**JAVA I/O:**

Data can be stored on a computer system, as per the code requirements, in two ways, either **permanently**or **temporarily**.

Temporary storage can be accomplished by storing the data in datastructures or instance variables.

The data is temporary because it is stored in RAM. For a permanent storage, the data should be stored on the hard disk either in the form of database tables or files.

There comes two entities – a **source** and a **destination**. Source is that from where data is read and the destination is that one to where data is written.

The source and destination need not be a file only; it can be a socket or keyboard input etc.

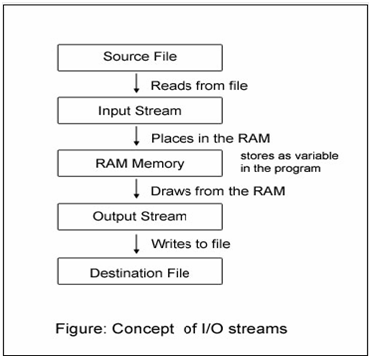
To do the job of reading and writing, there comes two types of streams – **input streams** and **output streams**.

An input stream job is to read from the source and the output stream job is to write to the destination.

That is, in the program, it is necessary to link the input stream object to the soruce and the output stream object to the destination.

I/O streams are carriers of data from one place to another. The input stream carries data from the source and places it temporarily in a variable (like int k or String str etc.) in the process (program).

The output stream takes the data from the variable and writes to the destination. The variable works like a temporary buffer between input stream and output stream.

[](https://way2java.com/wp-content/uploads/2011/04/ss7.bmp)

It is clear from the above figure, the input stream reads from the soruce and puts in the buffer (actually in programming, in a temporary variable, shown later). The output stream takes from the memory and writes to the destination.

All the classes needed to do with reading and writing (like file copying) are placed in the package **java.io** by the designers.

All the I/O streams do the file reading or writing **sequentially** (means, one byte after another from start to the end of file).

It can be done at **random** also. For this another class exists –**RandomAccessFile**.

The **java.io** package also includes another class, **File,**to know the properties of a file like the file has read permission or write permission etc.

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.



### 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**. **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.

**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();

}

}

}

}

## Standard Streams

All the programming languages provide support for standard I/O where the user's program can take input from a keyboard and then produce an output on the computer screen.

* **Standard Input** − This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as **System.in**.
* **Standard Output** − This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as **System.out**.
* **Standard Error** − This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as **System.err**.

Following is a simple program, which creates **InputStreamReader** to read standard input stream until the user types a "q" −

**Example**

import java.io.\*;

public class ReadConsole {

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

InputStreamReader cin = null;

try {

cin = new InputStreamReader(System.in);

System.out.println("Enter characters, 'q' to quit.");

char c;

do {

c = (char) cin.read();

System.out.print(c);

} while(c != 'q');

}finally {

if (cin != null) {

cin.close();

}

}

}

}

Enter characters, 'q' to quit.

1

1

e

e

q

q

## 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.**

### 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. |

There are other important input streams available, for more detail you can refer to the following links −

* [ByteArrayInputStream](https://www.tutorialspoint.com/java/java_bytearrayinputstream.htm)
* [DataInputStream](https://www.tutorialspoint.com/java/java_datainputstream.htm)

## 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);

|  |  |
| --- | --- |
| **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. |

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.

There are other important output streams available, for more detail you can refer to the following links −

* [ByteArrayOutputStream](https://www.tutorialspoint.com/java/java_bytearrayoutputstream.htm)
* [DataOutputStream](https://www.tutorialspoint.com/java/java_dataoutputstream.htm)

**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");

}

}

}

The above code would create file test.txt and would write given numbers in binary format. Same would be the output on the stdout screen.