A programozás alapjai 3.

Java GUI alapok

Goldschmidt Balázs
balage@iit.bme.hu

1

GUI alapok

- Főként ablakos alkalmazások
- Általában grafikus komponensekre (widget) épül
  - kisebb grafikus elemek (gomb, gördítő sáv, stb.)
  - ablakozó keretrendszeren alapul
- Számos keretrendszer elérhető
  - AWT, SWING, SWT, stb.
- Logikát definiálni kell
- GUI tervezőeszközök segítenek

2

Abstract Windowing Toolkit

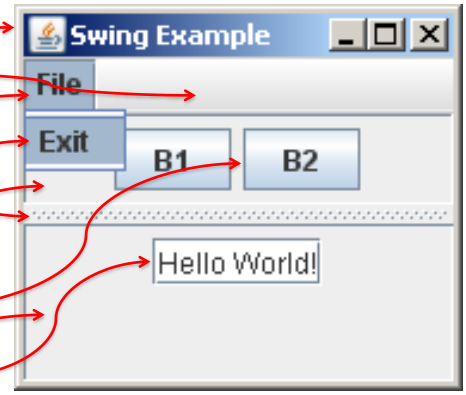
- Nehézsúlyú komponensek
  - az operációs rendszer képességeire épül
- Abstract facade
  - egységes programozói interfész, de minden platformon az OS-nek megfelelő kinézet
- Különböző implementációk
  - inkompatibilitási problémák
- Néhány képességét ma is használjuk
  - események, elemek elrendezése, stb.
- `java.awt.*`

SWING

- Gazdag GUI elemkészlet
  - könnyűsúlyú komponensek
  - Java-ban implementálva
- Erősen a model-view-controller mintára épül
  - összetett widgeteknek külön modell-osztályai vannak
  - megjelenítés külön van implementálva
- A kinézet konfigurálható
  - új osztályokkal, konfigurációs fájlokkal és konfigurációs paraméterekkel
- `javax.swing.*`

Ablakok architektúrája

- JFrame
  - JMenuBar
    - JMenu
      - JMenuItem
  - JSplitPane
    - JPanel
      - JButton
      - JButton
    - JPanel
      - JTextField



5

Ablakok életciklusa

- Ablak felépítése
  - JFrame vagy más ablak létrehozása
  - konténerek és komponensek rápakolása
- Eseménykezelők regisztrálása
  - felhasználói események kezelése külön objektumokban
- Láthatóvá tétel
  - az ablak megjelenik és használható
- *Használat*
- Bezárás
  - erőforrások felszabadítása

6

Komponensek és konténerek

Komponensek és konténerek

- Egyszerű komponensek
  - JButton
  - JTextField
  - JPanel
  - JFrame
- Összetett komponensek
  - MVC
  - JList + JScrollPane

Komponensek tulajdonságai

- Megjelenítés automatikus
- Közös funkcionalitás
  - méret (minimum, maximum, kívánt)
  - láthatóság
  - engedélyezettség
  - események kezelése
- Konténerbe rakható
- Konténerek is komponensek
  - komponens-hierarchia

Egyszerű komponensek: JButton

- Klasszikus nyomógomb
  - kattintható
  - *eseménykezelés később…*
- Szöveg és kép beállítható
- Méret automatikusan számítódik
  - növekszik ill. összemegy, ahogy a konténer szeretné
- Fontosabb függvények:
  - `setEnabled(boolean)`
  - `get/setText()`

Egyszerű komponensek: JTextField

- Klasszikus szövegbeviteli mező
  - szöveg begépelhető
- Kezdeti szöveg ill. méret beállítható
- Méret automatikusan számítódik
  - növekszik ill. összemegy, ahogy a konténer szeretné
- Fontosabb függvények:
  - `setEditable(boolean)`
  - `get/setText(String)`
  - `setCaretPosition(int)`
  - `setSelectionStart/End(int)`

Egyszerű komponensek: JPanel

- Alapvető konténer
  - komponensek tehetők bele
- Komponensek elrendezéséért (layout) felelős
- Méret automatikusan számítódik
  - növekszik ill. összemegy, ahogy a szülő konténer szeretné
- Fontosabb függvények:
  - `add(Component[, param])`
  - `setLayout(LayoutManager)`
  - `getComponentAt(int,int)`

Egyszerű ablak: JFrame

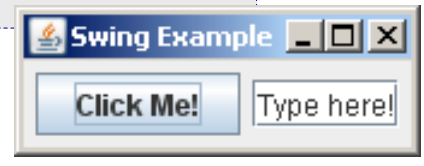
- Ablak kerettel és címsorral
- Minden ablakozó művelet támogatott
  - néhányat explicit engedélyezni kell
- Méret automatikusan számítódik
  - a tartalom mérete alapján
- Fontosabb függvények:
  - `add(Component, int where)`
  - `pack()`
  - `setVisible(boolean)`
  - `setDefaultCloseOperation(`*`JFrame.EXIT_ON_CLOSE`*`)`

Példa: GUI felépítése

```
JFrame f = new JFrame("Swing Example");
JPanel p = new JPanel();
JButton b = new JButton("Click Me!");
JTextField t = new JTextField("Type here!");

p.add(b);
p.add(t);
f.add(p, BorderLayout.NORTH);

f.pack();
f.setDefaultCloseOperation(f.EXIT_ON_CLOSE);
f.setVisible(true);
```



Eseménykezelés

15

Eseménykezelés alapjai

- Listener (observer) minta
  - két szereplő: *listener* (megfigyelő) és *subject* (megfigyelt)
    - subject megkapja az eseményt ...
    - ... és továbbítja az összes regisztrált listener felé
  - egyetlen *eseménykezelő szál* dolgozza fel az összes eseményt
- Sokfajta eseménykezelő típus van
  - `java.util.EventObject`
  - `java.awt.AWTEvent`
  - `java.awt.event.MouseEvent`
  - `java.awt.event.WindowEvent`
- Események és listener-ek szétválaszthatók
  - dedikált felelősség

16

Eseménykezelő interfészek

- *XEvent* → *X*`Listener`
  - interfész (implementálni kell)
  - implementációt regisztrálni kell a komponensnél
    - `add`*X*`Listener(`*X*`Listener el)`
  - *XEvent* eseményt kezelni kell az implementációban
    - `MouseEvent, KeyEvent, AdjustmentEvent, FocusEvent` stb.
  - implementáció: általában anonim belső osztály
    - kényelmesnek tűnik, de nehezen karbantartható

Eseménykezelő adapterek

- *X*`Adapter`
  - *X*`Listener` alapértelmezett implementációja
    - üres metódusokkal
  - kényelmi osztály
    - akkor hasznos, ha az eseményeknek csak egy részét kell kezelnünk
  - tisztább kódot eredményez
  - csak egyszeres öröklődés van Javában!

Eseménykezelési példa

- Módosítsuk az előző példát:
  - gombnyomásra az aktuális idő íródjon be a szövegmezőbe
- `ActionListener` implementáció kell
  - új osztály, fontos a láthatóság
  - ***`actionPerformed(ActionEvent ae)`*** metódus
- Nyomógomb azonosítása
  - ActionEvent ***`getSource`*** metódusa → Component
  - ActionEvent ***`getActionCommand`*** metódusa → String
    - gomb neve vagy a ***`setActionCommand(String c)`*** által beállított érték

Eseménykezelési példa

```
class MyActionListener implements ActionListener {
  JTextField t;
  public MyActionListener(JTextField tt) { t = tt;}
  public void actionPerformed(ActionEvent ae) {
    if (ae.getActionCommand().equals("date")) {
      t.setText((new Date()).toString());
    }
 }
}
```

```
...
JButton b = new JButton("Click Me!");
b.setActionCommand("date");
ActionListener al = new MyActionListener(t);
b.addActionListener(al);
...
```

Komponensek elrendezése (layout)

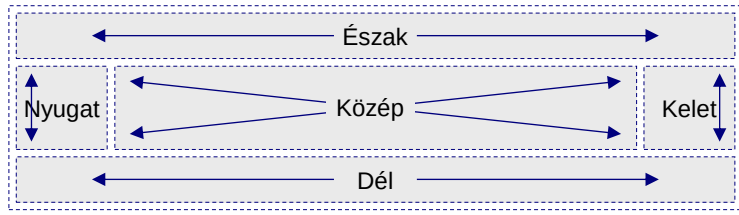
Hova kerülnek a komponensek?

- Problémák
  - pozícionálás
    - *ez az oldal 800x600-as felbontásra van optimalizálva*
  - mi történik átméretezéskor?
- Megoldás: Layout menedzserek
  - minden konténernek van egy alapértelmezett layout menedzsere
  - layout menedzserek megváltoztathatók (*setLayout(LayoutManager m)*)
  - komponensek elhelyezéséért felelősek
  - rekurzív számítás
  - átméretezéskor újraszámítás

Layout menedzserek

- `Container`
  - `void setLayout(LayoutManager mgr)`
  - `LayoutManager getLayout()`
  - `void validate()`
    - újraszámítja az elrendezést és elhelyezi a komponenseket (rekurzívan)
  - `Component add(Component comp [,int index])`
  - `void add(Component c, Object constraint, int index)`
    - hozzáadja a komponenst a konténerhez
    - opcionális paraméter: speciális elhelyezési kérés
- `LayoutManager`
  - hosszú történet...

BorderLayout



- Öt mező
  - észak (north), dél (south), nyugat (west), kelet (east), közép (center)
- `JFrame` alapértelmezett elrendezése
- Átméretezési lehetőség a nyilak irányába
- Egy mező – egy komponens

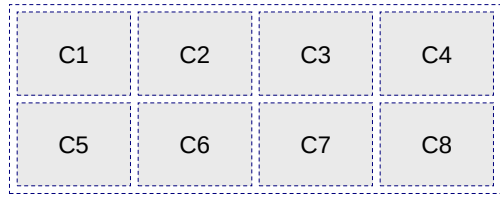
FlowLayout

- `JPanel` alapértelmezett elrendezése
- Egymás mellé teszi a komponenseket
  - ha nincs több hely, új sort kezd
- Nem méretezi át őket egyenként
- Irány függ a konténertől
  - **`ComponentOrientation.LEFT_TO_RIGHT, RIGHT_TO_LEFT`**
- Lehetséges igazítások:
  - **`LEFT, RIGHT, CENTER, LEADING, TRAILING`**

CardLayout

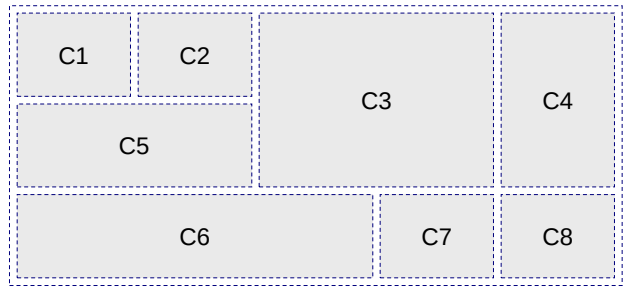
- Komponensek egymásra rakhatók
  - mint egy kártyapakli
  - egy komponens – egy kártya
- Mindig csak a legfelső látszik
- Pakli átrendezhető a menedzser metódusaival
- Komponensek elnevezhetők a gyorsabb eléréshez

GridLayout



- NxM-es rács
  - egy komponens – egy cella
- Minden komponens ugyanolyan méretű
  - szükség esetén átméretezi őket
- Irány függ a konténertől
  - **`LEFT_TO_RIGHT – RIGHT_TO_LEFT`**
- Ha a sorok száma fix, új oszlopok adhatók hozzá

GridBagLayout

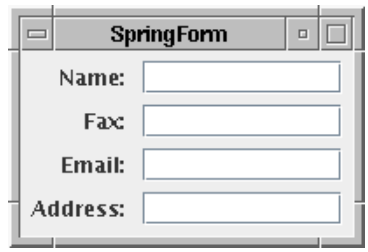


- *GridLayout* fejlettebb változata
- Egy komponensek több cellát is elfoglalhat
- *GridBagConstraint* adja meg a foglaltsági szabályokat

BoxLayout

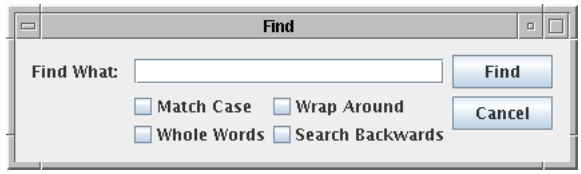
- Komponensek vízszintes vagy függőleges elrendezése
- Nem kezd új sort, ha megtelik
- Négy irány
  - **`X_AXIS`** – **`Y_AXIS`**: vízszintes vagy függőleges
  - **`LINE_AXIS`** – **`PAGE_AXIS`**: figyelembe veszi a **`ComponentOrientation`**-t is

SpringLayout



- Rugalmas, táblázat-szerű elrendezés
- Lényegében a komponensek szélei közötti kapcsolatot kell definiálni
- Csak GUI tervezőeszközök számára
  - nehéz kézzel programozni

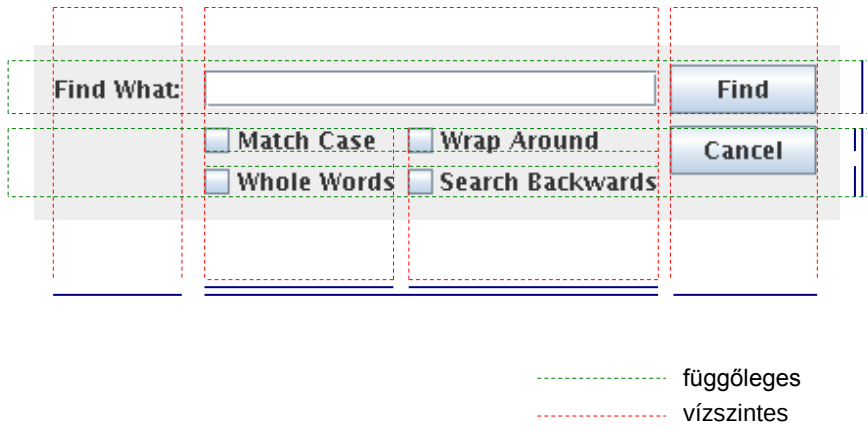
GroupLayout



- Vízszintes és függőleges dimenziók független megadása
  - minden komponenst kétszer kell hozzáadni
- Hierarchikus: csoportok egymásba ágyazhatók
- Soros ill. párhuzamos elhelyezés
- Komponens lecserélése:
  - `void replace(Component oldc, Component newc)`

31

GroupLayout (swing) 2



32

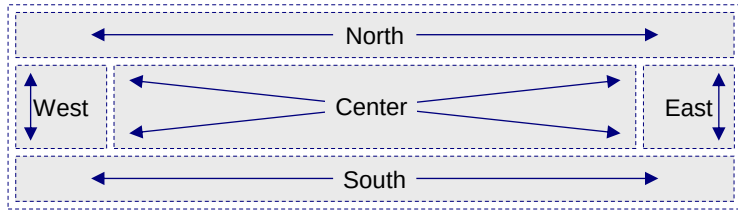
GroupLayout (swing) 3

```
layout.setHorizontalGroup(layout.createSequentialGroup()
  .addComponent(label)
  .addGroup(layout.createParallelGroup(LEADING)
     .addComponent(textField)
     .addGroup(layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(LEADING)
           .addComponent(caseCheckBox)
           .addComponent(wholeCheckBox))
        .addGroup(layout.createParallelGroup(LEADING)
           .addComponent(wrapCheckBox)
           .addComponent(backCheckBox))))
  .addGroup(layout.createParallelGroup(LEADING)
    .addComponent(findButton)
    .addComponent(cancelButton))
);
```


GroupLayout (swing) 4

```
layout.setVerticalGroup(layout.createSequentialGroup()
  .addGroup(layout.createParallelGroup(BASELINE)
    .addComponent(label)
    .addComponent(textField)
    .addComponent(findButton))
  .addGroup(layout.createParallelGroup(LEADING)
    .addGroup(layout.createSequentialGroup()
      .addGroup(layout.createParallelGroup(BASELINE)
        .addComponent(caseCheckBox)
        .addComponent(wrapCheckBox))
      .addGroup(layout.createParallelGroup(BASELINE)
        .addComponent(wholeCheckBox)
        .addComponent(backCheckBox)))
    .addComponent(cancelButton)))
);
```

BorderLayout megvalósítása GroupLayout segítségével



35

Border via Group 1. lépés

```
GroupLayout layout = new GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setAutoCreateGaps(true);
layout.setAutoCreateContainerGaps(true);

layout.setHorizontalGroup
      (layout.createParallelGroup(CENTER)
            .addComponent(north)
            .addGroup(layout.createSequentialGroup()
                      .addComponent(west)
                      .addComponent(center)
                      .addComponent(east)
                      )
            .addComponent(south)
      );
```


36

Border via Group 2. lépés

```
layout.setVerticalGroup
     (layout.createSequentialGroup()
            .addComponent(north)
            .addGroup(layout.createParallelGroup(CENTER)
                      .addComponent(west)
                      .addComponent(center)
                      .addComponent(east)
                      )
            .addComponent(south)
     );

layout.linkSize(SwingConstants.VERTICAL, north, south);
layout.linkSize(SwingConstants.HORIZONTAL, west, east);
```


Belső osztályok

- Osztály definíciója egy másik osztályon belül
  - metódusokon kívül vagy belül
  - függvényhívás paramétereként *(csúnya!!!)*
- Nincs limit az egymásba ágyazások számára
- Eléri a befoglaló osztály tagjait
  - metódusokat
  - de csak a *final* paramétereket és *final* lokális változókat
- Cél: egységbe zárás
  - pl. kicsi segédosztály egy nagy osztályon belül: *Map – Map.Entry*

Tag osztály (member class)

- van neve
- befoglaló osztálytól függetlenül is elérhető
  - amennyiben a láthatóság engedi
- befoglaló osztály törzsében van deklarálva (nem egy metódus belsejében)

```
class In1 {
  int k;
  class In2 {int x = k;}
  void bar() {
    In2 i2 = new In2();
    k = i2.x++;
  }
}
```

```
...
In1 i1 = new In1();
i1.k++;
In1.In2 i3 =
      new In1().new In2();
...
```


39

Lokális osztály (local class)

- van neve
- kapcsos zárójeles kódblokkban van deklarálva
  - csak a blokkon belül használható

```
public class Test {
  int i = 10;
  void xxx() { i++; }
  void foo(final int a) {
    class In1 {
      int k = a;
      int j = i;
      void bar() {
        k = i++;
        xxx();
      }
    }
    In1 i1 = new In1();
    i1.j++;
  }
}
```


40

Kombinált példa

```
public class Test {
  int i = 10;
  void xxx() { i++; }
  void foo(final int a) {
    class In1 {
      int k = a;
      int j = i;
      class In2 {
        int x = k;
        int y = i;
        int z = a;
      }
      void bar() {
        In2 i2 = new In2();
        k = i2.z++;
        xxx();
      }
    }
    In1 i1 = new In1();
    i1.j++;
    In1.In2 i3 =
      new In1().new In2();
  }
}
```


Anonim osztály (anonymous class)

- sosem *abstract*, sosem *static*, mindig *final*
- nincs deklarált konstruktora

```
public class MyFrame extends JFrame {
  MyFrame() {
    super("MyFrame");
    addWindowListener(new WindowListener() {
        public void windowClosing(WindowEvent e) {
          System.exit(0);
        }
        ... // egyéb függvények
      }
    );
  }
}
```

Köszönöm a figyelmet!