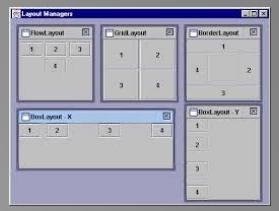


# Object Oriented Programming

## Graphical User Interface

Hirra Anwar

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science
Islamabad

# Agenda

- **Event Driven Programming**
- **Java Event Types and Listeners Interfaces**
  - Action Listener
  - Mouse Listeners
  - Key Events
  - ……
- **Adapter Classes**
- **Anonymous Inner Classes**

# Agenda

- **Event Driven Programming**
- **Java Event Types and Listeners Interfaces**
  - Action Listener
  - Mouse Listeners
  - Key Events
  - ……
- **Adapter Classes**
- **Anonymous Inner Classes**

- **Event Driven Programming**
- **Java Event Types and Listeners Interfaces**
  - Action Listener
  - Mouse Listeners
  - Key Events
  - ……
- **Adapter Classes**
- **Anonymous Inner Classes**

- **The Delegation Event Model**
  - Defines standard and consistent mechanisms to generate and process events
  - Describes how your program can respond to user interaction
  - Model used by Java to handle user interaction with GUI components

- A source generates an event and sends it to one or more listeners.
- In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.

## Three important players

1. Event Source
2. Event Listener/Handler
3. Event Object

- **Event Source**
  - GUI component that generates the event
  - Example: button
- **Event Listener/Handler**
  - Receives and handles events
  - Contains business logic
  - Example: displaying information useful to the user, computing a value
- **Event Object**
  - Created when an event occurs (i.e., user interacts with a GUI component)
  - Contains all necessary information
  - Represented by an Event class

- **Event Driven Programming**
- **Java Event Types and Listeners Interfaces**
  - Action Listener
  - Mouse Listeners
  - Key Events
  - ……
- **Adapter Classes**
- **Anonymous Inner Classes**

- All Events are objects of Event Classes.
- All Event Classes are derived from EventObject.
- When an Event occurs, Java sends a message to all registered Event Listeners from the Event source

- https://docs.oracle.com/javase/7/docs/api/javax/swing/event/package-tree.html

- **The *EventObject class***
  - Found in the *java.util package*
- **The *AWTEvent class***
  - An immediate subclass of *EventObject*
  - Defined in *java.awt package*
  - Root of all AWT-based events

1. Either **implements** a listener interface or extends **a** class that implements a listener interface

2. Register your listener

   someComponent.addActionListener(instanceOfMyClass);

3. Implement user action

   public void actionPerformed(ActionEvent e) {
   ...//code

   that reacts to the action... }

```
public class MyHelloGUI extends JFrame
{
      private JButton btn=new JButton("Click me");
      private JTextField jtf=new JTextField("value");
      private JLabel lab= new JLabel("Enter Value:");

      public MyHelloGUI()
      {
         Container con = this.getContentPane();
         this .setTitle("Title");
         con.setLayout(new FlowLayout());
         con.add(lab);
         con.add(jtf);
         con.add(btn);
         con.setSize(300,200);
  btn.addActionListener(new   MyActionListener());
        con.show();
        this.pack();
       this.setVisible(true);
   }
```

```
 public static void main(String[] args)
 {
      MyHelloGUI myHelloGUI = new MyHelloGUI();
 }
 class MyActionListener implements ActionListener
 {
    public void actionPerformed (ActionEvent e)
    {
         if(e.getSource() == btn)
         {
           JOptionPane.showMessageDialog(null,jtf.getText());
         }
     }
 }
}
```

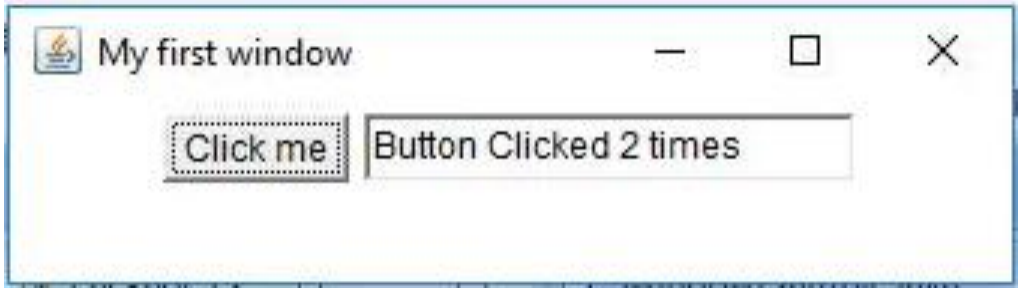
**addActionListener(*reference to the object of Handler class*)**

```
btn.addActionListener(new   MyActionListener());

class MyActionListener implements ActionListener
{
    public void actionPerformed (ActionEvent e)
    {
         if(e.getSource() == btn)
         {
           JOptionPane.showMessageDialog(null,jtf.getText());
        }
     }
}
```

```
btn.addActionListener(new ActionListener()
{
   public void actionPerformed (ActionEvent e)
   {
      if(e.getSource() == btn)
      {
        JOptionPane.showMessageDialog(null,jtf.getText());
      }
  }
});
```

# Event : Example



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

public class AL extends Frame implements ActionListener {
        TextField text = new TextField(20);
        Button b;
        private int numClicks = 0;

public AL(String title) {

                super(title);
                setLayout(new FlowLayout());
                b = new Button("Click me");
                add(b);
                add(text);
                b.addActionListener(this);
        }
```

addActionListener(*reference to the object of Handler class*)

```
public void actionPerformed(ActionEvent e) {
        numClicks++;
        text.setText("Button Clicked " + numClicks + " times");
    }

public static void main(String[] args) {
        AL myWindow = new AL("My first window");
        myWindow.setSize(350,100);
        myWindow.setVisible(true);
    }
}
```

addActionListener(*reference to the object of Handler class*)

```
import java.awt.Container;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class EventActionPerfomredExample extends JFrame
{
    public EventActionPerfomredExample()
    {
        JButton btn = new JButton("Click");
        Container cont= this.getContentPane();
        btn.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                JButton b= (JButton) e.getSource();
                JOptionPane.showMessageDialog(null, "Heloo :
"+b.getText());
            }
        });
```

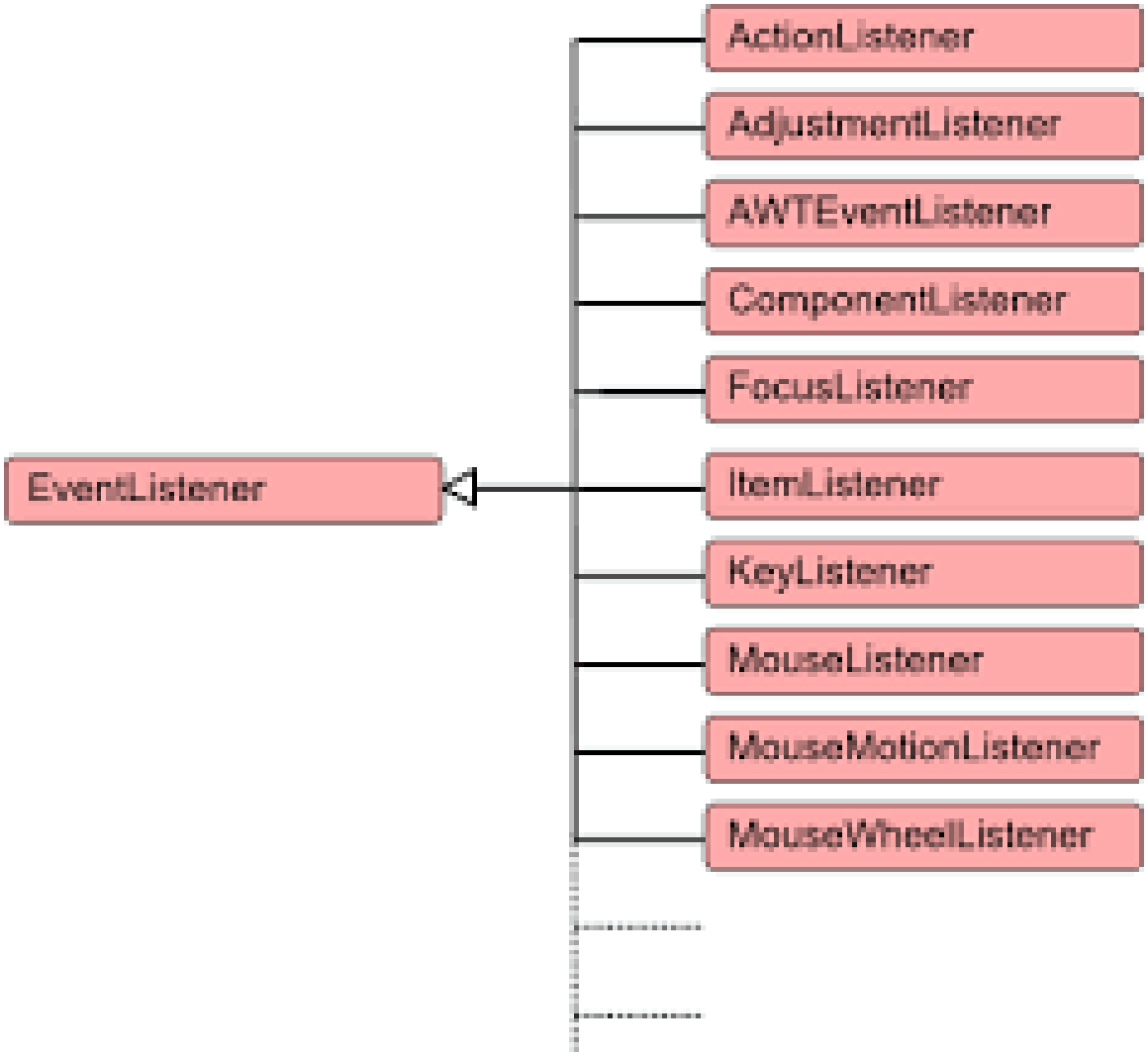
```
                cont.add(btn);
                this.setBounds(new Rectangle(200,200,200,200));
                this.setVisible(true);
        }
        public static void main(String atr [ ] )
        {
                new EventActionPerfomredExample();
        }
}
```

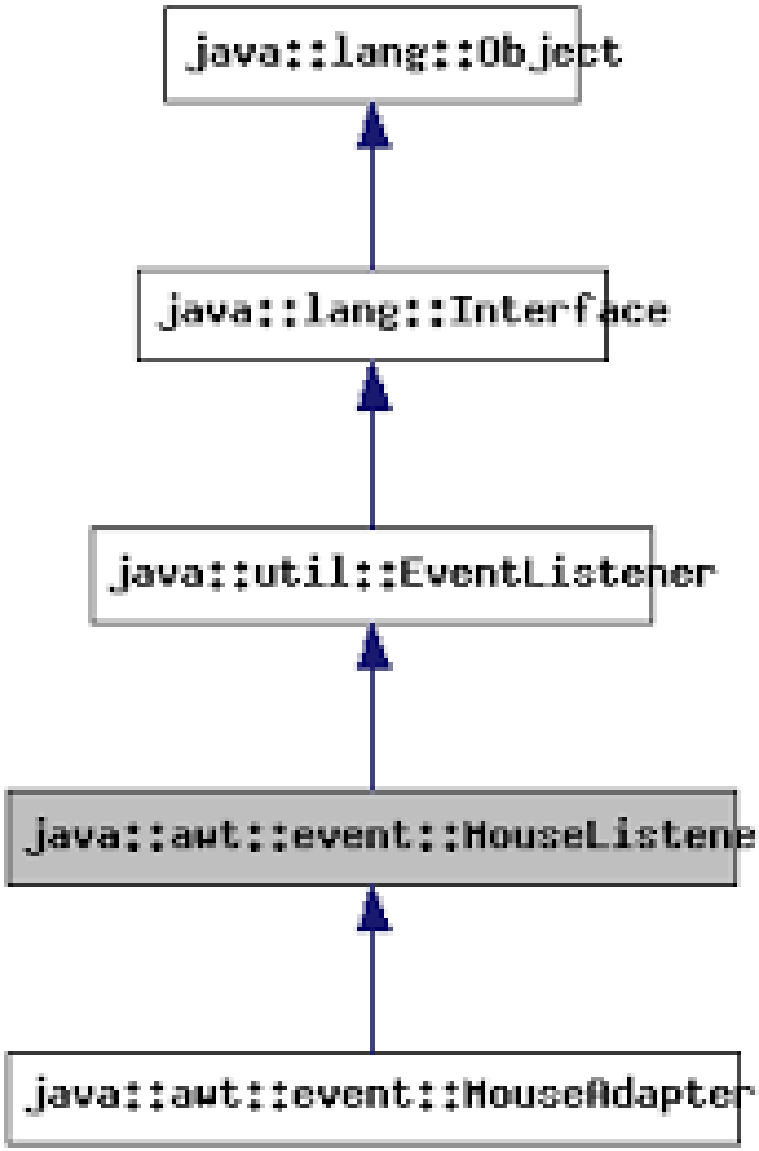

Click


Message
Heloo : Click
OK

- **Event Driven Programming**
- **Java Event Types and Listeners Interfaces**
  - Action Listener
  - Mouse Listeners
  - Key Events
  - ……
- **Adapter Classes**
- **Inner Classes and Anonymous Inner Classes**

| | |
|---|---|
| **Mouse Listener** | mouseClicked() |
| | mouseEntered() |
| | mouseExited() |
| | mousePressed() |
| | mouseReleased() |
| **MouseMotion Listener** | mouseDragged() |
| | mouseMoved() |


EventListener
ActionListener
AdjustmentListener
AWTEventListener
ComponentListener
FocusListener
ItemListener
KeyListener
MouseListener
MouseMotionListener
MouseWheelListener


java::lang::Object
java::lang::Interface
java::util::EventListener
java::awt::event::MouseListene
java::awt::event::MouseAdapter

# Listeners

- Process events
  - ActionListener (JButton, Timer, JComboBox)
  - ChangeListener (JSlider)
  - MouseListener, MouseMotionListener
- Listeners are interfaces; must implement *ALL* specified methods
  - ActionListener: void actionPerformed(ActionEvent e)
  - ChangeListener: void stateChanged(ChangeEvent e)
  - MouseListener: void mouseClicked(MouseEvent e)
    void mousePressed(MouseEvent e)
    void mouseReleased(MouseEvent e)
    void mouseEntered(MouseEvent e)
    void mouseExited(MouseEvent e)
  - MouseMotionListener: void mouseMoved(MouseEvent e)
    void mouseDragged(MouseEvent e)

| Adapter class | Listener interface |
| --- | --- |
| WindowAdapter | WindowListener |
| KeyAdapter | KeyListener |
| MouseAdapter | MouseListener |
| MouseMotionAdapter | MouseMotionListener |
| FocusAdapter | FocusListener |
| ComponentAdapter | ComponentListener |
| ContainerAdapter | ContainerListener |
| HierarchyBoundsAdapter | HierarchyBoundsListener |

# Adapter classes

- Convenience classes
  - server as intermediaries between available interfaces (e.g. listeners) and the user defined classes (e.g. listeners)
  - make it possible to implement only the “important” methods

# Adapter classes

- Convenience classes
  - server as intermediaries between available interfaces (e.g. listeners) and the user defined classes (e.g. listeners)
  - make it possible to implement only the “important” methods

```
abstract  class MouseAdapter implements MouseListener, MouseMotionListener
{
    void    mousePressed(MouseEvent e)    {  // empty body  }
    void    mouseReleased(MouseEvent e)   {  // empty body  }
    void    mouseEntered(MouseEvent e)    {  // empty body  }
    void    mouseExited(MouseEvent e)     {  // empty body  }
    void    mouseMoved(MouseEvent e)      {  // empty body  }
    void    mouseDragged(MouseEvent e)    {  // empty body  }
}
```

MouseListener methods

MouseMotionListener methods

# MouseListener : Example

```
public class EventMouseListnerExample extends JFrame
{
	public EventMouseListnerExample()
	{
		JButton btn = new JButton("Click");
		Container cont= this.getContentPane();
		btn.addMouseListener(new MouseListnerImp());
		cont.add(btn);
		this.setBounds(new Rectangle(200,200,200,200));
		this.setVisible(true);
	}
	class MouseListnerImp implements MouseListener
	{
		public void mouseClicked(MouseEvent e) {
			System.out.println("Click Count: "+e.getClickCount());
		}
		public void mouseEntered(MouseEvent e) {
			System.out.println("Mouse Entered in Button Area !");
		}
		public void mouseExited(MouseEvent e) {
			System.out.println("Mouse Exited from Button Area !");
		}
```

```
        public void mousePressed(MouseEvent e) {
                System.out.println("Mouse Pressed !");
        }
        public void mouseReleased(MouseEvent e) {
                System.out.println("Mouse Released !");
        }
    }

    public static void main(String atr [] )
    {
    new EventMouseListnerExample();
    }

}
```

| KeyListener | keyPressed() |
| --- | --- |
| | keyReleased() |
| | keyTyped() |

```
btn.addKeyListener(new KeyListenerImp());

class KeyListenerImp implements KeyListener
{
    public void keyPressed(KeyEvent e) {
        System.out.println("Key Presses :"+e.getKeyChar());
    }
    public void keyReleased(KeyEvent e) {
        System.out.println("Key Released :"+e.getKeyChar());
    }
    public void keyTyped(KeyEvent e) {
        System.out.println("Key Types :"+e.getKeyCode());
    }
}
```

# Java Event Types, Listeners & Listener Methods

| Event Class | Listener Interface | Listener Methods |
|---|---|---|
| **ActionEvent** | **ActionListener** | actionPerformed() |
| **AdjustmentEvent** | **AdjustmentListener** | adjustmentValueChanged() |
| **ComponentEvent** | **ComponentListener** | componentHidden() |
| | | componentMoved() |
| | | componentResized() |
| | | componentShown() |
| **ContainerEvent** | **ContainerListener** | componentAdded() |
| | | componentRemoved() |
| **FocusEvent** | **FocusListener** | focusGained() |
| | | focusLost() |

# Java Event Types, Listeners & Listener Methods

| Event Class | Listener Interface | Listener Methods |
|---|---|---|
| **WindowEvent** | **WindowListener** | windowActivated() |
| | | windowClosed() |
| | | windowClosing() |
| | | windowDeactivated() |
| | | windowDeiconified() |
| | | windowIconified() |
| | | windowOpened() |

- **Event Driven Programming**
- **Java Event Types and Listeners Interfaces**
  - Action Listener
  - Mouse Listeners
  - Key Events
  - ……
- **Adapter Classes**
- **Anonymous Inner Classes**

- **Often we only really want to implement one or two of the event handlers**
- **But interface implementation requires us to implement all of them**
- **Java adapter classes allow us to solve this problem**
- **Adapter classes**
  - Implement the event-handling interfaces in a trivial way, just to satisfy the interface
  - We extend the class and override the methods that we need

- **Some adapter classes:**
  - KeyAdapter
  - WindowAdapter
  - MouseAdapter
  - MouseMotionAdapter

```
btn.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        if(e.getButton() == 1)
        {
            System.out.println("Mouse Left Button Clicket !");
        }
        else if (e.getButton() ==2)
        {
            System.out.println("Mouse Center Button Clicket !");
        }
        else if(e.getButton() ==3 )
        {
            System.out.println("Mouse Right Button Clicket !");
        }
    }
});
```

- **Event Driven Programming**
- **Java Event Types and Listeners Interfaces**
  - Action Listener
  - Mouse Listeners
  - Key Events
  - ……
- **Adapter Classes**
- **Anonymous Inner Classes**

- **Unnamed inner classes**
- **Why use anonymous inner classes?**
  - Further simplify your codes
  - Especially in event handling
- This is a variation of the previous option.
- An anonymous inner class is an inner class without a name.
- You define the class "on the fly" when you need an instance of it (an object) to be created.

- **Class declared within another class**
- **Why use inner classes?**
  - Help simplify your programs
  - Especially in event handling

- you can implement your listeners inside the class that extends your JFrame, making it an inner class.
- This enables you to put everything in one class (and hence file).
- Inner classes have access to the private data of the outer class

# Agenda

- **Event Driven Programming**
- **Java Event Types and Listeners Interfaces**
  - Action Listener
  - Mouse Listeners
  - Key Events
  - ……
- **Adapter Classes**
- **Anonymous Inner Classes**

- Reading Material:
  - Chapter 14