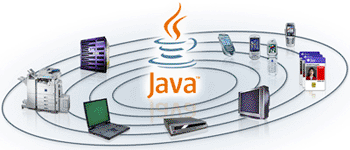
**7. Implementation**

**7.1.Technology Description**

**Technology Used:**

**About the Java Technology**

The Java platform consists of the Java application programming interfaces (APIs) and the Java virtual machine (JVM).



The following Java technology lets developers, designers, and business partners develop and deliver a consistent user experience, with one environment for applications on mobile and embedded devices. Java meshes the power of a rich stack with the ability to deliver customized experiences across such devices.

Java APIs are libraries of compiled code that you can use in your programs. They let you add ready-made and customizable functionality to save you programming time.  
Java programs are run (or interpreted) by another program called the Java Virtual Machine. Rather than running directly on the native operating system, the program is interpreted by the Java VM for the native operating system. This means that any computer system with the Java VM installed can run Java programs regardless of the computer system on which the applications were originally developed.

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains bytecodes — the machine language of the Java Virtual Machine (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris TM Operating System (Solaris OS), Linux, or Mac OS.

Java technology is both a programming language and a platform.

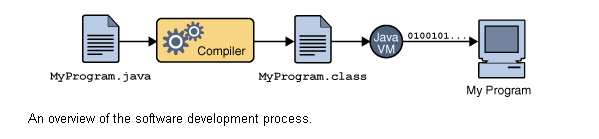
**The Java Programming Language**

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

|  |  |
| --- | --- |
| * Simple * Object oriented * Distributed * Multithreaded * Dynamic | * Architecture neutral * Portable * High performance * Robust * Secure |

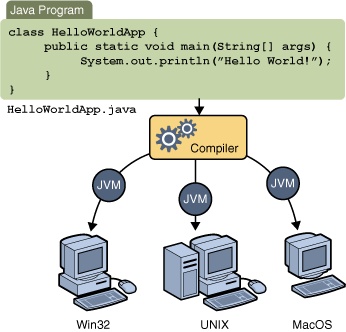
Each of the preceding buzzwords is explained in [*The Java Language Environment*](http://java.sun.com/docs/white/langenv/) , a white paper written by James Gosling and Henry McGilton.

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine[1](http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html#FOOT) (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.



An overview of the software development process.

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS. Some virtual machines, such as the [Java HotSpot virtual machine](http://java.sun.com/products/hotspot/), perform additional steps at runtime to give your application a performance boost. This include various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code



Through the Java VM, the same application is capable of running on multiple platforms.

**AWT**

AWT is the acronym of Abstract Window Toolkit.

**Introduction:**

GUI (Graphical User Interface) is a mediator (translator) between the end user and the program.

**Form :** It is a collection of GUIs designed for one specific purpose.

\* Console is a program, provided by the local OS, which supports only text output.

\* In order to display GUI outputs, we have another program (similar to the console) provided by the local os. This program is, window.

\* All the java classes, which are used to create GUIs are present in the package java.awt.

\* The theory behind the creation of GUIs is as follows.

\* The functionality of the constructor of the class, whose GUI is to be created, is to create that particular GUI.

\* Strictly speaking, this is not the functionality of the constructor of that class.

\* In fact, the constructor has some statements which are nothing but, a call to a specific file in the local os. This particular file will be responsible in creating the corresponding GUI. In the foreground, it seems as if the GUI is created just because of the constructor.

\* Whatever language we use to create the GUIs, (for example : java, html, vb e.t.c.) in all these cases, only the files present in the local os will be responsible for creating the corresponding GUIs.

\* Because of this, the GUIs created in all these cases will look alike.

\* Let us take java in particular.

\* When we want to create a GUI, we need to create object of that class which represents the corresponding GUI. Let us assume we need to create a GUI – a button. Then we need to write the following code, i.e.

\* Button b1 = new Button ();

\* So, to create the button GUI, JVM executes the constructor of the Button class. Inside the constructor we have statements which execute the files present in the local os to create the GUI.

\* The means, we are making JVM to execute the files present in the local operating system.

\* But remember that, JVM can execute only byte code, which is generated from the .class file and this .class file is results from a .java file.

\* The files present in the local operating system need not be in java and definitely they will not be in java.

\* They will be in some other native language.

**Question :** Then, how can JVM execute the files present in the local os to create the GUIs?

**Ans:** JVM depends upon some classes known as Peer Classes to execute the files present in the local os.

\* Thus, every awt component depends upon the files of the local os to create GUIs, making JVM to execute those files which are non-java files. This increases the overhead on the JVM. This is the reason why we call awt classes (components) as heavy-weight components

\* From the above discussion, it is clear that each os will have its own files for creating the GUIs.

\* Now, if we create a GUI using java on one os, then the look and feel of this GUI will be completely different when we execute the same java program on another OS.

**Question: Java is not purely platform independent. Justify.**

**Ans:** Because of the dependency of awt programs on the local os in creating GUIs, the outputs of these programs very from one os to another os. i.e. the outputs of the awt programs are platform dependent. This makes java to be platform-dependent.

This is the reason, why we say that java is not purely platform-independent.

**Disadvantages of AWT :**

(i) The AWT programs increase the overhead on the JVM.

(ii) The make java to be platform dependent.

\* The above disadvantages are over-come in Swings

\* Swings is an extension of AWT. The swing components do not depend upon the files of the local os in creating the GUIs and thereby, they decrease the overhead on the JVM.

\* Thus, swings are known as Light-Weight Components.

\* The following is the class hierarchy of java.awt package.

Picture1

NOTE: Applet class does not belong to java.awt package.

\* Applet class belongs to java. applet package.

\* We know that window is used for displaying the GUIs.

\* A blind rule is that, when we want to add a component to another component then the later component should have the properties of a container.

\* Whenever we want to view a GUI output, we add all the required components to the window and then they are displayed that means, window has the properties of a container.

\* From the class hierarchy, it is clear that window is the sub class of container class. Hence it has all the properties of a container.

\* If we want to add a Button to another button, it is not possible because, button is not a sub class of container and it does not have the properties of a container.

\* Same is the case with the other classes present in that level of hierarchy.

\* Similarly, we can not think of adding Text Filed to a Label, Text Area to a List e.t.c.

\* Thus, any component which has the properties of container is eligible to accommodate other components in it.

\* It is clear from the class hierarchy that panel is a sub-class of container and window is at the same level of hierarchy.

\* Panel has the properties of container and any no. of components can be added to the panel. To make them visible on the screen, we again need window, but observe that panel does not have the properties of a window.

\* Therefore, panel should be added to the window to make the components visible.

\* Similarly, we can create a Frame. Frame has the properties of window as well as container

\* As Button, Text Area e.t.c. have some properties, window also has some properties.

Picture1

- It has a title-bar

- It has some system Menus

- We should be able to drag it

- We should be able to resize it e.t.c.

\* Some of the best examples of the window application are Edit plus Editor, Notepad, IE browser e.t.c.

**NOTE:** As console cannot display GUIs, window cannot display text directly.

\* In order to display text on the window, we need to represent the text in the form of a GUI component and then add it to the window. For this, we make use of the Label class.

\* We always prefer to create a window through Frame because window alone has no feature of boundaries. The facility of having boundaries for a window is possible with Frame. This is the reason why we create window through Frame.

\* Using the functions of the Frame class we can set the dimensions (size) for the window.

**Ex:** f.setsize (400, 400);

**NOTE:** By default, window would always be created in the invisible mode. To make the window visible on the VDU, we should explicitly set the visibility property of the window to the Boolean value ‘true’.

**Ex:** f = set visible (true);

\* A constructor defined in the Frame class takes a string class object as an argument. Thus, whatever the text we want to see in the title bar of the window, we need to pass it as an argument (in the form of a string class object) to the constructor of the Frame class.

Let us develop a simple program which displays a GUI as shown aside.

Picture1

**Hint:** This GUI consists of six components

 two labels + two text fields + one button + one frame (window)

import java.awt.\*;

class Form1 Form1.java

{

public static void main (string args [])

{

Frame f1 = new Frame (“Login Page”);

Label l1 = new Label (“User Name”);

Label l2 = new Label (“Password”);

Text Field tf1 = new Text Field ();

Text Field tf2 = new Text Field ();

Button b1 = new Button (“submit”);

f1. add (l1);

f1. add (l2);

f1. add (tf1);

f1. add (tf2);

f1. add (b1);

f1. set size (437, 437);

f1. set visible (true);

}

}

Picture1

\* When we execute the above program, we can see only the button on the window. This is because, all the components have been added, but one upon another. The last component added is the button and hence we see only the button.

\* This resulted because we have not mentioned, to which part of the window, the components have to be added. Here they have occupied the entire window.

\* So, to make all the components visible we have to align them properly on the window.

\* We have some standard procedures using which we can align the components in the frame (container)

\* The standard procedures which exactly specify the way in which we align the components on the container are known as Layout Managers.

\* We have five Layout Managers in the awt package, where each Layout Manager is the name of a class.

(i) Border Layout (ii) Flow Layout

(iii) Grid Layout (iv) Grid Bag Layout

(v) Card Layout

\* Therefore, we should mention the layout according to which the components are to be aligned on the container.

\* We can mention the layout (specification or procedure) byusing the set Layout () method of the container class. This method is a non-static method.

\* The set Layout () method takes the object of any of the above five layouts as an argument, according to which the components would be aligned (arranged) on the containers.

**Border Layout:**

Picture1

\* According to the specifications of the Border Layout, the container area will be divided into five parts as shown in the fig.

\* The Border Layout class has some static constants to refer to these areas as Border Layout. SOUTH e.t.c.

\* The default layout associated with the Frame class object is Border Layout. The default location associated with the Border Layout is CENTER.

\* Therefore, when we call add () method on the Frame class object (i.e. f. add (xxx)), it takes the layout to be Border Layout and adds the component to the centre of it.

\* An important property of the Border Layout is, when a component is added to the CENTER, then the component not only occupies the CENTER but also stretches to the other four areas, if there are not components in those areas.

\* For example, if we add a button to the center, then it occupies the entire frame.

\* If we add two buttons – one to the NORTH and one to the center, then the button which is added to the CENTER also stretches to the EAST, WEST and SOUTH areas.

**Flow Layout:**

\* According to this layout, the container will be divided into rows and columns depending upon the components and their size.

\* When we start adding components to the frame according to Flow Layout, then, the components would be added row-wise i.e. first row  column-1, column-2, e.t.c. If the row is filled, then the next component would be added to first column of second row and so on.

**NOTE:** When we specify the layout as Flow Layout, then, initially it assumes the entire space as 1 row & 1 column.

\* When we add a component it would be added in the centre at the top (because initially it is 1 row & 1 column).

\* Now, when we add another component, control check whether this component can be accommodated to the right of the existing component.

\* It yes, it will be added, thus creating a second column.

\* If no, the component would be added in the next row.

\* Hence, size of the column completely depends upon the size of the component being added.

\* Therefore, it is obvious that, when we add components to the container (frame) according to Flow Layout, then the columns will be dynamically generated.

**NOTE:** The no. of rows and columns may or may not be equal. i.e. the no. of columns may vary from one row to another row, depending upon the size of the components being added.

**NOTE:** When we resize the frame (on which the components are arranged according to flow layout) then the columns are going to get adjusted accordingly and even the components are going to get adjusted accordingly.

**Ex:**  Picture1

// Modifying the program Form1. java

import java. awt. \*;

class Form2 Form2. java

{

Frame f1 = new Frame (“Login Page”);

Label l1 = new Label (“User Name”);

Label l2 = new Label (“Password”);

Text Field tf1 = new Text Field ();

Text Field tf2 = new Text Field ();

Button b1 = new Button (“SUBMIT”);

Form2 ()

{

f1. set Layout (new Flow Layout ());

f1. add (l1); f1. add (l2);

f1. add (tf1); f1. add (tf2);

f1. add (b1);

f1. set size (437, 440);

f1. set visible (true);

}

public static void main (string args [])

{

Form2 f2 = new Form2 ();

}

}

\* In the GUI programs which we have written till now, we have been creating objects of the Text Field class wherever necessary. Whether it is for a user name label or a password label, the Text Field is same. So, instead of creating another text field, can’t we add the text field (which was created once) again and again where necessary ?

**NOTE:** We can not reuse the GUI components once created in a container (frame) i.e. for example, if we have a text field already and if we need one more, then we can’t add the one created earlier. Instead, we should create another text field and add it.

**Grid Layout:**Picture1According to the specifications of Grid Layout, the container will be divided into specific no. of rows and columns of equal size.

\* Each division is called a cell or a grid.

\* In the Grid Layout, only one component can be added to a grid, and the component occupies entire grid.

\* The components will be added to the grids in a serial fashion and we can not skip the grids in between i.e. a component will not be added to grid-3, unless grid-2 is filled provided grid-1 is already occupied.

\* It is obvious that the constructor of Grid Layout class takes two integer arguments – row & column.

**Ex:** f1. set Layout (new Grid Layout(4, 4));

**NOTE:** When we use the Grid Layout, it is compulsory that all the grids should be occupied with the components. No grid should be left free.

**Panel:** Panel is a component which has the properties of container. It does not have the properties of a window. So, panel has to be added to the window (frame) to make it visible.

\* Since panel has the properties of a container, we can add any no. of components to the panel and now the panel will be treated as a single component.

**NOTE:** If we want to have more than one component in a grid, then we make use of the panel i.e. add all the required components to the panel and add this panel (which is a single component) to the grid.

\* If we want to have an empty grid, then add an empty panel (without any components on the panel) to the grid. This makes the grid look empty.

\* We do not have any separate class to create radio buttons. We create radio buttons using checkbox class.

\*After creating the checkboxes, if we group them, they will act as radio buttons, else they will act as checkboxes.

**Question:** Develop a GUI which meets the following requirements.

- It should have four panels.

- Panel-1 should have a drop down list and an ordinary list

and also a text area.

- Panel-2 and panel-3 are empty.

- panel-4 should have two checkboxes and two radio

buttons.

Ans: The GUI would be something as shown below.

Picture1

**Hint:** Here we have altogether 12 components as listed below

Frame-1 Choice-1 Text Area-1

Panels-4 List-1 Check boxes-2

Radio buttons-2

import java. awt. \*;

class Grid Demo1

{

Frame f1 = new Frame (“Grid-Demo”);

Panel p1 = new Panel ();

Panel p2 = new Panel ();

Panel p3 = new Panel ();

Panel p4 = new Panel ();

Choice ch = new Choice ();

List lst = new List ();

Text Area ta = new Text Area (9, 40);

Checkbox cb1 = new Checkbox (“sports”);

Checkbox r1, r2;

Grid Demo1

{

f1. set Layout (new Grid Layout (2, 2));

ch. add (“Andhra Pradesh”);

ch. add (“Karnataka”);

ch. add (“Tamilnadu”);

ch. add (“Kerala”);

lst. add (“Hyderabad”);

lst. add (“Bangalore”);

lst. add (“Chennai”);

lst. add (“Trivendrum”);

p1. add (ch);

p1. add (lst);

p1. add (ta);

p4. set Layout (new Grid Layout (4, 1));

p4. add (cb1);

p4. add (cb2);

checkbox Group cbg = new checkbox Group ();

r1 = new checkbox (“Male”, true, cbg);

r2 = new checkbox (“Female”, false, cbg);

p4. add (r1);

p4. add (r2);

f1. add (p1);

f1. add (p2);

f1. add (p3);

f1. add (p4);

f1. set size (437, 437);

f1. set visible (true);

}

public static void main (string args [])

{

Grid Demo1 gd = new Grid Demo1 ();

}

}

**NOTE:** The constructor of the checkbox class takes three arguments – strings object (i.e. name), Boolean value (to select or deselect the given radio button), the checkbox Group class object.

**Grid Bag Layout**

\* The Grid Bag Layout is similar to the geed Layout, but with some additional features

Picture1\* In the gird layout, we have the facility of adding the components to a specific part of the grid (by mentioning the coordination’s on all the four sides.

\* we can stretch the components between the girds either horizontally or vertically

**Note:-**

Even though grid Bag layout is advantageous it become it increases the complexity of the program

\* we have a layout by name box layout in the swings package we use this layout to achieve all that we do using grid Bag Layout and with less complexity

Picture1**Card Layout:**

The card layout specifications help us to add the components to a frame w.r.t z-axis i.e. one component behind the other as shown in the figure

\* card lay out is closely related to event handling

**Note:-**

Actually set layout () method is defined to accept, an object of an interface, as an argument the interface is layout manager

\* All the layout classes implements the layout manager interface thus, object of any class, which implements layout manager interface , can be passed as an arguments to the set layout () method

\* Obviously, null can also be passed as an argument to the method