# FILES

## Streams

A stream is a method to sequentially access a file. I/O Stream means an input source or output destination representing different types of sources e.g. disk files.The java.io package provides classes that allow you to convert between Unicode character streams and byte streams of non-Unicode text.

**Stream** – A sequence of data.
**Input Stream:** reads data from source.
**Output Stream:** writes data to destination.

## Character streams

In Java, characters are stored using Unicode conventions (Refer this for details). Character stream automatically allows us to read/write data character by character. For example FileReader and FileWriter are character streams used to read from source and write to destination.

Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**

## Byte Streams.

Byte streams process data byte by byte (8 bits). For example FileInputStream is used to read from source and FileOutputStream to write to the destination.

**When to use Character Stream over Byte Stream?**

- In Java, characters are stored using Unicode conventions. Character stream is useful when we want to process text files. These text files can be processed character by character. A character size is typically 16 bits.

**When to use Byte Stream over Character Stream?**

- Byte oriented reads byte by byte. A byte stream is suitable for processing raw data like binary files.

## Reader class

Java Reader is an abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods to provide higher efficiency, additional functionality, or both.

Some of the
implementation class are BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

```
1.  import java.io.*;
2.  public class ReaderExample {
3.     public static void main(String[] args) {
4.        try {
5.           Reader reader = new FileReader("file.txt");
6.           int data = reader.read();
7.           while (data != -1) {
8.              System.out.print((char) data);
9.              data = reader.read();
10.          }
11.          reader.close();
12.       } catch (Exception ex) {
13.          System.out.println(ex.getMessage());
14.       }
```

Writer class

It is an abstract class for writing to character streams. The methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses will override some of the methods defined here to provide higher efficiency, functionality or both.

```
1.  import java.io.*;
2.  public class WriterExample {
3.     public static void main(String[] args) {
4.        try {
5.           Writer w = new FileWriter("output.txt");
6.           String content = "I love my country";
7.           w.write(content);
8.           w.close();
9.           System.out.println("Done");
10.       } catch (IOException e) {
11.          e.printStackTrace();
12.       }
13.    }
```

Reading and writing to a file.

import java.io.*;
class fileio12

```
{
	public static void main(String args[]) throws IOException
	{
	FileInputStream infile=new FileInputStream("C:\\Users\\Desktop\\a1.txt");
	FileOutputStream outfile= new FileOutputStream ("C:\\Users\\Desktop\\a2.txt");


	try
		{

		int b;
		while ((b=infile.read())!=-1)
		{
		outfile.write(b);
		}
		}
	catch(Exception e)
	{
	System.out.println(e);
	}
infile.close();
}
}
```

## wrapper classes

**Wrapper class in java** provides the mechanism *to convert primitive into object and object into primitive*.

**autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.

## byte stream classes.

| | |
|---|---|
| BufferedInputStream | contains methods to read bytes from the buffer (memory area) |
| ByteArrayInputStream | contains methods to read bytes from a byte array |
| DataInputStream | contains methods to read Java primitive data types |
| FileInputStream | contains methods to read bytes from a file |
| FilterInputStream | contains methods to read bytes from other input streams which it uses as its basic source of data |
| ObjectInputStream | contains methods to read objects |

## The Character Stream class./ any 4

| Class | Description |
|---|---|
| BufferedReader | contains methods to read characters from the buffer |
| CharArrayReader | contains methods to read characters from a character array |
| FileReader | contains methods to read from a file |
| FilterReader | contains methods to read from underlying character-input stream |
| InputStreamReader | contains methods to convert bytes to characters |
| PipedReader | contains methods to read from the connected piped output stream |
| StringReader | contains methods to read from a string |

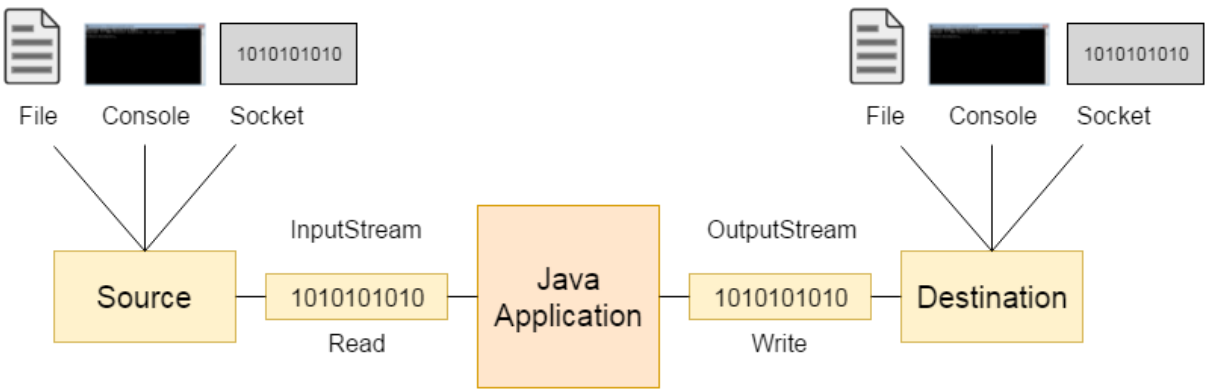
## InputStream class and Output stream class

## 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.



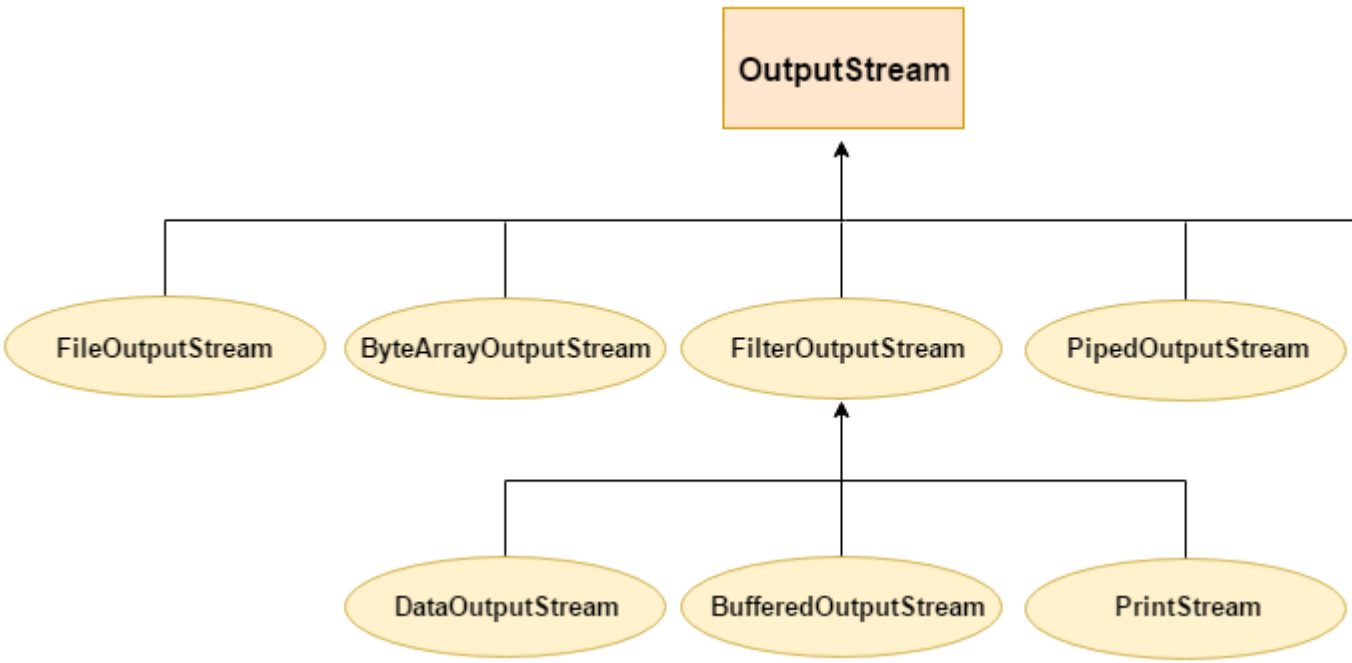

## OutputStream class

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 IOException | returns an estimate of the number of bytes that can be read from the current input stream. |
| 3) public void close()throws IOException | is used to close the current input stream. |

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. |

## flush() method

The **java.io.Writer.flush()** method flushes the stream. If the stream has saved any characters from the various write() methods in a buffer, write them immediately to their intended destination. Then, if that destination is another character or byte stream, flush it. Thus one flush() invocation will flush all the buffers in a chain of Writers and OutputStreams.

## isDirectory() method.

The **java.io.File.isDirectory()** checks whether the file denoted by this abstract pathname is a directory.