CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

Stream Classes

Overview

Introduction

Streams in Java represent an ordered sequence of data. Java performs input and output operations in the terms of streams.

It uses the concept of streams to make I/O operations fast. For example, when we read a sequence of bytes from a binary file, actually, we’re reading from an input stream.

Similarly, when we write a sequence of bytes to a binary file, we’re writing to an output stream.

Byte Streams and Character Streams

Modern versions of Java platform define two types of I/O streams:

- Byte streams
- Character streams

Let’s understand first the meaning of byte streams and character streams one-by-one.

Byte Streams in Java

Byte streams in Java are designed to provide a convenient way for handling the input and output of bytes (i.e., units of 8-bits data). We use them for reading or writing to binary data I/O.

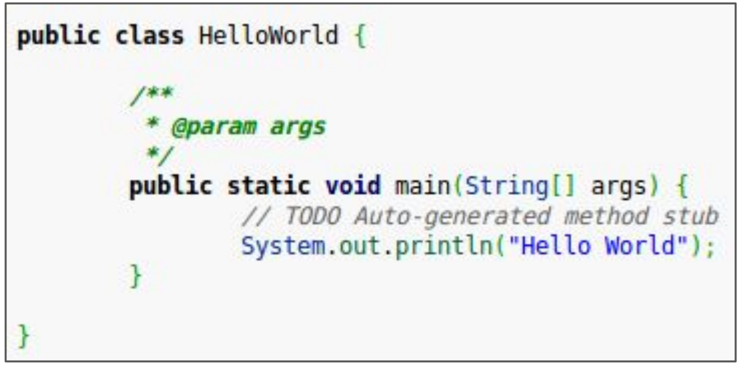
Byte streams are especially used when we are working with binary files such as executable files, image files, and files in low-level file formats such as .zip, .class, .obj, and .exe.

Binary files are those files that are machine readable.

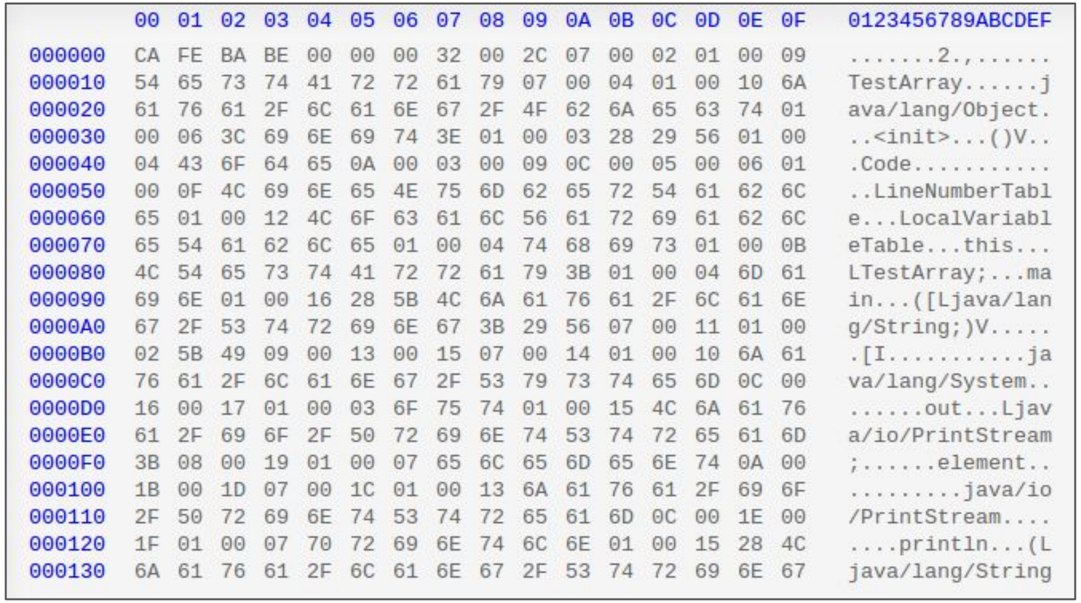
Byte streams that are used for reading are called **input streams** and for writing are called **output streams**. They are represented by the abstract classes of InputStream and OutputStream in Java.

Byte Streams - Real-time Examples

For example, a Java class file is an extension of “.class” and humans cannot read it.
It can be processed by low-level tools such as a JVM (executable java.exe in Windows) and java disassembler (executable javap.exe in Windows).



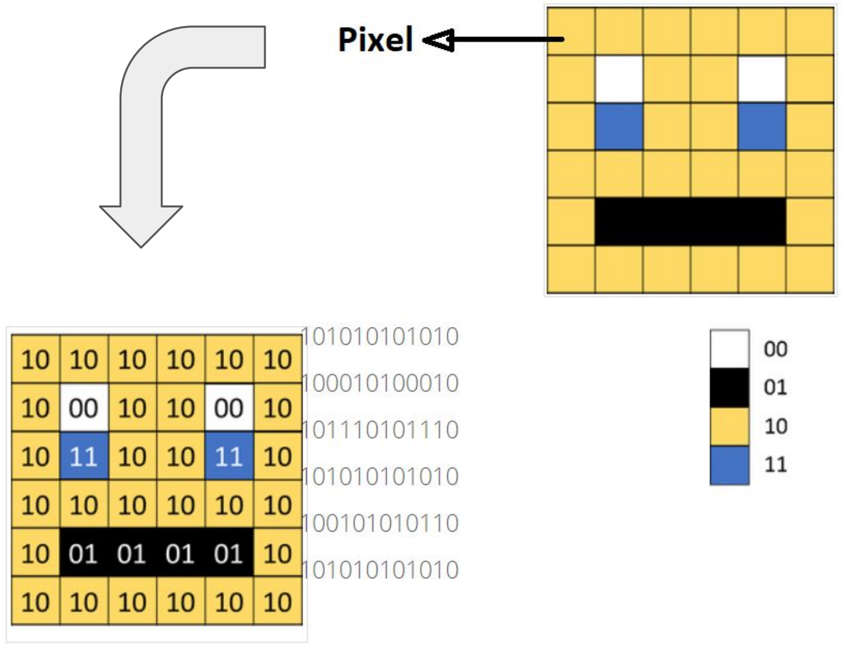
```java
public class HelloWorld {

        /**
         * @param args
         */
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                System.out.println("Hello World");
        }

}
```



```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   0123456789ABCDEF
000000  CA FE BA BE 00 00 00 32 00 2C 07 00 02 01 00 09   .......2.,......
000010  54 65 73 74 41 72 72 61 79 07 00 04 01 00 10 6A   TestArray......j
000020  61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 01   ava/lang/Object.
000030  00 06 3C 69 6E 69 74 3E 01 00 03 28 29 56 01 00   ..<init>...()V..
000040  04 43 6F 64 65 0A 00 03 00 09 0C 00 05 00 06 01   .Code...........
000050  00 0F 4C 69 6E 65 4E 75 6D 62 65 72 54 61 62 6C   ..LineNumberTabl
000060  65 01 00 12 4C 6F 63 61 6C 56 61 72 69 61 62 6C   e...LocalVariabl
000070  65 54 61 62 6C 65 01 00 04 74 68 69 73 01 00 0B   eTable...this...
000080  4C 54 65 73 74 41 72 72 61 79 3B 01 00 04 6D 61   LTestArray;...ma
000090  69 6E 01 00 16 28 5B 4C 6A 61 76 61 2F 6C 61 6E   in...([Ljava/lan
0000A0  67 2F 53 74 72 69 6E 67 3B 29 56 07 00 11 01 00   g/String;)V.....
0000B0  02 5B 49 09 00 13 00 15 07 00 14 01 00 10 6A 61   .[I.........ja
0000C0  76 61 2F 6C 61 6E 67 2F 53 79 73 74 65 6D 0C 00   va/lang/System..
0000D0  16 00 17 01 00 03 6F 75 74 01 00 15 4C 6A 61 76   ......out...Ljav
0000E0  61 2F 69 6F 2F 50 72 69 6E 74 53 74 72 65 61 6D   a/io/PrintStream
0000F0  3B 08 00 19 01 00 07 65 6C 65 6D 65 6E 74 0A 00   ;......element..
000100  1B 00 1D 07 00 1C 01 00 13 6A 61 76 61 2F 69 6F   .........java/io
000110  2F 50 72 69 6E 74 53 74 72 65 61 6D 0C 00 1E 00   /PrintStream....
000120  1F 01 00 07 70 72 69 6E 74 6C 6E 01 00 15 28 4C   ....println...(L
000130  6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67   java/lang/String
```

Byte Streams - Real-time Examples

Another realtime example is storing a photo in a .bmp or .jpeg file.



10	10	10	10	10	10
10	00	10	10	00	10
10	11	10	10	11	10
10	10	10	10	10	10
10	01	01	01	01	10
10	10	10	10	10	10

These files are certainly not human readable.

Photo editing or image manipulation software can only process them.

Character Streams in Java

Character streams in Java are designed for handling the input and output of characters. They use 16-bit Unicode characters.

Character streams are more efficient than byte streams. They are mainly used for reading or writing to character or text-based I/O such as text files, text documents, XML, and HTML files.

Text files are those files that are human readable. For example, a .txt file that contains human-readable text. This file is created with a text editor such as Notepad in Windows.

Character streams that are used for reading are called **readers** and for writing are called **writers**. They are represented by the abstract classes of Reader and Writer in Java.

Stream Classes in Java

All streams in Java are represented by classes in `java.io` package. This package contains a lot of stream classes that provide abilities for processing all types of data.

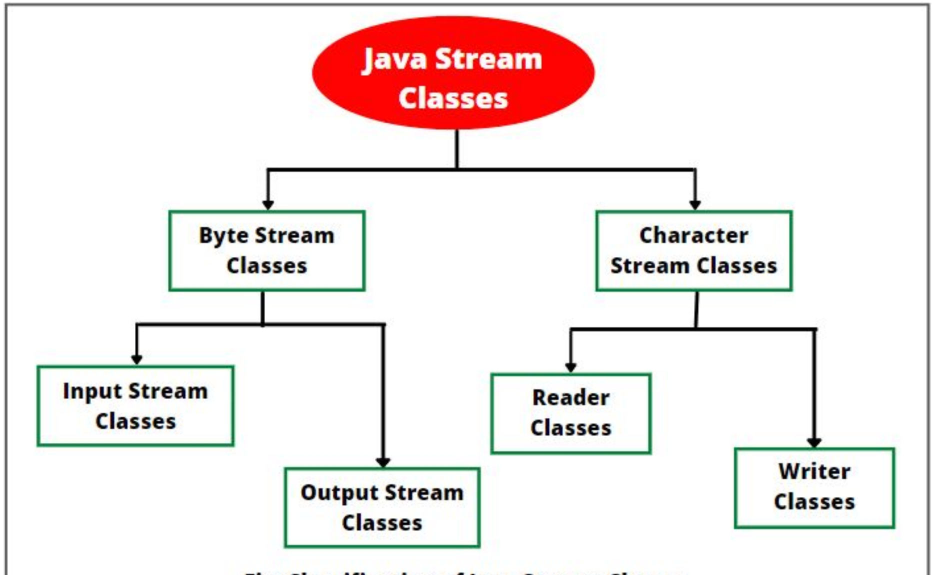


Fig: Classification of Java Stream Classes

We can classify these stream classes into two basic groups based on the date type. They are:

- Byte Stream Classes
  (support for handling I/O operations based on bytes)

- Character Stream Classes
  (support for managing I/O operations on characters)

Byte Stream Classes in Java

There are two kinds of byte stream classes in Java. They are as follows:

- InputStream classes
- OutputStream classes

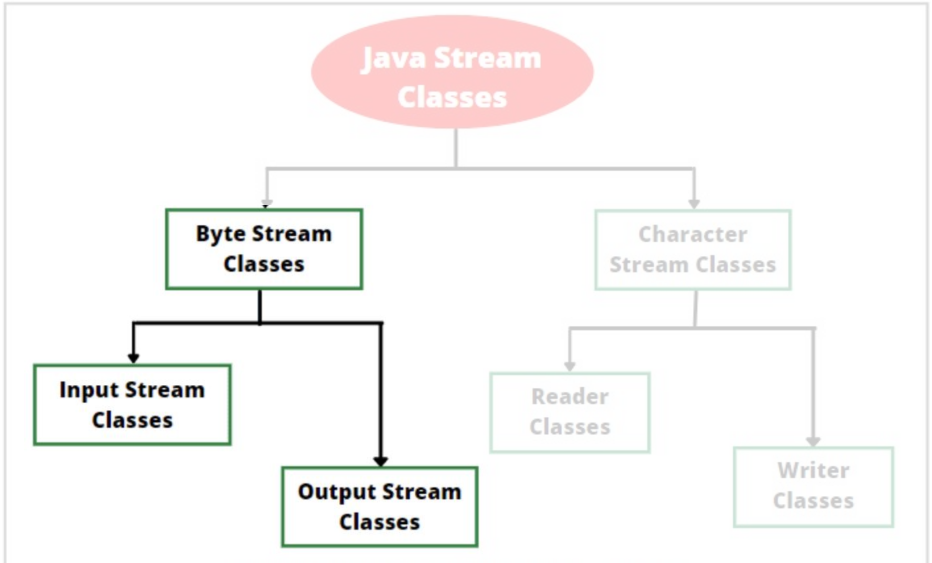


Fig: Classification of Java Stream Classes

InputStream Classes in Java

`InputStream` class is an abstract class. It is the root class for reading binary I/O data. It is the superclass of all classes representing an input stream of bytes.

Since `InputStream` class is an abstract class, we cannot create an object of this class. We must use subclasses of this class to create an object.

The **several subclasses** of Java `InputStream` class can be used for performing several input functions. They are listed with a brief description in the below table:

InputStream Classes in Java

InputStream Subclass	Description
BufferedInputStream	It adds buffering abilities to an input stream. In simple words, it buffered input stream.
ByteArrayInputStream	Input stream that reads data from a byte array.
DataInputStream	It reads bytes from the input stream and converts them into appropriate primitive-type values or strings.
FileInputStream	This input stream reads bytes from a file.
FilterInputStream	It filters bytes from an input stream.
ObjectInputStream	Input stream for objects
PipedInputStream	It creates a communication channel on which data can be received.
PushbackInputStream	It is a subclass of FilterInputStream that adds “pushback” functionality to an input stream.
SequenceInputStream	It is used to read input streams sequentially, one after the other.

InputStream Methods in Java

The abstract `InputStream` class in Java defines **several methods** for performing input functions such as reading bytes, closing streams, marking positions in streams, etc.

All these methods are present in `java.io.InputStream` package. `InputStream` methods are as follows:

InputStream Methods in Java

Method	Description
int read()	The read() method reads the next byte of data from the input stream. It returns input byte read as an int value in the range 0 to 255. If no byte is present because the end of the stream is reached, the value –1 is returned. If this method encounters an I/O error, it will throw an IOException. The read() method returns an int value -1 instead of a byte because it indicates the end of stream.
int read(byte[ ] b)	This method reads the number of bytes from the input stream and stores them into the array of bytes b. It returns the total number of bytes read as an int. If the end of stream is reached, returns -1.
int read(byte[ ] b, int n, int m)	It reads up to m bytes of data from the input stream starting from nth byte into an array b. It returns the total number of bytes read as an int. Returns -1 at the end of stream because of no more data.

InputStream Methods in Java

Method	Description
int available()	The available method returns an estimate of the number of bytes that can be read (or skipped over) from the input stream.
void close()	The close() method closes the input stream and releases any system resources associated with it.
long skip(long n)	This method skips over n bytes of data from this input stream. It returns the actual number of bytes skipped.
void reset()	The reset() method is used to go back to the beginning of the stream.
boolean markSupported()	This method tests this input stream supports the mark and reset methods. It returns true if the input stream supports mark and reset methods.

OutputStream Classes in Java

`OutputStream` class is an abstract class. It is the root class for writing binary data. It is a superclass of all classes that represents an output stream of bytes.

Since `OutputStream` is an abstract class we cannot directly create objects of it. The hierarchy of classification of `OutputStream` classes has shown in the above diagram.

The **several subclasses** of `OutputStream` class in Java can be used for performing several output functions. They are listed with a brief description in the below table:

OutputStream Classes in Java

OutputStream Subclass	Description
BufferedOutputStream	It adds buffering abilities to an output stream. In simple words, it buffered output stream.
ByteArrayOutputStream	Output stream that writes data to a byte array.
DataOutputStream	It converts primitive-type values or strings into bytes and outputs bytes to the stream.
FileOutputStream	It writes byte stream into a file.
FilterOutputStream	It filters bytes from an output stream.
ObjectOutputStream	Output stream for objects
PipedOutputStream	It creates a communication channel on which data can be sent.

OutputStream Methods in Java

The `OutputStream` class defines the following method in `java.io.OutputStream` package that are used to perform output tasks such as writing bytes, closing streams, and flushing streams.

The `OutputStream` **methods** with a brief description are as follows:

OutputStream Methods in Java

Method	Description
void write(int b)	The write() method writes the specified byte to the output stream. It accepts an int value as an input parameter. It throws an IOException if an I/O error occurs (e.g. output stream has been closed).
void write(byte[ ] b)	This method writes all the specified bytes in the array b to the output stream.
void write(byte[ ] b, int n, int m)	It writes m bytes from array b starting from nth byte to the output stream.
void close()	It closes the output stream and releases any system resources associated with this stream.
void flush()	It flushes the output stream and forces any buffered output bytes to be written out.

END