CODESPRING
OUR CODE YOUR SUCCESS
20 years

Codespring mentorprogram
Enterprise szoftverrendszerek fejlesztésének alapjai (MP I)

Let's win the race together!

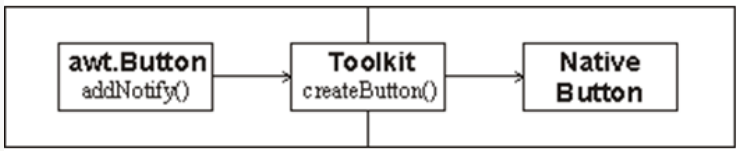
Grafikus felhasználói felületek készítése és eseménykezelés Java-ban

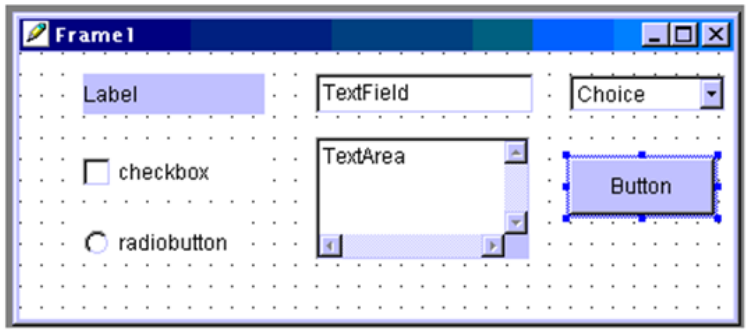
Abstract Window Toolkit, a `java.awt` és `java.awt.event` csomagok

Simon Károly
simon.karoly@codespring.ro

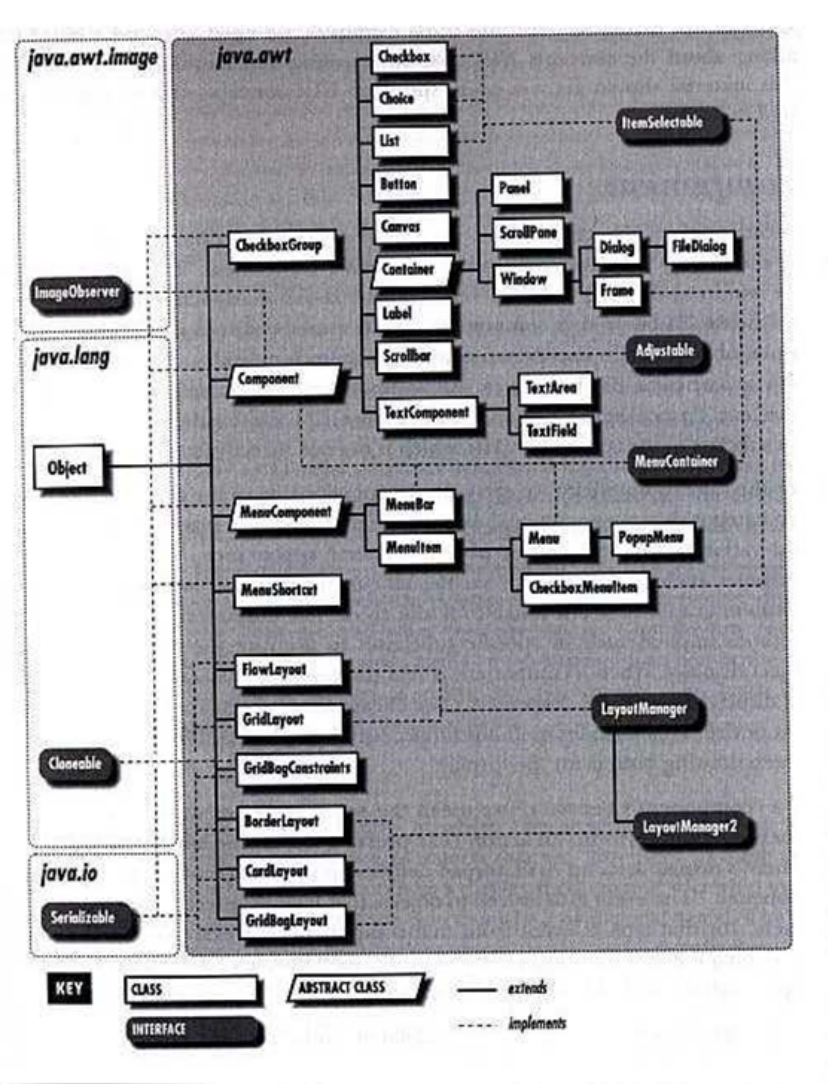
Abstract Window Toolkit

- Absztrakció: az osztályok és funkcionalitások azonosak, a komponensek az aktuális platformnak megfelelően jelennek meg, a platformfüggetlenséget „cserélhető”, platform-specifikus eszköztárak (toolkit) valósítják meg.
- Toolkit: component factory
- `java.awt.peer` – interfészek a komponens típusokhoz





AWT - osztályhierarchia



- Component: absztrakt metódusok (a különböző speciális komponensek valósítják meg ezeket), a megjelenítést és viselkedést kontroláló attribútumok és metódusok (szín, méret stb.).
- Container (tároló) osztályok: több komponenst tartalmaznak (panel, ablak stb.).
- Működés: a kommunikáció események segítségével történik, a felhasználótól származó „műveletekről" (pl. kattintás egérrel) egy AWT szál értesíti a regisztrált receptorokat (Listener objektumok).

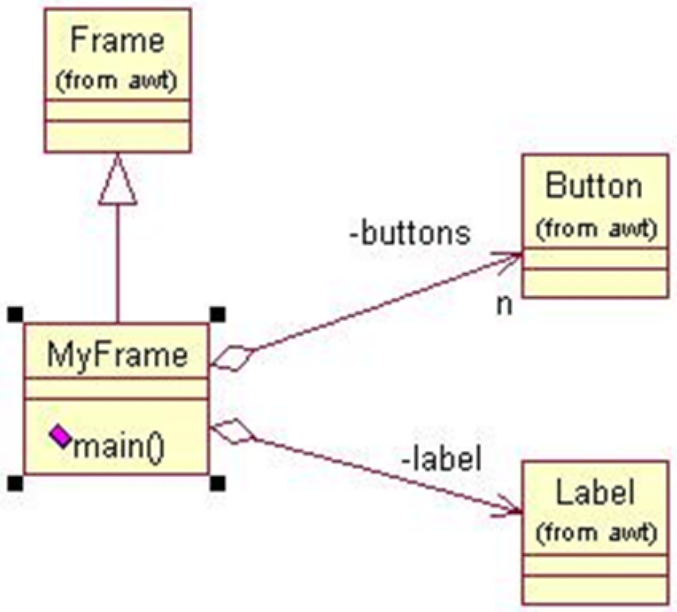
Model-View-Controller

- MVC szerkezeti minta: célja a modell (adatok, az alkalmazás által kezelt információk), a nézet (a modell megjelenítése, grafikus felhasználói felület) és a vezérlés (felhasználói műveletek, események feldolgozása) szétválasztása → növelni a rugalmasságot, egyszerűsíteni a szerkezetet.
- Példa: gomb (button):
  - Modell: logikai érték (az állapotoknak megfelelően)
  - View: a gomb megjelenítése (pozíció, méret, szín stb.)
  - Controller: a gombbal kapcsolatos események kezelése
- Általános működés:
  - A felhasználó hatást gyakorol a felületre (pl. lenyom egy gombot)
  - A vezérlő átveszi az eseményt a felülettől, majd - ha szükséges - az eseménynek megfelelően frissíti a modellt
  - A nézet (felület) a modell alapján frissítődik, és új eseményekre vár
  - Megjegyzés: a modellnek nincs tudomása a nézetről

Példa

- Ablak (frame) két gombbal (button) és egy címkével (label):

```
import java.awt.Frame;
import java.awt.Button;
import java.awt.Label;
import java.awt.BorderLayout;

public class MyFrame extends Frame {

  private Button button1;
  private Button button2;
  private Label  label;

  public MyFrame() {
      // gombok létrehozása
      button1 = new Button("Button 1");
      add(button1, BorderLayout.NORTH);
      button2 = new Button("Button 2");
      add(button2, BorderLayout.SOUTH);
      // a címke létrehozása
      label = new Label("Label");
      add(label, BorderLayout.CENTER);
  }

  public static void main(String[] args) {
      MyFrame f = new MyFrame();
      f.setBounds(10,10, 300, 300);
      f.setVisible(true);
  }
```



Komponensek elrendezése

- A komponensek tárolókon (container) belüli elrendezése elrendezés-menedzser (layout manager) osztályok segítségével történik: **`FlowLayout`**, **`GridLayout`**, **`CardLayout`**, **`BorderLayout`**, **`GridBagLayout`** stb.
- Példák (tároló osztályokon belül):

```
FlowLayout fLayout = new FlowLayout();
setLayout(fLayout);
Button okButton = new Button("Ok");
add(okButton);
Button cancelButton = new Button("Cancel");
add(cancelButton);

FlowLayout fLayout = new FlowLayout();
fLayout.setAlignment(FlowLayout.LEFT)
setLayout(fLayout);

setLayout(new BorderLayout(), 10, 6);
add(new Button("Egy"), BorderLayout.NORTH);
add(new Button("Ketto"), BorderLayout.EAST);
add(new Button("Harom"), BorderLayout.SOUTH);
add(new Button("Negy"), BorderLayout.CENTER);
```

Eseménykezelés

- A GUI komponensek közötti kommunikáció eseményeken (events) keresztül történik. Az események tulajdonképpen üzenetek (pl. egy gomb értesíti az alkalmazást, ha a felhasználó kattintott rá).
- Egy eseménynek egy forrása és egy vagy több címzettje (receptor) lehet. A címzett a fogadott eseményeknek megfelelő összes metódust implementálja.
- Esemény példák: `ActionEvent`, `MouseEvent`
- Az események eljuttatása a címzetthez **`megfigyelt – megfigyelő`** modell alapján történik:

Eseménykezelés

- A regisztrálás pillanatában a receptor hozzáadódik egy listához. Ez a lista tartalmazza mindazokat a receptorokat, akik érdekeltek az illető esemény megfigyelésében.
- Az esemény a címzetthez egy megfelelő metódus meghívásával jut el, ezek a metódusok az eseményeknek megfelelő Listener interfészek metódusai
- Pl. kattintás gombra:

`ActionEvent – ActionListener – public void actionPerformed(ActionEvent))`

Eseménykezelés példa

- Gomb (button) - ActionEvent típusú események forrása lehet:

```
Button b = new Button("OK");
```

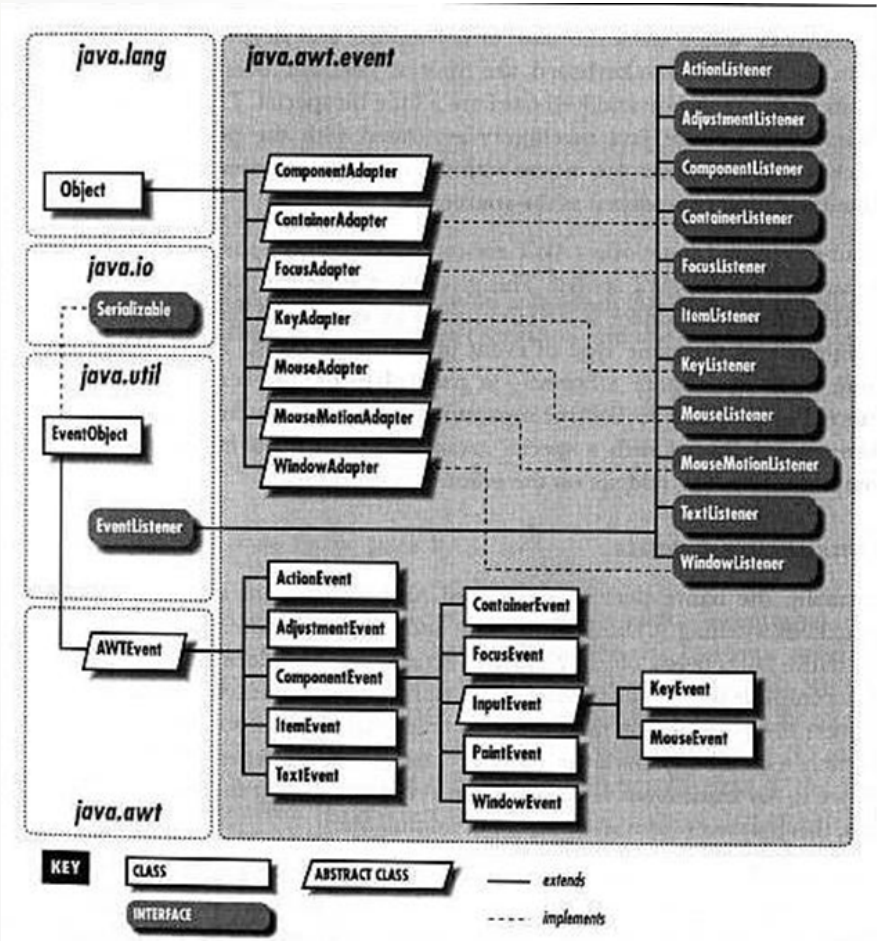
- Címzett:

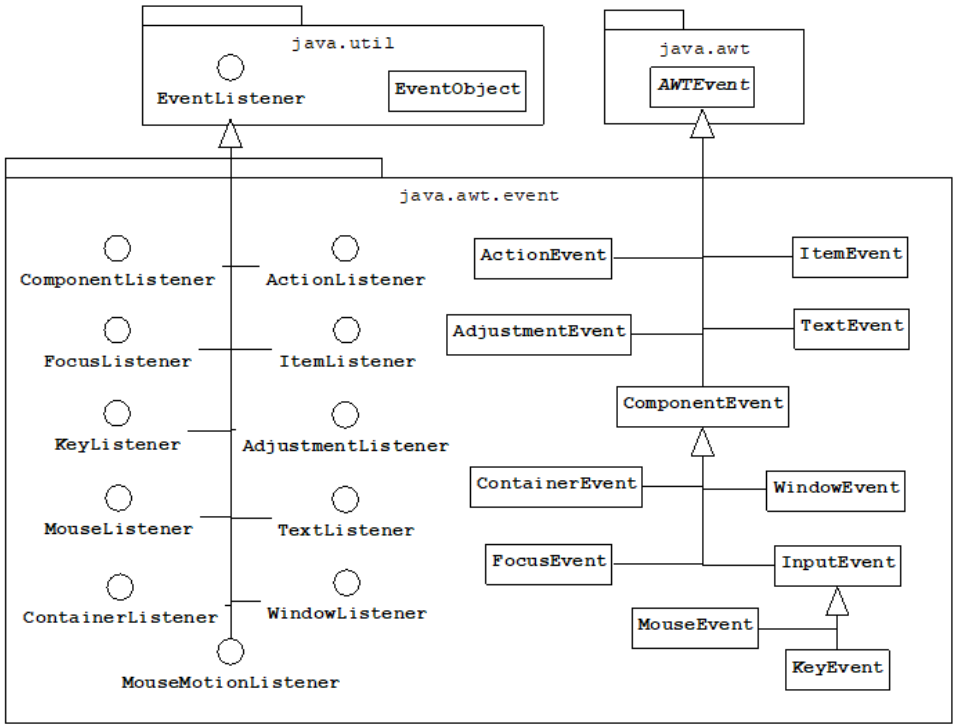
```
class Receptor implements ActionListener {
    private void setupReceiver() {
      ...
      b.addActionListener( this );
    }

    public void actionPerformed(ActionEvent e) {
      // mi történjen a gomb lenyomásakor
    }

}
```

- A forrásnak lehetőséget kell adnia a címzettek regisztrációjára. Ehhez esetünkben a Button osztály a következő metódusokat biztosítja:

```
public void addActionListener( ActionListener listener ){...}
public void removeActionListener( ActionListener listener ){...}
```

AWT események hierarchiája





AWT - általános események

Esemény	Listener interfész	Megfelelő metódusok

`ComponentEvent`	`ComponentListener`	`componentResized()`
		`componentMoved()`
		`componentShown()`
		`componentHidden()`

`FocusEvent`	`FocusListener`	`focusGained()` `focusLost()`
`ContainerEvent`	`ContainerListener`	`componentAdded()` `componentRemoved()`

AWT - specifikus események

Esemény	Előfordulás	Listener interfész	Megfelelő metódusok
ActionEvent	TextField MenuItem List Button	ActionListener	actionPerformed()
ItemEvent	List CheckBox Choice CheckboxMenuItem	ItemListener	itemStateChanged()
AdjustmentEvent	ScrollPane Scrollbar	AdjustmentListener	adjustmentValueChanged()
TextEvent	TextArea TextField	TextListener	textValueChanged()

AWT - window events

Esemény	Előfordulás	Listener interfész	Megfelelő metódusok

`WindowEvent`	`Window`
	`Frame`
	`Dialog`
		`WindowListener`	`windowOpened()`
			`windowClosing()`
			`windowClosed`
			`windowIconified()`
			`windowDeiconified()`
			`windowActivated()`
			`windowDeactivated()`

AWT - input events

Esemény	Listener interfész	Megfelelő metódusok
KeyEvent	KeyListener	keyTyped() keyPressed() keyReleased()
MouseEvent	MouseListener	mouseClicked() mousePressed() mouseReleased() mouseEntered() mouseExited()
MouseEvent	MouseMotionListener	mouseMoved() mouseDragged()

AWT - input events

- `java.awt.event.InputEvent`
- Konstansok: `SHIFT_MASK`, `CTRL_MASK`, `META_MASK`, `ALT_MASK`, `BUTTON1_MASK`, `BUTTON2_MASK`, `BUTTON#_MASK` stb. + metódusok

```
public void mousePressed(MouseEvent e){
    int mods = e.getModifiers();
    if( ( mods & InputEvent.SHIFT_MASK) != 0 ){
      // SHIFT lenyomva
      ...
    }
}
```

Figyelő osztályok

- Példa: italautomata - három gomb: kávé, tea, üdítő

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == teaButton) ...
    else if ...
  }
```

- A GUI egybeolvad a logikával, az MVC elv sérül. Jobb megoldás lehet külön figyelő osztályok alkalmazása:

```
class CoffeeListener implements ActionListener {

    MachineController c;

    CoffeeListener(MachineController c) {
      this.c =  c;
    }

    public void actionPerformed(ActionEvent e) {
      c.coffee();
    }

}
```

- Használat:

```
MachineController c;
...
coffeeButton.addActionListener(new CoffeeListener(c));
```

Figyelő osztályok és adapterek

- A figyelő osztályok implementációjánál legtöbbször belső osztályokat, vagy név nélküli belső osztályokat alkalmazunk a View osztályon belül
  - Megjegyzés: ekkor a vezérlőre mutató referencia átadása sem szükséges: ha a nézet rendelkezik referenciával, a belső osztályból elérhető az illető adattag)
- A több metódussal rendelkező figyelő interfészek esetében adapter osztályok bevezetése is indokolt lehet. Pl. az AWT minden figyelő interfészéhez biztosít ilyent. Ezek egyszerűen üresen hagyják a metódusok törzsét. Előnyük, hogy saját figyelő osztályainkat ezekből származtatva csak a ténylegesen szükséges metódusokat kell újradefiniálnunk (nem feltétlenül szükséges az összes metódus implementálása, mint az interfész közvetlen megvalósítása esetén)

```
myComponent.addMouseListener(new MouseAdapter() {

  @Override
  public void mousePressed(MouseEvent e ) {
    ...
  }

});
```

Feladatok

1. Hozzunk létre egy keretet (Frame), és ezen belül helyezzünk el egy panelt, valamint egy címkét (Label). Ha a felhasználó a panelre kattint az egérrel, a címkén jelenítsük meg az egérkattintás koordinátáit. A komponensek elhelyezésére használjunk egy megfelelő LayoutManager példányt, ne rögzítsük a pozíciókat és méreteket (ez a javaslat a legutolsó kivételével a következő feladatokra is érvényes).
2. Egészítsük ki a programot, olyan módon, hogy ne csak az egérkattintást figyeljük, hanem az egérmutató mozgását is. A címkére az aktuális esemény típusát is írjuk ki a koordináták mellé: amennyiben a felhasználó kattintott a „clicked” üzenet, amennyiben csak elmozdította a pointert a „moved” üzenet, amennyiben lenyomott gombbal mozdította a pointert a „dragged” üzenet jelenjen meg.

Feladatok

3. Hozzunk létre egy keretet, és ezen belül helyezzünk el egy többsoros szöveg megjelenítésére alkalmas komponenst (TextArea), egy egysoros szöveg bevitelére alkalmas szövegmezőt (TextField), valamint egy gombot. Ha a felhasználó a gombra kattint, vagy a szövegmezőn belül lenyomja az enter billentyűt, a szövegmező tartalmát hozzáadjuk a TextArea tartalmához, majd töröljük a szövegmezőből (lehetőséget adva egy új szöveg beírására).
4. Egy kereten belül egy címke szövegét változtassuk jelölőnégyzetek (Checkbox) segítségével: a címkén mindig az aktuálisan kijelölt jelölőnégyzeteknek megfelelő címkék szövegét jelenítsük meg. Alakítsuk át a programot, olyan módon, hogy egyszerre csak egy jelölőnégyzet legyen kiválasztható (a jelölőnégyzet komponensek „radio button” komponensekbe történő alakítása, a CheckboxGroup osztály segítségével).

Feladatok

5. Egy kereten belül helyezzünk el több különböző színű panelt, a színeket véletlenszerűen generálva. Ha az egérmutató belépik egy adott panel fölé, az illető panel véletlenszerűen színt vált.
6. Egy kereten belül helyezzünk el egy gombot, véletlenszerűen generált koordinátákra, a „Push me!” felirattal. Amikor a felhasználó megpróbál a gombra kattintani (az egérmutató a gomb fölé kerül), a gomb elmozdul (véletlenszerűen újrageneráljuk a koordinátákat, és áthelyezzük a gombot az új koordinátákra). A feladatnak elkészíthetjük egy olyan változatát is, amikor tényleg nem lehetséges a gomb lenyomása: az új koordináták generálásánál kiszűrjük annak a lehetőségét, hogy a gomb újra a mutató alá kerüljön.

Útmutatás: a koordináták véletlenszerű generálásához a java.util.Random osztályt használhatjuk (figyelem: a generátorból egyetlen példány elégséges, ezután ettől több érték is elkérhető a megfelelő metódusok meghívásával, ezért ne hozzunk létre minden változtatáskor egy új Random példányt). A feladatot úgy oldhatjuk meg legegyszerűbben, ha ezúttal (kivételesen) nem használunk LayoutManager példányt (a setLayout metódus null paramétert is elfogad), hanem meghatározzuk a gomb méretét és pozícióját (ez utóbbit változtatva a megfelelő esemény felléptekor).