Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése II

11. előadás

Platformspecifikus Xamarin alkalmazások

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

Készült Giachetta Roberto jegyzete alapján
http://www.inf.elte.hu/karunkrol/digitkonyv/

Platformspecifikus Xamarin alkalmazások

Alkalmazások környezete

- Az alkalmazások egy biztonságos környezetben futnak
  - nem férhetnek hozzá más alkalmazások adataihoz
  - csak korlátozott módon férhetnek hozzá a rendszer adataihoz (pl. fájlrendszer), és azt is csak engedéllyel
  - szintén engedéllyel használhatják csak az eszközöket (*application manifest*)

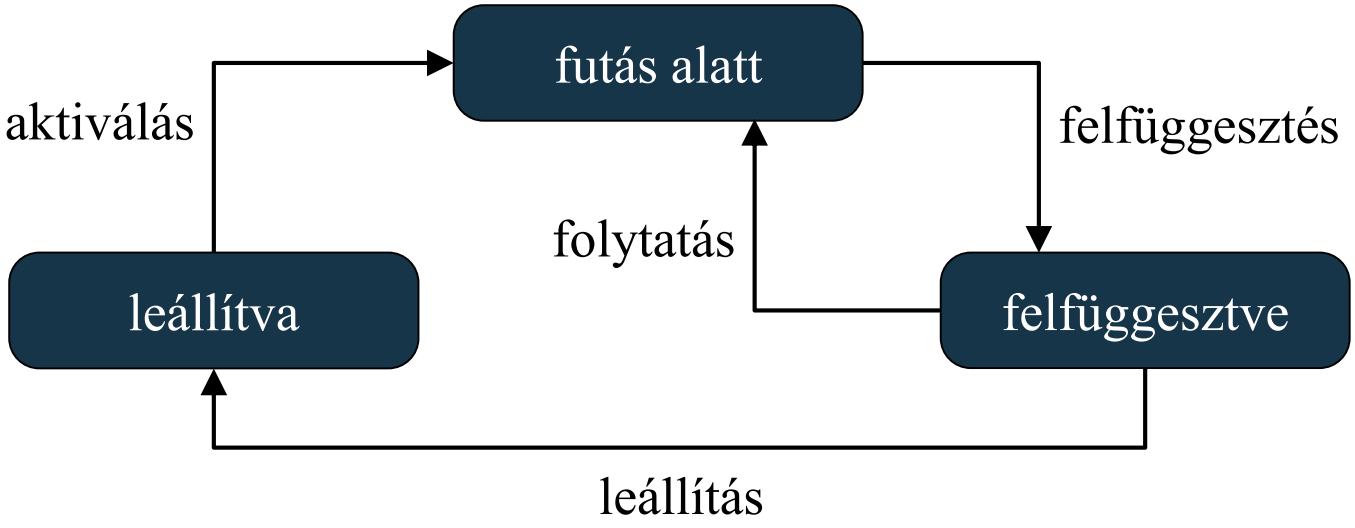
- Mindegyik alkalmazás számára rendelkezésre áll
  - egy beállítás/tulajdonság tároló, amelyben kulcs/érték párokat helyezhet el (a kulcs *string*)
  - egy lokális könyvtár, amely csak az alkalmazás fájljait tárolja

- Alkalmazás törlésekor a hozzá tartozó adatok is törlődnek

Platformspecifikus Xamarin alkalmazások

Alkalmazások életciklusa

- A mobil alkalmazások más életciklusban futnak, mint az asztali alkalmazások
  - a *futás alatt (running)* és a *terminált (not running)* állapotok mellett megjelenik a *felfüggesztett* (*suspended*) állapot is, amely akkor lép életbe, ha az alkalmazás a háttérbe (vagy a gép alvó állapotba) kerül, célja a takarékosság



Platformspecifikus Xamarin alkalmazások

Alkalmazások életciklusa

- A felfüggesztés célja az erőforrásokkal való takarékoskodás
  - a fejlesztőnek törekednie kell rá, hogy felfüggesztett állapotban az alkalmazás minél kevesebb erőforrást igényeljen
  - a futó tevékenységeket célszerű leállítani, az adatokat perzisztálni
- A rendszer úgy is dönthet (pl. ha kevés a memória), hogy a felfüggesztett alkalmazást leállítja, majd újraindítja, ha a felhasználó visszaváltott rá
  - célszerű, hogy a felhasználó ennek ellenére olyan állapotban kapja vissza az alkalmazást, amelyben hagyta, így ezt az állapotot vissza kell állítanunk
  - az állapot eltárolását felfüggesztéskor kell elvégeznünk

Platformspecifikus Xamarin alkalmazások

Alkalmazások életciklusa

- A Xamarin Forms alkalmazások egységes életciklus kezeléssel rendelkeznek
  - az alkalmazás (**`App`**) tartalmaz metódusokat indításkor (**`OnStart`**), felfüggesztéskor (**`OnSleep`**) és folytatáskor (**`OnResume`**) futtatandó tevékenységek végrehajtására
  - emellett az egyes platformokon külön is kezelhetjük az életciklust
  - a perzisztálás történhet
    - az alkalmazás beállításai/tulajdonságai közé, amelyet a rendszer automatikusan tárol (**`Application.Properties`**)
    - a fájlrendszerbe, vagy adatbázisba (pl. SQLite)

Platformspecifikus Xamarin alkalmazások

Alkalmazások életciklusa

- Pl.:

```
public class App : Application {
    protected override void OnSleep() {
        // felfüggesztés
        Properties["state"] = _data;
        // állapot elmentése a tulajdonság közé
        _data = null; // memória kiürítése
    }
    protected override void OnResume() {
        // folytatás
        if (Properties.ContainsKey("state"))
            _data = Properties["state"];
        // ha van elmentett állapot, visszatöltjük
    }
    …
```

Platformspecifikus Xamarin alkalmazások

Példa

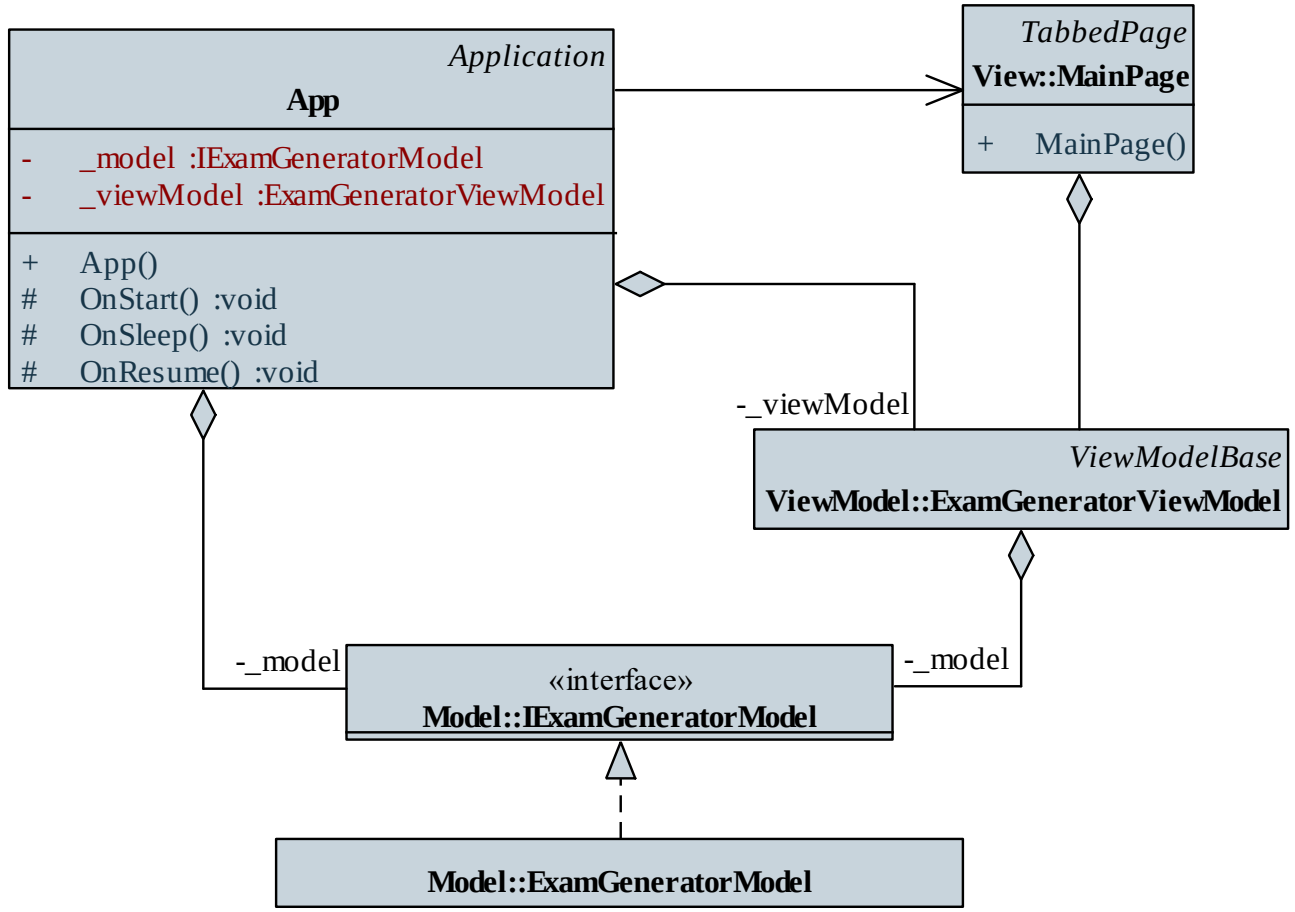
Feladat: Készítsünk egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- kiegészítjük életciklus kezeléssel az alkalmazást, eltároljuk a modell állapotát, valamint a generálás állapotát az alkalmazás beállításai (`Properties`) közé
- mivel nincs külön perzisztencia, közvetlenül a modelltől kérjük el az információkat, és tároljuk el egyenként
- indításkor, illetve folytatáskor a korábbi állapotot betöltjük, és ennek megfelelően inicializáljuk a modellt
- a hatékony erőforrás gazdálkodás érdekében felfüggesztéskor felszabadítjuk a modellt és nézetmodellt is

Platformspecifikus Xamarin alkalmazások

Példa

Tervezés:



Platformspecifikus Xamarin alkalmazások

Példa

Megvalósítás (`App.cs`*):*

```
protected override void OnSleep() {
    Properties["questionCount"] =
        _model.QuestionCount;
        // elmentjük a tételek számát
    Properties["periodCount"] = _model.PeriodCount;


    …
    _model = null;
    _viewModel = null;
    MainPage.BindingContext = null;
    // a szemétgyűjtő kitörli a memóriából a
    // modellt és a nézetmodellt
}
```

Platformspecifikus Xamarin alkalmazások

Példa

*Megvalósítás (**App.cs**):*

```
protected override void OnResume() {
    // ha el lett mentve az állapot, akkor
    // visszatöltjük
    if (Properties.ContainsKey("questionCount"))
    {
        _model = new ExamGeneratorModel(
            (Int32)Properties["questionCount"],
            (Int32)Properties["periodCount"]);
        // az elmentett adatokkal példányosítjuk a
        // modellt
    }
    …
}
```

Platformspecifikus Xamarin alkalmazások

Perzisztencia

- A Xamarin Forms nem biztosít egységes megoldást a perzisztenciára, mivel a megvalósítás platformonként jelentősen eltérhet
  - a perzisztencia platformonként történő megvalósítását a függőség befecskendezés teszi lehetővé
  - a hasonló platformfüggő függőségek kezelésének megkönnyítésére egy egységes módszer biztosított (**`DependencyService`**)
    - a függőségek megvalósítását megjelölhetjük (**`Dependency`** attribútum), így a rendszer automatikusan regisztrálja és betölti őket
    - a függőségeket az interfészen keresztül betölthetjük (**`DependencyService.Get`**)

Platformspecifikus Xamarin alkalmazások

Perzisztencia

- pl.:

```
// Android platformkód

[assembly: Dependency(typeof(DroidPersistence))]
   // megjelöljük a függőséget
namespace App.Droid
{
   public DroidPersistence : IPersistence
   {
      public void SaveData(String data)
      {
         … // perzisztencia megvalósítása
      }
      …
```

Platformspecifikus Xamarin alkalmazások

Perzisztencia

- pl.:

```
// Xamarin Forms kód
public interface IPersistence { … }
   // perzisztencia interfésze


…
IPersistence persistence =
   DependencyService.Get<IPersistence>();
   // megtalálja a perzisztencia megvalósítását
   // az adott platformon

IModel model = new Model(persistence);
   // így most már befecskendezhető
…
```

Platformspecifikus Xamarin alkalmazások

Perzisztencia Android plarformon

- Az Android (és iOS) platform(ok) a fájlkezelés összes lehetőségét biztosítják
  - könyvtárak (**`Directory`**) elérését, listázását (**`GetFiles`**, **`GetDirectories`**), benne könyvtárak létrehozását (**`CreateDirectory`**), fájlok és könyvtárak törlését (**`Delete`**)
  - fájlok (**`File`**) létrehozását, megnyitását (**`Open`**), olvasást (**`ReadAllBytes`**, **`ReadAllLines`**, **`ReadAllText`**), írást (**`WriteAllBytes`**, …), másolást (**`Copy`**), …
  - az írás és olvasás történhet adatfolyamok segítségével is (**`StreamReader`**, **`StreamWriter`**), amelyeket a megszokott módon használhatunk

Platformspecifikus Xamarin alkalmazások

Perzisztencia Android plarformon

- Android platformon a perzisztálás történhet:
  - az alkalmazás helyi könyvtárába (**`Environment.SpecialFolder.ApplicationData`**)
  - a felhasználó személyes könyvtárába (**`SpecialFolder.Personal`**), vagy tematikus könyvtárba (**`SpecialFolder.MyMusic`**, **`SpecialFolder.MyPictures`**)
  - az összes felhasználó által elérhető alkalmazás könyvtárba (**`SpecialFolder.LocalApplicationData`**)
- Az utak kezelését a **`Path`** osztály, speciális könyvtárak elérését az **`Environment.GetFolderPath`** metódus biztosítja

Platformspecifikus Xamarin alkalmazások

Perzisztencia Android plarformon

- pl.:

```
String documentsPath =
    Environment.GetFolderPath
        (Environment.SpecialFolder.Personal);
    // felhasználó könyvtárának lekérése

String filePath =
    Path.Combine(documentsPath, filename);
    // fájl elérési útvonalának létrehozása

File.WriteAllText(filePath, text);
    // szöveg kiírása a fájlba
```

Platformspecifikus Xamarin alkalmazások

Perzisztencia Windows platformon

- Windows platformon az alkalmazás fájljait a megadott könyvtáron keresztül (**`StorageFolder`**) kezelhetjük
  - listázhatunk (**`GetFilesAsync`**) létrehozhatunk (**`CreateFileAsync`**), betölthetünk (**`OpenAsync`**), törölhetünk (**`DeleteAsync`**) fájlokat, illetve könyvtárakat
  - a könyvtárakban fájlok (**`StorageFile`**) helyezkednek el, amelyeknek elérhetjük a tulajdonságait (**`Name`**, **`FileType`**, **`Properties`**, …)
  - a fájlokat a **`FileIO`** osztályon keresztül írhatjuk/olvashatjuk
    - szöveges, illetve bináris tartalmat tudunk kezelni (**`ReadTextAsync()`**, **`ReadLinesAsync()`**, **`ReadBufferAsync(…)`**, **`WriteTextAsync(…)`**)

Platformspecifikus Xamarin alkalmazások

Perzisztencia Windows platformon

- a megszokott adatfolyam kezelés is elérhető (**`StreamReader`**, **`StreamWriter`**), azonban funkcionalitása korlátozott

- Windows platform a perzisztálás történhet
  - lokálisan, a helyi fájlrendszerben (**`ApplicationData.Current.LocalFolder`**)
  - központilag (roaming), a felhasználói fiókhoz társítva (**`ApplicationData.Current.RoamingFolder`**), amelyek a felhasználó valamennyi eszközén elérhetőek
  - átmenetileg, futás közben elérhető tárhelyen (**`ApplicationData.Current.TemporaryFolder`**)

Platformspecifikus Xamarin alkalmazások

Perzisztencia Windows platformon

- 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
```

Platformspecifikus Xamarin alkalmazások

Példa

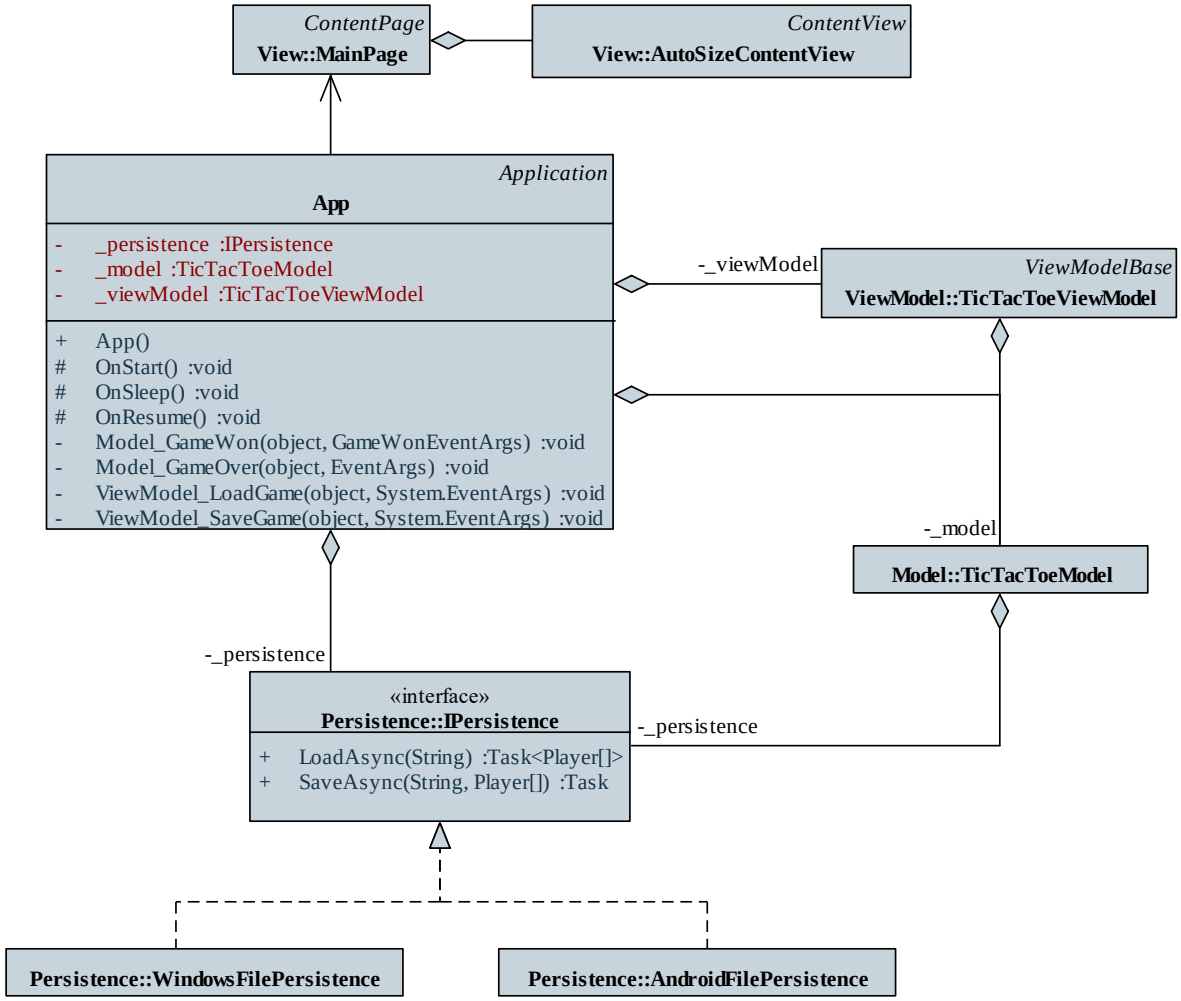
Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- elkésztjük a nézet Xamarin Forms megvalósítását, amelyben képekkel (**`Image`**) jelenítjük meg a mező tartalmát egy rácsban (**`FlowListView`**), a képeket csatoljuk a platformprojektekhez (**`circle.png`**, **`cross.png`**, **`empty.png`**)
- megvalósítjuk a perzisztenciát Android (**`AndroidFilePersistence`**) és Windows (**`WindowsFilePersistence`**) platformokra, amelyeket függőségként kezelünk
- a mentést a főprogram hajtja végre egy közös fájlba (így mindig csak az utolsó állapotot tudjuk menteni), valamint felfüggesztés esetén is mentjük az aktuális játékállapotot

Platformspecifikus Xamarin alkalmazások

Példa

Tervezés:



Platformspecifikus Xamarin alkalmazások

Példa

*Megvalósítás (**App.cs**):*

```
protected override void OnSleep()
{
    // elmentjük a jelenleg folyó játékot
    try
    {
        Task.Run(() => await
        _model.SaveGameAsync("SuspendedGame"));
       // az aszinkron tevékenységet taszkban
       // futtatjuk
    }
    catch { }
}
```

Platformspecifikus Xamarin alkalmazások

Példa

*Megvalósítás (***AndroidFilePersistence.cs***):*

```
public async Task<Player[]> LoadAsync(String path)
{
    // a betöltés a személyes könyvtárból történik
    String filePath =
      Path.Combine(Environment.GetFolderPath(
        Environment.SpecialFolder.Personal), path);

    // a fájlműveletet taszk segítségével végezzük
    String text = await Task.Run(() =>
            File.ReadAllText(filePath));
    return text.Split().Select(number =>
            (Player)Int32.Parse(number)).ToArray();
}
```

Platformspecifikus Xamarin alkalmazások

Példa

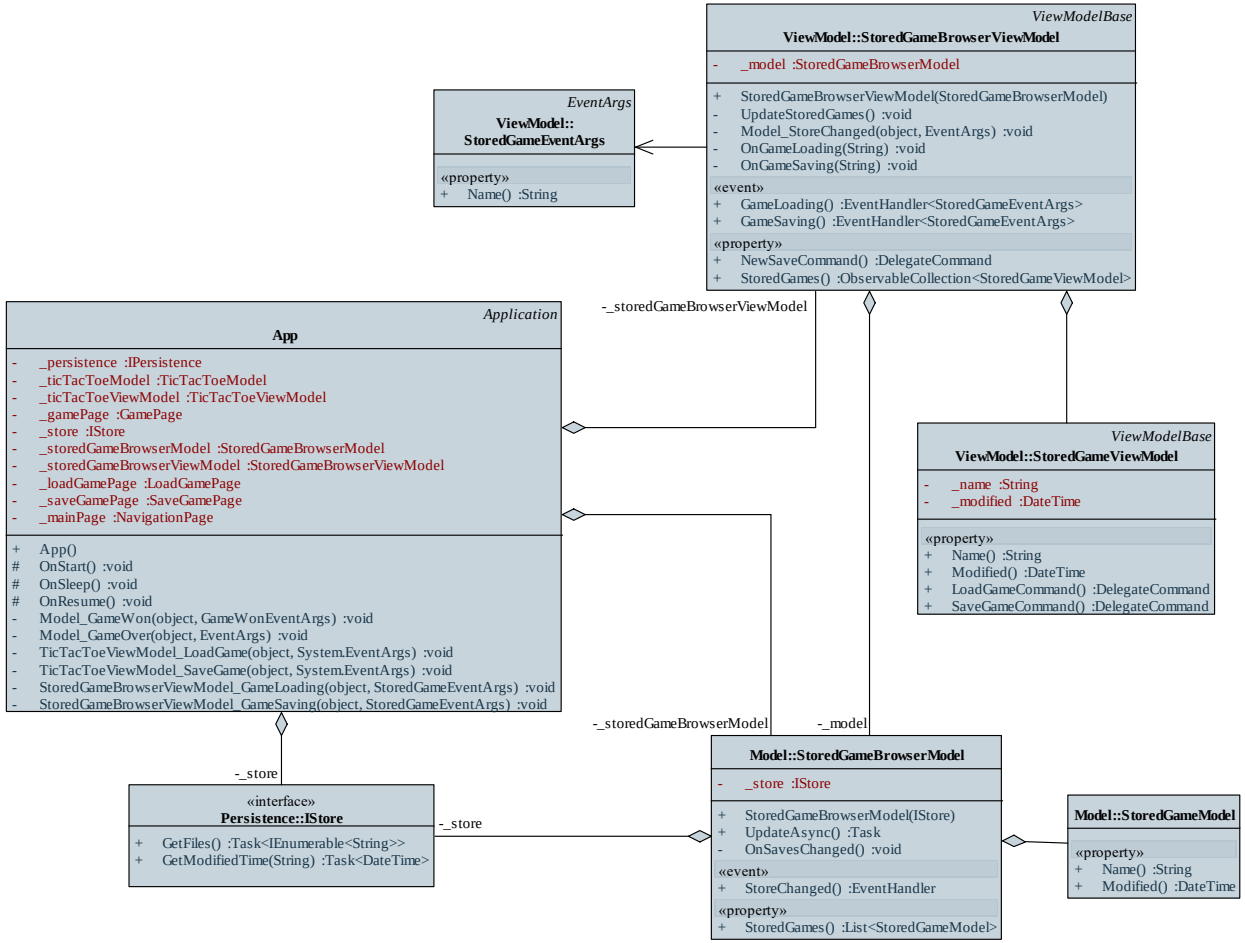
Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- lehetőséget adunk a felhasználónak a betöltött/mentett fájl kiválasztására két új oldal segítségével (**`LoadGamePage`**, **`SaveGamePage`**), amit navigáció (**`NavigationPage`**) segítségével kapcsolunk a főoldalhoz
- a nézetek mellett meg kell valósítanunk a fájlböngészés
  - perziszenciáját (**`IStore`**), amely beolvassa a könyvtárak tartalmát (platformonként eltérő módon)
  - modelljét (**`StoredGameBrowserModel`**), amely kezeli a játékok adatait
  - nézetmodelljét (**`StoredGameBrowserViewModel`**), amely biztosítja a frissítést és a parancsok végrehajtását

Platformspecifikus Xamarin alkalmazások

Példa

Tervezés:



Platformspecifikus Xamarin alkalmazások

Viselkedések

- Lehetőségünk van a grafikus felület működését deklaratív módon bővíteni viselkedések (`Behavior`) segítségével
  - a viselkedés egy olyan típus, amely adott vezérlőhöz biztosít hozzáférést, és adott típusú vezérlőhöz csatlakoztatható, további tevékenységeket
    - pl. tulajdonságok megváltoztatása, állapot ellenőrzése
  - megadhatunk vezérlő csatlakoztatásakor (`OnAttachedTo`) és lecsatlakoztatásakor (`OnDetachingFrom`) végrehajtható tevékenységet
    - általában egy eseménykezelőt társítunk, amely egészen a lecsatlakoztatásig végrehajtódik
    - a metódusokban mindig meg kell hívnunk az ős metódusát

Platformspecifikus Xamarin alkalmazások

Viselkedések

- pl.:

```
public class NumericValidationBehavior :
    Behavior<Entry>
    // egy viselkedés beviteli mezőkre
{
    protected override void OnAttachedTo(
        Entry entry)
    {
        entry.TextChanged += OnEntryTextChanged;
        // eseménykezelő hozzárendelése szöveg
        // megváltoztatásakor
        base.OnAttachedTo(entry);
    }
    … // az eseménykezelő validálja a tartalmat
```

Platformspecifikus Xamarin alkalmazások

Viselkedések

- a viselkedések XAML kódban hasznosíthatóak a vezérlőkre példányosítva (**`Behaviors`**)
  - így amennyiben a megfelelő viselkedés rendelkezésre áll, tetszőleges módon bővíthető deklaratívan a működés
- pl.:

```
<Entry Placeholder="Enter number">
    <Entry.Behaviors>
        <local:NumericValidationBehavior />
        <!-- az ellenőrzés automatikusan lefut a
             szöveg megváltoztatásakor -->
    </Entry.Behaviors>
</Entry>
```

Platformspecifikus Xamarin alkalmazások

Példa

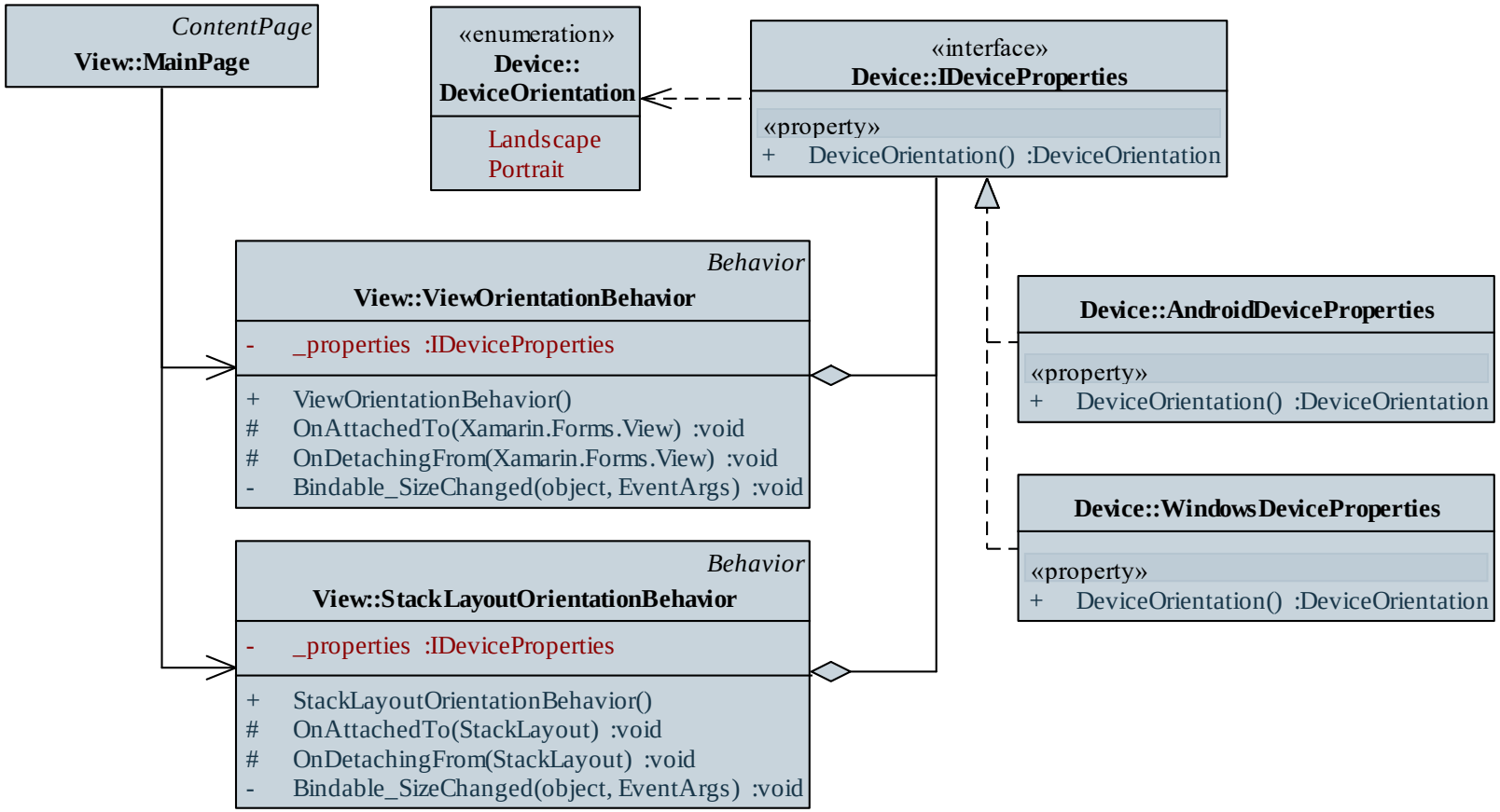
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- valósítsuk meg a tájoláskezelést viselkedések segítségével
- a tájolás platformonként eltérő módon kérhető le, ezért egy interfészen keresztül (**`IDeviceProperties`**) férünk hozzá a platformfüggő megvalósításhoz, mint függőséghez (**`DependencyService`**)
- két viselkedést veszünk fel
  - egyik tetszőleges vezérlő elhelyezését befolyásolja (**`ViewOrientationBehavior`**)
  - a másik a **`StackLayout`** elem elrendezési módját befolyásolja (**`StackLayoutOrientationBehavior`**)

Platformspecifikus Xamarin alkalmazások

Példa

Tervezés:



Platformspecifikus Xamarin alkalmazások

Példa

Megvalósítás (`StackLayoutOrientationBehavior.cs`*):*

```
private void Bindable_SizeChanged(object sender,
    EventArgs e)
{
    // az eszköz tájolásának függvényében
    // változtatunk a StackLayout tájolásán
    switch (_properties.DeviceOrientation){
        case DeviceOrientation.Landscape:
            if (stackLayout.Orientation !=
                StackOrientation.Horizonal)
                stackLayout.Orientation =
                    StackOrientation.Horizontal;
                break;
            …
```

Platformspecifikus Xamarin alkalmazások

Lokalizáció

- Általános elvárás, hogy az alkalmazások a felhasználó nyelvén kommunikáljanak, és annak megfelelő nyelvi környezetet használják (pl. számformátum, pénznem, …), ezt nevezzük az *alkalmazás lokalizációjának (application localization)*

- A .NET alkalmazások a lokalizációt egységes formában kezelik (a **`System.Globalization`** névtérben)

  - az alkalmazás nyelvi környezete (**`CultureInfo`**) bárhol elérhető, módosítható, illetve az alkalmazás átveheti a rendszer nyelvi környezetét (**`CultureInfo.CurrentCulture`**)

  - a megjelenített, nyelvfüggő tartalmakat (szöveg, képek) lehetőségünk van lokalizált erőforrásokból (*resource file*) betölteni

Platformspecifikus Xamarin alkalmazások

Lokalizáció

- A lokalizációt két lépésben határozhatjuk meg, nyelv, illetve terület (ország szintjén)
  - pl.: angol (Egyesült Államok): **`en-US`**, angol (Egyesült Királyság): **`en-GB`**, magyar (Magyaország): **`hu-HU`**
  - az adott lokalizációhoz kérhetjük le az erőforrásokat (**`CultureInfo`**), pl.:

```
CultureInfo info = new CultureInfo("en-US");
// amerikai lokalizáció betöltése
```

  - így lehetőségünk van elérni a lokális információkat, pl.: naptár (**`Calendar`**), időformátum (**`DateTimeFormat`**), számformátum (**`NumberFormat`**), tizedes elválasztó (**`NumberFormat.NumberDecimalSeparator`**)

Platformspecifikus Xamarin alkalmazások

Lokalizált erőforrások

- Az alkalmazás erőforrásait elhelyezhetjük erőforrás fájlokban (.resx), majd közvetlenül, statikus tulajdonságokként hivatkozhatjuk meg őket
  - az erőforrás fájl neve az osztálynév, a kulcs a tulajdonság neve lesz, pl.:

```
nameLabel.Text = AppText.NameText;
// az AppText.resx fájl NameText kulcsú
// szövegének betöltése, és a címkére állítása
```

  - az erőforrásfájlt több nyelvre is elkészíthetjük úgy, hogy a lokalizáció nevét hozzáillesztjük a fájlnévhez, pl.:

```
AppText.en-US.resx
   // szövegek amerikai nyelvi környezetre
AppText.hu-HU.resx // magyar környezetre
```

Platformspecifikus Xamarin alkalmazások

Lokalizált erőforrások

- a rendszer a beállított nyelvi környezetnek megfelelő erőforrásfájl tartalmát tölti be
- amennyiben a lokalizációnak megfelelő fájl nem található, vagy nincs benne megadott kulcsú elem, akkor a lokalizáció nélkül megadott erőforrásfájl tartalmát tölti be
- így célszerű minden esetben egy alapértelmezett erőforrás fájlt is készíteni, pl.:

```
AppText.resx // alapértelmezett szövegek
AppText.hu-HU.resx
    // lokalizáció magyar környezetre
```

- az erőforrások kezelése további lehetőségeket a `ResourceManager` típus biztosít

Platformspecifikus Xamarin alkalmazások

Lokalizált erőforrások a nézetben

- Amennyiben a lokalizációt nem kódban, hanem a grafikus felületen (XAML-ben) szeretnénk betölteni, ki kell bővítenünk a Xamarin funkcionalitását
  - a felületre történő beíráshoz szükséges egy bővítés a leíró nyelvhez (**`IMarkupExtension`**), amelyben megadjuk a lokalizált tartalom betöltésének módját (**`ProvideValue`**)
  - Android és iOS platformokon ezen felül a beépített nyelvi környezetet meg kell feleltetnünk .NET nyelvi környezetnek, így azt platformfüggő módon kell lekérnünk
    - pl. ki kell cserélnünk az összekötőt (`_` helyett `-`), egyes környezetek helyett mást kell használnunk (**`gsw-CH`** helyett **`de-CH`**)

Platformspecifikus Xamarin alkalmazások

Példa

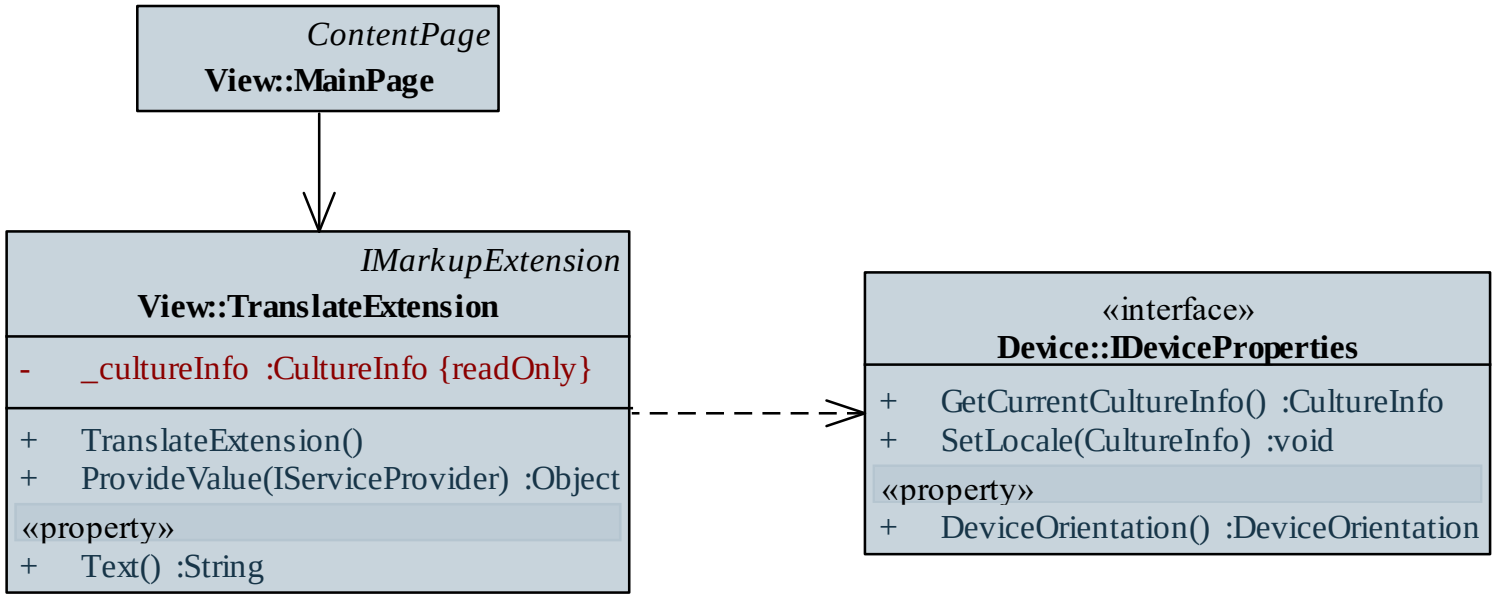
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- jelenítsük meg a szövegeket és a tizedes elválasztót a nyelvi beállításnak megfelelően angolul, vagy magyarul
- a tizedes elválasztót a nézetmodell közvetlenül elkérheti a környezettől
- a szövegeket erőforrásban (**`ApplicationText`**) tároljuk, amelyből készítünk alapértelmezett és magyar változatot
- kiegészítjük az eszköztulajdonságokat (**`IDeviceProperties`**) a nyelvi környezet kezelésével **`Android`** platformra
- megvalósítunk egy XAML bővítést, ami szintén elvégzi a konverziót (**`TranslateExtension`**)

Platformspecifikus Xamarin alkalmazások

Példa

Tervezés:



Platformspecifikus Xamarin alkalmazások

Példa

Megvalósítás (`MainPage.xaml`*):*

```
<ContentPage …
    xmlns:view="clr-namespace:ELTE.Calculator.View"
    …>

…
<Label Text="{view:Translate HistoryListTitle}"
       Style="{StaticResource
                NumberListViewLabelStyle}" />
    <!-- nyelvfüggő szöveg betöltése a
         TranslateExtension típus segítségével -->
…
```