VI. Grafikus Java alkalmazások

1. Bevezetés

Programjaink egészen eddig algoritmusvezérelt módon, konzolos környezetben kommunikáltunk a felhasználókkal. A program menetét az határozta meg, hogy a kódban milyen sorrendben helyeztük el az utasításokat. A program feltett egy kérdést és egészen addig várt, amíg a felhasználó reagált a kérdésre.
Napjainkban kevés olyan program működik, amely ne rendelkezne grafikus felhasználói felülettel. Ezek a programok pedig már ún. eseményvezérelt módon működnek. Ez azt jelenti, hogy a felhasználói interakció határozza meg a program menetét. Például a felhasználó egy nyomógombra kattint az egér segítségével, vagy lenyom egy billentyűkombinációt a billentyűzeten…stb.
A grafikus felhasználói felület (**G**raphical **U**ser **I**nterface) programozására a Java kétféle osztálygyűjteményt nyújt a felhasználónak, az **AWT**-t és a **Swing**et.

Abstract Window Toolkit (AWT)

- Ebben az esetben a Java csak szabványos felületet biztosít a grafikus interfésznek, a grafikus elemek az aktuális operációs rendszer ablakozó rendszerének elemei.
- A fentikeből kifolyólag eltérő platformokon eltérő formában fog megjelenni az azonos módon leprogramozott grafikus felület (például előfordulhat, hogy MS Windows alatt egy nyomógomb sarkai lekerekítettek, míg egy Unix rendszeren ugyanez a nyomógomb szögletes formában fog megjelenni)
- Az AWT eszközkészlete szegényesebb, mert itt csak a különböző operációs rendszerek nyújtotta grafikus elemek halmazainak a közös részhalmaza található meg (pl.: nyomógomb, szövegmező, rádiógomb…stb)

Swing

- minden elem tisztán Java-ban megvalósított
- platformfüggetlen megjelenés
- grafikus építőelemek színes választéka (pl.: naptár, fájl dialógus ablak…stb)
- tetszőleges új elem hozható létre
- lassabb megjelenés, mint az AWT esetén

2. Abstract Window Toolkit (AWT)

2.1 Hierarchia

Egy Java alkalmazás grafikus felhasználói interfésze (GUI) komponensekből áll. A komponenseknek meghatározott tulajdonságaik vannak (elhelyezés, méret, betűtípus, előtérszín, háttérszín, láthatóság…stb.) és előre meghatározott szabályok szerint reagálnak különböző eseményekre (egér-, billentyűesemény…stb.). Minden grafikus elem ősosztálya a **java.awt.Component** osztály.

A Component osztály leszármazottja a **Container**, amely képes több komponens összefogására:

- az *add()* metódussal adható hozzá komponens
- a *remove()* metódussal távolíthatók el a korábban hozzáadott komponensek
- ilyen például a keret , a panel, a lista …stb.

2.2 Alapkomponensek

2.2.1 Window

- a Container leszármazottja
- az ablakozó rendszer ablak fogalmának absztrakciója
- az operációs rendszer által biztosított „nyers” fogalom
    - nincs kerete
    - nincs fejléce
    - nincs menüsora
- önállóan nem létezhet

2.2.2 Frame

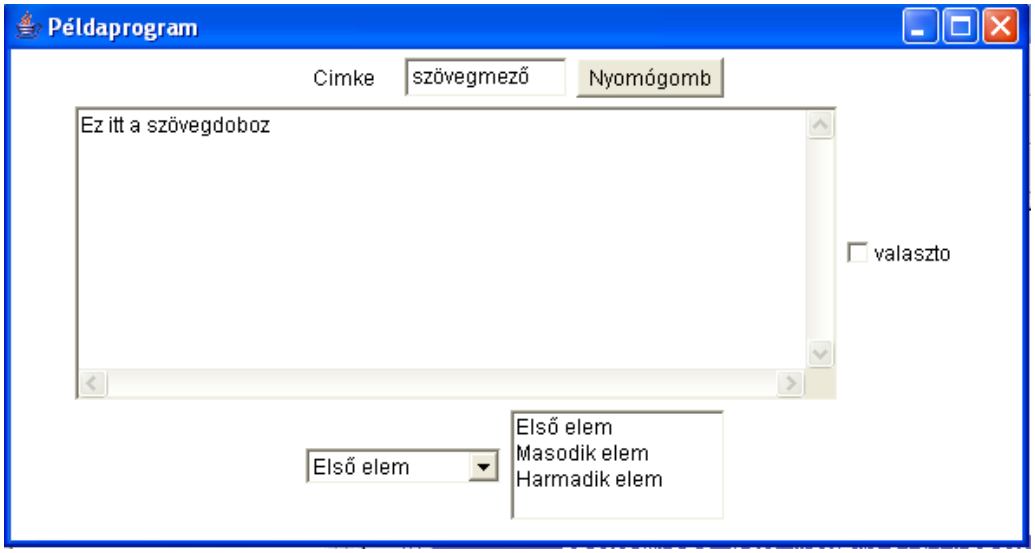
- A Window leszármazottja
- grafikus felülettel rendelkező program alapja
- ez ez ablak az OP rendszer ablakozó rendszeréből van kerete
    - van fejléce
    - adható neki menüsor

2.2.3 LayoutManager

- több komponens területi elrendezéséért felelős
- egy Container objektumhoz tartozik, a komponenseket pakolja ki a képernyőre a megadott szabálynak, stratégiának megfelelően (pl.: BorderLayout, FlowLayout, AbsoluteLayout…stb)
- a komponensek méretére vonatkozó tulajdonságait módosíthatja, ha a hozzá tartozó Container objektum mérete változik (rugalmaz felület)
- A Frame alapértelmezett LayoutManager-e a BorderLayout

2.2.4 További AWT komponensek

- Label
- Button
- TextField
- TestArea
- CheckBox
- Choice
- List
- MenuBar
- PopupMenu



Standard AWT komponensek

2.3 Egyszerű grafikus felület létrehozása

Ebben a fejezetben egy példán keresztül létre fogunk hozni egy primitív grafikus ablakot. A példa megértése azért nagyon fontos, mert a későbbiekben erre a mintára rá tudjuk „húzni” az összes grafikus programunkat!

```
import java.awt.*;

public class Pelda extends Frame
{
    //komponensek deklarálása

    public Pelda(String cim)
    {

    }

    public void inicializalas()
    {
    //komponensek és eseménykezelők felépítése
    }

    //komponensekhez tartozó események kezelése függvényekkel

    public static void main(String args[])
    {
    //a program indítása
    }
}
```

1. ábra A grafikus alkalmazás váza

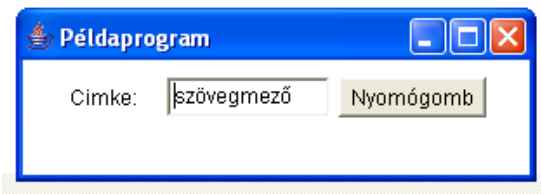
Ahogy azt már említettük, a grafikát kezelő osztályok a **java.awt** csomagban kaptak helyez, ezért szükséges a csomag importálása a program elején. Ezután létrehozunk egy osztályt, amely a ***Frame*** osztály kiterjesztése (öröklődés!!!).

A komponensek inicializálását és megjelenítését, valamint a program logikáját egy egyszerű példán kövessük végig.

Példa

A példa során hozzunk létre egy egyszerű grafikus felületet, amely egyetlen ablakból áll. Az ablaknak adjuk a „Példaprogram” címet. A 300x100 pixeles ablak területen jelenítsünk meg egy cimkét és egy szövegbeviteli mezőt, majd helyezzünk el egy nyomógombot.

A várt eredmény:



A programozás menete:

- Importáljuk a szükséges **java.awt** csomagot

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

- Hozzunk létre egy **Pelda** nevű osztályt a **Frame** osztály kiterjesztésével

```
public class Pelda extends Frame
{
```

- A Pelda osztály törzsében deklaráljuk a szükséges komponenseket

```
private Panel panel;
private Button gomb;
private TextField szovegmezo;
private Label cimke;
```

- Hozzunk létre egy konstruktort, amely paraméterként az ablak címét kapja. A konstruktor kezdeményezi a komponensek inicializálását és az ablak méterének a beállítását

```
public Pelda(String cim)
{
    inicializalas();
    setSize(300,100);
    this.setTitle(cim);
}
```

- Végezzük el a komponensek inicializálását, az ablak felépítését, az elrendezés menedzserek beállítását az ***inicializalas()*** metódusban. A komponenseket az add() metódus segítségével tudjuk betenni a Container típusú komponensekbe, jelen esetben a Frame és Panel komponensekbe.

```
public void inicializalas()
{
    //Komponensek létrehozása
    panel=new Panel();
    gomb=new Button("Nyomógomb");
    szovegmezo=new TextField("szövegmező");
    cimke=new Label("Cimke:");

    //Elrendezés menedzserek
    this.setLayout(new FlowLayout());
    panel.setLayout(new FlowLayout());

    //Komponensek felhúzása az ablakra
    add(panel);
    panel.add(cimke);
    panel.add(szovegmezo);
    panel.add(gomb);

    pack();
}
```

- Végül a kitüntetett szereppel bíró *main()* metódusban gondoskodjunk arról, hogy a programunk elinduljon. A Pelda osztály példányosítása után a *p* objektum *setVisible(true)* metódusa kezdeményezi az összeállított ablak.

```
public static void main(String[] args){
    Pelda p = new Pelda("Példaprogram");
    p.setVisible(true);
}
```

A fenti kódrészletet begépeljük és elmentjük ***Pelda.java*** néven (az osztály nevének megfelelően). JCreator-ral fordítsuk és futtassuk a programot. Látható, hogy a kiírásnak megfelelő grafikus felhasználói felületet kaptuk. Lehetőségünk van a szövegmezőbe új szöveget írni és a gombot is tudjuk „nyomkodni”. Az ablak jobb felső sarkában megjelennek az operációs rendszer ablakainál megszokott „kicsinyít”, „nagyít” és „bezár” gombok.

Első pillantásra úgy tűnhet, hogy megtanultuk a Java grafikus programozásának „javát”, de ez koránt sem igaz. Ha közelebbről megnézzük a programunkat, akkor észrevehető, hogy a bezár gomb megnyomása után nem záródik be az ablak és az is igaz, hogy a programunk – bár szép – de semmi értelme sincs, mert a gomb megnyomásának nincs semmilyen gyakorlati haszna .

Ahhoz hogy a grafikus felületű Java programunkat életre tudjuk kelteni, meg kell ismerkednünk az eseményvezérelt programozás fogalmával, amelyre a következő fejezetben kerítünk sort.

3. Eseményvezérelt programozás

A bevezetőben már tettünk utalást arra, hogy mi a különbség az algoritmusvezérelt és az eseményvezérelt programozás között. Tehát a lényeg az, hogy eseményvezérelt programozásnál a program menetét az határozza meg, hogy a felhasználó és a program milyen sorrendben milyen eseményeket vált ki (pl.: kattintás az egérrel vagy az egér görgőjének mozgatása…stb).

A felhasználó cselekedetei tehát eseményeket generálnak. Az alkalmazásunk értesül ezekről az eseményekről és ezen események alapján tudja eldönteni, hogy mit szeretne a felhasználó. A programozónak az a dolga, hogy lekódolja a programban, hogy az reagáljon-e illetve hogyan reagáljon az eseményekre.

A Javaban eseménykezelésre a ***java.awt.event*** csomag interfészei és osztályai használhatók.

Az eseménykezelés megértéséhez tisztáznunk kell néhány angol kifejezést:

- ***event*** : esemény
- ***action*** : akció
- ***listener*** : figyelő
- ***perform*** : végrehajt

Az esemény a vele összefüggő információkat (esemény forrása, típusa, időpontja…stb) magába foglaló objektum. Az események mindig valamilyen forrásobjektumon keletkeznek (pl.: nyomógomb, szövegmező…stb). Az eseményekre csak akkor reagálhatunk, ha figyeljük őket. Minden forrásobjektumhoz ki kell jelölnünk ún. figyelőobjektumokat (ezekben kezeljük le a forrásobjektumon keletkezett eseményeket). Egy forrásobjektumhoz több figyelőobjektumot hozzáadhatunk, amelyeket az add***Listener() metódussal lehet hozzáadni az objektumhoz (pl.: addActionListener()).

Az eseményeket két részre bonthatjuk:

- **alacsony szintű esemény**
  az operációs rendszer szintjén keletkező alacsony szintű esemény, amelynek forrása csak komponens lehet (pl.: egéresemény)
- **magas szintű esemény**
  minden ami nem alacsony szintű esemény, általában logikai események, amelyek nem feltétlenül komponenshez kötődnek (pl.: görgetősáv helyének megváltozása)

A tantárgy keretein belül csak néhány alacsony szintű eseménnyel foglalkozunk, amelyek jó alapot adnak ahhoz, hogy az érdeklődők a Java dokumentáció segítségével könnyen elsajátítsák a „komolyabb eseménykezelés” fogásait.

Egy objektum csak akkor figyelhet egy eseményt, ha hozzáadtuk a forrásobjektumhoz (szakkifejezéssel ezt úgy mondjuk, hogy a figyelőobjektum fel van fűzve a forrásobjektum megfelelő figyelőláncára) és osztálya implementálja a figyelőinterfészt.
Egy figyelőinterfész implementálásakor meg kell valósítanunk a benne megadott összes metódust, még azokat is, amelyeket nem akarunk működtetni. Sokszor megesik, hogy a sok eseményfajta közül csak 1-2 eseményt akarunk kezelni. Erre az esetre a Java az ún. adapterosztályokat biztosítja. Egy adapterosztály üres metódusokkal implementálja a megfelelő interfész összes metódusát, így az adapterosztály utódjában csak a számunkra fontos metódust kell felüldefiniálnunk.

A következő táblázatban egy helyre összegyűjtve megtalálhatók a legfontosabb alacsony szintű események és a használatukhoz szükséges információk:

Eseménytípus	**Forrás**	**Figyelő interfész**	**Felfűző metódus**	**Eseményadapter**	**Eseménykezelő metódusok**
ComponentEvent	Component	ComponentListener	addComponentListener	ComponentAdapter	componentResized componentMoved componentShown componentHidden
FocusEvent	Component	FocusListener	addFocusListener	FocusAdapter	focusGained focusLost
KeyEvent	Component	KeyListener	addKeyListener	KeyAdapter	keyTyped keyPressed keyReleased
MouseEvent	Component	MouseListener	addMouseListener	addMouseAdapter	mousePressed mouseReleased mouseEntered mouseExited mouseClicked
MouseEvent	Component	MouseMotionListener	addMouseMotionListener	MouseMotionAdapter	mouseDragged mouseMoved
WindowEvent	Window	WindowListener	addWindowListener	WindowAdapter	windowOpened windowClosing windowClosed windowActivated windowDeactivated
ActionEvent	*Button utódai*	*ActionListener*	*addActionListener*	*NINCS*	*actionPerformed*

Megjegyzés: az *ActionEvent* esemény már magas szintű eseménynek tekinthető.

Eseménykezeléssel kiegészített grafikus programok írásának a menete:

1) Grafikus felület megtervezése, a komponensek deklarációja
2) Grafikus felület felépítése, a komponensek elhelyezése
3) Komponensekhez kapcsolódó események eltervezése
4) Figyelő objektumok hozzárendelése a komponensekhez
5) Eseménykezelő metódusok megvalósítása, az események kezelése

Az előző fejezethez hasonlóan itt is kövessünk végig egy – most már – eseményvezérelt program megírásának menetét.

Példa

Készítsünk el egy grafikus Java programot, amely lehetőséget biztosít arra, hogy egy szövegmezőbe begépeljük a nevünket és a program gombnyomásra köszönjön nekünk. A program egyetlen ablakot használjon, aminek „bezár” gombja bezárja az ablakot. A programot tegyük attraktívvá azzal, hogy ha az egeret a nyomógomb fölé visszük, akkor annak színe megváltozik világoskékre (R:167 G:224 B:244) és ha elhúzzuk a nyomógomb felöl, akkor változzon vissza a színe az eredeti színre (R:236 G:233 B:216).

A várt eredmény:

A programozás menete:

- Importáljuk a szükséges csomagokat.

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

- Hozzunk létre egy ablakot egy szövegmező egy nyomógomb és 2db label deklarálásával, majd írjuk meg a konstruktort, ami kezdeményezi a felület inicializálását, beállítja az ablak méretét és megadja annak címét

```
public class Pelda extends Frame
{
    private Panel panel;
    private Button gomb;
    private TextField szovegmezo;
    private Label cimke;
    private Label cimke2;


    public Pelda(String cim)
    {
        inicializalas();
        setSize(600,100);
        this.setTitle(cim);
    }
```

- Az inicilaizáló metódusban hozzuk létre a komponenseket és helyezzük el őket a grafikus felületre folytonos elrendezésben (FlowLayout). A cimke2 nevű Label egyelőre üres legyen, ide fogjuk kiíratni az üdvözlő szöveget.

```
public void inicializalas()
{
    //Komponensek létrehozása
    panel=new Panel();
    gomb=new Button("Nyomógomb");
    szovegmezo=new TextField("írja ide a nevét");
    cimke=new Label("Név:");
    cimke2=new Label("                                                    ");


    //Elrendezés menedzserek
    this.setLayout(new FlowLayout());
    panel.setLayout(new FlowLayout());

    //Komponensek felhúzása az ablakra
    add(panel);
    panel.add(cimke);
    panel.add(szovegmezo);
    panel.add(gomb);
    panel.add(cimke2);
```

- Hozzunk létre eseményfigyelőt, amely az ablak bezárását figyeli (addWindowListener). Hozzunk létre egy eseményadaptert, amely implementálja a WindowListener osztály metódusait és azok közül definiáljuk újra a windowClosing() metódust, amely az ablak bezárásának eseményét írja le.

```
this.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        ablakbezaras(e);
    }
});
```

- Az előzőhöz hasonló elven hozzunk létre egy eseményfigyelőt, amely a gombnyomás eseményt felügyeli (addActionListener) és az ActionListener interfész actionPerformed() metódusával határozzuk meg a gombnyomáskor végrehajtandó kódrészletet.

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

- Majd hozzunk létre egy eseménykezelőt, ami az egér nyomógomb fölé húzását kezeli.

```
gomb.addMouseMotionListener(new MouseMotionAdapter()
{
    public void mouseMoved(MouseEvent e)
    {
        gombonEger(e);
    }
});
```

- Végül hozzunk létre egy olyan eseményfigyelőt, amely figyeli, hogy az egér mikor hagyta el a nyomógomb komponens területét

```
gomb.addMouseListener(new MouseAdapter()
{
    public void mouseExited(MouseEvent e)
    {
        gombonEgerElhagy(e);
    }
});

pack();
```

- Definiáljuk az eseménykezelő metódusokban megadott metódusokat.
    - Az *ablakbezaras()* metódus kilép a programból amellyel egyidejűleg bezáródik a grafikus ablak is.
    - A *gombNyomasKezeles()* metódus kiveszi (getText)a szövegmezőbe írt szöveget egy stringbe és ezt hozzáfűzi egy üdvözlethez kiírja (setText) a cimke2 nevű Label típusú komponensbe.
    - A *gombonEger()* és a *gombonEgerElhagy()* metódusok a nyomógomb háttérszínét változtatják.

```
public void ablakbezaras(WindowEvent e)
{
    System.out.println("Viszontlatasra...");
    System.exit(0);
}

public void gombNyomasKezeles(ActionEvent e)
{
    String s="Üdvözlöm "+szovegmezo.getText()+"!";
    cimke2.setText(s);
}

public void gombonEger(MouseEvent e)
{
    Color szin=new Color(167,224,244);
    gomb.setBackground(szin);
}

public void gombonEgerElhagy(MouseEvent e)
{
    Color szin=new Color(236,233,216);
    gomb.setBackground(szin);
}
```

- Végül a *main()* metódussal az osztály példányosításán keresztül elindítjuk a programunkat.

```
public static void main(String[] args){
    Pelda p = new Pelda("Példaprogram - eseménykezeléssel");
    p.setVisible(true);
}
```

A bemutatott módszer nem az egyetlen és nem is a legkönnyebb módszer grafikus felületek eseményvezérelt módon történő használatára, de követi a NetBeans IDE stratégiáját és segítségével átláthatóbbak, könnyebben bővíthetők lesznek programjaink. Egyéb módszerek tanulmányozásához tekintse át az ajánlott irodalom ide vonatkozó részeit. A számonkérés során természetesen mindenki a számára legszimpatikusabb módszert használhatja.

Megjegyzés(1):

A komponensekkel kapcsolatos manipulációs metódusok ismertetésére ebben az anyagban nem térünk ki, mert azok megtalálhatók a JDK help ide vonatkozó fejezeteiben. A következőkben már a NetBeans IDE-t fogjuk használni a laborgyakorlatok során, és ez az eszköz – a 4GL eszközökhöz híven – sok segítséget fog nyújtani számunkra a programozás azon részleteihez, amelyeket nem feltétlenül kell megjegyeznünk (ilyenek lesznek a komponensekhez tartozó metódusok is, amelyeket a NetBeans automatikusa fel fog ajánlani nekünk).

Megjegyzés(2):

Meggyőződésem, hogy az ebben a fejezetben ismertetett grafikus, eseményvezérelt programozási alapok elsajátításával és a tudás mélyítésével a hallgatók olyan ismereteket szereznek (szerezhetnek), amilyen ismeretekre e tantárgyon kívül a főiskolai informatikus képzésben csak elvétve esik szó. A jelenlegi piac pedig szinte kivétel nélkül az ilyen alkalmazásokra épít és a Java nyelv az egyik legszélesebb körben használt eszköz, amelynek mellesleg jövője is van. Ezért bátorítanám a hallgatókat, hogy az önálló házi-dolgozatot lelkiismeretesen készítsék el, mert csak nyerhetnek vele!