



**Eötvös Loránd Tudományegyetem
Informatikai Kar**

Eseményvezérelt alkalmazások fejlesztése II

10. előadás

Xamarin alapismeretek

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

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

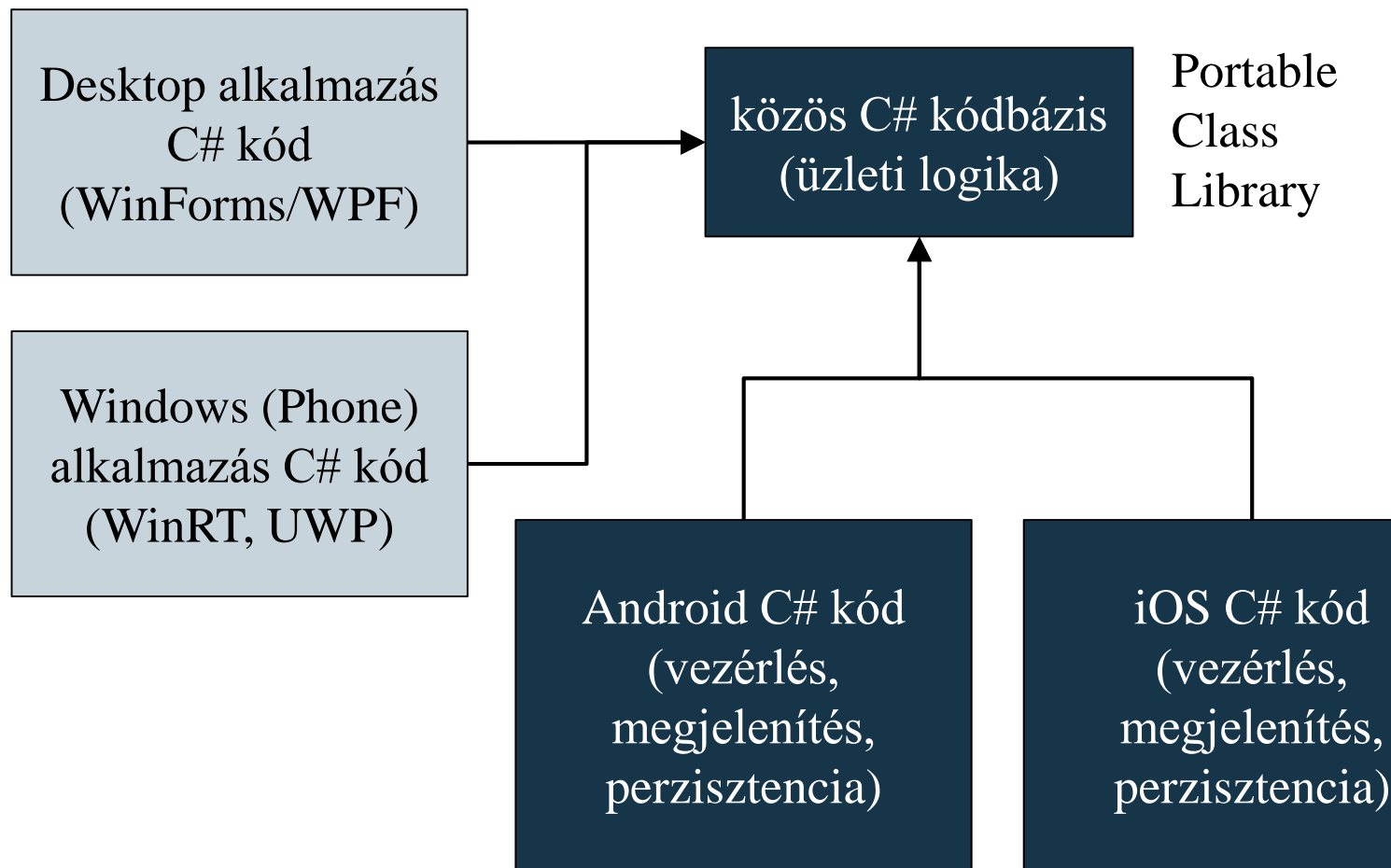
Xamarin alapismeretek

A Xamarin platform

- A *Xamarin* egy többplatformos szoftverfejlesztői környezet, amely lehetőséget ad Android, iOS, Windows Phone (8+) és Windows (8+) alkalmazások fejlesztésére a .NET keretrendszerre alapozva
 - lehetőséget ad közös kódbázissal rendelkező alkalmazások fejlesztésére, amelyeket natív alkalmazásokká (Objective-C, Java, ...) fordít át
 - a *Xamarin.Forms* csomag az architektúrát (MVVM) és a grafikus felületet is egységesíti
 - alapja a Mono, amely a .NET keretrendszer alternatív megvalósítása Linux és MacOS rendszerekre
 - a programcsomagok NuGet segítségével kezelhetők és frissíthetők

Xamarin alapismeretek

Alkalmazások felépítése



Xamarin alapismeretek

Xamarin.Forms alkalmazások felépítése

- Xamarin.Forms alkalmazások esetén a közös programegységek (osztálykönyvtárak) tartalmazzák
 - a modellt, amely tartalmazza az üzleti logikát, szokványos eszközök segítségével felépítve
 - a nézetmodellt, amelyet az alapvető eszközök segítségével tudunk felépíteni (**ICommand**, **INotifyPropertyChanged**, ...)
 - a nézetet, amely XAML alapon írunk le, és adatkötés (**Binding**) segítségével kapcsoljuk a nézetmodellhez
 - az alkalmazás vezérlését (**App**), amely meghatározza a közös viselkedést minden platformon
 - a perzisztencia interfészét, a konkrét megvalósítás nélkül, amely platformonként eltérhet

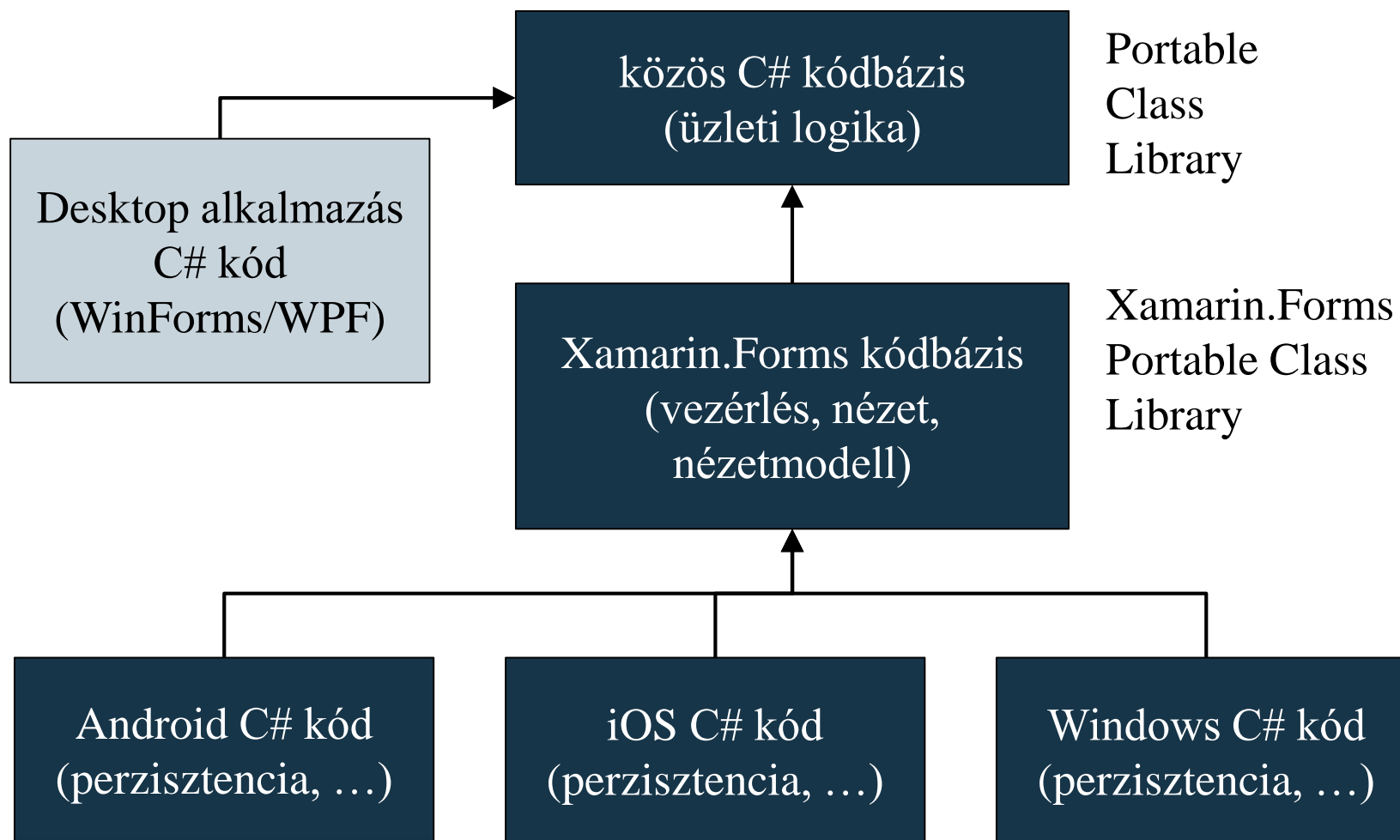
Xamarin alapismeretek

Xamarin.Forms alkalmazások felépítése

- Az alkalmazások közös funkcionalitást *Portable Class Library* segítségével valósíthatjuk meg
 - a közös programegységek is felruházhatóak platformspecifikus jellemzőkkel (**Device**)
- A platformspecifikus programegységek (Android, iOS, Windows, WindowsPhone) tovább bővíthetik a közös funkcionalitást
 - tartalmazza a perzisztencia megvalósítását, mivel az adattárolás módja platformonként eltér
 - tartalmazhatnak speciális nézetbeli elemeket, amelyek adott platformban érhetőek csak el, illetve lehetőséget a nézet adaptálására (pl. Material Design)

Xamarin alapismeretek

Xamarin.Forms alkalmazások felépítése



Xamarin alapismeretek

Xamarin.Forms alkalmazások felépítése

- Azt alkalmazások számára a közös belépési pontot az alkalmazás (**Application**) jelenti
 - tartalmazza a nyitóképernyőt (**MainPage**), valamint az alkalmazás életciklus eseménykezelőit (**OnStart**, **OnResume**, **OnSleep**)
 - kezeli az alkalmazás tulajdonságait (**Properties**), erőforrásait (**Resources**)
- Az alkalmazást minden platform egyedileg kezeli és tölti be, pl.:
 - Android platformon a főtevékenység (**MainActivity**) létrehozáskor (**OnCreate**)
 - Windows platformon a főképernyő (**MainPage**) a konstruktor lefutásakor

Xamarin alapismeretek

Xamarin.Forms grafikus felület felépítése

- A Xamarin.Forms alkalmazások egy egységes grafikus felülettel rendelkeznek, amelyeket a platform egyedi vezérlőire fordít le a rendszer
 - a felület lapokból (**Page**) áll, amelyek különböző felépítéssel rendelkezhetnek
 - pl. egyszerű tartalom (**ContentPage**), többlapos (**TabbedPage**), húzható (**CarouselPage**), többnézetű (**MasterDetailPage**)
 - a lapoknak lehet címe (**Title**), háttere (**BackgroundImage**) és ikonja (**Icon**), valamint platformspecifikus eszköztára (**ToolbarItems**)
 - megjeleníthetnek üzeneteket (**DisplayAlert**), valamint választódialógust (**DisplayActionSheet**)

Xamarin alapismeretek

Xamarin.Forms grafikus felület felépítése

- a tartalmat elrendező (**Layout**) elemekkel igazíthatjuk el, pl. rács (**Grid**), vízszintes/függőleges lista (**StackLayout**), abszolút pozíciós (**AbsoluteLayout**), relatív pozíciós (**RelativeLayout**)
- a tartalmat nézet (**View**) elemekből építhetjük fel, pl. **Label**, **Button**, **DatePicker**, **Entry** (egy soros szövegdoz), **Editor** (több soros szövegdoz), **Picker** (kiválasztó), **TableView**, **Image**, **WebView**
 - a nézeten megjelenő betűk alapértelmezetten platformspecifikusak, és mértük is platformnak megfelelően szabályozható (**Small**, **Medium**, ...)
 - lehetőségünk van egyedileg formázott szöveg megjelenítésére (**FormattedString**)

Xamarin alapismeretek

Xamarin.Forms grafikus felület felépítése

- Pl.:

```
<ContentPage ...>
  <!-- tartalomlap, amely lehet az alkalmazás
        képernyője -->
  <StackLayout Orientation="Vertical">
    <!-- függőleges elrendező -->
    <Label Text="Hello, Xamarin!" Margin="5"
          FontSize="Large"/> <!-- címke -->
    <Button Clicked="Button_Clicked"
            Text="Go on.." HorizontalOptions="Center"
            Margin="10" />
    <!-- gomb eseménykezelővel -->
  </StackLayout>
</ItemsControl>
```

Xamarin alapismeretek

Alkalmazás tulajdonságok és kiehlyezés

- Az alkalmazások tulajdonságait, képességeit az *alkalmazás leíró* (*application manifest*) segítségével írhatjuk le
 - tartalmazza az alkalmazás nevét, leírását, verzióját és a fejlesztő adatait
 - megadja az engedélyeket a rendszerhez, és más alkalmazásokhoz, pl. internet, kamera, pozicionálás, telefonkönyv, ...
 - Android esetén az **AndroidManifest.xml** fájl, Windows esetén a **Package.appmanifest** fájl tartalmazza a leírást
- A megfelelően konfigurált alkalmazások kihelyezhetőek fizikai eszközökre, illetve elhelyezhetőek a platform alkalmazásboltjában is, ehhez az alkalmazást megfelelő aláírással kell ellátnunk, amely szintén platformspecifikus

Xamarin alapismeretek

Alkalmazás erőforrások

- Az alkalmazások rendelkezhetnek platformfüggetlen, és platformfüggő erőforrásokkal
 - a nézethez használat erőforrások (stílusok, animációk, ...) a nézetben direkt (**Resources**), vagy erőforrásgyűjtemény (**ResourceDictionary**) formájában lehetnek jelen
 - az Android platform megkülönbözteti
 - a felügyelt erőforrásokat (*Resources*), amelyek tartalmat egyedileg illeszti be az alkalmazásba (pl. ikon)
 - a felügyeletmentes erőforrásokat (*Assets*), amelyek tartalma nyersen kerül az alkalmazásba
 - a Windows platform speciális erőforrásként kezeli az alkalmazás ikonjait (*Assets*)

Xamarin alapismeretek

Példa

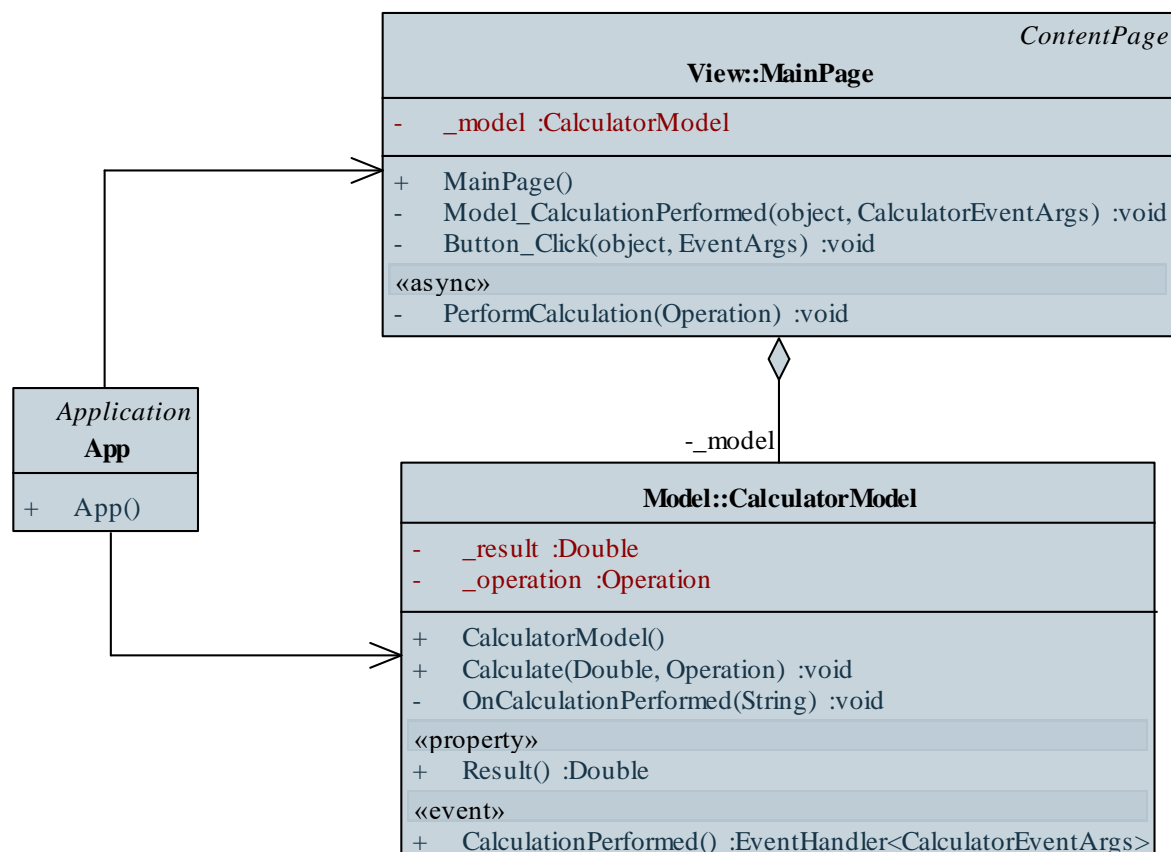
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- a modell (**CalculatorModel**) biztosítja a számológép funkcionalitást, ezt újrahasznosítjuk
- a nézetben (**MainPage**) elhelyezünk egy rácsot, benne a beviteli mezőt (**Entry**), a gombokat (**Button**), valamint a számítások listáját (**Label**)
- a gombokhoz közös eseménykezelőt rendelünk (**Button_Blick**), és a gomb szövege alapján döntünk a műveletről
- az eredményről figyelmeztető üzenetet küldünk (**DisplayAlert**)

Xamarin alapismeretek

Példa

Tervezés:



Xamarin alapismeretek

MVVM architektúra

- A Xamarin.Forms támogatja az MVVM architektúra alapú fejlesztést, így biztosított
 - az adatkötés (**Binding**) a nézet oldalon, amelynek megadhatunk tetszőleges forrást (**BindingContext**)
 - minden vezérlőnek külön is megadható forrás a **BindingContext** tulajdonság segítségével
 - az elnevezett elemekre (**x:Name**) is hivatkozhatunk a kötésben (**x:Reference**), pl.:

```
<Label x:Name="someLabel" />
```

```
<Label Text="{Binding  
                Source={x:Reference someLabel},  
                Path=Text}" />
```

```
<!-- a két címke ugyanazt írja ki -->
```


Xamarin alapismeretek

MVVM architektúra

- a változásjelzés (**INotifyPropertyChanged**, **ObservableCollection**), valamint a parancsvégrehajtás (**ICommand**, **Command**, **Command<T>**) nézetmodell oldalon
- az alkalmazás vezérlése az alkalmazás (**App** osztály) segítségével
 - a nézet adatforrását a **MainPage** tulajdonság **BindingContext** tulajdonságán keresztül adhatjuk meg, pl.:

```
ViewModel viewModel = new ViewModel(model);  
MainPage = new MainPage();  
MainPage.BindingContext = viewModel;  
// nézetmodell befecskendezése
```
 - a nézet cseréjét a **MainPage** tulajdonságon keresztül végezhetjük, ám ehelyett célszerű több lapot tartalmazó nézet használata (pl. **TabbedPage**, **MasterDetailPage**)

Xamarin alapismeretek

Példa

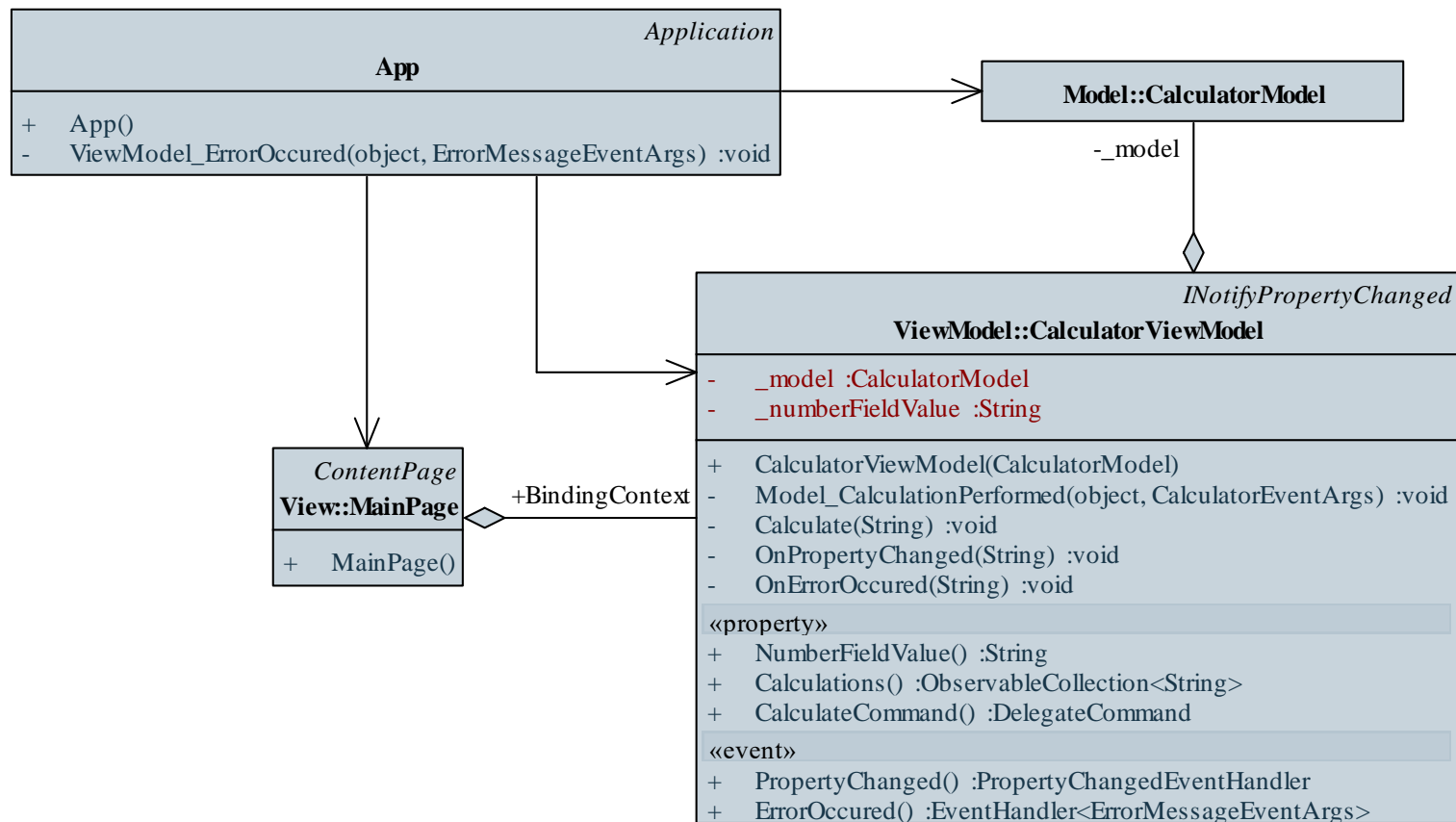
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- valósítsunk meg MVVM architektúrát a nézetmodell kiemelésével
- a nézetmodell (**CalculatorViewModel**) tartalmazza az aktuális értéket (**NumberFieldValue**), a számítások listáját (**Calculations**) és a számítást parancs formájában (**CalculateCommand**)
 - a számítási hibákkal kapcsolatosan eseményt küld (**ErrorOccured**)
- az alkalmazás példányosítja és összeállítja az alkalmazás rétegeit, és kezeli a számítási hibák eseményeit

Xamarin alapismeretek

Példa

Tervezés:



Xamarin alapismeretek

Példa

Megvalósítás (MainPage.xaml):

...

```
<!-- felhelyezzük a vezérlőket a rácsra, és  
      hozzákötjük őket a nézetmodellhez -->
```

```
<Entry Text="{Binding NumberFieldValue}"
```

```
      Grid.Row="0" Grid.Column="0"
```

```
      Grid.ColumnSpan="3" FontSize="40" />
```

```
<Button Command="{Binding CalculateCommand}"
```

```
      CommandParameter="+" Text="+"
```

```
      FontSize="30" HeightRequest="60"
```

```
      WidthRequest="70" HorizontalOptions="Start"
```

```
      Grid.Row="1" Grid.Column="0" />
```

...

Xamarin alapismeretek

Alkalmazások mobil környezetben

- A mobil/táblagépes környezetben alkalmazásunknak számos speciális követelményt kell teljesíteni
 - *könnyű áttekinthetőség*, infografikus megjelenítés
 - *folyamatos, gyors működés* aszinkron tevékenységekkel
 - *alkalmazkodás (resposiveness)*: az alkalmazás
 - különböző hardvereken,
 - különböző méretű/felbontású képernyőkön,
 - különböző tájolással (portré/tájkép),
 - különböző üzemmódokban (teljes képernyő, oldalra zárt, ...)
futhat
 - *kézmozdulatok* kezelése, és kihasználása
 - *speciális hardverek* igénybevétele (GPS, gyorsulásmérő, ...)

Xamarin alapismeretek

Kézmozdulatok kezelése

- Célszerű az alkalmazás vezérlésében a felhasználó kézmozdulataira támaszkodni
 - bármely vezérlőre állíthatunk kézmozdulat érzékelést (**GestureRecognizer**), így tetszőleges tevékenységet (**Command**) rendelhetünk bármely vezérlőhöz
 - támogatott az érintés (**TapGestureRecognizer**), a csíptetés (**PinchGestureRecognizer**) és a húzás (**PanGestureRecognizer**), illetve tetszőleges egyedi mozdulat megvalósítása (**IGestureRecognizer**), pl.:

```
<Label ...>
```

```
  <Label.GestureRecognizers> <!-- érzékelés -->
```

```
    <TapGestureRecognizer Command=... />
```

```
    <!-- érintés hatására fut a parancs -->
```

Xamarin alapismeretek

Méret kezelés

- A mobil eszközök képernyőmérete jelentősen eltérhet, és az alkalmazásunknak alkalmazkodnia kell a teljes képernyős megjelenítéshez
 - a vezérlők, szövegek méretezésénél használjunk relatív méreteket, pl.:

```
<StackLayout HorizontalOptions="Fill"
                VerticalOptions="Fill" ...>
  <!-- az elrendező kitölti a képernyőt -->
  <Label ... FontSize="Medium">
  <!-- a szövegméret közepes lesz -->
```
- lehetőségünk méretezni bármely vezérlőt a **Scale** tulajdonság segítségével

Xamarin alapismeretek

Méret kezelés

- a méretezést legcélszerűbb gyerekvezérlőkre alkalmazni (**ContentView**, **Layout**, **ContentPage** leszármazottakban)
 - kezelhető a vezérlő átméretezése (**OnSizeAllocated**)
 - kezelhető a tartalom átméretezése (**Content.SizeChanged**)
- pl.:

```
private void Content_SizeChanged(...) {  
    // ha változik a tartalom mérete, akkor  
    // méretezzük  
    Double height = Height / Content.Height;  
    Double width = Width / this.Content.Width;  
  
    if (width > 0 && height > 0)  
        Content.Scale = Math.Min(height, width);  
}
```

Xamarin alapismeretek

Tájolás kezelés

- A mobil eszközök többféle tájolásban helyezkedhetnek el
 - általában az portré/tájkép (álló/fekvő) tájolásokat különböztetjük meg, de ezeknek lehetnek speciális esetei
 - a támogatott tájolásokat eszközönként adhatjuk meg (korlátozhatjuk)
 - Android esetén a főtevékenység (**MainActivity**) egyik jellemzője a támogatott tájolás (**ScreenOrientation**)
 - Windows esetén az alkalmazás leírója (*application manifest*) tartalmazza a tájolást
 - az eszköz tájolását és az alkalmazás képernyőjének méretét egyszerre célszerű szabályozni a vezérlő méretváltogatásának kezelésével (**OnSizeAllocated**)

Xamarin alapismeretek

Tájolás kezelés

- pl.:

```
protected override void OnSizeAllocated(
    Double width, Double height)
    // megkapjuk az aktuális
    // szélességet/magasságot
    {
        base.OnSizeAllocated(width, height);

        // orientáció meghatározása
        if (width > height)
            ... // tájkép
        else
            ... // portré
    }
```

Xamarin alapismeretek

Eszközkezelés

- Az alkalmazások felhasználhatóak táblagépes, illetve mobil környezetben is, és mindkét környezetben megfelelő megjelenítést kell biztosítanunk

- kódból lekérhetjük az eszköz típusát (`Device.Idiom`), és arra megfelelően reagálhatunk, pl.:

```
if (Device.Idiom == TargetIdiom.Phone) {  
    image.Source =  
        ImageSource.FromFile("small.jpg");  
} else {  
    image.Source =  
        ImageSource.FromFile("large.jpg");  
}  
// táblagépen nagyobb képet használunk
```

Xamarin alapismeretek

Eszközkezelés

- nézetből az eszköznek megfelelő értékeket adhatjuk át (**OnIdiom**), ehhez meg kell adnunk az érték típusát (**x:TypeArguments**), illetve a két változatot (**Phone**, **Tablet**), pl.:

```
<Label ...>
```

```
  <Label.FontSize>
```

```
    <!-- a betűméret eszközfüggő lesz -->
```

```
    <OnIdiom x:TypeArguments="x:Double"
```

```
      Phone="10" Tablet="30"/>
```

```
    <!-- telefon esetén 10-es betűméretet  
      használunk, táblagép esetén  
      30-ast -->
```

```
  </Label.FontSize>
```

```
</Label>
```

Xamarin alapismeretek

Triggerek

- Lehetőségünk van tevékenységeket deklaratív módon, triggerek segítségével megfogalmazni a nézetben
 - triggerek megadhatók vezérlőkben és stílusokban
 - triggerek segítségével reagálhatunk
 - valamely vezérlő tulajdonságának megváltozására (*property trigger*), pl.:

```
<Trigger TargetType="Entry"
          Property="IsFocused" Value="True">
  <!-- ha fókuszba kerül a szövegdoboz -->
  <Setter Property="BackgroundColor"
          Value="Yellow" />
  <!-- a háttére sárga lesz -->
  ...
```

Xamarin alapismeretek

Triggerek

- a kötött adat megváltozására (*data trigger*), pl.:

```
<DataTrigger TargetType="Button"
    Binding="{Binding Count}" Value="0">
    <!-- ha a nézetmodell Count tulajdonsága
    0-ra vált -->
    <Setter Property="Text" Value="Empty" />
    <!-- a felirata Empty lesz -->
```

- eseményre (*event trigger*), amelynek hatására valamilyen akciót (*trigger action*) hajtunk végre, pl.:

```
<EventTrigger Event="TextChanged">
    <local:NumberValidateAction />
</EventTrigger>
```

- a triggerek kombinálhatóak (*multi trigger*)

Xamarin alapismeretek

Triggerek

- a trigger akciók (**TriggerAction<>**) olyan osztályok, amelyben az eseményre adott reakciót (**Invoke**) írhatjuk le, pl.:

```
public class NumberValidateAction
    : TriggerAction<Entry>
    // megadjuk az érintett vezérlő típusát
{
    protected override void Invoke (Entry entry) {
        // megadjuk a tevékenységet
        double result;
        entry.TextColor =
            Double.TryParse(entry.Text, out result)
            ? Color.Default : Color.Red;
        // ha nem szám, piros lesz a betűszín
    }
    ...
}
```

Xamarin alapismeretek

Példa

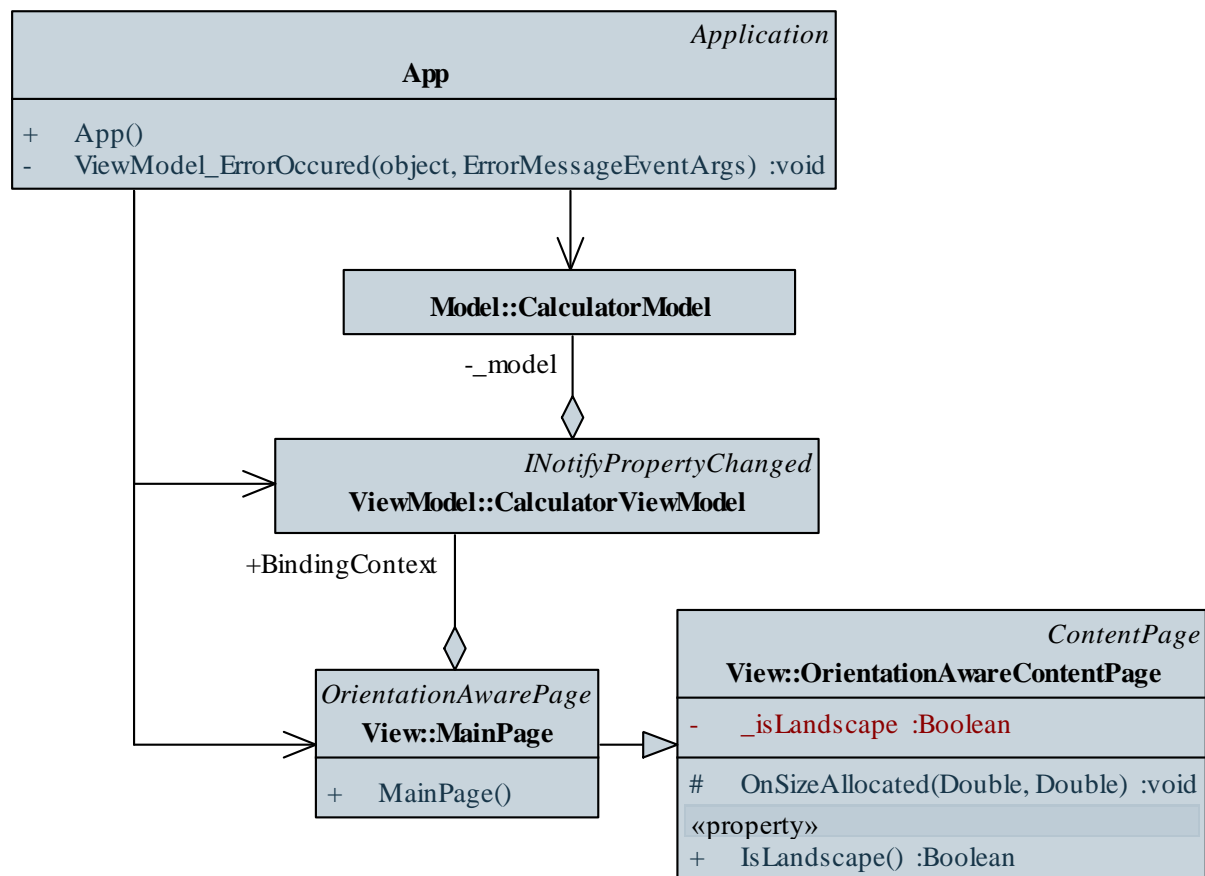
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- alakítsuk át a nézetet, hogy könnyen használható legyen mobil környezetben, és alkalmazkodjon a platformhoz
- szövegbevitel helyett gombokat használunk a számokhoz, ezért ki kell egészítenünk a nézetmodellt a számok kezelésével (**Calculate**)
- megvalósítunk egy új laptípust (**OrientationAwareContentPage**), amely kezeli a tájolást (**IsLandscape**), így triggerek segítségével tudunk reagálni az elforgatásokra
- külön méreteket definiálunk mobilra és táblagépekre (**OnIdiom**)

Xamarin alapismeretek

Példa

Tervezés:



Xamarin alapismeretek

Példa

Megvalósítás (MainPage.xaml):

```
<view:OrientationAwareContentPage ...>
...
<Style x:Key="BasicButtonStyle"
      TargetType="Button">
  <Setter Property="FontSize">
    <Setter.Value>
      <OnIdiom x:TypeArguments="x:Double"
              Phone="15" Tablet="45" />
      <!-- reagálunk az eszközre -->
    </Setter.Value>
  </Setter>
...

```

Xamarin alapismeretek

Példa

Megvalósítás (MainPage.xaml):

```
<view:OrientationAwarePage ...>
...
<StackLayout>
  <StackLayout.Triggers>
    <!-- reagálunk az elforgatásokra -->
    <DataTrigger TargetType="StackLayout"
      Binding="{Binding Source={x:Reference
        ContentPage}, Path=IsLandscape}"
      Value="True">
      <Setter Property="Orientation"
        Value="Horizontal" />
    ...
  </view:OrientationAwarePage>
```

Xamarin alapismeretek

Időzítés

- Felületi időzítésre használhatjuk a platformfüggetlen időzítőt (**Device.StartTimer**)
 - megadhatjuk az időintervallumot, és a végrehajtandó tevékenység lambda-kifejezését
 - a tevékenység egy logikai függvény, amely amennyiben hamissal tér vissza, az időzítő leáll, pl.:

```
elapsedTime = 0;
```

```
Device.StartTimer(TimeSpan.FromSeconds(1), () =>
```

```
{
```

```
    entry.Text = ++elapsedTime;
```

```
    // eltelt másodpercek kiírása
```

```
    return elapsedTime < 120;
```

```
    // 2 percig fut az időzítő
```

```
});
```

Xamarin alapismeretek

Időzítés

- Üzleti logikában történő időzítésre használható a párhuzamosított időzítő (**System.Threading.Timer**)
 - példányosításkor megadhatjuk a tevékenységet, egy felügyelt állapotot, a kezdési időt és az intervallumot, pl.:

```
elapsedTime = 0;  
timer = new Timer(state =>  
    { entry.Text = ++elapsedTime; },  
    state, 0, 1000); // 1 másodpercenként fut le
```
 - megsemmisítéssel (**Dispose**) leállíthatjuk az időzítőt
 - az időzítőt nem kell szinkronizálni a felülettel
- Amennyiben eseményvezérelt időzítőre van szükségünk (pl. **System.Timers.Timer**), a funkcionalitást becsomagolhatjuk egy annak megfelelő típusba

Xamarin alapismeretek

Példa

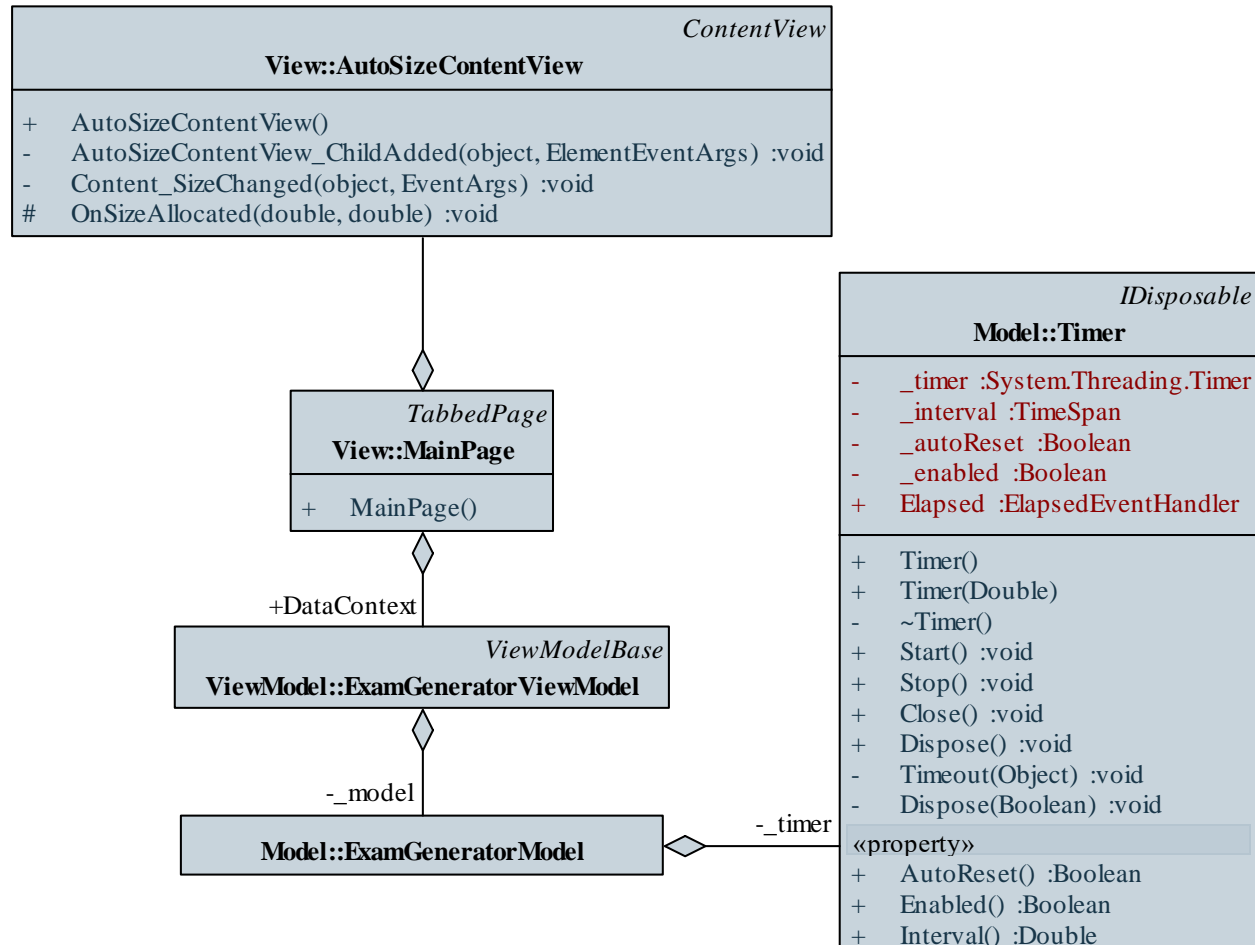
Feladat: Készítsünk egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- a megjelenítéshez két lapot (**ContentPage**) használunk, amelyek között lapozunk (**TabbedPage**)
- az első lapon csak a tétel sorszámát jelenítjük meg, és érintéssel tudjuk generálni a sorszámot (**TapGestureRecognizer**), a második lapon elhelyezzük a beállításokat
- a sorszám méretét automatikusan méretezzük, ehhez egy saját vezérlőt veszünk fel (**AutoSizeContentView**), amely méretezi a tartalmát
- megvalósítunk egy saját időzítőt (**Timer**), amely teljes egészében megfelel az üzleti logikában használt időzítőnek (az időzített eseményt már nem kell szinkronizálnunk a nézetmodellben)

Xamarin alapismeretek

Példa

Tervezés:



Xamarin alapismeretek

Példa

Megvalósítás (MainPage.xaml):

```
<TabbedPage ...>
```

...

```
<view:AutoSizeContentView>
```

```
<!-- automatikusan méreteződik a tartalom -->
```

```
<Label Text="{Binding QuestionNumber}" ...>
```

```
<Label.GestureRecognizers>
```

```
<!-- érintés kezelése -->
```

```
<TapGestureRecognizer
```

```
Command="{Binding StartStopCommand}" />
```

```
<!-- érintés -->
```

```
</Label.GestureRecognizers>
```

```
</Label>
```

```
</view:AutoSizeContentView >
```

...

Xamarin alapismeretek

Külső vezérlők

- Az egyedileg létrehozott vezérlők mellett számos programcsomag tartalmaz Xamarin Forms vezérlőt
 - XAML-ben a külső forrásban definiált vezérlő esetén be kell töltenünk a szerelvényt (*assembly*)
- Amennyiben dinamikusan méretezhető rácsot szeretnénk megvalósítani, használhatjuk a **DLToolkit.Forms.Controls** csomagot, amely tartalmaz egy **FlowListView** vezérlőt
 - szabályozható az egy sorban megjelenített vezérlők száma (**FlowColumnCount**)
 - szabályozható a megjelenített vezérlő (**FlowColumnTemplate**)
 - külön kell jelezni a vezérlő inicializálását (**FlowListView.Init()**)

Xamarin alapismeretek

Külső vezérlők

- Pl.:

```
<ContentPage ...
  xmlns:controls="clr-namespace:
    DLToolkit.Forms.Controls;
    assembly=DLToolkit.Forms.Controls.FlowListView">
<!-- külső szerelvény betöltése -->
...
<controls:FlowListView ...>
  <!-- külső vezérlő felhasználása -->
  <controls:FlowListView.FlowColumnTemplate>
    <DataTemplate>..</DataTemplate>
  </controls:FlowListView.FlowColumnTemplate>
</controls:FlowListView>
...
```

Xamarin alapismeretek

Példa

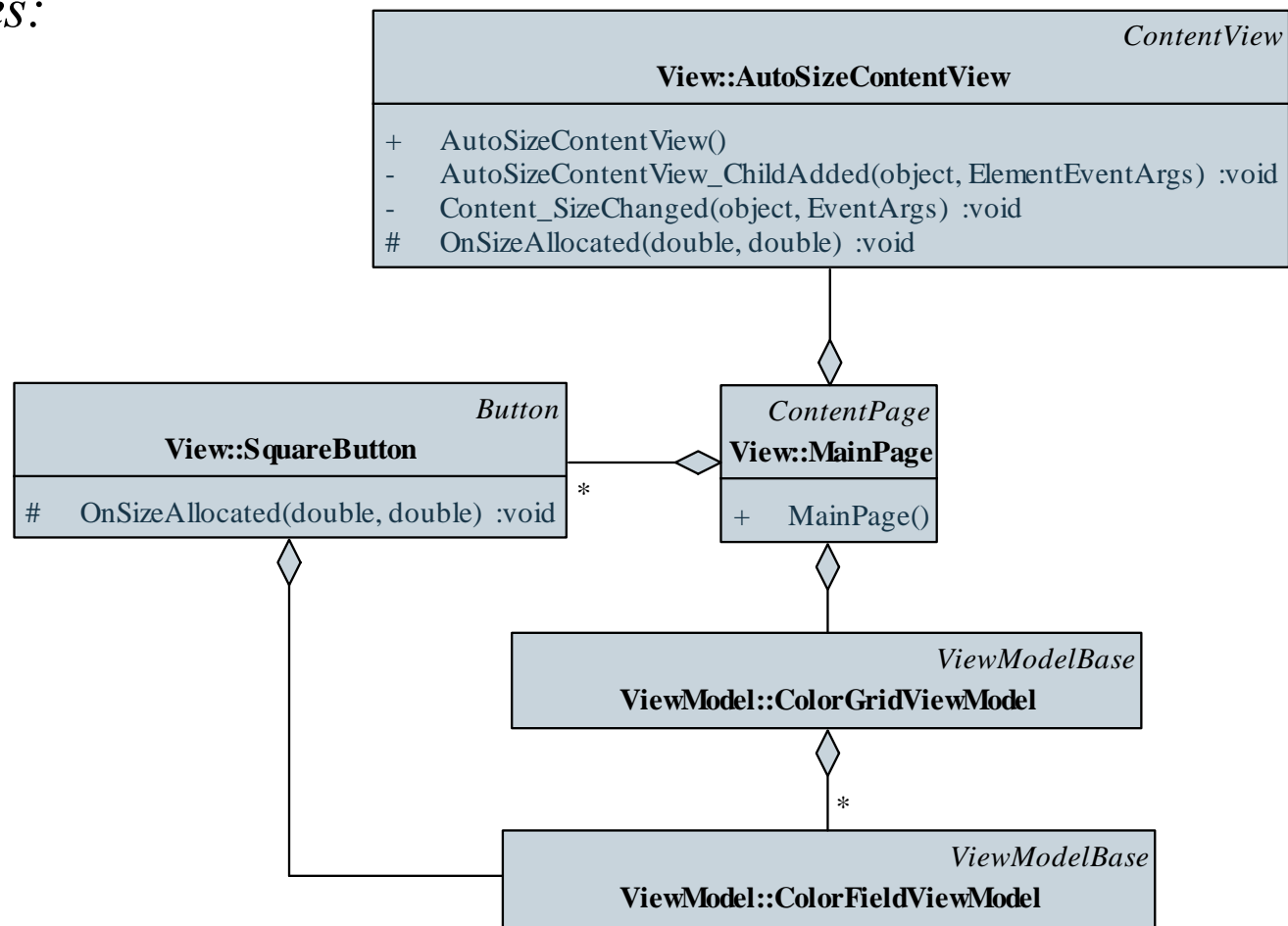
Feladat: Készítsünk egy dinamikus méretezhető táblát, amely három szín között (piros, fehér, zöld) állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- elhasználjuk a rendelkezésre álló a nézetmodell (**ColorGridViewModel**, **ColorFieldViewModel**), a színváltást trigger segítségével vezéreljük
- használjuk a **FlowLayoutView** vezérlőt a rács megjelenítéséhez, amelyet elhelyezünk az automatikusan méretező nézetben (**AutoSizeContentView**)
- annak érdekében, hogy a mezők szélessége megegyezzen a magassággal, létrehozunk egy egyedi, négyzet alakú gombot (**SquareButton**), ahol a magasságot a szélességgel tesszük egyenértékűvé (**OnSizeAllocated**)

Xamarin alapismeretek

Példa

Tervezés:



Xamarin alapismeretek

Példa

Megvalósítás (MainPage.xaml):

```
<view:AutoSizeContentView Grid.Row="3">
  <!-- automatikusan méreteződő tartalom -->
  <controls:FlowListView
    FlowItemsSource="{Binding Fields}"
    FlowColumnCount="{Binding ColumnCount}" ...>
  <!-- külső vezérlő, amelyből rácsot készítünk
    -->
  <controls:FlowListView.FlowColumnTemplate>
    <DataTemplate>
      <view:SquareButton ... />
      <!-- a rácsot négyzetes gombokkal töltjük
        fel -->
    ...
  </controls:FlowListView.FlowColumnTemplate>
</view:AutoSizeContentView>
```