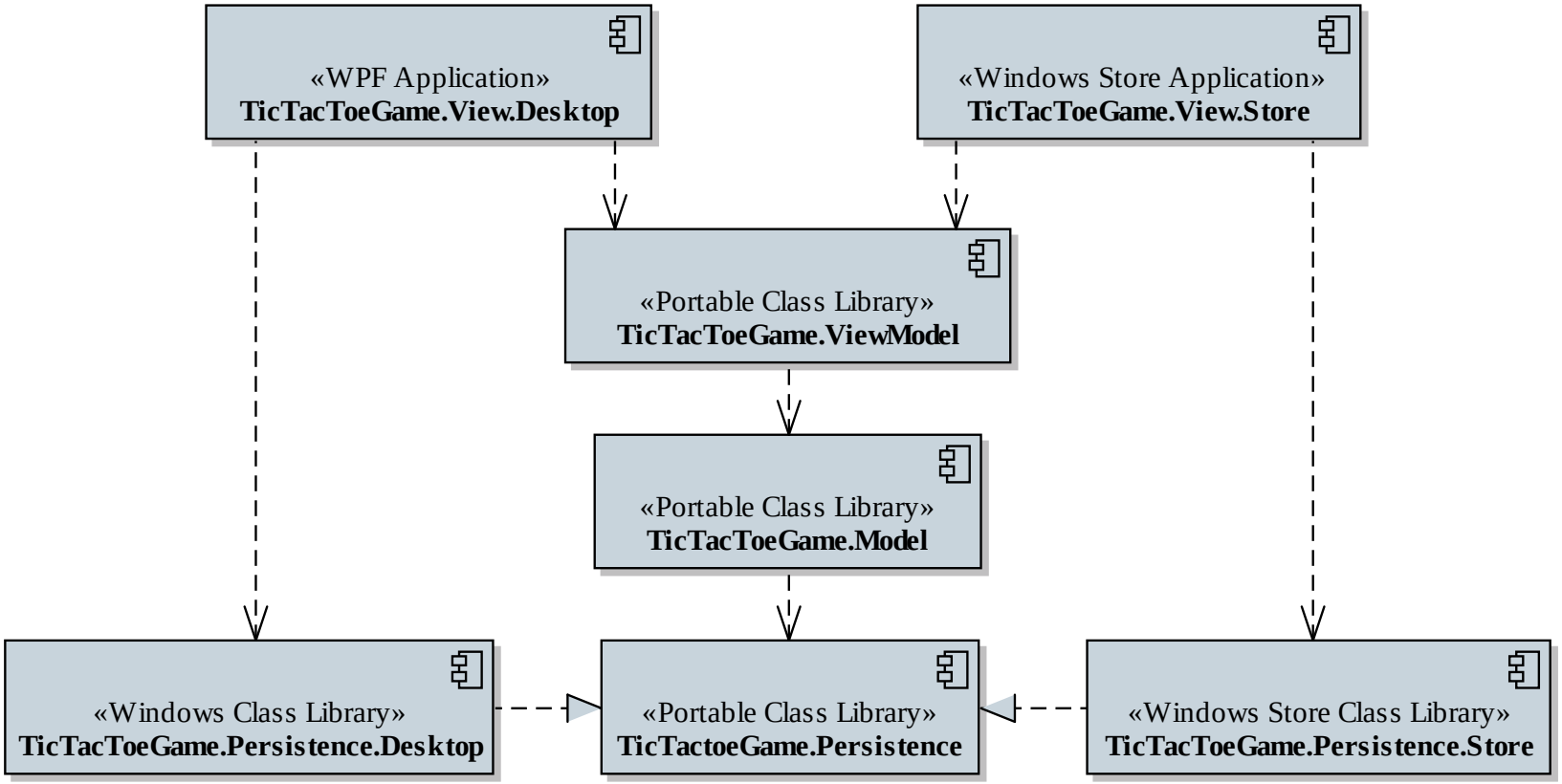
Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- az alkalmazás modellje, nézetmodellje és a perzisztencia felülete használható az asztali és a mobil alkalmazás esetén is, de *Portable Class Library* segítségével kell megvalósítanunk
- a perzisztencia megvalósítása különbözik a két alkalmazásban, ezért külön osztálykönyvtárként készítjük el mindkét esetre
- a nézetet szintén egy külön alkalmazásban valósítjuk meg, amelyben nem csak lehetőséget adunk fájlkezelésre, de automatikusan kezeljük az állapotot terminálás esetén

Windows Runtime specifikus alkalmazások megvalósítása

Példa

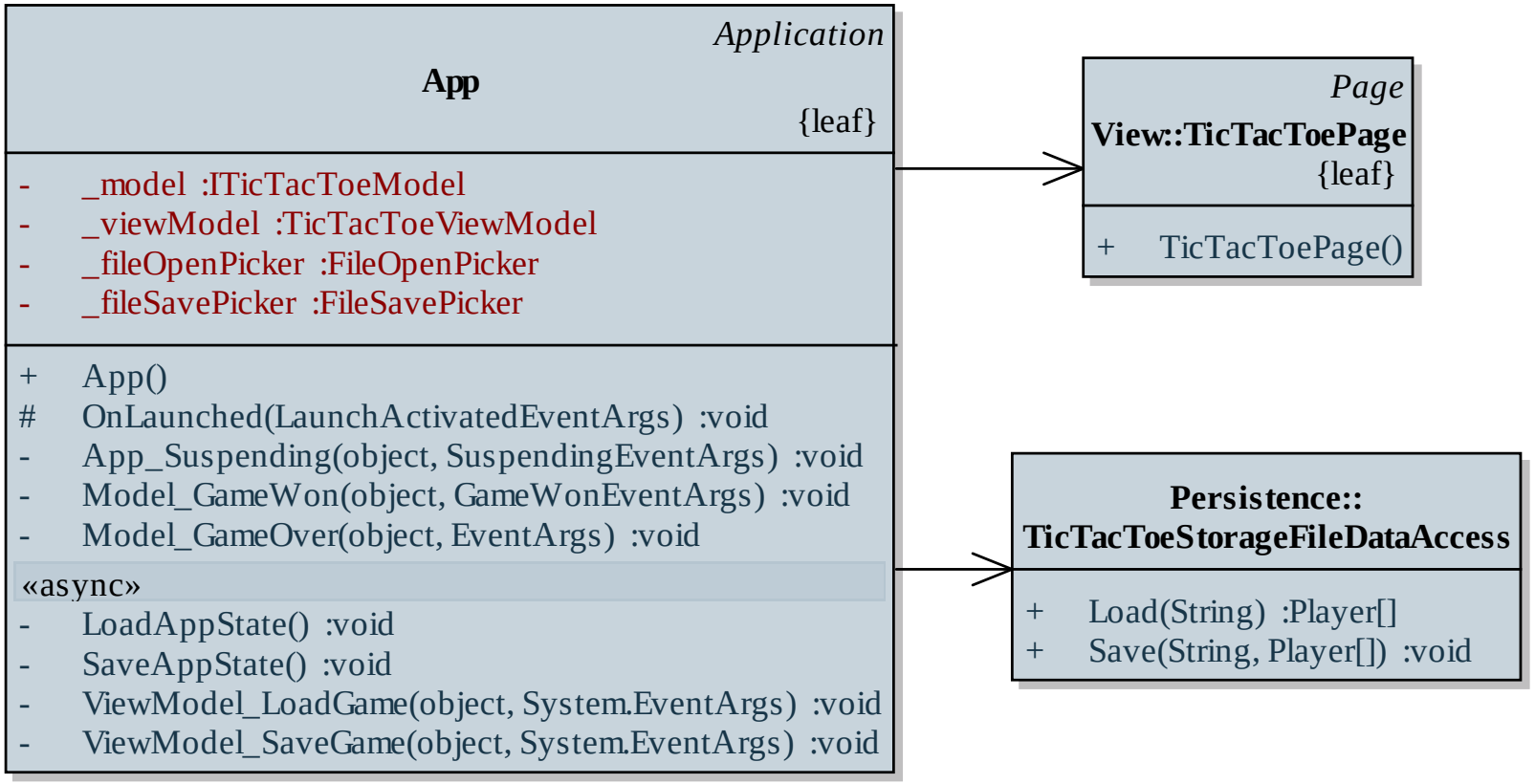
Tervezés:



Windows Runtime specifikus alkalmazások megvalósítása

Példa

Tervezés:



Windows Runtime specifikus alkalmazások megvalósítása

Példa

Megvalósítás (`App.xaml.cs`*):*

```
protected override void OnLaunched(…) {
    …
    // amennyiben nem a felhasználó zárta be az
    // alkalmazást, be kell töltenünk a korábbi
    // állapotot
    if (args.PreviousExecutionState ==
        ApplicationExecutionState.Terminated)
    {
        LoadAppState();
    }
}
```

Windows Runtime specifikus alkalmazások megvalósítása

Példa

Megvalósítás (`App.xaml.cs`*):*

```
private async void LoadAppState() {
   // a betöltést az alkalmazás könyvtárából
   // végezzük
   StorageFile file = await ApplicationData
            .Current.LocalFolder
            .GetFileAsync("lastgame.txt");
   …
   try {
      _model.LoadGame(file.Path);
   } catch { }
      // a hibát nem kell jeleznünk, csak
      // elveszítjük a korábbi állapotot
}
```

Windows Runtime specifikus alkalmazások megvalósítása

Példa

Megvalósítás (`TicTacToeStorageFileDataAccess.cs`*):*

```
public Player[] Load(String path)
    …
    StorageFile file = StorageFile
       .GetFileFromPathAsync(path).AsTask().Result;
       // betöltjük a fájlt a megadott útvonalból
    String[] numbers = FileIO.ReadTextAsync(file)
        .AsTask().Result.Split();
       // a fájl teljes tartalmát beolvassuk
       // egy szövegbe
    return numbers.Select(number =>
           (Player)Int32.Parse(number)).ToArray();
    …
}
```

Windows Runtime specifikus alkalmazások megvalósítása

Aszinkron tevékenységek megvalósítása

- Az aszinkron tevékenységek hasznosak, mivel lehetővé teszik a felület állandó hozzáférését
  - nem csupán szükségszerűen, de mind mobil, mind asztali környezetben célszerű az időigényes tevékenységeket (pl. adatkezelés) aszinkron módon megvalósítani
  - az aszinkron műveletek alapja a taszk (**`Task`**), amely biztosítja a párhuzamos futtatást
    - a művelet tulajdonképpen taszkkal tér vissza (**`void`** esetben is), amely tartalmazhat eredményt (**`Task<T>`**)
    - célszerű visszatérési típusként mindenhol a taszkot jelölni (pl. interfészben nem is használható az **`async`** kulcsszó)

Windows Runtime specifikus alkalmazások megvalósítása

Aszinkron tevékenységek megvalósítása

- pl.:

```
interface IMyInterface {
   Task MyMethodAsync(); // aszinkron művelet
}
…
class MyType : IMyInterface {
   public async Task MyMethodAsync() { }
   // aszinkron művelet megvalósítása
}
…
IMyInterface mt = new MyType();
await mt.MyMethodAsync();
   // megvárható a művelet
```

Windows Runtime specifikus alkalmazások megvalósítása

Példa

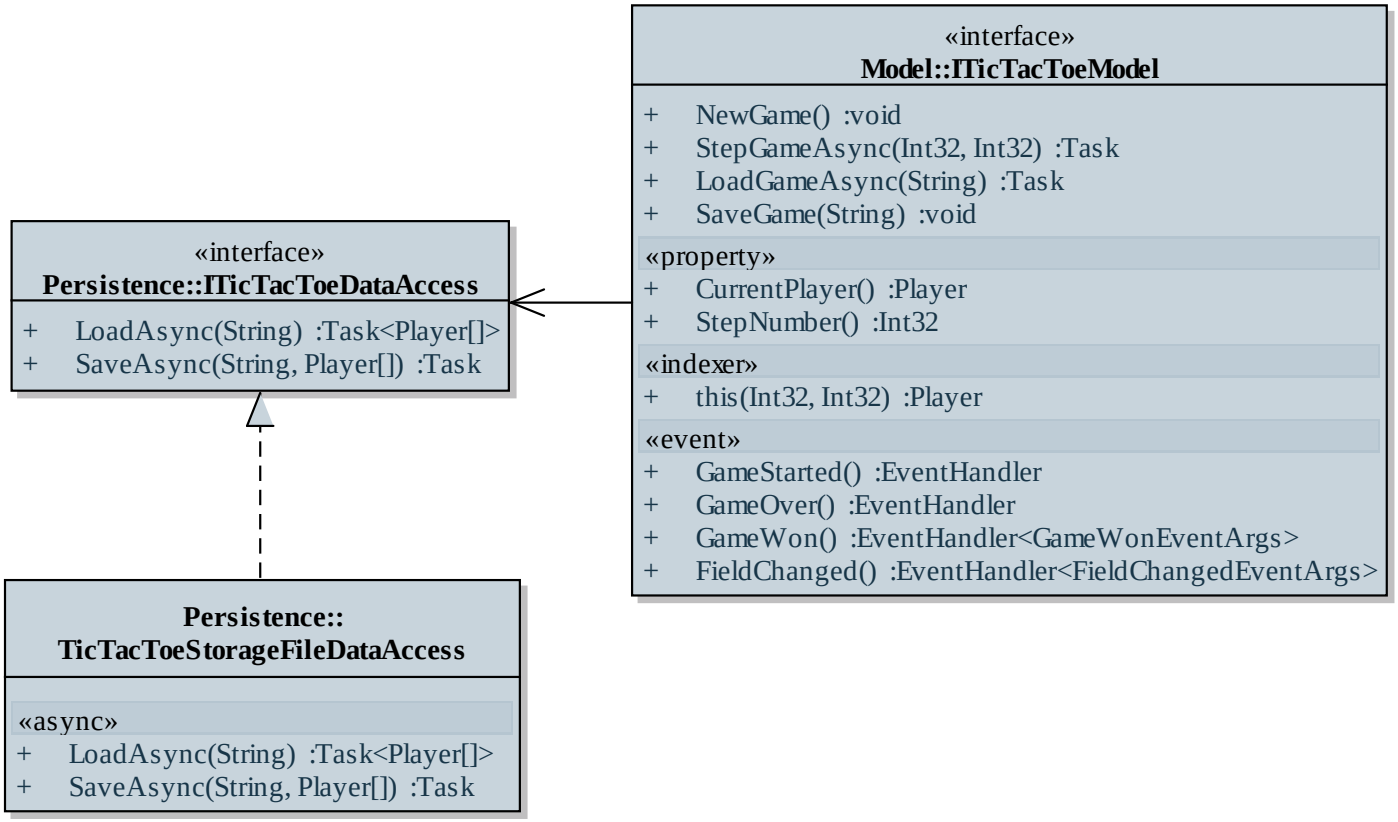
Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- hatékonysági okokból valósítsuk meg aszinkron módon a teljes fájlkezelést, így
  - az **`ITicTacToeDataAccess`** interfész **`Load`** és **`Save`** műveletei taszkkal térnek vissza
  - az **`ITicTacToeModel`** interfésze **`LoadGame`** és **`SaveGame`** műveletei is taszkkal térnek vissza
  - minden esetben a megvalósításban aszinkron műveleteket készítünk, és aszinkron műveleteket hívunk,
  - ennek megfelelően minden felhasználáskor bevárjuk az eredményt

Windows Runtime specifikus alkalmazások megvalósítása

Példa

Tervezés:



Windows Runtime specifikus alkalmazások megvalósítása

Példa

Megvalósítás (`TicTacToeStorageFileDataAccess.cs`*):*

```
public async Task<Player[]> Load(String path)
    …
    StorageFile file = await StorageFile
       .GetFileFromPathAsync(path);
       // betöltjük a fájlt a megadott útvonalból
    String[] numbers =
       (await FileIO.ReadTextAsync(file)).Split();
       // a fájl teljes tartalmát beolvassuk
       // egy szövegbe
    return numbers.Select(number =>
          (Player)Int32.Parse(number)).ToArray();
    …
}
```