

Applet Programming

KEY TERMS

Applet | Local applet | Remote applet | Web page | HTML tag | APPLET tag | Applet life cycle

14.1 INTRODUCTION

Applets are small Java programs that are primarily used in Internet computing. They can be transported over the Internet from one computer to another and run using the **Applet Viewer** or any Web browser that supports Java. An applet, like any application program, can do many things for us. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, and play interactive games.

Java has revolutionized the way the Internet users retrieve and use documents on the world wide network. Java has enabled them to create and use fully interactive multimedia Web documents. A web page can now contain not only a simple text or a static image but also a Java applet which, when run, can produce graphics, sounds and moving images. Java applets therefore have begun to make a significant impact on the World Wide Web.

Local and Remote Applets

We can embed applets into Web pages in two ways. One, we can write our own applets and embed them into Web pages. Second, we can download an applet from a remote computer system and then embed it into a Web page.

An applet developed locally and stored in a local system is known as a *local applet*. When a Web page is trying to find a local applet, it does not need to use the Internet and therefore the local system does not require the Internet connection. It simply searches the directories in the local system and locates and loads the specified applet (see Fig. 14.1).

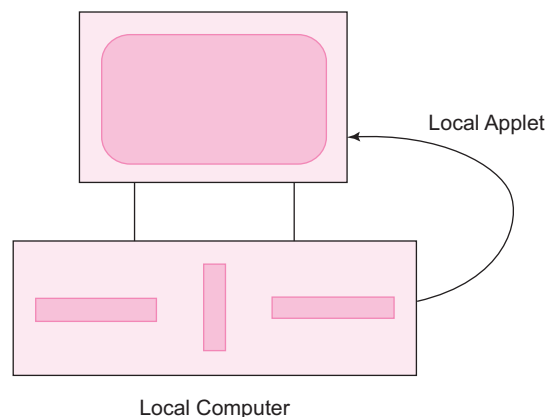


Fig. 14.1 Loading local applets

A *remote applet* is that which is developed by someone else and stored on a remote computer connected to the Internet. If our system is connected to the Internet, we can download the remote applet onto our system via at the Internet and run it (see Fig. 14.2).

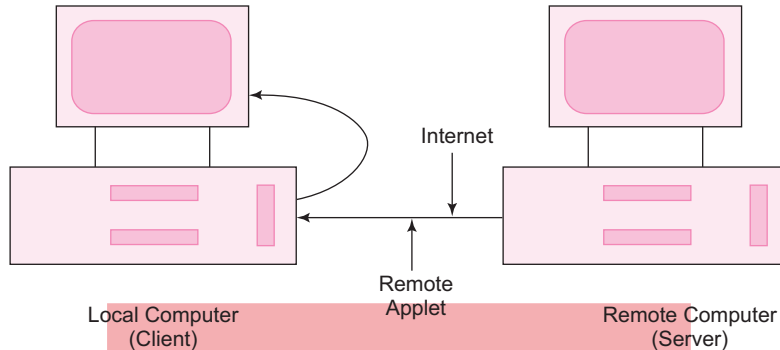


Fig. 14.2 Loading a remote applet

In order to locate and load a remote applet, we must know the applet's address on the Web. This address is known as *Uniform Resource Locator (URL)* and must be specified in the applet's HTML document as the value of the CODEBASE attribute (see Section 14.11). Example:

```
CODEBASE = http : // www.netserve.com / applets
```

In the case of local applets, CODEBASE may be absent or may specify a local directory. In this chapter, we shall discuss how applets are created, how they are located in the Web documents and how they are loaded and run in the local computer.

14.2 HOW APPLETS DIFFER FROM APPLICATIONS

Although both the applets and stand-alone applications are Java programs, there are significant differences between them. Applets are not full-featured application programs. They are usually written to accomplish a small task or a component of a task. Since they are usually designed for use on the Internet, they impose certain limitations and restrictions in their design.

1. Applets do not use the **main()** method for initiating the execution of the code. Applets, when loaded, automatically call certain methods of Applet class to start and execute the applet code.
2. Unlike stand-alone applications, applets cannot be run independently. They are run from inside a Web page using a special feature known as HTML tag.
3. Applets cannot read from or write to the files in the local computer.
4. Applets cannot communicate with other servers on the network.
5. Applets cannot run any program from the local computer.
6. Applets are restricted from using libraries from other languages such as C or C++. (Remember, Java language supports this feature through **native** methods).

All these restrictions and limitations are placed in the interest of security of systems. These restrictions ensure that an applet cannot do any damage to the local system.

14.3 PREPARING TO WRITE APPLETS

Until now, we have been creating simple Java application programs with a single **main()** method that created objects, set instance variables and ran methods. Here, we will be creating applets exclusively and therefore we will need to know

- When to use applets,
- How an applet works,
- What sort of features an applet has, and
- Where to start when we first create our own applets.

First of all, let us consider the situations when we might need to use applets.

1. When we need something dynamic to be included in the display of a Web page. For example, an applet that displays daily sensitivity index would be useful on a page that lists share prices of various companies or an applet that displays a bar chart would add value to a page that contains data tables.
2. When we require some “flash” outputs. For example, applets that produce sounds, animations or some special effects would be useful when displaying certain pages.
3. When we want to create a program and make it available on the Internet for us by others on their computers.

Before we try to write applets, we must make sure that Java is installed properly and also ensure that either the Java **appletviewer** or a Java-enabled browser is available. The steps involved in developing and testing in applet are:

1. Building an applet code (.java file)
2. Creating an executable applet (.class file)
3. Designing a Web page using HTML tags
4. Preparing <APPLET> tag
5. Incorporating <APPLET> tag into the Web page
6. Creating HTML file
7. Testing the applet code

Each of these steps is discussed in the following sections.

14.4 BUILDING APPLET CODE

It is essential that our applet code uses the services of two classes, namely, **Applet** and **Graphics** from the Java class library. The **Applet** class which is contained in the **java.applet** package provides life and behavior to the applet through its methods such as **init()**, **start()** and **paint()**. Unlike the applications, where Java calls the **main()** method directly to initiate the execution of the program, when an applet is loaded, Java automatically calls a series of **Applet** class methods for starting, running, and stopping the applet code. The **Applet** class therefore maintains the *lifecycle* of an applet.

The **paint()** method of the **Applet** class, when it is called, actually displays the result of the applet code on the screen. The output may be text, **graphics**, or sound. The **paint()** method, which requires a **Graphics** object as an argument, is defined as follows:

```
public void paint (Graphics g)
```

This requires that the applet code imports the **java.awt** package that contains the **Graphics** class. All output operations of an applet are performed using the methods defined in the **Graphics** class. It is thus clear from the above discussions that an applet code will have a general format as shown below.

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

.....
.....
public class appletclassname extends Applet
{
    .....
```

```

.....
public void paint (Graphics g)
{
    .....
    .....
    .....
}
.....
.....
}

```

// Applet operations code

The *appletclassname* is the main class for the applet. When the applet is loaded, Java creates an instance of this class, and then a series of **Applet** class methods are called on that instance to execute the code. Program 14.1 shows a simple HelloJava applet.

Program 14.1 *The HelloJava applet*

```

import java.awt.*;
import java.applet.*;
public class HelloJava extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString "Hello Java", 10, 100);
    }
}

```

The applet contains only one executable statement.

```
g.drawString("Hello Java", 10, 100);
```

which, when executed, draws the string

Hello Java

at the position 10, 100 (pixels) of the applet's reserved space as shown in Fig. 14.3.

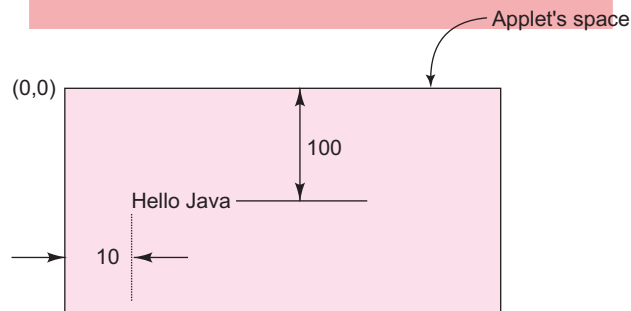


Fig. 14.3 *Output of Program 14.1*

Remember that the applet code in Program 14.1 should be saved with the file name **Hello Java.java**, in a java subdirectory. Note the **public** keyword for the class **HelloJava**. Java requires that the main applet class be declared public.

Remember that **Applet** class itself is a subclass of the **Panel** class, which is again a subclass of the **Container** class and so on as shown in Fig. 14.4. This shows that the main applet class inherits properties from a long chain of classes. An applet can, therefore, use variables and methods from all these classes.

14.5 APPLET LIFE CYCLE

Every Java applet inherits a set of default behaviors from the **Applet** class. As a result, when an applet is loaded, it undergoes a series of changes in its state as shown in Fig. 14.5. The applet states include:

1. Born on initialization state
2. Running state
3. Idle state
4. Dead or destroyed state

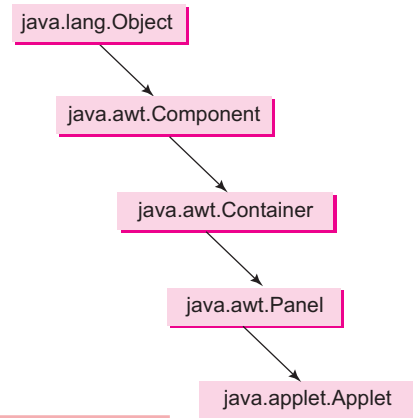


Fig. 14.4 Chain of classes inherited by Applet class

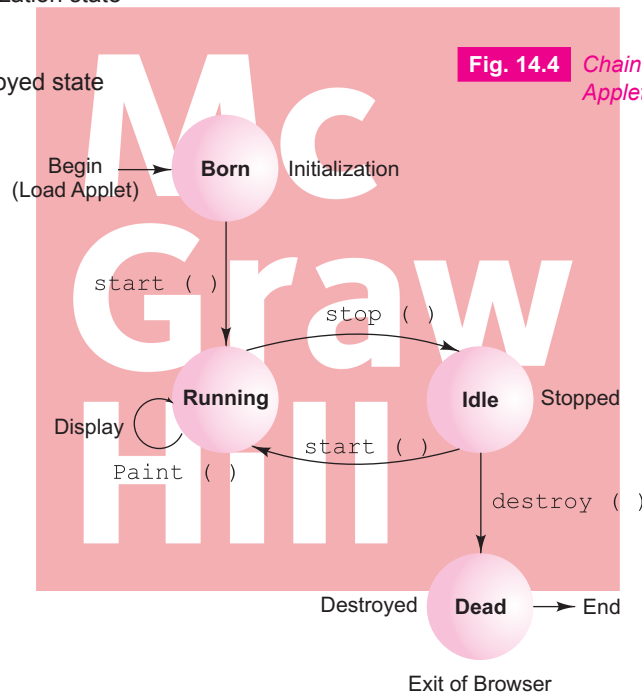


Fig. 14.5 An applet's state transition diagram

Initialization State

Applet enters the *initialization* state when it is first loaded. This is achieved by calling the **init()** method of **Applet** Class. The applet is born. At this stage, we may do the following, if required.

1. Create objects needed by the applet
2. Set up initial values
3. Load images or fonts
4. Set up colors

The initialization occurs only once in the applet's life cycle. To provide any of the behaviors mentioned above, we must override the **init()** method:

```

public void init( )
{
    .....
    ..... (Action)
}

```

Running State

Applet enters the **running** state when the system calls the **start()** method of **Applet** Class. This occurs automatically after the applet is initialized. Starting can also occur if the applet is already in “stopped” (idle) state. For example, we may leave the Web page containing the applet temporarily to another page and return back to the page. This again starts the applet running. Note that, unlike **init()** method, the **start()** method may be called more than once. We may override the **start()** method to create a thread to control the applet.

```

public voids start( )
{
    .....
    .....
    ..... (Action)
}

```

Idle or Stopped State

An applet becomes *idle* when it is stopped from running. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the **stop()** method explicitly. If we use a thread to run the applet, then we must use **stop()** method to terminate the thread. We can achieve this by overriding the **stop()** method;

```

public void stop( )
{
    .....
    ..... (Action)
    .....
}

```

Dead State

An applet is said to be *dead* when it is removed from memory. This occurs automatically by invoking the **destroy()** method when we quit the browser. Like initialization, destroying stage occurs only once in the applet’s life cycle. If the applet has created any resources, like threads, we may override the **destroy()** method to clean up these resources.

```

public void destroy ( )
{
    .....
    ..... (Action)
    .....
}

```

Display State

Applet moves to the *display* state whenever it has to perform some output operations on the screen. This happens immediately after the applet enters into the running state. The **paint()** method is called to accomplish this task. Almost every applet will have a **paint()** method. Like other methods in the life

cycle, the default version of **paint()** method does absolutely nothing. We must therefore override this method if we want anything to be displayed on the screen.

```
public void paint (Graphics g)
{
    .....
    .....      (Display statements)
    .....
}
```

It is to be noted that the display state is not considered as a part of the applet's life cycle. In fact, the **paint()** method is defined in the **Applet** class. It is inherited from the **Component** class, a super class of **Applet**.

14.6 CREATING AN EXECUTABLE APPLLET

Executable applet is nothing but the **.class** file of the applet, which is obtained by compiling the source code of the applet. Compiling an applet is exactly the same as compiling an application. Therefore, we can use the Java compiler to compile the applet.

Let us consider the **HelloJava** applet created in Section 14.4. This applet has been stored in a file called **HelloJava.java**. Here are the steps required for compiling the **HelloJava** applet.

1. Move to the directory containing the source code and type the following command:

```
javac HelloJava.java
```

2. The compiled output file called **HelloJava.class** is placed in the same directory as the source.
3. If any error message is received, then we must check for errors, correct them and compile the applet again.

14.7 DESIGNING A WEB PAGE

Recall the Java applets are programs that reside on Web pages. In order to run a Java applet, it is first necessary to have a Web page that references that applet.

A Web page is basically made up of text and HTML tags that can be interpreted by a Web browser or an applet viewer. Like Java source code, it can be prepared using any ASCII text editor. A Web page is also known as HTML page or HTML document. Web pages are stored using a file extension. **html** such as **MyApplet.html**. Such files are referred to as HTML files. HTML files should be stored in the same directory as the compiled code of the applets.

As pointed out earlier, Web pages include both text that we want to display and HTML tags (commands) to Web browsers. A Web page is marked by an opening HTML tag **< HTML >** and a closing HTML tag **</ HTML >** and is divided into the following three major sections:

1. Comment section (Optional)
2. Head section (Optional)
3. Body section

A Web page outline containing these three sections and the opening and closing HTML tags is illustrated in Fig. 14.6.

Comment Section

This section contains comments about the Web page. It is important to include comments that tell us what is going on in the Web page. A comment line begins with a **< !** and ends with a **>**. Web browsers will ignore the text enclosed between them. Although comments are important, they should be kept to a minimum as they will be downloaded along with the applet. Note that comments are optional

and can be included anywhere in the Web page.

Head Section

The head section is defined with a starting `<HEAD>` tag and a closing `</HEAD>` tag. This section usually contains a title for the Web page as shown below.

```
<HEAD>
    <TITLE> Welcome to
    Java Applets </
    TITLE>
</HEAD>
```

The text enclosed in the tags `<TITLE>` and `</TITLE>` will appear in the title bar of the Web browser when it displays the page. *The head section is also optional.*

Note that tags `< ... >` containing HTML commands usually appear in pairs such as `<HEAD>` and `</HEAD>`, and `<TITLE>` and `</TITLE>`. A slash (/) in a tag signifies the end of that tag section.

Body Section

After the head section, comes the body section. We call this as body section because this section contains the entire information about the Web page and its behavior. We can set up many options to indicate how our page must appear on the screen (like color, location, sound, etc.). Shown below is a simple body section.

```
<BODY>
    <CENTER>
        <H1> Welcome to the World of Applets </H1>
    </CENTER>
    <BR>
    <APPLET ...>
    </APPLET>
</BODY>
```

The body shown above contains instructions to display the message

Welcome to the World of Applets

followed by the applet output on the screen. Note that the `<CENTER>` tag makes sure that the text is centered and `<H1>` tag causes the text to be of the largest size. We may use other heading tags `<H2>` to `<H6>` to reduce the size of letters in the text.

14.8 APPLET TAG

Note that we have included a pair of `<APPLET...>` and `</APPLET>` tags in the body section discussed above. The `<APPLET ...>` tag supplies the name of the applet to be loaded and tells the browser how

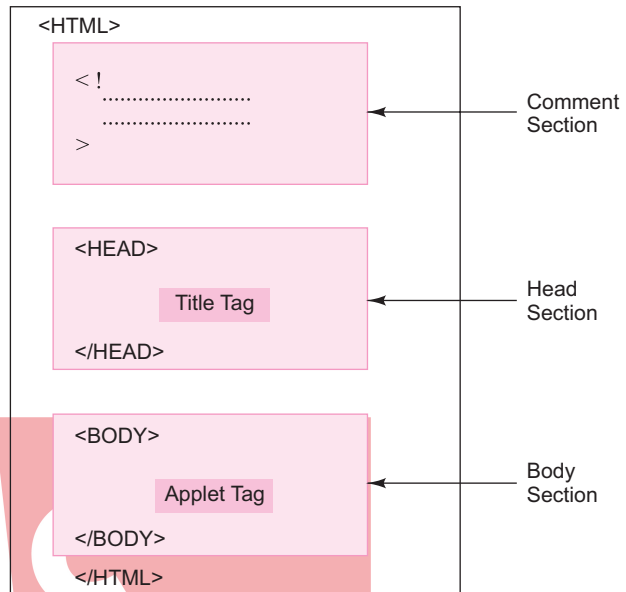


Fig. 14.6 A Web page template

much space the applet requires. The ellipsis in the tag `<APPLET ...>` indicates that it contains certain attributes that must be specified. The `<APPLET>` tag given below specifies the minimum requirements to place the **HelloJava** applet on a Web page:

```
<APPLET
    CODE = helloJava.class
    WIDTH = 400
    HEIGHT = 200 >
</APPLET >
```

This HTML code tells the browser to load the compiled Java applet **HelloJava.class**, which is in the same directory as the HTML file. And also specifies the display area for the applet output as 400 pixels width and 200 pixels height. We can make this display area appear in the centre of the screen by using the `CENTER` tags shown as follows:

```
<CENTER>
    <APPLET
        .....
        .....
        .....
    </APPLET>
</CENTER>
```

Note that `<APPLET>` tag discussed above specifies three things:

1. Name of the applet
2. Width of the applet (in pixels)
3. Height of the applet (in pixels)

14.9 ADDING APPLET TO HTML FILE

Now we can put together the various components of the Web page and create a file known as HTML file. Insert the `<APPLET>` tag in the page at the place where the output of the applet must appear. Following is the content of the HTML file that is embedded with the `<APPLET>` tag of our **HelloJava** applet.

```
<HTML>
    <!-- This page includes a welcome title in the title bar and also
    displays a welcome message. Then it specifies the applet to be
    loaded and executed.
    >
    <HEAD>
        <TITLE>
            Welcome to Java Applets
        </TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1> Welcome to the World of Applets </H1>
        </CENTER>
        <BR>
        <CENTER>
```

```

<APPLET
    CODE = HelloJava.class
    WIDTH = 400
    HEIGHT = 200>
</APPLET>
</CENTER>
</BODY>
</HTML>

```

We must name this file as **HelloJava.html** and save it in the same directory as the compiled applet.

14.10 RUNNING THE APPLET

Now that we have created applet files as well as the HTML file containing the applet, we must have the following files in our current directory:

```

HelloJava.java
HelloJava.class
HelloJava.html

```

To run an applet, we require one of the following tools:

1. Java-enabled Web browser (such as HotJava or Netscape)
2. Java appletviewer

If we use a Java-enabled Web browser, we will be able to see the entire Web page containing the applet. If we use the **appletviewer** tool, we will only see the applet output. Remember that the **appletviewer** is not a full-fledged Web browser and therefore it ignores all of the HTML tags except the part pertaining to the running of the applet.

The **appletviewer** is available as a part of the Java Development Kit that we have been using so far. We can use it to run our applet as follows:

```
appletviewer HelloJava.html
```

Notice that the argument of the **appletviewer** is not the **.java** file or the **.class** file, but rather **.html** file. The output of our applet will be as shown in Fig. 14.7.

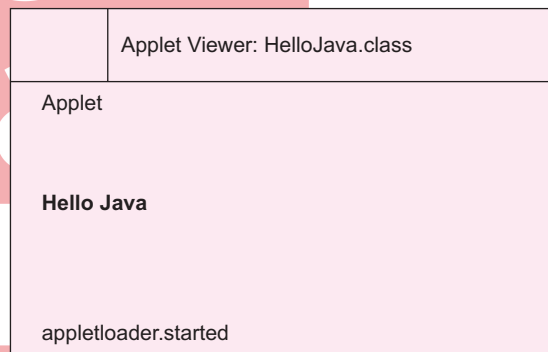


Fig. 14.7 Output of HelloJava applet by using appletviewer

14.11 MORE ABOUT APPLET TAG

We have used the **<APPLET>** tag in its simplest form. In its simplest form, it merely creates a space of the required size and then displays the applet output in that space. The syntax of the **<APPLET>** tag is a little more complex and includes several attributes that can help us better integrate our applet into the overall design of the Web page. The syntax of the **<APPLET>** tag in full form is shown as follows:

```

<APPLET
    [ CODEBASE = codebase_URL ]
    CODE = AppletFileName.class
    [ ALT = alternate_text ]
    [ NAME = applet_instance_name ]

```

```

        WIDTH = pixels
        HEIGHT = pixels
        [ ALIGN = alignment ]
        [ VSPACE = pixels ]
        [ HSPACE = pixels ]
    >
    [ < PARAM NAME = name1 VALUE = value1> ]
    [ < PARAM NAME = name2 VALUE = value2> ]
    .....
    .....
    [ Text to be displayed in the absence of Java ]
</APPLET>

```

The various attributes shown inside [] indicate the options that can be used when integrating an applet into a Web page. Note that the minimum required attributes are:

```

CODE = AppletFileName.class
WIDTH = pixels
HEIGHT = pixels

```

Table 14.1 lists all the attributes and their meaning.

Table 14.1 Attributes of **APPLET** Tag

Attribute	Meaning
CODE=AppletFileName.class	Specifies the name of the applet class to be loaded. That is, the name of the already-compiled .class file in which the executable Java bytecode for the applet is stored. This attribute must be specified.
CODEBASE=codebase_URL (Optional)	Specifies the URL of the directory in which the applet resides. If the applet resides in the same directory as the HTML file, then the CODEBASE attribute may be omitted entirely.
WIDTH=pixels	These attributes specify the width and height of the space on the HTML page that will be reserved for the applet.
HEIGHT=pixels	
NAME=applet_instance_name (Optional)	A name for the applet may optionally be specified so that other applets on the page may refer to this applet. This facilitates inter-applet communication.
ALIGN = alignment (Optional)	This optional attribute specifies where on the page the applet will appear. Possible values for alignment are: TOP, BOTTOM, LEFT, RIGHT, MIDDLE, ABSMIDDLE, ABSBOTTOM, TEXTTOP, and BASELINE.
HSPACE=pixels (Optional)	Used only when ALIGN is set to LEFT or RIGHT, this attribute specifies the amount of horizontal blank space the browser should leave surrounding the applet.
VSPACE=pixels (Optional)	Used only when some vertical alignment is specified with the ALIGN attribute (TOP, BOTTOM, etc.) VSPACE specifies the amount of vertical blank space the browser should leave surrounding the applet.
ALT=alternate_text (Optional)	Non-Java browsers will display this text where the applet would normally go. This attribute is optional.

We summarize below the list of things to be done for adding an applet to a HTML document:

1. Insert an <APPLET> tag at an appropriate place in the Web page.
2. Specify the name of the applet's .class file.
3. If the .class file is not in the current directory, use the codebase parameter to specify
 - the relative path if file is on the local system, or
 - the Uniform Resource Locator (URL) of the directory containing the file if it is on a remote computer.

4. Specify the space required for display of the applet in terms of width and height in pixels.
5. Add any user-defined parameters using <PARAM> tags.
6. Add alternate HTML text to be displayed when a non-Java browser is used.
7. Close the applet declaration with the </APPLET> tag.

14.12 PASSING PARAMETERS TO APPLETS

We can supply user-defined parameters to an applet using <PARAM...> tags. Each <PARAM...> tag has a **name** attribute such as **color**, and a **value** attribute such as **red**. Inside the applet code, the applet can refer to that parameter by name to find its value. For example, we can change the color of the text displayed to red by an applet by using a <PARAM...> tag as follows:

```
<APPLET>
<PARAM name = color VALUE = "red">
</APPLET>
```

Similarly, we can change the text to be displayed by an applet by supplying new text to the applet through a <PARAM...> tag as shown below.

```
<PARAM NAME = text VALUE = "I love Java">
```

Passing parameters to an applet code using <PARAM> tag is something similar to passing parameters to the **main()** method using command line arguments. To set up and handle parameters, we need to do two things:

1. Include appropriate <PARAM...> tags in the HTML document.
2. Provide Code in the applet to parse these parameters.

Parameters are passed on an applet when it is loaded. We can define the **init()** method in the applet to get hold of the parameters defined in the <PARAM> tags. This is done using the **getParameter()** method, which takes one string argument representing the **name** of the parameter and returns a string containing the value of that parameter.

Program 14.2 shows another version of **HelloJava** applet. Compile it so that we have a class file ready.

Program 14.2 *Applet HelloJavaParam*

```
import java.awt.*;
import java.applet.*;
public class HelloJavaParam extends Applet
{
    String str;
    public void init( )
    {
        str = getParameter( "string" ); // Receiving parameter value
        if (str == null)
            str = "Java";
        str = "Hello" + str; // Using the value
    }
    public void paint (Graphics g)
    {
        g.drawString(str, 10, 100).
    }
}
```

Now, let us create HTML file that contains this applet. Program 14.3 shows a Web page that passes a parameter whose NAME is “string” and whose VALUE is “APPLET!” to the applet **HelloJavaParam**.

Program 14.3 *The HTML file for HelloJavaParam applet*

```
<HTML>
  <!--Parameterized HTML file-->
  <HEAD>
    <TITLE> Welcome to Java Applets </TITLE>
  </HEAD>
  <BODY>
    <APPLET CODE = HelloJavaParam.class
      WIDTH = 400
      HEIGHT = 200>
      <PARAM NAME = "string"
        VALUE = "Applet!">
    </APPLET>
  </BODY>
</HTML>
```

Save this file as **HelloJavaParam.html** and then run the applet using the applet viewer as follows:

```
appletviewer HelloJavaParam.html
```

This will produce the result as shown in Fig. 14.8.

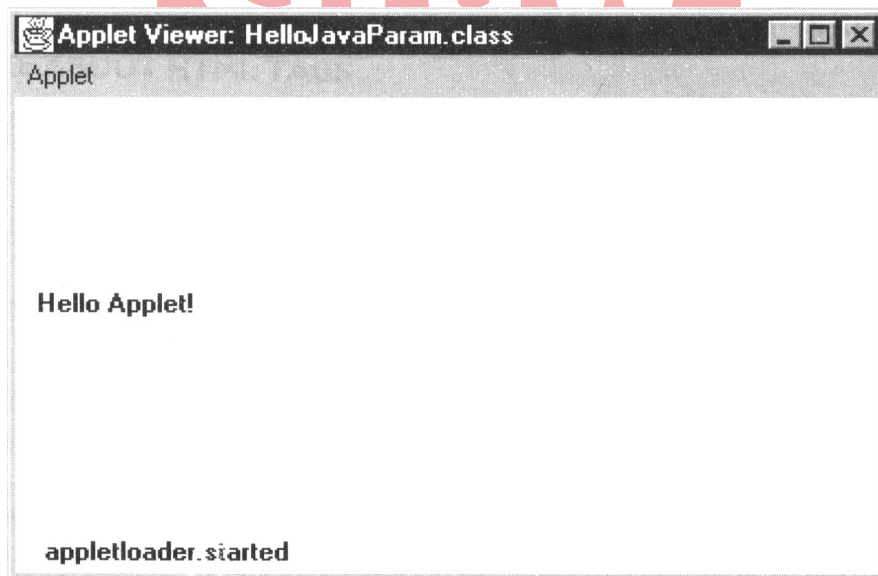


Fig. 14.8 *Displays of HelloJavaParam applet*

Now, remove the <PARAM> tag from the HTML file and then run the applet again. The result will be as shown in Fig. 14.9.

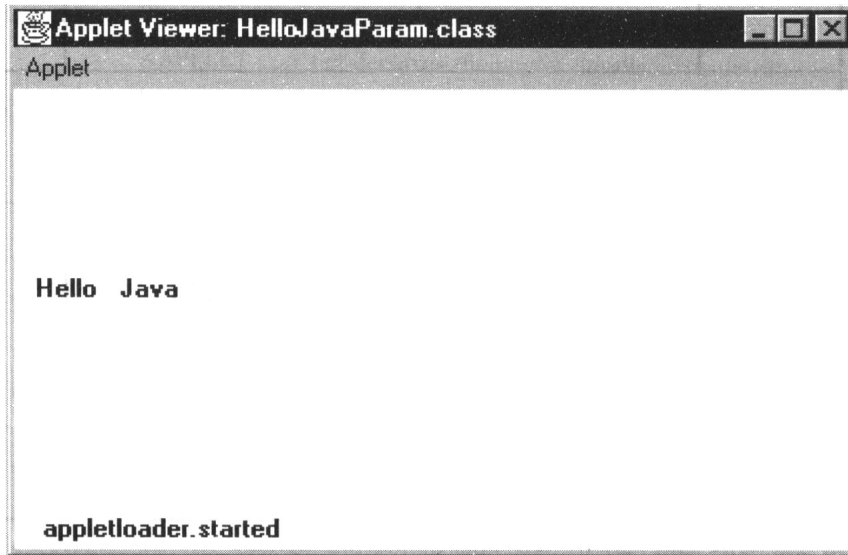


Fig. 14.9 *Output without PARAM tag*

14.13 ALIGNING THE DISPLAY

We can align the output of the applet using the ALIGN attribute. This attribute can have one of the nine values:

LEFT, RIGHT, TOP, TEXT TOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM.

For example, ALGN = LEFT will display the output at the left margin of the page. All text that follows the ALIGN in the Web page will be placed to the right of the display. Program 14.4 shows an HTML file for our **HelloJava** applet shown in Program 14.1.

Program 14.4 *HTML file with ALIGN attribute*

```
<HTML>
  <HEAD>
    <TITLE> Here is an applet </TITLE>
  </HEAD>
  <BODY>
    <APPLET CODE = HelloJava.class
      WIDTH  = 400
      HEIGHT = 200
      ALIGN  = RIGHT>
    </APPLET>
  </BODY>
</HTML>
```

The alignment of applet will be seen and appreciated only when we run the applet using a Java-capable browser. Figure 14.10 shows how an applet and text surrounding it might appear in a Java-capable browser. All the text following the applet appears to the left of that applet.

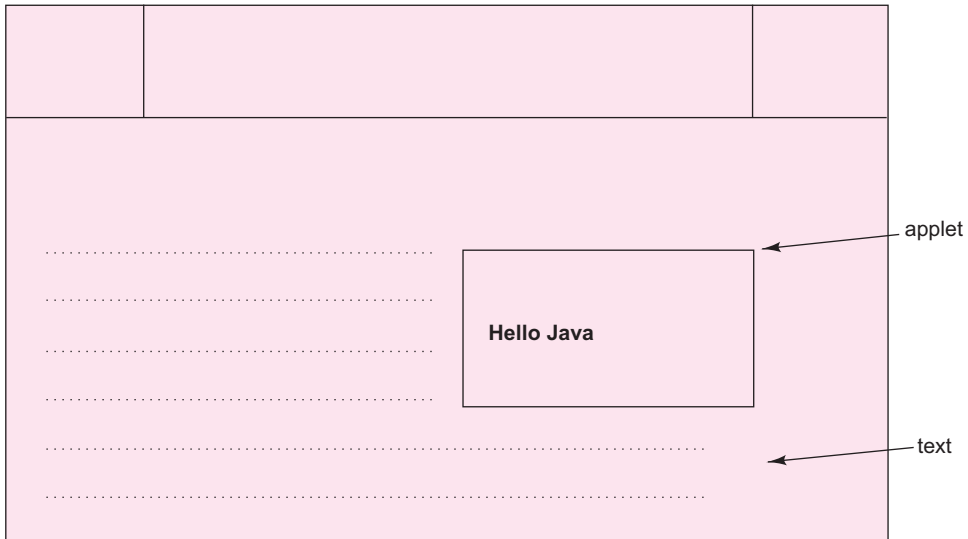


Fig. 14.10 An applet aligned right

14.14 MORE ABOUT HTML TAGS

We have seen and used a few HTML tags. HTML supports a large number of tags that can be used to control the style and format of the display of Web pages. Table 14.2 lists important HTML tags and their functions.

Table 14.2 HTML Tags and Their Functions

Tag	Function
<HTML>...</HTML>	Signifies the beginning and end of a HTML file.
<HEAD>...</HEAD>	This tag may include details about the Web page. Usually contains <TITLE> tag within it.
<TITLE>...</TITLE>	The text contained in it will appear in the title bar of the browser.
<BODY>...</BODY>	This tag contains the main text of the Web page. It is the place where the <APPLET> tag is declared.
<H1>...</H1> <H6>...</H6>	Header tags. Used to display headings. <H1> creates the largest font header, while <H6> creates the smallest one.
<CENTER> ...</CENTER>	Places the text contained in it at the center of the page.
<APPLET ...>	<APPLET ...> tag declares the applet details as its attributes.
<APPLET>...</APPLET>	May hold optionally user-defined parameters using <PARAM> Tags.
<PARAM...>	Supplies user-defined parameters. The <PARAM> tag needs to be placed between the <APPLET> and </APPLET> tags. We can use as many different <PARAM> tags as we wish for each applet.
...	Text between these tags will be displayed in bold type.

(Contd.)

Table 14.2 (Contd.)

Tag	Function
 	Line break tag. This will skip a line. Does not have an end tag.
<P>	Para tag. This tag moves us to the next line and starts a paragraph of text. No end tag is necessary.
<IMG...>	This tag declares attributes of an image to be displayed.
<HR>	Draws a horizontal rule.
<A...> 	Anchor tag used to add hyperlinks.
 ... 	We can change the color and size of the text that lies in between and tags using COLOR and SIZE attributes in the tag<FONT...>.
<!>	Any text starting with a <! mark and ending with a > mark is ignored by the Web browser. We may add comments here. A comment tag may be placed anywhere in a Web page.

14.15 DISPLAYING NUMERICAL VALUES

In applets, we can display numerical values by first converting them into strings and then using the **drawString()** method of **Graphics** class. We can do this easily by calling the **valueOf()** method of **String** class. Program 14.5 illustrates how an applet handles numerical values.

Program 14.5 *Displaying numerical values*

```
import java.awt.*;
import java.applet.*;
public class NumValues extends Applet
{
    public void paint (Graphics g)
    {
        int value1 = 10;
        int value2 = 20;
        int sum = value1 + value2;
        String s = "sum:" + String.valueOf(sum);
        g.drawString(s, 100, 100);
    }
}
```

The applet Program 14.5 when run using the following HTML file displays the output as shown in Fig. 14.11.

```
<html>
<applet
    code = NumValues.class
    width = 300
    height = 300 >
</applet>
</html>
```

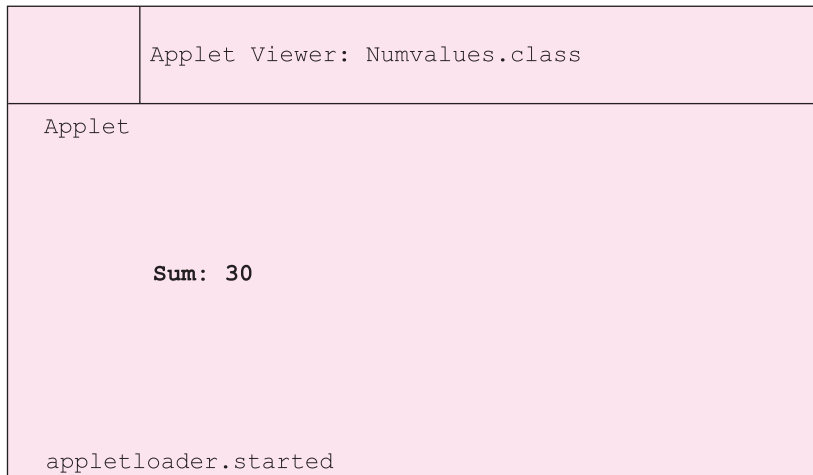


Fig. 14.11 Output of Program 14.5

14.16 GETTING INPUT FROM THE USER

Applets work in a graphical environment. Therefore, applets treat inputs as text strings. We must first create an area of the screen in which user can type and edit input items (which may be any data type). We can do this by using the **TextField** class of the applet package. Once text fields are created for receiving input, we can type the values in the fields and edit them, if necessary.

Next step is to retrieve the items from the fields for display of calculations, if any. Remember, the text fields contain items in string form. They need to be converted to the right form, before they are used in any computations. The results are then converted back to strings for display. Program 14.6 demonstrates how these steps are implemented.

Program 14.6 Interactive input to an applet

```
import java.awt.*;
import java.applet.*;
public class UserIn extends Applet
{
    TextField text1, text2;
    public void init( )
    {
        text1 = new TextField(8);
        text2 = new TextField(8);
        add (text1);
        add (text2);
        text1.setText ("0");
        text2.setText ("0");
    }
}
```

(Contd.)

Program 14.6 (Contd.)

```

public void paint (Graphics g)
{
    int x = 0, y = 0, z = 0;
    String s1, s2, s;
    g.drawString "Input a number in each box", 10, 50);
    try
    {
        s1 = text1.getText( );
        x = Integer.parseInt(s1);
        s2 = text2.getText( );
        y = Integer.parseInt(s2);

    }
    catch (Exception ex) { }
    z = x + y;
    s = String.valueOf (z);
    g.drawString ("THE SUM IS:", 10, 75);
    g.drawString (s, 100, 75);
}

public Boolean action (Event event, Object object)
{
    repaint ( );
    return true;
}
}

```

Run the applet **UserIn** using the following steps:

1. Type and save the program (.java file)
2. Compile the applet (.class file)
3. Write an HTML document (.html file)

```

<html>
<applet
    code = UserIn.class
    width = 300
    height = 200 >
</applet>
</html>

```

4. Use the **appletviewer** to display the results

When the applet is up and running, enter a number into each text field box displayed in the applet area, and then press the Return Key. Now, the applet computes the sum of these two numbers and displays the result as shown in Fig. 14.12.

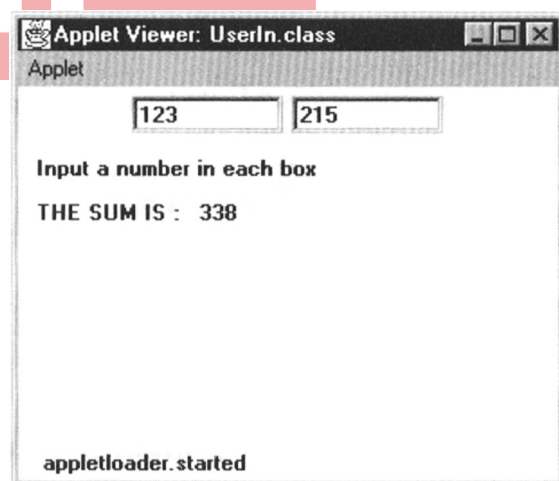


Fig. 14.12 Interactive computing with applets

Program Analysis

The applet declares two **TextField** objects at the beginning.

```
TextField text1, text2;
```

These two objects represent text boxes where we type the numbers to be added.

Next, we override the **init()** method to do the following:

1. To create two text field objects to hold strings (of eight character length).
2. To add the objects to the applet's display area.
3. To initialize the contents of the objects to zero.

Then comes the **paint()** method where all the actions take place. First, three integer variables are declared, followed by three string variables. Remember, the numbers entered in the text boxes are in string form and therefore they are retrieved as strings using the **getText()** method and then they are converted to numerical values using the **parseInt()** method of the **Integer** class.

After retrieving and converting both the strings to integer numbers, the **paint()** method sums them up and stores the result in the variable **z**. We must convert the numerical value in **z** to a string before we attempt to display the answer. This is done using the **valueOf()** method of the **String** class.

14.17 EVENT HANDLING

As the name suggests, event handling is a mechanism that is used to handle events generated by applets. An event could be the occurrence of any activity such as a mouse click or a key press. In Java, events are regarded as method calls with a certain task performed against the occurrence of each event.

Some of the key events in Java are:

1. **ActionEvent** is triggered whenever a user interface element is activated, such as selection of a menu item.
2. **ItemEvent** is triggered at the selection or deselection of an itemized or list element, such as check box.
3. **TextEvent** is triggered when a text field is modified.
4. **WindowEvent** is triggered whenever a window-related operation is performed, such as closing or activating a window.
5. **KeyEvent** is triggered whenever a key is pressed on the keyboard.

Each event is associated with an event source and an event handler. Event source refers to the object that is responsible for generating an event whereas event listener is the object that is referred by event source at the time of occurrence of an event.

Event Sources

It is not necessary that a source generates only one type of event; it can generate events of multiple types. However, it is mandatory that the source object registers listener objects corresponding to each of its event types. The registration of a listener object with an event ensures that on occurrence of the event, the corresponding listener object is notified for taking appropriate action. Following is the syntax for registering a listener for an event:

```
public void add<Type>Listener(<Type>Listener EveList)
```

Here, **Type** is the name of the event and **EveList** is the corresponding listener object reference.

Event Listeners

The **event listener** object contains methods for receiving and processing event notifications sent by the source object. These methods are implemented from the corresponding listener interface contained in the **java.awt.event** package.

Event Classes

All the events in Java have corresponding event classes associated with them. Each of these classes is derived from one single super class, i.e., **EventObject**. It is contained in the **java.util** package. The **EventObject** class contains the following two important methods for handling events:

1. **getSource()**: Returns the event source.
2. **toString()**: Returns a string containing information about the event source.

The immediate subclass of **EventObject** is the **AWTEvent** class from which all the AWT-based event classes are derived. Figure 14.13 shows hierarchy of event-based classes:

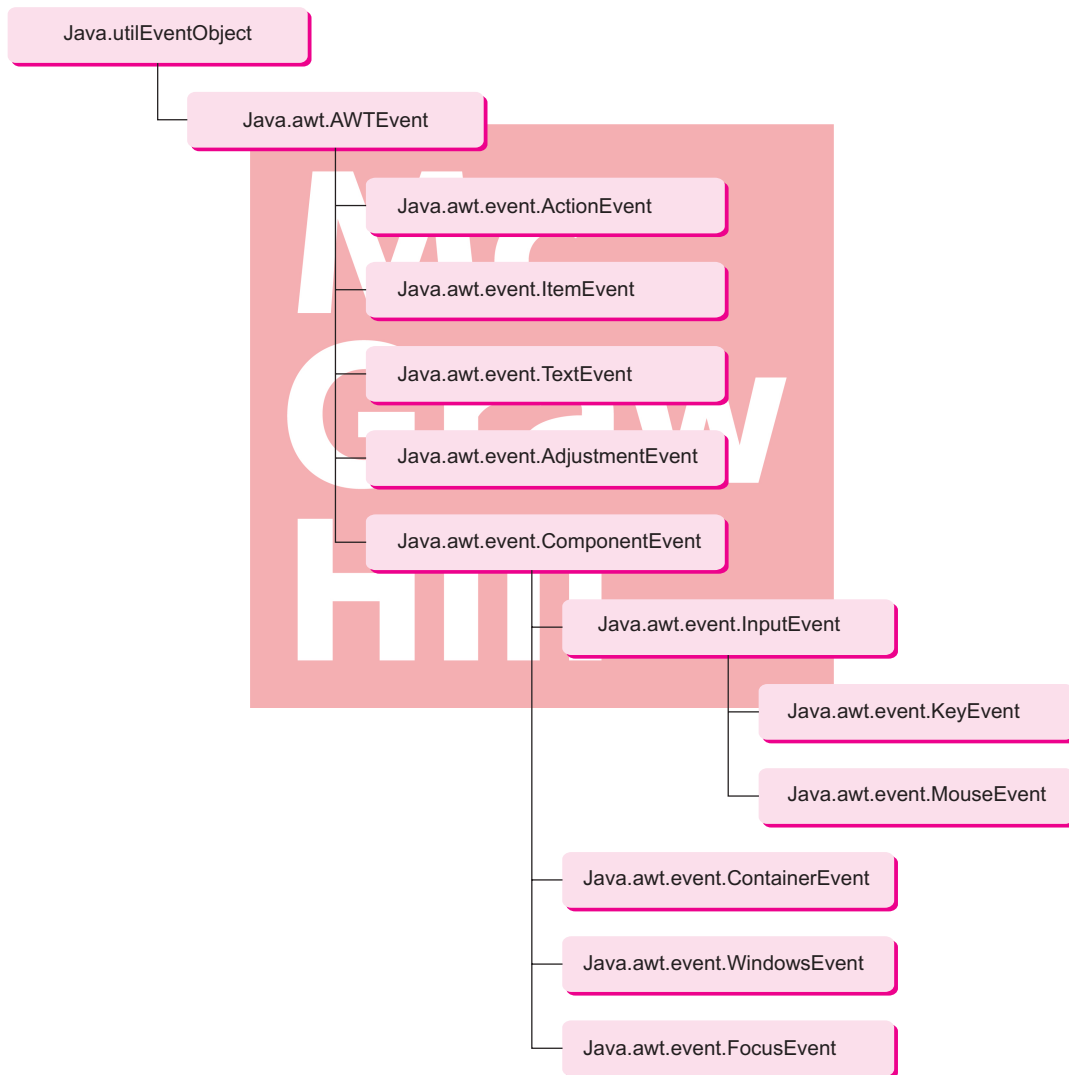


Fig. 14.13 Hierarchy of event-based classes in Java

Program 14.7 is a sample program depicting the usage of keyboard event in an applet.

Program 14.7 *Keyboard event in an applet*

```

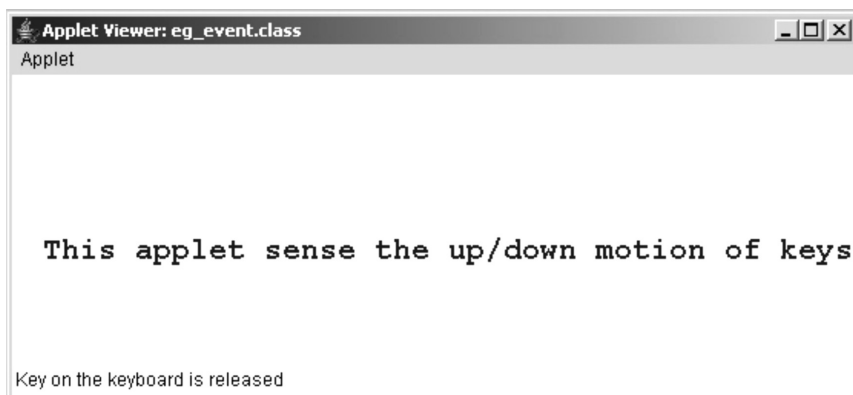
<html>
<body>
<applet code=eg_event.class width =200 height=200>
</applet>
</body>
</html>

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

public class eg_event extends Applet implements KeyListener
{
    public void init()
    {
        addKeyListener(this);
    }
    public void keyTyped(KeyEvent KB){}
    public void keyReleased(KeyEvent KB)
    {
        showStatus("Key on the keyboard is released");
    }
    public void keyPressed(KeyEvent KB){
        showStatus("A key on the keyboard is pressed");
    }
    Font f1 = new Font("Courier New",Font.BOLD, 20);
    public void paint(Graphics GA){
        GA.setFont(f1);
        GA.setColor(Color.blue);
        GA.drawString("This applet sense the up/down motion of keys",20,120);
    }
}

```

The output of Program 14.7 would be:



This program detects the key-press activities from the keyboard and flashes the corresponding message accordingly.

SUMMARY

Applets are Java programs developed for use on the Internet. They provide a means to distribute interesting, dynamic, and interactive applications over the World Wide Web. We have learned the following about applets in this chapter:

- How do applets differ from applications
- How to design applets
- How to design a Web page using HTML tags
- How to execute applets
- How to provide interactive input to applets

■ ■ ■

REVIEW QUESTIONS

- 14.1 What is an applet?
- 14.2 What is a local applet?
- 14.3 What is a remote applet?
- 14.4 Explain the client/server relationship as applied to Java applets.
- 14.5 How do applets differ from application programs?
- 14.6 Discuss the steps involved in developing and running a local applet.
- 14.7 Discuss the steps involved in loading and running a remote applet.
- 14.8 Describe the various sections of Web page.
- 14.9 How many arguments can be passed to an applet using <PARAM> tags?
- 14.10 Why do applet classes need to be declared as public?
- 14.11 Describe the different stages in the life cycle of an applet. Distinguish between **init()** and **start()** methods.
- 14.12 Develop an applet that receives three numeric values as input from the user and then displays the largest of the three on the screen. Write an HTML page and test the applet.

DEBUGGING EXERCISES

- 14.1 Find errors in the following code for drawing set of nested Rectangles.

```
import java.awt.*;
import java.applet.Applet;
public class Rectangles extends Applet
{
    public void paint(Graphics g)
    {
        int inset;
        int rectWidth, rectHeight;
        g.setColor(Color.blue);
        g.fillRect(0,0,300,160);
        inset = 0;
        rectWidth = 299;
        rectHeight= 159;
        g.setColor(Color.red);
        g.drawString("Rectangles",150,200);
        while (rectWidth>= 0 && rectHeight>= 0)
        {
```

```

        g.drawRect(inset, inset, rectWidth, rectHeight)
        inset += 15;
        rectWidth -= 30;
        rectHeight -= 30;
    }
}

```

- 14.2 The following code converts temperature values. Will the code display the new value on moving the scrollbar?

```

import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
public class CelsiusValue extends Applet implements AdjustmentListener
{
    private Scrollbar bar;
    private int old, newtemp = 0;
    private inft fahr = 32;
    public void init()
    {
        bar = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 100);
        bar.addAdjustmentListener(this);
        setLayout(new BorderLayout( ));
        add("North", bar);
    }
    public void paint(Graphics g)
    {
        g.drawString("Celsius = " + newtemp, 30, 50);
        g.drawString("Fahrenheit = " + fahr, 30, 70);
    }
    public void adjustmentValueChanged(AdjustmentEvent e)
    {
        newtemp = bar.getValue( );
        if (newtemp != old)
        {
            fahr = newtemp * 9 / 5 + 32;
            old = newtemp;
        }
    }
}

```

- 14.3 Given code creates an expanding ring on mouse click event. Does the code show the desired output?

```

import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class MouseRing extends Applet implements MouseListener
{
    private int    x = 100, y = 100;
    private int    pauseLength;
    public void    init( )
    {
        pauseLength = Integer.parseInt(getParameter("PauseLength"));
        setBackground(Color.white);
    }
}

```

```

public void paint(Graphics g)
{
    int count = 0;
    while (count < 100)
    {
        int radius = 5*count;
        int diameter = 2*radius;
        g.setColor(Color.black);
        g.drawOval(-radius, y-radius, diameter, diameter);
        // Draw pause(pauseLength);
        g.setColor(Color.white);
        g.drawOval(x-radius, y-radius, diameter, diameter)
        // Erase!
        count = count+1;
    }
}

private void pause(int howLong)
{
    for (int count = 0; count < howLong; count++);
}
public void mouseClicked(MouseEvent e)
{
    x = e.getX();
    y = e.getY();
    repaint();
}
public void mouseExited(MouseEvent e) { }
public void mouseEntered(MouseEvent e) { }
public void mousePressed(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
}

```

14.4 Using parameter, an applet provides answers to different questions. Correct the code.

```

Question.html
<html> <head>
<title>Questions and Answers</title>
</hhead>
<body>
<APPLET CODE=Question.class
        WIDTH=400 HEIGHT = 100>
    <PARAM NAME=question VALUE="What is Inheritance?">
    <PARAM NAME=answer VALUE="Getting the properties of one class into
    another">
</APPLET>
</body>
</html>

Question.java
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class Question extends Applet implements ActionListener
{
    String theQuestion;
    String theAnswer = " ";
    Button reveal = new Button("Click to know the answer");
}

```

```

    public void init()
    {
        theQuestion = getParameter("ques");
        add(reveal);
        reveal.addActionListener(this);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.black);
        g.drawString(theQuestion, 10, 50);
        g.setColor(Color.red);
        g.drawString(theAnswer, 10, 70);
    }
    public void actionPerformed(ActionEvent e)
    {
        theAnswer = getParameter("answer");
        repaint();
    }
}

```

- 14.5 Given applet shows the sequence of events called in an applet. Will the message defined in destroy() event be shown?

```

import java.awt.*;
import java.applet.Applet;
public class AllMethodsApplet extends Applet
{
    TextArea messages = new TextArea(8, 30);
    public AllMethodsApplet()
    {
        messages.append("Constructor called/n");
    }
    public void init()
    {
        add(messages);
        messages.append ("Init called/n");
    }
    public void start()
    {
        messages.append ("Start called/n" );
    }
    public void stop( )
    {
        messages.append ("Stop called/n");
    }
    public void destroy( )
    {
        messages.append ("Destroy called/n");
    }
    public void paint(Graphics display)
    {
        messages.append ("Paint called/n");
        Dimension size = getSize( );
        display.drawRect( 0, 0, size.width-1, size.height-1);
    }
}

```