07 working with Applets

IN THIS CHAPTER

Until now, we have studied about console-based Java programs. In this chapter, we study about another type of Java programs that allow us to communicate with the world of the Web. Such Java programs are known as applet. An applet is an Internet-based program written in Java programming language, which any computer can download. It is usually embedded in an HTML page on a Web site and can be executed from within a browser. Users are allowed to personalize an applet by using its various settings. For instance, you can insert an applet that works as a menu, and then you can specify an option from this menu to load the corresponding page. The page that contains an applet can be viewed by using the Java enabled browser, such as Internet Explorer. An applet can use various classes, such as Graphic, Font, and Color, to perform various tasks, such as, drawing rectangles and arcs, selecting fonts to write messages, and filling the color in rectangles and arcs. The `Graphics` class creates various shapes, such as lines, and rectangles, in applets. The `Color` class fills or sets the colors into the shapes, and the `Font` class selects the fonts to write text in applets.

Moreover, applets support event handling. Event handling is a process that allows the programmer to control an event occured in a graphical application, such as clicking a mouse button on a window. Here, an event is a mechanism that notifies the user when some action is performed. For example, when you click a mouse button, the mouse related event (mouse clicked) occurs within a graphical application.

In this chapter, you learn about applets, their life cycle, and how applets are different from other Java applications. Next, we explain how to create an applet by using HTML applet tag and various methods, such as `paint()`, `update()`, and `repaint()`; and passing parameters to applets. In addition, this chapter also discusses the `Graphics` class, which draws shapes such as lines and rectangle, in an applet. Further, you learn how the `Color` class is used to set colors in these shapes, and how the `Font` class is used to set various fonts such as Arial, and Sans Serif, in an applet. At the end, you learn how to handle events.

Let's take an overview of applets first.

7.1 Overview of Applets

An applet is a small Internet-based program that has the Graphical User Interface (GUI), written in the Java programming language. Applets are designed to run inside a Web browser or in applet viewer to facilitate the user to animate the graphics, play sounds, and design the GUI components such as text box, button, and radio button. An applet can be embedded into a webpage; much in the same way, an image can be embeded in a page. The page that contains an applet can be viewed by using the Java enabled browser, so that the applet's code is transferred from the source system to the user system and executed by the browser's Java Virtual Machine (JVM).

An applet is required to build dynamic and static Web pages of a Web application. For example, a Web page displays the examination result published on Web site or the reservation status of an Airline application. Applets are helpful in the following areas:

- Applets can load data from a file that is specified with a relative Uniform Resource Locator (URL).
- Applets display documents in Web browser. The `AppletContext` interface provides the `showDocument()` method to display documents in Web browser by using a specified URL.
- Applets are used to send messages on Internet by using the `getApplet()` method of the `AppletContext` interface.
- Applets can play sound files, by using the `AudioClip` interface of the `java.applet` package.

To create an applet (a Java program), first we need to import the packages, such as java.applet, and java.awt, and extend the user class with the Applet class. The java.applet package provides the Applet class that is used to create an applet. The java.awt package provides the Component class, which develops the User Interface (UI) components, such as button and checkbox. This Component class has the paint() method that allows to print a string in an applet.

In this section, we explain the following topics in context of applet:

- Life Cycle of an Applet
- Comparing Applets and Applications

Let's start discussion with the life cycle of an applet.

7.2 Life Cycle of an Applet

An applet executes within the Web browser or in an applet window. When an applet is executed, it goes through the four stages of its life cycle. These stages are:

- Initialization of an applet that loads the applet first time into the memory
- The applet starts running
- The applet stops running
- The applet is unloaded or destroyed from the memory

The life cycle of an applet is basically built on four methods of the Applet class. By calling these methods, a Web browser manages the life cycle of an applet. The following are the methods that involve in life cycle of an applet:

- **The `init()` method:** Loads the applet in the memory. The `public void init()` method is executed only once, that is, when an applet is loaded for the first time. When the init() method is called, then an applet loads in the memory and initializes all variables of this method.
- **The `start()` method:** Starts the execution of the applet. The `public void start()` method is called to start the applet activity. This method executes when every time an applet is displayed on the screen. For example, when a user minimizes a Web page to move to another page, the execution of the `public void start()` method is stopped, whereas maximizing the Web page resumes the execution of this method.
- **The `stop()` method:** Stops the execution of the applet.
- **The `destroy()` method:** Removes the applet from the memory of the computer.

It is not necessary to use these methods while creating an applet, because applet container manages these methods. However, when you use these methods, ensure that they are used in a proper sequence as described.

Moreover, the `main()` method is not required to execute the applets. You can use the following options in Java to run the applets:

- **appletviewer:** Allows you to display an applet on the computer's applet window.
- **Web browser:** Allows you to display an applet embedded in an HTML page. It can be any browser, such as Internet Explorer or Mozilla, but should be Java-enabled.

TEST YOUR KNOWLEDGE

Q1. Write a program to demonstrate life cycle of an applet.

Ans.

```
import java.applet.*;
import java.awt.*;
/*
<applet code="TestYourKnowledge1.class" width=200 height=200>
</applet>
*/
public class TestYourKnowledge1 extends Applet
{
  StringBuffer sb;
  public void init()
  {
      sb = new StringBuffer();
      addItem("Initializing applet...");
  }
  public void start()
  {
      addItem("Starting applet...");
  }
  public void stop()
  {
      addItem("Stopping applet...");
  }
  public void destroy()
  {
```

```
        addItem("Unloading applet...");
    }
    private void addItem(String newText)
    {
        System.out.println(newText);
        sb.append(newText);
        repaint();
    }
    public void paint(Graphics g)
    {
        // draw string contained in sb
        g.drawString(sb.toString(), 5,10);

    }
}
```

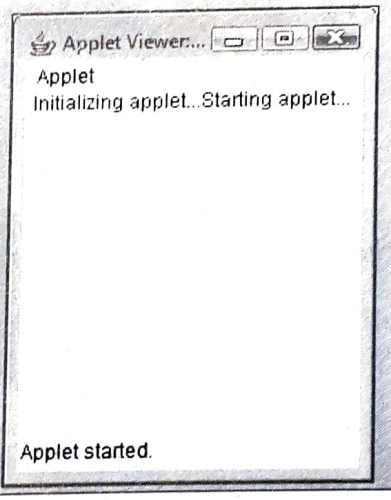
Execution:

```
D:\code\chapter7>Javac TestYourKnowledge1.java
D:\code\chapter7>appletviewer TestYourKnowledge1.html
```

Output:

```
Output of command window:

Initializing applet...
Starting applet...
Stopping applet
Unloading applet...
```

The following figure shows the output of the TestYourKnowledge1 applet:



7.3 Comparing Applets and Applications

An applet and an application have many differences. An application is used to design stand alone application where an applet is used to design Web application. The following are the common differences between applications and applets:

- An application has a special static method called main() method, which starts execution of a Java program. An applet, on the other hand, does not have the main() method.

- When an applet starts running, it inherits or overrides the life cycle methods such as init(), start(), stop(), and destroy(). The Web browser calls life cycle methods of applet at various times during the life cycle of an applet. In contrast, an application has no life cycle method.
- An applet can be embedded in HTML pages whereas an application has no support for HTML.
- An applet can be used to create static and dynamic Web pages whereas an application can be used to create stand alone application, such as network accessing.
- An Application can run with or without graphical user interface whereas an applet must run with a graphical user interface.
- An applet is a Java program that runs under the control of a browser, whereas an application runs as stand-alone application, which needs Java virtual machine for its execution.

Now, we describe how to create an applet.

7.4 Creating Applets

For creating an applet, make sure that Java is installed properly on your system. It is also necessary to ensure that either Java interpreter or Java enabled browser is available on your system, because it needs to run an applet. All applets are subclass of the Applet class, which is available in the `java.applet` package. Therefore, to create an applet, we must import the java.applet package and extend our class from the Applet class. In addition, import the java.awt package, as it is required to provide a user interface to an applet, and so that an applet can have controls such as buttons, checkboxes, graphics, and the rest.

You base applets on the `java.applet.Applet` class, which is itself a subclass of the `java.awt.Container` class as shown in following class hierarchy:

```
java.lang.Object
|____java.awt.Component
|____java.awt.Container
|_____java.awt.Panel
|____java.applet.Applet
```

Table 1 lists the methods of the `Applet` class:

Table 1: Methods of the Applet Class

Method	Description
`void destroy()`	Called by the browser or applet viewer when an applet disposes from the memory.
`AccessibleContext getAccessibleContext()`	Returns the `AccessibleContext` object associated with an applet.
`AppletContext getAppletContext()`	Returns an applet's context.
`String getAppletInfo()`	Returns information, such as copyright and version of an applet.

Table 1: Methods of the Applet Class

Method	Description
`AudioClip getAudioClip (URL url)`	Returns the `AudioClip` object specified by the `URL` argument.
`AudioClip getAudioClip (URL url, String name)`	Returns the `AudioClip` object specified by the `URL` and `name` arguments.
`URL getCodeBase()`	Returns the base URL.
`URL getDocumentBase()`	Returns the document URL.
`Image getImage(URL url)`	Returns an image that can be painted on the screen using the URL argument.
`Image getImage (URL url, String name)`	Returns an image that can be painted on the screen using the URL and name argument.
`Locale getLocale()`	Returns a locale for an applet.
`String getParameter (String name)`	Returns value of the name parameter in the HTML tag.
`String[ ][ ] getParameterInfo()`	Returns information about an applet parameters.
`void init()`	Called by the browser or appletviewer to initialize an applet.
`boolean isActive()`	Determines whether current applet is active or not.
`static AudioClip newAudioClip (URL url)`	Returns an audio clip from the given URL.
`void play(URL url)`	Plays the audio clip at the specified absolute URL.
`void play (URL url, String name)`	Plays an audio clip at the specified absolute URL and location of the audio clip.
`void resize(Dimension d)`	Resizes an applet by using the object of the Dimension class.
`void resize (int width, int height)`	Resizes an applet using width and height parameter.
`void setStub(AppletStub stub)`	Sets an applet's stub using reference of the AppletStub interface.
`void showStatus(String msg)`	Displays a message (contained in msg) in the status window of an applet.
`void start()`	Invokes by Web browser or appletviewer to execute an applet.
`void stop()`	Invokes by Web browser or appletviewer to stop an applet.

We use methods described in Table 1, whenever, we want to add new functionality, such as, showing a message in the status window, playing an audio clip to an applet. Now, we will create an applet. The following are the steps to create an applet:

1. To create an applet, called HelloApplet, first import two packages (java.applet.* and java.awt.*) and then extend the Applet class, as shown here:

```
import java.applet.*;
import java.awt.*;
public class HelloApplet extends Applet { . . . }
```

2. Invoke the paint() method by passing an object of the Graphics class in the HelloApplet applet, as shown here:

```
import java.applet.Applet;
import java.awt.*;
public class HelloApplet extends Applet {
  public void paint(Graphics g) {  }
}
```

3. Invoke the drawString() method in the body of the paint() method to display a message in an applet, as shown here:

```
import java.applet.Applet;
import java.awt.*;
public class HelloApplet extends Applet {
  public void paint(Graphics g) {
      g.drawString("Hello from Java!", 60, 100);
  }
}
```

4. Save the file as HelloApplet.java in D:\codes\chapter7 location and compile it by using the javac command from the command prompt. The following command line shows how to compile the HelloApplet applet from the command prompt:

```
D:\code\chapter7>Javac HelloApplet.java
```

After compiling the HelloApplet.java file, we get the HelloApplet.class file in the same location (D:\code\chapter7).

5. Creates an HTML file named HelloApplet.html (in D:\code\chapter7 location) and write the following code snippet:

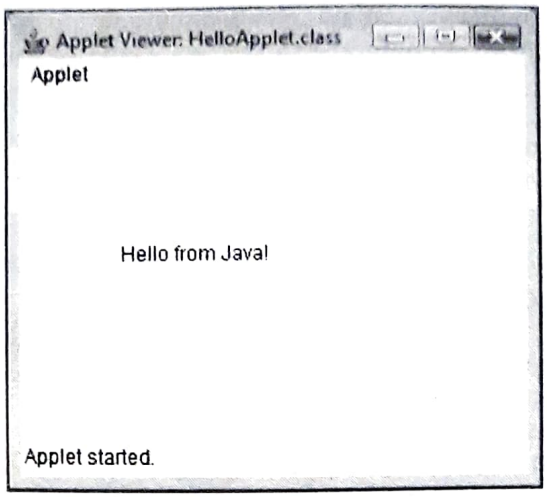
```
<APPLET
  CODE=HelloApplet.class
  WIDTH=300
  HEIGHT=200 >
</APPLET>
```

In the preceding codes snippet, the <applet> tag loads the HelloApplet applet in a window whose width is 300 point, and height is 200 point.

6. Now, execute the HelloApplet applet, by using the appletviewer command from the command prompt, as given in the following command line:

```
D:\code\chapter7>appletviewer HelloApplet.html
```

The output of the HelloApplet applet is shown in Figure 1:



▲ **Figure 1: An Applet at Work in the Appletviewer**

In Figure 1, you can see that the HelloApplet applet draws the text "Hello from Java!" at location (60, 100) in the applet window. Applet coordinates are in pixels; therefore, coordinates (60, 100) prints message (Hello from Java!) at 60 pixels from the left edge of the applet and 100 pixels from the upper edge of the applet.

Further in this section, we discuss about the following topics in context of applets:

- Using the HTML Applet Tag
- Initializing and Terminating an Applet
- Using the paint() Method
- Using the update() Method
- Invoking the repaint() Method
- Implementing Multithreading with Applets
- Passing Parameters to Applets

Let's discuss these in detail one by one.

■ Using the HTML Applet Tag

The APPLET tag loads a compiled applet (class file of an applet) in the applet viewer or in Web browser. Besides this, the `APPLET` tag is used to embed an applet in HTML page. The following syntax is given to use the APPLET tag:

```
<APPLET
  [CODEBASE = URL]
  CODE = filename
  [ALT = alternateText]
  [NAME = instancename]
  WIDTH = pixels
  HEIGHT = pixels
  [ALIGN = alignment]
  [VSPACE = pixels]
  [HSPACE = pixels]
  > [<PARAM NAME = name VALUE = value>]
  . . .
  [<PARAM = NAME name VALUE = value>]
</APPLET>
```

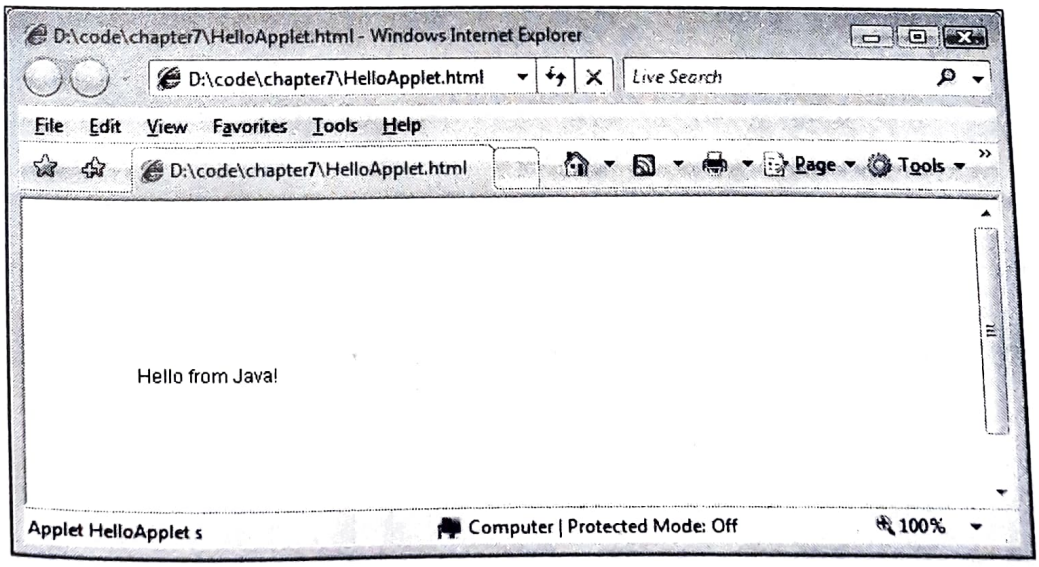
Here are the attributes of the `<APPLET>` tag:

- `CODEBASE`: Specifies the directory in which the applet code is to be found
- `CODE`: Refers to the name of the applet file, including the extension `.class`
- `ALT`: Specifies that if any text used with the ALT tag then this text should be displayed on Web browser, if the applet is not displayed due to any reason
- `NAME`: Refers to the name of the other applet displayed on the Web browser
- `WIDTH`: Refers to the width of the space reserved for the applet
- `HEIGHT`: Refers to the height of the space reserved for the applet
- `ALIGN`: Specifies the alignment of the applet: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, or ABSBOTTOM
- `VSPACE`: Allocates space vertically, above and below the applet
- `HSPACE`: Allocates space horizontally, to the right and left around the applet
- `PARAM NAME`: Refers to the name of a parameter to be passed to the applet
- `PARAM VALUE`: Refers to the value of a parameter

The following code snippet shows how to use the APPLET tag in a Web page:

```
<HTML>
  <BODY>
      <CENTER>
          <APPLET
              CODE=HelloApplet.class
              WIDTH=300
              HEIGHT=200 >
          </APPLET>
      </CENTER>
  </BODY>
</HTML>
```

In the preceding code snippet, we are not specifying a code base, therefore, we place `HelloApplet.class` in the same directory as `applet.html`. The output of applet.html file in the Windows Internet Explorer is shown in Figure 2:



▲ **Figure 2: An Applet Working in the Windows Internet Explorer**

In Figure 2, the text message (Hello from Java!) of the HelloApplet applet is shown in a Web page.

TEST YOUR KNOWLEDGE

Q2. Write an applet to draw an arc with red outline using the Applet tag.

Ans.

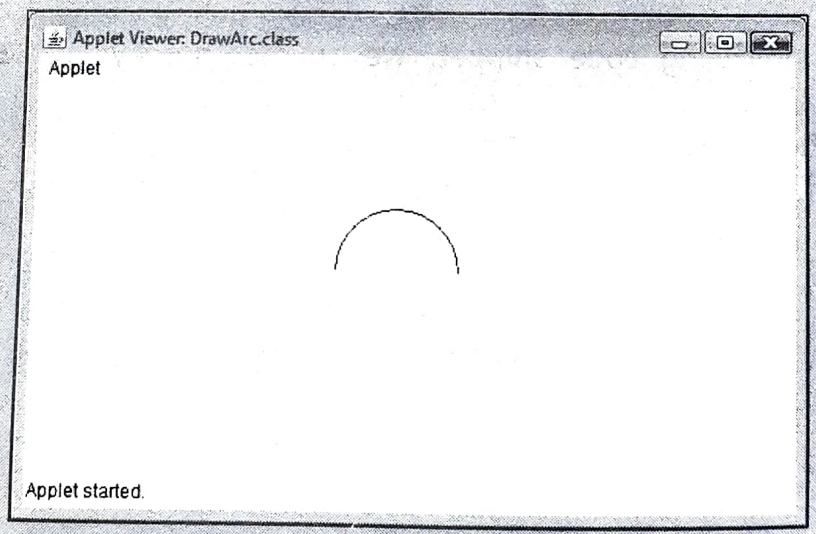
```
import java.awt.*;
import java.applet.*;
/*
<applet code="TestYourKnowledge2.class"
   width=500
   height=250>
</applet>
*/
public class TestYourKnowledge2 extends Applet
{
  public void paint(Graphics g)
  {
      setForeground(Color.red);
      g.drawArc(200, 80, 80, 80, 0, 180);
  }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge2.java
D:\code\chapter7>appletviewer TestYourKnowledge2.html
```

Output:

The following figure shows the output of the TestYourKnowledge2 applet:



Q3. Write an applet to draw A using the drawLine() method using the Applet tag.

Ans.

```
import java.awt.*;
import java.applet.*;
/*
<applet code="TestYourKnowledge3.class"
        width=500
        height=250>
```

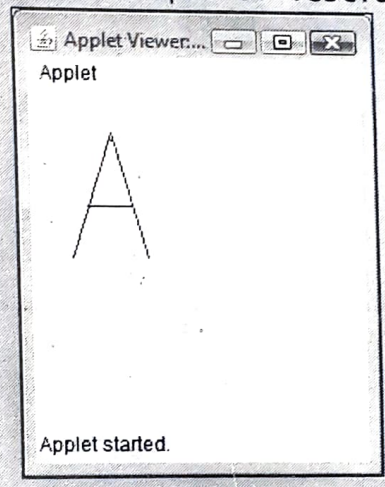
```
</applet>
*/
public class TestYourKnowledge3 extends Applet
{
 public void paint(Graphics g)
 {
     setForeground(Color.red);
     g.drawLine(48, 25, 24, 100);
     g.drawLine(48, 25, 69, 100);
     g.drawLine(33, 69, 59, 69);
 }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge3.java
D:\code\chapter7>appletviewer TestYourKnowledge3.html
```

Output:

The following figure shows the output of TestYourKnowledge3 applet:



Q4. Write a program to display lines in applet window.

Ans.

```
import java.awt.*;
import java.applet.*;
/*
<applet code="TestYourKnowledge4.class"
 width=300
 height=300>
</applet>
*/
public class TestYourKnowledge4 extends Applet
{
 int width, height;
 public void init()
 {
     width = getSize().width;
     height = getSize().height;
 }
 public void paint(Graphics g)
 {
     for ( int i = 0; i < 20; ++i ) {
```

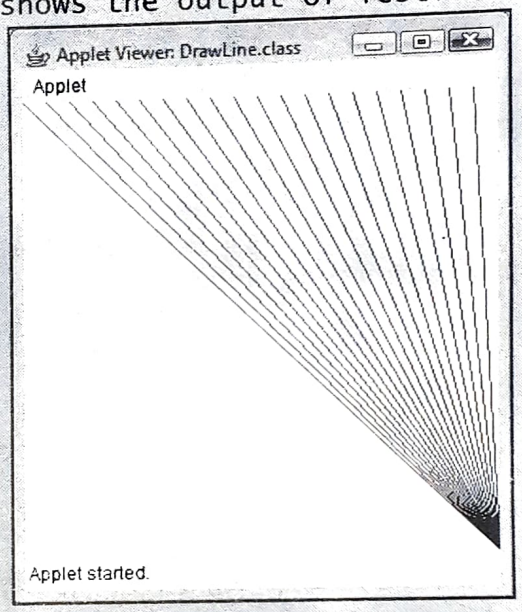
```
                g.drawLine( width, height, i * width / 20, 0 );
        }
    }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge4.java
D:\code\chapter7>appletviewer TestYourKnowledge4.html
```

Output:

The following figure shows the output of TestYourKnowledge4 applet:



Q5. Write an applet to draw a triangle.

Ans.

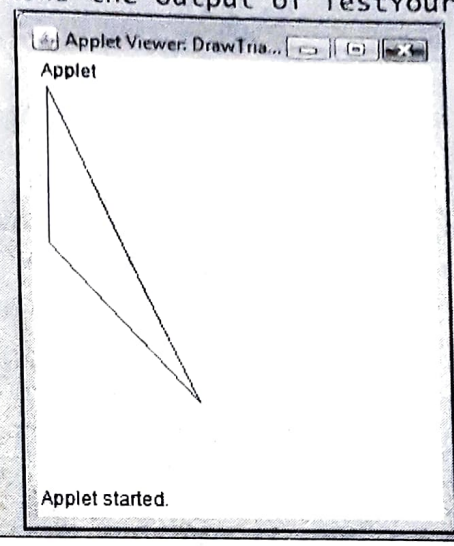
```
import java.awt.*;
import java.applet.*;
/*
<applet code=" TestYourKnowledge5.class"
 width=300
 height=300>
</applet>
*/
public class TestYourKnowledge5 extends Applet
{
 public void paint(Graphics g)
 {
     g.drawLine(10, 0, 10, 100);
     g.drawLine(10, 100, 100, 200);
     g.drawLine(100, 200, 10, 0);
 }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge5.java
D:\code\chapter7>appletviewer TestYourKnowledge5.html
```

Output:

The following figure shows the output of TestYourKnowledge5 applet:



If an applet does not work in your Web browser that means the Web browser does not support the current version of Java.

Initializing and Terminating an Applet

Initialization and termination of an applet is controlled by a set of methods. When an applet is loaded in the memory, it is initialized with some resources, such as variables used in the init() method and when an applet terminates, all the resources occupied by this applet is destroyed. When an applet starts running, AWT calls the following methods in a sequence:

1. init()
2. start()

When an applet is terminated, AWT calls the following methods in a sequence:

1. stop()
2. destroy()

The preceding methods can be overridden by a user to customize them as per the requirement. The following code snippet shows a skeletal implementation of the methods listed previously:

```
import java.applet.Applet;
import java.awt.*;
/* <APPLET
CODE=applet1.class
WIDTH=200
HEIGHT=200 >
</APPLET> */
public class applet1 extends Applet {
  public void init() { setBackground(Color.white); }
  public void start() {
      // called when applet is restarted
  }
  public void paint(Graphics g) {
  g.drawString("Hello from Java!", 60, 100); }
```

```
    public void stop() {
        // is called to suspend the exaction
    }
    public void destroy() {
        // is called when applet is terminated
    }
}
```

In the preceding code snippet, we override the `init()` method to change the background color of the applet to white. The background color is changed by using the setBackground() method of the Component class by passing the white field of the Color class.

■ Using the paint() Method

The paint() is used to display messages in an applet: For displaying messages, we must override the paint() method of the java.awt.Component class in an applet. The paint() method deals with the display of graphics in an applet. This paint() method takes one parameter of type Graphics class, which contains the graphics context to run the applets. This graphics context is used whenever we need to print or draw shapes such as line, and oval, in an applet. The following code snippet shows how to override the paint() method in an applet:

```
import java.applet.Applet;
import java.awt.*;
public class HelloApplet extends Applet
{
  public void paint(Graphics g)
  {
      // do some graphics stuff
  }
}
```

In the preceding code snippet, the paint() method takes one parameter g of type Graphics class to draw all kind of drawings (line, rectangle), and text in an applet.

■ Using the update() Method

The `update()` method is called when a portion of the applet is to be redrawn. This method can be used to make small changes to display an applet. This method is defined in the Component class and is called when an applet needs a portion of window to be redrawn. The default version of the `update()` method simply calls the `paint()` method before filling the default background color of an applet. It means that when you fill the background color using the `paint()` method, a flash of default background is occurred, each time when the `update()` method is called. To avoid this problem, override the `update()` method and then call it within the `paint()` method. The following code snippet shows how to override the `update()` method:

```
import java.applet.Applet;
import java.awt.*;
public class FirstApplet extends Applet
{
  public void update(Graphics g)
  {
      // redisplay your window.
  }
```

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

In the preceding code snippet, the FirstApplet applet overrides the update() method and calls this method within the paint() method. The update() method has one parameter of Graphics type that contains the graphics context to run applet in graphics environment. This graphic context is used to print message and various shapes, such as oval and polygon in an applet.

Invoking the repaint() Method

The `repaint()` method is used whenever an applet needs to update information in its window. AWT defines the `repaint()` method which causes the AWT run-time system to execute the `update()` method of an applet. Further this method calls the paint() method to display the stored information. For example, if we need a string value as an output, we store this string value in a String variable and then call the `repaint()` method. Inside the `paint()` method, this string value will be outputted using the drawString() method. The following are the four forms of the `repaint()` method:

- **void repaint()**: Repaints an entire window of an applet.
- **void repaint(int left, int top, int width, int height)**: Repaints a window of an applet with specified region in terms of left, top, width, height. Using this method, you can update a small portion of a window and repaint only specified region more efficiently.
- **void repaint(long maxDelay)**: Updates a window in the maximum number of milliseconds specified with `maxDelay`.
- **void repaint(long maxDelay, int x, int y, int width, int height)**: Updates a region of window specified with x, y, width, and height coordinating within `maxDelay` milliseconds.

The following code snippet shows how to use the repaint() method:

```
import java.applet.Applet;
import java.awt.*;
public class TestApplet extends Applet
{
  public void repaint(Graphics g)
  {
      // repaint your window.
      update(g)
  }
  public void update(Graphics g)
  {
      paint(g)
  }

  public void paint(Graphics g)
  {
      g.setColor(Color.RED);
  }
}
```

In the preceding code snippet, the repaint() method invokes the update() method and, finally the update() method calls the paint() method to repaint an applet window.

■ Implementing Multithreading with Applets

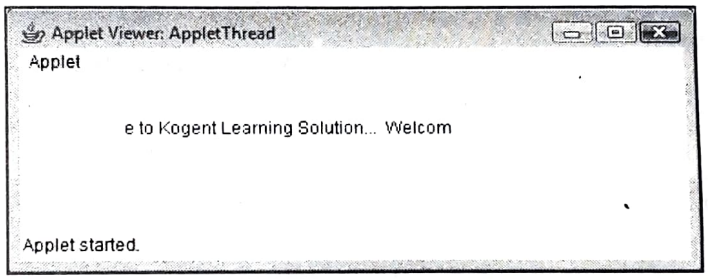
Every applet can run in multiple threads. Many Web browsers allocate a thread for each applet on a page. An applet that performs the same task repeatedly should keep under a thread with `while` or `do while` loop. For this, we keep these repetitive tasks in thread's `run()` method and we use applet's `stop()` method to stop a thread. An example of multithreading is an applet with thread that executes `repaint()` method at regular intervals. Listing 1 shows how to implement multithreading in an applet (you can find the AppletThread.java file on the CD in the code\chapter7 folder):

▶ **Listing 1: Implementing Multithreading with Applet**

```
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletThread" width=500 height=100>
</applet>
*/
public class AppletThread extends Applet implements Runnable
{
  String mssage = " Welcome to Kogent Learning Solution... ";
  Thread th;
  boolean flag;
  // Initialize th thread to null.
  public void init()
  {
      th = null;
  }
  // Start th thread
  public void start()
  {
      th = new Thread(this);
      flag = false;
      th.start();
  }
  // run the thread
  public void run()
  {
      char ch;
      // Display message
      for( ; ; )
      {
          try
          {
              repaint();
              Thread.sleep(250);
              ch = mssage.charAt(0);
              message = mssage.substring(1, mssage.length());
              mssage += ch;
              if(flag)
              break;
          } catch(InterruptedException exc) {}
      }
```

```
    }
    // Pause the message.
    public void stop()
    {
        flag = true;
        th = null;
    }
    // Display the message.
    public void paint(Graphics g)
    {
        g.drawString(mssage, 70, 40);
    }
}
```

In Listing 1, the AppletThread class extends the `Applet` class and implements the `Runnable` interface to implement multithreading in applet. In this AppletThread class, the `init()` method initializes the th thread with null and the `start()` method starts the th thread. Within the `run()` method, the message (Welcome to Kogent Learning Solution...) is repeatedly moved from right to left using the `repaint()` method. This `repaint()` method calls the `paint()` method to print the message (Welcome to Kogent Learning Solution...) on your screen. At last, the `stop()` method sets the flag boolean variable to true and initializes the th thread with null value. Figure 3 shows the output of Listing 1:



▲ **Figure 3: The AppletThread is Moving Message from Right to Left**

In Figure 3, the AppletThread is loaded using appletviewer command and the message (Welcome to Kogent Learning Solution...) is moved from right to left direction.

TEST YOUR KNOWLEDGE

Q6: Write a program to animate a circle (ball).

Ans.

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Rectangle;
/*
<applet code=" TestYourKnowledge6.class"
 width=400
 height=400>
</applet>
*/

public class TestYourKnowledge6 extends Applet implements Runnable
{
  int x = 150, y = 50, r = 30; // Position and radius of the circle
  int dx = 11, dy = 7; // Trajectory of circle
```

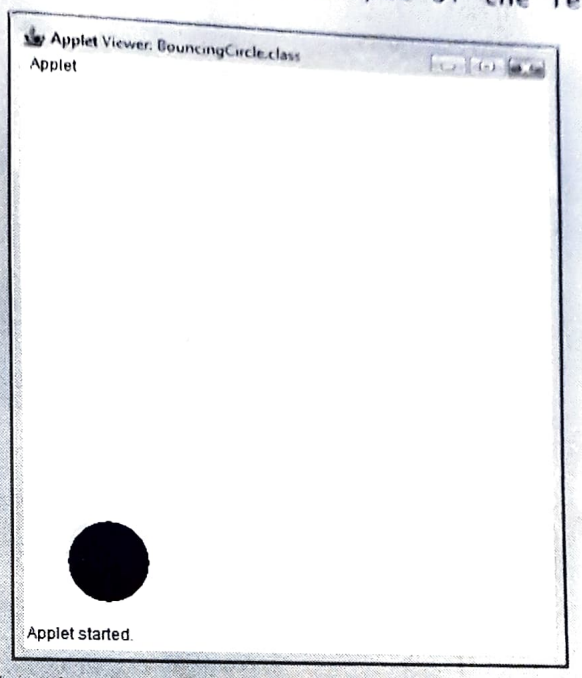
```
    Thread animatorTh; // The thread that perform the animation
    volatile boolean stopTh; // A flag to stop thread

    /** This method simply draws the circle at its current position */
    public void paint(Graphics g)
    {
        g.setColor(Color.blue);
        g.fillOval(x - r, y - r, r * 2, r * 2);
    }
    // This method moves circle.
    public void animateCircle()
    {
        // Bounce if hit an edge.
        Rectangle bounds = getBounds();
        if ((x - r + dx < 0) || (x + r + dx > bounds.width))
        dx = -dx;
        if ((y - r + dy < 0) || (y + r + dy > bounds.height))
        dy = -dy;

        // Move the circle.
        x += dx;
        y += dy;

        repaint(); // to draw the circle in its new position
    }
    //This method is used to perform animation.
    public void run()
    {
        while (!stopTh)
        {
            animateCircle(); // Update and request redraw
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {   }
        }
    }
    // Start animating when the browser starts the applet
    public void start()
    {
        animatorTh = new Thread(this); // Create a thread
        stopTh = false;
        animatorTh.start(); // Start the thread.
    }
    // Stop animating when the browser stops the applet
    public void stop()
    {
        stopTh = true;
    }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge6.java
D:\code\chapter7>appletviewer TestYourKnowledge6.html
```

Output:

The following code snippet shows the output of the TestYourKnowledge6 applet:



Q7. Write an applet to display time in an applet window.

Ans.

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.Label;
import java.text.DateFormat;
import java.util.Date;
/*
<applet code="TestYourKnowledge7.class"
 width=200
 height=100>
</applet>
*/
public class TestYourKnowledge7 extends Applet implements Runnable
{
 Label showTime;
 DateFormat timeFormat;
 Thread timerTh;
 volatile boolean running;
 public void init()
 {
     showTime = new Label();
     showTime.setFont(new Font("helvetica", Font.BOLD, 20));
     showTime.setAlignment(Label.CENTER);
     setLayout(new BorderLayout());
     add(showTime, BorderLayout.CENTER);
     timeFormat = DateFormat.getTimeInstance(DateFormat.MEDIUM);
 }
 public void start()
 {
     running = true;
     if (timerTh == null)
     {
```

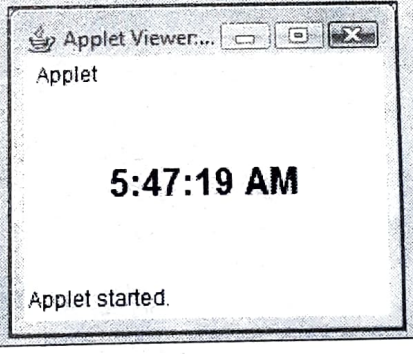
```
            timerTh = new Thread(this);
            timerTh.start();
        }
    }
    public void run()
    {
        while (running)
        {
            showTime.setText(timeFormat.format(new Date()));
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException e)
            { }
        }
        timerTh = null;
    }
    public void stop()
    {
        running = false;
    }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge7.java
D:\code\chapter7>appletviewer TestYourKnowledge7.html
```

Output:

The following code snippet shows the output of TestYourKnowledge7 applet:



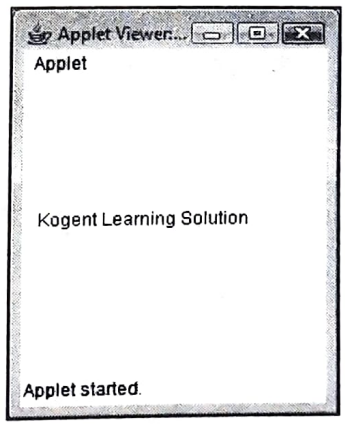
Passing Parameters to Applets

We pass parameters in an applet using the <APPLET> tag. An applet code reads the values of those parameters using the `getParameter()` method. For this, we have to only supply different parameters in the `<APPLET>` tag. To actually get the value of a parameter, we use the `getParameter()` method of the `Applet` class to pass the name of the parameter as specified in the `<PARAM>` attribute. The `getParameter()` method returns the value of the parameter that is set in the `<PARAM>` attribute. Listing 2 shows how to pass parameter to an applet (you can find the AppletParameterPassing.java file on the CD in the code\chapter7 folder):

▶ **Listing 2: Passing Parameter to an Applet**

```
import java.applet.Applet;
import java.awt.*;
/* <APPLET
  CODE=AppletParameterPassing.class
  WIDTH=200
  HEIGHT=200 >
  <PARAM NAME = company VALUE = "Kogent Learning Solution">
</APPLET>  */
public class AppletParameterPassing extends Applet
{
  String company;
  public void start()
  {
      company = getParameter("company");
  }
  public void paint(Graphics g)
  {
      g.drawString(company, 10, 100);
  }
}
```

In Listing 2, we pass a parameter, named company, to the AppletParameterPassing applet. The value of this parameter is string message (Kogent Learning Solution) and this applet displays the string message using the paint() method. Figure 4 shows the output of Listing 2:



▲ **Figure 4: Parameter Passing to an Applet**

In Figure 4, the text message (Kogent Learning Solution) is displayed by using the PARAM tag (<PARAM NAME = company VALUE = "Kogent Learning Solution">) and the `getParameter()` method.

TEST YOUR KNOWLEDGE

Q8: Write an applet to pass a parameter to an applet.

Ans.

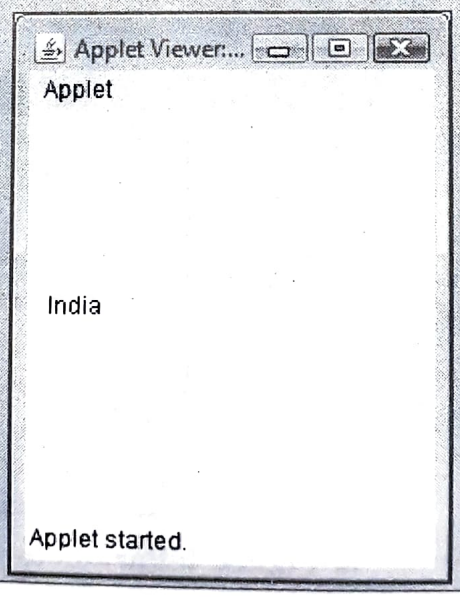
```
import java.applet.Applet;
import java.awt.*;
/* <APPLET
  CODE= TestYourKnowledge8.class
  WIDTH=200
  HEIGHT=200 >
  <PARAM NAME = country VALUE = "India">
</APPLET>>  */

public class TestYourKnowledge8 extends Applet
{
  String country;
  public void start()
  {
      country = getParameter("country");
  }
  public void paint(Graphics g)
  {
      g.drawString(country, 10, 100);
  }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge8.java
D:\code\chapter7>appletviewer TestYourKnowledge8.html
```

Output:

The following code snippet shows the output of the TestYourKnowledge8 applet:



Now, we know how to write applet using various methods such as `update()`, and `paint()`. Next, we use the Graphics class to draw various shapes such as line, rectangle in applets.

7.3 Working with the Graphics Class

The Graphics class is used to draw different shapes on an applet. These shapes include single lines, rectangles, ovals, and the rest. In addition, the Graphics class provides the methods to fill colors in the various shapes such as, rectangle, oval. Table 2 lists various methods of the Graphics class:

Table 2: Methods of the Graphics Class

Method	Description
`abstract void clearRect(int x, int y, int width, height)`	Clears a rectangle by filling with background color.
`abstract void clipRect (int x, int y, int width, int height)`	Intersects a clip region with the specified rectangle.
`abstract void copyArea (int x, int y, int width, int height, int dx, int dy)`	Copies an area of the component as specified by a region given by dx and dy. Here, dx is the horizontal distance to copy the pixels and dy is the vertical distance to copy the pixels.
`abstract Graphics create()`	Creates a new `Graphics` object.
`Graphics create(int x, int y, int width, int height)`	Creates a new `Graphics` object based on a clip area specified with x, y, width, and height parameter.
`abstract void dispose()`	Disposes a graphics context.
`void draw3DRect(int x, int y, int width, int height, boolean raised)`	Draws a 3D highlighted outline of a rectangle.
`abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`	Draws a circular or elliptical arc.
`void drawBytes (byte[] data, int offset, int length, int x, int y)`	Draws text given by the byte array.
`void drawChars (char[] data, int offset, int length, int x, int y)`	Draws the text given by the character array.
`abstract boolean drawImage (Image img, int x, int y, Color bgcolor, ImageObserver observer)`	Draws image specified with img, x, y, bgcolor, and observer parameters.

Table 2: Methods of the Graphics Class

Method	Description
`abstract boolean drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)`	Draws image specified with img, x, y, and bgcolor parameters.
`abstract void drawLine (int x1, int y1, int x2, int y2)`	Draws a line using current color, between the points (x1, y1) and (x2, y2).
`abstract void drawOval(int x, int y, int width, int height)`	Draws an oval.
`abstract void drawPolygon (int[] xPoints, int[] yPoints, int nPoints)`	Draws a closed polygon.
`void drawPolygon (Polygon p)`	Draws the outline of a polygon defined by the given `Polygon` object.
`abstract void drawPolyline (int[] xPoints, int[] yPoints, int nPoints)`	Draws a sequence of connected lines defined by arrays of x and y coordinates.
`void drawRect(int x, int y, int width, int height)`	Draws a rectangle.
`abstract void drawRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)`	Draws a rounded-corner rectangle.
`abstract void drawString (AttributedCharacterIterator iterator, int x, int y)`	Draws text given by the `AttributedCharacterIterator` interface.
`abstract void drawString (String str, int x, int y)`	Draws a text string.
`void fill3DRect(int x, int y, int width, int height, boolean raised)`	Paints a 3D highlighted rectangle filled with current color.
`abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`	Fills a circular or elliptical arc covering with given rectangle.
`abstract void fillOval (int x, int y, int width, int height)`	Fills an oval covering with a given rectangle.
`abstract void fillPolygon (int[] xPoints, int[] yPoints, int nPoints)`	Fills a closed polygon defined by arrays of x and y coordinates.

Table 2: Methods of the Graphics Class

Method	Description
`void fillPolygon (Polygon p)`	Fills the polygon defined by the given `Polygon` class with current color.
`abstract void fillRect(int x, int y, int width, int height)`	Fills the given rectangle.
`abstract void fillRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)`	Fills the given rounded-corner rectangle.
`void finalize()`	Handles garbage collection of a graphics context.
`abstract Shape getClip()`	Gets the clipping area of an arbitrary shape.
`abstract Rectangle getClipBounds()`	Gets bounded rectangle of an arbitrary shape.
`Rectangle getClipBounds (Rectangle r)`	Gets bounded rectangle of a clipping area with r parameter.
`abstract Color getColor()`	Gets the current color of a graphics context.
`abstract Font getFont()`	Gets a font.
`FontMetrics getFontMetrics()`	Gets a font metrics of a font.
`abstract FontMetrics getFontMetrics(Font f)`	Gets a font metrics of a font.
`boolean hitClip(int x, int y, int width, int height)`	Returns true if the rectangular area intersects the rectangle of a clipping area.
`abstract void setClip(int x, int y, int width, int height)`	Sets a clip to a rectangle shape formed by the coordinates passed as parameter.
`abstract void setClip (Shape clip)`	Sets a clipping area to an arbitrary clip shape specified with the object (clip) of the Shape Interface.
`abstract void setColor (Color c)`	Sets this graphics context's color to the given color.
`abstract void setFont (Font font)`	Sets this graphics context's font to given font.
`abstract void setPaintMode()`	Sets the paint mode of a graphics context with current color.
`abstract void setXORMode (Color c1)`	Sets a paint mode to alternate between graphics context's current color and new given color using XOR operation.
`abstract void translate (int x, int y)`	Translates the origin of a graphics context with the specified point `(x, y)`.

We can use methods listed in Table 2 to draw shape, such as line, rectangle, oval, arc, and polygon, and to get the current font and color of an applet.

TEST YOUR KNOWLEDGE

Q9: Write an applet to draw an oval filled with blue color.

Ans.

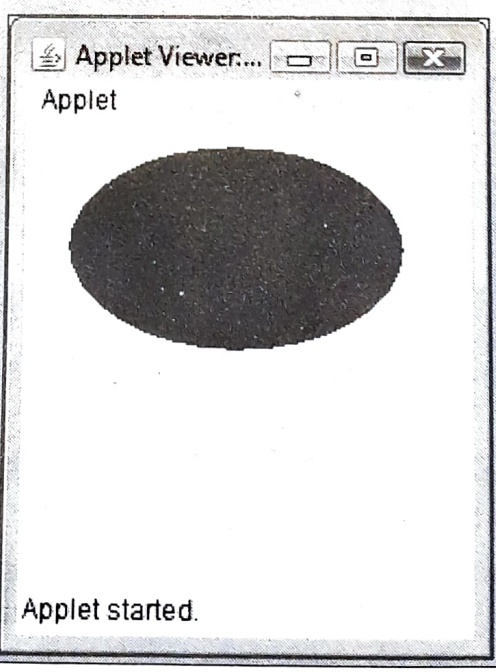
```
import java.awt.*;
import java.applet.*;
/*
<applet code="TestYourKnowledge9.class"
   width=200
   height=200>
</applet>
*/
public class TestYourKnowledge9 extends Applet
{
 public void paint(Graphics g)
 {
     setForeground(Color.blue);
     g.fillOval(20, 10, 150, 90);
 }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge9.java
D:\code\chapter7>appletviewer TestYourKnowledge9.html
```

Output:

The following figure shows the output of the TestYourKnowledge9 applet:



In the next section we draw following shapes by using the Graphics class:

- Lines
- Rectangles

- Ovals
- Arcs
- Polygon

Let's discuss these in detail one by one.

Drawing Lines

The `drawLine()` method of the Graphics class is used to draw a line. We can draw a line between (x1, y1) point and (x2, y2) points. The syntax to use the `drawLine()` method is:

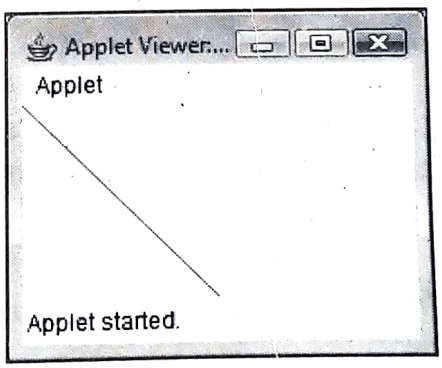
```
drawLine(int x1, int y1, int x2, int y2);
```

Here, the `drawLine()` method draws a line from the x1 and y1 point to the x2 and y2 point. Listing 3 shows the usage of the `drawLine()` method (you can find the DrawLine.java file on the CD in the code\chapter7 folder):

▶ **Listing 3: The DrawLine.java File**

```
import java.awt.*;
import java.applet.*;
/*
<applet code="DrawLine.class"
    width=200
    height=100>
</applet>
*/
public class DrawLine extends Applet
{
  public void paint(Graphics g)
  {
      g.drawLine(0, 0, 100, 100);
  }
}
```

In Listing 3, the DrawLine applet invokes the `drawLine()` method that draws a line starts with point (0,0) and ends with points (100,100). Figure 5 shows the output of Listing 3:



▲ **Figure 5: Drawing a Line Using the drawLine() Method**

Drawing Rectangles

The `drawRect()` method of the Graphics class is used to draw a rectangle. The syntax to use the `drawRect()` method is:

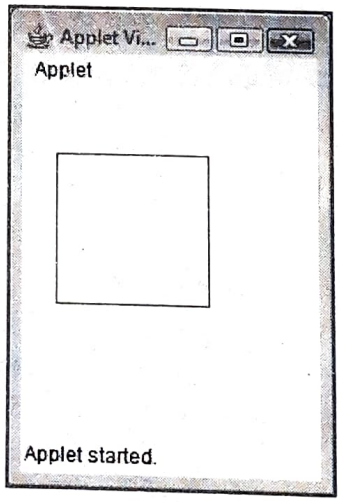
```
void drawRect(int top, int left, int right, int bottom)
```

Here, the `drawRect()` method draws a rectangle whose upper-left corner is specified with top, and left parameters and bottom-right corner is specified with right, and bottom parameters. Listing 4 shows usage of the drawRect() method (you can find the DrawRectangle.java file on the CD in the code\chapter7 folder):

▶ **Listing 4: The DrawRectangle.java File**

```
import java.awt.*;
import java.applet.*;
/*
<applet code="DrawRectangle.class"
  width=200
  height=250>
</applet>
*/
public class DrawRectangle extends Applet
{
  public void paint(Graphics g)
  {
      g.drawRect(20, 40, 90, 90);
  }
}
```

In Listing 5, the g object of the Graphics class invokes the `drawRect()` method that draws a rectangle from 20,40 corner to 90,90 corner. Figure 6 shows the output of Listing 4:



▲ **Figure 6: Drawing a Rectangle Using drawLine() Method**

■ Drawing Ovals

The `drawOval()` method of the Graphics class is used to draw an oval. The syntax to use the `drawOval()` method is:

```
void drawOval(int top, int left, int right, int bottom)
```

Here, the `drawOval()` method draws an oval whose upper-left corner is specified with top, and left parameters and bottom-right corner is specified with right, and bottom parameters.

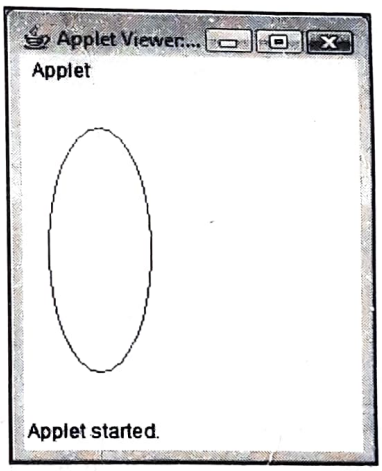
Listing 5 shows usage of the drawOval() method (you can find the DrawOval.java file on the CD in the code\chapter7 folder):

▶ **Listing 5: The DrawOval.java File**

```
import java.awt.*;
import java.applet.*;
/*
<applet code="DrawOval.class"
 width=200
 height=200>
</applet>
*/
public class DrawOval extends Applet
{
 public void paint(Graphics g)
 {
     g.drawOval(15, 25, 60, 150);
 }
}
```

In Listing 5, the DrawOval applet invokes the drawOval() method that draws an oval starts at 15,25 coordinate and ends at 60,150 coordinate.

Figure 7 shows the output of Listing 5:



▲ **Figure 7: Drawing an Oval Using drawOval() Method**

Drawing Arcs

The `drawArc()` method of the Graphics class is used to draw an arc. The syntax to use the `drawArc()` method is:

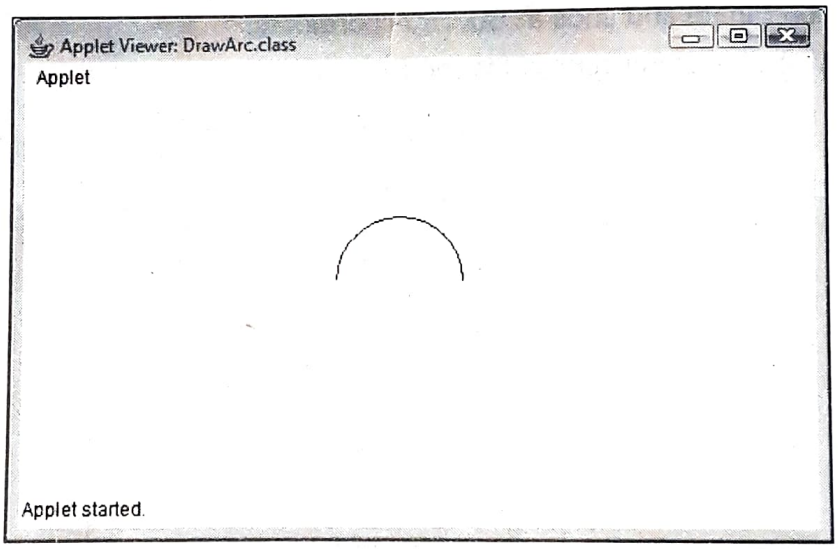
```
void drawArc(int top, int left, int right, int bottom, int startAngle, int
 endAngle)
```

Here, the `drawArc()` method draws an arc whose upper-left corner is specified with top, and left parameters and bottom-right corner is specified with right, and bottom parameters. The angle of an arc is specified with startAngle, and endAngle parameters. Listing 6 shows the usage of the drawArc() method (you can find the DrawArc.java file on the CD in the code\chapter7 folder):

▶ **Listing 6: The DrawArc.java File**

```
import java.awt.*;
import java.applet.*;
/*
<applet code="DrawArc.class"
 width=500
 height=250>
</applet>
*/
public class DrawArc extends Applet
{
 public void paint(Graphics g)
 {
     g.drawArc(200, 80, 80, 80, 0, 180);
 }
}
```

In Listing 6, the object of `Graphics` class (g) invokes the `drawArc()` method that draws an arc started at coordinate 200,80 and ended at coordinate 80,80. The starting and ending angles of this arc is 0 degree and 180 degree. Figure 8 shows the output of Listing 6:



▲ **Figure 8: Drawing an Arc Using drawArc() Method**

■ Drawing Polygon

The `drawPolygon()` method of the Graphics class is used to draw an arbitrary shape. It is used to draw a polygon with n sides, where n is number of sides of a polygon. The syntax to use the `drawPolygon()` method is:

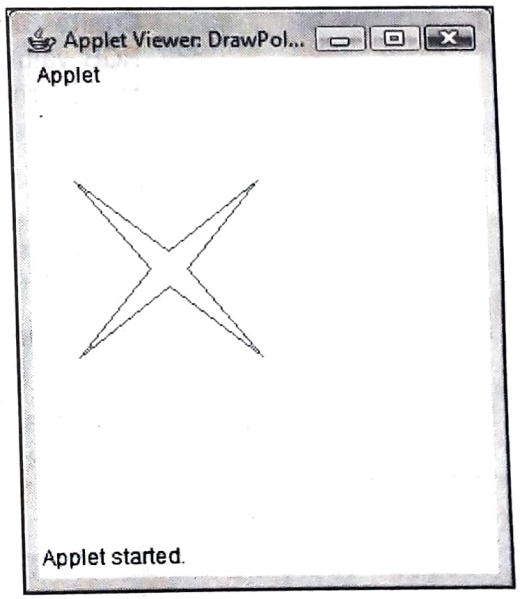
```
void drawPolygon(int xPoints[ ], int yPoints[ ], int numPoints)
```

Here, the `drawPolygon()` method draws a polygon whose starting and ending positions are specified by the pairs of co-ordinates contained within the xPoints and yPoints arrays. The number of elements defined in the xPoints or yPoints arrays is equal to the value of the numPoints variable. Listing 7 shows the usage of the drawPolygon() method (you can find the DrawPolygon.java file on the CD in the code\chapter7 folder):

▶ **Listing 7: The DrawPolygon.java File**

```
import java.awt.*;
import java.applet.*;
/*
<applet code="DrawPolygon.class"
 width=250
 height=250>
</applet>
*/
public class DrawPolygon extends Applet
{
  public void paint(Graphics g)
  {
      int xpoints[] = {25, 75, 125, 85, 125, 75, 25, 65};
      int ypoints[] = {50, 90, 50, 100, 150, 110, 150, 100};
      int num = 8;
      g.drawPolygon(xpoints, ypoints, num);
  }
}
```

In Listing 7, the DrawPolygon applet invokes the `drawPolygon()` method that draws a polygon of 8 sides. Figure 9 shows a polygon shape drawn on an applet:



▲ **Figure 9: Drawing a Polygon Using drawPolygon() Method**

Now, we know how to draw rectangle, oval and other shapes. Let's see how to fill color in these shapes.

7.6 Working with the Color Class

The `java.awt.Color` class is used to create new color, or use some predefined color constants, such as red, green in an applet or a standalone application. Java allows us to create up to 24 million different colors. Table 3 lists the fields of the `Color` class. These fields are used to set a color or get the current color of an applet:

Table 3: Fields of the Color Class		
static Color black	static Color blue	static Color cyan
static Color darkGray	static Color gray	static Colorgreen
static Color lightGray	static Color magenta	static Color orange
static Color pink	static Color red	static Color white
static Color yellow		

Table 4 lists the constructors of the Color class:

Table 4: Constructors of the Color Class	
Constructor	**Description**
`Color(ColorSpace cspace, float[] components, float alpha)`	Creates a color in the color space of the supplied `ColorSpace` class.
`Color(float r, float g, float b)`	Creates an opaque color with the given red, green, and blue values in the range 0.0 to 1.0.
`Color(float r, float g, float b, float a)`	Creates a color with the given red, green, blue, and alpha values in the range 0.0 to 1.0.
`Color(int rgb)`	Creates an opaque color with the given combined RGB value, consisting of the red value in bits 16 to 23, the green value in bits 8 to 15, and the blue value in bits 0 to 7.
`Color(int rgba, boolean hasalpha)`	Creates a color with the given combined RGB value, consisting of the alpha value in bits 24 to 31, the red value in bits 16 to 23, the green value in bits 8 to 15, and the blue value in bits 0 to 7.
`Color(int r, int g, int b)`	Creates an opaque color with the given red, green, and blue values in the range 0 to 255.
`Color(int r, int g, int b, int a)`	Creates a color with the given red, green, blue, and alpha values in the range 0 to 255.

Table 5 lists the methods of the Color class:

Table 5: Methods of the Color Class	
Method Does	**Description**
`Color brighter()`	Makes a brighter version of a color.
`PaintContext createContext(ColorModel cm, Rectangle r, Rectangle2D r2d, Affine Transform xform, RenderingHints hints)`	Creates and returns a paint context to generate a pattern in solid color.
`Color darker()`	Makes a darker version of a color.

Table 5: Methods of the Color Class

Method Does	Description
`static Color decode (String nm)`	Converts a string into an integer and returns the specified color.
`boolean equals(Object obj)`	Determines whether another color is equal to a specified color.
`int getAlpha()`	Gets a alpha value in the range of 0-255.
`int getBlue()`	Gets a blue value.
`static Color getColor (String nm)`	Finds a color in the system properties specified with the nm string.
`static Color getColor (String nm, Color v)`	Finds a color in the system properties specified with the nm string, and a color specified with the v field.
`static Color getColor (String nm, int v)`	Finds a color in the system properties.
`float[] getColorComponents (ColorSpace cspace, float[] compArray)`	Gets a float array containing the color components of the `Color` object in the ColorSpace class.
`float[] getColorComponents (float[]compArray)`	Gets a float array containing only the color components of the Color class.
`ColorSpace getColorSpace()`	Gets the color space of the `Color` object.
`float[] getComponents (ColorSpace cspace, float[] compArray)`	Gets a float array containing the color and alpha components of the Color class specified with an object of the ColorSpace class.
`float[] getComponents (float[ ] compArray)`	Gets a float array containing a color and alpha components of the `Color` object.
`int getGreen()`	Gets the green value in the range of 0-255.
`static Color getHSBColor (float h, float s, float b)`	Creates a `Color` object based on HSB (hue, saturation, and brightness) values.
`int getRed()`	Gets the red value in the range of 0-255.
`int getRGB()`	Gets the RGB value representing a color in the default color model.
`float[] getRGBColorComponents (float[]compArray)`	Gets a float array containing only the color components of the Color class.
`float[] getRGBComponents (float[]compArray)`	Gets a float array containing color components and alpha components of the Color class.
`int getTransparency()`	Returns the transparency mode for a color.
`int hashCode()`	Computes the hashcode for a color.
`static int HSBtoRGB(float hue, float saturation,float brightness)`	Converts the components of a color given by a HSB model into the default RGB model.

Table 5: Methods of the Color Class

Method Does	Description
`static float[ ] RGBtoHSB (int r, int g, int b, float[] hsbvals)`	Converts the components of a color, as given by the RGB model into an equivalent set of values for the HSB model.
`String toString()`	Gets a string representation of a color.

To create a new color, we need to pass the field (red, green, blue) of the `Color` class as parameters of the Color's constructor. The following code snippet shows how to create color:

```
Color c = new Color(red, green, blue);
```

Now, you are free to set this new color as the drawing color using `setForeground()` method as shown in the following code snippet:

```
setForeground(c);
```

Now, drawing operations takes place in the color (c) we have specified. We can also use a predefined color, such as `Color.blue`, as in this case, where we are setting the background color. he following code snippet shows how to use predefined color:

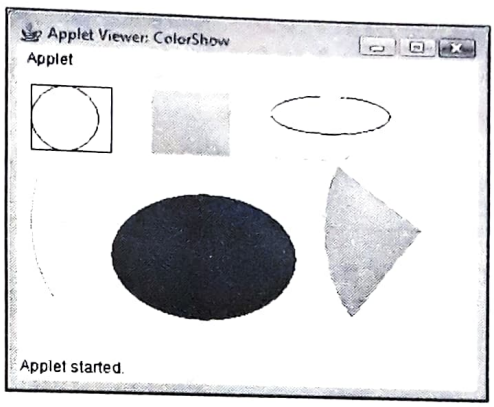
```
setBackground(Color.blue);
```

In fact, we can also fill color in figures using the `Graphics` methods, such as `fillArc()`, and `fillOval()`. Listing 8 shows how to use colors in an applet (you can find the ColorShow.java file on the CD in the code\chapter7 folder):

▶ **Listing 8: The ColorShow.java File**

```
import java.awt.*;
import java.applet.*;
/* <applet code ="ColorShow" width=300 height=200>
</applet> */
public class ColorShow extends Applet
{
 public void paint(Graphics g)
 {
     g.setColor(Color.red);
     g.drawOval(10, 10, 50, 50);
     g.fillOval(70, 90, 140, 100);
     g.setColor(Color.blue);
     g.drawOval(190, 10, 90, 30);
     g.drawRect(10, 10, 60, 50);
     g.setColor(Color.cyan);
     g.fillRect(100,10,60,50);
     g.drawRoundRect(190, 10, 60, 50, 15, 15);
     g.setColor(Color.green);
     g.drawArc(10, 20, 150, 190, 160,60);
     g.fillArc(230, 15, 150, 200, 150, 75);
 }
}
```

In Listing 8, the ColorShow applet invokes the `setColor()` method, several times, to set various colors for different shapes such as oval, rectangle, arc, and circle. Figure 10 shows various shapes with different colors:



▲ **Figure 10: Filling Objects with Color.**

Now, we know how to use colors in graphics. Let's see how to use the Font class to draw the text.

7.7 Working with the Font Class

The Font class is used to represent font to draw text, label, text field, and button. The name of the font is expressed as family name, such as Courier. Category of font is specified with logical name, such as Moonscape and the face name is specified with a specific font, such as Courier Italic.

We can select the type and style of text fonts with the `Font` class. Using this class, we can select a font (such as Roman, Arial, or Courier), set its size, and specify its styles such as bold, italic, and regular. Table 6 lists the fields of the `Font` class. These fields are used to set different style and size of the font:

Table 6: Fields of the Font Class

Field	Description
`static int BOLD`	The bold style.
`static int CENTER_BASELINE`	The baseline used in ideographic scripts such as Japanese.
`static int HANGING_BASELINE`	The baseline used in laying out scripts such as Devanagari.
`static int ITALIC`	The italicized style.
`Protected String name`	The logical name of a font.
`static int PLAIN`	The plain style.
`protected float pointSize`	The point size of a font in a float.
`static int ROMAN_BASELINE`	The baseline used in most roman scripts in laying out text.
`protected int size`	The point size of a font.
`protected int style`	The style of a font.

Table 7 lists the constructors of the Font class:

Table 7: Constructors of the Font Class

Constructor	Description
`Font (Map attributes)`	Creates a new font with specified attributes.
`Font(String name, int style, int size)`	Creates a new font from a given name, style, and point size.

Table 8 lists the methods of the Font Class:

Table 8: Methods of the Font Class

Method	Description
boolean canDisplay(char c)	Checks whether a font has a glyph for a given character (c).
int canDisplayUpTo(char[ ] text, int start, int limit)	Indicates whether this font can display the characters in the given text or not.
int canDisplayUpTo(CharacterIterator iter, int start, int limit)	Indicates whether a font can display a string or not in the specified reference (iter) of the `CharacterIterator` interface, starting at start and ending at limit parameters.
int canDisplayUpTo (String str)	Indicates whether a font can display a given string or not in the specified str parameter.
static Font createFont (int fontFormat, InputStream fontStream)	Returns a new font with the specified font type.
GlyphVector createGlyphVector(FontRenderContext frc, char[ ] chars)	Gets a new `GlyphVector` object.
GlyphVector createGlyphVector(FontRenderContext frc, CharacterIterator ci)	Gets a new `GlyphVector` object with a specified character iterator.
`GlyphVector createGlyphVector(FontRenderContext frc, int[]` glyphCodes)	Gets a new `GlyphVector` object with a given integer array and the specified `FontRenderContext` class.
createGlyphVector(FontRenderContext frc, String str)	Gets a new `GlyphVector` object created with the specified `FontRenderContext class.`
static Font decode(String str)	Decodes a font with specified string str.
Font deriveFont(AffineTransform trans)	Creates a new `Font` object by duplicating the current `Font` object and applying a new transform, which maps one set of coordinates to another.
Font deriveFont(float size)	Creates a new `Font` object by duplicating the current `Font` object and applying a new size.

Table 8: Methods of the Font Class

Method	Description
Font deriveFont(int style)	Creates a new `Font` object by duplicating the current `Font` object and applying a new style.
Font deriveFont(int style, AffineTransform trans)	Creates a new `Font` object by duplicating the current `Font` object and applying a new style and transform.
Font deriveFont(int style, float size)	Creates a new `Font` object by duplicating the current `Font` object and applying a new style and size.
Font deriveFont(Map attributes)	Creates a new `Font` object by duplicating the current `Font` object and applying a new set of font attributes.
boolean equals(Object obj)	Compares the `Font` object to the given object.
protected void finalize()	Disposes of a native `Font` object.
Map getAttributes()	Gets a map of font attributes in a font.
AttributedCharacterIterator.Attribute[] getAvailableAttributes()	Gets all key attributes of the current Font object.
byte getBaselineFor(char c)	Gets the baseline to display a character specified with c parameter.
String getFamily()	Gets the family name of a font.
String getFamily(Locale l)	Gets the family name of a font, localized with the l object of the Locale class.
static Font getFont(Map attributes)	Gets a font specified with the attributes parameter.
static Font getFont(String nm)	Gets `Font` object from a system properties list.
static Font getFont(String nm, Font font)	Gets given font from the system properties list.
String getFontName()	Gets the font face name.
`String getFontName` (Locale l)	Gets the font face name, localized with the l object of the Locale class.
float getItalicAngle()	Gets the italic angle of a font.
`LineMetrics getLineMetrics(char[] chars, int beginIndex, int limit,` FontRenderContext frc)	Gets a `LineMetrics` object by using the given parameters specified with chars, beginIndex, limit, and frc.
`LineMetrics getLineMetrics` `(CharacterIterator ci, int` `beginIndex, int limit,` FontRenderContext frc)	Gets a LineMetrics object using given parameters specified with ci, beginIndex, limit and frc.
LineMetrics getLineMetrics(String str, FontRenderContext frc)	Gets a `LineMetrics` object created with a given string and the `FontRenderContext` class.

Table 8: Methods of the Font Class

Method	Description
LineMetrics getLineMetrics(String str, int beginIndex, int limit, FontRenderContext frc)	Gets a `LineMetrics` object created with a given arguments.
Rectangle2D getMaxCharBounds(FontRenderContext frc)	Gets the maximum bounds of characters in a given graphics context.
int getMissingGlyphCode()	Gets the glyph code used when this font does not have a glyph for a given Unicode character.
String getName()	Gets the font's logical name.
int getNumGlyphs()	Gets the number of glyphs in a font.
String getPSName()	Gets the `PostScript` name of a font.
int getSize()	Gets the point size of a font.
float getSize2D()	Gets the point size of a font in a float value.
Rectangle2D getStringBounds(char[] chars, int beginIndex, int limit, FontRenderContext frc)	Gets the bounds of the array of characters in the specified `FontRenderContext` class.
Rectangle2D getStringBounds(CharacterIterator ci, int beginIndex, int limit, FontRenderContext frc)	Gets the bounds of a characters in the given character iterator in the specified `FontRenderContext` class.
Rectangle2D getStringBounds(String str, FontRenderContext frc)	Gets the bounds of a given string in the specified `FontRenderContext` class.
Rectangle2D getStringBounds(String str, int beginIndex, int limit, FontRenderContext frc)	Gets the bounds of a given string in the specified range of string and the `FontRenderContext` class.
int getStyle()	Gets the style of a font.
AffineTransform getTransform()	Gets a copy of the transform associated with a font.
int hashCode()	Gets a `hashcode` for a font.
boolean hasUniformLineMetrics()	Checks whether a font has uniform line metrics.
boolean isBold()	Indicates whether the style of the `Font` object is bold.
boolean isItalic()	Indicates whether the style of the `Font` object is italic.
boolean isPlain()	Indicates whether the style of the `Font` object is plain.
String toString()	Converts the `Font` object into a string representation.

In this section, we learn about the following topics in context of the Font class:

- Determining the Available Fonts
- Creating and Selecting Fonts
- Using the FontMetrics Class

Let's discuss these in detail one by one.

Determining the Available Fonts

When we work with fonts, it is necessary to know which fonts are available on your system. For this, we use the `getAvailableFontFamilyNames()` method of the `GraphicsEnvironment` class. The syntax to use this method is:

```
String[ ] getAvailableFontFamilyNames();
```

Here, the `getAvailableFontFamilyNames()` method returns an array of strings, which contains the names of the available font in your system.

In addition to the `getAvailableFontFamilyNames()` method, the `getAllFonts()` method is also used to know which fonts are available in your system. The syntax to use the `getAllFonts()` method is:

```
Font[ ] getAllFonts();
```

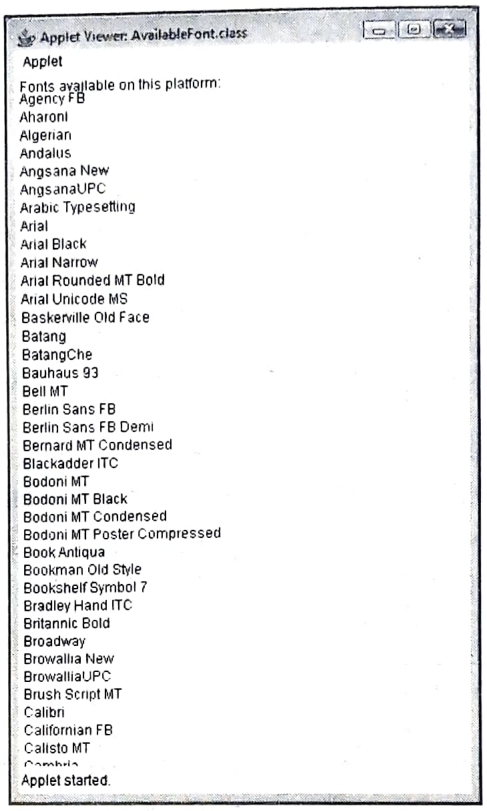
Here, this method returns an array of Font objects that is available in your system. Listing 9 shows the fonts available in your system (you can find the AvailableFont.java file on the CD in the code\chapter7 folder):

▶ **Listing 9: The AvailableFont.java File**

```
import java.applet.*;
import java.awt.*;
/*
<applet code="AvailableFont.class" width=550 height=600>
</applet>
*/
public class AvailableFont extends Applet
{
  public void paint(Graphics g)
  {
      int j=25;
      GraphicsEnvironment env =
      GraphicsEnvironment.getLocalGraphicsEnvironment();
      String[] fontNames = env.getAvailableFontFamilyNames();
      g.drawString("Fonts available on this platform:", 4, 15);
      for (int i = 0; i < fontNames.length; i++)
      {
          g.drawString(fontNames[i], 4, j);
          j+=15;
      }
  }
}
```

In Listing 9, the AvailableFont applet declares the object (env) of the `GraphicsEnvironment` class. This object invokes the `getAvailableFontFamily`

Names() method that stores all fonts in the fontNames string. Further, the drawString() method is called to display all the available fonts in a system. Figure 11 shows the output of Listing 9:



▲ **Figure 11: List of Available Fonts in the System**

■ Creating and Selecting Fonts

To select a new font, first we need to construct a Font object that describes a font. For this, we need to use constructor of the Font class as shown here:

```
Font(String fontName, int fontStyle, int pointSize)
```

Here, fontName is the name of the font, such as Sans Serif. fontStyle is the style of a font, such as Font.PLAIN, Font.BOLD, and Font.ITALIC and the size of a font is specified with pointSize.

We must select the setFont() method to use a font that we have created. This method can be used as:

```
void setFont(Font fontObject)
```

Here, fontObject is the object of the Font class. Let's look at an example of how to create and select a font. Listing 10 shows how to select and create a font (you can find the MyFont.java file on the CD in the code\chapter7 folder):

▶ **Listing 10: The MyFont.java File**

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

```
/*
<applet code="MyFont.class" width=300 height=300>
</applet>
*/
public class MyFont extends Applet
{
  public void paint(Graphics g)
  {
      Font sans = new Font("SansSerif", Font.BOLD, 30);
      g.setFont(sans);
      g.drawString("Sans Serif", 10, 75);
      Font arial = new Font("Arial", Font.BOLD, 20);
      g.setFont(arial);
      g.drawString("Arial", 10, 150);
      Font times = new Font("Times New Roman", Font.BOLD, 30);
      g.setFont(times);
      g.drawString("Times New Roman", 10, 200);

  }
}
```

In Listing 10, the `MyFont` applet declares the object (big) of the Font class. This Font class sets the font name to SansSerif with bold and the size of font is 30 point. Further, the drawString() method displays the message ("Sans Serif") in SansSerif font with bold and the size font is 30 point. Similarly, the Font class the sets the font name to Arial, and Times New Roman, and the drawstring() method displays messages "Arial" and "Times New Roman". Figure 12 shows the output of Listing 10:

▲ **Figure 12: Selecting and Creating a New Font.**

TEST YOUR KNOWLEDGE

Q10. Write an applet to set the font as Times New Roman and print some message.

Ans.

```
import java.applet.*;
import java.awt.*;
/*
<applet code="TestYourKnowledge10.class" width=300 height=300>
```

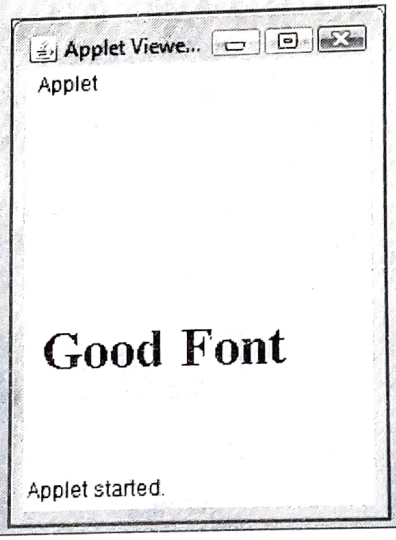
```
</applet>
*/
public class TestYourKnowledge10 extends Applet
{
  public void paint(Graphics g)
  {
      setForeground(Color.red);
      Font times = new Font("Times New Roman", Font.BOLD, 30);
      g.setFont(times);
      g.drawString("Good Font", 10, 150);
  }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge10.java
D:\code\chapter7>appletviewer TestYourKnowledge10.html
```

Output:

The following code snippet shows the output of the TestYourKnowledge10 applet:



Using the FontMetrics Class

The FontMetrics class is used to specify the physical dimensions of fonts. This class is used to determine the measurements of characters and strings. One of the most common uses of the `FontMetrics` class is to determine height and width of text when you are displaying the multiline text. Table 9 lists the methods of the `FontMetrics` class:

Table 9: Methods of the FontMetrics Class

Method	Description
`int bytesWidth(byte[ ] data, int off, int len)`	Gets the total advance width to show the given array of bytes in a font.
`int charsWidth(char[] data, int off, int len)`	Gets the total advance width to show the given array of characters in a font.
`int charWidth(char ch)`	Gets the advance width of the given character in a font.

Table 9: Methods of the FontMetrics Class

Method	Description
`int charWidth(int codePoint)`	Gets the advance width of the given unicode character in a font.
`int getAscent()`	Indicates the font ascent of a font.
`int getDescent()`	Indicates the font descent of a font.
`Font getFont()`	Gets the font described by the FontMetrics object.
`int getHeight()`	Gets the standard height of a line of text in a font.
`int getLeading()`	Indicates the leading of a font.
`LineMetrics getLineMetrics (char[]chars,int beginIndex, int limit, Graphics context)`	Gets the `LineMetrics` object for a given character array in the specified Graphics context.
`LineMetrics getLineMetrics (CharacterIterator ci, int beginIndex, int limit, Graphics context)`	Gets the `LineMetrics` object for a character iterator in the specified Graphics context.
`LineMetrics getLineMetrics (String str, Graphics context)`	Gets the `LineMetrics` object for a given string in the specified Graphics context.
`LineMetrics getLineMetrics (String str, int beginIndex, int limit, Graphics context)`	Gets the `LineMetrics` object for a given string in a given Graphics context.
`int getMaxAdvance()`	Gets the maximum advance width of any character in a font.
`int getMaxAscent()`	Indicates the maximum ascent of a font described by the `FontMetrics` object.
`Rectangle2D getMaxCharBounds (Graphics context)`	Gets the maximum bounds of characters in a given Graphics context.
`int getMaxDescent()`	Indicates the maximum descent of a font described by this `FontMetrics` object.
`Rectangle2D getStringBounds(Character Iterator ci, int beginIndex, int limit, Graphics context)`	Gets the bounds of a characters in the given character iterator in the specified Graphics context.

Table 9: Methods of the FontMetrics Class

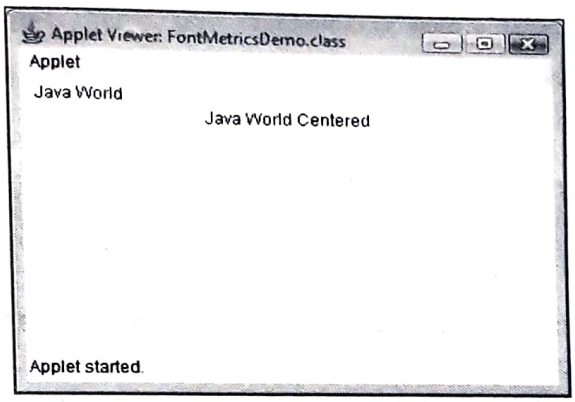
Method	Description
`Rectangle2D getStringBounds(String str, Graphics context)`	Gets the bounds of a given string in the specified Graphics context.
`Rectangle2D getStringBounds(String str, int beginIndex, int limit, Graphics context)`	Gets the bounds of a given string in the specified range of string in the specified Graphics context.
`int[] getWidths()`	Gets the advance width of the first 256 characters in a font.
`boolean hasUniformLineMetrics()`	Checks whether the font has uniform line metrics or not.
`int stringWidth(String str)`	Gets the total advance width of a string to show the given string in a font.
`String toString()`	Gets the `FontMetrics` object's values as a string.

Listing 11 shows the usage of the FontMetrics class (you can find the FontMetricsDemo.java file on the CD in the code\chapter7 folder):

▶ **Listing 11: The FontMetricsDemo.java File**

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

/* <APPLET
  CODE=FontMetricsDemo.class
  WIDTH=200
  HEIGHT=200 >
</APPLET>  */
public class FontMetricsDemo extends Applet
{
  public void paint(Graphics g)
  {
      FontMetrics fontMatrics = g.getFontMetrics();
      g.setColor(Color.BLACK);

      // first set y to the first line on screen
      int y = fontMatrics.getHeight();

      // draw a line of text
      g.drawString("Java World",10,y);

      // center a line of text
      String str = "Java World Centered";
      int x = (getWidth()/2)-(fontMatrics.stringWidth(str)/2);
      y+=fontMatrics.getHeight(); // move down one line of text
      g.drawString(str,x,y);
  }
}
```

In Listing 11, the `fontMatrics` object of the FontMetricsDemo class is obtained by using the `getFontMetrics()` method of the Graphics class. Further, this object is used to determine height and width of text using the `getHeight()`, and `getWidth()` method. The height of text is stored in y variable and width of text is stored in x variable. These variables (x and y) are used to display text messages in first line and centered line of a screen. Figure 13 shows the output of Listing 11:



▲ **Figure 13: Displaying Text in Centered Using FontMetrics Class**

After working with font, let's see how to handle events generated by mouse and keyboard.

7.8 Handling Events

An event is an action performed by a user. For example, when you press a keyboard button, the keyboard related event occurs within an application. Event handling is a process that allows the programmer to control the activities that occur with an event, such as a when the close button of a window is clicked, the window gets closed.

One of the biggest aspects of creating applets and applications is allowing the user to interact with the program, and we do that with the help of *event handlers*. When a user performs some actions, such as clicking a button, closing a window, selecting an item in a list, or using the mouse, Java considers these as events.

In the following sections, we discuss the following topics in the context of event handling:

- The Delegation Event Model
- Events Source
- Explaining Event Listeners
- Implementing the Delegation Event Model
- Using Adapter Classes
- Using Anonymous Inner Adapter Classes

Let's begin our discussion with the Delegation Event Model.

The Delegation Event Model

The delegation event model is the modern approach to handle events in which, we define standard and consistent mechanisms to generate events. In this approach, a source generates an event and sends it to one or more listeners. The listener waits for an event, and when it receives an event, then the listener process that event. The advantage of delegation event model is that the application logic is separated from the user interface logic.

In delegation event model, event types are encapsulated in a class hierarchy on the top of the `java.util.EventObject` class. To receive an event notification, listeners must register with a source that generates an event. This provides an important benefit to send notifications to only those listeners that want to receive them. The event handling using delegation event modal is more efficient to handle event than old Java 1.0 version. The delegation event model eliminates the overhead of components required to receive events that are not in process.

■ Events Source

An object that generates an event is known as event source. An event source object must register a listener, to receive the notifications about a specific type of event. The general form to register an event listener to an event source is:

```
public void addTypeListener(TypeListener eventListener)
```

Here, `eventListener` is a reference to an event listener and Type is the name of an event. For example, the `addKeyListener()` method is used to register a keyboard event listener.

Similarly, the `addMouseMotionListener()` method is used to register a mouse motion event listener. It is to be noted that an event source can register one or more listeners. When an event source registers a single event listener, known as unicast event; however if it registers multiple listeners, known as multicast events.

The following example shows the unicast events:

```
public void add<ListenerType>( <ListenerType> listener) throws
  java.util.TooManyListenersException;
public void remove<ListenerType>( <ListenerType> listener);
```

The following example shows the multicast events:

```
public void add<ListenerType>( <ListenerType> listener);
public void remove<ListenerType>( <ListenerType> listener);
```

Note that in both the previous examples, the `removeTypeListener()` method is used to remove or to unregister the event listener from the event source.

■ Explaining Event Listeners

A listener is an object that notifies when an event occurs. It has two major requirements to notify an event. First, one or more sources must be registered to receive notifications about specific types of event. Second, to receive and process event notifications, listener must implement specific methods. The methods of receiving and processing events are defined in a set of interfaces defined in the `java.awt.event` package. For example, this package (`java.awt.event`) provides the `MouseMotionListener` interface, which has two methods such as, mouseDragged(MouseEvent e), and mouseMoved(MouseEvent e), to receive notifications when a mouse is dragged or moved. Let's explore the event listener classes and interfaces in the next sections.

➧ Exploring Event Listener Classes

The event listener classes represent events between the user and the GUI applications, which are the core of Java's event handling mechanism. These event listener classes such as

ActionEvent, and InputEvent, capture the fundamental characteristics of all the events. `EventObject` is the super class of all event listener classes. The constructor of the `EventObject` class is the `EventObject(Object srcEvent)`, where `srcEvent` is the object that generates this event.

The `EventObject` contains two methods namely, `getSource()` and `toString()`. The `getSource()` method returns the source of the event and the `toString()` method returns string equivalent of the event.

The `AWTEvent` class is a subclass of the `EventObject`. It is defined in the java.awt package. This class has the `getID()` method that is used to determine the type of event.

The `java.awt.event` package defines several types of event listener classes that generate various types of user interface elements. Some of the most important event listener classes are described here:

- ActionEvent Class
- AdjustmentEvent Class
- ComponentEvent Class
- InputEvent Class
- KeyEvent Class
- MouseEvent Class
- ItemEvent Class
- WindowEvent Class

Let's now study each of the event classes in detail.

ActionEvent Class

This class is used to generate an action event when a user presses a button, double-clicks in a list of items, or selects a menu item in an application. The `ActionEvent` contains four integer constants that identify any modifiers associated with an action event. These integer constants are `ALT_MASK`, `CTRL_MASK`, `META_MASK`, and `SHIFT_MASK`. The ActionEvent class has the following three constructors:

- ActionEvent(Object srcEvent, int type, String cmd)
- ActionEvent(Object srcEvent, int type, String cmd, int modifiers)
- ActionEvent(Object srcEvent, int type, String cmd, long timestamp int modifiers)

In the second `constructor` just mentioned, the `srcEvent` is the parameter used to generate an event. The 'type' parameter defines the type of `event`, `cmd` is a command string that represents the type of the command and the 'modifiers' defines which modifier keys such as `ALT`, `CTRL`, and `SHIFT` are pressed to generate the event.

The third constructor is same as the second constructor except that it contains one more parameter that is timestamp, which represents the time when the event occurred.

This class contains two important methods, namely, `getActionCommand()` and `getModifiers()` method. The `getActionCommand()` method invokes an ActionEvent object by using the command name that returns a string value. The `getModifiers()` method returns an integer value that represents which modifier key has been pressed during the generation of an event.

AdjustmentEvent Class

The `AdjustmentEvent` class is used to generate an event when some adjustments such as scrolling a scroll bar, dragging a slider are made on a component. Table 10 lists the adjustment constants of the `AdjustmentEvent` class:

Table 10: Adjustment Constants of the AdjustmentEvent Class

Constant	Description
`BLOCK_DECREMENT`	Decreases the inside value of the scroll bar when a user clicks the scroll bar.
`BLOCK_INCREMENT`	Increases the value of the scroll bar when a user clicks inside the scroll bar.
`TRACK`	Helps in dragging a slider.
`UNIT_DECREMENT`	Decreases the value of the scroll bar when the button at the end of the scroll bar is clicked.
`UNIT_INCREMENT`	Increases the value of the scroll bar when the button at the end of the scroll bar is clicked.

The constructor of the `AdjustmentEvent` class is `AdjustmentEvent(Adjustable srcEvent, int id, int type int data);` where, `srcEvent` is the reference of the object that generates an event, 'id' is equal to `ADJUSTMENT_VALUE_CHANGED` constant that indicates the changes that occurred in the scroll bar, 'type' is used for the type of event and data that is used for associated data of an event.

The important methods of `AdjustmentEvent` class are `getAdjustable()` and `getValue()` methods. The `getAdjustable()` method returns one constant, as discussed earlier, of the generated event. The `getValue()` method returns the changed values of a scroll bar.

ComponentEvent Class

ComponentEvent class is used to generate an event when a component is hidden, shown, moved or resized. Table 11 lists the constants of the ComponentEvent class:

Table 11: Constants of the ComponentEvent Class

Constant	Description
`COMPONENT_HIDDEN`	Hides the component.
`COMPONENT_MOVED`	Moves the component.
`COMPONENT_RESIZED`	Resizes the component.
`COMPONENT_SHOWN`	Makes the component visible.

The constructor of `ComponentEvent` is `ComponentEvent(Component srcEvent, int type)`, where `srcEvent` is a reference to an object that generates the event and type specifies the type of event used. The `getComponent()` method is an important method of `ComponentEvent` class that returns the component object.

InputEvent Class

The `InputEvent` class is an abstract class, which is the subclass of the `ComponentEvent` class that is used to generate an event when a component is hidden, shown, moved, or

resized. The subclasses of `InputEvent` are `KeyEvent` and `MouseEvent`. The constant variable such as `ALT_MASK`, `BUTTON2_MASK`, `META_MASK`, `ALT_GRAPH_MASK`, `BUTTON3_MASK`, `SHIFT_MASK`, `BUTTON1_MASK`, and `CTRL_MASK` is used to obtain information about any modifiers associated with the event. The following are the most commonly used methods of the InputEvent class:

- **getModifiers()**: Gets the modifier mask for an event
- **isAltDown()**: Gets whether the Alt modifier is down on an event or not
- **isConsumed()**: Gets an event has been consumed or not
- **isShiftDown()**: Gets whether the Shift modifier is down on an event or not
- **isControlDown()**: Gets whether the Control modifier is down on an event or not

KeyEvent Class

The `KeyEvent` class is used when a `KeyEvent` occurs during an input from the keyboard. There are three types of events identified by constants, namely, `KEY_PRESSED`, `KEY_RELEASED`, and `KEY_TYPED`. The first two events occur, when a key is pressed or released and the last event occurs when a character is typed.

Several integer constants are associated with KeyEvent. These constants are from `VK_0` to `VK_9`, used for numbers between 0 and 9, and `VK_A` to `VK_Z`, used for `ASCII` characters between `A and Z`. There are also some special types of constants such as `VK_ENTER`, `VK_ESCAPE`, `VK_SHIFT`, `VK_CONTROL` that are used for handling special characters.

The `KeyEvent` class provides several methods. Some of the important methods are, `getKeyChar()` which returns the character entered, and `getKeyCode()`, which returns the key code such as `VK_UNDEFINED` that is used for invalid characters.

MouseEvent Class

The `MouseEvent` class provides the integer constants that are used to generate a mouse event. Table 12 lists the constants of the MouseEvent class:

Table 12: Constants of the MouseEvent class

Constant	Description
`MOUSE_CLICKED`	The mouse is clicked.
`MOUSE_DRAGGED`	The mouse is dragged.
`MOUSE_ENTERED`	Mouse pointer is entered in a component area.
`MOUSE_EXITED`	Mouse pointer is exited from a component area.
`MOUSE_MOVED`	The mouse is moved.
`MOUSE_PRESSED`	The mouse button is pressed for a particular span of time.
`MOUSE_RELEASED`	The mouse button is released after being pressed for a specific time period.

The MouseEvent class provides `getX()` and `getY()` methods which return X and Y coordinates of the mouse pointer position, respectively. To get both X and Y co-ordinate, both `getX()` and `getY()` methods have to be used. An alternate method is the `getPoint()` method which can also be used to obtain the coordinates of the mouse pointer position. The `getClickCount()` method is used to obtain the number of mouse clicks for an event.

ItemEvent Class

The `ItemEvent` class generates an event when a check box or a list item is clicked or when a menu item is selected or deselected. The `SELECTED` and `DESELECTED` constants are used for selecting and deselecting an item. The following are the most commonly used methods of the ItemEvent class:

- `getItem()`: Returns an object, which is a reference of an item that generates an event.
- `getItemSelectable()`: Retrieves the reference of ItemSelectable object that is generator of an event.
- `getStateChange()`: Returns the changed states, such as `SELECTED` or `DESELECTED` of an event only.

WindowEvent Class

The `WindowEvent` class provides handling for window related events, that is, the event is generated if an operation is performed on a window. Table 13 describes a list of constants of the WindowEvent class:

Table 13: Constants of the WindowEvent Class

Constant	Description
`WINDOW_ACTIVATED`	A window is activated.
`WINDOW_CLOSED`	A window is closed.
`WINDOW_CLOSING`	Used when a user requests the window to be closed.
`WINDOW_DEACTIVATED`	Awindow is deactivated.
`WINDOW_ICONIFIED`	Iconifies a window, means, reducing the window to an icon on the desktop.
`WINDOW_DEICONIFIED`	Deiconifies a window that is, restoring the window to its normal state after iconifying it.
`WINDOW_OPENED`	A window is opened.

The most commonly used method of `WindowEvent` class is the `getWindow()` method, which returns object of a window that generates the event.

Exploring Event Listener Interfaces

When events occur, we need to detect and respond them so that we can handle those events. An event can be detected by creating an event listener (event handler). An event listener is simply an execution of a specific kind of event that guides the application about what to do. An event listener can be created from two operations, first is to create a method to handle a specific event, and second is to register the listener with the application to listen that specific type of event. The following code snippet shows how to handle mouse click event:

```
// Handling mouse clicked.
public void mouseClicked(MouseEvent mouseEv) {
// handle mouse click event
System.out.println("mouse button clicked");
}
// Registering the listener with application
addMouseListener(this);
```

In the preceding code snippet, the first operation is to listen a mouse click that is in the body of `mouseClicked()` method and add actions as the response. The parameter `mouseEv` in the `mouseClicked()` method refers to an event. The next operation is to register a listener with application.

The `java.awt.event` package provides several listener interfaces for event handling. When an event occurs, the listener invokes the appropriate method to handle an event. The following are the types of event listener interfaces that are generally used for handling events:

- ActionListener interface
- ComponentListener interface
- KeyListener interface
- MouseListener interface
- AdjustmentListener interface

Let's discuss all of them in detail one by one.

ActionListener Interface

This interface is used for receiving action events. A Java class implements this interface for processing an action event. The component's `addActionListener()` method is used to register the object of the ActionListener interface with a component. This interface provides the `actionPerformed()` method that is invoked when an action event occurs. The general form of `actionPerformed()method` is void `actionPerformed(`ActionEvent actionEvent). The user must perform the following steps to handle the event by using the ActionListener interface:

- Define an event handler class that implements the ActionListener interface, as shown in the following code snippet:

```
public class MyEventClass implements ActionListener { ….}
```

Here, MyEventClass is an event handler class that implements the ActionListener interface.

- Add the addActionListener() method with an instance of the event handler class containing one or more components in the defined event handler class, as shown in the following code snippet:

```
public class MyEventClass implements ActionListener {
someComponent.addActionListener(instanceOfMyClass);
}
```

Here, the addActionListener() method is added in the MyEventClass class.

- Finally, add the following code snippet that implements the actionPerformed() method in the defined event handler class:

```
public class MyEventClass implements ActionListener {
  someComponent.addActionListener(instanceOfMyClass);
  public void actionPerformed(ActionEvent e) {

  //code that perform the action...
}
}
```

Here, the `actionPerformed()` method is added in the MyEventClass class.

ComponentListener Interface

The `ComponentListener` interface is used for receiving a component event. The component's `addComponentListener()` method is used to register an object of the `ComponentListener` interface with a component. Table 14 lists the methods of the `ComponentListener` interface:

Table 14: Methods of the ComponentListener Interface

Method	Description
`componentHidden (ComponentEvent ce)`	Makes the component invisible.
`componentMoved (ComponentEvent ce)`	Moves a component from one position to another.
`componentResized (ComponentEvent ce)`	Resizes/changes the size of the component according to the user's requirement.
`componentShown (ComponentEvent ce):`	Makes the component visible.

KeyListener Interface

The `KeyListener` interface generates a key event when the user types from a keyboard. When a user presses or releases keyboard keys, the key events are fired. Table 15 lists the methods of the `KeyListener` interface:

Table 15: Methods of KeyListener Interface

Method	Description
`void keyPressed(KeyEvent keyEvent)`	Invoked when a key is pressed on the keyboard.
`void keyReleased(KeyEvent keyEvent)`	Invoked when a key is released on the keyboard.
`void keyTyped(KeyEvent keyEvent`	Invoked when a character is entered.

MouseListener Interface

The `MouseListener` interface is used when the user uses a mouse to interact with a component. The mouse event occurs when the user presses or releases the mouse button. Table 16 lists the methods of the `MouseListener` interface:

Table 16: Methods of the MouseListener Interface

Method	Description
`void mouseClicked(MouseEvent mouseE)`	Invoked when a user clicks (presses or releases) the mouse button on a component. Note that a click event involves pressing and releasing the mouse button in quick succession. This method is invoked only if the user clicks the mouse button.

Table 16: Methods of the MouseListener Interface

Method	Description
`void mouseEntered(MouseEvent mouseE)`	Invoked when a user moves the mouse cursor/ mouse pointer on the component.
`void mouseExited(MouseEvent mouseE)`	Invoked when the mouse cursor exits from the component.
`void mousePressed(MouseEvent mouseE)`	Invoked when the mouse button is pressed. The method is invoked only if the mouse button is pressed, not released.
`void mouseReleased(MouseEvent mouseE)`	Invoked when the mouse button is released. The method is invoked only if the mouse button is released after being pressed for a short duration.

TEST YOUR KNOWLEDGE

Q11. Write a program to handle mouse event.

Ans.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet CODE= TestYourKnowledge11.class WIDTH=300 HEIGHT=200>
</applet>

*/
public class TestYourKnowledge11 extends Applet implements
 MouseListener
{
  public String message = "x = ?, y = ?";

  public void init()
  {
      // tell the applet to listen for mouse related events
      addMouseListener(this);
      repaint();
  }
  public void paint(Graphics g)
  {
      g.drawString(message, 50, 50);
  }

  // Respond to a mouse press on applet
  public void mousePressed(MouseEvent mEvent)
  {
      // display the x and y coordinate of mouse pointer
      message = "x = " + mEvent.getX() + ", y = " + mEvent.getY();
      System.out.println("mouse button pressed");
      repaint();
  }
  // you must implement all the methods of MouseListener interface.
  public void mouseClicked(MouseEvent mEvent)
```

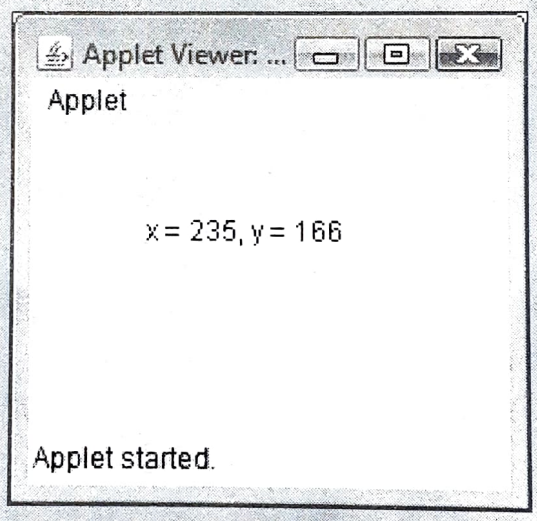
```
    {
        System.out.println("mouse button clicked");
    }
    public void mouseEntered(MouseEvent mEvent)
    {
        System.out.println("mouse enter in applet window");
    }
    public void mouseExited(MouseEvent mEvent)
    {
        System.out.println("mouse exit in applet window");
    }
    public void mouseReleased(MouseEvent mEvent)
    {
        System.out.println("mouse released in applet window");
    }

}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge11.java
D:\code\chapter7>appletviewer TestYourKnowledge11.html
```

Output:

The following code snippet shows the output of the TestYourKnowledge11 applet:



Q12. Write a program to draw a free sketch using mouse over an applet window using the MouseListener interface.

Ans.

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

/*
<applet code=" TestYourKnowledge12.class"
 width=300
 height=300>
</applet>
*/
```

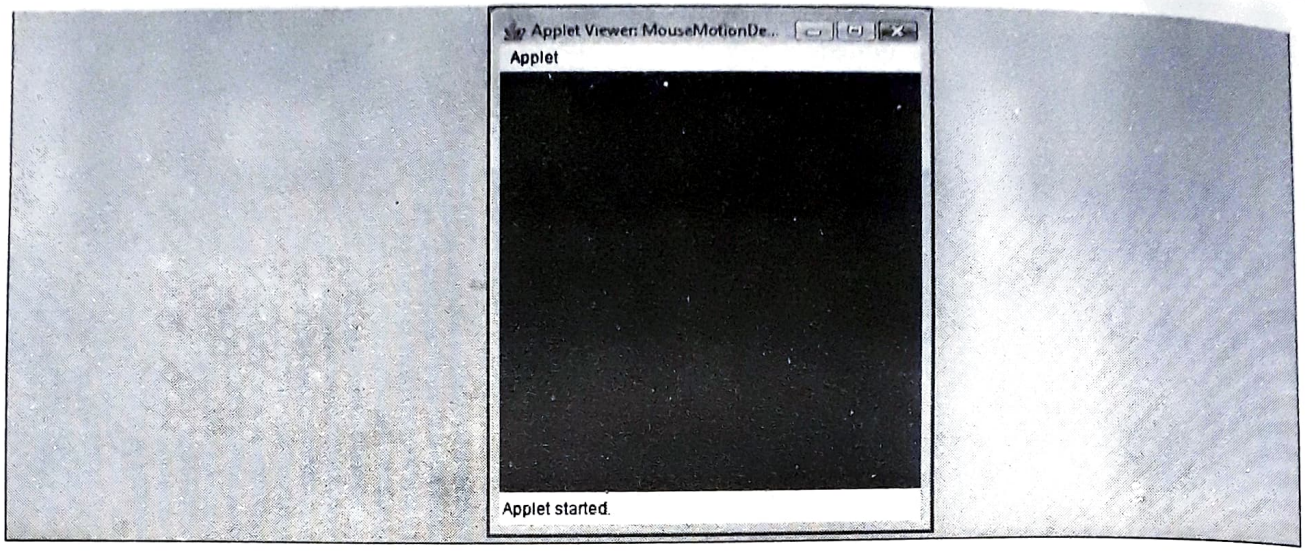
```
public class TestYourKnowledge12 extends Applet implements
 MouseListener, MouseMotionListener
{
 int width, height;
 Vector listPositions;
 public void init()
 {
     width = getSize().width;
     height = getSize().height;
     setBackground( Color.BLACK );
     listPositions = new Vector();
     addMouseListener( this );
     addMouseMotionListener( this );
 }
 public void mouseEntered( MouseEvent me ) { }
 public void mouseExited( MouseEvent me ) { }
 public void mouseClicked( MouseEvent me ) { }
 public void mousePressed( MouseEvent me ) { }
 public void mouseReleased( MouseEvent me ) { }
 public void mouseMoved( MouseEvent me )
 {
     if ( listPositions.size() >= 50 )
     {
         // delete the first element in the listPositions list
         listPositions.removeElementAt( 0 );
     }
     // add the new position to the end of the listPositions list
     listPositions.addElement( new Point(me.getX(), me.getY()) );
     repaint();
     me.consume();
 }
 public void mouseDragged( MouseEvent me ) { }
 public void paint( Graphics g )
 {
     g.setColor( Color.RED );
     for ( int i = 1; i < listPositions.size(); ++i )
     {
         Point P1 = (Point)(listPositions.elementAt(i-1));
         Point P2 = (Point)(listPositions.elementAt(i));
         g.drawLine( P1.x, P1.y, P2.x, P2.y );
     }
 }
}
```

Execution:

```
D:\code\chapter7>javac TestYourKnowledge12.java
D:\code\chapter7>appletviewer TestYourKnowledge12.html
```

Output:

The following code snippet shows the output of the TestYourKnowledge12 applet:



AdjustmentListener Interface

The `AdjustmentListener` interface receives the adjustment events that are generated in the AdjustmentEvent class. This interface provides the `adjustmentValueChanged()` method that is invoked when an adjustment event occurs. The `adjustmentValueChanged()` method takes one argument, that is, the object of `AdjustmentEvent`. The general form of the `adjustmentValueChanged()` method is `void adjustmentValueChanged (AdjustmentEvent adjustmentEvent)`.

■ Implementing the Delegation Event Model

Now, we have learned the theory of delegation event model and its various event listener classes and interfaces. It is the time to implement delegation event model in an applet. For implementing delegation event model, first we need an appropriate listener class to receive an event and then, implement the code to register and unregister component for that event.

Let's see how to handle an event using mouse and keyboard.

➡ Handling Mouse Events

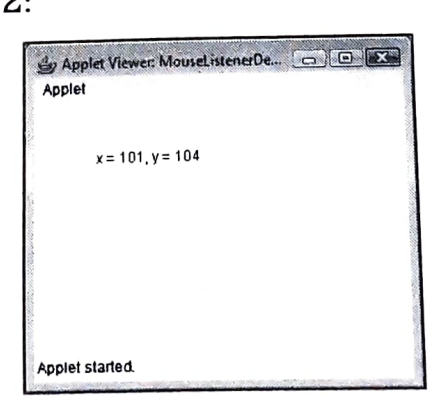
For handling a mouse event, the MouseListener interface must be implemented in an applet. We must implement all the methods of the `MouseListener` interface, such as `mouseEntered()`, `mouseExited()`, `mouseReleased()`, and `mouseClicked()` in an applet. Listing 12 shows how mouse event is handled by using mouse (you can find the `MouseListenerDemo.java` file on the CD in the code\chapter7 folder):

▶ Listing 12: Event Handling Using MouseListener Interface

```
// Example of using the MouseListener interface
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet CODE=MouseListenerDemo.class WIDTH=300 HEIGHT=200>
</applet>
*/
public class MouseListenerDemo extends Applet implements MouseListener
{
 public String myMessage = "x = ?, y = ?";
```

```
    public void init()
    {
        // tell the applet to listen for mouse related events
        addMouseListener(this);
        repaint();
    }
    public void paint(Graphics g)
    {
        System.out.println("paint called");
        g.drawString(myMessage, 50, 50);
    }
    // Respond to a mouse press on applet
    public void mousePressed(MouseEvent me)
    {
        // display the x and y coordinate of mouse pointer
        myMessage = "x = " + me.getX() + ", y = " + me.getY();
        System.out.println("mouse button pressed");
        repaint();
    }
    // you must implement all the methods of MouseListener interface.
    public void mouseClicked(MouseEvent e)
    {
        System.out.println("mouse button clicked");
    }
    public void mouseEntered(MouseEvent e)
    {
        System.out.println("mouse enter in application area");
    }
    public void mouseExited(MouseEvent e)
    {
        System.out.println("mouse exit in application area");
    }
    public void mouseReleased(MouseEvent e)
    {
        System.out.println("mouse button released");
    }
}
```

In Listing 12, when user enters the mouse pointer, the `mouseEntered()` method is called and displays "mouse enter in application area" on a console window. When user clicks a mouse button within the applet window, the value of x and y co-ordinate of mouse pointer is displayed on this window. The MouseListenerDemo applet also executes the `mouseClicked()` and `mouseReleased()` method of the MouseListener interface. Figure 14 shows the output of Listing 12:



▲ **Figure 14: An Output of Mouse Event**

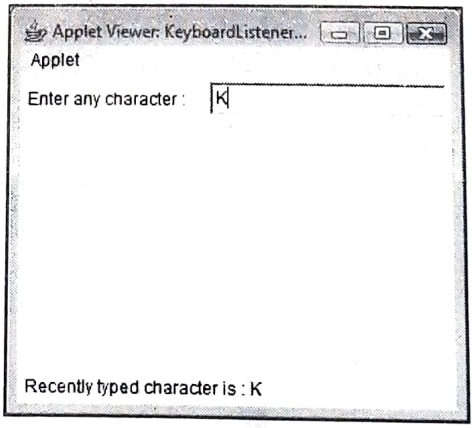
Handling Keyboard Events

In this section, we will describe how to handle keyboard related events. For handling keyboard events, the `KeyListener` interface must be implemented. When the keyboard event is executed, several key events can occur such as key pressed, key released and key typed. Listing 13 shows how keyboard event is handled (you can find the KeyboardListenerDemo.java file on the CD in the code\chapter7 folder):

▶ **Listing 13: Event Handling Using the KeyListener Interface**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
/*
<applet CODE=KeyboardListenerDemo.class WIDTH=300 HEIGHT=200>
</applet>
*/
public class KeyboardListenerDemo extends Applet implements KeyListener
{
  public void init() {
      Label label = new Label ("Enter any character :");
      add(label);
      TextField textField = new TextField(20);
      add(textField);
      textField.addKeyListener(this);
  }
  public void keyPressed(KeyEvent e)
  {}
  public void keyReleased(KeyEvent e)
  {}
  public void keyTyped(KeyEvent e) {
      showStatus(" Recently typed character is : " + e.getKeyChar());
  }
}
```

In Listing 13, when the user presses any character, the corresponding character is displayed on the screen. This is done by invoking the `keyTyped()` method of the `KeyListener` interface. Figure 15 shows when you press character 'K', it is displayed on the screen:



▲ **Figure 15: An Output of Keyboard Event**

In Figure 15, you can see that when a character 'K' is entered in the provided text field, the keyTyped() method is called, and a message 'Recently typed character is: K' is displayed at the bottom of the applet.

Using Adapter Classes

Adapter classes are used when we want to receive and process only some events that need to be handled by a particular event listener interface. Without the use of Adapter classes, all the methods of an interface have to be implemented in your program, whether these methods are required or not. With Adapter classes, only the required methods need to be implemented. Therefore, in this way, Adapter classes help you to work with only the required method. Java provides various classes corresponding to various listener interfaces, for example, `MouseAdapter` class in correspondence to the `MouseListenerInterface` interface.

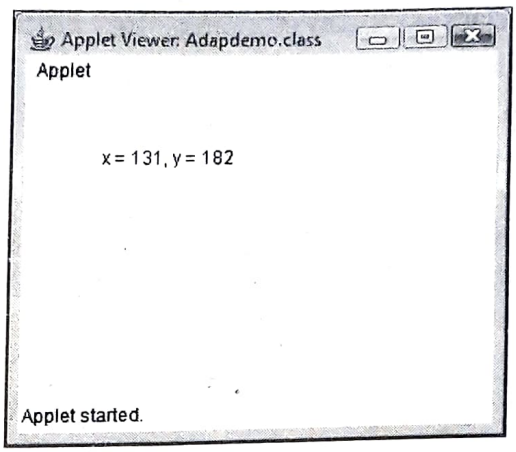
For example, when a Java program implements the `MouseListener` interface, all the methods of the `MouseListenter` interface must be implemented even when only one method is required.

With the implementation of the Adapter classes, only the required methods need to be implemented. Listing 14 shows the usage of an adapter class (you can find the Adapdemo.java file on the CD in the code\chapter7 folder):

▶ **Listing 14: Using MouseAdapter Class**

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet CODE=Adapdemo.class WIDTH=300 HEIGHT=200>
</applet>
*/
public class Adapdemo extends Applet
{
  public String myMessage = "x = ?, y = ?";
  public void init()
  {
      // tell the applet to listen for mouse related events
      addMouseListener(new MyAdapter (this));
  }
  public void paint(Graphics g)
  {
      System.out.println("paint called");
      g.drawString(myMessage, 50, 50);
  }
}
class MyAdapter extends MouseAdapter
{
  Adapdemo ad;
  MyAdapter(Adapdemo ad)
  {
      this.ad=ad;
  }
  // Respond to a mouse press on applet
  public void mousePressed(MouseEvent me)
  {
      // display the x and y coordinate of mouse pointer
      ad.myMessage = "x = " + me.getX() + ", y = " + me.getY();
      ad.repaint();
  }
}
```

The working of Listing 14 is just same as Listing 12. This application has been rewritten so that the use of `MouseAdapter` class can be understood. In Listing 14, we have created two classes; first, the `Adapdemo` class is defined to create an applet and to call the specific mouse related event. The second class is the adapter, which extends the `MouseAdapter` class and shows the implementation of the `mousePressed()` method. Figure 16 shows the output of Listing 14:



▲ **Figure 16: An Output of Mouse Event Using the MouseAdapter class**

In Figure 16, you can see the position of mouse pointer, displaying in the form of x and y co-ordinates. This position is shown, whenever the left mouse button is clicked in the applet window.

■ Using Anonymous Inner Adapter Classes

An inner class that is not assigned a name is known as anonymous inner class. This class has the following special format:

```
new SuperType(constructor parameters) {//methods and data }
```

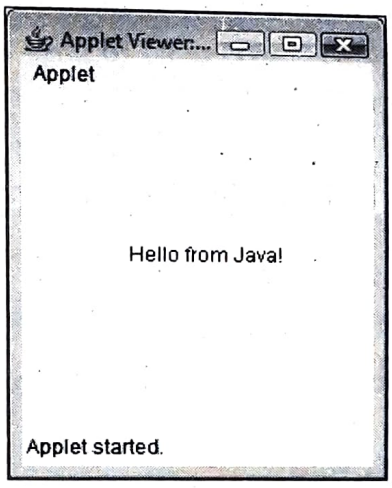
Here, `SuperType` is the class or interface where anonymous inner class is derived from. In this section, we explain how an anonymous inner class is used for writing event handling program. We are going to derive an anonymous inner class from the `MouseAdapter` class in an applet. We are also overriding the `mousePressed()` method in an anonymous inner class, which makes the code much more compact. Listing 15 shows how we use anonymous inner adapter class (MouseAdapter) for handling mouse event (you can find the AppletAnonymous.java file on the CD in the code\chapter7 folder):

▶ Listing 15: Using Anonymous Inner Adapter Class

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
/* <APPLET
 CODE=AppletAnonymous.class
 WIDTH=200
 HEIGHT=200 >
 </APPLET> */
 public class AppletAnonymous extends Applet
 {
 public String s = "Hello from Java!";
```

```
    public void init()
    {
        addMouseListener(new MouseAdapter()
        {
        public void mousePressed(MouseEvent me)
        {
        s = "Hello to Java!"; repaint(); }});
    }
    public void paint(Graphics g) { g.drawString(s, 60, 100); }
}
```

In Listing 15, the `init()` method calls the addMouseListener() method that defines and instantiates an anonymous inner class (MouseAdapter). This anonymous inner class is defined within the scope of the AppletAnonymous class. Therefore, it can call the `repaint()` method directly. Figure 17 shows the output of Listing 15:



▲ **Figure 17: Anonymous Inner Class for Handling Mouse Event**

With this, we have reached at the end of this chapter. Before closing this chapter, let's have a look at the topics covered in this chapter.

Summary

In this chapter, you learned about applets and its life cycle. Further, this chapter discussed how to draw lines, ovals, arcs, and polygons in applets. In addition, you learned how to set or fill colors in various shapes (rectangle, oval) in applets, and how to select and create font for writing some text. At the end of the chapter, we described event handling in the context of mouse and keyboard, with and without using adapter classes.

Review Questions

Here, let's answer the following types of questions:

- ❒ True or False
- ❒ Multiple Choice
- ❒ Short Answers
- ❒ Debugging

True False

State True or False for the following:

1. The Applet class has the paint() method. **False**
2. The getParameter() method returns the value of the named parameter in the HTML tag. **True**
3. The getStyle() method returns the style of a font. **True**
4. An applet can make network connections to any host on the Internet. **False**
5. The MouseClickListener interface defines the methods for handling mouse clicks. **False**
6. The MouseListener interface defines the methods for handling mouse clicks. **True**
7. The getAvailableFontFamilyNames() method is used to determine available fonts in a system. **True**
8. The play() method is used to play an audio clip. **True**
9. In HTML, the <PARAM> tag is used to pass a parameter to an applet. **True**
10. The setFont() method is used to create a font. **False**
11. The init() method is in the Applet class. **True**
12. The mouseExited() method is invoked when mouse exits a component. **True**

Multiple Choice Questions

Answer the following multiple choice questions.

Q1. Which of the following are the mandatory tags when creating HTML page to display an applet?

A. name, height, width
B. code, name
C. code, height, width
D. codebase, height, width

Ans. Option C is correct.

Q2. Which of the following is the highest-level event class of the event-delegation model?

A. java.util.EventListener
B. java.util.EventObject
C. java.awt.AWTEvent
D. java.awt.event.AWTEvent

Ans. Option B is correct.

Q3. Which class comes at the top of the AWT event hierarchy?

A. HierarchyEvent
B. ActionEvent
C. ComponentEvent
D. AWTEvent

Ans. Option D is correct.

Q4. Which listener object is invoked first to handle the event, when two or more objects are added as listeners for the same event?

A. Last object added as listener
B. First object added as listener
C. Cannot provide more than one listener for a given event
D. Cannot determine which listener is invoked first

Ans. Option D is correct.

Q5. An applet executes these methods in which sequence?

A. init(), start(), paint(), stop(), and destroy() methods
B. init(), start(), stop(), and destroy() methods
C. init(), start(), paint(), and stop() methods
D. init(), start(), paint(), and destroy() methods

Ans. Option A is correct.

Q6. Which of the following methods are available in the MouseMotionListener interface?

A. void mouseDragged(MouseEvent e)
B. void mouseMoved(MouseEvent e)
C. void mouseExited(MouseEvent e)
D. void mouseReleased(MouseEvent e)

Ans. Options A and B are correct.

Q7. Which of the following fields are available in the Font class?

A. ROUND
B. PLAIN
C. ITALIC
D. BOLD

Ans. Options B, C, and D are correct.

Q8. Which class is used to extend and create an applet?

A. The Applet class
B. The Font class
C. The Component class
D. The Container class

Ans. Option A is correct.

Q9. Which of the following attribute is not in the <APPLET> tag?

A. HEIGHT
B. TOP
C. PARAM NAME
D. CODE

Ans. Option B is correct.

Q10. Which of the following fields are not in the Color class?

A. BLACK
B. RED
C. LIGHTRED
D. LIGHTBLACK

Ans. Options C, and D are correct.

Q11. Which of the following packages are used to create an applet?

A. The java.util package
B. The java.io package
C. The java.math package
D. The java.applet package

Ans. Option D is correct.

Q12. Which of the following method is used to create a line?

A. drawLine()
B. drawRect()
C. drawOval()
D. drawArc()

Ans. Option A is correct.

Short Answer Questions

Q1. What is an Applet? Should applets have any constructor?

Ans. An applet is a program written in the Java programming language that can be included in an HTML Web page, in the same way an image is included in a Web page. It can be used in both static and dynamic Web pages to either display content or share information through the Web application.

In applets, there is no concept of constructor. It can be invoked through either Web browser or appletviewer utility provided by JDK.

Q2. What are the Applet's Life Cycle methods? Explain them.

Ans. A Web browser manages the life cycle of an applet, by calling certain methods. The Applet or JApplet class executes these methods automatically in a proper sequence known as the life cycle methods of an applet. The following are the methods in the life cycle of an Applet:

- `init()` **method:** This is the first method to be called. The public void `init()` method is executed only once, when an applet is loaded for the first time. This method is used to set up menus, variable values, and create threads.
- `start()` **method:** This method is executed after the execution of the `init()` method. The public void `start()` method is called to start the applet activity. This is the heart of the applet. This method executes every time an HTML page containing applets is displayed on the screen. For example, when a user minimizes a Web page to move to another page, the execution of the public void `start()` method is stopped, whereas maximizing the Web page resumes the execution of this method.
- `stop()` **method:** This method stops an applet from executing.
- `destroy()` **method:** This method removes an applet from the memory of the computer.

It is not necessary to use these methods while creating an applet, because the applet container manages these methods.

Q3. In which sequence, an applet calls methods involved in applet's life cycle?

Ans. When an applet starts, the AWT calls the following methods in a sequence:

1. init()
2. start()

When an applet is terminated, the following sequence of method is called:

1. stop()
2. destroy()

Q4. What are the differences between an applet and applications?

Ans. An applets and an application has many difference. An application is used to design stand alone application where as an applet is used to design Web application. Some of the common differences between applications and applets are:

- An applet has an HTML document but there is no HTML document in an application.

- An application has a special static method called `main()` method which serves as the execution entry point of an application. An applet does not have a `main()` method.
- When an application starts to run, it does not need application object, whereas in case of applet, it cannot run until an applet object has been created.
- When an applet starts to run, it inherits or overrides life cycle methods. These life cycle methods are called by the Web browser at various times during the life cycle of an applet. In contrast, an application has no life cycle method.
- An applet can be used to create static and dynamic Web pages where as an application can be used to created full blown application, such as network accessing.

An applet is subjected to more security restriction in terms of file and network access, whereas an application can have free security restriction of files and network access.

Q5. How can we pass parameters to an applet from HTML page?

Ans. We can pass parameters to an applet using <PARAM> tag as:

```
<PARAM name="param1" value="value1">
<PARAM name="param2" value="value2">
```


Values of these parameters (param1, and param2) can be accessed by calling the `getParameter()` method in an applet. This `getParameter()` method returns string value corresponding to the parameter name. The following code snippet shows how we call parameters (param1, and param2) in an applet:

```
String param1, param2;
public void start() {
 param1= getParameter("param1");
 param2= getParameter("param2");
}
```

Q6. How can an applet communicate with other applet on Web page?

Ans. When we need an applet to communicate with other applet, we must follow these two steps:

1. Provide the name of applet using the HTML <APPLET> tag.
2. Invoke the AppletContext's getApplet() method in your applet code to obtain references to other applets on a Web page.

Q7. How do we determine the width and height of an applet?

Ans. We can use the `getSize()` method of Applet class that inherits from the Component class. The `getSize()` method returns the size of an applet in an object of the Dimension class using this object, we can get width, and height fields of an applet. The following code snippet shows how to determine the width and height of an applet:

```
Dimension dim = getSize();
int appletWidth = dim.width();
int appletHeight = dim.height();
```

Q8. Which interfaces are in java.applet package?

Ans. The java.applet package has the following three interfaces:

- AppletContext
- AppletStub
- AudioClip

Q9. What are the steps involved in developing an Applet?

Ans. The following steps are involved in developing an Applet:

1. Import the Applet class and `awt` package, and put name of your applet by extending the Applet class. This can be done in the following code snippet:

```
import java.applet.Applet;
import java.awt.*;
public class HelloApplet extends Applet { . . . }
```

2. Use the `paint()` method to display message in the applet. This `paint()` method passes an object of the `Graphics` class, which we use to draw a string of text (Hello from Java) in that applet. The following code snippet shows how to use the `paint()` method to draw a string of text in the HelloApplet applet:

```
import java.applet.Applet;
import java.awt.*;
public class HelloApplet extends Applet {
 public void paint(Graphics g) {
     g.drawString("Hello from Java!", 60, 100);
     }
}
```

3. Save the preceding code snippet as HelloApplet.java and compile it using `javac` command at command prompt. After compiling this file, the HelloApplet.class file is created on your specified directory.
4. Now, create an HTML file named HelloApplet.html and write the following codes:

```
<APPLET
 CODE=HelloApplet.class
 WIDTH=300
 HEIGHT=200 >
</APPLET>
```

5. Type the following command at command prompt to execute the HelloApplet applet using `appletviewer`:

```
D:\code\chapter7>appletviewer HelloApplet.html
```

Q10. Which method is called when an applet is running and which method is called when an applet is removed from the memory?

Ans. When an applet starts running, the init() method is called and when an applet's execution is finished, the destroy() method is called.

Q11. What are Adapter Classes in Event handling?

Ans. Adapter classes are used when we want to receive and process only some events that need to be handled by a particular event listener interface. Without the use of Adapter classes, all the methods of an interface have to be implemented in a program whether these methods are required or not. With Adapter classes, only the required method needs to be implemented. Therefore, in this way, Adapter classes help you working with only the required method. Java provides various classes corresponding to various listener interfaces, for example, the `MouseAdapter` class in correspondence to the `MouseListenerInterface`.

Q12. How to pass parameter to an applet?

Ans. The <PARAM> attribute of the <APPLET> tag is used to pass a parameter in an applet. We use the getParameter() method of the Applet class to get the value of a parameter specified in the <PARAM> attribute. The following program shows how to pass parameter and get the value of a parameter in an applet:

```
import java.applet.Applet;
import java.awt.*;
/* <APPLET
 CODE=ParameterPassing.class
 WIDTH=200
 HEIGHT=200 >
 <PARAM NAME = company VALUE = "Welcome to Kogent">
</APPLET>  */
public class ParameterPassing extends Applet
{
 String company;
 public void start()
 {
     company = getParameter("company");
 }
 public void paint(Graphics g)
 {
     g.drawString(company, 10, 100);
 }
}
```

In the preceding program, the <PARAM> attribute has one parameter named company, and its value is "Welcome to Kogent". The getParameter() method returns the value stored in the company parameter.

Debugging Exercises

Q1. What happen when you compile and run the following applet?

```
import java.applet.Applet;
import java.awt.*;
public class HelloApplet extends Applet {
 public void repaint(Graphics g) {
     g.drawString("Hello from Java!", 60, 100);
 }
}
```

Ans. The preceding applet compiles successfully, but when you run this applet using appletviewer command, it does not print message (Hello from Java), because of improper use of the repaint() method in this applet. For running this program and displaying message (Hello from Java), replace the repaint() method to the paint() method.

Q2. What happen when you compile and run the following program?

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
/*
<applet CODE=KeyboardListenerTest.class WIDTH=300 HEIGHT=200>
</applet>
*/
public class KeyboardListenerTest extends Applet
{
 public void init()
 {
     Label label = new Label ("Enter any character :");
     add(label);
     TextField textField = new TextField(20);
     add(textField);
     textField.addKeyListener(this);
 }
 public void keyPressed(KeyEvent e)
 {}
 public void keyReleased(KeyEvent e)
 {}
 public void keyTyped(KeyEvent e)
 {
     showStatus(" Recently typed character is : " + e.getKeyChar());
 }
}
```

Ans. The preceding program does not compile because, we need to implement the KeyListener interface in this program to handle the keyboard event.

Q3. What happen when you compile and run the following applet?

```
import java.applet.Applet;
import java.awt.*;
public class TestApplet extends Applet {
 public void print(Graphics g) {
     g.drawString("Welcome all of you", 60, 100);
}
}
```

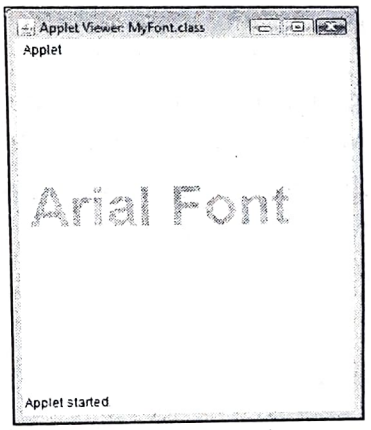
Ans. The preceding applet compiles successfully, but does not display any output in the applet window, because the TestApplet applet overrides the print() method. To display the message (Welcome all of you), we need to use the paint() method.

Q4. What will be the output of the following applet?

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

```
/*
<applet code="MyFont.class" width=300 height=300>
</applet>
*/
public class MyFont extends Applet
{
 public void paint(Graphics g)
 {
     setForeground(Color.green);
     Font arial = new Font("Arial", Font.BOLD, 50);
     g.setFont(arial);
     g.drawString("Arial Font", 10, 150);
 }
}
```

Ans. The output of preceding applet appears, as shown in Figure 18:



▲ **Figure 18: Showing the Message Arial Font in Green Color**

Q5. What happen when you compile and run the following program?

```
// Example of using the MouseListener interface
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet CODE=MouseListenerDemo.class WIDTH=300 HEIGHT=200>
</applet>
*/
public class MouseListenerDemo extends Applet implements MouseListener
{
 public String myMessage = "x = ?, y = ?";
 public void init()
 {
     // tell the applet to listen for mouse related events
     addMouseListener(this);
     repaint();
 }
 public void paint(Graphics g)
 {
     System.out.println("paint called");
     g.drawString(myMessage, 50, 50);
 }
 // Respond to a mouse press on applet
 public void mousePressed(MouseEvent me)
```

```
    {
        // display the x and y coordinate of mouse pointer
        myMessage = "x = " + me.getX() + ", y = " + me.getY();
        System.out.println("mouse button pressed");
        repaint();
    }
    // you must implement all the methods of MouseListener interface.
    public void mouseClicked(MouseEvent e)
    {
        System.out.println("mouse button clicked");
    }
    public void mouseEntered(MouseEvent e) {
        System.out.println("mouse enter in application area");
    }
    public void mouseExited(MouseEvent e) {
        System.out.println("mouse exit in application area");
    }
}
```

Ans. The preceding program does not compile successfully because this program implements the MouseListener interface that has the following methods:

- void mouseClicked(MouseEvent mouseE)
- void mouseEntered(MouseEvent mouseE)
- void mouseExited(MouseEvent mouseE)
- void mousePressed(MouseEvent mouseE)
- void mouseReleased(MouseEvent mouseE)

To implement the MouseListener interface in a program, we must override all methods of this interface. In the preceding program, the mouseReleased(MouseEvent mouseE) method is not overridden. Therefore, we need to override this method (mouseReleased(MouseEvent mouseE)) to successfully compile the preceding program.

Q6. What happen when you compile and run the following applet?

```
import java.awt.*;
public class MyApplet extends Applet {
 public void paint(Graphics g) {
     g.drawString("Hello from Java!", 60, 100);
     }
}
```

Ans. The preceding applet does not compile because the MyApplet applet extends the Applet class and, we need to import the java.applet.Applet class to compile the MyApplet applet.

Q7. What happen when you compile and run the following applet?

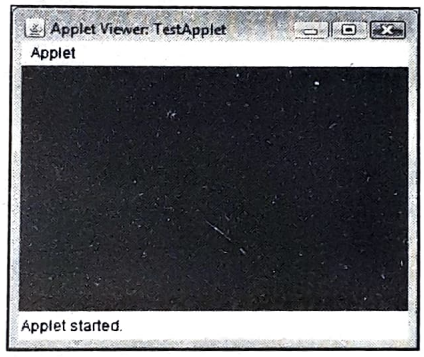
```
import java.applet.Applet;
public class MyApplet extends Applet {
 public void paint(Graphics g) {
     g.drawString("Hello from Java!", 60, 100);
     }
}
```

Ans. The preceding applet does not compile because the Graphics class is available in the java.awt package. The compiler generates error "cannot find symbol" at statement "public void paint(Graphics g)". For successful compilation of this applet, we need to import the java.awt package in this (MyApplet) applet.

Q8. What will be the output of the following applet?

```
import java.awt.*;
import java.applet.*;
/* <applet code ="TestApplet" width=300 height=200>
</applet> */
public class TestApplet extends Applet
{
 public void paint(Graphics g)
 {
     setBackground(Color.red);
 }
}
```

Ans. The output of preceding applet appears, as shown in Figure 19:



▲ **Figure 19: Showing the Red Color as a Background Color**

Q9. What happen when you compile and run the following applet?

```
import java.applet.Applet;
import java.awt.Graphics;
public class MsgApplet extends Applet {
 public void paint(Graphics g) {
     System.out.println("Welcome to Kogent");
     }
}
```

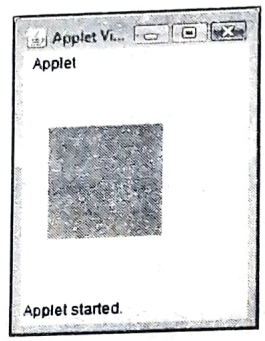
Ans. The preceding applet compiles and runs successfully, and the output ("Welcome to Kogent" message) is displayed on the command prompt window, not in the applet window. It is because the drawstring() method is not used in the `MsgApplet` applet to display the message.

Q10. What will be the output of the following applet?

```
import java.awt.*;
import java.applet.*;
/*
<applet code="FillColorRectangle.class"
 width=200
 height=250>
```

```
</applet>
*/
public class FillColorRectangle extends Applet {
 public void paint(Graphics g) {
     setForeground(Color.green);
     g.fillRect(20, 40, 90, 90);
 }
}
```

Ans. The output of preceding applet appears, as shown in Figure 20:



▲ **Figure 20: Drawing a Rectangle Filled with Green Color**

Q11. What happen when you compile and run the following applet?

```
import java.applet.Applet;
import java.awt.Graphics;
public class TestApplet extends Applet {
 public void paint(Graphics g) {
     drawString("Testing an applet", 10, 150);
     }
}
```

Ans. The preceding applet generates a compile time error, because to display string in applet window, we need to invoke the `drawString()` method using an object of the `Graphics` class, as shown in the following code snippet:

```
g.drawString("Testing an applet", 10, 150);
```

Q12. What happen when you compile and run the following applet?

```
import java.applet.*;
import java.awt.*;
public class TryThisApplet extends Applet {
 public void update(Graphics g) {
     g.drawString("Hi, How are u", 60, 100);
     }
}
```

Ans. The preceding applet compiles successfully, but does not display any output. It is because the TryThisApplet applet overrides the update() method instead of the paint() method, to display the message "Hi, How are u".