

File handling in Java using byte streams involves reading and writing data as sequences of bytes. This approach is suitable for handling binary data, such as images, audio files, and other non-textual content.

Key Classes for Byte Stream File Handling:

* FileInputStream:

Used for reading data from a file as a stream of bytes. It provides methods like read() to fetch byte data.

* FileOutputStream:

Used for writing data to a file as a stream of bytes. It provides methods like write() to send byte data.

* File:

Represents a file or directory path, used for creating, deleting, and retrieving file information.

Reading from a File using Byte Streams:

* Create a FileInputStream object, passing the File object or file path as an argument.
* Use the read() method to read bytes from the file. This method returns the next byte or -1 if the end of the file is reached.
* Handle the data as needed.
* Close the stream using the close() method in a finally block to release resources.

Writing to a File using Byte Streams:

* Create a FileOutputStream object, passing the File object or file path as an argument.
* Use the write() method to write bytes to the file.
* Close the stream using the close() method in a finally block to release resources.

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class ByteStreamFileHandling {

public static void main(String[] args) {

String inputFile = "input.txt";

String outputFile = "output.txt";

try {

// Write to a file

FileOutputStream fos = new FileOutputStream(outputFile);

String data = "Hello, World!";

fos.write(data.getBytes());

fos.close();

System.out.println("Data written to " + outputFile);

// Read from a file

FileInputStream fis = new FileInputStream(inputFile);

int i;

while ((i = fis.read()) != -1) {

System.out.print((char) i);

}

fis.close();

} catch (IOException e) {

e.printStackTrace();

}

}

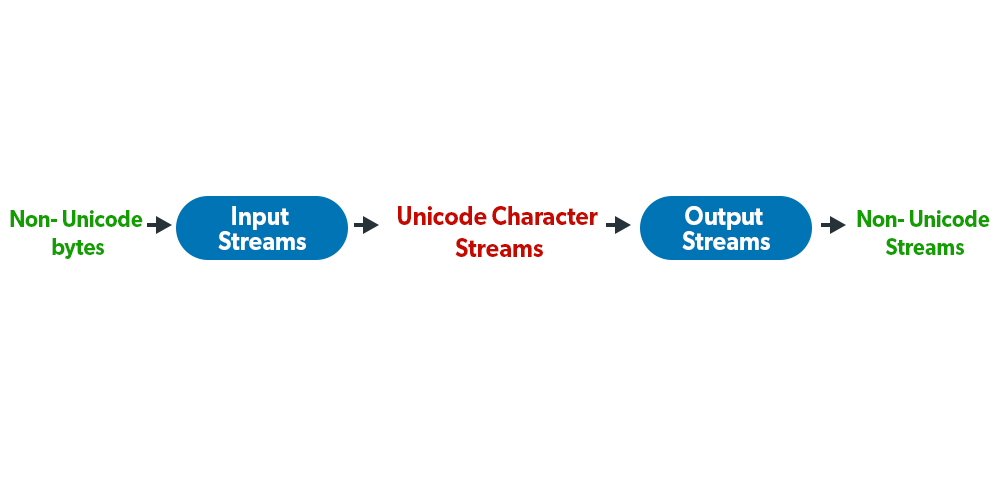
}

Important Considerations:

* **Resource Management**: Always close streams in a finally block to avoid resource leaks.
* **Error Handling**: Handle IOException to manage input/output errors properly.
* **Buffering**: For better performance, consider using BufferedInputStream and BufferedOutputStream to buffer input and output operations.
* **Character Streams**: For text files, character streams like FileReader and FileWriter are more suitable.
* **File Operations**: Use the File class to manage files, create directories, and check file attributes.

Character Stream

 Character stream automatically allows us to read/write data character by character. For example, FileReader and FileWriter are character streams used to read from the source and write to the destination.

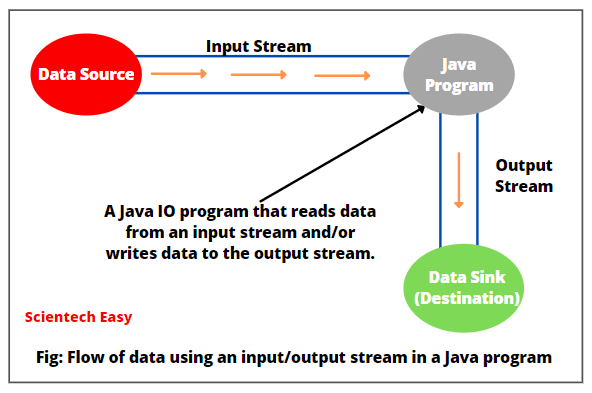


**Stream in Java** represents sequential flow (or unbroken flow) of data from one place to another place. In other words, a stream is a path along which data flows (like a pipe along which water flows).

It is required to accept data as input from the keyboard. The data in the form of stream may be bytes, characters, objects, etc. Let’s understand it with the help of a realtime example.

A river is a stream of water that flows from one place (i.e. source) to another place (i.e. destination) in a continuous sequence. Source and destination are connected through the continuous flow of water.

Similarly, a stream carries data from one place (source) to another place (destination). In Java I/O, data flows from a source known as data source to a destination known as data sink as shown in the below figure. Data source and data sink are connected through a Java program.

[](https://www.scientecheasy.com/2021/05/stream-in-java.html/)

A stream is always required if we want to move data from data source to data sink. For example, a stream can carry data from:

* keyboard to memory
* memory to printer
* memory to a text file

**Types of Stream in Java**

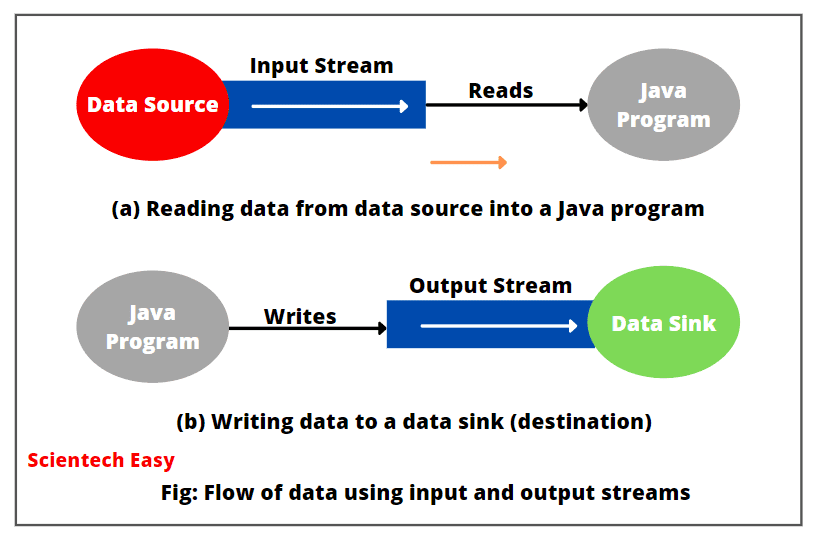
Basically, there are two types of streams in Java. They are as follows:

* Input stream
* Output stream

**Input Stream:**

A stream that receives or reads data from a data source and sends it to a Java program is called input stream. Input represents the flow of data into a program.

The input stream connects the data source and a Java program, as shown in the below figure.

[](https://www.scientecheasy.com/2021/05/stream-in-java.html/)

As you can see in the above figure, the data is read from a data source and flow sequentially to a Java program.

**Output Stream:**

A stream that takes data from a Java program and sends or writes data to the destination (data sink) is called output stream. A output represents the flow of data out of a program.

Output stream connects java program and a data sink. A Java program writes or sends data to the data sink.

The data flow from a data source to a Java program via an input stream. The data flow from a Java program to a data sink via an output stream.

In other words, Java program reads data from an input stream and writes data to the output stream. In both input and output streams, Java program does not know the details of source and destination.

**Note:**

Java represents all streams by classes defined in java.io (input and output) package. This package contains a lot of classes, all of which are divided into two basic categories: input streams and output streams.

**Keyboard as a Field**

Java represents a keyboard by a field, called “in” in System class. System.in represents a standard input device, i.e. keyboard, by default.

System class is found in java.lang (language) package. It contains three fields that represent some type of stream. These fileds are as:

a) **System.in:** This field represents an InputStream object, which is a standard input device, known as a keyboard by default.

b) **System.out:** This field represents a PrintStream object, which represents a standard output device, known as a monitor by default. Normally, we use it to display normal messages.

c) **System.err:** It also represents PrintStream object, which by default represents a monitor. Normally, we use it to display error messages.

Remember that we use both System.out and System.err to represent the monitor. Hence, we can use any of these to send data to a monitor.

**How to Read data from Data source into Java Program?**

To read data from data source into a Java program, we need to perform the following steps that are as follows:

* Identify the data source. It may be a keyboard, file, array, string, network connection, etc.
* Create an input stream by using the data source identified.
* Read data from the input stream. Basically, we read data in a loop as long as we have received all data from input stream.
* Now, close the input stream when reading data has finished. From Java 7 onwards, we can use the try-with-resources block that closes the input stream automatically.

**How to Write Data to Destination from Java Program?**

To write data to a destination (data sink) from the Java program, perform the following steps:

* Identify the destination where data will be written. A destination may be a monitor, a file, array, string, network connection, etc.
* Create an output stream using data sink that we have identified.
* Now, write data to the output stream.
* Close the output stream once writing data is completed. From Java 7 onwards, we can use a try-with-resources block to close the output stream automatically.