Java I/O

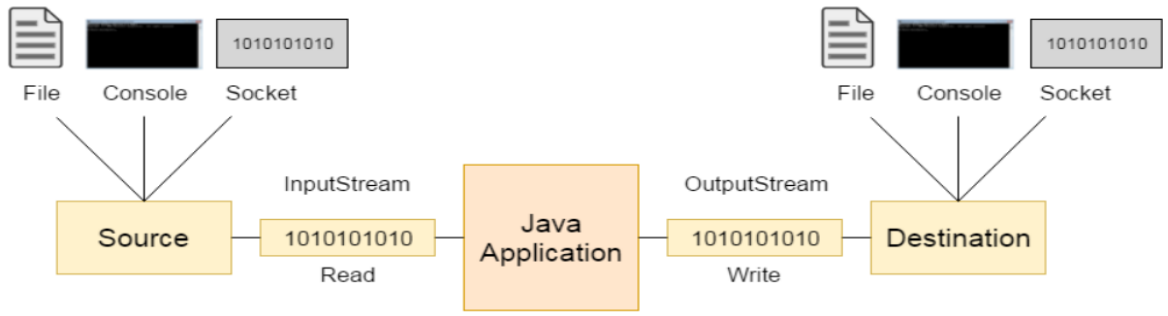
- Java I/O is used to process the input and produce the output.
- Java uses the concept of a stream to make I/O operation fast.
- The java.io package contains all the classes required for input and output operations.
- We can perform file handling in Java by Java I/O API.

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.

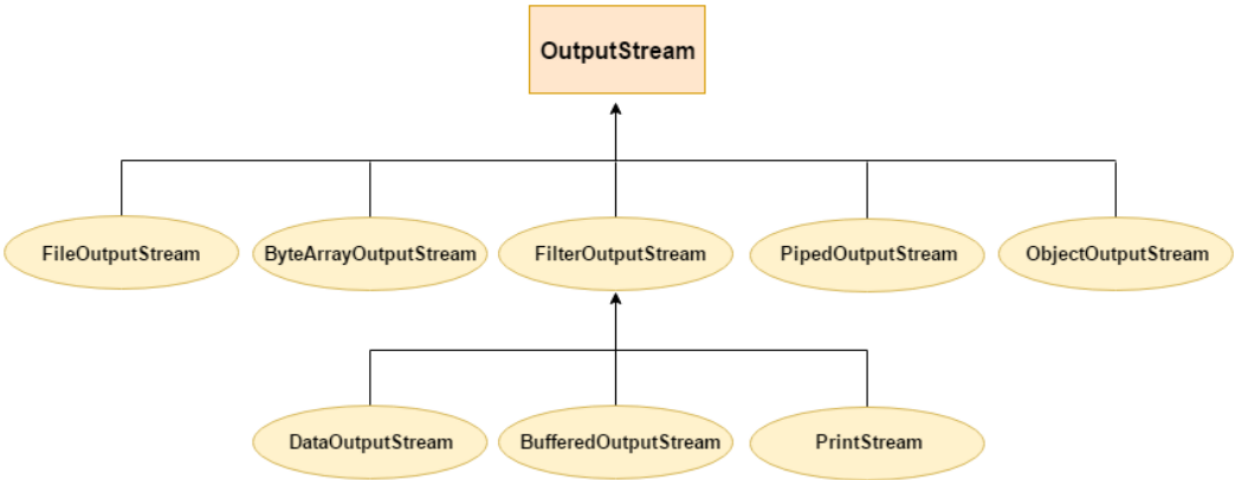
1) System.out: standard output stream
2) System.in: standard input stream
3) System.err: standard error stream

InputStream versus OutputStream



OutputStream

- Java applications use an output stream to write data to a destination;
- It may be a file, an array, peripheral device or socket.
- OutputStream class is an abstract class.
- It is the superclass of all classes representing an output stream of bytes.

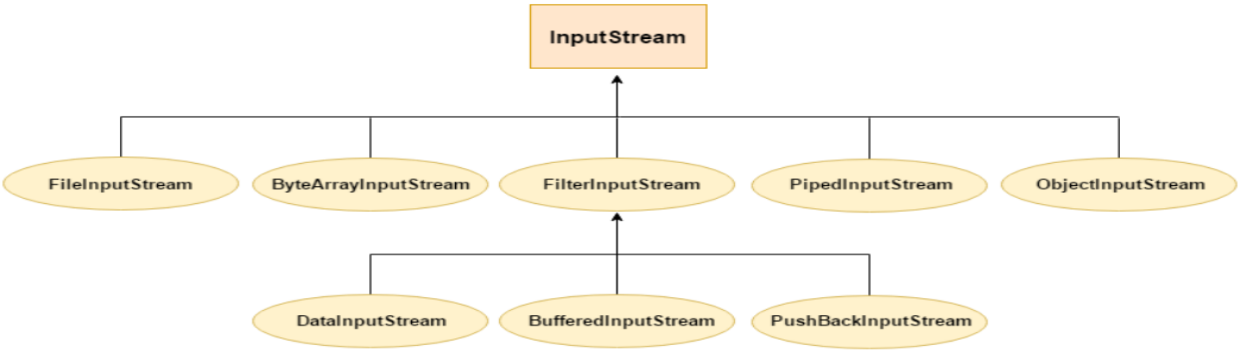


Methods:

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.

InputStream

- Java applications use an input stream to read data from a source;
- It may be a file, an array, peripheral device or socket.
- InputStream class is an abstract class.
- It is the superclass of all classes representing an input stream of bytes.



Methods:

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.

Character versus ByteStreams:

In Java the streams are used for input and output operations by allowing data to be read from or written to a source or destination.

Java offers two types of streams:

character streams

byte streams.

These streams can be different in how they are handling data and the type of data they are handling.

1. Character Streams:

Character streams are designed to address character based records, which includes textual records inclusive of letters, digits, symbols, and other characters. These streams are represented by a way of training that ends with the phrase "Reader" or "Writer" of their names, inclusive of FileReader, BufferedReader, FileWriter, and BufferedWriter.

Character streams offer a convenient manner to read and write textual content-primarily based information due to the fact they mechanically manage character encoding and decoding. They convert the individual statistics to and from the underlying byte circulation the usage of a particular individual encoding, such as UTF-eight or ASCII.It makes person streams suitable for operating with textual content files, analyzing and writing strings, and processing human-readable statistics. Play

2. Byte Streams:

Byte streams are designed to deal with raw binary data, which includes all kinds of data, including characters, pictures, audio, and video. These streams are represented through classes that cease with the word "InputStream" or "OutputStream" of their names,along with FileInputStream,BufferedInputStream, FileOutputStream and BufferedOutputStream.

Byte streams offer a low-stage interface for studying and writing character bytes or blocks of bytes. They are normally used for coping with non-textual statistics, studying and writing files of their binary form, and running with network sockets. Byte streams don't perform any individual encoding or deciphering. They treat the data as a sequence of bytes and don't interpret it as characters.

FileInputStream

- 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.
- package com.javatpoint;

Real all the Characters

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=0;
          while((i=fin.read())!=-1){
           System.out.print((char)i);
          }
          fin.close();
        }catch(Exception e){System.out.println(e);}
       }
```

}

FileOutputStream

- 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.

BufferedInputStrem

- Java BufferedInputStream class is used to read information from stream.
- It internally uses buffer mechanism to make the performance fast.
- The important points about BufferedInputStream are:
- When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
- When a BufferedInputStream is created, an internal buffer array is created.

```
public class BufferedInputStream extends FilterInputStream
```

Java BufferedInputStream class constructors

Constructor	Description
BufferedInputStream(InputStream IS)	It creates the BufferedInputStream and saves it argument, the input stream IS, for later use.
BufferedInputStream(InputStream IS, int size)	It creates the BufferedInputStream with a specified buffer size and saves it argument, the input stream IS, for later use.

Example:

```
import java.io.*;
public class BufferedInputStreamExample{
 public static void main(String args[]){
  try{
    FileInputStream fin=new FileInputStream("D:\\testout.txt");
    BufferedInputStream bin=new BufferedInputStream(fin);
    int i;
```

```
   while((i=bin.read())!=-1){
    System.out.print((char)i);
   }
   bin.close();
   fin.close();
  }catch(Exception e){System.out.println(e);}
 }
}
```

Java BufferedOutputStream Class

Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

For adding the buffer in an OutputStream, use the BufferedOutputStream class. Let's see the syntax for adding the buffer in an OutputStream:

```
OutputStream os= new BufferedOutputStream(new FileOutputStream("D:\\IO Package\\testout.txt"));
```

Java BufferedOutputStream class declaration

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

```
public class BufferedOutputStream extends FilterOutputStream
```

Java BufferedOutputStream class constructors

Constructor	Description
BufferedOutputStream(OutputStream os)	It creates the new buffered output stream which is used for writing the data to the specified output stream.
BufferedOutputStream(OutputStream os, int size)	It creates the new buffered output stream which is used for writing the data to the specified output stream with a specified buffer size.

```
import java.io.*;
public class BufferedOutputStreamExample{
public static void main(String args[])throws Exception{
    FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
    BufferedOutputStream bout=new BufferedOutputStream(fout);
    String s="Welcome to javaTpoint.";
    byte b[]=s.getBytes();
    bout.write(b);
    bout.flush();
    bout.close();
    fout.close();
    System.out.println("success");
}
}
```

Java DataInputStream Class

Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataInputStream class declaration

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

> **public class** DataInputStream **extends** FilterInputStream **implements** DataInput

```
package com.javatpoint;
import java.io.*;
public class DataStreamExample {
 public static void main(String[] args) throws IOException {
  InputStream input = new FileInputStream("D:\\testout.txt");
  DataInputStream inst = new DataInputStream(input);
  int count = input.available();
  byte[] ary = new byte[count];
  inst.read(ary);
  for (byte bt : ary) {
   char k = (char) bt;
   System.out.print(k+"-");
  }
 }
}
```

Java DataOutputStream Class

Java DataOutputStream class allows an application to write primitive Java data types to the output stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataOutputStream class declaration

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

> **public class** DataOutputStream **extends** FilterOutputStream **implements** DataOutput

```
import java.io.*;
public class OutputExample {
   public static void main(String[] args) throws IOException {
      FileOutputStream file = new FileOutputStream(D:\\testout.txt);
      DataOutputStream data = new DataOutputStream(file);
      data.writeInt(65);
      data.flush();
      data.close();
      System.out.println("Succcess...");
   }
}
```

Category 1: Byte stream programs (Deals with binary data)
Category 2: Character stream programs (deals with character data).

Category 1: Byte stream programs (Deals with binary data)

Important classes related to file handling operations
i) FileInputStream class : contains methods to read binary data from the files
For example: **read()** method is in FileInputStream class.
To use read() method first create object for the FileInputStream class

FileInputStream fin=new FileInputStream("sample.txt");

Now read() method can be called as fin.read() which reads one byte at time from the file saple.txt.

ii) FileOutStream class : contains methods to read binary data from the files
For example: **write()** method is in FileOutputStream class.
To use write() method first create object for the FileInputStream class

FileOutputStream fin=new FileOutputStream("newfile.txt");

Now write() method can be called as fin.write() which writes one byte at time to the file newfile.txt

Note: while dealing binary files => the last byte in the file is -1.

Write a java program to copy the contents of one file to another file

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
class CopyFile
{
   public static void main(String args[])
   {
      FileInputStream fin = null;
      FileOutputStream fout = null;
      try
      {
         fin = new FileInputStream("sample.txt");
```

```
        fout = new FileOutputStream("demo.txt");
        int c;
        while ((c = fin.read()) != -1) //reading each byte from the file sample.txt
        {
           fout.write(c);   //writeing each byte to the newly created file demo.txt
        }
        System.out.println("File copied successfully.");
     }
     catch (IOException e)
     {
        System.err.println("Error during file operation: " + e.getMessage());
     }
  }
}
```

Output:

```
C:\Users\anilk\OneDrive\Desktop\files>javac CopyFile.java

C:\Users\anilk\OneDrive\Desktop\files>java CopyFile
File copied successfully.

C:\Users\anilk\OneDrive\Desktop\files>type sample.txt
hi this is a sample file.

C:\Users\anilk\OneDrive\Desktop\files>type demo.txt
hi this is a sample file.
```

//**java program to display the file contents on the screen**

```
import java.io.FileInputStream;
import java.io.IOException;

class displayfilecontent
{
   public static void main(String args[])
   {
      FileInputStream fin = null;
      try
      {
         fin = new FileInputStream("sample.txt");
         int c;
         while ((c = fin.read()) != -1) //reading each byte from the file sample.txt
         {
```

```
            System.out.print((char)c);
        }
      }
      catch (IOException e)
      {
         System.err.println("Error during file operation: " + e.getMessage());
      }
   }
}
```

Output:

```
C:\Users\anilk\OneDrive\Desktop\files>javac displayfilecontent.

C:\Users\anilk\OneDrive\Desktop\files>java displayfilecontent
hi this is a sample file.

C:\Users\anilk\OneDrive\Desktop\files>type sample.txt
hi this is a sample file.
```

Category 2: Character stream programs (deals with character data).

- The java.io package provides CharacterStream classes to overcome the limitations of ByteStream classes, which can only handle the 8-bit bytes and is not compatible to work directly with the Unicode characters.
- CharacterStream classes are used to work with 16-bit Unicode characters.
- They can perform operations on characters, char arrays and Strings.

Important classes to handle character data in files
FileReader: To read 16 bit unicode characters from the file
read() used to read integer data from the file
FileWriter: To write 16 bit unicode character to the file
write() used to write integer data to the file

//java program to copy the contents of one file to another file

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

class copycharfiles {
   public static void main(String args[]) {
      FileReader fr = null;
```

```
        FileWriter fw = null;

        try {
            fr = new FileReader("sample.txt");  // Reading from sample.txt
            fw = new FileWriter("demo123.txt");    // Writing to demo.txt

            int c;  // Use int to properly handle EOF (-1)
            while ((c = fr.read()) != -1) {  // Reading each character
                fw.write(c);  // Writing each character to demo.txt
            }
            System.out.println("File copied successfully.");
        }
        catch (IOException e) {
            System.err.println("Error during file operation: " + e.getMessage());
        }
    }
}
```

Output:

```
C:\Users\anilk\OneDrive\Desktop\files>javac copycharfiles.jav

C:\Users\anilk\OneDrive\Desktop\files>java copycharfiles
File copied successfully.

C:\Users\anilk\OneDrive\Desktop\files>type sample.txt
hi this is a sample file.

C:\Users\anilk\OneDrive\Desktop\files>type demo123.txt
hi this is a sample file.
```

Here are the some of the differences listed:

Aspect	Character Streams	Byte Streams
Data Handling	Handle character-based data	Handle raw binary data

Representation	Classes end with "Reader" or "Writer"	Classes end with "InputStream" or "OutputStream"
Suitable for	Textual data, strings, human-readable info	Non-textual data, binary files, multimedia
Character Encoding	Automatic encoding and decoding	No encoding or decoding
Text vs non-Text data	Text-based data, strings	Binary data, images, audio, video
Performance	Additional conversion may impact performance	Efficient for handling large binary data
Handle Large Text Files	May impact performance due to encoding	Efficient, no encoding overhead
String Operations	Convenient methods for string operations	Not specifically designed for string operations
Convenience Methods	Higher-level abstractions for text data	Low-level interface for byte data
Reading Line by Line	Convenient methods for reading lines	Byte-oriented, no built-in line-reading methods
File Handling	Read/write text files	Read/write binary files
Network Communication	Sending/receiving text data	Sending/receiving binary data

Handling Images/Audio/Video	Not designed for handling binary data directly	Suitable for handling binary multimedia data
Text Encoding	Supports various character encodings	No specific text encoding support