Unit-2: Applet and AWT

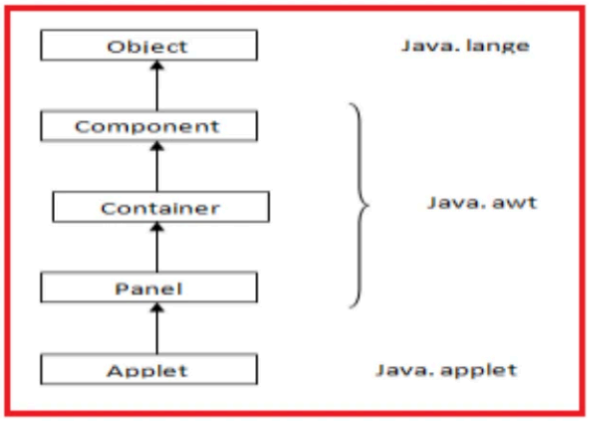
UNIT-II

Java applets, AWT controls (Button, Labels, Combo box, list and other Listeners, menu bar) layout manager, string handling (only main functions)

Java Applets

- An **applet** is a **small Java program** that runs inside a web browser or an **applet viewer**.
- It is mainly used to create **interactive and dynamic web content**.
- Any Java applet is a class that extends the class of `java.applet.Applet`.
- There is no `main()` method in an Applet class.
- The JVM can operate an applet application using either a Web browser plug-in or a distinct runtime environment.
- JVM generates an applet class instance and invokes `init()` to initialize an applet.

Applet Hierarchy in Java



Hierarchy (top to bottom): `Object` → `Component` → `Container` → `Panel` → `Applet`

- `Object` and `Component` belong to `java.lang` / base classes
- `Component`, `Container`, `Panel` belong to `java.awt`
- `Applet` belongs to `java.applet`

Types of Applets in Java

There are two sorts of applets:

1. **Applet** — based directly on the Applet class. These applets use the Abstract Window Toolkit (AWT) to supply the graphic user interface (GUI). This sort of applet has been available since Java was first created.

2. **JApplet** — supported by the Swing class. Swing applets use the Swing classes to supply the GUI. Swing offers a richer and easier-to-use interface than the AWT. Thus, Swing-based applets are now the more popular option.

Note: `Applet` class is used for enabling the AWT package to feature UI components, whereas `JApplet` class is used for enabling the Swing package to feature UI components.

Advantages of Java Applet

1. It works at the client-side, therefore the reaction time is extremely less.
2. It is secured.
3. It is often executed by browsers running under many platforms, including Linux, Windows, Mac OS, etc.

Drawbacks of Applet in Java

- The plugin is required at the client browser to execute the applet.
- An applet cannot load libraries or define native methods.
- An applet cannot read or write files on the execution host or certain system properties.
- An applet cannot make network connections except to the host that it came from.

Running an Applet in Java

- Java Applets are compiled by the `javac` command.
- They can be run in **two ways** (not using the `java` command):
    1. Using the **appletviewer** java tool.
    2. By loading the Applet into a web browser using the `<applet>` tag in HTML.

1) Using Web Browser

To view the applet in a web browser, you require a java enabled browser. To enable java in the browser, go to browser advanced setting and enable java. The following steps should be followed:

- Create an HTML file as above, containing the `APPLET` tag.
- Compile the applet source code using `javac`.
- Open the html file in the web browser.

2) Using appletviewer

It is a java tool provided to view applets. It is like a small browser provided to have a preview of applet as they would look in browser. It understands the `APPLET` tag and is used in the creation phase. The APPLET tag should be written in source code file enclosing in comments. The following steps should be followed:

- Write HTML APPLET tag in comments in the source file.
- Compile the applet source code using `javac`.
- Use `appletviewer ClassName.class` to view the applet.

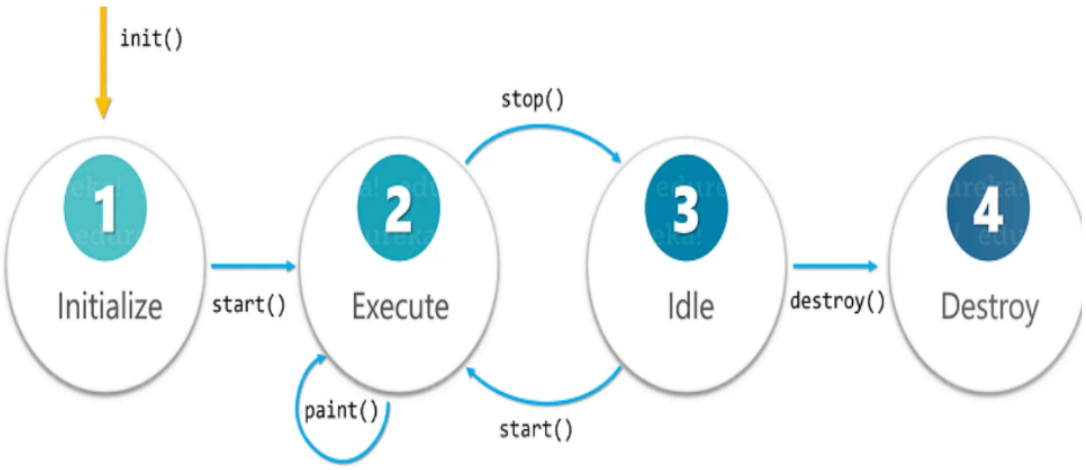
What is Appletviewer?

AppletViewer is a standalone command-line program from Sun Microsystems to run Java applets. Appletviewer is usually employed by developers for testing their applets before deploying them to an internet site. The appletviewer command connects to the documents or resources designated by URLs and displays each applet referenced by the documents in its own window.

Applet Life Cycle

Every Java Applet needs to go through a series of phases from initialization to destruction in order to complete its execution. For that, the very first step is to inherit the `java.applet.Applet` class. This class aids with various methods which help in holding up a basic framework for the Java Applets. The various methods involved in the life cycle of Applet has been depicted by the below diagram.

Life cycle stages (with the diagram flow):



There are 4 main methods which are mandatory for any Java Applet to override:

1. **`public void init()`:** This is the very first method to be invoked during the life cycle of an applet. In this method, the variable that will be used further in the applet is initialized. One thing you must note here is that this method can be invoked only once per applet life cycle.
2. **`public void start()`:** This is the second method that is invoked just after the `init()` method is called by the browser. Each time a user revisits the web page containing the applet, `start()` method is invoked and the applet is started.
3. **`public void stop()`:** This method is invoked whenever a user leaves the web page containing applet. In other words, the `stop()` method is used to suspend the threads which are not required when the applet is in the background or is not visible on the screen. These can be easily resumed using the `start()` method.
4. **`public void destroy()`:** Finally, we have the `destroy()` method which is invoked in order to completely remove an applet from the memory. This method is invoked

only once per applet life cycle and all the engaged resources must be freed up before this method is called.

One more method that is mostly used along with the above four is `paint()`.

- **`public void paint(Graphics g):`** This method is invoked whenever an applet needs to be redrawn or repainted in the browser, irrespective of the cause. The `paint()` method takes one Graphic object as a parameter that contains the graphics context in which the applet is being executed. Also, this method is invoked each time output is expected from the applet.

Example

```
import java.applet.*;
import java.awt.*;
/*
<applet code = "ALC" width = "300" height = "300">
</applet>
*/
public class ALC extends Applet{
    public void init(){
        System.out.println("Applet is Initialized");
    }
    public void start(){
        System.out.println("Applet is being Executed");
    }
    public void stop()
    {
        System.out.println("Applet execution has Stopped");
    }
    public void paint(Graphics g)
    {
        System.out.println("Painting the Applet...");
    }
    public void destroy()
    {
        System.out.println("Applet has been Destroyed");
    }
}
```

Simple Applet Program

```
import java.applet.*;
import java.awt.*;
/*
<applet code = "Simple" width = "300" height = "300">
</applet>
*/
public class Simple extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Raghav", 20, 20);
    }
}
```

Example:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="textdemo" width="500" height="450">
</applet>
*/
public class textdemo extends Applet
{
    Font f1;
    public void init()
    {
        setBackground(Color.pink);
        f1 = new Font("Comic Sans",Font.BOLD,65);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.green);
        g.setFont(f1);
        g.drawString("Alisha, Arifa, Prateeksha",70,85);
    }
}
```

Example:

```
import java.applet.*;
import java.awt.*;
/*
<applet code = "MyApplet" width = "300" height = "300">
</applet>
*/
public class MyApplet extends Applet
{
    int height, width;
    public void init()
```

```
    {
        height = getSize().height;
        width = getSize().width;
        setName("MyApplet");
    }
    public void paint(Graphics g)
    {
        g.drawRoundRect(10, 30, 120, 120, 2, 3);
    }
}
```

Parameter in Applet

User-defined Parameters can be applied in applet using <PARAM…> tags. Each <PARAM…> tag has a name and value attribute.

Example: - name = color - Value = red

Syntax:

```
<PARAM name = ......... Value = "........." >
```

Example:

```
import java.applet.*;
import java.awt.*;
public class param extends Applet
{
    String str;
    public void init()
    {
        str = getParameter("pname");
        if (str == null)
            str = "Hello " + str;
    }
    public void paint(Graphics g)
    {
        g.drawString(str, 200, 200);
    }
}
```

```
<html>
<applet code=param.class height=300 width=300>
<param Name="pname" value="Raghav">
</applet>
</html>
```

Methods of Java Applet Class

The Applet class defines the following methods:

1. **`void destroy()`:** It is called by the browser just before an applet is terminated. If it needs to perform any clean up prior to its destruction, your applet will override this method.
2. **`AccessibleContext getAccessibleContext()`:** It returns the accessibility context for the invoking object.
3. **`AppletContext getAppletContext()`:** It returns the context associated with the applet.
4. **`String getAppletInfo()`:** It returns a String that describes the applet.
5. **`AudioClip getAudioclip(URL url)`:** It returns an AudioClip found at the location specified by URL.
6. **`AudioClip getAudioClip(URL url, String clipName)`:** It returns an AudioClip object found at the location specified by URL and having the name specified by clipName.
7. **`URL getCodeBase()`:** It returns the URL associated with the invoking applet.
8. **`URL getDocumentBase()`:** It returns the URL of the HTML document.
9. **`Image getImage(URL url)`:** It returns an Image object that encapsulates the image of the location specified by the URL.
10. **`Image getImage(URL url, String imageName)`:** Returns an Image object that encapsulates the image found at the location specified by URL and having the name specified by imageName.
11. **`Locale getLocale()`:** It returns a Locale object used by various classes and methods.
12. **`String getParameter(String paramName)`:** It returns the parameter associated with paramName and returned Null if the parameter is not found.
13. **`String[] [] getParameterInfo()`:** It returns a String table that contains the name of the parameter, a description of its type and/or range, and an explanation of its purpose.
14. **`void init()`:** It is the first method called for any applet. It is called when an applet begins execution.
15. **`boolean isActive()`:** It returns true if the applet has been started and returns false if the applet has been stopped.
16. **`static final AudioClip newAudioClip(URL url)`:** It returns an AudioClip found at the location specified by the URL. This method is similar to `getAudioClip()` except that it is static and can be executed without the need for an Applet object.
17. **`void play(URL url)`:** If an audio clip is found at the location specified by the url, the clip is played.
18. **`void play(URL url, String clipName)`:** If an audio clip is found at the location specified by the url with the name specified by clipName.

19. **`void resize(Dimension dim):`** It is used to resize the applet according to the dimensions specified by dim.
20. **`void resize(int width, int height):`** It is used to resize the applet according to the dimensions specified by width and height.
21. **`final void setStub(AppletStub stubObj):`** Makes stubObj the stub for the applet. This method is used by the run-time system and is not usually called by your applet.
22. **`void showStatus(String str):`** It is used to display str in the status window of the browser or applet viewer.
23. **`void start():`** It is called by the browser when an applet should start (or resume) execution. It is automatically called after the `init()` method.
24. **`void stop():`** It is called by the browser to suspend the execution of the applet. Once stopped, an applet is restarted when the browser calls `start()`.

Draw 3D Rectangle

```
import java.applet.*;
import java.awt.*;
/*
<applet code = "Draw3D" width = "300" height = "300">
</applet>
*/
public class Draw3D extends Applet{
    public void paint(Graphics g){
        g.setColor(Color.green);
        //this will draw a 3-D rectangle of width 50 & height 100 at (10,10)
        g.draw3DRect(10,10,50,100,true);
        g.draw3DRect(100,100,50,50,true);
        g.setColor(Color.orange);
        //this will draw a filled 3-D rectangle of width 50 & height 100 at
(10,10)
        g.fill3DRect(10,150,50,100,true);
        //this will draw a filled 3-D square
        g.fill3DRect(100,200,50,50,true);
    }
}
```

Draw Arc

```
import java.applet.*;
import java.awt.*;
/*
<applet code="DrawArc" width=500 height=500>
</applet>
*/
public class DrawArc extends Applet{
    public void paint(Graphics g){
        setForeground(Color.red);
        //this will draw an arc of width 50 & height 100 at (10,10)
        g.drawArc(10,10,50,100,10,45);
        g.fillArc(100,10,100,100,0,90);
```

```
    }
}
```

Draw Random Dots

```
import java.applet.Applet;
import java.awt.Dimension;
import java.awt.Graphics;
/*
<applet code = "DrawDots" width = 500 height = 300>
</applet>
*/
public class DrawDots extends Applet implements Runnable{
    Thread t;
    public void init(){
        //start new Thread
        t = new Thread(this);
        t.start();
    }
    public void run(){
        try{
            while(true){
                // Request repaint
                repaint();
                Thread.sleep(200);
            }
        }
        catch(Exception e){
        }
    }
    public void update(Graphics g){
        paint(g);
    }
    public void paint(Graphics g){
        Dimension d = getSize();
        int x = (int)(Math.random() * d.width);
        int y = (int)(Math.random() * d.height);
        g.fillRect(x,y,2,2);
    }
}
```

Working with Images in Applet

In Applet programs, images also can be used. **`java.awt.Image`** class is used for representing an image.

`java.applet`, **`java.awt`** and **`java.awt.image`** are the packages which are used for event handling.

Loading an Image

In Applet, images are loaded using the **`getImage()`** method. This method works when the constructor of the Applet is called. It is always suggested to call the constructor in the **`init()`** method.

Example:

```
import java.applet.*;
import java.awt.*;
/*
<applet code = "Aimage" width = 500 height = 300>
</applet>
*/
public class Aimage extends Applet
{
    Image img1;
    public void init()
    {
        img1 = getImage(getDocumentBase(),"komal.jpeg");
    }
    public void paint(Graphics g)
    {
        g.drawImage(img1,20,10,this);
    }
}
```

Example:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="linedemo" width="300" height="300">
</applet>
*/
public class linedemo extends Applet
{
    Font f1;
    public void init()
    {
        setBackground(Color.cyan);
        f1 = new Font("Comic Sans",Font.BOLD,55);
    }
```

```
    public void paint(Graphics g)
    {
        g.setFont(f1);
        g.setColor(Color.red);
        g.drawLine(100,200,400,600);
        g.drawRect(125,60,180,120);
        g.drawString("Java",130,120);
        g.draw3DRect(10,10,50,100,true);
        g.drawArc(10,10,50,100,10,45);
        g.drawOval(300,210,450,280);
        g.setColor(Color.gray);
        g.fillOval(300,210,450,280);
        g.setColor(Color.red);
        g.drawString("Alisha",450,400);
    }
}
```

Introduction to Java AWT

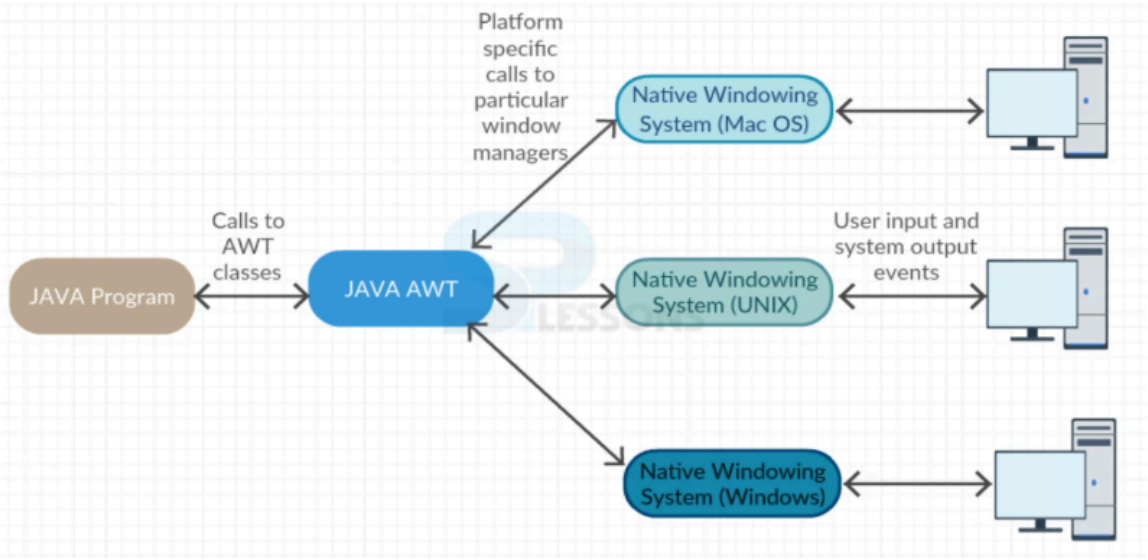
Java AWT (Abstract Window Toolkit) package is a library used for designing graphical user interfaces.

- It contains classes for placing various user intractable components and graphics. However, the components of this class are platform dependent. They are heavy and rely on the OS for their functioning.
- The **AWT package** has classes using which we can create TextBoxes, CheckBoxes, RadioButton, List, etc.

The interface between the user and application program is known as a user-interface. A user-interface has multiple forms, which ranges from commands to graphic clicks.

- In low-level operating systems, the user has to communicate with the commands to interact with the application. Now, a user can interact with the application in just a click by tracking the mouse and reading the keyboard. Abstract Window Toolkit provides a better object-oriented interface to these low-level operating systems by developing its design and the performance.
- AWT has a set of tools that are used in various platforms by including them in the graphical user interface (GUI). Every platform has its GUI Toolkit and the interface element enhances the look & feel of the applications.

Flow diagram:



When a user wants to interact with an application, the user has to provide some information to the application and it can be done in two ways:

- **Character User Interface (CUI):** This interface is used to interact with the application by typing some characters. This interface is not user friendly because the user has to type all the commands and the user has to remember all the commands. Example: DOS
- **Graphical User Interface (GUI):** This interface will interact with the application by using some graphics like menus, icons, images, etc. This interface is user friendly because it prompts the user by providing the options and menus. Example: Windows XP, Windows 7, etc.

Why AWT is platform dependent?

Java AWT components are platform-dependent because components are displayed according to the view of the operating system. Java AWT calls Operating systems subroutine for creating components such as textbox, button, etc. An application built on AWT looks like a windows application when it runs on Windows, but the same application would look like a Mac application when runs on Mac OS.

> **Note:** AWT is rarely used nowadays because of its platform-dependent and heavy-weight nature. AWT components are considered heavyweight because they are being generated by the underlying operating system (OS).

Advantages of GUI over CUI

1. They are easy to learn and use, i.e. users without experience can also use the system quickly.
2. The user can interact with several different applications by switching quickly from one task to another and.
3. Information remains visible in its own window when attention is switched.

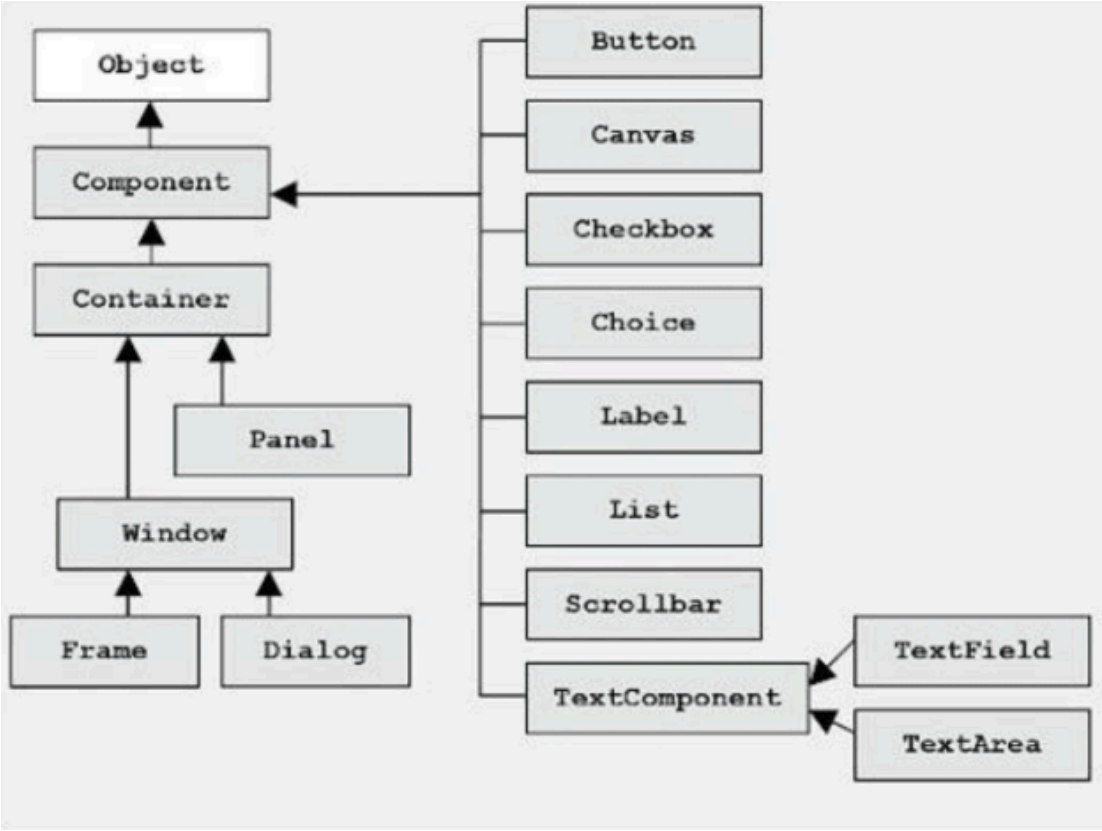
4. Fast, full-screen interaction is possible with immediate access to anywhere on the screen.
5. Some types of error situations can be avoided because they are not allowed to occur through input.

AWT Hierarchy

The AWT classes are contained in **java.awt** package. It is one of Java's largest packages. Fortunately, because it is logically organized in a top-down, hierarchical fashion, it is easier to understand and use than you might at first believe.

The AWT defines windows according to a class hierarchy that is used to add functionality and specificity to each level. The two most common windows are **Panel**, which is used by applets, and **Frame**, which creates a standard application window. Most of the functionalities of these windows are derived from their parent classes.

Hierarchy diagram:



AWT Component

At the top of the AWT hierarchy is the component class. **The component** is an abstract class that encapsulates all of the attributes of a visual component. All user interface elements that interact with the user are subclasses of Component and are displayed on the screen.

Some useful methods of Component Class:

1. **`public void add(Component c):`** It is used to inserts a component on this component.
2. **`public void setSize(int width, int height):`** It is used to set the size (height and width) of the component.
3. **`public void setLayout(LayoutManager m):`** It defines the layout manager of the component.
4. **`public void setStatus(boolean status):`** It is used to change the visibility of the component, by default false.

Container

The Container class is a subclass of Component Class. It has additional methods that allow other component objects to be nested within it. A container provides a space where a component can be located. A Container in AWT is a component itself and it adds the capability to add the component to itself.

1. **Window:** It is an instance of the Window class having neither border nor title. It is used for creating a top-level window.
2. **Frame:** Frame is a subclass of Window and contains title, border and menu bars. It comes with a resizing canvas and is the most widely used container for developing AWT applications. It is capable of holding various components such as buttons, text fields, scrollbars, etc. You can create a Java AWT Frame in two ways:
    i. By Instantiating Frame class
    ii. By extending Frame class
3. **Dialog:** Dialog class is also a subclass of Window and comes with the border as well as the title. Dialog class's instance always needs an associated Frame class instance to exist.
4. **Panel:** Panel is the concrete subclass of Container and doesn't contain any title bar, menu bar or border. Panel class is a generic container for holding the GUI components. You need the instance of the Panel class in order to add the components.

Basic Methods of Component Class

- **`public void add(Component c):`** This method would insert a component on this component.

- **`public void setSize(int width, int height);`** This method would set the size (width and height) of the particular component.
- **`public void setVisible(boolean status);`** This method would change the visibility of the component, which is by default false.
- **`public void setLayout(LayoutManager m);`** This method would define the layout manager for the particular component.

Frame's Constructors

There are two types of Frame's constructors:

1. **`Frame();`** It creates a standard window that does not contain a title.
2. **`Frame(String title);`** It creates a window with the title specified by the title.

   Note: You cannot specify the dimensions of the window. Instead, you must set the size of the window after it has been created.

Methods of Frames

1. **`void setSize(int newWidth, int newHeight);`** It is used to set the dimensions of the window by specifying the width and height fields of the dimensions. The dimensions are specified in terms of pixels.
2. **`void setSize(Dimension newSize);`** The new size of the window is specified by newWidth and newHeight, or by the width and height fields of the Dimension object passed in newSize.
3. **`Dimension getSize();`** The getSize() method is used to obtain the current size of a window. This method returns the current size of the window contained within the width and height fields of a Dimension object.
4. **`void setVisible(Boolean visibleFlag);`** The component is visible if the argument to this method is true. Otherwise, it is hidden.
5. **`void setTitle(String newTitle);`** You can change the title in a frame window using setTitle(). Here, newTitle is the new title for the window.
6. **`windowClosing();`** When using a frame window, your program must remove that window from the screen when it is closed, by calling setVisible(false). To intercept a window-close event, you must implement the windowClosing() method of the WindowListener interface. Inside windowClosing(), you must remove the window from the screen.

Creating a Frame Window in an Applet

The frame can be created in two ways:

1. Create the object of the Frame class directly. `Frame f = new Frame();`
2. Create the object of any class that extends Frame class.
   `MyFrame mf = new MyFrame();`

Sample Program to create a frame by creating the object of frame class directly

```
import java.awt.*;
public class FrameDemo1
{
    public static void main (String[]args)
    {
        Frame f = new Frame ();
        Label lb = new Label ("Hello World");
        f.add (lb);
        f.setVisible (true);
        f.setSize (300, 300);
    }
}
```

Sample Program to create a frame by extending the frame class

```
import java.awt.*;
public class FrameDemo2 extends Frame
{
    public static void main (String[]args)
    {
        FrameDemo2 fd = new FrameDemo2 ();
        Button btn = new Button ("Hello World");
        fd.add (btn);
        fd.setVisible (true);
        fd.setSize (500, 200);
    }
}
```

Label

The easiest control to use is a label. A label contains a string and is an object of type Label. Labels are passive controls that do not support any interaction with the user.

Creating Label: `Label l = new Label(String);`

Label Constructors:

1. **`Label() throws HeadlessException`:** It creates a blank label.
2. **`Label(String str) throws HeadlessException`:** It creates a label that contains the string specified by str.
3. **`Label(String str, int how)`:** It creates a label that contains the string using the alignment specified by how. The value of how must be one of these three constants: **LEFT, Label.RIGHT, Label.CENTER**.

Label Methods:

1. **`void setText(String str)`:** It is used to set or change the text in a label by using the setText() method. Here, str specifies the new label.
2. **`String getText()`:** It is used to obtain the current label by calling getText() method. Here, the current label is returned.

3. **`void setAlignment(int how)`:** It is used to set the alignment of the string within the label by calling setAlignment() method. Here, how is one of the alignment constants?
4. **`int getAlignment()`:** It is used to obtain the current alignment, getAlignment() is called.

Example (Label):

```
import java.awt.*;
import java.applet.*;
/*
<applet code = "LabelDemo" width=300 height=200>
</applet>
*/
public class LabelDemo extends Applet
{
    public void init ()
    {
        Label one = new Label ("One");
        Label two = new Label ("Two");
        Label three = new Label ("Three");
        add (one);
        add (two);
        add (three);
    }
}
```

AWT Buttons Control

The most widely used control is Button. A button is a component that contains a label and that generates an event when it is pressed.

Creating Button: `Button b = new Button(String label);`

Button Constructors:

1. **`Button() throws HeadlessException`:** It creates an empty button.
2. **`Button(String str) throws HeadlessException`:** It creates a button that contains str as a label.

Button Methods:

1. **`void setLabel(String str)`:** You can set its label by calling setLabel(). Here, str is the new Label for the button.
2. **`String getLabel()`:** You can retrieve its label by calling getLabel() method.

Example:

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

```
public class ButtonDemo extends Frame
{
    Button b1, b2;
    ButtonDemo ()
    {
        b1 = new Button ("OK");
        b2 = new Button ("CANCEL");
        this.setLayout (null);
        b1.setBounds (100, 100, 80, 40);;
        b2.setBounds (200, 100, 80, 40);
        this.add (b1);
        this.add (b2);
        this.setVisible (true);
        this.setSize (300, 300);
        this.setTitle ("button");
        this.addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);}
            }
        );
    }
    public static void main (String args[])
    {
        ButtonDemo new ButtonDemo ();
    }
}
```

AWT TextComponent Control in Java

The TextComponent class is the superclass of any component that permits the editing of some text. A text component embodies a string of text. The TextComponent class defines a group of methods that determine whether or not this text is editable. There are two types of TextComponent:

1. TextField
2. TextArea

AWT TextField Control in Java

The TextField component will allow the user to enter some text. It is used to implement a single-line text-entry area, usually called an edit control. It also allows the user to enter strings and edit the text using the arrow keys, cut and paste keys, and mouse selections. TextField is a subclass of TextComponent.

TextField Constructors

1. **`TextField() throws HeadlessException`:** It creates a default textfield.

2. **`TextField(int numChars) throws HeadlessException`:** It creates a text field that is numChars characters wide.
3. **`TextField(String str) throws HeadlessException`:** It initializes the text field with the string contained in str.
4. **`TextField(String str, int numChars) throws HeadlessException`:** It initializes a text field and sets its width.

TextField Methods

1. **`String getText()`:** It is used to obtain the string currently contained in the text field.
2. **`void setText(String str)`:** It is used to set the text. Here, str is the new String.
3. **`void select(int startIndex, int endIndex)`:** It is used to select a portion of the text under program control. It selects the characters beginning at startIndex and ending at endIndex-1.
4. **`String getSelectedText()`:** It returns the currently selected text.
5. **`boolean isEditable()`:** It is used to determine editability. It returns true if the text may be changed and false if not.
6. **`void setEditable(boolean canEdit)`:** It is used to control whether the contents of a text field may be modified by the user. If canEdit is True, the text may be changed. If it is false, the text cannot be altered.
7. **`void setEchoChar(char ch)`:** It is used to disable the echoing of the characters as they are typed. This method specifies a single character that TextField will display when characters are entered.
8. **`boolean echoCharIsSet()`:** By this method, you can check a text field to see if it is in this mode.
9. **`char getEchochar()`:** It is used to retrieve the echo character.

Example (TextField):

```
import java.awt.*;
import java.awt.event.*;
public class TextFieldDemo {
    public static void main(String k[]) {
        Frame f = new Frame("TextField Example");
        TextField tf = new TextField(20);
        Label l = new Label("Enter your name:");
        Button b = new Button("Show");
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("You entered: " + tf.getText());
            }
        });
        f.setLayout(new FlowLayout());
        f.add(l);
        f.add(tf);
        f.add(b);
```

```
        f.setSize(300, 200);
        f.setVisible(true);
    }
}
```

AWT TextArea Control in Java

Sometimes one line of text input isn't enough for a given task. To handle these situations, the AWT includes an easy multiline editor called TextArea.

Creating TextArea: `TextArea ta = new TextArea();`

TextArea Constructor

1. **`TextArea() throws HeadlessException`:** It creates a default textarea.
2. **`TextArea(int numLines, int numChars) throws HeadlessException`:** It creates a text area that is numChars characters wide. Here, numLines specifies the height, in lines of the text area.
3. **`TextArea(String str) throws HeadlessException`:** It initializes the text area with the string contained in str.
4. **`TextArea(String str, int numLines, int numChars) throws HeadlessException`:** It initializes a text field and sets its width. Initial text can be specified by str.
5. **`TextArea(String str, int numLines, int numChars, int sBars) throws HeadlessException`:** Here, you can specify the scroll bars that you want the control to have. sBars must be one of these values:
    1. SCROLLBARS_BOTH
    2. SCROLLBARS_NONE
    3. SCROLLBARS_HORIZONTAL_ONLY
    4. SCROLLBARS_VERTICAL_ONLY

TextArea Methods

TextArea is a subclass of TextComponent. Therefore, it supports the `getText()`, `setText()`, `getSelectedText()`, `select()`, `isEditable()`, and `setEditable()` methods described in the TextField section. TextArea adds the following methods:

1. **`void append(String str)`:** It appends the string specified by str to the end of the current text.
2. **`void insert(String str, int index)`:** It inserts the string passed in str at the specified index.
3. **`void replaceRange(String str, int startIndex, int endIndex)`:** It is used to replace the text. It replaces the characters from startIndex to endIndex-1.

Example (TextArea):

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

```
public class TextAreaDemo {
    public static void main(String k[]) {
        Frame f = new Frame("TextArea Example");
        Label l = new Label("Enter feedback:");
        TextArea ta = new TextArea(5, 30);
        Button b = new Button("Submit");
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String text = ta.getText();
                System.out.println("Your feedback: " + text);
            }
        });
        f.setLayout(new FlowLayout());
        f.add(l);
        f.add(ta);
        f.add(b);
        f.setSize(400, 300);
        f.setVisible(true);
    }
}
```

Event Handling in TextComponent

TextComponent generates **TextEvent**, which is handled by implementing the TextListener interface.

```
import java.awt.*;
import java.awt.event.*;
public class TextListenerDemo {
    public static void main(String k[]) {
        Frame f = new Frame("TextListener Example");
        TextField tf = new TextField(20);
        Label l = new Label("Typing...");
        tf.addTextListener(new TextListener() {
            public void textValueChanged(TextEvent e) {
                l.setText("Text Changed: " + tf.getText());
            }
        });
        f.setLayout(new FlowLayout());
        f.add(tf);
        f.add(l);
        f.setSize(400, 200);
        f.setVisible(true);
    }
}
```

Checkbox Control

A checkbox may be a control that's used to turn an option on or off. It consists of a little box that will either contain a check or not. There's a label related to each checkbox that

describes what option the box represents. You modify the state of a checkbox by clicking on. Checkboxes are often used individually or as a part of a gaggle. Checkboxes are objects of the Checkbox class.

Creating Checkbox: `Checkbox cb = new Checkbox(Label);`

Methods of Checkbox

1. **`boolean getState()`:** To retrieve the present state of a checkbox.
2. **`void setState(boolean on)`:** To line its state, call setState(). Here, if one is true, the box is checked. If it's false, the box is cleared.
3. **`String getLabel()`:** you'll obtain the present label related to a checkbox by calling getLabel().
4. **`void setLabel(String str)`:** To line the label, call setLabel(). The string passed in str becomes the new label related to the invoking checkbox.

Example:

```
import java.awt.*;
import java.awt.event.*;
class chk
{
    public static void main(String k[])
    {
        Frame f=new Frame();
        String s="";
        Label l=new Label("Gender");
        CheckboxGroup cbg=new CheckboxGroup();
        Checkbox m=new Checkbox("Male",cbg,false);
        Checkbox ff=new Checkbox("Female",cbg,false);
        Button b=new Button("Show");

        b.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent a)
            {
                Checkbox selected = cbg.getSelectedCheckbox();
                if (selected != null)
                    System.out.println("Selected Gender: " +
selected.getLabel());
                else
                    System.out.println("No selection made.");
            });
        f.add(l);
        f.add(m);
        f.add(ff);
        f.add(b);
        f.setSize(800,600);
        f.setLayout(new FlowLayout());
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter(){
```

```
            public void windowClosing(WindowEvent we){ System.exit(0); }
        });
    }
}
```

Example-2:

```
import java.awt.*;
import java.awt.event.*;
class checkbox
{
    public static void main(String k[])
    {
        Frame f=new Frame();
        Label quali=new Label("Qualification");
        Checkbox c1=new Checkbox("BCA");
        Checkbox c2=new Checkbox("BBA");
        Checkbox c3=new Checkbox("MCA");
        Checkbox c4=new Checkbox("MBA");
        Button b=new Button("Display");
        Label result=new Label();
        f.add(quali);
        f.add(c1);
        f.add(c2);
        f.add(c3);
        f.add(c4);
        f.add(b);
        f.add(result);
        f.setSize(600,600);
        f.setVisible(true);
        f.setFont(new Font("Arial",Font.BOLD,40));
        f.setLayout(new FlowLayout());
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent w)
            {
                System.exit(0);
            }
        });
        b.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent a)
            {
                String msg = "Selected: ";
                if (c1.getState()) msg += "BCA ";
                if (c2.getState()) msg += "BBA ";
                if (c3.getState()) msg += "MBA ";
                if (c4.getState()) msg += "MCA ";

                if (!c1.getState() && !c2.getState() && !c3.getState() &&
!c4.getState()) {
                    msg = "No qualification selected!";
```

```
                }

                result.setText(msg);
            }
        });
    }
}
```

AWT Choice Control in Java

This component will display a group of times as a drop-down menu from which a user can select only one item. The choice component is used to create a pop-up list of items from which the user may choose. Therefore, Choice control is a form of a menu. When it is inactive, a Choice component takes up only enough space to show the currently selected item. When the user clicks on a Choice component, the whole list of choices pops up, and a new selection can be made.

Note: Choice only defines the default constructor, which creates an empty list.

Creating Choice: `Choice ch = new Choice();`

Choice Methods

1. **`void add(String name):`** To add a selection to the list, use add(). Here, the name is the name of the item being added.
2. **`String getSelectedItem():`** It determines which item is currently selected. It returns a string containing the name of the item.
3. **`int getSelectedIndex():`** It determines which item is currently selected. It returns the index of the item.
4. **`int getItemCount():`** It obtains the number of items in the list.
5. **`void select(int index):`** It is used to set the currently selected item with a zero-based integer index.
6. **`void select(String name):`** It is used to set the currently selected item with a string that will match a name in the list.
7. **`String getItem(int index):`** It is used to obtain the name associated with the item at the given index. Here, the index specifies the index of the desired items.

Example:

```
import java.awt.*;
import java.awt.event.*;
class choicedemo1
{
    public static void main(String k[])
    {
        Frame f=new Frame();
        Label l=new Label("Select your programming");
        Choice c=new Choice();
        Button b=new Button("DISPLAY");
```

```
        c.addItem("...select...");
        c.addItem("C");
        c.addItem("Java");
        c.addItem("Python");
        c.addItem("C++");
        c.addItem("DotNet");
        f.setFont(new Font("Cambria",Font.BOLD,25));
        f.setSize(600,600);
        f.setVisible(true);
        f.setLayout(new FlowLayout());
        f.add(l);
        f.add(c);
        f.add(b);
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent w)
            {
                System.exit(0);
            }
        });
        b.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent a)
            {
                System.out.println(c.getSelectedItem());
            }
        });
    }
}
```

Menu

A **Menu** in AWT is used to create a **drop-down list of options** in a **menu bar**.

It allows users to perform actions in a structured and easy way — similar to menus in applications like Notepad or MS Word.

Menu Hierarchy in AWT

The **AWT Menu System** includes the following main classes:

```
java.lang.Object
   ├── java.awt.MenuComponent
   │       ├── java.awt.MenuBar
   │       ├── java.awt.Menu
   │       ├── java.awt.MenuItem
   │       └── java.awt.CheckboxMenuItem
```

Class	Description
MenuBar	The top-level bar that holds all menus (like “File”, “Edit”, etc.)
Menu	Represents a drop-down list of items (like File → New, Open, Exit)

Class MenuItem	Description Represents a single selectable item inside a menu
CheckboxMenuItem	A menu item that can be checked or unchecked

MenuBar

```
MenuBar mbar = new MenuBar();
```

Menu

```
Menu menu = new Menu("File");
```

MenuItem

```
MenuItem mi1 = new MenuItem("New");
MenuItem mi2 = new MenuItem("Open");
```

CheckboxMenuItem

```
CheckboxMenuItem cb = new CheckboxMenuItem("Show Toolbar");
```

Commonly Used Methods

Method add(MenuItem item)	Description Adds a menu item to a menu
add(Menu menu)	Adds a submenu to a menu
setMenuBar(MenuBar m)	Sets the menu bar to a frame
addSeparator()	Adds a separator line between menu items
setEnabled(boolean b)	Enables/disables a menu item
getLabel()	Returns the label of a menu or menu item
setLabel(String str)	Changes the label of a menu or menu item

Example (Menu):

```
import java.awt.*;
import java.awt.event.*;
class MenuExample {
    public static void main(String k[]) {
        // Create Frame
        Frame f = new Frame("AWT Menu Example");
        // Create MenuBar
        MenuBar mbar = new MenuBar();
        // Create Menus
        Menu file = new Menu("File");
        Menu edit = new Menu("Edit");
        Menu view = new Menu("View");
        // Create MenuItems for File Menu
        MenuItem newFile = new MenuItem("New");
```

```
        MenuItem open = new MenuItem("Open");
        MenuItem save = new MenuItem("Save");
        MenuItem exit = new MenuItem("Exit");
        // Add MenuItems to File Menu
        file.add(newFile);
        file.add(open);
        file.add(save);
        file.addSeparator(); // Adds a line separator
        file.add(exit);
        // Add MenuItems to Edit Menu
        MenuItem cut = new MenuItem("Cut");
        MenuItem copy = new MenuItem("Copy");
        MenuItem paste = new MenuItem("Paste");
        edit.add(cut);
        edit.add(copy);
        edit.add(paste);
        // Add CheckboxMenuItem in View Menu
        CheckboxMenuItem toolbar = new CheckboxMenuItem("Show Toolbar");
        view.add(toolbar);
        // Add Menus to MenuBar
        mbar.add(file);
        mbar.add(edit);
        mbar.add(view);
        // Set MenuBar to Frame
        f.setMenuBar(mbar);
        // Add action for Exit MenuItem
        exit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                f.dispose(); // Close frame
            }
        });
        // Set Frame Layout and Visibility
        f.setSize(400, 300);
        f.setLayout(new FlowLayout());
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                f.dispose();
            }
        });
    }
}
```

Layout Manager

In Java **AWT (Abstract Window Toolkit)**, a Layout Manager is responsible for **arranging GUI components** (like buttons, text fields, labels, etc.) inside a **container** such as Frame, Panel, or Applet.

Without a layout manager, all components must be placed manually using coordinates — which is difficult and non-flexible.

Applet Layout Manager in Java

A **Layout Manager** in an Applet is used to **automatically arrange GUI components** such as **Button, Label, TextField, Checkbox, etc.** inside the applet.

Instead of specifying the exact position `(x, y)` of every component, we can use a layout manager.

Why do we use Layout Manager?

Suppose we add 5 buttons to an Applet. Without a layout manager, we have to manually decide:

button.setBounds(20, 50, 80, 30); //Check the Code of ThreeButton.java in GitHub

With a Layout Manager, Java automatically decides where the components should be placed.

Types of Layout Managers

AWT package provides the following types of Layout Managers:

1. Flow Layout (BY Default) //Check the Code of LayoutDemo.java in GitHub
2. Border Layout //Check the Code of BorderDemo.java in GitHub
3. Card Layout //Check the Code of CardDemo.java in GitHub
4. Grid Layout
5. GridBag Layout

Flow Layout

This layout will display the components from left to right, from top to bottom. The components will always be displayed in the first line and if the first line is filled, these components are displayed on the next line automatically.

> **Note:** If the row contains only one component, then the component is aligned in the center position of that row.

Creation of Flow Layout

```
FlowLayout f1 = new FlowLayout();
FlowLayout f1 = new FlowLayout(int align);
FlowLayout f1 = new FlowLayout(int align, int hgap, int vgap);
```

Example:

```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutDemo
{
    JFrame f;
    FlowLayoutDemo ()
    {
        f = new JFrame ();
        JLabel l1 = new JLabel ("Enter Name");
        JTextField tf1 = new JTextField (10);
        JButton b1 = new JButton ("SUBMIT");
        f.add (l1);
        f.add (tf1);
        f.add (b1);
        f.setLayout (new FlowLayout (FlowLayout.RIGHT));

        //setting flow layout of right alignment
        f.setSize (300, 300);
        f.setVisible (true);
    }
    public static void main (String k[])
    {
        new FlowLayoutDemo ();
    }
}
```