Chapter – 4

Java Input Output

JAVA INPUT/OUTPUT PACKAGE:

All the input/output operations are handled by java.io package. All the classes that are needed to perform input/output operations are contained inside java.io package. A stream can be defined as a sequence of data and it produce or consumes information. A stream is linked to a physical device by java input/output system. The InputStream is used to read from a source and OutputStream is used for writing data to a destination. Streams represent the source and destination.

BYTE STREAMS:

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, FileInputStream and FileOutputStream. Following is an example which makes use of these two classes to copy an input file into an output file.

Example:

```
import java.io.*;
public class CopyFile {
  public static void main(String args[]) throws IOException {
    FileInputStream in = null;
    FileOutputStream out = null;
      in = new FileInputStream("input.txt");
      out = new FileOutputStream("output.txt");
      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
         System.out.print((char)c);
      }
        in.close();
        out.close();
  }
}
```

CHARACTER STREAMS:

Java Byte streams are used to perform input and output of 8-bit bytes, whereas Java Character streams are used to perform input and output for 16-bit Unicode. Though there are many classes related to character streams but the most frequently used classes are, FileReader and FileWriter. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can re-write the above example, which makes the use of these two classes to copy an input file (having unicode characters) into an output file:

Example:

```
import java.io.*;
public class CopyFile {
  public static void main(String args[]) throws IOException {
    FileReader in = null;
    FileWriter out = null;
      in = new FileReader("input.txt");
      out = new FileWriter("output.txt");
      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
         System.out.print((char)c);
      }
        in.close();
        out.close();
  }
}
```

JAVA BUFFERED OUTPUT STREAM CLASS:

Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast. For adding the buffer in an OutputStream, use the BufferedOutputStream class.

Syntax:

OutputStream os= **new** BufferedOutputStream(**new** FileOutputStream("test.txt"));

Declaration:

public **class** BufferedOutputStream **extends** FilterOutputStream

Class Constructors:

Constructor	Description
BufferedOutputStream(OutputStream os)	It creates the new buffered output stream which is used for writing the data to the specified output stream.
BufferedOutputStream(OutputStream os, int size)	It creates the new buffered output stream which is used for writing the data to the specified output stream with a specified buffer size.

Class Methods:

Method	Description
void write(int b)	It writes the specified byte to the buffered output stream.
void write(byte[ ] b, int off, int len)	It write the bytes from the specified byte-input stream into a specified byte array, starting with the given offset
void flush()	It flushes the buffered output stream.

Example:

```
import java.io.*;
public class BufferedOutputStreamExample{
public static void main(String args[])throws IOException{
   FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
   BufferedOutputStream bout=new BufferedOutputStream(fout);
   String s="Welcome to javaTpoint.";
   byte b[ ]=s.getBytes();
   bout.write(b);
   bout.flush();
   bout.close();
   fout.close();
   System.out.println("success");
}
}
```

JAVA BUFFERED INPUT STREAM CLASS:

Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

The important points about BufferedInputStream are:

- When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
- When a BufferedInputStream is created, an internal buffer array is created.

Syntax:

InputStream in= **new** BufferedIutputStream(**new** FileInputStream("test.txt"));

Class Declaration:

public class BufferedInputStream extends FilterInputStream

Class Constructors:

Constructor	Description
BufferedInputStream(InputStream IS)	It creates the BufferedInputStream and saves it argument, the input stream IS, for later use.
BufferedInputStream(InputStream IS, int size)	It creates the BufferedInputStream with a specified buffer size and saves it argument, the input stream IS, for later use.

Class Methods:

Method	Description
int available()	It returns an estimate number of bytes that can be read from the input stream without blocking by the next invocation method for the input stream.
int read()	It read the next byte of data from the input stream.
int read(byte[] b, int off, int ln)	It read the bytes from the specified byte-input stream into a specified byte array, starting with the given offset.
void close()	It closes the input stream and releases any of the system resources associated with the stream.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readlimit)	It sees the general contract of the mark method for the input stream.
long skip(long x)	It skips over and discards x bytes of data from the input stream.
boolean markSupported()	It tests for the input stream to support the mark and reset methods.

Example:

```
import java.io.*;
public class BufferedInputStreamExample{
 public static void main(String args[]){
  try{
    FileInputStream fin=new FileInputStream("D:\\testout.txt");
    BufferedInputStream bin=new BufferedInputStream(fin);
    int i;
    while((i=bin.read())!=-1){
     System.out.print((char)i);
    }
    bin.close();
    fin.close();
  }catch(Exception e){System.out.println(e);}
 }
}
```

JAVA BUFFERED READER CLASS:

Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.

Class Declaration:

public class BufferedReader **extends** Reader

Class Constructors:

Constructor	Description
BufferedReader(Reader rd)	It is used to create a buffered character input stream that uses the default size for an input buffer.
BufferedReader(Reader rd, int size)	It is used to create a buffered character input stream that uses the specified size for an input buffer.

Class Methods:

Method	Description
int read()	It is used for reading a single character.
int read(char[] cbuf, int off, int len)	It is used for reading characters into a portion of an array.
boolean markSupported()	It is used to test the input stream support for the mark and reset method.
String readLine()	It is used for reading a line of text.
boolean ready()	It is used to test whether the input stream is ready to be read.
long skip(long n)	It is used for skipping the characters.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readAheadLimit)	It is used for marking the present position in a stream.
void close()	It closes the input stream and releases any of the system resources associated with the stream.

Example 1:

```
import java.io.*;
public class BufferedReaderExample {
  public static void main(String args[])throws Exception{
     FileReader fr=new FileReader("D:\\testout.txt");
     BufferedReader br=new BufferedReader(fr);
     int i;
     while((i=br.read())!=-1){
     System.out.print((char)i);
     }
     br.close();
     fr.close();
  }
}
```

Example 2:

```
import java.io.*;
public class BufferedReaderExample{
public static void main(String args[])throws Exception{
   InputStreamReader r=new InputStreamReader(System.in);
   BufferedReader br=new BufferedReader(r);
   System.out.println("Enter your name");
   String name=br.readLine();
   System.out.println("Welcome "+name);
}
}
```

JAVA BUFFERED WRITER CLASS:

Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.

Class Declaration:

public class BufferedWriter **extends** Writer

Class constructors:

Constructor	Description
BufferedWriter(Writer wrt)	It is used to create a buffered character output stream that uses the default size for an output buffer.
BufferedWriter(Writer wrt, int size)	It is used to create a buffered character output stream that uses the specified size for an output buffer.

Class methods:

Method	Description
void newLine()	It is used to add a new line by writing a line separator.
void write(int c)	It is used to write a single character.

void write(char[] cbuf, int off, int len)	It is used to write a portion of an array of characters.
void write(String s, int off, int len)	It is used to write a portion of a string.
void flush()	It is used to flushes the input stream.
void close()	It is used to closes the input stream

Example:

```
import java.io.*;
public class BufferedWriterExample {
public static void main(String[] args) throws Exception {
   FileWriter writer = new FileWriter("D:\\testout.txt");
   BufferedWriter buffer = new BufferedWriter(writer);
   buffer.write("Welcome to javaTpoint.");
   buffer.close();
   System.out.println("Success");
   }
}
```

JAVA FILE READER CLASS:

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class. It is character-oriented class which is used for file handling in java.

Class Declaration:

public class FileReader **extends** InputStreamReader

Class Constructor:

Constructor	Description
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Class Method:

Method	Description
int read()	It is used to return a character in ASCII form. It returns -1 at the end of file.
void close()	It is used to close the FileReader class.

Example:

```
import java.io.FileReader;
public class FileReaderExample {
    public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        int i;
        while((i=fr.read())!=-1)
        System.out.print((char)i);
        fr.close();
    }
}
```

JAVA PRINT WRITER CLASS:

Java PrintWriter class is the implementation of Writer class. It is used to print the formatted representation of objects to the text-output stream.

Class Declaration:

public class PrintWriter **extends** Writer

Methods of PrintWriter class:

Method	Description
void println(boolean x)	It is used to print the boolean value.
void println(char[] x)	It is used to print an array of characters.
void println(int x)	It is used to print an integer.
PrintWriter append(char c)	It is used to append the specified character to the writer.
PrintWriter append(CharSequence ch)	It is used to append the specified character sequence to the writer.
PrintWriter append(CharSequence ch, int start, int end)	It is used to append a subsequence of specified character to the writer.
boolean checkError()	It is used to flushes the stream and check its error state.
protected void setError()	It is used to indicate that an error occurs.
protected void clearError()	It is used to clear the error state of a stream.
PrintWriter format(String format, Object... args)	It is used to write a formatted string to the writer using specified arguments and format string.
void print(Object obj)	It is used to print an object.
void flush()	It is used to flushes the stream.
void close()	It is used to close the stream.

Example:

```
import java.io.File;
import java.io.PrintWriter;
public class PrintWriterExample {
    public static void main(String[] args) throws Exception {
        //Data to write on Console using PrintWriter
     PrintWriter writer = new PrintWriter(System.out);
     writer.write("Javatpoint provides tutorials of all technology.");
 writer.flush();
     writer.close();
//Data to write in File using PrintWriter
     PrintWriter writer1 =null;
       writer1 = new PrintWriter(new File("D:\\testout.txt"));
      writer1.write("Like Java, Spring, Hibernate, Android, PHP etc.");
                 writer1.flush();
       writer1.close();
    }
}
```

FILE SEQUENTIAL/RANDOM:

Sequential file access allows data to be read from a file or written to a file from beginning to end. It is not possible to read data starting in the middle of the file, nor is it possible to write data to the file starting in the middle using sequential methods. The input and output streams in Java are sequential.

Random file access allows the programmer to read or write a data at a random position in the file. The programmer specifies a position in the file with a special SEEK operation and then read or write a data at that position. Thus, a random access file has the logical behavior of array and support much faster access then sequential access file.

The main point is that the programmer must be able to specify the position of a data in terms of a numbers of bytes between it and the beginning of the file. To do this the programmer must also know the type of each data and the number of bytes required to store it.

The random file access supports random access file processing. The most convenient constructor expect a file name and a mode as a parameter. The mode 'rw' and 'r' specify read/write and read only access prospectively. We use 'rw' for output files and 'r' for input files.

Example:

```
import java.io.*;
public class RandomAccessDemo{
    public static void main(String [] args) throws IOException{
            char a[] = {'a','e','i','o','u'};
            RandomAccessFile rand = new RandomAccessFile("rand.txt","rw");
            System.out.println("Writing vowel letters to file");
            for(int i=0;i<a.length;i++){
                    rand.writeChar(a[i]);
            }
            System.out.println("Finished Writing");
            rand.close();
    }
}
```