**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.

**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. The java.awt.event package provides many event classes and Listener interfaces for event handling.

**Delegation Event Model**

This model defines the standard mechanism to generate and handle the events. It defines standard and consistent mechanisms to generate and process events. Its concept is quite simple 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 an event is received the listener processes the event and then returns. 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. A user interface element is able to “delegate” the processing of an event to a separate piece of code.

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.

**Event Classes**

The Event classes represent the event. Java provides us various Event classes.

|  |  |
| --- | --- |
| **Control** | **Description** |
| [**AWTEvent**](https://www.tutorialspoint.com/awt/awt_awt_event.htm) | It is the root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class. |
| [**ActionEvent**](https://www.tutorialspoint.com/awt/awt_action_event.htm) | The ActionEvent is generated when button is clicked or the item of a list is double clicked. |
| [**InputEvent**](https://www.tutorialspoint.com/awt/awt_input_event.htm) | The InputEvent class is root event class for all component-level input events. |
| [**KeyEvent**](https://www.tutorialspoint.com/awt/awt_key_event.htm) | On entering the character the Key event is generated. |
| [**MouseEvent**](https://www.tutorialspoint.com/awt/awt_mouse_event.htm) | This event indicates a mouse action occurred in a component. |
| [**TextEvent**](https://www.tutorialspoint.com/awt/awt_text_event.htm) | The object of this class represents the text events. |
| [**WindowEvent**](https://www.tutorialspoint.com/awt/awt_window_event.htm) | The object of this class represents the change in state of a window. |
| [**AdjustmentEvent**](https://www.tutorialspoint.com/awt/awt_adjustment_event.htm) | The object of this class represents the adjustment event emitted by Adjustable objects. |
| [**ComponentEvent**](https://www.tutorialspoint.com/awt/awt_component_event.htm) | The object of this class represents the change in state of a window. |
| [**ContainerEvent**](https://www.tutorialspoint.com/awt/awt_container_event.htm) | The object of this class represents the change in state of a window. |
| [**MouseMotionEvent**](https://www.tutorialspoint.com/awt/awt_mousemotion_event.htm) | The object of this class represents the change in state of a window. |
| [**PaintEvent**](https://www.tutorialspoint.com/awt/awt_paint_event.htm) | The object of this class represents the change in state of a window. |

**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.

**Steps to perform Event Handling**

Following steps are required to perform event handling:

* Implement the Listener interface and overrides its methods
* Register the component with the Listener(ActionListener, MouseListener, KeyListener, WindowListener)

**Example of Mouse Event Handling**

import javax.swing.\*;

import java.awt.event.\*;

public class JFrameClickEventExample extends JFrame implements MouseListener

{

public JLabel label;

public static void main(String args[])

{

new JFrameClickEventExample();

}

JFrameClickEventExample()

{

setSize(500, 400);

setTitle("Java Swing - JFrame Detect Mouse Events");

label = new JLabel("No Mouse Event Captured", JLabel.CENTER);

add(label);

addMouseListener(this);

setVisible(true);

}

//These methods are called automatically when corresponding mouse event occurs

public void mousePressed(MouseEvent e)

{ label.setText("Mouse Pressed");

}

public void mouseEntered(MouseEvent e)

{ label.setText("Mouse Entered");

}

public void mouseClicked(MouseEvent e)

{

label.setText("Mouse Clicked");

}

}

**Example of event handling within class:**

import java.awt.\*;

import java.awt.event.\*;

class AEvent extends Frame implements ActionListener

{

TextField tf;

AEvent()

{

tf=new TextField();

tf.setBounds(60,50,170,20);

Button b=new Button("click me");

b.setBounds(100,120,80,30);

b.addActionListener(this);

add(b);add(tf);

setSize(300,300);

setLayout(null);

setVisible(true);

}

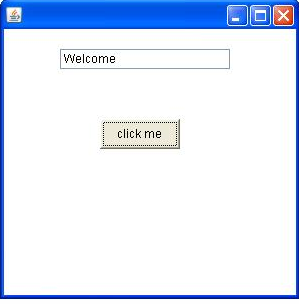
public void actionPerformed(ActionEvent e)

{ tf.setText("Welcome"); }

public static void main(String args[])

{ new AEvent(); }

}



**Swing**

It is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. It acts as replacement of AWT API as it has almost every control corresponding to AWT controls.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Swing API is set of extensible GUI Components to ease developer's life to create JAVA based Front End/ GUI Applications. It is used to create window-based applications. Swing component follows a Model-View-Controller architecture to fulfill the following criteria’s.

* A single API is to be sufficient to support multiple look and feel.
* API is to model driven so that highest level API is not required to have the data.
* API is to use the Java Bean model so that Builder Tools and IDE can provide better services to the developers to use it.

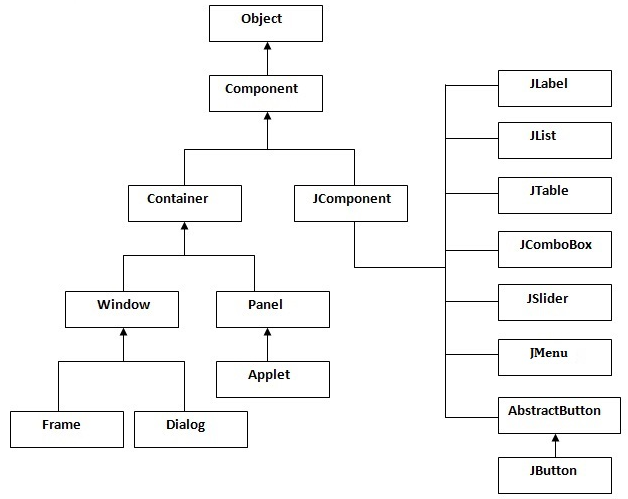
**Swing features**

* **Light Weight** - Swing component are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
* **Rich controls** - Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, color picker, table controls.
* **Highly Customizable** - Swing controls can be customized in very easy way as visual appearance is independent of internal representation.
* **Pluggable look-and-feel**- SWING based GUI Application look and feel can be changed at run time based on available values.

**Difference between AWT and Swing**

|  |  |
| --- | --- |
| **Java AWT** | **Java Swing** |
| AWT components are **platform-dependent**. | Java swing components are **platform-independent**. |
| AWT components are **heavyweight**. | Swing components are **lightweight**. |
| AWT **doesn't support pluggable look and feel**. | Swing **supports pluggable look and feel**. |
| AWT provides **less components** than Swing. | Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. |
| AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. | Swing **follows MVC**. |

**Hierarchy of Java Swing classes**



**Commonly used Methods of Component class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void add(Component c) | add a component on another component. |
| public void setSize(int width,int height) | sets size of the component. |
| public void setLayout(LayoutManager m) | sets the layout manager for the component. |
| public void setVisible(boolean b) | sets the visibility of the component. It is by default false. |

There are two ways to create a frame:

* By creating the object of Frame class (association)
* By extending Frame class (inheritance)

Simple Java Swing example(*FirstSwingExample.java*)

Where we are creating one button and adding it on the JFrame object inside the main() method.

|  |
| --- |
| 1. **import** javax.swing.\*; 2. **public** **class** FirstSwingExample { 3. **public** **static** **void** main(String[] args) { 4. JFrame f=**new** JFrame();//creating instance of JFrame 5. JButton b=**new** JButton("click");//creating instance of JButton 6. b.setBounds(130,100,100, 40);//x axis, y axis, width, height 7. f.add(b);//adding button in JFrame   f.setSize(400,500);//400 width and 500 height  f.setLayout(**null**);//using no layout managers  f.setVisible(**true**);//making the frame visible  }  } |

**Example of Swing by Association inside constructor**

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

**import** javax.swing.\*;

**public** **class** Simple

{

JFrame f;

Simple()

{

f=**new** JFrame();//creating instance of JFrame

JButton b=**new** JButton("click");//creating instance of JButton

b.setBounds(130,100,100, 40);

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height

f.setLayout(**null**);//using no layout managers

f.setVisible(**true**);//making the frame visible

}

**public** **static** **void** main(String[] args)

{

**new** Simple();

}

}

**Simple example of Swing by inheritance**

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

**import** javax.swing.\*;

**public** **class** Simple2 **extends** JFrame

{ JFrame f;

Simple2()

{

JButton b=**new** JButton("click");//create button

b.setBounds(130,100,100, 40);

add(b);//adding button on frame

setSize(400,500);

setLayout(**null**);

setVisible(**true**);

}

**public** **static** **void** main(String[] args)

{  **new** Simple2();

}

}

**Example: When click on button, display full name by taking first name and last name from text boxes**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class SwingControlDemo1 {

private JFrame mainFrame;

private JLabel statusLabel;

private JPanel controlPanel;

public SwingControlDemo1(){

prepareGUI();

}

public static void main(String[] args){

SwingControlDemo1 swingControlDemo = new SwingControlDemo1();

swingControlDemo.showTextFieldDemo();

}

private void prepareGUI(){

mainFrame = new JFrame("Java Swing Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

statusLabel = new JLabel("",JLabel.CENTER);

statusLabel.setSize(350,100);

controlPanel = new JPanel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showTextFieldDemo(){

JLabel firstnamelabel= new JLabel("First Name : ", JLabel.RIGHT);

JLabel lastNameLabel = new JLabel("Last Name : ", JLabel.CENTER);

final JTextField firstNameText = new JTextField(6);

final JTextField lastNameText = new JTextField(6);

JButton displayButton= new JButton("Display");

displayButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String data = "Full name: " + firstNameText.getText() + " " +lastNameText.getText();

statusLabel.setText(data);

}

});

controlPanel.add(firstnamelabel);

controlPanel.add(firstNameText);

controlPanel.add(lastNameLabel);

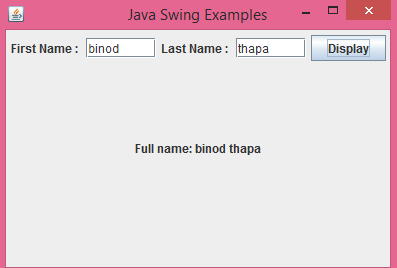
controlPanel.add(lastNameText);

controlPanel.add(displayButton);

mainFrame.setVisible(true);

}

}



**Java AWT (Abstract Window Toolkit)**

Java AWT is an API to develop GUI (Graphical User Interface) or window-based application in java.

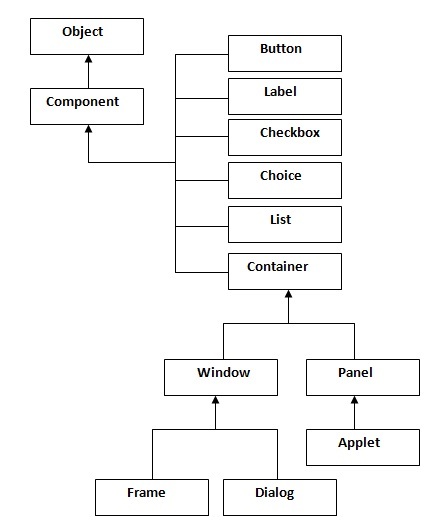
Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components uses the resources of system.

It offers user interaction via some graphical components

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

**Java AWT Hierarchy**

The hierarchy of Java AWT classes are given below.



**Container**

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

**Window**

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

**Panel**

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

**Frame**

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

**Useful Methods of Component class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void add(Component c) | inserts a component on this component. |
| public void setSize(int width,int height) | sets the size (width and height) of the component. |
| public void setLayout(LayoutManager m) | defines the layout manager for the component. |
| public void setVisible(boolean status) | changes the visibility of the component, by default false. |

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

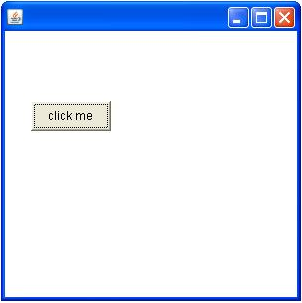
By extending Frame class (inheritance)

By creating the object of Frame class (association)

**Simple example of AWT by inheritance**

|  |
| --- |
| 1. **import** java.awt.\*; 2. **class** First **extends** Frame{ 3. First(){ 4. Button b=**new** Button("click me"); 5. b.setBounds(30,100,80,30);// setting button position 6. add(b);//adding button into frame 7. setSize(300,300);//frame size 300 width and 300 height 8. setLayout(**null**);//no layout manager   setVisible(**true**);//now frame will be visible, by default not visible  }  **public** **static** **void** main(String args[]){  First f=**new** First();  }} |

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.



**Simple example of AWT by association**

|  |
| --- |
| 1. **import** java.awt.\*; 2. **class** First2{ 3. First2(){ 4. Frame f=**new** Frame(); 5. Button b=**new** Button("click me"); 6. b.setBounds(30,50,80,30); 7. f.add(b);   f.setSize(300,300);  f.setLayout(**null**);  f.setVisible(**true**);  }  **public** **static** **void** main(String args[]){  First2 f=**new** First2();  }} |