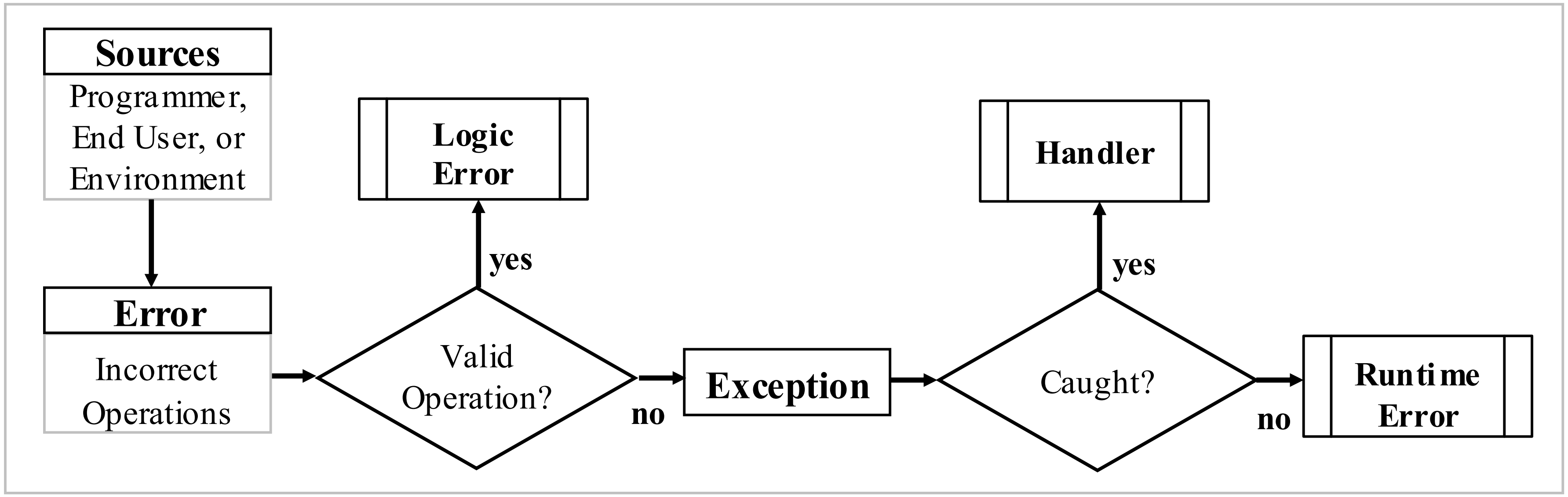
EECS 1720
Building Interactive Systems

Lecture 15 :: Event Handling [1]

Introduction to Events and Event Handling

Recap: Exception Handling



- An error source can lead to an incorrect operation
- An incorrect operations may be valid or invalid
- An invalid operation throws an exception
- An exception becomes a runtime error unless caught
- Caught exceptions can be handled gracefully

What is an Event?

- Change in state of an Object

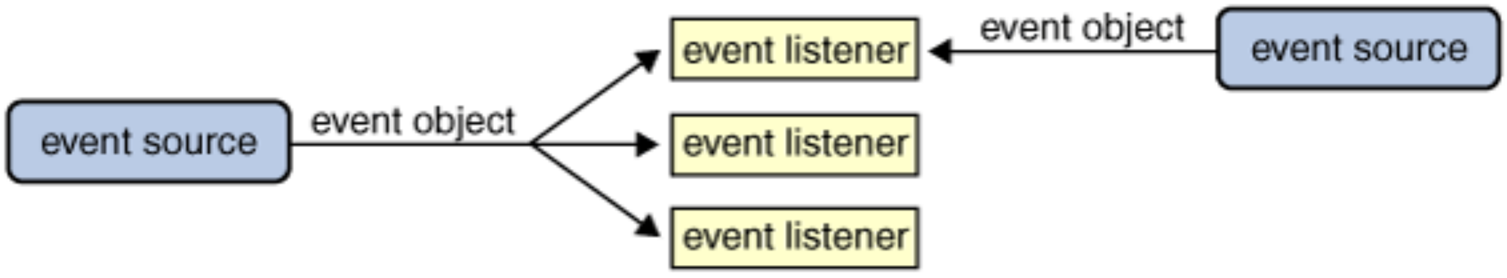
- Generated as a result of:
    - User interaction with an input Peripheral
        - Entering a character through the keyboard
        - Clicking a mouse button or moving the mouse
    - User Interaction with a GUI component
        - Typing into a text field
        - Clicking on a button in the GUI
        - Moving the mouse onto, or off a GUI element
        - Selecting an item from a list
        - Checking a checkbox/radiobutton item
        - Scrolling

Events

- **Foreground Events**
  - **Require direct interaction of user**
  - **e.g. with graphical component in the GUI**
    - **(button click, mouse click, mouse move, entering character through keyboard, select item from list, menu or radio button)**
- Background Events
  - May/may not require interaction of an end user
  - E.g. OS interruptions, hardware/software failure, timer expiry, operation completion

Event handling

• Event:	carries details of the event type
• Event Source:	generates event objects
• Event Listener:	listens for event objects
• Response:	processes event



https://docs.oracle.com/javase/tutorial/uiswing/events/intro.html

Events vs. Exceptions

- An event is “routed” much like an “exception” is routed
- An **Exception** is routed back through the calling chain (call stack) until it is handled
  - if it does not get captured by methods in the call stack, then handled by the JVM
- An **Event** is typically routed to one or more **registered** listener classes
  - An event will not get routed anywhere if no listener is registered!
  - The listener class is any class that implements the “ActionListener” interface (or extends from a class that implements this interface)

Event delegation

1. The user interacts with a GUI component and the event is generated.

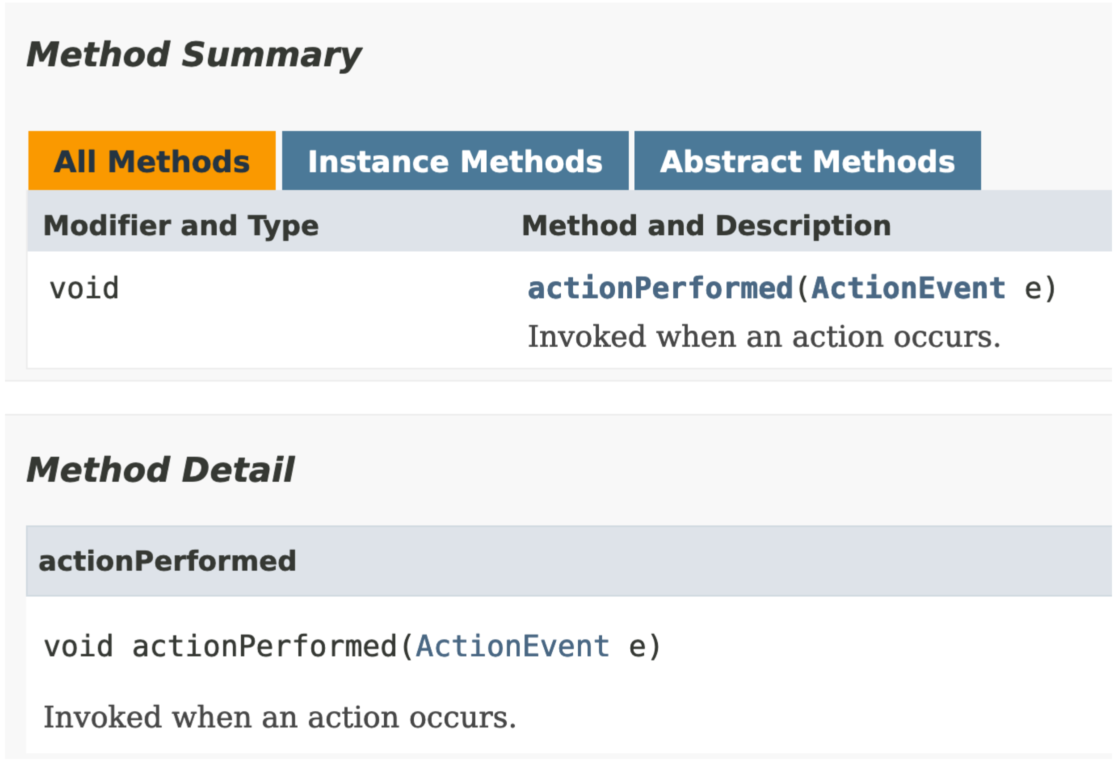
2. Now an object of the concerned **event class** is created automatically and information about the source and the event get initialized into that event object.
    - event types are in ActionEvent hierarchy

3. The event object is forwarded (delegated) to the method of registered listener class

4. the method is now executed and returns

ActionListener Interface

```
public interface ActionListener extends EventListener
```



Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
`void`	`actionPerformed(ActionEvent e)` Invoked when an action occurs.

Method Detail

actionPerformed

```
void actionPerformed(ActionEvent e)
```

Invoked when an action occurs.

https://docs.oracle.com/javase/8/docs/api/java/awt/event/ActionListener.html

Dealing with a generic ActionEvent (e)

- Much like dealing with an Exception object
- Methods to access state, and use state to decide what action to take

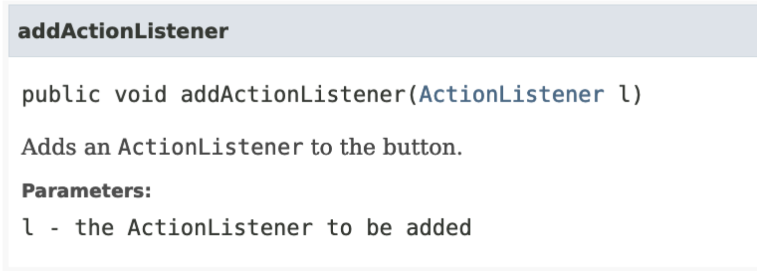
```
e.getActionCommand();     // access the state of the object
                          // for many GUI components this is
                          // a string value

e.getSource();            // access source object – the
                          // object that dispatched ("fired")
                          // the event
```

Example 1:

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

public class HelloEvent extends JFrame {

    public HelloEvent(String name) {
        super(name);
        this.setLayout(new FlowLayout());

        JLabel heading = new JLabel("HelloEvent:");
        JButton button = new JButton("Click Me");



        this.add(heading);
        this.add(button);
        this.setSize(480,400);
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);

    }
}
```

Example 1:

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

public class HelloEvent extends JFrame implements ActionListener {

    public HelloEvent(String name) {
        super(name);
        this.setLayout(new FlowLayout());

        JLabel heading = new JLabel("HelloEvent:");
        JButton button = new JButton("Click Me");

        button.addActionListener(this);

        this.add(heading);
        this.add(button);
        this.setSize(480,400);
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);

    }
}
```



addActionListener

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

Adds an `ActionListener` to the button.

Parameters:

`l - the ActionListener to be added`

Example 1:

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

public class HelloEvent extends JFrame implements ActionListener {

    public HelloEvent(String name) {

        // (previous slide)

    }


    @Override
    public void actionPerformed(ActionEvent e) {

        System.out.println(e.getActionCommand()); // get the command string
        System.out.println(e.getSource());        // what generated e ?

        System.out.println("Clicked on button");


    }

}
```

YORK
UNIVERSITÉ
UNIVERSITY

- Previous example:
  - Our application (HelloEvent class) implemented ActionListener **directly**
  - Could also create a separate class and register to an instance of that class
    - Completely independent class
    - OR.. as a nested class
      - Nested: class defined within another class

Example 2 (external listener class):

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

public class HelloEvent2Listener implements ActionListener {

    // default constructor not defined (but implicit)

    @Override
    public void actionPerformed(ActionEvent e) {

        System.out.println(e.getActionCommand()); // get the command string
        System.out.println(e.getSource());        // what generated e ?

        System.out.println("Clicked on button");


    }


}
```

Example 2 (external listener class):

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

public class HelloEvent2 extends JFrame {

    public HelloEvent2(String name) {

        super(name);
        this.setLayout(new FlowLayout());
        heading = new JLabel("HelloEvent:");
        button = new JButton("Click Me");

        button.addActionListener(new HelloEvent2Listener());

        this.add(heading);
        this.add(button);
        this.setSize(480,400);
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);


    }
}
```

Example 3 (nested listener class):

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

public class HelloEvent3 extends JFrame {

    public HelloEvent3(String name) {

        super(name);
        this.setLayout(new FlowLayout());
        heading = new JLabel("HelloEvent:");
        button = new JButton("Click Me");

        button.addActionListener(new HelloEvent3Listener());

        // setup not shown (add to content pane, etc)
    }

    class HelloEvent3Listener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            // ...
            System.out.println(”HelloEvent3: Clicked on button");
        }
    }
}
```

Implications?

- Why nest?
  - Listener might only be relevant to GUI controls defined within a given class (i.e. never used by other classes)
  - Nested class can automatically see all class fields of top level class
- Why implement as a separate class vs direct?
  - Can instantiate once, and re-use (same actionPerformed)
  - Can define different actionPerformed methods..
  - Can have separate instances that behave differently..

Instantiate once, and re-use

```
public class HelloEvent4 extends JFrame {

    public HelloEvent4(String name) {

        super(name);
        this.setLayout(new FlowLayout());
        JLabel heading = new JLabel("HelloEvent:");
        JButton button1 = new JButton("B1: Click Me");
        JButton button2 = new JButton("B2: Click Me 2");

        MyListener myListener = new MyListener("B* clicked");

        button1.addActionListener(myListener);
        button2.addActionListener(myListener);

        this.add(heading);
        this.add(button1);
        this.add(button2);

        // add to JFrame etc
    }
    public static void main(String[] args) {
        HelloEvent4 frame = new HelloEvent4("Hello Swing");
    }
}
```

Instantiate separately

```
public class HelloEvent4 extends JFrame {

    public HelloEvent4(String name) {

        super(name);
        this.setLayout(new FlowLayout());
        JLabel heading = new JLabel("HelloEvent:");
        JButton button1 = new JButton("B1: Click Me");
        JButton button2 = new JButton("B2: Click Me 2");

        button1.addActionListener(new MyListener("B1 clicked"));
        button2.addActionListener(new MyListener("B2 clicked"));

        this.add(heading);
        this.add(button1);
        this.add(button2);

        // add to JFrame etc
    }
    public static void main(String[] args) {
        HelloEvent4 frame = new HelloEvent4("Hello Swing");
    }
}
```

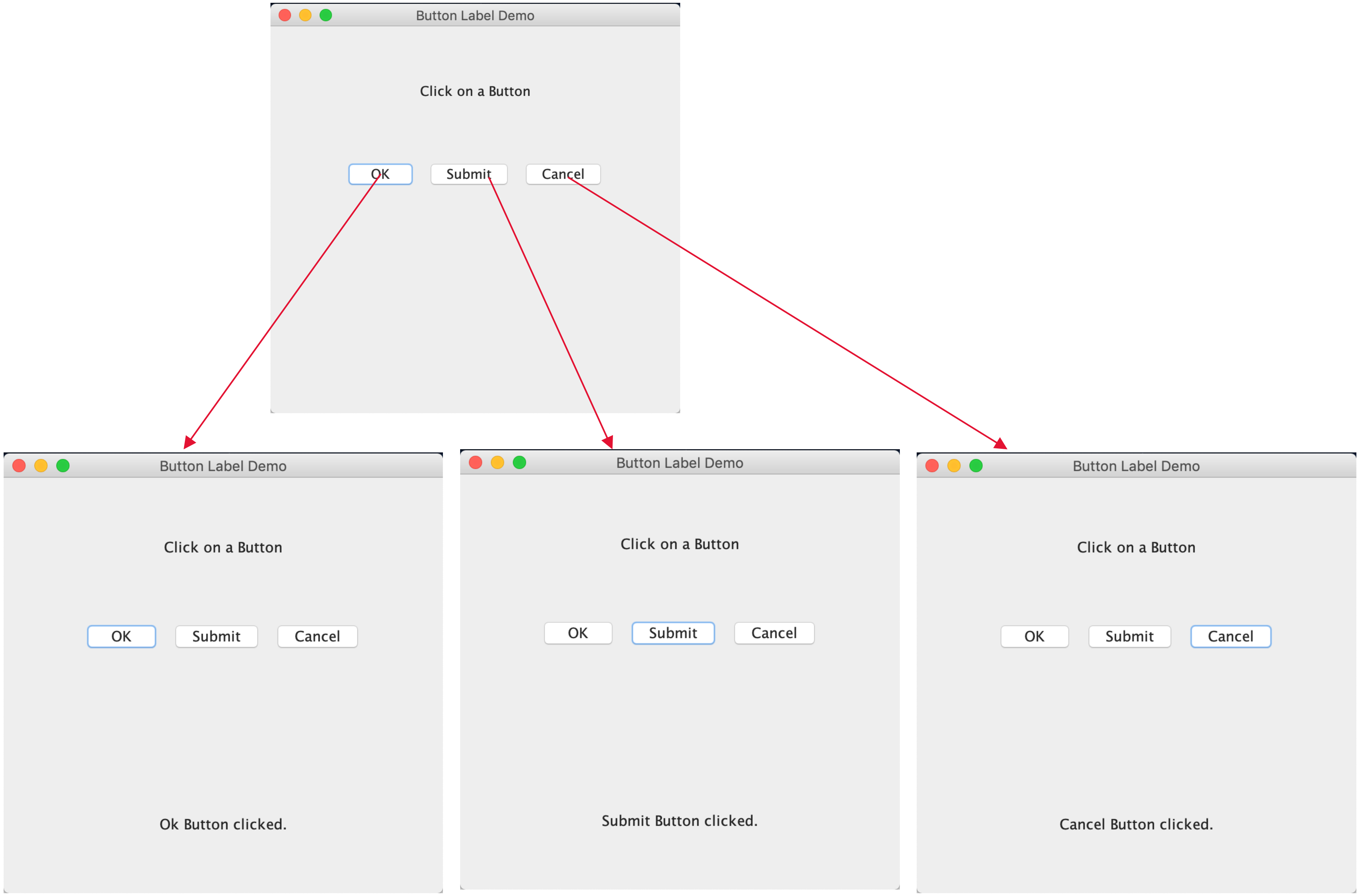
Action Command

- setActionCommand(String str)
- getActionCommand()

- set & get a string that has been associated with the GUI control
  - Buttons & Labels by default use the text associated with the control (can set this independently)

```java
public class HelloEvent5 extends JFrame implements ActionListener {

    JLabel heading;
    JButton button;

    public HelloEvent5(String name) {
        super(name);
        this.setLayout(new FlowLayout());

        heading = new JLabel("HelloEvent:");
        button = new JButton("Click Me", new ImageIcon("images/3d_file.png"));

        button.setActionCommand("Action click!");
        button.addActionListener(this);

        // add to content pane etc
    }

    @Override
    public void actionPerformed(ActionEvent e) {

        if (e.getSource()==this.button) {
            System.out.println(this.button.getText());
            System.out.println(e.getActionCommand());
        }
        else {
            System.out.println("something other event");
        }
    }
}
```

ButtonLabelDemo



Click on a Button

Ok Button clicked.

Submit Button clicked.

Cancel Button clicked.

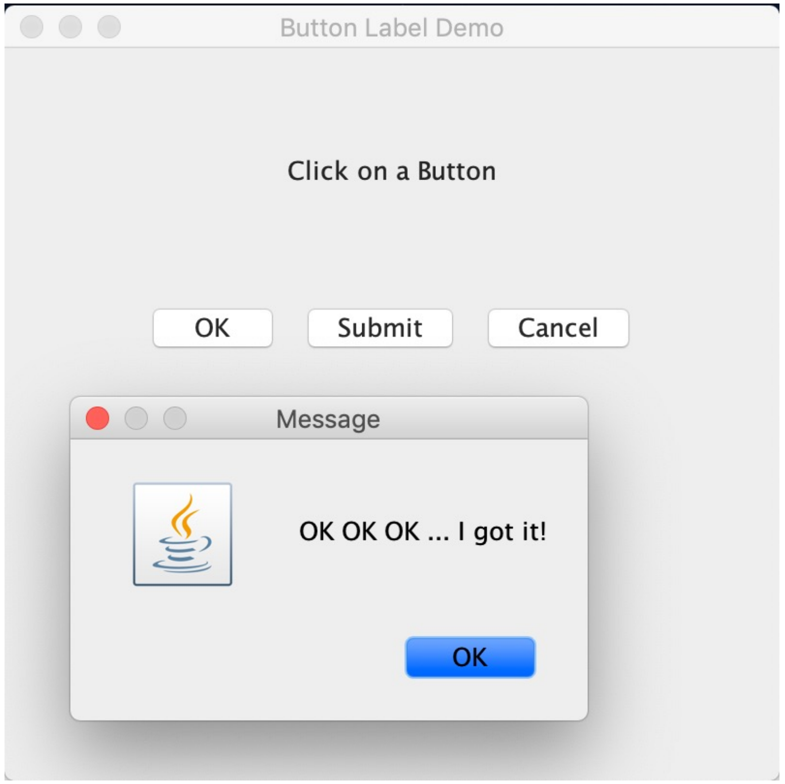
Using ActionCommands:

```
public class ButtonClickListener implements ActionListener{

    public void actionPerformed(ActionEvent e) {

        String command = e.getActionCommand();

        if( command.equals( "OK" ))  {
                statusLabel.setText("Ok Button clicked.");
        }
        else if( command.equals( "Submit" ) )  {
                statusLabel.setText("Submit Button clicked.");
        }
        else  {
                statusLabel.setText("Cancel Button clicked.");
        }
    }
}
```

```java
public class ButtonLabelDemo extends JFrame {

private Label statusLabel;

public ButtonLabelDemo() {
      super("Button Label Demo");
      this.setLayout(new GridLayout(3, 1));

      JLabel headerLabel = new JLabel();
      headerLabel.setText("Click on a Button");
      this.statusLabel = new Label();

      Panel controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());
      Button okButton = new Button("OK");
      Button submitButton = new Button("Submit");
      Button cancelButton = new Button("Cancel");

      okButton.setActionCommand("OK");
      submitButton.setActionCommand("Submit");
      cancelButton.setActionCommand("Cancel");

      okButton.addActionListener(new ButtonClickListener());
      submitButton.addActionListener(new ButtonClickListener());
      cancelButton.addActionListener(new ButtonClickListener());

      controlPanel.add(okButton);
      controlPanel.add(submitButton);
      controlPanel.add(cancelButton);

      this.add(headerLabel);
      this.add(controlPanel);
      this.add(statusLabel);
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      this.setVisible(true);

}
public static void main(String[] args){
      ButtonLabelDemo  demo = new ButtonLabelDemo();
}

// ActionListener (nested)
}
```

```java
private class ButtonClickListener implements ActionListener{

      public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();
            if( command.equals( "OK" ))  {
            statusLabel.setText("Ok Button clicked.");
            }
            else if( command.equals( "Submit" ) )  {
            statusLabel.setText("Submit Button clicked.");
            }
            else  {
            statusLabel.setText("Cancel Button clicked.");
            }
      }
}
```

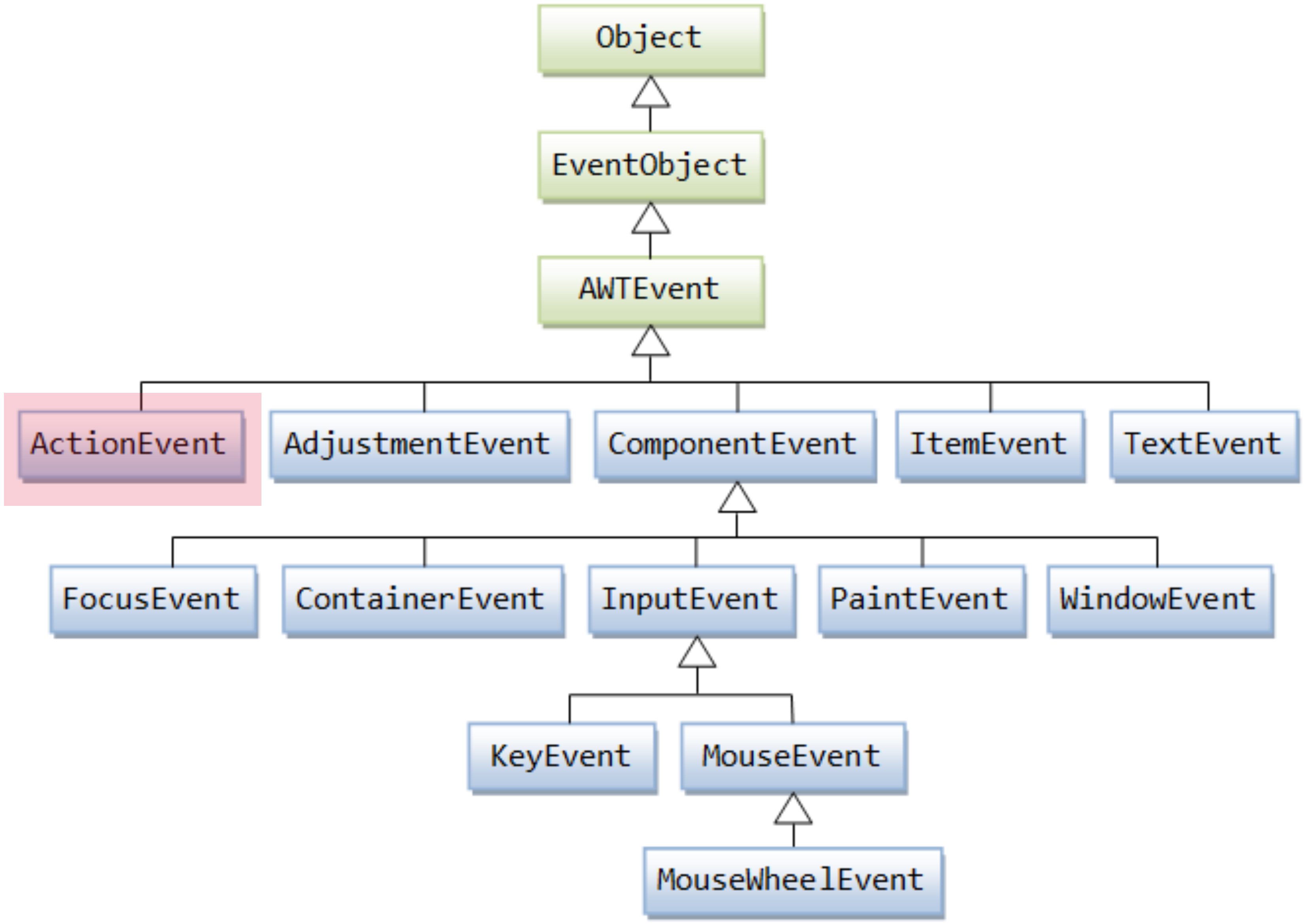
```java
private class ButtonClickListener implements ActionListener{

      public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();

            if( command.equals( "OK" ))  {
                  statusLabel.setText("Ok Button clicked.");
                  // open a dialog window
                  JOptionPane.showMessageDialog((Component) e.getSource(),
                                          "OK OK OK ... I got it!");
            }
            else if( command.equals( "Submit" ) )  {
                  statusLabel.setText("Submit Button clicked.");
            }
            else  {
                  statusLabel.setText("Cancel Button clicked.");

            }
      }
}
```



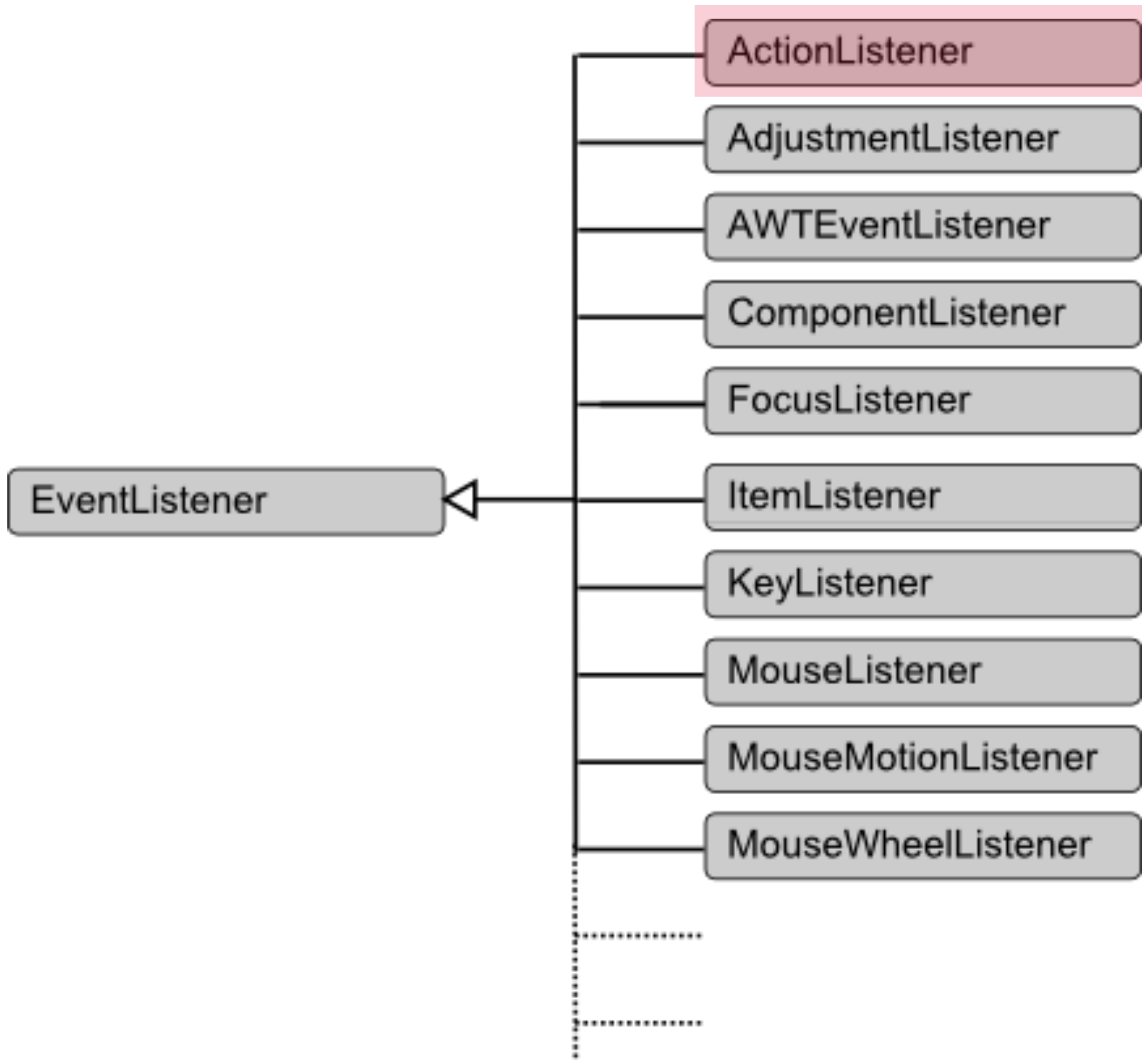
Click on a Button

OK OK OK ... I got it!

Events


Object
EventObject
AWTEvent
ActionEvent
AdjustmentEvent
ComponentEvent
ItemEvent
TextEvent
FocusEvent
ContainerEvent
InputEvent
PaintEvent
WindowEvent
KeyEvent
MouseEvent
MouseWheelEvent

Event Listeners (interfaces)



Event Listeners

User Action	**Event Triggered**	**Event Listener interface**
Click a `Button`, `JButton`	**`ActionEvent`**	**`ActionListener`**
Open, iconify, close `Frame`, `JFrame`	`WindowEvent`	`WindowListener`
Click a `Component`, `JComponent`	`MouseEvent`	`MouseListener`
Change texts in a `TextField`, `JTextField`	`TextEvent`	`TextListener`
Type a key	`KeyEvent`	`KeyListener`
Click/Select an item in a `Choice`, `JCheckbox`, `JRadioButton`, `JComboBox`	`ItemEvent`, **`ActionEvent`**	`ItemListener`, **`ActionListener`**

- Java Tutorials on Event Listeners:
  - https://docs.oracle.com/javase/tutorial/uiswing/events/index.html

- More next lecture!