

Let's win the race together!



Swing

A `javax.swing` csomag

Simon Károly
simon.karoly@codespring.ro

-
- ```

classDiagram
 class Object["Object (from lang)"]
 class Component["Component (from awt)"]
 class Container["Container (from awt)"]
 class JComponent["JComponent (from swing)"]
 class JSlider["JSlider (from swing)"]
 class JScrollBar["JScrollBar (from swing)"]
 class JTextComponent["JTextComponent (from text)"]
 class JLabel["JLabel (from swing)"]
 class AbstractButton["AbstractButton (from swing)"]
 class JTextField["JTextField (from swing)"]
 class JPasswordField["JPasswordField (from swing)"]
 class JEditorPane["JEditorPane (from swing)"]
 class JTextArea["JTextArea (from swing)"]
 class JTextPane["JTextPane (from swing)"]
 class JButton["JButton (from swing)"]

 Object --|> Component
 Component --|> Container
 Container --|> JComponent
 JComponent --|> JSlider
 JComponent --|> JScrollBar
 JComponent --|> JTextComponent
 JComponent --|> JLabel
 JComponent --|> AbstractButton
 JTextComponent --|> JTextField
 JTextComponent --|> JEditorPane
 JTextComponent --|> JTextArea
 JPasswordField --|> JTextField
 JTextPane --|> JEditorPane
 JButton --|> AbstractButton

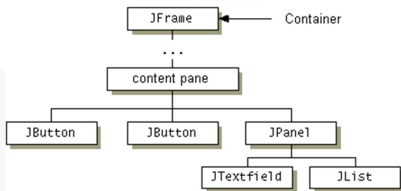
```

- ▶ JFrame: a Frame kiterjesztése
- ▶ NEM adhatunk hozzá direkt módon SWING komponenseket
- ▶ Tartozik hozzá egy **ContentPane** (**Container** típus), ehhez adjuk hozzá a komponenseinket:

```
Container contentPane = this.getContentPane();
contentPane.setLayout(new FlowLayout());
contentPane.add(button1);
contentPane.add(button2);
contentPane.add(panel);
```

- ▶ Vagy:

```
JPanel panel = new JPanel();
panel.setLayout(new FlowLayout());
panel.add(textField);
panel.add(list);
...
this.setContentPane(panel);
```



- ▶ A komponensek megjelenítése már nem platformfüggő, az **UIManager** osztály segítségével befolyásolhatjuk a felszínünk grafikus megjelenítését, különböző „look and feel” osztályokat rendelhetünk hozzá:

```
try {
 UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
} catch (Exception e) {
 System.err.println("Couldn't use the metal look and feel: " + e);
}
```

- ▶ **JLabel**: az `awt.Label` SWING-es megfelelője. Ikont rendelhetünk hozzá, lehetővé teszi a tartalom pozicionálását. Megjegyzés: a SWING címkék esetében html formázás is alkalmazható.
- ▶  **JButton**: az `awt.Button` SWING-es megfelelője, ikon rendelhető hozzá
- ▶  **JTextComponent**:
  - ▶ Hasznos metódusok: `copy()`, `cut()`, `paste()`, `getSelectedText()`, `setSelectionStart()`, `setSelectionEnd()`, `selectAll()`, `replaceSelection()`, `getText()`, `setText()`, `setEditable()`, `setCaretPosition()`
  - ▶ Származtatott osztályok: `JTextField`, `JTextArea`, `JTextPane`.
  - ▶ Továbbá a `JTextField` leszármazottja: `JPasswordField` (jelszavak beolvasására)
- ▶  **JScrollBar**: az `awt.ScrollBar` SWING-es megfelelője
- ▶  **JSlider**: a `JScrollBar`-hoz hasonló, plusz funkciókkal
- ▶  **JProgressBar**: folyamatok, műveletsorok állapotának monitorizálására
- ▶  **JComboBox**: az `awt.Choice` komponenshez hasonló, lehetővé teszi egy szerkeszthető mező hozzáadását
- ▶  **JList**: az `awt.List` megfelelője
- ▶ Más eszközök (nem grafikus komponensek): pl.  **Timer** (bizonyos időintervallum eltelte után `ActionEvent`-et generál) stb.

```
public interface Icon{
 void paintIcon(Component c, Graphics g, int x, int y);
 int getIconWidth();
 int getIconHeight();
}
```

## ► Használata:

```
public class RedOval implements Icon {

 public void paintIcon(Component c, Graphics g, int x, int y) {
 g.setColor(Color.red);
 g.drawOval(x, y, getIconWidth(), getIconHeight());
 }

 public int getIconWidth() {
 return 10;
 }

 public int getIconHeight() {
 return 10;
 }
}
```

## ► Vagy:

```
Icon iconPicture = new ImageIcon("Apple.gif");
```

```
import javax.swing.*;
import java.awt.*;

public class SwingPanel extends JPanel {

 public SwingPanel() {
 setLayout(new GridLayout(3, 1));
 JLabel simplelabel = new JLabel("This is a simple label");
 add(simplelabel);
 JLabel iconlabel = new JLabel("This is a fancy label");
 Icon icon = new ImageIcon("icon.gif");
 iconlabel.setIcon(icon);
 Font font = new Font("Serif", Font.BOLD|Font.ITALIC, 30);
 iconlabel.setFont(font);
 add(iconlabel);
 JLabel myiconlabel = new JLabel("This is my icon label");
 Icon myicon = new RedOval();
 myiconlabel.setIcon(myicon);
 add(myiconlabel);
 }
}
```

```
import javax.swing.JFrame;

public class SwingFrame extends JFrame {

 private SwingPanel p;

 public SwingFrame() {
 p = new SwingPanel();
 getContentPane().add(p);
 }

 public static void main(String args[]) {
 SwingFrame f = new SwingFrame();
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 f.setBounds(1, 1, 400, 300);
 f.setVisible(true);
 }
}
```





- ▶ Az AWT heavyweight komponenseket használ (mindegyiknek megfelel egy natív peer), a SWING lightweight komponenseket (nincs natív párjuk).
- ▶ A Swing komponensek egy natív konténeren belül lesznek megjelenítve, az AWT komponensek mindegyikének megfelel egy natív ablak
  - ▶ Z-ordering problémák (a SWING komponensek öröklik az őket tartalmazó tároló szintjét, így az AWT komponensek elfedik őket).
  - ▶ egy alkalmazáson (tárolón) belül nem javasolt a különböző típusú komponensek keverése.
- ▶ A lightweight komponensek tartalmazhatnak „átlátszó” részeket (a felület nem lesz feltétlenül kitöltve háttérszínű pixelekkel → a komponensek nem feltétlenül „téglalap alakúak”, az egérműveletek „továbbítódhatnak” az őket tartalmazó tárolókhoz).

- ▶ AWT:
  - ▶ A (natív) rendszer által kiváltott (system triggered) frissítés: mikor először jelenik meg a komponens, mikor újraméreteződik, mikor „sérül” és frissíteni kell → **paint** metódus meghívása (a teljes érintett felület újrarajzolódik)
  - ▶ Az alkalmazás által kiváltott (application triggered) frissítés: az alkalmazás kéri a komponens frissítését (valamilyen állapotváltozás következményeként) → a **repaint** metódus meghívása az **update**, majd a **paint** meghívását eredményezi (az update alap implementációja „törli” a komponens felületét, de újradefiniálható → incremental painting)
- ▶ SWING:
  - ▶ A tárolók frissítése először saját felületük, majd az általuk tartalmazott komponensek felületének frissítését eredményezi → a **paint** metódus újradefiniálásánál fontos a **super.paint()** metódushívás alkalmazása (ellenkező esetben a tartalmazott komponensek nem jelennek meg, mivel a **Container.update()** alap implementációja nem biztosít rekurzív **update** és **paint** metódushívásokat a tartalmazott komponensekre)
  - ▶ A system triggered frissítést nem csak a natív rendszer (első megjelenés), hanem a lightweight framework is kiválthatja (későbbi frissítés), és ez **repaint** metódushíváson keresztül történik → a SWING esetében nincs igazi különbség az **update/paint** között

- ▶ Double buffering támogatás: offscreen buffer hozzárendelése a tárolókhhoz
  - ▶ `public boolean isDoubleBuffered()` – alapértelmezetten `true`
- ▶ Transparency és overlapping: a komponensek felületén nincs minden pixel kirajzolva, és a komponensek részlegesen fedhetik egymást → a frissítésnél a komponens alatti, illetve az egymást fedő részeket is frissíteni kell (a hierarchia bejárása).
  - ▶ `public boolean isOptimizedDrawingEnabled()`
  - ▶ `true`, ha a komponens biztosítja, hogy nem tartalmaz egymást fedő komponenseket (nincs szükség a hierarchia bejárására).
- ▶ Opacity: `public boolean isOpaque()`
  - ▶ `true` – a komponensnek megfelelő téglalap alakú felület minden pixele meg lesz jelenítve (a legtöbb esetben)
  - ▶ `false` – a komponens nem garantálja minden pixel megjelenítését
  - ▶ Nem átlátszóságot jelent, tulajdonképpen csak egy „szerződés” a grafikus frissítést végző rendszerrel. A komponenseknek maguknak kell biztosítaniuk a megfelelő implementációt.
- ▶ Paint metódusok:

```
protected void paintComponent(Graphics g)
protected void paintBorder(Graphics g)
protected void paintChildren(Graphics g)
```

  - ▶ általában csak a `paintComponent` metódust akarjuk újradefiniálni (és akkor csak ezt tesszük!)

- ▶ Az AWT grafikával kapcsolatos feladat (alakzatok kirajzolása) megoldása során elkészített program grafikus felületét írjuk át SWING komponensek felhasználásával. A programot egészítsük ki néhány új funkcionalitással. Egy **JSlider** segítségével legyen változtatható az alakzat mérete, olyan módon, hogy az meg is haladhassa a vászon (esetünkben **JPanel** komponens) aktuális méretét. Amennyiben az alakzat „kilóg” a vászonból, jelenjenek meg görgetőszalagok, amelyek segítségével változtathatjuk az éppen látható felületet (**JScrollPane** komponens alkalmazhatunk). A szín kiválasztására ezúttal egy külön grafikus felületet is biztosítsunk, a **JColorChooser** komponens felhasználásával.