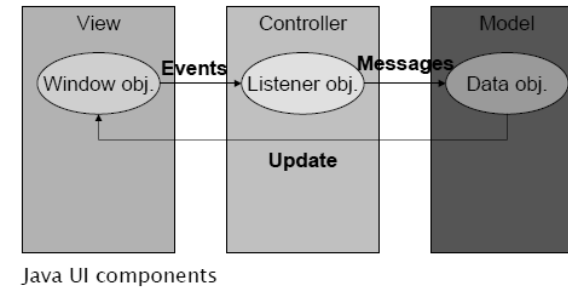


## Programozás III

RENDEZETT  
JLIST

### A SWING SZERKEZETE



### ISMÉTLÉS: A SWING SZERKEZETE

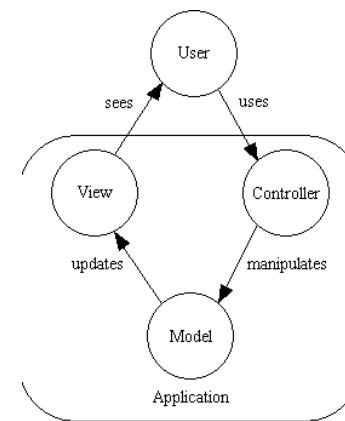
A Swing komponenseket az **MVC (Model-View-Controller)** architektúra (tervezési minta) alapján készítették.

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

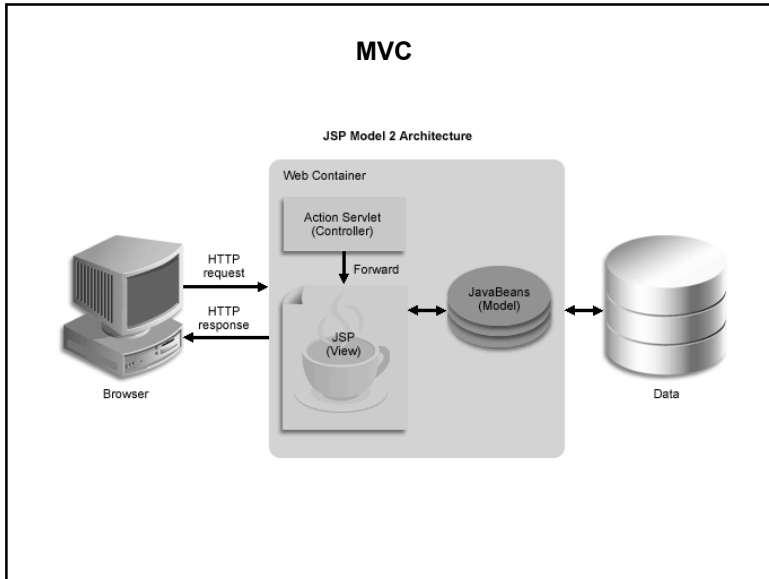
**View** (nézet): A komponens megjelenése a képernyőn. A felhasználói eseményeket továbbítja a vezérlő rétegnek. (pl. JList)

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

### A SWING SZERKEZETE



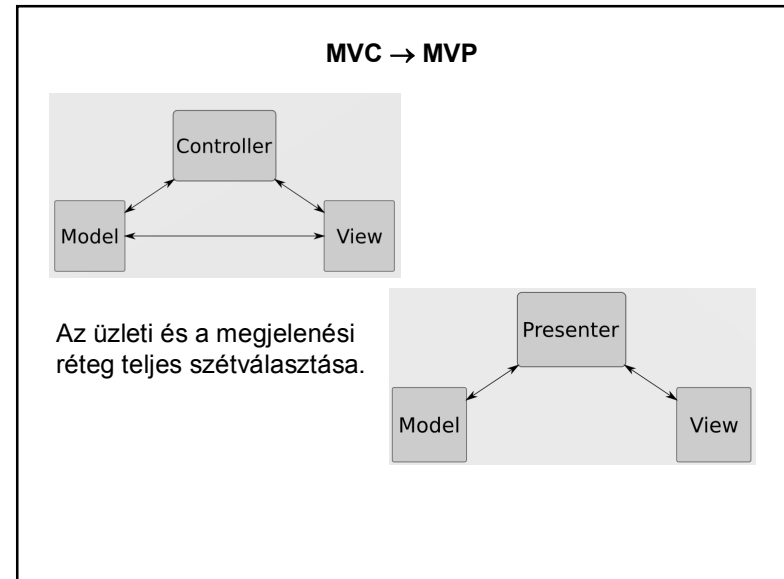
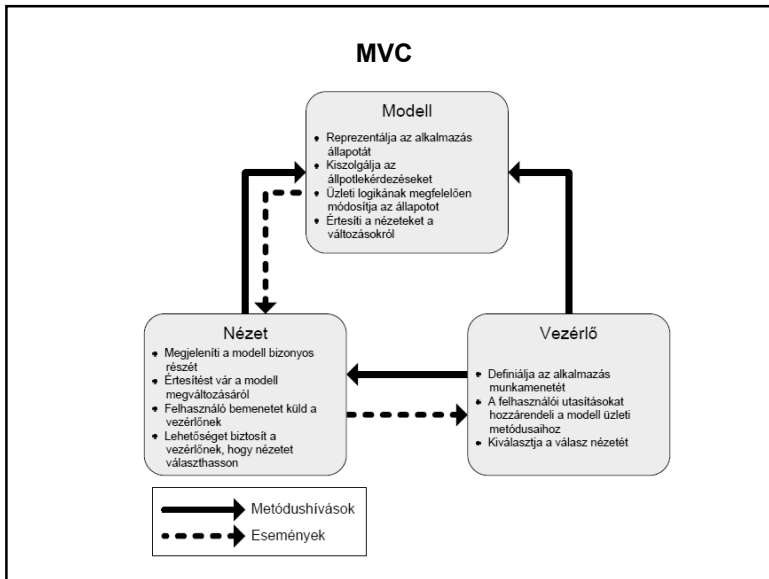
A Swing-ben az MVC egyszerűsített változatát alkalmazzák, azaz a vezérlés és a megjelenítés össze van vonva. A grafikus komponens saját maga felelős a megjelenítésért és a felhasználói események feldolgozásáért. Egy Swing komponens a modell és a grafikus megjelenítés közti kommunikációt vezérli.



### MVC

Miért fontos a szétválasztás?

Egy gyöngyszem. 😊



## 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 LISTAKEZELÉS (ISMÉTLÉS VÉGE )

A modellek szintén konténerek, vagyis a listákhoz hasonló módon kezelhetők. (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 SajátListModel osztályban létre kell hoznunk egy saját metódust az új elem hozzáadásához, majd a hozzáadás után:  

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

## 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

## LIST vs MODELL

a/ Létrehozunk List-et is és modellt is, és oda-vissza rakosgatjuk az elemeket.

Miért nem jó, ill. mikor jó, mikor nem?

b/ Ez jobb?

Object[] tomb = modell.toArray(); vagy

List<Tipus> temp =Collections.list(modell.elements());

(vagy egy for ciklusban rakjuk át)

– és ezt rendezzük, majd visszaírjuk;

Valamivel jobb, de nem tökéletes.

## JLIST ELEMEINEK RENDEZÉSE

Ha rendezetten akarjuk kiíratni egy JList elemeit, akkor a modellt kell rendezni.

A DefaultListModel-nek nincs sort() metódusa. ☹



Saját modellt kell írni.

(Általában: egy kicsit is érdekesebb feladathoz illik saját modellt írni, vagyis ez nem nagy igény.)

## SAJÁT MODELL

```
public void addDiak(Diak diak) {
    this.diakLista.add(diak);
    int ujIndex = diakLista.size() - 1;
    this.fireIntervalAdded(diak, ujIndex, ujIndex);
}

public void removeDiak(Diak diak) {
    int toroltIndex = diakLista.indexOf(diak);
    this.diakLista.remove(diak);
    this.fireIntervalRemoved(diak, toroltIndex, toroltIndex);
}

public void sort() {
    Collections.sort(diakLista);
    System.out.println(diakLista);
    this.fireContentsChanged(diakLista, 0, diakLista.size() - 1);
}
```

stb... ld. ea\_mintapeldak\sajat\_modelll

```
public class DiakModell<E extends Diak> extends AbstractListModel{

    private List<E> diakLista;

    public DiakModell() {
        super();
        this.diakLista = new ArrayList<>();
    }

    @Override
    public int getSize() {
        return diakLista.size();
    }

    // Ezek kötelezően implementálandó metódusok
    @Override
    public Diak getElementAt(int index) {
        return diakLista.get(index);
    }

    public void addDiak(E obj) {
        this.diakLista.add(obj);
        int ujIndex = diakLista.size()-1;
        this.fireIntervalAdded(obj, ujIndex, ujIndex);
    }
}
```

## JLIST ELEMEINEK „AZONNALI” RENDEZÉSE

```
public class SortableListModel<T extends Comparable<T>>
    extends AbstractListModel {

    private List<T> model = new ArrayList<>();
    private boolean isSorted = false;

    // Kötelező implementálni a köv. két metódust (generálható):
    @Override
    public Object getElementAt(int index) {
        return model.get(index);
    }

    @Override
    public int getSize() {
        return model.size();
    }
}
```

## JLIST ELEMEINEK „AZONNALI” RENDEZÉSE

A SortableListModel további metódusai:

```
/**
 * Ha utólag akarjuk rendezni a modellt, akkor ezt
 * a metódust kell hívni, de csak akkor hajtódik végre,
 * ha még nincs rendezve.
 */
public void sort() {
    if (!isSorted) {
        Collections.sort(model);
        fireContentsChanged(this, 0, model.size() - 1);
    }
}
```

Minden változásról értesíteni kell a JList-et.  
(a DefaultListModel-ben is van ilyen)

```
/**
 * Ha sort értéke true, akkor már eleve rendezetten rakja be az elemet,
 * vagyis az eddig berakottakat is rendezi, és ennek megfelelő helyre
 * rakja az újat.
 * Ha a sort értéke false, akkor a lista végére teszi az új elemet.
 * @param element
 * @param sort
 */
public void addElement(T element, boolean sort) {
    if (!sort) {
        addElement(element);
        isSorted = false;
    } else {
        if (!isSorted) {
            sort();
        }
        int index = Collections.binarySearch(model, element);

        if (index < 0)
            addElement(element, -index - 1);
        else
            addElement(element, index);
        isSorted = true;
    }
}
```

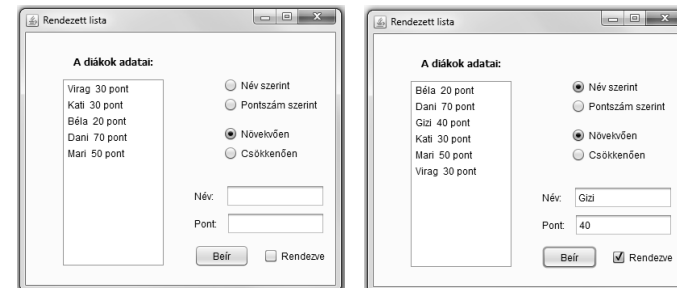
## JLIST ELEMEINEK „AZONNALI” RENDEZÉSE

A SortableListModel további metódusai:

```
/**
 * a modell végére szűr be egy elemet
 * @param element
 */
private void addElement(T element) {
    this.addElement(element, model.size());
    this.fireIntervalAdded(element, model.size()-1, model.size()-1);
}

/**
 * a modell adott indexű helyére szűrja be az elemet
 * @param element
 * @param index
 */
private void addElement(T element, int index) {
    model.add(index, element);
    this.fireIntervalAdded(this, index, index);
}
```

## JLIST ELEMEINEK RENDEZÉSE



„Magyarított” és kommentezett változatát ld.

ea\_mintapeldak/rendezettJList

(nem teljes, próbálja meg befejezni)

### **JLIST ELEMEINEK RENDEZÉSE**

A modellben a bináris rendezést használtuk.

Ennek Java megvalósításához ld. rekurzió.