Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése II

3. előadás

Windows Forms alapismeretek, eseményvezérlés

Giachetta Roberto

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

Windows Forms alapismeretek

Grafikus felületek .NET keretrendszerben

- A .NET keretrendszer több grafikus felület megvalósítási technológiát is biztosít:
  - a *Windows Forms* (*WinForms*) az elsőként kifejlesztett felület (.NET 1.0), amely raszteres grafikára (GDI+) épül, és teljes mértékben processzor által vezérelt
  - a *Windows Presentation Foundation* (*WPF*) a később kifejlesztett felület (.NET 3.0), mely vektoros grafikára épül, célja a 3D gyorsítókártyák lehetőségeinek kihasználása
  - a *Modern UI* direkt hordozható eszközökre szánt egyszerűsített, letisztult felület, amely a WPF architektúrára épít

Windows Forms alapismeretek

A Visual Studio eszközei

- A *Microsoft Visual Studio* biztosít egy felülettervező eszközt, amivel grafikusan készítjük el a GUI-t, a hozzá tartozó kód pedig legenerálódik, így nem szükséges a teljes kódot megírnunk
  - egy eszköztárból (*ToolBox*) válogathatunk a vezérlők közül, és ezt grafikusan („drag and drop” módszerrel) helyezhetjük a felületre
  - a vezérlő tulajdonságait (*Properties*) és eseményeit (*Events*) külön menüben állíthatjuk
  - bármikor válthatunk a kód és a tervező nézet között, és felváltva szerkeszthetjük a felületet, a szerkesztett kód nem lesz hatással a generált kódra

Windows Forms alapismeretek

Vezérlők

- A *Windows Forms* grafikus felülete vezérlőkből épül fel
  - az osztályok a **`System.Windows.Forms`** névtérben helyezkednek el
  - minden vezérlő egy ősosztály leszármazottja (**`Control`**)
- A vezérlők csoportosítása:
  - ablak (**`Form`**)
  - elrendező elemek (**`FlowLayoutPanel`**, **`TableLayoutPanel`**, **`SplitConatiner`**, **`GroupBox`**, …)
  - menük és eszköztárak (**`ToolStrip`**, **`StatusStrip`**, …)
  - adatkezelő vezérlők (**`DataGridView`**, **`BindingSource`**, …)

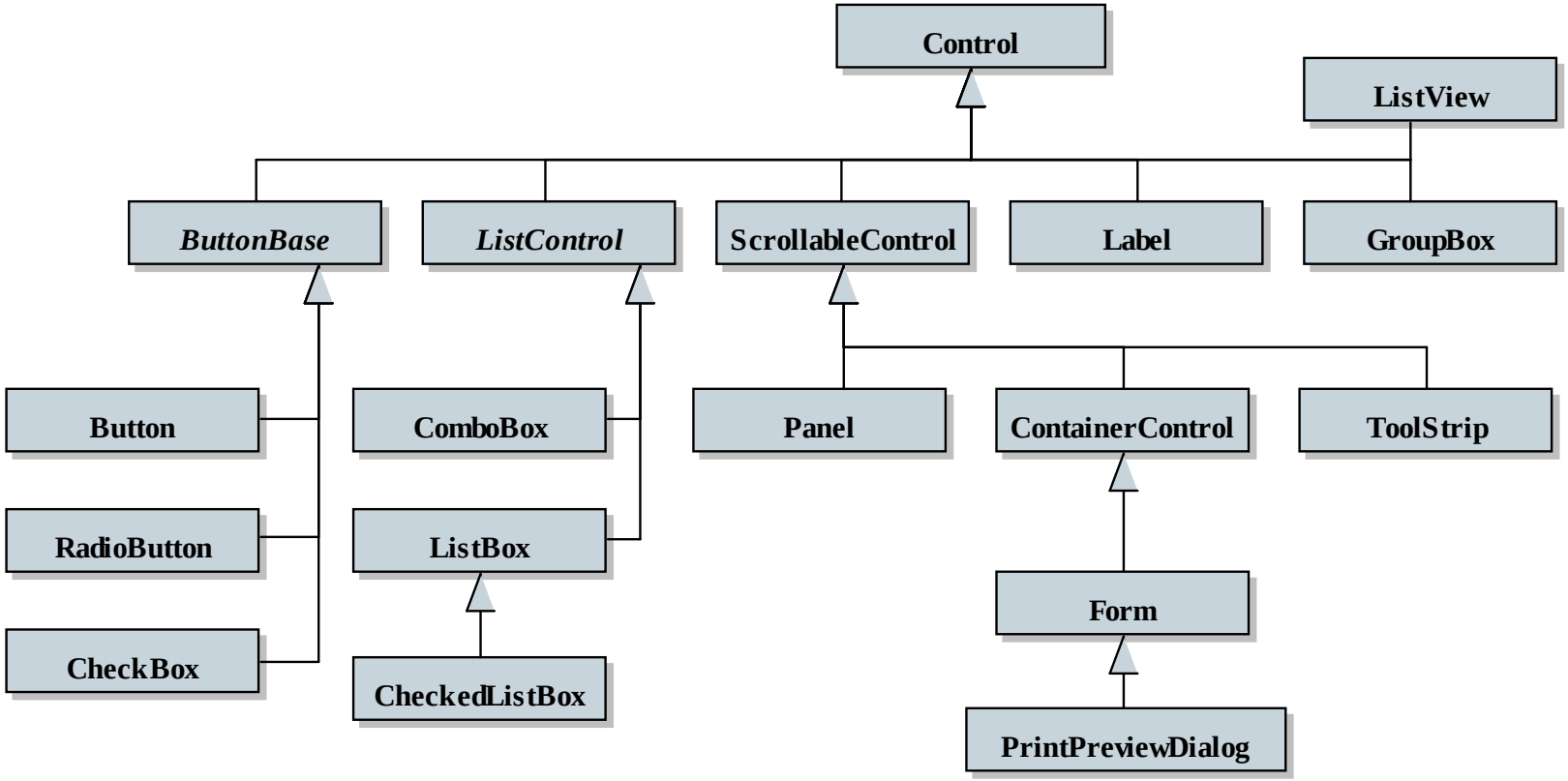
Windows Forms alapismeretek

Vezérlők

- felületi elemek:
  - megjelenítők (**`Label`** , **`ListView`**, **`Panel`**, **`WebBrowser`**, …)
  - gombok (**`Button`**, **`RadioButton`**, **`CheckBox`**, …)
  - beviteli mezők (**`TextBox`**, **`NumericUpDown`**, **`ComboBox`**, **`ListBox`**, …)
  - egyedi vezérlők (**`UserControl`**)
- dialógusablakok (**`MessageBox`**, **`OpenFileDialog`**, **`FolderBrowserDialog`**, **`PrintDialog`**, …)
- segédelemek (**`ToolTip`**, **`ErrorProvider`**, **`Help`**, …)

Windows Forms alapismeretek

Vezérlők



Windows Forms alapismeretek

Vezérlők

- A vezérlők tulajdonságait általában létrehozás után állítjuk be (de konstruktorban is megadhatóak alapvető értékek)
- Fontosabb tulajdonságok:
  - pozícionálás és méretezés (**`Location`**, **`Size`**, **`Anchor`**, **`AutoSize`**, **`Dock`**)
  - engedélyezettség (**`Enabled`**), fókuszáltság (**`Focused`**)
  - láthatóság (**`Visible`**), áttetszőség (**`Opacity`**)
  - tabulátorkezelés (**`TabIndex`**, **`TabStop`**)
  - felirat (**`Text`**), szöveget tartalmazó elemekben
  - név (**`Name`**), a későbbi azonosításra szolgál

- szülő és gyerek vezérlők (**`Parent`**, **`Controls`**)
- színezés (**`ForeColor`**, **`BackColor`**), amelyek a **`Color`** osztály segítségével állíthatunk be tetszőleges RGB kombinációra, vagy fix értékre (pl. **`Color.Red`**)

- Pl. egy címke esetén:

```
Label myLabel = new Label();
myLabel.Location = new Point(6, 18); // pozíció
myLabel.Name = "cimke"; // név
myLabel.Size = new Size(65, 13); // méret
myLabel.TabIndex = 1; // tabulátor index
myText.ForeColor = Color.Blue; // kék feliratszín
myLabel.Text = "valami felirat"; // felirat
```

Windows Forms alapismeretek

Események és eseménykezelés

- A C# nyelvi szinten valósítja meg az eseménykezelést, amelyhez eseményeket (**`event`**) és delegáltakat (**`delegate`**) használ
  - az eseménykezelő egy szabványos metódus, de mindig két paramétere van, a küldő objektum (**`object sender`**), és az eseménytulajdonságok (**`EventArgs e`**), amelyek leszármazottai hordozhatnak speciális értéket
  - a delegált szabja meg az eseménytulajdonságok (**`EventArgs`**) típusát
    - az alapértelmezett delegált az **`EventHandler`**
    - lehet delegáltakat létrehozni, vagy sablont használni más tulajdonságokhoz

- Az eseménykezelő hozzárendelésekor az eseménykezelő nevét kell megadnunk:

```
<objektumnév>.<eseménynév>
    += new EventHandler(<metódusnév>);
```

  - a **+=** operátor lehetővé teszi, hogy egy eseményhez több eseménykezelőt is hozzárendeljünk
  - a társításban bármely objektum eseményét rendelhetjük bármely, azonos szintaktikájú eseménykelőhöz
  - a **-=** operátor segítségével tudjuk bontatni a kapcsolatot
- Pl.:

```
class EventClass {
   public event EventHandler MyEvent; // esemény
}
```

- Pl.:

```
class HandlerClass {
   private EventClass ec;

   public HandlerClass(){
      ec = new EventClass();
      ec.MyEvent +=
         new EventHandler(MyEventHandler);
      // eseménykezelő társítás
   }
   private void MyEventHandler(object sender,
                               EventArgs e){ … }
      // eseménykezelő metódus
}
```

- Események kiváltása az esemény meghívásával történik, ahol átadjuk a megfelelő paramétereket
  - esemény csak akkor váltható ki, ha van hozzárendelve eseménykezelő, különben az esemény **`null`** értéknek felel meg (és így kivételt kapunk)
  - általában a kiváltást külön metódusban végezzük
- Pl.:

```
if (ec.MyEvent != null)
      // ha van hozzárendelve eseménykezelő
   ec.MyEvent(this, null); // kiváltjuk
      // a küldő az aktuális objektum, az
      // eseményargumentumok üresek
```

Windows Forms alapismeretek

Vezérlők eseményei

- A vezérlők számos eseménnyel rendelkeznek, több csoportban:
  - egér és billentyűzet tevékenységek (**`Click`**, **`MouseClick`**, **`MouseHover`**, **`KeyDown`**, **`KeyUp`**, …)
  - vezérlőállapot megváltozása (**`Validating`**, **`Validated`**, **`Resize`**, **`Paint`**, **`GotFocus`**, …)
  - tulajdonságok megváltozása (**`BackColorChanged`**, **`TabIndexChanged`** , **`TextChanged`**, **`SizeChanged`**, …)
- Bizonyos események csak akkor váltódnak ki, ha a vezérlő fókuszban van (**`Focus()`**), pl. billentyűzetesemények
  - ugyanakkor a billentyűzet lekezelhető az ablak szintjén is

Windows Forms alapismeretek

Vezérlők eseményei

- Pl.:

```
Button b = new Button();
b.Click += new EventHandler(B_Click); // társítás
b.MouseDoubleClick +=
   new MouseEventHandler(B_DClick); // társítás
…
void B_Click(object sender, EventArgs e) { … }
   // eseménykezelő
…
void B_DClick(object sender, MouseEventArgs e) {
   // speciális eseményargumentum, amelytől
   // lekérdezhető az egérgomb (Button) és a
   // pozíció (Location)
}
```

Windows Forms alapismeretek

Ablakok

- Az ablak ősosztálya a **`Form`**, amely példányosítással, vagy specializációval használható, fontosabb tulajdonságai:
  - vezérlő eszköztár (**`ControlBox`**, **`MinimizeBox`**, **`MaximizeBox`**)
  - menü (**`Menu`**)
  - kezdőpozíció (**`StartPosition`**)
  - ablakállapot (**`WindowState`**)
  - vezérlőgombok (**`AcceptButton`**, **`CancelButton`**)
  - dialóguseredmény (**`DialogResult`**)
  - billentyű-esemény elfogás (**`KeyPreview`**)

Windows Forms alapismeretek

Ablakok

- Pl. (`MyForm.cs`):

```
class MyForm : Form // saját ablak osztály
{
   private Button okButton; // mezők
   …
   public MyForm() { // konstruktor
     Text = "Az ablak"; // ablakcím

     okButton = new Button();
        // vezérlők inicializálása
     okButton.Text = "OK";
     okButton.Location = new Point(5, 5);
     …
  }
  …
```

Windows Forms alapismeretek

Ablakok

- Felülettervező használata esetén az ablakaink több fájlban helyezkednek el
  - a felülettervező által generált kód egy másik (**`Designer`**) fájlba kerül, így két fájl alkotja az ablak osztályát

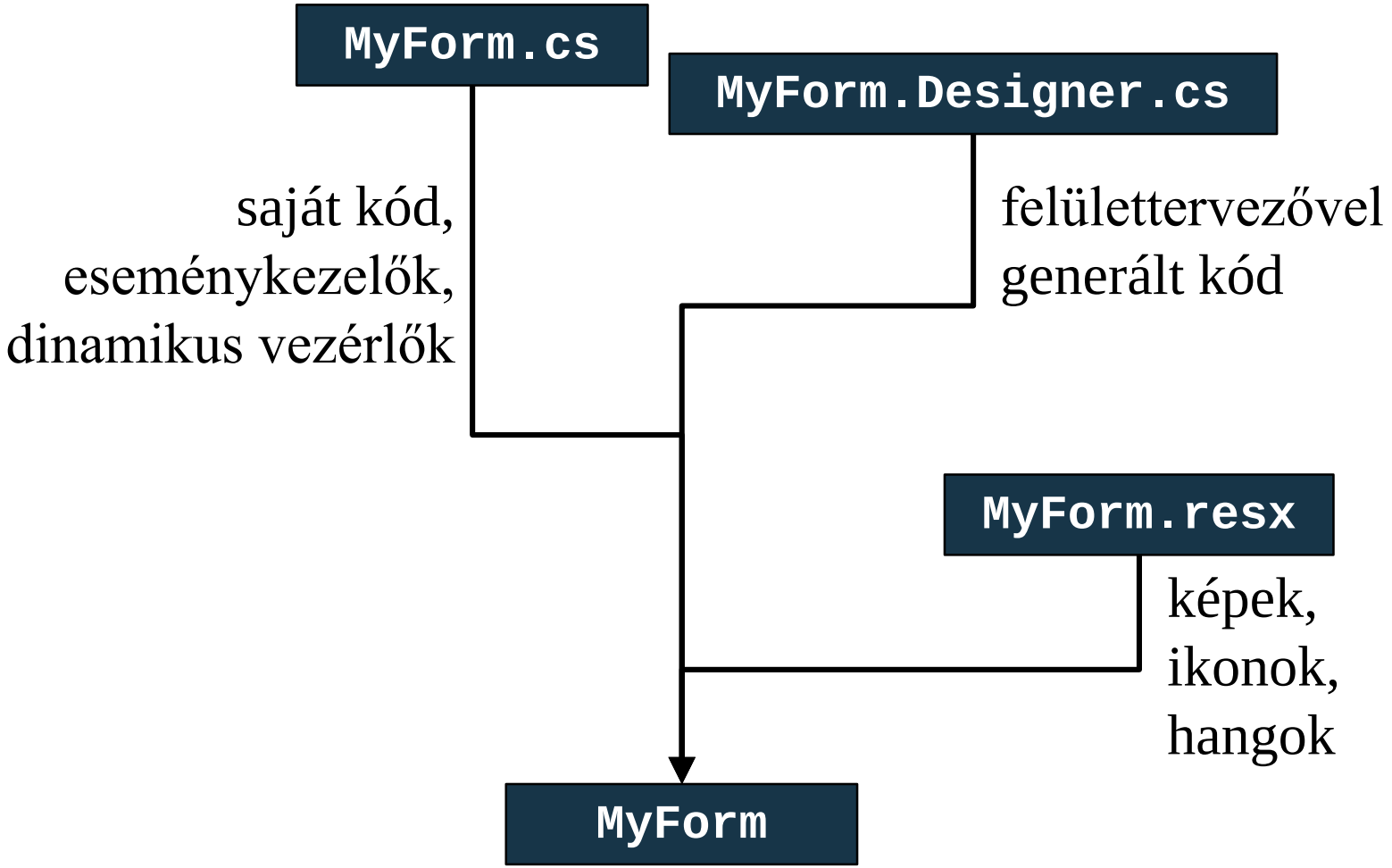
    `<`*`osztálynév`*`>.cs`

    `<`*`osztálynév`*`>.Designer.cs`
  - ezen felül az ablakhoz használt erőforrások (képek, ikonok, ...) erőforrásfájlban helyezhetőek el:

    `<`*`osztálynév`*`>.resx`
  - az ablakok úgynevezett parciális (**`partial`**) osztályokká válnak, amelyek több fájlban helyezkedhetnek el

Windows Forms alapismeretek

Ablakok



Windows Forms alapismeretek

Ablakok

- A felülettervező által generált kód is olvasható, átírható, azonban a módosítása hatással lehet a felülettervező ablakra, ezért a saját kódot (eseménykezelők, dinamikus vezérlők, …) ne keverjük a generált kóddal
- A tervezőben létrehozott felület (vezérlők létrehozása, tulajdonságok beállítása) az **`InitializeComponent()`** metódus segítségével jön létre
  - az osztály konstruktorának első utasítása, ez után tetszőlegesen bővíthetjük a kódot
- Emellett a tervező felülírja automatikusan a **`Dispose()`** metódust, ami a vezérlők törlését végzi

Windows Forms alapismeretek

Ablakok

- Pl. (`MyForm.cs`):

```
namespace MyFormsApplication
{
   partial class MyForm : Form {
      // parciális ablak osztály
      public MyForm() { // konstruktor
         InitializeComponent();
            // felülettervező által létrehozott
            // vezérlők inicializálása

         //… további inicializáció ezt követően
      }
   }
}
```

Windows Forms alapismeretek

Ablakok

- Pl. (`MyForm.Designer.cs`):

```
namespace MyFormsApplication
{
    partial class MyForm {
        // parciális ablak osztály másik része

        public void Dispose() { … }
            // ablak megsemmisítése
        public void InitializeComponent(){ … }
            // vezérlők inicializálása

        // … vezérlők
    }
}
```

Windows Forms alapismeretek

Dialógusablakok

- Amellett, hogy bármely ablakot kezelhetünk dialógusablakként, vannak előre legyártott dialógusablakok is, a legegyszerűbb az előugró üzenet (**`MessageBox`**)
  - a statikus **`Show(…)`** művelettel használható, amely paraméterezhető (pl. üzenet, gombok, ikon, …)
  - a művelet visszatérési értéke **`DialogResult`**, így lekérdezhető, milyen gombot használt a felhasználó
  - pl.:

```
MessageBox.Show("Really quit?",
    "My Application", // cím
    MessageBoxButtons.YesNo, // gombok
    MessageBoxIcon.Question); // ikon
```

Dialógusablakok

- A további dialógusablakok megegyeznek az operációs rendszerben fellelhető ablakokkal, pl.:
  - fájl megnyitó (**`OpenFileDialog`**), fájl mentő (**`SaveFileDialog`**), könyvtárböngésző (**`FolderBrowserDialog`**)
  - betűtípus-választó (**`FontDialog`**), színválasztó (**`ColorDialog`**)
  - nyomtatási beállítások (**`PrintDialog`**), előnézet (**`PrintPreviewDialog`**), oldalbeállítás (**`PageSetupDialog`**)
- További dialógusablakok (pl. szövegbeviteli mező) egyedileg készíthetőek

Windows Forms alapismeretek

Dialógusablakok

- Pl. :

```
SaveFileDialog dialog = new SaveFileDialog();
   // fájl mentő dialógus
dialog.Title = "Save file"; // cím
dialog.Filter =
   "txt files (*.txt)|*.txt|All files (*.*)|*.*";
   // szűrés a megjelenített tartalomra
if (dialog.ShowDialog() == DialogResult.OK) {
   // ha OK-val zárták le az ablakot
   StreamWriter writer =
      new StreamWriter(dialog.FileName);
   // a megadott fájlnévre mentünk
   …
}
```

Windows Forms alapismeretek

Alkalmazás osztályok

- A grafikus felületű alkalmazásokat egy *alkalmazásnak* (**`Application`**) kell vezérelnie
  - statikus osztály, a főprogramban használjuk
  - legfőbb művelete a futtatás (**`Run`**), amely paraméterben megkapja az első indítandó képernyő objektumát, illetve lehetőséget ad a kilépésre is (**`Exit`**)
  - ezen felül alkalmas a környezet beállítására (**`EnableVisualStyles`**, **`UseWaitCursor`**, …), valamint információgyűjtésre (**`StartupPath`**, **`OpenForms`**, **`ProductName`**, …)
  - eseményeivel követhetjük a programfutást (**`ApplicationExit`**, **`Idle`**)

Windows Forms alapismeretek

Alkalmazás osztályok

- Pl. (**Program.cs**):

```
namespace MyFormsApplication
{
    class Program
    {
        static void Main() // főprogram
        {
            Application.EnableVisualStyles();
            Application.Run(new MyForm());
                // alkalmazás indítása a megadott
                // ablakkal
        }
    }
}
```

Windows Forms alapismeretek

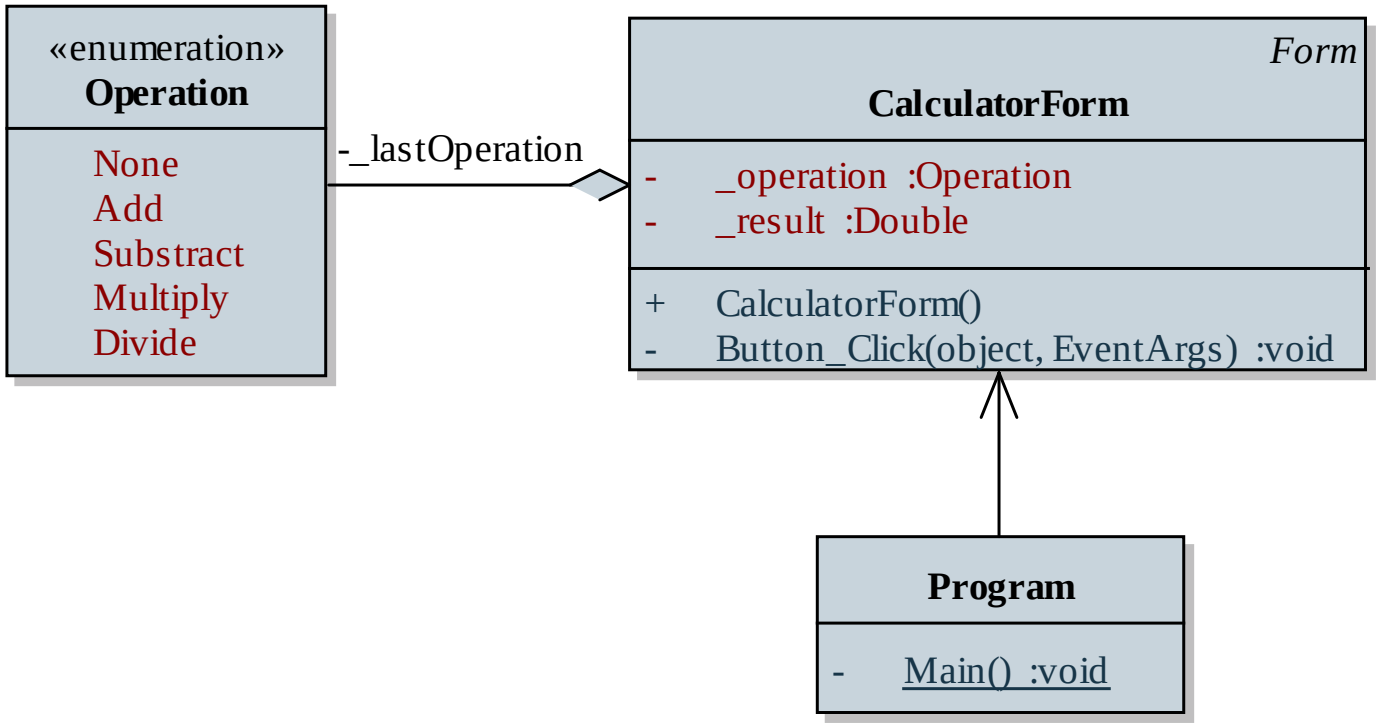
Példa

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

- az alkalmazás felületét a felülettervezővel készítjük el, elhelyezünk 5 gombot (**`Button`**), egy szövegbeviteli mezőt (**`TextBox`**), valamint egy listát (**`ListBox`**)
- az ablak osztályban (**`CalculatorForm`**) létrehozunk egy eseménykezelőt (**`Button_Click`**) a gombokra, amely a megfelelő műveleteket végzi el
- egy felsorolási típussal (**`Operation`**) tároljuk el a műveletet
- ellenőrizzük kivételkezeléssel, hogy a bevitt érték megfelelő-e

Példa

Tervezés:



Windows Forms alapismeretek

Példa

Megvalósítás (`CalculatorForm.cs`*):*

```
public partial class CalculatorForm : Form {
    …
    // egy közös eseménykezelő az összes gombnak
    private void Button_Click(object sender,
                              EventArgs e){
       try {
          …
          // minden esetben:
          _fistNumber =
             Double.Parse(_textNumber.Text);
             // eltároljuk az első operandust
```

Windows Forms alapismeretek

Példa

Megvalósítás (`CalculatorForm.cs`*):*

```
            switch (((sender as Button).Text) {
                // megvizsgáljuk, milyen az eseményt
                // kiváltó gomb felirata, így
                // eldönthetjük, melyik gombot
                // nyomták le
                …
        }
        catch (OverflowException) {
            MessageBox.Show("Your input has to many
                digits!", "Calculation Error",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        …
```

Windows Forms alapismeretek

Ablakok használata

- Ablakok megnyitására két lehetőségünk van:
  - a **`Show()`** művelet megnyitja az ablakot, de utána tovább fut a megnyitó ablak kódja
  - a **`ShowDialog()`** művelet dialógusablakként nyitja meg, ekkor a megnyitó ablak blokkolódik, és csak az új ablak bezárása után lehet bármely más tevékenységet végezni
  - utóbbi esetben kaphatunk eredményt (**`DialogResult`**) az ablaktól a lezárást illetően (pl. **`None`**, **`OK`**, **`Cancel`**, **`Yes`**, …), amelyet lekérdezhetünk, pl.:
    ```
    if (myForm.ShowDialog() == DialogResult.Yes) …
    ```
- Ablak bezárása a **`Close()`** művelettel történik

Windows Forms alapismeretek

Időzítő

- Az időzítő kezelést egyfelől szálak segítségével, másfelől a **`Timer`** osztályon keresztül vehetjük igénybe
  - lehetőségünk van indításra (**`Start`**), leállításra (**`Stop`**), állapotlekérdezésre (**`Enabled`**), valamint az intervallum (**`Interval`**) beállítására, az idő eltelésekor a **`Tick`** esemény váltódik ki
  - pl.:

```
Timer myTimer = new Timer(); // időzítő
myTimer.Interval = 1000;
   // 1 másodpercenként váltódik ki az esemény
myTimer.Tick += new EventHandler(Timer_Tick);
   // eseménykezelő társítás
myTimer.Start(); // indítás
```

Windows Forms alapismeretek

Példa

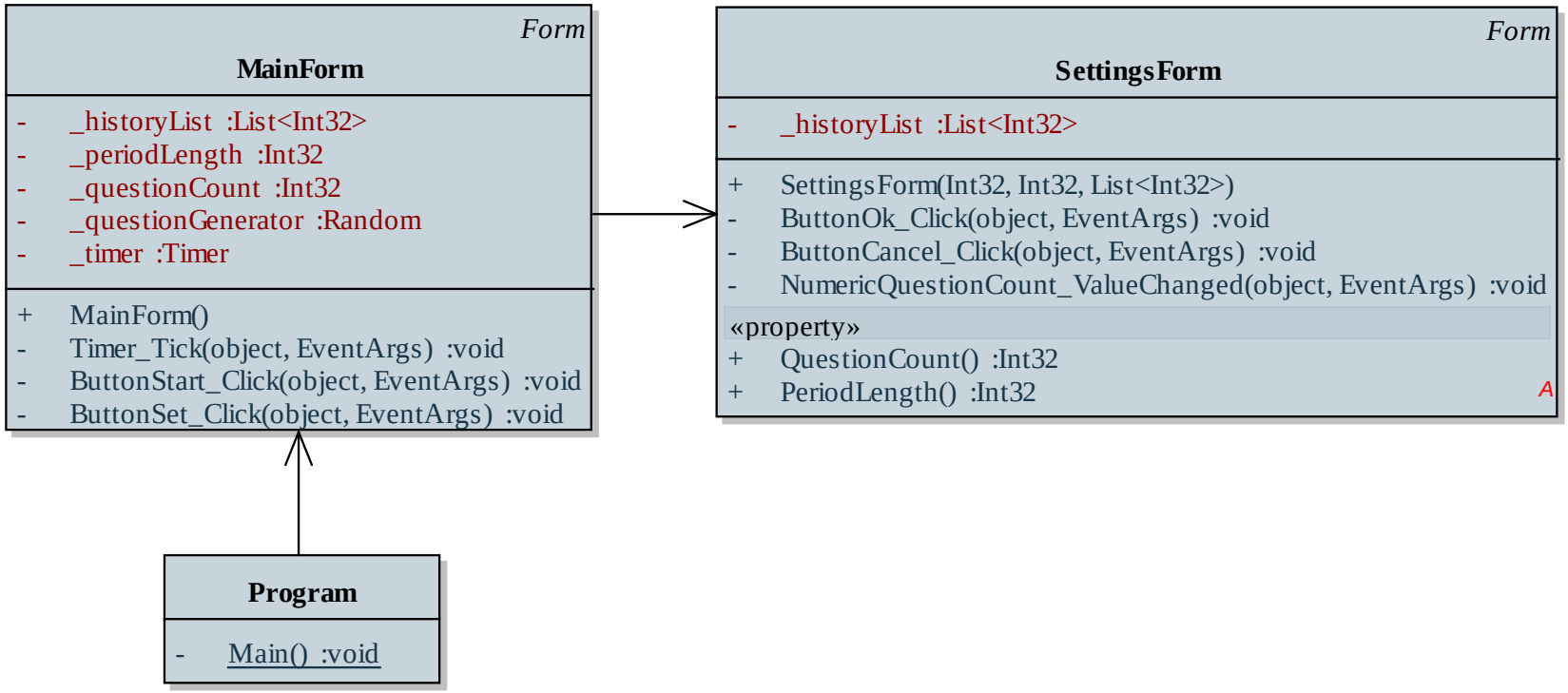
Feladat: Készítsünk egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- a főablakban két gombot (*Start/Stop*, *Beállít*), valamint egy szövegmezőt helyezünk el, a generálást időzítővel (**`Timer`**) valósítjuk meg, a generált számokat elmentjük egy listába az ellenőrzéshez
- egy segédablakban két számbeállító (**`NumericUpDown`**) segítségével állítjuk be a tételek számát és a bent lévő hallgatók számát
- egy kijelölhető lista (**`CheckedListBox`**) segítségével ellenőrizhetjük és korrigálhatjuk a kiadott tételszámokat

Windows Forms alapismeretek

Példa

Tervezés:



Windows Forms alapismeretek

Példa

Megvalósítás (`MainForm.cs`*):*

```
void Timer_Tick(object sender, EventArgs e) {
   Int32 number = _questionGenerator.Next(1,
      _questionCount + 1);
      // új szám generálása 1 és a tételszám
      // között
   while (_historyList.Contains(number))
      // ha a szám szerepel a korábbiak között
      number = _questionGenerator.Next(1,
         _questionCount + 1);
         // akkor új generálása

   _textNumber.Text = number.ToString();
}
```

Windows Forms alapismeretek

Példa

Megvalósítás (`MainForm.cs`*):*

```
void ButtonSet_Click(object sender, EventArgs e) {
   SettingsForm f = new SettingsForm(
      _questionCount, _periodLength,
      _historyList);
      // dialógusablak létrehozása paraméterekkel

   if (f.ShowDialog() == DialogResult.OK) {
      // dialógusablak megjelenítése
      _questionCount = f.QuestionCount;
         // elmentjük az új értékeket
      _periodLength = f.PeriodLength;
}
```

Windows Forms alapismeretek

Billentyűzet és egérkezelés

- A billentyűzet kezelésére lehetőség van a fókuszált vezérlőn, de az ablak is le tudja kezelni a billentyű eseményeket (**`PreviewKeyDown`**, **`KeyDown`**, **`KeyUp`**, **`KeyPress`**)
  - az ablaknál engedélyeznünk kell a kezelést (**`KeyPreview`**), különben nem fogja el az eseményt
  - eseményargumentumban (**`KeyEventArgs`**) megkapjuk a billentyűzet adatait (**`KeyCode`**, **`KeyData`**, **`Modifiers`**, ...)
  - az ablak mellett a vezérlő is megkapja az eseményt, amennyiben ezt nem szeretnénk, lehetőség van beavatkozni (**`SuppressKeyPress`**)

Windows Forms alapismeretek

Billentyűzetkezelés

- Pl.:

```
KeyPreview = true;
   // az ablak lekezeli a billentyűzetet
KeyDown += new KeyEventHandler(Form_KeyDown);
   // billentyű lenyomásának eseménye
…
void Form_KeyDown(object sender, KeyEventArgs e) {
   if (e.KeyCode == Keys.Enter) // Enter hatására
   {
      … // tevékenység elvégzése
      e.SuppressKeyPress = true;
         // a vezérlő nem kapja meg az eseményt
   }
}
```

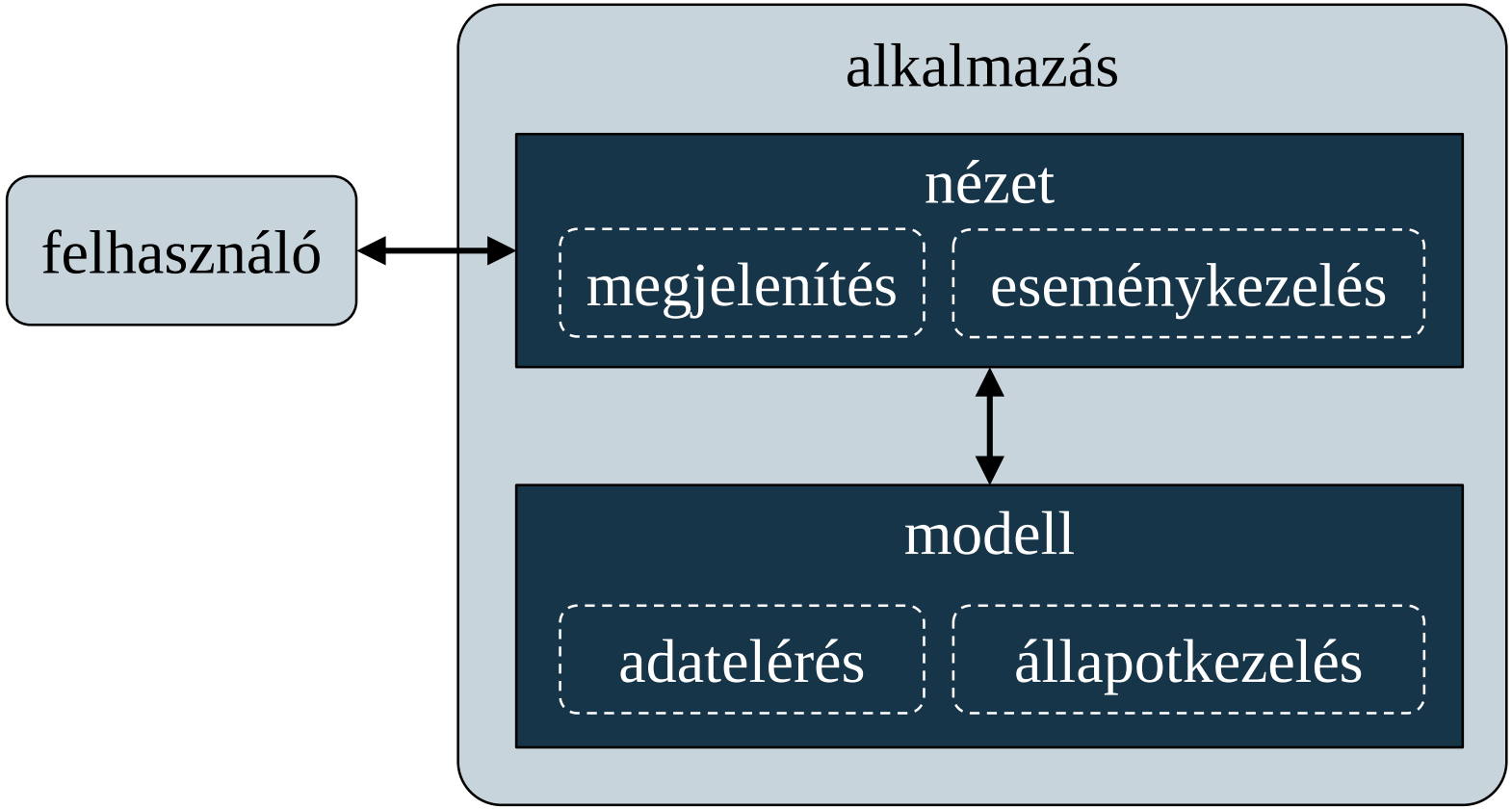
A modell/nézet architektúra

- Összetettebb alkalmazásoknál az egyrétegű felépítés korlátozza a program
  - áttekinthetőségét, tesztelését (pl. nehezen látható át, hol tároljuk a számításokhoz szükséges adatokat)
  - módosíthatóságát, bővíthetőségét (pl. nehezen lehet a felület kinézetét módosítani)
  - újrafelhasználhatóságát (pl. komponens kiemelése és áthelyezése másik alkalmazásba)
- A legegyszerűbb felbontás a felhasználói felület leválasztása a háttérbeli tevékenységekről, ezt nevezzük , *modell/nézet* (*MV*, *model-view*) architektúrának

A modell/nézet architektúra

- A modell/nézet architektúrában
  - a *modell* tartalmazza a háttérben futó logikát, azaz a tevékenységek végrehajtását, az állapotkezelést, valamint az adatkezelést, ezt nevezzük *alkalmazáslogiká*nak, vagy *üzleti logiká*nak
  - a *nézet* tartalmazza a grafikus felhasználói felület megvalósítását, beleértve a vezérlőket és eseménykezelőket
  - a felhasználó a nézettel kommunikál, a modell és a nézet egymással
  - a modell nem függ a nézettől, függetlenül, önmagában is felhasználható, ezért könnyen átvihető másik alkalmazásba, és más felülettel is üzemképes

A modell/nézet architektúra



Példa

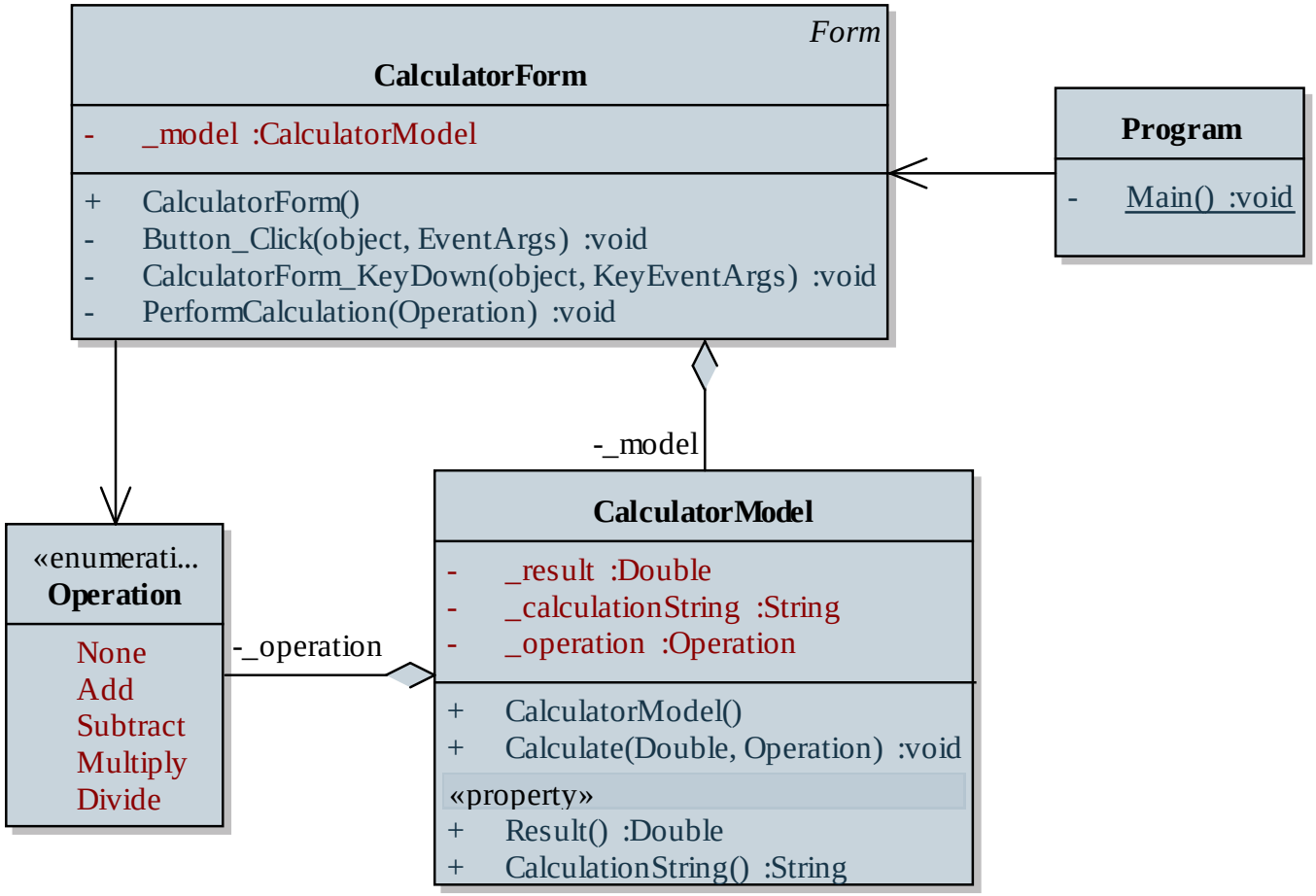
Feladat: Készítsük a számológép alkalmazást modell/nézet architektúrában.

- leválasztjuk a modellt a felületről, így létrejön a számológép (**`CalculatorModel`**), amely végrehajtja a műveletet (**`Calculate`**), tárolja az eredményt (**`Result`**), valamint a művelet szöveges leírását (**`CalculationString`**)
- a nézet (**`CalculatorForm`**) feladata a modell példányosítása és használata
- a gombok eseménykezelése mellett célszerű a billentyűzetet is kezelni az ablakon keresztül, a tevékenység végrehajtását pedig külön alprogramba helyezzük (**`PerformCalculation`**)

Windows Forms alapismeretek

Példa

Tervezés:



Windows Forms alapismeretek

Példa

Megvalósítás (`CalculatorForm.cs`*):*

```
private void CalculatorForm_KeyDown(object sender,
                    KeyEventArgs e)
{
   switch (e.KeyCode) { // megkapjuk a billentyűt
      case Keys.Add:
         PerformCalculation(Operation.Add);
         e.SuppressKeyPress = true;
            // az eseményt nem adjuk tovább a
            // vezérlőnek
         break;
   …
}
```

Windows Forms alapismeretek

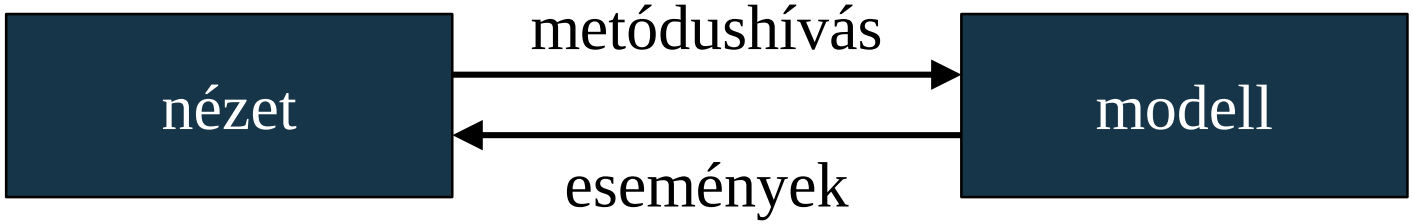
Példa

Megvalósítás (`CalculatorForm.cs`*):*

```
private void PerformCalculation(Operation
                                operation){
   try {
      _model.Calculate(
         Double.Parse(_textNumber.Text),
         operation); // művelet végrehajtása
      _textNumber.Text = _model.Result.ToString();
         // eredmény kiírása
      if (operation != Operation.None)
         _listHistory.Items.Add(
            _model.CalculationString);
            // művelet kiírása a listába
      …
```

A modell/nézet architektúra

- A modell és a nézet kapcsolatát úgy kell megvalósítani, hogy
  - *a nézet ismerheti a modell felületét* (interfészét), és hívhatja annak (publikus) műveleteit
  - *a modellnek semmilyen tudomása sem lehet a nézetről*, ezért nem hívhatja annak műveleteit, de eseményeken keresztül kommunikálhat vele



- A megvalósításban a nézet hivatkozhat a modellre, tartalmazhatja annak példányát

Windows Forms alapismeretek

Példa

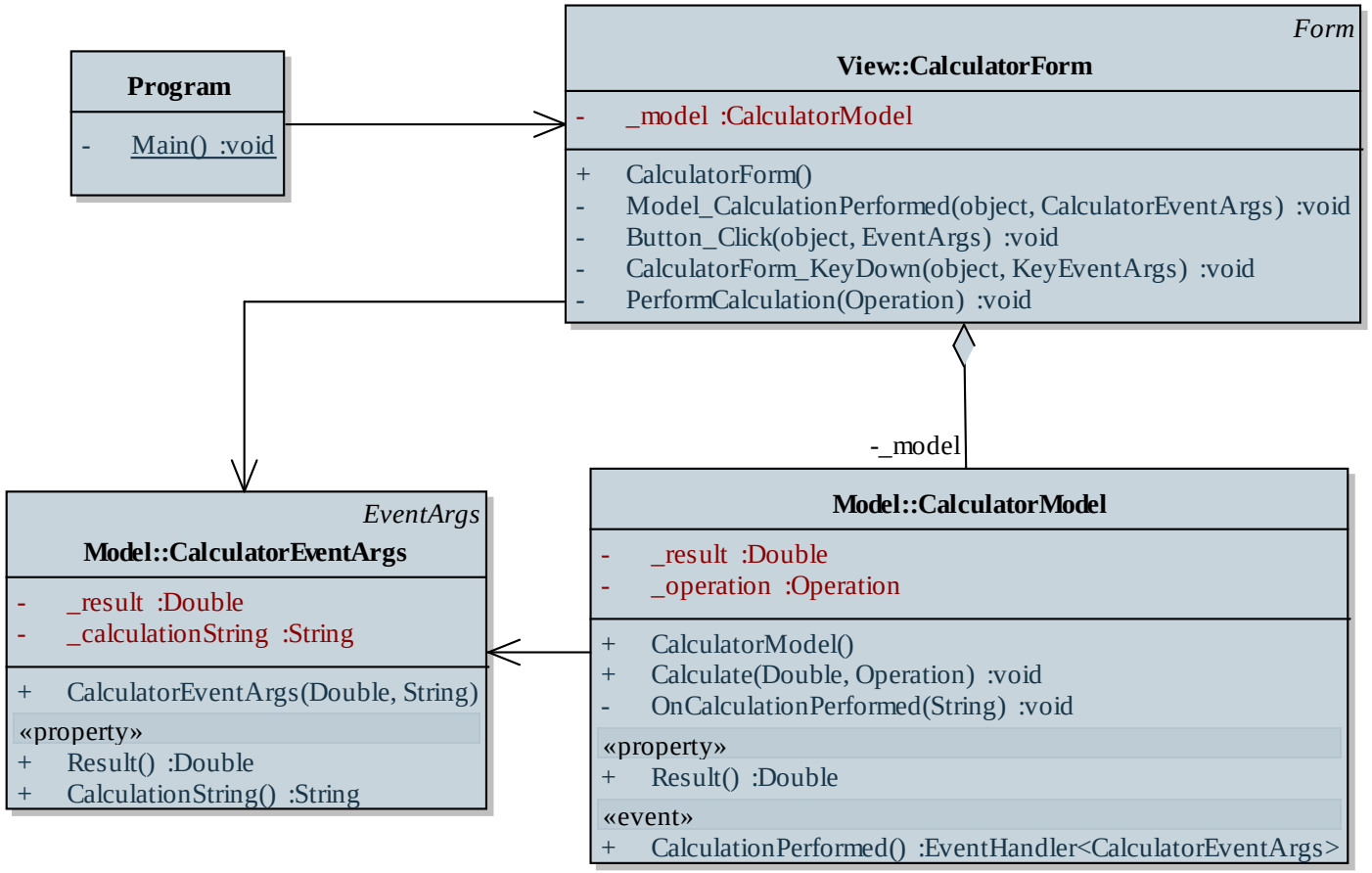
Feladat: Módosítsuk a számológép alkalmazást úgy, hogy a modell jelezze a számítás befejeződését, és annak eredményét.

- a modellhez felveszünk egy új eseményt (**`CalculationPerformed`**), amelyet a nézet feldolgoz
- az esemény mellé adatokat is szolgáltatunk, ezért szükség van egy speciális eseményargumentumra (**`CalculatorEventArgs`**), amely tartalmazza az eredményt, és a szöveges kiírást
- a nézetnek így már nem kell lekérdeznie a számítás eredményét, mert automatikusan megkapja
- az osztályokat helyezzük külön névterekbe

Windows Forms alapismeretek

Példa

Tervezés:



Példa

Megvalósítás (`CalculatorModel.cs`*):*

```
…
public event EventHandler<CalculatorEventArgs>
   CalculationPerformed;
   // számítás végrehajtásának eseménye
…
private void OnCalculationPerformed(String
                    calculationString){
   if (CalculationPerformed != null)
      CalculationPerformed(this,
         new CalculatorEventArgs(_result,
            calculationString));
      // feltöltjük az eseményargumentumot
}
```

Windows Forms alapismeretek

Időzítés modell/nézet architektúrában

- Amennyiben a modell szintjén akarunk időzítést végezni, nem használhatunk felületi időzítőt, de használhatjuk a **`System.Timers.Timer`** időzítőt, amely független a felülettől
  - hasonlóan kezelhető az intervallum (**`Interval`**), indítás és leállítás (**`Start`**, **`Stop`**), valamint az időzített esemény kiváltása (**`Elapsed`**)
  - hátránya, hogy nem szinkronizál a grafikus felülettel, és emiatt a közvetlen manipulálása a felületnek hibát vált ki
  - ez feloldható a vezérlő **`Invoke`** műveletével, amely egy lambda-kifejezéssel megadott akciót (**`Action`**) tud futtatni a felület szálán

Windows Forms alapismeretek

Időzítés modell/nézet architektúrában

- Pl.:

```
Timers.Timer myTimer = new Timer(); // időzítő
myTimer.Elapsed +=
    new ElapsedEventHandler(Timer_Elapsed);
    // időzített esemény
…
void Timer_Elapsed(…){
    // itt nem használhatjuk a felületet
    Invoke(new Action(() => {
        // itt már igen
        myLabel.Text = e.SignalTime.ToString();
            // kiírjuk az eltelt időt a felületre
    }));
}
```

Windows Forms alapismeretek

Példa

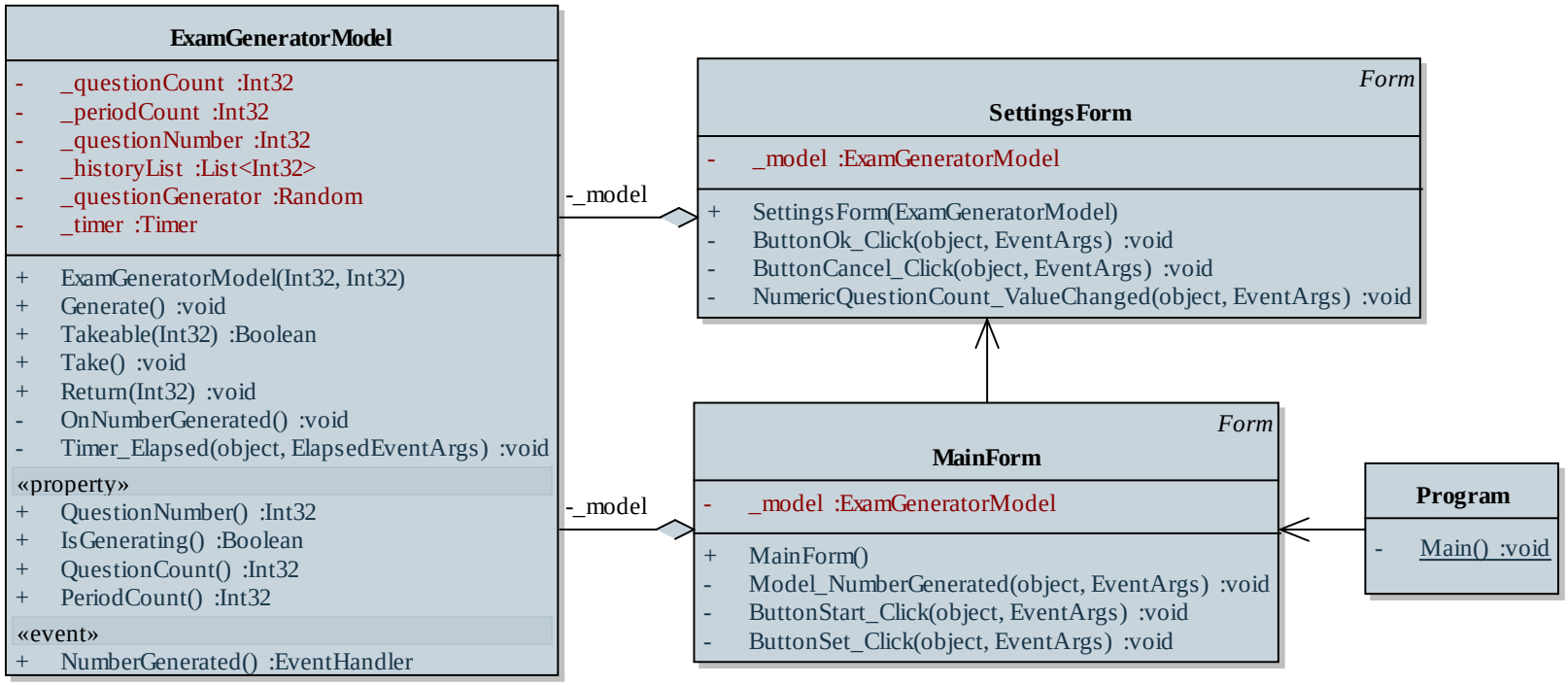
Feladat: Készítsünk egy vizsgatétel generáló alkalmazást modell/nézet architektúrában.

- a modell (**`ExamGeneratorModel`**) végzi a tételek generálását (**`Generate`**), amihez időzítőt használ, továbbá eseménnyel (**`NumberGenerated`**) jelzi, ha generált egy új számot
- emellett lehetőség van a tétel elfogadására (**`Take`**), illetve a korábban húzott tételek visszahelyezésére (**`Return`**)
- mindkét nézet kapcsolatban áll a modellel, a főablak az esemény hatására frissíti a megjelenítést (ügyelve a szinkronizációra)

Windows Forms alapismeretek

Példa

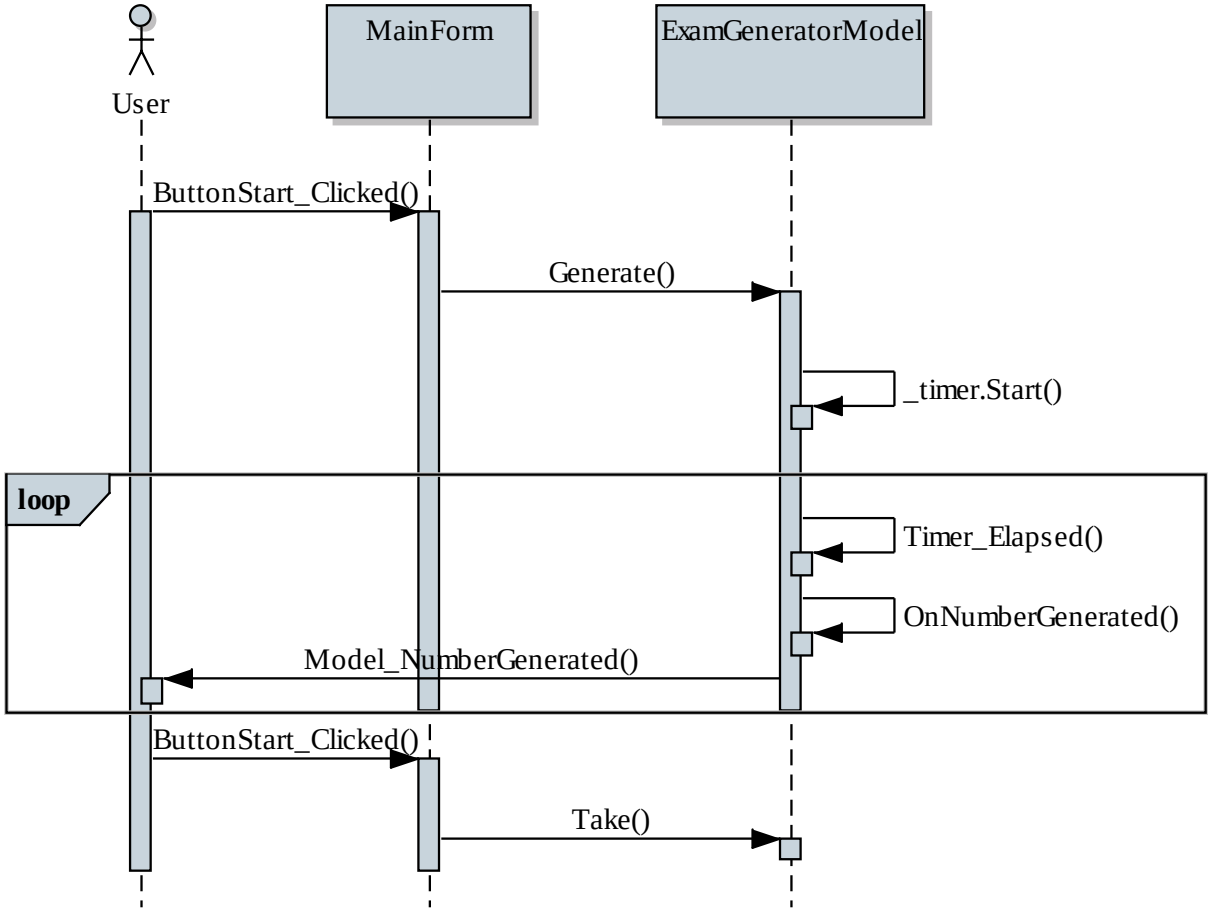
Tervezés:



Windows Forms alapismeretek

Példa

Tervezés:



Windows Forms alapismeretek

Példa

Megvalósítás (`MainForm.cs`*):*

```
public MainForm(){
   _model = new ExamGeneratorModel(10, 0);
   _model.NumberGenerated +=
      new EventHandler(Model_NumberGenerated);
         // modell eseménye
}
private void Model_NumberGenerated(object sender,
                  EventArgs e){
  Invoke(new Action(() => {
     _textNumber.Text =
         _model.QuestionNumber.ToString();
  })); // szinkronizált végrehajtás
}
```