Object Oriented Programming

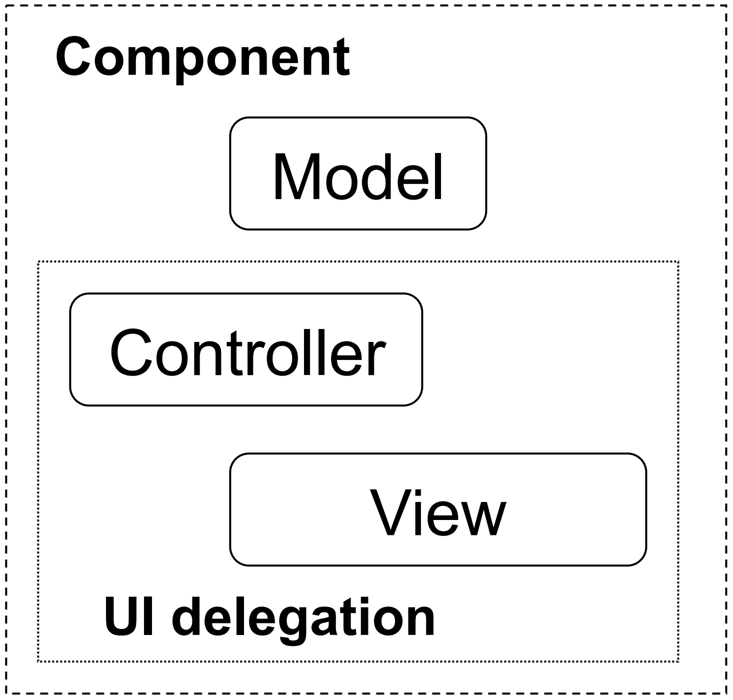
Graphical user interfaces

Introduction (1)

- Java provides packages with **graphical user interfaces (GUI)**.
  - `AWT` (Abstract Window Toolkit)
    - package java.awt (`import java.awt.*;`)
    - available in JSE 1.1
    - depends on native code
  - `Swing`
    - package javax.swing (`import javax.swing.*;`)
    - available in JSE 1.2
    - extends AWT (17 packages in version 1.4)

Introduction (2)

- Swing has components, that follow the architecture Model-View-Controller (MVC):
  - **Model:** Component data and rules that govern access to and updates of this data.
  - **View:** Renders the contents of a model. It specifies exactly how the model data should be presented.
  - **Controller:** Translates the user's interactions with the view into actions that the model will perform.



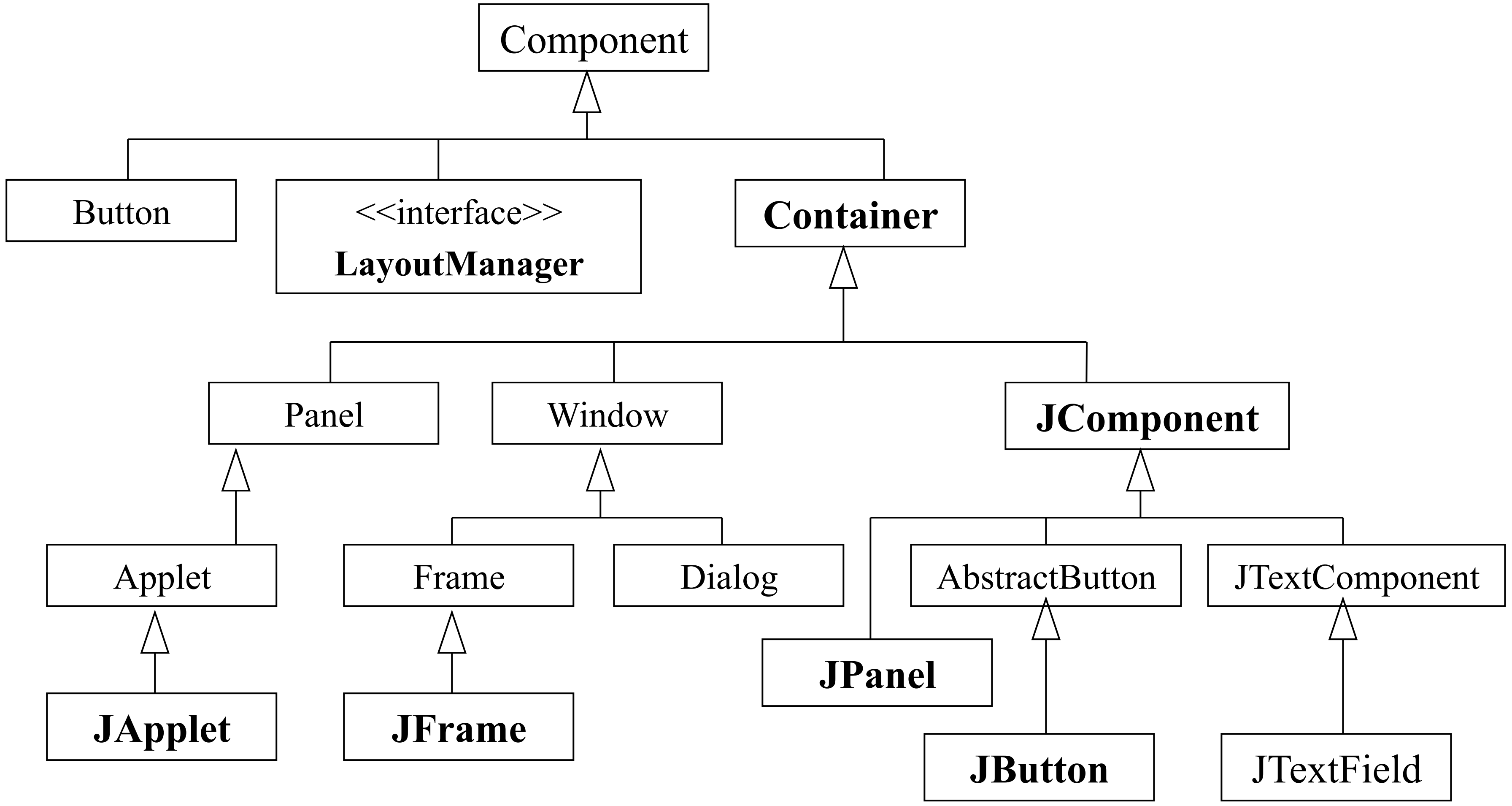
Introduction (3)

- Swing* components:
  - `JComponent`
    - `JPanel`
    - `JButton`
    - …
  - `JFrame`
  - `JDialog`
  - `JApplet`

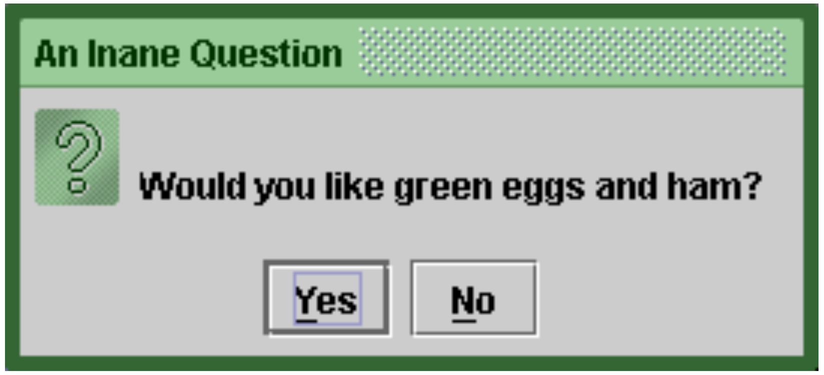
  All that appears in a window belongs to an hierarchy of one (or more) containers which are instances of `JFrame`, `JDialog` or `JApplet`.

  Note: AWT components have the same identifier, without the prefix J!

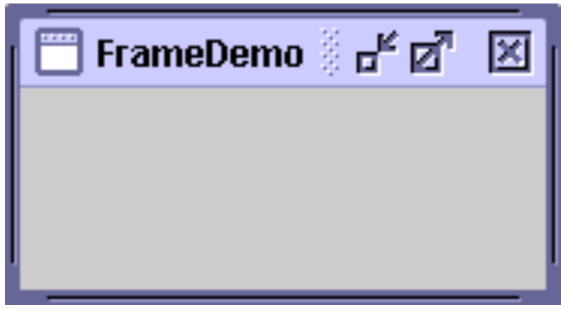
Introduction (4)



Swing components: top-level containers

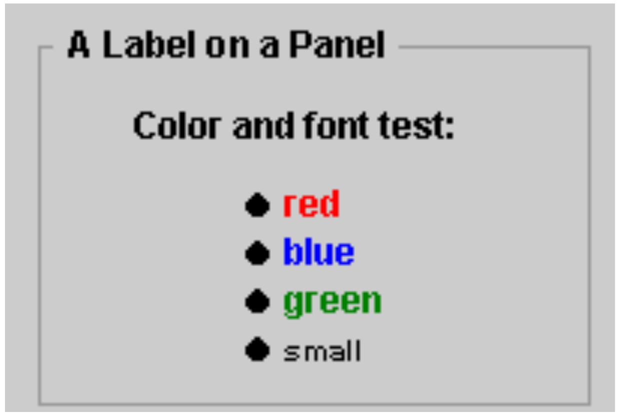


`JDialog`

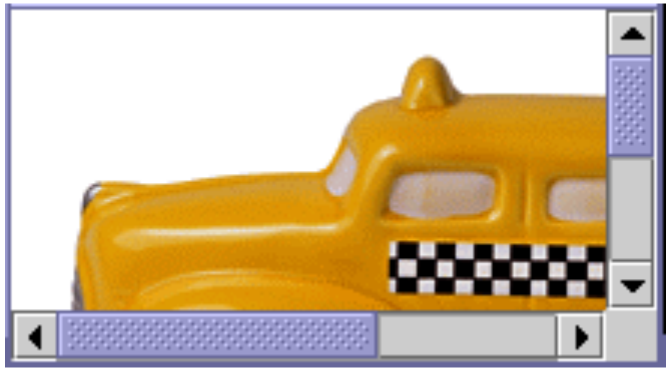


`JFrame`

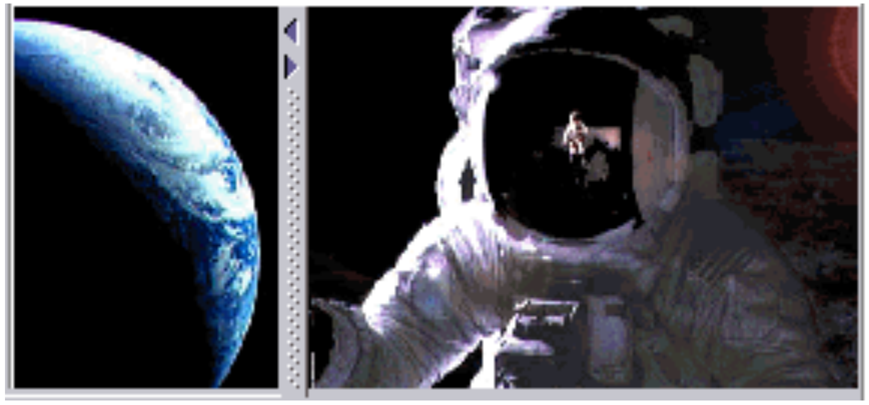
Swing components: intermediate containers



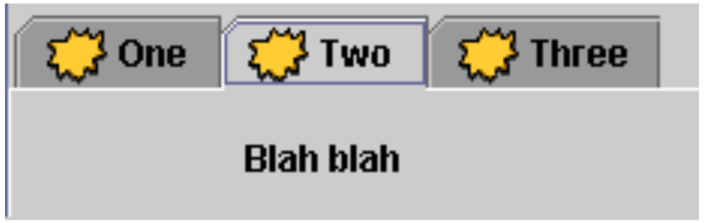
`JPanel`



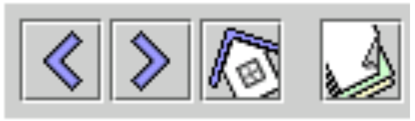
`JScrollPane`



`JSplitPane`

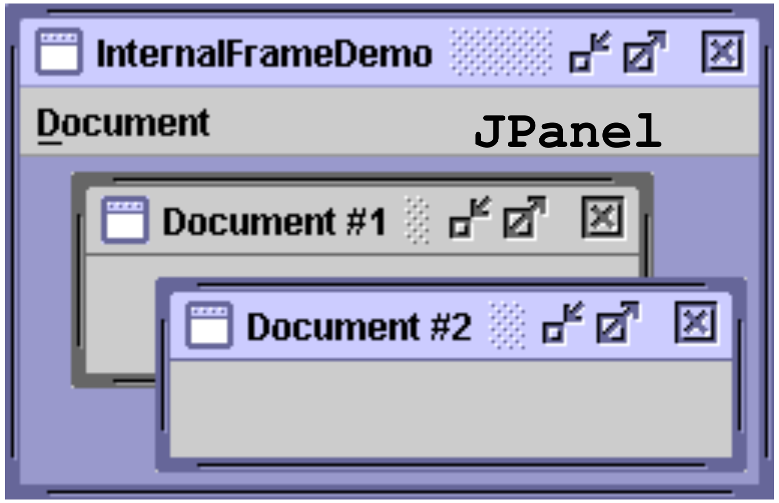


`JTabbedPane`



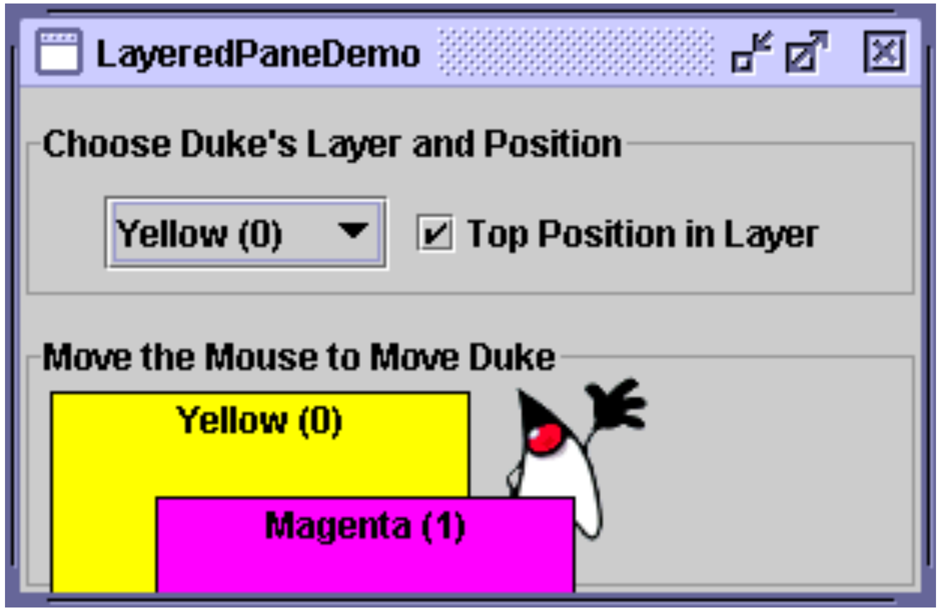
`JToolBar`

Swing components: special containers



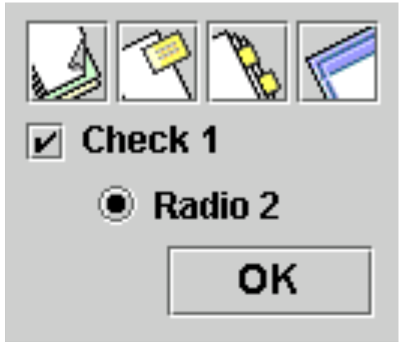
JPanel

JInternalFrame

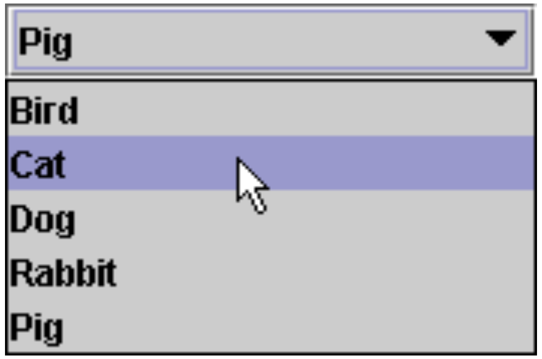


JLayeredPane

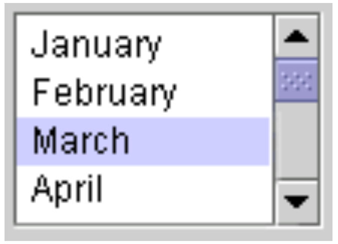
Swing components: atomic components



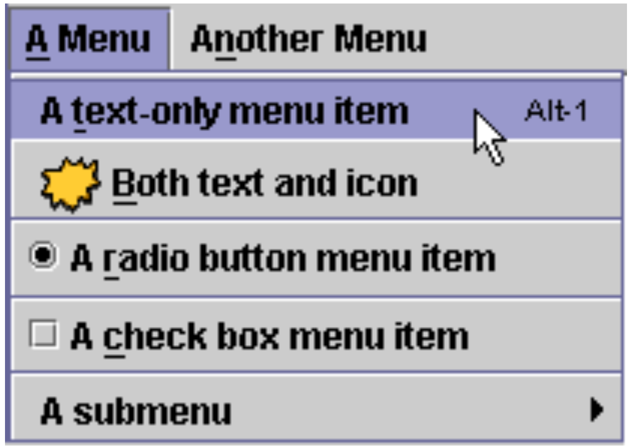
`JButton`



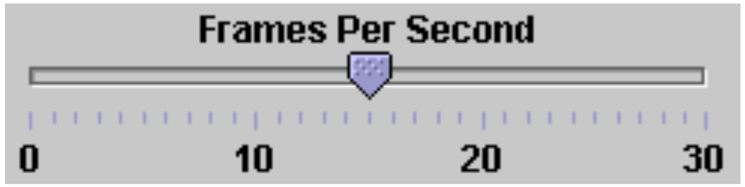
`JComboBox`



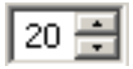
`JList`



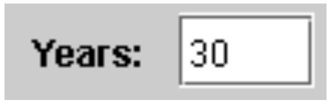
`JMenu`



`JSlider`

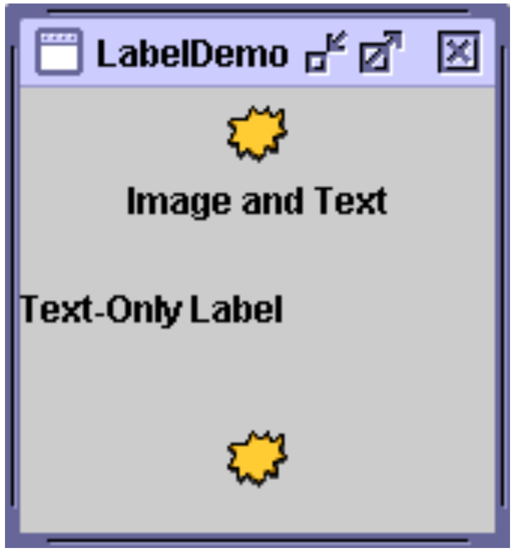


`JSpinner`

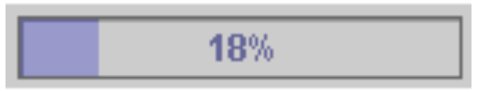


`JTextField`

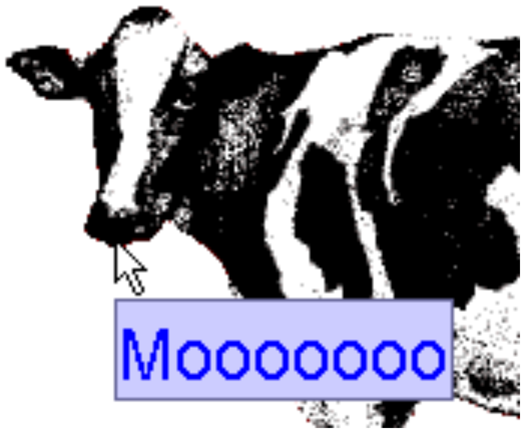
Swing components: atomic components



`JLabel`

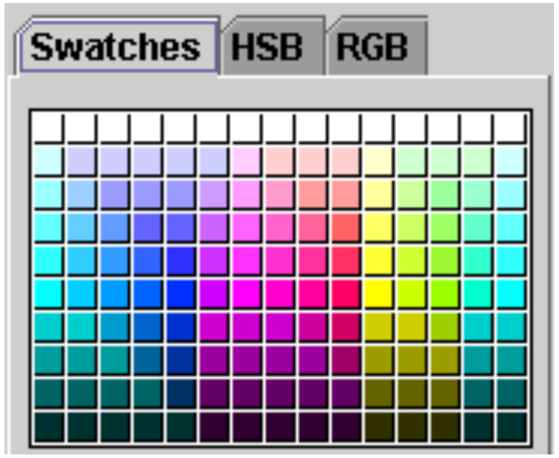


`JProgressBar`

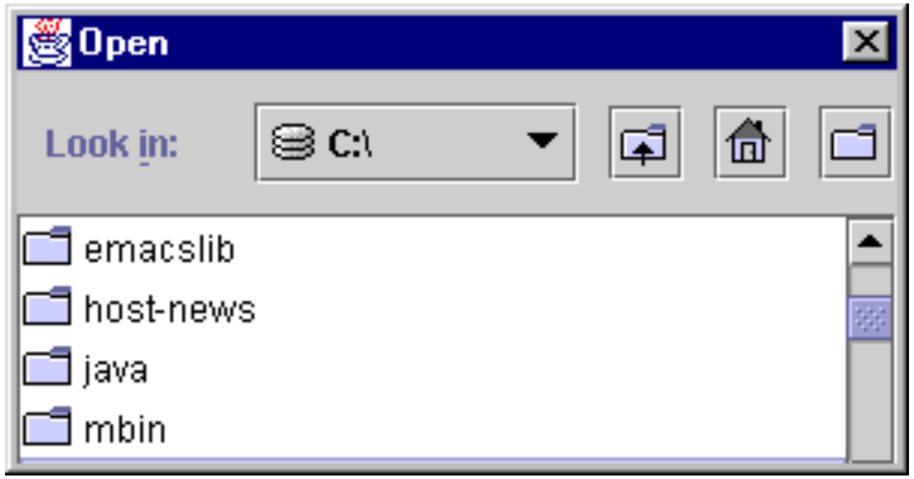


`JToolTip`

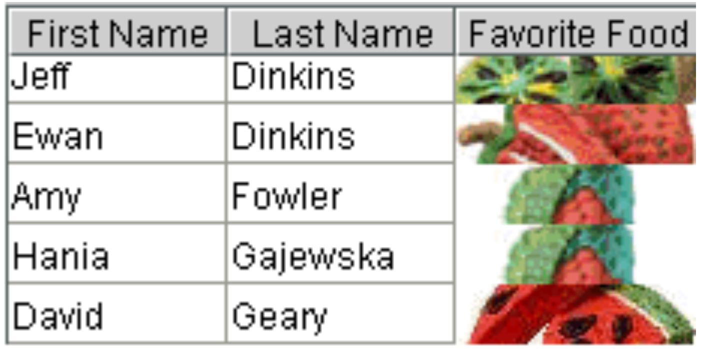
Swing components: advanced components



JColorChooser

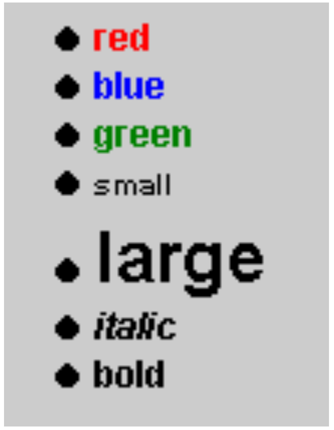


JFileChooser



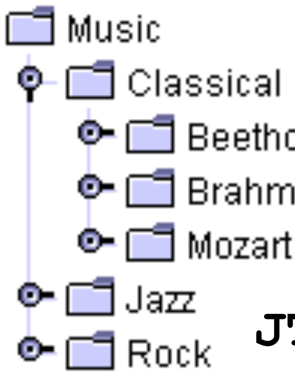
First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

JTable



- red
- blue
- green
- small
- large
- *italic*
- **bold**

JText



JTree

Container (1)

- A **container** is an object that contains other components:
  - Components added into a container are tracked in a list.
  - The order of the list will define the components' front-to-back staking order within the container.
  - It is possible to change the index of a component within this list, as it is possible to delete a component from a container.
- There are two types of containers:
  - **`Panel`** (rectangle with components)
  - **`Window`**

Container (2)

- Methods:
  - `Component add(Component comp)`
    appends the specified component to the end of this container
  - `Component add(Component comp, int index)`
    adds the specified component to this container at the given position
  - `void remove(int index)`
    remove the component at the specified index from this container
  - `void remove(Component comp)`
    remove the specified component from the container

Component (1)

- A **component** is an object having a graphical representation that can be displayed on the screen and that can interact with the user
- The boundaries are mandarorily rectangulars (even if the object is a circle):
  - The **dimension** of a component is an object **`Dimension`**, with fields: `int height`, `width`
  - The 4 positions of a component in a container is an object **`Insets`**, with fields: `int bottom`, `left`, `right`, `top`.

```
java.lang.Object
 └── java.awt.Component
       └── java.awt.Container
             └── javax.swing.Component
```

Component (2)

- Each component has its own coordinate system ((0,0) in the left top corner).
  - A **position** is an object **`Point`**, with fields: `int x, y`.

Component (3)

- Changing the position of the component in its container does not change its local coordinates!
- Some methods (227 at all, most inherited from `Container`):
  1. **Position and area**
     - `Dimension getSize(Dimension rv)` [`setSize`]
     - `Dimension getPreferredSize()` [`setPreferredSize`]
     - `Point getLocation(Point rv)` [`setLocation`]
     - `int getHeigth()` [`getWidth`]
     - `int getX()` [`getY`]
     - `Rectangle getBounds()` [`setBounds`]
     - `Insets getInsets(Component comp)`

Component (5)

- `Container getParent()`
  returns the container where the component is inserted

2. Aspect

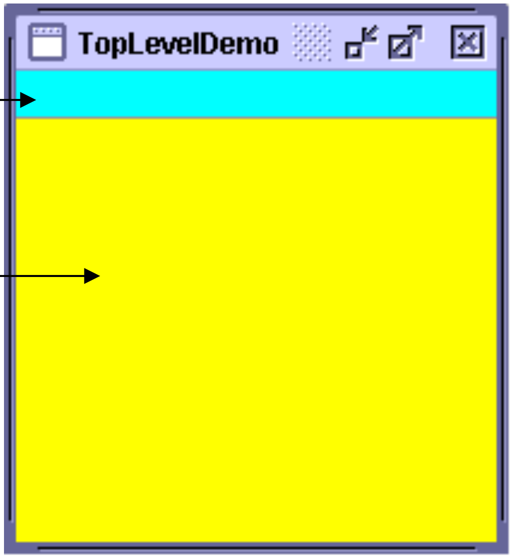
- `void paint(Graphics comp)`
  displays the component in the screen

3. Reaction to events

4. State

`Jframe` class (1)

- The **`JFrame`** container has:
  - Menus (opcional):
    Menus contain buttons that invoke methods when the buttons are cliked.
  - Instance of **`JPanel`**:
    The panel may contain containers, `JFrame`, `JDialog` or `JApplet`, on top of each other.



```
java.lang.Object
  └── java.awt.Component
        └──java.awt.Container
              └── java.awt.Window
                    └── java.awt.Frame
                          └── javax.swing.JFrame
```

`Jframe` class (2)

- **`Jframe` construtor** parameters:
  - **`String title`**: title of the window.
  - **`GraphicsConfiguration gc`**: window configuration.

    • fields that depend on the adopted graphical system (X11, MS Windows, …).

    • by default, the initial frame is invisible.
- Fields:
  - **`int EXIT_ON_CLOSE`**: action to perform on the close.

`Jframe` class (3)

- Methods:
  - **`void setVisible(boolean b)`**
    turns the window visible (`true`) or invisible (`false`)
  - **`setDefaultCloseOperation(int operation)`**
    defines the action to perform when the close button is clicked on the window (usually, `JFrame.EXIT_ON_CLOSE`)
  - **`setSize(int width,int height)`**
    fixes the dimension of the frame
  - **`pack()`**
    causes the window to be sized to fit the preferred size and layouts of its subcomponents.
  - **`Container getContentPane()`**
    returns the panel
  - **`void setContentPane(Container pane)`**
    defines the panel

`Jframe` class (4)

- In this slides only basic aspects of insertion in a `JPanel` will be considered:
  - Texts (`JLabel` objects)
  - Graphics 2D (`Graphics2D` objects)
  - Buttons (`JButton` objects)

(Swing is much more powerful, *Java Swing* manual of O'Reilly has 1200+ pages!)

Visualization (1)

- Texts and images are visualized by instances of `JLabel`.
- Parameters of **`Jlabel constructor`**:
  - **`String text`**: text to visualize
  - **`Icon image`**: image to visualize
  - **`int horizontalAlignment`**: alignment

```
java.lang.Object
 └── java.awt.Component
      └──java.awt.Container
           └── javax.swing.JComponent
                └── javax.swing.JLabel
```

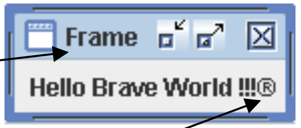
- SwingConstants.LEFT
- SwingConstants.CENTER (image default)
- SwingConstants.RIGHT
- SwingConstants.LEADING (text default)
- SwingConstants.TRAILING

Visualization (2)

- Methods:
  - `void setText(String text)`
    defines the single line of text this component will display.
  - `void setHorizontalAlignment(int align)`
    sets the alignment of the label’s content along the X axis.

Visualization (3)

```
import java.awt.*;
import javax.swing.*;
public static void main(String[] args) {
      // build frame
      JFrame.setDefaultLookAndFeelDecorated(true);
      JFrame frame = new JFrame("Frame");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // build painel with text
      JPanel p = new JPanel();
      p.add(new JLabel("Hello Brave World !!!\u00ae "));

      frame.setContentPane(p);
      frame.pack();
      frame.setVisible(true);
}
```

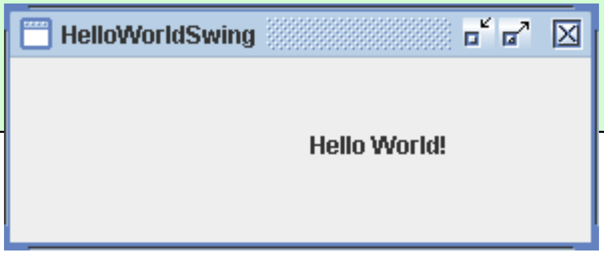


Visualization (4)

- Repositioning a graphical object can be performed in the following steps:
  1. Get the dimension of the object (`gePreferredSize`)
  2. Move the object (`setBounds`)
  3. Change the dimension of the frame (`setSize`)
  4. Turns the frame visible (`setVisible`)

Visualization (5)

```
// *** imprime mensagem deslocada
public static void main(String[] args) {
     JFrame.setDefaultLookAndFeelDecorated(true);
     JFrame frame = new JFrame("HelloWorldSwing");
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

     Container pane = frame.getContentPane();
     pane.setLayout(null);

     JLabel b1 = new JLabel("Hello World");
     pane.add(b1);

     // *** moves the text
     Dimension size = b1.getPreferredSize();
     b1.setBounds(150,25,size.width,size.height);
     // *** changes the dimension of the window
     frame.setSize(300,100);
     frame.setVisible(true);
}
```



Graphics (1)

- Graphic capabilities of `Graphics2D`:
  - Texts
  - Lines, rectangles, poligns, ovals
  - Colors, fonts and area fill

```
java.lang.Object
  └── java.awt.Graphics
         └──java.awt.Graphics2D
```

Graphics (2)

- Grafics 2D are paint, in a extension of `JPanel` class, in method:
  - **`paintComponent(Graphics g)`**
- The `paintComponent` method is never invoked directly.
- All objects are inserted in fixed coordinates: if the windiw is redimensioned to a smaller size, some of the components may disappear or may be cut.

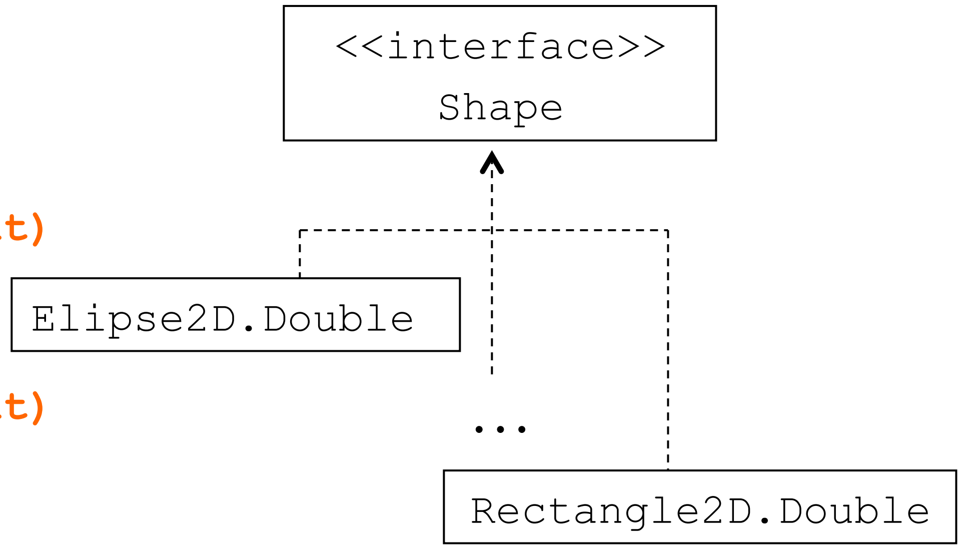
```
public class DrawingPanel extends JPanel {
      public void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D) g;
            super.paintComponent(g);
            // insertion in g of graphical objects
            // ...
      }
}
```

Graphics (3)

- Methods:
  - `drawLine(int x1, int y1, int x2, int y2)`
    draws a line between (x1,y1) and (x2,y2)
  - `drawOval(int x, int y, int width, int height)`
    draws an oval
  - `drawPolygon(int[] xPoints, int[] yPoints, int nPoints)`
    draws a closed polign, defined by arrays of coordinates
  - `drawRect(int x, int y, int width, int height)`
    draws a rectangle
  - `drawString(String str, int x, int y)`
    draws a string

Graphics (4)

- The **`Shape`** interface determines basic definitions of geometric objects. Implementing classes are found in the **`java.awt.geom`** package.
  - **`Elipse2D.Double( double x, double y, double width, double height)`**
  - **`Rectangle2D.Double( double  x, double y, double width, double height)`**
  - **`RoundRectangle2D.Double( double  x, double y, double width, double height)`**



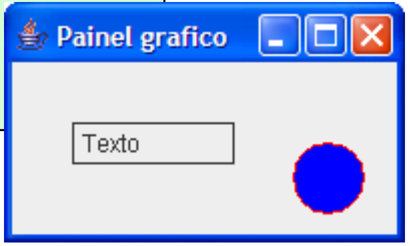
Graphics (5)

- The `Color` class defines fields for main colors (`black`, `blue`, `green`, `red`,...).
- Colors are changes through the method:
  - `setPaint(Color c)`
- The `Graphics2D` methods to change colors are:
  - `draw(Shape s)`: shape of the object
  - `fill(Shape s)`: filling of the object

```
java.lang.Object
  └── java.awt.Color
```

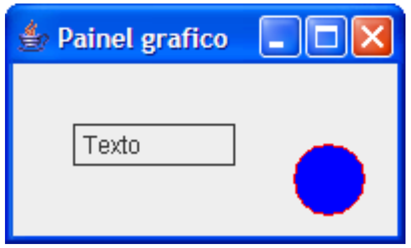
Graphics (6)

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
public class DrawingPanel extends JPanel {
     public void paintComponent(Graphics g) {
          Graphics2D g2 = (Graphics2D)g;
          super.paintComponent(g);
          //*** inserts rectangle with text
          g2.drawRect(30,30,80,20);
          g2.drawString("Texto",35,45);
          //*** inserts blue circle
          Ellipse2D circ = new Ellipse2D.Double(140,40,35,35);
          g2.setPaint(Color.blue);
          g2.fill(circ);
          g2.setPaint(Color.red);
          g2.draw(circ);
     }
}
```



Graphics (7)

```
public static void main(String[] args) {
      JFrame f = new JFrame("Painel grafico");
      f.setSize(200,120);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.getContentPane().add(new DrawingPanel());
      f.setVisible(true);
}
```



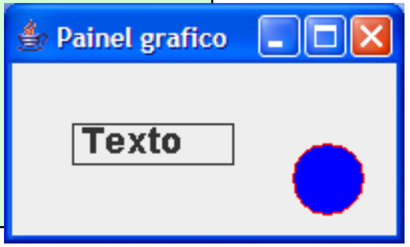
Graphics (8)

- Fonts are objects of type **`Font`** and its **constructor** has 3 parameters:
  - **`String name`** (e.g., “SansSerif”)
  - **`int style`** (fields of `Font`, e.g., `Font.PLAIN` and `Font.BOLD`)
  - **`int size`** (font size)

```
java.lang.Object
  └── java.awt.Font
```

Graphics (9)

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
public class DrawingPanel extends JPanel {
     public void paintComponent(Graphics g) {
          Graphics2D g2 = (Graphics2D)g;
          super.paintComponent(g);
          //*** inserts a rectangle with text with desired font
          g2.drawRect(30,30,80,20);
          Font f = new Font("SansSerif",Font.BOLD,18);
          g2.setFont(f);
          g2.drawString("Texto",35,45);
          //*** insert blue circle
          Ellipse2D circ = new Ellipse2D.Double(140,40,35,35);
          g2.setPaint(Color.blue);
          g2.fill(circ);
          g2.setPaint(Color.red);
          g2.draw(circ);
     }
}
```



Graphics' layout (1)

- Swing provides diverse ways to lay out containers:
  - The `LayoutManager` interface defines basic operations to lay out containers.
  - Swing offers 7 implementations of this interface:
    1. `BorderLayout`: [Window default] arrange and resize its components to fit five regions: north (PAGE_START), south (PAGE_END) and three other in the middle corresponding to west, center and east (LINE_START, CENTER, LINE_END).
    2. `BoxLayout`: allows multiple components to be laid out either vertically or horizontally.

Graphics’ layout (2)

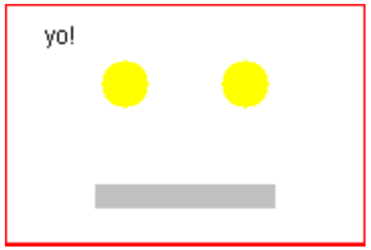
3. `CardLayout`: it treats each component as a card: only one card is visible at a time, and the container acts as a stack of cards.
4. `FlowLayout`: [Panel default] arranges components in a directional flow, much like lines of text in a paragraph; the flow direction is determined by a containers’ field.
5. `GridBagLayout`: aligns component vertically, horizontally or along their baseline without requiring that the components be of the same size.
6. `GridLayout`: lays out a container's components in a rectangular grid.
7. `SpringLayout`: lays out the children of its associated container according to a set of constarints.

`BoxLayout` (1)

- Parameters of **`BoxLayout` constructor**:
  - **`Container target`**
  - **`int axis`**: lay out components along the given axis
    - **`BoxLayout.Y_AXIS`** – components are laid out horizontally, from left to right
    - **`BoxLayout.X_AXIS`** – components are laid out vertically, from top to bottom

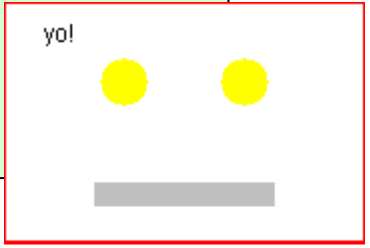
BoxLayout (2)

```
import java.awt.*;
import javax.swing.*;
public class MyComponent extends JComponent {
     public void paintComponent(Graphics g) {
          super.paintComponent(g);

          // *** get dimensions
          int width = getWidth();
          int height = getHeight();

          // *** draw red rectangle
          g.setColor(Color.red);
          g.drawRect(0, 0, width-1, height-1);

          // *** draw yo!
          g.setColor(Color.black);
          g.drawString("yo!",20,20);
```



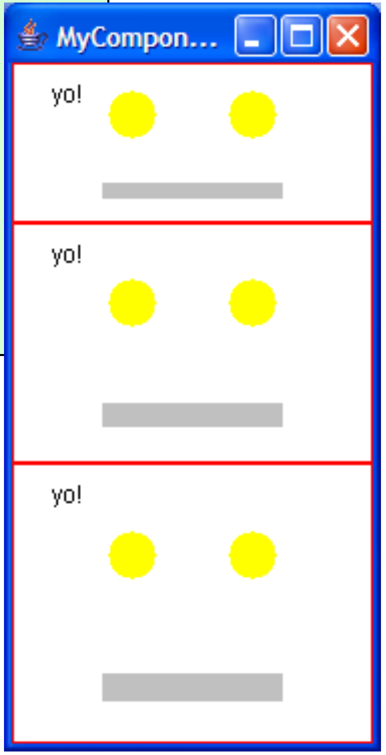
BoxLayout (3)

```
          // *** draw eyes
          int eyeY = height/3;
          int left = width/3;
          int right = 2*width/3;
          int radius = width/15;
          g.setColor(Color.yellow);
          g.fillOval(left-radius, eyeY-radius, radius*2, radius*2);
          g.fillOval(right-radius, eyeY-radius, radius*2, radius*2);

          // *** draw mouse
          g.setColor(Color.lightGray);
          g.fillRect(width/4,3*height/4, width/2, height/10);
     }

     public MyComponent(int width, int height) {
          super();
          setPreferredSize(new Dimension(width,height));
     }
}
```



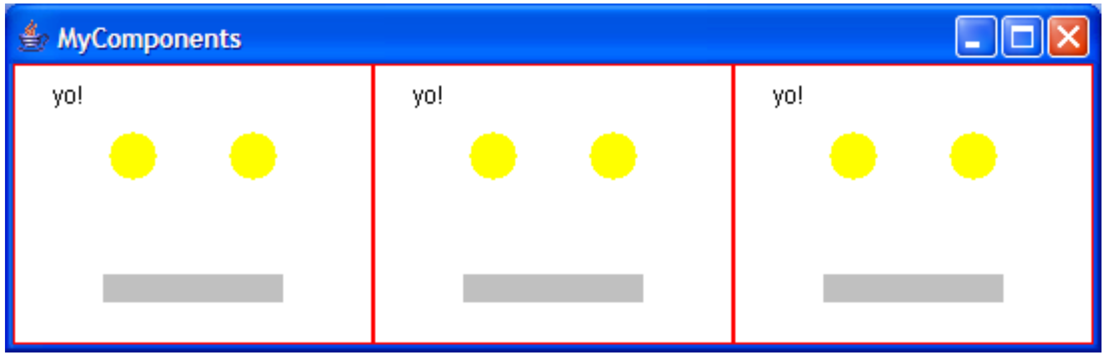
BoxLayout (4)

```
public static void main(String[] args) {
      JFrame frame = new JFrame("MyComponents");
      JComponent content = (JComponent) frame.getContentPane();
      content.setBackground(Color.white);
      content.setLayout(new BoxLayout(content,BoxLayout.Y_AXIS));
      content.add(new MyComponent(180,80));
      content.add(new MyComponent(180,120));
      content.add(new MyComponent(180,140));
      frame.pack();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
}
```



BoxLayout (5)

```
public static void main(String[] args) {
      JFrame frame = new JFrame("MyComponents");
      JComponent content = (JComponent) frame.getContentPane();
      content.setBackground(Color.white);
      content.setLayout(new BoxLayout(content,BoxLayout.X_AXIS));
      content.add(new MyComponent(180,80));
      content.add(new MyComponent(180,120));
      content.add(new MyComponent(180,140));
      frame.pack();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
}
```

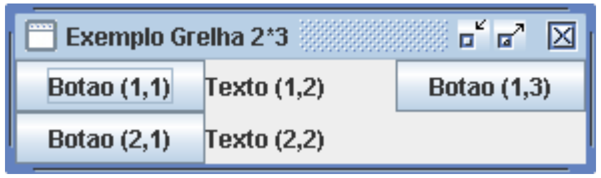


`GridLayout` (1)

- Parameters of **`GridLayout` constructor**:
  - **`int rows`**: number of lines (1st parameter)
  - **`int columns`**: number of columns (2nd parameter)
  - **`int hgap`**: horizontal gap between components (3rd parameter)
  - **`int vgap`**: vertical gap between components (4th parameter)

GridLayout (2)

```
public static void main(String[] args) {
      JFrame.setDefaultLookAndFeelDecorated(true);
      JFrame frame = new JFrame("Exemplo Grelha 2*3");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JPanel pane = new JPanel();
      pane.setLayout(new GridLayout(2,3));
      pane.add(new JButton("Botao (1,1)"));
      pane.add(new JLabel("Texto (1,2)"));
      pane.add(new JButton("Botao (1,3)"));
      pane.add(new JButton("Botao (2,1)"));
      pane.add(new JLabel("Texto (2,2)"));

      frame.setContentPane(pane);
      frame.pack();
      frame.setVisible(true);
}
```



Events (1)

- An **event** is triggered by an external device (e.g.: click in the mouse button or keyboard).
- **When an event is triggered a specific method is invoked; the programmer must indicate the listener object that is waiting for the event.**
- Classes that deal with event listeners should import
  ```
  import java.awt.event.*;
  ```

Events (2)

- Swing defines 10 types of events, depending on the event triggered:
  - **`ActionEvent`** `(components: Button, List, MenuItem)`
  - `AdjustmentEvent (component: Scrollbar)`
  - `ComponentEvent (components: Choice, Component)`
  - `ContainerEvent (component: Container)`
  - `FocusEvent (component: Component)`
  - `ItemEvent (components: CheckBox, List)`
  - `KeyEvent (component: Component)`
  - `MouseEvent (component: Component)`
  - `TextEvent (component: TextComponent)`
  - `WindowEvent (component: Window)`

Events (3)

- The listener object should implement an interface, that depends on the triggered event.

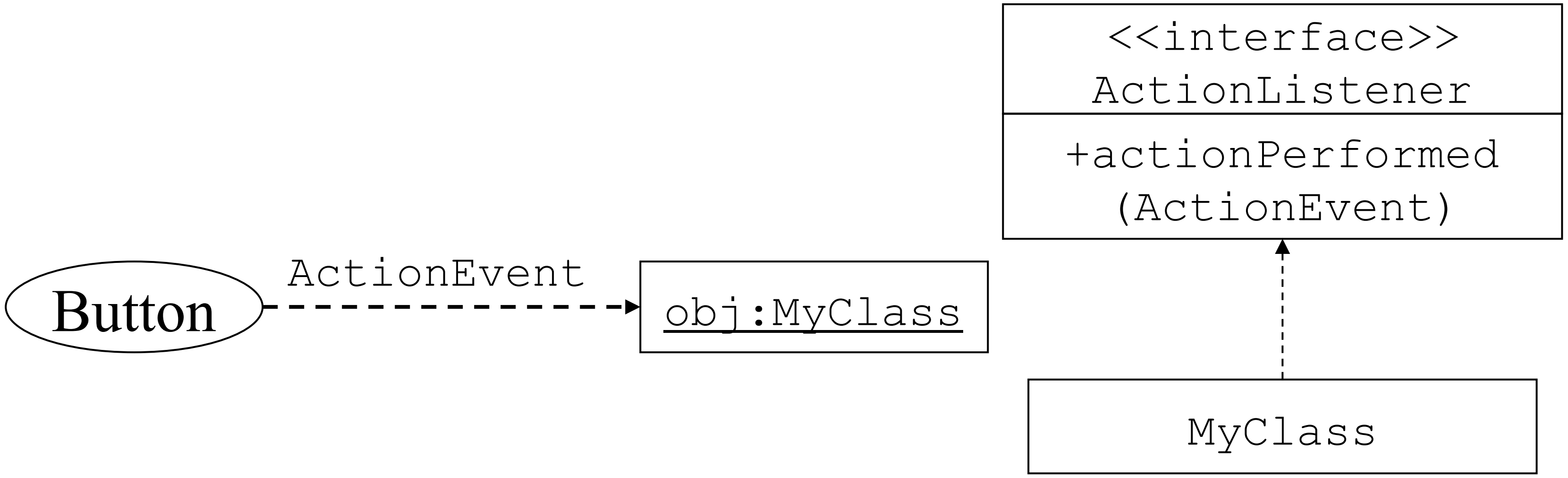
Event	Interface	Methods
`ActionEvent`	**`ActionListener`**	**`actionPerformed()`**
`AdjustmentEvent`	`AdjustmentListener`	`adjustmentValueChanged()`

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

`ContainerEvent`	`ContainerListener`	`componentAdded()`
		`componentRemoved()`

Events (4)

- A button is an instance of `JButton`.
- In AWT/Swing, the class implements the interface `ActionListener`, and the `actionPerformed` method processes the event.



Events (5)

- **Dealing with an `ActionEvent` in Swing:**
  1. **Implement the `ActionListener` interface**

     ```
     class MyClass implements ActionListener {
     ```

  2. **Inside the class define the method**

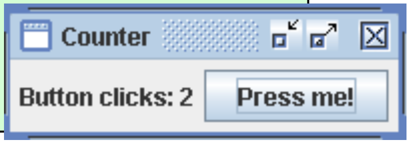
     ```
               void actionPerformed(ActionEvent e) {
                         // código de reacção ao evento
               }
     }
     ```

  3. **Add, in the button, the object that processes the event**

     ```
     void addActionListener(ActionListener l)
     ```

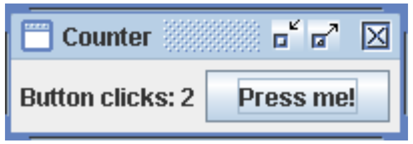
Events (6)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class SwingApplication {
     private int numClicks = 0;
     private JLabel label = new JLabel("Button clicks: " + numClicks);
     private JButton button = new JButton("Press me!");

     public SwingApplication(JPanel jp) {
          jp.add(label);
          jp.add(button);
          button.addActionListener(new Handler());
     }

     private class Handler implements ActionListener {
          public void actionPerformed(ActionEvent e) {
               numClicks++;
               label.setText("Button clicks: " + numClicks);
          }
     }
}
```



Events (7)

```
import javax.swing.*;
import java.awt.*;
public class Main {
     public static void main(String[] args) {
          JFrame.setDefaultLookAndFeelDecorated(true);
          JFrame frame = new JFrame("Counter");
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

          JPanel p = new JPanel();
          frame.setContentPane(p);
          SwingApplication app = new SwingApplication(p);
          frame.pack();
          frame.setVisible(true);
     }
}
```


Counter
Button clicks: 2
Press me!