

Java GUI

K.P. Chow

University of Hong Kong

The First GUI

```
import javax.swing.*;

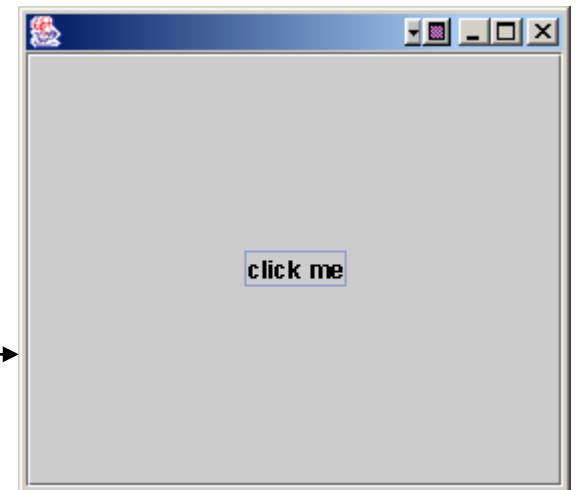
public class SimpleGUI {
    public static void main(String[] args) {
        JFrame jf = new JFrame();
        JButton jb = new JButton("click me");
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.getContentPane().add(jb);
        jf.setSize(300,300);
        jf.setVisible(true);
    }
}
```

Make a frame and a button

Make the program quit
when the window is closed

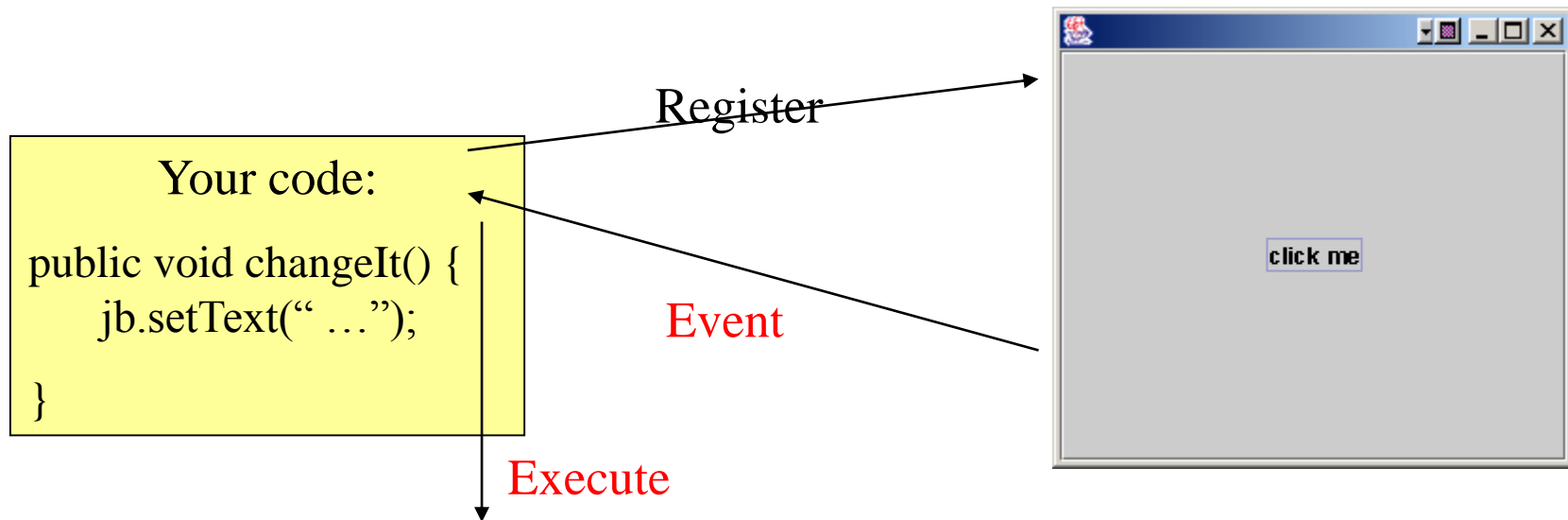
Add the button to the frame's
content pane, not the frame

Finally, display it: a big button



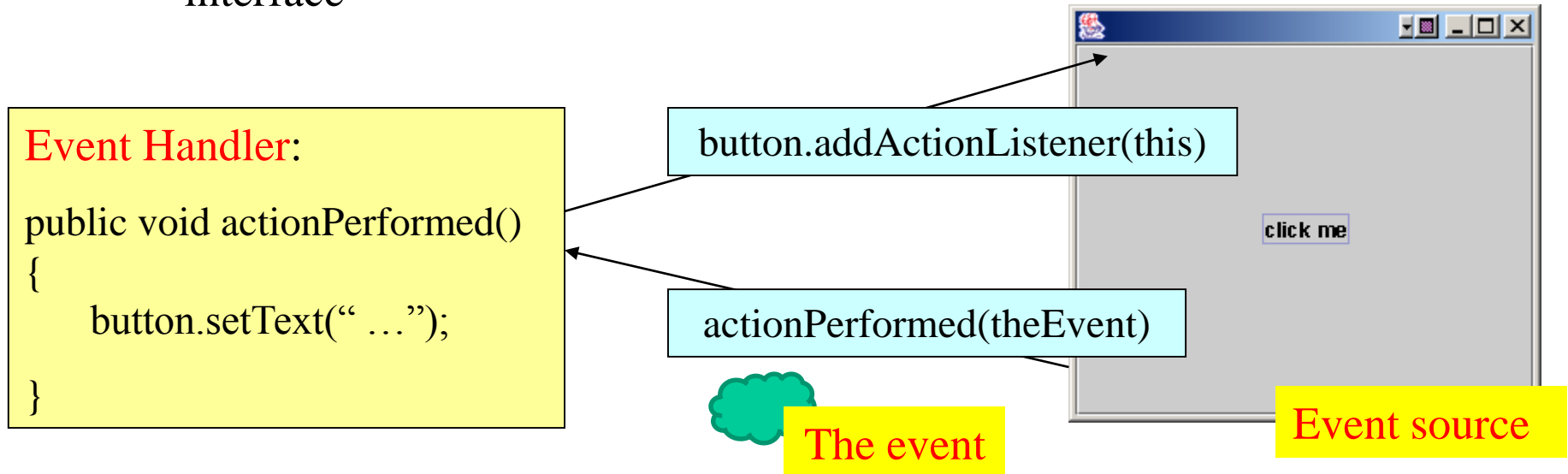
Event Handling

- How do get the button to do something when the user clicks it?
 - A *method* to be called when the user clicks the button
 - A way to know when to trigger that *method*



Listener Interface

- A listener interface is the bridge between the listener (you) and event source (the button)
- If you want to carry out some actions when the user clicks on the button (generate a button event), you need to implement the interface “I’m listening for your events.”
 - An event source creates an event object when the user clicks the button
 - Your code (the event handler) will receive the event and carry out the action
 - Every event type has a matching listener interface, e.g. MouseEvents require MouseListener interface, WindowEvents require WindowListener interface



The First Event Handler

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

public class SimpleGUI1 implements ActionListener {
    JButton jb;
    public static void main(String[] args) {
        SimpleGUI1 gui = new SimpleGUI1();
        gui.go();
    }
    public void go() {
        JFrame jf = new JFrame();
        jb = new JButton("click me");
        jb.addActionListener(this);
        jf.getContentPane().add(jb);
        jf.setSize(300,300);
        jf.setVisible(true);
    }
    public void actionPerformed(ActionEvent ae) {
        jb.setText("I have been clicked");
    }
}
```

Implement the ActionListener interface, which must include the method actionPerformed()

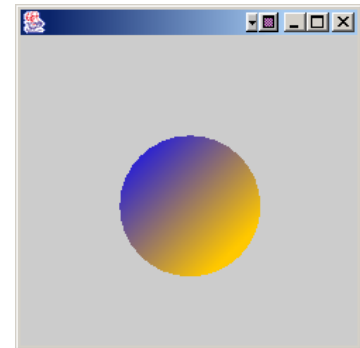
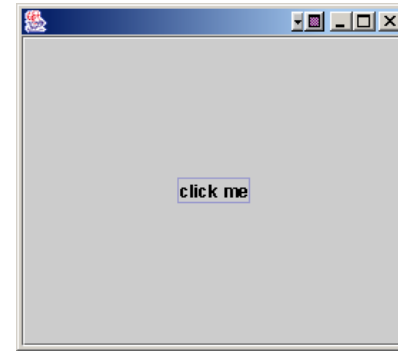
Register the action with the button

Implement the ActionListener interface's actionPerformed method

The button calls this method to let you know an event is triggered

Put things on GUI: 3 ways

1. Put widget on a frame, e.g. add buttons, menus, etc
 - `frame.getContentPane().add(myButton);`
2. Draw 2D graphics on a widget, e.g. use a graphic bject to paint shapes
 - `graphics.fillOval(70,70,100,100);`
3. Put a JPEG on a widget, e.g. put some images on a widget
 - `graphics.drawImage(myPic,10,10,this);`



Paintable Widget : Draw 2D Graphics

- To put graphics on the screen: create a paintable widget
 - Make a subclass of JPanel and override one method: `paintComponent()`
 - All graphics code goes inside the `paintComponent()`
- **PaintComponent** (Graphics g)
 - The method called by the *system*: paint yourself
 - E.g. when the frame holding the drawing panel is displayed, `paintComponent()` will be called, all objects in the panel will be displayed
 - Never need to call the method `paintComponent()` directly: can ask the *system* to refresh the display
 - Argument “Graphics g” is the actual drawing canvas that gets slapped onto the real display: can’t get this yourself, handed to you by the *system*

Graphics

- Put your own graphics on the screen:
 - Create your own paintable widget
 - Put the widget on the frame
 - Display it

```
import java.awt.*;
import javax.swing.*;
class MyDrawPanel extends JPanel {
    public void paintComponent(Graphics g) {
        g.setColor(Color.orange);
        g.fillRect(20,50,100,100);
    }
}
```

Make a subclass of JPanel, a widget that can be added to a frame, and override the paintComponent() method

paintComponent() is called by the *system* when it is time to paint the widget (*you never call this method yourself*)

g is a Graphic object that is the actual drawing surface that mapped to the real display, and is handed to you by the *system*

Draw a rectangle with orange color

Put a JPEG on a Widget

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

public class SimpleGUI {
    public static void main(String[] args) {
        JFrame jf = new JFrame();
        MyDrawPanel p = new MyDrawPanel();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.getContentPane().add(p);
        jf.setSize(300,300);
        jf.setVisible(true);
    }
}

class MyDrawPanel extends JPanel {
    public void paintComponent(Graphics g) {
        Image image = new ImageIcon("yours.jpg").getImage();
        g.drawImage(image,3,4,this);
    }
}
```

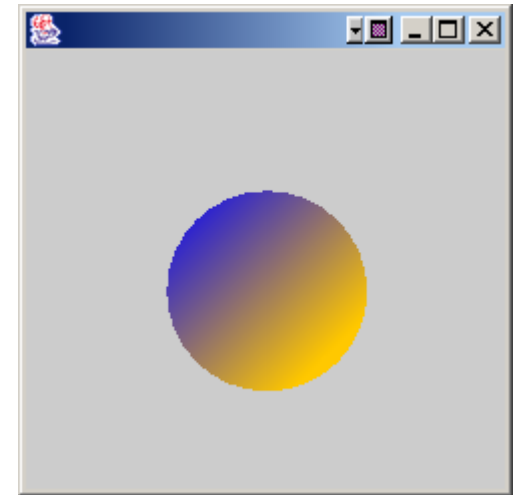


3 pixels from the left edge of the panel (this) and 4 pixels from the top edge of the panel (this)

Graphics2D

- Graphics g: g is actually an instance of the Graphics2D class
- More methods are available in Graphics2D than Graphics, e.g. rotate, scale, transform, ...
- E.g.

```
public void paintComponent(Graphics g) {  
    Graphics2D g2d = (Graphics2D) g;  
    GradientPaint gp = new  
    GradientPaint(70,70,Color.blue,  
                150,150, Color.orange);  
    g2d.setPaint(gp);  
    g2d.fillOval(70,70,100,100);  
}
```



The GradientPaint class provides a way to fill a shape with a linear color pattern: the color on the line connecting (70,70) to (150,150) is changed from blue to orange.

Graphics and Graphics2D

Graphics

- drawImage()
- drawLine()
- drawPolygon()
- drawRect()
- drawOval()
- fillRect()
- fillRoundRect()
- setColor()
- ...

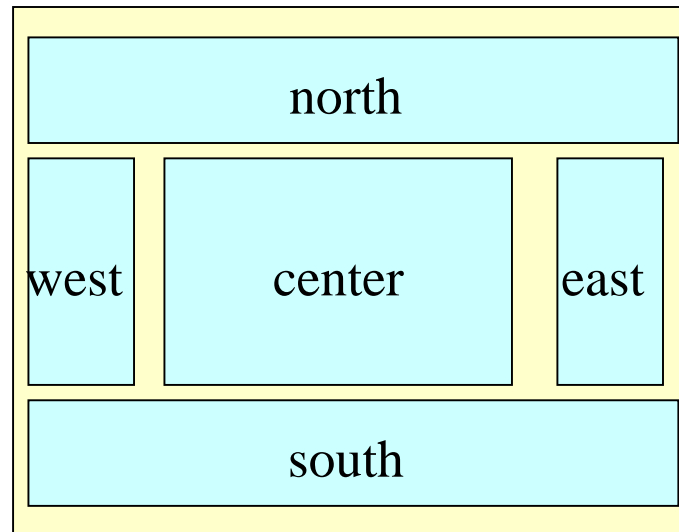
Graphics2D

- fill3DRect()
- draw3DRect()
- rotate()
- scale()
- shear()
- transform()
- setRenderingHints()
- ...

Multiple Widgets on a Frame

- A frame has 5 regions that you can add widget to
- Only one component can be added to each region
- One can add a panel to a region, and multiple widgets can be added to the panel
- One can specify the region to be added:

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



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

public class SimpleGUI3 implements ActionListener {
    JFrame jf;

    public static void main(String[] args) {
        SimpleGUI3 gui = new SimpleGUI3();
        gui.go();
    }

    public void go() {
        jf = new JFrame();
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton jb = new JButton("Change colors");
        jb.addActionListener(this);
        MyDrawPanel dp = new MyDrawPanel();
        jf.getContentPane().add(BorderLayout.SOUTH,jb);
        jf.getContentPane().add(BorderLayout.CENTER,dp);
        jf.setSize(30,300);
        jf.setVisible(true);
    }
}
```

The Circle that Changes Color

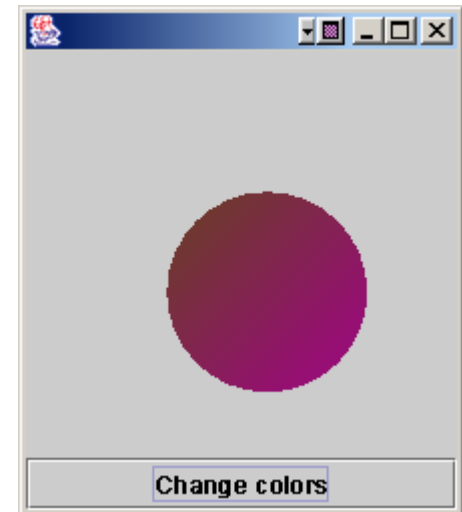
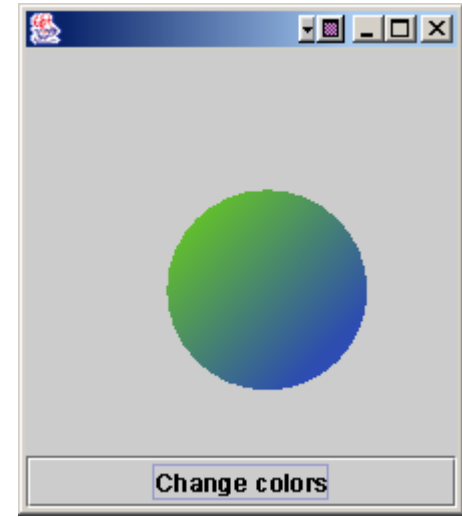
When the button is pressed, the circle's color will be changed.

When the user clicks, the frame will repaint itself → `paintComponent()` is called on every widget in the frame

```
public void actionPerformed(ActionEvent ae) {
    jf.repaint();
}
```

The Circle that Changes Color

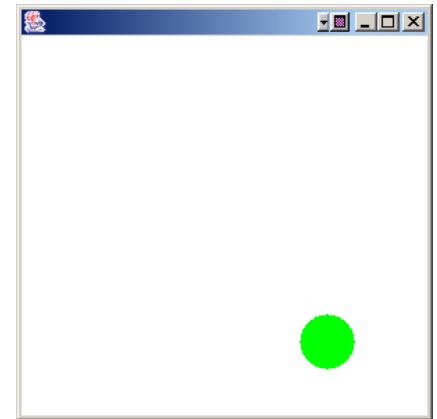
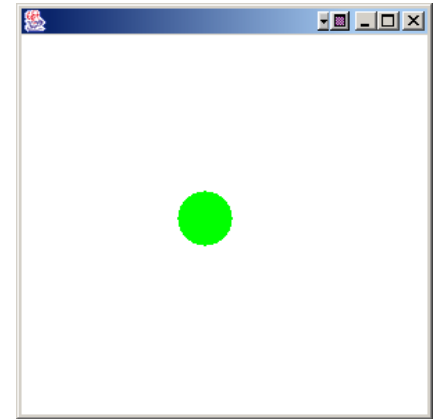
```
public void paintComponent(Graphics g) {  
    Graphics2D g2d = (Graphics2D) g;  
    int red = (int) (Math.random() * 255);  
    int green = (int) (Math.random() * 255);  
    int blue = (int) (Math.random() * 255);  
    Color start = new Color(red,green,blue);  
    int red = (int) (Math.random() * 255);  
    int green = (int) (Math.random() * 255);  
    int blue = (int) (Math.random() * 255);  
    Color end = new Color(red,green,blue);  
  
    GradientPaint gp = new GradientPaint(  
        70,70,start,150,150,end);  
  
    g2d.setPaint(gp);  
    g2d.fillOval(70,70,100,100);  
}
```



Simple Animation

- A moving circle from upper left corner to lower right corner
- Steps:
 1. Paint an object at a particular x and y coordinate
`g.fillOval(20,50,100,100)`
 2. Paint the object at a different x and y coordinate
`g.fillOval(25,55,100,100)`
 3. Repeat the previous step with changing x and y values
- Will use inner class:

```
class MyDrawPanel extends JPanel {  
    public void paintComponent(Graphics g) {  
        g.setColor(Color.white);  
        g.fillOval(x,y,this.getWidth(),this.getHeight());  
        g.setColor(Color.green); g.fillOval(x,y,100,100);  
    }  
}
```



Erase the
previous circle

Simple Animation Code

```
public class SimpleAnimation {
    int x = 70;  int y = 70;

    public static void main(String[] args) {
        SimpleAnimation gui =
            new SimpleAnimation();
        gui.go();
    }

    public void go() {
        JFrame jf = new JFrame();
        jf.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);

        MyDrawPanel dp =
            new MyDrawPanel();
        jf.getContentPane().add(dp);

        jf.setSize(300,300);
        jf.setVisible(true);
    }
}
```

The real animation code

```
for ( int i=0; i<130; i++ )
{
    x++;
    y++;

    dp.repaint();

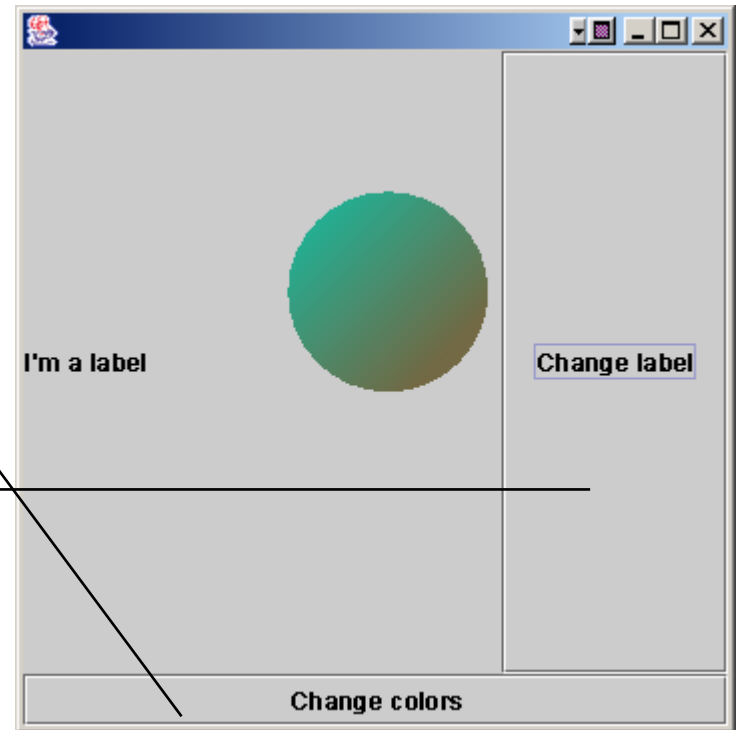
    try {
        Thread.sleep(50);
    } catch (Exception ex) { }
}
}
```


- How do we get actions for 2 different buttons, when each button needs to do something different?

Two Buttons

class MyGUI implements ActionListener

```
{  
    public void actionPerformed  
        (ActionEvent ae) {  
        frame.repaint();  
    }  
  
    public void actionPerformed  
        (ActionEvent ae) {  
        label.setLabel("Label changed");  
    }  
}
```

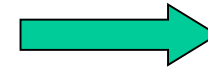


Doesn't Work, why?

2 ActionListeners for 2 Buttons

```
class MyGUI {  
    JFrame frame;  
    JLabel label;  
    ...  
}
```

```
class ColorButtonListener implements ActionListener {  
    public void actionPerformed(ActionEvent ae) {  
        frame.repaint();  
    }  
}
```



Deson't work: frame
is in MyGUI class

```
class LabelButtonListener implements ActionListener {  
    public void actionPerformed(ActionEvent ae) {  
        label.setLabel("Label changed.");  
    }  
}
```



Deson't work: label
is in MyGUI class

Inner Class

```
class OuterClass {  
    private int x;  
  
    class InnerClass {  
        void go () {  
            x = 42;  
        }  
    }  
}
```

- An inner class can use all the methods and variables of the outer class, even the private ones
- The inner class gets to use those variables and methods just as if the methods and variables were declared within the inner class
- The above properties apply to instance of the inner class accessing *something* in an instance of the outer class
 - An inner object must be tied to a specific outer object

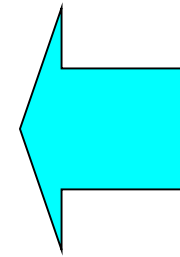
How to make an instance of an inner class?

```
class Outer {  
    private int x;  
  
    Inner in = new Inner();  
  
    public void doThis() {  
        in.go();  
    }  
  
    class Inner {  
        void go () { ... }  
    }  
}
```

- Instantiate an inner class from code within an outer class
- The instance of the outer class is the one that the inner object will “bond” with
- E.g. when a new Outer object is created:
Outer out = new Outer();
 - A new Inner object is created and is “bond” with the Outer object out

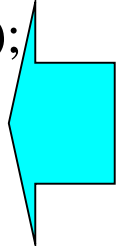
2-button Code

```
public class TwoButtons {  
    JFrame jf;  
    JLabel jl;  
    public static void main(String[] args) {  
        TwoButtons gui = new TwoButtons();  
        gui.go();  
    }  
    public void go() {  
        jf = new JFrame();  
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        JButton jlb = new JButton("Change Label");  
        jlb.addActionListener(new LabelListener());  
        JButton jcb = new JButton("Change Circle");  
        jcb.addActionListener(new ColorListener());  
        jl = new JLabel("I'm a label");  
        MyDrawPanel dp = new MyDrawPanel();
```



Create the
widgets and
add the
listeners

```
jf.getContentPane().add(BorderLayout.SOUTH,jcb);  
jf.getContentPane().add(BorderLayout.CENTER,dp);  
jf.getContentPane().add(BorderLayout.EAST,jlb);  
jf.getContentPane().add(BorderLayout.WEST,jl);
```



Place the
widgets

```
    jf.setSize(300,300);  
    jf.setVisible(true);  
}
```

2-button Code

```
class LabelListener implements ActionListener {  
    public void actionPerformed(ActionEvent ae) {  
        jl.setText("Label changed");  
    }  
}
```



Label
listener

```
class ColorListener implements ActionListener {  
    public void actionPerformed(ActionEvent ae) {  
        jf.repaint();  
    }  
}
```



Color
listener

MORE EVENT HANDLER

Java Events

- java.util.EventObject (implements java.io.Serializable)
 - java.awt.AWTEvent ←
 - java.awt.event.ActionEvent
 - java.awt.event.AdjustmentEvent
 - java.awt.event.ComponentEvent
 - java.awt.event.ContainerEvent
 - java.awt.event.FocusEvent
 - java.awt.event.InputEvent
 - » java.awt.event.KeyEvent
 - » java.awt.event.MouseEvent
 - » java.awt.event.MouseWheelEvent
 - java.awt.event.PaintEvent
 - java.awt.event.WindowEvent
 - java.awt.event.HierarchyEvent
 - java.awt.event.InputMethodEvent
 - java.awt.event.InvocationEvent
 - java.awt.event.ItemEvent
 - java.awt.event.TextEvent

Contains information about how it is consumed

Events for individual controls, each contains event specific information, e.g. the exact key pressed, where is the mouse clicked,

Event Handler (1)

General Category	Event that it generates	Interfaces that the event-handler implements
Mouse	Dragging, moving mouse causes a MouseEvent	MouseMotionListener
	Click, selecting, releasing causes a MouseEvent	MouseListener
Keyboard	Key press or key release causes a KeyEvent	KeyListener
Selecting (an item from a list, checkbox, etc)	When item is selected causes an ItemEvent	ItemListener
Text input controls	When newline is entered causes a TextEvent	TextListener
Scrolling controls	When a scrollbar slider is moved causes an AdjustmentEvent	AdjustmentListener

Event Handler (2)

General Category	Event that it generates	Interfaces that the event-handler implements
Other controls (button, menu, etc)	When pressed causes an <code>ActionEvent</code>	<code>ActionListener</code>
Window changes	Open, close, iconify, etc., causes a <code>WindowEvent</code>	<code>WindowListener</code>
KeyboardFocus changes	Tabbing to next field or requesting focus causes a <code>FocusEvent</code> . A component must have the focus to generate key events	<code>FocusListener</code>
Component changes	Resizing, hiding, revealing, or moving a component causes a <code>ComponentEvent</code>	<code>ComponentListener</code>
Container changes	Adding or removing a component to a container causes a <code>ContainerEvent</code>	<code>ContainerListener</code>

Implementing WindowListener

```
public class CloseDemo implements WindowListener {  
  
    public static void main(String[] args) {  
        JFrame if = new JFrame();  
        if.setSize(400,100);  
        if.setVisible(true);  
        if.addWindowListener( new CloseDemo());  
    }  
  
    public void windowClosing(WindowEvent e) { System.exit(0); }  
    public void windowClosed(WindowEvent e) { }  
    public void windowIconified(WindowEvent e) { }  
    public void windowDeiconified(WindowEvent e) { }  
    public void windowActivated(WindowEvent e) { }  
    public void windowDeactivated(WindowEvent e) { }  
}
```

Using WindowAdapter

```
public class CloseDemo extends WindowAdapter {  
  
    public static void main(String[] args) {  
        JFrame if = new JFrame();  
        if.setSize(400,100);  
        if.setVisible(true);  
        if.addWindowListener( new CloseDemo());  
    }  
  
    public void windowClosing(WindowEvent e) { System.exit(0); }  
  
    // default methods are defined in WindowAdapter  
}
```

MouseListener

```
public interface java.awt.event.MouseListener extends  
    java.lang.Object implements java.util.EventListener{  
  
    public void mouseClicked(java.awt.event.MouseEvent);  
    public void mousePressed(java.awt.event.MouseEvent);  
    public void mouseReleased(java.awt.event.MouseEvent);  
    public void mouseEntered(java.awt.event.MouseEvent);  
    public void mouseExited(java.awt.event.MouseEvent);  
  
}
```