Unit-IV

APPLETS

Java applets- Life cycle of an applet – Adding images to an applet – Adding sound to an applet. Passing parameters to an applet. Event Handling. Introducing AWT: Working with Windows Graphics and Text. Using AWT Controls, Layout Managers and Menus. Servlet – life cycle of a servlet. The Servlet API, Handling HTTP Request and Response, using Cookies, Session Tracking. Introduction to JSP.

Java Applets:

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following −

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet

Four methods in the Applet class gives you the framework on which you build any serious applet −

- **init** − This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start** − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

A "Hello, World" Applet

Following is a simple applet named HelloWorldApplet.java −

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

public class HelloWorldApplet extends Applet {
```

```
  public void paint (Graphics g) {
    g.drawString ("Hello World", 25, 50);
  }
}
```

These import statements bring the classes into the scope of our applet class −

- java.applet.Applet
- java.awt.Graphics

Without those import statements, the Java compiler would not recognize the classes Applet and Graphics, which the applet class refers to.

The Applet Class

Every applet is an extension of the *java.applet.Applet class*. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following −

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may −

- Request information about the author, version, and copyright of the applet
- Request a description of the parameters the applet recognizes
- Initialize the applet
- Destroy the applet
- Start the applet's execution
- Stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

The "Hello, World" applet is complete as it stands. The only method overridden is the paint method.

Invoking an Applet

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.

The <applet> tag is the basis for embedding an applet in an HTML file. Following is an example that invokes the "Hello, World" applet −

```
<html>
  <title>The Hello, World Applet</title>
  <hr>
  <applet code = "HelloWorldApplet.class" width = "320" height = "120">
    If your browser was Java-enabled, a "Hello, World"
    message would appear here.
  </applet>
  <hr>
</html>
```

Note − You can refer to HTML Applet Tag to understand more about calling applet from HTML.

The code attribute of the <applet> tag is required. It specifies the Applet class to run. Width and height are also required to specify the initial size of the panel in which an applet runs. The applet directive must be closed with an </applet> tag.

If an applet takes parameters, values may be passed for the parameters by adding <param> tags between <applet> and </applet>. The browser ignores text and other tags between the applet tags.

Non-Java-enabled browsers do not process <applet> and </applet>. Therefore, anything that appears between the tags, not related to the applet, is visible in non-Java-enabled browsers.

The viewer or browser looks for the compiled Java code at the location of the document. To specify otherwise, use the codebase attribute of the <applet> tag as shown −

```
<applet codebase = "https://amrood.com/applets" code = "HelloWorldApplet.class"
  width = "320" height = "120">
```

If an applet resides in a package other than the default, the holding package must be specified in the code attribute using the period character (.) to separate package/class components. For example −

```
<applet  = "mypackage.subpackage.TestApplet.class"
  width = "320" height = "120">
```

Getting Applet Parameters

The following example demonstrates how to make an applet respond to setup parameters specified in the document. This applet displays a checkerboard pattern of black and a second color.

The second color and the size of each square may be specified as parameters to the applet within the document.

CheckerApplet gets its parameters in the init() method. It may also get its parameters in the paint() method. However, getting the values and saving the settings once at the start of the applet, instead of at every refresh, is convenient and efficient.

The applet viewer or browser calls the init() method of each applet it runs. The viewer calls init() once, immediately after loading the applet. (Applet.init() is implemented to do nothing.) Override the default implementation to insert custom initialization code.

The Applet.getParameter() method fetches a parameter given the parameter's name (the value of a parameter is always a string). If the value is numeric or other non-character data, the string must be parsed.

The following is a skeleton of CheckerApplet.java −

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

public class CheckerApplet extends Applet {
  int squareSize = 50;  // initialized to default size
  public void init() {}
  private void parseSquareSize (String param) {}
  private Color parseColor (String param) {}
  public void paint (Graphics g) {}
}
```

Here are CheckerApplet's init() and private parseSquareSize() methods −

```
public void init () {
  String squareSizeParam = getParameter ("squareSize");
  parseSquareSize (squareSizeParam);
```

```
  String colorParam = getParameter ("color");
  Color fg = parseColor (colorParam);

  setBackground (Color.black);
  setForeground (fg);
}

private void parseSquareSize (String param) {
  if (param == null) return;
  try {
    squareSize = Integer.parseInt (param);
  } catch (Exception e) {
    // Let default value remain
  }
}
```

The applet calls parseSquareSize() to parse the squareSize parameter. parseSquareSize() calls the library method Integer.parseInt(), which parses a string and returns an integer. Integer.parseInt() throws an exception whenever its argument is invalid.

Therefore, parseSquareSize() catches exceptions, rather than allowing the applet to fail on bad input.

The applet calls parseColor() to parse the color parameter into a Color value. parseColor() does a series of string comparisons to match the parameter value to the name of a predefined color. You need to implement these methods to make this applet work.

Specifying Applet Parameters

The following is an example of an HTML file with a CheckerApplet embedded in it. The HTML file specifies both parameters to the applet by means of the <param> tag.

```
<html>
  <title>Checkerboard Applet</title>
  <hr>
  <applet code = "CheckerApplet.class" width = "480" height = "320">
    <param name = "color" value = "blue">
    <param name = "squaresize" value = "30">
  </applet>
  <hr>
</html>
```

Note − Parameter names are not case sensitive.

Application Conversion to Applets

It is easy to convert a graphical Java application (that is, an application that uses the AWT and that you can start with the Java program launcher) into an applet that you can embed in a web page.

Following are the specific steps for converting an application to an applet.

- Make an HTML page with the appropriate tag to load the applet code.
- Supply a subclass of the JApplet class. Make this class public. Otherwise, the applet cannot be loaded.
- Eliminate the main method in the application. Do not construct a frame window for the application. Your application will be displayed inside the browser.

- Move any initialization code from the frame window constructor to the init method of the applet. You don't need to explicitly construct the applet object. The browser instantiates it for you and calls the init method.
- Remove the call to setSize; for applets, sizing is done with the width and height parameters in the HTML file.
- Remove the call to setDefaultCloseOperation. An applet cannot be closed; it terminates when the browser exits.
- If the application calls setTitle, eliminate the call to the method. Applets cannot have title bars. (You can, of course, title the web page itself, using the HTML title tag.)
- Don't call setVisible(true). The applet is displayed automatically.

Event Handling

Applets inherit a group of event-handling methods from the Container class. The Container class defines several methods, such as processKeyEvent and processMouseEvent, for handling particular types of events, and then one catch-all method called processEvent.

In order to react to an event, an applet must override the appropriate event-specific method.

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.applet.Applet;
import java.awt.Graphics;

public class ExampleEventHandling extends Applet implements MouseListener {
  StringBuffer strBuffer;

  public void init() {
    addMouseListener(this);
    strBuffer = new StringBuffer();
    addItem("initializing the apple ");
  }

  public void start() {
    addItem("starting the applet ");
  }

  public void stop() {
    addItem("stopping the applet ");
  }

  public void destroy() {
    addItem("unloading the applet");
  }

  void addItem(String word) {
    System.out.println(word);
    strBuffer.append(word);
    repaint();
  }

  public void paint(Graphics g) {
    // Draw a Rectangle around the applet's display area.
    g.drawRect(0, 0,
```

```
      getWidth() - 1,
      getHeight() - 1);

      // display the string inside the rectangle.
      g.drawString(strBuffer.toString(), 10, 20);
   }


   public void mouseEntered(MouseEvent event) {
   }
   public void mouseExited(MouseEvent event) {
   }
   public void mousePressed(MouseEvent event) {
   }
   public void mouseReleased(MouseEvent event) {
   }
   public void mouseClicked(MouseEvent event) {
      addItem("mouse clicked! ");
   }
}
```

Now, let us call this applet as follows −

```
<html>
   <title>Event Handling</title>
   <hr>
   <applet code = "ExampleEventHandling.class"
      width = "300" height = "300">
   </applet>
   <hr>
</html>
```

Initially, the applet will display "initializing the applet. Starting the applet." Then once you click inside the rectangle, "mouse clicked" will be displayed as well.

Displaying Images

An applet can display images of the format GIF, JPEG, BMP, and others. To display an image within the applet, you use the drawImage() method found in the java.awt.Graphics class.

Following is an example illustrating all the steps to show images −

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

public class ImageDemo extends Applet {
   private Image image;
   private AppletContext context;

   public void init() {
      context = this.getAppletContext();
      String imageURL = this.getParameter("image");
      if(imageURL == null) {
         imageURL = "java.jpg";
```

```
    }
    try {
      URL url = new URL(this.getDocumentBase(), imageURL);
      image = context.getImage(url);
    } catch (MalformedURLException e) {
      e.printStackTrace();
      // Display in browser status bar
      context.showStatus("Could not load image!");
    }
  }

  public void paint(Graphics g) {
    context.showStatus("Displaying image");
    g.drawImage(image, 0, 0, 200, 84, null);
    g.drawString("www.javalicense.com", 35, 100);
  }
}
```

Now, let us call this applet as follows −

```
<html>
  <title>The ImageDemo applet</title>
  <hr>
  <applet code = "ImageDemo.class" width = "300" height = "200">
    <param name = "image" value = "java.jpg">
  </applet>
  <hr>
</html>
```

Playing Audio

An applet can play an audio file represented by the AudioClip interface in the java.applet package. The AudioClip interface has three methods, including −

- **public void play()** − Plays the audio clip one time, from the beginning.
- **public void loop()** − Causes the audio clip to replay continually.
- **public void stop()** − Stops playing the audio clip.

To obtain an AudioClip object, you must invoke the getAudioClip() method of the Applet class. The getAudioClip() method returns immediately, whether or not the URL resolves to an actual audio file. The audio file is not downloaded until an attempt is made to play the audio clip.

Following is an example illustrating all the steps to play an audio −

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

public class AudioDemo extends Applet {
  private AudioClip clip;
  private AppletContext context;

  public void init() {
    context = this.getAppletContext();
    String audioURL = this.getParameter("audio");
    if(audioURL == null) {
```

```
      audioURL = "default.au";
    }
    try {
      URL url = new URL(this.getDocumentBase(), audioURL);
      clip = context.getAudioClip(url);
    } catch (MalformedURLException e) {
      e.printStackTrace();
      context.showStatus("Could not load audio file!");
    }
  }

  public void start() {
    if(clip != null) {
      clip.loop();
    }
  }

  public void stop() {
    if(clip != null) {
      clip.stop();
    }
  }
}
```

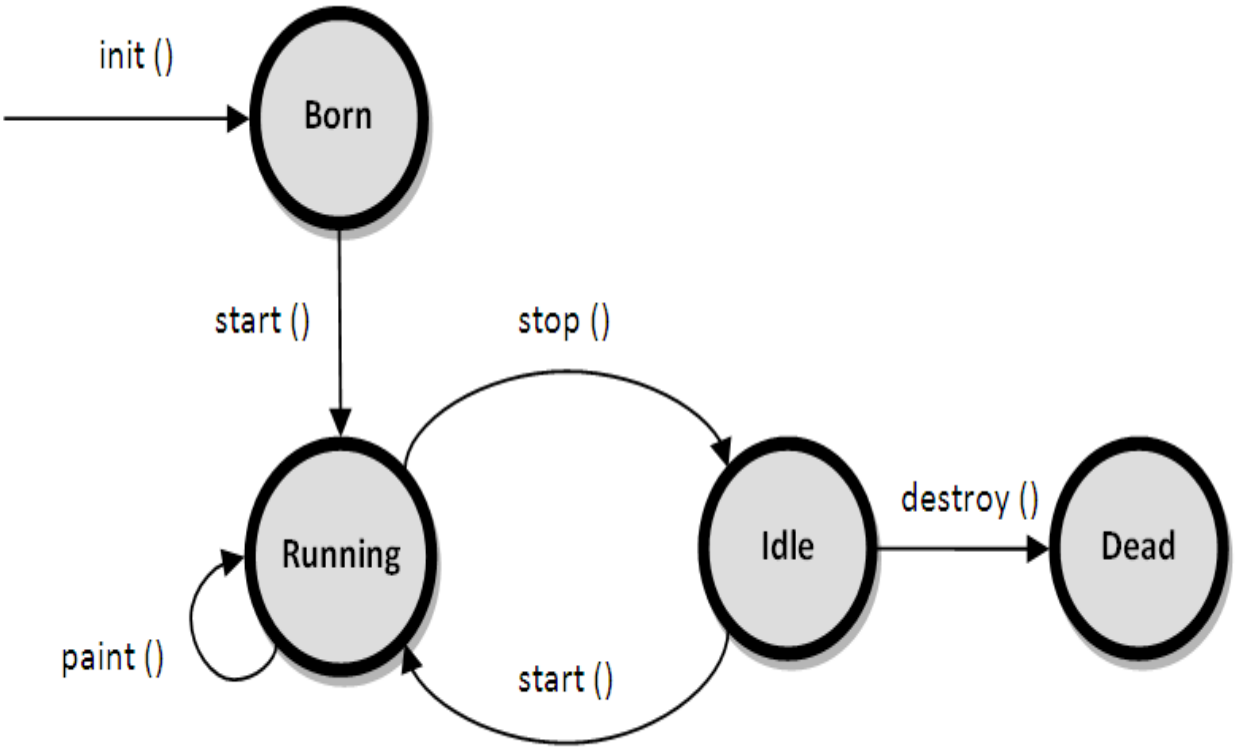
Now, let us call this applet as follows −

```
<html>
  <title>The ImageDemo applet</title>
  <hr>
  <applet code = "ImageDemo.class" width = "0" height = "0">
    <param name = "audio" value = "test.wav">
  </applet>
  <hr>
</html>
```

Life Cycle of an Applet:

In this article we will learn about applet life cycle and various life cycle methods of an applet along with example program.

The life cycle of an applet is as shown in the figure below:



As shown in the above diagram, the life cycle of an applet starts with *init()* method and ends with *destroy()* method. Other life cycle methods are *start()*, *stop()* and *paint()*. The methods to execute only once in the applet life cycle are *init()* and *destroy()*. Other methods execute multiple times.

Below is the description of each applet life cycle method:

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.

The method execution sequence when an applet is executed is:

- init()

- start()
- paint()

The method execution sequence when an applet is closed is:

- stop()
- destroy()
- Example program that demonstrates the life cycle of an applet is as follows:
-

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
   public void init()
   {
            System.out.println("Applet initialized");
   }
   public void start()
   {
            System.out.println("Applet execution started");
   }
   public void stop()
   {
            System.out.println("Applet execution stopped");
   }
   public void paint(Graphics g)
   {
            System.out.println("Painting...");
   }
   public void destroy()
   {
            System.out.println("Applet destroyed");
   }
}
```

- Output of the above applet program when run using appletviewer tool is:

- Applet initialized
  Applet execution started
  Painting…
  Painting…
  Applet execution stopped
  Applet destroyed

Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

```
1. public Image getImage(URL u, String image){}
```

Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet is embedded.
2. **public URL getCodeBase():** is used to return the base URL.

Example of displaying image in applet:

```
1. import java.awt.*;
2. import java.applet.*;
3.
4.
5. public class DisplayImage extends Applet {
6.
7.   Image picture;
8.
9.   public void init() {
10.    picture = getImage(getDocumentBase(),"sonoo.jpg");
11.  }
12.
13.  public void paint(Graphics g) {
14.    g.drawImage(picture, 30,30, this);
15.  }
16.
17.  }
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
1. <html>
2. <body>
3. <applet code="DisplayImage.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

Adding sound to an applet:

Problem Description

How to play sound using Applet?

Solution

Following example demonstrates how to play a sound using an applet image using getAudioClip(), play() & stop() methods of AudioClip() class.

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

public class PlaySoundApplet extends Applet implements ActionListener {
  Button play,stop;
  AudioClip audioClip;

  public void init() {
    play = new Button(" Play in Loop ");
    add(play);
    play.addActionListener(this);
    stop = new Button(" Stop ");
    add(stop);
    stop.addActionListener(this);
    audioClip = getAudioClip(getCodeBase(), "Sound.wav");
  }
  public void actionPerformed(ActionEvent ae) {
    Button source = (Button)ae.getSource();
    if (source.getLabel() == " Play in Loop ") {
      audioClip.play();
    } else if(source.getLabel() == "  Stop  "){
      audioClip.stop();
    }
  }
}
```

Result

The above code sample will produce the following result in a java enabled web browser.

View in Browser.

Passing parameter to an applet:

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter(). Syntax:

1. **public** String getParameter(String parameterName)

Example of using parameter in Applet:

```
1. import java.applet.Applet;
2. import java.awt.Graphics;
3.
4. public class UseParam extends Applet{
5.
6. public void paint(Graphics g){
7. String str=getParameter("msg");
8. g.drawString(str,50, 50);
9. }
10.
11. }
```

myapplet.html

```
1. <html>
```

```
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```

Event handling:

EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple in applet that prints a message by click on the button.

Example of EventHandling in applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener{
Button b;
TextField tf;

public void init(){
tf=new TextField();
tf.setBounds(30,40,150,20);

b=new Button("Click");
b.setBounds(80,150,60,50);

add(b);add(tf);
b.addActionListener(this);

setLayout(null);
}

 public void actionPerformed(ActionEvent e){
  tf.setText("Welcome");
 }
}
```

In the above example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

```
<html>
<body>
<applet code="EventApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

Introducing AWT:

Java AWT Tutorial

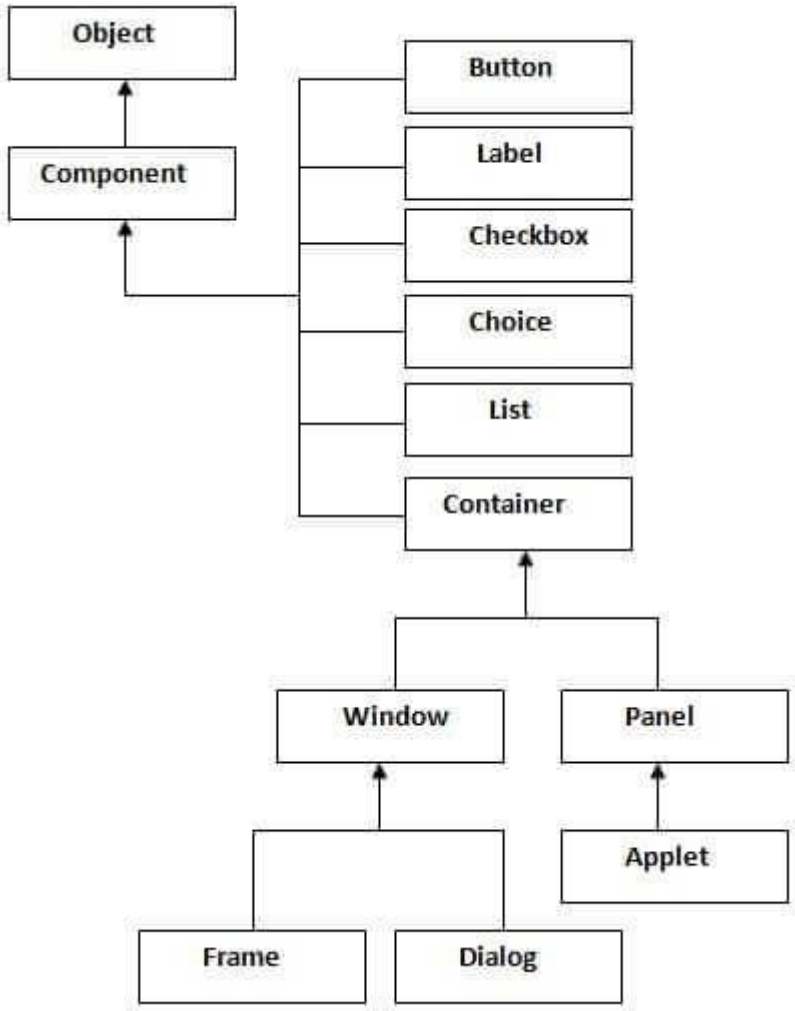
Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

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

The hierarchy of Java AWT classes are given below.



Container

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

Window

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

Panel

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

Frame

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

Useful Methods of Component class

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

Java AWT Example

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

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

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
1. import java.awt.*;
2. class First extends Frame{
3. First(){
4. Button b=new Button("click me");
5. b.setBounds(30,100,80,30);// setting button position
6. add(b);//adding button into frame
7. setSize(300,300);//frame size 300 width and 300 height
8. setLayout(null);//no layout manager
9. setVisible(true);//now frame will be visible, by default not visible
10. }
11. public static void main(String args[]){
12. First f=new First();
13. }}
```

download this example

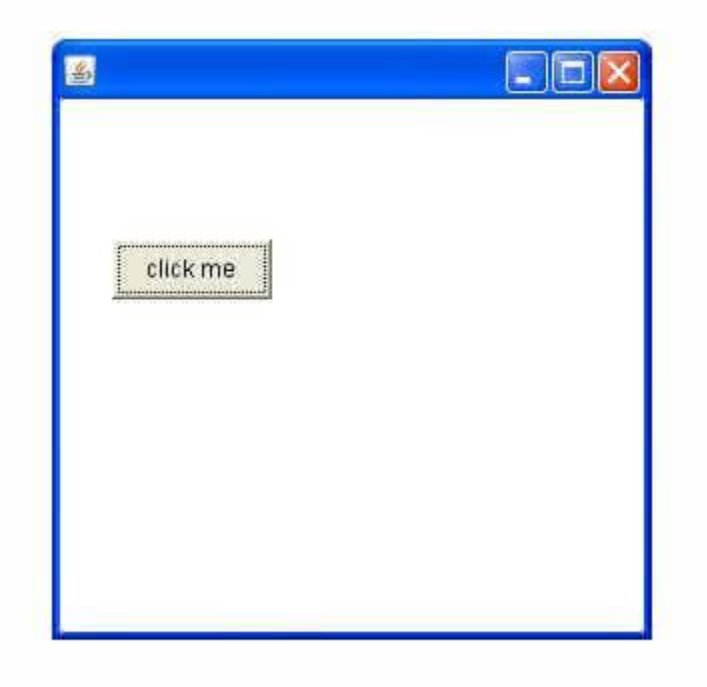
The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
1. import java.awt.*;
2. class First2{
3. First2(){
4. Frame f=new Frame();
5. Button b=new Button("click me");
6. b.setBounds(30,50,80,30);
7. f.add(b);
8. f.setSize(300,300);
9. f.setLayout(null);
10. f.setVisible(true);
11. }
12. public static void main(String args[]){
13. First2 f=new First2();
14. }}
```

download this example



Working with Windows Graphics and Text.

Introduction

The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well.

A Graphics object encapsulates all state information required for the basic rendering operations that Java supports. State information includes the following properties.

- The Component object on which to draw.
- A translation origin for rendering and clipping coordinates.
- The current clip.
- The current color.
- The current font.
- The current logical pixel operation function.
- The current XOR alternation color

Class declaration

Following is the declaration for **java.awt.Graphics** class:

```
public abstract class Graphics
  extends Object
```

Class constructors

S.N.	Constructor & Description
1	**Graphics() ()** Constructs a new Graphics object.

Class methods

S.N.	Method & Description
1	**abstract void clearRect(int x, int y, int width, int height)** Clears the specified rectangle by filling it with the background color of the current drawing surface.
2	**abstract void clipRect(int x, int y, int width, int height)** Intersects the current clip with the specified rectangle.
3	**abstract void copyArea(int x, int y, int width, int height, int dx, int dy)** Copies an area of the component by a distance specified by dx and dy.
4	**abstract Graphics create()** Creates a new Graphics object that is a copy of this Graphics object.
5	**Graphics create(int x, int y, int width, int height)** Creates a new Graphics object based on this Graphics object, but with a new translation and clip area.
6	**abstract void dispose()** Disposes of this graphics context and releases any system resources that it is using.
7	**void draw3DRect(int x, int y, int width, int height, boolean raised)** Draws a 3-D highlighted outline of the specified rectangle.
8	**abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)** Draws the outline of a circular or elliptical arc covering the specified rectangle.
9	**void drawBytes(byte[] data, int offset, int length, int x, int y)** Draws the text given by the specified byte array, using this graphics context's current font and color.
10	**void drawChars(char[] data, int offset, int length, int x, int y)** Draws the text given by the specified character array, using this graphics context's current font and color.
11	**abstract boolean drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)** Draws as much of the specified image as is currently available.
12	**abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)** Draws as much of the specified image as is currently available.
13	**abstract boolean drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)** Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
14	**abstract boolean drawImage(Image img, int x, int y, int width, int height,**

	ImageObserver observer) Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
15	**abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, Color bgcolor, ImageObserver observer)** Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
16	**abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)** Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
17	**abstract void drawLine(int x1, int y1, int x2, int y2)** Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
18	**abstract void drawOval(int x, int y, int width, int height)** Draws the outline of an oval.
19	**abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)** Draws a closed polygon defined by arrays of x and y coordinates.
20	**void drawPolygon(Polygon p)** Draws the outline of a polygon defined by the specified Polygon object.
21	**abstract void drawPolyline(int[] xPoints, int[] yPoints, int nPoints)** Draws a sequence of connected lines defined by arrays of x and y coordinates.
22	**void drawRect(int x, int y, int width, int height)** Draws the outline of the specified rectangle.
23	**abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)** Draws an outlined round-cornered rectangle using this graphics context's current color.
24	**abstract void drawString(AttributedCharacterIterator iterator, int x, int y)** Renders the text of the specified iterator applying its attributes in accordance with the specification of the TextAttribute class.
25	**abstract void drawString(String str, int x, int y)** Draws the text given by the specified string, using this graphics context's current font and color.
26	**void fill3DRect(int x, int y, int width, int height, boolean raised)** Paints a 3-D highlighted rectangle filled with the current color.
27	**abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)** Fills a circular or elliptical arc covering the specified rectangle.

28	**abstract void fillOval(int x, int y, int width, int height)** Fills an oval bounded by the specified rectangle with the current color.
29	**abstract void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)** Fills a closed polygon defined by arrays of x and y coordinates.
30	**void fillPolygon(Polygon p)** Fills the polygon defined by the specified Polygon object with the graphics context's current color.
31	**abstract void fillRect(int x, int y, int width, int height)** Fills the specified rectangle.
32	**abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)** Fills the specified rounded corner rectangle with the current color.
33	**void finalize()** Disposes of this graphics context once it is no longer referenced.
34	**abstract Shape getClip()** Gets the current clipping area.
35	**abstract Rectangle getClipBounds()** Returns the bounding rectangle of the current clipping area.
36	**Rectangle getClipBounds(Rectangle r)** Returns the bounding rectangle of the current clipping area.
37	**Rectangle getClipRect()** Deprecated. As of JDK version 1.1, replaced by getClipBounds().
38	**abstract Color getColor()** Gets this graphics context's current color.
39	**abstract Font getFont()** Gets the current font.
40	**FontMetrics getFontMetrics()** Gets the font metrics of the current font.
41	**abstract FontMetrics getFontMetrics(Font f)** Gets the font metrics for the specified font.
42	**boolean hitClip(int x, int y, int width, int height)** Returns true if the specified rectangular area might intersect the current clipping area.
43	**abstract void setClip(int x, int y, int width, int height)** Sets the current clip to the rectangle specified by the given coordinates.

44	**abstract void setClip(Shape clip)** Sets the current clipping area to an arbitrary clip shape.
45	**abstract void setColor(Color c)** Sets this graphics context's current color to the specified color.
46	**abstract void setFont(Font font)** Sets this graphics context's font to the specified font.
47	**abstract void setPaintMode()** Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color.
48	**abstract void setXORMode(Color c1)** Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color.
49	**String toString()** Returns a String object representing this Graphics object's value.
50	**abstract void translate(int x, int y)** Translates the origin of the graphics context to the point (x, y) in the current coordinate system.

Methods inherited

This class inherits methods from the following classes:

- java.lang.Object

Graphics Example

Create the following java program using any editor of your choice in say **D:/ > AWT > com > tutorialspoint > gui >**

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class AWTGraphicsDemo extends Frame {

  public AWTGraphicsDemo(){
    super("Java AWT Examples");
    prepareGUI();
  }

  public static void main(String[] args){
    AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
    awtGraphicsDemo.setVisible(true);
  }

  private void prepareGUI(){
```

```
    setSize(400,400);
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
        System.exit(0);
      }
    });
  }

  @Override
  public void paint(Graphics g) {
    g.setColor(Color.GRAY);
    Font font = new Font("Serif", Font.PLAIN, 24);
    g.setFont(font);
    g.drawString("Welcome to TutorialsPoint", 50, 150);
  }
}
```

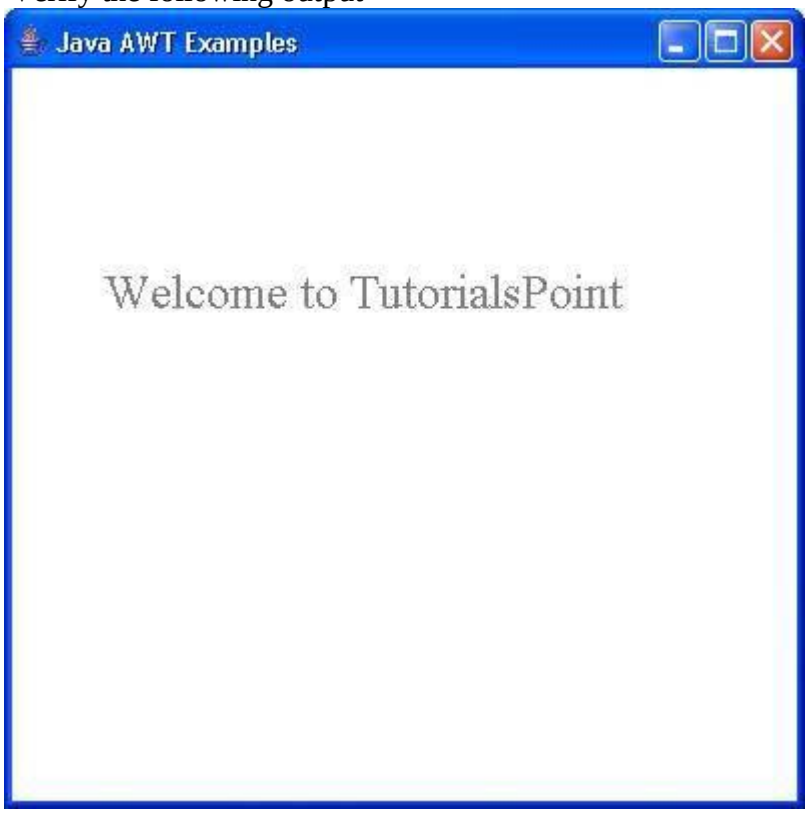
Compile the program using command prompt. Go to **D:/ > AWT** and type the following command.

```
D:\AWT>javac com\tutorialspoint\gui\AWTGraphicsDemo.java
```

If no error comes that means compilation is successful. Run the program using following command.

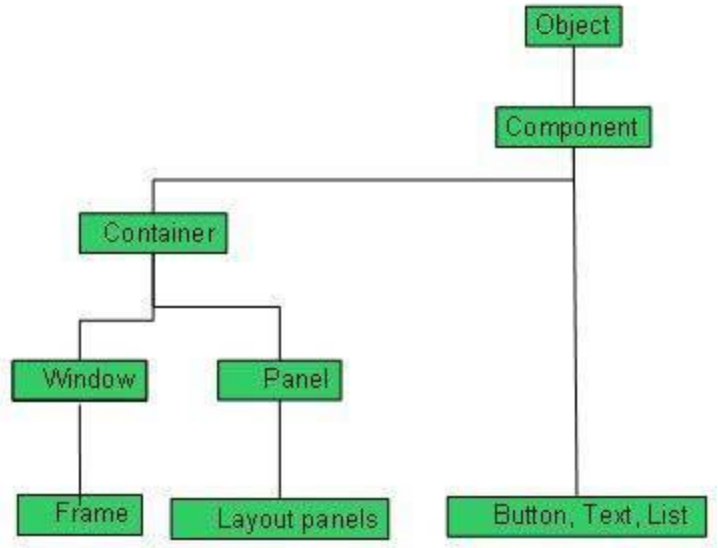
```
D:\AWT>java com.tutorialspoint.gui.AWTGraphicsDemo
```

Verify the following output



Every user interface considers the following three main aspects:

- **UI elements** : Thes are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex which we will cover in this tutorial.
- **Layouts:** They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in Layout chapter.
- **Behavior:** These are events which occur when the user interacts with UI elements. This part will be covered in Event Handling chapter.



Every AWT controls inherits properties from Component class.

Sr. No.	Control & Description
1	Component A Component is an abstract super class for GUI controls and it represents an object with graphical representation.

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.

Layout Managers and Menus:

Introduction

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. It is possible to layout the controls by hand but it becomes very difficult because of the following two reasons.

- It is very tedious to handle a large number of controls within the container.
- Oftenly the width and height information of a component is not given when we need to arrange them.

Java provide 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.

Sr. No.	Interface & Description
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.

Sr. No.	LayoutManager & Description
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.

Servlet:

Servlets | Servlet Tutorial

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.
There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

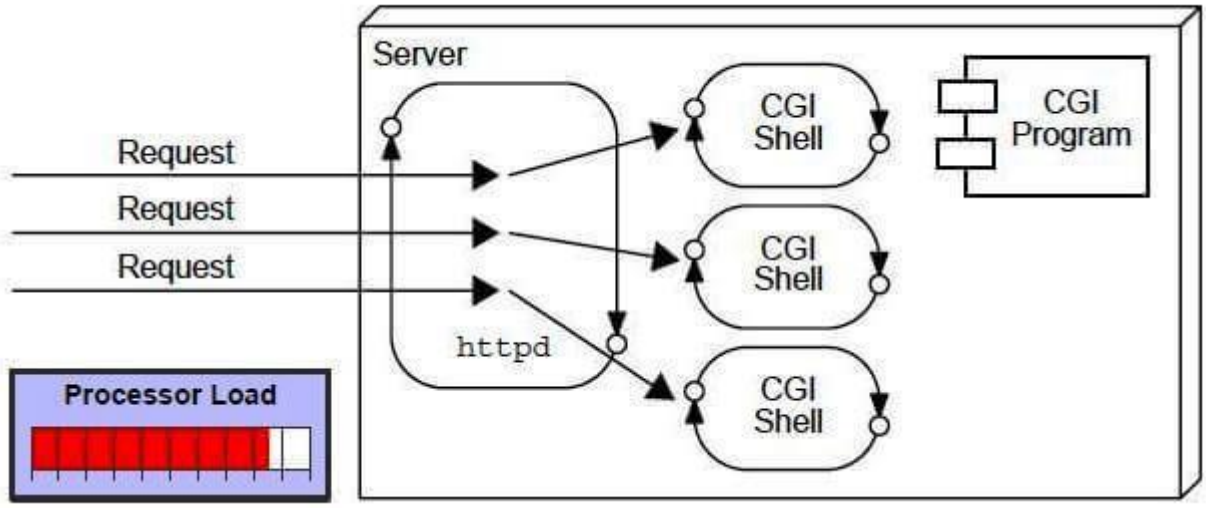
What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

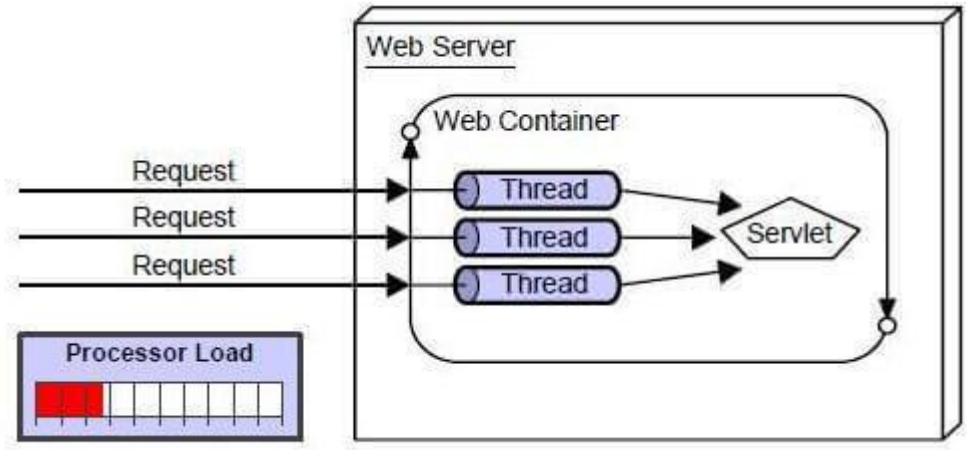


Disadvantages of CGI

There are many problems in CGI technology:

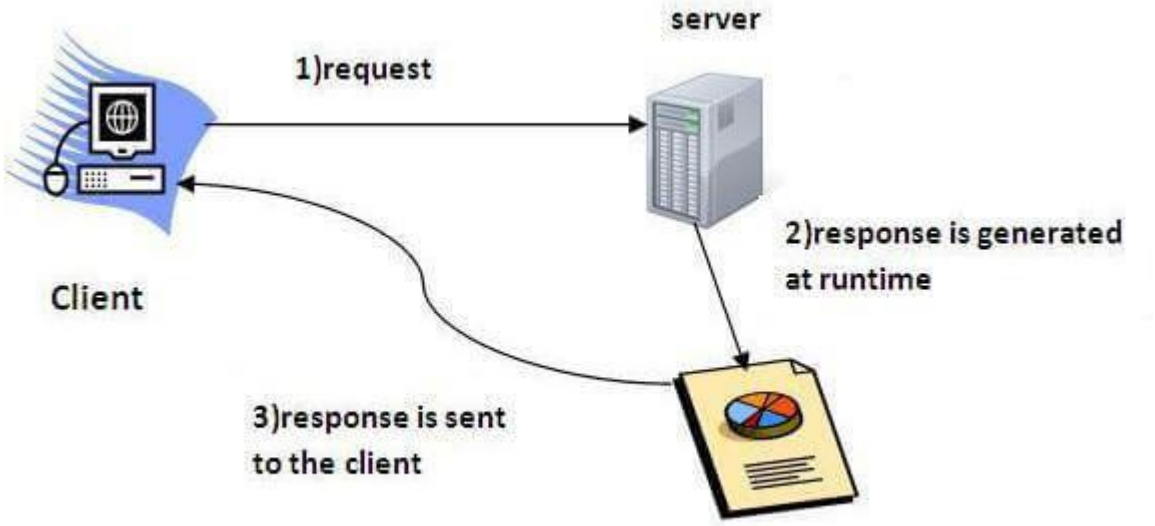
1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantages of Servlet



There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

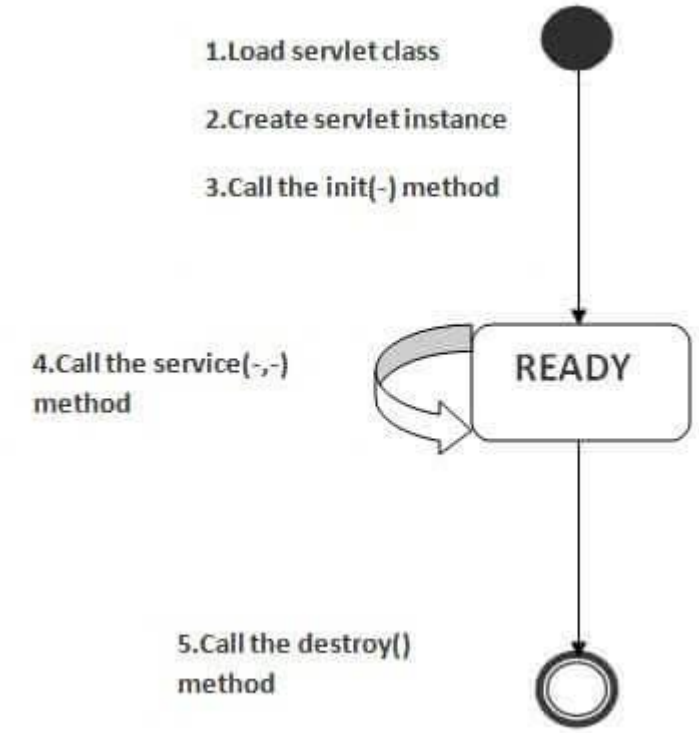
1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.



Life cycle of an Servlet:

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

---

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the Javax.servlet.Servlet interface. Syntax of the init method is given below:

```
1. public void init(ServletConfig config) throws ServletException
```

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

```
1. public void service(ServletRequest request, ServletResponse response)
2.   throws ServletException, IOException
```

---

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

```
1. public void destroy()
```

The Servlet API:

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api. The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Handling HTTP Request and Response:

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

- A Request-line
- Zero or more header (General|Request|Entity) fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF)
- indicating the end of the header fields
- Optionally a message-body

The following sections explain each of the entities used in an HTTP request message.

Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by space SP characters.

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Let's discuss each of the parts mentioned in the Request-Line.

Request Method

The request **method** indicates the method to be performed on the resource identified by the given **Request-URI**. The method is case-sensitive and should always be mentioned in uppercase. The following table lists all the supported methods in HTTP/1.1.

S.N.	Method and Description
1	**GET** The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** Same as GET, but it transfers the status line and the header section only.
3	**POST** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4	**PUT** Replaces all the current representations of the target resource with the uploaded content.
5	**DELETE** Removes all the current representations of the target resource given by URI.
6	**CONNECT** Establishes a tunnel to the server identified by a given URI.
7	**OPTIONS** Describe the communication options for the target resource.
8	**TRACE** Performs a message loop back test along with the path to the target resource.

Request-URI

The Request-URI is a Uniform Resource Identifier and identifies the resource upon which to apply the request. Following are the most commonly used forms to specify an URI:

Request-URI = "*" | absoluteURI | abs_path | authority

S.N.	Method and Description
1	The asterisk ***** is used when an HTTP request does not apply to a particular resource, but to the server itself, and is only allowed when the method used does not necessarily apply to a resource. For example: **OPTIONS * HTTP/1.1**
2	The **absoluteURI** is used when an HTTP request is being made to a proxy. The proxy is requested to forward the request or service from a valid cache, and return the response. For example: **GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1**
3	The most common form of Request-URI is that used to identify a resource on an origin server or gateway. For example, a client wishing to retrieve a resource directly from the origin server would create a TCP connection to port 80 of the host "www.w3.org" and send the following lines: **GET /pub/WWW/TheProject.html HTTP/1.1** **Host: www.w3.org** Note that the absolute path cannot be empty; if none is present in the original URI, it MUST be given as "/" (the server root).

Request Header Fields

We will study General-header and Entity-header in a separate chapter when we will learn HTTP header fields. For now, let's check what Request header fields are.

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers.Here is a list of some important Request-header fields that can be used based on the requirement:

- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Expect
- From
- Host
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- Proxy-Authorization
- Range
- Referer
- TE
- User-Agent

You can introduce your custom fields in case you are going to write your own custom Client and Web Server.

Examples of Request Message

Now let's put it all together to form an HTTP request to fetch **hello.htm** page from the web server running on tutorialspoint.com

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Here we are not sending any request data to the server because we are fetching a plain HTML page from the server. Connection is a general-header, and the rest of the headers are request headers. The following example shows how to send form data to the server using request message body:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

Here the given URL */cgi-bin/process.cgi* will be used to process the passed data and accordingly, a response will be returned. Here **content-type** tells the server that the passed data is a simple web form data and **length** will be the actual length of the data put in the message body. The following example shows how you can pass plain XML to your web server:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

Using Cookies:

In the last guide, I have covered Sessions in Servlet. Here we will discuss Cookies which is also used for session management. Let’s recall few things here from last tutorial so that we can relate sessions and cookies. When a user visits web application first time, the servlet container crates new HttpSession object by calling request.getSession(). A unique Id is assigned to the session. The **Servlet container also sets a Cookie in the header of the HTTP response with cookie name and the unique session ID as its value.**

The cookie is stored in the user browser, the client (user's browser) sends this cookie back to the server for all the subsequent requests until the cookie is valid. **The Servlet container checks the request header for cookies and get the session information from the cookie and use the associated session from the server memory.**
The session remains active for the time specified in tag in web.xml. If tag in not set in web.xml then the session remains active for 30 minutes. **Cookie remains active as long as the user's browser is running**, as soon as the browser is closed, the cookie and associated session info is destroyed. So when the user opens the browser again and sends request to web server, the new session is being created.

Types of Cookies

We can classify the cookie based on their expiry time:

1. Session
2. Persistent

1) SessionCookies:

Session cookies do not have expiration time. It lives in the browser memory. As soon as the web browser is closed this cookie gets destroyed.

2) Persistent Cookies:

Unlike Session cookies they have expiration time, they are stored in the user hard drive and gets destroyed based on the expiry time.

How to send Cookies to the Client

Here are steps for sending cookie to the client:

1. Create a Cookie object.
2. Set the maximum Age.
3. Place the Cookie in HTTP response header.

1) Create a Cookie object:

```
Cookie c = new Cookie("userName","Chaitanya");
```

2) Set the maximum Age:

By using **setMaxAge ()** method we can set the maximum age for the particular cookie in seconds.

```
c.setMaxAge(1800);
```

3) Place the Cookie in HTTP response header:

We can send the cookie to the client browser through response.addCookie() method.

```
response.addCookie(c);
```

How to read cookies

```
Cookie c[]=request.getCookies();
//c.length gives the cookie count
for(int i=0;i<c.length;i++){
 out.print("Name: "+c[i].getName()+" & Value: "+c[i].getValue());
}
```

Example of Cookies in java servlet

index.html

```
<form action="login">
 User Name:<input type="text" name="userName"/><br/>
 Password:<input type="password" name="userPassword"/><br/>
 <input type="submit" value="submit"/>
</form>
```

MyServlet1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class MyServlet1 extends HttpServlet
{
  public void doGet(HttpServletRequest request,
    HttpServletResponse response) {
    try{
       response.setContentType("text/html");
       PrintWriter pwriter = response.getWriter();

       String name = request.getParameter("userName");
       String password = request.getParameter("userPassword");
       pwriter.print("Hello "+name);
       pwriter.print("Your Password is: "+password);

       //Creating two cookies
       Cookie c1=new Cookie("userName",name);
       Cookie c2=new Cookie("userPassword",password);

       //Adding the cookies to response header
       response.addCookie(c1);
       response.addCookie(c2);
       pwriter.print("<br><a href='welcome'>View Details</a>");
       pwriter.close();
  }catch(Exception exp){
     System.out.println(exp);
   }
  }
}
```

MyServlet2.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet2 extends HttpServlet {
 public void doGet(HttpServletRequest request,
   HttpServletResponse response){
   try{
     response.setContentType("text/html");
     PrintWriter pwriter = response.getWriter();

     //Reading cookies
     Cookie c[]=request.getCookies();
     //Displaying User name value from cookie
     pwriter.print("Name: "+c[1].getValue());
     //Displaying user password value from cookie
     pwriter.print("Password: "+c[2].getValue());

     pwriter.close();
   }catch(Exception exp){
     System.out.println(exp);
    }
  }
```

```
}
```

web.xml

```
<web-app>
<display-name>BeginnersBookDemo</display-name>
 <welcome-file-list>
 <welcome-file>index.html</welcome-file>
 </welcome-file-list>
<servlet>
 <servlet-name>Servlet1</servlet-name>
 <servlet-class>MyServlet1</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>Servlet1</servlet-name>
 <url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet>
 <servlet-name>Servlet2</servlet-name>
 <servlet-class>MyServlet2</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>Servlet2</servlet-name>
 <url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Output:

Welcome Screen:

After clicking Submit:

After clicking View Details:

Methods of Cookie class

public void setComment(String purpose): This method is used for setting up comments in the cookie. This is basically used for describing the purpose of the cookie.

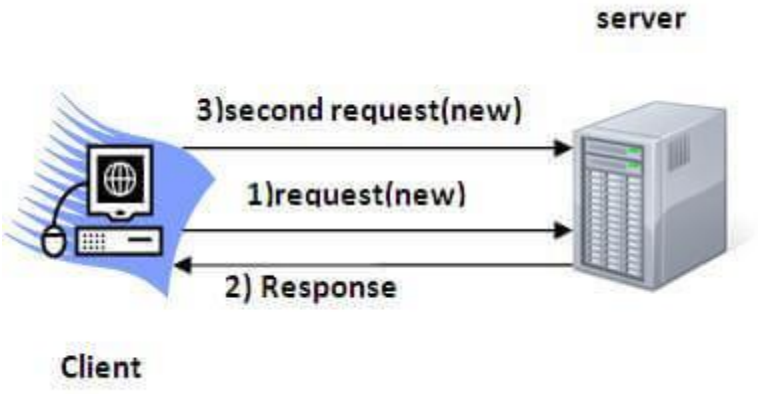
public String getComment(): Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
public void setMaxAge(int expiry): Sets the maximum age of the cookie in seconds.
public int getMaxAge(): Gets the maximum age in seconds of this Cookie.
By default, -1 is returned, which indicates that the cookie will persist until browser shutdown.
public String getName(): Returns the name of the cookie. The name cannot be changed after creation.
public void setValue(String newValue): Assigns a new value to this Cookie.
public String getValue(): Gets the current value of this Cookie.
The list above has only commonly used methods. To get the complete list of methods of Cookie class refer official documentation.

Session Tracking:

Session Tracking in Servlets

1. Session Tracking
2. Session Tracking Techniques

Session simply means a particular interval of time.
Session Tracking is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.
HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?
To recognize the user It is used to recognize the particular user.

Session Tracking Techniques
There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

Introduction to JSP:

What is JavaServer Pages?
JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the **Common Gateway Interface (CGI)**. But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including **JDBC, JNDI, EJB, JAXP,** etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

Advantages of JSP

Following table lists out the other advantages of using JSP over other technologies −

vs. Active Server Pages (ASP)

The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

vs. Pure Servlets

It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

vs. Server-Side Includes (SSI)

SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

vs. JavaScript

JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

vs. Static HTML

Regular HTML, of course, cannot contain dynamic information.