The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, object, localized characters, etc.

Stream

A stream can be defined as a sequence of data. There are two kinds of Streams −

InPutStream − The InputStream is used to read data from a source.

OutPutStream − The OutputStream is used for writing data to a destination.

Java provides strong but flexible support for I/O related to files and networks but this tutorial covers very basic functionality related to streams and I/O. We will see the most commonly used examples one by one −

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;

    try {

      in = new FileInputStream("input.txt");
```

```
        out = new FileOutputStream("output.txt");
        int c;
        while ((c = in.read()) != -1) {
           out.write(c);
        }
      }finally {
        if (in != null) {
           in.close();
        }
        if (out != null) {
           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

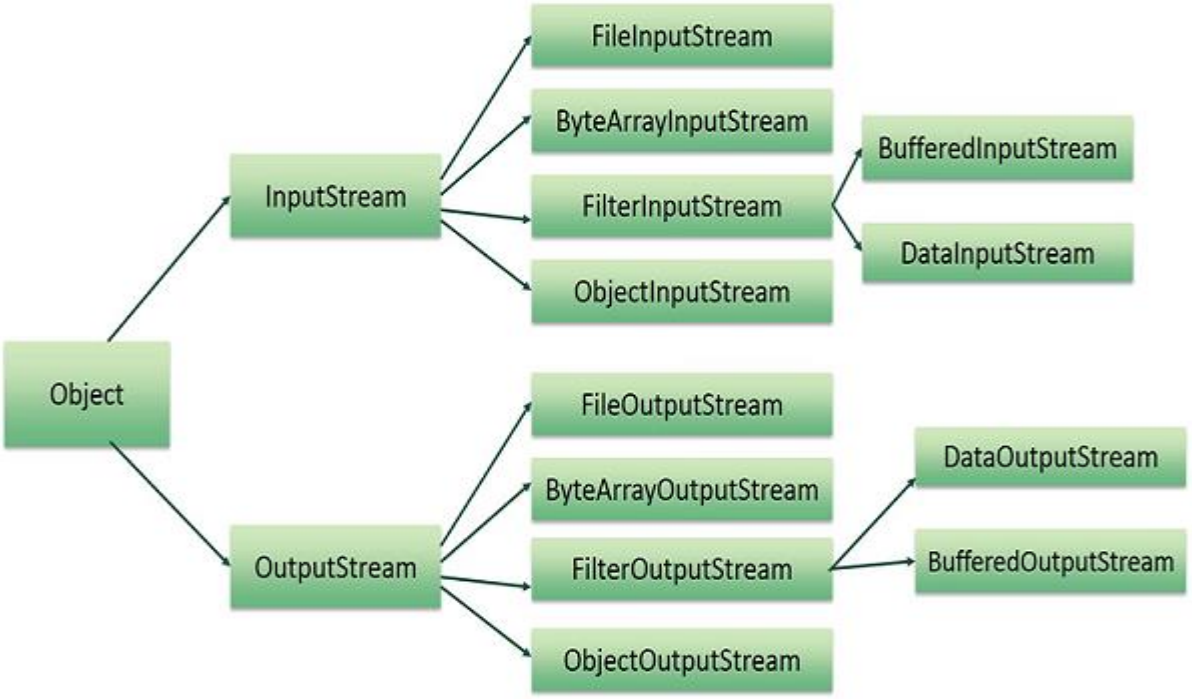
```
import java.io.*;
```

```
public class CopyFile {

  public static void main(String args[]) throws IOException {

    FileReader in = null;

    FileWriter out = null;

    try {

      in = new FileReader("input.txt");

      out = new FileWriter("output.txt");

      int c;

      while ((c = in.read()) != -1) {

        out.write(c);

      }

    }finally {

      if (in != null) {

        in.close();

      }

      if (out != null) {

        out.close();

      }

    }

  }

}
```

Reading and Writing Files

As described earlier, a stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

Here is a hierarchy of classes to deal with Input and Output streams.



The two important streams are FileInputStream and FileOutputStream, which would be discussed in this tutorial.

FileInputStream

This stream is used for reading data from the files. Objects can be created using the keyword new and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file −

InputStream f = new FileInputStream("C:/java/hello");

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows −

File f = new File("C:/java/hello");

InputStream f = new FileInputStream(f);

Once you have InputStream object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

Sr.No.	Method & Description

1	public void close() throws IOException{} This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
2	protected void finalize()throws IOException {} This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
3	public int read(int r)throws IOException{} This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's the end of the file.
4	public int read(byte[] r) throws IOException{} This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If it is the end of the file, -1 will be returned.
5	public int available() throws IOException{} Gives the number of bytes that can be read from this file input stream. Returns an int.

FileOutputStream

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Here are two constructors which can be used to create a FileOutputStream object.

Following constructor takes a file name as a string to create an input stream object to write the file −

OutputStream f = new FileOutputStream("C:/java/hello")

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows −

File f = new File("C:/java/hello");

OutputStream f = new FileOutputStream(f);

Once you have OutputStream object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

Sr.No.	Method & Description
1	public void close() throws IOException{} This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
2	protected void finalize()throws IOException {} This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
3	public void write(int w)throws IOException{} This methods writes the specified byte to the output stream.
4	public void write(byte[] w) Writes w.length bytes from the mentioned byte array to the OutputStream.

Example

Following is the example to demonstrate InputStream and OutputStream −

import java.io.*;

```
public class fileStreamTest {

  public static void main(String args[]) {

    try {

      byte bWrite [] = {11,21,3,40,5};

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

      for(int x = 0; x < bWrite.length ; x++) {

        os.write( bWrite[x] );   // writes the bytes

      }

      os.close();

      InputStream is = new FileInputStream("test.txt");

      int size = is.available();

      for(int i = 0; i < size; i++) {

        System.out.print((char)is.read() + "  ");

      }

      is.close();

    } catch (IOException e) {

      System.out.print("Exception");

    }

  }

}
```