

## *Programozás III*

GRAFIKUS FELÜLETEK

### PROBLÉMAFELVETÉS

Majd lehessen

- újabb ügyféllel szerződést kötni
- tetszőleges kártya egyenlegét feltölteni
- tetszőleges kártyával vásárolni

### PROBLÉMAFELVETÉS

A már meglévő banki ügyfelek adatait tároljuk egy adatfájlban. Az egyszerűség kedvéért legyen ilyen a text fájl:  
név;kártyaszám;esetleges hitelkeret.

Kovács Lajos;123587  
Faragó Edit;456788;10000  
Nagy Géza;124568;30000  
Balog Mária;325646  
Horváth Teréz;652149;20000

Ha csak két adat van, akkor bankkártya,  
ha van hitelkeret is, akkor hitelkártya.

Ékezetes betűk miatt mentés UTF8-ban

### BANKOS PÉLDA

Bankkártyás feladat

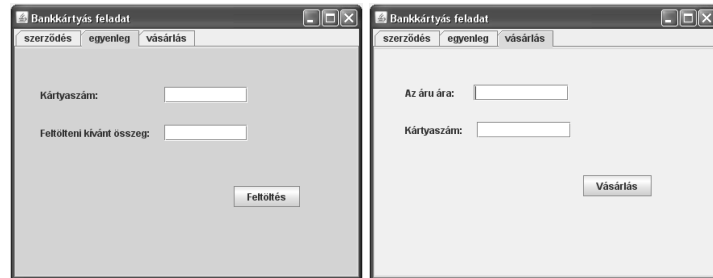
szerződés egyenleg vásárlás

Az ügyfél neve:

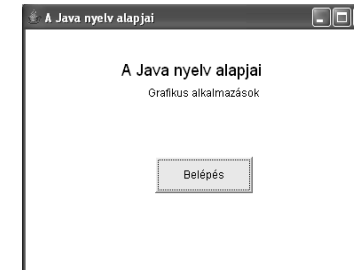
Kártyatípus:

Szerződéskötés

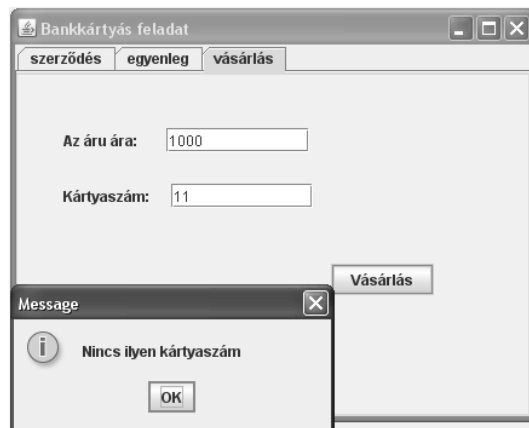
## BANKOS PÉLDA



## GRAFIKUS JAVA ALKALMAZÁSOK



## BANKOS PÉLDA



## GRAFIKA – ALAPOK

Eddigi 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

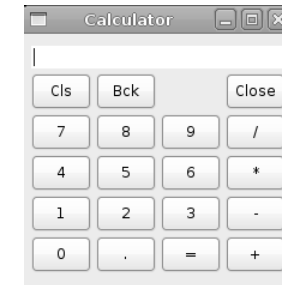


csomag: java.awt

## SWT (STANDARD WIDGET TOOLKIT)

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

Például:



## SWING

Minden elem Java-ban van megvalósítva



- platformfüggetlen, de lassúbb megjelenés
- bővebb eszközkészlet



csomag: javax.swing

A Swing komponensosztályok azonosítói J betűvel kezdődnek.

## 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

#### **Frame:**

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

### AWT ALAPKOMPONENSEK

#### **Window:**

A Container leszármazottja,  
az ablakozó rendszer ablak fogalmának absztrakciója,

„nyers” fogalom:

nincs kerete

nincs fejléce

nincs menüSOR

Önállóan nem létezhet!!!

### 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

## A GRAFIKUS PROGRAMOZÁS MENETE

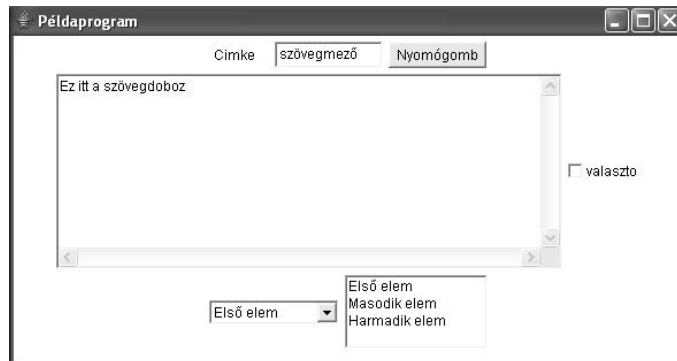
Az AWT alapkomponeensek a **java.awt** csomagban kaptak helyet, ezért ezt szükséges importálni.

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

Példa: Hozzuk létre az alábbi grafikus felületet:



## AWT ALAPKOMPONENSEK



## 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 eredmény;
```

## 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

Végül készítjük 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

## A GRAFIKUS PROGRAMOZÁS MENETE

```
public void inicializalas(){
```

Az inicializáló metódus:

```
// komponensek létrehozása
panel = new JPanel();
szam1 = new JTextField("",10);
szam2 = new JTextField("",10);
gomb = new JButton(" = ");
eredmeny = new Label("eredmény");

// elrendezésmenedzser
this.setLayout(new BorderLayout());
panel.setLayout(new BorderLayout());

//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();
}
```

## 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 KEZELÉSE

Az eseményekre csak akkor reagálhatunk, ha figyeljük őket.

Minden forrásobjektumhoz ki kell jelölni úgy nevezett figyelőobjektumokat (ezekben kezeljük a forrásobjektumon keletkezett eseményeket).

Egy forrásobjektumhoz több figyelőobjektumot adhatunk hozzá az **add\*\*\*Listener()** segítségével.

Például: addActionListener()

## 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

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 működőképessé az előző példát!

## ESEMÉNYEK KEZELÉSE – PÉLDA

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

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

A deklaráció, konstruktor ugyanaz, mint eddig.

Az inicializálás() 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



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

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

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

belső osztály

## 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

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?

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.)

## MELYIKET SZERESSEM?

AWT vagy Swing?

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

Az AWT kissé(?) elavult, nem fejlesztik tovább, a Swing modernebb, többet tud.

Ugyanakkor az AWT hierarchiája kicsit áttekinthetőbb, ezért könnyebb evvel kezdeni a tanulást.

## SWING PÉLDA

```
public class Ablak extends JFrame{
    private int szelesseg, magassag;
    private String cim;
    private JTable diakTabla;
    private JLabel diakNevLabel = new JLabel();
    private int index = -1;
    private List<Diak> diakok;

    public Ablak(int szelesseg, int magassag, String cim) {
        this.szelesseg = szelesseg;
        this.magassag = magassag;
        this.cim = cim;
        inicializalas();
    }
}
```

## SWING PÉLDA

```
private void inicializalas() {
    this.setSize(szelesseg, magassag);
    this.setTitle(cim);
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setLocationRelativeTo(null);

    this.setVisible(true);
}
```

## SWING PÉLDA



The screenshot shows a window titled "Tárgyak" with a table containing the following data:

| név           | kredit |
|---------------|--------|
| Programozás 1 | 3      |
| Programozás 2 | 4      |
| Programozás 3 | 5      |
| Matematika    | 4      |
| Hálózatok 1   | 2      |
| Hálózatok 2   | 3      |
| Fizika        | 2      |

```
public void neZarodjon() {
    this.setDefaultCloseOperation(HIDE_ON_CLOSE);
}

public void targyIr(List<Tantargy> targyak) {
    this.neZarodjon();
    this.setLocationRelativeTo(this);
    int sor = 0, oszlop = 2;
    String adatok[][] = new String[targyak.size()][oszlop];
    String[] oszlopNevek = {"név", "kredit"};
    for (Tantargy targy : targyak) {

        adatok[sor][0] = targy.getNev();
        adatok[sor][1] = String.valueOf(targy.getKredit());

        sor++;
    }
    JTable tabla = new JTable(adatok, oszlopNevek);

    JScrollPane scrollPane = new JScrollPane(tabla);
    this.getContentPane().add(scrollPane);

    this.revalidate();
}
```