Java Swing

Dr. Partha Pratim Sarangi

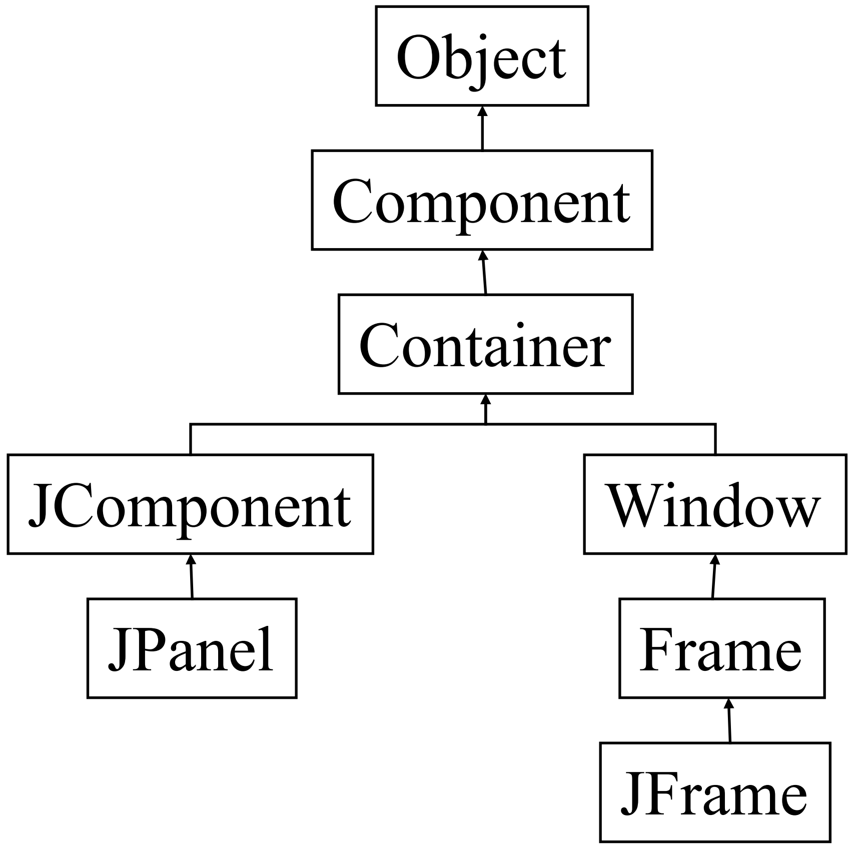
School of Computer Engineering

Java Swing

- Java Swing is a set of GUI (Graphical User Interface) components for Java.
- It provides a comprehensive toolkit for creating sophisticated and platform-independent graphical user interfaces.
- Swing is built on top of the Abstract Window Toolkit (AWT) but provides more powerful and flexible components.
- Commonly used classes in javax.swing package:
    - JButton, JTextBox, JTextArea, JPanel, JFrame, JMenu, JSlider, JLabel, JIcon, …
    - There are many, many such classes to do anything imaginable with GUIs
    - Here we only study the basic architecture and do simple examples

Swing components

- Each component is a Java class with a fairly extensive inheritance hierarchy:



Using Swing Components

- Very simple, just create object from appropriate class – examples:
    - JButton but = new JButton();
    - JTextField text = new JTextField();
    - JTextArea text = new JTextArea();
    - JLabel lab = new JLabel();
- Many more classes. We don’t need to know every one to get started.

Adding components

- Once a component is created, it can be added to a container by calling the container’s add method:

```
JFrame f = new JFrame("Title");
     JPanel p = new JPanel();
     JButton b = new JButton("Click Me");
p.add(b); //add Button to Panel
f.setContentPane(p); //add Panel to Frame
f.setVisible(true);
setLayout(new FlowLayout()); //set Layout Manager
```

- Actually, how components are laid out is determined by the layout manager.
- If we choose not to use a layout manager, you need to set the layout of the container (in this case, the JFrame) to null (setLayout(null)), which means you will need to manually specify the position and size of each component.

Layout Managers

- Java comes with 7 or 8. Most common and easiest to use are
  - FlowLayout
  - BorderLayout
  - GridLayout
- Using just these three it is possible to attain fairly precise layout for most simple applications.

Setting layout managers

- Very easy to associate a layout manager with a component. Simply call the **setLayout** method on the Container:

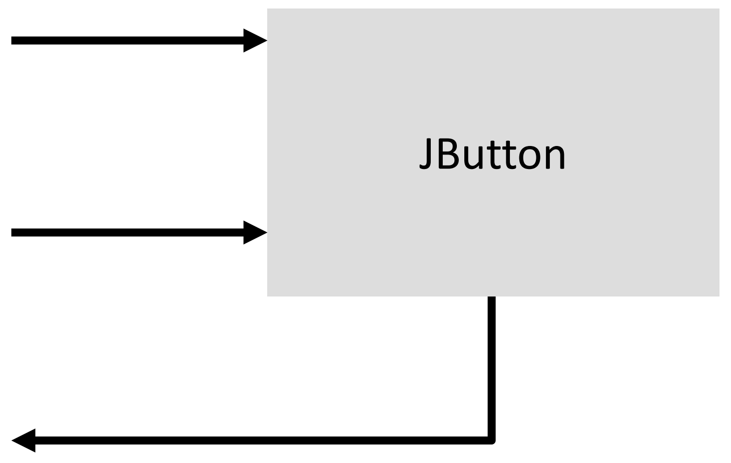
```
JPanel p1 = new JPanel();
p1.setLayout(new
FlowLayout(FlowLayout.LEFT));

JPanel p2 = new JPanel();
p2.setLayout(new BorderLayout());
```

As Components are added to the container, the layout manager determines their size and positioning.

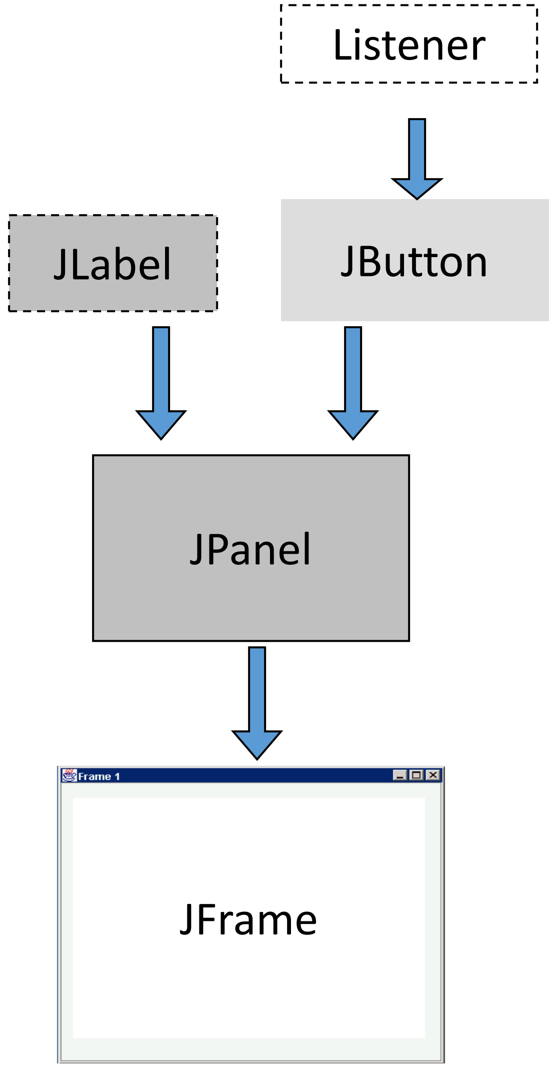
GUI Component API

- Java: GUI component = class
- Properties
- Methods
- Events



Build from bottom up

- Create:
  - Frame
  - Panel
  - Components
  - Listeners
- Add:(bottom up)
  - listeners into components
  - components into panel
  - panel into frame



Event handling

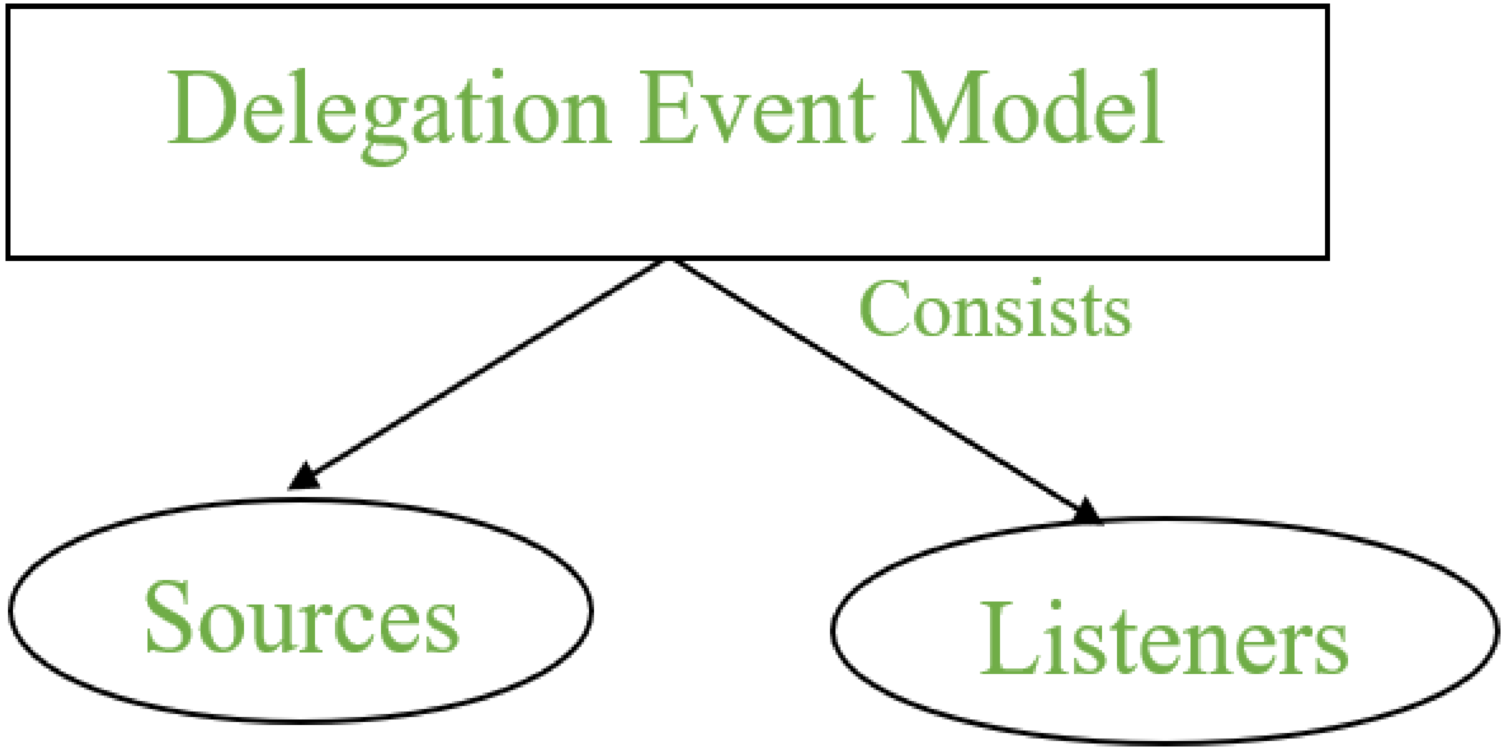
- **What are events?**
- In computer programming, the event is an **action** occurred in GUI components; that is recognized by a program as object and finally this object may be handled in the program.
- Changing the state of an object is known as an Event.
- Events can be generated by various sources, such as user interaction with the **graphical user interface (GUI)** components.
- All components can listen for one or more events.

  Typical examples are:
  - Mouse clicks
  - Mouse movements
  - Hitting any key
  - Hitting return key
  - etc.

Delegation Event Model

- Java uses the “Delegation Event Model” to handle the events. This model defines the standard mechanism to generate and handle the events.
- The delegation event model has the following three key concepts namely:

1. Source:
   - Source is an object that generates an event.
   - Event occurs when the internal state of that object changes in some way.
   - Sources may generate more than one type of event.
   - Source is responsible for providing information of the occurred event to it’s handler.
2. Listener:
   - It is also known as event handler.
   - A listener is an object that is notified when an event occurs.
   - Listener is responsible for generating response to an event.
   - From java implementation point of view the listener is also an object.
   - Listener waits until it receives an event. Once the event is received, the listener process the event.


Delegation Event Model
Consists
Sources
Listeners

Steps involved in Event Handling

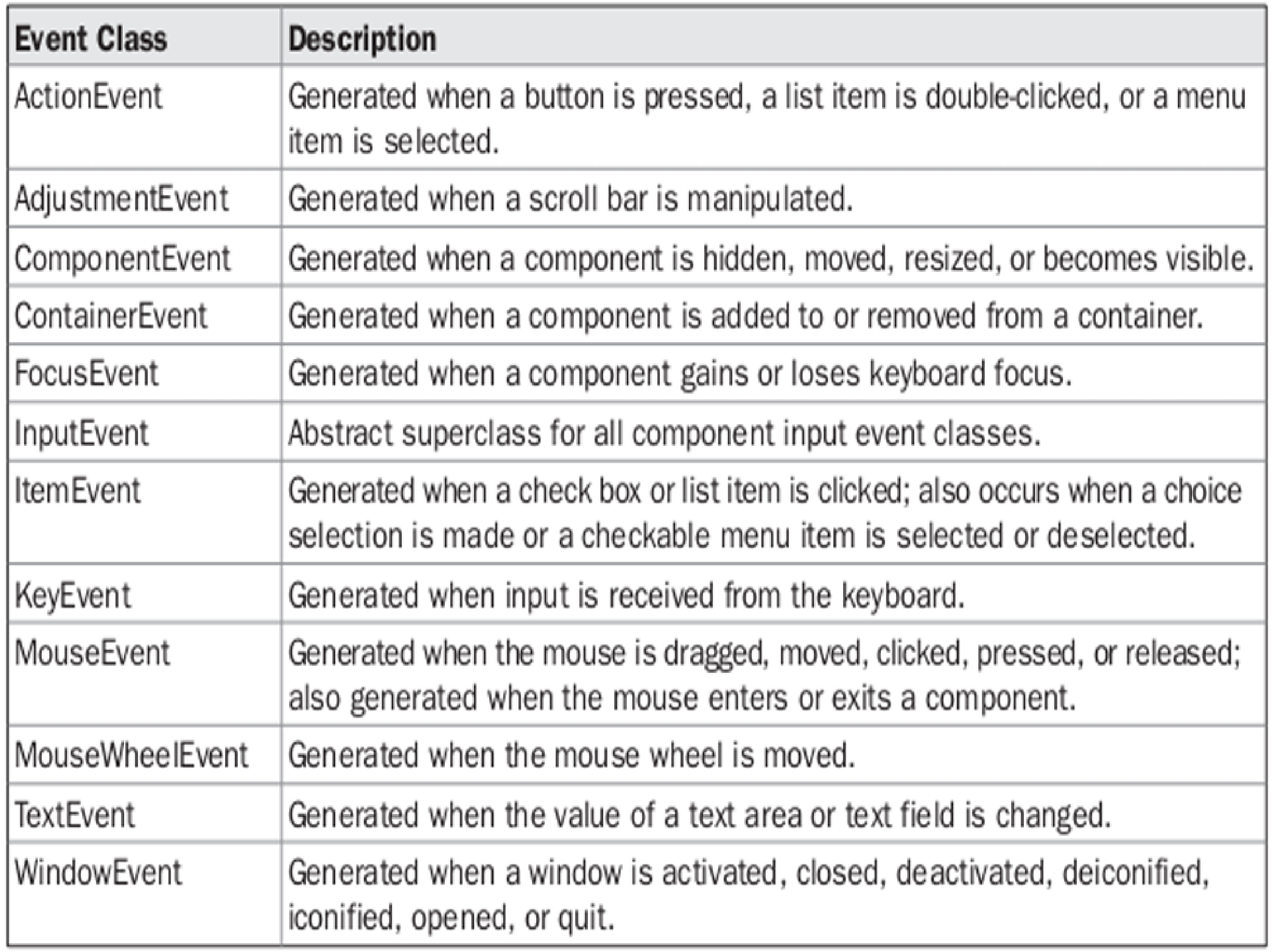
- **Step 1:**
  - The user clicks the button and the event is generated.
- **Step 2:**
  - The object of related event class is created automatically and information about the event source and the event stored with in the same object.
- **Step 3:**
  - Event object is forwarded to the method of registered listener class.
  - Event source must register listeners in order for the listeners to receive notification about a specific type of event.
  - Each type of event has its own registration method.
- **Step 4:**
  - The method processes these notifications and then return results.

Advantages of Delegation Event Model

- The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.

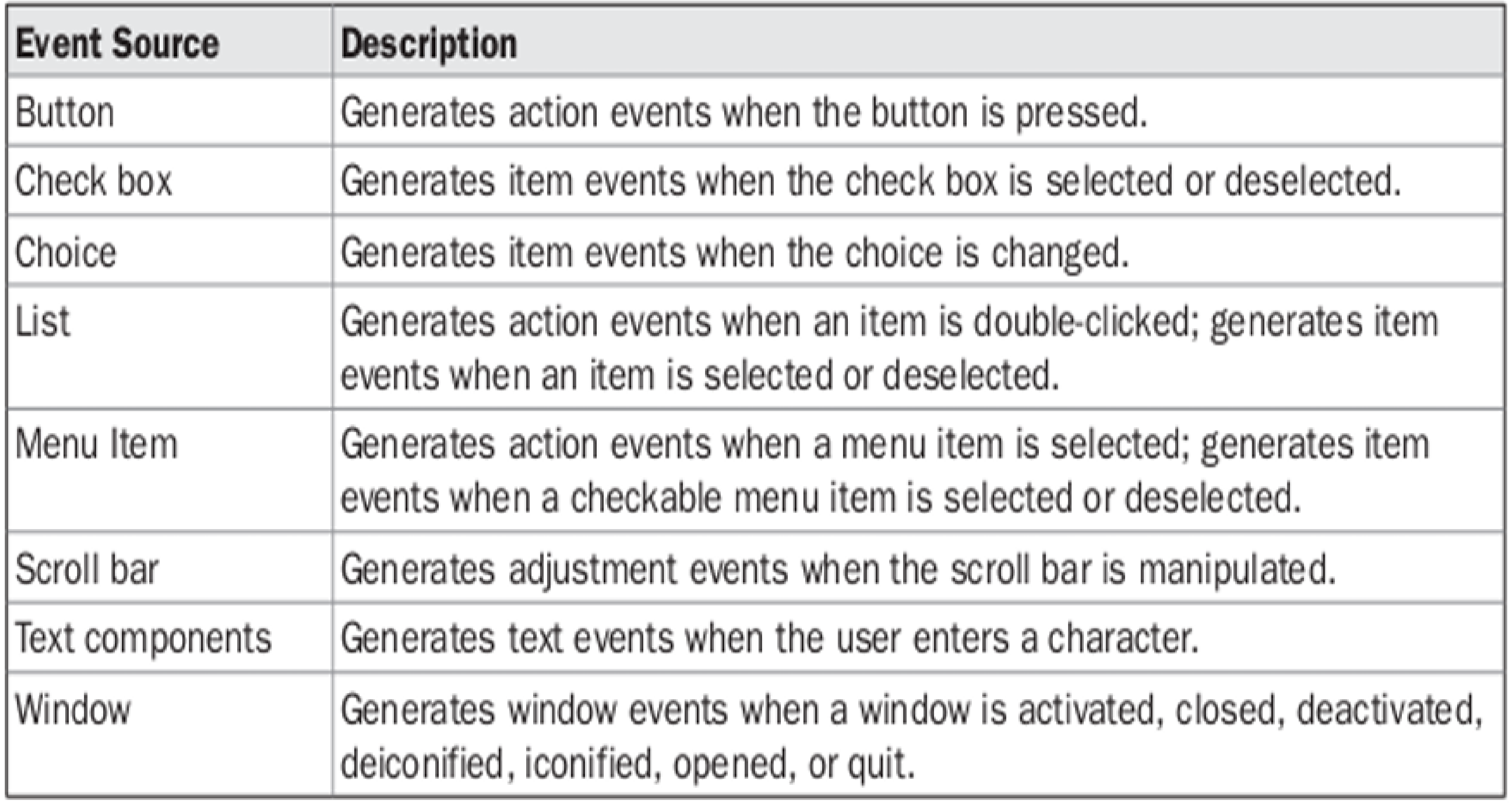
- In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them.

Main Event Classes in java.awt.event



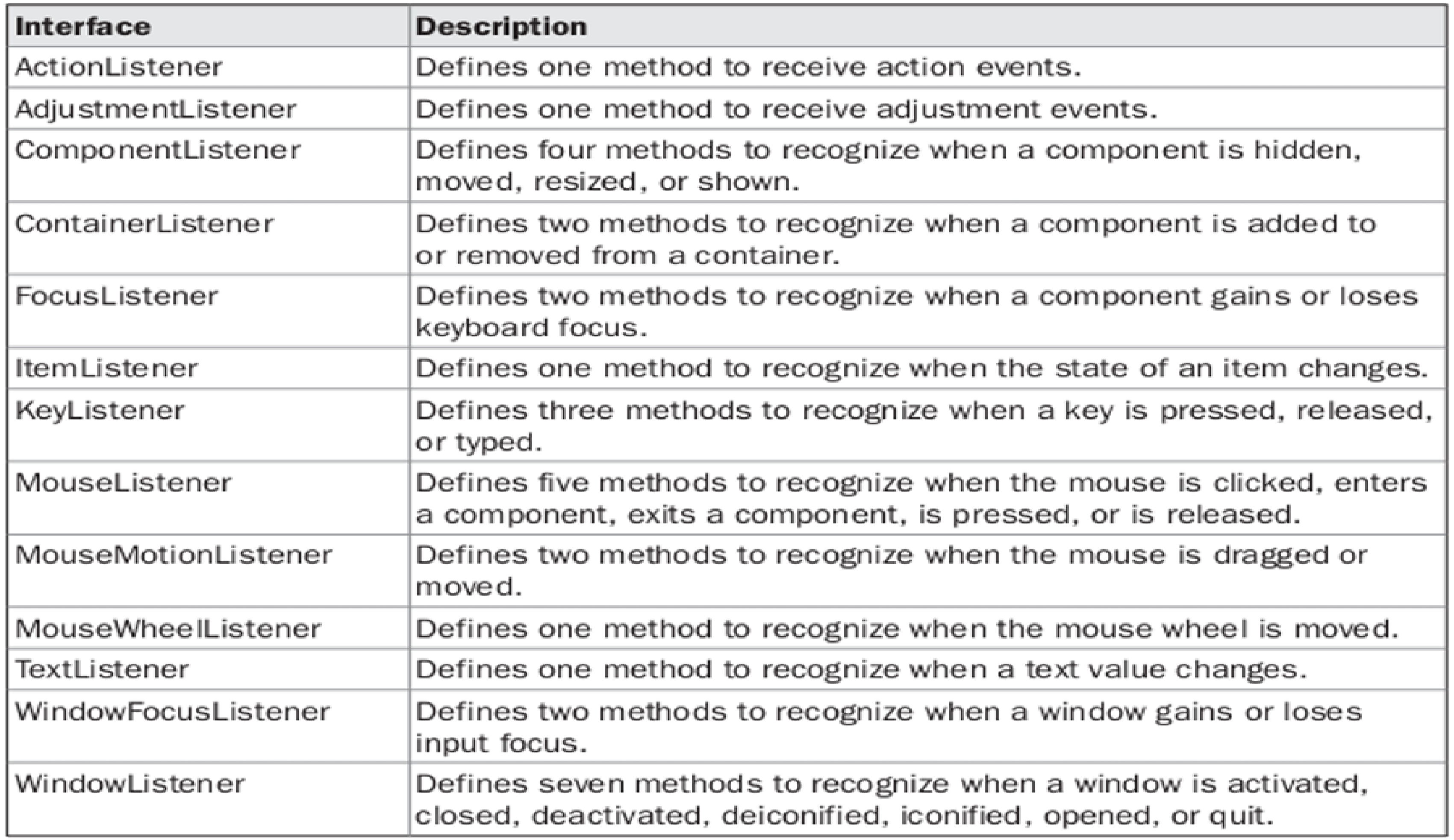
Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Source Examples



Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu Item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listener Interfaces



Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

```
// Java program to demonstrate the
// event handling within the class

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

class FirstSwingDemo extends JFrame implements ActionListener {

    JTextField textField;

    FirstSwingDemo()
    {
       setTitle("First Swing Demo");
        setSize(300, 200);
```

```
 // Component Creation
textField = new JTextField();

// setBounds method is used to provide
// position and size of the component
textField.setBounds(60, 50, 180, 25);
JButton button = new JButton("Click Here");
button.setBounds(150, 80, 80, 30);

// Registering component with listener
// this refers to current instance
button.addActionListener(this);

// add Components
add(textField);
add(button);
```

```java
        // set visibility
        setVisible(true);

           setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    // implementing method of actionListener
    public void actionPerformed(ActionEvent e)
    {
        // Setting text to field
        textField.setText("First Swing Demo");
    }

    public static void main(String[] args)
    {
      new FirstSwingDemo();
    }
}
```