

---

# UNIT 1 APPLET

---

Structure	Page Nos.
1.0 Introduction	5
1.1 Objectives	5
1.2 The Applet Class	5
1.3 Applet Architecture	6
1.4 An Applet Skeleton: Initialization and Termination	8
1.5 Handling Events	12
1.6 HTML Applet Tag	16
1.7 Summary	21
1.8 Solutions/Answers	22

---

## 1.0 INTRODUCTION

---

You are already familiar with several aspects of Java applications programming discussed in earlier blocks. In this unit you will learn another type of Java programs called Java Applets. As you know in Java you can write two types of programmes,—Applications and Applets. Unlike a Java application that executes from a command window, an Applet runs in the *Appletviewer* ( *a test utility for Applets that is included with the J2SDK*) or a World Wide Web browser such as Microsoft Internet Explorer or Netscape Communicator.

In this unit you will also learn how to work with *event driven programming*. You are very familiar with Windows applications. In these application environments program logic doesn't flow from the top to the bottom of the program as it does in most procedural code. Rather, the operating system collects *events* and the program responds to them. These events may be mouse clicks, key presses, network data arriving on the Ethernet port, or any from about two dozen other possibilities. The operating system looks at each event, determines what program it was intended for, and places the event in the appropriate program's *event queue*.

Every application program has an *event loop*. This is just a while loop which loops continuously. On every pass through the loop the application retrieves the next event from its event queue and responds accordingly.

---

## 1.1 OBJECTIVES

---

Our objective is to introduce you to Java Applets. After going through this unit, you will be able to:

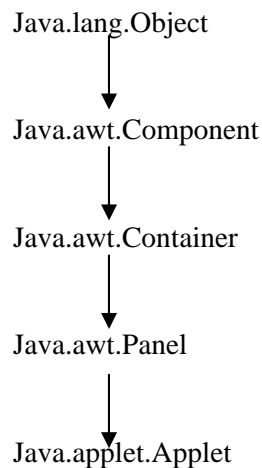
- describe Java Applets and its Importance;
  - explain Applet architecture;
  - compile and Execute Java Applet programs;
  - use Event Handling constructs of Java programs, and
  - apply various HTML tags to execute the Applet programs.
- 

## 1.2 THE APPLET CLASS

---

Java is an Object Oriented Programming language, supported by various classes. The Applet class is packed in the *Java. Applet* package which has several interfaces. These interfaces enable the creation of Applets, interaction of Applets with the browser, and playing audio clips in Applets. In Java 2, class *Javax.swing. JApplet* is used to define an Applet that uses the *Swing GUI components*.

As you know, in Java class hierarchy Object is the base class of Java.lang package. The Applet is placed into the hierarchy as follows:



Now let us see what you should do using Java Applet and what you should not do using it.

Below is the list given for Do's and Don'ts of Java Applets:

### **Do's**

- Draw pictures on a web page
- Create a new window and draw the picture in it.
- Play sounds.
- Receive input from the user through the keyboard or the mouse.
- Make a network connection to the server from where the Applet is downloaded, and send to and receive arbitrary data from that server.

### **Don'ts**

- Write data on any of the host's disks.
- Read any data from the host's disks without the user's permission. In some environments, notably Netscape, an Applet cannot read data from the user's disks even with permission.
- Delete files
- Read from or write to arbitrary blocks of memory, even on a non-memory-protected operating system like the MacOS
- Make a network connection to a host on the Internet other than the one from which it was downloaded.
- Call the native API directly (though Java API calls may eventually lead back to native API calls).
- Introduce a virus or Trojan horse into the host system.

Now we will discuss different components in Applet architecture.

---

## **1.3 APPLET ARCHITECTURE**

---

Java Applets are essentially Java programs that run within a web page. Applet programs are Java classes that extend the Java.Applet.Applet class and are embedded by reference within a HTML page. You can observe that when Applets are combined with HTML, they can make an interface more dynamic and powerful than with HTML alone. While some Applets do nothing more than scroll text or play animations, but by incorporating these basic features in web pages you can make them dynamic. These dynamic web pages can be used in an enterprise application to view or manipulate data coming from some source on the server. For example, an

Applet may be used to browse and modify records in a database or control runtime aspects of some other application running on the server.

Besides the class file defining the Java Applet itself, Applets can use a collection of utility classes, either by themselves or archived into a JAR file. The Applets and their class files are distributed through standard HTTP requests and therefore can be sent across firewalls with the web page data. Applet code is refreshed automatically each time the user revisits the hosting web site. Therefore, keeps full application up to date on each client desktop on which it is running.

Since Applets are extensions of the Java platform, you can reuse existing Java components when you build web application interface with Applets. As we'll see in example programs in this unit, we can use complex Java objects developed originally for server-side applications as components of your Applets. In fact, you can write Java code that can operate as either an Applet or an application.

In Figure 1, you can see that using Applet program running in a Java-enabled web browser you can communicate to the server.

### Basic WWW/Applet Architecture

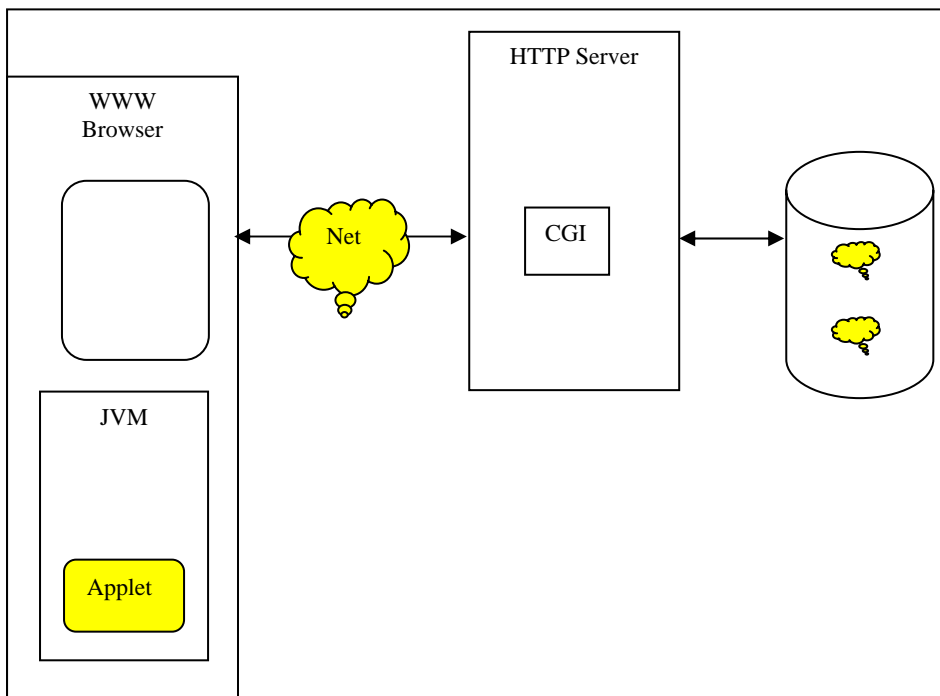


Figure1: Applet Architecture



### Check Your Progress 1

- 1) Explain what an Applet is.

.....

.....

.....

.....

...

.....

...

- .....
- ...
- 2) Explain of Applets security.

- .....
- .....
- .....
- .....
- 3) What are the various ways to execute an Applet?

- .....
- .....
- .....
- .....
- 4) Why do you think that OOPs concepts are important in the current scenario?

Now we will discuss the Applet methods around which Applet programming moves.

---

## 1.4 AN APPLLET SKELETON: INITIALIZATION AND TERMINATION

---

In this section we will discuss about the Applet life cycle. If you think about **the Basic Applet Life Cycle, the points listed below should be thought of:**

1. The browser reads the HTML page and finds any <APPLET> tags.
2. The browser parses the <APPLET> tag to find the CODE and possibly CODEBASE attribute.
3. The browser downloads the. **Class** file for the Applet from the URL (Uniform Resource Locator) found in the last step.
4. The browser converts the raw bytes downloaded into a Java class, that is a Java.lang.Class object.
5. The browser instantiates the Applet class to form an Applet object. This requires the Applet to have a no-args constructor.
6. The browser calls the Applet's init () method.
7. The browser calls the Applet's start () method.
8. While the Applet is running, the browser passes all the events intended for the Applet, like mouse clicks, key presses, etc. to the Applet's handle Event () method.
9. The default method paint() in Applets just draw messages or graphics (such as lines, ovals etc.) on the screen. Update events are used to tell the Applet that it needs to repaint itself.
10. The browser calls the Applet's stop () method.

11. The browser calls the Applet's destroy () method.

In brief, you can say that, all Applets use their five following methods:

```
public void init ();
public void start();
public void paint();
public void stop();
public void destroy();
```

Every Applet program use these methods in its life cycle. Their methods are defined in super class, Java.applet. Applet (It has others too, but right now I just want to talk about these five.

In the super class, these are simply do-nothing methods. Subclasses may override these methods to accomplish certain tasks at certain times. For example,

```
public void init() { }
```

init () method is used to read parameters that were passed to the Applet via <PARAM> tags because it's called exactly once when the Applet starts up. If there is such need in your program you can override init() method. Since these methods are declared in the super class, the Web browser can invoke them when it needs, without knowing in advance whether the method is implemented in the super class or the subclass. This is a good example of polymorphism.

A brief description of **init ()**, **start ()**, **paint ()**, **stop ()**, and **destroy ()** methods are given below.

**init () method:** The init() method is called *exactly once* in an Applet's life, when the Applet is first loaded. It's normally used to read PARAM tags, start downloading any other images or media files you need, and to set up the user interface. Most Applets have init () methods.

**start () method:** The start() method is called at *least once* in an Applet's life, when the Applet is started or restarted. In some cases it may be called more than once. Many Applets you write will not have explicit start () method and will merely inherit one from their super class. A start() method is often used to start any threads the Applet will need while it runs.

**paint () method:** The task of paint () method is to draw graphics (such as lines, rectangles, string on characters on the screen).

**stop() method:** The stop () method is called at least once in an Applet's life, when the browser leaves the page in which the Applet is embedded. The Applet's start () method will be called if at some later point the browser returns to the page containing the Applet. In some cases the stop () method may be called multiple times in an Applet's life. Many Applets you write will not have explicit stop () methods and will merely inherit one from their super class. Your Applet should use the stop () method to pause any running threads. When your Applet is stopped, it *should* not use any CPU cycles.

**destroy () method:** The destroy() method is called exactly once in an Applet's life, just before the web browser unloads the Applet. This method is generally used to perform any final clean-up. For example, an Applet that stores state on the server might send some data back to the server before it is terminated. Many Applet programs generally don't have explicit destroy () methods and just inherit one from their super class.

Let us take one example to explain the use of the methods explained above. For example in a video Applet, the init () method might draw the controls and start loading the video file. The start () method would wait until the file was loaded, and then start playing it. The stop () method would pause the video, but not rewind it. If the start () method were called again, the video would pick up from where it left off; it would not start over from the beginning. However, if destroy () were called and then init (), the video would start over from the beginning.

The point to note here is, if you run an Applet program using Appletviewer, selecting the restart menu item calls stop () and then start (). Selecting the Reload menu item calls stop (), destroy (), and init (), in the order.

**Note 1:** The Applet start () and stop () methods are not related to the similarly named methods in the Java.lang.Thread class, you have studied in block 3 unit 1.

**Note 2:** i) Your own code may occasionally invoke start() and stop(). For example, it is customary to stop playing an animation when the user clicks the mouse in the Applet and restart it when s/he clicks the mouse again.

ii) Now we are familiar with the basic needs and ways to write Applet program. You can see a simple Applet program given below to say Hello World.

```
import Java.applet.Applet;
import Java.awt.Graphics;
public class HelloWorldApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Hello world!", 50, 25);
    }
}
```

If you observe, that this Applet version of Hello World is a little more complicated than to write an application program to say Hello World, it will take a little more effort to run it as well.

First you have to type in the source code and save it into file called HelloWorldApplet.Java.

Compile this file in the usual way.

If all is well a file called HelloWorldApplet.class will be created.

Now you need to create an HTML file that will include your Applet. The following simple HTML file will do.

```
<HTML>
<HEAD>
<TITLE> Hello World </TITLE>
</HEAD>
<BODY>
This is the Applet:<P>
<Applet code="HelloWorldApplet" width="150" height="50">
</Applet>
</BODY>
</HTML>
```

Save this file as HelloWorldApplet.html in the same directory as the HelloWorldApplet.class file.

When you've done that, load the HTML file into a Java enabled browser like Internet Explorer or Sun's Applet viewer. You should see something like below, though of course the exact details depend on which browser you use.

If the Applet is compiled without error and produced a HelloWorldApplet.class file, and yet you don't see the string "Hello World" in your browser chances are that the class file is in the wrong place. Make sure HelloWorldApplet.class is in the same directory as HelloWorld.html.

Also make sure that you're using a version of Netscape or Internet Explorer which supports Java. Not all versions do.

In any case Netscape's Java support is less than the perfect, so if you have trouble with an Applet, the first thing to try is load it into Sun's Applet Viewer. If the Applet Viewer has a problem, then chances are pretty good the problem is with the Applet and not with the browser.

According to Sun *"An Applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application. The Applet class provides a standard interface between Applets and their environment."*

The output of the program will be like:



You know GUI programming is heavily dependent on events like mouse click, button pressed, key pressed, etc. Java also supports event handling. Now let us see how these events are handled in Java.

## Check Your Progress 2

- 1) What is the sequence of interpretation, compilation of a Java Applet?  
.....  
.....  
.....
- 2) How can you re-execute an Applet from Appletviewer?  
.....  
.....  
.....  
.....
- 3) Give in brief the description of an Applet life cycle.  
.....  
.....

.....  
.....  
.....  
...  
.....  
...

---

## 1.5 HANDLING EVENTS

---

You are leaving for work in the morning and someone rings the doorbell....  
That is an event!

In life, you encounter events that force you to suspend other activities and respond to them immediately. In Java, events represent all actions that go on between the user and the application. Java's Abstract Windowing Toolkit (AWT) communicates these actions to the programs using events. When the user interacts with a program let us say by clicking a command button, the system creates an event representing the action and delegates it to the event-handling code within the program. This code determines how to handle the event so the user gets the appropriate response.

Originally, JDK 1.0.2 applications handled events using an inheritance model. A container sub class inherits the `action()` and `handleEvent()` methods of its parent and handled the events for all components it contained. For instance, the following code would represent the event handler for a panel containing an OK and a Cancel button:

```
public boolean handleEvent(Java.awt.Event e)
{
    if (e.id == Java.awt.Event.ACTION_EVENT)
    {
        if (e.target == buttonOK)
        {
            buttonOKPressed();
        }
        else if (e.target == buttonCancel)
        (
            buttonCancelPressed();
        )
    }
    return super.handleEvent(e);
}
```

The problem with this method is that events cannot be delivered to specific objects. Instead, they have to be routed through a universal handler, which increases complexity and therefore, weakens your design.

But Java 1.1 onward has introduced the concepts of the *event delegation* model. This model allows special classes, known as "adapter classes" to be built and be registered with a component in order to handle certain events. Three simple steps are required to use this model:

1. Implement the desired listener interface in your adapter class. Depending on what event you're handling, a number of listener interfaces are available. These include: `ActionListener`, `WindowListener`, `MouseListener`, `MouseMotionListener`, `ComponentListener`, `FocusListener`, and `ListSelectionListener`.
2. Register the adapter listener with the desired component(s). This can be in the



form of an add XXX Listener () method supported by the component for example include add ActionListener (), add MouseListener (), and add FocusListener ().

3. Implement the listener interface's methods in your adapter class. It is in this code that you will actually handle the event.

The event delegation model allows the developer to separate the component's display (user interface) from the event handling (application data) which results in a cleaner and more object-oriented design.

**Components of an Event:** Can be put under the following categories.

1. **Event Object:** When the user interacts with the application by clicking a mouse button or pressing a key an event is generated. The Operating System traps this event and the data associated with it. For example, info about time at which the event occurred, the event types (like keypress or mouse click etc.). This data is then passed on to the application to which the event belongs.

You must note that, in Java, objects, which describe the events themselves, represent events. Java has a number of classes that describe and handle different categories of events.

2. **Event Source:** An event source is the object that generated the event, for example, if you click a button an ActionEvent Object is generated. The object of the ActionEvent class contains information about the event (button click).
3. **Event-Handler:** Is a method that understands the event and processes it. The event-handler method takes the Event object as a parameter. You can specify the objects that are to be notified when a specific event occurs. If the event is irrelevant, it is discarded.

The four main components based on this model are *Event classes*, *Event Listeners*, *Explicit event handling and Adapters*.

Let me give you a closer look at them one by one.

**Event Classes:** The EventObject class is at the top of the event class hierarchy. It belongs to the **Java.util** package. While most of the other event classes are present in Java.awt.event package.

The get Source () method of the EventObject class returns the object that initiated the event.

The getId () method returns the nature of the event. For example, if a mouse event occurs, you can find out whether the event was a click, a press, a move or release from the event object.

AWT provides two conceptual types of events: **Semantic and Low-level events**.

**Semantic event :** These are defined at a higher-level to encapsulate the semantics of user interface component's model.

Now let us see what the various semantic event classes are and when they are generated:

An **ActionEvent** object is generated when a component is activated.

An **Adjustment Event** Object is generated when scrollbars and other adjustment elements are used.

A **Text Event** object is generated when text of a component is modified.

An **Item Event** is generated when an item from a list, a choice or checkbox is selected.

**Low-Level Events:** These events are those that represent a low-level input or windows-system occurrence on a visual component on the screen.

The various low-level event classes and what they generate are as follows:

- A **Container Event** Object is generated when components are added or removed from container.
- A **Component Event** object is generated when a component is resized moved etc.
- A **Focus Event** object is generated when component receives focus for input.
- A **Key Event** object is generated when key on keyboard is pressed, released etc.
- A **Window Event** object is generated when a window activity, like maximizing or close occurs.
- A **Mouse Event** object is generated when a mouse is used.
- A **Paint Event** object is generated when component is painted.

**Event Listeners:** An object delegates the task of handling an event to an **event listener**. When an event occurs, an event object of the appropriate type (as explained below) is created. This object is passed to a **Listener**. The listener must **implement the interface** that has the method for event handling. A component can have multiple listeners, and a listener can be removed using **remove Action Listener ()** method. You can understand a listener as a person who has listened to your command and is doing the work which you commanded him to do so.

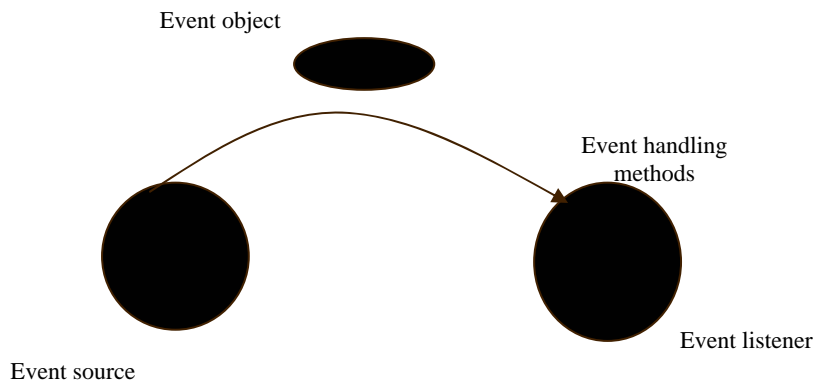
As you have studied about interfaces in Block 2 Unit 3 of this course, an **Interface** contains constant values and method declaration. The Java.awt.event package contains definitions of all event classes and listener interface. The **semantic listener interfaces** defined by AWT for the above-mentioned semantic events are:

- ActionListener
- AdjustmentListener
- ItemListener
- TextListener

The **low-level event listeners** are as follows:

- ComponentListener
- ContainerListener
- FocusListener
- KeyListener
- MouseListener
- MouseMotionListener
- WindowListener.

**Action Event using the ActionListener interface:** In *Figure 2* the Event Handling Model of AWT is given. This figure illustrates the usage of `ActionEvent` and `ActionListener` interface in a Classic Java Application.



**Figure 2: Event Handling Model of AWT**

You can see in the program given below that the Button Listener is handling the event generated by pressing button “Click Me”. Each time the button “Click Me” is pressed; the message “The Button is pressed” is printed in the output area. As shown in the output of this program, the message. “The Button is pressed” is printed three times, since “Click Me” is pressed thrice.

```

import Java.awt.event.*;
import Java.awt.*;
public class MyEvent extends Frame
{
    public MyEvent()
    {
        super("Window Title: Event Handling");
        Button b1;
        b1 = new Button("Click Me");
        //getContentPane().add(b1);
        add(b1);
        Button Listener listen = new Button Listener();
        b1.add Action Listener(listen);
        set Visible (true);
        set Size (200,200);
    }
    public static void main (String args[])
    {
        My Event event = new My Event();
    }
}; // note the semicolon
class Button Listener implements ActionListener
{
    public void action Performed (ActionEvent evt)
    {
        Button source1 = (Button) evt. get Source ();
        System. out. print ln ("The Button is Pressed");
    }
}

```

Output of the program:



The Button is Pressed  
The Button is Pressed  
The Button is Pressed

How does the above Application work? The answer to this question is given below in steps.

1. The execution begins with the main method.
2. An Object of the MyEvent class is created in the main method, by invoking the constructor of the MyEvent class.
3. Super () method calls the constructor of the base class and sets the title of the window as given, "Windows Title: Event Handling".
4. A button object is created and placed at the center of the window.
5. Button object is added in the event.
6. A Listener Object is created.
7. The addAction Listener () method registers the listener object for the button.
8. SetVisible () method displays the window.
9. The Application waits for the user to interact with it.
10. Then the user clicks on the button labeled "Click Me": The "ActionEvent" event is generated. Then the(ActionEvent) object is created and delegated to the registered listener object for processing. The Listener object contains the actionPerformed () method which processes the(ActionEvent) In the actionPerformed () method, the reference to the event source is retrieved using getSource () method. The message "The Button is Pressed" is printed.

---

## 1.6 HTML APPLET TAG

---

Till now you have written some Applets and have run them in the browser or Appletviewer. You have used basic tags needed for running an Applet program. Now you will learn some more tag that contains various attributes.

Applets are embedded in web pages using the <APPLET> and </APPLET> tags. APPLET elements accept The. *class* file with the CODE attribute. The CODE attribute tells the browser where to look for the compiled .class file. It is relative to the location of the source document.

If the Applet resides somewhere other than the same directory where the page lives on, you don't just give a URL to its location. Rather, you have to point at the CODEBASE.

The CODEBASE attribute is a URL that points at the directory where the *.class* file is. The CODE attribute is the name of the *.class* file itself. For instance if on the HTML page in the previous section had you written

```
<APPLET CODE="HelloWorldApplet" CODEBASE="classes"
WIDTH="200" HEIGHT="200">
</APPLET>
```

then the browser would have tried to find HelloWorldApplet.class in the classes subdirectory inside in directory where the HTML page that included the Applet is contained. On the other hand if you had written

```
<APPLET CODE="HelloWorldApplet"
CODEBASE="http://www.mysite.com/classes" WIDTH="200" HEIGHT="200">
</APPLET>
```

then the browser would try to retrieve the Applet from <http://www.mysite.com/classes/HelloWorldApplet.class> regardless of where the HTML page is residing.

If the Applet is in a non-default package, then the full package qualified name must be used. For example,

```
<APPLET CODE="mypackage. HelloWorldApplet"
CODEBASE="c:\Folder\Java\" WIDTH="200" HEIGHT="200">
</APPLET>
```

The HEIGHT and WIDTH attributes work exactly as they do with IMG, specifying how big a rectangle the browser should set aside for the Applet. These numbers are specified in pixels.

## Spacing Preferences

The <APPLET> tag has several attributes to define how it is positioned on the page. The ALIGN attribute defines how the Applet's rectangle is placed on the page relative to other elements. Possible values include LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM and ABSBOTTOM. This attribute is optional.

You can specify an HSPACE and a VSPACE in pixels to set the amount of blank space between an Applet and the surrounding text. The HSPACE and VSPACE attributes are optional.

```
<Applet code="HelloWorldApplet" width=200 height=200
ALIGN=RIGHT HSPACE=5 VSPACE=10>
</APPLET>
```

The ALIGN, HSPACE, and VSPACE attributes are identical to the attributes used by the <IMG> tag.

## Alternate Text

The <APPLET> has an ALT attribute. An ALT attribute is used by a browser that understands the APPLETTAG tag but for some reason cannot play the Applet. For instance, if you've turned off Java in Netscape Navigator 3.0, then the browser should display the ALT text. The ALT tag is optional.

```
<Applet code="HelloWorldApplet"
CODEBASE="c:\Folder\classes" width="200" height="200"
ALIGN="RIGHT" HSPACE="5" VSPACE="10"
ALT="Hello World!">
</APPLET>
```

ALT is not used by browsers that do not understand <APPLET> at all. For that purpose <APPLET> has been defined to require a closing tag, </APPLET>.

All raw text between the opening and closing <APPLET> tags is ignored by a Java capable browser. However, a non-Java-capable browser will ignore the <APPLET> tags instead and read the text between them. For example, the following HTML fragment says Hello World!, both with and without Java-capable browsers.

```
<Applet code="HelloWorldApplet" width=200 height=200  
ALIGN=RIGHT HSPACE=5 VSPACE=10  
ALT="Hello World!">  
Hello World!<P>  
</APPLET>
```

## **Naming Applets**

You can give an Applet a name by using the NAME attribute of the APPLET tag. This allows communication between different Applets on the same Web page.

```
<APPLET CODE="HelloWorldApplet" NAME="Applet1"  
CODEBASE="c:\Folder\Classes" WIDTH="200" HEIGHT="200"  
align="right" HSPACE="5" VSPACE="10"  
ALT="Hello World!">  
Hello World!<P>  
</APPLET>
```

## **JAR Archives**

HTTP 1.0 uses a separate connection for each request. When you're downloading many small files, the time required to set up and tear down the connections can be a significant fraction of the total amount of time needed to load a page. It would be better if you could load all the HTML documents, images, Applets, and sounds in a page in one connection.

One way to do this without changing the HTTP protocol is to pack all those different files into a single archive file, perhaps a zip archive, and just download that.

We aren't quite there yet. Browsers do not yet understand archive files, but in Java 1.1 Applets do. You can pack all the images, sounds; and.class files that an Applet needs into one JAR archive and load that instead of the individual files. Applet classes do not have to be loaded directly. They can also be stored in JAR archives.

To do this you use the ARCHIVES attribute of the APPLET tag.

```
<APPLET CODE="HelloWorldApplet" WIDTH="200" HEIGHT="100"  
ARCHIVES="HelloWorld.jar">  
<hr>  
Hello World!  
<hr>  
</APPLET>
```

In this example, the Applet class is still HelloWorldApplet. However, there is no HelloWorldApplet.class file to be downloaded. Instead the class is stored inside the archive file HelloWorld.jar.

Sun provides a tool for creating JAR archives with its JDK 1.1 onwards.

## **For Example**

```
% jar cf HelloWorld.jar *.class
```

This puts all the .class files in the current directory in a file named "HelloWorld.jar". The syntax of the jar command is similar to the Unix tar command.

## The Object Tag

HTML 4.0 deprecates the <APPLET> tag. Instead you are supposed to use the <OBJECT> tag. For the purposes of embedding Applets, the <OBJECT> tag is used almost exactly like the <APPLET> tag except that the class attribute becomes the classid attribute. For example,

```
<OBJECT classid="MyApplet" CODEBASE="c:\Folder\Classes" width=200
height=200 ALIGN=RIGHT HSPACE=5 VSPACE=10>
</OBJECT>
```

The <OBJECT> tag is also used to embed ActiveX controls and other kinds of active content. It has a few additional attributes to allow it to do that. However, for the purposes of Java you don't need to know about these.

The <OBJECT> tag is supported by Netscape and Internet Explorer. It is not supported by earlier versions of these browsers. <APPLET> is unlikely to disappear anytime soon in the further.

You can support both by placing an <APPLET> element inside an <OBJECT> element like this:

```
<OBJECT classid="MyApplet" width="200" height="200">
<APPLET code="MyApplet" width="200" height="200">
</APPLET>
</OBJECT>
```

You will notice that browsers that understand <OBJECT> will ignore its content while the browsers will display its content.

PARAM elements are the same for <OBJECT> as for <APPLET>.

## Passing Parameters to Applets

Parameters are passed to Applets in NAME and VALUE attribute pairs in <PARAM> tags between the opening and closing APPLET tags. Inside the Applet, you read the values passed through the PARAM tags with the getParameter() method of the Java.Applet.Applet class.

The program below demonstrates this with a generic string drawing Applet. The Applet parameter "Message" is the string to be drawn.

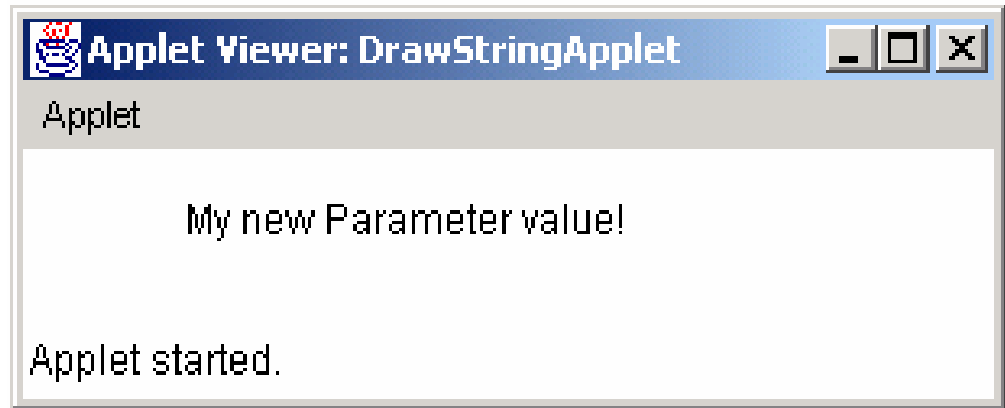
```
import Java.Applet.*;
import Java.awt.*;
public class DrawStringApplet extends Applet {
    private String str = "Hello!";

    public void paint(Graphics g) {
        String str = this.getParameter("Message");
        g.drawString(str, 50, 25);
    }
}
```

You also need an HTML file that references your Applet. The following simple HTML file will do:

```
<HTML>
<HEAD>
<TITLE> Draw String </TITLE>
</HEAD>
<BODY>
This is the Applet:<P>
<APPLET code="Draw String Applet" width="300" height="50">
<PARAM name="Message" value="My new Parameter value!">

</APPLET>
</BODY>
</HTML>
```



Of course you are free to change “My new Parameter value!” to a “message” of your choice. You only need to change the HTML, not the Java source code. PARAMs let you customize Applets without changing or recompiling the code.

This Applet is very similar to the HelloWorldApplet. However rather than hardcoding the message to be printed it’s read into the variable str through get parameter () method from a PARAM in the HTML.

You pass get Parameter() a string that names the parameter you want. This string should match the name of a <PARAM> tag in the HTML page. getParameter() returns the value of the parameter. All values are passed as strings. If you want to get another type like an integer, then you’ll need to pass it as a string and convert it to the type you really want.

The <PARAM> HTML tag is also straightforward. It occurs between <APPLET> and </APPLET>. It has two attributes of its own, NAME and VALUE. NAME identifies which PARAM this is. VALUE is the value of the PARAM as a String. Both should be enclosed in double quote marks if they contain white space.

An Applet is not limited to one PARAM. You can pass as many named PARAMs to an Applet as you like. An Applet does not necessarily need to use all the PARAMs that are in the HTML. You can be safely ignore additional PARAMs.

### Processing An Unknown Number Of Parameters

Sometimes the parameters are not known to you, in that case most of the time you have a fairly good idea of what parameters will and won’t be passed to your Applet or perhaps you want to write an Applet that displays several lines of text. While it would be possible to cram all this information into one long string, that’s not too friendly to authors who want to use your Applet on their pages. It’s much more sensible to give each line its own <PARAM> tag. If this is the case, you should name the tags via some predictable and numeric scheme. For instance in the text example the following set of <PARAM> tags would be sensible:

```
<PARAM name="param1" value="Hello Good Morning">
```





These three methods are `init`, `start` and `paint`, and they are guaranteed to be called in that order. These methods are called from Appletviewer or browser in which the Applet is executing. The `<Applet>` tag first attribute or indicates the file containing the compiled Applet class. It specifies height and width of the Applet tag to process an event you must register an event listener and implement one or more event handlers. The use of event listeners in event handling is known as Event Delegation Model.

You had seen various events and event listeners associated with each component. The information about a GUI event is stored in an object of a class that extends AWT Event.

---

## **1.8 SOLUTIONS/ANSWERS**

---

### **Check Your Progress 1**

- 1) An Applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application. The Applet class provides a standard interface between Applets and their environment. It is a secure program that runs inside a web browser. Applet can be embedded in an HTML page. Applets differ from Java applications in the way that they are not allowed to access certain resources on the local computer, such as files and serial devices (modems, printers, etc.), and are prohibited from communicating with most other computers across a network. The common rule is that an Applet can only make an Internet connection to the computer from which the Applet was sent.
- 2) You can surf the web without worrying that a Java Applet will format your hard disk or introduce a virus into your system. In fact you must have noticed that Java Applets and applications are much safer in practice than code written in traditional languages. This is because even code from trusted sources is likely to have bugs. However Java programs are much less susceptible to common bugs involving memory access than are programs written in traditional languages like C. Furthermore, the *Java runtime environment* provides a fairly robust means of trapping bugs before they bring down your system. Most users have many more problems with bugs than they do with deliberately malicious code. Although users of Java applications aren't protected from out and out malicious code, they are largely protected from programmer errors. Applets implement additional security restrictions that protect users from malicious code too. This is accomplished through the `Java.lang.SecurityManager` class. This class is subclassed to provide different security environments in different virtual machines. Regrettably implementing this additional level of protection does somewhat restrict the actions an Applet can perform.

Applets are not allowed to write data on any of the host's disks or read any data from the host's disks without the user's permission. In some environments, notably Netscape, an Applet cannot read data from the user's disks even with permission.

- 3) An Applet is a Java program that runs in the Appletviewer or a World Wide Web browser such as Netscape Communicator or Microsoft Internet Explorer. The Applet viewer (or browser) executes an Applet when a Hypertext Markup Language (HTML) document containing the Applet is opened in the Applet viewer (or web browser).
- 4) Object Oriented Programming (OOP) models real world objects with software counterparts. It takes advantage of class relationships where objects of a certain class have the same characteristics. It takes advantage of inheritance relationships where newly created classes are derived by inheriting characteristics of existing classes; yet contain unique characteristics of their

own. Object Oriented concepts implemented through Applet programming can communicate one place to other through web pages.

## Check Your Progress 2

- 1) A Java program is first compiled to bytecode which is stored in a '.class file'. This file is downloaded as an Applet to a browser, which then interprets it by converting into machine code appropriate to the hardware platform on which the Applet program is running.
- 2) In the Appletviewer, you can execute an Applet again by clicking the Applet viewer's Applet menu and selecting the **Reload** option from the menu. To terminate an Applet, click the Appletviewer's Applet menu and select the **Quit** option.
- 3) When the Applet is executed from the Appletviewer, The Appletviewer only understands the <Applet> and </Applet> HTML tags, so it is sometimes referred to as the "minimal browser".(It ignores all other HTML tags). The starting sequence of method calls made by the Appletviewer or browser for every Applet is always init (), start () and paint () – this provides a start-up sequence of method calls as every Applet begins execution.

The stop () method would pause the Applet, However, destroy () will terminate the application.

## Check Your Progress 3

- 1) There is no way that an Applet can access network resources. You have to implement a Java Query Sever that will be running on the same machine from where you are receiving your Applet (that is Internet Server). Your Applet will communicate with that server which fulfill all the requirement of the Applet. You communicate between Applet and Java query server using sockets Remote Method Invocation (RMI), this may be given preference because it will return the whole object.

2)

```
import Java.Applet.*;
import Java.awt.*;
import Java.awt.event.*;
public class TestMouse1
{
public static void main (String[] args)
{
    Frame f = new Frame("TestMouseListener");
    f.setSize(500,500);
    f.setVisible(true);
    f.addMouseListener(new MouseAdapter()
    {
        public void mouseClicked(MouseEvent e)
        {
            System.out.println("Mouse clicked: (" +e.getX()+" "+e.getY()+"");
        }
    });
}
}
```

3)

DrawStringApplet1.Java file:

```
import Java.Applet.*;
import Java.awt.*;
```

```
public class DrawStringApplet1 extends Applet
{
    public void paint(Graphics g)
    {
        String str1 = this.getParameter("Message1");
        g.drawString(str1, 50, 25);
        String str2 = this.getParameter("Message2");
        g.drawString(str2, 50, 50);
    }
}
```

DrawStringApplet1.html file:

```
<HTML>
<HEAD>
<TITLE> Draw String </TITLE>
</HEAD>
<BODY>
<APPLET code="DrawStringApplet1" width="300" height="250">
<PARAM name="Message1" value="M. P. Mishra">
<PARAM name="Message2" value="SOCIS, IGNOU, New Delhi-68">
</APPLET>
</BODY>
</HTML>
```

Compile DrawStringApplet1.Java file then run DrawStringApplet1.html file either in web browser or through Appletviewer.

---

## UNIT 2 GRAPHICS AND USER INTERFACES

---

Structure	Page Nos.
2.0 Introduction	25
2.1 Objectives	25
2.2 Graphics Contexts and Graphics Objects	26
2.2.1 Color Control	
2.2.2 Fonts	
2.2.3 Coordinate System	
2.2.3.1 Drawing Lines	
2.2.3.2 Drawing Rectangle	
2.2.3.3 Drawing Ovals and Circles	
2.2.3.4 Drawing Polygons	
2.3 User Interface Components	32
2.4 Building User Interface with AWT	33
2.5 Swing-based GUI	38
2.6 Layouts and Layout Manager	39
2.7 Container	45
2.8 Summary	46
2.9 Solutions/Answer	47

---

### 2.0 INTRODUCTION

---

We all wonder that on seeing 2D or 3D movies or graphics, even on the Internet, we see and admire good animation on various sites. The java technology provides you the platform to develop these graphics using the Graphics class. In this unit you have to overview several java capabilities for drawing two-dimensional shapes, colors and fonts. You will learn to make your own interface, which will be user-interactive and friendly. These interfaces can be made either using java AWT components or java SWING components. In this unit you will learn to use some basic components like button, text field, text area, etc.

Interfaces using GUI allow the user to spend less time trying to remember which keystroke sequences do what and allow spend more time using the program in a productive manner. It is very important to learn how you can beautify your components placed on the canvas area using FONT and COLOR class. For placing components on different layouts knowing use of various Layout managers is essential.

---

### 2.1 OBJECTIVES

---

Our objective is to give you the concept of making the look and feel attractive. After going through this unit, you will be able to:

- explain the principles of graphical user interfaces;
- differentiate between AWT and SWING components;
- design a user friendly Interface;
- applying color to interfaces;
- using Font class in programs;
- use the Layout Manager, and
- use Container Classes.

---

## 2.2 GRAPHICS CONTEXTS AND GRAPHICS OBJECTS

---

How can you build your own animation using Graphics Class? A Java graphics context enables drawing on the screen. A Graphics object manages a graphics context by controlling how objects are drawn. Graphics objects contain methods for drawing, font manipulation, color manipulation and the other related operations. It has been developed using the Graphics object *g* (the argument to the applet's paint method) to manage the applet's graphics context. In other words you can say that Java's Graphics is capable of:

- Drawing 2D shapes
- Controlling colors
- Controlling fonts
- Providing Java 2D API
- Using More sophisticated graphics capabilities
- Drawing custom 2D shapes
- Filling shapes with colors and patterns.

Before we begin drawing with Graphics, you must understand Java's coordinate system, which is a scheme for identifying every possible point on the screen. Let us start with Graphics Context and Graphics Class.

### Graphics Context and Graphics Class

- Enables drawing on screen
- Graphics object manages graphics context
- Controls how objects is drawn
- Class Graphics is abstract
- Cannot be instantiated
- Contributes to Java's portability
- Class Component method paint takes Graphics object.

The *Graphics* class is the abstract base class for all graphics contexts. It allows an application to draw onto components that are realized on various devices, as well as on to off-screen images.

### Public Abstract Class Graphics Extends Object

You have seen that every applet performs drawing on the screen.

### Graphics Objects

In Java all drawing takes place via a Graphics object. This is an instance of the class *java.awt.Graphics*.

Initially the Graphics object you use will be passed as an argument to an applet's paint() method. The drawing can be done Applet Panels, Frames, Buttons, Canvases etc.

Each Graphics object has its own coordinate system, and methods for drawing strings, lines, rectangles, circles, polygons etc. Drawing in Java starts with particular Graphics object. You get access to the Graphics object through the *paint(Graphics g)* method of your applet.

Each draw method call will look like

```
g.drawString("Hello World", 0, 50);
```

Where *g* is the particular Graphics object with which you're drawing. For convenience sake in this unit the variable *g* will always refer to a pre-existing object of the Graphics class. It is not a rule you are free to use some other name for the particular Graphics context, such as *myGraphics* or *applet-Graphics* or anything else.

## 2.2.1 Color Control

It is known that Color enhances the appearance of a program and helps in conveying meanings. To provide color to your objects use class *Color*, which defines methods and constants for manipulation colors. Colors are created from **red**, **green** and **blue** components **RGB** values.

All three RGB components can be integers in the range 0 to 255, or floating point values in the range 0.0 to 1.0

The first part defines the amount of *red*, the second defines the amount of *green* and the third defines the amount of *blue*. So, if you want to give dark red color to your graphics you will have to give the first parameter value 255 and two parameter zero.

Some of the most common colors are available by name and their RGB values in *Table 1*.

**Table 1: Colors and their RGB values**

Color Constant	Color	RGB Values
Public final static Color ORANGE	Orange	255, 200, 0
Public final static Color PINK	Pink	255, 175, 175
Public final static Color CYAN	Cyan	0, 255, 255
Public final static Color MAGENTA	Magenta	255, 0, 255
Public final static Color YELLOW	Yellow	255, 255, 0
Public final static Color BLACK	Black	0, 0, 0
Public final static Color WHITE	White	255, 255, 255
Public final static Color GRAY	Gray	128, 128, 128
Public final static Color LIGHT_GRAY	light gray	192, 192, 192
Public final static Color DARK_GRAY	dark gray	64, 64, 64
Public final static Color RED	Red	255, 0, 0
Public final static Color GREEN	Green	0, 255, 0
Public final static Color BLUE	Blue	0, 0, 255

## Color Methods

To apply color in your graphical picture (objects) or text two Color methods *get Color* and *set Color* are provided. Method *get Color* returns a Color object representing the current drawing color and method *set Color* used to sets the current drawing color.

## Color constructors

`public Color(int r, int g, int b)`: Creates a color based on the values of red, green and blue components expressed as integers from 0 to 255.

`public Color (float r, float g, float b )` : Creates a color based the values of on red, green and blue components expressed as floating-point values from 0.0 to 1.0.

## Color methods:

`public int getRed()`: Returns a value between 0 and 255 representing the red content

`public int getGreen()`: Returns a value between 0 and 255 representing the green content

`public int getBlue()` . Returns a value between 0 and 255 representing the blue content.

## Graphics Methods For Manipulating Colors

`public Color getColor ()`: Returns a Color object representing the current color for the graphics context.

`public void setColor (Color c )`: Sets the current color for drawing with the graphics context.

As you do with any variable you should preferably give your colors descriptive names. For instance

`Color medGray = new Color(127, 127, 127);`

`Color cream = new Color(255, 231, 187);`

`Color lightGreen = new Color(0, 55, 0);`

You should note that Color is not a property of a particular rectangle, string or other object you may draw, rather color is a part of the Graphics object that does the drawing. You change the color of your Graphics object and everything you draw from that point forward will be in the new color, at least until you change it again.

When an applet starts running , its color is set to *black by default*. You can change this to red by calling `g.setColor(Color.red)`. You can change it back to black by calling `g.setColor(Color.black)`.

## Check Your Progress 1

- 1) What are the various color constructors?

.....  
.....

- 2) Write a program to set the Color of a String to red.

.....  
.....

- 3) What is the method to retrieve the color of the text? Write a program to retrieve R G B values in a given color.

.....  
.....

Now we will discuss about how different types of fonts can be provided to your strings.

## 2.2.2 Fonts

You must have noticed that until now all the applets have used the default font. However unlike HTML Java allows you to choose your fonts. Java implementations



are guaranteed to have a *serif font* like *Times* that can be accessed with the name "*Serif*", a monospaced font like *courier* that can be accessed with the name "*Mono*", and a *sans serif* font like *Helvetica* that can be accessed with the name "*SansSerif*".

How can you know the available fonts on your system for an applet program? You can list the fonts available on the system by using the `getFontList()` method from `java.awt.Toolkit`. This method returns an array of strings containing the names of the available fonts. These may or may not be the same as the fonts to installed on your system. It is implementation is dependent on whether or not all the fonts in a system are available to the applet.

Choosing a font face is very easy. You just create a new `Font` object and then call `setFont(Font f)`.

To instantiate a `Font` object the constructor

```
public Font(String name, int style, int size)
```

can be used. **name** is the name of the font family, e.g. "*Serif*", "*SansSerif*", or "*Mono*".

**size** is the size of the font in points. In computer graphics a point is considered to be equal to one pixel. 12 points is a normal size font.

**style** is an mnemonic constant from `java.awt.Font` that tells whether the text will be bold, italics or plain. The three constants are `Font.PLAIN`, `Font.BOLD`, and `Font.ITALIC`.

In other words, The Class `Font` contains methods and constants for font control. `Font` constructor takes three arguments

Font name	Monospaced, SansSerif, Serif, etc.
Font style	<code>Font.PLAIN</code> , <code>Font.ITALIC</code> and <code>Font.BOLD</code>
Font size	Measured in points (1/72 of inch)

## Graphics method for Manipulating Fonts

`public Font get Font()`: Returns a `Font` object reference representing the current `Font`.

`public void setFont(Font f)`: Sets the current font, style and size specified by the `Font` object reference `f`.

### 2.2.2.1 FontMetrics

Sometimes you will need to know how much space a particular string will occupy. You can find this out with a `FontMetrics` object.

`FontMetrics` allow you to determine the height, width or other useful characteristics of a particular string, character, or array of characters in a particular font. In *Figure 1* you can see a string with some of its basic characteristics.



Figure 1: String Characteristics

In order to tell where and whether to wrap a String, you need to measure the string, not its length in characters, which can be variable in its width, and height in pixels. Measurements of this sort on strings clearly depend on the font that is used to draw the string. All other things being equal a 14-point string will be wider than the same string in 12 or 10-point type.

To measure character and string sizes you need to look at the FontMetrics of the current font. To get a FontMetrics object for the current Graphics object you use the `java.awt.Graphics.getFontMetrics()` method.

`java.awt.FontMetrics` provide method `stringWidth(String s)` to return the width of a string in a particular font, and method `getLeading()` to get the appropriate line spacing for the font. There are many more methods in `java.awt.FontMetrics` that let you measure the height and width of specific characters as well as ascenders, descenders and more, but these three methods will be sufficient for basic programs.

## Check Your Progress 2

- 1) Write a program to set the font of your String as font name as “Arial”, font size as 12 and font style as `FONT.ITALIC`.

.....  
.....

- 2) What is the method to retrieve the font of the text? Write a program for font retrieval.

.....  
.....

- 3) Write a program that will give you the Fontmetrics parameters of a String.

.....  
.....

Knowledge of co-ordinate system is essential to play with positioning of objects in any drawing. Now you will see how coordinates are used in java drawings.

## 2.2.3 Coordinate System

By Default the upper left corner of a GUI component (such as applet or window) has the coordinates (0,0). A Coordinate pair is composed of x-coordinate (the horizontal coordinate) and a y-coordinate (the vertical coordinate). The x-coordinate is the horizontal distance moving right from the upper left corner.

The y-coordinate is the vertical distance moving down from the upper left corner. The x-axis describes every horizontal coordinate, and the y-axis describes every vertical coordinate. You must note that different display cards have different resolutions (i.e. the density of pixels varies). *Figure 2* represents coordinate system. This may cause graphics to appear to be different sizes on different monitors.

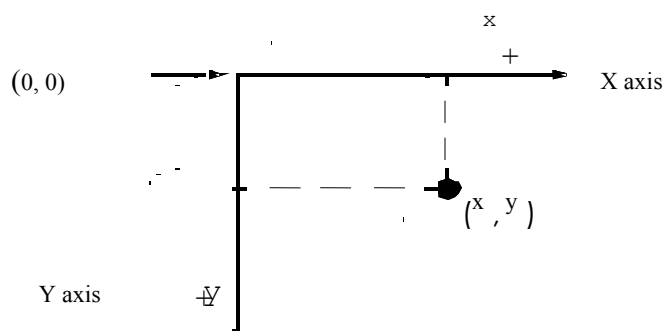


Figure 2: Co-ordinate System

Now we will move towards drawing of different objects. In this section, I will demonstrate drawing in applications.

### 2.2.3.1 Drawing Lines

Drawing straight lines with Java can be done as follows:

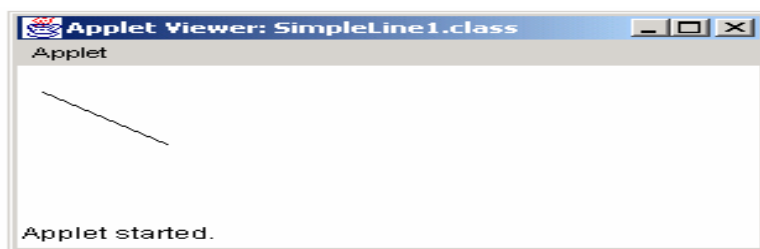
call

```
g.drawLine(x1,y1,x2,y2)
```

method, where (x1, y1) and (x2, y2) are the endpoints of your lines and g is the Graphics object you are drawing with. The following program will result in a line on the applet.

```
import java.applet.*;
import java.awt.*;
public class SimpleLine extends Applet
{
    public void paint(Graphics g)
    {
        g.drawLine(10, 20, 30, 40);
    }
}
```

Output:



### 2.2.3.2 Drawing Rectangle

Drawing rectangles is simple. Start with a Graphics object g and call its drawRect() method:

```
public void drawRect(int x, int y, int width, int height)
```

The first argument int is the left hand side of the rectangle, the second is the top of the rectangle, the third is the width and the fourth is the height. This is in contrast to some APIs where the four sides of the rectangle are given.

Remember that the upper left hand corner of the applet starts at (0, 0), not at (1, 1). This means that a 100 by 200 pixel applet includes the points with x coordinates between 0 and 99, not between 0 and 100. Similarly the y coordinates are between 0 and 199 inclusive, not 0 and 200.

### 2.2.3.3 Drawing Ovals and Circles

Java has methods to draw outlined and filled ovals. These methods are called drawOval() and fillOval() respectively. These two methods are:

```
public void drawOval(int left, int top, int width, int height)
public void fillOval(int left, int top, int width, int height)
```

Instead of dimensions of the oval itself, the dimensions of the smallest rectangle, which can enclose the oval, are specified. The oval is drawn as large as it can be to touch the rectangle's edges at their centers. *Figure 3* may help you to understand properly.

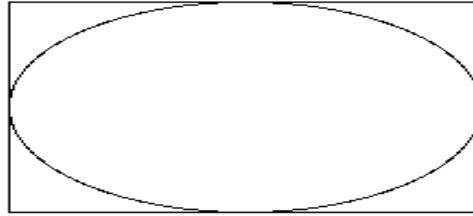


Figure 3: An Oval

The arguments to `drawOval()` are the same as the arguments to `drawRect()`. The first `int` is the left hand side of the enclosing rectangle, the second is the top of the enclosing rectangle, the third is the width and the fourth is the height. There is no special method to draw a circle. Just draw an oval inside a square.

#### 2.2.3.4 Drawing Polygons and Polylines

You have already seen that in Java rectangles are defined by the position of their upper left hand corner, their height, and their width. However it is implicitly assumed that there is in fact an upper left hand corner. What's been assumed so far is that the sides of the rectangle are parallel to the coordinate axes. You can't yet handle a rectangle that has been rotated at an arbitrary angle.

There are some other things you can't handle either, triangles, stars, rhombuses, kites, octagons and more. To take care of this broad class of shapes Java has a **Polygon** class.

*Polygons* are defined by their corners. No assumptions are made about them except that they lie in a 2-D plane. The basic constructor for the Polygon class is

```
public Polygon(int[] xpoints, int[] ypoints, int npoints)
```

**xpoints** is an array that contains the x coordinates of the polygon. **ypoints** is an array that contains the y coordinates. Both should have the length **npoints**. Thus to construct a right triangle with the right angle on the origin you would type

```
int[] xpoints = {0, 3, 0};  
int[] ypoints = {0, 0, 4};  
Polygon myTriangle = new Polygon(xpoints, ypoints, 3);
```

To draw the polygon you can use `java.awt.Graphics`'s `drawPolygon(Polygon p)` method within your `paint()` method like this:

```
g.drawPolygon(myTriangle);
```

You can pass the arrays and number of points directly to the `drawPolygon()` method if you prefer:

```
g.drawPolygon(xpoints, ypoints, xpoints.length);
```

There's also an overloaded `fillPolygon()` method, you can call this method like

```
g.fillPolygon(myTriangle);  
g.fillPolygon(xpoints, ypoints, xpoints.length());
```

To simplify user interaction and make data entry easier, Java provides different controls and interfaces. Now let us see some of the basic user interface components of Java.

---

## 2.3 USER INTERFACE COMPONENTS

---

Java provides many controls. Controls are components, such as buttons, labels and text boxes that can be added to containers like frames, panels and applets. The `Java.awt` package provides an integrated set of classes to manage user interface components.

Components are placed on the user interface by adding them to a container. A container itself is a component. The easiest way to demonstrate interface design is by using the container you have been working with, i.e., the Applet class. The simplest form of Java AWT component is the basic *User Interface Component*. You can create and add these to your applet without any need to know anything about creating containers or panels. In fact, your applet, even before you start painting and drawing and handling events, is an AWT container. Because an applet is a container, you can put any of AWT components, and (or) other containers, in it.

In the next section of this Unit, you will learn about the basic User Interface components (controls) like labels, buttons, check boxes, choice menus, and text fields. You can see in Table 2a, a list of all the Controls in Java AWT and their respective functions. In Table 2b list of classes for these control are given.

## 2.4 BUILDING USER INTERFACE WITH AWT

In order to add a control to a container, you need to perform the following two steps:

1. Create an object of the control by passing the required arguments to the constructor.
2. Add the component (control) to the container.

**Table 2a: Controls in Java**

CONTROLS	FUNCTIONS
Textbox	Accepts single line alphanumeric entry.
TextArea	Accepts multiple line alphanumeric entry.
Push button	Triggers a sequence of actions.
Label	Displays Text.
Check box	Accepts data that has a yes/no value. More than one checkbox can be selected.
Radio button	Similar to check box except that it allows the user to select a single option from a group.
Combo box	Displays a drop-down list for single item selection. It allows new value to be entered.
List box	Similar to combo box except that it allows a user to select single or multiple items. New values cannot be entered.

**Table 2b: Classes for Controls**

CONTROLS	CLASS
Textbox	TextField
TextArea	TextArea
Push button	Button
Check box	CheckBox
Radio button	CheckboxGroup with CheckBox
Combo box	Choice
List box	List

## The Button

Let us first start with one of the simplest of UI components: the button. Buttons are used to trigger events in a GUI environment (we have discussed Event Handling in detail in the previous Unit: Unit 1 Block 4 of this course). The Button class is used to create buttons. When you add components to the container, you don't specify a set of coordinates that indicate where the components are to be placed. A layout manager in effect for the container handles the arrangement of components. The default layout for a container is flow layout (for an applet also default layout will be flow layout). More about different layouts you will learn in later section of this unit. Now let us write a simple code to test our button class.

To create a button use, one of the following constructors:

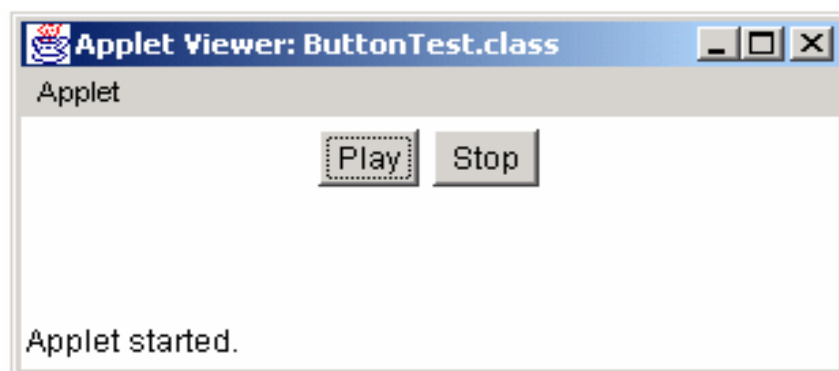
Button() creates a button with no text label.

Button(String) creates a button with the given string as label.

Example Program:

```
/*
<Applet code= "ButtonTest.class"
Width = 500
Height = 100>
</applet>
*/
import java.awt.*;
import java.applet.Applet;
public class ButtonTest extends Applet
{
    Button b1 = new Button ("Play");
    Button b2 = new Button ("Stop");
    public void init(){
        add(b1);
        add(b2);
    }
}
```

Output:



As you can see this program will place two buttons on the Applet with the caption Play and Stop.

## The Label

Labels are created using the Label class. Labels are basically used to identify the purpose of other components on a given interface; they cannot be edited directly by the user. Using a label is much easier than using a drawString( ) method because

labels are drawn automatically and don't have to be handled explicitly in the `paint()` method. Labels can be laid out according to the layout manager, instead of using `[x, y]` coordinates, as in `drawString()`.

To create a Label, use any one of the following constructors:

`Label()`: creates a label with its string aligned to the left.

`Label(String)`: creates a label initialized with the given string, and aligned left.

`Label(String, int)`: creates a label with specified text and alignment indicated by any one of the three `int` arguments. `Label.Right`, `Label.Left` and `Label.Center`.

`getText()` method is used to indicate the current label's text `setText()` method to change the label's text. `setFont()` method is used to change the label's font.

## The Checkbox

Check Boxes are labeled or unlabeled boxes that can be either "Checked off" or "Empty". Typically, they are used to select or deselect an option in a program.

Sometimes Check are *nonexclusive*, which means that if you have six check boxes in a container, all the six can either be checked or unchecked at the same time. This component can be organized into Check Box Group, which is sometimes called radio buttons. Both kinds of check boxes are created using the `Checkbox` class. To create a nonexclusive check box you can use one of the following constructors:

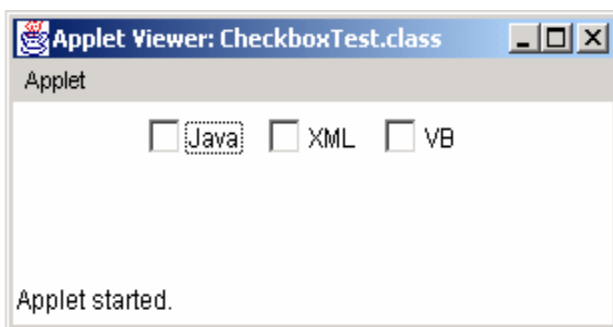
`Checkbox()` creates an unlabeled checkbox that is not checked.

`Checkbox(String)` creates an unchecked checkbox with the given label as its string.

After you create a checkbox object, you can use the `setState(boolean)` method with a true value as argument for checked checkboxes, and false to get unchecked. Three checkboxes are created in the example given below, which is an applet to enable you to select up to three courses at a time.

```
import java.awt.*;
public class CheckboxTest extends java.applet.Applet
{
    Checkbox c1 = new Checkbox ("Java");
    Checkbox c2 = new Checkbox ("XML");
    Checkbox c3 = new Checkbox ("VB");
    public void init(){
        add(c1);
        add(c2);
        add(c3);
    }
}
```

Output:



## The Checkbox group

`CheckboxGroup` is also called like a radio button or exclusive check boxes. To organize several Checkboxes into a group so that only one can be selected at a time,

you can create CheckboxGroup object as follows:

```
CheckboxGroup radio = new CheckboxGroup ();
```

The CheckboxGroup keeps track of all the check boxes in its group. We have to use this object as an extra argument to the *Checkbox* constructor.

Checkbox (String, CheckboxGroup, Boolean) creates a checkbox labeled with the given string that belongs to the CheckboxGroup indicated in the second argument. The last argument equals true if box is checked and false otherwise.

The set Current (checkbox) method can be used to make the set of currently selected check boxes in the group. There is also a get Current () method, which returns the currently selected checkbox.

## The Choice List

Choice List is created from the Choice class. List has components that enable a single item to be picked from a pull-down list. We encounter this control very often on the web when filling out forms.

The first step in creating a Choice

You can create a choice object to hold the list, as shown below:

```
Choice cgender = new Choice();
```

Items are added to the Choice List by using addItem(String) method the object. The following code adds two items to the gender choice list.

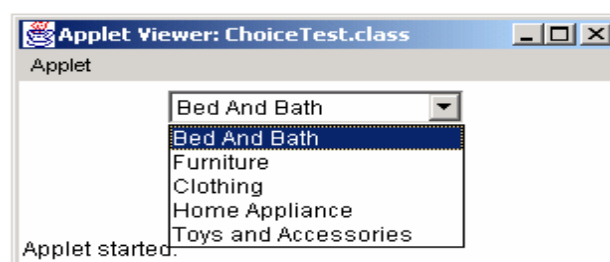
```
cgender.addItem("Female");  
cgender.addItem("Male");
```

After you add the Choice List it is added to the container like any other component using the add() method.

The following example shows an Applet that contains a list of shopping items in the store.

```
import java.awt.*;  
Public class ChoiceTest extends java.applet.Applet  
{  
    Choice shoplist = new Choice();  
  
    Public void init(){  
        shoplist.addItem("Bed And Bath");  
        shoplist.addItem("Furniture");  
        shoplist.addItem("Clothing");  
        shoplist.addItem("Home Appliance");  
        shoplist.addItem("Toys and Accessories");  
        add(shoplist);  
    }  
}
```

Output:





The choice list class has several methods, which are given in *Table 3*.

**Table 3: Choice List Class Methods**

Method	Action
getItem()	Returns the string item at the given position (items inside a choice begin at 0, just like arrays)
countItems()	Returns the number of items in the menu
getSelectedIndex()	Returns the index position of the item that's selected
getSelectedItem()	Returns the currently selected item as a string
select(int)	Selects the item at the given position
select(String)	Selects the item with the given string

## The Text Field

To accept textual data from user, AWT provided two classes, *TextField* and *TextArea*. The *TextField* handles a single line of text and does not have scrollbars, whereas the *TextArea* class handles multiple lines of text. Both the classes are derived from the *TextComponent* class. Hence they share many common methods. *TextFields* provide an area where you can enter and edit a single line of text. To create a text field, use one of the following constructors:

*TextField()*: creates an empty *TextField* with no specified width.

*TextField(int)*: creates an empty text field with enough width to display the specified number of characters (this has been depreciated in Java2).

*TextField(String)*: creates a text field initialized with the given string.

*TextField(String, int)*: creates a text field with specified text and specified width.

For example, the following line creates a text field 25 characters wide with the string "Brewing Java" as its initial contents:

```
TextField txtfld = new TextField ("Brewing Java", 25); add(txtfld);
```

*TextField*, can use methods like:

*setText()*: Used to set the text in text field.

*getText()*: Used to get the text currently contained by text field.

*setEditable()*: Used to provide control whether the content of text field may be modified by user or not.

*isEditable()*: It return **true** if the text in text filed may be changed and **false** otherwise.

## Text Area

The *TextArea* is an editable text field that can handle more than one line of input. Text areas have horizontal and vertical scrollbars to scroll through the text. Adding a text area to a container is similar to adding a text field. To create a text area you can use one of the following constructors:

*TextArea()*: creates an empty text area with unspecified width and height.

*TextArea(int, int)*: creates an empty text area with indicated number of lines and specified width in characters.

*TextArea(String)*: creates a text area initialized with the given string.

*TextField(String, int, int)*: creates a text area containing the indicated text and specified number of lines and width in the characters.

The *TextArea*, similar to *TextField*, can use methods like *setText()*, *getText()*, *setEditable()*, and *isEditable()*.

In addition, there are two more methods like these. The first is the *insertText(String, int)* method, used to insert indicated strings at the character index specified by the

second argument. The next one is `replaceText(String, int, int)` method, used to replace text between given integer position specified by second and third argument with the indicated string.

The basic idea behind the AWT is that a graphical Java program is a set of nested components, starting from the outermost window all the way down to the smallest UI component. Components can include things you can actually see on the screen, such as windows, menu bars, buttons, and text fields, and they can also include containers, which in turn can contain other components.

Hope you have got a clear picture of Java AWT and its some basic UI components, In the next section of the Unit we will deal with more advance user interface components.

---

## 2.5 SWING - BASED GUI

---

You must be thinking that when you can make GUI interface with AWT package then what is the purpose of learning Swing-based GUI? Actually Swing has *lightweight* components and does not write itself to the screen, but redirects it to the component it builds on. On the other hand AWT are heavyweight and have their own view port, which sends the output to the screen. Heavyweight components also have their own z-ordering (look and feel) dependent on the machine on which the program is running. This is the reason why you can't combine AWT and Swing in the same container. If you do, AWT will always be drawn on top of the Swing components.

Another difference is that Swing is pure Java, and therefore platform independent. Swing looks identically on all platforms, while AWT looks different on different platforms.

See, basically Swing provides a rich set of GUI components; features include model-UI separation and a plug able look and feel. Actually you can make your GUI also with AWT but with Swing you can make it more user-friendly and interactive. Swing components make programs efficient.

Swing GUI components are packaged into Package `javax.swing`. In the Java class hierarchy there is a class  
Class `Component` which contains method `paint` for drawing `Component` onscreen  
Class `Container` which is a collection of related components and contains method `add` for adding components and Class `JComponent` which has  
*Pluggable look and feel* for customizing look and feel  
Shortcut keys (*mnemonics*)  
Common event-handling capabilities

The Hierarchy is as follows:

Object-----> Component-->Container--->JComponent

In Swings we have classes prefixed with the letter 'J' like  
`JLabel` -> Displays single line of read only text

`JTextField` -> Displays or accepts input in a single line

`JTextArea` -> Displays or accepts input in multiple lines

`JCheckBox` -> Gives choices for multiple options

`JButton` -> Accepts command and does the action

JList -> Gives multiple choices and display for selection  
JRadioButton -> Gives choices for multiple option, but can select one at a time.

### Check Your Progress 3

- 1) Write a program which draws a line, a rectangle, and an oval on the applet.  
.....  
.....
- 2) Write a program that draws a color-filled line, a color-filled rectangle, and a color filled oval on the applet.  
.....  
.....
- 3) Write a program to add various checkboxes under the CheckboxGroup  
.....  
.....
- 4) Write a program in which the Applet displays a text area that is filled with a string, when the programs begin running.  
.....  
.....
- 5) Describe the features of the Swing components that subclass J Component.  
What are the difference between Swing and AWT?  
.....  
.....

Now we will discuss about different layouts to represent components in a container.

---

## 2.6 LAYOUTS AND LAYOUT MANAGER

---

When you add a component to an applet or a container, the container uses its *layout manager* to decide where to put the component. Different LayoutManager classes use different rules to place components.

java.awt.LayoutManager is an interface. Five classes in the java packages implement it:

- FlowLayout
- BorderLayout
- CardLayout
- GridLayout
- GridBagLayout
- plus javax.swing.BoxLayout

### FlowLayout

A FlowLayout arranges widgets from left to right until there's no more space left. Then it begins a row lower and moves from left to right again. Each component in a FlowLayout gets as much space as it needs and no more.  
This is the *default LayoutManager* for applets and panels. FlowLayout is the default layout for java.awt.Panel of which java.applet.Applet is a subclasses.  
Therefore you don't need to do anything special to create a FlowLayout in an applet. However you do need to use the following constructors if you want to use a FlowLayout in a Window. LayoutManagers have constructors like any other class.

The constructor for a FlowLayout is

```
public FlowLayout()
```

Thus to create a new FlowLayout object you write

```
FlowLayout fl;
```

```
fl = new FlowLayout();
```

As usual this can be shortened to

```
FlowLayout fl = new FlowLayout();
```

You tell an applet to use a particular LayoutManager instance by passing the object to the applet's setLayout() method like this: this.setLayout(fl);

Most of the time setLayout() is called in the init() method. You normally just create the LayoutManager right inside the call to setLayout() like this

```
this.setLayout(new FlowLayout());
```

For example the following applet uses a FlowLayout to position a series of buttons that mimic the buttons on a tape deck.

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

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

```
public class FlowTest extends Applet {
```

```
    public void init() {
```

```
        this.setLayout(new FlowLayout());
```

```
        this.add( new Button("Add"));
```

```
        this.add( new Button("Modify"));
```

```
        this.add( new Button("Delete"));
```

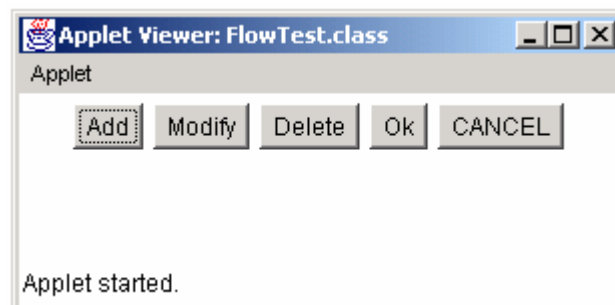
```
        this.add( new Button("Ok"));
```

```
        this.add( new Button("CANCEL"));
```

```
    }
```

```
}
```

Output:



You can change the alignment of a FlowLayout in the constructor. Components are normally centered in an applet. You can make them left or right justified. To do this just passes one of the defined constants FlowLayout.LEFT, FlowLayout.RIGHT or FlowLayout.CENTER to the constructor, e.g.

```
this.setLayout(new FlowLayout(FlowLayout.LEFT));
```

Another constructor allows you spacing option in FlowLayout:

```
public FlowLayout(int alignment, int horizontalSpace, int verticalSpace);
```

For instance to set up a FlowLayout with a ten pixel horizontal gap and a twenty pixel vertical gap, aligned with the left edge of the panel, you would use the constructor

```
FlowLayout fl = new FlowLayout(FlowLayout.LEFT, 20, 10);
```

Buttons arranged according to a center-aligned FlowLayout with a 20 pixel horizontal spacing and a 10 pixel vertical spacing

## BorderLayout

A BorderLayout organizes an applet into North, South, East, West and Center sections. North, South, East and West are the rectangular edges of the applet. They're continually resized to fit the sizes of the widgets included in them. Center is whatever is left over in the middle.

A BorderLayout places objects in the North, South, East, West and center of an applet. You create a new BorderLayout object much like a FlowLayout object, in the init() method call to setLayout like this:

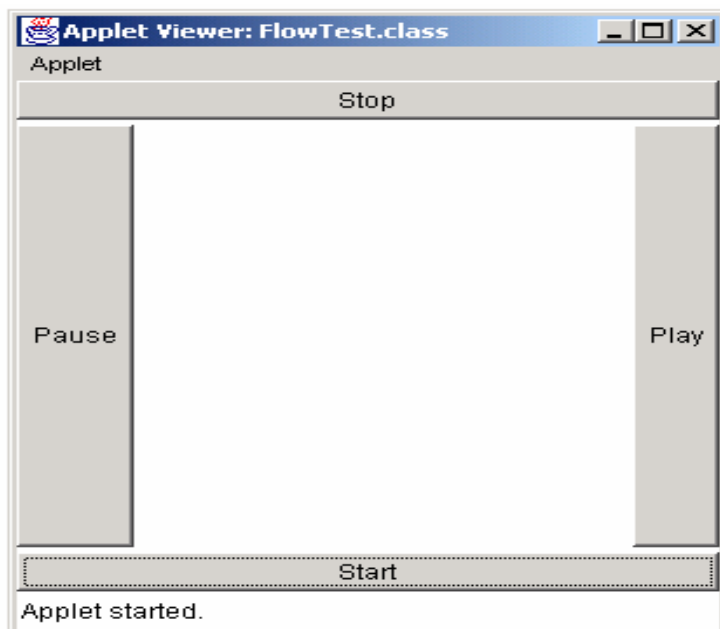
```
this.setLayout(new BorderLayout());
```

There's no centering, left alignment, or right alignment in a BorderLayout. However, you can add horizontal and vertical gaps between the areas. Here is how you would add a two pixel horizontal gap and a three pixel vertical gap to a BorderLayout:

```
this.setLayout(new BorderLayout(2, 3));
```

To add components to a BorderLayout include the name of the section you wish to add them to do like done in the program given below.

```
this.add("South", new Button("Start"));
import java.applet.*;
import java.awt.*;
public class BorderLayouttest extends Applet
{
    public void init() {
        this.setLayout(new BorderLayout(2, 3));
        this.add("South", new Button("Start"));
        this.add("North", new Button("Stop"));
        this.add("East", new Button("Play"));
        this.add("West", new Button("Pause"));
    }
}
```



## Card Layout

A CardLayout breaks the applet into a deck of cards, each of which has its own Layout Manager. Only one card appears on the screen at a time. The user flips between cards, each of which shows a different set of components. The common analogy is with HyperCard on the Mac and Tool book on Windows. In Java this might

be used for a series of data input screens, where more input is needed than will comfortably fit on a single screen.

## Grid Layout

A GridLayout divides an applet into a specified number of rows and columns, which form a grid of cells, *each equally sized and spaced*. It is important to note that each is equally sized and spaced as there is another similar named Layout known as GridBagLayout. As Components are added to the layout they are placed in the cells, starting at the upper left hand corner and moving to the right and down the page. Each component is sized to fit into its cell. This tends to squeeze and stretch components unnecessarily.

*You will find the GridLayout is great for arranging Panels.* A GridLayout specifies the number of rows and columns into which components will be placed. The applet is broken up into a table of equal sized cells.

GridLayout is useful when you want to place a number of similarly sized objects. It is great for putting together lists of checkboxes and radio buttons as you did in the Ingredients applet. GridLayout looks like *Figure 3*.



Figure 3: GridLayout Demo

## Grid Bag Layout

GridBagLayout is the most precise of the five AWT Layout Managers. It is similar to the GridLayout, but components do not need to be of the same size. Each component can occupy one or more cells of the layout. Furthermore, components are not necessarily placed in the cells beginning at the upper left-hand corner and moving to the right and down.

In simple applets with just a few components you often need only one layout manager. In more complicated applets, however, you will often split your applet into panels, lay out the panels according to a layout manager, and give each panel its own layout manager that arranges the components inside it.

The GridBagLayout constructor is trivial, GridBagLayout() with no arguments.

```
GridBagLayout gbl = new GridBagLayout();
```

Unlike the GridLayout() constructor, this does not say how many rows or columns there will be. The cells your program refers to determine this. If you put a component in row 8 and column 2, then Java will make sure there are at least nine rows and three columns. (Rows and columns start counting at zero.) If you later put a component in row 10 and column 4, Java will add the necessary extra rows and columns. You may have a picture in your mind of the finished grid, but Java does not need to know this when you create a GridBagLayout.

Unlike most other LayoutManagers you should not create a GridBagLayout inside a cell to setLayout(). You will need access to the GridBagLayout object later in the applet when you use a GridBagConstraints.

A **GridBagConstraints** object specifies the location and area of the component's display area within the container (normally the applet panel) and how the component is laid out inside its display area. The GridBagConstraints, in conjunction with the component's minimum size and the preferred size of the component's container, determines where the display area is placed within the applet.

The GridBagConstraints() constructor is trivial

```
GridBagConstraints gbc = new GridBagConstraints();
```

Your interaction with a GridBagConstraints object takes place through its eleven fields and fifteen mnemonic constants.

### **gridx and gridy**

The gridx and gridy fields specify the x and y coordinates of the cell at the upper left of the Component's display area. The upper-left-most cell has coordinates (0, 0). The mnemonic constant GridBagConstraints.RELATIVE specifies that the Component is placed immediately to the right of (gridx) or immediately below (gridy) the previous Component added to this container.

### **Gridwidth and Gridheight**

The gridwidth and gridheight fields specify the number of cells in a row (gridwidth) or column (gridheight) in the Component's display area. The mnemonic constant GridBagConstraints.REMAINDER specifies that the Component should use all remaining cells in its row (for gridwidth) or column (for gridheight). The mnemonic constant GridBagConstraints.RELATIVE specifies that the Component should fill all but the last cell in its row (gridwidth) or column (gridheight).

### **Fill**

The GridBagConstraints fill field determines whether and how a component is resized if the component's display area is larger than the component itself. The mnemonic constants you use to set this variable are

GridBagConstraints.NONE :Don't resize the component

GridBagConstraints.HORIZONTAL: Make the component wide enough to fill the display area, but don't change its height.

GridBagConstraints.VERTICAL: Make the component tall enough to fill its display area, but don't change its width.

GridBagConstraints.BOTH: Resize the component enough to completely fill its display area both vertically and horizontally.

### **Ipadx and Ipady**

Each component has a minimum width and a minimum height, smaller than which it will not be. If the component's minimum size is smaller than the component's display area, then only part of the component will be shown.

The ipadx and ipady fields let you increase this minimum size by padding the edges of the component with extra pixels. For instance setting ipadx to two will guarantee that the component is at least four pixels wider than its normal minimum. (ipadx adds two pixels to each side.)

## Insets

The insets field is an instance of the java.awt.Insets class. It specifies the padding between the component and the edges of its display area.

## Anchor

When a component is smaller than its display area, the anchor field specifies where to place it in the grid cell. The mnemonic constants you use for this purpose are similar to those used in a BorderLayout but a little more specific.

They are

GridBagConstraints.CENTER  
GridBagConstraints.NORTH  
GridBagConstraints.NORTHEAST  
GridBagConstraints.EAST  
GridBagConstraints.SOUTHEAST  
GridBagConstraints.SOUTH  
GridBagConstraints.SOUTHWEST  
GridBagConstraints.WEST  
GridBagConstraints.NORTHWEST  
The default is GridBagConstraints.CENTER.

## weightx and weighty

The weightx and weighty fields determine how the cells are distributed in the container when the total size of the cells is less than the size of the container. With weights of zero (the default) the cells all have the minimum size they need, and everything clumps together in the center. All the extra space is pushed to the edges of the container. It doesn't matter where they go and the default of center is fine. See the program given below for visualizing GridBagLayout.

```
import java.awt.*;
public class GridbagLayouttest extends Frame
{
    Button b1,b2,b3,b4,b5;
    GridBagLayout gbl=new GridBagLayout();
    GridBagConstraints gbc=new GridBagConstraints();
    public GridbagLayouttest()
    {
        setLayout(gbl);
        gbc.gridx=0;
        gbc.gridy=0;
        gbl.setConstraints(b1=new Button("0,0"),gbc);
        gbc.gridx=4; //4th column
        gbc.gridy=3; //3rd row
        gbl.setConstraints(b2=new Button("4,3"),gbc);
        gbc.gridx=8; //8th column
        gbc.gridy=5; //5rd row
        gbl.setConstraints(b3=new Button("8,5"),gbc);
        gbc.gridx=10; //10th column
        gbc.gridy=3; //3rd row
        gbl.setConstraints(b4=new Button("10,3"),gbc);
        gbc.gridx=20; //20th column
        gbc.gridy=3; //3rd row
        gbl.setConstraints(b5=new Button("20,3"),gbc);
        add(b1);
    }
}
```

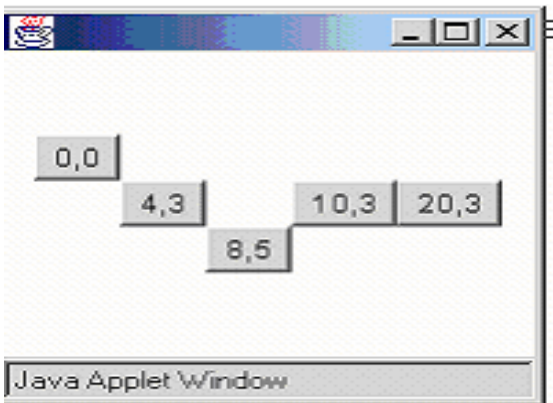


```

add(b2);
add(b3);
add(b4);
add(b5);
setSize(200,200);
setVisible(true);
}
public static void main(String a[])
{
GridbagLayouttest gb= new GridbagLayouttest();
}
}

```

Output:



Now we will discuss about different containers.

---

## 2.7 CONTAINER

---

You must be thinking that container will be a thing that contains something, like a bowl. Then you are right!! Actually container object is derived from the `java.awt.Container` class and is one of (or inherited from) three primary classes: `java.awt.Window`, `java.awt.Panel`, `java.awt.ScrollPane`.

The `Window` class represents a standalone window (either an application window in the form of a `java.awt.Frame`, or a dialog box in the form of a `java.awt.Dialog`). The `java.awt.Panel` class is not a standalone window by itself; instead, it acts as a background container for all other components on a form. For instance, the `java.awt.Applet` class is a direct descendant of `java.awt.Panel`.

The three steps common for all Java GUI applications are:

1. Creation of a container
2. Layout of GUI components.
3. Handling of events.

The `Container` class contains the `setLayout()` method so that you can set the default `LayoutManager` to be used by your GUI. To actually add components to the container, you can use the container's `add()` method:

```

Panel p = new java.awt.Panel();
Button b = new java.awt.Button("OK");
p.add(b);

```

A `JPanel` is a `Container`, which means that it can contain other components. GUI design in Java relies on a layered approach where each layer uses an appropriate layout manager.

FlowLayout is the default for JPanel objects. To use a different manager use either of the following:

```
JPanel pane2 = new JPanel()           // make the panel first
pane2.setLayout(new BorderLayout()); // then reset its manager
JPanel pane3 = new JPanel(new BorderLayout()); // all in one!
```

### **Check Your Progress 4**

- 1) Why do you think Layout Manager is important?  
.....  
.....  
.....  
.....
- 2) How does repaint() method work with Applet?  
.....  
.....  
.....
- 3) Each type of container comes with a default layout manager. Default for a Frame, Window or Dialog is \_\_\_\_\_
- 4) Give examples of stretchable components and non-stretchable components  
.....  
.....  
.....  
.....
- 5) The Listener interfaces inherit directly from which package.  
.....  
.....  
.....  
.....
- 6) How many Listeners are there for trapping mouse movements.  
.....  
.....  
.....  
.....

---

## **2.8 SUMMARY**

---

This unit is designed to know about the graphical interfaces in Java. In this unit you become familiar with AWT and SWING packages, which are used to add components in the container or a kind of tray. Swings are lightweight components and more flexible as compared to AWT. You learned to beautify your text by using Font class and Color class. For Geometric figures various in built classes of Java like for drawing line, circle, polygons etc are used. You learned to place the components in various

layouts. Java provides various layouts some of which are: FlowLayout, GridBagLayout, GridLayout, CardLayout, BorderLayout.

---

## 2.9 SOLUTIONS/ANSWERS

---

### Check Your Progress 1

- 1) Java defines two constructors for the Color Class of which one constructor takes three integer arguments and another takes three float arguments.

```
public Color (int r, int g, int b)
```

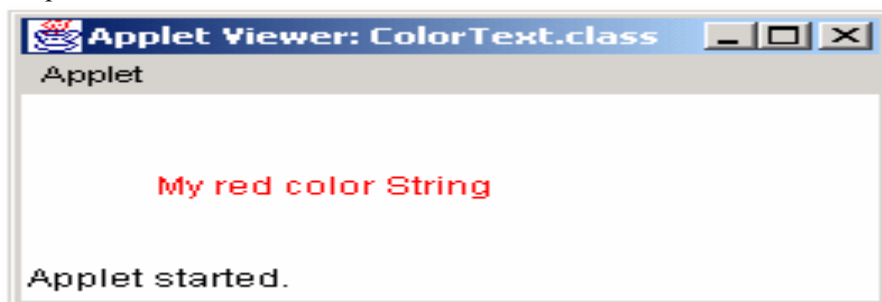
Creates a color based on red, green and blue components expressed as integers from 0 to 255. Here, the r, g, b means red, green and blue contents respectively.

`public Color(float r, float g, float b)` Creates a color based on red, green and blue components expressed as floating-point values from 0.0 to 1.0. Here, the r, g, b means red, green and blue contents respectively.

2)

```
import java.awt.*;
import java.applet.Applet;
public class ColorText extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.red);
        g.drawString("My Red color String",40,50);
    }
}
```

Output:



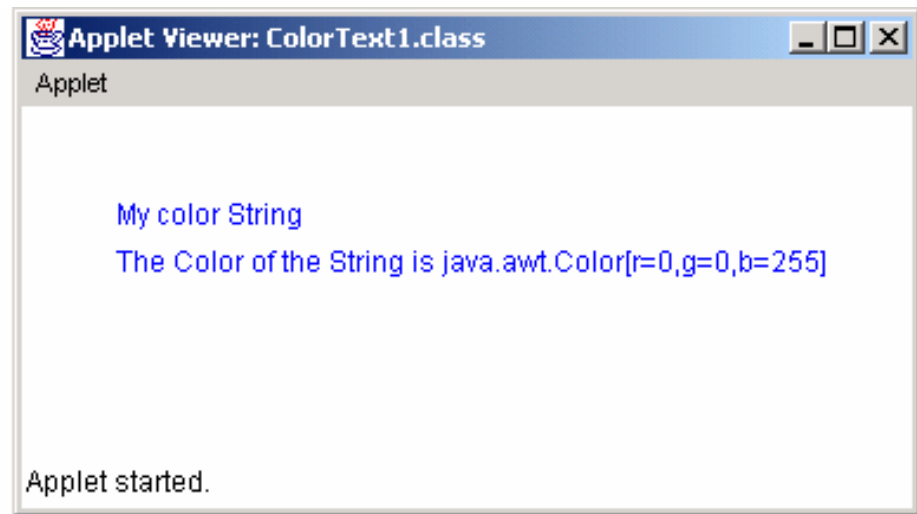
- 3) If you want to retrieve the Color of a text or a String you can use the method `public int getColor()`. It returns a value between 0 and 255 representing the color content.

The Program given bellow is written for getting RGB components of the color  
`import java.awt.*`

```
import java.applet.Applet;
public class ColorText extends Applet
{
    public void paint(Graphics g)
    {
        int color;
        g.drawString("My Pink color String",40,50);
    }
}
```

```
color = getColor(g);  
g.drawString("The Color of the String is "+color , 40,35);  
}  
}
```

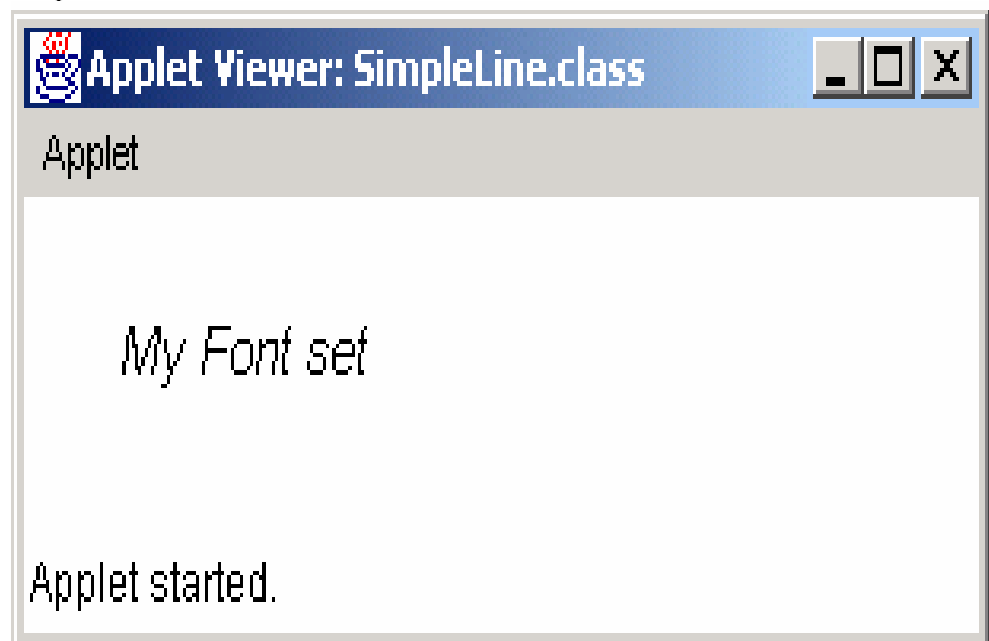
Output:



## Check Your Progress 2

```
1)  
import java.applet.*;  
import java.awt.*;  
public class SimpleLine extends Applet  
{  
    public void paint(Graphics g)  
    {  
        g.drawString("My Font set" , 30, 40);  
        g.setFont("Arial",Font.ITALIC,15);  
    }  
}
```

Output:

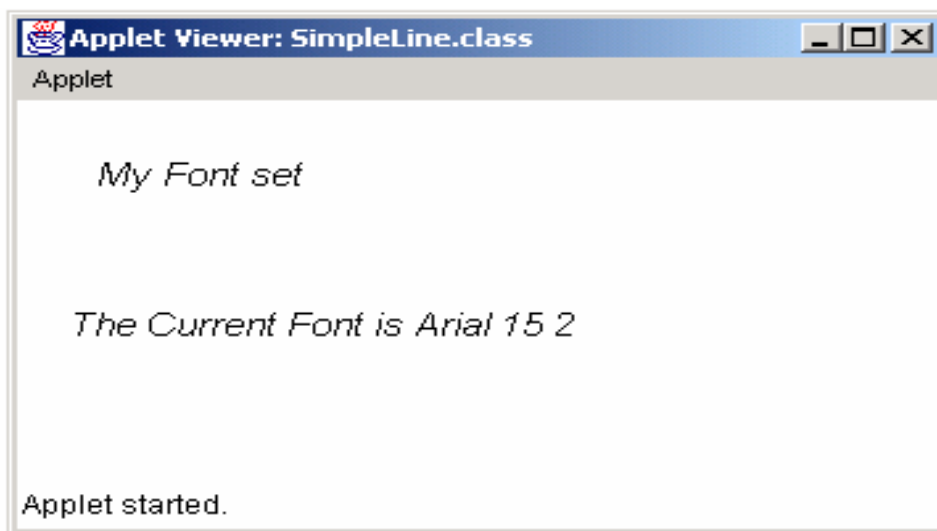


2) To retrieve the font of the string the method **getFont()** is used.

Program for font retrieval:

```
import java.applet.*;
import java.awt.*;
public class SimpleLine extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("My Font set", 30, 40);
        g.setFont("Arial",Font.ITALIC,15);
        g.drawString("current Font is " + g.getFont(),40,50);
        g.drawString( g.getFont().getName() + " " + g.getFont().getSize()+" "
        +g.getFont().getStyle(), 20, 110 );
    }
}
```

Output:



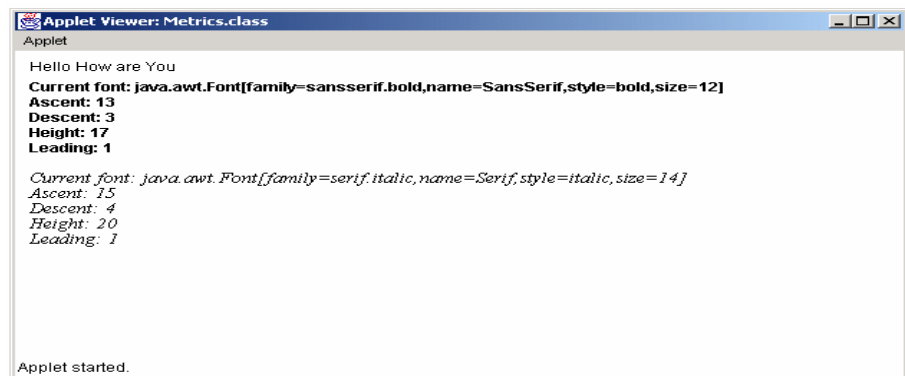
Note: The `getStyle()` method returns the integer value e.g above value 2 represent ITALIC.

3)

```
import java.awt.*;
import java.applet.Applet;
public class Metrics extends Applet {
    public void paint(Graphics g)
    {
        g.drawString("Hello How are You",50,60);
        g.setFont( new Font( "SansSerif", Font.BOLD, 12 ) );
        FontMetrics metrics = g.getFontMetrics();
        g.drawString( "Current font: " + g.getFont(), 10, 40 );
        g.drawString( "Ascent: " + metrics.getAscent(), 10, 55 );
        g.drawString( "Descent: " + metrics.getDescent(), 10, 70 );
        g.drawString( "Height: " + metrics.getHeight(), 10, 85 );
        g.drawString( "Leading: " + metrics.getLeading(), 10, 100 );
        Font font = new Font( "Serif", Font.ITALIC, 14 );
        metrics = g.getFontMetrics( font );
        g.setFont( font );
        g.drawString( "Current font: " + font, 10, 130 );
        g.drawString( "Ascent: " + metrics.getAscent(), 10, 145 );
        g.drawString( "Descent: " + metrics.getDescent(), 10, 160 );
    }
}
```

```
        g.drawString( "Height: " + metrics.getHeight(), 10, 175 );  
        g.drawString( "Leading: " + metrics.getLeading(), 10, 190 );  
    } // end method paint  
}
```

Output:



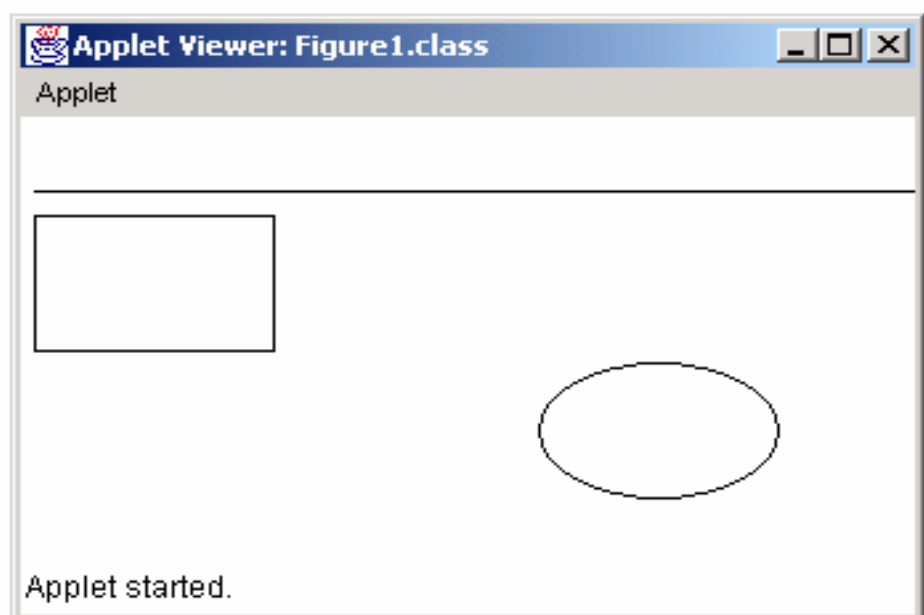
### Check Your Progress 3

1)

// Drawing lines, rectangles and ovals.

```
import java.awt.*;  
import java.applet.Applet;  
public class LinesRectsOvals extends Applet {  
    // display various lines, rectangles and ovals  
    public void paint( Graphics g )  
    {  
        g.drawLine( 5, 30, 350, 30 );  
        g.drawRect( 5, 40, 90, 55 );  
        g.drawOval( 195, 100, 90, 55 );  
    } // end method paint  
}
```

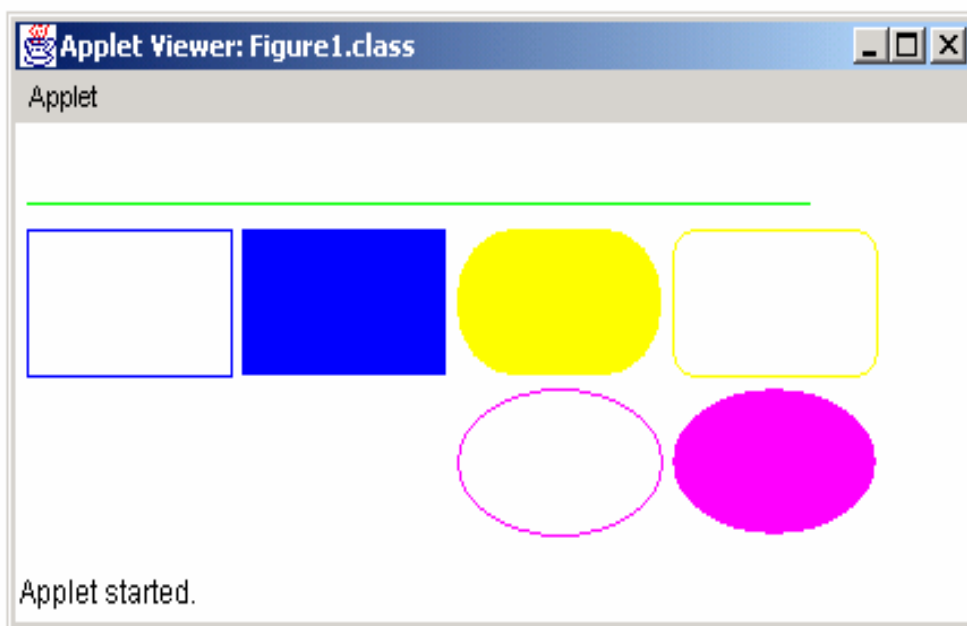
Output:



2)

```
import java.awt.*;
import javax.swing.*;
public class LinesRectsOvals extends JFrame
{
    public void paint( Graphics g )
    {
        g.setColor( Color.GREEN );
        g.drawLine( 5, 30, 350, 30 );
        g.setColor( Color.BLUE );
        g.drawRect( 5, 40, 90, 55 );
        g.fillRect( 100, 40, 90, 55 );
        //below two lines will draw a filled rounded rectangle with color yellow
        g.setColor( Color.YELLOW );
        g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
        //below two lines will draw a rounded figure of rectangle with color Pink
        g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
        g.setColor( Color.PINK );
        //below lines will draw an Oval figure with color Magenta
        g.setColor( Color.MAGENTA );
        g.drawOval( 195, 100, 90, 55 );
        g.fillOval( 290, 100, 90, 55 );
    }
}
```

Output:



3)

```
import java.awt.*;
class ChkGroup extends java.applet.Applet
{
    CheckboxGroup cbgr = new CheckboxGroup();
    Panel panel=new Panel();
    Frame fm=new Frame();
    Checkbox c1 = new Checkbox ("America Online");
    Checkbox c2 = new Checkbox ("MSN");
    Checkbox c3 = new Checkbox ("NetZero", cbgr, false);
}
```

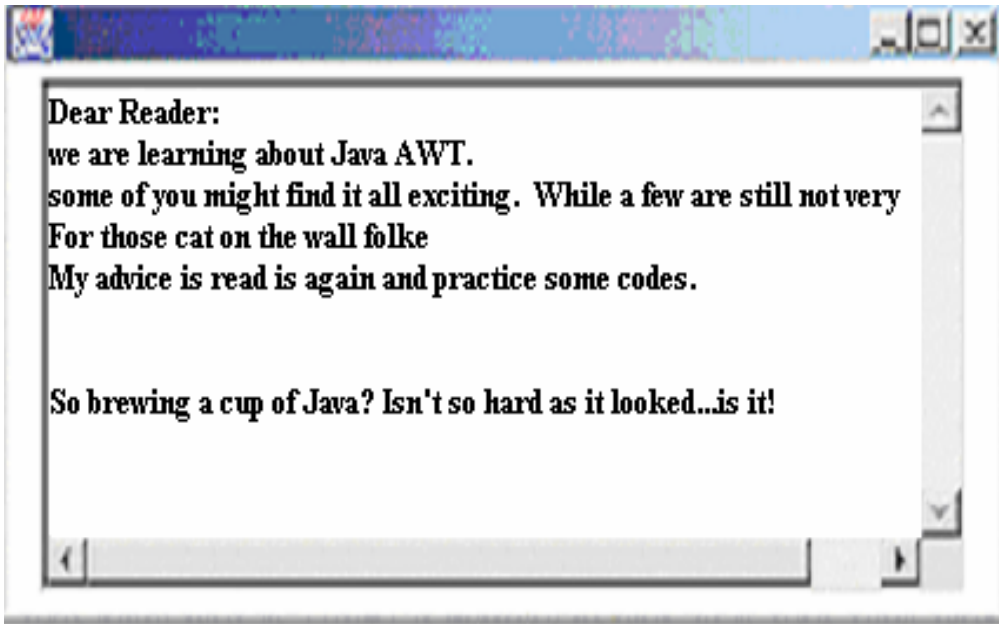
```
Checkbox c4 = new Checkbox ("EarthLink", cbgr, false);
Checkbox c5 = new Checkbox ("Bellsouth DSL", cbgr, false);
public void init()
{
    fm.setVisible(true);
    fm.setSize(300,400);
    fm.add(panel);
    panel.add(c1);
    panel.add(c2);
    panel.add(c3);
    panel.add(c4);
    panel.add(c5);
}
public static void main(String args[])
{
    ChkGroup ch=new ChkGroup();
    ch.init();
}
}
```

In the above example you will notice that the control which is not in the group has become a checkbox and the control which is in the group has become a radio button because checkbox allows you multiple selection but a radio button allows you a single selection at a time.

4)

```
import java.awt.*;
class TextFieldTest extends java.applet.Applet
{
    Frame fm=new Frame();
    Panel panel=new Panel();
    String letter = "Dear Readers: \n" +
    "We are learning about Java AWT. \n" +
    "Some of you might find it all exciting, while a few are still not very sure \n" +
    "For those cat on the wall folks \n" +
    "My advice is read it again and practice some codes. \n \n" +
    "So brewing a cup of Java? Isn't so hard as it looked...is it! " ;
    TextArea ltArea;
    public void init(){
        ltArea = new TextArea(letter, 10, 50);
        fm.setVisible(true);
        fm.setSize(300,400);
        fm.add(panel);
        panel.add(ltArea);
    }
    public static void main(String args[])
    {
        TextFieldTest tt=new TextFieldTest();
        tt.init();
    }
}
```





Basically you will notice that along with the text area control you will see the scrollbars.

- 5) Swing components that subclass JComponent has many features, including:  
A pluggable look and feel that can be used to customize the look and feel when the program executes on different platforms.

We can have Shortcut keys (called) mnemonics) for direct access to GUI components through the keyboard. It has very common event handling capabilities for cases where several GUI components initiate the same actions in the program. It gives the brief description of a GUI component's means (tool tips) that are displayed when the mouse cursor is positioned over the component for a short time. It has a support for technologies such as Braille screen for blind people. It has support for user interface localization-customizing the user interface for display in different languages and cultural conventions.

## Check Your Progress 4

- 1) A LayoutManager rearranges the components in the container based on their size relative to the size of the container.

Consider the window that just popped up. It has got five buttons of varying sizes. Resize the window and watch how the buttons move. In particular try making it just wide enough so that all the buttons fit on one line. Then try making it narrow and tall so that there is only one button on line. See if you can manage to cover up some of the buttons. Then uncover them. Note that whatever you try to do, the order of the buttons is maintained in a logical way. Button 1 is always before button 2, which is always before button 3 and so on.

It is harder to show, but imagine if the components changed sizes, as they might if you viewed this page in different browsers or on different platforms with different fonts.

The layout manager handles all these different cases for you to the greatest extent possible. If you had used absolute positioning and the window were smaller than expected or the components larger than you expected, some components would likely be truncated or completely hidden. In essence a layout manager defers decisions about positioning until runtime.

- 2) The repaint() method will cause AWT to invoke a component's update() method. AWT passes a Graphics object to update() –the same one that it passes to paint(). So, repaint() calls update() which calls paint().
- 3) BorderLayout
- 4) Stretchable: Button, Label and TextField  
Non-Stretchable: CheckBox
- 5) java.awt.event
- 6) MouseListener and MouseMotionListener

---

## UNIT 3 NETWORKING FEATURES

---

Structure	Page Nos.
3.0 Introduction	55
3.1 Objectives	55
3.2 Socket Overview	55
3.3 Reserved Parts and Proxy Servers	59
3.4 Internet Addressing: Domain Naming Services (DNS)	60
3.5 JAVA and the net: URL	61
3.6 TCP/IP Sockets	64
3.7 Datagrams	66
3.8 Summary	69
3.9 Solutions/ Answers	69

---

### 3.0 INTRODUCTION

---

Client/server applications are need of the time. It is challenging and interesting to develop Client/server applications. Java provides easier way of doing it than other programming such as C. Socket programming in Java is seamless. The java.net package provides a powerful and flexible infrastructure for network programming. Sun.\* packages have some classes for networking. In this unit you will learn the java.net package, using socket based communications which enable applications to view networking operations as I/O operation. A program can read from a socket or write to a socket as simply as reading a file or writing to a file.

With datagram and stream sockets you will be developing connection less and connection oriented applications respectively. Going through various classes and interfaces in java. net, will be useful in learning, how to develop networking applications easily. In this unit you will also learn use of stream sockets and the TCP protocol, which is the most desirable for the majority of java programmers for developing networking applications.

---

### 3.1 OBJECTIVES

---

After going though this Unit, you will be able to:

- define socket and elements of java networking;
  - describe the stream and datagram sockets, and their usage;
  - Explain how to implement clients and servers programs that communicates with each other;
  - define reserved sockets and proxy servers;
  - implement Java networking applications using TCP/IP Server Sockets, and
  - implement Java networking applications using Datagram Server Sockets.
- 

### 3.2 SOCKET OVERVIEW

---

Most of the inter process communication uses the *client server model*. The terms client and server refer to as the two processes, which will be communicating with each other. One of the two processes, the *client*, connects to the other process, the *server*, typically to make a request for information. A good analogy of this type of communication can be a person who makes a railway engineering from other person

Socket is a data structure that maintains necessary information used for communication between client & server. Therefore both end of communication has its own sockets.

may be through phone call: person making enquiry is a client and another person providing information is a server.

Notice that the client needs to know about the existence and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established. Once a connection is established, both sides can send and receive information.

The system calls for establishing a connection are different for the client and the server, but both involve the basic construct of a *socket*.

Java introduces socket based communications, which enable applications to view networking as if it were file I/O— a program which can read from socket or write to a socket with the simplicity as reading from a file or writing to a file. Java provides *stream sockets and datagram sockets*.

## **Stream Sockets**

Stream Sockets are used to provide a connection-oriented service (i.e. TCP-Transmission Control Protocol).

With stream sockets a process establishes a connection to another process. Once the connection is in place, data flows between processes in continuous streams.

## **Datagram Sockets**

This socket are used to provide a connection-less service, which does not guarantee that packets reach the destination and they are in the order at the destination. In this, individual packets of information are transmitted. In fact it is observed that packets can be lost, can be duplicated, and can even be out of sequence.

In this section you will get answers of some of the most common problems to be addressed in sockets programming using Java. Then you will see some example programs to learn how to write client and server applications.

The very first problem you have to address is “How will you open a socket?” Before reaching to the answer to this question you will definitely think that what type of socket is required? Now the answer can be given as follows:

1. If you were programming a client, then you would open a socket like this:

Socket MyClient;

MyClient = new Socket("Machine\_Name", PortNumber);

Where “Machine name” is the server machine you are trying to open a connection to, and “PortNumber” is the number of the port on server, on which you are trying to connect it. When selecting a port number, you should note that port numbers between 0 and 1,023 are reserved for privileged users or slandered services like e-mail, HTTP etc. For example, port number 21 is for FTP, 23 is for TLNET, and 80 is for HTTP. It is a good idea to handle exceptions while creating a new Socket.

Socket MyClient;

```
try {  
    MyClient = new Socket("Machine name", PortNumber);  
}  
catch (IOException e)
```

Port is a unique number association with a socket on a machine. In other word's port is a numbered socket on a machine.

```

{
    System.out.println(e);
}

```

2. If you are programming a server, then this is how you open a socket:

```

ServerSocket MyService;
try {
    MyService = new ServerSocket(PortNumber);
}
catch (IOException e)
{
    System.out.println(e);
}

```

While implementing a server you also need to create a socket object from the ServerSocket in order to listen for client and accept connections from clients.

```

Socket clientSocket = null;
try {
    serviceSocket = MyService.accept();
}
catch (IOException e)
{
    System.out.println(e);
}

```

You can notice that for a Client side programming you use the Socket class and for the Server side programming you use the ServerSocket class.

Now you know how to create client socket and server socket. Now you have to create input stream and output stream to receive and send data respectively.

### InputStream

On the client side, you can use the DataInputStream class to create an input stream to receive response from the server:

```

DataInputStream input;
try {
    input = new DataInputStream(MyClient.getInputStream());
}
catch (IOException e)
{
    System.out.println(e);
}

```

The class DataInputStream allows you to read lines of text and Java primitive data types in a portable way. It has methods such as read, readChar, readInt, readDouble, and readLine. Use whichever function you think suits your needs depending on the type of data that you receive from the server.

On the server side, you can use DataInputStream to receive input from the client:

```

DataInputStream input;
try {
    input = new DataInputStream(serviceSocket.getInputStream());
}
catch (IOException e)

```

```
        {  
            System.out.println(e);  
        }  
    }  
OutputStream
```

On the client side, you can create an output stream to send information to the server socket using the class `PrintStream` or `DataOutputStream` of `java.io`:

```
PrintStream output;  
try {  
    output = new PrintStream(MyClient.getOutputStream());  
}  
catch (IOException e)  
{  
    System.out.println(e);  
}
```

The class `PrintStream` has methods for displaying textual representation of Java primitive data types. Its `write ()` and `println ()` methods are important here. Also, you can use the `DataOutputStream`:

```
DataOutputStream output;  
try {  
    output = new DataOutputStream(MyClient.getOutputStream());  
}  
catch (IOException e)  
{  
    System.out.println(e);  
}
```

The class `DataOutputStream` allows you to write Java primitive data types; many of its methods write a single Java primitive type to the output stream. The method `writeBytes ()` is a useful one.

On the server side, you can use the class `PrintStream` to send information to the client.

```
PrintStream output;  
try {  
    output = new PrintStream(serviceSocket.getOutputStream());  
}  
catch (IOException e)  
{  
    System.out.println(e);  
}
```

If you have opened a socket for connection, after performing the desired operations your socket should be closed. You should always close the output and input stream before you close the socket.

### **On the client side:**

```
try  
{  
    output.close();  
    input.close();  
    MyClient.close();  
}  
catch (IOException e)
```

```
{
    System.out.println(e);
}
```

### On the server side:

```
try
{
    output.close();
    input.close();
    serviceSocket.close();
    MyService.close();
}
catch (IOException e)
{
    System.out.println(e);
}
```

### Check Your Progress 1

- 1) Write a program to show that from the client side you send a string to a server that reverse the string which is displayed on the client side.

.....

.....

.....

- 2) Describe different types of sockets.

.....

.....

.....

- 3) What are Datagram and Stream Protocols?

.....

.....

.....

---

## 3.3 RESERVED PORTS AND PROXY SERVERS

---

Now let us see some reserve ports. As we have discussed earlier, there are some port numbers, which are **reserved** for specific purposes on any computer working as a server and connected to the Internet. Most standard applications and protocols use **reserved** port numbers, such as email, FTP, and HTTP.

When you want two programs to talk to each other across the Internet, you have to find a way to initiate the connection. So at least one of the ‘partners’ in the conversation has to know where to find the other one or in other words the address of other one. This can be done by address (IP number + port number) of the one side to the other.

However, a problem could arise if this address must not be taken over by any other program. In order to avoid this, there are some port numbers, which are reserved for specific purposes on any computer connected to the Internet. Such ports are reserved for programs such as ‘Telnet’, ‘Ftp’ and others. For example, Telnet uses port 23, and

FTP uses port 21. Note that for each kind of service, not only a port number is given, but also a protocol name (usually TCP or UDP).

Two services may use the same port number, provided that they use different protocols. This is possible due to the fact that different protocols have different address spaces: port 23 of a one machine in the TCP protocol address space is not equivalent to port 23 on the same machine, in the UDP protocol address space.

## **Proxy Servers**

A proxy server is a kind of buffer between your computer and the Internet resources you are accessing. They accumulate and save files that are most often requested by thousands of Internet users in a special database, called “cache”. Therefore, proxy servers are able to increase the speed of your connection to the Internet. The cache of a proxy server may already contain information you need by the time of your request, making it possible for the proxy to deliver it immediately. The overall increase in performance may be very high.

Proxy servers can help in cases when some owners of the Internet resources impose some restrictions on users from certain countries or geographical regions. In addition to that, a type of proxy server called anonymous proxy servers can hide your IP address thereby saving you from vulnerabilities concerned with it.

## **Anonymous Proxy Servers**

Anonymous proxy servers hide your IP address and thereby prevent your data from unauthorized access to your computer through the Internet. They do not provide anyone with your IP address and effectively hide any information about you. Besides that, they don’t even let anyone know that you are surfing through a proxy server. Anonymous proxy servers can be used for all kinds of Web-services, such as Web-Mail (MSN Hot Mail, Yahoo mail), web-chat rooms, FTP archives, etc. *ProxySite.com* will provide a huge list of public proxies.

Any web resource you access can gather personal information about you through your unique IP address – your ID in the Internet. They can monitor your reading interests, spy upon you, and according to some policies of the Internet resources, deny accessing any information you might need. You might become a target for many marketers and advertising agencies that, having information about your interests and knowing your IP address as well as your e-mail. They will be able to send you regularly their spam and junk e-mails.

---

## **3.4 INTERNET ADDRESSING: DOMAIN NAMING SERVICES (DNS)**

---

You know there are thousands of computers in a network; it is not possible to remember the IP address of each system. Domain name system provides a convenient way of finding computer systems in network based on their name and IP address. Domain name services resolves names to the IP addresses of a machine and vice-versa. Domain name system is a hierarchical system where you have a top-level domain name server sub domain and clients with names & IP address.

When you use a desktop client application, such as e-mail or a Web browser, to connect to a remote computer, your computer needs to resolve the addresses you have entered, into the IP addresses it needs to connect to the remote server. DNS is a way to resolve domain name to IP addresses on a TCP/IP network.



The major components of DNS are:

Domain Name Space,  
Domain Name Servers and Resource Records (RR),  
Domain Name Resolvers (DNRs).

The Domain Name Space: It is a tree-structured name space that contains the domain names and data associated with the names. For example, *astrospeak.indiatimes.com* is a node within the *indiatimes.com* domain, which is a node in the *com* domain. Data associated with *astrospeak.indiatimes.com* includes its IP address. When you use DNS to find a host address, you are querying the Domain Name Space to extract information.

*A Domain Name Server provides information about a subset of the Domain Name Space.*

The Domain Name Space for an entity is the name by which the entity is known on the Internet. For example, in the organization shown in, you have two entities with two address spaces. The *Times of India* organization is known on the Internet as the *timesofindia.com* domain, and our sample Internet organization is known as the *indiatimes.com* domain. Hosts at these organizations are known as a host name plus the domain name; for example, *money.timesofindia.com*. Similarly, users in these domains can be found by their e-mail aliases; for example, *abc@indiatimes.com*.

The Domain Name Server points to other Domain Name Servers that have information about other subsets of the Domain Name Space. When you query a Domain Name Server, it returns information if it is an authoritative server for that domain. If the Domain Name Server doesn't have the information, it refers you to a higher level Domain Name Server, which in turn can refer you to another Domain Name Server, until it locates the one with the requested information. In this way, no single server needs to have all the information for every host you might need to contact.

A Domain Name Resolver extracts information from Domain Name Servers so you can use host addresses instead of IP addresses in clients such as a Web browser or a File Transfer Protocol (FTP) client, or with utilities such as ping, tracer, or finger. The DNR is typically built into the TCP/IP implementation on the desktop and needs to know only the IP address of the Domain Name Server. Configuring the DNR on the desktop is usually a matter of filling in the TCP/IP configuration data.

---

## 3.5 JAVA AND THE NET: URL

---

You know each package defines a number of classes, interfaces, exceptions, and errors. The *java.net* package contains these, interfaces, classes, and exceptions: This package is used in programming where you need to know some information regarding the internet or if you want to communicate between two host computers.

### Interfaces in *java.net*

ContentHandlerFactory  
FileNameMap  
SocketImplFactory  
URLStreamHandlerFactory

### Exceptions in *java.net*

BindException  
ConnectException  
MalformedURLException  
NoRouteToHostException

### Classes in *java.net*

ContentHandler  
DatagramSocket  
DatagramPacket  
DatagramSocketImpl  
HttpURLConnection  
InetAddress

ProtocolException  
SocketException  
UnknownHostException  
UnknownServiceException

MulticastSocket  
ServerSocket  
Socket  
SocketImpl  
URL  
URLConnection  
URLEncoder  
URLStreamHandler

## **URL**

URL is the acronym for Uniform Resource Locator. It represent the addresses of resources on the Internet. You need to provide URLs to your favorite Web browser so that it can locate files on the Internet. In other words you can see URL as addresses on letters so that the post office can locate for correspondents URL class is provided in the `java.net` package to represent a URL address.

URL object represents a URL address. The URL object always refers to an absolute URL. You can construct from an absolute URL, a relative URL. URL class provides accessor methods to get all of the information from the URL without doing any string parsing.

You can connect to a URL by calling `openConnection()` on the URL. The `openConnection()` method returns a `URLConnection` object. `URLConnection` object is used for general communications with the URL, such as reading from it, writing to it, or querying it for content and other information.

## **Reading from and Writing to a URL Connection**

Some URLs, such as many that are connected to cgi-bin scripts, allow you to write information to the URL. For example, if you have to search something, then search script may require detailed query data to be written to the URL before the search can be performed.

Before interacting with the URL you have to first establish a connection with the Web server that is responsible for the document identified by the URL.

Then you can use TCP socket for the connection is constructed by invoking the `openConnection` method on the URL object. This method also performs the name resolution necessary to determine the IP address of the Web server.

The `openConnection` method returns an object of type `URLConnection` to the Web server which is requested by calling `connection` on the `URLConnection` object. For input and output handling for the document identified by the URL `InputStream` and `OutputStream` are used respectively.

Below are the various constructors and methods of URL class of `java.net` package.

```
public URL(String protocol, String host, int port, String file)  
throws MalformedURLException
```

```
public URL(String protocol, String host, String file)  
throws MalformedURLException
```

```
public URL(String spec) throws MalformedURLException  
public URL(URL context, String spec) throws MalformedURLException  
public int getPort()
```

```

public String getFile()
public String getProtocol()
public String getHost()
public String getRef()
public boolean equals(Object obj)
public int hashCode()
public boolean sameFile(URL other)
public String toString()
public URLConnection openConnection() throws IOException
public final InputStream openStream() throws IOException
public static synchronized void setURLStreamHandlerFactory(
URLStreamHandlerFactory factory)

```

In the example program given below, URL of homepage of “rediff.com” is created.

```

import java.net.*;
class URL_Test
{
    public static void main(String[] args) throws MalformedURLException
    {
        URL redURL = new URL("http://in.rediff.com/index.html");
        System.out.println("URI Protocol:"+redURL.getProtocol());
        System.out.println("URI Port:"+redURL.getPort());
        System.out.println("URI Host:"+redURL.getHost());
        System.out.println("URI File"+redURL.getFile());
    }
}

```

Output:  
URI Protocol:http  
URI Port:-1  
URI Host:in.rediff.com  
URI File/index.html

## Check Your Progress 2

- 1) When should you use Anonymous Proxy Servers?

.....

.....

.....

.....

.....

- 2) Explain various kinds of domain name servers.

.....

.....

.....

.....

- 3) Write a program in Java to know the Protocol, Host, Port, File and Ref of a particular URL using *java.net.URL* package.

.....

.....

.....

*TCP/IP is a set of protocols used for communication between different types of computers and networks.*

---

## 3.6 TCP/IP SOCKETS

---

TCP/IP refers to two of the protocols in the suite: the *Transmission Control Protocol* and the *Internet Protocol*.

These protocols utilize sockets to exchange data between machines. The TCP protocol requires that the machines communicate with one another in a reliable, ordered stream. Therefore, all data sent from one side must be received in correct order and acknowledged by the other side. This takes care of lost and dropped data by means of acknowledgement and re-transmission. UDP, however, simply sends out the data without requiring knowing that the data be received.

In order to do communication over the TCP protocol, a connection must first be established between the pair of sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once two sockets have been connected, they can be used to transmit data in both (or either one of the) directions.

UDP is an unreliable protocol; there is no guarantee that the datagrams you have sent will be put in the same order by the receiving socket. On the other hand, TCP is a reliable protocol; it is guaranteed that the packets you send will be put in the order in which they were sent.

In short, TCP is useful for implementing network services: such as remote login (rlogin, telnet) and file transfer (FTP). These services require data of indefinite length to be transferred. UDP is less complex and incurs fewer overheads. UDP is often used in implementing client/server applications in distributed systems, which are built over local area networks.

We have already discussed about client and server Sockets in section 3.2 of this Unit. Recall the discussions in section 3.2, two packages `java.net.ServerSocket` and `java.net.Socket` were used to create sockets.

Before we discuss about `Socket` and `ServerSocket` class, it is important to know about `InetAddress` class. Let us see what is `InetAddress` class.

### **InetAddress**

This class is used for encapsulating numerical IP address and domain name for that address.

Because `InetAddress` is not having any constructor, its objects are created by using any of the following three methods

```
static InetAddress getLocalHost() throws UnknownHostException:
    returns the InetAddress object representing local host
static InetAddress getByName() throws UnknownHostException:
    returns the InetAddress object for the host name passed to
    it.
```

```
static InetAddress getAllByName() throws UnknownHostException: returns an array
of InetAddresses representing all the addresses that a particular name is resolves to.
```

### **Java.net.Socket**

#### Constructors

```
public Socket(InetAddress addr, int port): creates a stream socket and connects it to
the specified port number at the specified IP address
```

`public Socket (String host, int port):` creates a stream socket and connects it to the specified port number at the specified host

Methods:

`InetAddress getAddress() :` Return Inet Address of object associated with Socket

`int getPort() :` Return remote port to which socket is connected

`int getLocalPort() :` Return local port to which socket object is connected.

`public InputStream getInputStream():` Get InputStream associated with Socket

`public OutputStream getOutputStream():` Return OutputStream associated with socket

`public synchronized void close() :` closes the Socket.

## Java.net.ServerSocket

Constructors:

`public ServerSocket(int port):` creates a server socket on a specified port with a queue length of 50. A port number 0 creates a socket on any free port.

`public ServerSocket(int port, int QueLen):` creates a server socket on a specified port with a queue length of QueLen.

`public ServerSocket(int port, int QueLen, InetAddress localAdd):` creates a server socket on a specified port with a queue length specified by QueLen. On a multihomed host, localAdd specifies the IP address to which this socket binds.

Methods:

`public Socket accept():` listens for a connection to be made to this socket and accepts it. `public void close():` closes the socket.

## Java TCP Socket Example

A Server (web server) at ohm.uwaterloo.ca

- listens to port 80 for Client Connection Requests
- Establish InputStream for sending data to client
- Establish OutputStream for receiving data from client

TCP connection example: (Server)

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
public class myserver {
    public static void main( String [] s) {
        try {
            ServerSocket s = new ServerSocket( 80 );
            While (true) {
                // wait for a connection request from client
                Socket clientConn = s.accept();
                InputStream in = clientConn.getInputStream();
                OutputStream out = clientConn.getOutputStream();
                // communicate with client
                // ..
                clientConn.close(); // close client connection
            }
        } catch (Exception e) {
            System.out.println("Exception!");
            // do something about the exception
        }
    }
}
```

```
TCP connection example: (Client)
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
public class myclient {
    public static void main( String [] s) {
        try {
            InetAddress addr = InetAddress.getByName(
                "ohm.uwaterloo.ca");
            Socket s = new Socket(addr, 80);
            InputStream in = s.getInputStream();
            OutputStream out = s.getOutputStream();
            // communicate with remote process
            // e.g. GET document /~ece454/index.html
            s.close();
        } catch(Exception e) {
            System.out.println("Exception");
            // do something about the Exception
        }
    }
}
```

---

## 3.7 DATAGRAMS

---

A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. Datagrams runs over UDP protocol.

The UDP protocol provides a mode of network communication where packets sent by applications are called datagrams. A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. The datagram Packet and Datagram Socket classes in the java.net package implement system independent datagram communication using UDP.

Actually the DatagramPacket class is a wrapper for an array of bytes from which data will be sent or into which data will be received. It also contains the address and port to which the packet will be sent.

DatagramPacket constructors:

```
public DatagramPacket(byte[] data, int length)
public DatagramPacket(byte[] data, int length, InetAddress host, int port)
```

You can construct a DatagramPacket object by passing an array of bytes and the number of those bytes to the DatagramPacket() constructor:

```
String s = "My first UDP Packet"
byte[] b = s.getBytes();
DatagramPacket dp = new DatagramPacket(b, b.length());
```

Normally the object of DatagramPacket is created by passing in the host and port to which you want to send the packet with data and its length. For example, object m in the code given below:

```
try
{
    InetAddress m = new InetAddress("http://mail.yahoo.com");
    int chargen = 19;
```

```
String s = "My second UDP Packet"
byte[] b = s.getBytes();
DatagramPacket dp = new DatagramPacket(b, b.length, m, chargen);
}
catch (UnknownHostException ex)
{
System.err.println(ex);
}
```

The byte array that's passed to the constructor is stored by reference, not by value. If you change its contents elsewhere, the contents of the `DatagramPacket` change as well.

`DatagramPackets` themselves are not immutable. You can change the data, the length of the data, the port, or the address at any time using the following four methods:

```
public void setAddress(InetAddress host)
public void setPort(int port)
public void setData(byte buffer[])
public void setLength(int length)
```

You can retrieve address, port, data , and length of data using the following four get methods:

```
public InetAddress getAddress()
public int getPort()
public byte[] getData()
public int getLength()
```

`DatagramSocket` constructors: The `java.net.DatagramSocket` class has three constructors:

```
public DatagramSocket() throws SocketException
public DatagramSocket(int port) throws SocketException
public DatagramSocket(int port, InetAddress laddr) throws SocketException
```

The first is used for datagram sockets that are primarily intended to act as clients, i.e., a sockets that will send datagrams before receiving anything from anywhere.

The second constructors that specify the port and optionally the IP address of the socket, are primarily intended for servers that must run on a well-known port.

## Sending UDP Datagrams

To send data to a particular server, you first must convert the data into byte array. Next you pass this byte array, the length of the data in the array (most of the time this will be the length of the array) the `InetAddress` and port to `DatagramPacket()` constructor.

For example, first you create `atagramPacket` object

```
try
{
InetAddress m = new InetAddress("http://mail.yahoo.com");
int chargen = 19;
String s = "My second UDP Packet";
byte[] b = s.getBytes();
DatagramPacket dp = new DatagramPacket(b, b.length, m, chargen);
}
catch (UnknownHostException ex)
{
System.err.println(ex);
}
```

```
}
```

Now create a DatagramSocket object and pass the packet to its send() method as try

```
{
    DatagramSocket sender = new DatagramSocket();
    sender.send(dp);
}
catch (IOException ex)
{
    System.err.println(ex);
}
```

## Receiving UDP Datagrams

To receive data sent to you, construct a DatagramSocket object bound to the port on which you want to receive the data. Then you pass an empty DatagramPacket object to the DatagramSocket's receive() method.

public void receive(DatagramPacket dp) throws IOException.

The calling threads blocks until the datagram is received. Then dp is filled with the data from that datagram. You can then use getPort() and getAddress() to tell where the packet came from, getData() to retrieve the data, and getLength() to see how many bytes were in the data. If the received packet is too long for the buffer, then it is truncated to the length of the buffer. You can write program with the help of code written below:

```
try
{
    byte buffer = new byte[65536]; // maximum size of an IP packet
    DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
    DatagramSocket ds = new DatagramSocket(2134);
    ds.receive(dp);
    byte[] data = dp.getData();
    String s = new String(data, 0, data.getLength());
    System.out.println("Port " + dp.getPort() + " on " + dp.getAddress()
        + " sent this message:");
    System.out.println(s);
}
catch (IOException ex)
{
    System.err.println(ex);
}
```

## Check Your Progress 3

- 1) Write a program to print the address of your local machine and Internet web site rediff.com and webduniya.com.

.....

.....

.....

.....

- 2) Give a brief explanation of TCP Client/Server Interaction.

.....

.....

.....



- 3) Differentiate between TCP or UDP Protocols.

.....

.....

.....

- 4) Write a program, which does UDP Port scanner by checking out various port numbers status (i.e.) Are they occupied or free?

.....

.....

.....

.....

.....

---

## 3.8 SUMMARY

---

You have learnt that Java provides stream sockets and datagram sockets. With stream sockets a process establishes a connection to another process. While the connection is in place, data flows between the processes in continuous streams.

Stream sockets provide connection-oriented service. The TCP protocol is used for this purpose. With datagram sockets individual packets of information are transmitted. UDP protocol is used for this kind of communication. Stream based connections are managed with Sockets objects.

Datagram packets are used to create the packets to send and receive information using Datagram Sockets. Connection oriented services can be seen as your telephone service and connection-less services can be seen as Radio Broadcast.

Domain naming services solve the problem of remembering the long IP address of various web sites and computers. You also have learn that there are some dedicated Port numbers for various protocols which are known as reserved port like for FTP you have port no. 21.

With Proxy servers you can prevent your computer not accessible to anyone you don't want. Your computer data cannot be traced easily if you are using proxy servers, as the IP addresses will not be known to the second person directly.

---

## 3.9 SOLUTIONS/ANSWERS

---

### Check Your Progress 1

1)  
Client side Programming  
import java.io.\*;  
import java.net.\*;

```
public class Client
{
    public static final int DEFAULT_PORT = 8000;
    public static void usage()
    {
        System.out.println("Usage: java Client <hostname> [<port>]");
        System.exit(0);
    }
}
```

```
}
public static void main(String[] args)
{
    int port = DEFAULT_PORT;
    Socket s = null;

    // Parse the port specification
    if ((args.length != 1) && (args.length != 2)) usage();
    if (args.length == 1) port = DEFAULT_PORT;
    else
    {
        try
        {
            port = Integer.parseInt(args[1]);
        }
        catch (NumberFormatException e)
        { usage();
        }
    }
    try
    {
        // Here is a socket to communicate to the specified host and port
        s = new Socket(args[0], port);

        BufferedReader sin = new BufferedReader(new
        InputStreamReader(s.getInputStream())); //stream for reading
        PrintStream sout = new PrintStream(s.getOutputStream());
            // stream for writing lines of text
        // Here is a stream for reading lines of text from the console
        BufferedReader in = new BufferedReader(new
        InputStreamReader(System.in));

        System.out.println("Connected to " + s.getInetAddress()
            + ":" + s.getPort());
        String line;
        while(true)
        {
            // print a prompt
            System.out.print("> ");
            System.out.flush();
            // read a line from the console; check for EOF
            line = in.readLine();
            if (line == null) break;
            // Send it to the server
            sout.println(line);
            // Read a line from the server.
            line = sin.readLine();
            // Check if connection is closed (i.e. for EOF)
            if (line == null)
            {
                System.out.println("Connection closed by server.");
                break;
            }
            // And write the line to the console.
            System.out.println(line);
        }
    }
    catch (IOException e)
```

```

    {
        System.err.println(e);
    }
    finally
    {
        try
        {
            if (s != null)
                s.close();
        }
        catch (IOException e2) { ; }
    }
}
}

```

## Server Side Programming

```

import java.io.*;
import java.net.*;
public class Server extends Thread
{
    public final static int DEFAULT_PORT = 8000;
    protected int port;
    protected ServerSocket listen_socket;
    public static void fail(Exception e, String msg)
    {
        System.err.println(msg + ": " + e);
        System.exit(1);
    }
    // Creating a ServerSocket to listen for connections on;
    public Server(int port)
    {
        if (port == 0) port = DEFAULT_PORT;
        this.port = port;
        try { listen_socket = new ServerSocket(port);
        }
        catch (IOException e)
        {
            fail(e, "Exception creating server socket");
        }
        System.out.println("Server: listening on port " + port);
        this.start();
    }
    // create a Connection object to handle communication through the new Socket.
    public void run()
    {
        try
        {
            while(true)
            {
                Socket client_socket = listen_socket.accept();
                Connection c = new Connection(client_socket);
            }
        }
        catch (IOException e)
        {
            fail(e, "Exception while listening for connections");
        }
    }
}

```

```
    }  
    }  
    public static void main(String[] args)  
    {  
        int port = 0;  
        if (args.length == 1)  
        {  
            try  
            {  
                port = Integer.parseInt(args[0]);  
            }  
            catch (NumberFormatException e)  
            {  
                port = 0;  
            }  
        }  
        new Server(port);  
    }  
}  
  
// A thread class that handles all communication with a client  
class Connection extends Thread  
{  
    protected Socket client;  
    protected BufferedReader in;  
    protected PrintStream out;  
    // Initialize the streams and start the thread  
    public Connection(Socket client_socket)  
    {  
        client = client_socket;  
        try  
        {  
            in = new BufferedReader(new InputStreamReader(client.getInputStream()));  
            out = new PrintStream(client.getOutputStream());  
        }  
        catch (IOException e)  
        {  
            try  
            {  
                client.close();  
            }  
            catch (IOException e2) { ; }  
            System.err.println("Exception while getting socket streams: " + e);  
            return;  
        }  
        this.start();  
    }  
    public void run()  
    {  
        String line;  
        StringBuffer revline;  
        int len;  
        try  
        {  
            for(;;)  
            {  
                // read in a line  
                line = in.readLine();  
            }  
        }  
    }  
}
```

```

        if (line == null) break;
        // reverse it
        len = line.length();
        revline = new StringBuffer(len);
        for(int i = len-1; i >= 0; i--)
            revline.insert(len-1-i, line.charAt(i));
        // and write out the reversed line
        out.println(revline);
    }
}
catch (IOException e) { ; }
finally
{
    try
    {
        client.close();
    }
    catch (IOException e2) {;}
}
}
}

```

### Output :

The screenshot shows two overlapping Windows command prompt windows. The top window, titled 'C:\WINNT\system32\cmd.exe - java Server', shows the server's startup sequence: navigating to 'C:\jdk1.3\bin' and running 'java Server'. The output indicates the server is listening on port 8000. The bottom window, titled 'C:\WINNT\system32\cmd.exe - java Client Archana', shows the client's startup sequence: navigating to 'C:\jdk1.3\bin' and running 'java Client Archana'. The output shows the client connecting to 'Archana/10.10.13.215:8000'. After sending the message 'Hello Good Morning', the client receives the reversed response 'gnirroM dooG olleH'.

In the output screen you can see that that server is running and listening to on port number 8000.

At the client side you can see that whenever you will write a string as “Hello Good Morning”

Then the string goes to server side and the server reverses it as the reverse function is written on the server.

2) In Table given below Socket type protocols and their description is given:

**Table 1:Types of Sockets**

Socket type	Protocol	Description
SOCK_STREAM	Transmission Control Protocol (TCP)	The stream socket (SOCK_STREAM) interface defines a reliable connection-oriented service. Data is sent without errors or duplication and is received in the same order as it is sent.

SOCK_DGRAM	User Datagram Protocol (UDP)	The datagram socket (SOCK_DGRAM) interface defines a connectionless service for datagrams, or messages. Datagrams are sent as independent packets. The reliability is not guaranteed, data can be lost or duplicated, and datagrams can arrive out of order. However, datagram sockets have improved performance capability over stream sockets and are easier to use.
SOCK_RAW	IP, ICMP, RAW	The raw socket (SOCK_RAW) interface allows direct access to lower-layer protocols such as Internet Protocol (IP).

### **Note:**

The type of socket you use is determined by the data you are transmitting:

- When you are transmitting data where the integrity of the data is high priority, you ***must*** use stream sockets.
  - When the data integrity is not of high priority (for example, for terminal inquiries), use datagram sockets because of their ease of use and higher performance capability.
- 3) There are two communication protocols that one can use for socket programming: datagram communication and stream communication.

### **Datagram communication:**

The datagram communication protocol, known as UDP (user datagram protocol), is a connectionless protocol, meaning that each time you send datagrams, you also need to send the local socket descriptor and the receiving socket's address. As you can tell, socket address is required each time a communication is made.

### **Stream communication:**

The stream communication protocol is known as TCP (Transfer Control Protocol). Unlike UDP, TCP is a connection-oriented protocol. In order to do communication over the TCP protocol, a connection must first be established between the pair of sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once two sockets have been connected, they can be used to transmit data in both (or either one of the) directions.

### **Check Your Progress 2**

1. Using an anonymous proxy server you don't give anybody a chance to find out your IP address to use it in their own interests. We can offer you two ways to solve your IP problem:
  - i) Secure Tunnel - a pay proxy server with plenty of features. Effective for personal use, when your Internet activities are not involved in very active surfing, web site development, mass form submitting, etc. In short, Secure Tunnel is the best solution for most of Internet users. Ultimate protection of privacy - nobody can find out where you are engaged in surfing. Blocks all methods of tracking. Provides an encrypted connection for all forms of web browsing, including http, news, mail, and the especially vulnerable IRC and ICQ. Comes with special totally preconfigured software.

- ii) ProxyWay - a proxy server agent which you use together with your web browser to ensure your anonymity when you surf the Internet. It contains a database of anonymous proxy servers and allows you to easily test their anonymity. Using a network of publicly accessible servers ProxyWay shields your current connection when you visit websites, download files, or use web-based e-mail accounts.
- 2) Our own small proxy list is also a good place to start with if you are a novice.

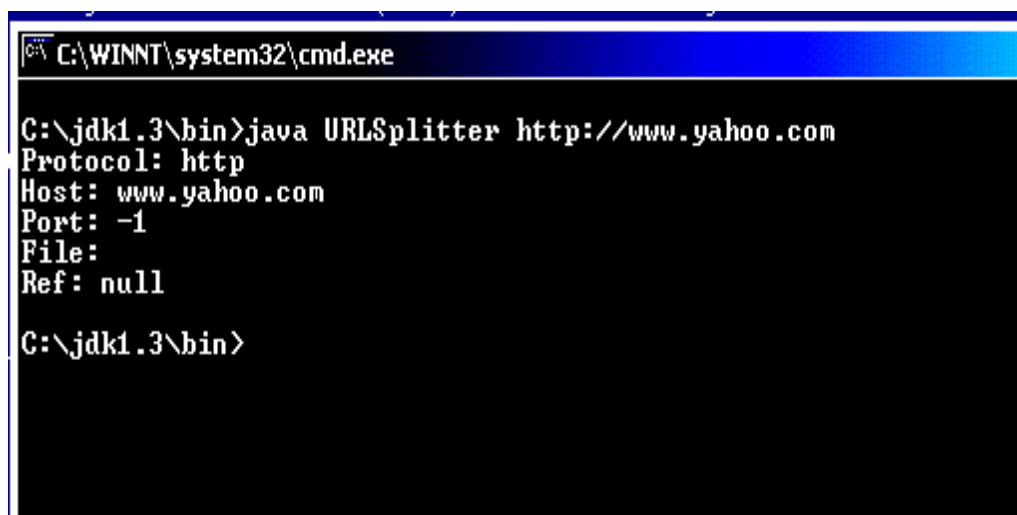
There are two types of Domain Name Servers: primary and secondary.

A primary server maintains a set of configuration files that contain information for the subset of the name space for which the server is authoritative. For example, the primary server for *indiatimes.com* contains IP addresses for all hosts in the *indiatimes.com* domain in configuration files. Resource Records are the entries in the configuration files that contain the actual data. A secondary server does not maintain any configuration files, but it copies the configuration files from the primary server in a process called a zone transfer. A secondary name server can respond to requests for name resolution, and it looks just like a primary name server from a user's perspective. Primary and secondary Domain Name Servers provide both performance and fault-tolerance benefits because you can split the workload between the servers, and if one goes down, the other can take over.

3)

```
import java.net.URL;
public class URLSplitter {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            try {
                java.net.URL u = new java.net.URL(args[i]);
                System.out.println("Protocol: " + u.getProtocol());
                System.out.println("Host: " + u.getHost());
                System.out.println("Port: " + u.getPort());
                System.out.println("File: " + u.getFile());
                System.out.println("Ref: " + u.getRef());
            }
            catch (java.net.MalformedURLException e) {
                System.err.println(args[i] + " is not a valid URL");
            }
        }
    }
}
```

Here's the output:



```
C:\WINNT\system32\cmd.exe

C:\jdk1.3\bin>java URLSplitter http://www.yahoo.com
Protocol: http
Host: www.yahoo.com
Port: -1
File:
Ref: null

C:\jdk1.3\bin>
```

## Check Your Progress 3

### Program to print the addresses:

```
1)
import java.net.*;
class I_Address
{
    public static void main(String[] args) throws UnknownHostException
    {
        InetAddress Addr = InetAddress.getLocalHost() ;
        System.out.println(Addr);
        Addr = InetAddress.getByName("yahoo.com");
        System.out.println(Addr);
        InetAddress Addr1[] = InetAddress.getAllByName("indiatimes.com") ;
        for( int i = 0; i < Addr1.length; i++)
            System.out.println(Addr1[i]);
    }
}
```

Output:

Run this program on your machine while connected to Internet to get proper output otherwise UnkwnHostException will occur.

Output on the machine is:

shashibhushan/190.10.19.205

rediff.com/208.184.138.70

webduniya.com/65.182.162.66

### 2) TCP Client/Server Interaction:

**The Server** starts by getting ready to receive client connections...

#### Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

#### Server

1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

3) In UDP, as you have read above, every time you send a datagram, you have to send the local descriptor and the socket address of the receiving socket along with it. Since TCP is a connection-oriented protocol, on the other hand, a connection must be established before communications between the pair of sockets start. So there is a connection setup time in TCP.

Once a connection is established, the pair of sockets behaves like streams: All available data are read immediately in the same order in which they are received.

UDP is an unreliable protocol- there is no guarantee that the datagrams you have sent will be received in the same order by the receiving socket. On the other hand, TCP is a reliable protocol; it is guaranteed that the packets you receive are put in the order in which they were sent.

In short, TCP is useful for implementing network services-such as remote login (rlogin, telnet) and file transfer (FTP)- which require data of indefinite length to be transferred. UDP is less complex and incurs fewer overheads. It is often used in



implementing client/server applications in distributed systems built over local area networks.

In client-server applications, the server provides some service, such as processing database queries or sending out current stock prices. The client uses the service provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it.

TCP provides a reliable, point-to-point communication channel those client-server applications use on the Internet to communicate with each other. To communicate over TCP, a client program and a server program establish a connection between them. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

- 4) The LocalPortScanner developed earlier only found TCP ports. The following program detects UDP ports in use. As with TCP ports, you must be root on Unix systems to bind to ports below 1024.

```
import java.net.*;
import java.io.IOException;
public class UDPPortScanner {
    public static void main(String[] args) {
        // first test to see whether or not we can bind to ports
        // below 1024
        boolean rootaccess = false;
        for (int port = 1; port < 1024; port += 50) {
            try {
                ServerSocket ss = new ServerSocket(port);
                // if successful
                rootaccess = true;
                ss.close();
                break;
            }
            catch (IOException ex) {
            }
        }
        int startport = 1;

        if (!rootaccess) startport = 1024;
        int stopport = 65535;

        for (int port = startport; port <= stopport; port++)
        {
            try {
                DatagramSocket ds = new DatagramSocket(port);
                ds.close();
            }
            catch (IOException ex) {
                System.out.println("UDP Port " + port + " is occupied.");
            }
        }
    }
}
```

Output

```
C:\WINNT\system32\cmd.exe

C:\jdk1.3\bin>java URLSplitter http://www.yahoo.com
Protocol: http
Host: www.yahoo.com
Port: -1
File:
Ref: null

C:\jdk1.3\bin>javac UDPPortScanner.java

C:\jdk1.3\bin>java UDPPortScanner
UDP Port 137 is occupied.
UDP Port 138 is occupied.
UDP Port 445 is occupied.
UDP Port 500 is occupied.
UDP Port 1027 is occupied.
UDP Port 1045 is occupied.
UDP Port 1072 is occupied.
UDP Port 4500 is occupied.

C:\jdk1.3\bin>
C:\jdk1.3\bin>
```

Since UDP is connectionless it is not possible to write a remote UDP port scanner. The only way you know whether or not a UDP server is listening on a remote port is if it sends something back to you.

---

## UNIT 4 ADVANCE JAVA

---

Structure	Page Nos.
4.0 Introduction	79
4.1 Objectives	79
4.2 Java Database Connectivity	80
4.2.1 Establishing A Connection	
4.2.2 Transactions with Database	
4.3 An Overview of RMI Applications	84
4.3.1 Remote Classes and Interfaces	
4.3.2 RMI Architecture	
4.3.3 RMI Object Hierarchy	
4.3.4 Security	
4.4 Java Servlets	88
4.4.1 Servlet Life Cycle	
4.4.2 Get and Post Methods	
4.4.3 Session Handling	
4.5 Java Beans	94
4.6 Summary	97
4.7 Solutions/Answers	97

---

### 4.0 INTRODUCTION

---

This unit will introduce you to the advanced features of Java. To save data, you have used the file system, which gives you functionality of accessing the data but it does not offer any capability for querying on data conveniently.

You are familiar with various databases like Oracle, Sybase, SQL Server etc. They do not only provide the file-processing capabilities, but also organize data in a manner that facilitates applying queries.

Structured Query Language (SQL) is almost universally used in relational database systems to make queries based on certain criteria. In this unit you will learn how you can interact to a database using **Java Database Connectivity (JDBC)** feature of Java.

You will also learn about RMI (Remote Method Invocation) feature of Java. This will give the notion of client/server distributed computing. **RMI** allows Java objects running on the same or separate computers to communicate with one another via remote method calls.

A request-response model of communication is essential for the highest level of networking. **The Servlets** feature of Java provides functionality to extend the capabilities of servers that host applications accessed via a request-response programming model. In this unit we will learn the basics of Servlet programming.

**Java beans** are nothing but small reusable pieces of components that you can add to a program without disturbing the existing program code, are also introduced in this unit.

---

### 4.1 OBJECTIVES

---

After going through of this unit you will be able to:

- interact with databases through java programs;
- use the classes and interfaces of the *java.sql* package;

- use basic database queries using Structured Query Language (SQL) in your programs;
- explain Servlet Life Cycle;
- write simple servlets programs;
- explain the model of client/server distributed computing;
- explain architecture of RMI, and
- describe Java Beans and how they facilitate component-oriented software construction.

---

## 4.2 JAVA DATABASE CONNECTIVITY

---

During programming you may need to interact with database to solve your problem. Java provides JDBC to connect to databases and work with it. Using standard library routines, you can open a connection to the database. Basically JDBC allows the integration of SQL calls into a general programming environment by providing library routines, which interface with the database. In particular, Java's JDBC has a rich collection of routines which makes such an interface extremely simple and intuitive.

### 4.2.1 Establishing A Connection

The first thing to do, of course, is to install Java, JDBC and the DBMS on the working machines. Since you want to interface with a database, you would need a driver for this specific database.

#### Load the vendor specific driver

This is very important because you have to ensure portability and code reuse. The API should be designed as independent of the version or the vendor of a database as possible. Since different DBMS's have different behaviour, you need to tell the driver manager which DBMS you wish to use, so that it can invoke the correct driver.

For example, an Oracle driver is loaded using the following code snippet:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")
```

#### Make the connection

Once the driver is loaded and ready for a connection to be made, you may create an instance of a Connection object using:

```
Connection con = DriverManager.getConnection(url, username, password);
```

Let us see what are these parameters passed to get Connection method of DriverManager class. The first string is the URL for the database including the protocol, the vendor, the driver, and the server the port number. The username and password are the name of the user of database and password is user password. The connection *con* returned in the last step is an open connection, which will be used to pass SQL statements to the database.

#### Creating JDBC Statements

A JDBC Statement object is used to send the SQL statements to the DBMS. It is entirely different from the SQL statement. A JDBC Statement object is an open connection, and not any single SQL Statement. You can think of a JDBC Statement object as a channel sitting on a connection, and passing one or more of the SQL statements to the DBMS.

An active connection is needed to create a Statement object. The following code is a snippet, using our Connection object *con*

```
Statement stmt = con.createStatement();
```

At this point, you will notice that a Statement object exists, but it does not have any SQL statement to pass on to the DBMS.

## Creating JDBC PreparedStatement

PreparedStatement object is more convenient and efficient for sending SQL statements to the DBMS. The main feature, which distinguishes PreparedStatement object from objects of Statement class, is that it gives an SQL statement right when it is created. This SQL statement is then sent to the DBMS right away, where it is compiled. Thus, in effect, a PreparedStatement is associated as a channel with a connection and a compiled SQL statement.

Another advantage offered by PreparedStatement object is that if you need to use the same or similar query with different parameters multiple times, the statement can be compiled and optimized by the DBMS just once. While with a normal Statement, each use of the same SQL statement requires a compilation all over again.

PreparedStatements are also created with a Connection method. The following code shows how to create a parameterized SQL statement with three input parameters:

```
PreparedStatement prepareUpdatePrice
= con.prepareStatement("UPDATE Employee SET emp_address=? WHERE
emp_code=? AND emp_name=?");
```

You can see two? symbol in the above PreparedStatement *prepareUpdatePrice*. This means that you have to provide values for two variables *emp\_address* and *emp\_name* in PreparedStatement before you execute it. Calling one of the setXXX methods defined in the class PreparedStatement can provide values. Most often used methods are setInt, setFloat, setDouble, setString, etc. You can set these values before each execution of the prepared statement.

You can write something like:

```
prepareUpdatePrice.setInt(1, 3);
prepareUpdatePrice.setString(2, "Renuka");
prepareUpdatePrice.setString(3, "101, Sector-8, Vasundhara, M.P");
```

## Executing CREATE/INSERT/UPDATE Statements of SQL

Executing SQL statements in JDBC varies depending on the intention of the SQL statement. DDL (Data Definition Language) statements such as table creation and table alteration statements, as well as statements to update the table contents, all are executed using the *executeUpdate* method. The following snippet has examples of executeUpdate statements.

```
Statement stmt = con.createStatement();
stmt.executeUpdate("CREATE TABLE Employee " +
"(emp_name VARCHAR2(40), emp_address VARCHAR2(40), emp_sal REAL)");
stmt.executeUpdate("INSERT INTO Employee " +
"VALUES ('Archana', '10, Down California', 30000)");
String sqlString = "CREATE TABLE Employee " +
"(name VARCHAR2(40), address VARCHAR2(80), license INT)";
stmt.executeUpdate(sqlString);
```

Since the SQL statement will not quite fit on one line on the page, you can split it into two or more strings concatenated by a plus sign(+).

"INSERT INTO Employee" to separate it in the resulting string from "VALUES".

The point to note here is that the same Statement object is reused rather than to create a new one each time.

When `executeUpdate` is used to call DDL statements, the return value is always zero, while data modification statement executions will return a value greater than or equal to zero, which is the number of tuples affected in the relation by execution of modification statement.

While working with a `PreparedStatement`, you should execute such a statement by first plugging in the values of the parameters (as you can see above), and then invoking the `executeUpdate` on it. For example:

```
int n = prepareUpdateEmployee.executeUpdate() ;
```

## Executing SELECT Statements

A query is expected to return a set of tuples as the result, and not change the state of the database. Not surprisingly, there is a corresponding method called `executeQuery`, which returns its results as a `ResultSet` object. It is a table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A `ResultSet` object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The *next method* moves the cursor to the next row, and because it returns false when there are no more rows in the `ResultSet` object, it can be used in a while loop to iterate through the result set. A default `ResultSet` object is not updatable and has a cursor that moves forward only

In the program code given below:

```
String ename,eaddress;  
float esal;  
  
ResultSet rs = stmt.executeQuery("SELECT * FROM Employee");  
while ( rs.next() ) {  
    ename = rs.getString("emp_name");  
    eaddress = rs.getString("emp_address");  
    esal = rs.getFloat("emp_salary");  
    System.out.println(ename + " address is" + eaddress + " draws salary " + esal + "  
in dollars");  
}
```

The tuples resulting from the query are contained in the variable `rs` which is an instance of `ResultSet`. A set is of not much use to you unless you can access each row and the attributes in each row. The?

Now you should note that each invocation of the *next method* causes it to move to the next row, if one exists and returns true, or returns false if there is no remaining row.

You can use the `getXXX` method of the appropriate type to retrieve the attributes of a row. In the above program code `getString` and `getFloat` methods are used to access the column values. One more thing you can observe that the name of the column whose value is desired is provided as a parameter to the method.

Similarly,while working with a `PreparedStatement`, you can execute a query by first plugging in the values of the parameters, and then invoking the `executeQuery` on it.

```
1. ename = rs.getString(1);  
   eaddress = rs.getFloat(3);  
   esal = rs.getString(2);
```

2. `ResultSet rs = prepareUpdateEmployee.executeQuery() ;`

## Accessing ResultSet

Now to reach each record of the database, JDBC provides methods like `getRow`, `isFirst`, `isBeforeFirst`, `isLast`, `isAfterLast` to access `ResultSet`. Also there are means to make scroll-able cursors to allow free access of any row in the `ResultSet`. By default, cursors scroll forward only and are read only. When creating a `Statement` for a `Connection`, we can change the type of `ResultSet` to a more flexible scrolling or updatable model:

```
Statement stmt = con.createStatement  
ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);  
ResultSet rs = stmt.executeQuery("SELECT * FROM Sells");
```

The different options for types are `TYPE_FORWARD_ONLY`, `TYPE_SCROLL_INSENSITIVE`, and `TYPE_SCROLL_SENSITIVE`. We can choose whether the cursor is read-only or updatable using the options `CONCUR_READ_ONLY`, and `CONCUR_UPDATABLE`.

With the default cursor, we can scroll forward using `rs.next()`. With scroll-able cursors we have more options:

```
rs.absolute(3);      // moves to the third tuple or row  
rs.previous();       // moves back one tuple (tuple 2)  
rs.relative(2);      // moves forward two tuples (tuple 4)  
rs.relative(-3);     // moves back three tuples (tuple 1)
```

### 4.2.2 Transactions with Database

When you go to some bank for deposit or withdrawal of money, you get your bank account updated, or in other words you can say some transaction takes place.

JDBC allows SQL statements to be grouped together into a single transaction. Thus, you can ensure the ACID (Atomicity, Consistency, Isolation, Durability) properties using JDBC transactional features.

The `Connection` object performs transaction control. When a connection is created, by default it is in the *auto-commit* mode. This means that each individual SQL statement is treated as a transaction by itself, and will be committed as soon as its execution is finished.

You can turn off *auto-commit* mode for an active connection with:

```
con.setAutoCommit(false) ;
```

And turn it on again if needed with:

```
con.setAutoCommit(true) ;
```

Once *auto-commit* is off, no SQL statements will be committed (that is, the database will not be permanently updated) until you have explicitly told it to commit by invoking the `commit ()` method:

```
con.commit() ;
```

At any point before commit, you may invoke `rollback ()` to rollback the transaction, and restore values to the last commit point (before the attempted updates).

## Check Your Progress 1

1) How is a program written in java to access database?

.....

.....

.....

- 2) What are the different kinds of drivers for JDBC?  
.....  
.....
- 3) Write a program code to show how you will perform commit() and rollback().  
.....  
.....
- 4) Read the following program assuming mytable already exists in database and answer the questions given below:
- What is the use of rs.next()?
  - Value of which attribute will be obtained by rs.getString(3).
  - If the statement "Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")" is removed from the program what will happen.

```
import java.sql.*;
import java.io.*;
public class TestJDBC
{
    public static void main(String[] args)
    {
        String dataSourceName = "mp";
        String dbURL = "jdbc:odbc:" + dataSourceName;
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(dbURL, "", "");
            Statement s = con.createStatement();
            s.execute("insert into mytable values('AKM',20,'azn')");
            s.executeQuery("select * from mytable ");
            ResultSet rs = s.getResultSet();
            rs.next();
            String n = rs.getString(1);
            System.out.println("Name:"+ n);
            if (rs != null)
                while ( rs.next() )
                {
                    System.out.println("Data from column_name: " + rs.getString(1) );
                    System.out.println("Data from column_age: " + rs.getInt(2) );
                    System.out.println("Data from column_address: " + rs.getString(3) );
                }
        }
        catch (Exception err)
        {
            System.out.println( "Error: " + err );
        }
    }
}
```

---

## 4.3 AN OVERVIEW OF RMI APPLICATIONS

---

Many times you want to communicate between two computers. One of the examples of this type of communication is a chatting program. How do chatting happens or two computers communicate each other? RPC (Remote Procedure Call) is one of the ways to perform this type of communication. In this section you will learn about RMI (Remote Method Invocation).



Java provides RMI (Remote Method Invocation), which is “*a mechanism that allows one to invoke a method on an object that exists in another address space. The other address space could be on the same machine or a different one. The RMI mechanism is basically an object-oriented RPC mechanism.*”

RPC (Remote Procedure Call) organizes the types of messages which an application can receive in the form of functions. Basically it is a management of streams of data transmission.

RMI applications often comprised two separate programs: *a server and a client*.

A typical server application creates some remote objects, makes references to them accessible, and waits for clients to invoke methods on these remote objects.

A typical client application gets a remote reference to one or more remote objects in the server and then invokes methods on them.

RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a *distributed object application*.

There are three processes that participate in developing applications based on remote method invocation.

1. The *Client* is the process that is invoking a method on a remote object.
2. The *Server* is the process that owns the remote object. The remote object is an ordinary object in the address space of the server process.
3. The *Object Registry* is a name server that relates objects with names. Objects are *registered* with the Object Registry. Once an object has been registered, one can use the Object Registry to obtain access to a remote object using the name of the object.

### 4.3.1 Remote Classes and Interfaces

A *Remote* class is one whose instances can be used remotely. An object of such a class can be referenced in two different ways:

1. Within the address space where the object was constructed, the object is an ordinary object, which can be used like any other object.
2. Within other address spaces, the object can be referenced using an object handle. While there are limitations on how one can use an object handle compared to an object, for the most part one can use object handles in the same way as an ordinary object.

For simplicity, an instance of a Remote class is called a *remote object*.

A Remote class has two parts: the interface and the class itself.

The Remote interface must have the following properties:

Interface must be public.

Interface must extend the *java.rmi.Remote* interface. Every method in the interface must declare that it throws *java.rmi.RemoteException*. Maybe other exceptions also ought to be thrown.

The Remote class itself has the following properties:

It must implement a Remote interface.

It should extend the *java.rmi.server.UnicastRemoteObject* class. Objects of such a class exist in the address space of the server and can be invoked remotely. While there are other ways to define a Remote class, this is the simplest way to ensure that objects of a class can be used as remote objects.

It can have methods that are not in its Remote interface. These can only be invoked locally. It is not necessary for both the Client and the Server to have access to the definition of the Remote class.

The Server requires the definition of both the Remote class and the Remote interface, but the client only uses the Remote interface.

All of the Remote interfaces and classes should be compiled using *javac*. Once this has been completed, the stubs and skeletons for the Remote interfaces should be compiled by using the *rmic stub* compiler. The stub and skeleton of the example Remote interface are compiled with the command:  
`rmic <filename.class>`

### 4.3.2 RMI Architecture

It consists of three layers as given in Figure 1

1. Stub/Skeleton layer – client-side stubs and server-side skeletons.
2. Remote reference layer-invocation to single or replicated object
3. Transport layer-connection set up and management, also remote object tracking.

#### JAVA RMI Architecture

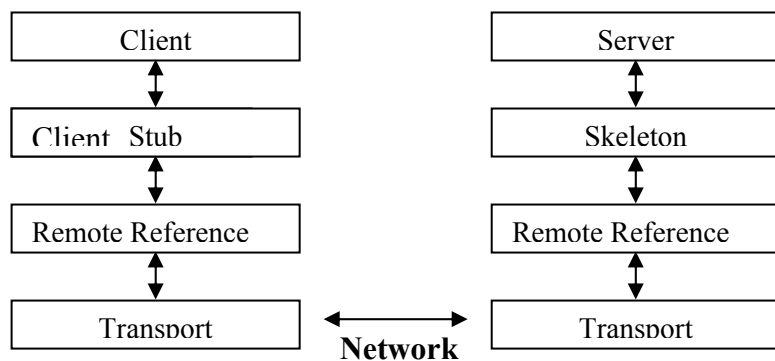


Figure 1: Java RMI Architecture

### 4.3.3 RMI Object Hierarchy

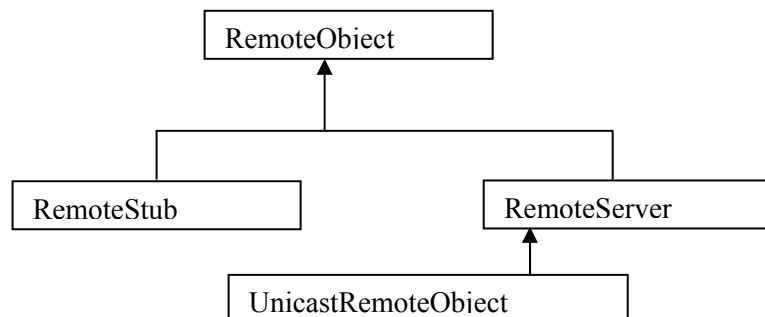


Figure 2:RemoteObject Hierarchy

In *Figure 2* above, the RMI Object Hierarchy is shown. The most general feature set associated with RMI is found in the `java.rmi.Remote` interface. Abstract class `java.rmi.server.RemoteObject` supports the needed modifications to the Java object model to cope with the indirect references.

Remote Server is a base class, which encapsulates transport semantics for RemoteObjects. Currently RMI ships with a `UnicastRemoteObject`(single object)

A server in RMI is a named service which is registered with the RMI registry, and listens for remote requests. For security reasons, an application can bind or unbind only in the registry running on the same host.

#### 4.3.4 Security

One of the most common problems with RMI is a failure due to security constraints. Let us see Java the security model related to RMI. A Java program may specify a security manager that determines its security policy. A program will not have any security manager unless one is specified. You can set the security policy by constructing a *SecurityManager* object and calling the *setSecurityManager* method of the *System* class. Certain operations require that there be a security manager. For example, RMI will download a Serializable class from another machine only if there is a security manager and the security manager permits the downloading of the class from that machine. The `RMISecurityManager` class defines an example of a security manager that normally permits such download. However, many Java installations have instituted security policies that are more restrictive than the default. There are good reasons for instituting such policies, and you should not override them carelessly.

### Creating Distributed Applications Using RMI

The following are the basic steps be followed to develop a distributed application using RMI:

- Design and implement the components of your distributed application.
- Compile sources and generate stubs.
- Make classes network accessible.
- Start the application.

#### Compile Sources and Generate Stubs

This is a two-step process. In the first step you use the `javac` compiler to compile the source files, which contain the implementation of the remote interfaces and implementations, of the server classes and the client classes. In the second step you use the `rmic` compiler to create stubs for the remote objects. RMI uses a remote object's stub class as a proxy in clients so that clients can communicate with a particular remote object.

#### Make Classes Network Accessible

In this step you have to make everything: the class files associated with the remote interfaces, stubs, and other classes that need to be downloaded to clients, accessible via a Web server.

#### Start the Application

Starting the application includes running the RMI remote object registry, the server, and the client.

## Check Your Progress 2

- 1) What is Stub in RMI?  
.....  
.....
- 2) What are the basic actions performed by receiver object on server side?  
.....  
.....
- 3) What is the need of and Registry Service of RMI?  
.....  
.....

---

## 4.4 JAVA SERVLETS

---

In this section you will be introduced to server side-programming. Java has utility known as servlets for server side-programming.

A *servlet* is a class of Java programming language used to extend the capabilities of servers that host applications accessed via a request-response programming model.

Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. Java Servlet technology also defines HTTP-specific servlet classes. The `javax.servlet` and `java.servlet.http` packages provide interfaces and classes for writing servlets. All servlets must implement the `Servlet` interface, which defines life-cycle methods.

When implementing a generic service, you can use or extend the `GenericServlet` class provided with the Java Servlet API. The `HttpServlet` class provides methods, such as `doGet` and `doPost`, for handling HTTP-specific services.

In this section we will focus on writing servlets that generate responses to HTTP requests. Here it is assumed that you are familiar with HTTP protocol.

### 4.4.1 Servlet Life Cycle

The container in which the servlet has been deployed controls the life cycle of a servlet. When a request is mapped to a servlet, the container performs the following steps.

Loads the servlet class.

Creates an instance of the servlet class.

Initializes the servlet instance by calling the `init()` method.

When servlet is executed it invokes the service method, passing a request and response object.

If the container needs to remove the servlet, it finalizes the servlet by calling the servlet's `destroy` method.

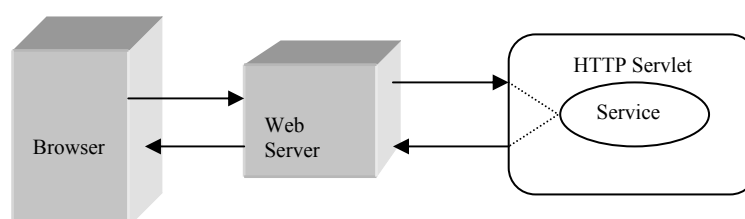


Figure 3: Interaction with Servlet

Servlets are programs that run on servers, such as a web server. You all do net surfing and well known the data on which the web is submitted and you get the respond accordingly. On web pages the data is retrieved from the corporate databases, which should be secure. For these kinds of operations you can use servlets.

#### 4.4.2 GET and POST Methods

The GET methods is a request made by browsers when the user types in a URL on the address line, follows a link from a Web page, or makes an HTML form that does not specify a METHOD. Servlets can also very easily handle POST requests, which are generated when someone creates an HTML form that specifies METHOD="POST".

The program code given below will give you some idea to write a servlet program:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SomeServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // Use "request" to read incoming HTTP headers (e.g. cookies)
        // and HTML form data (e.g. data the user entered and submitted)

        // Use "response" to specify the HTTP response line and headers
        // (e.g. specifying the content type, setting cookies).
        PrintWriter out = response.getWriter();
        // Use "out" to send content to browser
    }
}
```

To act as a servlet, a class should extend `HttpServlet` and override `doGet` or `doPost` (or both), depending on whether the data is being sent by GET or by POST. These methods take two arguments: an `HttpServletRequest` and an `HttpServletResponse` objects.

The `HttpServletRequest` has methods for information about incoming information such as FORM data, HTTP request headers etc.

The `httpServletResponse` has methods that let you specify the HTTP response line (200, 404, etc.), response headers (Content-Type, Set-Cookie, etc.), and, most importantly, a `PrintWriter` used to send output back to the client.

#### A Simple Servlet: Generating Plain Text

Here is a simple servlet that just generates plain text:

//Program file name: HelloWorld.java

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

public class HelloWorld extends HttpServlet
{
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}
```

## Compiling and Installing the Servlet

Note that the specific details for installing servlets vary from Web server to Web server. Please refer to the Web server documentation for definitive directions. The on-line examples are running on Java Web Server (JWS) 2.0, where servlets are expected to be in a directory called `servlets` in the JWS installation hierarchy.

You have to set the `CLASSPATH` to point to the directory above the one actually containing the servlets. You can then compile normally from within the directory.

```
DOS> set CLASSPATH=C:\JavaWebServer\servlets;%CLASSPATH%
DOS> cd C:\JavaWebServer\servlets\
DOS> javac HelloWorld.java
```

## Running the Servlet

With the Java Web Server, servlets are placed in the `servlets` directory within the main JWS installation directory, and are invoked via `http://host/servlet/ServletName`. Note that the directory is `servlets`, plural, while the URL refers to `servlet`, singular. Other Web servers may have slightly different conventions on where to install servlets and how to invoke them. Most servers also let you define aliases for servlets, so that a servlet can be invoked via `http://host/any-path/any-file.html`.

The Url that you will give on the explorer will be:

`http://localhost:8080/servlet/HelloWorld` then you will get the output as follows:



## A Servlet that Generates HTML

Most servlets generate HTML, not plain text as in the previous example. To do that, you need two additional steps: tell the browser that you're sending back HTML, and modify the `println` statements to build a legal Web page. First set the Content-Type response header. In general, headers can be set via the `setHeader` method of `HttpServletResponse`, but setting the content type is such a common task that there is also a special `setContentType` method just for this purpose. You need to set response headers *before* actually returning any of the content via the `PrintWriter`. Here is an example for the same:

```
//HelloWWW.java
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWWW extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
                    \"Transitional//EN\">\n" +
                    "<HTML>\n" +
```

```

"<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
"<BODY>\n" +
"<H1>Hello WWW</H1>\n" +
"</BODY></HTML>");
}
}
URL: http://localhost:8080/servlet/HelloWWW
HelloWWW Output:

```



### 4.4.3 Session Handling

It is essential to track client's requests. To perform this task, Java servlets offers two different ways:

1. It is possible to save information about client state on the server using a *Session* object
2. It is possible to save information on the client system using cookies.

HTTP is a **stateless protocol**. If a client makes a **series** of requests on a server, HTTP provides no help whatsoever to determine if those requests originated from the **same** client. There is no way in HTTP to link two separate requests to the same client. Hence, there is no way to maintain state between client requests in HTTP.

*A **session** is sequence of HTTP requests, from the same client, over a period of time.*

You need to maintain the state on the web for e-commerce type of applications. Just like other software systems, web applications want and need state. The classic web application example is the shopping cart that maintains a list of items you wish to purchase at a web site. The shopping cart's state is the items in the shopping basket at any given time. This state, or shopping items, needs to be maintained over a series of client requests. HTTP alone cannot do this; it needs help.

Now the question arises, for how long can you maintain the state of the same client? Of course, this figure is application-dependent and brings into play the concept of a web session. If a session is configured to last for 30 minutes, once it has expired the client will need to start a new session. Each session requires a unique identifier that can be used by the client.

There are various ways through which you maintain the state.

- Hidden Form Fields
- URL Rewriting
- Session Handling
- Cookies.

### Hidden Form Fields

Hidden form fields are HTTP tags that are used to store information that is invisible to the user. In terms of session tracking, the hidden form field would be used to hold a client's unique session id that is passed from the client to the server on each HTTP request. This way the server can extract the session id from the submitted form, like it does for any of form field, and use it to identify which client has made the request and act accordingly.

For example, using servlets you could submit the following search form:

```
<form method="post" action="/servlet/search">
  <input type="text" name="searchtext">
  <input type="hidden" name="sessionid" value="1211xyz">
  ...
</form>
```

When it is submitted to the servlet registered with the name **search**, it pulls out the sessionid from the form as follows:

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
{
    ...
    String theSessionId = request.getParameterValue("sessionid");
    if( isAllowToPerformSearch(theSessionId) )
    {
        ...
    }
    ...
}
```

In this approach' the search servlet gets the session id from the hidden form field and uses it to determine whether it allows performing any more searches.

Hidden form fields implement the required **anonymous** session tracking features the client needs but not without cost. For hidden fields to work the client must send a hidden form field to the server and the server must always return that same hidden form field. This tightly coupled dependency between client requests and server responses requires sessions involving hidden form fields to be an unbreakable chain of dynamically generated web pages. If at any point during the session the client accesses a static page that is not point of the chain, the hidden form field is lost, and with it the session is also lost.

## URL Rewriting

URL rewriting stores session details as part of the URL itself. You can see below how we look at request information for our search servlet:

- i) <http://www.archana.com/servlet/search>
- ii) <http://www.archana.com/servlet/search/23434abc>
- iii) <http://www.archana.com/servlet/search?sessionid=23434abc>

For the original servlet [i] the URL is clean. In [ii] we have URL re-written at the server to add extra path information as embedded links in the pages we send back to the client. When the client clicks on one of these links, the search servlet will do the following:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
{
    ...
    String sessionid = request.getPathInfo(); // return 2343abc from [ii]
    ...
}
```

Extra path information work for both GET and POST methods involved from inside as well as outside of forms with static links.



Technique [iii] simply re-writes the URL with parameter information that can be accessed as follows:

```
request.getParameterValue("sessionid");
```

URL re-writing, like, hidden forms provide a means to implement anonymous session tracking. However, with URL rewriting you are not limited to forms and you can re-write URLs in static documents to contain the required session information. But URL re-writing suffers from the same major disadvantage that hidden form fields do, in that they must be dynamically generated and the chain of HTML page generation cannot be broken.

## Session object

A *HttpSession* object (derived from *Session* object) allows the servlet to solve part of the HTTP stateless protocol problems. After its creation a *Session* is available until an explicit invalidating command is called on it (or when a default timeout occurs). The same *Session* object can be shared by two or more cooperating servlets. This means each servlet can track client's service request history.

A servlet accesses a *Session* using the *getSession()* method implemented in the *HttpServletRequest* interface.

*getSession()* method returns the *Session* related to the current *HttpServletRequest*.

### Note:

1. It is important to remember that each instance of *HttpServletRequest* has its own *Session*. If a *Session* object has not been created before, *getSession()* creates a new one.
2. *HttpSession* interface implements necessary methods to manage with a session.

### Following are the various methods related to session tracking:

public abstract String getId( ): Returns a string containing session's name. This name is unique and is set by *HttpSessionContext()*.

public abstract void putValue (String Name, Object Value): Connect the object *Value* to the *Session* object identified by *Name* parameter. If another object has been connected to the same session before, it is automatically replaced.

public abstract Object getValue (String name): Returns the object currently held by current session. It returns a *null* value if no object has been connected before to the session.

public abstract void removeValue (String name): Remove, if existing, the object connected to the session identified by the *Name* parameter.

public abstract void invalidate( ): Invalidate the session.

## Cookies

Using JSDK (Java Servlet Development Kit) it is possible to save client's state sending cookies. Cookies are sent from the server and saved on client's system. On client's system cookies are collected and managed by the web browser. When a cookie is sent, the server can retrieve it in a successive client's connection. Using this strategy it is possible to track client's connections history.

*Cookie* class of Java is derives directly from the *Object* class. Each *Cookie* object instance has some attributes like max age, version, server identification, comment.

A cookie is a text file with a name, a value and a set of attributes.

Below are some cookie methods:

`public Cookie (String Name, String Value)`: Cookie class' constructor. It has two parameters. The first one is the name that will identity the cookie in the future; the second one, *Value*, is a text representing the cookie value. Notice that *Name* parameter must be a "token" according to the standard defined in RFC2068 and RFC2109.

`Public String getName( )`: Returns cookie's name. A cookie name is set when the cookie is created and can't be changed.

`public void setValue(String NewValue)`: This method can be used to set or change cookie's value.

`public String getValue( )`: Returns a string containing cookie's value.

`public void setComment(Sting Comment)`: It is used to set cookie's comment attribute.

`public String getComment( )`: Returns cookie's comment attribute as a string.

`public void setMaxAge (int MaxAge)`: Sets cookie's max age in seconds. This means client's browser will delete the cookie in *MaxAge* seconds. A negative value indicate the cookie has to be deleted when client's web browser exits.

`public int getMaxAge( )`: Returns cookie's max age.

### Check Your Progress 3

- 1) What are the advantages of Servlets?  
.....  
.....
- 2) What is session tracking?  
.....  
.....
- 3) What is the difference between doGet() and doPost()?  
.....  
.....
- 4) How does HTTP Servlet handle client requests?  
.....  
.....

---

## 4.5 JAVA BEANS

---

Java Beans are reusable software component model which allow a great flexibility and addition of features in the existing piece of software. You will find it very interesting and useful to use them by linking together the components to create applets or even new beans for reuse by others. Graphical programming and design environments often called builder tools give a good visual support to bean programmers. The builder tool does all the work of associating of various components together.

A “JavaBeans-enabled” builder tool examines the Bean’s patterns, discern its features, and exposes those features for visual manipulation. A builder tool maintains Beans in a palette or toolbox. You can select a Bean from the toolbox, drop it into a form, modify its appearance and behaviour, define its interaction with other Beans, and compose it into applets, application, or new Bean. All this can be done without writing a line of code.

**Definition:** *A Java Bean is a reusable software component that can be visually manipulated in builder tools*

To understand the precise meaning of this definition of a Bean, you must understand the following terms:

- Software component
- Builder tool
- Visual manipulation.

Reusable software components are designed to apply the power and benefit of reusable, interchangeable parts from other industries to the field of software construction. Other industries have long profited from reusable components. Reusable electronic components are found on circuit boards. A typical part in your car can be replaced by a component made from one of many different competing manufacturers. Lucrative industries are built around parts construction and supply in most competitive fields. The idea is that standard interfaces allow for interchangeable, reusable components.

Reusable software components can be simple like familiar push buttons, text fields list boxes, scrollbars, dialogs boxes etc.

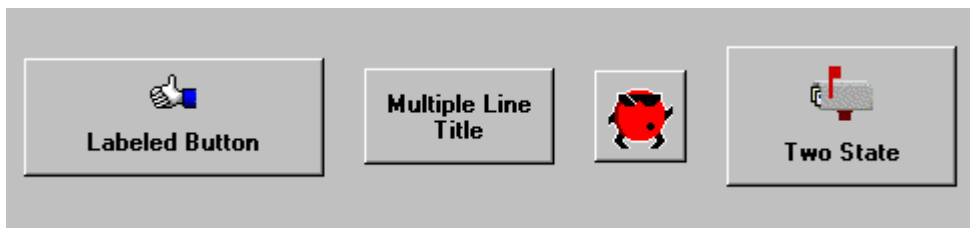


Figure 4: Button Beans

## Features of JavaBeans

- Individual Java Beans will vary in functionality, but most share certain common defining features.
- Support for **introspection** allowing a builder tool to analyze how a bean works.
- Support for **customization** allowing a user to alter the appearance and behaviour of a bean.
- Support for **events** allowing beans to fire events, and informing builder tools about both the events they can fire and the events they can handle.
- Support for **properties** allowing beans to be manipulated programmatically, as well as to support the customization mentioned above.
- Support for **persistence** allowing beans that have been customized in an application builder to have their state saved and restored. Typically persistence is used with an application builder’s save and load menu commands to restore any work that has gone into constructing an application.

It is not essential that Beans can only be primarily with builder tools. Beans can also be manually manipulated by programmatic interfaces of Text tools. All key APIs,

including support for events, properties, and persistence, have been designed to be easily read and understood by human programmers as well as by builder tools.

## **BeanBox**

BeanBox is a utility from the JavaBeans Development Kit (*BDK*). Basically the BeanBox is a test container for your JavaBeans. It is designed to allow programmers to preview how a Bean created by user will be displayed and manipulated in a builder tool. The BeanBox is not a builder tool. It allows programmers to preview how a bean will be displayed and used by a builder tool.

## **Example of Java Bean Class**

Create a new SimpleBean.java program containing the code given below:

```
/WEB-INF/classes/com/myBean/bean/test/ folder

package com.mybean.bean.test;
public class SimpleBean implements java.io.Serializable
{
    /* Properties */
    private String ename = null;
    private int eage = 0;
    /* Empty Constructor */
    public SimpleBean() {}
    /* Getter and Setter Methods */
    public String getEname()
    {
        return ename;
    }
    public void setEname(String s)
    {
        ename = s;
    }
    public int getAge()
    {
        return eage;
    }

    public void setAge(int i)
    {
        eage = i;
    }
}
```

The class SimpleBean implements java.io.Serializable interface.

There are two variables which hold the name and age of an employee. These variables inside a JavaBean are called properties. These properties are private and are thus not directly accessible by other classes. To make them accessible, methods are defined.

## **Compiling JavaBean**

You can compile JavaBean like you compile any other Java Class file. After compilation, a SimpleBean.class file is created and is ready for use. Finally you can say, JavaBeans are Java classes which adhere to an extremely simple coding convention. All you have to do is to implement java.io.Serializable interface,

use a public empty argument constructor and provide public methods to get and set the values of private variables (properties).

### Check Your Progress 4

- 1) What are JavaBeans?  
.....  
.....
- 2) What do you understand by Introspection?  
.....  
.....
- 3) What is the difference between a JavaBean and an instance of a normal Java class?  
.....  
.....
- 4) In a single line answer what is the main responsibility of a Bean Developer.  
.....  
.....

---

## 4.6 SUMMARY

---

In this unit you have learn Java JDBC are used to connect databases and work with it. JDBC allows the integration of SQL call into a general programming environment. Vender specific drivers are needed in JDBC programming to make a code portable. getConnection() method of DriverManager class is used to create connection object. By PreparedStatement similar queries can be performed in efficient way. Tuples in ResultSet are accessed by using next () method.

Distributed programming can be done using Java RMI (Remote Methods Invocation). Every RMI program has two sets one for client side and other for server side. Remote interface is essentially implemented in RMI programs.

Servlets are used with web servers. The HttpServlet class is an extension of GenericServlet that include methods for handling HTTP. HTTP request for specific data are handled by using doGet () and doPost () methods of HttpServlet. HttpSession objects are used to solve the problems of HTTP caused due to the stateless nature of HTTP.

Java Beans are a new dimension in software component model. Beans provide introspection and persistency.

---

## 4.7 SOLUTIONS/ANSWERS

---

### Check Your Progress 1

- 1) Programs are written according to the JDBC driver API would talk to the JDBC driver Manager. JDBC driver Manager would use the drivers that were plugged into it at that moment to access the actual database.
- 2) Four types of drivers are there for JDBC. They are:
  - a) JDBC-ODBC Bridge Driver: Talks to an ODBC connection using largely non-Java code.

- b) Native API,(Partly Java) : Uses foreign functions to talk to a non-Java API; the non-Java component talks to the database any way it likes.(Written partly in Java and partly in native code, that communicate with the native API of a database).
  - c) Net Protocol (pure Java): Talks to a middleware layer over a network connection using the middleware's own protocol (Client library is independent of the actual database).
  - d) Native Protocol (pure Java): Talks directly to the RDBMS over a network connection using an RDBMS-specific protocol. (pure Java library that translates JDBC requests directly to a database-specific protocol).
- 3) //It is assumed that Employee database is already existing.
- ```
con.setAutoCommit(false);  
The Statement stmt = con.createStatement();  
stmt.executeUpdate("INSERT INTO Employee VALUES('Archana', 'xyz',  
30000)");  
con.rollback();  
stmt.executeUpdate("INSERT INTO Sells Employee('Archie', 'ABC', 40000)");  
con.commit();  
con.setAutoCommit(true);
```
- 4)
- i. rs.next() is used to move to next row of the table.
  - ii. rs.getString(3) will give the value of the third attribute in the current row. In this program the third attribute is Address.
  - iii. The statement "Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")" is essential for the program execution. Database driver cannot be included in JDBC runtime method. If you want to remove this statement from program you must have to provide jdbc.drivers property using command line parameter.

## **Check Your Progress 2**

- 1) When you invoke a remote method on a remote object' the remote method calls a method of java programming language that is encapsulated in a surrogate object called Stub. The Following information is built by Stub:
  - i. An identifier of the remote object to be used.
  - ii. A description of the method to be called.
  - iii. The marshalled parameters.
- 2) The basic actions performed by receiver object on server side are:
  - i. Unmarshaling of the parameters.
  - ii. Locating the object to be called.
  - iii. Calling the desired method
  - iv. Capturing the marshals and returning the value or exception of the call.
  - v. Sending a package consisting of the marshalled return data back to the stub on the client.
- 3) RMI Registry is required to provide RMI Naming Service which is used to simplify the location of remote objects. The naming service is a JDK utility called rmiregistry that runs at a well-known address.

- 1) Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI than many alternative CGI-like technologies.

### Following are the advantages of Services

**Efficient.** With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are  $N$  simultaneous requests to the same CGI program, then the code for the CGI program is loaded into memory  $N$  times. With servlets, however, there are  $N$  threads but only a single copy of the servlet class.

**Convenient.** Servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.

**Powerful.** Java servlets let us easily do several things that are difficult or impossible with regular CGI. For example servlets can talk directly to the Web server (regular CGI programs can't). This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement.

**Portable.** Servlets are written in Java and follow a well-standardized API. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or WebStar. Servlets are supported directly or via a plugin on almost every major Web server.

- 2) Session tracking is a concept which allows you to maintain a relationship between two successive requests made to a server on the Internet by the same client. Servlet's Provide an API named HttpSession is used in session tracking programming.
- 3)
  - i. The doGet() method is limited with 2k of data only to be sent, but this limitation is not with doPost() method.
  - ii. A request string for doGet() looks like the following:

`http://www.abc.com/svt1?p1=v1&p2=v2&...&pN=vN`

But doPost() method does not need a long text tail after a servlet name in a request.

- 4) An HTTP Servlet handles client requests through its service method, which supports standard HTTP client requests. The service method dispatches each request to a method designed to handle that request.

## Check Your Progress 4

- 1) Java Beans are components that can be used to assemble a larger Java application. Beans are basically classes that have properties, and can trigger events. To define a property, a bean writer provides accessor methods which are used to get and set the value of a property.
- 2) Introspection is the process of implicitly or explicitly interrogating Bean.  
**Implicit Introspection:** Bean runtime supplies the default introspection mechanism which uses the Reflection API and a well established set of Naming Conventions.

**Explicit Introspection:** A bean designer can provide additional information through an object which implements the Bean Info interface.

In a nutshell, Introspection is a how a builder or designer can get information about how to connect a Bean with an Application.

- 3) The difference in Beans from typical Java classes is **introspection**. Tools that recognize predefined patterns in method signatures and class definitions can “look inside” a Bean to determine its properties and behavior. A Bean’s state can be manipulated at the time it is being assembled as a part within a larger application. The application assembly is referred to as **design time** in contrast to **run time**. In order for this scheme to work, method signatures within Beans must follow a certain pattern for introspection tools to recognize how Beans can be manipulated, both at design time, and run time.
- 4) To minimize the effort in turning a component into a Bean.