

## Event Handlers

called "Action Listeners"

---

---

---

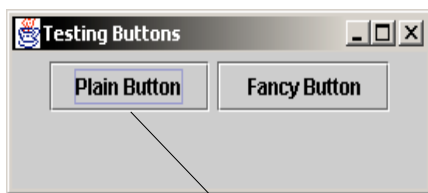
---

---

---

---

## JButtons



An "Event"

Java code that "Listens" for an event is called an "Action Listener"

---

---

---

---

---

---

---

## ActionListener class

- An interface – written by someone else, and you must adhere to the contract
- Class that implements the ActionListener must meet one obligation:
  - provide a method called `actionPerformed` that receives an `ActionEvent` object, as a means of telling "what just happened?".
  - The `actionPerformed` method runs based on a certain specified asynchronous event

---

---

---

---

---

---

---

```
private class eventClass implements ActionListener
{
    public void actionPerformed ( ActionEvent e )
    {
        System.out.println("Something just happened:");
        System.out.println( e.getActionCommand() );
    }
}
```

---

---

---

---

---

---

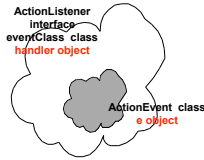
---

---

## ActionEvent objects

- An object of the ActionEvent class has the following property:

- It gets notified of computer events.
- It is an object of a class that contains methods that deal with events



---

---

---

---

---

---

---

---

## Asynchronous

- On NO schedule
- Happens without warning

---

---

---

---

---

---

---

---

## Objects of eventClass

Useful when this becomes an object somewhere in the immediate code (for instance, a class constructor)

```
eventClass handler = new eventClass( );
```

- handler is an object of `eventClass` (handler name is arbitrary)
- `eventClass` implements `ActionListener` interface (eventClass name is arbitrary)
- therefore, object handler is an `ActionListener`

---

---

---

---

---

---

---

---

## the JButton

- grabs one `ActionListener` object as it's own
- sends its activity to that `ActionListener`
- remember:
  - JButton needs an `ActionListener`
  - handler is an object of `eventClass`
  - `eventClass` implements `ActionListener` interface
  - therefore, handler is an `ActionListener`
  - therefore, JButton can ***attach itself*** to handler

---

---

---

---

---

---

---

---

## Tying a "component" to an ActionListener

- Components: `JButtons`, `JComboBoxes`, `JTextFields`, and other GUI controls.

```
JButton helloButton = new JButton( "Hello");  
eventClass handler = new eventClass( );  
helloButton.addActionListener( handler );
```

---

---

---

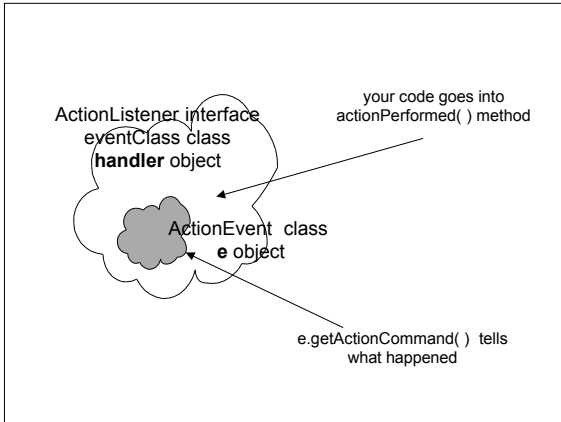
---

---

---

---

---




---

---

---

---

---

---

---

---

### Inner Classes

- the eventClass implementation of the ActionListener interface class, is wholly defined as private within the bounds of your main class

```
public class ButtonTest extends JFrame
{
    private class eventClass implements ActionListener
    {
        public void actionPerformed ( ActionEvent e )
        {
        }
    }
}
```

---

---

---

---

---

---

---

---

```
import java.awt.event.*;
public class ButtonTest extends JFrame    JFrame class
{
    public ButtonTest ( )                Constructor
    {
    }
    private class eventClass implements ActionListener
    {
        public void actionPerformed ( ActionEvent e )
        {
        }
    }                                     Inner class
    public static void main (String [ ] args)
    {
    }                                     main method
}
```

---

---

---

---

---

---

---

---

## Summary of Action Listeners

- Implementation of the ActionListener interface.
- Attached to a JButton, or other component.

---

---

---

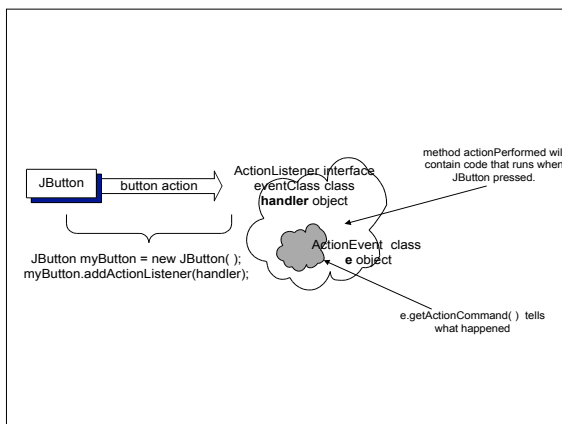
---

---

---

---

---



---

---

---

---

---

---

---

---

## Recipe: 10 E-Z steps

- Step 1: Write the main class framework

```
import java.awt.event.*;
public class ButtonTest extends JFrame
{
    public ButtonTest ( )
    {
    }
    public static void main (String [ ] args)
    {
    }
}
```

---

---

---

---

---

---

---

---

## Recipe

- Step 2: add an inner class called eventClass that implements ActionListener.
- Step 3: in eventClass, write the actionPerformed( ) method that uses an ActionEvent object as a parameter.
- Step 4: in actionPerformed( ActionEvent e), use e.getActionCommand( ) to get a description of what happened.

---

---

---

---

---

---

---

```
// inner class for button event handling
private class eventClass implements ActionListener
{
    public void actionPerformed( ActionEvent e )
    {
        System.out.println(e.getActionCommand( ) );
    }
} // end inner class
```

---

---

---

---

---

---

---

- Step 5 - in constructor of your JFrame extension, construct a JButton
- Step 6 - get ContentPane
- Step 7 – start up a FlowLayout
- Step 8 - add Button to ContentPane

---

---

---

---

---

---

---

```
public class ButtonTest extends JFrame
{
public ButtonTest()
{
setSize( 275, 100 );
setLocation (100,100);
Container c = getContentPane( );
c.setLayout( new FlowLayout( ) ); // new

// create button
JButton exitButton = new JButton( "Exit" );
c.add( exitButton );

setVisible(true);
}
}
```

---

---

---

---

---

---

---

---

- Step 9 – create an object of your eventClass inner class... call it handler.
- Step 10 – attach the object to your button

---

---

---

---

---

---

---

---

```
public class ButtonTest extends JFrame
{
public ButtonTest()
{
Container c = getContentPane();
JButton exitButton = new JButton( "Exit" );
c.setLayout( new FlowLayout( ) );
c.add( exitButton );
eventClass handler = new eventClass( );
exitButton.addActionListener( handler );
setSize( 275, 100 );
setVisible(true);
}
}
```

---

---

---

---

---

---

---

---

### The ActionEvent object

```
private class eventClass implements ActionListener
{
    public void actionPerformed ( ActionEvent e )
    {
        // use e.getActionCommand( ); to print
        // button label
        System.out.println(e.getActionCommand( ));
    }
}
```

---

---

---

---

---

---

---

---

### Combo boxes – ItemEvents not ActionEvents

```
// inner class for combo item handling
private class ComboHandler implements ItemListener
{
    public void itemStateChanged( ItemEvent x )
    {
        System.out.println( x.getItem( ) );
    }
} // end inner class
```

---

---

---

---

---

---

---

---

### in Constructor

```
String names[ ] = {"stay", "leave", "ignore"};
JComboBox myCombo = new JComboBox( names );
c.add( myCombo);
ComboHandler myComboHandler = new
    ComboHandler( );
myCombo.addItemListener(myComboHandler);
```

---

---

---

---

---

---

---

---

## Labels – no handler

```
JLabel label3 = new JLabel("Demo");
```

---

---

---

---

---

---

---

## JTextField

- Action Listener handler
- Same as button handler
- need access to the actual text field

---

---

---

---

---

---

---

## e.getSource

- Pull JButtons, JTextFields out of constructor brackets.
- Make them public and known to every one.
- Can be tested in if statements

**Defines the “two step” instantiation**

---

---

---

---

---

---

---

- Be sensitive ALWAYS to the bracket-boundaries of code that you're writing

```
public class myClass
{
    public myClass ()
        { // code here runs on instantiation
        }
    public static void main (String [ ] args)
        { // code here runs on startup
        }
}
// code here never runs
```

---

---

---

---

---

---

---

---

```
public class myClass extends
{
    // variables here can be known everywhere
    public myClass ()
        {
        }
    public static void main (String [ ] args)
        {
        }
}
```

---

---

---

---

---

---

---

---

### Two-step Instantiation

```
public class ButtonTest extends JFrame
{
    public JButton exitButton;
    public ButtonTest()
    {
        exitButton = new JButton("I want to leave");
        // other stuff
    }
}
```

---

---

---

---

---

---

---

---

### test in actionPerformed

```
private class ButtonHandler implements ActionListener
{
    public void actionPerformed( ActionEvent e )
    {
        System.out.println(e.getActionCommand());
        if (e.getSource() == exitButton)
        {
            System.out.println("Exiting");
            System.exit(0);
        }
    }
}
```

---

---

---

---

---

---

---

---

```
public class TextFieldTest extends JFrame
{
    JTextField writeBox;
    public TextFieldTest()
    {
        Container c = getContentPane();
        c.setLayout( new FlowLayout( ) );

        writeBox = new JTextField( "Hello" );
        c.add( writeBox );

        eventClass handler = new eventClass( );
        writeBox.addActionListener( handler );

        setVisible(true);
    }
}
```

---

---

---

---

---

---

---

---

```
private class eventClass implements ActionListener
{
    public void actionPerformed ( ActionEvent e )
    {
        System.out.println( writeBox.getText() );
    } // end actionPerformed method

} // end private class

} // end class TextBoxTest
```

---

---

---

---

---

---

---

---