File handling 11

In this chapter, we will explore how to access, identify and manipulate files and directories on the hard disk through java programs. Through out the chapter we will focus on input/output processing and file handling. We will use the most common classes available in java.io package and few classes of java.util package. Let us begin the discussion with an overview of files and directories.

Understanding Computer Files

Storage devices of a computer system can be broadly classified into two categories: volatile storage and non-volatile Storage.

Volatile storage is temporary; values stored in variables are lost when a computer is shutdown. A Java program that stores a value in a variable uses Random Access Memory (RAM). Apart from variables, objects and their references are generally stored in RAM, once the program terminates or the computer shuts down, the data is lost.

Non-volatile storage is permanent storage; data is not lost when a computer loses power. When a Java program is saved on a disk, we are using permanent storage. A computer file is a collection of data stored on a non-volatile device. Files exist on permanent storage devices, such as hard disks, USB drives, optical disks and compact discs. Data stored in files is often called persistent data.

Files can be further classified broadly into two categories: text files and binary files.

Text files contain data that can be read in a text editor because the data has been encoded using a scheme such as ASCII or Unicode. Text files can be data files that contain facts, such as a payroll file that contains employee numbers, names, and salaries; or some text files can be program files or application files that store software instructions. Files created through editors like gedit, vi, pico are example of text files. They may have extensions like txt, java or c.

Binary files contain data that has not been encoded as text. Their contents are in binary format, which means the data is accessed in terms of bytes. Some example extensions of binary files are jpeg, mp3 and class.

Java language supports various operations that can be performed on file or on directories. Few operations that can be performed on files using java programs are listed below :

- Determining the path of a file or a directory
- Opening a file
- Writing to a file
- Reading from a file
- Closing a file

- Deleting a file
- Querying the attributes of a file

Java provides built-in classes that contain methods to help us with these tasks. These classes are present in java.io package. Java uses the concepts of streams and it provides two different categories of java classes to perform I/O operations on bytes and characters. A detailed discussion of reading from files and writing to files will be covered in the subsequent sections.

File Class in Java

The java.io.File class encapsulates information about the properties of a file or a directory. File class can be used to access attributes of files and directories. We can create/rename/delete a file or a directory. We can also access the attributes of a file like its permissions, length of a file, or last modification time. The creation of a file object that belongs to File class does not imply that the file or directory exists. A File object encapsulates a pathname or reference to a physical file or a directory on the hard disk.

There are nearly 30 methods of File Class that can be used to perform various operations on a file or a directory. However, File class does not provide any method to read from a file or write into a file, there are several stream classes to perform such operations. So, let us begin our study of few widely used constructors and methods of the File class.

Constructors of File Class

By using the File class, we can create a reference to any file by providing its absolute path in string format or by providing the relative path. The File class provides following constructors to refer a file or a directory.

```
File(String path)
File(String directory_path, String file_name)
File(File directory, String file_name)
```

Let us take an example for the above mentioned constructors. In Linux, "passwd" file present in "/etc" directory stores the information of the users existing in the system. Suppose we want to display its attributes, then its java file object can be created using three ways as mentioned below:

- By specifying the path as File fileobj = new File("/etc/passwd");
- By specfying directory and filename as two separate arguments

  ```
  File fileobj = new File("/etc", "passwd");
  ```

- By using the reference to directory encapsulated in dirobj object

  ```
  File dirobj = new File("/etc");

  File fileobj = new File(dirobj, "passwd");
  ```

Methods of File Class

The Table 11.1 summarizes few widely used methods of File class.

Method	Description
boolean exists()	Returns true if the file or directory exists, otherwise returns false
boolean isFile()	Returns true if the file exists, otherwise returns false
boolean isDirectory()	Returns true if the directory exists, otherwise returns false
boolean isHidden()	Returns true if the file or directory is hidden
String getAbsolutePath()	Returns the absolute path of the file or directory
String getName()	Returns the name of the file or directory referred by the object.
String getPath()	Returns the path to the file or directory
long length()	Returns the number of bytes in that file
String[] list()	Returns the name of files and directories in a directory
File[] listFiles()	Returns an array of abstract pathnames denoting the files in the directory.

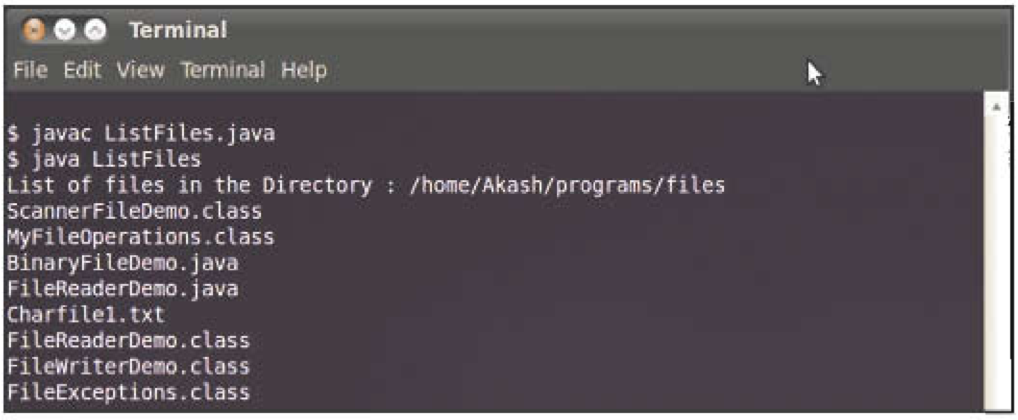
Table 11.1 : Few widely used method of File Class

Let us now try to understand a program that lists the file names in a given directory. After creating an object of file class that refers to a particular directory, we can use the list() method to list all the files present in that directory. The program given in code listing 11.1 demonstrates the listing of files present in "/home/Akash/programs/files" directory. In the program, we have used a variable to count the number of files and directories for a given directory, an array, listOfFiles of the File class is used to store the file objects present in the directory.

```
//List the contents of a Directory
import java.io.File;
public class ListFiles
{
        public static void main(String args[])
        {
                //Provide a Directory Path
                String path = "/home/Akash/programs/files";
                String files;
                int countOfFiles;
                try
                {
                    File folder = new File(path);
                    //Store the list of files in an array of File[] objects
                    File[] listOfFiles = folder.listFiles();
                    //count the number of files in the folder
                    countOfFiles = listOfFiles.length;

                    System.out.print("List of files in the Directory : ");
                    System.out.println(folder.getAbsolutePath());

                    //Iterate to display the name of each file
                    for(int i=0; i<countOfFiles; i++)
                    {
                         if(listOfFiles[i].isFile())
                         {
                              files = listOfFiles[i].getName();
                              System.out.println(files);
                         }
                    }
               }
               catch(Exception eobj)
               {
                    System.out.println(eobj);
               }
          }
     }
```

Code Listing 11.1 : Program to list files in a given directory

The output of code listing 11.1 is shown in figure 11.1



```
$ javac ListFiles.java
$ java ListFiles
List of files in the Directory : /home/Akash/programs/files
ScannerFileDemo.class
MyFileOperations.class
BinaryFileDemo.java
FileReaderDemo.java
Charfile1.txt
FileReaderDemo.class
FileWriterDemo.class
FileExceptions.class
```

Figure 11.1 : Output of Code Listing 11.1

Note : The above code lists files and directories present in the "/home/Akash/programs/files" directory. You may use appropriate file name and path to see the output.

Introduction to Streams

Till now, we haven't seen how to modify a file or display the contents of a file. To perform such operations we need to understand the concept of streams. Java uses stream classes to carry out read and write operations on files.

We have already studied the types of devices used for input and output. For instance, keyboard is an input device while monitor is an output device. Hard disk can be classified as both input and output device as we can store and read data from the files. There are various devices available in the market from different manufacturers and different capabilities. For example, hard disk are manufactured by various companies and come with different storage capacities like 500GB or 1 TB. Apart from this, hard disk can be connected using different cables like USB or SATA. However, a Java programmer does not need to worry about the technical details like type of hard disk or its capacity while developing a program to perform read/write operations over the files. This is possible because java language provides functionality of streams.

A stream is an abstract representation of an input or output device that is used as a source or destination for data. We can visualize a stream as a sequence of bytes that flows into the program or that flows out of our program. We can write data or read data using streams.

When we write data to stream, the stream is called an output stream. The output stream can transfer data from the program to a file on a hard disk or a monitor or to some other computer over the network. An input stream is used to read data from an external device to the program; it can transfer data from keyboard or from the file on a hard disk to the program.

The main reason for using streams for input or output operations is to make our program independent of the devices involved. Two advantages of streams are:

- Programmer does not need to worry about the technical details of the device
- The program can work for a variety of input/output devices without any changes to the source code.

Stream Classes in Java

To understand byte streams and character streams, let us first try to differentiate between character and byte representation. Let us take an example of number "5"; we can represent the number in two different ways as shown in table 11.2.

Representation	Particulars	Binary Representation
Character 5	ASCII Value : 53	00110101
Binary Number 5	Binary Value : 5	00000101

Table 11.2 : Character and Binary Representation

A character is generally stored using ASCII or Unicode format but when it is used for calculation purpose; its binary value is meaningful. Let us take one more example, the statement int i = 32; declares 'i' as an integer type variable that stores number 32. However 32 can be represented as two seperate characters '3' and '2'. The character representation is advisable when we write sentences like "Human beings have 32 teeth", where we are not performing any kind of operations over the number. But when we apply numerical calculations, we use data types like int, float or double that allows us to store the numbers in binary format.

Java supports two types of streams, byte stream and character stream. Streams that transfer data in the form of bytes to the file or devices are known as byte stream or binary stream. The files that are created using byte stream are known as binary files. Suppose if we wish to store variables like integer, double or boolean into a file, then we must use binary files. Binary files can also be used to store arrays or objects. Similarly text files and program codes are created using character stream. They can be opened in text editors like vi or SciTE.

Java provides two set of classes, character stream class and binary stream class. Character Stream classes present in java.io package deal with the character/text data while byte stream classes present in java.io package deal with binary data. Figure 11.2 shows the classification of stream classes.

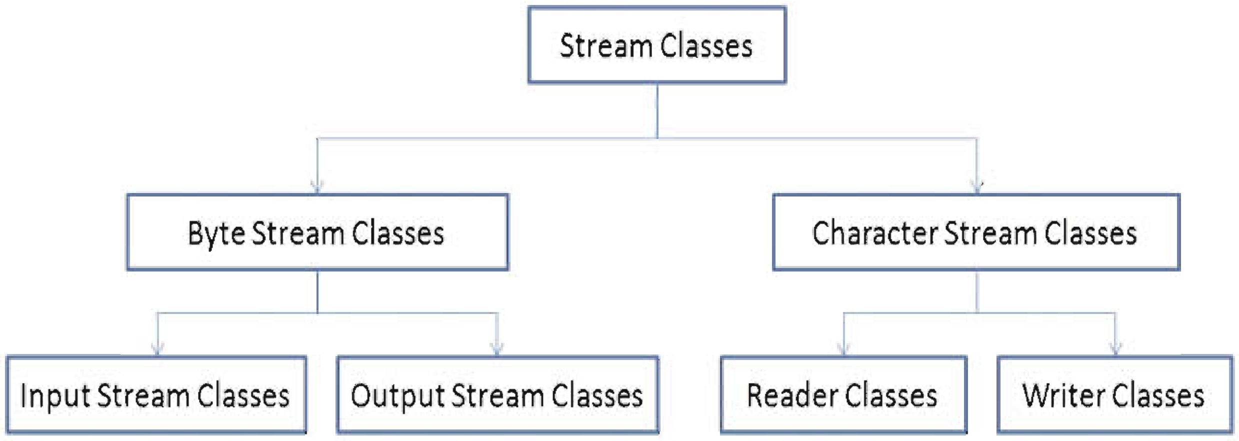


Figure 11.2 : Classification of Stream Classes

Java streams can be classified into two basic types, namely input stream and output stream. An input stream reads data from the source (file, keyboard) while the output stream writes data to the destination (file, output device).

The java.io package contains a collection of stream classes that support reading and writing in a file. To use these classes, a program needs to import the java.io package. Although there are many classes to perform character and byte operations, we will discuss the most widely used classes. Discussion of each class belonging to java.io package is out of the scope for this textbook.

Character Stream Classes

Character stream classes are a group of classes available in java.io package. They can be used to read and write 16-bit Unicode characters. Character stream classes can be further classified into Reader and Writer classes. Reader classes are a group of classes designed to read characters from files. The Writer classes are a group of classes designed to write characters into a file. Figure 11.3 shows the hierarchy of Character Stream Classes.

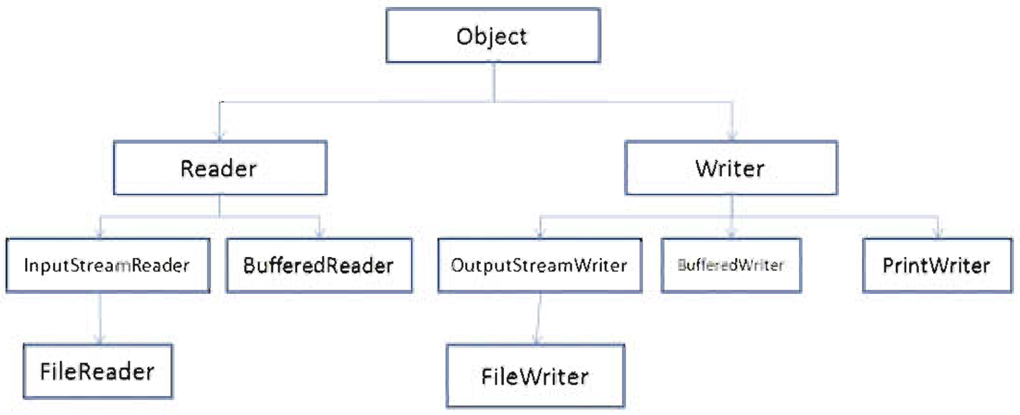


Figure 11.3 : Hierarchy of Character Stream Classes

As seen in the figure 11.3, java.io.Reader class and java.io.Writer class are inherited from the Object class. They are abstract classes (classes that cannot be used to create an object) and come with set of methods to be implemented by its subclasses. InputStreamReader and BufferedReader are the subclass of Reader Class. FileReader class is the subclass of InputStreamReader class. Similarly, OutputStreamWriter, BufferedWriter and PrintWriter are the subclasses of Writer class. FileWriter class is the subclass of OutputStreamWriter class.

Now, let us understand the constructor and methods of some of the above mentioned classes. A detailed description of methods, constructors can be obtained from the online Java documentation at http://docs.oracle.com/javase/6/docs/api/.

Writer Classes

Writer class is the base class for writing a character stream. The abstract Writer class defines the functionality that is available for all character output streams. We will be using the FileWriter class in our program to perform write operations.

The methods of Writer class can throw IOException. An IOException occurs when there is a failed I/O operation. IOException is a checked exception, so we must take care of it otherwise there will be a compiling error. Table 11.3 lists few of the methods of Writer class; these methods are used by its subclasses.

Method	Description
void close()	Closes the stream
void write(int c)	Writes the lower 16 bits of 'c' to the stream
void write(String s)	Writes string 's' to the stream

Table 11.3 : Few methods of FileWriter Class

The OutputStreamWriter class extends Writer class. It converts stream of characters to a stream of bytes. The FileWriter class extends OutputStreamWriter and outputs characters to a file. Some of its constructors are :

FileWriter(String filepath) throws IOException

FileWriter(File fileobj) throws IOException

FileWriter(String filepath, boolean append) throws IOException

In the above constructors, the parameter filepath is the full path name of a file and fileobj is a File class object that describes the file. In the last constructor, if append is true, characters are appended to the end of file, otherwise the existing contents of the file are overwritten. For example we can create an object of FileWriter as shown below:

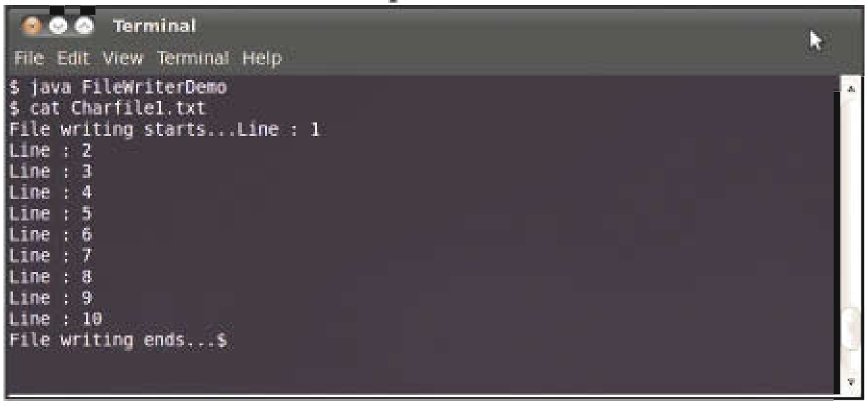
FileWriter fwobject = new FileWriter("/java/files/Charfile1.txt");

Let us take an example that illustrates how to write to a file. We will assume that the files are saved in the working directory. The code listing 11.2 shows how to create a file "Charfile1.txt" that does not exist. Further, we write few lines into that file using the write() method. Methods like write() and close() used in this program are inherited from the Writer class. Output of the code listing is shown in figure 11.4, after execution of the program, we use the cat command to display contents within the newly created file "Charfile1.txt".

```
// Write to a file using character stream
import java.io.*;
class FileWriterDemo
{
        public static void main(String args[])
        {
                FileWriter fwobject = null;
                try   {

                        // Create an object of FileWriter
                        fwobject = new FileWriter("Charfile1.txt");

                        //Write strings to the file
                        fwobject.write("File writing starts...");

                        for(int i = 1; i < 11 ; i++)
                                fwobject.write("Line : " + i + "\n");

                        fwobject.write("File writing ends...");
                }
                catch(Exception eobj)
                {
                        System.out.println(eobj);
                }
                finally
                {
                        try {
                        // Close the filewriter
                        fwobject.close();
                        }
                        catch(Exception eobj)
                        {
                                System.out.println(eobj);
                        }
                }
        }
}
```

Code Listing 11.2 : Program to illustrate file write operation

It is important to close the stream object after writing to a file is accomplished. Open files consume system resources, and depending on the file mode, other programs may not be able to access them. It's important to close files as soon as the operations are over.



```
$ java FileWriterDemo
$ cat Charfile1.txt
File writing starts...Line : 1
Line : 2
Line : 3
Line : 4
Line : 5
Line : 6
Line : 7
Line : 8
Line : 9
Line : 10
File writing ends...$
```

Figure 11.4 : Output of code listing 11.2

Reader Classes

Reader class is the base class for reading a character stream. The abstract Reader class defines the functionality that is available for all character input streams. We will be using the FileReader class in our program to perform read operations. Table 11.4 lists few of the methods of Reader class; these methods are used by its subclasses.

Method	Description
void close()	Closes the stream
int read()	Reads next available character from the stream, it returns "-1" to indicate the end of stream.

Table 11.4 : Few methods of FileReader Class

The InputStreamReader class extends Reader class. It converts a stream of bytes to a stream of characters. The FileReader class extends InputStreamReader class and reads characters from a file. Some of its constructors are:

FileReader(String filepath) throws FileNotFoundException

FileReader(File fileobj) throws FileNotFoundException

We can create an object of FileReader as shown below:

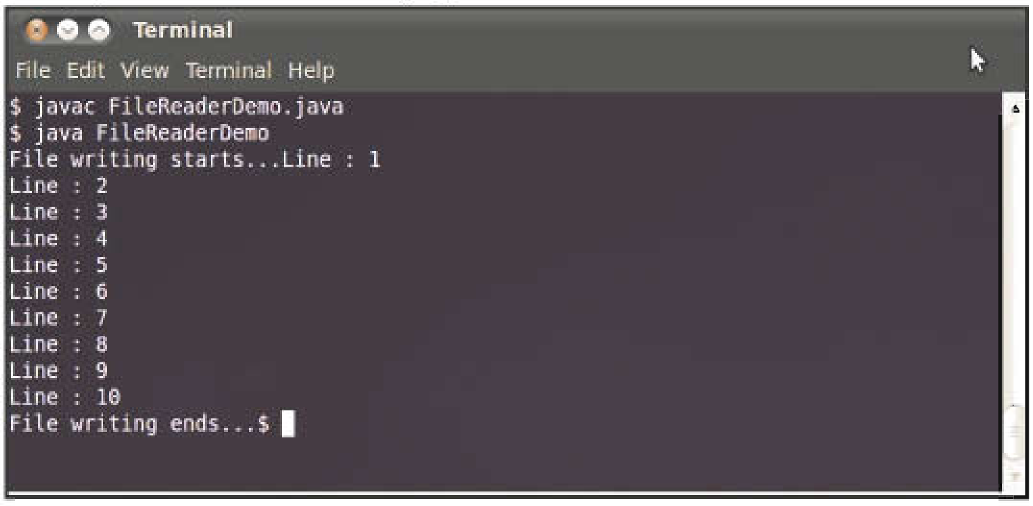
FileReader frobject = new FileReader("/java/files/Charfile1.txt");

In the program shown in code listing 11.2, we attempted to write to a file "Charfile1.txt". Now, let us write a program that reads from the file that we have already created. The program shown in code listing 11.3 reads data from the file using read() method of FileReader class. This read method is inherited from the Reader class.

```
// Reading from a file using character stream
import java.io.*;
class FileReaderDemo
{
        public static void main(String args[])
        {
                FileReader frobject = null;
                try   {

                        // Create an object of FileReader
                        frobject = new FileReader("Charfile1.txt");
                        int i;
                        char ch;
                        while ( ( i = frobject.read()) != -1)
                        {
                                ch = (char) i ;
                                System.out.print(ch);
                        }
                }
                catch(Exception eobj)
                {
                        System.out.println(eobj);
                }
                finally
                {
                        try {
                        // Close the filewriter
                        frobject.close();
                    }
                    catch(Exception eobj)
                    {
                        System.out.println(eobj);
                    }
                }
            }
        }
```

Code Listing 11.3 : Program to illustrate file read operation

Figure 11.5 shows the output of code listing 11.3, here it can be noticed that the execution of program displays the output on the screen. It displays all the contents of the file "Charfile1.txt".



```
$ javac FileReaderDemo.java
$ java FileReaderDemo
File writing starts...Line : 1
Line : 2
Line : 3
Line : 4
Line : 5
Line : 6
Line : 7
Line : 8
Line : 9
Line : 10
File writing ends...$
```

Figure 11.5 : Output of code listing 11.3

There is a special symbol to identify End of File (EOF). While reading from a file, program must identify that the file has ended. However the program reads from input stream so java read() method returns "-1" to identify the end of data in the stream.

Byte Stream Classes

Till now, we have seen how to process characters using java.io package. Most of the real life applications require numeric calculations like storing the details of inventory in a file and calculating the costs involved or like storing the employee details and their salaries in a file. For such kind of processing, java provides set of binary streams and their associated classes.

The FileInputStream and FileOutputStream classes in the java.io package give us the ability to read and write bytes from and into any files in the disk. These classes are the sub-classes of InputStream and OutputSream classes.

FileOutputStream

The FileOutputStream is a subclass of OutputStream and is used to write bytes to the file or some output stream. To use this class and its methods, first we need to create a file object, then we could use the write method derived from the abstract class OutputStream to write byte data into a file. Few widely used methods of FileOutputStream classes are shown in table 11.5:

Method	Description
void close()	Closes this file output stream and releases any system resources associated with the stream.
void write(int b)	Writes the specified byte to this file output stream.
void write(byte[] b)	Writes b.length bytes from the specified byte array to this file output stream.

Table 11.5 : Few methods of FileOutputStream Class

The constructors of FileOutputStream can accept either a string containing the path to the file location or an object of the File class. Let us see few constructors that are normally used :

FileOutputStream(String name) throws FileNotFoundException

Or

FileOutputStream(File file) throws FileNotFoundException

Examples of creating instances of FileOutputStream are as shown below:

```
FileOutputStream fosobject = new FileOutputStream("/home/Akash/myfile.txt");
```

Alternatively we can also use

```
File fobj = new File("/home/Akash/myfile.txt");
FileOutputStream fosobject = new FileOutputStream(fobj);
```

The above listed constructors can throw FileNotFoundException, if the file name refers to directory rather than a regular file, file does not exist, or file cannot be opened for some reason.

FileInputStream

FileInputStream is a subclass of InputStream and is generally used to read byte data from the files. It provides set of methods to perform write operations over the files as shown in table 11.6.

Method	Description
void close()	Closes this file input stream and releases any system resources associated with the stream.
int read()	Reads a byte of data from this input stream.
int read(byte[] b)	Reads up to b.length bytes of data from this input stream into an array of bytes.

Table 11.6 : Few methods of FileInputStream Class

Let us now see a program that uses the above described classes to perform write and read operations over a file as shown in code listing 11.4.

```
//Program to read and write bytes to binary file
import java.io.*;
class BinaryFileDemo {
        public static void main(String args[])
        {
                FileOutputStream outobject = null;
                FileInputStream inobject = null;
                String cities = " Rajkot \n Ahmedabad \n Vadodara \n Vapi \n";

                //Convert cities into byte array
                byte citiesarray[] = cities.getBytes();

                try {
                        // Create object of Binary output stream
                        outobject = new FileOutputStream("Binaryfile.dat");
                        //Write the array of bytes into file
                        outobject.write(citiesarray);
                        outobject.close();

                        //Create object of Binary input stream
                        inobject = new FileInputStream("Binaryfile.dat");
                        //Variable to read each byte
                        int i;
                        //Read each byte from the file and display
                        while((i = inobject.read())!=-1)
                        {
                                System.out.print((char)i);
                        }
                        inobject.close();

                }
                catch(Exception eobj)
                {
                        System.out.println(eobj);
                }
        }
}
```

Code Listing 11.4 : Program to read and write bytes to binary file

The program shown in code listing 11.4 writes a string to a file "Binaryfile.dat", later it creates an object of FileInputStream to read the data from the same file and display it on the screen. We must note that in the above program, typecasting is applied to convert integer to character after read operation is performed.

Note: to observe the difference between text file and binary file, open the files created using the programs given in code listing 11.2 and 11.4.

Processing Input from Keyboard

In this section, let us explore the different ways to input the data to java program through the keyboard. As we know, a program can get input of data from live interaction through keyboard/GUI or it may take input as command line arguments or from files. Although there are many classes that facilitate the input from keyboard, here we will discuss two of the most widely used technique, using Scanner class of java.util package, and using the Console class of java.io package.

Scanner Class

Scanner class belongs to the java.util package; it provides various methods to read input from the keyboard or from the file. A special feature of this class is that it breaks the input string into tokens (words) using a delimiter. (White space is the default delimiter). Each token can be of different type, for example a string like "India-1947" can be read as "String-int" values. Let us explore the constructors and few methods of the Scanner class.

Scanner(String str)

Scanner(InputStream isobject)

Scanner(File fobject) throws FileNotFoundException

A Scanner object can be created from a string, file object or InputStream object. For instance we may use the constructor in the following way to read from a file and keyboard respectively:

Scanner fileinput = new Scanner(new File("Students.dat"));

Scanner kbinput = new Scanner(System.in);

Some of the important methods available with the Scanner class are listed in table 11.7.

Method	Descrption
void close()	Closes the Scanner
String next()	Returns the next token
boolean hasNext()	Returns true if there is a token in input
int nextInt()	Scans the next token of the input as Int.
float nextFloat()	Scans the next token of the input as Float.
String nextLine()	Scans the next token of the input as Line

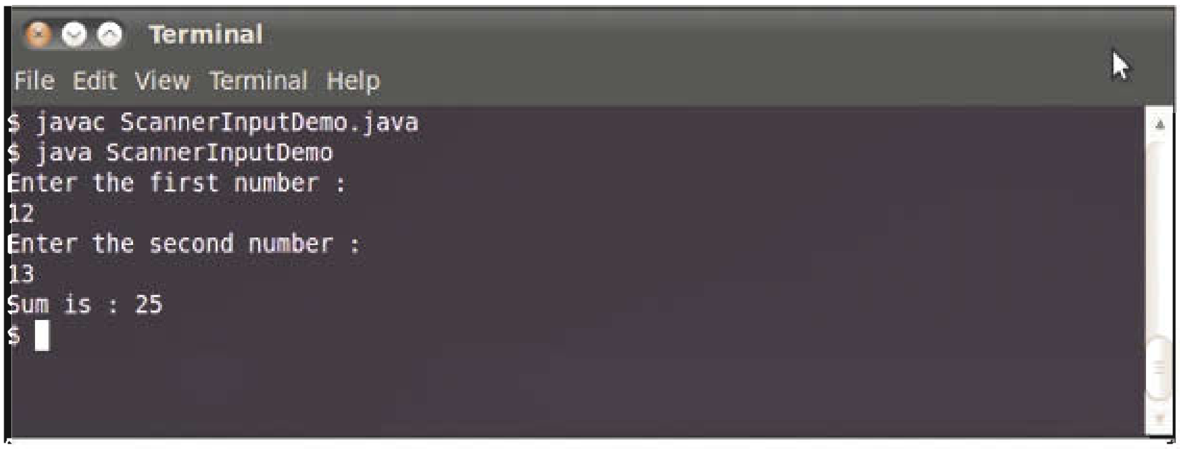
Table 11.7 : Methods of Scanner Class

Let us now see a program that reads two numbers interactively from the user and displays the addition of those two numbers. Code listing 11.5 shows the program.

```
//Accepts input at command prompt
import java.io.*;
import java.util.*;
class ScannerInputDemo
{
        public static void main(String args[])
        {
                Scanner kbinput = null;
                int number1;
                int number2;
                int sum=0;
                try
                {
                        // Create an object of Scanner class
                        // that reads from Standard Input
                        kbinput = new Scanner(System.in);
                        System.out.println("Enter the first number : ");
                        //Read the integer number from console
                        number1 = kbinput.nextInt();
                        System.out.println("Enter the second number : ");
                        //Read the integer number from console
                        number2 = kbinput.nextInt();
                        sum = number1 + number2;
                        System.out.println("Sum is : " + sum);
                }
                catch(Exception eobj)
                {
                        System.out.println(eobj);
                }
                finally {
                        try {
                                kbinput.close();
                        }
                        catch(Exception eobj)
                        {
                                System.out.println(eobj);
                        }
                }

        }
}
```

Code listing 11.5 : Program to add two numbers

In the code listing 11.5, we have created an object of Scanner class, the constructor of Scanner class accepts "System.in" as an argument so that it reads from the standard input (keyboard). Both the numbers are scanned as integer numbers using the nextInt() method of the Scanner class. Figure 11.6 shows the output of the program.



```
$ javac ScannerInputDemo.java
$ java ScannerInputDemo
Enter the first number :
12
Enter the second number :
13
Sum is : 25
$ 
```

Figure 11.6 : Output of Code Listing 11.5

Scanner class can also be used to read from a file. Let us see an example where we use the Scanner class to read the data from a file that contains information about few students. It is assumed that the file "Students.dat" already exists. It contains five fields, student_no, student_name, marks of three subjects. The format and data of the file Students.dat is as shown below:

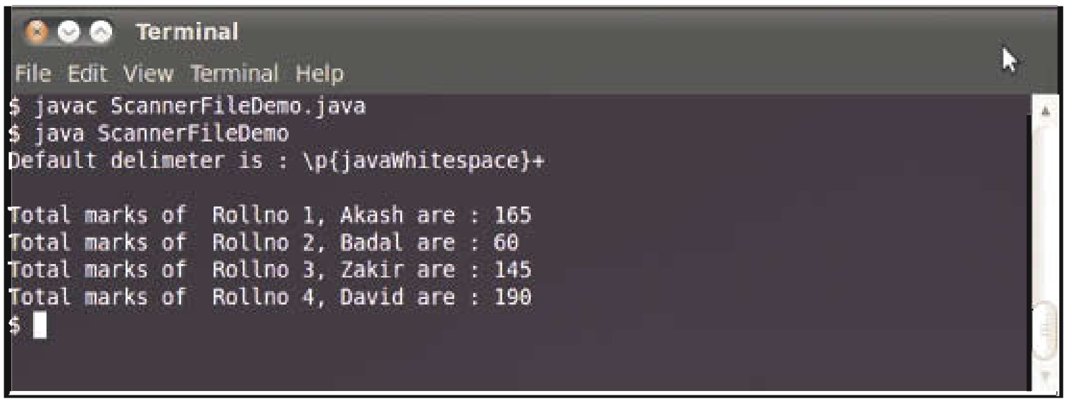
```
1 Akash 45 65 55
2 Badal 10 20 30
3 Zakir 45 40 60
4 David 65 50 75
```

We will write a program as shown in code listing 11.6 to read the data of each student and perform operations like calculating the total marks and displaying them on the output.

```
//Accepts input from a file "Students.dat" and calculate the total marks of each student
import java.io.*;
import java.util.*;
class ScannerFileDemo
{
        public static void main(String args[])
        {
                Scanner fileinput = null;
                int rollno, mark1, mark2, mark3, totalmarks;
                String name = null;
                File fobject;
        try
        {
                // Specify the file from where data is to be read
                fobject = new File("Students.dat");
                // Create an object of Scanner class that reads from File
                fileinput = new Scanner(fobject);
                //Display the default Delimiter to separate fields within a file
                System.out.println("Default delimeter is : " + fileinput.delimiter() + "\n");
                // Iterate to read the values of each record
                while(fileinput.hasNext()) {
                rollno = fileinput.nextInt();
                name = fileinput.next();
                mark1=fileinput.nextInt();
                mark2=fileinput.nextInt();
                mark3=fileinput.nextInt();
                totalmarks = mark1 + mark2 + mark3;
                System.out.println("Total marks of Rollno " + rollno + " ," +name+ " are
                                                                    : " + totalmarks);
                }
                fileinput.close();
            }
            catch(Exception eobj)
            {
                System.out.println(eobj);
            }
    }
    }
```

Code Listing 11.6 : Program to calculate the total marks of each student

The output of the program is displayed on the monitor as shown in figure 11.7.



```
$ javac ScannerFileDemo.java
$ java ScannerFileDemo
Default delimeter is : \p{javaWhitespace}+

Total marks of  Rollno 1, Akash are : 165
Total marks of  Rollno 2, Badal are : 60
Total marks of  Rollno 3, Zakir are : 145
Total marks of  Rollno 4, David are : 190
$
```

Figure 11.7 : Output of code Listing 11.6

Console Class

In the previous example, we used the Scanner class to read user input. Apart from the Scanner class, there is another class java.io.Console which can be used to get the input from the keyboard. We use this class especially when the input is to be typed in hidden form (characters must not be echoed on screen).

The Console class provides a method for reading password. When reading the password the user input will be hidden or not shown in the console screen. And it will return an array of character as the return type. The Console class belongs to java.io package. Few widely used methods of Console class are listed in table 11.8:

Methods	Description
String readLine()	This method reads a single line of text from the console.
char[] readPassword()	This method reads a password or passphrase from the console with echoing disabled.
Console printf(String format, Object args)	This method is used to write a formatted string to this console's output stream using the specified format string and arguments.

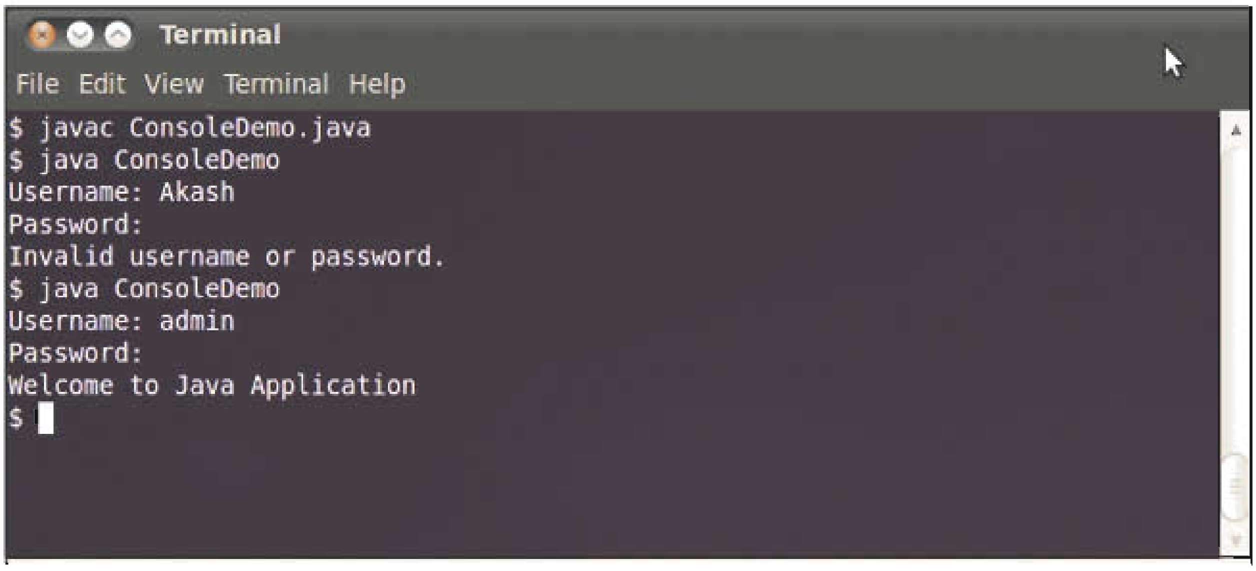
Table 11.8 : Few Methods of Console Class

The program shown in code listing 11.7 demonstrates the use of Console class to read username and password, further, it validates whether the username and password are correct or not.

```
// Program to reading passwords
import java.io.Console;
import java.util.Arrays;
public class ConsoleDemo {

    public static void main(String[] args) {
     Console console = System.console();
    String username = console.readLine("Username: ");
     char[] password = console.readPassword("Password: ");

   if (username.equals("admin") && String.valueOf(password).equals("secret")) {
      console.printf("Welcome to Java Application \n");
      } else {
      console.printf("Invalid username or password.\n");
      }
   }
}
```

Code Listing 11.7 : Program to read username and password

The output of the program is shown in figure 11.8. We have used two attempts to input the username and password. In the first attempt the username is false and that is why we use invalid username message. In the second attempt, we entered the correct username and password.



```
$ javac ConsoleDemo.java
$ java ConsoleDemo
Username: Akash
Password:
Invalid username or password.
$ java ConsoleDemo
Username: admin
Password:
Welcome to Java Application
$ 
```

Figure 11.8 : Output of Code Listing 11.7

The program can be extended to read list of users and match their passwords from a file in the presence of multi-user environment.

Apart from various classes discussed in this chapter, java provides classes to store and retrieve objects using files. It also provides classes and methods to access a file randomly. Till now, we have seen the methods that perform sequential operations; however, there are classes which allow us to directly jump to nth record instead of accessing it sequentially. Discussion of these classes being out of scope for this text, we leave it to the students for exploring various other interesting features of java.io package.

Summary

In this chapter we learnt about file handling operations. We saw how to use the java.io.File class to perform file operations. We learnt the concept of stream and saw how input and output streams of different types can be used. We also learnt how to use the Scanner class to access data from a keyboard or a file. Finally we learnt how to use the Console class to input data from keyboard.

EXERCISE

1. Why is a file important in java programming? Under what circumstances will you store the data in files ?
2. State various operations that can be performed on a file and directory.

3. Why is the concept of Streams introduced in java, give the advantages of using streams.

4. Choose the most appropriate option from the following :

   (1) Which of the following statements is true ?

   (a) Volatile storage lasts only a few seconds.

   (b) Volatile storage is lost when a computer is shutdown.

   (c) Computer disks are volatile storage devices.

   (d) All of the above are true.

   (2) Which of the following refers to a collection of data stored on a nonvolatile device in a computer system ?

   (a) file (b) application (c) volatile data (d) hard-disk

   (3) The data hierarchy occurs in which of the following order from the smallest to largest piece of data ?

   (a) file:character:field:record (b) file:character:record:field

   (c) character:field:file:record (d) character:field:record:file

   (4) Which of the following is true about streams ?

   (a) Streams always flow in two directions

   (b) Streams are channels through which the data flow

   (c) Only one stream can be open in a program at a time

   (d) All of the above are true

   (5) Which of the following is used as a separator between fields of a record ?

   (a) path (b) delimiter (c) variable (d) space

   (6) Scanner class can be used to for performing which of the following operations ?

   (a) accept input from the keyboard (b) read from the file

   (c) parse a string separated by delimiters (d) All of the above

LABORATORY EXERCISES

1. Write a java program to list all the files in a given directory with ".txt" extension.

2. Write a java program to enter a filename, your program must check whether the file exists or not, if it exists then display the properties of that file.

3. Write a java program to create a file "friends.dat"; write the name of your friends in that file using Writer classes.