Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások

---

6. előadás

Windows Presentation Foundation (WPF) alapismeretek

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

WPF alapismeretek

Tulajdonságai

- A *Windows Presentation Foundation* (*WPF*) a .NET környezet vektoros alapú grafikus felületi rendszere
  - lehetővé teszi a 3D grafikus kártyák kihasználását (*Direct3D*)
  - jóval nagyobb testre szabhatóságot biztosít (megjelenítés és stílusok átdefiniálási lehetősége, megjelenítési tulajdonságok erőforrás-alapú tárolása)
  - lehetőséget ad a felület deklaratív leírására (*XAML*)
  - függetleníti a megjelenést és a vezérlést, így jelentősen javít az alkalmazás architektúrán (*MVVM*)
  - hátránya, hogy csak Windows rendszerekre érhető el

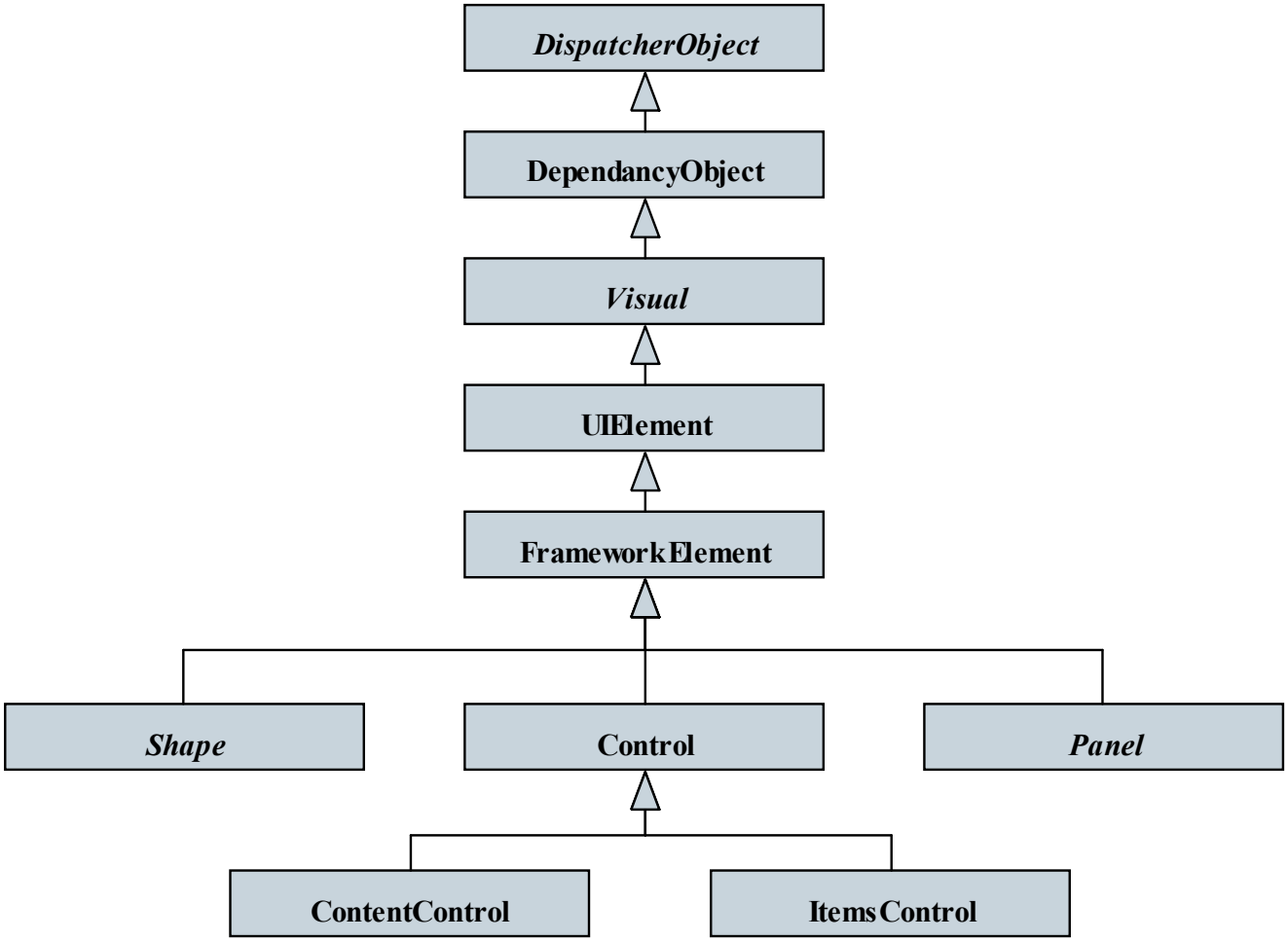
WPF alapismeretek

Felépítés

- A WPF grafikus felület vektoros grafikus elemekből épül fel
  - az elemek (**`UIElement`**) lehetnek vezérlők (**`Control`**), alakzatok (**`Shape`**), gyűjtőelemek (**`Panel`**), …
  - az osztályok a **`System.Windows`** névtérben helyezkednek el
  - az elemek grafikailag összetettek, alapértelmezés szerint hasonlítanak a Windows vezérlőkre, de ez módosítható
- Az alkalmazások futása, és a kirajzolás folyamata jóval összetettebb
  - a képalkotást külön szál (*rendering thread*) végzi az elemkezeléstől (*dispatcher thread*), utóbbi egy prioritásos üzenetciklussal kezeli az elemeket (**`DispatcherObject`**)

WPF alapismeretek

Felépítés



WPF alapismeretek

Az XAML nyelv

- Az *eXtensible Application Markup Language* (*XAML*) olyan XML alapú deklaratív nyelv, mely biztosítja a grafikus felület teljes leírását
  - lehetőséget ad 2D/3D elemek, transzformációk, animációk, valamint további effektek leírására
  - pl.:

```
<Canvas Name="myCanvas"> <!-- vászon -->
   <Label Name="myLabel" BorderBrush="Red">
      <!-- címke a vászonban -->
      Hello World! <!-- címke tartalma
                        (nem csak szöveg lehet) -->
   </Label> <!-- címke vége -->
</Canvas> <!-- vászon vége -->
```

WPF alapismeretek

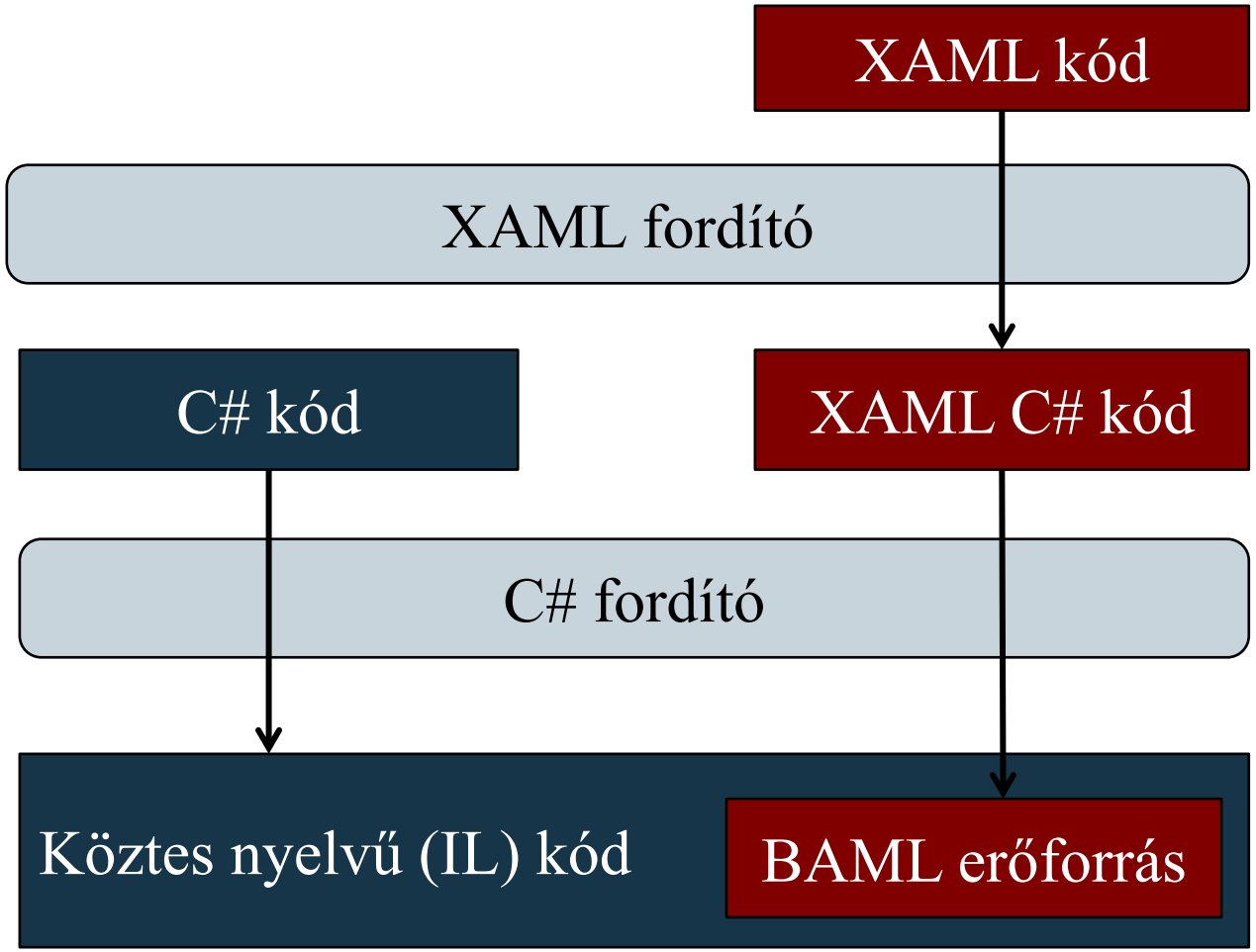
Az XAML nyelv

- Minden XAML elemtípus megfeleltethető egy .NET osztálynak, és a deklaratív leírás imperatív kódnak
  - így minden, amit XAML-ben leírunk, leírható kóddal is, és dinamikusan is létrehozhatunk vezérlőket
  - pl.:

```
Canvas myCanvas = new Canvas(); // vászon
Label myLabel = new Label(); // címke
myLabel.Content = "Hello World!"; // tartalom
myLabel.BorderBrush = Brushes.Red; // szegély
myCanvas.Children.Add(myLabel); // behelyezés
```

- Az XAML kód átalakul *BAML* (*Binary XAML*) formátumra, amely erőforrásként csatolható a felügyelt kódhoz

Az XAML fordítása



WPF alapismeretek

Az XAML felépülése

- Az XAML attribútumokkal, illetve tartalmazással írja le a tulajdonságokat és a magukban foglalt elemeket, pl.:

```
<Grid> <!-- elemek tároló rács -->
    <Grid.RowDefinitions> <!-- rács felépítés -->
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        …
    <Label Content="Enter Name: " Grid.Row="0"
           Grid.Column="0" />
        <!-- címke az 1. sor 1. oszlopában -->
    <TextBox Grid.Row="0" Grid.Column="1"
             MinWidth="50"/> <!-- szövegdoboz -->
</Grid> <!-- rács vége -->
```

WPF alapismeretek

Ablakok

- Az ablakok a **`Window`** osztály leszármazottai, amelyek parciális osztályként rendelkeznek felületi kóddal (**`.xaml`**), valamint háttérkóddal (**`.xaml.cs`**)
  - a felületi kódban adjuk meg a deklaratív leírást, pl.:

```
<Window x:Class="MyApplication.MyWindow" …
    Title="My Window" Height="350" Width="525">
    <!- megadjuk címét és méreteit -->
    <Grid> … </Grid>
    <!--- rács a további elemeknek -->
</Window>
```

  - meg kell adnunk az osztálynevet (az **`x:Class`**), valamint a felhasznált sémákat és névtereket

WPF alapismeretek

Ablakok

- az ablakba csak egy elem helyezhető (ez általában rács, vagy vászon, amely további elemeket tartalmaz)
- a háttérkódban írhatjuk meg a további tevékenységeket, pl. eseménykezelők
- az eseménykezelő társítás történhet a háttérkódban (**+=**), illetve a felületi kódban is, pl.:

```
// MyWindow.xaml:
<Button Name="myButton" Click="myButton_Click">
    <!-- gomb eseménykezelő társítással -->

// MyWindow.xaml.cs:
void myButton_Click(…) { … }
```

WPF alapismeretek

Ablakok és alkalmazások

- minden felületi kódot a konstruktor fog lefuttatni az **`InitializeComponent()`** művelet segítségével, pl.:

```
partial class MyWindow { // háttérkód osztálya
    public MyWindow() {
        InitializeComponent(); …
    }
}
```

- Az alkalmazást egy **`Application`** leszármazott osztály vezérli, amely szintén megadható XAML segítségével, pl.:

```
<Application x:Class="MyApplication.App" …
    StartupUri="MainWindow.xaml">
    <!-- megadjuk a kezdőablakot -->
</Application>
```

WPF alapismeretek

Példa

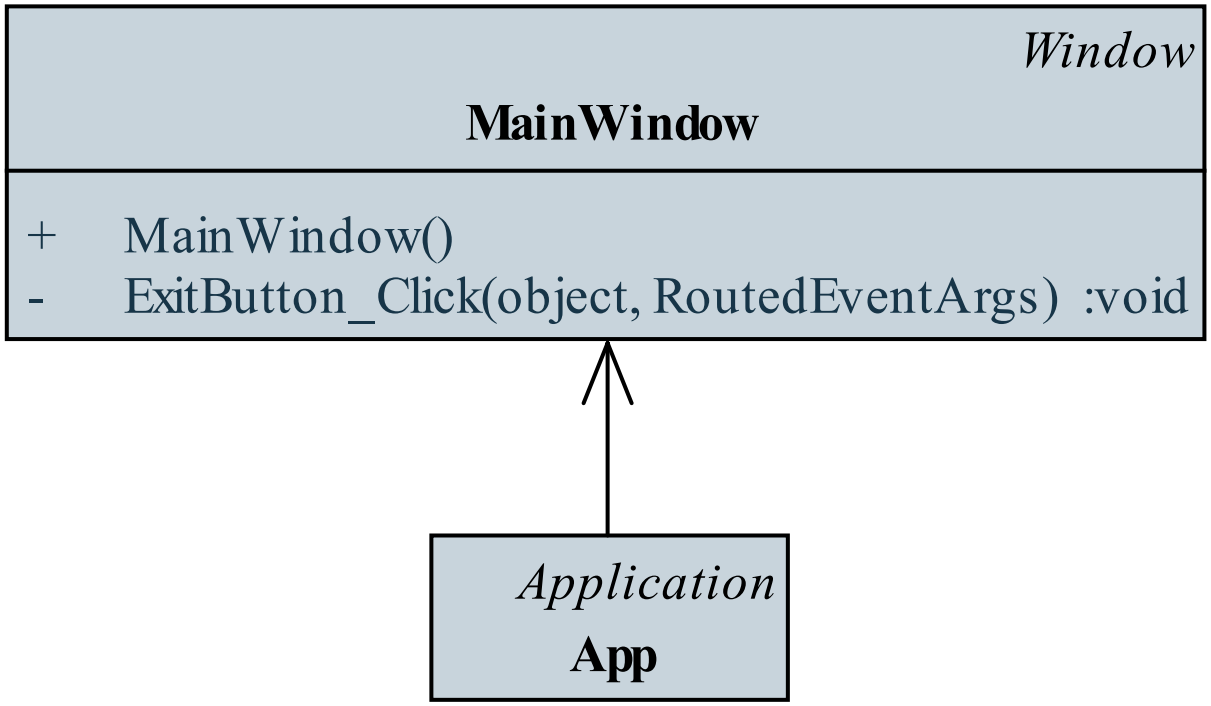
Feladat: Készítsünk egy egyszerű programot, amelyben egy ablak közepére helyezünk egy kilépésre szolgáló gombot.

- a programot készítsük el deklaratív leírás, illetve tisztán kód használatával
- deklaratív leírás esetén csak az eseménykezelő függvényt kell megírnunk a kódban, amelynek feladata az ablak bezárása (**`Close`**)
- kódban történő megvalósítás esetén felparaméterezzük az alkalmazást a főprogramban, és megvalósítjuk az indítás (**`Application_Startup`**) és befejezés (**`Application_Exit`**) eseménykezelését, továbbá a saját ablak osztály (**`MainWindow`**) konstruktorában definiáljuk a megjelenést

WPF alapismeretek

Példa

Tervezés:



Példa

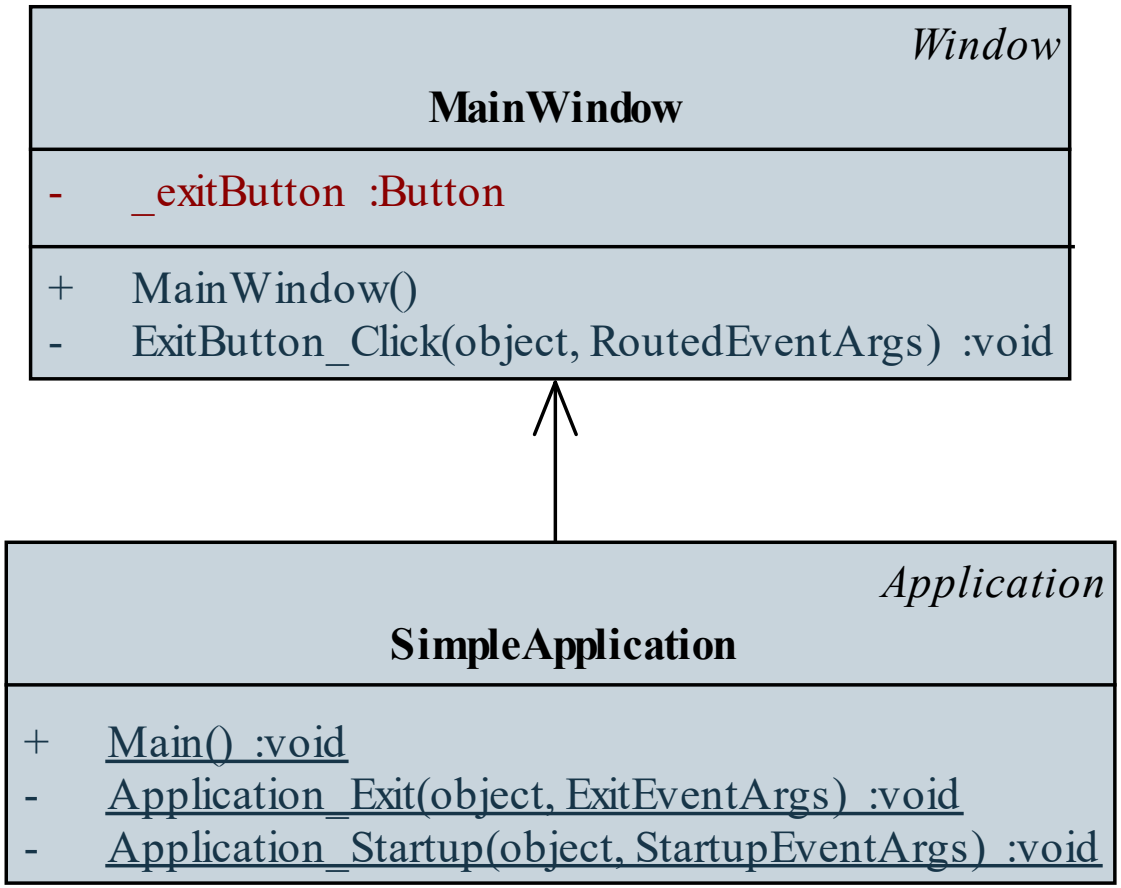
Megvalósítás (`MainWindow.xaml`*):*

```
<Window x:Class="ELTE.SimpleWindowByDesign.
                 MainWindow"
    …
    Title="Egyszerű ablak" Height="200" Width="300"
    WindowStartupLocation="CenterScreen">
    <Grid>
       <Button Name="_ExitButton" Content="Kilépés"
          HorizontalAlignment="Center"
          VerticalAlignment="Center"
          Height="25" Width="100"
          Click="ExitButton_Click" />
     </Grid>
</Window>
```

WPF alapismeretek

Példa

Tervezés:



WPF alapismeretek

Példa

Megvalósítás (`MainWindow.cs`*):*

```
class MainWindow : Window {
   // a Window osztály leszármazottja
   …
   public MainWindow(){
      Width = 300;
         // ablak tulajdonságainak beállítása
      …
      _exitButton = new Button();
         // gomb létrehozása és felkonfigurálása
      …
      AddChild(_exitButton);
          // gomb felvétele az ablakra
      …
```

WPF alapismeretek

Példa

*Megvalósítás (**SimpleApplication.cs**):*

```
class SimpleApplication : Application
{
  public static void Main() {
    SimpleApplication app = new SimpleApplication();
    app.Startup += Application_Startup;
    app.Run(); // futtatás
  }

  static void Application_Startup(object sender,
                                  StartupEventArgs e) {
    MainWindow window = new MainWindow();
    window.Show(); // megjelenítjük az ablakot
  }
}
```

WPF alapismeretek

Vezérlők

- A Windows Forms-ban megszokott vezérlőket jórészt megtalálhatjuk a WPF-ben is (esetlegesen más néven)
  - általában jóval szélesebb körben személyre szabhatóan
- Fontosabb tulajdonságok:
  - objektumnév (**`Name`**, **`x:Name`**)
  - erőforrások (**`Resources`**)
  - sablon (**`Template`**), amellyel több vezérlő tulajdonságait tudjuk közösen állítani
  - kinézet (**`Background`**, **`Foreground`**, **`BorderBrush`**, **`BorderThickness`**, …)

- betűkezelés (**`FontFamily`**, **`FontSize`**, **`FontStretch`**, …)
- kurzorkinézet (**`Cursor`**)
- pozícionálás és méretezés (**`Width`**, **`ActualWidth`**, **`MaxWidth`**, **`Padding`**, **`Margin`**, **`VerticalAlignment`**, **`VerticalContentAlignment`**, **`RenderTransform`**, …)
- engedélyezettség (**`IsEnabled`**), láthatóság (**`IsVisible`**), fókuszáltság (**`IsFocused`**)
- tabulátorkezelés (**`TabIndex`**, **`IsTabStop`**)

- A vezérlők eseményei is jórészt megegyeznek a Windows Forms eseményekkel, így tartalmazzák a különböző egér-/billentyűállapotok kezelését, a tulajdonságok változását, stb.

WPF alapismeretek

Példa

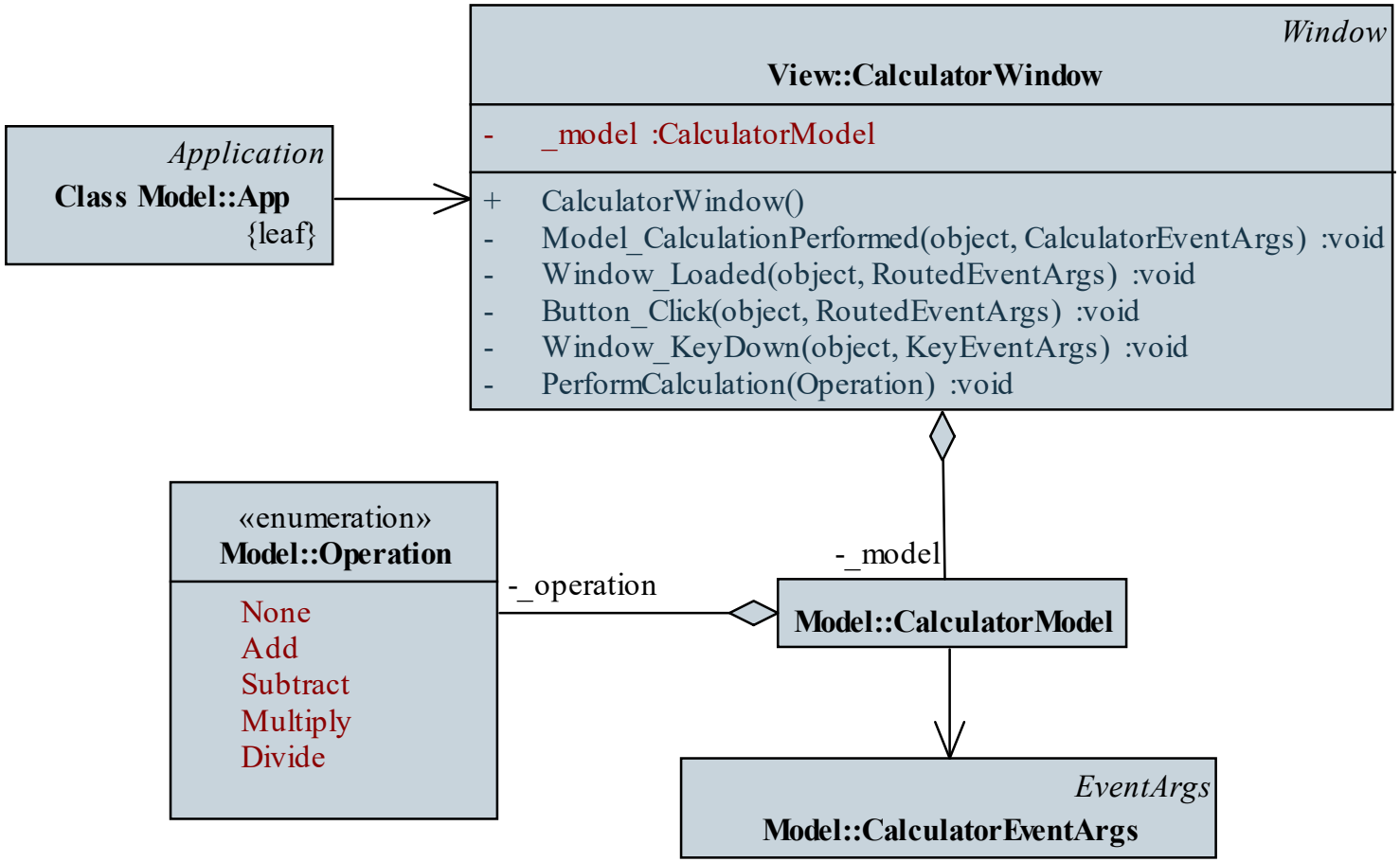
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- az alkalmazást modell/nézet architektúrában valósítjuk meg
- a modell (**`CalculatorModel`**) biztosítja a műveletek végrehajtását, eseménnyel jelzi az eredmény megváltozását
- a nézet (**`CalculatorWindow`**) példányosítja a nézetet, és gombokon keresztül biztosítja a műveletek végrehajtását (**`Button_Click`**), továbbá kezeli a billentyűzet eseményeit is (**`Window_KeyDown`**)
  - az elemeket magasság (**`Height`**), illetve margó (**`Margin`**) megadásával pozícionáljuk

WPF alapismeretek

Példa

Tervezés:



WPF alapismeretek

Példa

Megvalósítás (`CalculatorWindow.xaml`*):*

```
<Window x:Class="ELTE.Windows.
                    Calculator.View.CalculatorWindow"
    …
    Title="Calculator" Height="280" Width="275" …
    Loaded="Window_Loaded"
    KeyDown="Window_KeyDown">
    <Grid>
        <TextBox Name="_textNumber" Height="42"
           VerticalAlignment="Top" FontSize="28"
           TextAlignment="Right" FontWeight="Bold"
           />
        …
</Window>
```

WPF alapismeretek

Vezérlők

- Sok vezérlő tartalmazhat további vezérlő(ke)t, illetve grafikus elemeket, így:
  - a **`ContentControl`** leszármazottai tartalmazhatnak egy másik elemet a **`Content`** mezőjükben, pl.:

```
<Button … >
    <Image Source="…" />
    <!-- a vezérlő Content értékét töltjük fel
         egy képpel -->
</Button>
```

  - az **`ItemsControl`** leszármazottai tetszőlegesen sok elemet tartalmazhatnak (pl. **`ListBox`**, **`ListView`**, **`ComboBox`**)
  - a vezérlőknek lehetnek fejlécei is (pl. **`GroupBox`**, **`TreeView`**)

WPF alapismeretek

Vezérlők elhelyezése

- A vezérlők elhelyezése több tényezővel vezérelhető:
  - igazítás (**`VerticalAlignment`**, **`HorizontalAlignment`**)
  - külső margó (**`Margin`**, a vezérlő széle és a tartalmazó elem között) és belső margó (**`Padding`**, a vezérlő tartalma és széle között)
  - méret (**`Width`**, **`Height`**), korlátok (**`MinWidth`**, **`MaxWidth`**) valamint lekérdezhető az aktuális érték is (**`ActualWidth`**, **`ActualHeight`**)
  - túlfutás kezelése (**`ClipToBounds`**)
- A vezérlők nézetdobozba (**`Viewbox`**) helyezhetőek, amely automatikusan méretezi tartalmát

WPF alapismeretek

Vezérlők elhelyezése

- Több vezérlő elhelyezése panelek (**`Panel`**) segítségével történik, amelynek leszármazottai:
  - *vászon* (**`Canvas`**), amelyben a bal felső sarokhoz viszonyított koordinátarendszert használhatunk
  - *rács* (**`Grid`**), amelyben szabályozható a sorok és oszlopok mérete, illetve lehet egységes rács (**`UniformGrid`**)
  - igazító panelek (**`StackPanel`**, **`WrapPanel`**, **`DockPanel`**)
- Egyik elhelyezés sem görgethető, de behelyezhető görgetett területbe (**`ScrollViewer`**)
- Az egyes elhelyezések automatikusan különböző elhelyezési tulajdonságokat vesznek figyelembe a beágyazott elemeken

WPF alapismeretek

Vezérlők elhelyezése és megjelenése

- A vezérlőkre különböző transzformációk alkalmazhatóak: forgatás (**`RotateTransform`**), nagyítás (**`ScaleTransform`**), eltolás (**`TraslateTransform`**), ferdítés (**`SkewTransform`**)
  - a transzformációk csoportosíthatóak (**`TransformGroup`**)
- A vezérlők megjelenése számos módon testre szabható
  - a legtöbb vezérlőnél külön kezelhető a határvonal (**`Border`**/**`Stroke`**), illetve a kitöltés (**`Background`**/**`Fill`**), valamint a különböző hatások (**`Effect`**)
  - a színekhez különböző ecsetek használhatóak (pl. **`SolidColorBrush`**, **`LinearGradientBrush`**)
  - a megjelenítés stílusba (**`Style`**) foglalható

WPF alapismeretek

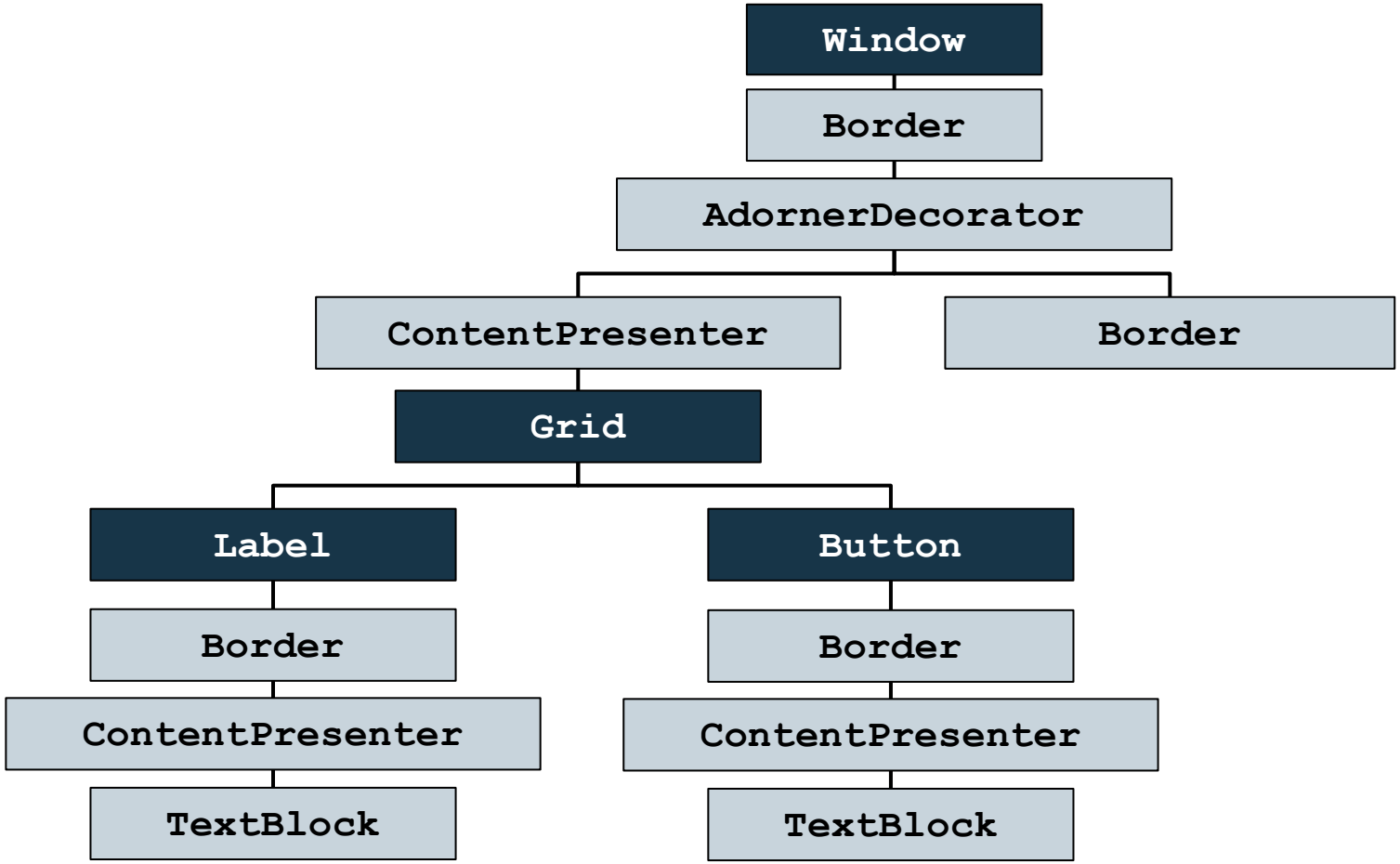
Vezérlők megjelenése

- A megjelenő vezérlők összetettek, több elemből állnak
  - az egyes elemek különböző tulajdonságokat szolgáltatnak a teljes vezérlő számára
  - a *logikai fa* írja le az elemek közötti kapcsolatokat, a *vizuális fa* írja a logikai elemek összes alkotóelemének kapcsolatát (pl. elhelyezés, áttetszőség, engedélyezettség)
  - a felépítés a *sablon*nal (**`ControlTemplate`**) szabályozható
  - pl.:

```
<Window><Grid> <!-- ablakba helyezett rács -->
    <Label … /> <!-- címke -->
    <Button … /> <!-- gomb -->
</Grid></Window>
```

WPF alapismeretek

Vezérlők megjelenése



WPF alapismeretek

Képkezelés

- A képek kezelését a memóriában több osztály segítségével is végezhetjük, amelyek speciális eszközöket biztosítanak
  - alapvető képtípus a **`BitmapImage`**, amely felhasználható a különböző felületi elemeken (pl. **`Image`** vezérlő, vagy **`ImageBrush`** ecset)
  - amennyiben pixelszintű manipulációt szeretnénk, a **`WritableBitmap`** biztosít írási/olvasási lehetőségeket
  - ügyelnünk kell arra, hogy a WPF-ben már minden elérési útvonal Uri segítségével van megfogalmazva, pl.:

```
Uri iUri = new Uri(@"Images\smiley.png",
                   UriKind.Relative);
BitmapImage bImage = new BitmapImage(iUri);
```

WPF alapismeretek

Elemi grafika

- Lehetőségünk van elemi alakzatok rajzolására rajzeszköz (**`DrawingContext`**) segítségével
  - a rajzoláshoz számtalan rajzolómetódus használható (pl. **`DrawRectangle`**, **`DrawText`**, **`DrawImage`**, **`DrawVideo`**)
  - a rajzobjektumot egy kezelőre (pl. **`DrawingGroup`**) kell ráállítani, azt pedig egy rajzfelületre (pl. **`DrawingImage`**)
  - igazából nem rajzol, hanem utasításokat állít össze a 3D rendereléshez, és lehetőség van állapotkezelésre is
- Az elemi rajzolás használata nem javasolt, mivel a primitív alakzatok (**`Rectangle`**, **`Ellipse`**, …) már osztályként meg vannak valósítva, ezért használatuk egyszerűbb és gyorsabb

- Pl.:

```
Image myImage = new Image(); // képmegjelenítő
DrawingGroup drGroup = new DrawingGroup();
   // rajzkezelő
using (DrawingContext dx = drGroup.Open())
{  // rajzeszköz létrehozása
   Pen myPen = new Pen(Brushes.Black, 2); // toll
   dx.DrawRectangle(Brushes.Blue, myPen,
                    new Rect(0, 0, 25, 25));
   …
}
DrawingImage img = new DrawingImage(drGroup);
   // rajzfelület
myImage.Source = img; // kirajzolás
```

WPF alapismeretek

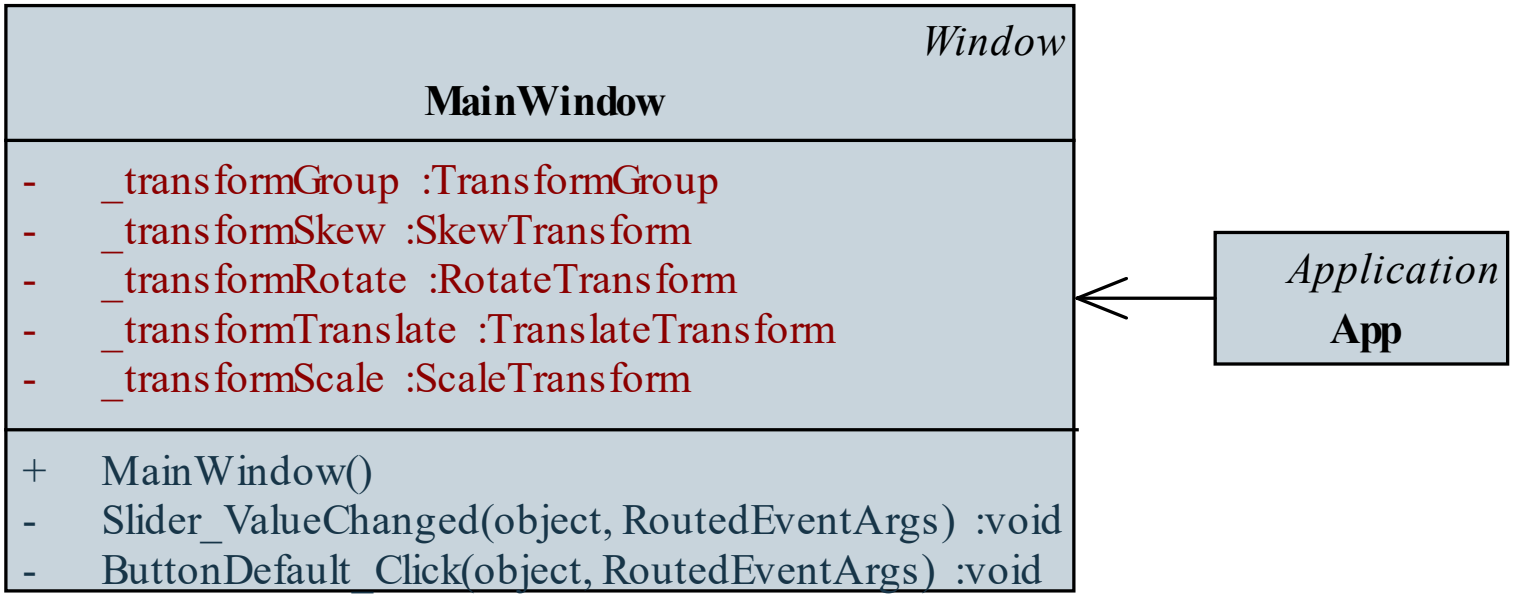
Példa

Feladat: Készítsünk egy programot, amellyel egy képet tudunk transzformálni.

- a képet egy **`ImageBrush`** objektumban helyezzük egy négyzetben (**`Rectangle`**) a képernyő közepén
- a négyzetre definiálunk egy transzformációs csoportot, amely a transzformáció-típusok egy-egy példányát tartalmazza
- ezek paramétereit szabályozzuk csúszkák (**`Slider`**) segítségével, amelyekhez közös eseménykezelőt (**`Slider_ValueChanged`**) rendelünk
- felveszünk egy gombot, amivel az alapértelmezett értékeket visszaállíthatjuk

Példa

Tervezés:



Példa

*Megvalósítás (***MainWindow.xaml.cs***):*

```
public partial class MainWindow : Window {
    private TransformGroup _transformGroup;
        // transzformációs csoport
    private SkewTransform _transformSkew;
    …

    public MainWindow(){
        …
        _transformGroup.Children.Add(
            _TransformSkew);
            // felvesszük a transzformációs csoport
            // elemeit
        …
```

WPF alapismeretek

Példa

Megvalósítás (`MainWindow.xaml.cs`*):*

```
        _rectangleSmiley.RenderTransform =
            _transformGroup;
            // transzformációk megadása
        …
    }

    protected void Slider_ValueChanged(object
                          sender, RoutedEventArgs e) {
        _transformRotate.Angle =
            _sliderRotateAngle.Value;
            // transzformációk értékeinek megadása
        …
```

WPF alapismeretek

Függőségi tulajdonságok

- A WPF bevezette a tulajdonság egy speciális változatát, a *függőségi tulajdonság*ot (*dependency property*)
  - lehetővé teszi, hogy egy adott objektum tulajdonságait más objektumon keresztül definiáljuk, és úgy szabjunk rá értéket, hogy az környezettől függően változzon
  - a **`DependencyObject`** statikus **`GetValue`** és **`SetValue`** metódusaival kezelhetőek a tulajdonság átadásával, amely statikus tulajdonságként definiált
  - a legtöbb tulajdonság WPF-ben függőségi tulajdonság, XAML-ben is kihasználható
    - pl. lehetőséget ad a szülőelemek tulajdonságainak elérése, és beállítására

WPF alapismeretek

Függőségi tulajdonságok

- Pl.:

```
Canvas myCanvas = new Canvas();
Label myLabel = new Label();
myLabel.SetValue(Canvas.LeftProperty, 100);
myLabel.SetValue(Canvas.TopProperty, 50);
   // függőségi tulajdonságok beállítása
myCanvas.Children.Add(myLabel);
   // elem felvétele gyerekelemként
…
Grid myGrid = new Grid();
myLabel.SetValue(Grid.RowProperty, 1);
myLabel.SetValue(Grid.ColumnProperty, 3);
   // rácsban sort és oszlopot kell beállítanunk
myGrid.Children.Add(myLabel);
```

WPF alapismeretek

Függőségi tulajdonságok

- Pl.:

```
<Canvas Name="myCanvas"> <!-- vászon -->
    <Label Name="myLabel" Content="Hello!"
           Canvas.Left="100" Canvas.Top="50" />
    <!-- a címkében állítjuk be a vászonbeli
         pozíciót -->
…
<Grid Name="myGrid"> <!-- rács -->
    …
    <Label Name="myLabel" Content="Hello!"
           Grid.Row="1" Grid.Column="3" />
    <!-- rácsban sort és oszlopot kell
         beállítanunk -->
```

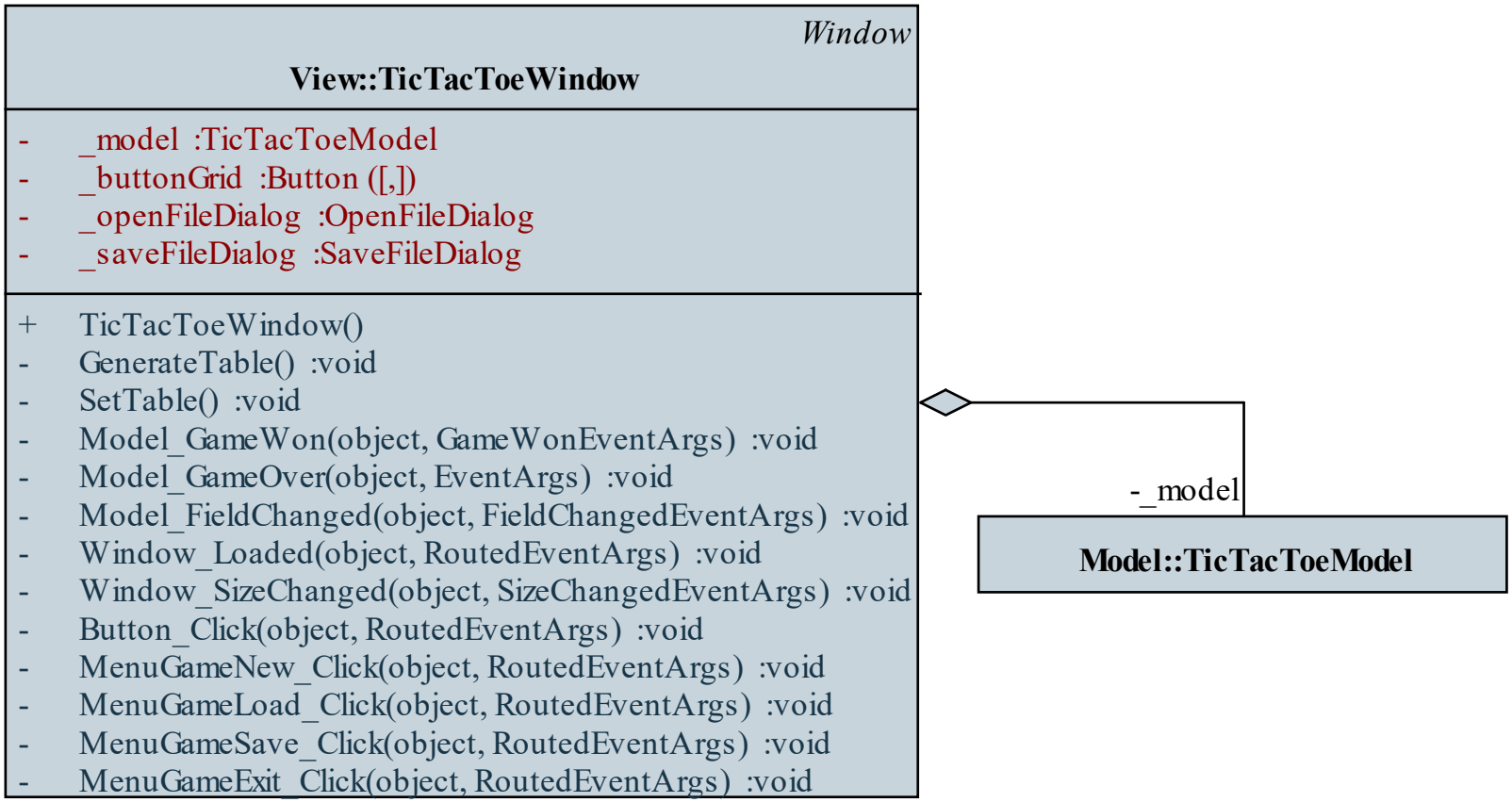
WPF alapismeretek

Példa

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

- megvalósítunk egy új felhasználói felületet WPF segítségével (**`TicTacToeWindow`**)
- a felületre felhelyezünk egy menüt (**`Menu`**), valamint egy rácsot (**`Grid`**) a játéktáblának, utóbbi tartalmát dinamikusan generáljuk, gombokból (**`Button`**) építünk mátrixot
  - mivel minden cella kitöltésre kerül, alternatív megoldásként **`UniformGrid`** is használható
- eseménykezelőket veszünk fel a betöltésre (**`Loaded`**), a méretváltásra (**`SizeChanged`**), a menüpontokra, a gombokra (**`Button_Clicked`**), valamint a modell eseményeire
- a fájl betöltés/mentés dialógusablakait a kódban hozzuk létre

Tervezés:


Window
View::TicTacToeWindow
- _model :TicTacToeModel
- _buttonGrid :Button ([,])
- _openFileDialog :OpenFileDialog
- _saveFileDialog :SaveFileDialog
+ TicTacToeWindow()
- GenerateTable() :void
- SetTable() :void
- Model_GameWon(object, GameWonEventArgs) :void
- Model_GameOver(object, EventArgs) :void
- Model_FieldChanged(object, FieldChangedEventArgs) :void
- Window_Loaded(object, RoutedEventArgs) :void
- Window_SizeChanged(object, SizeChangedEventArgs) :void
- Button_Click(object, RoutedEventArgs) :void
- MenuGameNew_Click(object, RoutedEventArgs) :void
- MenuGameLoad_Click(object, RoutedEventArgs) :void
- MenuGameSave_Click(object, RoutedEventArgs) :void
- MenuGameExit_Click(object, RoutedEventArgs) :void
-_model
Model::TicTacToeModel

WPF alapismeretek

Példa

Megvalósítás (`TicTacToeWindow.xaml`*):*

```
<Window … >
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <Menu Grid.Row="0">…</Menu>
        <Grid x:Name="_gameGrid" Grid.Row="1">
            <Grid.RowDefinitions … />
            <Grid.ColumnDefinitions … />
        </Grid>
    </Grid>
</Window>
```

WPF alapismeretek

Példa

Megvalósítás (`TicTacToeWindow.xaml.cs`*):*

```
private void GenerateTable(){
    _buttonGrid = new Button[3, 3];
    for (Int32 i = 0; i < 3; i++)
       for (Int32 j = 0; j < 3; j++){
          _buttonGrid[i, j] = new Button();
          …
          _buttonGrid[i, j].SetValue(
             Grid.RowProperty, i);
          _buttonGrid[i, j].SetValue(
             Grid.ColumnProperty, j);
          // beállítjuk a függőségi tulajdonságokat
          …
}
```

Példa

Alternatív megvalósítás (`TicTacToeWindow.xaml`*):*

```
<Window … >
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <Menu Grid.Row="0">…</Menu>
        <UniformGrid Name="_gameGrid"
            Grid.Row="1" Rows="3" Columns="3" />
    </Grid>
</Window>
```

WPF alapismeretek

Példa

Alternatív megvalósítás (`TicTacToeWindow.xaml.cs`*):*

```
private void GenerateTable(){
    _buttonGrid = new Button[3, 3];
    for (Int32 i = 0; i < 3; i++)
       for (Int32 j = 0; j < 3; j++){
          _buttonGrid[i, j] = new Button();
          …
          _buttonGrid[i, j].SetValue(
             Grid.RowProperty, i);
          _buttonGrid[i, j].SetValue(
             Grid.ColumnProperty, j);
          // a UniformGrid feltöltése
          // mindig folytonos
          …
}
```