Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése II

10. előadás

Window Runtime alapismeretek, Modern UI alapú alkalmazások

Giachetta Roberto

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

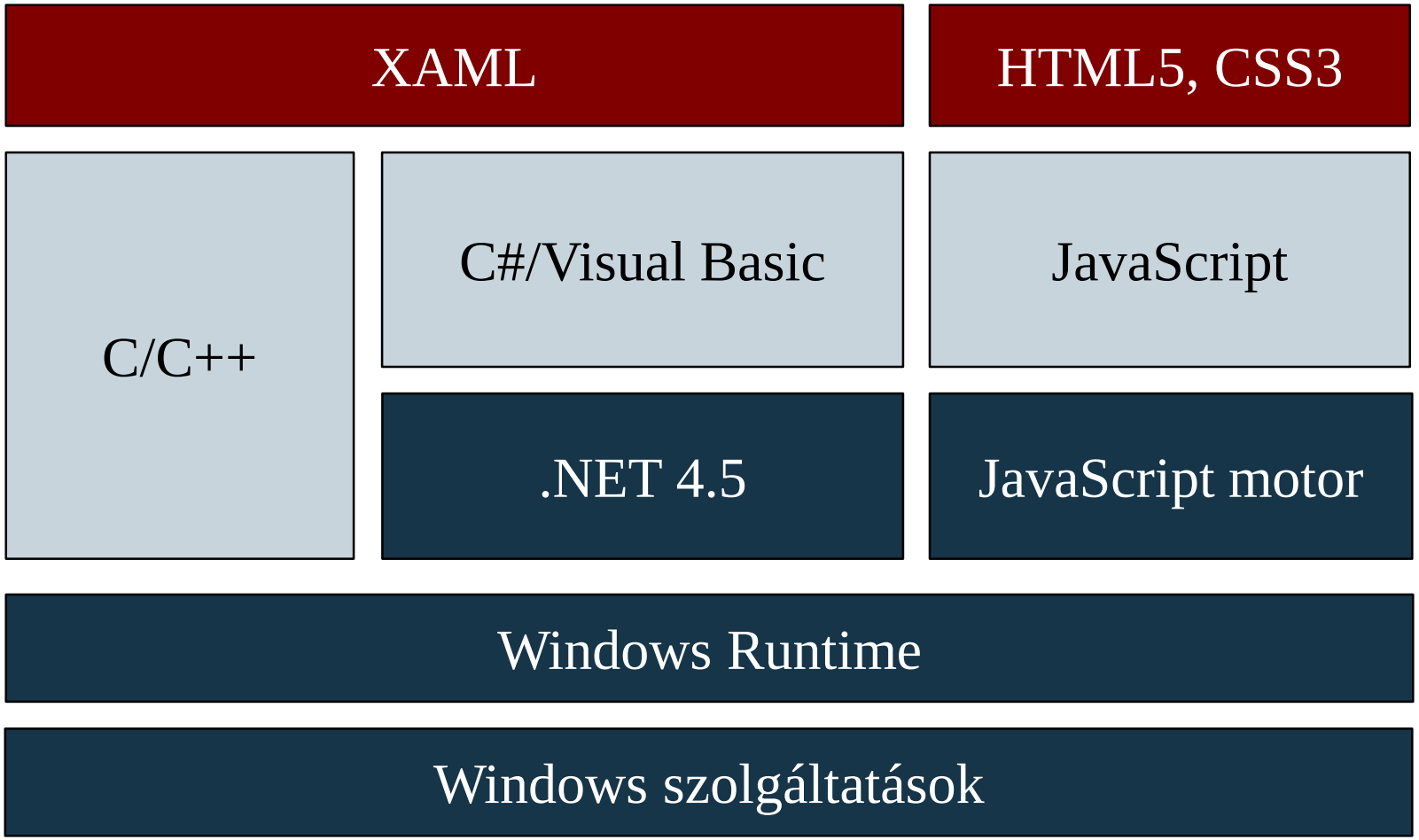
Windows Runtime alapismeretek

Kialakulás

- A .NET keretrendszer felügyelt platformja több hátránnyal is rendelkezik (teljesítmény, együttműködés)
  - sok esetben vegyes kódbázisú alkalmazásokat kell építeni
  - a cél a felügyelet mentes és felügyelt technológiák egységesítése, ugyanakkor alkalmassá tétele több programozási nyelvvel/technológiával való kompatibilitásra (*C/C++*, *.NET*, *JavaScript*)
- A *Windows Runtime* (*WinRT*) a Windows 8 rendszerek fejlesztői platformja
  - egységes, felügyeletmentes API, amely különböző technológiákon (.NET, JavaScript) keresztül is elérhető

Windows Runtime alapismeretek

Architektúra



Windows Runtime alapismeretek

Koncepciók

- Koncepciók:
  - technológiákkal való együttműködés, összehangolás, metaadat-kezelés
  - hatékonyságnövelés, aszinkron végrehajtási modell
  - biztonságos működés, korlátozott rendszerhozzáférés
  - célhardverek kezelése, hozzáférés szabályozás
- Mobiltelefonokra szánt változata a *Windows Phone Runtime*
- Az egyes platformok (WinRT, .NET, Silverlight, XBox) között megoszthatóak programegységek a *Portable Class Library* (*PCL*) segítségével

Windows Runtime alapismeretek

Alkalmazások mobil környezetben

- A mobil/táblagépes környezetben alkalmazásunknak számos olyan követelményt kell teljesíteni, amit az asztali alkalmazásnak nem
  - fontos az *adaptivitás*: az alkalmazás
    - különböző hardvereken,
    - különböző méretű/felbontású képernyőkön,
    - különböző tájolással (portré/tájkép),
    - különböző üzemmódokban (teljes képernyő, oldalra zárt, …) futhat
  - fontos a *beviteli gesztus*ok kezelése
  - elérhetőek *speciális hardverek* (GPS, gyorsulásmérő, …)

Windows Runtime alapismeretek

Alkalmazások grafikus felülete

- Az alkalmazások grafikus felülete a *Modern UI* (korábban *Metro UI*) tervezési koncepció segítségével valósul meg
  - *infografikus*, célja a minimalizmus, a letisztultság, a kezelhetőség növelése (kisebb méretű kijelzőkön is)
  - az alkalmazás futtatása legyen gyors és folyamatos (*aszinkron* végrehajtása a tevékenységeknek)
  - egy meghatározott tervezési vonalat, és annak sajátosságait (pl. *Segoe UI* betűtípus) kell követni
  - nélkülözi a bonyolult grafikákat, animációkat
  - az ikonokat felváltja csempékkel, amelyek tartalma dinamikus is lehet

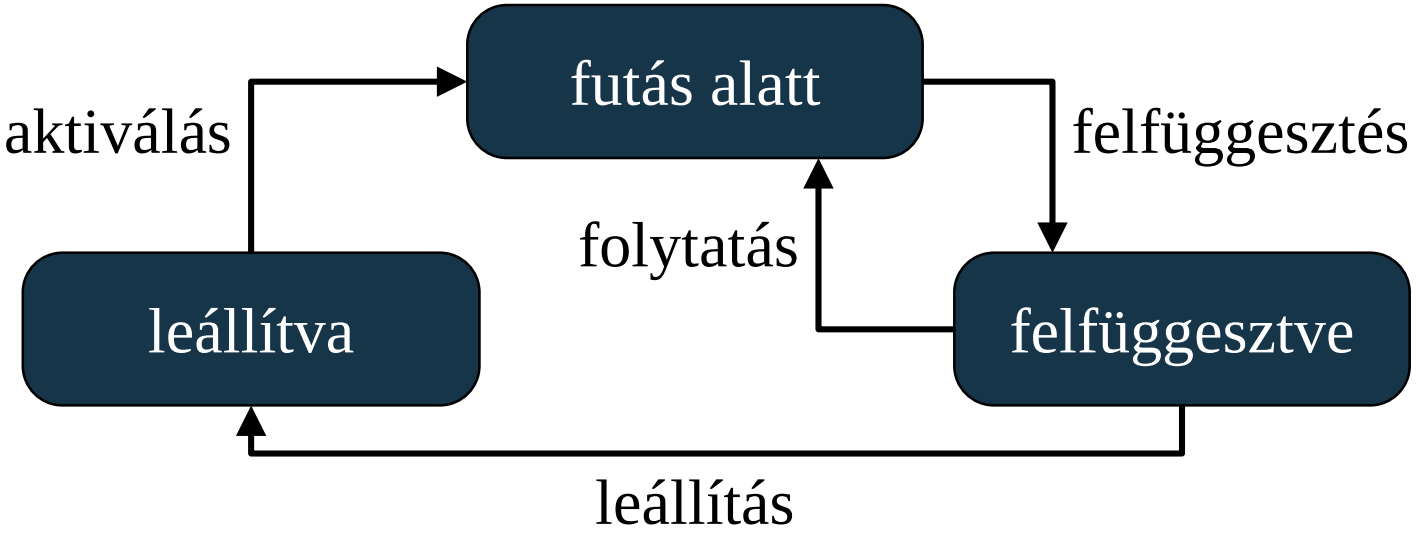
Windows Runtime alapismeretek

Alkalmazások futtatása

- 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
  - emiatt a fejlesztői eszközkészletük is korlátozott
- Az alkalmazás tulajdonságait, illetve engedélyeit az *alkalmazás azonosító* (*Application Manifest*) tartalmazza, amely része az alkalmazás csomagjának
  - az engedélyeket a telepítéskor jóvá kell hagyni

Alkalmazások életciklusa

- A mobil alkalmazások más életciklusban futnak, mint az asztali alkalmazások
  - a *futás* és a *terminált* állapotok mellett megjelenik a *felfüggesztett* állapot is, amely akkor lép életbe, ha az alkalmazás a háttérbe (vagy a gép alvó állapotba) kerül



- a felfüggesztés célja az erőforrásokkal való takarékosság
- a terminálást kezdeményezheti a felhasználó (pl. gesztussal), illetve az operációs rendszer felfüggesztett alkalmazások esetén (pl. amennyiben kevés a memória)
  - a program nem zárható be más módon (kódban)
  - az alkalmazás nem kap figyelmezetést a bezárásról (célszerű felfüggesztéskor menteni az adatokat, illetve elengedni az erőforrásokat)
- a program elszállhat kivétellel, ekkor eltűnik, és a felhasználó visszakerül a Start képernyőre (*Windows crash experience*)
  - ebben az esetben ne állítsuk vissza a mentett adatokat

Windows Runtime alapismeretek

Alkalmazások felépítése

- Az alkalmazások (projektek) elemei:
  - szabványos kódfájlok, illetve grafikus felület (XAML és háttérkód)
  - erőforrás gyűjtemények (*Resource Dictionary*), illetve más erőforrások (pl. csempe és nyitó képernyő grafikák)
  - személyes azonosító fájl (*Personal Information Exchange* file)
  - alkalmazás azonosító (*Application Manifest*)
- Az alkalmazás (és a hozzá tartozó tartalom) egy csomagot alkot (**appx**), amely publikálható a *Windows Áruház*ban, amennyiben megfelel az elvárásoknak

Windows Runtime alapismeretek

Alkalmazások architektúrája

- A szoftvereket *.NET for Windows Store* keretrendszerben fejlesztjük, amely a WinRT felügyelt interfésze, illetve egy lehatárolt részhalmaza a .NET keretrendszernek
  - az osztályok sok esetben a *Silverlight* programozási modelljére támaszkodnak
- Az alkalmazásokat fejleszthetjük:
  - *modell/nézet/perzisztencia* architektúrában
    - az eszközkészlet jó része elérhető, az eseménykezelés a megszokott módon (kiegészülve pl. gesztúrákkal)
  - *modell/nézet/nézetmodell* (*MVVM*) architektúrában
    - több megszorítás található az eszközkészletre

Windows Runtime alapismeretek

A grafikus felület felépítése

- A grafikus felület *oldalak*ból (**`Page`**) áll
  - felépítése hasonló az ablakokéhoz, de mindig kitölti a rendelkezésére álló teret,
  - a mérete előre nem meghatározható, ezért elrendezéseket (**`Grid`**, **`Canvas`**, **`StackPanel`**, …), transzformációkat (**`Viewbox`**) kell alkalmazni
  - az oldal figyelhet az elhelyezésre is
  - az elemek megjelenítéséhez adott egy stíluskészlet (amelyet a **`ThemeResource`** hivatkozással érhetünk el)
  - lehet előugró üzenetek (**`MessageDialog`**), illetve speciális funkciójú oldalakat (**`FileOpenPicker`**, …) használni

Windows Runtime alapismeretek

Az alkalmazás vezérlése

- A programot az alkalmazás (**`App`**) vezérli, amely egyesíti az erőforrásokat
  - az alkalmazás aktiválásakor (**`OnLaunched`**) végezzük az inicializálást
  - adatmentést végezhetünk felfüggesztéskor (**`Suspending`**), megjelenítés frissítést aktiváláskor (**`Resuming`**)
  - az oldalakat *keret*be helyezzük (**`Frame`**), amely szabályozza az oldalak közötti átmenetet (**`Navigate(…)`**, **`GoBack()`**, **`GoForward()`**), ehhez tárolja az eddig megnyitott ablakokat
  - a keretet helyezzük az aktuális ablakba (**`Window.Current`**), majd ezt aktiváljuk (**`Activate()`**)

Windows Runtime alapismeretek

Példa

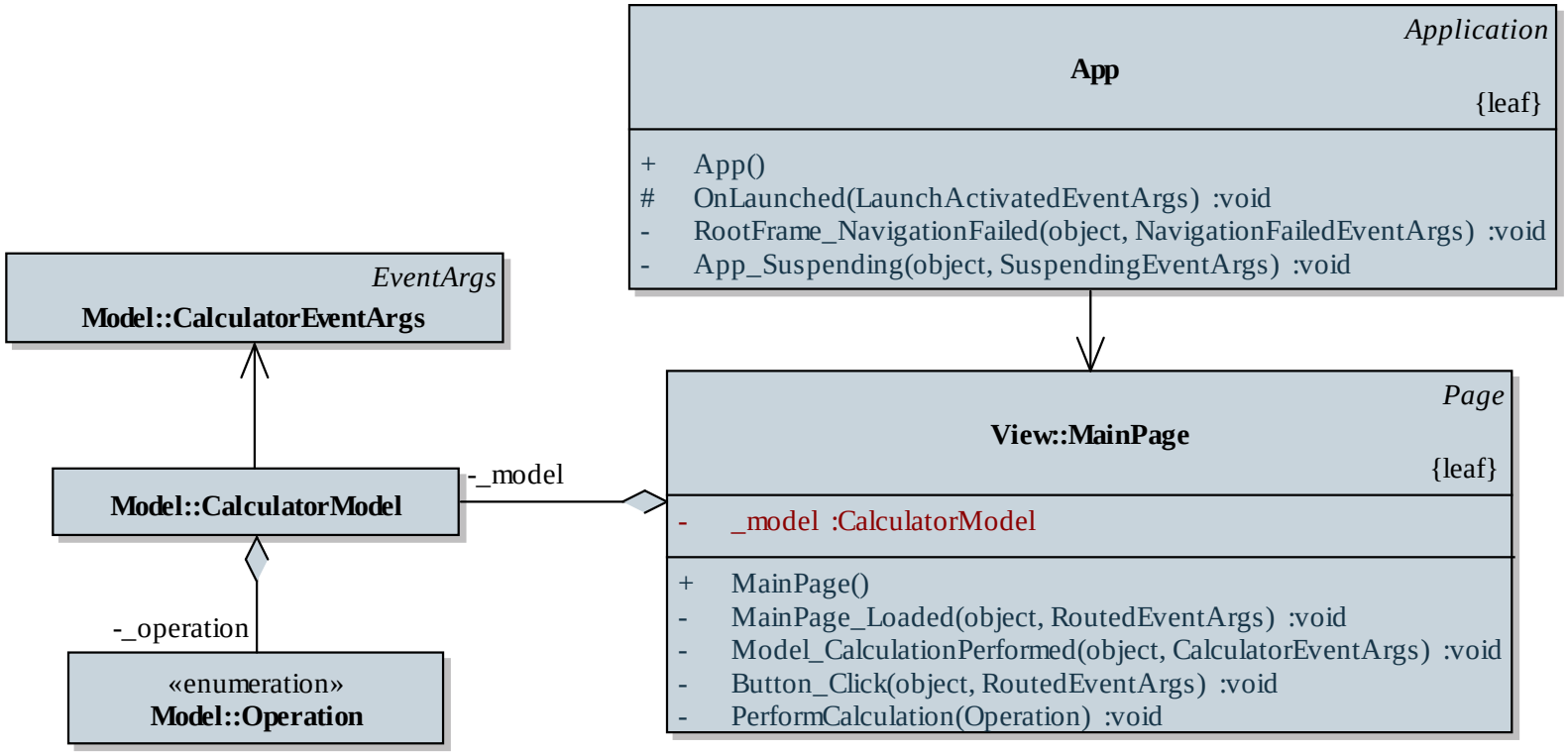
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- az alkalmazást modell/nézet architektúrában valósítjuk meg, a korábban készített modell (**`CalculatorModel`**) újra felhasználjuk
- a képernyőn (**`MainPage`**) felvesszük a megfelelő vezérlőket, amelyek stílusok segítségével egyedire szabunk
- eseménykezelővel végrehajtjuk a tevékenységeket, előugró dialógussal (**`MessageDialog`**) figyelmeztetünk a hibás bemenetre

Windows Runtime alapismeretek

Példa

Tervezés:



Windows Runtime alapismeretek

Példa

Megvalósítás (`MainPage.xaml`*):*

```
<Page …>
    <Page.Resources>
      <!-- új stílusokat adunk meg a vezérlőkre -->
      <Style x:Key="FunctionButtonStyle"
             TargetType="Button">…</Style>
      …
    </Page.Resources>
    <Grid Background="{ThemeResource
            ApplicationPageBackgroundThemeBrush}">
       <!-- felhelyezzük a vezérlőket a rácsra -->
       …
    </Grid>
</Page>
```

Windows Runtime alapismeretek

Példa

Megvalósítás (`MainPage.xaml.cs`*):*

```
private void MainPage_Loaded(…)
{
    // akkor példányosítunk mindent, amikor az
    // oldal betöltött
    _model = new CalculatorModel();
    _model.CalculationPerformed += …
        // modell eseményének társítása
    _textNumber.Text = _model.Result.ToString();

    _textNumber.Focus(FocusState.Keyboard);
    _textNumber.SelectAll();
}
```

Az MVVM architektúra

- Alkalmazásainkat megvalósíthatjuk MVVM architektúrában is, a következő tényezők figyelembe vételével
  - a nézetmodell és a modell a megszokott módon kezelhető a környezetben (pl. alkalmazás osztályban), a nézetmodellt a keret adatforrásaként kell megadni (`Frame.DataContext`)
  - a parancsok (`ICommand`) kiválthatóságát manuálisan kell kezelni, mivel nem elérhető a parancskezelő (`CommandManager`)
  - speciális parancsok (pl. billentyűzetkezelés) nem használhatóak (ugyanakkor használatok nem is célszerű a mobil környezetben)

Példa

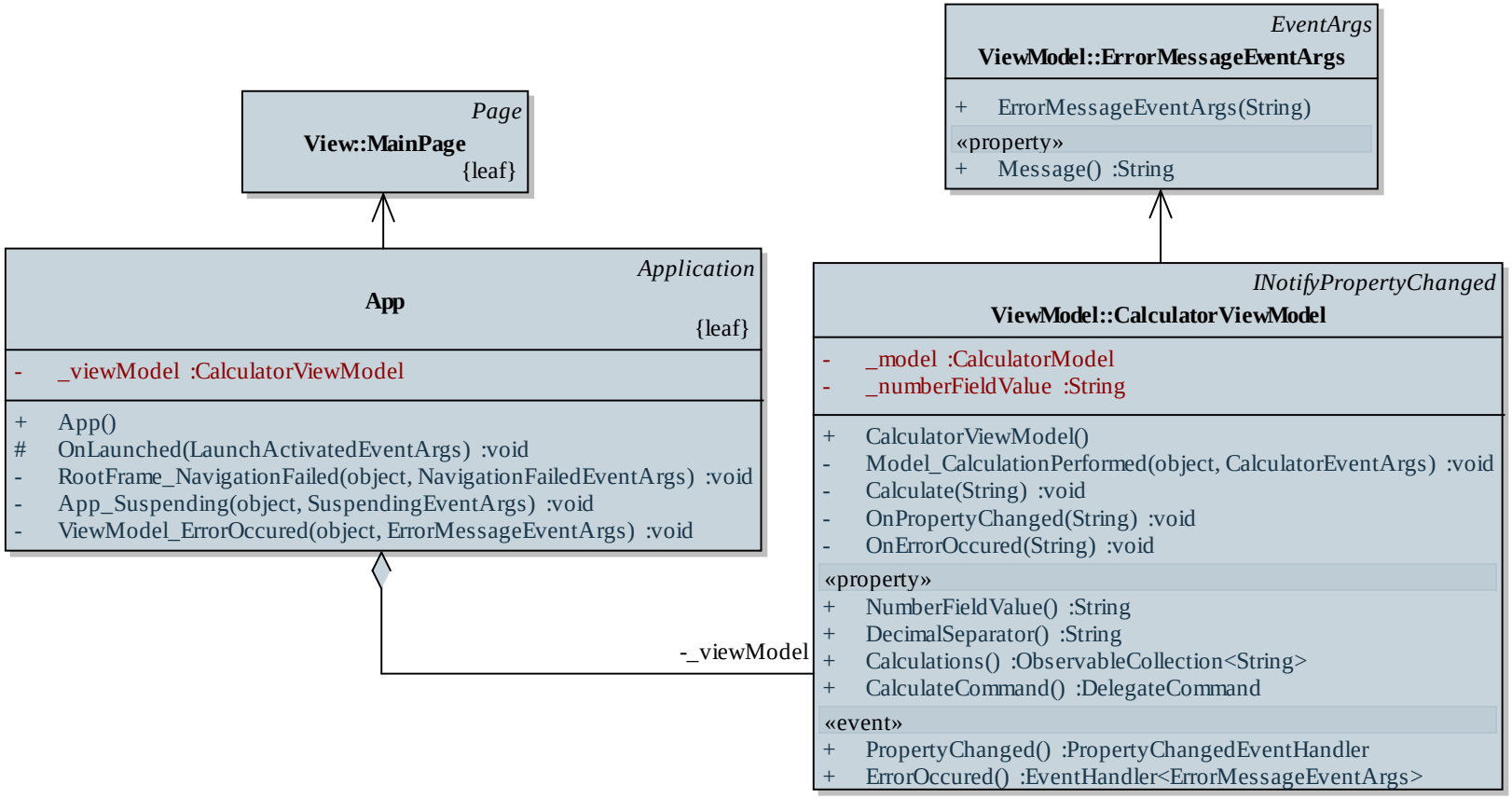
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- annak érdekében, hogy táblagépen is könnyen használható legyen a felületen gombokat helyezünk el a bevitelhez (számok, tizedespont és törlés), így nem kell billentyűzetet használnunk
- az alkalmazást MVVM architektúrában építjük fel, a korábban készített modellt (**`CalculatorModel`**) és a nézetmodellt (**`CalculatorViewModel`**) újra felhasználjuk
- a modellt kiegészítjük a törlés funkciójával, a nézetmodellt kiegészítjük az új gombok tevékenységeivel

Windows Runtime alapismeretek

Példa

Tervezés:



Windows Runtime alapismeretek

Példa

*Megvalósítás (**CalculatorViewModel.cs**):*

```
…

private void Calculate(String operatorString){
   …
   switch (operatorString) {
      // művelet végrehajtása, amely most már
      // számjegy, tizedesjel és törlés is lehet
      …
      case "C": // törlés
         _model.Clear();
         NumberFieldValue = "0";
         break;
```

Windows Runtime alapismeretek

Példa

Megvalósítás (`CalculatorViewModel.cs`*):*

```
            default:
                if (_numberFieldValue == "0" &&
                    operatorString != DecimalSeparator)
                    // 0 esetén lecseréljük a tartalmat
                    // (kivéve, ha tizedesjel jön),
                    // egyébként hozzáírjuk
                    _numberFieldValue = operatorString;
                else
                    // minden más esetben csak hozzáírjuk
                    _numberFieldValue += operatorString;
                    OnPropertyChanged("NumberFieldValue");
                break;
            …
```

Windows Runtime alapismeretek

Példa

Megvalósítás (`App.xaml.cs`*):*

```
protected override void OnLaunched(
                    LaunchActivatedEventArgs args) {
   …
   _viewModel.ErrorOccured +=
      new EventHandler<ErrorMessageEventArgs>(
                           ViewModel_ErrorOccured);
   …
private void ViewModel_ErrorOccured(object sender,
                           ErrorMessageEventArgs e){
   MessageDialog dialog =
      new MessageDialog(e.Message);
   dialog.ShowAsync();
}
```

Példa

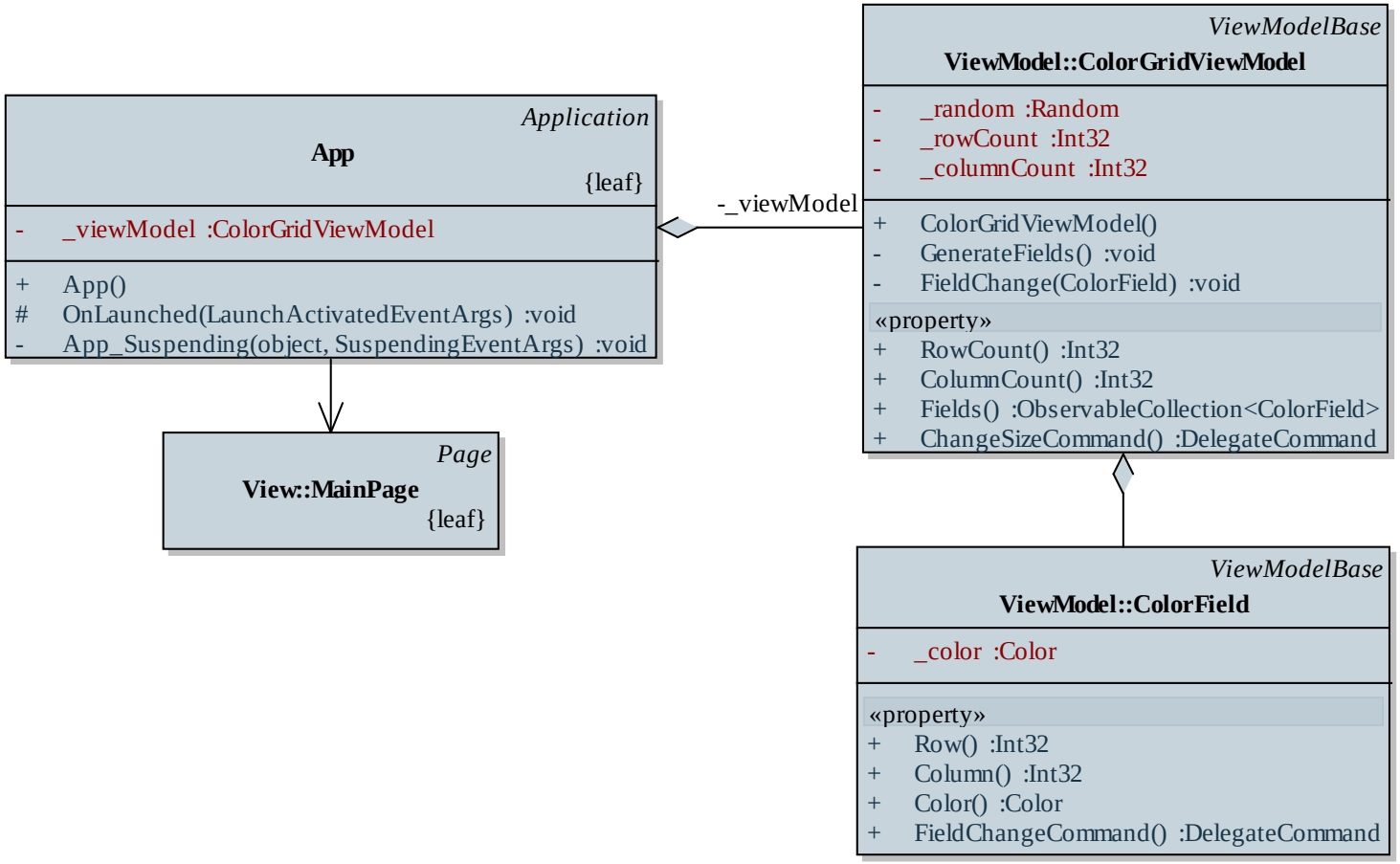
Feladat: Készítsünk egy dinamikus méretezhető táblát, amely véletlenszerű színre állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- MVVM architektúrát használunk, és a korábbi megoldásból felhasználjuk a nézetmodellt (**`ColorGridViewModel`**, **`ColorField`**)
- a nézetet ellátjuk oldalcímmel, és betartjuk az elhelyezési konvenciókat, továbbá csúszkákat (**`Slider`**) használunk az értékbeállításhoz (amelyet külön címkén is megjelenítünk)
- a rácsot **`ItemsControl`** segítségével helyezzük el, amelyben **`WrapGrid`** megjelenítőt használunk (mivel nincs **`UniformGrid`** vezérlő)

Windows Runtime alapismeretek

Példa

Tervezés:



Windows Runtime alapismeretek

Példa

Megvalósítás (`MainPage.xaml`):

```
…
<ItemsControl ItemsSource="{Binding Fields}">
   <!-- elemek gyűjtő vezérlője -->
   <ItemsControl.ItemsPanel>
      <ItemsPanelTemplate>
         <!-- az elemek egy rácsban fognak
              elhelyezkedni, amelyet
              sorfolytonosan töltünk fel -->
         <WrapGrid Orientation="Horizontal"
                   MaximumRowsOrColumns="{Binding
                                ColumnCount}" />
      </ItemsPanelTemplate>
  …
```

Windows Runtime alapismeretek

Szinkron és aszinkron programozás

- A tevékenységek végrehajtásának két megközelítése van:
  - *szinkron*: a tevékenység kezdeményezője megvárja annak lefutását
    - a hívó szál blokkolódik, amíg a tevékenység lefut
    - ha sokáig tart a tevékenység, akkor az a program felületén is észrevehető
  - *aszinkron*: a tevékenység kezdeményezője nem várja meg a lefutást, illetve az eredményt
    - a tevékenység (metódus) külön szálon fut
    - az eredményt később megkapjuk (pl. eseményen át)
    - a hívó szál nem blokkolódik, folytathatja a végrehajtást

Windows Runtime alapismeretek

Aszinkron műveletek

- A Windows Runtime működési elveiben fontos, hogy
  - gyorsan reagáljunk a felhasználói interakcióra, a felhasználói felület mindig aktív legyen
  - amennyiben egy nagyobb műveletet hajtunk végre, azt aszinkron módon, háttérben végezzük
- A tevékenységek jelentős része (pl. ablaknyitás, fájlolvasás, hálózatkezelés) aszinkron műveletként van megvalósítva
  - az a műveletek nevében jelzett (**`Async`**)
  - pl.:

```
MessageDialog dialog = …;
dialog.ShowAsync(); // aszinkron művelet
```

Windows Runtime alapismeretek

Aszinkron műveletek

- Aszinkron műveleteket az **async** kulcsszóval hozhatunk létre
  - aszinkron műveletben lehetőségünk más aszinkron műveletet futtatni, és azt bevárni az **await** utasítással, pl.:

```
private async void ReadStream(Stream str)
{
    StreamReader reader = new StreamReader(str);
    String line = await reader.ReadLineAsync();
       // aszinkron módon olvasunk, és megvárjuk
       // a művelet lefutását
    MessageDialog dialog =
       new MessageDialog(line);
    await dialog.ShowAsync();
       // megvárjuk a dialógus bezárását
}
```

Windows Runtime alapismeretek

Aszinkron műveletek

- A .NET nyelvi könyvtárban több műveletnek adott szinkron/aszinkron megvalósítása is
  - pl. **`ReadLine()`**, **`ReadLineAsync()`**
  - a Windows Runtime nyelvi könyvtárában jórészt csak az aszinkron műveletek találhatóak meg
- Lehetőségünk van egy szinkron műveletben megvárni az aszinkron művelet eredményét
  - ehhez blokkolnunk kell a műveletet, az **`AsTask().Result`** lekérdezés biztosítja a várakozást
  - pl.:

```
result = dialog.ShowAsync().AsTask().Result;
```

Windows Runtime alapismeretek

Időzítés

- A felület szintű időzítő adott (**`DispatcherTimer`**), azonban a modell szintű időzítés (**`System.Timers.Timer`**) nem került be az osztálykönyvtárba
  - a funkcionalitása kiváltható a **`ThreadPoolTimer`** típussal
    - lehet periodikus, vagy egyszeri
    - nem esemény alapú, konstruktorparaméterben megadható a futtatandó művelet
    - a felhasználásával lehet készíteni saját időzítőt
  - pl.

```
timer = ThreadPoolTimer.CreatePeriodicTimer(
    Timeout, TimeSpan.FromMinutes(1));
    // a Timeout függvényt hívja meg percenként
```

Windows Runtime alapismeretek

Szinkronizáció

- Amennyiben párhuzamosan végzünk tevékenységet, ügyelnünk kell a szinkronizációra (a felülettel)
  - a felületi időzítő, illetve az aszinkron tevékenységek szinkronizáltak
  - a szálbiztos végrehajtáshoz használnunk kell a **`Dispatcher.RunAsync(<`*`tevékenység`*`>)`** metódust, pl.

```
CoreApplication.MainView.CoreWindow.Dispatcher.
  RunAsync(CoreDispatcherPriority.Normal,
    // a végrehajtás prioritását is megadjuk
  () => {
      // a tevékenység lambda-kifejezése
      textBox.Text = "Hello World!";
  }))
```

Windows Runtime alapismeretek

Példa

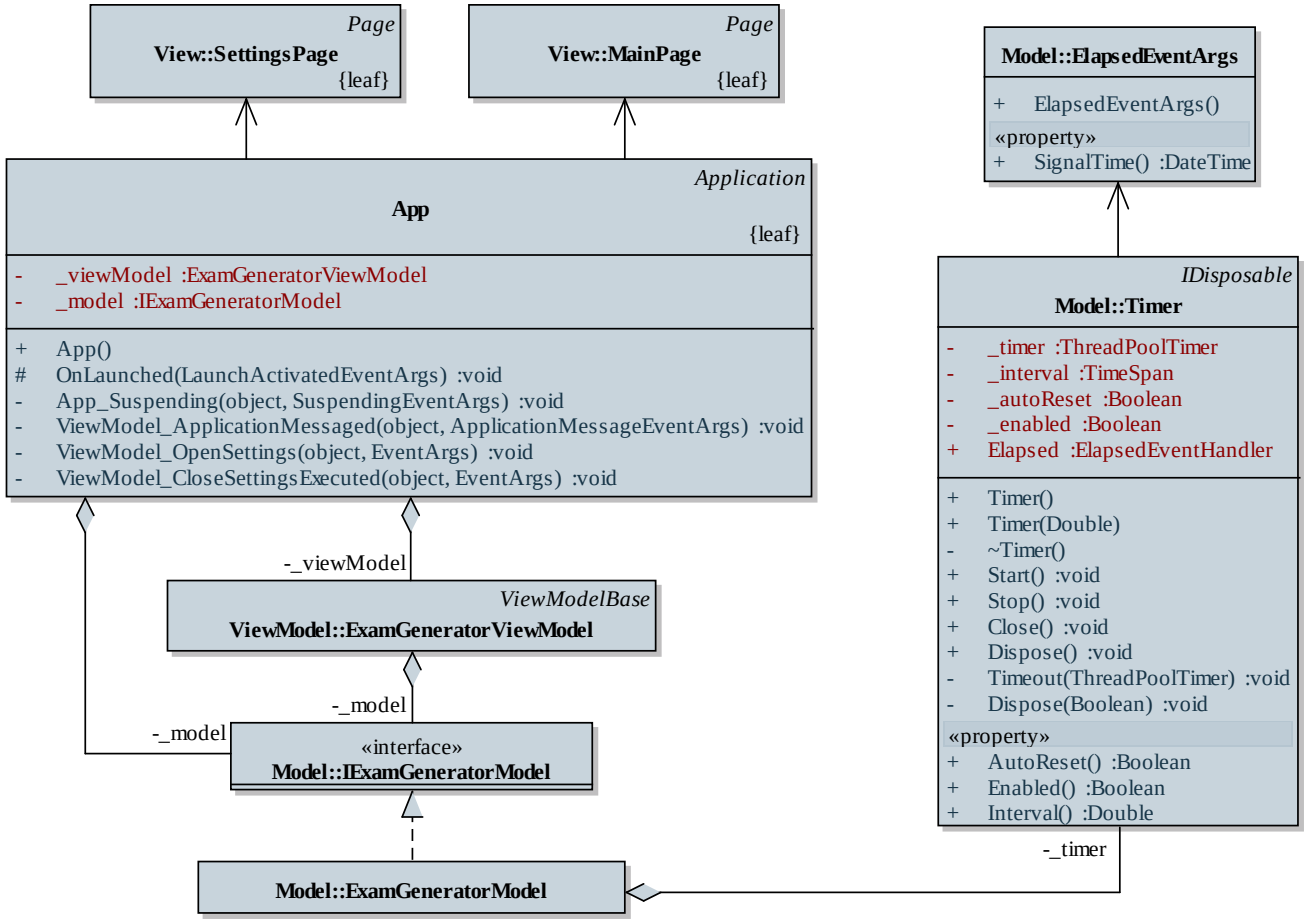
Feladat: Készítsünk egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- használjunk MVVM architektúrát, és a korábbi megvalósítást
- a modellt (**`ExamGeneratorModel`**) nem kell módosítanunk, de ki kell egészítenünk egy időzítő (**`Timer`**) típussal
- a nézetmodellben (**`ExamGeneratorViewModel`**) szinkronizáltan kell lekezelnünk az időzítő párhuzamos eseményeit
- a nézet két lapot is tartalmaz (**`MainPage`**, **`SettingsPage`**), amelyek között navigálunk

Windows Runtime alapismeretek

Példa

Tervezés:



Windows Runtime alapismeretek

Példa

Megvalósítás (`App.xaml.cs`*):*

```
private void ViewModel_OpenSettings(object sender,
                                    EventArgs e) {
    Frame rootFrame =
        Window.Current.Content as Frame;

    if (rootFrame.CanGoForward) // ha már egyszer
        // átnavigáltunk a beállítások oldalra
        rootFrame.GoForward();
        // akkor ismét elnavigálhatunk oda
    else
        rootFrame.Navigate(typeof(SettingsPage));
        // különben felvesszük, mint új oldalt
}
```

Windows Runtime alapismeretek

Példa

Megvalósítás (`ExamGeneratorViewModel.cs`*):*

```
  private async void Model_NumberGenerated(
                          object sender, EventArgs e) {
     // szinkron műveletben megvárhatjuk az
     // szinkron végrehajtás végét
     await CoreApplication.MainView.
           CoreWindow.Dispatcher.
           RunAsync(CoreDispatcherPriority.Normal,
              () => OnPropertyChanged(
                       "QuestionNumber"));
        // szinkronizált végrehajtás
  }
```