# Event Handling

## What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page are the activities that causes an event to happen.

## Types of Event

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user.They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page etc.
- **Background Events** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

## What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.
- **Listener** - It is also known as event handler.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 an then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model ,Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

## Steps involved in event handling

- The User clicks the button and the event is generated.

- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- the method is now get executed and returns.

# Event Classes

## ActionEvent Class

This class is defined in java.awt.event package. The ActionEvent is generated when button is clicked or the item of a list is double clicked.

# Class methods

| S.N. | Method & Description |
|---|---|
| 1 | **java.lang.String getActionCommand()**<br>Returns the command string associated with this action. |
| 2 | **int getModifiers()**<br>Returns the modifier keys held down during this action event. |
| 3 | **long getWhen()**<br>Returns the timestamp of when this event occurred. |
| 4 | **java.lang.String paramString()**<br>Returns a parameter string identifying this action event. |

## KeyEvent Class

On entering the character the Key event is generated. There are three types of key events which are represented by the integer constants. These key events are following

- KEY_PRESSED
- KEY_RELASED
- KEY_TYPED

# Class methods

| S.N. | Method & Description |
|---|---|
| 1 | **char getKeyChar()**<br>Returns the character associated with the key in this event. |
| 2 | **int getKeyCode()**<br>Returns the integer keyCode associated with the key in this event. |

| | |
|---|---|
| 3 | **int getKeyLocation()**<br>Returns the location of the key that originated this key event. |
| 4 | **static String getKeyModifiersText(int modifiers)**<br>Returns a String describing the modifier key(s), such as "Shift", or "Ctrl+Shift". |
| 5 | **static String getKeyText(int keyCode)**<br>Returns a String describing the keyCode, such as "HOME", "F1" or "A". |
| 6 | **boolean isActionKey()**<br>Returns whether the key in this event is an "action" key. |
| 7 | **String paramString()**<br>Returns a parameter string identifying this event. |
| 8 | **void setKeyChar(char keyChar)**<br>Set the keyChar value to indicate a logical character. |
| 9 | **void setKeyCode(int keyCode)**<br>Set the keyCode value to indicate a physical key. |

## MouseEvent Class

This event indicates a mouse action occurred in a component. This low-level event is generated by a component object for Mouse Events and Mouse motion events.

- a mouse button is pressed
- a mouse button is released
- a mouse button is clicked (pressed and released)
- a mouse cursor enters the unobscured part of component's geometry
- a mouse cursor exits the unobscured part of component's geometry
- a mouse is moved
- the mouse is dragged

# Class methods

| S.N. | Method & Description |
|---|---|
| 1 | **int getButton()**<br>Returns which, if any, of the mouse buttons has changed state. |
| 2 | **int getClickCount()**<br>Returns the number of mouse clicks associated with this event. |
| 3 | **Point getLocationOnScreen()**<br>Returns the absolute x, y position of the event. |
| 4 | **static String getMouseModifiersText(int modifiers)** |

| | |
|---|---|
| | Returns a String describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift". |
| 5 | **Point getPoint()**<br>Returns the x,y position of the event relative to the source component. |
| 6 | **int getX()**<br>Returns the horizontal x position of the event relative to the source component. |
| 7 | **int getXOnScreen()**<br>Returns the absolute horizontal x position of the event. |
| 8 | **int getY()**<br>Returns the vertical y position of the event relative to the source component. |
| 9 | **int getYOnScreen()**<br>Returns the absolute vertical y position of the event. |
| 10 | **boolean isPopupTrigger() Returns whether or not this mouse event is the popup menu trigger event for the platform.** |
| 11 | **String paramString()**<br>Returns a parameter string identifying this event. |
| 12 | **void translatePoint(int x, int y)**<br>Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets. |

## TextEvent Class

The object of this class represents the text events.The TextEvent is generated when character is entered in the text fields or text area. The TextEvent instance does not include the characters currently in the text component that generated the event rather we are provided with other methods to retrieve that information.

# Class methods

| S.N. | Method & Description |
|---|---|
| 1 | **String paramString()**<br>Returns a parameter string identifying this text event. |

## WindowEvent Class

The object of this class represents the change in state of a window.This low-level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or deiconified, or when focus is transfered into or out of the Window.

# Class methods

| S.N. | Method & Description |
|---|---|
| 1 | **int getNewState()**<br>For WINDOW_STATE_CHANGED events returns the new state of the window. |
| 2 | **int getOldState()**<br>For WINDOW_STATE_CHANGED events returns the previous state of the window. |
| 3 | **Window getOppositeWindow()**<br>Returns the other Window involved in this focus or activation change. |
| 4 | **Window getWindow()**<br>Returns the originator of the event. |
| 5 | **String paramString()**<br>Returns a parameter string identifying this event. |

# Event Listeners

## Event Listeners Interface

It is a marker interface which every listener interface has to extend.This class is defined in **java.util package**.

1. **ActionListener Interface**
2. **ItemListener Interface**
3. **KeyListener Interface**
4. **MouseListener Interface**
5. **MouseMotionListener Interface**
6. **TextListener Interface**
7. **WindowListener Interface**

# 1. ActionListener Interface

The class which processes the ActionEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addActionListener() method. When the action event occurs, that object's actionPerformed method is invoked.

## Interface declaration

Following is the declaration for **java.awt.event.ActionListener** interface:

```
public interface ActionListener
   extends EventListener
```

## Interface methods

| S.N. | Method & Description |
|---|---|
| 1 | **void actionPerformed(ActionEvent e)**<br>Invoked when an action occurs. |

## Methods inherited

This interface inherits methods from the following interfaces:

- java.awt.EventListener

## ActionListener Example

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showActionListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
```

```
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });

      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showActionListenerDemo(){
      headerLabel.setText("Listener in action: ActionListener");

      ScrollPane panel = new ScrollPane();
      panel.setBackground(Color.magenta);

      Button okButton = new Button("OK");

      okButton.addActionListener(new CustomActionListener());
      panel.add(okButton);
      controlPanel.add(panel);

      mainFrame.setVisible(true);
   }

   class CustomActionListener implements ActionListener{

      public void actionPerformed(ActionEvent e) {
         statusLabel.setText("Ok Button Clicked.");
      }
   }
}
```

**Output :**


## 2. ItemListener Interface

The class which processes the ItemEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addItemListener() method. When the action event occurs, that object's itemStateChanged method is invoked.

### Interface declaration

Following is the declaration for **java.awt.event.ItemListener** interface:

```
public interface ItemListener
   extends EventListener
```

### Interface methods

| S.N. | Method & Description |
|---|---|
| 1 | **void itemStateChanged(ItemEvent e)**<br>Invoked when an item has been selected or deselected by the user. |

### Methods inherited

This interface inherits methods from the following interfaces:

- java.awt.EventListener

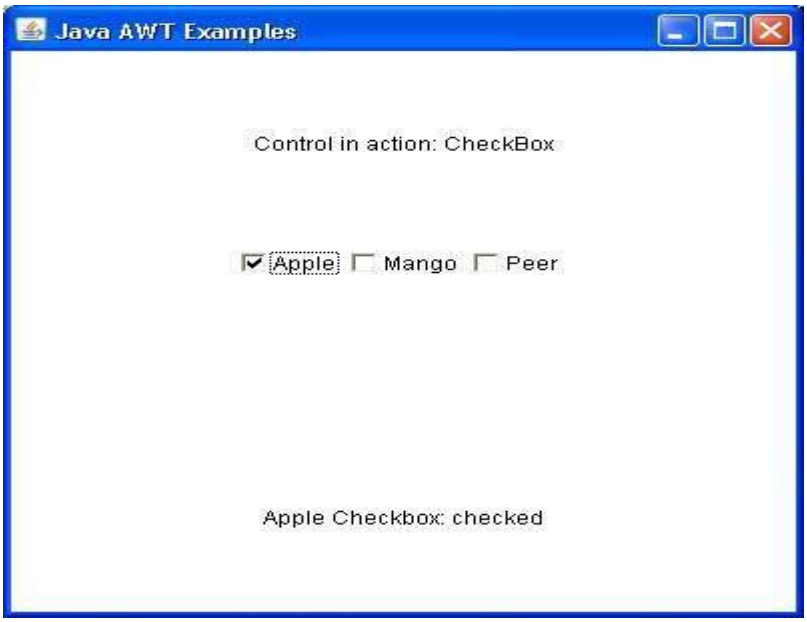
### ItemListener Example

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
```

```
public AwtListenerDemo(){
   prepareGUI();
}

public static void main(String[] args){
   AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
   awtListenerDemo.showItemListenerDemo();
}

private void prepareGUI(){
   mainFrame = new Frame("Java AWT Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));
   mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
         System.exit(0);
      }
   });

   headerLabel = new Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

   controlPanel = new Panel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}

private void showItemListenerDemo(){
   headerLabel.setText("Listener in action: ItemListener");
   Checkbox chkApple = new Checkbox("Apple");
   Checkbox chkMango = new Checkbox("Mango");
   Checkbox chkPeer = new Checkbox("Peer");

   chkApple.addItemListener(new CustomItemListener());
   chkMango.addItemListener(new CustomItemListener());
   chkPeer.addItemListener(new CustomItemListener());

   controlPanel.add(chkApple);
   controlPanel.add(chkMango);
   controlPanel.add(chkPeer);
   mainFrame.setVisible(true);
}

class CustomItemListener implements ItemListener {
   public void itemStateChanged(ItemEvent e) {
      statusLabel.setText(e.getItem()
```

```
         +" Checkbox: "
         + (e.getStateChange()==1?"checked":"unchecked"));
      }
   }
}
```

**Output :**




## 3. KeyListener Interface

The class which processes the KeyEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addKeyListener() method.

## Interface declaration

Following is the declaration for **java.awt.event.KeyListener** interface:

```
public interface KeyListener
   extends EventListener
```

## Interface methods

| S.N. | Method & Description |
|---|---|
| 1 | **void keyPressed(KeyEvent e)**<br>Invoked when a key has been pressed. |

| | |
|---|---|
| 2 | **void keyReleased(KeyEvent e)**<br>Invoked when a key has been released. |
| 3 | **void keyTyped(KeyEvent e)**<br>Invoked when a key has been typed. |

## Methods inherited

This interface inherits methods from the following interfaces:

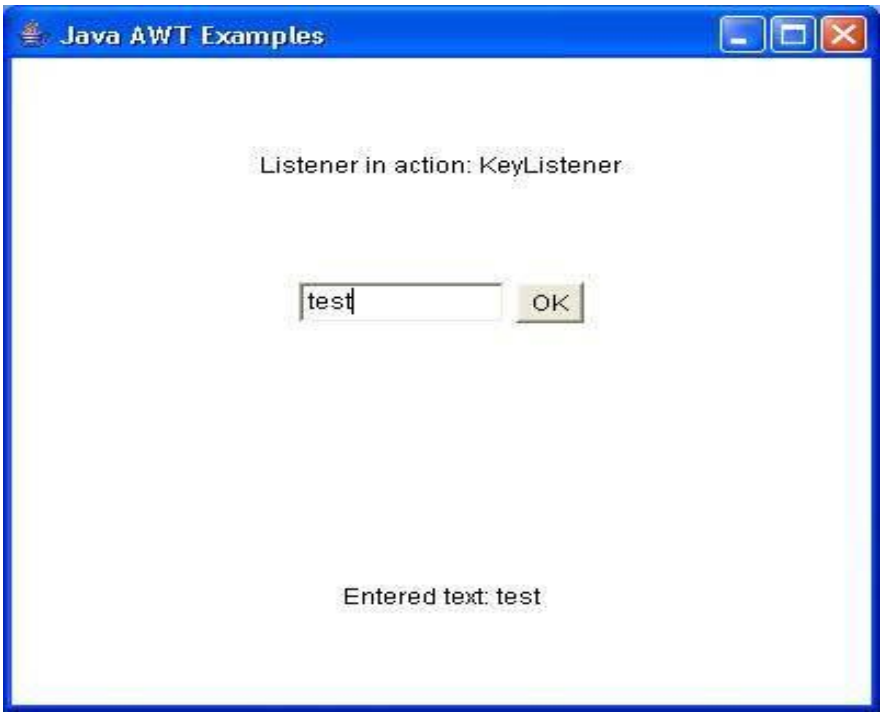
- java.awt.EventListener

## KeyListener Example

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private TextField textField;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showKeyListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });

      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
```

```
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showKeyListenerDemo(){
      headerLabel.setText("Listener in action: KeyListener");

      textField  = new TextField(10);

      textField.addKeyListener(new CustomKeyListener());
      Button okButton = new Button("OK");
      okButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            statusLabel.setText("Entered text: " + textField.getText());
         }
      });

      controlPanel.add(textField);
      controlPanel.add(okButton);
      mainFrame.setVisible(true);
   }

   class CustomKeyListener implements KeyListener{
      public void keyTyped(KeyEvent e) {
      }

      public void keyPressed(KeyEvent e) {
         if(e.getKeyCode() == KeyEvent.VK_ENTER){
            statusLabel.setText("Entered text: " + textField.getText());
         }
      }

      public void keyReleased(KeyEvent e) {
      }
   }
}
```

**Output :**




# 4. MouseListener Interface

The class which processes the MouseEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addMouseListener() method.

## Interface declaration

Following is the declaration for **java.awt.event.MouseListener** interface:

```
public interface MouseListener
   extends EventListener
```

## Interface methods

| S.N. | Method & Description |
|---|---|
| 1 | **void mouseClicked(MouseEvent e)**<br>Invoked when the mouse button has been clicked (pressed and released) on a component. |
| 2 | **void mouseEntered(MouseEvent e)**<br>Invoked when the mouse enters a component. |
| 3 | **void mouseExited(MouseEvent e)**<br>Invoked when the mouse exits a component. |
| 4 | **void mousePressed(MouseEvent e)** |

| | |
|---|---|
| | Invoked when a mouse button has been pressed on a component. |
| 5 | **void mouseReleased(MouseEvent e)**<br>Invoked when a mouse button has been released on a component. |

## Methods inherited

This interface inherits methods from the following interfaces:

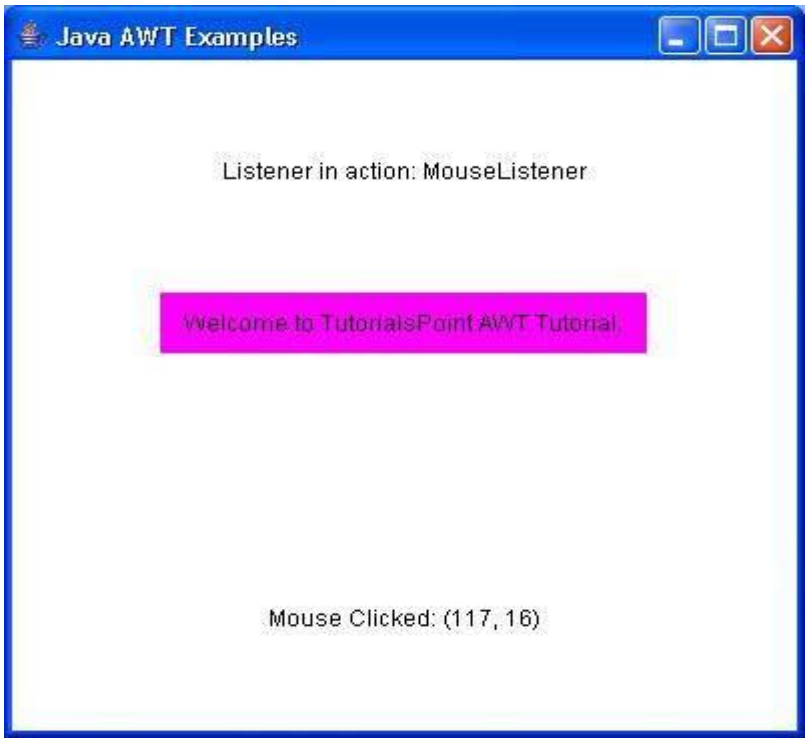
- java.awt.EventListener

## MouseListener Example

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showMouseListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });

      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
```

```
      mainFrame.setVisible(true);
   }

   private void showMouseListenerDemo(){
      headerLabel.setText("Listener in action: MouseListener");

      Panel panel = new Panel();
      panel.setBackground(Color.magenta);
      panel.setLayout(new FlowLayout());
      panel.addMouseListener(new CustomMouseListener());

      Label msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

      msglabel.addMouseListener(new CustomMouseListener());
      panel.add(msglabel);

      controlPanel.add(panel);

      mainFrame.setVisible(true);
   }
   class CustomMouseListener implements MouseListener{

      public void mouseClicked(MouseEvent e) {
         statusLabel.setText("Mouse Clicked: ("
         +e.getX()+", "+e.getY() +")");
      }

      public void mousePressed(MouseEvent e) {
      }

      public void mouseReleased(MouseEvent e) {
      }

      public void mouseEntered(MouseEvent e) {
      }

      public void mouseExited(MouseEvent e) {
      }
   }
}
```

**Output :**




# 5. MouseMotionListener Interface

## Introduction

The interface**MouseMotionListener** is used for receiving mouse motion events on a component. The class that process mouse motion events needs to implements this interface.

## Class declaration

Following is the declaration for **java.awt.event.MouseMotionListener**interface:

```
public interface MouseMotionListener
extends EventListener
```

## Interface methods

| S.N. | Method & Description |
|---|---|
| 1 | **void mouseDragged(MouseEvent e)**<br>Invoked when a mouse button is pressed on a component and then dragged. |
| 2 | **void mouseMoved(MouseEvent e)**<br>Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed. |

## Methods inherited

This class inherits methods from the following interfaces:

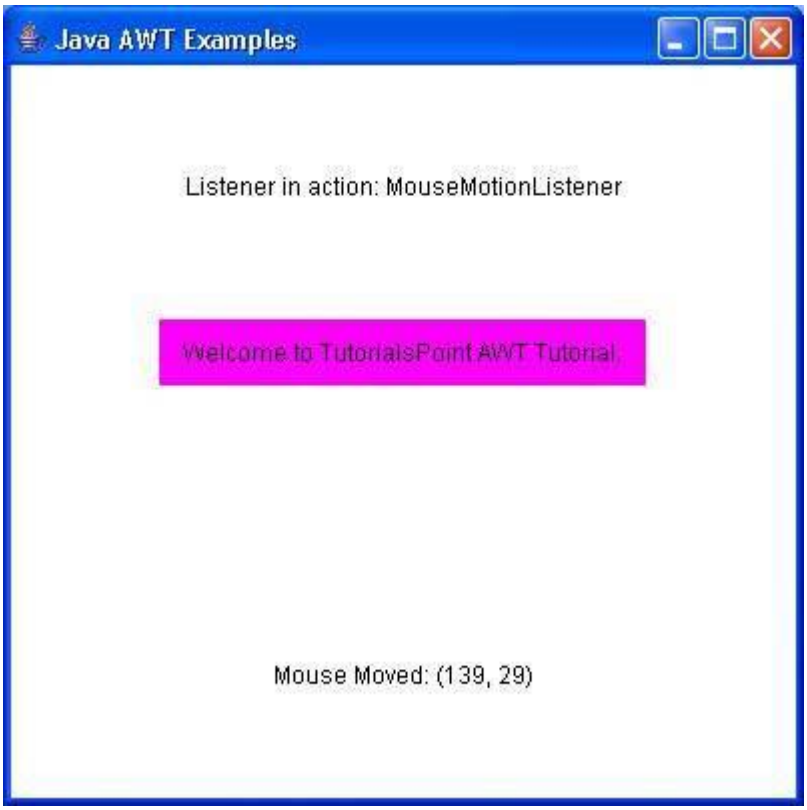
- java.awt.event.EventListener

# MouseMotionListener Example

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showMouseMotionListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });

      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showMouseMotionListenerDemo(){
      headerLabel.setText("Listener in action: MouseMotionListener");

      Panel panel = new Panel();
      panel.setBackground(Color.magenta);
      panel.setLayout(new FlowLayout());
      panel.addMouseMotionListener(new CustomMouseMotionListener());

      Label msglabel = new Label();
```

```
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
      panel.add(msglabel);

      controlPanel.add(panel);

      mainFrame.setVisible(true);
   }

   class CustomMouseMotionListener implements MouseMotionListener {

      public void mouseDragged(MouseEvent e) {
         statusLabel.setText("Mouse Dragged: ("+e.getX()+", "+e.getY() +")");
      }

      public void mouseMoved(MouseEvent e) {
         statusLabel.setText("Mouse Moved: ("+e.getX()+", "+e.getY() +")");
      }
   }
}
```

**Output :**



## 6. TextListener Interface

The class which processes the TextEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addTextListener() method.

## Interface declaration

Following is the declaration for **java.awt.event.TextListener** interface:

```
public interface TextListener

   extends EventListener
```

## Interface methods

| S.N. | Method & Description |
|---|---|
| 1 | **void textValueChanged(TextEvent e)**<br>Invoked when the value of the text has changed. |

## Methods inherited

This interface inherits methods from the following interfaces:

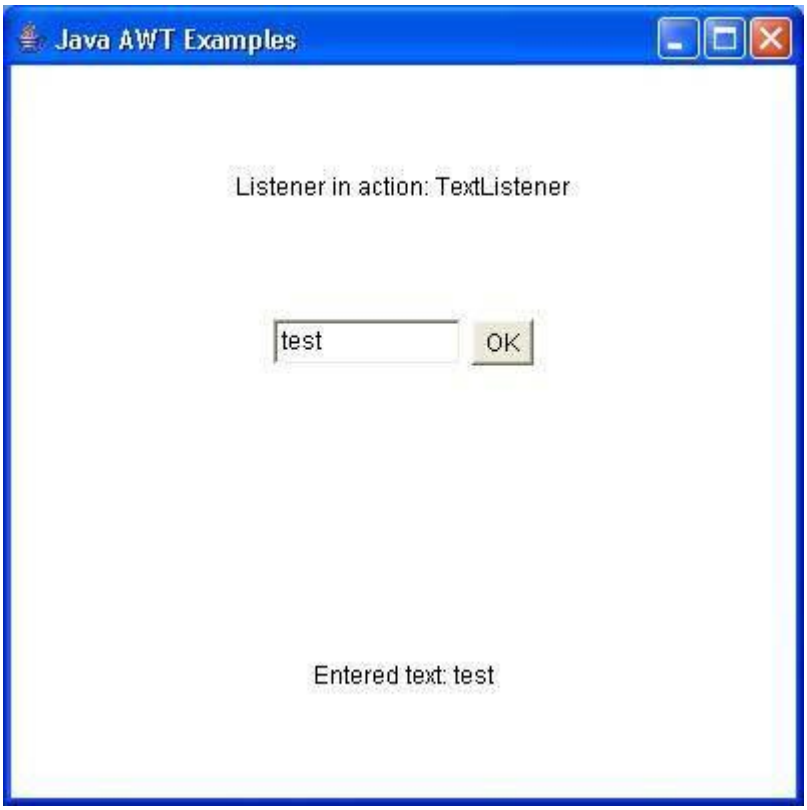
- java.awt.EventListener

## TextListener Example

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private TextField textField;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showTextListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
```

```
      });

      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showTextListenerDemo(){
      headerLabel.setText("Listener in action: TextListener");

      textField  = new TextField(10);

      textField.addTextListener(new CustomTextListener());
      Button okButton = new Button("OK");
      okButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            statusLabel.setText("Entered text: "
            + textField.getText());
         }
      });

      controlPanel.add(textField);
      controlPanel.add(okButton);
      mainFrame.setVisible(true);
   }

   class CustomTextListener implements TextListener {
      public void textValueChanged(TextEvent e) {
         statusLabel.setText("Entered text: " + textField.getText());
      }
   }
}
```

**Output :**




# 7. WindowListener Interface

The class which processes the WindowEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addWindowListener() method.

## Interface declaration

Following is the declaration for **java.awt.event.WindowListener** interface:

```
public interface WindowListener
   extends EventListener
```

## Interface methods

| S.N. | Method & Description |
|---|---|
| 1 | **void windowActivated(WindowEvent e)**<br>Invoked when the Window is set to be the active Window. |
| 2 | **void windowClosed(WindowEvent e)**<br>Invoked when a window has been closed as the result of calling dispose on the window. |
| 3 | **void windowClosing(WindowEvent e)**<br>Invoked when the user attempts to close the window from the window's system menu. |

| | |
|---|---|
| 4 | **void windowDeactivated(WindowEvent e)**<br>Invoked when a Window is no longer the active Window. |
| 5 | **void windowDeiconified(WindowEvent e)**<br>Invoked when a window is changed from a minimized to a normal state. |
| 6 | **void windowIconified(WindowEvent e)**<br>Invoked when a window is changed from a normal to a minimized state. |
| 7 | **void windowOpened(WindowEvent e)**<br>Invoked the first time a window is made visible. |

## Methods inherited

This interface inherits methods from the following interfaces:

- java.awt.EventListener

## WindowListener Example

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

public class AwtListenerDemo {
   private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public AwtListenerDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
      awtListenerDemo.showWindowListenerDemo();
   }

   private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });

      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
```

```
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showWindowListenerDemo(){
      headerLabel.setText("Listener in action: WindowListener");

      Button okButton = new Button("OK");

      aboutFrame = new Frame();
      aboutFrame.setSize(300,200);;
      aboutFrame.setTitle("WindowListener Demo");
      aboutFrame.addWindowListener(new CustomWindowListener());

      Label msgLabel = new Label("Welcome to tutorialspoint.");
      msgLabel.setAlignment(Label.CENTER);
      msgLabel.setSize(100,100);
      aboutFrame.add(msgLabel);
      aboutFrame.setVisible(true);
   }

   class CustomWindowListener implements WindowListener {
      public void windowOpened(WindowEvent e) {
      }

      public void windowClosing(WindowEvent e) {
         aboutFrame.dispose();
      }

      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) {
      }
   }
}
```

**Output :**