Byte Stream in Java - Scaler Topics

scaler.com/topics/java/byte-stream-in-java/

Overview

The stream method helps to **sequentially access** a file. There are two types of streams in Java- Byte Stream and Character Stream. Byte streams in Java are used to perform input and output operations of 8-bit bytes while the Character stream is used to perform input and output operations for 16-bits Unicode.

Character streams are useful in reading or writing text files which are processed character by character. Byte Streams are useful to read/write data from raw binary files.

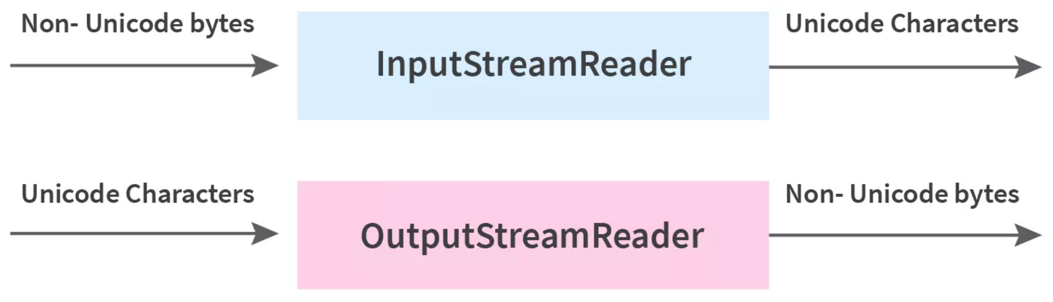
Introduction to Byte Stream and Character Stream

Have you ever wondered, how the various files that we write in Java are accessed and processed or how Java internally manages to handle operations on such files? Suppose, if we want to copy a file from our laptop to a pen drive, how does that happen?

An I/O (Input/Output) stream is used to represent an **input source** or an **output destination**. It can represent many kinds of sources and destinations like disc files (systems that manage data on permanent storage devices eg. hard disc or magnetic disc) or devices.

The stream method helps to **sequentially access** a file. Some streams simply pass on the data while some manipulate and transform the data in a useful way. For example, some streams copy the contents of the file to another and do not modify them or some streams perform manipulations on them like adding or filtering data etc. Streams support many kinds of data including bytes, primitive data types, characters and objects.

The *java.io* package contains classes that allow the user to convert between Unicode character streams and byte streams of non-Unicode text.



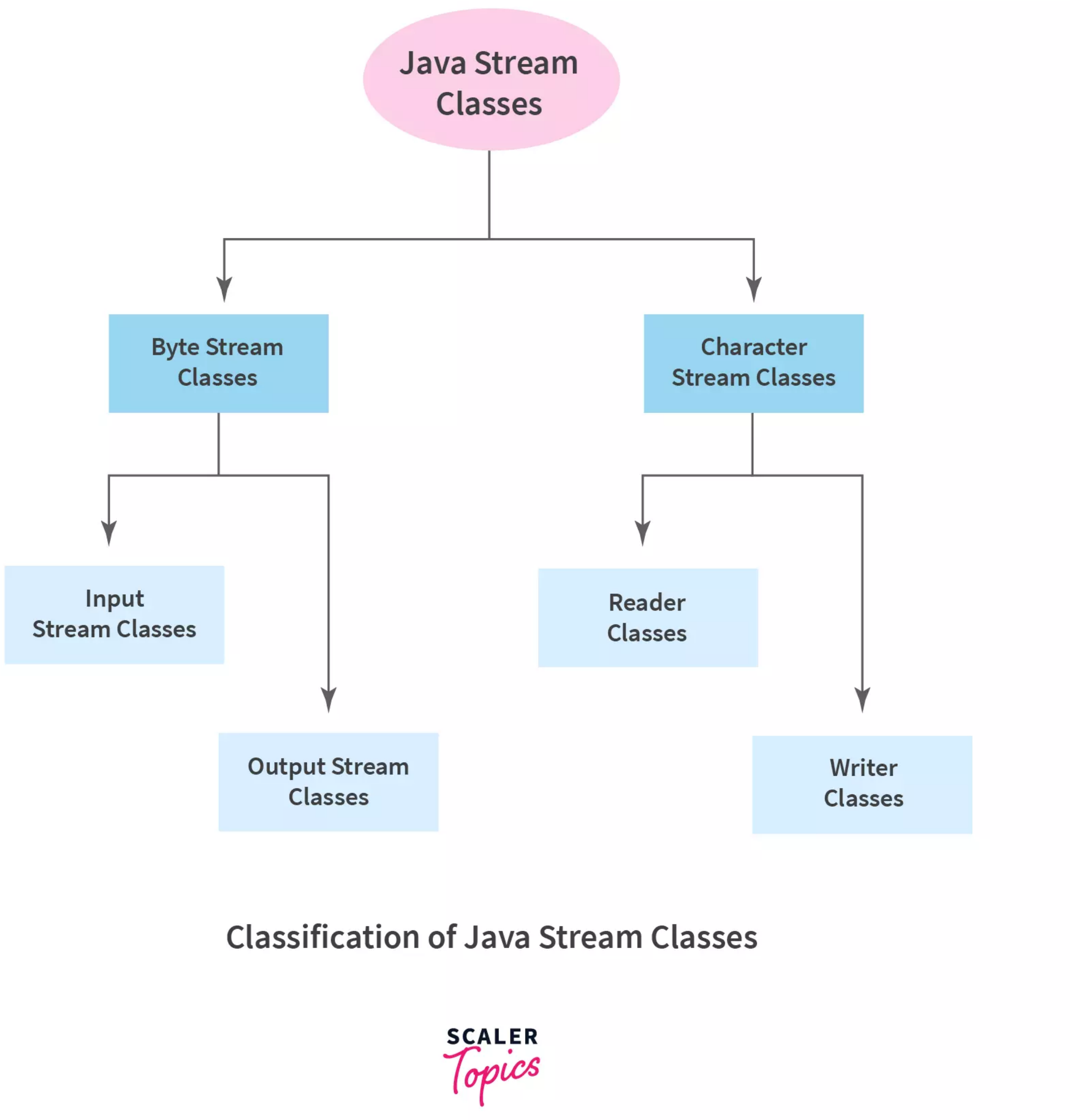
Classes of Java.io which help to convert between
Unicode character streams and bytes streams of non- Unicode text

SCALER Topics

In the above image, we are converting non-Unicode bytes to Unicode characters and vice-versa using InputStreamReader and OutputStreamWriter classes respectively.

Let us understand some terminologies associated with the same.

- **Stream-** It is a sequence of data/objects that supports various methods. They are used to read or write data. Eg- files, input-output devices etc.
- **Input Stream Reader-** It reads data from a source, one item at a time. An InputStreamReader class is a bridge from byte streams to character streams. It reads bytes and decodes them into characters.
- **Output Stream Writer-** It writes data to a destination, one item at a time. An OutputStreamWriter class is a bridge from character streams to byte streams. Characters written to it are encoded into bytes.
- **Unicode** is an international character encoding standard by which each letter, digit or symbol is assigned a unique numeric value across all platforms.
- **Non-Unicode text** are modules or character encodings that do not support Unicode standards. It only supports English language representation.



What is Byte Stream in Java?

Byte streams are used to perform **input** and **output** of **8-bit bytes**. They are used to read bytes from the input stream and write bytes to the output stream. Mostly, they are used to read or write raw binary data.

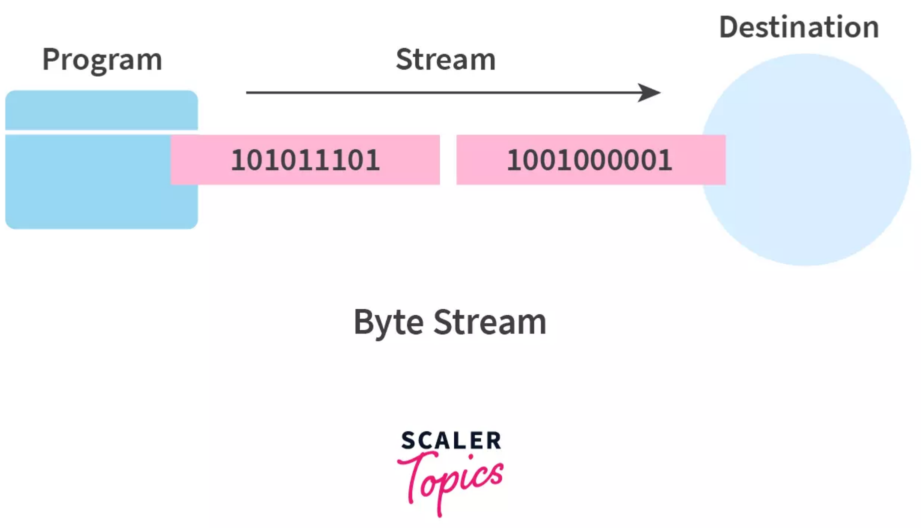
In Java, the byte streams have a 3 phase mechanism:

- **Split-** The input data source is split into a stream by a spliterator. Java Spliterator interface is an internal iterator that breaks the stream into smaller parts for traversing over them.
- **Apply-** The elements in the stream are processed.
- **Combine-** After the elements are processed, they are again combined together to create a single result.

Java provides many byte stream classes, but the most common ones are-

FileInputStream- This class is used to read data from a file/source. The FileInputStream class has constructors which we can use to create an instance of the FileInputStream class.

FileOutputStream- This class is used to write data to the destination. The following is the constructor to create an instance of the FileOutputStream class.



The above image shows an example of the byte stream. From a program, the bytes are being transferred in the form of a stream to the destination.

Example of Byte Stream

Let us take a look at an example to use Byte Stream to copy the contents of one file to another. In the example, we will create two objects of the FileInputStream and the FileOutputStream classes.

The source and the destination files names are given as parameters to the FileInputStream and the FileOutputStream classes respectively. Then the content of the source file will be copied to the destination file.

Explanation- Let us assume that we have already created a file named *source.txt* which has the following content-

After we execute the above program, it will create a file called *destination.txt* in the same directory as the program file which has the **same** content as of *source.txt*. If a file with the same name already exists, an exception of FileAlreadyExistsException is thrown.

What is Character Stream in Java?

In Java, **character** values are stored using **Unicode** conventions. As we saw above, the Byte stream is used to perform input and output operations of 8-bit bytes, but the Character stream is used to perform **input** and **output** operations of **16-bit Unicode**. If we want to copy a text file containing characters from one source to another destination using streams, character streams would be advantageous as it deals with characters. Characters in Java are 2 bytes or 16 bits in size.

In Java, the character streams too have a 3 phase mechanism similar to that of Byte Streams as explained above.

Java provides many character stream classes, but the most common ones are-
FileReader- It is used to read two bytes at a time from the source. The following is the constructor to create an instance of the FileReader class.

FileWriter- It is used to write two bytes at a time to the destination. The following is the constructor to create an instance of the FileWriter class.

Example of Character Stream

This example deals with the usage of Character Stream to copy the contents of one file to another. In the example, we will create two objects of the FileReader and the FileWriter classes. The source and the destination files names are given as parameters to the FileReader and the FileWriter classes respectively. Then the content of the source file will be copied to the destination file.

Explanation - Let us assume that we have already created a file named *source.txt* which has the following content-

After we execute the above program, it will create a file called *destination.txt* which has the **same** content as of *source.txt*. If a file with the same name already exists, an exception of FileAlreadyExistsException is thrown.

Difference Between Byte Stream and Character Stream

Byte Stream	Character Stream
Byte stream is used to perform input and output operations of 8-bit bytes.	Character stream is used to perform input and output operations of 16-bit Unicode.
It processes data byte by byte.	It processes data character by character.
Common classes for Byte stream are FileInputStream and FileOutputStream.	Common classes for Character streams are FileReader and FileWriter.
Example- Byte streams are used to read or write binary data.	Example- Character streams are used to read/write characters.

When to use Character Stream over Byte Stream?

Character streams are used when we want to process **text files**. As we know Java stores characters in Unicode format. Character stream processes data **character by character**. Character stream performs input and output operations of **16-bit Unicode** which is equivalent to the size of a character in Java.

Example- Character streams are used to read/write characters.

When to use Byte Stream over Character Stream?

Byte streams are used to process raw data like **binary files**. If we have a file that contains binary data, then it will be appropriate to use Byte stream. They can be used to **read/write** data of **8-bit bytes**.

Example- Byte streams are used to read or write binary data.

Benefits of Byte Stream

- Byte stream provides a convenient way to handle the input and output of bytes. If your file is too large, byte stream handles data in chunks rather than having the entire data altogether in the memory.
- They are useful when we want to read/write binary data.

Benefits of Character Stream

- Character stream provides convenient means to handle character-based inputs and outputs.
- Since they use Unicode, Character streams can be internationalized. *Internationalization is the process of preparing an application that supports linguistic, regional, cultural or political-specific data.*
- In some cases, character streams are more efficient than byte streams especially when the file contains characters.
- Character streams automatically translate the internal format of the file (the content of the file) to and from the local character set without extra effort by the programmer.

- In short, character streams make it easy to write programs that are not dependent upon a specific character encoding, which becomes easy to internationalize.

Note:

- The original version of Java(Java 1.0) did not include character streams. Thus, earlier all I/O was **Byte Oriented**. Character streams were added in Java 1.1.
- Names of byte stream classes end with *InputStream/OutputStream* while the names of character stream classes end with *Reader/Writer*.
- It is recommended to close the stream when it is no longer in use. This is to ensure that the streams shouldn't be affected if any error occurs.
- The above codes may not run in online compilers as the source files may not exist.

Conclusion

- The stream method helps to sequentially access a file.
- There are 2 Streams in Java- Byte Stream and Character Stream.
- Byte streams are used to perform input and output of 8-bit bytes.
- Byte streams are useful when we want to read/write binary data.
- Character stream is used to perform input and output operations of 16-bit Unicode.
- Character streams are used to read/write characters.
- Byte Streams and Character Streams have a three phase mechanism which includes Split, Apply and Combine.
- FileInputStream and FileOutputStream are common classes to read/write data using byte streams.
- FileReader and FileWriter are commonly used to read/write data using character streams.