Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások

---

2. előadás

Windows Forms alkalmazások architektúrája

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

Windows Forms alkalmazások architektúrája

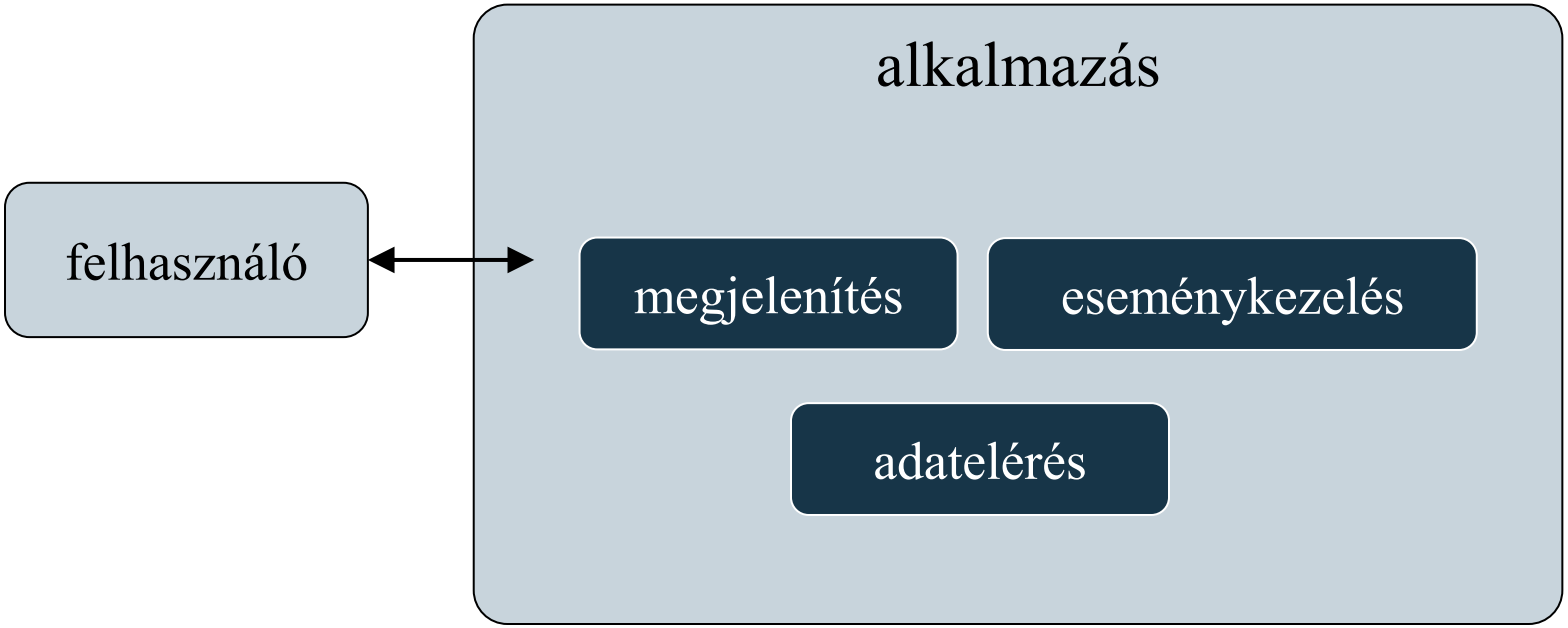
Szoftver architektúra

- A *szoftver architektúra* elsődleges feladata *a rendszer magas szintű felépítésének és működésének meghatározása*:
  - megnevezi a szoftver fő komponenseit,
  - megmutatja azok kapcsolatait a szolgáltatott és elvárt interfészek, a kommunikációs csatornák és csatlakozási pontok jellemzésével.
- A szoftver architektúra megválasztása a szoftver fejlesztése során meghozott *elsődleges tervezési döntések* eredménye, amely
  - kihat a rendszer felépítésére, viselkedésére, kommunikációjára, nem funkcionális jellemzőire és megvalósítására,
  - amely későbbi megváltoztatása a szoftver jelentős újratervezését vonná maga után.

Windows Forms alkalmazások architektúrája

Monolitikus architektúra

- A legegyszerűbb felépítéssel a monolitikus architektúra (*monolithic architecture*) rendelkezik, amely nem különíti el egymástól az egyes feladatköröket (pl. megjelenítés, adatkezelés).



Windows Forms alkalmazások architektúrája

Monolitikus architektúra

Windows Forms alkalmazások architektúrája

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

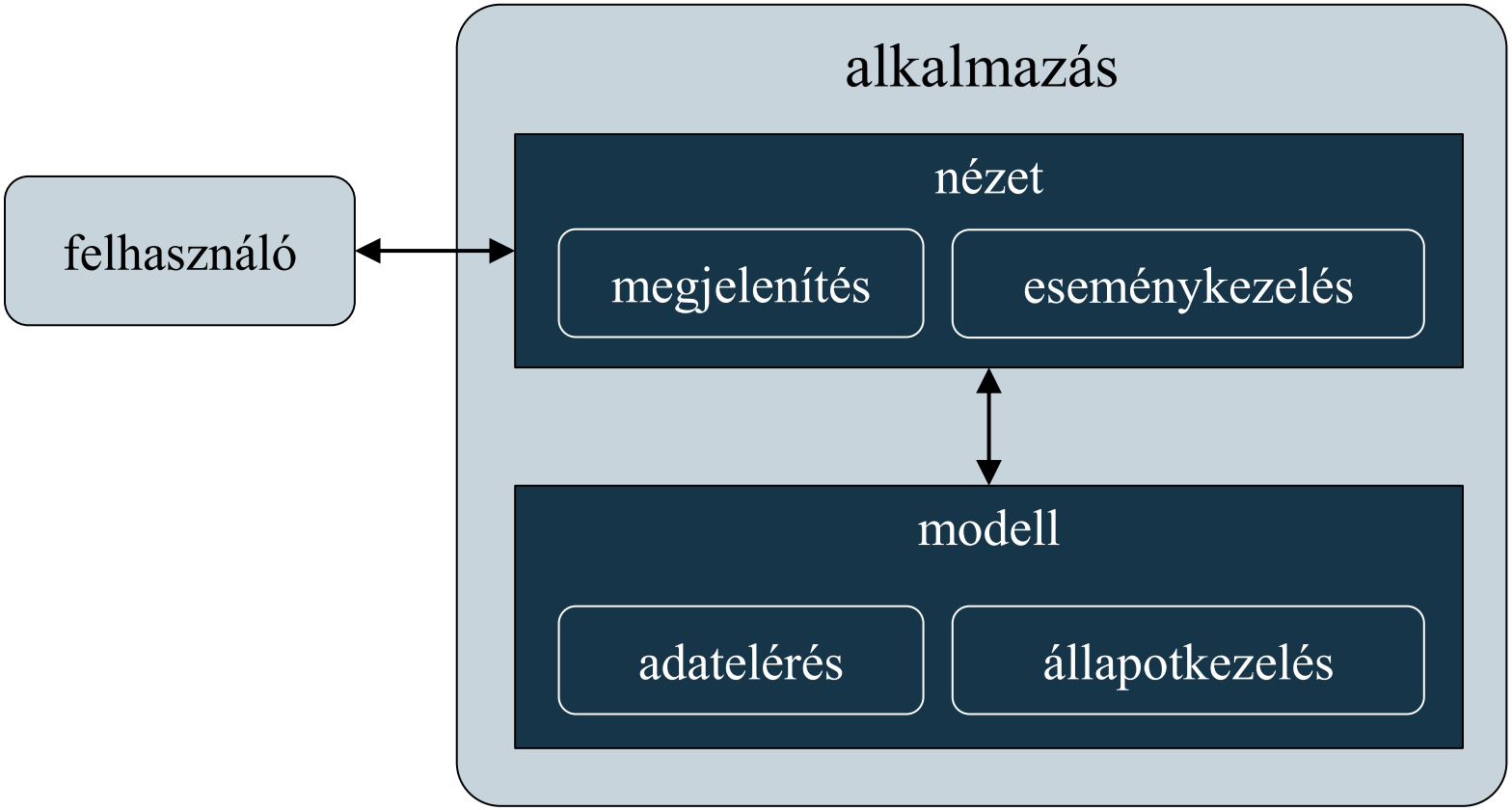
Windows Forms alkalmazások architektúrája

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 *alkalmazáslogikának*, vagy *üzleti logikának*
  - 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

Windows Forms alkalmazások architektúrája

A modell/nézet architektúra



Windows Forms alkalmazások architektúrája

Billentyűzetkezelés

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

Windows Forms alkalmazások architektúrája

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
   }
}
```

Windows Forms alkalmazások architektúrája

Példa

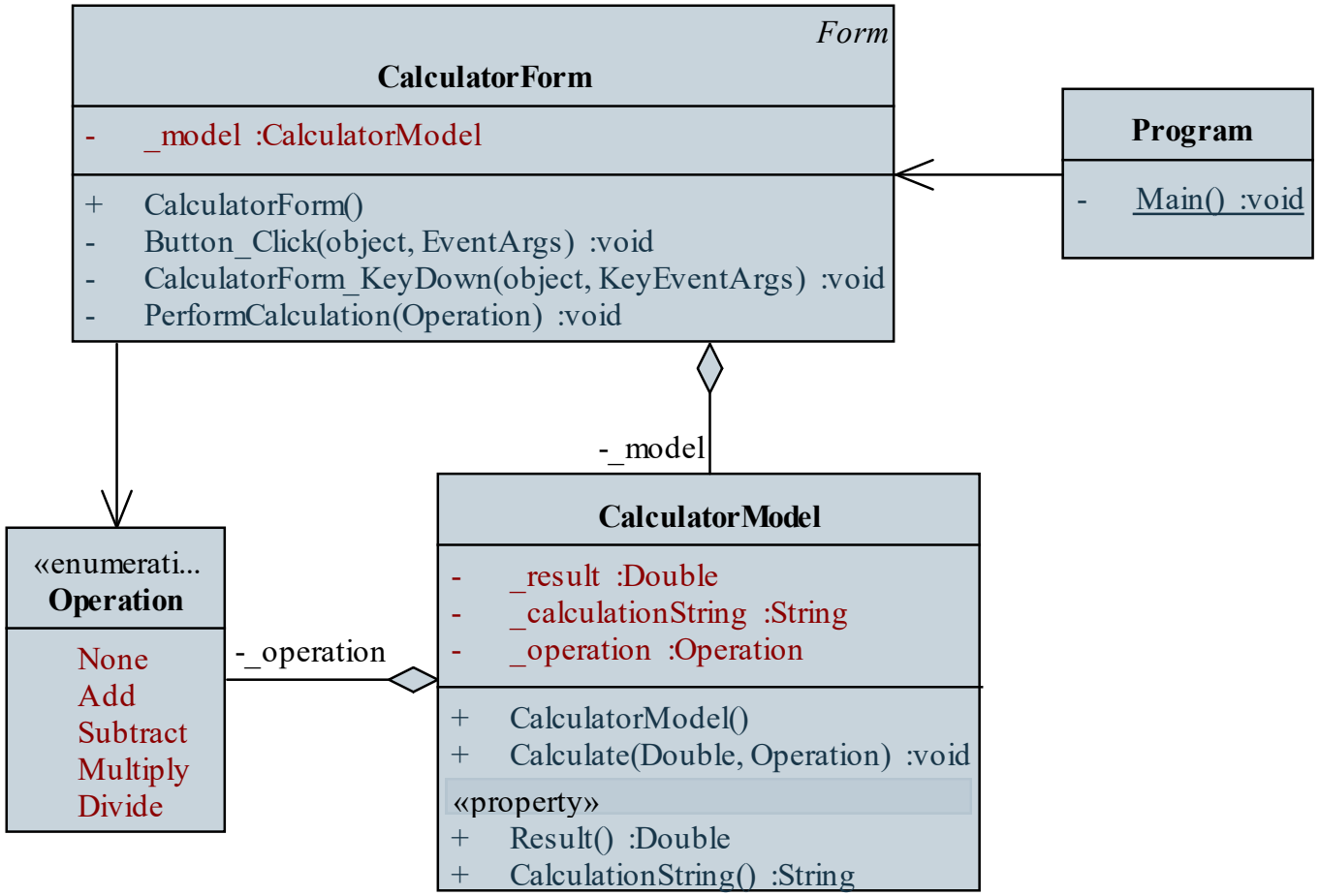
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- 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, a tevékenység végrehajtását pedig külön alprogramba helyezzük (**`PerformCalculation`**)

Windows Forms alkalmazások architektúrája

Példa

Tervezés:



Windows Forms alkalmazások architektúrája

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 alkalmazások architektúrája

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
        …
```

Windows Forms alkalmazások architektúrája

Események létrehozása és kiváltása

- Amennyiben adatokat szeretnénk továbbítani az eseménnyel, célszerű saját argumentumtípust létrehozni, ehhez
  - az **`EventArgs`** típusból származtatunk egy speciális típust, pl.:

```
class MyEventArgs : EventArgs {
    Object SomeData { get; set; }
}
```

  - a saját eseményargumentumot (vagy általánosabban bármilyen típust), mint sablonparaméter rögzíthetjük az esemény delegáltjában, pl.:

```
class EventClass {
    event EventHandler<MyEventArgs> MyEvent;
}
```

Windows Forms alkalmazások architektúrája

Események létrehozása és kiváltása

- 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, new MyEventArgs{ … });
        // kiváltjuk: a küldő az aktuális objektum,
        // az eseményargumentumokat megadjuk
```

Windows Forms alkalmazások architektúrája

Események létrehozása és kiváltása

- A szintaxist egyszerűsíthetjük a *null-conditional operator* (`?.`) használatával, amellyel egy objektum tagja csak akkor kerül kiértékelésre, ha az objektum nem **`null`** érték volt.
  - Pl.:
    ```
    var address = order?.User?.Address;
    ```
  - Az **`address`** változó értéke null lesz, ha akár az **`order`**, akár az **`order.User`** értéke **`null`** volt.
- Így egyszerűsíthetjük az esemény kiváltását:
  ```
  ec.MyEvent?.Invoke(this, new MyEventArgs{ … });
  // kiváltjuk az eseményt,
  // ha van hozzárendelve eseménykezelő
  ```

Windows Forms alkalmazások architektúrája

Események létrehozása és kiváltása

- C# 8.0 vagy újabb verzió esetén a *nullable reference types* használata esetén érdemes az esemény delegáltjának típusát ennek megfelelően *nullable*-nek jelölni.
  - Pl.:

```
class EventClass {
    event EventHandler<MyEventArgs>? MyEvent;
}
```

  - Így fordítási időben figyelmeztetést kapunk, ha ellenőrizetlen módon váltunk ki egy eseményt.

Windows Forms alkalmazások architektúrája

Példa

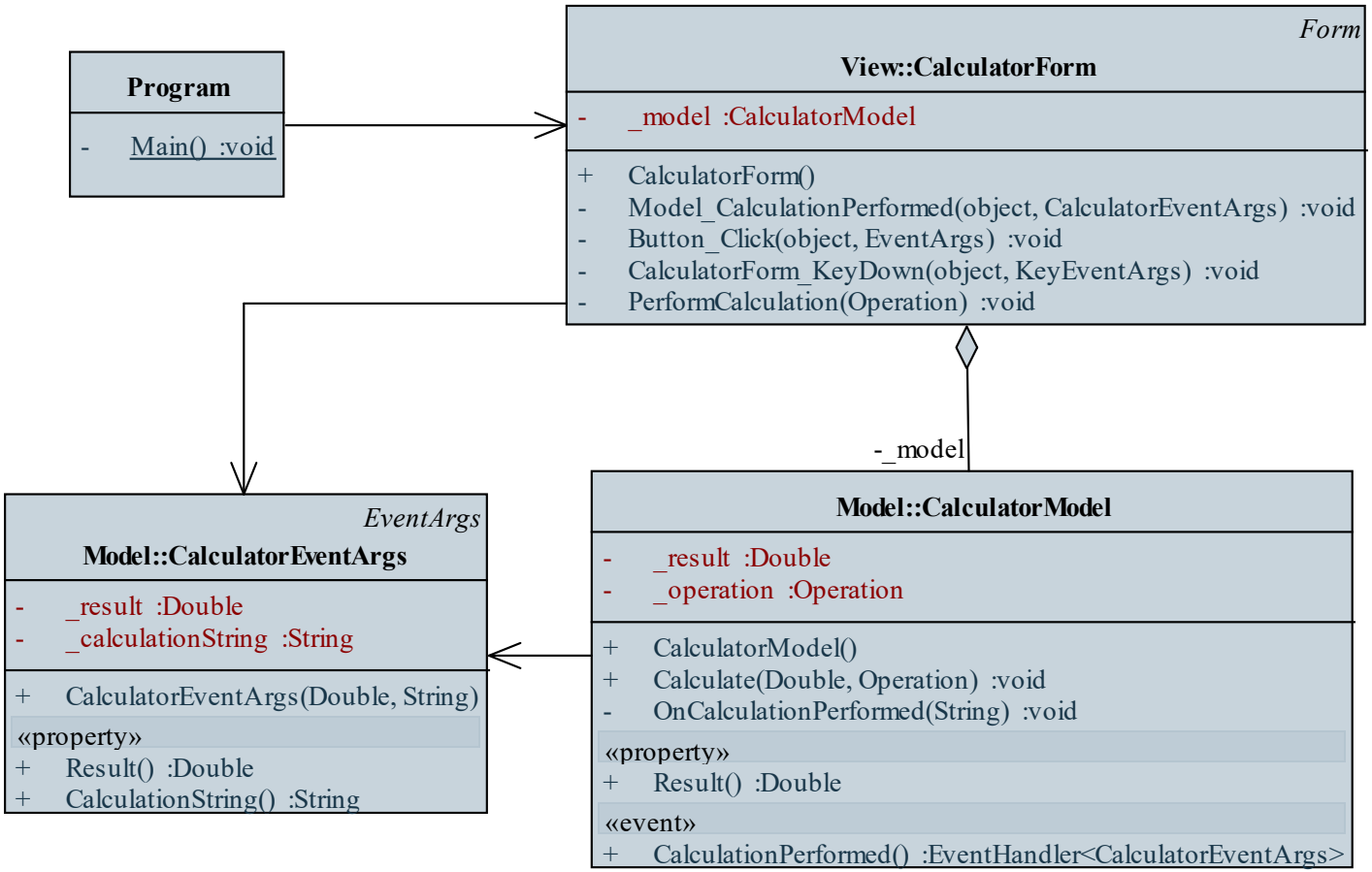
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- a modell fogja jelezni a számítás befejezését, ehhez felveszünk egy új eseményt (**`CalculationPerformed`**), amelyet a nézet feldolgoz
- 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 alkalmazások architektúrája

Példa

Tervezés:



Windows Forms alkalmazások architektúrája

Példa

Megvalósítás (`CalculatorModel.cs`*):*

- Esemény deklarálása:

```
public event EventHandler<CalculatorEventArgs>?
    CalculationPerformed;
    // számítás végrehajtásának eseménye
```

- Esemény kiváltása:

```
CalculationPerformed?.Invoke(this,
            new CalculatorEventArgs(
                _result,
                calculationString));
            // feltöltjük az eseményargumentumot
```