

Programozás III

GRAFIKUS FELÜLETEK

GRAFIKUS JAVA ALKALMAZÁSOK



GRAFIKA – ALAPOK

Korábbi programjaink

- algoritmusvezérelt
- konzolos programok

Mire jók a konzolos programok?

Fő ok: fontos a fogalmak pontos megismeréséhez

A Java két lehetőséget biztosít grafikus felületek készítésére

AWT – Abstract Window Toolkit

SWING

és **SWT** ☺

AWT

Az adott operációs rendszer ablakozó rendszeréhez biztosít szabványos hozzáférést.



- különböző platformokon különböző, de gyors megjelenés
- szűkebb eszközkészlet



csomag: java.awt

SWING

Minden elem Java-ban van megvalósítva



- platformfüggetlen, de lassúbb megjelenés
- bővebb eszközkészlet



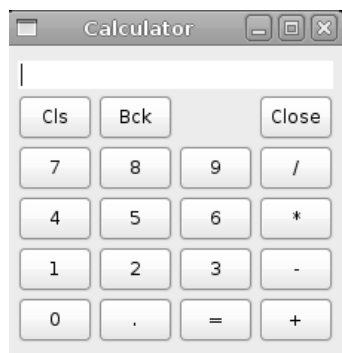
csomag: javax.swing

A Swing komponensosztályok azonosítói J betűvel kezdődnek.

SWT (STANDARD WIDGET TOOLKIT)

Nem része a szabvány Java API-nak.
Eclipse alternatíva az AWT és a Swing helyett.

Például:



AWT HIERARCHIA

Egy Java alkalmazás felhasználói interfésze (GUI)
komponensekből áll

A komponensek őssztálya a **java.awt.Component** osztály

Példák komponensekre

nyomógomb

ablak

szövegmező...stb.

AWT HIERARCHIA

A Component osztály leszármazottja a **Container**

Képes több komponens összefogására:

az *add()* metódussal adható hozzá komponens

a *remove()* metódussal távolíthatók el

önálló *Layout*-tal rendelkezik

ez azt mondja meg, hogy a benne található
komponensek milyen elrendezésben
jelenjenek meg

AWT ALAPKOMPONENSEK

Window:

A Container leszármazottja,
az ablakozó rendszer ablak fogalmának absztrakciója,
„nyers” fogalom:

- nincs kerete
- nincs fejléce
- nincs menüsora

Önállóan nem létezhet!!!

AWT ALAPKOMPONENSEK

Frame:

A Window leszármazottja,
grafikus felületű programok alapja,
az operációs rendszer ablakozó rendszeréből való
van fejléce
van kerete
adható hozzá menüsor

AWT ALAPKOMPONENSEK

LayoutManager

több komponens területi elrendezéséért felelős
egy Container objektumhoz tartozik
a komponenseket a megadott szabálynak megfelelően
rakja ki a képernyőre
módosíthatja a komponensek méretre vonatkozó
tulajdonságait (rugalmas igazodás a Container-hez)

Néhány LayoutManager:

FlowLayout – sorfolytonos elrendezés

BorderLayout – határmenti elrendezés

GridLayout – rácsos elrendezés

AbsoluteLayout...stb.

AWT ALAPKOMPONENSEK

További komponensek

Panel – panel

Label – címke

Button – nyomógomb

TextField – szövegmező

TextArea – szövegdoboz

CheckBox – jelölő négyzet

Choice – legördülő menü

List – lista

AWT ALAPKOMPONENSEK



A GRAFIKUS PROGRAMOZÁS MENETE

Az AWT alapkomponeensek a **java.awt** csomagban kaptak helyet, ezért ezt szükséges importálni.

A program alapja a Frame, ezért az osztályunk a **Frame** osztály leszármazottja lesz (öröklődés!)

Példa: Hozzuk létre az alábbi grafikus felületet:



A GRAFIKUS PROGRAMOZÁS MENETE

Importáljuk a grafikához szükséges csomagot, illetve hozzuk létre a Pelda osztályt a **Frame** osztály kiterjesztésével és az osztály törzsében deklaráljuk a szükséges komponenseket:

```
import java.awt.*;

public class Pelda extends Frame {

    private Panel panel;
    private Button gomb;
    private TextField szam1, szam2;
    private Label eredmeny;
```

A GRAFIKUS PROGRAMOZÁS MENETE

Hozzunk létre egy konstruktort, amely paraméterként kapja az ablak címét:

```
public Pelda(String cimke, int szelesseg, int magassag){
    inicializalas();
    this.setSize(szelesseg, magassag);
    this.setTitle(cimke);
}
```


A GRAFIKUS PROGRAMOZÁS MENETE

Az inicializáló
metódus:

```
public void inicializalas(){  
  
    // komponensek létrehozása  
    panel = new Panel();  
    szam1 = new TextField("",10);  
    szam2 = new TextField("",10);  
    gomb = new Button(" = ");  
    eredmeny = new Label("eredmény");  
  
    // elrendezésmenedzser  
    this.setLayout(new FlowLayout());  
    panel.setLayout(new FlowLayout());  
  
    //komponensek felrakása  
  
    add(panel);  
    panel.add(new Label("kérek két számot: "));  
    panel.add(szam1);  
    panel.add(new Label(" + "));  
    panel.add(szam2);  
    panel.add(gomb);  
    panel.add(eredmeny);  
  
    pack();  
}
```

A GRAFIKUS PROGRAMOZÁS MENETE

Végül készítsük el az indító main() metódust:

```
public static void main(String args[]){  
    Pelda g = new Pelda("peldaprogram", 600, 100);  
    g.setVisible(true);  
}
```

☺: megjelenik a várt ablak

☹: de nem jó semmire – még az ablakbezáró gomb sem működik



eseményvezérelt programozás kell

ESEMÉNYVEZÉRELT PROGRAMOZÁS

Algoritmus-vezérelt programozás:
a kód határozta meg a program menetét

Eseményvezérelt programozás:
a felhasználó határozza meg a program menetét a beavatkozásaival.

ilyen beavatkozások például:
kattintás az egérrel
billentyű leütés ... stb.

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.

ESEMÉNYEK – EVENTS

Az alacsony szintű események (pl. billentyű-, egér-esemény) az operációs rendszer szintjén keletkeznek, melyek egy eseménysorba (event queue) kerülnek.

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 úgynevezett 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.

??? – Ne essen pánikba, inkább tegyük működőképpé az előző példát!

ESEMÉNYEK KEZELÉSE – PÉLDA

Az események kezelése a java.awt.event csomaggal oldható meg:

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

A deklarálás, konstruktor ugyanaz, mint eddig.

Az inicializalas() metódus bővül majd az események figyelésével:

az ablak bezáró gombját és

az általunk felrakott nyomógombot kell figyelni.

Vagyis az eddigi metódusba még bele kell írni a következőket:

ESEMÉNYEK KEZELÉSE – PÉLDA

Az ablak bezáró gombjának működése:

Az esemény a **WindowEvent** esemény

Az eseménykezelő a **WindowListener**, amelyet az **addWindowListener** metódussal adunk hozzá a keret objektumhoz

A **WindowAdapter** osztály **windowClosing()** metódusát definiáljuk az ablakbezárás művelet végrehajtásához.

```
//ablakesemény kezelése
this.addWindowListener(new WindowAdapter(){
    @Override
    public void windowClosing(WindowEvent e){
        ablakbezaras();
    }
});
```

ESEMÉNYEK KEZELÉSE – PÉLDA



Jaj!

```
//ablak esemény kezelése
Ablak w = new Ablak();
this.addWindowListener(w);
```

```
class Ablak extends WindowAdapter{
    @Override
    public void windowClosing(WindowEvent e){
        ablakbezaras();
    }
}
```

Tömörebben:

```
//ablakesemény kezelése
this.addWindowListener(new WindowAdapter(){
    @Override
    public void windowClosing(WindowEvent e){
        ablakbezaras();
    }
});
```

belső osztály

ESEMÉNYEK KEZELÉSE – PÉLDA

A gombnyomás eseménykezelése

ActionEvent esemény

ActionListener eseményfigyelő

ActionListener interfész – egyetlen – metódusának az **actionPerformed()** metódusnak a definiálása

```
gomb.addActionListener(new ActionListener(){  
    @Override  
    public void actionPerformed(ActionEvent e){  
        gombnyomas(e);  
    }  
});
```

addActionListener(ActionListener l)

Adds the specified action listener to receive action events from this button.

actionPerformed(ActionEvent e)

Invoked when an action occurs.

ESEMÉNYEK KEZELÉSE – PÉLDA

Az eseménykezelő metódusok törzse ezekre a metódusokra hivatkozik:

```
public void ablakbezaras(){  
    System.out.println("Viszlát");  
    System.exit(0);  
}  
  
public void gombnyomas(){  
    int a = Integer.parseInt(szam1.getText());  
    int b = Integer.parseInt(szam2.getText());  
  
    eredmeny.setText(String.valueOf(a+b));  
}
```

A main() metódus ugyanaz, mint eddig.

MELYIKET SZERESSEM?

AWT vagy Swing?

A két osztálykönyvtár nem teljesen azonos feladatkört lát el, csak részben van átfedés.

Az AWT elavult, nem fejlesztik tovább, a Swing modernebb, többet tud.

Ugyanakkor az AWT hierarchiája kicsit áttekinthetőbb, ezért vettük ezt. Ha ezt megérti, akkor utána már könnyedén érti a Swing-et is.

MELYIKET SZERESSEM?

Swing vagy valami más keretrendszer?
(Pl. GWT, Vaadin, stb.)

Mára már a Swing is kissé elavult.

Ugyanakkor még mindig jól használható alkalmazásokat lehet fejleszteni vele, és jó alap a továbblépéshez.

Érdemes elolvasni:

<http://itechlog.blogspot.hu/2011/07/vastag-kliens-java-ban-netbeans.html>

MELYIKET SZERESSEM?

Osztályok importálása:

AWT számára:

```
import java.awt.*;           // AWT-komponensek,  
                             // elrendezés-kezelő  
  
import java.awt.event.*;     // eseménykezelő
```

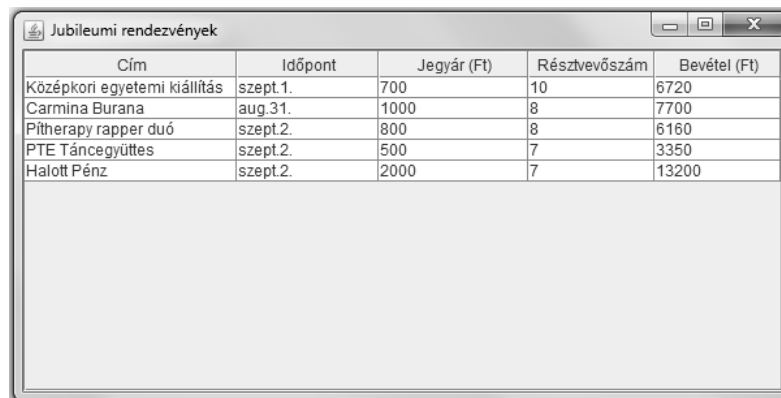
Swing számára:

```
import javax.swing.*;        // Swing-komponensek  
  
import java.awt.*;           // elrendezés-kezelő  
  
import java.awt.event.*;     // eseménykezelő
```

(A javax.swing.event csomag további eseményeket is definiál.)

SWING PÉLDA

Ld. gyak3 indító, ill. későbbi példája – nézze is át az indító feladat ide vonatkozó részét és a generált initComponents() metódusokat is a most tárgyaltak alapján.



| Cím | Időpont | Jegyár (Ft) | Részvevőszám | Bevétel (Ft) |
|------------------------------|----------|-------------|--------------|--------------|
| Középkori egyetemi kiállítás | szept.1. | 700 | 10 | 6720 |
| Carmina Burana | aug.31. | 1000 | 8 | 7700 |
| Pítherapy rapper duó | szept.2. | 800 | 8 | 6160 |
| PTE Táncegyüttes | szept.2. | 500 | 7 | 3350 |
| Halott Pénz | szept.2. | 2000 | 7 | 13200 |