JAVA I/O - Chap 7 & 8

Java I/O Tutorial

Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

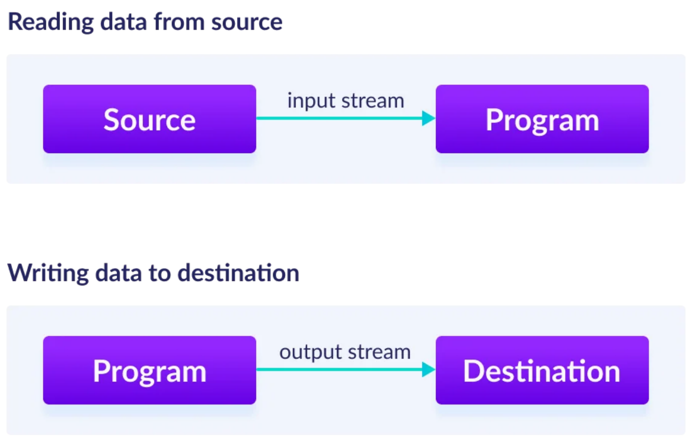
We can perform file handling in Java by Java I/O API.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream



Types of Streams

Depending upon the data a stream holds, it can be classified into:

- Byte Stream
- Character Stream

ByteStream Classes in Java

ByteStream classes are used to read bytes from the input stream and write bytes to the output stream. In other words, we can say that ByteStream classes read/write the data of 8-bits. We can store video, audio, characters, etc., by using ByteStream classes. These classes are part of the java.io package.

The ByteStream classes are divided into two types of classes, i.e., InputStream and OutputStream. These classes are abstract and the super classes of all the Input/Output stream classes.

InputStream	OutputStream
FIleInputStream	FileOutputStream
ByteArrayInputStream	ByteArrayOutputStream
ObjectInputStream	ObjectOutputStream
PipedInputStream	PipedOutputStream
FilteredInputStream	FilteredOutputStream

BufferedInputStream	BufferedOutputStream
DataInputStream	DataOutputStream

CharacterStream Classes in Java

The java.io package provides CharacterStream classes to overcome the limitations of ByteStream classes, which can only handle the 8-bit bytes and is not compatible to work directly with the Unicode characters. CharacterStream classes are used to work with 16-bit Unicode characters. They can perform operations on characters, char arrays and Strings.

However, the CharacterStream classes are mainly used to read characters from the source and write them to the destination. For this purpose, the CharacterStream classes are divided into two types of classes, I.e., Reader class and Writer class.

Reader	Writer
BufferedReader	BufferedWriter
CharacterArrayReader	CharacterArrayWriter
StringReader	StringWriter
FileReader	FileWriter
InputStreamReader	InputStreamWriter
FileReader	FileWriter

Which classes we need to Learn for our chapter 7 & 8

- File Class
- FileInputStream class : Parent class is InputStream class : Used to Read from File : Byte By Byte
- FileOutputStream class : Parent class is OutputStream class : Used to Write to File : Byte By Byte
- FileReader : Parent class is Reader class : Used to Read from File : Character By Character
- FileWriter : Parent class is Writer class : Used to Write to File : Character By Character
- BufferedReader : Parent class is Reader : Used to Read from Various sources like console or file with buffer facility
- InputStreamReader : Parent class is Reader : Used to Read from CONSOLE with the help of BufferedReader

  NOTE: kindly import java.io.*;

1. File Class

   The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

   The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

Constructors

Constructor	Description
File(File parent, String child)	It creates a new File instance from a parent abstract pathname and a child pathname string.
File(String pathname)	It creates a new File instance by converting the given pathname string into an abstract pathname.

File(String parent, String child)	It creates a new File instance from a parent pathname string and a child pathname string.
File(URI uri)	It creates a new File instance by converting the given file: URI into an abstract pathname.

Methods of File Class

boolean	createNewFile()	It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
boolean	canWrite()	It tests whether the application can modify the file denoted by this abstract pathname.String[]
boolean	canExecute()	It tests whether the application can execute the file denoted by this abstract pathname.
boolean	canRead()	It tests whether the application can read the file denoted by this abstract pathname.
boolean	isDirectory()	It tests whether the file denoted by this abstract pathname is a directory.
boolean	isFile()	It tests whether the file denoted by this abstract pathname is a normal file.

String	getName()	It returns the name of the file or directory denoted by this abstract pathname.
String	getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
Path	getPath()	It returns a java.nio.file.Path object constructed from the this abstract path.
File[]	listFiles()	It returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname
String[]	list(Filename)	It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
boolean	mkdir()	It creates the directory named by this abstract pathname.
exists()	boolean	Checks weather the file exists or not
length()	int	Returns the length of file
equals()	Boolean	Compare both files objects

Example:

```
 import java.io.*;
class FileEx
{
   public static void main(String[] args) throws IOException
   {
      File f = new File("fileProg1.txt");
      File f1 = new File("JAVAPAT/filed.txt   ");
      File f3 = new File("JAVAPAT/filed.txt   ");
      if(f1.createNewFile()) // createNewFile() is boolean method
      {
         System.out.println("f1 New file created");
      }
      else
      {
         System.out.println("f1 File already exist");
      }
      if(f.createNewFile())
      {
         System.out.println("f New file created");
      }
      else
      {
         System.out.println("f File already exist");
      }
            System.out.println("File path="+f1.getPath());
            System.out.println("Absolute path="+f1.getAbsolutePath());
            System.out.println("Parent="+f1.getParent());
            System.out.println("Path="+f1.getPath());
            System.out.println("Name="+f1.getName());
            System.out.println("File path="+f.getPath());
            System.out.println("Absolute path="+f.getAbsolutePath());
            System.out.println("Parent="+f.getParent());
            System.out.println("Path="+f.getPath());
            System.out.println("Name="+f.getName());

            System.out.println("is file ?"+f.isFile());
            System.out.println("is Directory ?"+f.isDirectory());
            File f2 = new File("D://PAT");
            f2.mkdir();
            System.out.println("is Directory ?"+f2.isDirectory());


            System.out.println(f1.canRead());
            System.out.println(f1.canWrite());
            System.out.println(f1.exists());
            System.out.println(f1.canExecute());

            System.out.println(f1.equals(f3));
            System.out.println(f1.length());

            File f9 = new File("C://Users/Abc/Desktop/JAVA 2");
            String fileNames[] = f9.list();

            for(String s : fileNames)
            {
               System.out.println("Name =" + s);
            }
            File all[] = f9.listFiles();
            for(File o : all)
            {
               System.out.println("NAme= "+o.getName());
               System.out.println("Absolute Path="+ o.getPath());
            }
   }
}
```

Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a file.

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

void write(byte[] ary)	It is used to write ary.length bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write len bytes from the byte array starting at offset off to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.

Example 1:

```
import java.io.FileOutputStream;

public class FileOutputStreamExample {

   public static void main(String args[]){

        try{

          FileOutputStream fos =new FileOutputStream("D:\\testout.txt");

          fos.write(65);

          fos.close();

          System.out.println("success...");

         }catch(Exception e){System.out.println(e);}

     }

}
```

Example 2:

```
import java.io.FileOutputStream;

public class FileOutputStreamExample {

   public static void main(String args[]){

        try{

          FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
        String s="Welcome to javaTpoint.";

        byte b[]=s.getBytes();//converting string into byte array

        fout.write(b);

        fout.close();

        System.out.println("success...");

       }catch(Exception e){System.out.println(e);}

   }

}
```

Example 3:

```
import java.io.FileOutputStream;
import java.io.IOException;
public class FileOutputStream1
{
   public static void main(String[] args) throws IOException
   {
      FileOutputStream fos = new FileOutputStream("D://PAT/t2.txt");
      fos.write(15);
      // fos.close();
      String s = "Parth";
      byte[]  b= s.getBytes();
      fos.write(b);
      fos.close();

   }
}
```

Java FileInputStream Class

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class public class FileInputStream extends InputStream

int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to b.length bytes of data from the input stream.
void close()	It is used to closes the stream.

Example 1:

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fis=new FileInputStream("D:\\testout.txt");
          int i=fis.read();    // read a single character from file
          System.out.print((char)i);
          fis.close();
        }catch(Exception e){System.out.println(e);}
       }
      }
```

Example 2: How to read all characters from the file

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=0;
          while((i=fin.read())!=-1){
           System.out.print((char)i);
          }
          fin.close();
        }catch(Exception e){System.out.println(e);}
       }
      }
```

Java FileWriter Class

Java FileWriter class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.

Unlike FileOutputStream class, you don't need to convert string into byte array because it provides method to write string directly.

Constructor

FileWriter(String file)	Creates a new file. It gets file name in string.
FileWriter(File file)	Creates a new file. It gets file name in File object.

Methods

void write(String text)	It is used to write the string into FileWriter.
void write(char c)	It is used to write the char into FileWriter.
void write(char[] c)	It is used to write char array into FileWriter.

Example 1:

```
import java.io.FileWriter;
public class FileWriterExample {
   public static void main(String args[]){
        try{
          FileWriter fw=new FileWriter("D:\\testout.txt");
          fw.write("Welcome to javaTpoint.");
          fw.close();
         }catch(Exception e){System.out.println(e);}
         System.out.println("Success...");
    }
}
```

Java FileReader 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.

Constructors:

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.

Methods

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 1:

```
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();
    }
}
```

Example 2: [FileWriter & FileReader Examples]

```
import java.io.FileReader;
import java.io.FileWriter;
public class FileWriterReaderExample
{
    public static void main(String[] args) throws Exception
    {
        FileWriter fw1 = new FileWriter("D://PAT/abc.txt"); // open file in writing
mode
        fw1.write("Hello Pat");  // write string
        fw1.write(3);    // write integer to fiel
        fw1.write("xyxyxyx");    // write whole string
        fw1.write(new char[] {'a','v','d'});  // write character array
        fw1.close();

        FileReader fr1 = new FileReader("D://PAT/abc.txt") ; //open file in reading
        int i = fr1.read();   // read only int value or 1st caracter
        while(i != -1)
        {
            System.out.print((char)i);
            i = fr1.read();
        }
        fr1.close();
    }
}
```

Java BufferedReader 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.

Constructor:

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

Methods

int read()	It is used for reading a single character.

String readLine()	It is used for reading a line of text.

Example 1: Read from file using BufferedReader

```
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.BufferedReader;
import java.io.FileReader;
public class Buffer1 {
   public static void main(String[] args) throws Exception
   {
      FileReader fr = new FileReader("D://PAT/dpt.txt");
      BufferedReader br = new BufferedReader(fr);
      String s = br.readLine();
      while(s != null)
      {
         System.out.println(s);
         s = br.readLine();
      }
   }
}
```

Class - Java InputStreamReader // use to read from console - we can use this instead of scanner class. But need to use BufferedReader with it to readLine from console.

An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Constructors

InputStreamReader(InputStream in)	It creates an InputStreamReader that uses the default charset.

Methods:

int	read()	It reads a single character.

```
//console
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
public class InputstreamreaderEx {
   public static void main(String[] args) throws IOException
   {
      InputStreamReader isr1 = new InputStreamReader(System.in);
      BufferedReader br1 = new BufferedReader(isr1);
      System.out.println("Enter your views");
      String view = br1.readLine();
      System.out.println(view);

   }
}
```

FINAL ALL PROGRAMS OF CHAP 7 & 8

```
/*
 *
 *Program 1 :  Replace word1 by word2 from file 1 and write to file2.
 *
*/
import java.io.*;
import java.util.*;


public class FileReplace {
   public static void main(String[] args)  throws IOException
   {
      File f1 = new File("jjs.txt");  // read from this and replace
      File f2 = new File("dju.txt"); // write to this after replace

      FileWriter fw1 = new FileWriter(f1);
      fw1.write("Hi baby Hi Hi its too High");
      fw1.close();

      Scanner sc = new Scanner(System.in);
      System.out.println("Enter Word 1");
      String word1 = sc.next();
      System.out.println("Enter Word 2");
      String word2 = sc.next();

      FileWriter fw2 = new FileWriter(f2);

      FileReader fr1 = new FileReader(f1);
      BufferedReader br1 = new BufferedReader(fr1); //

      String rep = "";
      String s="";
```

```
        while((s=br1.readLine())!=null)
        {
            String s1[] = s.split(" ");
            for(String s2 : s1){
                if(s2.equals(word1))
                {
                    s2=word2;
                }
                rep = rep+" "+s2;
            }

            fw2.write(rep+"\n");

        }
        fw2.close();


    }
}
```

Java - RandomAccessFile

This class is used for reading and writing to random access file. A random access file behaves like a large array of bytes. There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Constructor

RandomAccessFile(File file, String mode)	Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile(String name, String mode)	Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Methods

int	readInt()	It reads a signed 32-bit integer from this file.
String	readUTF()	It reads in a string from this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void	writeDouble(double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then

		writes that long value to the file as an eight-byte quantity, high byte first.
void	writeFloat(float v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
void	write(int b)	It writes the specified byte to this file.

Suppose I have file with name - jjs.txt
Content of jjs file is :
Hey Programmers....!!!
Welcome to files.

Example 1

```
import java.io.IOException;
import java.io.RandomAccessFile;
public class myFile {

 public static void main(String[] args) throws IOException {
  // Object of RandomAccessFile
  RandomAccessFile raf = new RandomAccessFile(“jjs.txt", "r");

  // moving file pointer
  raf.seek(4);

  byte[] bytes = new byte[12];

  // storing file data
  raf.read(bytes);
  raf.close();

  // printing file data
  System.out.println(new String(bytes));
 }
}
```

Output

```
Programmers....!!!
```

Example 2:

```
// Java Program illustrating use of io.RandomAccessFile class methods
// read(), read(byte[] b), readBoolean(), readByte(), readInt()
// readFully(byte[] b, int off, int len), readFully(), readFloat()
// readChar(), readDouble(),

import java.io.*;
public class NewClass
{
        public static void main(String[] args)
        {
                try
                {
                        double d = 1.5;
                        float f = 14.56f;

                        // Creating a new RandomAccessFile - "GEEK"
                        RandomAccessFile        geek        =        new
RandomAccessFile("GEEK.txt", "rw");

                        // Writing to file
                        geek.writeUTF("Hello Geeks For Geeks");

                        // File Pointer at index position - 0
                        geek.seek(0);

                        // read() method :
                        System.out.println("Use of read() method : " + geek.read());

                        geek.seek(0);
```

```
                    byte[] b = {1, 2, 3};

                    // Use of .read(byte[] b) method :
                    System.out.println("Use of .read(byte[] b) : " + geek.read(b));

                    // readBoolean() method :
                    System.out.println("Use of readBoolean() : " +
geek.readBoolean());

                    // readByte() method :
                    System.out.println("Use of readByte() : " + geek.readByte());

                    geek.writeChar('c');
                    geek.seek(0);

                    // readChar() :
                    System.out.println("Use of readChar() : " + geek.readChar());

                    geek.seek(0);
                    geek.writeDouble(d);
                    geek.seek(0);

                    // read double
                    System.out.println("Use of readDouble() : " +
geek.readDouble());

                    geek.seek(0);
                    geek.writeFloat(f);
                    geek.seek(0);

                    // readFloat() :
                    System.out.println("Use of readFloat() : " +
geek.readFloat());

                    geek.seek(0);
                    // Create array upto geek.length
                    byte[] arr = new byte[(int) geek.length()];
                    // readFully() :
```

```
                geek.readFully(arr);

                String str1 = new String(arr);
                System.out.println("Use of readFully() : " + str1);

                geek.seek(0);

                // readFully(byte[] b, int off, int len) :
                geek.readFully(arr, 0, 8);

                String str2 = new String(arr);
                System.out.println("Use of readFully(byte[] b, int off, int len)
: " + str2);
            }
            catch (IOException ex)
            {
                System.out.println("Something went Wrong");
                ex.printStackTrace();
            }
    }
}
```

Output

```
Use of read() method : 0
Use of .read(byte[] b) : 3
Use of readBoolean() : true
Use of readByte() : 108
Use of readChar() : c
Use of readDouble() : 1.5
Use of readFloat() : 14.56
Use of readFully() : Geeks For Geeks
Use of readFully(byte[] b, int off, int len) : Geeks For Geeks
```