Programozás III

GAFIKA 2

SWING ALKALMAZÁSOK

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.

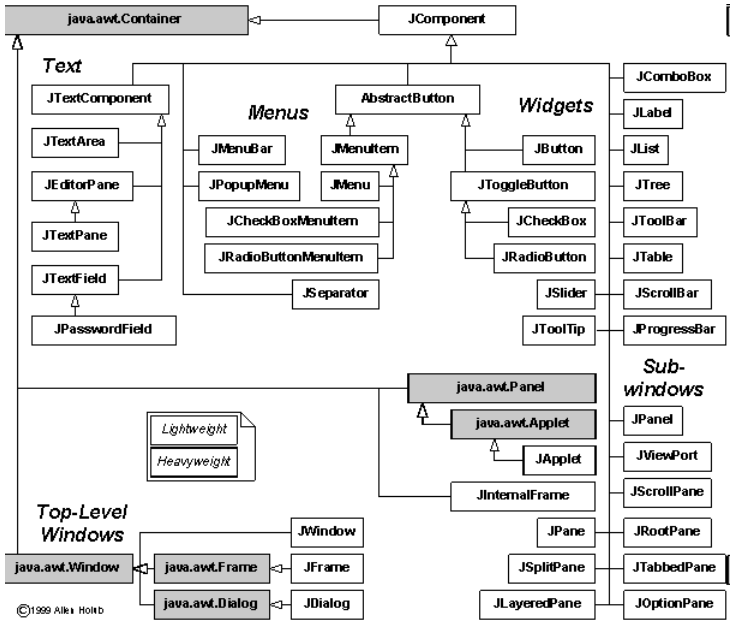
   Szerepük: ezek hatására hajtódik végre a feladat.

SWING ALKALMAZÁSOK

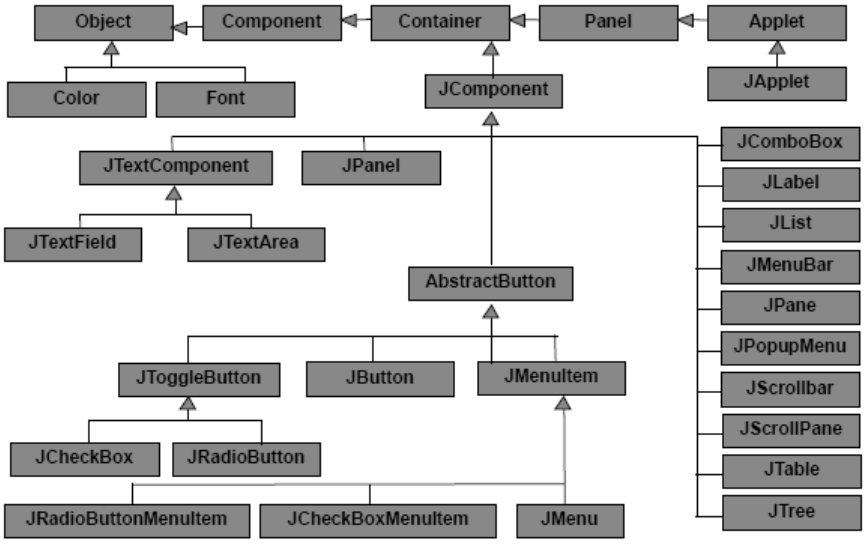
Swing felületű, eseményvezérelt alkalmazás inicializálása:

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

A SWING SZERKEZETE



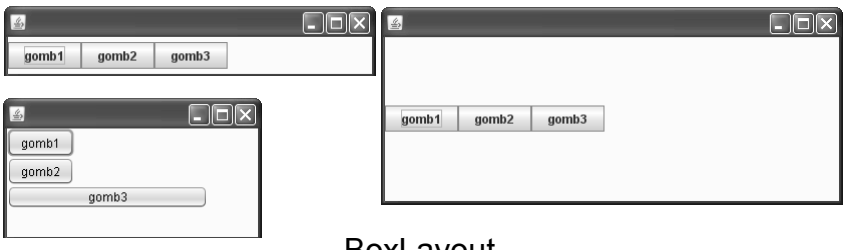
A SWING SZERKEZETE



ELRENDEZÉS-MENEDZSER

Néhány elrendezés:

FlowLayout

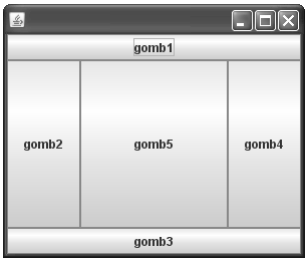


BoxLayout

ELRENDEZÉS-MENEDZSER

Néhány elrendezés:

GridLayout



BorderLayout

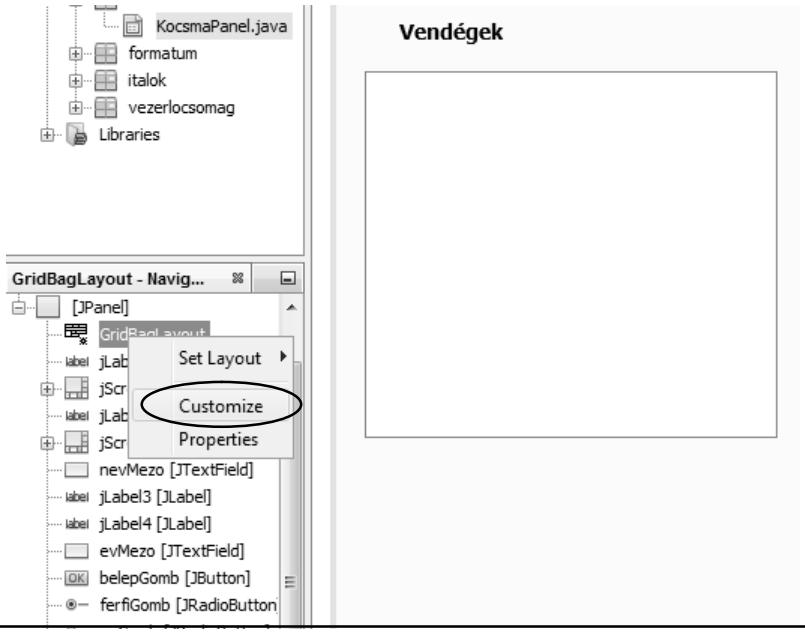
ELRENDEZÉS-MENEDZSER

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:

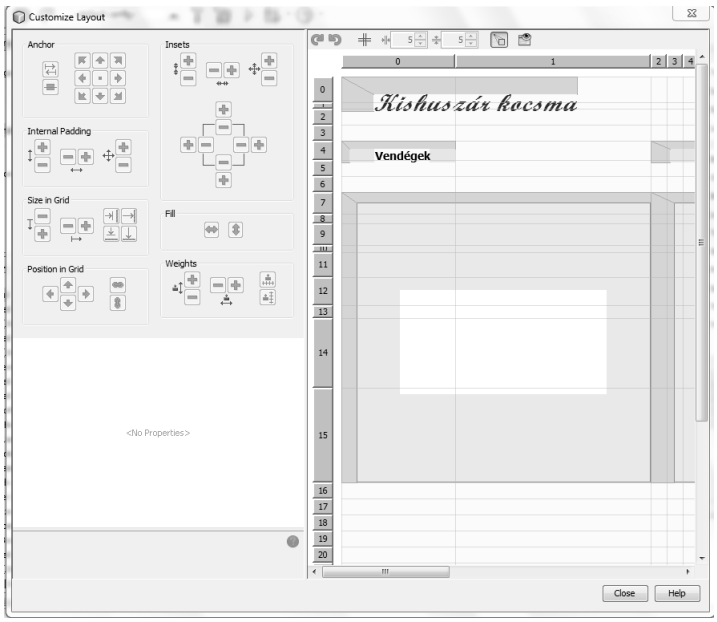
1. BorderLayout

2. GridBagLayout

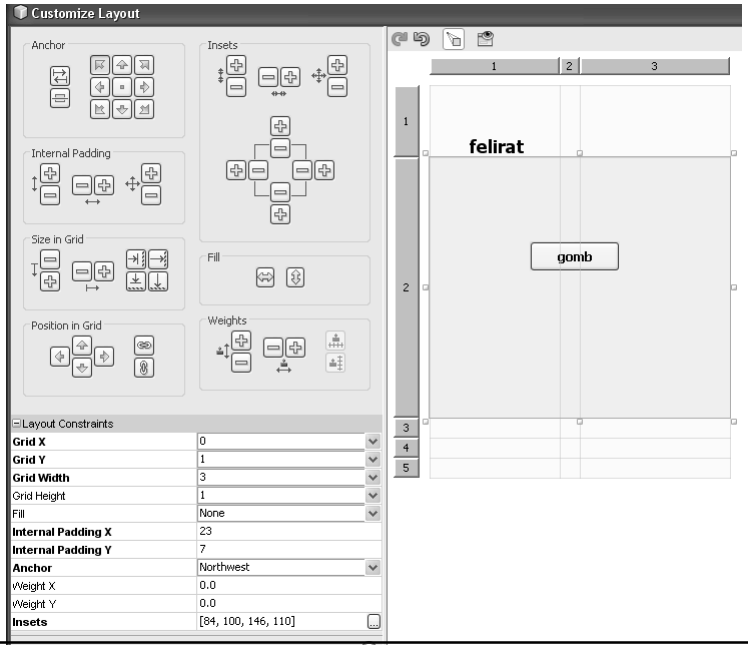
GRIDBAGLAYOUT



GRIDBAGLAYOUT



GRIDBAGLAYOUT



GRIDBAGLAYOUT

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

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

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.

Csomag: java.awt.event

ESEMÉNYEK – EVENTS

Az alacsony szintű események (pl. billentyű-, egér-esemény) az operációs rendszer szintjén keletkeznek, melyeket egy eseménysorban (event queue) helyez el.

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 úgy nevezett 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.

Pl.:

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

ESEMÉNYEK KEZELÉSE

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 ElrendezesJFrame() {
    initComponents();
    EgerAdapter adapter = new EgerAdapter();
    cimke.addMouseListener(adapter);
}

class EgerAdapter extends MouseAdapter {
    @Override
    public void mouseEntered(java.awt.event.MouseEvent evt) {
        cimkeMouseEntered(evt);
    }
}
```

belső osztály

```
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:

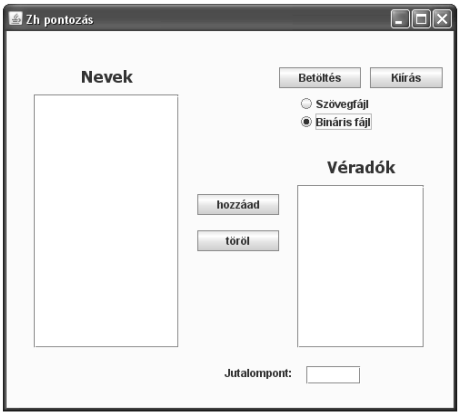
– statikus beágyazott osztály (static nested class),
– belső osztály (inner class) – ezek közvetlenül elérik a tartalmazó osztály más tagjait is.

Mind a négy hozzáférés lehet.

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

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

PÉLDA



A felület kialakításához szükséges komponensek:

- JList,
- JButton,
- JRadioButton + Button Group,
- JTextField,
- JLabel,
- JFileChooser
- JOptionPane

SWING LISTAKEZELÉS

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

Lista-modelleket használunk:

valamilyenModel modell = new valamilyenModel();

jlstValami.setModel(modell);

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 Object típusú objektumok kezelésére (azaz nem generikus).

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 addItemListener isEnabled isSelected	DefaultButtonModel	AbstractButton
ListModel	addListDataListener getElementAt getSize	DefaultListModel	JList
ListSelectionModel	addListSelectionListener clearSelection getSelectionMode	DefaultListSelection-Model	JList
BoundedRange-Model	addChangeListener getMinimum getValue	DefaultBounded-RangeModel	JScrollBar
Document	addDocumentListener getLength getText insertString	AbstractDocument	JTextComponent

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

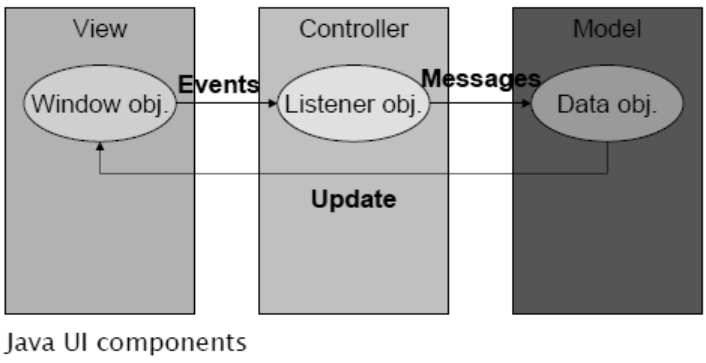
A Swing komponenseket az **MVC (Model-View-Controller)** architektúra (tervezési minta) alapján készítették.

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)

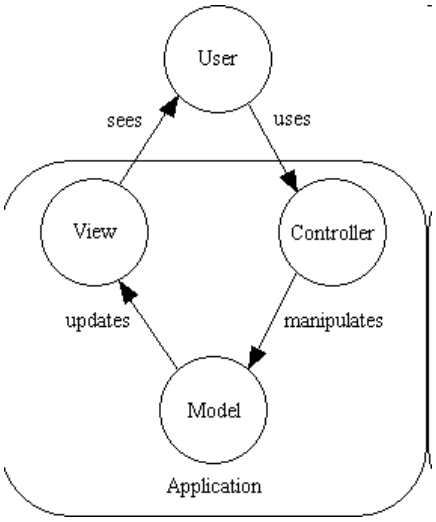
Controller (vezérlő): A felhasználói eseményeket feldolgozó programlogika. Felelős a külvilág eseményeire való reagálás módjáért. Reagálásként megváltoztathatja az adatmodell adatait.

A SWING SZERKEZETE



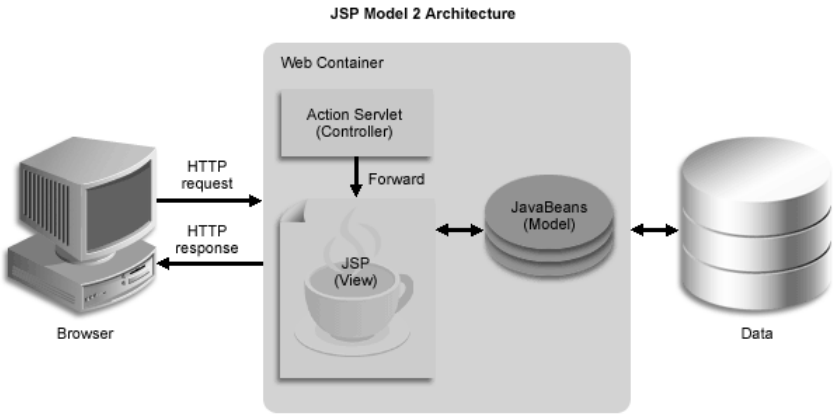
Java UI components

A SWING SZERKEZETE

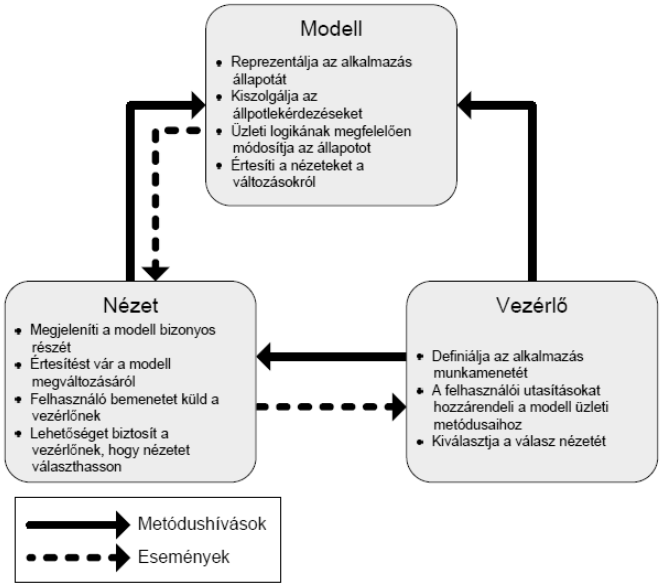


A Swing-ben az MVC egyszerűsített változatát alkalmazzák, azaz a vezérlés és a megjelenítés össze van vonva.
A grafikus komponens saját maga felelős a megjelenítésért és a felhasználói események feldolgozásáért.
Egy Swing komponens a modell és a grafikus megjelenítés közti kommunikációt vezérli.

MVC

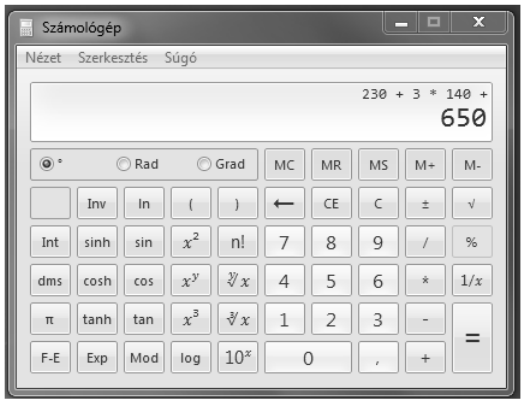


MVC



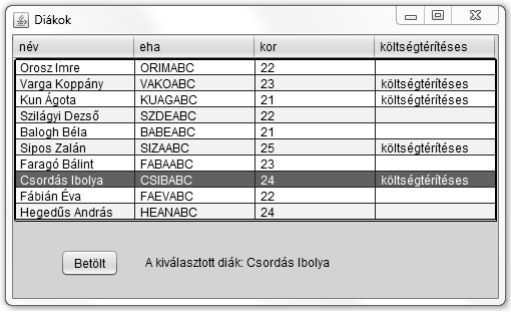
MVC

Miért fontos a szétválasztás?



Egy gyöngyszem. ☺

TÁBLÁZATKEZELÉS AZ EDDIGIEK FÉNYÉBEN



„Tisztességes” megoldás: táblamodell használatával.

TÁBLÁZATKEZELÉS AZ EDDIGIEK FÉNYÉBEN

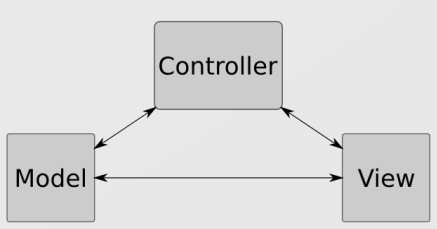
Lehetséges megoldások:

1. DefaultTableModel használata
2. Saját TableModel.

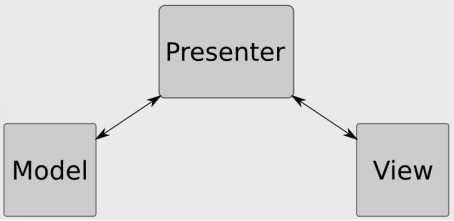
1.a – az oszlopfejet a modellben adjuk meg

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