A programozás alapjai 3.

Java GUI: MVC

Goldschmidt Balázs
balage@iit.bme.hu

1

Összetett komponensek

- JList
  - elemek kiválasztása egy lisából
- JComboBox
  - legördülő lista opcionális szövegdobozzal
- JTable
  - táblázat cellákban lévő adatokkal
- JTree
  - hierarchikus adatreprezentáció fastruktúrában
- …

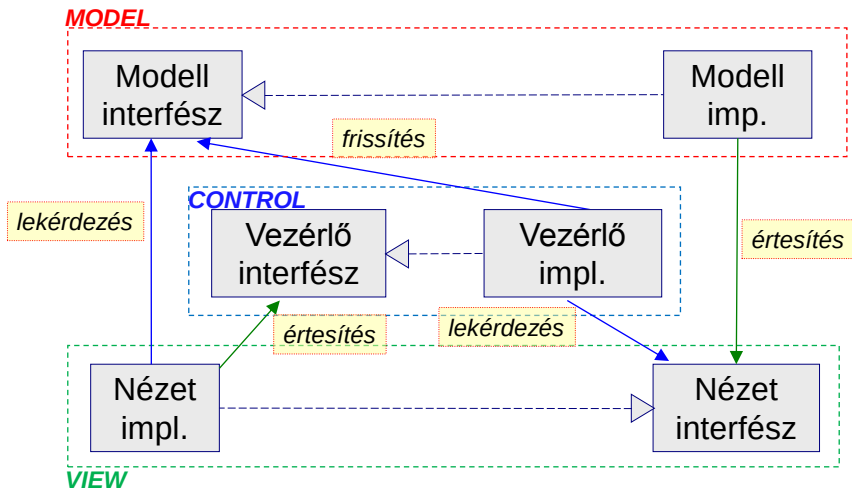
2

Felelősségek szétválasztása

- Model-View-Controller (MVC) minta
  - modell (model): *adatok kezelése és tárolása*
  - nézet (view): *GUI reprezentáció és vizuális információ*
  - vezérlő (controller): *eseménykezelés listener implementálásával*
- Egyszerű komponensek
  - a *listener* van külön, az adat és a nézet együtt marad
- Összetett komponensek
  - nézet és adat is külön
    - pl. *JList* és *ListModel*, *JTable* és *TableModel*
    - van alapértelmezett modellimplementáció

3

MVC architektúra



6

Összetett komponens: `JList`

- Objektumok listáját mutatja
  - alapértelezetten a *toString()* metódus eredménye látszik
- Objektumok külön tárolva
  - *Object[ ]* (nem módosítható)
  - *Vector<T>* (nem módosítható)
  - *ListModel* implementáció (pl. *DefaultListModel*)
- Görgetést a *JScrollPane* végzi
  - *JScrollPane* egy konténer, amely görgetést támogat
  - *JList* csak a listát rajzolja ki

Modell rész: **`ListModel`**

- **`interface javax.swing.ListModel`**
  - **`Object getElementAt(int index)`**
    - visszaadja az *index* pozíción lévő elemet
  - **`int getSize()`**
    - visszaadja a tárolt elemek számát
  - **`void addListDataListener (ListDataListener l)`**
  - **`void removeListDataListener (ListDataListener l)`**
    - listener hozzáadása/eltávolítása

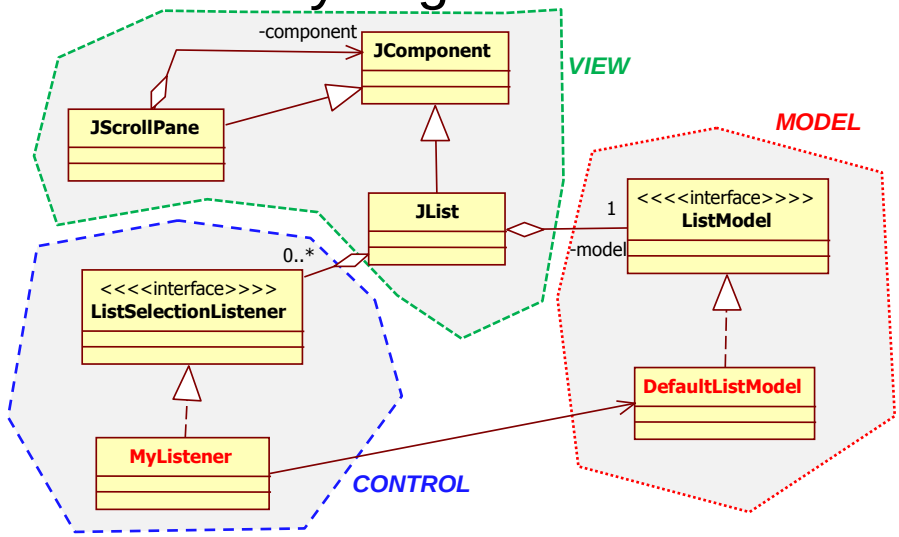
ListDataListener

- Amikor a modell változik, a listener-ek értesítést kapnak
- *JList*-nek saját implementációja van rá:
  - **`BasicListUI.ListDataHandler`**
- Metódusok:
  - **`void intervalAdded(ListDataEvent e)`**
  - **`void intervalRemoved(ListDataEvent e)`**
    - az *e* paraméter által jelölt intervallum módosult
  - **`void contentsChanged(ListDataEvent e)`**
    - komplex változás történt

DefaultListModel

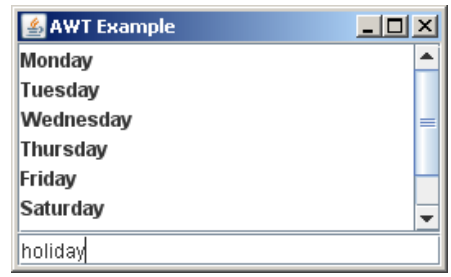
- Implementálja a *ListModel* interfészt
- Metódusok hasonlóak a *java.util.List* metódusaihoz:
  - **`void add(int index, Object o)`**
  - **`int size()`**
  - **`Object get(int index)`**
  - **`Object remove(int index)`**
  - ...

JList osztálydiagram



JList példa

- Készítsünk egy ablakot az alábbi tartalommal:
  - egy *lista* karakterláncok megjelenítésére
  - *egy szövegdoboz* karakterláncok bevitelére
    - bevitel után a lista frissül



JList példa

```
public class SwingList implements  ActionListener{
    JTextField tf;
    JList list;
    DefaultListModel model;
    public void actionPerformed(ActionEvent ae) {
        model.addElement(tf.getText());
        tf.setText("");
    }
    static public void main(String args[]) {
        (new SwingList()).run();
    }
    ...
```


JList példa

```
    ...
    public void run() {
        JFrame f = new JFrame("AWT Example");
        tf = new JTextField("", 20);
        model = new DefaultListModel();
        list  = new JList(model);
        JScrollPane pane = new JScrollPane(list);
        tf.addActionListener(this);
        f.add(tf, BorderLayout.SOUTH);
        f.add(pane, BorderLayout.CENTER);
        f.pack();
        f.setVisible(true);
    }
}
```

MVC működés közben

- Ha a modell változik, a nézet frissül
  - a *model.addElement("Holiday")* hívás hatására
  - a nézet frissül
- Ha a nézet kap egy eseményt, a vezérlő értesül
  - *ListSelectionListener*
    - `void valueChanged(ListSelectionEvent e)`
  - A kiválasztott elem(ek) elérése:
    - *JList-ben* `getSelectedIndex()` vagy `getSelectedIndices()`
    - *ListModel-ben* `getElementAt()` metódus

MVC elemzés

- **Modell (M)**: **`DefaultListModel`**
  - elemek gyűjteményét kezeli
  - ha változik, értesíti a *Nézetet*
- **Nézet (V)**: **`JList`***`(+BasicListUI.ListDataHandler)`*
  - csak az elemek megjelenítéséért felel
- **Vezérlő (C)**: **`SwingList`***`(ActionListener impl.)`*
  - frissíti a *Modellt* a nézet elemek eseményei alapján

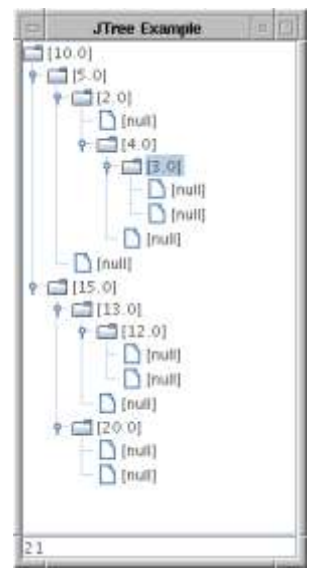
Összetett komponens: **JTree**

- Fastruktúra (hierarchia) megjelenítése
- Modell: **`TreeModel`**

- Inicializálható egy **`TreeNode`**-fával
  - a fa gyökerét kell megadni

JTree példa

- **Feladat**: bináris keresőfa megjelenítése
  - double értékeket tárolunk
  - új elemek adhatók hozzá



JTree példa

- **`BinTree`**
  - bináris keresőfa implementáció (DIY)
  - ha nincs gyerek: *Null* objektum *null* érték helyett
    - egyszerűbb implementáció
    - *Null Object* tervezési minta
    - láncolt lista strázsa elemeihez hasonló megoldás
    - drága!
  - beszúrás után az új elemhez vezető utat kell visszaadni

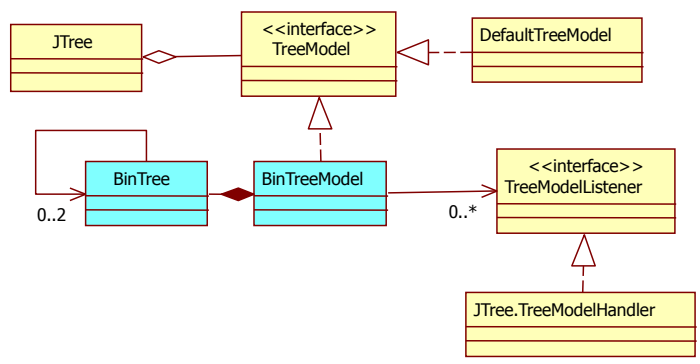
JTree példa

- **`BinTreeModel`**
  - modell a **`JTree`** számára
  - hozzáférést ad a **`BinTree`** implementációhoz
    - `BinTree root = new BinTree()`
  - **`TreeModelListener`**-ek kezelése
    - `Vector<TreeModelListener> listeners`
    - `public void addTreeModelListener(TreeModelListener l)`
    - `public void removeTreeModelListener(TreeModelListener l)`

JTree példa

- **`BinTreeModel`** (folytatás)
  - modell elérése:
    - `public Object getChild(Object parent, int index)`
    - `public int getChildCount(Object parent)`
    - `public int getIndexOfChild(Object parent, Object child)`
    - `public Object getRoot()`
    - `public boolean isLeaf(Object node)`
    - ***`public void valueForPathChanged(TreePath path, Object newValue)`***
    - **`public void insert(double d)`**

JTree osztálydiagram



JTree példa: alkalmazás

```
JTextField tf;
BinTreeModel btm;
```

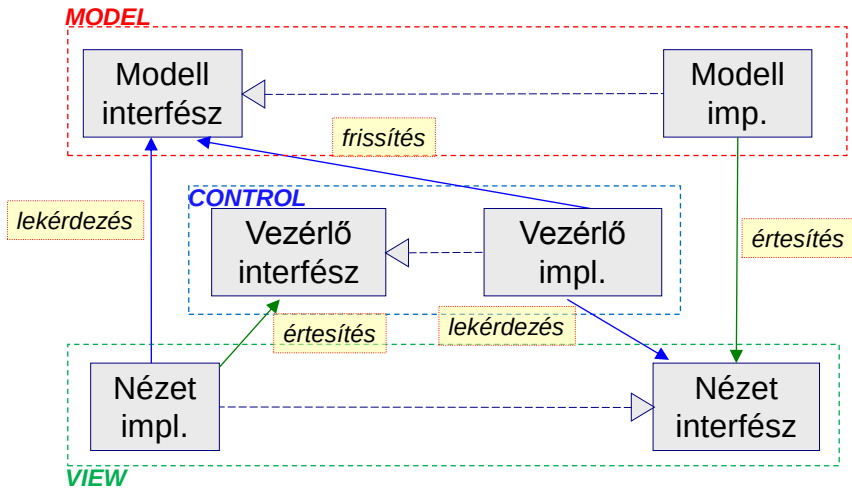
```
public void actionPerformed(ActionEvent ae) {
  double d = Double.parseDouble(tf.getText());
  btm.insert(d);
  tf.setText("");
}
```

```
tf = new JTextField("", 20);
btm = new BinTreeModel();
JTree tree = new JTree(btm);
JScrollPane scrollPane = new JScrollPane(tree);
tf.addActionListener(this);
```

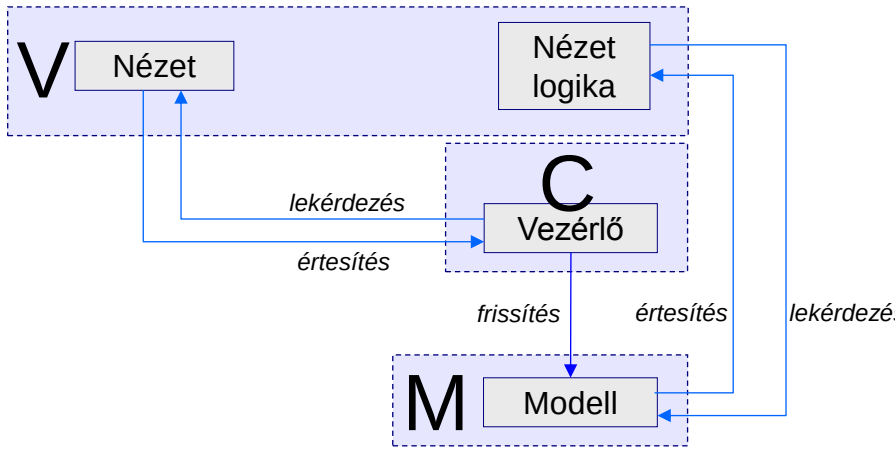

Rétegezési kérdések

- Vékony vagy vastag kliens
  - hol van az üzleti logika?
    - MVC: mennyi üzleti logika van a nézetben?
- Többrétegű architektúra
  - böngésző, alkalmazásszerver, adatbázis
  - nézet, szolgáltatás, üzleti logika, infrastruktúra
- Okos böngésző
  - Ajax, GWT, HTML5, stb.
    - böngészőben van a nézet és a *modell* is
  - még mindig vékony?

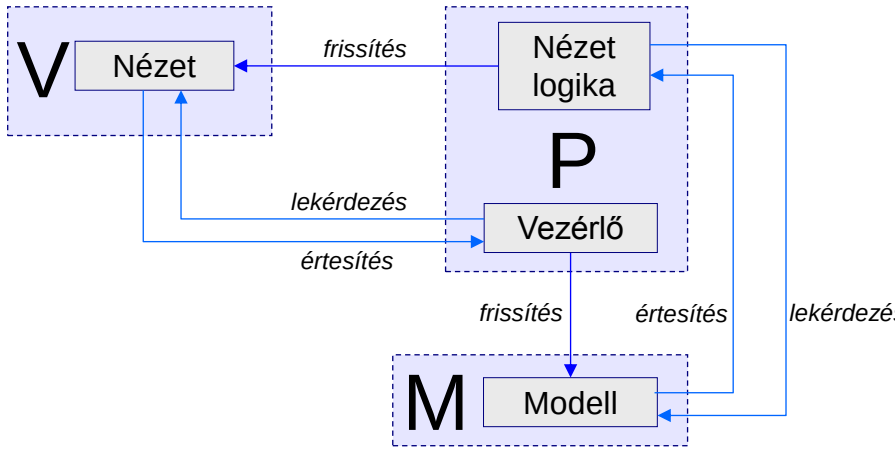
Klasszikus MVC (ismétlés)



Model-View-Controller

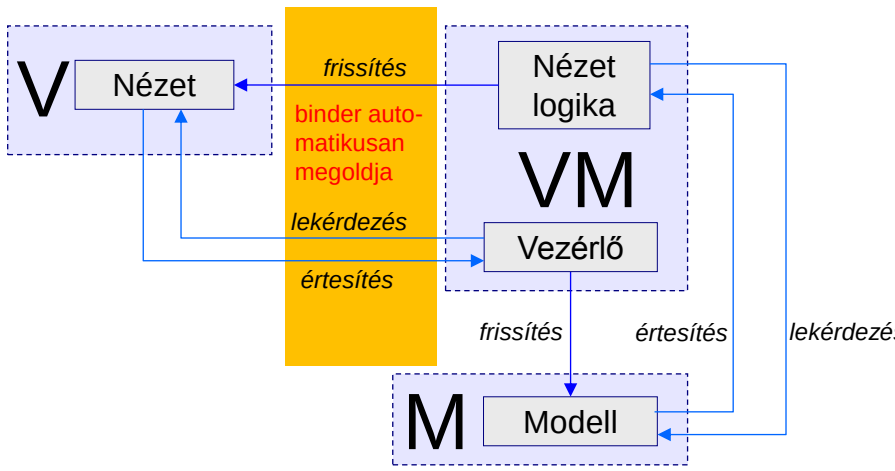


Model-View-Presenter



28

Model-View-ViewModel



29

Alacsony szintű grafika

JComponent és rajzolás

- `JComponent`: a komponens-osztályhierarchia gyökere
  - paint(Graphics g)
    - g-re rajzol, amikor szükséges
  - repaint()
    - értesítés, hogy újra kell rajzolni a komponenst
    - meg kell adni az újrarajzolandó terület téglalapját
- `Graphics`: geometriai primitívek rajzolása
  - vonalak, ívek, szöveg, stb.
- `Graphics2d`: fejlett grafika
  - alakzatok, képek, stb.

`Graphics` osztály

- Geometriai primitívek
  - *drawLine*, *drawRect*, *drawPolygon*, *drawText* etc.
- Képek kezelése
  - *drawImage*, lásd még az *Image* osztály
- Geometriai primitívek kitöltése
  - *fillRect*, *fillOval*, *fillPolygon*, etc.
- Geometriai primitívek kitöltése az alapértelmezett háttérszínnel
  - *clearRect*
- Színek és betűtípusok
  - *setColor*, *setFont*, stb.

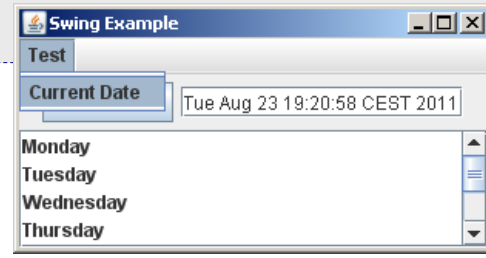
Graphics osztály elérése

- `paint(Graphics g)` metóduson belül
  - a metódus paramétere
  - csak a metóduson belül szabad használni
- `paint` metóduson kívül
  - pl. ha nem a képernyőre szeretnénk rajzolni
  - `JComponent.getGraphics()` metódus
- Rajzolás a memóriában (pufferelés)
  - `BufferedImage` osztály
    - `java.awt.image` csomagban

Speciális komponensek

Menü hozzáadása az ablakhoz

```
ActionListener al = new MyActionListener();
b.addActionListener(al);
JMenuItem mi1 = new JMenuItem("Current Date");
mi1.setActionCommand("date"); // action command beállítása
mi1.addActionListener(al);    // listener hozzáadása
JMenu m1 = new JMenu("Test");
m1.add(mi1);
JMenuBar bar = new JMenuBar();
bar.add(m1);
f.setJMenuBar(bar);
```



Párbeszédablakok használata

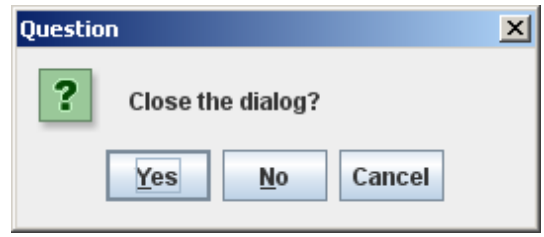
- A párbeszédablakok csak ideiglenesek
- Alapértelmezett Swing dialógusok
  - JFileChooser
    - fájl kiválasztása; OK gomb szövege megváltoztatható, fájl keresési minta beállítható, stb.
  - JColorChooser
    - szín kiválasztása különböző palettamodellek alapján
  - JOptionPane
    - opciók kiválasztása (OK, OK-mégsem, igen-nem-mégsem, stb.)
  - JDialog
    - általános célú dialógusablak: üres, fel kell tölteni tartalommal

Párbeszédablak példa

```
JFileChooser chooser = new JFileChooser();
// szülő ablakot és a kiválasztógomb szövegét definiálni kell
int returnVal = chooser.showDialog(f, "Select");
if(returnVal == JFileChooser.APPROVE_OPTION) {
       System.out.println(chooser.getSelectedFile()
              .getName());
}
```

Párbeszédablak példa

```
JOptionPane opt = new JOptionPane("Close the dialog?",
      JOptionPane.QUESTION_MESSAGE, // típus
      JOptionPane.YES_NO_CANCEL_OPTION); // opciók
// szülő ablak és a dialógusablak címe
JDialog jd = opt.createDialog(f, "Question");
jd.setVisible(true);
System.out.println(opt.getValue().toString());
```



További lehetőségek

További swing lehetőségek

- Kényelmes *tooltip* (felugró szöveg) menedzsment
  - pl. *JComponent.setToolTipText*
- Belső ablakok
  - más ablakon belül, saját ablakkezelővel
- Drag and drop (fogd és vidd)
  - szöveg és más objektumok áthúzása akár más alkalmazásokkal együttműködve
- Look and feel (kinézet)
  - testre szabható egy egyszerű JAR hozzáadásával
- …

Köszönöm a figyelmet!