# *Programozás III*

GRAFIKUS FELÜLETEK,
SWING

## GRAFIKUS JAVA ALKALMAZÁSOK

## GRAFIKA – ALAPOK

Korábbi programjaink
- algoritmusvezérelt
- konzolos programok

Mire jók a konzolos programok?

Fő ok: fontos a fogalmak pontos megismeréséhez

A Java két lehetőséget biztosít grafikus felületek készítésére

**AWT** – Abstract Window Toolkit

**SWING**

és **SWT** ☺

## AWT

Az adott operációs rendszer ablakozó rendszeréhez biztosít szabványos hozzáférést.

– különböző platformokon különböző, de gyors megjelenés
– szűkebb eszközkészlet


csomag: java.awt

## SWING

Minden elem Java-ban van megvalósítva

– platformfüggetlen, de lassúbb megjelenés
– bővebb eszközkészlet

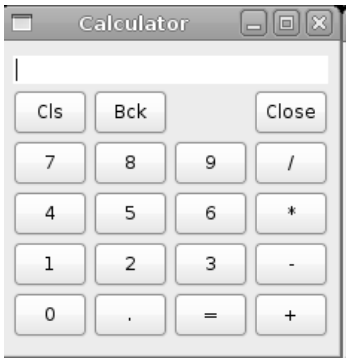

csomag: javax.swing

A Swing komponensosztályok azonosítói J betűvel kezdődnek.

## SWT (STANDARD WIDGET TOOLKIT)

Nem része a szabvány Java API-nak.
Eclipse alternatíva az AWT és a Swing helyett.

Például:



## AWT HIERARCHIA

Egy Java alkalmazás felhasználói interfésze (GUI) **komponensek**ből áll

A komponensek ősosztálya a **java.awt.Component** osztály

Példák komponensekre

- nyomógomb
- ablak
- szövegmező…stb.

## AWT HIERARCHIA

A Component osztály leszármazottja a **Container**

Képes több komponens összefogására:

- *az add()* metódussal adható hozzá komponens
- a *remove()* metódussal távolíthatók el
- önálló *Layout*-tal rendelkezik
  - ez azt mondja meg, hogy a benne található komponensek milyen elrendezésben jelenjenek meg

## AWT ALAPKOMPONENSEK

**Window:**

A Container leszármazottja,
az ablakozó rendszer ablak fogalmának absztrakciója,
„nyers” fogalom:
- nincs kerete
- nincs fejléce
- nincs menüsora

Önállóan nem létezhet!!!

## AWT ALAPKOMPONENSEK

**Frame:**

A Window leszármazottja,
grafikus felületű programok alapja,
az operációs rendszer ablakozó rendszeréből való
- van fejléce
- van kerete
- adható hozzá menüsor

## AWT ALAPKOMPONENSEK

**LayoutManager**

- több komponens területi elrendezéséért felelős
- egy Container objektumhoz tartozik
- a komponenseket a megadott szabálynak megfelelően rakja ki a képernyőre
- módosíthatja a komponensek méretre vonatkozó tulajdonságait (rugalmas igazodás a Container-hez)

Néhány LayoutManager:

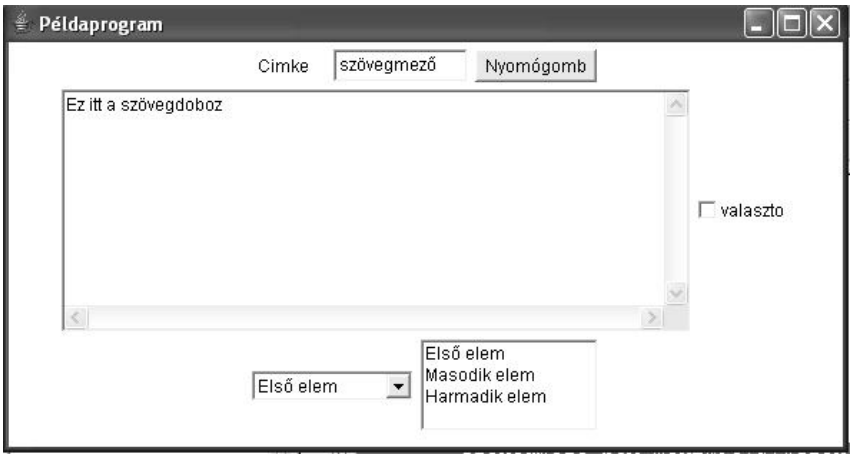
- FlowLayout – sorfolytonos elrendezés
- BorderLayout – határmenti elrendezés
- GridLayout – rácsos elrendezés
- AbsoluteLayout…stb.

## AWT ALAPKOMPONENSEK

További komponensek

- **Panel** – panel
- **Label** – címke
- **Button** – nyomógomb
- **TextField** – szövegmező
- **TextArea** – szövegdoboz
- **CheckBox** – jelölő négyzet
- **Choice** – legördülő menü
- **List** – lista

## AWT ALAPKOMPONENSEK




## A GRAFIKUS PROGRAMOZÁS MENETE

Az AWT alapkomponensek a **java.awt** csomagban kaptak helyet, ezért ezt szükséges importálni.

A program alapja a Frame, ezért az osztályunk a **Frame** osztály leszármazottja lesz (öröklődés!)

Példa: Hozzuk létre az alábbi grafikus felületet:

## A GRAFIKUS PROGRAMOZÁS MENETE

Importáljuk a grafikához szükséges csomagot, illetve hozzuk létre a Pelda osztályt a **Frame** osztály kiterjesztésével és az osztály törzsében deklaráljuk a szükséges komponenseket:

```
import java.awt.*;

public class Pelda extends Frame {

    private Panel panel;
    private Button gomb;
    private TextField szam1, szam2;
    private Label eredmeny;
```

## A GRAFIKUS PROGRAMOZÁS MENETE

Hozzunk létre egy konstruktort, amely paraméterként kapja az ablak címét:

```
public Pelda(String cimke, int szelesseg, int magassag){
    inicializalas();
    this.setSize(szelesseg, magassag);
    this.setTitle(cimke);
}
```

## A GRAFIKUS PROGRAMOZÁS MENETE

Az inicializáló metódus:

```
public void inicializalas(){

    // komponensek létrehozása
    panel = new Panel();
    szam1 = new TextField("",10);
    szam2 = new TextField("",10);
    gomb = new Button(" = ");
    eredmeny = new Label("eredmény");

    // elrendezésmenedzser
    this.setLayout(new FlowLayout());
    panel.setLayout(new FlowLayout());

    //komponensek felrakása

    add(panel);
    panel.add(new Label("kérek két számot: "));
    panel.add(szam1);
    panel.add(new Label(" + "));
    panel.add(szam2);
    panel.add(gomb);
    panel.add(eredmeny);

    pack();
}
```

## A GRAFIKUS PROGRAMOZÁS MENETE

Végül készítsük el az indító main() metódust:

```
public static void main(String args[]){
    Pelda g = new Pelda("példaprogram", 600, 100);
    g.setVisible(true);
}
```

☺: megjelenik a várt ablak

☹: de nem jó semmire – még az ablakbezáró gomb sem működik

eseményvezérelt programozás kell

## ESEMÉNYVEZÉRELT PROGRAMOZÁS

Algoritmus-vezérelt programozás:
a kód határozta meg a program menetét

Eseményvezérelt programozás:
a felhasználó határozza meg a program menetét a beavatkozásaival.

ilyen beavatkozások például:
kattintás az egérrel
billentyű leütés … stb.

## ESEMÉNYEK – EVENTS

Az esemény a vele összefüggő információkat magába foglaló objektum.

Ezek az információk:

- az esemény forrása
- az esemény típusa
- az esemény időpontja…stb.

Az események mindig valamilyen forrásobjektumon keletkeznek:
nyomógombon,
szövegmezőn,…stb.

## ESEMÉNYEK – EVENTS

Az alacsony szintű események (pl. billentyű-, egér-esemény) az operációs rendszer szintjén keletkeznek, melyek egy eseménysorba (event queue) kerülnek.

Az eseményt először a forrásobjektum kapja meg, majd továbbítja azt az esemény figyelőinek. (Szóhasználat: „forrásobjektumon keletkezik”.)

Az események mindig sorban, egymás után keletkeznek, nem keletkezhet egyszerre két esemény.

Egy komponensen csak akkor keletkezhet esemény, ha az eleme az alkalmazás komponens-hierarchiájának, és **látható**.

## ESEMÉNYEK KEZELÉSE

Az eseményekre csak akkor reagálhatunk, ha figyeljük őket.

Minden forrásobjektumhoz ki kell jelölni úgynevezett figyelőobjektumokat (ezekben kezeljük a forrásobjektumon keletkezett eseményeket).

Egy forrásobjektumhoz több figyelőobjektumot adhatunk hozzá az ***add***Listener()*** segítségével.

Például: addActionListener()

## ESEMÉNYEK KEZELÉSE

Egy objektum csak akkor figyelhet egy eseményt, ha hozzáadtuk a forrásobjektumhoz, és osztálya implementálja a figyelőinterfészt.

**Adapterosztályokkal** kiküszöbölhető, hogy implementálnunk kelljen az összes – figyelőinterfészbeli – metódust.

??? – Ne essen pánikba, inkább tegyük működőképessé az előző példát!

## ESEMÉNYEK KEZELÉSE – PÉLDA

Az események kezelése a java.awt.event csomaggal oldható meg:

```
import java.awt.*;
import java.awt.event.*;
```

A deklarálás, konstruktor ugyanaz, mint eddig.

Az inicializalas() metódus bővül majd az események figyelésével:

- az ablak bezáró gombját és
- az általunk felrakott nyomógombot kell figyelni.

Vagyis az eddigi metódusba még bele kell írni a következőket:

## ESEMÉNYEK KEZELÉSE – PÉLDA

Az ablak bezáró gombjának működése:

- Az esemény a **WindowEvent** esemény
- Az eseménykezelő a **WindowListener**, amelyet az **addWindowListener** metódussal adunk hozzá a keret objektumhoz
- A **WindowAdapter** osztály **windowClosing()** metódusát definiáljuk az ablakbezárás művelet végrehajtásához.

```
//ablakesemény kezelése
this.addWindowListener(new WindowAdapter(){
    @Override
    public void windowClosing(WindowEvent e){
        ablakbezaras();
    }
});
```

## ESEMÉNYEK KEZELÉSE – PÉLDA


```
// ablakesemény kezelése
AblakFigyelo af = new AblakFigyelo();
this.addWindowListener(af);
```

```
public class AblakFigyelo extends WindowAdapter{

    @Override
    public void windowClosing(WindowEvent e) {
        ablakbezaras();
    }
```

Tömörebben:

```
//ablakesemény kezelése
this.addWindowListener(new WindowAdapter(){
    @Override
    public void windowClosing(WindowEvent e){
        ablakbezaras();
    }
});
```

## ESEMÉNYEK KEZELÉSE – PÉLDA

A gombnyomás eseménykezelése

**ActionEvent** esemény

**ActionListener** eseményfigyelő

**ActionListener** interfész – egyetlen – metódusának az ***actionPerformed()*** metódusnak a definiálása

```
gomb.addActionListener(new ActionListener(){
    @Override
    public void actionPerformed(ActionEvent e){
        gombnyomas(e);
    }
});
```

**addActionListener**(ActionListener l)
Adds the specified action listener to receive action events from this button.

**actionPerformed**(ActionEvent e)
Invoked when an action occurs.

## ESEMÉNYEK KEZELÉSE – PÉLDA

Az eseménykezelő metódusok törzse ezekre a metódusokra hivatkozik:

```
public void ablakbezaras(){
    System.out.println("Viszlát");
    System.exit(0);
}

public void gombnyomas(){
    int a = Integer.parseInt(szam1.getText());
    int b = Integer.parseInt(szam2.getText());

    eredmeny.setText(String.valueOf(a+b));
}
```

A main() metódus ugyanaz, mint eddig.

## ESEMÉNYEK KEZELÉSE – „GENERÁLT”

A NetBeans ugyanilyeneket generál:

```
cimke.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseEntered(java.awt.event.MouseEvent evt) {
        cimkeMouseEntered(evt);
    }
});
```

Természetesen több metódus is implementálható, pl.:

```
cimke.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseEntered(java.awt.event.MouseEvent evt) {
        cimkeMouseEntered(evt);
    }
    public void mouseExited(java.awt.event.MouseEvent evt) {
        cimkeMouseExited(evt);
    }
});
```

## ESEMÉNYEK KEZELÉSE

A gombnyomás eseményfigyelőjének azonban csak egyetlen metódusa van:

```
gomb.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        gombActionPerformed(evt);
    }
});
```

**FONTOS**: A gombnyomás-esemény mindig az **actionPerformed** és sohasem az egérkattintás!

De mit jelent ez az egymásba ágyazott struktúra?

## ESEMÉNYEK KEZELÉSE – PÉLDA

```
public class PeldaFrame extends javax.swing.JFrame {

    public PeldaFrame() {
        initComponents();
        EgerAdapter adapter = new EgerAdapter();
        this.addMouseListener(adapter);
    }

    private static class EgerAdapter extends MouseAdapter {

        public EgerAdapter() {
        }

        @Override
        public void mouseClicked(MouseEvent me) {
            kattintasHatasa(me);
        }

        private void kattintasHatasa(MouseEvent me) {
            System.out.println("x: " + me.getX() + " y: " + me.getY());
        }
    }
```

belső osztály

## ESEMÉNYEK KEZELÉSE – PÉLDA

```
cimke.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseEntered(java.awt.event.MouseEvent evt) {
        cimkeMouseEntered(evt);
    }
});
```

beágyazott osztály

## PÉLDA BEÁGYAZOTT OSZTÁLYRA

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            DiakosFrame frame = new DiakosFrame();
            frame.setVisible(true);
            frame.indit();
        }
    });
}
```

## KITÉRŐ: OSZTÁLYTÍPUSOK

Az eddig használtak, azaz amelyek nincsenek beágyazva másik osztályba vagy interfészbe: **top level class**
Csak public vagy módosító nélküli (csomag szinten public) lehet.

Osztályokon belül deklarált osztályok: **beágyazott osztályok** (**nested class**). Fajtái:

– statikus beágyazott osztály (static nested class),
– belső osztály (inner class) – ezek közvetlenül elérik a tartalmazó osztály más tagjait is.

Mind a négy hozzáférés lehet.

http://www.developer.com/java/article.php/859381

http://javagyik.blogspot.com/2011/03/osztalyok-tipusai.html

## MELYIKET SZERESSEM?

AWT vagy Swing?

A két osztálykönyvtár nem teljesen azonos feladatkört lát el, csak részben van átfedés.

Az AWT elavult, nem fejlesztik tovább, a Swing modernebb, többet tud.

Ugyanakkor az AWT hierarchiája kicsit áttekinthetőbb, ezért vettük ezt. Ha ezt megérti, akkor utána már könnyedén érti a Swing-et is.

## MELYIKET SZERESSEM?

Swing vagy valami más keretrendszer?
(Pl. GWT, Vaadin, stb.)

Mára már a Swing is kissé elavult.

Ugyanakkor még mindig jól használható alkalmazásokat lehet fejleszteni vele, és jó alap a továbblépéshez.

## MELYIKET SZERESSEM?

Osztályok importálása:

AWT számára:

```
import java.awt.*;          // AWT-komponensek,
                            // elrendezés-kezelő
import java.awt.event.*;    // eseménykezelő
```

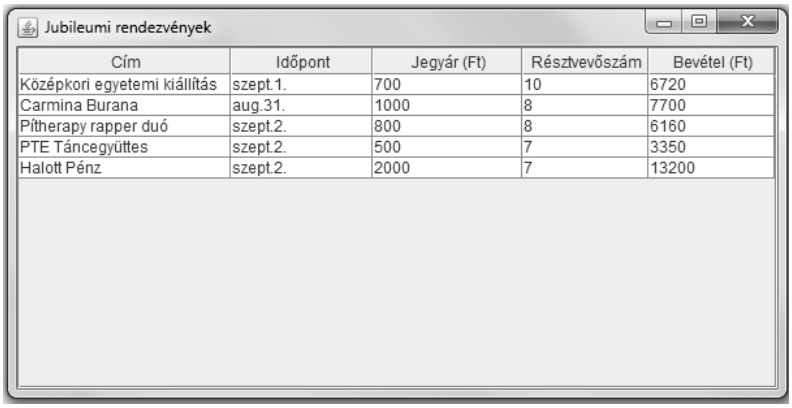
Swing számára:

```
import javax.swing.*;       // Swing-komponensek
import java.awt.*;          // elrendezés-kezelő
import java.awt.event.*;    // eseménykezelő
```

(A javax.swing.event csomag további eseményeket is definiál.)

## SWING PÉLDA

Ld. gyak3 indító, ill. későbbi példája – nézze is át az indító feladat ide vonatkozó részét és a generált initComponents() metódusokat is a most tárgyaltak alapján.




| Cím | Időpont | Jegyár (Ft) | Résztvevőszám | Bevétel (Ft) |
|---|---|---|---|---|
| Középkori egyetemi kiállítás | szept.1. | 700 | 10 | 6720 |
| Carmina Burana | aug.31. | 1000 | 8 | 7700 |
| Pítherapy rapper duó | szept.2. | 800 | 8 | 6160 |
| PTE Táncegyüttes | szept.2. | 500 | 7 | 3350 |
| Halott Pénz | szept.2. | 2000 | 7 | 13200 |

## SWING ALKALMAZÁSOK (ISM.)

Swing felületű, eseményvezérelt alkalmazás létrehozása:

1. JFrame alapú osztály létrehozása

   Szerepe: vezérlés

2. A frame-re rákerül egy vagy több panel.

   Szerepük: erre kerülnek az egyéb komponensek

3. Az egyes komponensekhez eseményeket rendelünk.

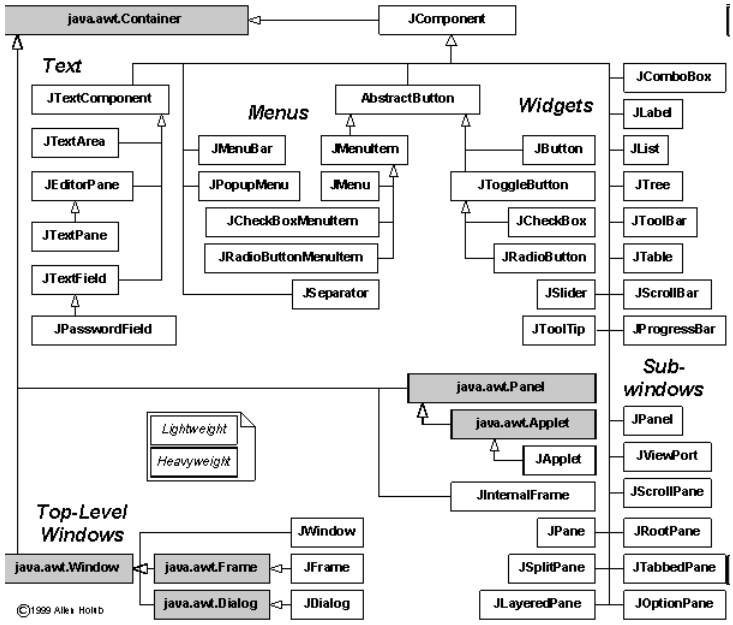
   Szerepük: ezek hatására hajtódik végre a feladat.

## SWING ALKALMAZÁSOK (ISM.)

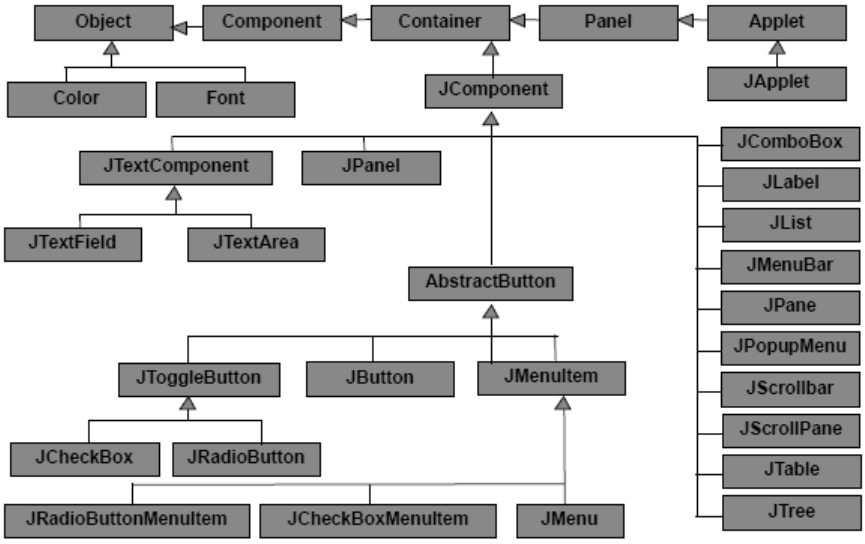
Swing felületű, eseményvezérelt alkalmazás inicializálása:

1. Komponensek definiálása, tulajdonságaik beállítása.
2. Elrendezés-menedzser beállítása.
3. Komponensek felrakása.
4. Eseményfigyelők beállítása.

## A SWING SZERKEZETE




## A SWING SZERKEZETE



## ELRENDEZÉS-MENEDZSER

Néhány elrendezés:

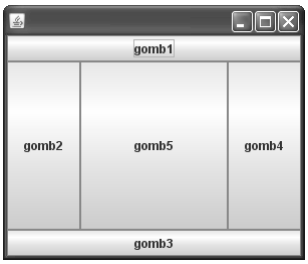

FlowLayout


BoxLayout

## ELRENDEZÉS-MENEDZSER

Néhány elrendezés:


GridLayout




BorderLayout

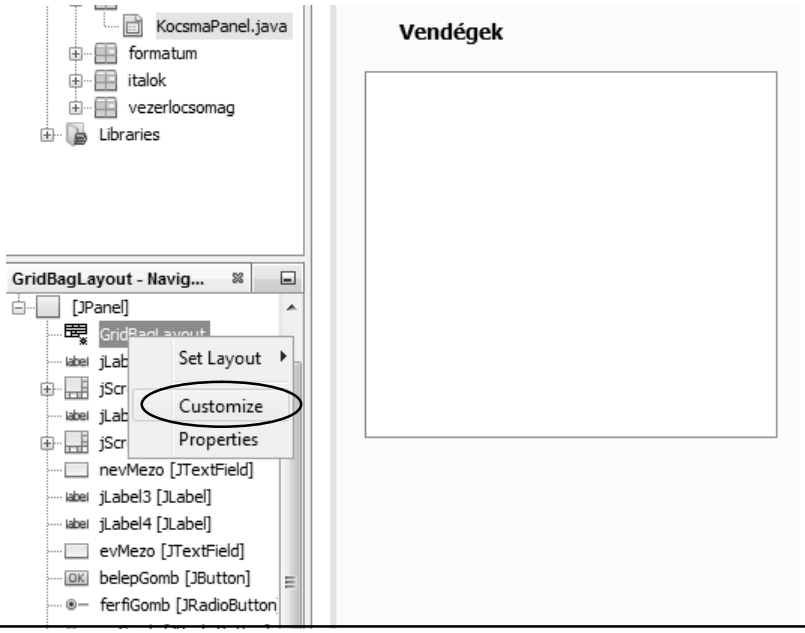
## ELRENDEZÉS-MENEDZSER

Egyszerű feladatok esetén használhatjuk a Netbeans által felkínált free-design-t vagy a Null Layout-ot, de komolyabb (vagy igényesebb) feladatok esetén két elrendezés javasolt:

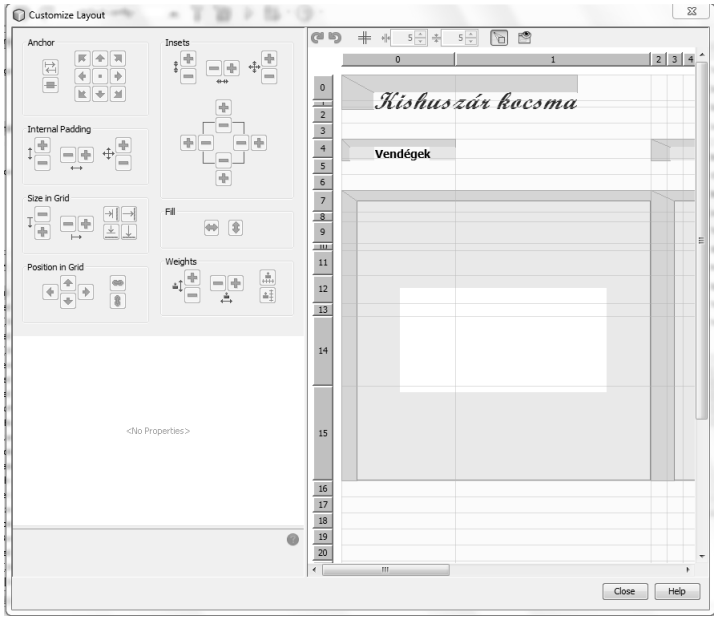
1. BorderLayout

2. GridBagLayout

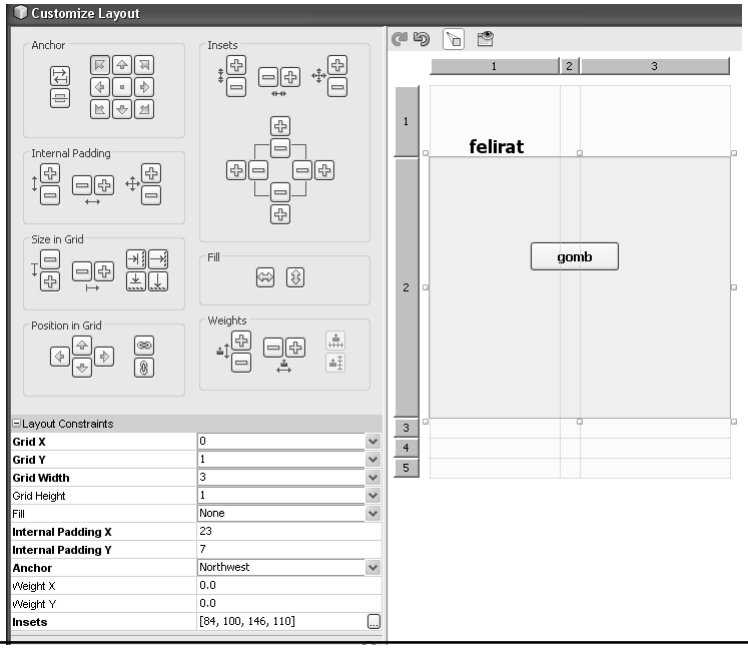
## GRIDBAGLAYOUT



## GRIDBAGLAYOUT




## GRIDBAGLAYOUT



## GRIDBAGLAYOUT

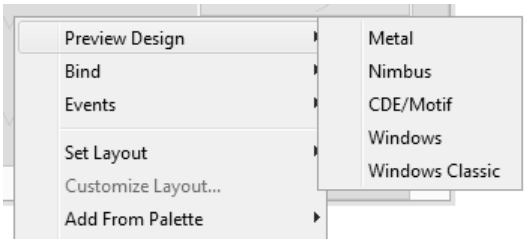
http://download.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

http://netbeans.org/kb/docs/java/gbcustomizer-basic.html

## KITÉRŐ: LOOK AND FEEL

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)
```

A NetBeans-ben ez is beállítható.
Egy felületre kattintva (jobb gomb) meg lehet nézni a választható kinézeteket, és igény esetén át lehet állítani.




https://www.youtube.com/watch?v=ieT1fPSL6Jw

## SWING LISTAKEZELÉS

Egy JList Stringek sorozatát jeleníti meg, de hogyan kerülhetnek a listába objektumok?

Lista-modelleket használunk:

valamilyenModel modell = new valamilyenModel();

**jlstValami.setModel(modell);**

## SWING LISTAKEZELÉS

A használt modellek a ListModel interface implementált osztályai.

1. AbstractListModel:

   Segítségével tetszőleges objektumok kezelésére vonatkozó saját modellt generálhatunk.

2. DefaultListModel:

   Az AbstractListModel egy kiterjesztése.

## SWING LISTAKEZELÉS

A modellek szintén konténerek, vagyis a listákhoz hasonló módon kezelhetőek. (sok saját metódus)

Ha új elem kerül beléjük, akkor arról értesíteni kell a megfelelő JList-et (különben nem jeleníti meg az új elemet, pontosabban az új elem toString()-jét):

• DefaultListModel esetén ez az értesítés automatikus;

• AbstractListModel használatakor a SajatListModel osztályban létre kell hoznunk egy saját metódust az új elem hozzáadásához, majd a hozzáadás után:

```
this.fireIntervalAdded(objektum, kezdoIndex, vegIndex);
```

## A SWING SZERKEZETE

A Java-ban mindegyik modell interfészhez (ButtonModel, ListModel, stb.) készítettek egy alapértelmezett modellt (DefaultButtonModel, DefaultListModel, stb.), melyet a megfelelő komponens alapértelmezésben használ, de ez a modell kicserélhető.

A modellek eseményt dobhatnak, ennek megfelelően vannak figyelőláncaik.

A felhasználó a komponenssel van közvetlen kapcsolatban.

## SWING MODELLEK

Pl.:

| Modell interfész | néhány metódusa | Alapértelmezett osztály | Mi használja? |
|---|---|---|---|
| ButtonModel | addActionListener<br>addItemListener<br>isEnabled<br>isSelected | DefaultButtonModel | AbstractButton |
| ListModel | addListDataListener<br>getElementAt<br>getSize | DefaultListModel | JList |
| ListSelectionModel | addListSelectionListener<br>clearSelection<br>getSelectionMode | DefaultListSelection-Model | JList |
| BoundedRange-Model | addChangeListener<br>getMinimum<br>getValue | DefaultBounded-RangeModel | JScrollBar |
| Document | addDocumentListener<br>getLength<br>getText<br>insertString | AbstractDocument | JTextComponent |

## SWING LISTAKEZELÉS

Látható, hogy a listakezelés három komponensre bontható:

1. Manipulálhatjuk a listához tartozó adatokat.

   modell (model)

2. Megjelenítjük a képernyőn.

   nézet (view)

3. Eseményeket rendelhetünk hozzá.

   vezérlő (controller)

## A SWING SZERKEZETE

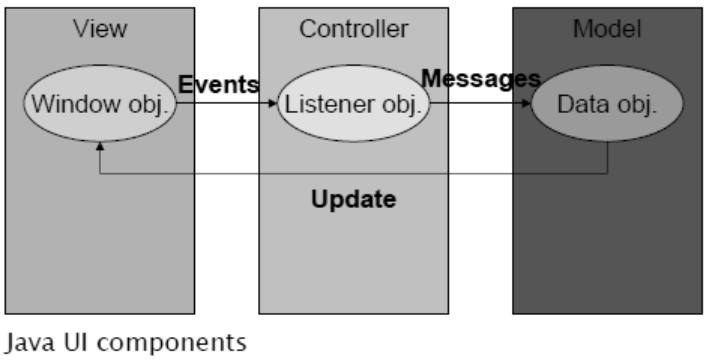
A Swing komponenseket az **MVC (Model-View-Controller)** architektúra (tervezési minta) alapján készítették.

**Model** (modell): A komponens adatai, állapota.
A modell felelős a komponens adatainak tárolásáért.
Egy modellen több nézet is osztozhat. (pl. ListModel)

**View** (nézet): A komponens megjelenése a képernyőn.
A felhasználói eseményeket továbbítja a vezérlő rétegnek. (pl. JList)

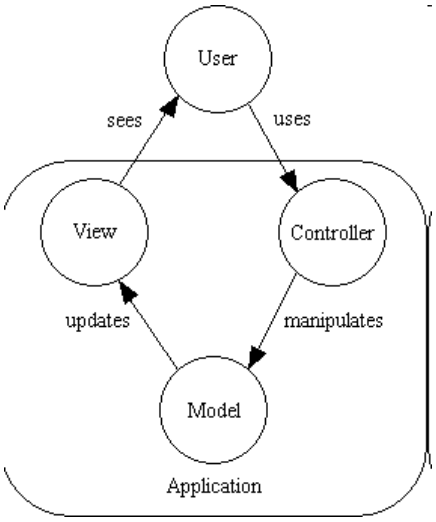
**Controller** (vezérlő): A felhasználói eseményeket feldolgozó programlogika. Felelős a külvilág eseményeire való reagálás módjáért. Reagálásként megváltoztathatja az adatmodell adatait.

## A SWING SZERKEZETE



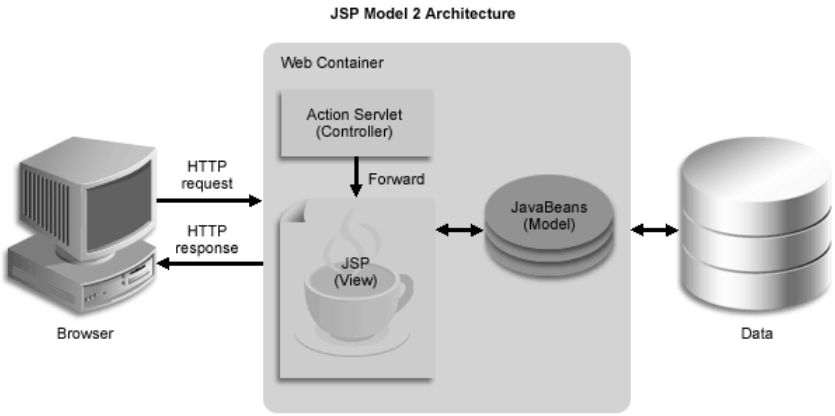

Java UI components

## A SWING SZERKEZETE

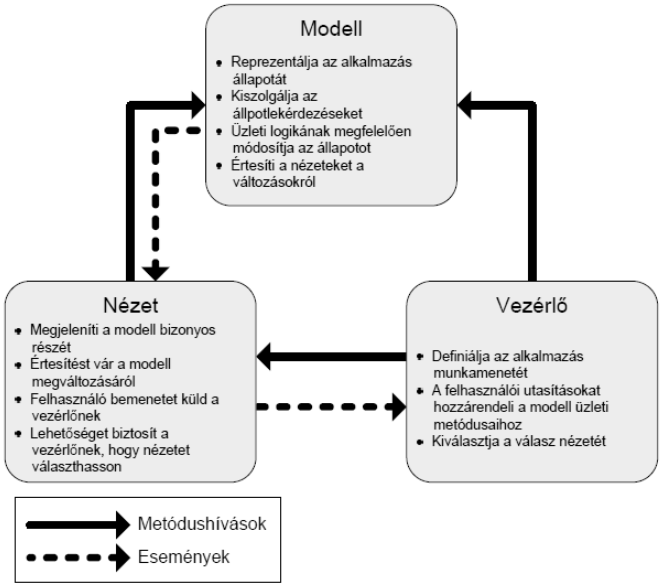



A Swing-ben az MVC egyszerűsített változatát alkalmazzák, azaz a vezérlés és a megjelenítés össze van vonva.
A grafikus komponens saját maga felelős a megjelenítésért és a felhasználói események feldolgozásáért.
Egy Swing komponens a modell és a grafikus megjelenítés közti kommunikációt vezérli.

## MVC

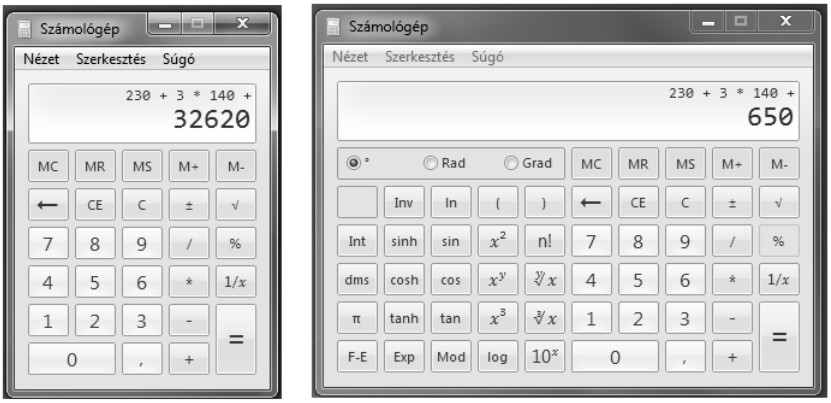


## MVC



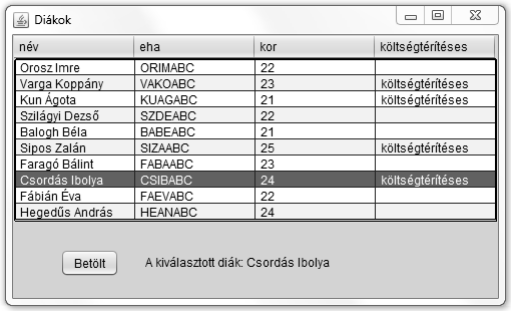

## MVC

Miért fontos a szétválasztás?




Egy gyöngyszem. ☺

## TÁBLÁZATKEZELÉS AZ EDDIGIEK FÉNYÉBEN




„Tisztességes” megoldás: táblamodell használatával.

## TÁBLÁZATKEZELÉS AZ EDDIGIEK FÉNYÉBEN

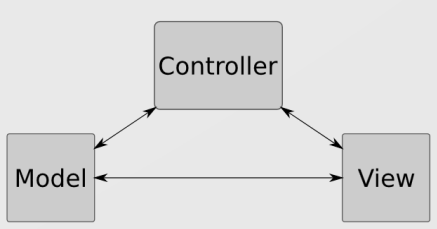
Lehetséges megoldások:

1. DefaultTableModel használata
2. Saját TableModel.

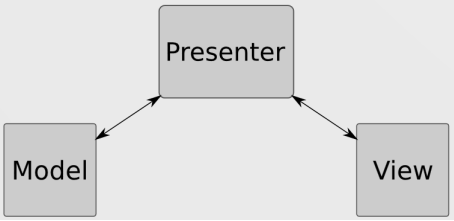
1.a – az oszlopfejet a modellben adjuk meg

1.b – a táblázatban

## MVC → MVP




Az üzleti és a megjelenési réteg teljes szétválasztása.




## NÉHÁNY LINK

http://download.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

http://netbeans.org/kb/docs/java/gbcustomizer-basic.html

http://download.oracle.com/javase/tutorial/uiswing/components/

http://download.oracle.com/javase/tutorial/uiswing/examples/components/index.html

http://www.oracle.com/technetwork/java/architecture-142923.html