Table 12.1 Difference between Applet and Application

Applet	Application
The execution of the applet does not start from `main()` method, as it does not have one.	The execution of an application program starts from `main()`.
Applets cannot run on their own. They have to be embedded inside a web page to get executed.	These can run on their own. In order to get executed, they need not be embedded inside any web page.
Applets can only be executed inside a browser or appletviewer.	Applications are executed at command line.
Applets execute under strict security limitations that disallow certain operations (sandbox model security).	Applications have no inherent security restrictions.
Applets have their own life cycle `init()`→`start()`→`paint()`→`stop`→`destroy()`	Applications have their own life cycle. Their execution begins at `main()`.

12.2 APPLETS

`java.applet.Applet` is the superclass of all the applets. Thus all the applets, directly or indirectly, inherently use the methods of `Applet` belonging `java.applet` package. This class provides all the necessary methods for starting, stopping, and manipulating applets. It also has methods providing multimedia support to an applet. `Applet` class has a predefined hierarchy in Java, which shows the classes extended by `Applet` class.

Figure 12.1 simply makes it easy for you to understand that an applet, which is a subclass of `java.applet.Applet`, also inherits the methods of the other classes like `java.awt.Panel`, `java.awt.Container`, `java.awt.Component`, and `java.lang.Object`, indirectly.

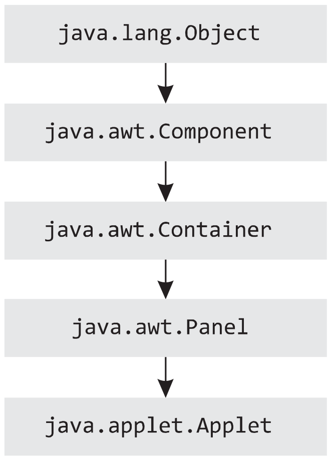


Fig. 12.1 Hierarchy of `Applet` Class

You can see that these classes are the ones which provide support for Java's window-based GUI, thus making an applet capable of supporting window-based activities. The common methods belonging to the `Applet` class are mentioned in Table 12.2.

Table 12.2 `Applet` Class Methods

Method	Description
`void init()`	First method to be called when an applet begins execution.
`boolean isActive()`	Returns true if the applet is running, otherwise false.
`URL getDocumentBase()`	Gets the URL of the document in which this applet is embedded.
`URL getCodeBase()`	Returns the URL of the directory where the class file of the invoking applet exists.
`String getParameter (String name)`	Returns the value of the parameter associated with parameter's name. Null is returned if the parameter is not specified.
`AppletContext getAppletContext()`	Determines this applet's context, which allows the applet to query and affect the environment in which it runs.
`void resize (int width,int height)`	Resizes the applet according to the parameters, *width* and *height*.
`void showStatus(String msg)`	Displays the string, *msg*, in the status window of the browser or appletviewer (only if they support status window).
`Image getImage(URL url)`	Returns an object of image, which binds the image found at the URL, specified as the argument of the method.
`Image getImage(URL url, String imgName)`	Returns the image object which encapsulates the image found at the specified URL and having the name specified by imgName.
`static final AudioClip newAudioClip(URL url)`	Returns an `AudioClip` object that encapsulates the audio found at the URL specified as the argument.
`void start()`	Starts or resumes the execution of applet.
`void stop()`	Stop or suspends the applet.
`void destroy()`	Terminates the applet.
`AccessibleContext getAccessibleContext()`	Returns the accessibility context for the invoking object.
`AudioClip getAudioClip(URL url)`	Returns the `AudioClip` object, which encapsulates the audio clip found at the URL, specified as the argument to the method.
`AudioClip getAudioClip(URL url,String clipName)`	Returns the `AudioClip` object, which encapsulates the audio clip found at URL, specified as the argument to the method and having the name specified by `clipName`.
`String getAppletInfo()`	Returns the string describing the applet.
`Locale getLocale()`	Returns the `Locale` object that is used by various locale sensitive classes and methods.
`String[][] getParameterInfo()`	Returns a string table that describes the parameter recognized by the applet.

12.3 APPLET STRUCTURE

Apart from using the services of `Applet` class, an applet also uses the services of `Graphics` class of the `java.awt` package. The `Applet` class has methods such as `init()`, `start()`, `destroy()`, and `stop()`, which are responsible for the birth and behavior of an applet. We have already mentioned in Section 12.1 that unlike an application program, Java runtime system does not call the `main()` method to start the execution of an applet, rather it just loads the methods of `applet` class which are responsible for starting, running, stopping, and manipulating an applet. The complete life cycle of the applet will be taken up in the next section.

There is a method, `paint()` in the `Container` class, which is inherited by `Applet` class (as you can make out from Fig. 12.1), carrying the signature,

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

This method, when called, displays the output of applet as per the code written on the applet's panel. You can see the argument of this method; it is nothing but an object of `Graphics` class. The object makes it possible for an applet to output text, graphics, sound, etc. One thing you must remember, you cannot take the services of `Graphics` class unless you import the package it belongs to, i.e., `java.awt`. The output operations for an applet requires the methods contained in the `Graphics` class, that is why its `Graphics` object is passed as argument to `paint()`. Now that you know some details about the internals of an applet, we can discuss the program structure of an applet.

When an applet is first loaded, Java runtime system creates an instance of the main class, which is `FirstApplet` in this case. Then the methods belonging to the `Applet` class are called through this object.

12.4 AN EXAMPLE APPLET PROGRAM

Let us take an example applet, which displays the statement "This is my first applet program."

Applet Program Structure

```
import java.awt.*;
//so as to make Graphics class available

import java.applet.*;
//so as to make Applet class available
………………………………
………………………………

public class NewApplet extends Applet
// new applet with the name, newApplet declared
{
     ………………………………
     ………………………………

     public void paint(Graphics g)
     // paint() of Applet class overridden to contain output operations
     {
     ………………………………
     ………………………………
     }
………………………………
………………………………
}
```

Example 12.1(a) First Applet Example

```
L1   import java.applet.*;
L2   import java.awt.*;
L3   public class FirstApplet extends Applet
     {
L4       public void paint(Graphics g) {
L5       g.drawString("This is my First Applet", 10, 10);
L6       }
L7   }
```

Explanation

L1 All applets are the subclasses of `Applet` class. All applets must import the `java.applet` package.
L2 The applet uses the methods of `java.awt` package; it must be imported.
L3 The `FirstApplet` class is declared public so that the program that executes the applet (a Java-enabled browser or applet viewer, which might not be local to the program) can access it. This class extends the `Applet` class of `java.applet` package, thus inheriting the features of `Applet` class.
L4 The `paint()` method defined by AWT `Container` class is overridden. Any output to be shown by an applet has to be taken care by this method only. Please note that an object of `Graphics` class is passed as parameter to this method.
L5 The object of the `Graphics` class is used to invoke `drawString()` method, which is responsible for printing the string ("This is my First Applet") at *x*-coordinate 10 and *y*-coordinate 10.

12.4.1 How to Run an Applet?

There are two ways to run an applet. We will explain these in context to the above example. These approaches are

(a) Save the file as `FirstApplet.java` and compile it by using `javac`. Now, type in the following HTML code in your editor and save the file as `FirstApplet.html` (here, the file name is not necessarily the same as the class name, as it was for the `java` file.)

```
<HTML><BODY>
<APPLET code = "FirstApplet.class" WIDTH = 200 HEIGHT =
150></APPLET>
</BODY></HTML>
```

You can execute the HTML file by giving

```
appletviewer FirstApplet.html
```

Note If you wish to run the above html file in any web browser, instead of using applet viewer, you must have Java-enabled web browser. Otherwise, you will have to install Java plug-in, which lets you run your applets as web pages under 1.2 version of JVM instead of the web browser's default virtual machine.

(b) Just as above, save the file as `FirstApplet.java` and compile it by using `javac`. In order to run the applet, you have to give the below HTML coding as a comment in `FirstApplet.java`.

```
/* <APPLET code = "FirstApplet.class" WIDTH = 200 HEIGHT = 150></APPLET> */
```

Execute the applet as,

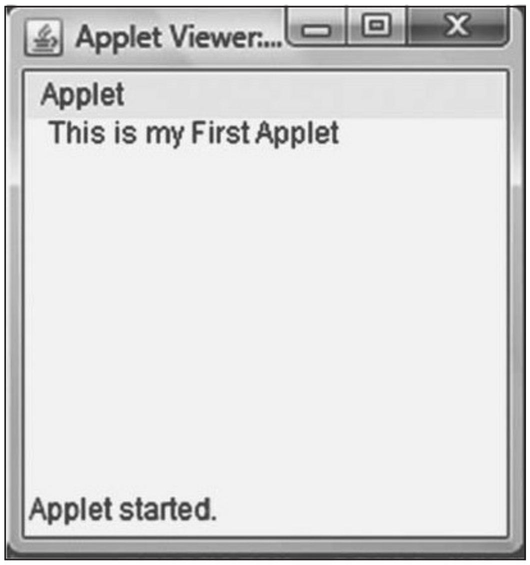
```
appletviewer FirstApplet.java
```

In this chapter, we will be using the second approach throughout. So Example 12.1(a) should have been actually written as shown in Example 12.1(b).

Example 12.1 (b) First Applet Example (Revised)

```
/* <APPLET code = "FirstApplet.class" WIDTH = 200 HEIGHT = 150></APPLET>
*/ import java.applet.*;
import java.awt.*;
public class FirstApplet extends Applet {
  public void paint(Graphics g) {
    g.drawString("This is my First Applet",10,10);
}}
```

Output



```
This is my First Applet
```

```
Applet started.
```

Fig. 12.2 Output Shown with the Help of Applet Viewer

12.5 APPLET LIFE CYCLE

An applet may move from one state to another depending upon a set of default behaviors inherited in the form of methods from `Applet` class. These states can be summed up as,

- Born
- Running

- Idle
- Dead

Figure 12.3 shows the flow an applet takes while moving from one state to another.

As mentioned before, an applet may override some of the basic methods of class `Applet`. Note that these methods are responsible for the lifecycle of an applet. These methods are

- `init()`
- `start()`
- `stop()`
- `destroy()`

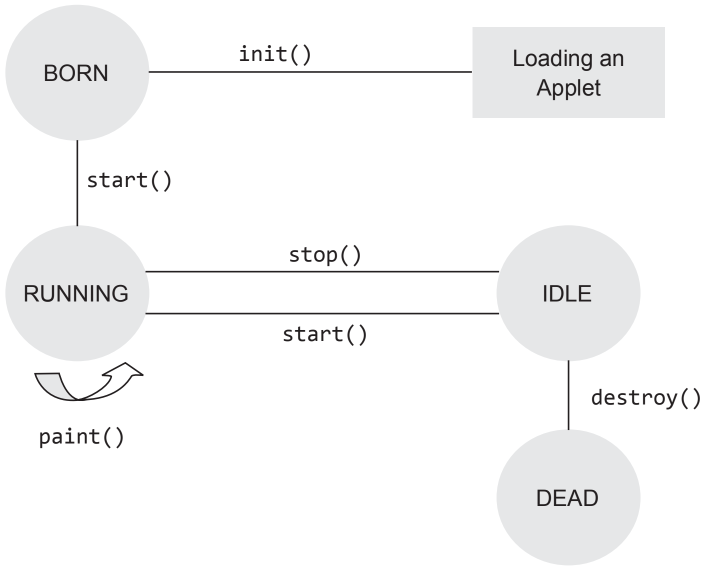


Fig. 12.3 Applet's State Diagram

Let us discuss all these states of an applet in greater detail.

Born State

You can easily see from Fig. 12.3, that an applet enters this phase as soon as it is first loaded by Java. This is made possible by calling `init()` of `Applet` class. Now, what are the things that Java runtime system does, while loading the applet, i.e., when `init()` is called? It creates the objects needed by the applet; it might set initial values, load font and images or set up colors. The method `init()` is called only once during the lifetime of an applet.

Note In order to initialize an applet, we must override the `init()` method of `Applet` class.

Running State

Applet moves to the running state by calling `start()`. An applet moves to this phase automatically after the initialization state. But if the applet is stopped or it goes to idle state, `start()` must be called in order to force the applet again to the running state. Suppose you have opened a web page (having an applet) and you move temporarily to another web page (by minimizing it) the first one goes to the idle state; when you return back to the first page, `start()` is called to put the applet in the running state again. Unlike `init()`, `start()` can be called more than once.

Note `start()` can be overridden to create a thread to control an applet.

You can see the `paint()` method in Fig. 12.3. This method is responsible for forcing the applet to an intermediary state (display state), which is actually a part of the running state itself. While running, an applet may need to perform some output and display it on the panel of the applet. The `paint()` method, which is a part of `Container` class (a superclass of `Applet` class), needs to be overridden for the purpose. This method is called each time to draw and redraw the output of an applet. We already know the drawing of output of an applet. Let us discuss redrawing the output of an applet with an example. An applet window may be minimized and then restored. This restoration is nothing but redrawing of applet's output and could be achieved by calling `paint()`. Actually when an applet in restored, `start()` and `paint()` are called in sequence. We will revisit this method in Section 12.7.1.

Idle State

An applet goes to idle state, once it is stopped from running. If we leave a web page containing an applet (i.e., minimize it), the applet automatically goes to idle state. An applet can also be forced to stop or go to idle state by calling `stop()`.

Note If a thread has been created to control an applet by overriding `start()`, then we must use `stop()` to stop the thread, by overriding the `stop()` method of the `Applet` class.

Dead State

Terminating or stopping an applet should not be confused with destroying an applet. An applet goes to dead state when it is destroyed by invoking the `destroy()` method of `Applet` class. It results in complete removal of applet from the memory. Whenever we quit the browser, `destroy()` is called automatically. You should free up the resources being used by applet (if any) by overriding the `destroy()` method. Like `init()`, `destroy()` is also called only once. `stop()` is always called before `destroy()`.

12.6 COMMON METHODS USED IN DISPLAYING THE OUTPUT

There are certain methods which you should be acquainted with, as they might be used in applet programming further down the chapter. These are the methods belonging to different classes, which can handle the AWT windowed environment.

`drawString()`

This method is a member of `Graphics` class, used to output a string to an applet. It is typically called from within the `paint()` or `update()` method. Its form is

```
    void drawString(String msg, int a, int b)
```

Here, the `string msg` is the string output to be displayed by the applet and `a`, `b` are the `x`, `y` coordinates respectively of the window, where the output has to be displayed.

setBackground()

This method belongs to component class. It is used to set the background color of the applet window. Its form is

```
void setBackground(Color anyColor)
```

The above method takes the color to be set as background, as argument. The `Color` class has certain predefined constants for each color, such as `Color.red, Color.blue, Color. green,` and `Color.pink`.

setForeground()

This method is similar to `setBackground` method, except that these are used to set the color of the text to be displayed on the foreground of the applet window. Its form is

```
void setForeground(Color anyColor)
```

Component class has two more methods `getBackground()` and `getForeground()`, having the following forms:

```
Color getBackground();
Color getForeground();
```

You can very well see that these methods return the current context of the `Color`, showing the background and foreground colors, respectively.

showStatus()

This method is a member of `Applet` class. It is used to display any string in the status window of the browser or `appletviewer`. Its from is

```
void showStatus(String text)
```

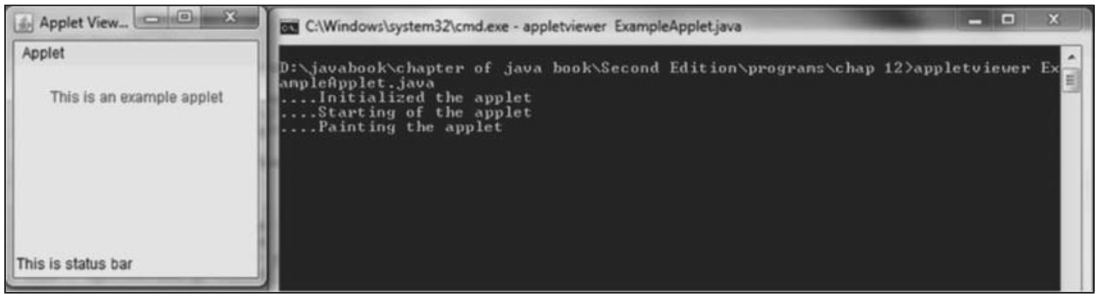
Here, the argument of the method is basically the string which you want to be displayed in the status window.

Before going any further, we should better take an example which uses these methods, discussed until now.

Example 12.2 Applet Methods

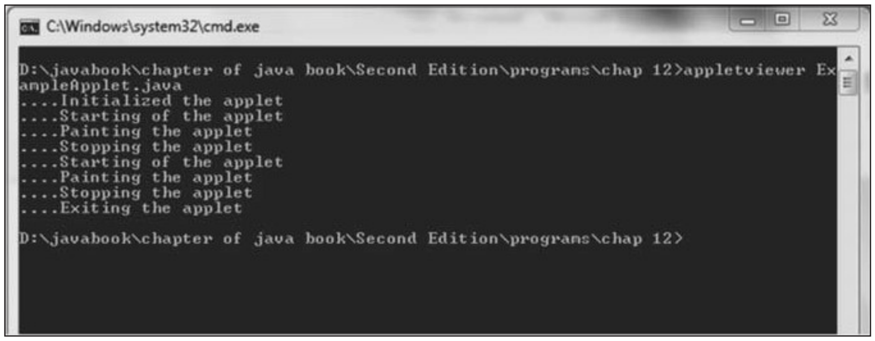
```
/* <APPLET code = “ExampleApplet.class” WIDTH = 200 HEIGHT = 150></APPLET> */
  import java.applet.Applet;
  import java.awt.Color;
  import java.awt.Graphics;
  public class ExampleApplet extends Applet{
  String text;
  public void init() {
     setBackground(Color.white);
     setForeground(Color.red);
     text = “This is an example applet”;
  System.out.println(“....Initialized the applet”); }
L11     public void start() {
L12       System.out.println("....Starting of the applet");
L13     }
L14     public void stop() {
L15       System.out.println("....Stopping the applet");
L16     }
L17     public void destroy() {
L18       System.out.println("....Exiting the applet");
L19     }
L20     public void paint(Graphics g) {
L21       System.out.println("....Painting the applet");
L22       g.drawString(text, 30, 30);
L23       showStatus("This is status bar"); }}
```

Output


Applet View...
Applet
This is an example applet
This is status bar
C:\Windows\system32\cmd.exe - appletviewer ExampleApplet.java

```
D:\javabook\chapter of java book\Second Edition\programs\chap 12>appletviewer Ex
ampleApplet.java
....Initialized the applet
....Starting of the applet
....Painting the applet
```

Fig. 12.4(a) Applet Initialized Using Applet Viewer



```
D:\javabook\chapter of java book\Second Edition\programs\chap 12>appletviewer Ex
ampleApplet.java
....Initialized the applet
....Starting of the applet
....Painting the applet
....Stopping the applet
....Starting of the applet
....Painting the applet
....Stopping the applet
....Exiting the applet

D:\javabook\chapter of java book\Second Edition\programs\chap 12>
```

Fig. 12.4(b) Strings Printed by Various Methods of the Applet

Explanation

L1–3 All the important classes (belonging to their respective packages), whose members are to be used in the applet are imported.

L6–10 This section shows the implementation of `init()`, where the background and foreground of the applet is set to white and red, respectively (see L7–8). White and red are static fields of the `Color` class (part of `java.awt` package). In L9, the text is initialized by a string, `This is an example applet`.

L11–13 These lines account for the implementation of `start()`, responsible for forcing the applet in running state. L13 displays the message about the start of the applet on the screen. Note that this message will not be displayed on the applet window;

it will be displayed as seen by you in earlier chapters (by the use of `System.out.println()`).

L14–16 These lines take care of the implementation of `stop()`. L16 just displays the message about stopping an applet. As many a times you will stop the applet, this message will be displayed on the screen. You can visualize easily that even minimizing the applet window stops or forces the applet into idle state. If restored after getting minimized, it will again invoke `start()` and `paint()`.

L17–19 These lines take care of the implementation of `destroy()`. Try closing the applet window and you will see the message "……Exiting the applet" (L19). Here, this message simply means that the applet is destroyed or has moved to the dead state.

L20–23 These lines are accountable for the implementation of `paint()` method. In L22, `Graphics` object `g` is used to invoke its `drawString()` method, which is actually used for writing on an applet window. See the arguments passed to this method: *text*, which contains the string, "This is an example applet" and the *x*, *y* coordinates from where this text will start in the displayable part of the applet. `paint()` is also called when the window containing applet is covered by another window and they later uncovered. (not minimized and restored)

L22 You can see the method, `showStatus()`, having the text, which has to be shown in the status window of the applet, as argument.

12.7 `paint()`, `update()`, and `repaint()`

All components and containers (since containers are actually components) in the JDK have two methods that are called by the system to paint their surface. These methods are `paint()` and `update()`, belonging to component class (see Fig. 12.1). The signature of these methods are shown below,

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

If you wish that a drawing should appear in a window, you shall override either or both of the methods. Let us discuss these methods in detail.

12.7.1 `paint()` Method

When a component needs to draw/redraw itself, its `paint()` method is called. The component draws itself when it first becomes visible. The component `paint()` method is also invoked when the window containing it is uncovered, if it is covered by another window.

The simplest `paint()` method looks like the following:

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

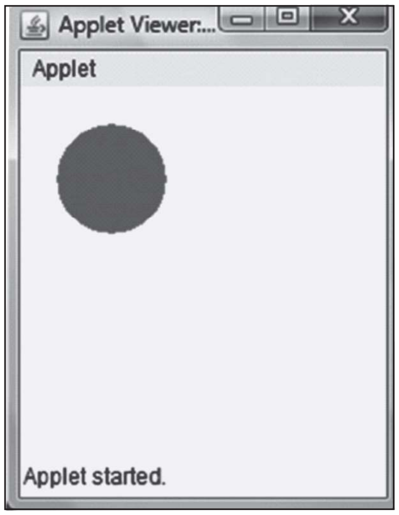
We have discussed the `Graphics` object passed to the method earlier. It will be discussed in more detail in the next chapter.

Example 12.3 **Set the Color of the Applet and Draws a Fill Oval**

```
     /* <APPLET code = "FillOval.class" WIDTH = 200 HEIGHT = 200></APPLET> */
L1     import java.applet.Applet;
L2     import java.awt.Color;
L3     import java.awt.Graphics;
L4     public class FillOval extends Applet
       {
L5     public void paint(Graphics g)
       {
```

```
L6      g.setColor(Color.red);
L7      g.fillOval(20, 20, 60, 60);
L8      }
L9    }
```

Output



Applet started.

Fig. 12.5 Fill Oval with Red Color

Explanation

L5–8 These lines are accountable for the implementation of `paint()` method. The `setColor()` method of `Graphics` class is used to set the drawing color of the applet to red (L6). Another method, `fillOval()`, belonging to the `Graphics` class is invoked at L7. It fills an oval bounded by the specified rectangle with the current color. The parameters passed to the method are the *x*-coordinate, *y*-coordinate, width, and height, respectively.

12.7.2 `update()` Method

Another method which does the same job as `paint()` method, which is called by AWT components to paint its surface is the `update()` method.

Now let us discuss what this method does. It clears the surface of the calling component to its background color and then calls `paint()` to paint the rest of the component. It makes the job easier because one does not have to draw the whole component within a `paint()` method, as the background is already filled. Then, when one overrides `paint()`, he/she only needs to draw what should appear on the foreground.

It simply does not mean that you will never override `update()`. Let us consider a case where you would like to draw a large, red oval inside a window having yellow background. Now take the case where you do not override `update()`, the window's entire background will be drawn by the `update()` method, and then the red oval will be drawn by the `paint()` method. A large area of one color is first drawn and then the large area of the oval in another color (red in this case) is redrawn. You, as a user can see some slight flickering while displaying the result, especially if you try to draw the oval a number of times in succession.

The above problem can be overcome by overriding `update()`. You would override `update()` to call `paint()`. Then this `paint()`will first draw only the background areas surrounding the red oval and then draw the red oval. Obviously, the flickering problem found earlier is removed because of elimination of the drawing of two overlapping objects of different colors.

12.7.3 `repaint()` Method

Sometimes you may want to force a component to be repainted manually. For example, if you have changed certain properties of a component to reflect its new appearance, you can call the `repaint()` method. Here is an example:

```
text.setBackground(Color.blue);
text.repaint();
```

Calling the `repaint()` method causes the whole component to be repainted.

```
repaint() → update() → paint()
```

`repaint()` in its default implementation calls `update()` which in turn calls `paint(). repaint()` method requests the AWT to call `update` and it returns. The AWT combines multiple rapid `repaint` requests into one request (usually this happens when you `repaint` inside a loop). So the last repaint in the sequence actually causes `paint()`. We will discuss these topics in Chapters 13 and 14, where we will illustrate the use of `repaint()` with proper examples.

12.8 MORE ABOUT APPLET TAG

We have used the APPLET tag while writing code for applets. An applet has to be specified in an HTML file. This is done by using APPLET tag in an HTML file. Applets are executed by a Java-enabled web browser as soon as it encounters the APPLET tag inside the HTML file. If you want to view and test an applet using the utility, `appletviewer`, of JDK, you just have to include a comment containing the APPLET tag, just above the actual applet code. We have already discussed how that has to be done.

Till now, we have been using the following form of APPLET tag:

```
<APPLET CODE = filename WIDTH = pixels HEIGHT = pixels></APPLET>
```

This is the most simplified form of APPLET tag, having only the mandatory fields as attributes. Actually this particular tag has many more attributes which are optional but worth discussing. The full syntax of the APPLET tag is shown below.

```
<APPLET [CODEBASE= codebasedURL]
CODE = appletFile [ALT= alternateText] [NAME = appletInstanceName] WIDTH
= pix els HEIGHT = pixels [ALIGN = alignment] [VSPACE = pixels]
[HSPACE = pixels]>
[<PARAM NAME = attributeName VALUE = attributeValue>]
[<PARAM NAME = attributeName VALUE = attributeValue>]
......................................
</APPLET>
```

In the above syntax, the attributes which are put inside the big braces are optional ones. Let us discuss about the use of these attributes in detail.

Codebase Here, we may specify the URL of the directory where the executable class file (specified by CODE attribute) of the applet will be searched for.

Code It gives the name of the file containing the applet's compiled class file. It is a mandatory attribute, which should always be present in APPLET tag.

Alt It is an attribute, which is used to specify the alternate short text message that should be displayed in case the browser recognizes the HTML tag but cannot actually run the applet because of some reason.

Name It is possible to give a name to an applet's instance using this optional attribute. If any other applet on the same web page wants to communicate with this applet, it is referenced through its NAME only.

Width It gives the width of the applet display area in terms of pixels.

Height It gives the height of the applet display area in terms of pixels.

Align This optional attribute is used to set the alignment of an applet. The alignment can be set as LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

Vspace These are used to specify the space, in pixels, above and below the applet.

Hspace These are used to specify the space, in pixels, on each side of the applet.

You can use PARAM tags between the <APPLET> and </APPLET> tags to provide information about parameters, or arguments, to be used by the Java applet. The <PARAM> tag is simple—it NAMES a parameter the JAVA applet needs to run, and provides a VALUE for that parameter.

Note User-defined parameters can be supplied to an applet using <PARAM…….> tags.

This tag has two parameters: NAME and VALUE.
Name Attribute name.
Value Value of the attribute named by corresponding PARAM NAME.
The applets access their attributes using the `getParameter()` method. Its signature is as follows:

```
String getParameter(String name);
```

Let us take an example applet which uses the concept of passing parameters.

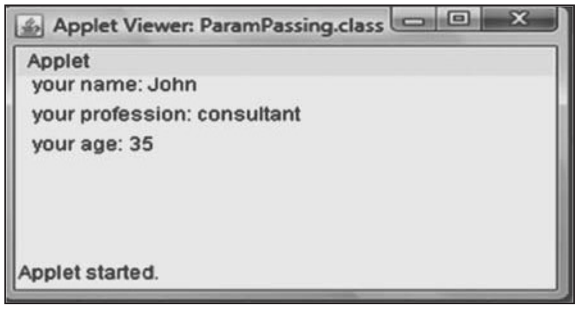
Example 12.4 Param Tag

```
/*<APPLET CODE = ParamPassing.class WIDTH = 300 HEIGHT = 250>
<param NAME = yourName VALUE = John>
<param NAME = yourProfession VALUE = consultant>
<param NAME = yourAge VALUE = 35>
</applet>*/
```

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

```
L3     public class ParamPassing extends Applet {
L4     String name;
L5     String profession;
L6     int age;
L7     public void start() {
L8          String str;
L9          name = getParameter("yourName");
L10    if (name == null) name = "not found";
L11    str = getParameter("yourProfession");
L12    if (str != null) profession = str;
L13    else profession = "No job";
L14    str = getParameter("yourAge");
L15    try {
L16    if (str != null) age = Integer.parseInt(str);
L17    else age = 0;
L18  } catch (NumberFormatException e) {}
L19  }
L20  public void paint(Graphics g) {
L21  g.drawString("your name: "+name, 10, 10);
L22  g.drawString("your profession: "+profession, 10, 30);
L23  g.drawString("your age: " +age, 10, 50);
L24  }}
```

Output



```
your name: John
your profession: consultant
your age: 35
```

```
Applet started.
```

Fig. 12.6 Parameter Passing through Applets

Explanation

Let us start with the APPLET tag placed as comment before the actual code for the program starts. Three PARAM tags are between the start and close of the APPLET tag. All the three PARAM tags have NAME and its corresponding VALUE, such as *yourName* has the value *John*, *yourProfession* has the value *consultant* and *yourAge* has the value 35.

L1–2 For importing necessary classes from their respective packages.

L3 A class named as `ParamPassing` is declared to extend the `Applet` class of the `java.applet` package.

L4–6 References to the `String` class is declared as `name` and `profession` in L4 and L5, respectively. A variable, `age`, of integer type is declared in L6.

L7–19 Implementation of `start()` method is shown. At L9, `getParameter()` method returns the value of the parameter name, `yourName,` passed as argument. The returned value is stored in `name`, declared at L4 as string reference. Similarly, two more `getParameter()` methods are used at L11 and L14, returning the values for the respective parameters names passed as arguments. The use of