UNIT 1 APPLETS

Structure **Page Nos.**

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

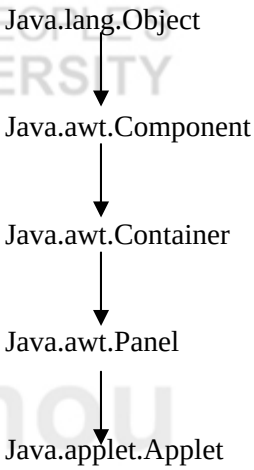
Our objective is to introduce you to Java Applets. After going through this unit, you will be able to:

- 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

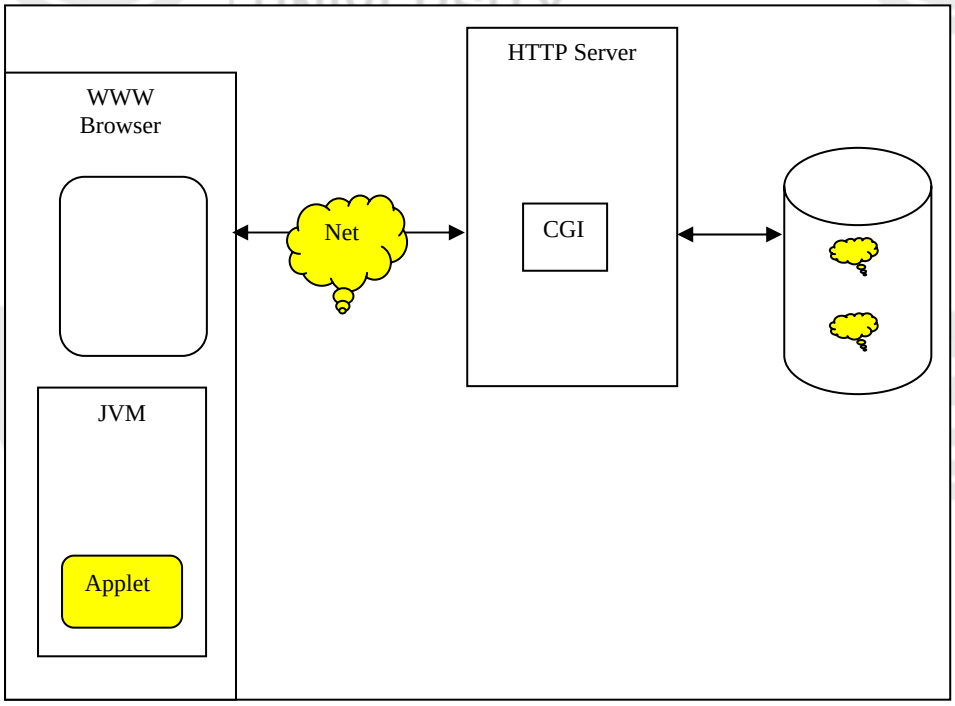


Figure1: Applet Architecture

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 APPLET 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 star t() 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 handle Event () 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
- AjdustmentListener
- ItemListener
- TextListener

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

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

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.

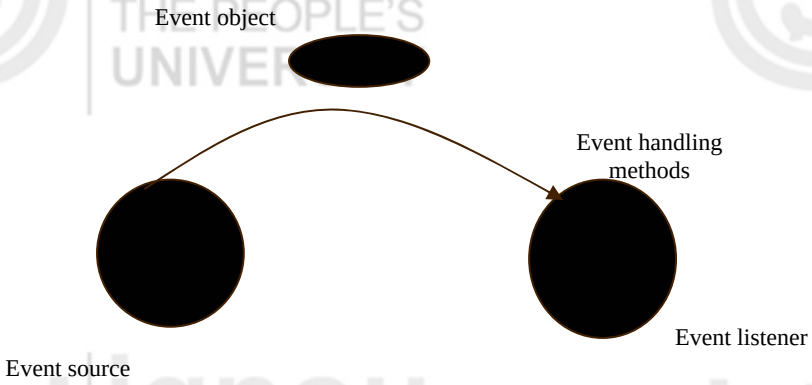


Figure 2: Event Handling Model of AWT

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 APPLET 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

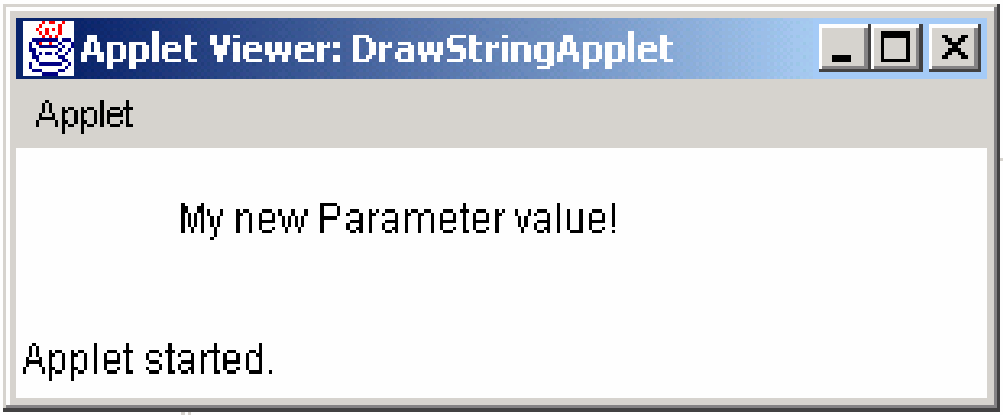
Parameters are passed to Applets in NAME and VALUE attribute pairs in <PARAM> tags between the opening and closing APPLET tags. Inside the Applet, you read the values passed through the PARAM tags with the getParameter() method of the Java.Applet.Applet class.

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">
```

<PARAM name="param2" value="Here I am ">

Check Your Progress 3

1) Is it possible to access the network resources through an Applet on the browser?

……

……

……

……

……

……

……

2) Write a program in which whenever you click the mouse on the frame, the coordinates of the point on which the mouse is clicked are displayed on the screen.

……

……

……

……

……

……

……

……

……

3) Write an Applet program in which you place a button and a textarea. When you click on button, in text area Your name and address is displayed. You have to take your name and address using < PARAM>.

……

……

……

……

……

.

……

……

1.7 SUMMARY

In this unit we have discussed Applet programs. Applets are secure programs that run inside a web browser and it's a subclass of Java.Applet. Applet.or its an instance of subclass of Java.Applet. Applet goes through five phases such as:

start(), init(),paint(),stop() and destroy(). paint() is one of the three methods that are guaranteed to be called automatically for you when any Applet begins execution.

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 sub classed 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 Invokation (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.