The ****java.io**** package is used to handle input and output operations. Java IO has various classes that handle input and output sources. A **stream** is a sequence of data.

Java input stream classes can be used to read data from input sources such as ****keyboard**** or a ****file****. Similarly output stream classes can be used to write data on a ****display**** or a ****file**** again.

We can also perform ****File Handling**** using ****Java IO API****.

## Introduction to I/O Streams in Java

Before understanding IO streams, let us discuss streams. ****A Stream is also a sequence of data.**** It is neither a data structure nor it stores data. Take an example of a river stream, where water flows from source to destination. Similarly, these are data streams; data flows through one point to another.

To handle these sequences, we introduce a term called IO streams.

The ****java.io**** package helps the user to perform all types of input-output operations. Java IO package is primarily focused on input-output files, network streams, internal memory buffers, etc. Data is read and written from ****Java IO****'s InputStream and OutputStream classes.

In other words, IO streams in java help to read the data from an input stream such as a file and write the data into an output stream such as the standard display or a file again. It represents source as input and destination as output. It can handle all types of data, from primitive values to advanced objects.

## What is Java IO?

The ****java.io**** package consists of input and output streams used to read and write data to files or other input and output sources.

There are 3 categories of classes in java.io package:

* Input Streams.
* Output Streams.
* Error Streams.

Java supports three streams that are automatically attached with the console.

1. ****System.out:**** Standard output stream
2. ****System.in:**** Standard input stream
3. ****System.err:**** Standard error stream

### Input Streams

As we know input source consists of data that needs to be read in order to extract information from it. Input Streams help us to read data from the input source. It is an **abstract**class that provides a programming interface for all input streams.

Input streams are opened implicitly as soon as it is created. To close the input stream, we use a close() method on the source object.

### Output Streams

The output of the executed program has to be stored in a file for further use. Output streams help us to write data to a output source(may be file). Similarly like input streams output streams are also abstract classes that provides a programming interface for all output streams.

The output stream is opened as soon as it is created and explicitly closed by using the close() method.

### Error Streams

Error streams are the same as output streams. In some ide’s error is displayed in different colors (other than the color of output color). It gives output on the console the same as output streams.

## Why We Need IO Streams in Java?

In day-to-day work, we do not enter the input into the programs manually. Also, the result of the program needs to be stored somewhere for further use.

So, IO streams in Java provide us with input and output streams that help us to extract data from the files and write the data into the files. Normally, we can create, delete, and edit files using Java.io.

In short, all the file manipulation is done using Java IO streams. Java IO streams also handle user input functionality.

## Types of Streams in Java

Depending on the types of operations, streams are divided into 2 primary classes.

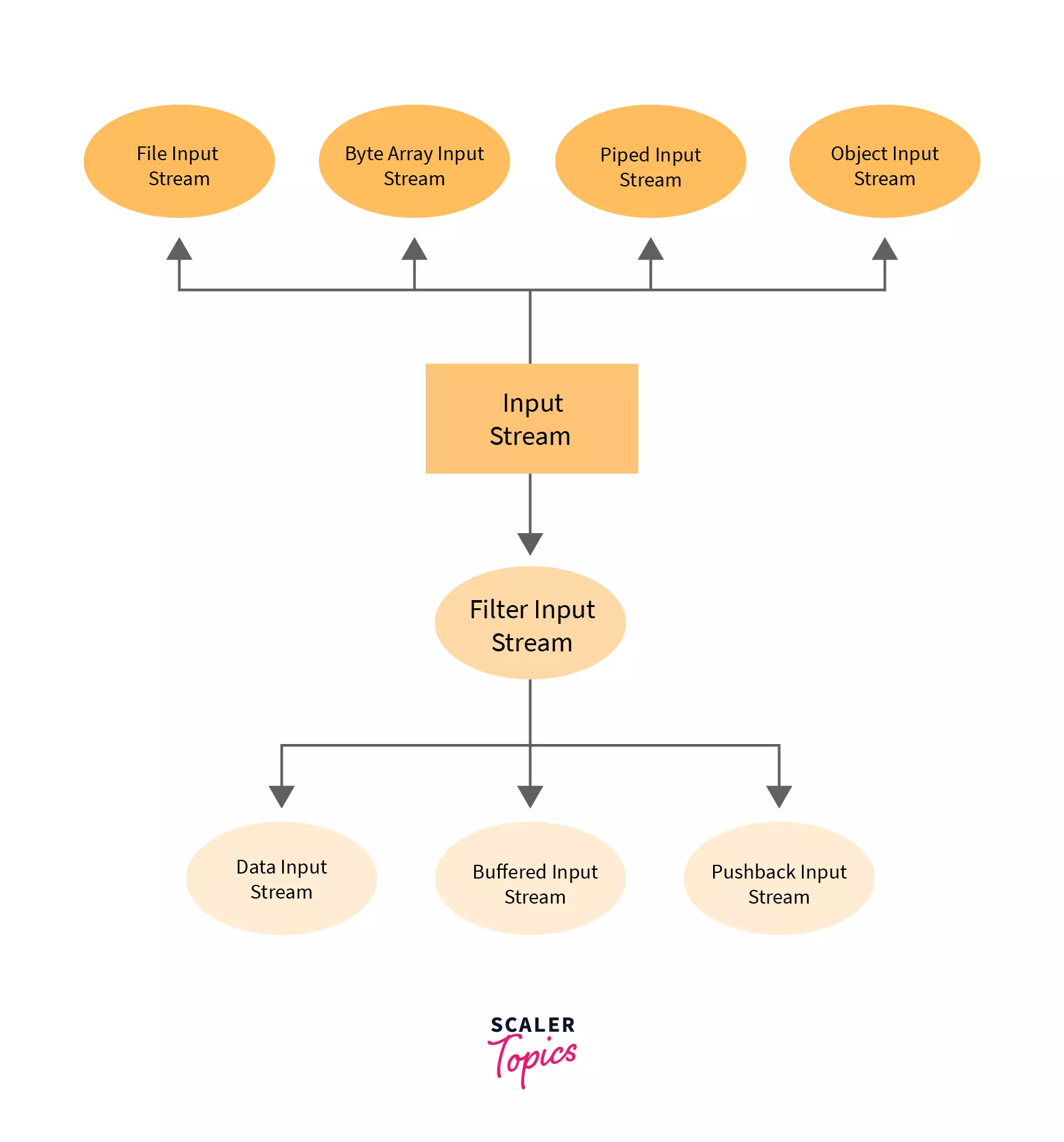
### Input Stream

It is an abstract superclass of the ****java.io**** package and is used to read the data from an input source. In other words, reading data from files or from a keyboard, etc. We can create an object of the input stream class using the new keyword. The input stream class has several types of constructors.

The following code takes the file name as a string, to read the data stored in the file.

InputStream f = new FileInputStream("input.txt");

### InputStream Hierarchy



#### Useful methods of InputStream

****1. public abstract int**read()**throws IOException****

The method above helps to return the data of the next byte in the input stream. The value returned is between 0 to 255. If no byte is read, the code returns -1, which indicates the end of the file.

****2. public int**available()**throws IOException****

The method above returns the number of bytes that can be read from the input stream.

****3. public void**close()**throws IOException****

The method above closes the current input stream and releases any system resources associated with it.

****4. public void**mark(int readlimit)**

It marks the current position in the input stream. The readlimit argument tells the input stream to read that many bytes to read before the mark position gets invalid.

****5. public boolean**markSupported()**

It tells whether the mark() and reset() method is supported in a particular input stream. It returns true if the mark and reset methods are supported by the particular input stream or else return false.

****6. public int**read(byte[ ] b)**throws IOException****

The method above reads the bytes from the input stream and stores every byte in the buffer array. ****It returns the total number of bytes stored in the buffer array****. If there is no byte in the input stream, it returns -1 as the stream is at the end of the file.

****7. public int**read(byte[ ] b , int off , len)**throws IOException****

It reads up to len bytes of data from the input stream. It returns the total number of bytes stored in the buffer. Here the “off” is start offset in buffer array b where the data is written, and the “len” represents the maximum number of bytes to read.

****8. public void**reset()**throws IOException****

It repositions the stream to the last called mark position. The reset method does nothing for input stream class except throwing an exception.

****9. public long**skip(long n)**throws IOException****

This method discards n bytes of data from the input stream.

### ****Examples****

1. In the below example, we will use ****FileInputStream**** class to read the input file: ****input.txt****.

* Create a file input.txt and place it in the same directory as Main.java
* Let us suppose ****input.txt**** contains the following content:

Scaler TopicsFrom InterviewBit

****Code:****

import java.io.\*;class Main { public static void main(String[] args) throws IOException { try { *// loading a file into f variable* FileInputStream f = new FileInputStream("input.txt"); *// initializing x to 0* int x = 0; *// while loop untill the end of the file.* while ((x = f.read()) != -1) { *// printing the character* System.out.print((char) x); } *// closing a file* f.close(); } catch (Exception e) { *// printing exception* System.out.println(e);

}

}

}

****Output:****

Scaler TopicsFrom InterviewBit

1. In the below example, we will use ****BufferedInputStream**** class to read the file.

****Code:****

import java.io.\*;class Main { public static void main(String[] args) throws IOException { try { *// loading a file into f1 variable using FileInputStream* FileInputStream f1 = new FileInputStream("input.txt"); *// loading a file into f2 variable using BufferInputStream* BufferedInputStream f2 = new BufferedInputStream(f1); *// using the available method* System.out.println("Available bytes: " + f2.available()); int x = 0; *// while loop untill the end of the file.* while ((x = f2.read()) != -1) { *// printing the character* System.out.print((char) x); }

System.out.println(); *// closing a file* f2.close(); } catch (Exception e) { *// printing exception* System.out.println(e);

}

}

}

****Output:****

Available bytes: 31Scaler Topics

From InterviewBit

1. In the below example we will use ****ByteArrayInputStream**** class to read the file.

****Code:****

import java.io.\*;class Main { public static void main(String[] args) throws IOException { try { *// loading a file into f1 variable using FileInputStream* FileInputStream f1 = new FileInputStream("input.txt"); int x = 0; String S = ""; *// while loop untill the end of the file.* while ((x = f1.read()) != -1) { *// printing the character* S = S + (char) x; } *// closing a input stream* f1.close();

*// converting string to array of bytes* byte[] b = S.getBytes(); *// declaring ByteArrayInputStream* ByteArrayInputStream b1 = new ByteArrayInputStream(b);

x = b1.read(); while (x != -1) { System.out.print((char) x); x = b1.read();

}

System.out.println(); *// close the input stream* b1.close(); } catch (Exception e) { *// printing exception* System.out.println(e);

}

}

}

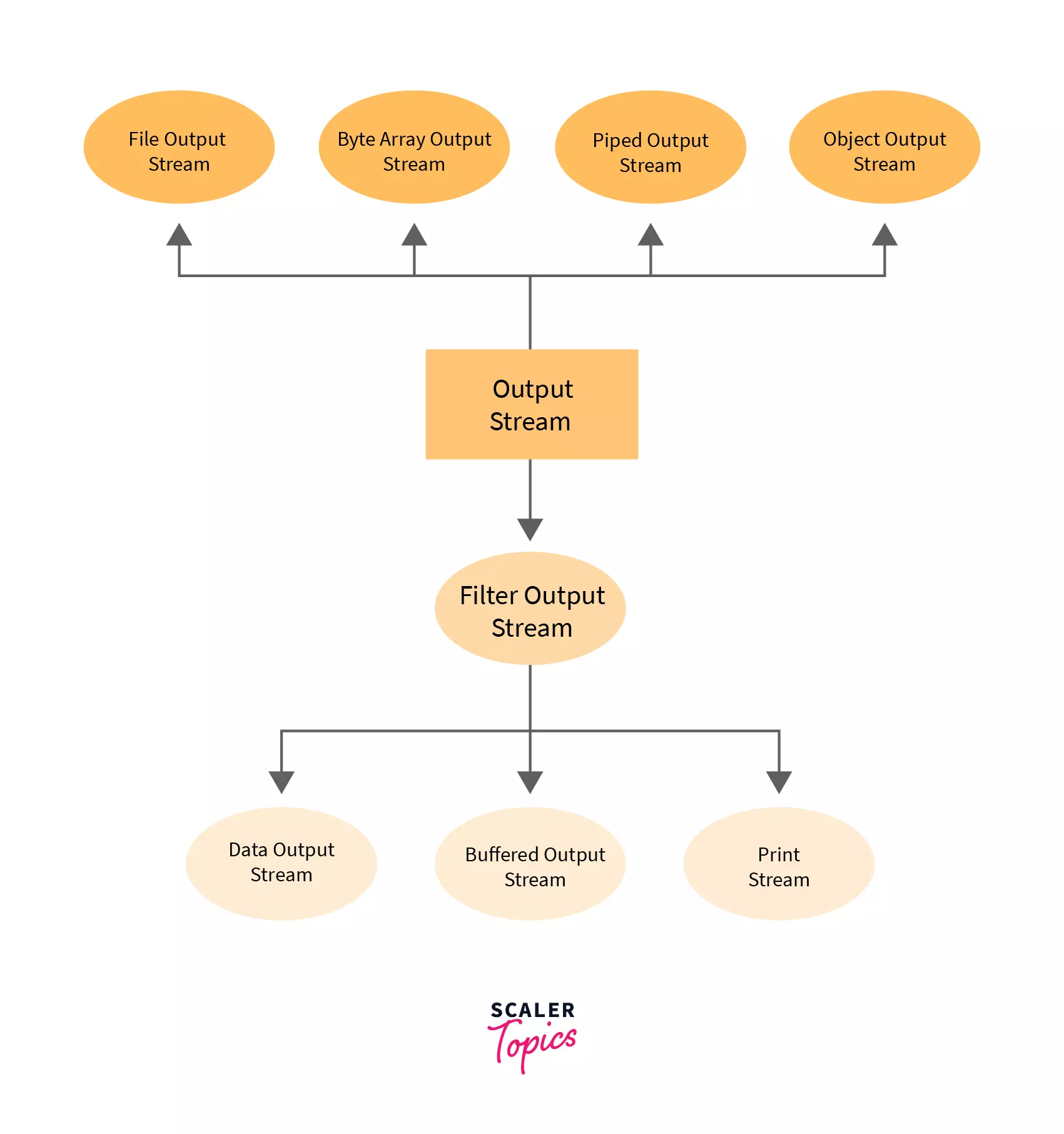
****Output:****

Scaler TopicsFrom InterviewBit

### Output Stream

It is an abstract superclass of the java.io package and writes data to an output resource. In other words, writing the data into the files. We can create an object of the output stream class using the new keyword. The output stream class has several types of constructors.

### OutputStream Hierarchy



### Useful methods of OutputStream

****1. public void**close()**throws IOException****

This method closes the current output stream and releases any system resources associated with it. The closed stream cannot be reopened and operations cannot be performed within it.

****2. public void**flush()**throws IOException****

It flushes the current output stream and forces any buffered output to be written out.

****3. Public void**write(byte[ ] b)**throws IOException****

This method writes the b.length bytes from the specified byte array to the output stream.

****4. Public void**write(byte[ ] b ,int off ,int len)**throws IOException****

It writes upto len bytes of data to the output stream. Here the “off” is the start offset in buffer array b, and the “len” represents the maximum number of bytes to be written in the output stream.

****5. Public abstract void**write(int b)**throws IOException****

The method above writes the specific bytes to the output stream. It does not return a value.

Some methods that are inherited from class java.lang.Object. These methods are used for both input stream and output stream purposes.

Example: clone, equals, finalise, getclass, hashCode, notify, notifyAll, toString, wait.

### ****Examples****

1. In the below example we will use ****FileOutputStream**** class to read the file.

****Code:****

import java.io.\*;class Main { public static void main(String[] args) throws IOException { try { *// loading a file into f variable* FileOutputStream f = new FileOutputStream("output.txt"); String s = "Scaler Topics"; char arr[] = s.toCharArray(); *// initializing x to 0* int x = 0; *// while loop untill the end of the string.* while (x < s.length()) { *// writing a byte into "output.txt" file* f.write(arr[x++]);

} *// closing a file* f.close(); } catch (Exception e) { *// printing exception* System.out.println(e);

}

}

}

****Output:****

The ****output.txt**** file will contain the following text:

Scaler Topics

1. In the below example we will use ****BufferedOutputStream**** class to read the file.

****Code:****

import java.io.\*;class Main { public static void main(String[] args) throws IOException { try { *// loading a file into f variable* FileOutputStream f1 = new FileOutputStream("output.txt"); *// declaring a f1 as BufferedOutputStream* BufferedOutputStream f2 = new BufferedOutputStream(f1); String s = "Scaler Topics"; char arr[] = s.toCharArray(); *// initializing x to 0* int x = 0; *// while loop untill the end of the string.* while (x < s.length()) { *// writing a byte into "output.txt" file* f2.write(arr[x++]);

} *// closing a file* f2.close();

f1.close(); } catch (Exception e) { *// printing exception* System.out.println(e);

}

}

}

****Output:****

The ****output.txt**** file will contain the following text:

Scaler Topics

1. In the below example we will use ****ByteArrayOutputStream**** class to read the file.

****Code:****

import java.io.\*;class Main { public static void main(String[] args) throws IOException { try { *// loading a file into f variable* FileOutputStream f = new FileOutputStream("output.txt"); String s = "Scaler Topics"; *// declaring ByteArrayOutputStream* ByteArrayOutputStream b1 = new ByteArrayOutputStream(); *// writing a data to "output.txt" file* b1.write(s.getBytes());

b1.writeTo(f);

*// closing a file* b1.close(); } catch (Exception e) { *// printing exception* System.out.println(e);

}

}

}

****Output:****

The ****output.txt****`` file will contain the following text:

Scaler Topics

## Conclusion

* Java has three main types of IO streams.
  + Input Streams
  + Output Streams
  + Error Streams
* Input Stream and Output Stream are abstract superclasses of the ****java.io**** package.
* Input and Output Streams are used to read data from input sources (such as Standard Input) and write data into output sources (such as files, console) respectively.