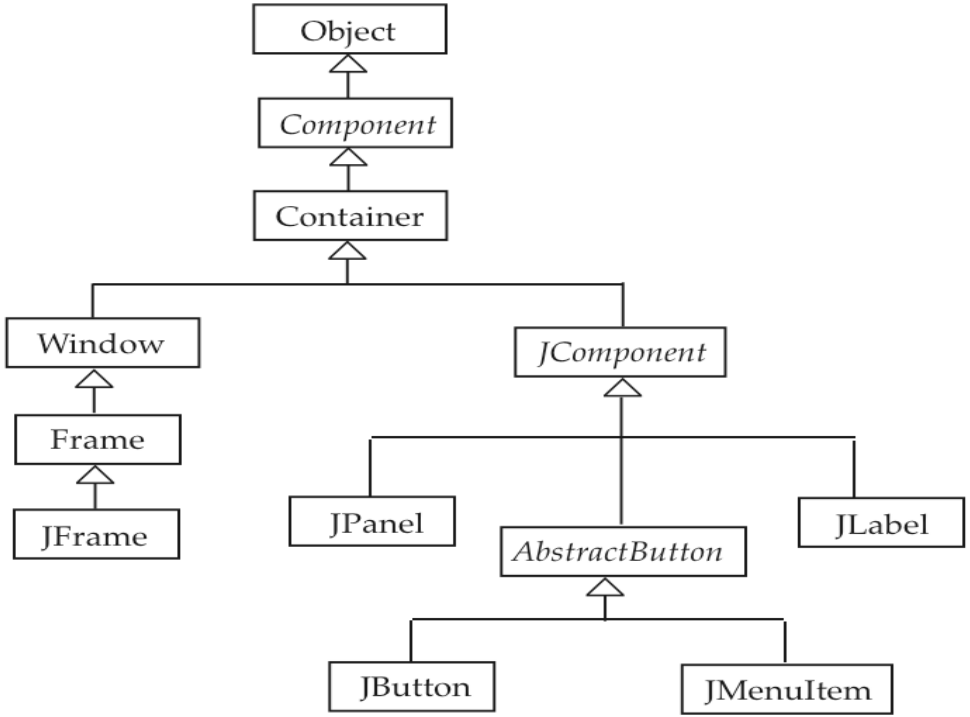
Advanced Programming

Graphic User Interface in Java

Introduction

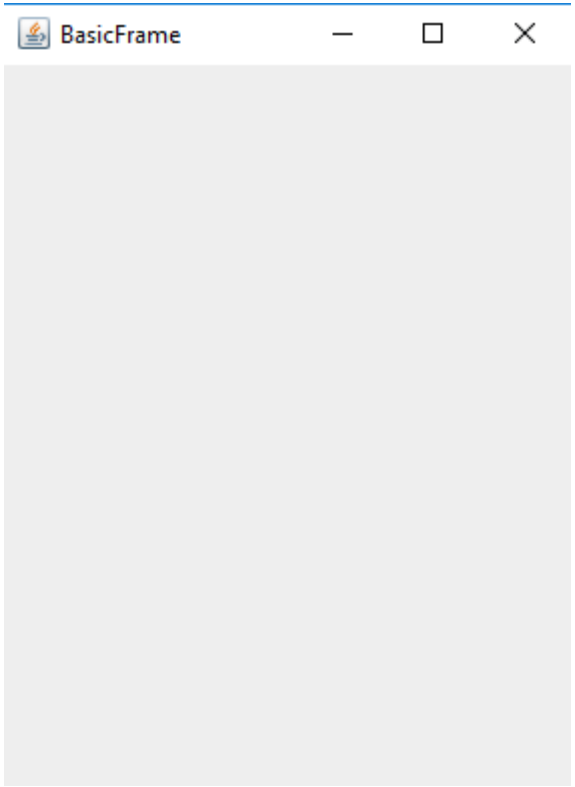
- Java 1.0: Contained AWT (Abstract Window Toolkit) class library for basic GUI programming.
- SWING: GUI Toolkit available from Java 1.1
- Swing has a rich and convenient set of user interface elements.
- Swing has few dependencies on the underlying platform; it is therefore less prone to platform-specific bugs.
- Swing gives a consistent user experience across platforms.

Inheritance Hierarchy for Frame and Component Classes



Create a Frame

- Top-level Window
- JFrame Class in Swing (extends Frame Class)
- Not painted on a canvas



```
import javax.swing.*;

public class SizedFrameTest
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();

        frame.setTitle("BasicFrame");

        frame.setSize(300,400);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setVisible(true);
     }
  }
```

Adding Information in the Frame

- Frames are designed to be containers for components
- Normally draw on another component added to the frame
- When designing a frame, you add components into the content pane

```
Container contentPane = frame.getContentPane();
Component c = . . .;
contentPane.add(c);
```

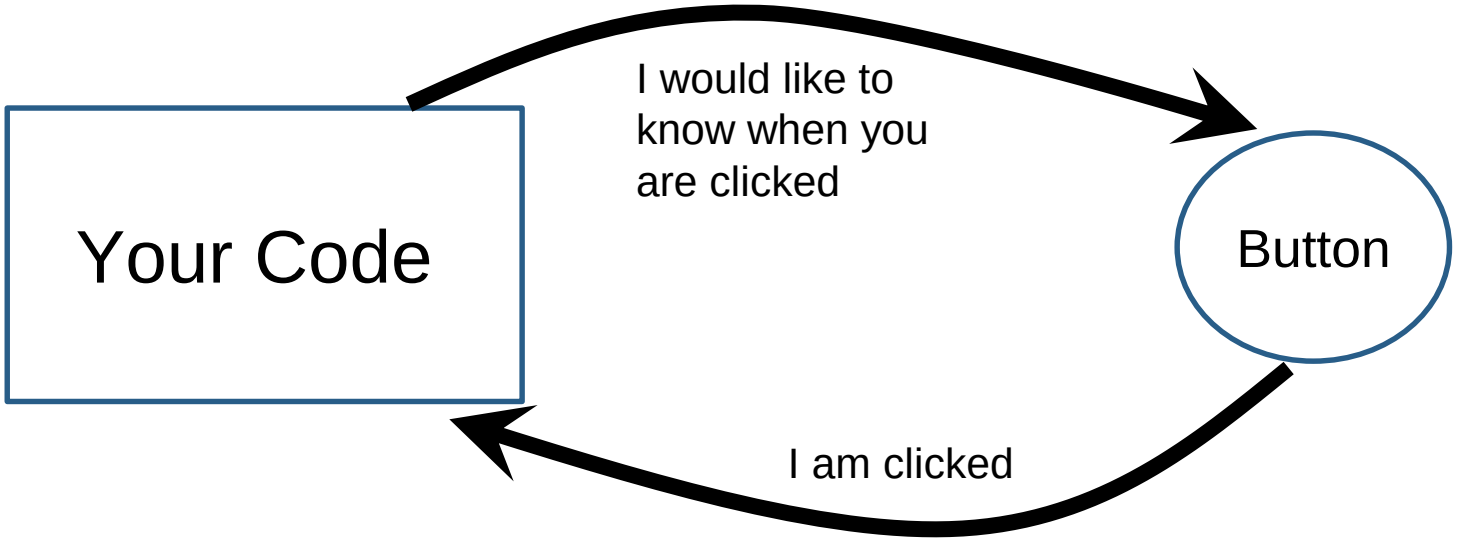
```
import javax.swing.*;

public class SimpleGui1 {

  public static void main (String[] args) {

    JFrame frame = new JFrame();

    JButton button = new JButton("click me");

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.getContentPane().add(button);

    frame.setSize(300,300);

    frame.setVisible(true);
  }
}
```

Adding Your Own Components

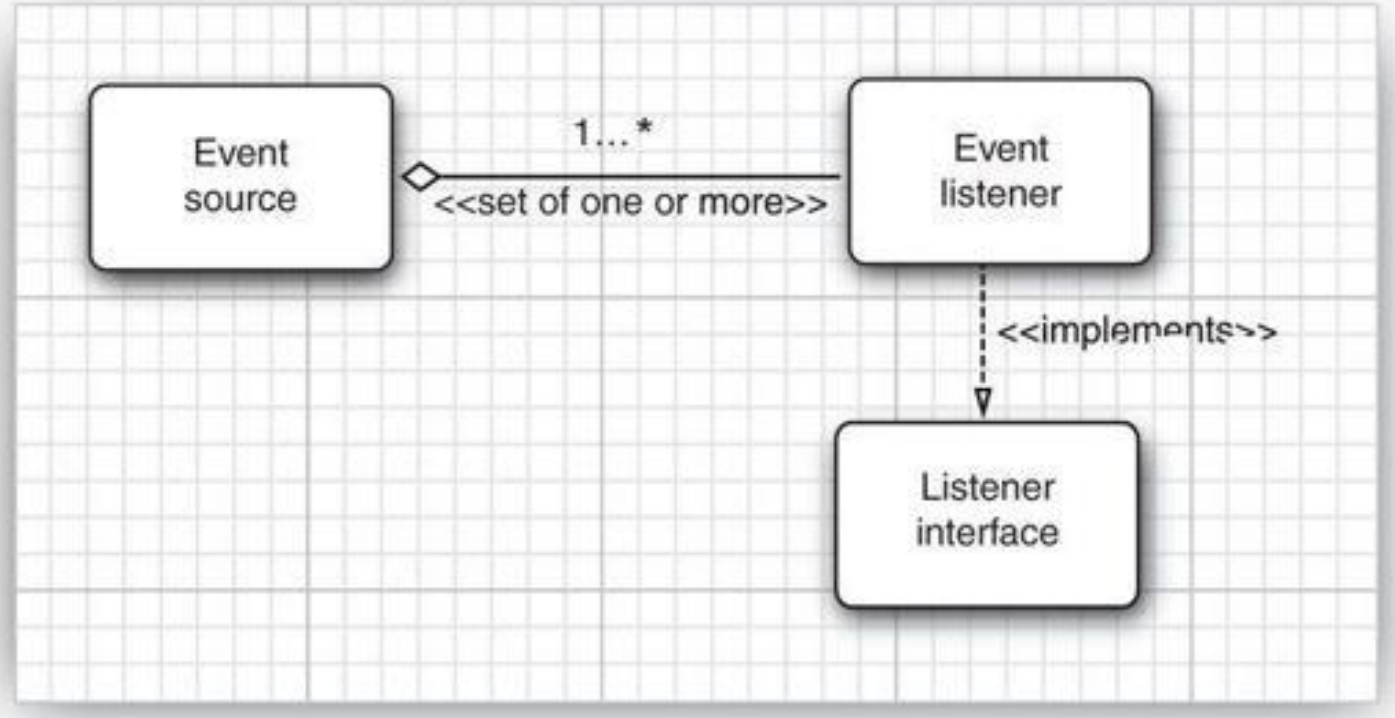
```
public class MyComponent extends JComponent {
  public static final int MESSAGE_X = 75;
  public static final int MESSAGE_Y = 100;
  private static final int DEFAULT_WIDTH = 300;
  private static final int DEFAULT_HEIGHT = 200;

  public void paintComponent(Graphics g) {
    g.drawString("This is the text from my component", MESSAGE_X, MESSAGE_Y);
  }

  public Dimension getPreferredSize() {
    return new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT);
  }
}
```

Event Handling

Any operating environment that supports GUIs constantly monitors events such as keystrokes or mouse clicks



Event Handling



Event Handling

- A listener object is an instance of a class that implements a special interface called (naturally enough) a listener interface.
- An event source is an object that can register listener objects and send them event objects.
- The event source sends out event objects to all registered listeners when that event occurs.
- The listener objects will then use the information in the event object to determine their reaction to the event

Event Handling

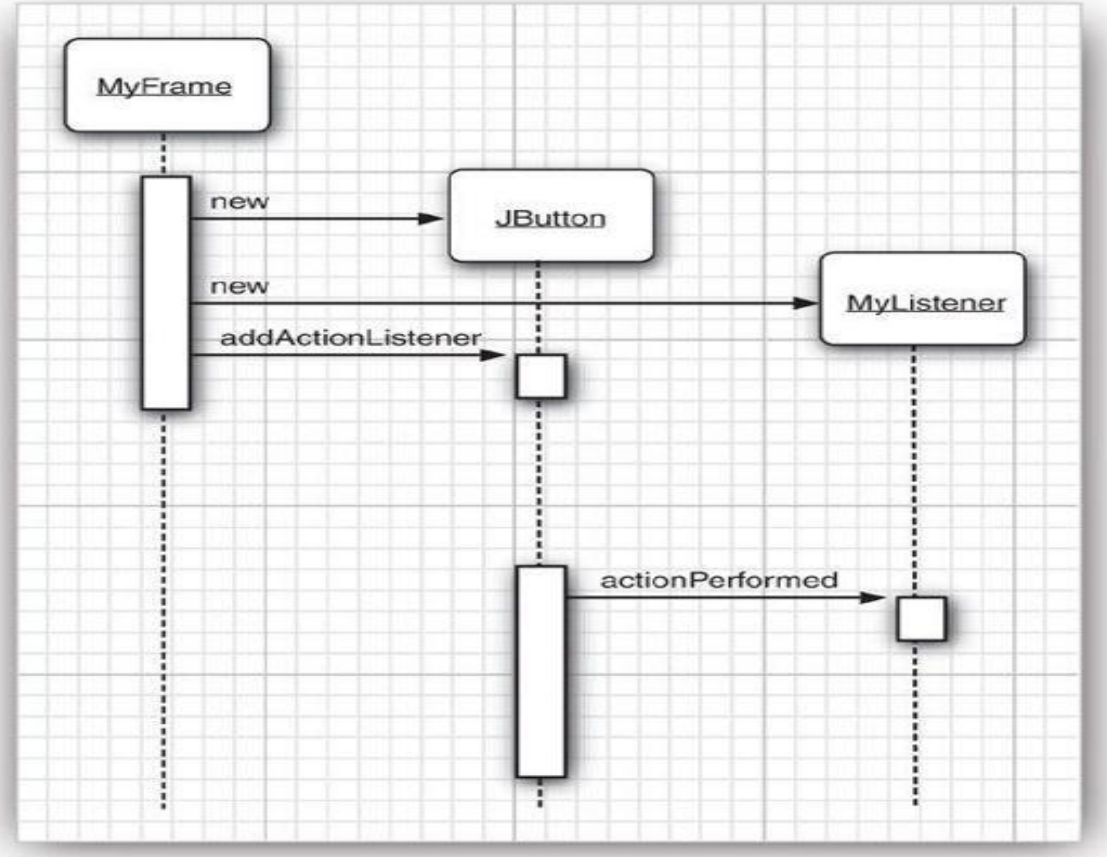
- Specify Listener

```
ActionListener listener = . . .;
JButton button = new JButton("Ok");
button.addActionListener(listener);
```

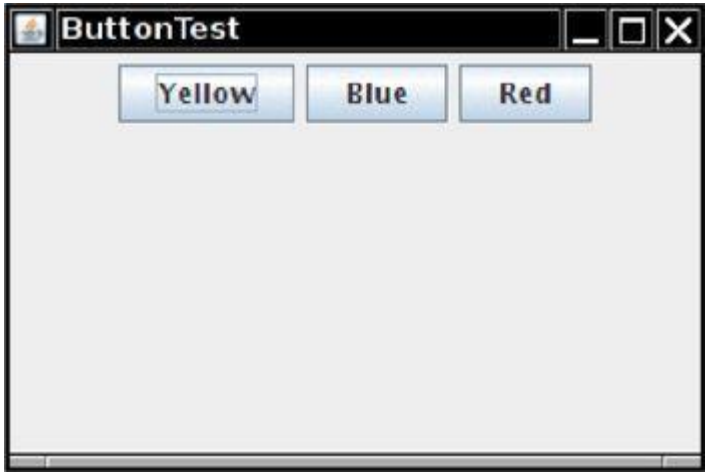
Event Handling

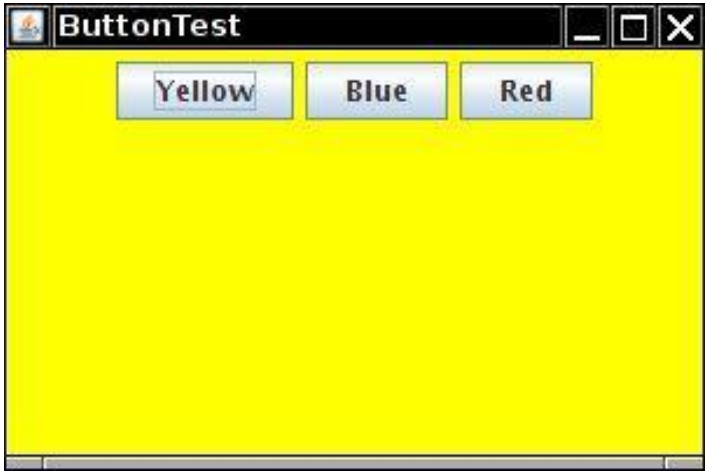
- To implement the ActionListener interface, the listener class must have a method called actionPerformed that receives an ActionEvent object as a parameter.

```
class MyListener implements ActionListener
{
   . . .
   public void actionPerformed(ActionEvent event)
   {
      // reaction to button click goes here
      . . .
   }
}
```


MyFrame
new
JButton
new
MyListener
addActionListener
actionPerformed

Example: Handling a Button Click





```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
 /**
  * A frame with a button panel
  */
 public class ButtonFrame extends JFrame
  {
    private JPanel buttonPanel;
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT =
200;

     public ButtonFrame()
     {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        JButton yellowButton = new
JButton("Yellow");
        JButton blueButton = new JButton("Blue");
        JButton redButton = new JButton("Red");

        buttonPanel = new JPanel();
```

```
// add buttons to panel
buttonPanel.add(yellowButton);
buttonPanel.add(blueButton);
buttonPanel.add(redButton);

// add panel to frame
add(buttonPanel);

// create button actions
ColorAction yellowAction = new
ColorAction(Color.YELLOW);

ColorAction blueAction = new
ColorAction(Color.BLUE);

ColorAction redAction = new
ColorAction(Color.RED);

// associate actions with buttons
yellowButton.addActionListener(yellowAct
ion);
blueButton.addActionListener(blueAction)
;
redButton.addActionListener(redAction);
}
```

```
   /**
    * An action listener that sets the panel's background color.
    */
   private class ColorAction implements ActionListener
   {
      private Color backgroundColor;

      public ColorAction(Color c)
      {
         backgroundColor = c;
      }

      public void actionPerformed(ActionEvent event)
      {
         buttonPanel.setBackground(backgroundColor);
      }
   }
}
```

Alternative

```
public void actionPerformed(ActionEvent e) {

       Color color = display.getBackground();

       int red = color.getRed();
       int green = color.getGreen();
       int blue = color.getBlue();

       if (e.getActionCommand().equals("Red")) {
           if (red == 0) {
               red = 255;
           } else {
               red = 0;
           }
       }
```

```
if (e.getActionCommand().equals("Green")) {
           if (green == 0) {
               green = 255;
           } else {
               green = 0;
           }
       }

if (e.getActionCommand().equals("Blue")) {
           if (blue == 0) {
               blue = 255;
           } else {
               blue = 0;
           }
       }
Color setCol = new Color(red, green, blue);
       display.setBackground(setCol);
   }
```

Problems

- Adding a specific ActionListener looks more Object Oriented but with a lot of hard coding

- If-else code looks very non-OOP

What is the best of both the worlds?

How can ActionListener access the GUI Vars

- Remember inner class!

```
this.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent e) {
    String data = "Username " + userText.getText();
    data += ", Password: " + new
      String(passwordText.getPassword());
    statusLabel.setText(data);
  }
});
```

Other Listener Interfaces

- ActionListener: actionPerformed
- ItemListener: itemStateChanged
- KeyListener: keyPressed, keyReleased, keyTyped

How to make your own GUI

- Put component: Menus, Buttons etc.
- Draw graphics on a component.
- Put an image on a component.

Layout Managers

- Border
- Flow
- Box

BorderLayout

- Five regions: East, West, North, South, Center

```
frame.getContentPane().add(BorderLayout.EAST, button);
```

BorderLayout

- You can add only one component per region to a background controlled by a BorderLayout manager.

- Components laid out by this manager usually don’t get to have their preferred size.

- BorderLayout is the default layout manager for a frame

FlowLayout

- A FlowLayout manager acts kind of like a word processor, except with components instead of words.
- Each component is the size it wants to be, and they're laid out left to right in the order that they're added, with "word-wrap" turned on.
- When a component won't fit horizontally, it drops to the next "line" in the layout.
- FlowLayout is the default layout manager for a panel.

BoxLayout

- Each component gets to have its own size, and the components are placed in the order in which they’re added.
- BoxLayout manager can stack the components vertically (or horizontally, but usually we’re just concerned with vertically).
- You can insert a sort of ‘component return key’ and force the components to start a new line