



Applet Programming

14.1 INTRODUCTION

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

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

Local and Remote Applets

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

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

A *remote applet* is that which is developed by someone else and stored on a remote computer

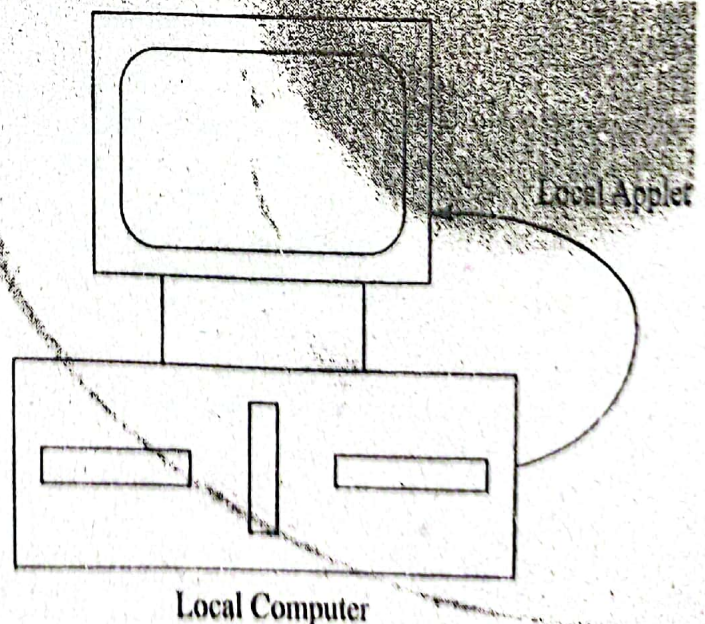


Fig. 14.1 Loading local applets

connected to the Internet. If our system is connected to the Internet, we can download the remote applet onto our system via at the Internet and run it (see Fig. 14.2).

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

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

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

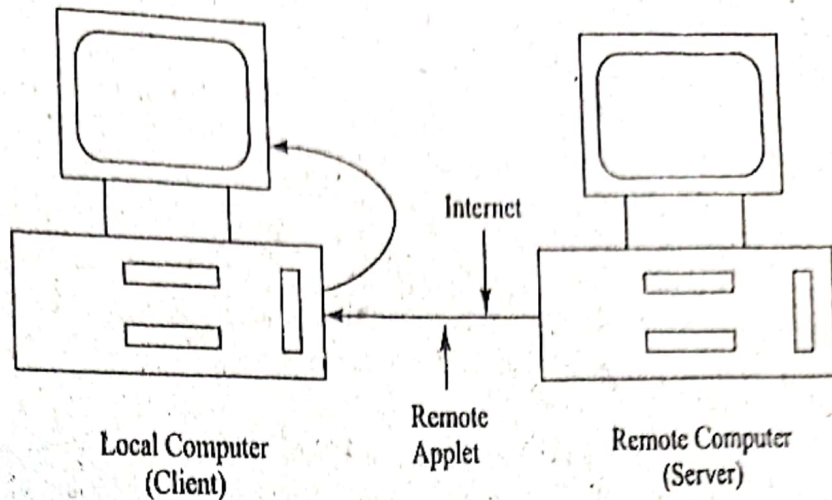


Fig. 14.2 Loading a remote applet

14.2 HOW APPLETS DIFFER FROM APPLICATIONS

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

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

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

14.3 PREPARING TO WRITE APPLETS

Creating simple Java application programs with a single main() method that runs. Here, we will be creating applets exclusively and

Graphical Input and Output

Program 16.11 creates a simple sequential student file interactively using window frames. The program uses the `TextFiled` class to create text fields that receive information from the user at the keyboard and then writes the information to a file. A record of information contains roll number, name, and marks obtained by a student in a test.

Program 16.11 Creating a file using text fields in windows

```
import java.io.*;
import java.awt.*;
class StudentFile extends Frame
{
    // Defining window components
    TextField number, name, marks;
    Button enter, done;
    Label numLabel, nameLabel, markLabel;
    DataOutputStream dos;

    // Initialize the Frame
    public StudentFile ( )
    {
        super ("Create Student File") ;
    }

    // Setup the window
    public void setup ( )
    {
        resize (400, 200) ;
        setLayout (new GridLayout (4, 2) ) ;
        // Create the components of the Frame
        number = new TextField (25) ;
        numLabel = new Label ("Roll Number") ;
        name = new TextField (25) ;
```



```

nameLabel = new Label ("Student name") ;
marks = new TextField (25) ;
markLabel = new Label ("Marks") ;
enter = new Button ("ENTER") ;
done = new Button ("DONE") ;

// Add the components to the Frame
add (numLabel) ;
add (number) ;
add (nameLabel) ;
add (name) ;
add (markLabel) ;
add (marks) ;
add (enter) ;
add (done) ;
// Show the Frame
show ( ) ;
// Open the file
try
{
    dos = new DataOutputStream (
        new FileOutputStream ("student.dat") ) ;
}
catch (IOException e)
{
    System.err.println (e.toString ( ) ) ;
    System.exit (1) ;
}
}
// Write to the file
public void addRecord ( )
{
    int num;
    Double d;
    num = (new Integer (number.getText ( ) ) ). intValue ( ) ;
    try
    {
        dos.writeInt(num);
        dos.writeUTF(name.getText ( ) ) ;
        d = new Double (marks.getText ( ) ) ;
        dos.writeDouble (d. doubleValue ( ) ) ;
    }
    catch (IOException e) { }

    // Clear the text fields
    number.setText (" ") ;
    name.setText (" ") ;
    marks.setText (" ") ;
}

// Adding the record and clearing the TextFields
public void cleanup ( )

```



```

{
    if (! number.getText ( ) . equals ( " " ) )
    {
        addRecord ( ) ;
    }
}
try
{
    dos.flush ( ) ;
    dos.close ( ) ;
}
catch (IOException e) { }

// Processing the event
public boolean action (Event event, object o)
{
    if (event.getSource instanceof Button)
    {
        if (event.getActionCommand ( ) . equals ( "ENTER" ) )
        {
            addRecord ( ) ;
            return true;
        }
    }
    return super.action (event, o) ;
}

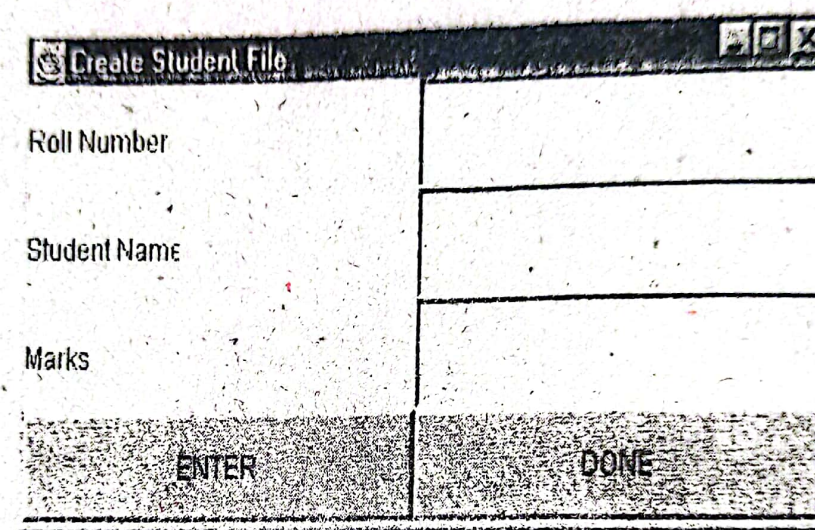
public boolean handleEvent (Event event)
{
    if (event.getSource instanceof Button)
    {
        if (event.getActionCommand ( ) . equals ( "DONE" ) )
        {
            cleanup ( ) ;
            System.exit (0) ;
            return true;
        }
    }
    return super.handleEvent (event) ;
}

// Execute the program
public static void main (String args [ ] )
{
    StudentFile student = new StudentFile ( ) ;
    student.setup ( ) ;
}
}

```

The program uses classes **Frame**, **TextField**, **Button**, and **Label** of **java.awt** package to create the window and the text fields required to receive a student record. The method **setup()** does the job of setting up the window. The method **addRecord()** writes the information to the "student.dat" file created earlier.

When we execute the program, a window appears on the screen to enable us to enter data (Fig. 16.16).



Create Student File	
Roll Number	<input type="text"/>
Student Name	<input type="text"/>
Marks	<input type="text"/>
ENTER	DONE

Fig. 16.16 Screen output produced by Program 16.11

After entering data in the appropriate text fields, we must click the **ENTER** button to write data to the file. This invokes the `addRecord()` which performs the task of writing to the file. When we click the **"DONE"** button, the program will call the method `cleanup()` which will close the file stream and terminate the program execution.

Program 16.12 reads the data stored in "student.dat" file by the previous program. The program opens the file for reading and sets up a window that is similar to the one created for writing. When we click the **NEXT** button, the program reads a record from the file and displays its content in the text fields of the window. A click on the **DONE** button closes the file stream and then terminates the execution.

Program 16.12 Reading a file using text fields

```
import java.io.*;
```