Budapesti Műszaki és Gazdaságtudományi Egyetem

Java technológia

Java grafikus könyvtárak

Sipos Róbert

siposr@hit.bme.hu

2014. 03. 27.

Bevezetés

- A Java két alapvető lehetőséget biztosít grafikus felhasználói felület építésére
  - AWT (Abstract Window Toolkit)
  - Swing
- AWT
  - Java 1.0 óta, **java.awt.***
  - a Java csak egy szabványos interfészt biztosít (absztrakt), a grafikus elemek az operációs rendszer natív ablakozórendszerének elemei
  - különféle platformokon különféle natív megjelenés, inkompatibilitás
  - szűkebb eszközkészlet (különféle operációs rendszerek nyújtotta grafikus elemek halmazainak a közös részhalmaza, metszete)

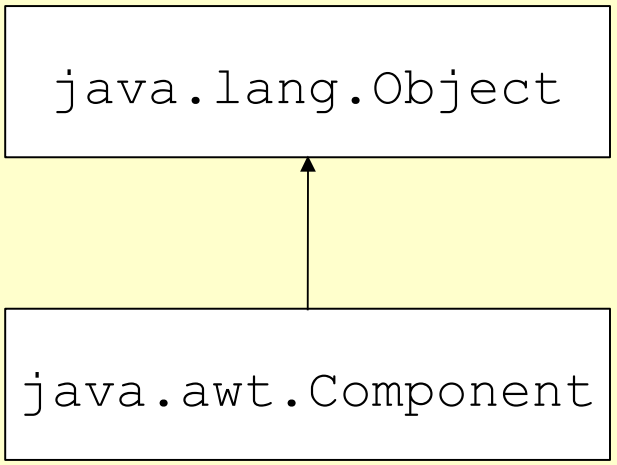
Bevezetés

- Swing
  - minden elem tisztán Javaban megvalósítva
  - platformfüggetlen megjelenés, Look and Feel
  - grafikus építőelemek széles választéka
  - tetszőleges új elem készíthető
  - átgondoltabb tervezés, MVC architektúra megjelenik
  - lassabb programvégrehajtás
  - **javax.swing.***

AWT alapkomponensek és hierarchia

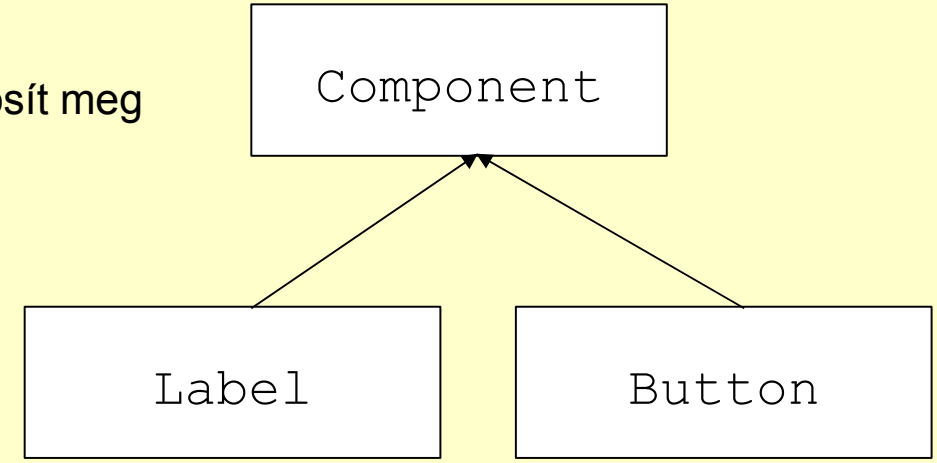
Component

- Minden grafikus elem ősosztálya a **Component**
- Egy AWT elem alapfunkcionalitását valósítja meg
  - Méret (minimum, maximum és preferált)
  - Előtér- és háttérszín
  - Betűtípus, locale
  - Engedélyezettség
  - Láthatóság
  - Általános eseménykezelők regisztrálása
- Absztrakt osztály



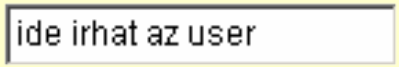
AWT – Komponensek 1

- **Label**
  - egy szöveg megjelenítésére alkalmas
  - csak egysoros szöveg
  - nem módosítható a felhasználó által
  - megadható a szöveg pozíciója (LEFT, RIGHT, CENTER)
- **Button**
  - egy címkézett nyomógombot valósít meg
  - regisztrálható ActionListener



AWT – Komponensek 2

- **TextField**



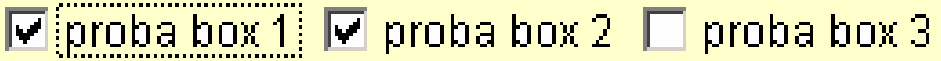
  - szövegbeviteli mező
  - egy sor szöveg bevitelére / szerkesztérére
  - a beirt szöveg elrejtése: setEchoChar
  - szöveg kinyerése: `String getText()`
  - beállítása: `setText(String text)`
  - regisztrálható TextListener, ami a szöveg megváltozásáról értesül

AWT – Komponensek 3

- **TextArea**
  - szövegbeviteli mező
  - több sor szöveg bevitelére / szerkesztérére
  - szöveg megadása `setText`, hozzáfűzés `append`, kinyerés `getText`
  - méret megadása: `setRows`, `setColumns`
  - regisztrálható `TextListener`, ami a szöveg megváltozásáról értesül

AWT – Komponensek 4

- **Checkbox**



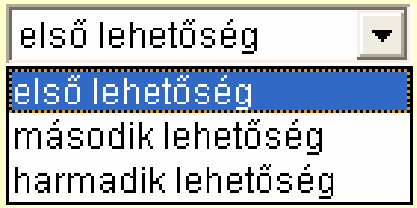
  - kétállapotú kapcsoló
  - a box és a címke egyaránt az objektum része
  - állapotkezelés
    - `boolean getState()`
    - `setState(boolean state)`
  - az állapot figyelhető `ItemListener`-rel is

AWT – Komponensek 5

- **CheckboxGroup**
  - több Checkbox közös kezelésére
  - egyszerre csak egy Checkbox lehet bekapcsolt állapotban (váltáskor az összes többi automatikusan kikapcsol)
  - nem a Checkbox-okat kell a grouphoz adni, hanem a Checkbox létrehozásakor kell a konstruktorban megadni, hogy melyik csoport része

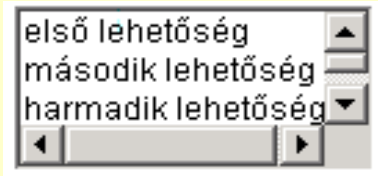
AWT – Komponensek 6

- **Choice**
  - véges sok szöveges elemből való választás
  - kiválasztott elem lekérdezése
    - `getSelectedItem`
    - `getSelectedIndex`
  - szintén figyelhető `ItemListener`-rel



AWT – Komponensek 7

- **List**
  - véges sok szöveges elemből való választás
  - kétféle üzemmód: single / multiple selection
  - kiválasztott elem lekérdezése
    - `Srtring([ ]) getSelectedItem(s)`
    - `int([ ]) getSelectedIndex(es)`
  - figyelhető `ItemListener`-rel (kijelölés megváltozása) vagy
  - `ActionListener`-rel (double-click egy elemre)

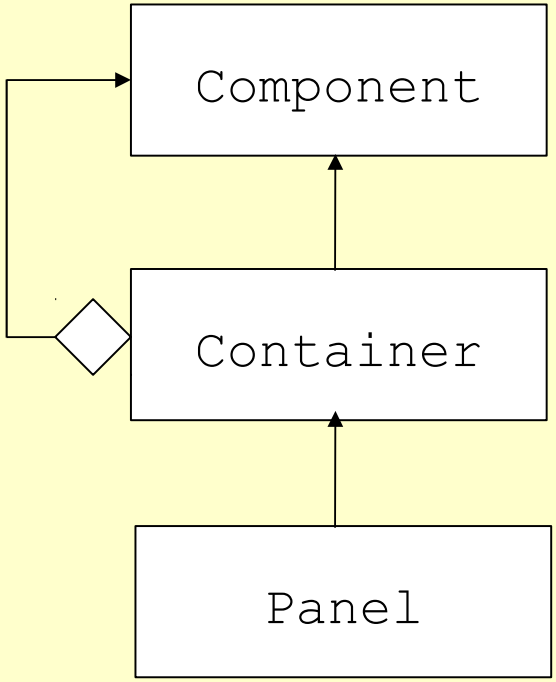


AWT – Komponensek 8

- **Canvas**
  - natív vászon, melyre tetszőleges képet rajzolhatunk
  - a kirajzoláshoz egy leszármazottját kell elkészíteni
  - rajzolás a `public void paint(Graphics g)` metódus felüldefiniálásával
  - az átvett Graphics objektum megfelelő metódusaival rajzolhatunk (lásd: később a Java2D-nél)

AWT – Hierarchia

- A Component leszármazottja a **Container**, ami képes komponensek több komponens összefogására (Homogén összetétel)
  - az **add/remove(Component)** metódussal adható hozzá vagy távolítható el egy Component
  - rendelkezik ún. layouttal, ami azt mondja meg, hogy az általa tartalmazott komponensek milyen elrendezésben kerüljenek megjelenítésre **setLayout(LayoutManager)**
- Konténerek egymásba ágyazhatóak
  → hierarchikus tartalmazási struktúra készíthető (fa)
- A konténer felelőssége a tartalmazott komponensek elhelyezése, méretezése (rekurzió) és nyilvántartása
- A legegyszerűbb konténer a **Panel**



Window

- **Window**
  - a Container leszármazottja
  - az ablakozó rendszer ablak fogalmának absztrakciója
  - az operációs rendszer által biztosított “nyers” komponens
    - nincs kerete
    - nincs fejléce
    - nem lehet menüsora
  - önállóan nem létezhet

Frame

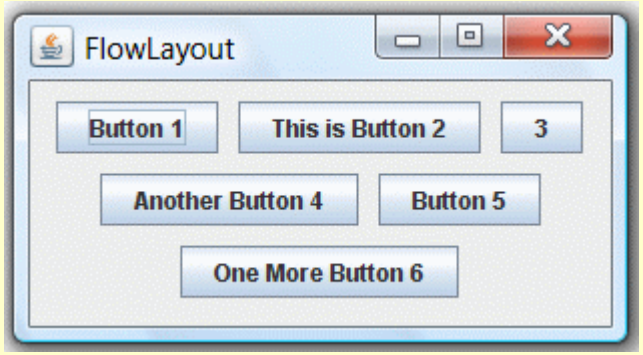
- **Frame**
  - a Window leszármazottja
  - grafikus felülettel rendelkező program alapja
  - egy ablak az operációs rendszer ablakozó rendszerétől
    - kerettel
    - fejléccel
    - megadható neki menüsor is

LayoutManager

- Az alapvető kérdés, hogy hogyan legyenek elhelyezve az egyes komponensek egy konténeren belül? És hogyan kezeljük az átméretezéseket?
- Emiatt tartozik minden Container objektumhoz egy LayoutManager (van default, de megváltoztatható), amely a komponenseit helyezi el a képernyőn a megadott szabályoknak megfelelően (Stratégia)
- A komponensek méretre vonatkozó tulajdonságait módosíthatja rekurzívan ha a hozzá tartozó Container objektum mérete változik (rugalmas felület)
- Különválik a tartalmazás és az elrendezés
- Csak egy interface, a konkrét implementációi konkrét elrendezési irányelveket határoznak meg

FlowLayout

- Sorban, vízszintesen helyezi el egymás mellé az elemeket
- Alapértelmezés szerint balról-jobbra, de megfordítható
- Szükség esetén új sort nyit
- Igazítás állítható (alapértelmezés szerint középre igazít)
- Nem növeli a komponensek méretét

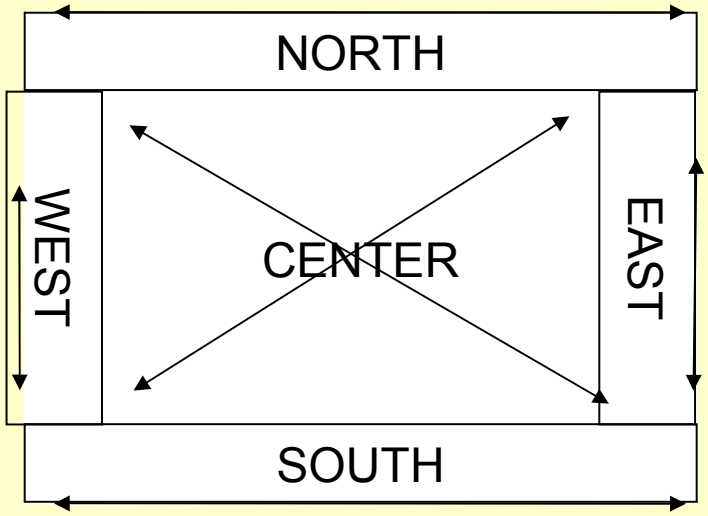


CardLayout

- „Kártyapakli” hasonlat, ahol minden komponens egy lap a pakliban
- Mindig csak a legfelső lap látható
- Sorszám, vagy a komponensekhez rendelt nevek alapján rendezhető

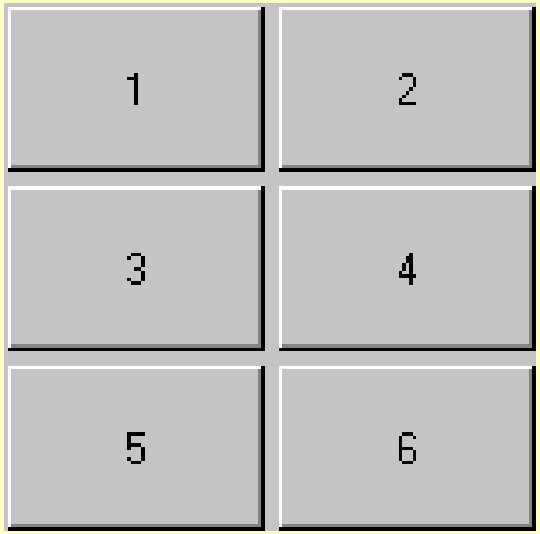
BorderLayout

- 5 területet definiál az alábbi elrendezésben
- Minden mezőbe legfeljebb egy komponens kerülhet
- A Frame alapértelmezett LayoutManager-e
- Az 5 területet 5 konstanssal definiálja, ezzel adható meg, hogy hova kívánjuk elhelyezni a komponenst
  - `BorderLayout.NORTH`
  - `BorderLayout.SOUTH`
  - `BorderLayout.WEST`
  - `BorderLayout.EAST`
  - `BorderLayout.CENTER`



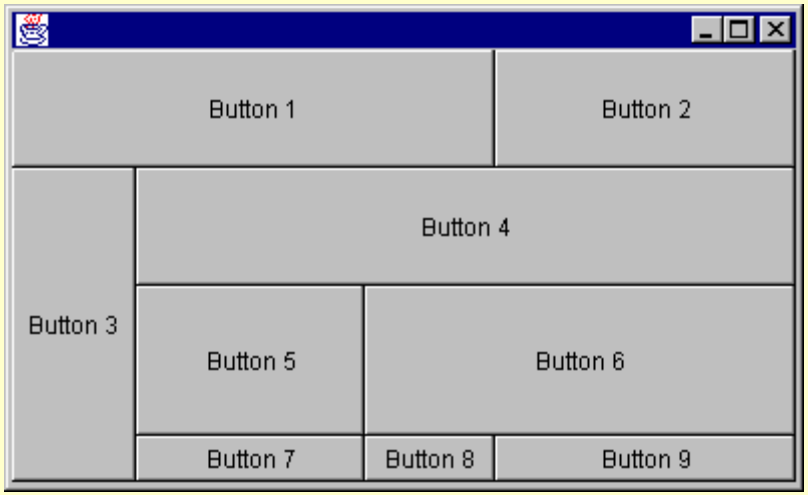
GridLayout

- Azonos cellaméretű mátrixra osztja fel a rendelkezésre álló területet
  → minden komponens egyforma méretű lesz
- Sorok és oszlopok száma megadható
- Sorfolytonosan tölti fel
- Elemek közötti vízszintes és függőleges hely is megadható



GridBagLayout

- A GridLayout bonyolultabb változata
- Egy elem több cellát is elfoglalhat



Példaprogram

AWT – Első példaprogram

```
public class PeldaProgram {
   private Frame peldaFrame;


   public PeldaProgram() {
      peldaFrame = new PeldaFrame();
      peldaFrame.setVisible(true);
   }


   public static void main(String[] args) {
      PeldaProgram p = new PeldaProgram();
   }
}
```

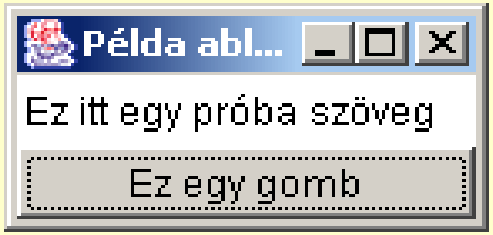
AWT – Első példaprogram 2

```
import java.awt.*;

public class PeldaFrame extends Frame {
    public PeldaFrame() {
        super("Példa ablak");
        Label label =
            new Label("Ez itt egy próba szöveg");
        Button button=new Button("Ez egy gomb");
        add(label, BorderLayout.CENTER);
        add(button, BorderLayout.SOUTH);
        pack(); //a komponensekhez méretezi az ablakot
    }
}
```

AWT – Első példaprogram 3

- eredmény



Eseménykezelés

AWT – Eseménykezelés 1

- a felhasználó cselekedetei eseményeket generálnak (pl: egérmozgatás, billentyűleütés)
- az alkalmazói program értesül ezekről az eseményekről, ez alapján tudja, hogy mi a felhasználó akarata
- a program maga dönti el mely eseményekre kíván reagálni, így nem kell minden eseményt nyomon követnie (pl. egérmozgatás)
- az általunk le nem kezelt eseményekre a rendszer az alapértelmezett módon válaszol

AWT – Eseménykeyelés 2

- az egéresemények közvetlenül az kurzor feletti objektumnak továbbítódnak
- a billentyűleütéseket az aktív ablak dolgozza fel, továbbítja a beviteli fókusszal rendelkező komponensnek
- az események között absztrakciós hierarchia van, így azon a szinten foglalkozhatunk az eseménnyel, amilyen absztrakciós szinten szükségünk van rá (pl. egér gombjának lenyomása+felengedése = egér klikkelés esemény)

AWT – Eseménylezelés 3

- alapvetően kétféle esemény van:
  - közvetlenül a beviteli eszköz által kiváltott esemény
  - magasabb szintű, rendszeresemény (pl. nyomógomb megnyomás, ablak bezárás, menüpont kiválasztás)
- minden eseményt külön objektum ír le
- az eseményosztályok konstruktora nyilvános, ezért a programozó is válthat ki rendszer-eseményeket
- minden esemény ősosztálya a **java.util.EventObject**

AWT – Eseménykezelés 4

- a komponensek a processXEvent metódusokkal dolgozzák fel az eseményeket
- ezek protected láthatóságúak, ezért új komponensosztály definiálásakor felüldefiniálhatóak
- az esemény feldolgozásakor új (magasabb szintű) esemény generálható (pl: nyomógomb felett az egérklikkelés gomb lenyomása eseményt vált ki)

AWT – Eseménykezelés 5

- Minden **XEvent** eseménytípushoz tartozik egy **XListener** interfészt, amelyet az eseménykezelők megvalósítanak
  - pl. MouseEvent → MouseListener, KeyEvent → KeyListener, WindowEvent → WindowListener, ActionEvent → ActionListener, …
  - Az XEvent példánytól elkérhetőek az esemény paraméterei
  - A **java.util.EventListener** interfészből származnak (marker interfész)
- Az egyes eseménykezelő osztályok regisztrálhatóak az adott forrás komponensnél, feliratkozhatnak egy adott típusú eseményre (**Observer** tervezési minta)
  - `public void addXListener(XListener listener)`
  - `public void removeXListener(XListener listener)`
- Egy eseményre több eseménykezelő, és egy eseménykezelő több eseményre is feliratkozhat

AWT – Eseménykezelés 6

```
import java.awt.*;

public class PeldaFrame extends Frame implements    ActionListener
{
    public PeldaFrame() {
        ...
        button.addActionListener(this);
        ...
    }


    public void actionPerformed(ActionEvent e) {
            System.exit(0);
    }

    }
```

AWT – Eseménykezelés 7

```
import java.awt.*;

public class PeldaFrame extends Frame implements WindowListener {
    public PeldaFrame() {
        ...
        addWindowListener(this);
        ...
    }
    public void windowActivated(WindowEvent e) {}
    public void windowClosing(WindowEvent e) { System.exit(0); }
    public void windowDeactivated(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowOpened(WindowEvent e) {}
}
```

AWT – Eseménykezelés 8

- Minden listener XListener interfészhez, amelynek egynél több metódusa van, tartozik egy **XAdapter** adapter osztály
  - pl. WindowListener → WindowAdapter
- Üres törzzsel implementálja az összes eseménykezelő metódust
- Kényelmi funkció, kevesebb és áttekinthetőbb kódot kell írni
- Saját eseménykezelő az adapterből való leszármaztatással

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

public class WindowHandler extends WindowAdapter {
   public void windowClosing(WindowEvent e) {
      System.exit(0);
   }
}
```

AWT – Eseménykezelés 9

- kényelmesebb, ha egy helyen van az ablak és a hozzá tartozó eseménykezelés kódja ⇒ beágyazott osztály

```
public class ProbaFrame extends Frame {
   public ProbaFrame() {
      addWindowListener(new Closer());
   }

   class Closer extends WindowAdapter {
      public void windowClosing(WindowEvent e) {
         System.exit(0);
      }
   }
}
```

AWT – Eseménykezelés 10

- ha csak egy helyen kell az eseményt lekezelni ⇒ anonymous inner class

```
public class ProbaFrame extends Frame {
   public ProbaFrame() {
      addWindowListener(new WindowAdapter() {

         public void windowClosing(WindowEvent e) {
            System.exit(0);
         }

      });
   }
}
```

AWT – Eseménykezelés 11

- több gomb lenyomásának kezelése
  - beágyazott osztályokkal, gombonként külön-külön
    - sok kódot kell írni
    - sok osztály keletkezik
  - egy metódusba összevonva, közös eseménykezelővel
    - szét kell válogatni a beérkező eseményeket attól függően, hogy melyik gombtól jött
    - ez megvalósítható az ActionEvent `String getActionCommand()` vagy a `Component getSource()` metódusával

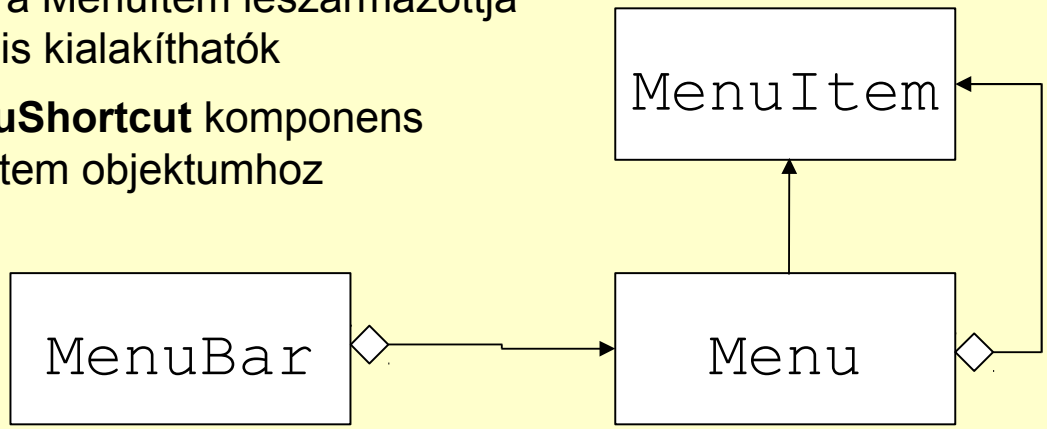
AWT – Eseménykezelés 12

```
public class ProbaFrame extends Frame implements ActionListener {
   ...
   public void actionPerformed(ActionEvent e) {
      if(e.getActionCommand().equals("exit") {
      } else if(e.getActionCommand().equals("open") {
      } else if(e.getActionCommand().equals("save") {
      } else if(e.getActionCommand().equals("new") {
      }
   }
   ...
}
```

Menükezelés

Menükezelés 1

- Menük használata
  - felső menüsort a **MenuBar** reprezentálja
  - ehhez adhatók **Menu** objektumok
  - a Menu objektumban **MenuItem**-ek reprezentálják a menü sorait, és szeparátor is hozzáadható az addSeparator metódussal
  - a Menu objektum maga is a MenuItem leszármazottja
    ⇒ hierarchikus struktúrák is kialakíthatók
  - a gyorsbillentyűket a **MenuShortcut** komponens reprezentálja, ezt a MenuItem objektumhoz lehet adni



Menükezelés 2

- Menük használata
  - MenuShortcut létrehozásánál egy betűt kell megadni, ami (implicite) a CTRL billentyűvel együtt lesz érvényes (lehet kérni a SHIFT-et mint módosítóbillentyűt)
  - készíthető a Checkbox-hoz hasonló, kétállapotú menüpont is a **CheckboxMenuItem** osztály felhasználásával
  - egyszerű menüpontokhoz az ActionListener figyelő regisztrálható, míg CheckboxMenuItem-ekhez az ItemListener

Menükezelés példa

```
MenuBar menuBar = new MenuBar();

Menu menu = new Menu("File",true);

menu.add("New");

menu.add(new MenuItem("Open"));

MenuItem menuItem = new MenuItem("Exit",new
      MenuShortcut(KeyEvent.VK_Q));

menu.add(menuItem);

menuBar.add(menu);

setMenuBar(menuBar);
```

tear-off opció

PopupMenu

- a Menu leszármazottja, tehát mindenhol szerepelhet ahol Menu objektum szerepelhet
- megjelenítése a **show** metódussal, ahol meg kell adni azt a komponenst, ahonnan ered a megjelenítés igénye, továbbá a megjelenítés relatív koordinátáit a megadott komponens bal felső sarkától
  - ezek például egy MouseEvent-től elkérhetőek
- nem hívható meg a show metódus, ha a PopupMenu-t a menühierarchiában használjuk
- csak akkor lesz eredményes a show, ha a menü szülője, és az igényforrás is látszik

Dialógusok

- abban tér el a hagyományos ablaktól (Frame), hogy lehet modális is
  - amíg a képernyőn látszik, más komponens nem kaphat eseményt
- csak akkor tér vissza a show metódusból a vezérlés, ha a dialógus megszűnik látszani
- a java.awt csomag előre elkészített dialógusa a **FileDialog**, ami file-ok kiválasztására szolgál
  - mindig modális
  - lehet SAVE és LOAD módban

Swing koncepció

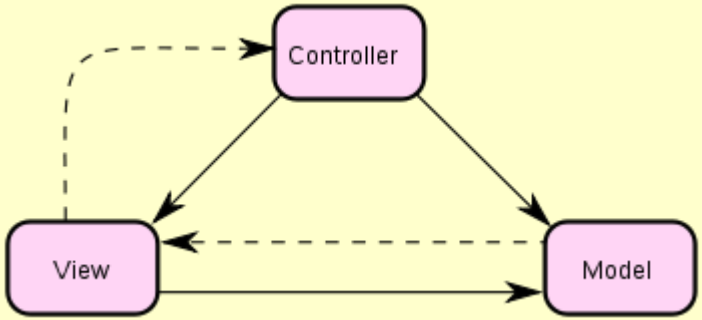
Pehelysúlyú komponensek

- nem tartozik hozzá operációs rendszerbeli peer-objektum
- közvetlenül a Component, a Container vagy már létező pehelysúlyú komponensből kell származnia
- meg kell valósítani a komponens megjelenését és viselkedését is (hogyan reagál az eseményekre)
- tetszőleges alakú (nem feltétlenül téglalap) lehet, lehet részben vagy teljesen átlátszó

Koncepció

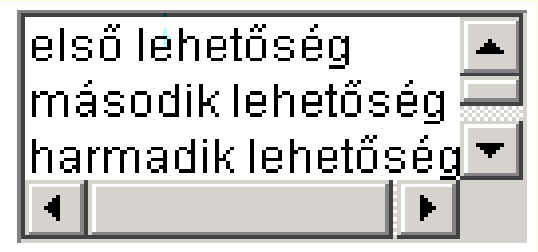
- pehelysúlyú grafikus komponensek gyűjteménye
- AWT eseménykezelő modelljére építenek
- teljesen Java-ban készült, ezért a megjelenés teljesen független a platformtól
- komplex grafikus elemek
- önállóan használható (használandó) ezért tartalmazza az AWT összes nehézsúlyú komponensének pehelysúlyú megvalósítását
  - az AWT nehézsúlyú komponensek pehelysúlyú megfelelőinek neve megegyezik az AWT komponens nevével, csak egy J betű került elé (pl. JTextField)

Model – View – Controller



- A Swingben az MVC architektúrát alkalmazzák
- A komponensek három részre tagolódnak
  - adatmodell: tárolja és kezeli az adatokat
  - megjelenítés: az adatmodell által szolgáltatott adatokat jeleníti meg, és a felhasználói eseményeket továbbítja a vezérlésnek (GUI)
  - vezérlés: a felhasználói események alapján manipulálja az adatmodellt (eseménykezelők)
- Az egyszerű komponensek esetén nincsen külön adatmodell
  (pl. `JButton, JLabel`)
- Összetett komponensek esetén elérhető default adatmodell implementáció
  - pl. `JList` → `ListModel` interfész → `DefaultListModel`

JList



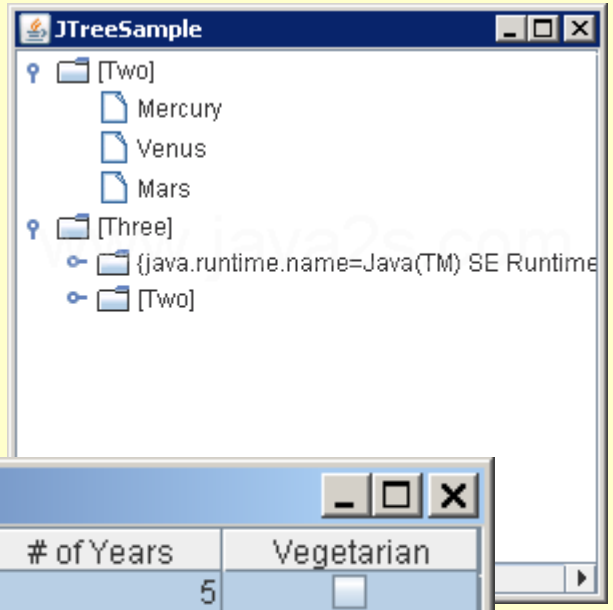
- Az AWT-s List Swinges párja
- Az elemek külön vannak a **ListModel**-ben tárolva
  - String[] helyett a **Object[]**

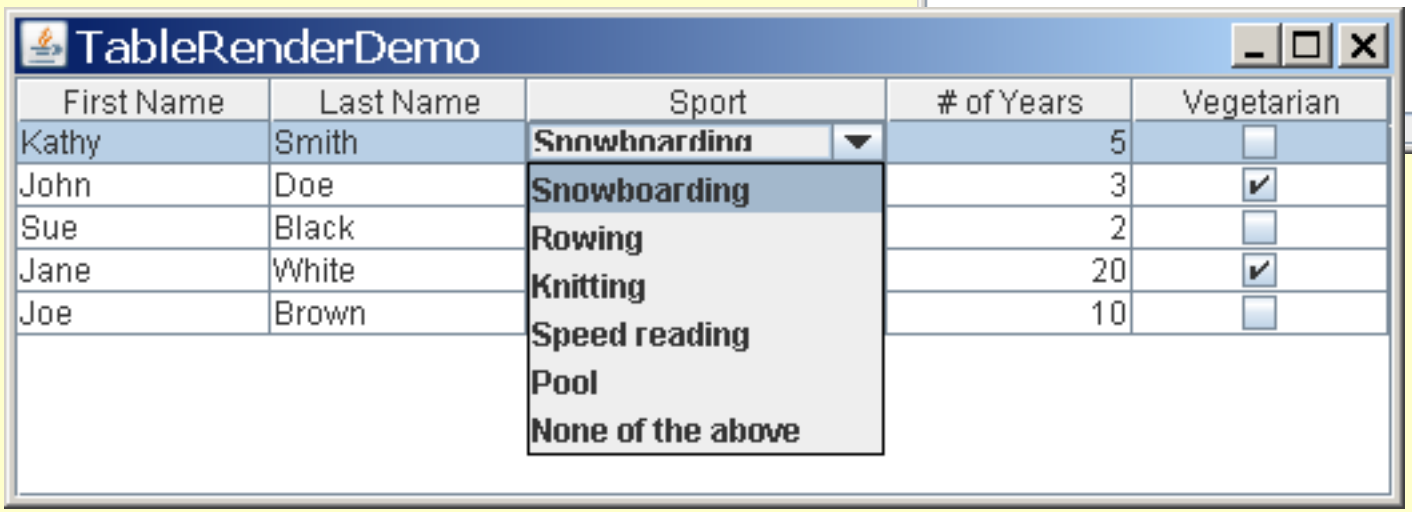
```
DefaultListModel model = new DefaultListModel();
model.addElement(”első lehetőség”);
model.addElement(”második lehetőség”);
JList list = new JList(model);
```

- Alapértelmezetten megjelenítésnél az objektumok **toString()** metódusát hívja
  - A **ListCellRenderer** interfészen keresztül felüldefiniálható
  - **DefaultListCellRenderer** default implementáció

További összetett vezérlők

- **JComboBox**
- **JTable**
- **JTree**
- **...**

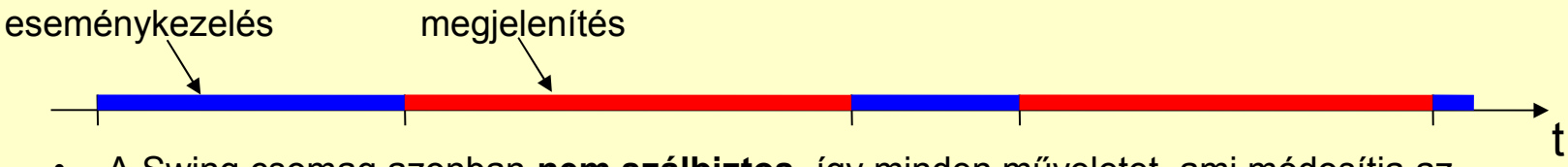




Swing szálkezelés

Swing szálkezelés 1

- az MVC megvalósítás igényli a kirajzolás és az adatmodell megváltoztatásának szinkronizációját, ugyanis, ha a megjelenítés közben változik az adatmodell az inkonzisztens képet eredményezhet
- a Swing egy szálon végzi (**AWT Event Dispatch Thread**) a komponensek megjelenítését, és az események kezelését, ezáltal automatikusan megoldódik a fenti szinkronizációs probléma



- A Swing csomag azonban **nem szálbiztos**, így minden műveletet, ami módosítja az adatmodellt vagy a megjelenítést az AWT EDT-ben kell végrehajtani
- A szálbiztos (több szálú) GUI deadlockot és race conditiont eredményezne

Swing szálkezelés 2

- Már a GUI felépítése is problémát okoz, ha más szálból történik
- Az adatmodellt is csak az eseménykezelő szálból szabad módosítani
  - pl. esetleges kivételek miatt
- Viszont ennek hátulütője, hogy egy eseménykezelő metódus, ha hosszú ideig tartó műveletsorozatot végez, akkor blokkolja a program grafikus felületét
- Ezért érdemes a hosszú műveleteket külön szálon végrehajtani, és az eseménykezelő metódusból azonnal visszatérni
  - Hogyan maradjon szálbiztos?

Swing szálkezelés 3

Mindezeknek a megoldására szolgálnak a **SwingUtilities** osztály alábbi statikus metódusai:

- **`invokeLater`**: a paraméterként átadott Runnable objektum run metódusának (Thread) végrehajtását úgy ütemezi, hogy az az AWT EDT-ben történjen meg, a metódus egyből visszatér
- **`invokeAndWait`**: ugyanazt teszi mint az előző, azzal a kivétellel, hogy megvárja a programrész lefutását, és csak azután tér vissza (AWT-EDT-ben nem szabad meghívni)
- **`isEventDispatchThread`**: true-val tér vissza, ha az aktuális szál az AWT Event Dispatch Thread

Swing szálkezelés (1. példa)

- egy folyamatirányító rendszerben a felhasználói felületen kezdeményezett akció egy hosszú műveletsort eredményez
- a folyamat végéről értesíteni akarjuk a felhasználót, de el akarjuk kerülni a program blokkolását a végrehajtás idejére, mert azt a felhasználó lefagyásnak is tekintheti, illetve azért, mert a végrehajtás ideje alatt egyéb feladatokat is el tud végezni

Swing szálkezelés (1. példa)

- az imént említett szituációkra példa az alábbi pszeudo-program:

```
public void actionPerformed(ActionEvent e) {
   HosszuMuvelet hosszuMuvelet = new HosszuMuvelet();
   hosszuMuvelet.start();
}

public class HosszuMuvelet extends Thread {
   public void run() {
      ...
      invokeLater(new LámpaKigyujto());
   }
}
```

Swing szálkezelés (2. példa)

- a felhasználói esemény bekövetkeztére ismét egy hosszú folyamat indul el, és nem akarjuk a programot blokkolni
- a folyamatnak több fázisa van, melyek végének eléréséről értesíteni kell a felhasználót, és csak akkor szabad továbbhaladni, ha biztosak lehetünk abban, hogy a felhasználó az értesítést megkapta
- például a folyamat attól függően, hogy éppen melyik fázisban van vagy áramot enged egy kábelbe vagy kikapcsolja azt, és a kikapcsolt fázisban szeretnénk a kábelhez érni

Swing szálkezelés (2. példa)

```
public void actionPerformed(ActionEvent e) {
   HosszuMuvelet hosszuMuvelet = new HosszuMuvelet();
   hosszuMuvelet.start();
}

public class HosszuMuvelet extends Thread {
   public void run() {
      for(int i=0; i<5; i++) { // 5 fazis
         ... // egyéb műveletek
         invokeAndWait(new ErtesitoModalisDialogus());
      }
   }
}
```

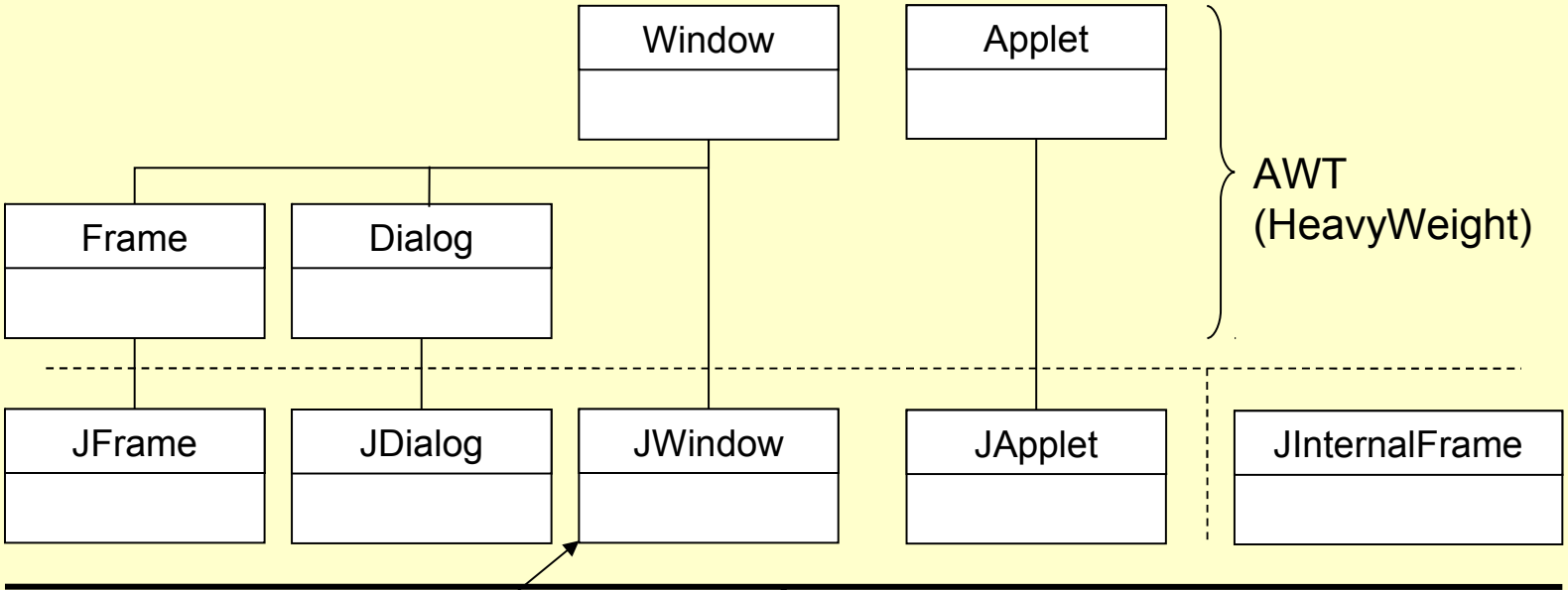
Biztosan csak akkor megyünk tovább,
ha a felhasználó jóváhagyta

Swing szálkezelés 4

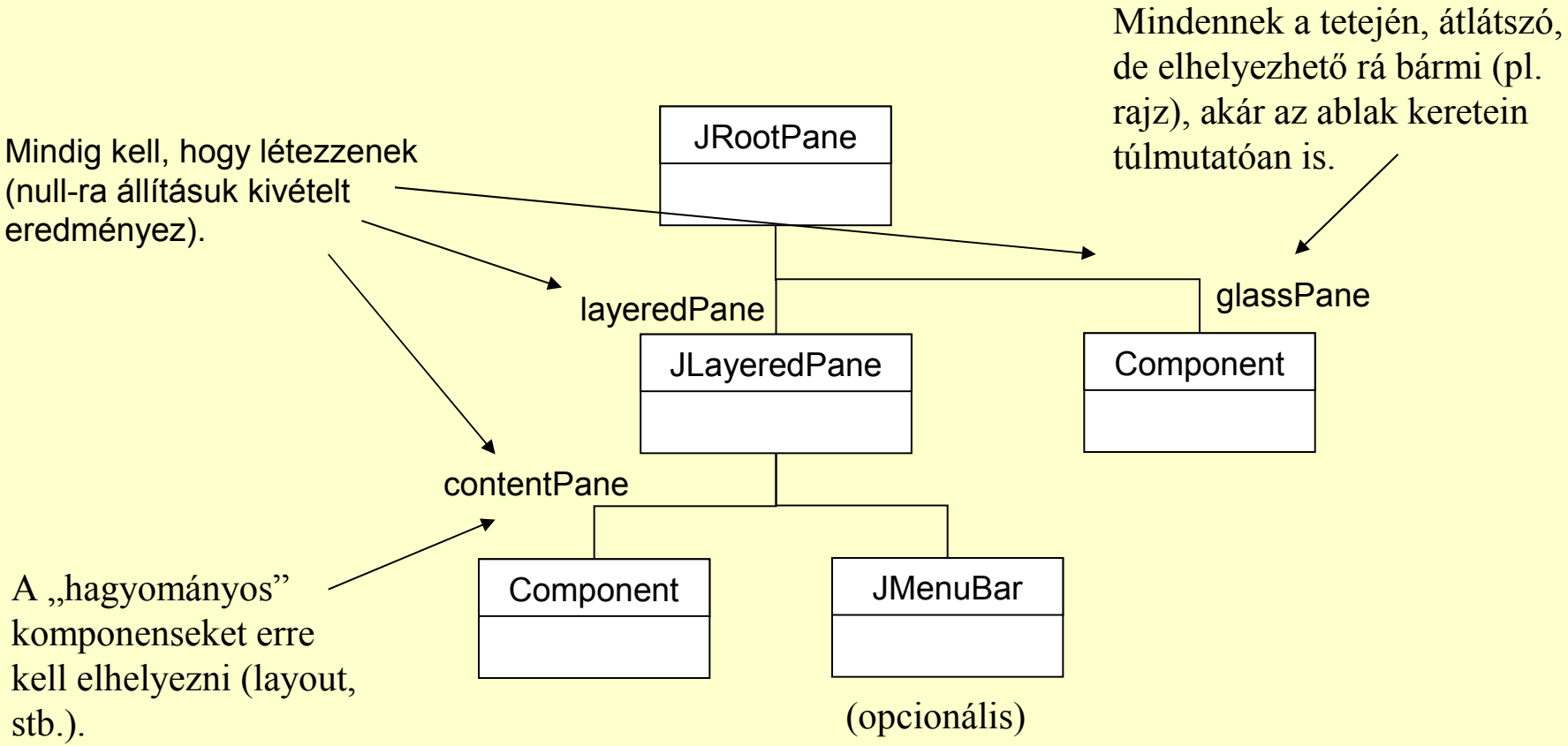
- az egyes komponensek újrarajzolását kérő repaint és revalidate metódusokat bármely programszálból meg lehet hívni, mivel az újrarajzolást úgyis a RepaintManager végzi, ami pedig ügyel arra, hogy az újrafestés csakis az AWT Event Dispatch Thread-ben történjék meg
- ismétlődő vagy késleltetett végrehajtást igénylő és az adatmodellt módosító feladatok esetén lehet használni a javax.swing csomag **Timer** osztályát (nem a java.util Timer-jét)
- a Timer-be lehet regisztrálni ActionListener-eket, és a Timer garantálja, hogy a listener-ek actionPerformed metódusa az AWT-EDT-ben kerül végrehajtásra
- pl. villogó lámpa a grafikus felületen

Swing hierarchia

Swing – Hierarchia 1



Swing – Hierarchia 2



JFrame 1

- a Swing-ben alkalmazott hierarchia miatt a JFrame nem teljesen kompatibilis a Frame-mel, ugyanis nem lehet komponenst közvetlenül a JFrame-hez adni, csak a ContentPane-jéhez, ellenkező esetben futásidejű hiba keletkezik

```
Container contentPane = getContentPane();

contentPane.add(new JLabel("a helyes út"));

contentPane.add(new JButton("OK"));
```

JFrame 2

- a JFrame-ben definiálva van alapértelmezett művelet az ablak bezárására:
  - DISPOSE_ON_CLOSE
  - DO_NOTHING_ON_CLOSE
  - HIDE_ON_CLOSE
  - EXIT_ON_CLOSE
- ezek a konstansok a WindowConstants interface-ben találhatóak (!)
- az alapértelmezett érték a HIDE_ON_CLOSE, ezért megtévesztő lehet (nem áll le a program végrehajtása, csak eltűnik az ablak)

Különbségek az AWT-hez képest 1

- a szövegkomponensek nem rendelkeznek automatikusan ScrollBar-ral, és kerettel sem, ezeket kívánság szerint a programozó rendelheti az objektumhoz
- ScrollBar használatához a szövegkomponenst át kell adni egy JScrollPane objektumnak
- megadhatók a ScrollBar-ok megjelenítési szempontjai

```
JTextArea textArea = new JTextArea("gördíthető",10,2);
JScrollPane scrollPane = new JScrollPane(textArea);
add(scrollPane);
```

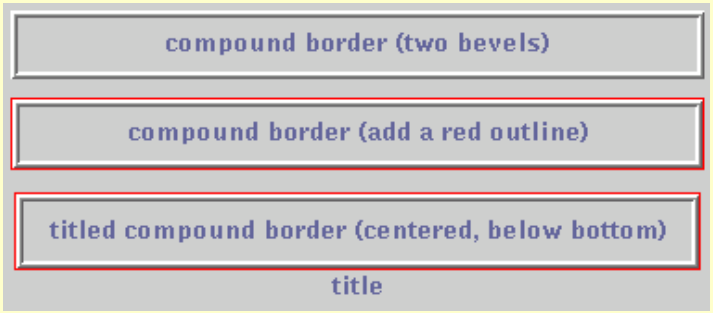
Különbségek az AWT-hez képest 2

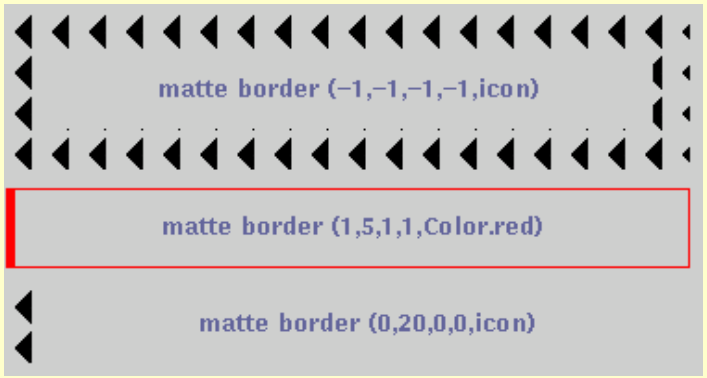
- egy komponenshez úgy rendelhető keret, hogy a setBorder metódusának átadunk egy keretobjektumot
- keretet létrehozhatunk a javax.swing.border package osztályainak felhasználásával
- hatékonyabb erőforrásfelhasználást érünk el, ha dedikált keretpéldányok helyett a javax.swing csomagban található **BorderFactory** osztály különféle statikus metódusaival gyártott kereteket adjuk át az objektumoknak, mivel a BorderFactory ahol lehetséges egy keret-példányt több objektum között oszt meg, így redukálódik a memóriafelhasználás

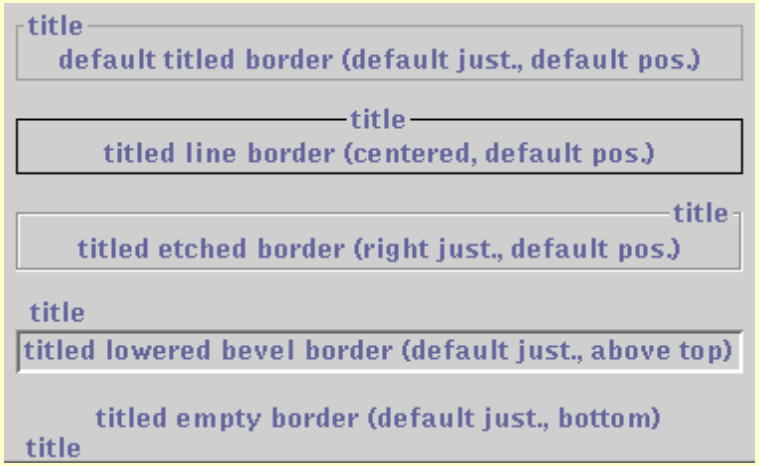
Különbségek az AWT-hez képest 3

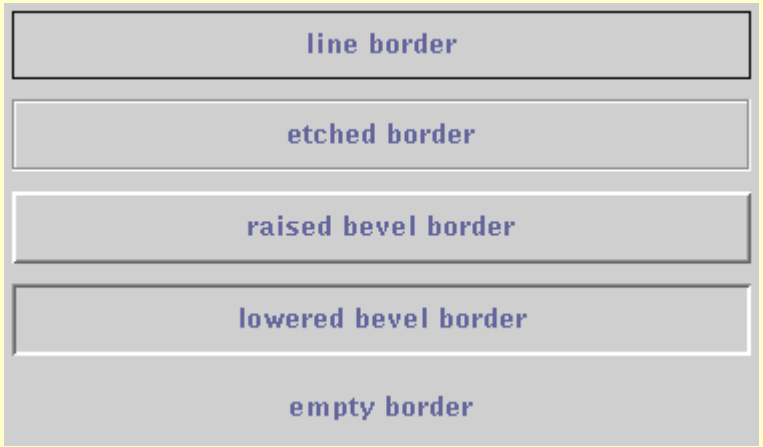
- a Swing-ben az alábbi kerettípusok használhatók
  - BevelBorder (RaisedBevelBorder / LoweredBevelBorder): mélyített, vagy emelt
  - EmptyBorder: üres perem megadásához
  - EtchedBorder: csak egy mélyített, vagy emelt horony
  - LineBorder: azonos vastagságú egyszínű keret
  - MatteBorder: oldalanként megadható vastagságú, lehet kép is
  - TitledBorder: egy EtchedBorder felirattal
  - CompoundBorder: keretek egymásba ágyazásához

Különbségek az AWT-hez képest 4









Look and Feel

Look and Feel

- az MVC modell alkalmazásának további előnye, hogy minden komponens megjelenése külön állítható (setUI / updateUI metódus) és akár futásidőben is változtatható, függetlenül a programlogikától
- egy egységes grafikus stílust nevezünk a Look And Feel-nek (javax.swing.plaf csomagban (Pluggable LAF))
- a Java alapértelmezetten három LAF-ot definiál:
  - MetalLookAndFeel
  - MotifLookAndFeel
  - WindowsLookAndFeel
- a LAF változtatása az **UIManager** metódusaival, valamint property-k segítségével lehetséges

Look and Feel példa

```
try {
    UIManager.setLookAndFeel
("javax.swing.plaf.metal.MetalLookAndFeel");
    UIManager.setLookAndFeel
     ("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
    UIManager.setLookAndFeel
     ("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
  UIManager.setLookAndFeel
     (UIManager.getSystemLookAndFeelClassName());
} catch (Exception e) {
  }
```

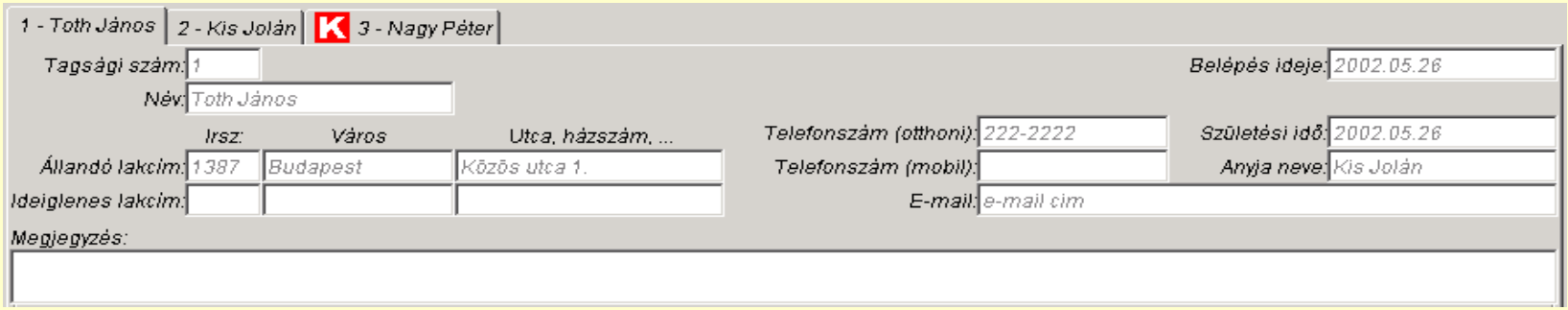
Komplex komponensek

Swing – Komplex komponensek 1

- **JSplitPane**: kettéosztható panel

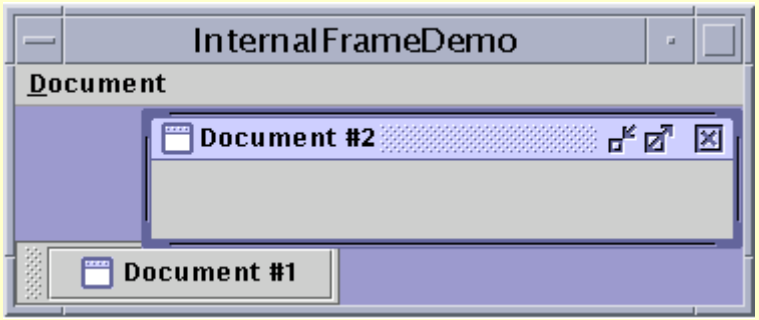
Swing – Komplex komponensek 2

- **JTabbedPane**: többboldalas panel



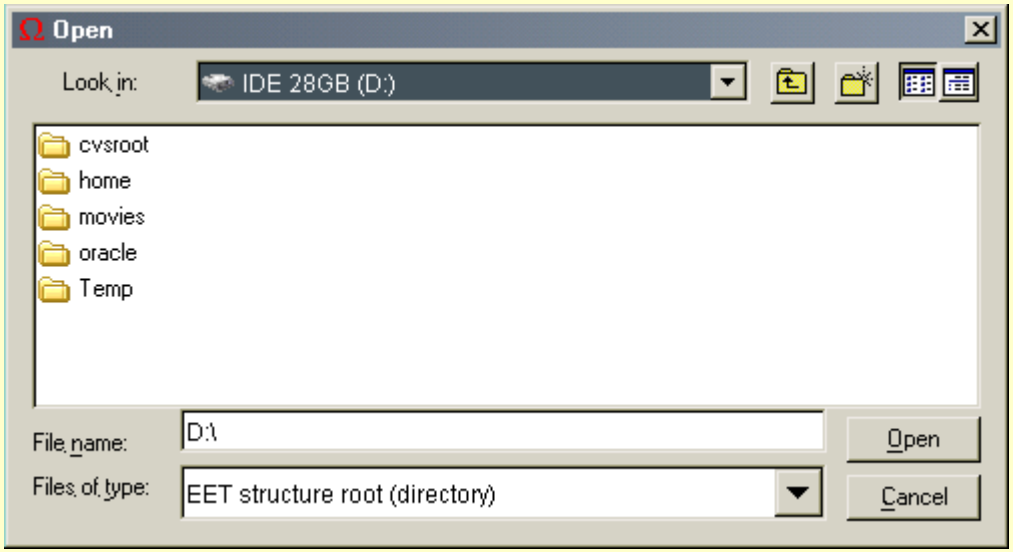
Swing – Komplex Komponensek 3

- **JToolBar**: floating toolbar megvalósítás
- **JDesktopPane**: belső ablakok kényelmes használatához
  - bármely komponens tartalmazhat JInternalFrame-et mivel az a JComponent leszármazottja, de a JDesktopPane bizosítja a hagyományos InternalFrame-ek megszokott működését



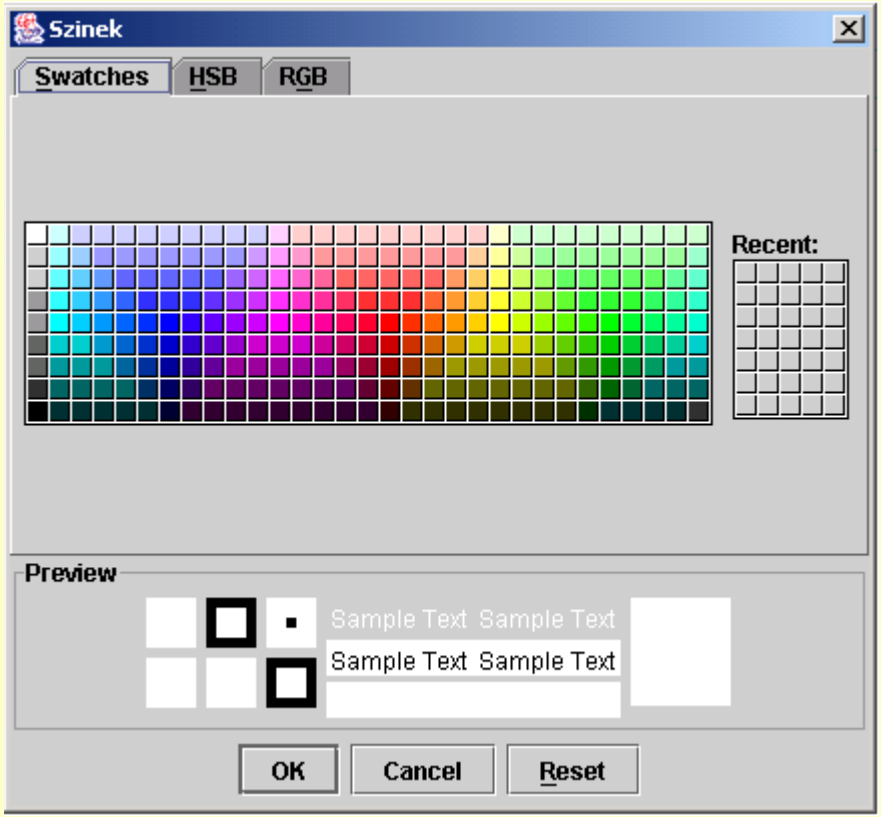
Swing – Komplex Komponensek 4

- Előre megvalósított konfigurálható típus-dialógusok:
  - **JFileChooser**: felhasználó általi file-kiválasztásra



Swing – Komplex Komponensek 5

- **JColorChooser**
  színkiválasztásra



Swing – Komplex Komponensek 6

- **JOptionPane**
  - a gyakran használt dialógusok egyszerű, általános megvalósítása
    - hibaüzenet
    - figyelmeztetés
    - megerősítés
    - egyszerű szöveges input
    - néhány opció közül választani (az opciókat gombok jelenítik meg)

Swing – Komplex Komponensek 7

```
JOptionPane.showConfirmDialog(this,"Biztosan törölni
akarja?","Kérdés",YES_NO_OPTION)
```