Java I/O

Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

Let's see the code to print **output and an error** message to the console.

```
1. System.out.println("simple message");
2. System.err.println("error message");
```

Let's see the code to get **input** from console.

```
1. int i=System.in.read();//returns ASCII code of 1st character
2. System.out.println((char)i);//will print the character
```

OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

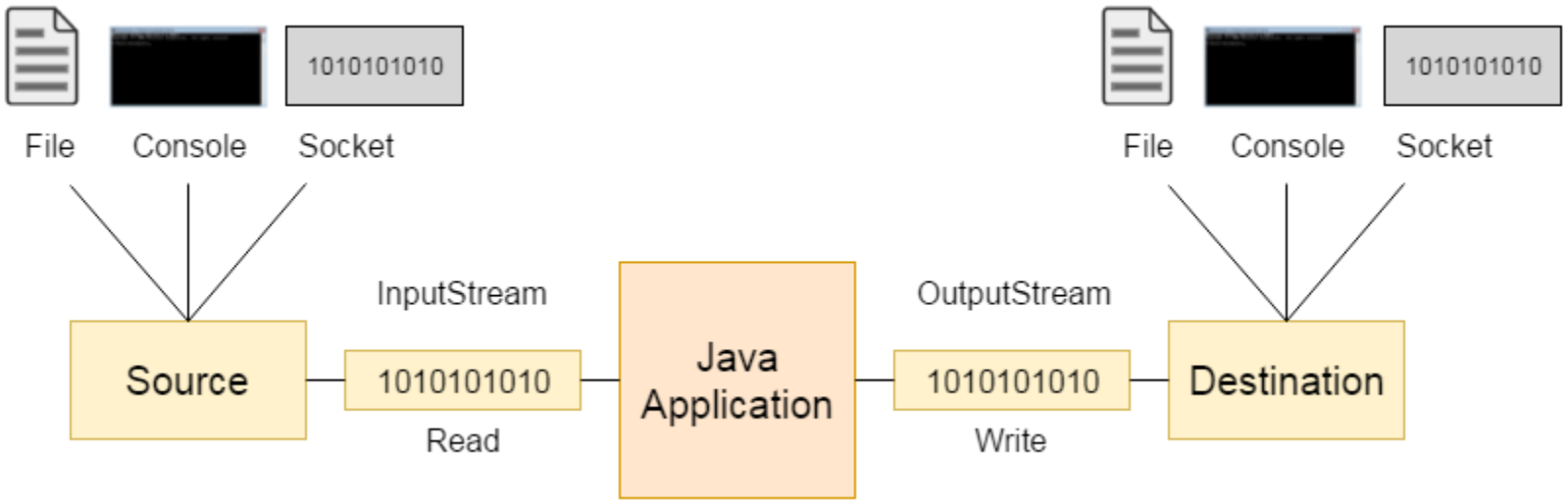
OutputStream

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

Let's understand the working of Java OutputStream and InputStream by the figure given below.



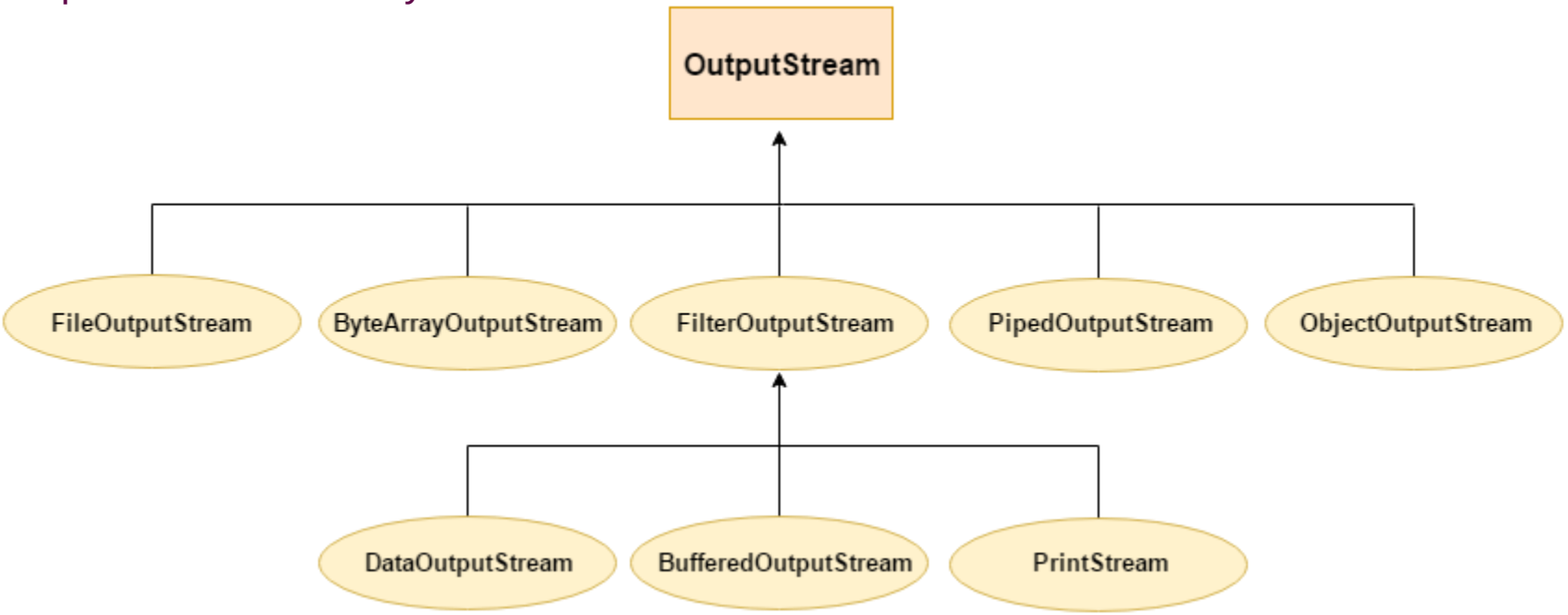
OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Useful methods of OutputStream

Method	Description
1) public void write(int)throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException	flushes the current output stream.
4) public void close()throws IOException	is used to close the current output stream.

OutputStream Hierarchy



InputStream class

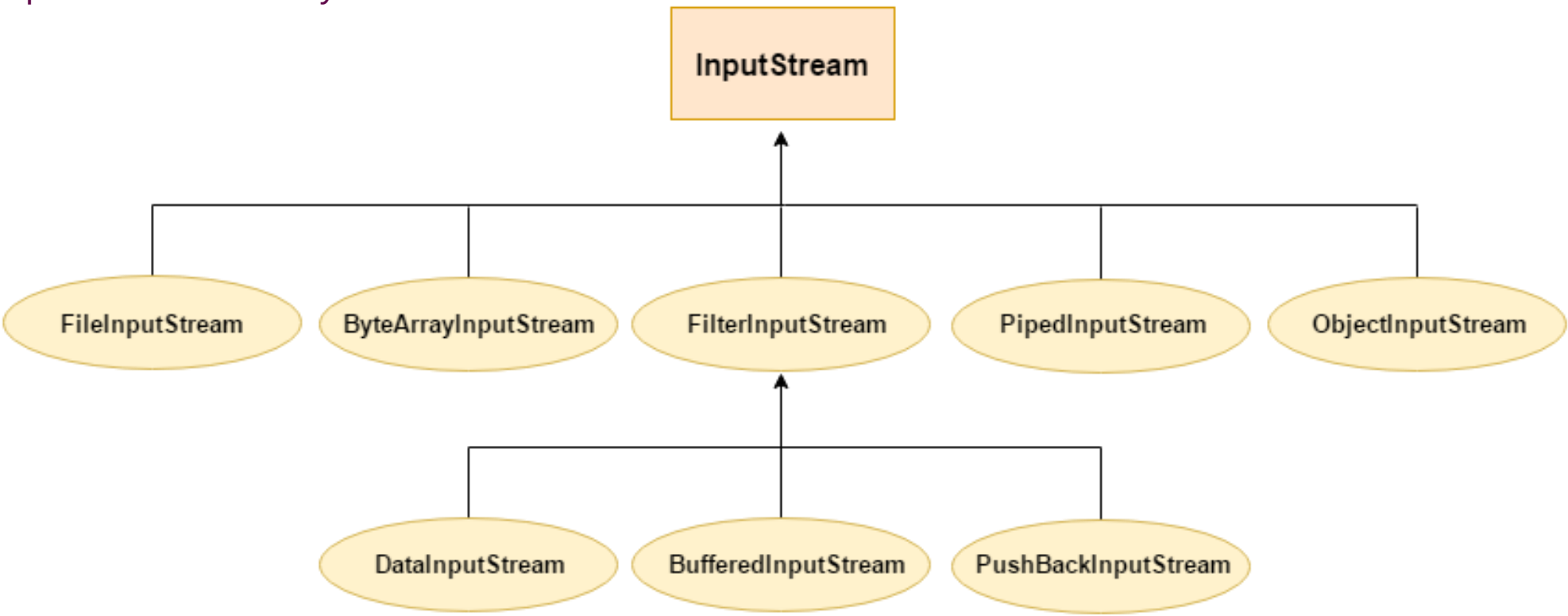
InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

Useful methods of InputStream

Method	Description
1) public abstract int read()throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available()throws	returns an estimate of the number of bytes that can be read from the current input stream.

IOException	
3) public void close()throws IOException	is used to close the current input stream.

InputStream Hierarchy



Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a file.

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than

FileOutputStream.

FileOutputStream class declaration

Let's see the declaration for Java.io.FileOutputStream class:

```
1. public class FileOutputStream extends OutputStream
```

FileOutputStream class methods

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

Java FileOutputStream Example 1: write byte

```
1. import java.io.FileOutputStream;
2. public class FileOutputStreamExample {
3.     public static void main(String args[]){
4.           try{
5.             FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
6.             fout.write(65);
7.             fout.close();
8.             System.out.println("success...");
9.            }catch(Exception e){System.out.println(e);}
10.     }
11. }
```

Output:

```
Success...
```

The content of a text file **testout.txt** is set with the data **A**.

testout.txt

```
A
```

Java FileOutputStream example 2: write string

```
1. import java.io.FileOutputStream;
2. public class FileOutputStreamExample {
3.     public static void main(String args[]){
4.           try{
5.             FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
6.             String s="Welcome to javaTpoint.";
7.             byte b[]=s.getBytes();//converting string into byte array
8.             fout.write(b);
9.             fout.close();
```

```
10.             System.out.println("success...");
11.            }catch(Exception e){System.out.println(e);}
12.      }
13. }
```

Output:

```
Success...
```

Java FileInputStream Class

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

Java FileInputStream class declaration

Let's see the declaration for java.io.FileInputStream class:

```
1. public class FileInputStream extends InputStream
```

Java FileInputStream class methods

Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.

int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream.

Java FileInputStream example 1: read single character

```
1.  import java.io.FileInputStream;
2.  public class DataStreamExample {
3.      public static void main(String args[]){
4.          try{
5.            FileInputStream fin=new FileInputStream("D:\\testout.txt");
6.            int i=fin.read();
7.            System.out.print((char)i);
8.
9.            fin.close();
10.         }catch(Exception e){System.out.println(e);}
11.        }
12.       }
```

Note: Before running the code, a text file named as **"testout.txt"** is required to be created. In this file, we are having following content:

```
Welcome to javatpoint.
```

After executing the above program, you will get a single character from the file which is 87 (in byte form). To see the text, you need to convert it into character.

Output:

```
W
```

Java FileInputStream example 2: read all characters

```
1.  package com.javatpoint;
2.
3.  import java.io.FileInputStream;
4.  public class DataStreamExample {
5.      public static void main(String args[]){
6.          try{
7.            FileInputStream fin=new FileInputStream("D:\\testout.txt");
8.            int i=0;
9.            while((i=fin.read())!=-1){
10.            System.out.print((char)i);
11.           }
12.           fin.close();
13.         }catch(Exception e){System.out.println(e);}
14.        }
15.       }
```

Output:

```
Welcome to javaTpoint
```