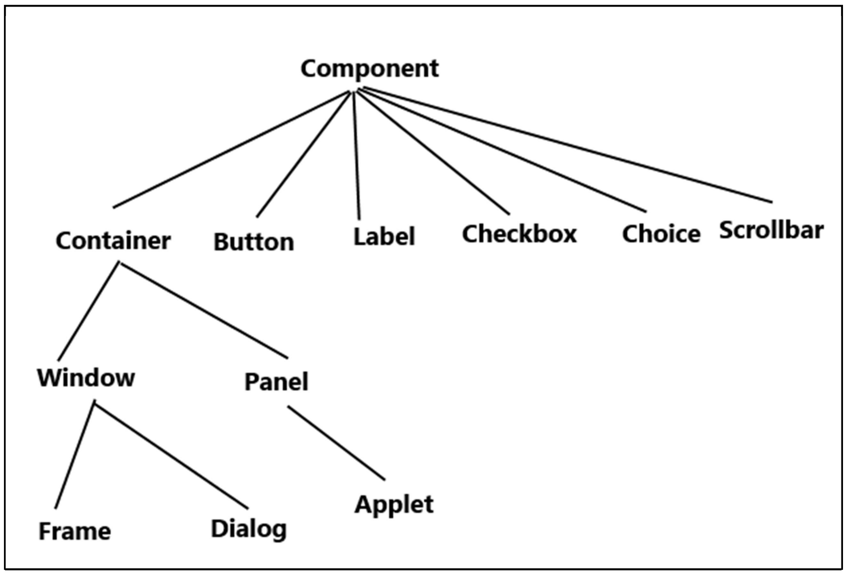
Java AWT

Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications 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 are using the resources of OS. The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy



Component class: Component class is at the top of AWT hierarchy. Component is an abstract class that encapsulates all the attributes of visual component. A component object is responsible for remembering the current foreground and background colors and the currently selected text font.

Container: Container is a component in AWT that contains another component like button, text field, tables etc. Container is a subclass of component class. Container class keeps track of components that are added to another component.

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

Panel: Panel class is a concrete subclass of Container. Panel does not contain title bar, menu bar or border. It is container that is used for holding components. It can have other components like button, textfield etc.

Frame: Frame is a subclass of Window and has resizing canvas. It is a container that contains several different components like button, title bar, textfield, label etc. In Java, most of the AWT applications are created using Frame window. Frame class has two different constructors,

Syntax:

```
Frame() throws HeadlessException

Frame(String title) throws HeadlessException
```

Creating a Frame

There are two ways to create a Frame. They are,

1. By Instantiating Frame class
2. By extending Frame class

1. Creating Frame Window by Instantiating Frame class

```
import java.awt.*;

public class Testawt

{

 Testawt()

 {

  Frame fm=new Frame();        //Creating a frame.

  Label lb = new Label("welcome to java graphics");     //Creating a label

  fm.add(lb);                        //adding label to the frame.

  fm.setSize(300, 300);              //setting frame size.

  fm.setVisible(true);              //set frame visibilty true.

 }

 public static void main(String args[]){
  Testawt ta = new Testawt();
 }
}
```

2. Creating Frame window by extending Frame class

```
package testawt;
import java.awt.*;
import java.awt.event.*;
public class Testawt extends Frame
{
  public Testawt()
 {
  Button btn=new Button("Hello World");
  add(btn);                    //adding a new Button.
  setSize(400, 500);       //setting size.
  setTitle("StudyTonight");  //setting title.
  setLayout(new FlowLayout());           //set default layout for frame.
  setVisible(true);          //set frame visibilty true.


 }
 public static void main (String[] args) {
  Testawt ta = new Testawt();   //creating a frame.
 }
}
```

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.

AWT UI Elements:

Following is the list of commonly used controls while designed GUI using AWT.

Sr. No.	Control	Description
1.	Label	A Label object is a component for placing text in a container.
2.	Button	This class creates a labeled button.
3.	Check Box	A check box is a graphical component that can be in either an on (true) or off (false) state.
4.	Check Box Group	The CheckboxGroup class is used to group the set of checkbox.
5.	List	The List component presents the user with a scrolling list of text items.
6.	Text Field	A TextField object is a text component that allows for the editing of a single line of text.
7.	Text Area	A TextArea object is a text component that allows for the editing of a multiple lines of text.
8.	Choice	A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
9.	Canvas	A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
10.	Image	An Image control is superclass for all image classes representing graphical images.
11.	Scroll Bar	A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
12.	Dialog	A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
13.	File Dialog	A FileDialog control represents a dialog window from which the user can select a file.

AWT Event Handling

Event: Changing the state of an object is known as an **event.**

Types of Event

Foreground Events - Those events which require the direct interaction of user.

Example: clicking on a button, moving the mouse, scrolling the page etc.

Background Events - Those events that require the interaction of end user are known as background events. **Example:** Operating system interrupts, hardware or software failure, etc.

Event Handling

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs.

Java Event Handling Code

We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

Layout

Layout means the arrangement of components within the container. In other way we can say that placing the components at a particular position within the container. The task of layouting the controls is done automatically by the Layout Manager.

Layout Manager

The layout manager automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager.

Java provides us with various layout manager to position the controls. The properties like size, shape and arrangement varies from one layout manager to other layout manager. When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

The layout manager is associated with every Container object. Each layout manager is an object of the class that implements the LayoutManager interface.

Following are the interfaces defining functionalities of Layout Managers.

1) LayoutManager: The LayoutManager interface declares those methods which need to be implemented by the class whose object will act as a layout manager.

2) LayoutManager2: The LayoutManager2 is the sub-interface of the LayoutManager.This interface is for those classes that know how to layout containers based on layout constraint object.

AWT Layout Manager Classes:

Following is the list of commonly used controls while designed GUI using AWT.

1) **BorderLayout:** The BorderLayout arranges the components to fit in the five regions: east, west, north, south and center.
2) **CardLayout:** The CardLayout object treats each component in the container as a card. Only one card is visible at a time
3) **FlowLayout:** The FlowLayout is the default layout.It layouts the components in a directional flow.
4) **GridLayout:** The GridLayout manages the components in form of a rectangular grid.
5) **GridBagLayout:** This is the most flexible layout manager class.The object of GridBagLayout aligns the component vertically,horizontally or along their baseline without requiring the components of same size.

Delegation of event model

Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The event handling in Java involves four types of classes:

1. Event Sources: Event sources are components, subclasses of java.awt.Component, capable to generate events. The event source can be a button, TextField or a Frame etc.

2. Event classes: Almost every event source generates an event and is named by some Java class. For example, the event generated by button is known as ActionEvent and that of Checkbox is known as ItemEvent. All the events are listed in java.awt.event package. Following list gives a few components and their corresponding listeners.

COMPONENT	**EVENT IT GENERATES**
Button, TextField, List, Menu	ActionEvent
Frame	WindowEvent
Checkbox, Choice, List	ItemEvent
Scrollbar	AdjustmentEvent
Mouse (hardware)	MouseEvent
Keyboard (hardware)	KeyEvent

3. **Event Listeners:** The events generated by the GUI components are handled by a special group of interfaces known as "listeners". Listener is an interface. Every component has its own listener, say, AdjustmentListener handles the events of scrollbar. Some listeners handle the events of multiple components. For example, ActionListener handles the events of Button, TextField, List and Menus. Listeners are from java.awt.event package.

Important Event Classes and Interface

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

4. **Event Adapters:** When a listener includes many abstract methods to override, the coding becomes heavy to the programmer. For example, to close the frame, you override seven abstract methods of WindowListener, in which, infact you are using only one method. To avoid this heavy coding, the designers come with another group of classes known as "adapters". Adapters are abstract classes defined in java.awt.event package. Every listener that has more than one abstract method has got a corresponding adapter class.

Following is the list of commonly used adapters while listening GUI events in AWT.

Sr. No.	**Adapter**	**Description**
1.	FocusAdapter	An abstract adapter class for receiving focus events.
2.	KeyAdapter	An abstract adapter class for receiving key events.
3.	MouseAdapter	An abstract adapter class for receiving mouse events.
4.	MouseMotionAdapter	An abstract adapter class for receiving mouse motion events.
5.	WindowAdapter	An abstract adapter class for receiving window events.

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Implement the Listener interface and overrides its methods
2. Register the component with the Listener

Java event handling by implementing ActionListener

```
import java.awt.*;

import java.awt.event.*;

class AEvent extends Frame implements ActionListener{

TextField tf;

AEvent(){

//create components

tf=new TextField();

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

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

b.setBounds(100,120,80,30);
//register listener

b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility

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();

}

}
```

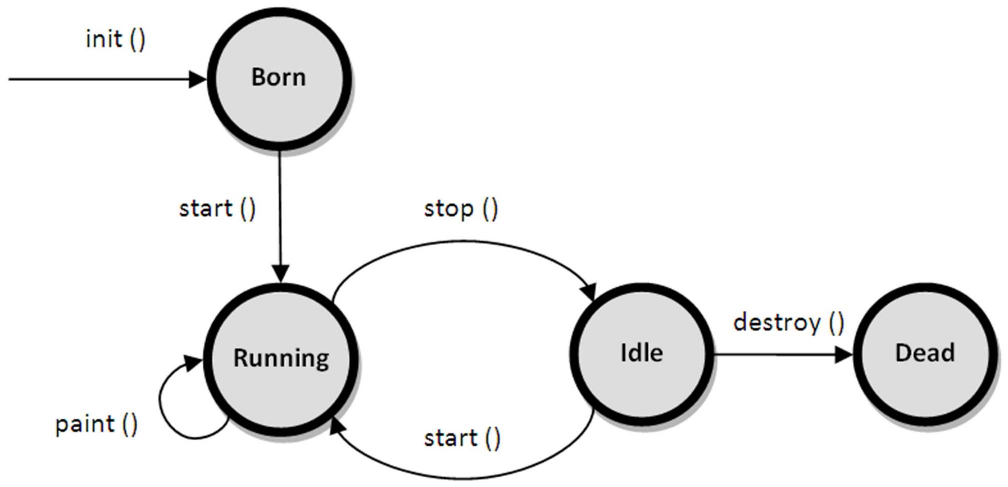
public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.

Applet in Java

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Applets are small Java applications that can be accessed on an Internet server, transported over Internet, and can be automatically installed and run as a part of a web document.

Applet Life Cycle



init(): The init() method is the first method to execute when the applet is executed. Variable declaration and initialization operations are performed in this method.

start(): The start() method contains the actual code of the applet that should run. The start() method executes immediately after the init() method. It also executes whenever the applet is restored, maximized or moving from one tab to another tab in the browser.

stop(): The stop() method stops the execution of the applet. The stop() method executes when the applet is minimized or when moving from one tab to another in the browser.

destroy(): The destroy() method executes when the applet window is closed or when the tab containing the webpage is closed. stop() method executes just before when destroy() method is invoked. The destroy() method removes the applet object from memory.

paint(): The paint() method is used to redraw the output on the applet display area. The paint() method executes after the execution of start() method and whenever the applet or browser is resized.

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.

3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Passing Parameters to Applets

Parameters specify extra information that can be passed to an applet from the HTML page. Parameters are specified using the HTML's *param* tag.

Param Tag

The <param> tag is a sub tag of the <applet> tag. The <param> tag contains two attributes: name and value which are used to specify the name of the parameter and the value of the parameter respectively. For example, the param tags for passing name and age parameters looks as shown below:

```
<param name=”name” value=”Ramesh” />

<param name=”age” value=”25” />
```

Now, these two parameters can be accessed in the applet program using the getParameter() method of the Applet class.

getParameter() Method

The getParameter() method of the Applet class can be used to retrieve the parameters passed from the HTML page. The syntax of getParameter() method is as follows:

```
String getParameter(String param-name)
```

Let’s look at a sample program which demonstrates the <param> HTML tag and the getParameter() method:

```
import java.awt.*;

import java.applet.*;

public class MyApplet extends Applet

{

    String n;

    String a;

    public void init()

    {

        n = getParameter("name");

        a = getParameter("age");

    }

    public void paint(Graphics g)

    {

        g.drawString("Name is: " + n, 20, 20);

        g.drawString("Age is: " + a, 20, 40);

    }

}

/*

    <applet code="MyApplet" height="300" width="500">
```

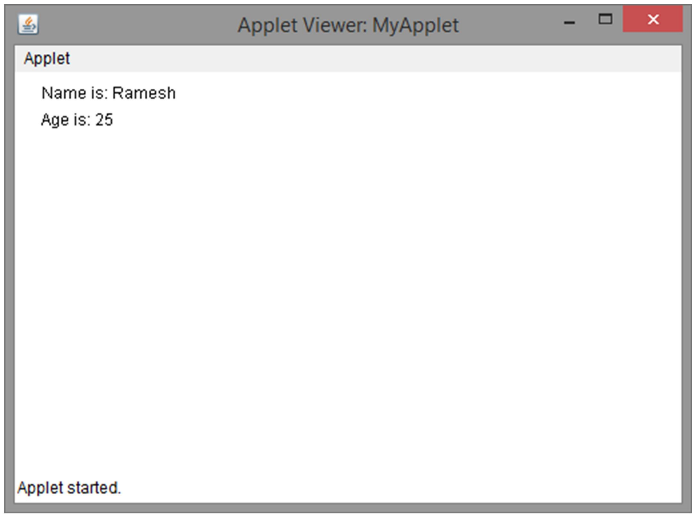
```
                <param name="name" value="Ramesh" />

                <param name="age" value="25" />

            </applet>

        */
```

Output of the above program is as follows:



```
Name is: Ramesh
Age is: 25
```

```
Applet started.
```