UNIT - IV

EXPLORING JAVA.IO

SYLLABUS [B.E - OU]

Java I/O Classes and Interfaces, Files, Stream and Byte Classes, Character Streams, Serialization.

OUTLINES

4.1 JAVA I/O CLASSES AND INTERFACES

The java.io package includes several classes for handling files. These can be categorized as follows,

(1) File.

(2) InputStream.

(3) OutputStream.

(4) Reader.

(5) Writer.

(6) RandomAccessFile.

4.2 FILES CLASS

A file is a container for storing information. Most programs use files as the fundamental source and destination for data. Java treat directories in the same way as files. A directory is simply a file where we can store file names and other directory names.

File Class : The File class interact directly with files and the file system, unlike other classes that operate on streams. An object of file class contains the information about a disk file such as type of file, permissions, created data, modified data, full directory path.

Constructors

(1) **File(String path) :** This constructor creates the file object from the path name of the file.

(2) **File(String path, String fname) :** This constructor creates the file object from a path and file name.

(3) **File(File dirObject, String fname) :** This constructor creates the file object from the given directory and file name.

(4) **File(URI uriObject) :** This constructor creates file object from the file specified by uriObject.

Let us consider an example that creates three files.

```
File a1 = new File ("/");
File a2 = new File ("/", "myfile.txt");
File a3 = new File ("a1", "myfile.txt");
```

The first file 'a1' is created using only one argument i.e., directory path.

The second file 'a2' is created using two arguments i.e., directory path and file name.

The third file 'a3' is created using filepath (assigned to a1) and file name.

Methods : File class defines the following methods to manipulate file information,

(1) **String getName( ) :** Returns the name of the file or directory.

(2) **String getParent( ) :** Returns the parent directory name.

(3) **boolean exists( ) :** Returns true, if this file exists otherwise returns false.

(4) **boolean isFile( ) :** Returns true, if the invoking object is a file. Otherwise returns false. It returns false if file is not a normal file.

(5) **boolean isDirectory( ) :** Returns true, if the file denoted by object is a directory. Otherwise returns false.

(6) **boolean renameTo(File newfname) :** Renames the file with newfname and returns true, if the file is successfully renamed. Otherwise returns false.

(7) **boolean delete( ) :** Delete the file or the directory. Directory is deleted if and only if it is empty. It returns true, if the file is deleted. Otherwise it returns false.

(8) **String[ ] list( ) :** Returns a string array consisting of a list of files and directories present in a directory denoted by file object.

(9) **boolean mkdir( ) :** Creates a directory with the name specified by file object and returns true, if directory is created. Otherwise returns false.

(10) **String getAbsolutePath( ) :** Returns the absolute path of a file or directory denoted by file object.

(11) **boolean canRead( ) :** Determines whether a file has read permission or not. Returns true, if the file is readable. Otherwise returns false.

(12) **boolean canWrite( ) :** Determines whether a file has write permission or not. Returns true, if file is writable. Otherwise returns false.

Example

```
import java.io.*;
class UsingFile
{
    public static void main(String args[ ]) throws IOException
    {
        String s;
        File f = new File("c:/mydir/text.txt");
        s = f.exists( )? "File exists" : "File does not exist";
```

```
        System.out.println(s);
        s = f.isFile( )? "Is a normal file" : "Is a special file";
        System.out.println(s);
        File f1 = new File("c:/mydir/subdir");
        fl.mkdir( );
        if(f1.isDirectory( ))
        {
            System.out.println("Directory name : " + f1.getName( ));
            System.out.print("Directory contents : ");
            String str[ ] = f1.list( );
            if(str.length == 0)
            System.out.println("Directory is empty");
            else
            for(int i=0; i<str.length; i++);
            System.out.println(str[i]);
        }
        else
            System.out.println("Is not a directory");
    }
}
```

Output

```
File exists
Is a normal file
Directory name : subdir
Directory contents :
Directory is empty
```

4.3 STREAM AND BYTE CLASSES

4.3.1 Stream Classes

Stream is a flow of data between source and destination through a program. If the data flows from a source to a program, it is called input stream. If the data flows from a program to destination, it is called output stream. The source may be a file on hard disk, keyboard or a socket connected to other system. The destination may be computer screen, a file or a socket. These streams can be handled using the classes and interfaces provided by java.io package.

Java defines four abstract classes, InputStream, OutputStream, Reader and Writer. The two classes InputStream and OutputStream handles the byte streams and two classes Reader and Writer handles the character streams. These classes gives the basic functionality that is common to all subclass streams. The byte stream classes are used to read and write bytes or binary objects. The character stream classes are used to read or write characters or string.

The byte stream classes handles byte oriented I/O such as reading or writing the data bytes or executable programs or numerical data. The classes that support byte stream include,

(1) InputStream.

(2) OutputStream.

(3) FileInputStream.

(4) FileOutputStream.

(5) ByteArrayInputStream.

(6) ByteArrayOutputStream.

(7) SequenceInputStream.

(8) FilterInputStream.

(9) FilterOutputStream.

(10) BufferedInputStream.

(11) BufferedOutputStream.

(12) PushbackInputStream.

(13) PrintStream.

(14) RandomAccessFile.

The InputStream, OutputStream, FilterInputStream and FilterOutputStream are abstract classes. The InputStream and OutputStream classes are at the top of hierarchy that supports byte streams. Character-based stream have several advantages over byte streams. They can read or write 16-bit unicode character at a time whereas byte streams can read or write 8 bit byte at a time. The various classes that support character streams include,

(1) Reader.

(2) Writer.

(3) FileReader.

(4) FileWriter.

(5) CharArrayReader.

(6) CharArrayWriter.

(7) BufferedReader.

(8) BufferedWriter.

(9) PushbackReader.

(10) PrintWriter.

Reader and Writer classes are at the top of hierarchy that support character streams.

4.3.2 Byte Stream Classes

Byte streams operate on bytes and were introduced with JDK 1.0 version of Java. Byte streams operate on ASCII range of characters ranging from 0 to 255 (8 bit data).

Example : FileInputStream, FileOutputStream etc.

Byte streams can be divided into two divisions input stream classes (subclasses of InputStream class) and output stream classes (subclasses of OutputStream class). Input stream classes read byte by byte from the source (like file) and output stream classes write byte by byte to the destination (like file).

By inheritance, all classes derive from InputStream have basic methods, like read( ) for reading a single byte or an array of bytes. Likewise, all classes derived from OutputStream have basic methods, like write( ) for writing a single byte or an array of bytes.

Byte Stream Classes : All the I/O classes are placed in java.io package of Java API. The following figures show the class hierarchy of InputStream and OutputStream classes.

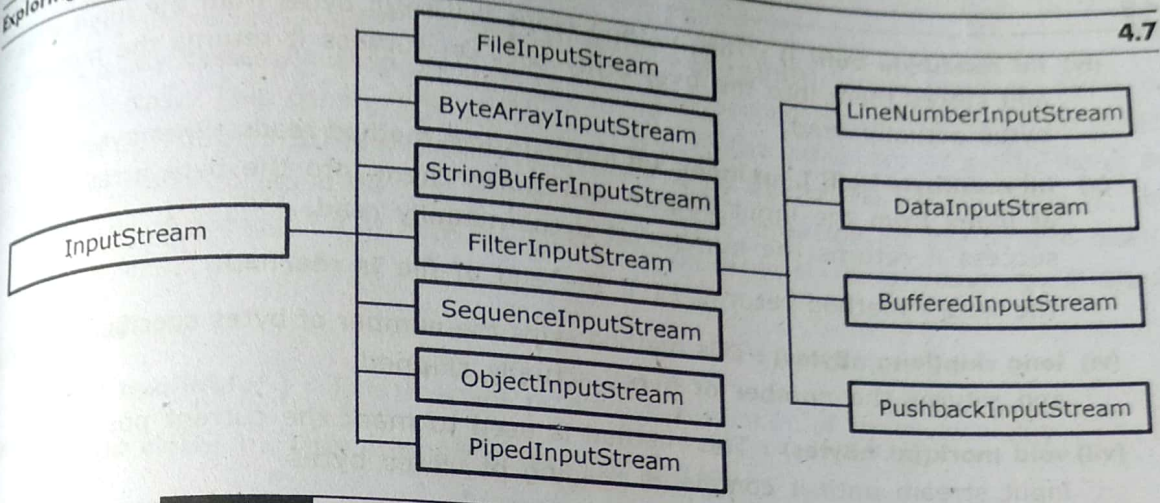


Fig. 4.3.1 The Class Hierarchy for Input Streams in Java.io

In the above hierarchy both InputStream and FilterInput Stream are abstract classes.

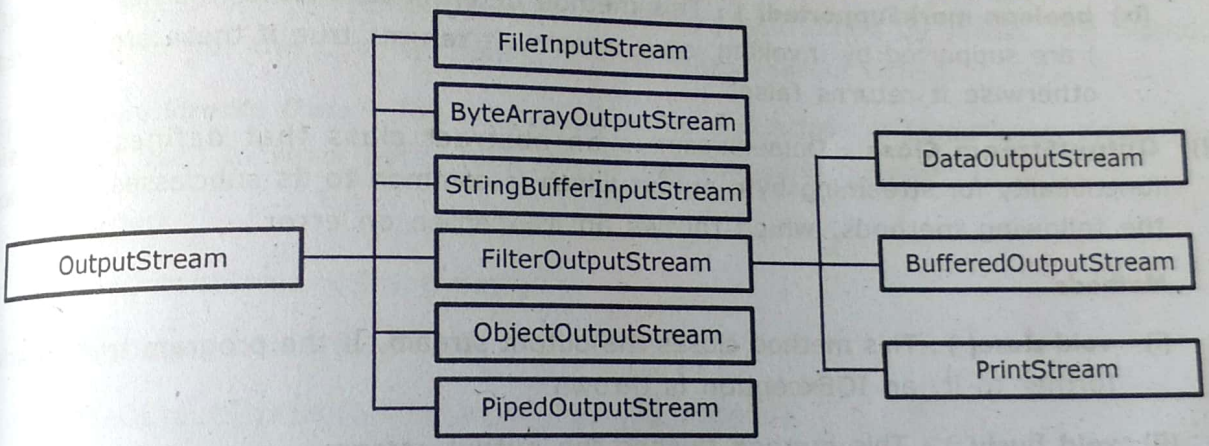


Fig. 4.3.2 The Class Hierarchy for Output Stream Class in Java.io

(1) **InputStream Class :** InputStream is an abstract class that defines the basic functionality for streaming byte input which is common to its subclasses. It defines the following methods, which throws an IOException on error.

Methods

(i) **int available( ) :** This method returns the number of bytes that are available for reading.

(ii) **void close( ) :** This method closes the input stream. If the program tries to read further from it, an IOExcpeiton is thrown.

(iii) **int read( ) :** This method reads and returns the integer representation of the next available byte from the input stream.

(iv) **int read(byte buff[ ])** : This method reads buff.length bytes from the input stream and stores them into the byte array buff. On success it returns the number of bytes actually read.

(v) **int read(byte buff[ ], int index, int nBytes)** : This method reads nBytes bytes starting at index from the input stream and stores them into the byte array buff, on success it returns the number of bytes actually read.

The read( ) method returns –1 if the end of file is reached.

(vi) **long skip(long nBytes)** : This method skips the number of bytes specified by nBytes and returns the number of bytes actually skipped.

(vii) **void mark(int nBytes)** : This method is used to mark the current position in the input stream until it completes reading of nBytes bytes.

(viii) **void reset( )** : This method resets the mark that was previously set on the input stream.

(ix) **boolean markSupported( )** : This method determines whether the mark( ) or reset( ) are supported by invoking input stream. It returns true if these are supported otherwise it returns false.

(2) **OutputStream Class** : OutputStream is an abstract class that defines the basic functionality for streaming byte output which is common to its subclasses. It defines the following methods, which throws an IOException on error.

Methods

(i) **void close( )** : This method closes the output stream. If the program tries to write further to it, an IOException is thrown.

(ii) **void flush( )** : This method flushes the output stream.

(iii) **void write(int b)** : This method writes a single byte to the output stream.

(iv) **void write(byte buff[ ])** : This method writes an array of bytes specified by buff to the output stream.

(v) **void write(byte buff[ ], int index, int nBytes)** : This method writes nBytes starting at index from the byte array buff to the output stream.

(3) **FileInputStream Class** : The FileInputStream is a subclass of InputStream class. This class is used to read bytes from a file. The object of it can be created by using following constructors,

Contructors

(i) FileInputStream(String path).

(ii) FileInputStream(File obj).

A FileNotFoundException is thrown by these constructors, the first form of constructor takes the absolute path of a file. The second constructor takes an object of File class. This constructor is useful to get all the attributes of a file. When an object of FileInputStream class is created it also opens the file for reading. The FileInputStream class does not provide the implementation to mark( ) and reset( ) methods. A IOException is thrown if there is an attempt to access these methods on FileInputStream object.

Methods

(i) **int available( ) :** This method returns the number of bytes available.

(ii) **void close( ) :** This method closes the stream.

(iii) **int read( ) :** This method reads a single byte from the input stream.

(iv) **long skip(long nBytes)( ) :** This method skips the specified number of bytes.

The getChannel( ) method is added to FileInputStream class by Java2. This method is used to get the channel connection to FileInputStream object.

(4) **FileOutputStream Class :** The FileOutputStream is a subclass of OutputStream class. This class is used to write bytes to a file. It has following constructors,

Contructors

(i) FileOutputStream(String path).

(ii) FileOutputStream(File obj).

(iii) FileOutputStream(String path, boolean append).

(iv) FileOutputStream(File obj, boolean append).

FileNotFoundException or a SecurityException is thrown by these constructors. The first constructor takes the absolute path of a file, the second constructor takes File object. The 'append' argument to third and fourth constructor tells whether a file can be appended or not. It can be either true or false. An IOException is thrown if you try to write to a read only file. A file is created when the object of FileOutputStream class is instantiated.

Methods

(i) **void close( ) :** Closes the output stream.

(ii) **void write( ) :** Writes the specified byte to this file output stream.

The getChannel( ) method is added to FileOutputStream classs by Java2. This method is used to get the channel connected to FileOutputStream object.

(5) **ByteArrayInputStream Class :** ByteArrayInputStream is a subclass of InputStream class. It takes a byte array as an input stream. This class has following constructors,

Constructors

(i) ByteArrayInputStream(byte a[ ])

(ii) ByteArrayInputStream(byte a[ ], int index, int nBytes).

These two constructors takes a byte array 'a' to read data from it. The second constructor takes the subset of array starting with the character at the specified index and containing nBytes bytes.

Methods : This class overrides both the mark( ) and reset( ) methods of InputStream class. If reset( ) is called without the call to mark( ) then stream pointer is set to the starting of the input stream.

(6) **ByteArrayOutputStream Class :** ByteArrayOutputStream is a subclass of OutputStream class. It takes a byte array as an output stream. This class has following constructors,

Constructors

(i) ByteArrayOutputStream( )

(ii) ByteArrayOutputStream(int nBytes)

The first form of constructor creates a default buffer of size 32 bytes. The second constructor specifies the size of the buffer to be created. The size of buffer is increased automatically when required. The buffer and number of bytes of a buffer are stored in the protected buf field and protected count field of ByteArrayOutputStream class respectively.

Methods

(i) void reset( ) : Resets the mark that was previously set.

(ii) int size( ) : Returns the size of the buffer.

(iii) void write(int b) : Writes a single byte.

(iv) void close( ) : Closes the stream.

(7) **SequenceInputStream Class :** The SequenceInputStream is a subclass of InputStream class. It takes several input streams and group them into a single input stream. It has following constructors,

Contructors

(i) SequenceInputStream(InputStream one, InputStream two).

(ii) SequenceInputStream(Enumeration enum).

The first form of constructor takes a pair of input streams. The program first reads the input stream 'one'. When it reaches to end of the input stream, 'one' then it switches to the next input stream in sequence i.e., input stream 'two'. The second form of constructor takes enumeration of input streams. In this case it continues through all of the input streams, until it reaches the end of the last input stream.

Methods

(i) **int available( ) :** Returns the number of available bytes.

(ii) **void close( ) :** Closes the input stream.

(iii) **int read( ) :** Read the next byte from the input stream. It returns –1, if the end of file is reached.

(8) **Filtered Byte Stream Classes :** Filtering means providing additional functionality such as buffering, character translation, and row data translation to the input or output streams. There are two filtered byte stream classes.

The FilterInputStream class filters the input stream and the FilterOutputStream class filters output stream. The constructor of these classes takes the following form,

Constructors

(i) FilterInputStream(InputStream inStream).

(ii) FilterOutputStream(OutputStream outStream).

FilterInputStream and FilterOutputStream are abstract classes. Their concrete subclasses gives some of the filtering capabilities provided by them. The methods defined by these classes are same as those defined by InputStream and OutputStream classes.

(9) **Buffered Byte Stream Classes :** Buffering an I/O operation increases the system performance. To achieve byte oriented buffering, a memory buffer is attached to the I/O streams and these streams extends a filtered stream class. There are two buffered byte stream classes,

(i) **BufferedInputStream Class :** The BufferedInputStream is a subclass of FilterInputStream class. This class allows to read a large amount (chunk) of logical data from a file into a memory buffer. When the program requests for the input data, it is read from the buffer instead of reading from the input device. When the buffer is empty the next local chunk of data from file is read into the buffer. Thus reading from the buffer reduces the number of actual physical input operations performed on the memory buffer when compared to the number of read request issued by the program.

BufferedInputStream class has following constructors,

Constructors

- BufferedInputStream(InputStream inStream).
- BufferedInputStream(InputStream inStream, int size).

The first form of constructor creates a buffered stream of default buffer size. The second constructor specifies the size of the buffer. Using the buffer size as multiples of memory pages, disk block and so on can increase the performance significantly.

Buffering an input stream we can move backward in the stream of the available buffer. In addition to read( ) and skip( ) methods, BufferedInputStream also supports the mark( ) and reset( ) methods.

(ii) **BufferedOutputStream Class :** The BufferedOutputStream is a subclass of FilterOutputStream class. This class allows to write the output to a memory buffer. When this buffer gets full, it is written to the actual output device. The practically filled buffer can also be written to the actual output device by calling the flush( ) method. The buffering output does not provide additional functionality unlike buffering input instead it increases the performance.

Constructors

- BufferedOutputStream(OutputStream outStream).
- BufferedOutputStream(OutputStream outStream, int size).

The first form of constructor creates a buffered output stream using a buffer of size 512 bytes. The second constructor specifies the size of the buffer.

(10) **PushbackInputStream Class :** The PushbackInputStream class uses buffering to implement pushback buffer. The pushback buffer is used to read (push) a byte from an input stream and then return it to the stream.

Constructors

(i) PushbackInputStream(InputStream inStream).

(ii) PushbackInputStream(InputStream inStream, int nBytes).

The first form of constructor creates a buffered stream that can return one byte to the input stream. The second constructor creates a buffered stream having a pushback buffer of length nBytes long. So this stream object can return multiple bytes to the input stream.

In addition to the methods of InputStream class, the PushbackInputStream class defines skip( ) and unread( ) methods.

(11) **PrintStream Class :** The PrintStream is a subclass of FilterOutputStream class. This class is used to output the formatted information to the specified stream. The System.out and System.err are the objects of PrintStream class. It has following constructors,

Constructors

(i) PrintStream(OutputStream outStream).

(ii) PrintStream(OutputStream outStream, boolean flush).

The first form of constructor outputs the text to the specified stream without flushing the buffer. The second form of constructor takes a boolean value 'flush' to flush the output stream automatically everytime a newline character(\n) is output. The flushing takes place if 'flush' is true. If it is false, flushing does not take place.

Methods : In addition to close( ), flush( ) and write( ) methods the PrintStream class defines two methods print( ) and println( ) for all types of objects including object class. If the argument passes to these methods is not a simple type then it calls the toString( ) method of the invoking object to print the result.

(12) **RandomAccessFile Class :** The RandomAccessFile class implements the interfaces DataInput and DataOutput. RandomAccessFile class allows a file to be accessed randomly for reading, writing and appending the data, the data in a file can be altered at any point.

Constructors

(i) RandomAccessFile(File obj, String mode).

(ii) RandomAccessFile(String filename, String mode).

The first form creates a stream to read from or write to file specified by the file object 'obj'. The second form creates a stream to read from or write to a file specified by 'filename'. 'mode' in both the constructors specifies the file access permission like read (r), read writer (rw) read write and every change made to data or metadata will be immediately written to the physical device,

Both constructors throw FileNotFoundException.

The RandomAccessFile class defines the following important methods,

Methods

(i) **getFilePointer( ) :** This method returns the current position of cursor (i.e., offset) in this file.

(ii) **read( ) :** This method reads a single byte of data.

(iii) **readDouble( ) :** This method reads a double from this file.

(iv) **readInt( ) :** This method reads a signed 32-bit integer from this file.

(v) **readLine( ) :** This method reads the next line of text from this file.

(vi) **write(int b) :** This method writes a single byte to this file.

(vii) **writeBytes(String str)** : This method writes the string str to the file as a sequence of bytes.

(viii) **close( )** : This method closes the stream.

(ix) **length( )** : This method returns the length of the file.

(x) **setLength(long length)** : This method sets the new length specified by 'length' of the file.

(xi) **seek(long pos)** : This is very important method. It is used to set the file pointer to the position specified by 'pos' in bytes within the file. Having set the file pointer using seek( ) method, the next read or write operation will occur at this new position.

(xii) **getChannel( )** : This method returns the channel connected to invoking object.

4.4 CHARACTER STREAM CLASSES

Character streams operate on characters and were introduced with JDK 1.1 version of Java. Character streams operate on Unicode range of characters ranging from 0 to 65,535 (16 bitdata).

Example : FileReader, FileWriter etc.

Character streams can be divided into two divisions reader classes and writer classes. Reader classes read character by character from a source and writer classes write character by character to the destination. Both FileInputStream and FileReader are capable to read data from a physical file, but they belong two different streams byte streams and character streams.

The byte streams and character streams (like FileInputStream of FileReader) are all sequential access streams; that is, they can read a file from a point-to-point, say start to finish. On contrast, **RandomAccessFile** is a byte stream class that is used to randomly access the contents of a file; that is, we can go to any point in a file and Character Stream Classes.

The following figures shows the class hierarchy of Reader and Writer class.

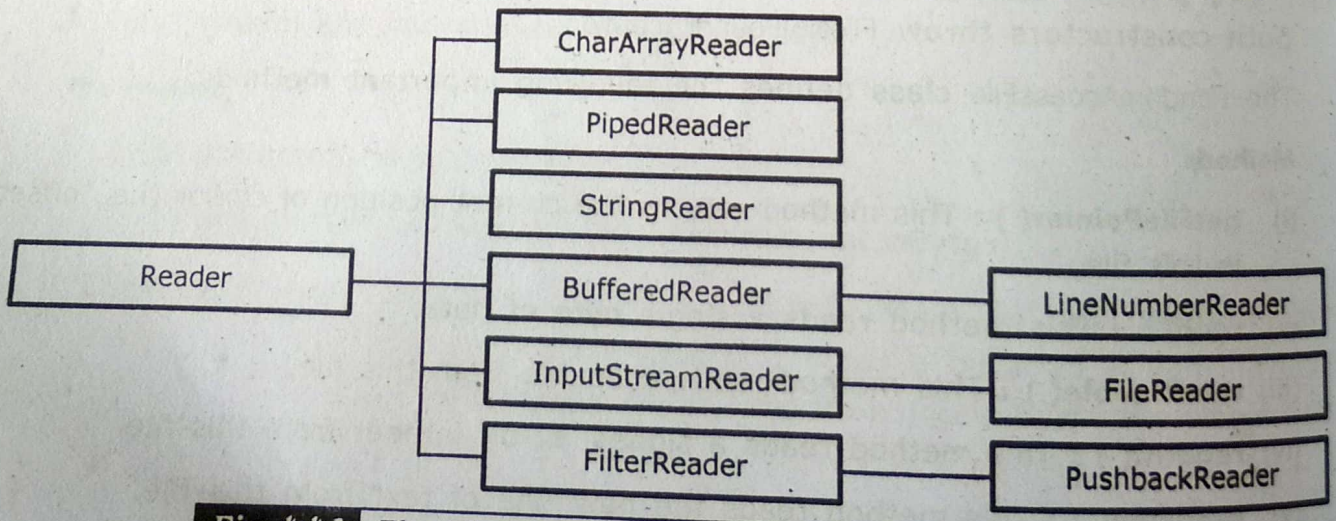


Fig. 4.4.1 The Class Hierarchy for Reader Classes in Java.io

In the above hierarchy Reader and FilterReader are abstract classes.

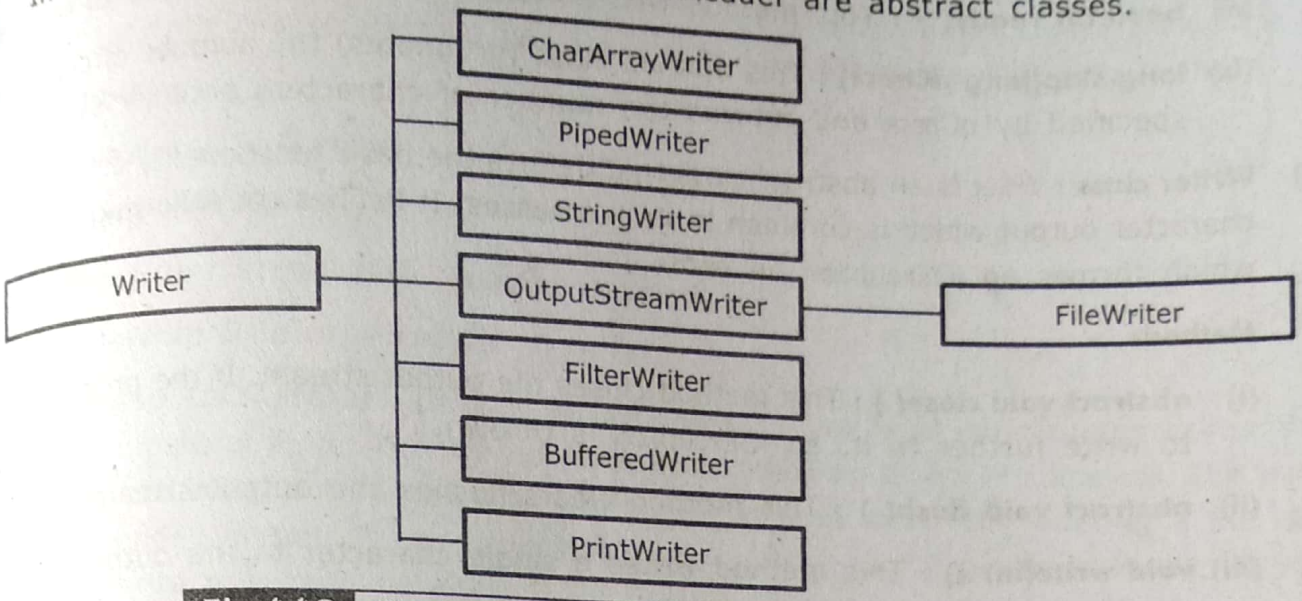


Fig. 4.4.2 The Class Hierarchy for Writer Classes in Java.io

(I) **Reader Class :** Reader is an abstract class that defines the basic functionality for streaming character input which is common to its subclasses. It defines the following methods, which throws an IOException on error.

Methods

(i) **abstract void close( ) :** This method closes the input stream. If the program tries to read further from it, an IOException is thrown.

(ii) **void mark(int nChars) :** This method is used to mark the current position in the input stream until it completes reading of nChars characters.

(iii) **void reset( ) :** This method resets the mark that was previously set on the input stream.

(iv) **boolean markSupported( ) :** This method determines whether the mark( ) or reset( ) are supported by invoking input stream. It returns true if these are supported, otherwise it returns false.

(v) **int read( ) :** This method reads and returns the integer representation of the next available character from input stream.

(vi) **int read(char buff[ ]) :** This method reads buff.length characters from the input stream and stores them into the character array buff. On success it returns the number of characters actually read.

(vii) **abstract int read(char buff[ ], int index, int nChars) :** This method reads nChars characters starting at index from the input stream and stores them into the character array buff. On success, it returns the number of characters actually read.

The read( ) method returns -1 if the end of file is reached.

(viii) **boolean ready( ) :** This method tells whether the stream is ready to be read.

(ix) **long skip(long nChars) :** This method skips (or ignores) the number of characters specified by nChars and returns the number of characters actually skipped.

(2) **Writer class :** Writer is an abstract class that defines the basic functionality or streaming character output which is common to its subclasses. It defines the following methods, which throws an IOException on error.

Methods

(i) **abstract void close( ) :** This method closes the output stream. If the program tries to write further to it, an IOException is thrown.

(ii) **abstract void flush( ) :** This method clears/flushes the output stream.

(iii) **void write(int c) :** This method writes a single character to the output stream.

(iv) **void write(char buff[ ]) :** This method writes an array of characters specified by buff to the output stream.

(v) **abstract void write(char buff[ ], int index, in nChars) :** This method writes nChars characters starting at index from the character array buff to the output stream.

(vi) **void write(String s) :** This method writes the string specified by s to the output stream.

(vii) **void write(String s, int index, int nChars) :** This method writes nChars characters starting at index from the specified string s to the output stream.

(3) **FileReader Class :** The FileReader is a subsclass of InputStreamReader class. It is used to read characters from a file. The object of it can be created using following constructors,

Constructors

(i) FileReader(String path).

(ii) FileReader(File obj).

A FileNotFoundException is thrown by these constructors.

The first form of constructor takes the absolute path of a file. The second constructor takes an object of file class.

Methods

(i) **int read( ) :** Reads a single character.

(ii) **int read(char buff[ ], int index, int nChars) :** Reads specified characters into an array.

(iii) **long skip(long nChars) :** Skips specified number of characters.

(iv) **void close( ) :** closes the stream.

(4) **FileWriter Class :** The FileWriter is a subclass of InputStreamWriter class. This class is used to write characters to a file. Its constructor takes the following forms,

Constructors

(i) FileWriter(String path).

(ii) FileWriter(File obj).

(iii) FileWriter(String path, boolean append).

(iv) FileWriter(File obj, boolean append).

These constructors throws an IOException. The first constructor takes the absolute path of a file. The second constructor takes an object of File class. The third and fourth constructors takes an argument 'append' that tells whether a file can be appended or not. It can be either true or false. An IOException is thrown if you try to write to a read only file.

Methods

(i) **void write(int c) :** Writes a single character to a file.

(ii) **void flush( ) :** Flushes the stream.

(iii) **void close( ) :** Closes the stream.

(5) **CharArrayReader Class :** CharArrayReader is a subclass of Reader class. It takes a character array as an input stream. This class has the following constructors,

Constructors

(i) CharArrayReader(char a[ ]).

(ii) CharArrayReader(char a[ ], int index, int nChars).

These two constructors takes a character array 'a' to read data from it. The second constructor takes the subset of array starting with the character at the specified index and containing nChars characters.

Methods

(i) **int read( ) :** Reads a single character.

(ii) **boolean ready( ) :** Determines whether the stream is ready to read.

(iii) **long skip(long nChars) :** Skips the specified number of characters.

(iv) **void close( ) :** Closes the stream.

(6) **CharArrayWriter Class :** CharArrayWriter is a subclass of Writer class. It takes a character array as an output stream. This class has following constructors,

Constructors

(i) CharArrayWriter( ).

(ii) CharArrayWriter(int nChars).

The first form of constructor creates a default buffer of size 32 bytes. The second constructor specifies the size of the buffer to be created. The size of buffer is increased automatically when required. The buffer and number of bytes of a buffer are stored in the protected buff field and protected count field of CharArrayWriter class respectively.

Methods

(i) **CharArrayWriter append(char c) :** Appends a single character to this writer.

(ii) **void flush( ) :** Flushes the stream

(iii) **void write(int c) :** Writes a single character.

(iv) **void writeTo(Writer obj) :** Writes the contents of the buffer to another character stream.

(v) **void close( ) :** Closes the stream

(vi) **char[ ] toCharArray( ) :** Returns character array containing the characters that have been written to this stream.

(7) **BufferedReader Class :** BufferedReader is a subclass of Reader class. This class buffers the input. Use of BufferedReader class improves the performance by reducing the number of times the data is actually read from the input stream. BufferedReader class has following constructors,

Constructors

(i) BufferedReader(Reader inStream).

(ii) BufferedReader(Reader inStream, int size).

The first constructor creates a buffered character stream of default buffer size. The second constructor specifies the size of the buffer. Like byte oriented streams, buffering an input character stream provides the foundation required to move backward in the stream of the available buffer. To support this the BufferedReader class overrides the mark( ) and reset( ) methods.

Methods

(i) int read( ) : Reads a single character.

(ii) String readLine( ) : Reads a line of text.

(iii) long skip(long nChars) : Skips the specified number of characters.

(iv) void close( ) : Closes the stream.

(8) **BufferedWriter Class :** BufferedWriter is a subclass of Writer class. This class improves the performance by buffering output. It has following constructors,

Constructors

(i) BufferedWriter(Writer outStream).

(ii) BufferedWriter(Writer outStream, int size).

The first form of constructor creates a buffered character stream of default buffer size. The second constructor specifies the size of the buffer.

Methods

(i) **void close( ) :** Closes the stream.

(ii) **void flush( ) :** Flushes the stream.

(iii) **void write(int c) :** Writers a single character to the stream.

(9) **PushbackReader Class :** The PushbackReader is a subclass of FilterReader class. It implements a pushback buffer which is used to store and return one or more characters to the input stream. It has following constructors,

Constructors

(i) PushbackReader(Reader inStream).

(ii) PushbackReader(Reader inStream, int size).

The first form of constructor creates a buffered stream that can return one character to the input stream. The second constructor creates a buffered stream having a pushback buffer of length specified by size. This constructor can return multiple characters to the input stream.

Methods

(i) **void close( ) :** Closes the stream

(ii) **int read( ) :** Reads a single characters.

(iii) **void unread(int c) :** Pushes back a single character

(iv) **void unread(char buff[ ]) :** Pushes back an array of characters into pushback buffer.

(v) **void unread(char buff[ ], int index, int nChars) :** Pushes back portion of array of characters into pushback buffer.

The unread( ) method throws an IOException if there is an attempt to return a character when the buffer is full.

(10) PrintWriter Class : PrintWriter is a subclass of Writer class. This class is used to output the formatted information to the specified stream. It has following constructors,

Constructors

(i) PrintWriter(OutputStream outStream).

(ii) PrintWriter(Writer outStream).

(iii) PrintWriter(OutputStream outStream, boolean flush).

(iv) PrintWriter(Writer outStream, boolean flush).

The first two constructors output the text to the specified stream without flushing the buffer. The last two constructors flushes the output stream everytime a newline character(\n) is output. The flushing is automatic if 'flush' is true. If it is false, flushing does not take place.

Methods : In addition to close( ), flush( ) and write( ) methods the PrintWriter class defines two methods print( ) and println( ) for all types of objects including object class. If the arguments passed to these methods is not a simple type then it calls the toString( ) method of the invoking object to print the result.

4.4.1 Example Programs

EXAMPLE PROGRAM 1

Write a program to read input from the keyboard and echo it on the monitor using Input StreamReader.

SOLUTION

```
import java.io.*;
class Kread
{
    public static void main(String args[ ]) throws IOException
    {
        char c;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        PrintWriter pw = new PrintWriter(System.out, true);
        pw.println("Enter character : ");
        c = (char)br.read( );
        pw.println(c);
    }
}
```

Output


```
Enter character : a
a
```

EXAMPLE PROGRAM 2

Write a program that uses a SequenceInputStream to output the contents of two files.

SOLUTION

```
import java.io.*;
class CopyTwoFiles
{
    public static void main(String args[ ]) throws IOException
    {
        FileInputStream f1 = new FileInputStream("First.txt");
        FileInputStream f2 = new FileInputStream("Second.txt");
        SequenceInputStream s = new SequenceInputStream(f1, f2);
        FileOutputStream out = new FileOutputStream("Third.txt");
        int ch;
            System.out.println("Output is on screen and in file third.txt");
        while((ch=s.read( ))!=-1)
        {
            System.out.print((char)ch);   // output on screen
            out.write(ch);                // output to file
        }
    }
}
```

Output

```
Output is on screen and in file third.txt
This is first file. This is second file.
```

4.5 SERIALIZATION

Serialization is the process of storing object's state where an object's state is nothing, but the values of the instance variable. The objects may have references to some other objects or to themselves forming a directed object graph. Serializing an object serializes all the other referenced objects in object graph recursively. Deserialization is the process of retrieving the object's state. Deserializing an object restores all of these objects and their references correctly.

Serialization is important if you want to store the state of your program example in a file then restore it at a later time using deserialization. Serialization is extensively used in Remote Method Invocation (RMI). In RMI the object of one class on one machine invokes the method of another class on a different machine. Here two sending machine serializes the object and transmits it. So that it could be reconstructed on a different machine in the same state as it was existing before on the original machine. The receiving machine deserializes the object to restore its state.

The interfaces and classes that support serialization are described below.

4.5.1 Serializable Interface

Serializable Interface : Objects of a class that implements Serializable interface can only be stored and retrieved i.e., a class that implements this interface is a serializable class. This interface does not define any methods. It is used to indicate that the objects of this class are serializable. Subclasses of a class that implements serializable interface are also serializable. Serialization does not store the variables that are declared as transient and static.

4.5.2 Externalizable Interface

Externalizable interface give the programmer control over saving and restoring the state of an object. This interfaces defines two methods.

void readExternal(ObjectInput in)

void writeExternal(ObjectOutput out)

Here 'in' is an object of byte stream to read the object from it 'out' is an object of byte stream to write the object to it. These two methods throws an IOException. The readExternal( ) method may also throw ClassNotFoundException.

4.5.3 DataOutput Interface

The DataOutput interface defines several methods to convert data from any of the Java's primitive types to a series of bytes and write these bytes to a binary output stream. It also defines the methods that convert a string into a Java modified UTF-8 format and write these bytes to a binary output stream. The DataOutput interface defines the following methods.

For a byte or for a byte array it defines the overloaded versions of write( ) method. For unicode strings, it defines writeBoolean( ) writeByte( ), writeBytes( ), writeChar( ) and writeChars() methods.

To display modified text for unicode, it defines writeDouble( ), writeFloat( ), writeInt(0, writeLong( ), writeShort( ) and writeUTF( ) methods.

4.5.4 Object Output Interface

The ObjectOutput interface is derived from DataOutput interface. This interface defines following methods to support object serialization. These methods throws an IOException on any error.

Methods

void close( ) : Closes the stream.

void flush( ) : Flushes the stream.

void write(byte buff[ ]) : Writes the contents byte array buff to the invoking stream.

void write(byte buff[ ], int index, int nBytes) : Writes the portion of byte array buff to the invoking stream.

void write(int b) : Writes a single byte

void writeObject(Object object) : This is an important method used to serialize an object. This method writes an object "object" to the invoking stream.

4.5.5 ObjectOutputStream Class

The ObjectOutputStream class is derived from the OutputStream class and ObjectOutput interface. This class is used for writing objects to a stream. It has the following constructor.

ObjectoutputStream(OutputStream out).

This constructor throws an IOException. It defines the following methods. All of the methods throws an IOException on any error.

void close( ) : Closes the stream.

void flush( ) : Flushes the stream.

void write(int b) : Writer a single byte.

void writeBoolean(boolean bool) : Writes a boolean.

void writeChar(int ch) : Writes a single character.

4.5.6 ObjectInputStream Interface

The ObjectInput interface is derived from DataInput interface. This interface defines the following methods to support object serialization. These methods throws an IOException on any error.

int available( ) : Returns the number of bytes available for reading.

void close( ) : Closes the stream.

int read( ) : This method reads and returns the integer representation of the next available byte from input stream.

int read(byte buff[ ]) : This method reads buff.length bytes from the input stream into byte array buff[ ]. On success it returns the number of bytes actually read.

int read(byte buff[ ], int index, int nBytes) : This method reads nBytes bytes into byte array buff from the input stream. On success it returns the number of bytes actually read.

The read( ) method returns -1 if the end of file is reached.

Object readObject( ) : This is an important method used to deserialize an object. This method reads an object from the input stream.

long skip(long nBytes) : Skips the specified number of bytes and returns actual number of bytes skipped.

Example : The program below, illustrates the object serialization and deserialization.

```
import java.io.*;
class SerializedClass implements Serializable
{
    String name;
    int pages;
    double price;
    public SerializedClass(String n, int p, double m)
    {
        name = n;
        pages = p;
        price = m;
    }
    public String toString( )
    {
        return "Name : " + name + "\n Pages : "+ pages + "\n Price : " + price;
    }
}
```

```
public class Serialization
{
    public static void main(String args[ ])
    {
        // Object's serialization
        try
        {
            SerializedClass obj1 = new SerializedClass("Java", 250, 350.00);
                System.out.println("Object's contents saved : \n" + obj1);
                FileOutputStream fStream = new FileOutputSream("output");
                ObjectOutputStream oStream = new ObjectOutputStream(fStream);
                oStream.writeObject(obj1);
                oStream.flush( );
                oStream.close( );
        }
        catch(Exception e)
        {
            System.out.println("Error while Serialization : " + e);
            System.exit(0);
        }
        // Object's deserialization
        try
        {
            SerializedClass obj2;
            FileInputStream fileStream = new FileInputStream("output");
            ObjectInputStream objStream = new ObjectInputStream(fileStream);
            obj2 = (SerializedClass)objStream.readObject( );
            objStream.close( );
            System.out.println("Object's contents restored : \n" + obj2);
        }
```

```
            catch(Exception e)
            {
                System.out.println("Error while Desiralization : " + e);
                System.exit(0);
            }
        }
    }
```

Output

```
Object's contents saved :
Name : Java
Pages : 250
Price : 350.00
Object's contents restored :
Name : Java
Pages : 250
Price : 350.00
```

4.6 EXAMPLE PROGRAMS

EXAMPLE PROGRAM 1

Copy first 3/4 of the source file into one destination file and the last 1/4 to another destination file.

SOLUTION

In the following program, WhileDemo.java is the source file and abc.txt and def.txt are the destination files. length( ) method of File class returns the size of the file.

File Name : TwoDestinations.java

```
import java.io.*;
public class TwoDestinations
{
    public static void main(String args[ ]) throws IOException
    {
        File myfile = new File("whileDemo.Java");
        long size = myfile.length( );
        int count = 0;
```

```
        FileInputStream fis = new FileInputStream(myfile);
        FileOutputStream fos 1 = new FileOutputStream("abc.txt");
        FileOutputStream fos2 = new FileOutputStream("def.txt");
        int k;
        while ((k = fis.read( ) ) ! = -1)
        {
            if(count++ <= size *0.75)
                fos1.write(k);
            else
                fos2.write(k);
        }
        fos2.close( ); fos1.close( ); fis.close( );
    }
}
```

EXAMPLE PROGRAM 2

Print the contents of a file in the reverse order at DOS prompt (1st character in the file as the last one and last character as the 1st one).

SOLUTION

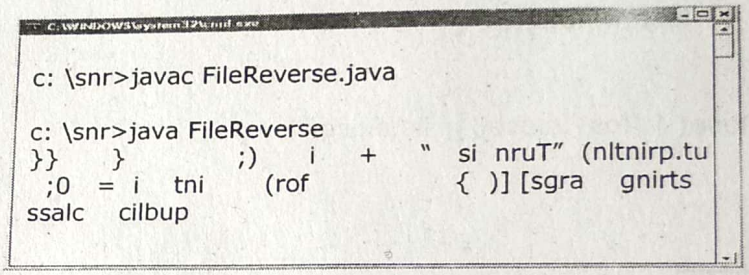
In this program, all the contents of the file read line by line is appended to a StringBuffer. Later we used reverse( ) method of StringBuffer to reverse whole thing present in the StringBuffer. It is to be noted that reverse( ) is a method of StringBuffer not does not exist in String class.

File Name : FileReverse.java

```
import.java.io.*;
public class FileReverse
{
public static void main(String args[ ]) throws IOException
{
    FileReader fr = new FileReader("ForLoop.java");
    BufferedReader br = new BufferedReader(fr);
    StringBuffer sb = new StringBuffer( );
    String str;
```

```
        while ((str = br.readLine( ) ) ! = null)
            {
                sb.append(str);
            }
            System.out.println(sb.reverse( ));
            br.close( );
            fr.close( );
        }
    }
```

Output



EXAMPLE PROGRAM 3

Print each line of the file in the reverse order (that is, each line is to be reversed) and print at DOS prompt.

SOLUTION

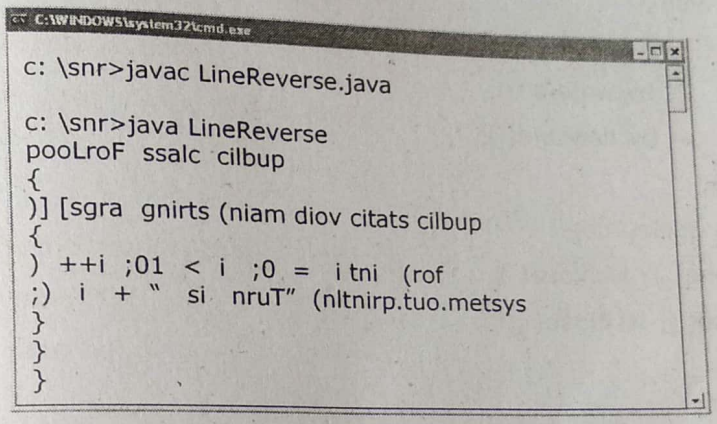
It is slightly a modification of previous program. Instead of reversing whole data of the file, we reverse each line. That is, when we read a line, we reverse and print immediately before another line is read.

File Name : LineReverse.java

```
import java.io.*;
public class LineReverse
{
    public static void main(String args[ ]) throws IOException
    {
        FileReader fr = new FileReader("ForLoop.java");
        BufferedReader br = new BufferedReader(fr);
        String str;
        while ((str = br.readLine( ) ) ! = null)
        {
```

```
            StringBuffer sb = new StringBuffer(str);
            System.out.println(sb.reverse( ));
        }
        br.close( );
        fr.close( );
    }
}
```

Output



EXAMPLE PROGRAM 4

Prompt the user to enter something. What are the enters is to be written to a disk file until the type $ symbol?

SOLUTION

File Name : TypedAndSaved.java

```
import java.io.*;
public class TypedAndSaved
{
    public static void main(String args[ ]) throws IOException
                                                    //to read from Keyboard
    {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);  //to write to a disk file
        FileWriter fw = new FileWriter("abc.txt");
        BufferedWriter bw = new BufferedWriter(fw);
        String str;
            do
```

```
        {
            System.out.println("Enter sme thing and type $ to end: ");
            str = br.readLine( );
            if(str.equals("$"))
            {
                break;
            }
            else
            {
                bw.write(str);
                bw.newLine( );
            }
            } while(true);
        bw.close( ); fw.close( );
        br.close( ); isr.close( );
    }
}
```

(i) InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
The above two lines can be replaced with as follows;
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

(ii) FileWriter fw = new FileWriter("abc.txt");
BufferedWriter bw = new BufferedWriter(fw);
The above two lines can be replaced with as follows:
BufferedWriter bw = new BufferedWriter(new FileWriter("abc.txt"));
While closing, close only bw and br.

EXAMPLE PROGRAM 5

Ask the user to enter a multiplication table number. The table is to be printed in a file and also at DOS prompt.

SOLUTION

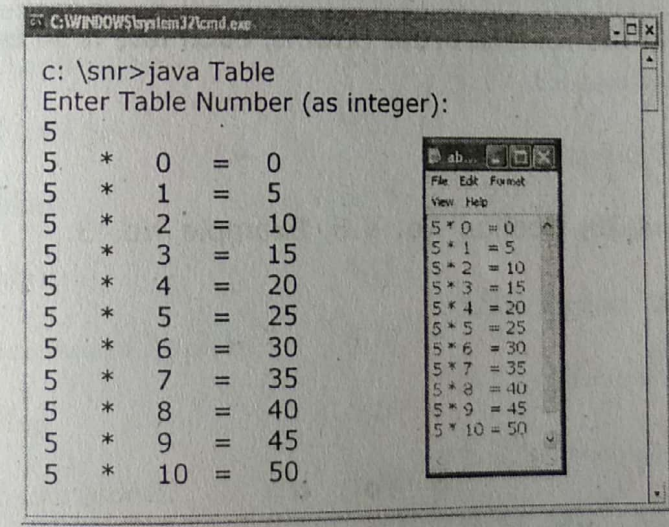
File Name : Table.java

```
import java.io.*;
public class Table
{
```

```
public static void main(String args[ ]) throws IOException
{
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));   //to read from keyboard

    BufferedWriter bw = new BufferedWriter(new FileWriter("abc.txt"));          //to write to a disk file
    System.out.println("Enter Table Number(as integer):");
    String str = br.readLine( );                                                //reading input as a string
    int num = Integer.parseInt(str);                                            //parsing string to int value
    for(int i = 0; i < 11; i++)
    {
        String s1 = num + "*" + i + "=" + num*i;
        System.out.println(s1);                                                 //writing at DOS prompt
        bw.write(s1);                                                           //writing to disk, abc.txt
        bw.newLine( );
    }
    bw.close( );
    br.close( );
    }
}
```

Output



❖ ❖ ❖

EXPECTED UNIVERSITY QUESTIONS WITH ANSWERS

Q1) What is a file? What are the constructors and methods of file class?

Answer :

Refer Page No. 4.2, Section No. 4.2.

Q2) Briefly discuss about the byte stream and classes.

Answer :

Refer Page No. 4.5, Section No. 4.3.

Q3) What are the character stream classes present in java.io package. Explain.

Answer :

Refer Page No. 4.14, Section No. 4.4.

Q4) Write a program that uses a SequenceInputStream to output the contents of two files.

Answer :

Refer Page No. 4.21, Section No. 4.4.1, Example No. 2.

Q5) What is serialization? Explain the purpose of serialization.

Answer :

Refer Page No. 4.22, Section No. 4.5.

Q6) Print each line in the reverse order (that is, each line is to be reversed) and print at DOS prompt.

Answer :

Refer Page No. 4.28, Section No. 4.6, Example No. 3.