

Java AWT Tutorial

1. Graphical User Interfaces
2. What is AWT in java
3. Describe the AWT Components
4. AWT Components
 1. [Canvas](#)
 2. [Checkbox](#)
 3. [Label](#)
 4. [Scrollbar](#)
 5. [TextField](#)
5. Common Component Methods
6. Containers
7. Describe different types of event to provide by the AWT Components in java
8. Events
9. Event Adapters
10. Button Pressing Example
11. Adapters Example
12. GUI-based Applications ← this presentation makes it to here as of 04/08/2008
13. Menus
14. Menu Shortcuts
15. Pop-up Menus
16. Creating a Frame
17. Setting the Icon for a Frame
18. Setting the close button operation for the frame
19. Making a Frame Non-Resizable
20. Removing the Title Bar of a Frame
+ 10 more examples

Used this site <http://www.roseindia.net/java/example/java/awt/>

As well as Deitel & Deitel Small Java How To Program 6th edition

AWT - Building a graphical interface

- Terminology
 - AWT – Abstract Windows Toolkit
 - GUI – Graphical User Interface

What is AWT

- **Interfaces and Descriptions of AWT Package:**

- **ActionEvent** This interface is used for handling events.
- **Adjustable** This interface takes numeric value to adjust within the bounded range.
- **Composite** This interface defines methods to draw a graphical area. It combines a shape, text, or image etc.
- **CompositeContext** This interface allows the existence of several context simultaneously for a single composite object. It handles the state of the operations.
- **ItemSelectable** This interface is used for maintaining zero or more selection for items from the item list.
- **KeyEventDispatcher** The KeyEventDispatcher implements the current KeyboardFocusManager and it receives KeyEvents before dispatching their targets.
- **KeyEventPostProcessor** This interface also implements the current KeyboardFocusManager. The KeyboardFocusManager receives the KeyEvents after that dispatching their targets.
- **LayoutManager** It defines the interface class and it has layout containers.
- **LayoutManager2** This is the interface extends from the LayoutManager and is subinterface of that.
- **MenuContainer** This interface has all menu containers.
- **Paint** This interface is used to [color](#) pattern. It used for the Graphics2D operations.
- **PaintContext** This interface also used the color pattern. It provides an important color for the Graphics2D operation and uses the ColorModel.
- **PaintGraphics** This interface provides [print](#) a graphics context for a page.
- **Shape** This interface used for represent the geometric shapes.
- **Stroke** This interface allows the Graphics2D object and contains the shapes to outline or stylistic representation of outline.
- **Transparency** This interface defines the transparency mode for implementing classes.

What is AWT

- **Classes and Descriptions of AWT Package:**

- **AlphaComposite** This class implements the basic alpha compositing rules. It combines the source and destination pixels to achieve transparency effects to graphics and images.
- **AWTEvent** This is a super class of all AWT Events.
- **AWTEventMulticaster** This class implements thread-safe multi-cast event and it is despatching for the AWT event. The AWT events defined in the java.awt.event package.
- **AWTKeyStroke** This class used to key action on the keyboard or equivalent input devices.
- **AWTPermission** This class uses for the AWT permissions.
- **BasicStroke** This class defines the basic set of rendering attributes for using outlines of graphics.
- **BorderLayout** This class uses to arranging the components. It has five components such as: east, west, north, south and the center.
- **BufferCapabilities** This class has properties of buffers.BufferCapabilities.
- **FlipContents** This class has a type-safe enumeration of buffer. It contains after page-flipping.
- **Button** This class used to create a label button
- **Canvas** It represents the blank rectangular area on screen. It can draw or trap input events from the user.
- **CardLayout** It is a layout manager for a container.
- **Checkbox** It is a graphical component. It has two states. True state that means "on" or false state that means "off".
- **CheckboxGroup** This class to be used together multiple checkbox buttons.
- **CheckboxMenuItem** This class represents the checkbox and also include the menu.
- **Choice** This class represents pop-up menu to user's choice.
- **Color** This class has colors. The default color is RGB color. Color library specify the all color, it identified by ColorSpace.
- **Component** This is a graphical representation to interacted by user. It displays on the screen.
- **ComponentOrientation** This class encapsulates the language-sensitive orientation. It also used the order the element of component or text..
- **Container** A generic AWT container object has other AWT components.

What is AWT

- **Classes and Descriptions of AWT Package:**

- **ContainerOrderFocusTraversalPolicy** It determines the traversal order based on the order of child components in a container.
- **Cursor** This class represents the bitmap representation of the mouse cursor.
- **DefaultFocusTraversalPolicy** This class determines the traversal order on the order of child components of container.
- **DefaultKeyboardFocusManager** This class used for handle the AWT applications.
- **Dialog** This is a top label window. It has title and border. It can be used for taking a some input of users.
- **Dimension** This class describe the height and width of a component in a single object.
- **DisplayMode** This class encapsulates the bit depth, height, width and refresh rate of a GraphicsDevice.
- **Event** This class available only for the backwards compatibility.
- **EventQueue** It is a platform independent class. It has both classes underlying peer class and trusted application class.
- **FileDialog** This class displays dialog window. Here user can be select the file.
- **FlowLayout** This class arrange the components and flow the left to right. It uses to write lines in a paragraph.
- **FocusTraversalPolicy** This class defines the order in which components traverse particular focus cycle root.
- **Font** This class defines [fonts](#) and it uses render text that is visible.
- **FontMetrics** This class defines font matrix object.. It encapsulate the information and rendering the paritcular fonts.
- **Frame** This class defines top-level window and it designs the any area of border.
- **GradientPaint** With the help of GradientPaint you fill any shapes.
- **Graphics** This class uses to drawing all types of graphics such as: oval, rectangle etc. Graphics2D This class controls all geometry, coordinate transformation, color management etc. It extends form the Graphics class.
- **GraphicsConfigTemplate** This class contains a valid GraphicConfiguration.
- **GraphicsConfiguration** This class describes the characteristics of graphics destination such as [printer](#) and monitor.
- **GraphicsDevice** This class describes the graphics devices and it available particular graphics environment.
- **GraphicsEnvironment** This class is a collection of GraphicsDevices object and Font objects. The GraphicsDevices objects are screen, images and printers etc.
- **GridBagConstraints** This class specify the constraint for components by using the GrideBagLayout class.
- **GridBagLayout** This class uses the layout manager and uses the vertically and horizontally components.
- **GridLayout** This class is a layout manager. It has rectangular grid components. Image This class is a super class of all graphical images.
- **ImageCompabilities** It has compabilities and properties of images. Insets This class represents all types of border's container. It includes borders, blank space and titles.

What is AWT

- **Classes and Descriptions of AWT Package:**

- **JobAttributes** This class control the print job. JobAttributes.
- **DefaultSelectionType** It has default selection states and extends from the java.awt.AttributeValue package. JobAttributes.
- **DestinatinType** It possible for the job destinations and extends form the java.awt.AttributeValue package. JobAttributes.
- **DialogType** It displays the user dialog and extends from the java.awt.AttributeValue package. JobAttributes.
- **MultipleDocumentHandlingType** This class handles the multiple copy states and extends form the java.awt.AttributeValue package. JobAttributes.
- **SidesType** It uses multi-page impositions and extends from the java.awt.AttributeValue package.
- **KeyboardFocusManager** This class manage the current focus owner, active and focused windows
- **Label** It is a component which contains the text in container.
- **List** This component uses by the uses and it choose the list of item.
- **MediaTracker** This class has status of a number of media objects. It is a utility class.
- **Menu** It has pull-down menu components that displayed as like menu bar.
- **MenuBar** This class has the concept of menu bar and it also bounded into a frame.
- **MenuComponent** This is super class of all menu related components.
- **MenuItem** This is a super class and it represents the item of menu.
- **MenuShortcut** This class represents the handling MenuItem through help of keyboard .
- **PageAttributes** It controls the output of the printed page.
- **ColorType** It handles the color states and extends form the java.awt.AttributeValue package. PageAttributes. MediaTypelt handles the paper size and extends from the java.awt.AttributeValue package. PageAttributes.
- **OrientationRequestedType** It handles the possible orientations and extends from the java.awt.AttributeValue package PageAttributes.
- **OriginType** It handles the origins and extends from the java.awt.AttributeValue package. PageAttributes.
- **PrintQualityType** It handles the [print qualities](#) and extends from the java.awt.AttributeValue package.
- **Panel** This is a simplest container class. It includes components and other panels. It extends Container and implements to Accessible.
- **Point** The point represents the location of coordinate (x, y) space. It extends Point2D.
- **Polygon** It has two dimensional region and it bounded by the multiple number of lines.
- **PopupMenu** It extends the Menu and specify the positions of components.
- **PrintJob** This class executes a print job and extends from the Object.
- **Rectangle** A rectangle object has length and width and it also specify an area in a coordinate space. It extends Rectangle2D.
- **RenderingHints** This class contains rendering hints by using the Graphics2D class. RenderingHints.
- **Key** This class used to control the randering and imaging pipelines.
- **Robot** This class used to generate the native system input events and it automatically test the [java platform](#) implementations.
- **Scrollbar** This class provide the user interface components and also include the scroll bar. Which implements the Adjustable interface.
- **ScrollPane** It includes the horizontal and vertical scrolling for a single child components. The horizontal and vertical state represented by the ScrollPaneAdjustable objects.
- **ScrollPaneAdjustable** This class represents the state of horizontal and vertical scrollbar of ScrollPane.
- **SystemColor** This class represents the system's color through the symbolic representation color. The value depends on the actual value of RGB.

What is AWT

- **Classes and Descriptions of AWT Package:**

- **TextArea** It displays multi line text.
- **TextComponent** This is a super class of any component. It allows to editing the some text.
- **TextField** It has text component and It allows to editing a single line of text.
- **TexturePaint** It provides a way to fill a shape with a texture and specify by the BufferedImage.
- **Toolkit** This is a super class of all Abstract Windowing Toolkit.
- **Window** It is a top-level window. It has no borders and menubar. It is capable of generating the window events like: WindowOpend, WindowClosed.

- **Exceptions and Descriptions of AWT Package:**

- **AWTException** This signal displays when an Abstract Windowing Toolkit exception has occurred.
- **FontFormatException** When the specified font is bad then this exception to be occurred. It thrown by the createFont method in the Font class.
- **HeadlessException** This exception to be occurs when the codes are not supported by the keyboard, display and Mouse.
- **IllegalComponentStateException** The AWT has not suitable state for the requesting operation then it thrown by the IllegalComponentStateException.
- **AWTError** It thrown when the Abstract Windowing Toolkit error has occurred.

Components – Label and Button

- **Labels** : This is the simplest component of Java Abstract Window Toolkit. This component is generally used to show the text or string in your [application](#) and label never perform any type of action. Syntax for defining the label only and with justification :

```
Label label_name = new Label ("This is the label text.");
```

Above code simply represents the text for the label.

```
Label label_name = new Label ("This is the label text.", Label.CENTER);
```

Justification of label can be left, right or centered. Above declaration used the center justification of the label using the Label.CENTER.

- **Buttons** : This is the component of Java Abstract Window Toolkit and is used to trigger actions and other events required for your application. The syntax of defining the button is as follows :

```
Button button_name = new Button ("This is the label of the button.");
```

You can change the Button's label or get the label's text by using the [Button.setLabel\(String\)](#) and [Button.getLabel\(\)](#) method. Buttons are added to the it's container using the [add\(button_name\)](#) method.

Components – Check Boxes and Radial Button

- **Check Boxes** : This component of Java AWT allows you to create check boxes in your applications. The syntax of the definition of Checkbox is as follows :

```
Checkbox checkbox_name = new Checkbox ("Optional check box 1", false);
```

Above code constructs the unchecked Checkbox by passing the boolean valued argument *false* with the Checkbox label through the `Checkbox()` constructor. Defined Checkbox is added to it's container using `add (checkbox_name)` method. You can change and get the checkbox's label using the `setLabel (String)` and `getLabel()` method. You can also set and get the state of the checkbox using the `setState(boolean)` and `getState()` method provided by the `Checkbox` class.

- **Radio Button** : This is the special case of the Checkbox component of Java AWT package. This is used as a group of checkboxes which group name is same. Only one Checkbox from a Checkbox Group can be selected at a time. Syntax for creating radio buttons is as follows :

```
CheckboxGroup chkgp = new CheckboxGroup();  
add (new Checkbox ("One", chkgp, false);  
add (new Checkbox ("Two", chkgp, false);  
add (new Checkbox ("Three", chkgp, false);
```

In the above code we are making three check boxes with the label "One", "Two" and "Three". If you mention more than one true valued for checkboxes then your program takes the last true and show the last check box as checked.

Components – Text Area and Text Field

- **Text Area:** This is the text container component of Java AWT package. The Text Area contains plain text. TextArea can be declared as follows:

```
TextArea txtArea_name = new TextArea();
```

You can make the Text Area editable or not using the `setEditable (boolean)` method. If you pass the boolean valued argument *false* then the text area will be non-editable otherwise it will be editable. The text area is by default in editable mode. Text are set in the text area using the `setText(string)` method of the **TextArea** class.

- **Text Field:** This is also the text container component of Java AWT package. This component contains single line and limited text information. This is declared as follows :

```
TextField txtfield = new TextField(20);
```

You can fix the number of columns in the text field by specifying the number in the constructor. In the above code we have fixed the number of columns to 20.

Component – Button in an applet

- The **class Component** is extended by all the **AWT components**. More code can be put into this class to design lots of **AWT components**. Most of the **AWT components** shown directly extend Component like **Button, Canvas, Label** etc.
- **Buttons**
 - As shown in the example below, a button is represented by a **single label**. That is the label shown in the example can be pushed with a click of a mouse.

Example

```
import java.awt.*;
import java.applet.Applet;

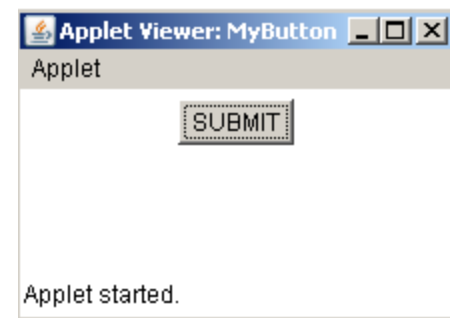
public class MyButton extends Applet {
    public void init() {
        Button button = new Button("SUBMIT");
        add(button);
    }
}
```

- **Here is the HTML code:**

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<APPLET ALIGN="CENTER" CODE="MyButton" WIDTH="400" HEIGHT="200"></APPLET>
</BODY>
</HTML>
```

- **Here is the Output:**

```
C:\newprgrm>javac MyButton.java
C:\newprgrm>appletviewer MyButton.html
```



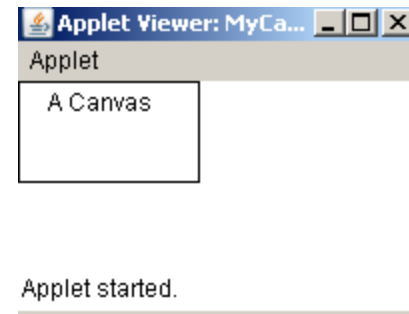
Component – Canvas in an applet

- As the name indicates a **Canvas** is a region where you can draw things such as circles, triangles, ovals or any other shape. Basically it is a graphical component that represents a region. It has got a default method which is **paint()** method. Canvas class can be sub classed to override this default method to define your own components as shown below in the example.

```
import java.awt.*;
import java.applet.*;

public class MyCanvas extends Applet {
    public MyCanvas() {
        setSize(80, 40);
    }
    public void paint(Graphics g) {
        g.drawRect(0, 0, 90, 50);
        g.drawString("A Canvas", 15,15);
    }
}
```

- Here is the Output:
- C:\newprgrm>javac MyCanvas.java
C:\newprgrm>appletviewer MyCanvas.html



Component – Textfield in an applet

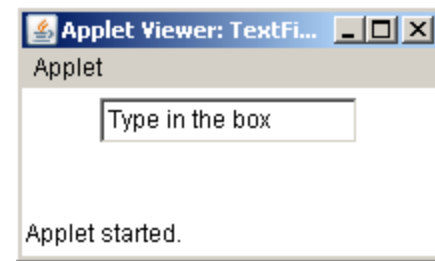
- A scrollable text display object with one row of characters is known as the `TextField`. In the example given below, a `TextField` is placed in the applet by creating its `TextField` object and invoking its `add` method.

```
import java.awt.*;
import java.applet.Applet;

public class TextFieldDemo extends Applet{
    public void init(){
        TextField tf = new TextField("Type in the box");
        add(tf);
    }
}
```

Here is the output:

```
C:\newprgrm>javac TextFieldDemo.java
C:\newprgrm>appletviewer TextFieldDemo.html
```



Component – Checkbox and Scrollbar in an applet

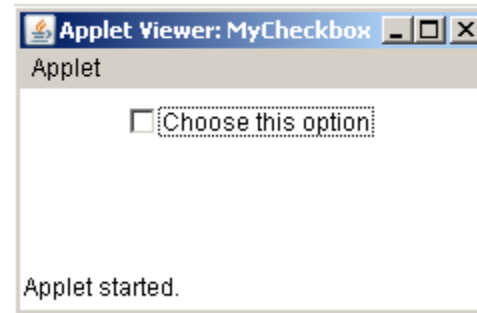
- A **Checkbox** is again a label which is displayed as a pushbutton as shown in the example below. This pushbutton can either be checked or unchecked. Therefore, the state of the checkbox is either true or false. However, the initial state is false which is the default one. The example below shows the checkbox whose state gets toggled

```
import java.awt.*;
import java.applet.Applet;

public class MyCheckbox extends Applet {
    public void init() {
        Checkbox c = new Checkbox("Choose this option");
        add(c);
    }
}
```

Here is the Output:

- C:\newprgrm>javac MyCheckbox.java
C:\newprgrm>appletviewer MyCheckbox.html

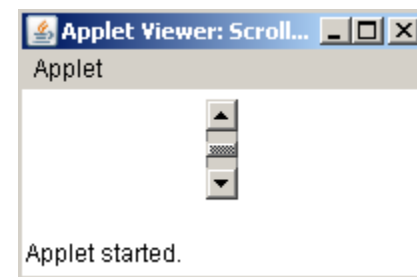


- A **Scrollbar** is represented by a "slider" widget. The characteristics of it are specified by integer values which are being set at the time of scrollbar construction. Both the types of Sliders are available i.e. **horizontal and vertical**.

The example below shows the code for the scrollbar construction. The subtraction of scrollbar width from the maximum setting gives the maximum value of the Scrollbar. In the program code, '0' is the initial value of the scrollbar, '8' is the width of the scrollbar.

- ```
import java.awt.*;
import java.applet.Applet;

public class ScrollbarDemo extends Applet {
 public void init() {
 Scrollbar sb = new Scrollbar
 (Scrollbar.VERTICAL, 0, 8, -100, 100);
 add(sb);
 }
}
```



# The common methods of AWT components

1. **getLocation()** - This method is used to get position of the component, as a Point. The usage of the method is shown below.

```
Point p = someComponent.getLocation();
int x = p.x;
int y = p.y;
```

One point to note here is you do not need to create a new point object. The x and y parts of the location can be easily accessed by using **getX()** and **getY()**. It is always efficient to use **getX()** and **getY()** methods. For example,

```
int x = someComponent.getX();
int y = someComponent.getY();
```

2. **getLocationOnScreen()** - This method is used to get the position of the upper-left corner of the screen of the component, as a Point. The usage of the method is shown below.

```
Point p = someComponent.getLocationOnScreen();
int x = p.x;
int y = p.y;
```

It is always advisable to use **getLocation()** method

# The common methods of AWT components

3. `getBounds()` - This method is used to get the current bounding Rectangle of component. The usage of the method is shown below.

```
Rectangle r = someComponent.getBounds();
int height = r.height;
int width = r.width;
int x = r.x;
int y = r.y;
```

One point to note here is that if you need a Rectangle object then the efficient way is to use `getX()`, `getY()`, `getWidth()`, and `getHeight()` methods while working on Java 2 platform.

4. `getSize()` - This method is used to get the current size of component, as a Dimension. The usage of the method is shown below.

```
Dimension d = someComponent.getSize();
int height = d.height;
int width = d.width;
```

Again, it is always advisable to use `getWidth()` and `getHeight()` methods to directly access the width and height. You can also use `getSize()` if you require a Dimension object. For Example,

```
int height = someComponent.getHeight();
int width = someComponent.getWidth();
```

# The common methods of AWT components

5. **setBackground(Color)/setForeground(Color)** - This method is used to change the background/foreground colors of the component.
6. **setFont(Font)** - This method is used to change the font of text within a component.
7. **setVisible(boolean)** - This method is used for the visibility state of the component. The component appears on the screen if **setVisible()** is set to **true** and if its set to **false** then the component will not appear on the screen. Furthermore, if we mark the component as not visible then the component will disappear while reserving its space in the GUI.
8. **setEnabled(boolean)** - This method is used to toggle the state of the component. The component will appear if set to **true** and it will also react to the user. ON the contrary, if set to **false** then the component will not appear hence no user interaction will be there.

# The Container

As discussed earlier a [container](#) is a component that can be nested. The most widely used [Panel](#) is the Class [Panel](#) which can be extended further to partition GUIs. There is a [Panel](#) which is used for running the programs. This [Panel](#) is known as Class [Applet](#) which is used for running the programs within the Browser.

## Common Container Methods

All the subclasses of the [Container](#) class inherit the behavior of more than 50 common methods of [Container](#). These subclasses of the container mostly override the method of component. Some of the methods of container which are most widely used are as follow:

```
getComponents();
add();
getComponentCount();
getComponent(int);
```

# The Container

## ScrollPane

The `ScrollPane` container provides an automatic scrolling of any larger component which was introduced with the 1.1 release of the Java Runtime Environment (JRE). Any image which is bigger in size for the display area or a bunch of spreadsheet cells is considered as a large object. Moreover there is no `LayoutManager` for a `ScrollPane` because only a single object exists within it. However, the mechanism of `Event Handling` is being managed for scrolling.

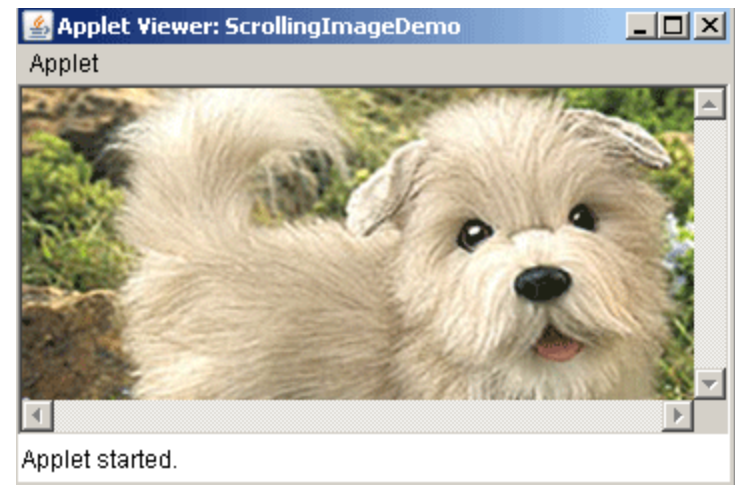
The example below shows the `ScrollPane`. This scrollpane demonstrates the scrolling of the large image. In the program code below, first of all we have created a scrollpane by creating its object, then we have passed the parameter of image in it. We have also set the border layout as centre, as shown.

```
import java.awt.*;
import java.applet.*;
class Scrollpane extends Component {
 private Image image;
 public Scrollpane(Image m) {
 image = m;
 }
 public void paint(Graphics g) {
 if (image != null)
 g.drawImage(image, 0, 0, this);
 }
}

public class ScrollingImageDemo extends Applet {
 public void init() {
 setLayout(new BorderLayout());
 ScrollPane SC = new ScrollPane(ScrollPane.SCROLLBARS_ALWAYS);
 Image mg = getImage(getCodeBase(), "cute-puppy.gif");
 SC.add(new Scrollpane(mg));
 add(SC, BorderLayout.CENTER);
 }
}
```

## Here is the output:

```
C:\newprgrm>javac ScrollingImageDemo.java
C:\newprgrm>appletviewer ScrollingImageDemo.html
```



# Events in the AWT

**Introduction** There are many types of events that are generated by your AWT [Application](#). These events are used to make the application more effective and efficient. Generally, there are twelve types of event are used in [Java](#) AWT. These are as follows :

1. ActionEvent
2. AdjustmentEvent
3. ComponentEvent
4. ContainerEvent
5. FocusEvent
6. InputEvent
7. ItemEvent
8. KeyEvent
9. MouseEvent
10. PaintEvent
11. TextEvent
12. WindowEvent

# Events in the AWT

These are twelve mentioned events are explained as follows :

1. **ActionEvent**: This is the **ActionEvent** class extends from the **AWTEvent** class. It indicates the component-defined events occurred i.e. the event generated by the component like Button, Checkboxes etc. The generated event is passed to every **EventListener** objects that receives such types of events using the **addActionListener()** method of the object.
2. **AdjustmentEvent**: This is the **AdjustmentEvent** class extends from the **AWTEvent** class. When the Adjustable Value is changed then the event is generated.
3. **ComponentEvent**: **ComponentEvent** class also extends from the **AWTEvent** class. This class creates the low-level event which indicates if the object moved, changed and it's states (visibility of the object). This class only performs the notification about the state of the object. The **ComponentEvent** class performs like root class for other component-level events.
4. **ContainerEvent**: The **ContainerEvent** class extends from the **ComponentEvent** class. This is a low-level event which is generated when container's contents changes because of addition or removal of a components.

# Events in the AWT

These are twelve mentioned events are explained as follows :

5. **FocusEvent:** The **FocusEvent** class also extends from the **ComponentEvent** class. This class indicates about the focus where the focus has gained or lost by the object. The generated event is passed to every objects that is registered to receive such type of events using the **addFocusListener()** method of the object.
6. **InputEvent:** The **InputEvent** class also extends from the **ComponentEvent** class. This event class handles all the component-level input events. This class acts as a root class for all component-level input events.
7. **ItemEvent:** The **ItemEvent** class extends from the **AWTEvent** class. The **ItemEvent** class handles all the indication about the selection of the object i.e. whether selected or not. The generated event is passed to every **ItemListener** objects that is registered to receive such types of event using the **addItemListener()** method of the object.
8. **KeyEvent:** **KeyEvent** class extends from the **InputEvent** class. The **KeyEvent** class handles all the indication related to the key operation in the application if you press any key for any purposes of the object then the generated event gives the information about the pressed key. This type of events check whether the pressed key left key or right key, 'A' or 'a' etc.

# Events in the AWT

These are twelve mentioned events are explained as follows :

9. **MouseEvent**: **MouseEvent** class also extends from the **InputEvent** class. The **MouseEvent** class handle all events generated during the mouse operation for the object. That contains the information whether mouse is clicked or not if clicked then checks the pressed key is left or right.
10. **PaintEvent**: **PaintEvent** class also extends from the **ComponentEvent** class. The **PaintEvent** class only ensures that the **paint()** or **update()** are serialized along with the other events delivered from the event queue.
11. **TextEvent**: **TextEvent** class extends from the **AWTEvent** class. **TextEvent** is generated when the text of the object is changed. The generated events are passed to every **TextListener** object which is registered to receive such type of events using the **addTextListener()** method of the object.
12. **WindowEvent** : **WindowEvent** class extends from the **ComponentEvent** class. If the window or the frame of your application is changed (**Opened**, **closed**, **activated**, **deactivated** or any other events are generated), **WindowEvent** is generated.

# Events in the AWT

For any **event** to occur, the **objects** registers themselves as **listeners**.

- No event takes place if there is no listener
- i.e. nothing happens when an event takes place if there is no listener.
- No matter how many listeners there are, each and every listener is capable of processing an event.
- For example, a **SimpleButtonEvent** applet registers itself as the listener for the button's action events that creates a **Button** instance.

**The first event was ActionEvent what is its corresponding listener?**

**ActionListener** can be implemented by any Class including **Applet**.

- One point to remember here is that all the listeners are always notified.

**What if I don't want the event to be handled by a listener?**

- Moreover, you can also call **AWTEvent.consume()** method whenever you don't want an event to be processed further.
- There is another method which is used by a listener to check for the consumption. The method is **isConsumed()** method. The processing of the events gets stopped with the consumption of the events by the system once a listener is notified. Consumption only works for **InputEvent** and its subclasses.
- Moreover, if you don't want any input from the user through keyboard then you can use **consume()** method for the **KeyEvent**.

# Events in the AWT

The step by step procedure of Event handling is as follow:

1. An event occurs

When anything interesting happens then the subclasses of **AWTEvent** are generated by the **component**. (One of the following events occurs)

**Direct Known Subclasses:**

ActionEvent, AdjustmentEvent, AncestorEvent, ComponentEvent, HierarchyEvent,  
InputMethodEvent, InternalFrameEvent, InvocationEvent, ItemEvent, TextEvent

2. A listener is notified

Any class can act like a Listener class permitted by the Event sources.

For example, **addActionListener()** method is used for any action to be performed, where **Action** is the event type. There is another method by which you can **remove** the listener class which is **removeXXXListener()** method, where XXX is the event type.

3. The action is performed

A listener type has to be implemented for an event handling such as **ActionListener**.

4. Sometimes multiple items need to be implemented

There are some special type of listener types as well for which you need to implement multiple methods like **key Events**.

There are three methods which are required to be implemented for Key events and to register them

i.e.            **one for key release,**  
                 **key typed and**  
                 **one for key press.**

# Types of Events

## **AWTEvent**

Most of the time every event-type has a **Listener interface** as **Events** subclass the **AWTEvent class**. However, **MouseEvent** and **KeyEvent** don't have the Listener interface because only the **paint() method** can be overridden with **MouseEvent** etc.

## **Low-level Events**

A **low-level input or window operation** is represented by the **Low-level events**.

Types of Low-level events are **mouse movement, window opening, a key press** etc.

For example, three events are generated by typing the letter 'A' on the Keyboard one for releasing, one for pressing, and one for typing.

The different type of **low-level events** and **operations that generate each event** are show below in the form of a table.

**FocusEvent** Used for Getting/losing focus.

**MouseEvent** Used for entering, exiting, clicking, dragging, moving, pressing, or releasing.

**ContainerEvent** Used for Adding/removing component.

**KeyEvent** Used for releasing, pressing, or typing (both) a key.

**WindowEvent** Used for opening, deactivating, closing, Iconifying, deiconifying, really closed.

**ComponentEvent** Used for moving, resizing, hiding, showing.

# Types of Events

## Semantic Events

The interaction with GUI component is represented by the **Semantic events** like changing the text of a text field, selecting a button etc. The different events generated by different components is shown below.

|                        |                          |
|------------------------|--------------------------|
| <b>ItemEvent</b>       | Used for state changed.  |
| <b>ActionEvent</b>     | Used for do the command. |
| <b>TextEvent</b>       | Used for text changed.   |
| <b>AdjustmentEvent</b> | Used for value adjusted. |

## Event Sources

If a component is an event source for something then the same happens with its subclasses. The different event sources are represented by the following table.

| Low-Level Events |                     | Semantic Events     |                    |
|------------------|---------------------|---------------------|--------------------|
| Window           | WindowListener      | Scrollbar           | AdjustmentListener |
| Container        | ContainerListener   | TextArea            |                    |
| Component        | ComponentListener   | or TextField        | TextListener       |
|                  | FocusListener       | Button              |                    |
|                  | KeyListener         | or List             |                    |
|                  | MouseListener       | or MenuItem         |                    |
|                  | MouseMotionListener | or TextField        | ActionListener     |
|                  |                     | Choice              |                    |
|                  |                     | or Checkbox         |                    |
|                  |                     | or CheckboxMenuItem |                    |
|                  |                     | or List             | ItemListener       |

# Event Listeners

Every **listener interface** has at least one event type. Moreover, it also contains a method for each type of event the event class incorporates. For example as discussed earlier, the **KeyListener** has three methods, one for each type of event that the **KeyEvent** has: **keyTyped()**, **keyPressed()**, and **keyReleased()**.

- The Listener interfaces and their methods are as follow:

| <b>Interface</b>    | <b>Methods</b>                                                                                                                    |                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| WindowListener      | windowActivated(WindowEvent e)<br>windowOpened(WindowEvent e)<br>windowClosing(WindowEvent e)<br>windowDeactivated(WindowEvent e) | windowDeiconified(WindowEvent e)<br>windowClosed(WindowEvent e)<br>windowIconified(WindowEvent e) |
| ActionListener      | actionPerformed(ActionEvent e)                                                                                                    |                                                                                                   |
| AdjustmentListener  | adjustmentValueChanged(AdjustmentEvent e)                                                                                         |                                                                                                   |
| MouseListener       | mouseClicked(MouseEvent e)<br>mouseExited(MouseEvent e)<br>mouseReleased(MouseEvent e)                                            | mouseEntered(MouseEvent e)<br>mousePressed(MouseEvent e)                                          |
| FocusListener       | focusGained(FocusEvent e)<br>focusLost(FocusEvent e)                                                                              |                                                                                                   |
| ItemListener        | itemStateChanged(ItemEvent e)                                                                                                     |                                                                                                   |
| KeyListener         | keyReleased(KeyEvent e)<br>keyPressed(KeyEvent e)                                                                                 | keyTyped(KeyEvent e)                                                                              |
| ComponentListener   | componentHidden(ComponentEvent e)<br>componentShown(ComponentEvent e)                                                             | componentMoved(ComponentEvent e)<br>componentResized(ComponentEvent e)                            |
| MouseMotionListener | mouseMoved(MouseEvent e)                                                                                                          | mouseDragged(MouseEvent e)                                                                        |
| TextListener        | textValueChanged(TextEvent e)                                                                                                     |                                                                                                   |
| ContainerListener   | componentAdded(ContainerEvent e)                                                                                                  | componentRemoved(ContainerEvent e)                                                                |

# Event Adaptors

There are some event listeners that have multiple methods to implement. That is some of the listener interfaces contain more than one method.

For instance, the **MouseListener interface** contains five methods such as **mouseClicked**, **mousePressed**, **mouseReleased** etc. If you want to use only one method out of these then also you will have to implement all of them. Thus, the methods which you do not want to care about can have empty bodies. To avoid such thing, we have **adapter class**.

**Adapter classes** help us in avoiding the implementation of the empty method bodies. Generally an adapter class is there for each listener interface having more than one method. For instance, the **MouseAdapter class** implements the **MouseListener interface**. An **adapter class** can be used by **creating a subclass of it** and then **overriding the methods** which are of use only. Hence avoiding the implementation of all the methods of the listener interface. The following example shows the implementation of a listener interface directly.

```
public class MyClass implements MouseListener {
 ...
 someObject.addMouseListener(this);
 ...
 /* Empty method definition. */
 public void mouseEntered(MouseEvent e) {
 }

 /* Empty method definition. */
 public void mouseExited(MouseEvent e) {
 }

 /* Empty method definition. */
 public void mousePressed(MouseEvent e) {
 }
}
```

# Button Pressing Example

In the program code given below, the **Frame** contains three buttons named - "**Bonjour**", "**Good Day**", "**Aurevoir**". The purpose of these three buttons need to be preserved within a command associated with the button. That is a different action takes place on the click of each button.

You need to run this program outside the browser because this is an [application](#). The following program demonstrates the event generated by each button.

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

```
public class ButtonPressDemo {
 public static void main(String[] args){
 Button b;
 ActionListener a = new MyActionListener();
 Frame f = new Frame("Java Applet");
 f.add(b = new Button("Bonjour"), BorderLayout.NORTH);
 b.setActionCommand("Good Morning");
 b.addActionListener(a);
 f.add(b = new Button("Good Day"), BorderLayout.CENTER);
 b.addActionListener(a);
 f.add(b = new Button("Aurevoir"), BorderLayout.SOUTH);
 b.setActionCommand("Exit");
 b.addActionListener(a);

 f.pack();
 f.show();
 }
}
```

```
class MyActionListener implements ActionListener {
 public void actionPerformed(ActionEvent ae) {
 String s = ae.getActionCommand();
 if (s.equals("Exit")) {
 System.exit(0);
 }
 else if (s.equals("Bonjour")) {
 System.out.println("Good Morning");
 }
 else {
 System.out.println(s + " clicked");
 }
 }
}
```

## Output of the program:

```
C:\newprgrm>javac ButtonPressDemo.java
C:\newprgrm>java ButtonPressDemo
Good Morning clicked
Good Day clicked
```



# Button Pressing Example explained

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

public class ButtonPressDemo {

 public static void main(String[] args) {

 Button b;

 ActionListener a = new MyActionListener();

 Frame f = new Frame("Java Applet");

 f.add(b = new Button("Bonjour"), BorderLayout.NORTH);
 b.setActionCommand("Good Morning");
 b.addActionListener(a);

 f.add(b = new Button("Good Day"), BorderLayout.CENTER);
 b.addActionListener(a);

 f.add(b = new Button("Aurevoir"), BorderLayout.SOUTH);
 b.setActionCommand("Exit");
 b.addActionListener(a);

 f.pack();
 f.show();
 }
}
```

// AN APPLICATION NOT AN APPLETT  
// GOING TO USE A BUTTON(S)  
// WILL IMPLEMENT A LISTENER IN A  
// SEPARATE CLASS, INSTANTIATED HERE  
// A FRAME TO CONTAIN OUR GUI  
// COMPONENTS  
// ADD A BUTTON AT THE TOP  
// SET ITS ACTION  
// ASSIGN OUR LISTENER TO THE BUTTON  
// ADD ANOTHER BUTTON IN THE CENTER  
// ASSIGN OUR LISTENER TO THE BUTTON  
// ADD ANOTHER BUTTON AT THE BOTTOM  
// SET ITS ACTION  
// ASSIGN OUR LISTENER TO THE BUTTON  
// MAKE THE WINDOW FIT THE COMPONENTS  
// MAKE THE CHANGES VISIBLE

# Button Pressing Example explained

```
class MyActionListener implements ActionListener { // IMPLEMENT THE NEEDED LISTENER

 public void actionPerformed(ActionEvent ae) { // IMPLEMENT THE NECESSARY METHOD

 String s = ae.getActionCommand(); // USE THE ACTION EVENT PASSED TO THE METHOD

 if (s.equals("Exit")) { // USE THE COMMAND FROM THE BUTTON
 System.exit(0); // TO DECIDE WHAT TO DO
 }
 else if (s.equals("Bonjour")) {
 System.out.println("Good Morning");
 }
 else {
 System.out.println(s + " clicked");
 }
 }
}
```

# Adaptor Example

In the following program code, the **adaptor class** has been used. This class has been used as an **anonymous inner class** to draw a **rectangle within an applet**.

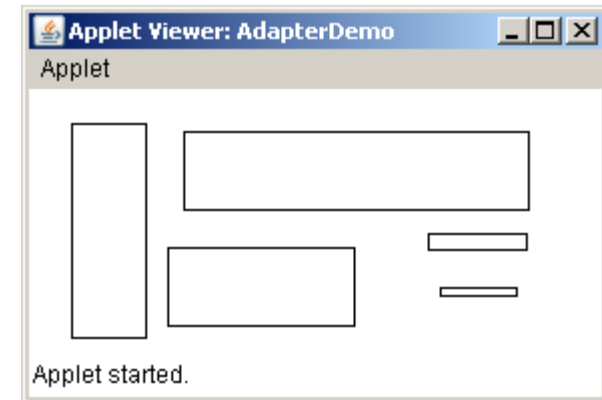
This example demonstrates the functionality of the mouse press. That is on every click of the mouse from **top left corner**, we get a rectangle on the release of the bottom right. The following program demonstrates the functionality of **adaptor class**.

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

public class AdapterDemo extends Applet{
 public void init(){
 addMouseListener(
 new MouseAdapter(){
 int topX, bottomY;
 public void mousePressed(MouseEvent me){
 topX = me.getX();
 bottomY = me.getY();
 }
 public void mouseReleased(MouseEvent me){
 Graphics g = AdapterDemo.this.getGraphics();
 g.drawRect(topX, bottomY, me.getX()-topX, me.getY()-bottomY);
 }
 });
 }
}
```

## Output of the program:

```
C:\newprgrm>javac AdapterDemo.java
C:\newprgrm>appletviewer AdapterDemo.html
C:\newprgrm>
```



# Adaptor Example explained

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

public class AdapterDemo extends Applet{ // extends Applet

 public void init(){ // for Applets implement the init() method

 addMouseListener(new MouseAdapter(){ // MouseAdaptor is a class that implements
 // MouseListener with empty methods for you to override

 int topX, bottomY;

 public void mousePressed(MouseEvent me){ // on mouse pressed get the location of the cursor
 topX = me.getX();
 bottomY = me.getY();
 }

 public void mouseReleased(MouseEvent me){ // on mouse release draw a rectangle draw a rectangle
 Graphics g = AdapterDemo.this.getGraphics(); // using the original and final locations
 g.drawRect(topX, bottomY, me.getX()-topX, me.getY()-bottomY);
 }
 }
 }
}
```

# GUI Programming

In this section you will learn about how to create a **window** for your **application**.

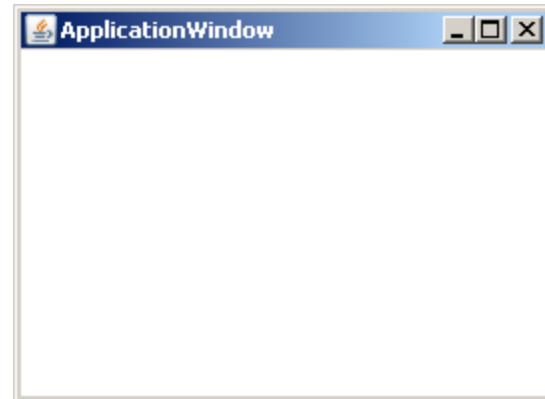
- You need to define a subclass of Frame to create a window for your application. The Frame consists of a **Window with a title, menubar, and border**.
- Moreover, you need to construct an instance of that class through the main method as shown below.
- One point to remember here is that **Applications** respond to events in the same way as applets do. In the example given below, you will see that the Application responds to the native window toolkit i.e. quit or close.

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

public class ApplicationWindow extends Frame{
 public ApplicationWindow(){
 super("ApplicationWindow");
 setSize(200, 200);

 addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent we){
 setVisible(false); dispose();
 System.exit(0);
 }
 });
 }

 public static void main(String[] args){
 ApplicationWindow aw = new ApplicationWindow();
 aw.setVisible(true);
 }
}
```



**Output of the program:**

```
C:\newprgrm>javac ApplicationWindow.java
C:\newprgrm>java ApplicationWindow
```

# GUI Programming

More to come....