Java I/O Tutorial

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.

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

Do You Know?

- How to write a common data to multiple files using a single stream only?
- How can we access multiple files by a single stream?
- How can we improve the performance of Input and Output operation?
- How many ways can we read data from the keyboard?
- What does the console class?
- How to compress and uncompress the data of a file?

OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

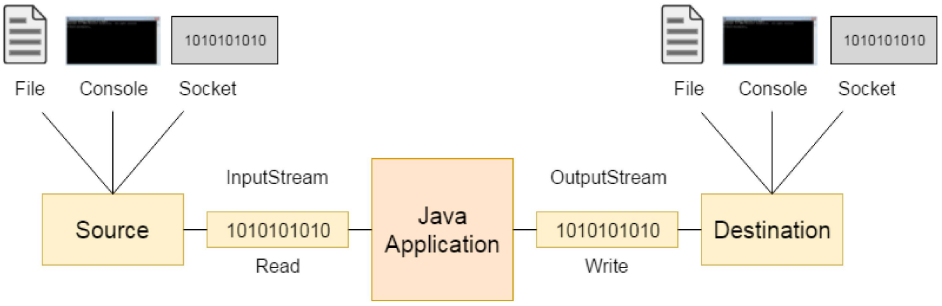
OutputStream

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.



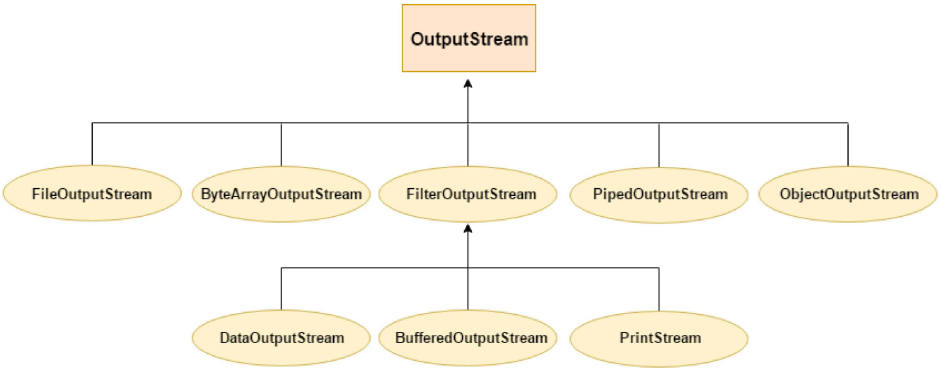
OutputStream class

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



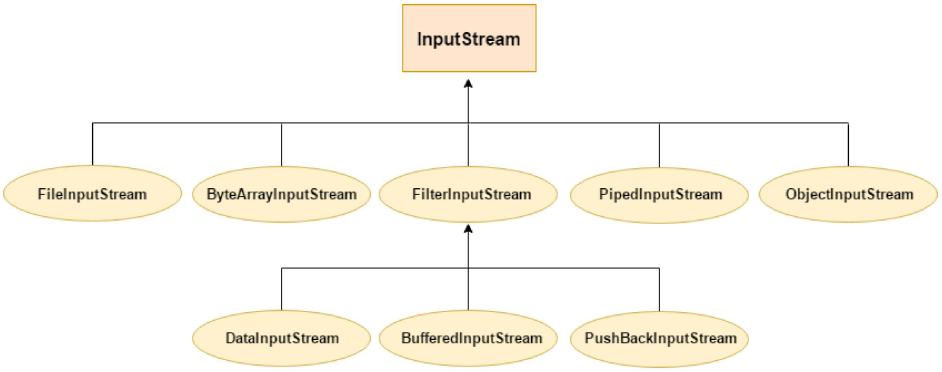
InputStream class

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 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.

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:

```
5. 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

```
6.  import java.io.FileOutputStream;
7.  public class FileOutputStreamExample {
8.      public static void main(String args[]){
9.          try{
10.           FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
11.           fout.write(65);
12.           fout.close();
13.           System.out.println("success...");
14.          }catch(Exception e){System.out.println(e);}
15.     }
16. }
```

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

```
17. import java.io.FileOutputStream;
18. public class FileOutputStreamExample {
19.     public static void main(String args[]){
20.         try{
21.           FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
22.           String s="Welcome to javaTpoint.";
23.           byte b[]=s.getBytes();//converting string into byte array
24.           fout.write(b);
25.           fout.close();
26.           System.out.println("success...");
27.          }catch(Exception e){System.out.println(e);}
28.     }
29. }
```

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:

```
30. 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

```
31. import java.io.FileInputStream;
32. public class DataStreamExample {
33.     public static void main(String args[]){
34.         try{
35.           FileInputStream fin=new FileInputStream("D:\\testout.txt");
36.           int i=fin.read();
37.           System.out.print((char)i);
38.
39.           fin.close();
40.         }catch(Exception e){System.out.println(e);}
41.        }
42.       }
```

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

```
43. package com.javatpoint;
44.
45. import java.io.FileInputStream;
46. public class DataStreamExample {
47.     public static void main(String args[]){
```

```
48.         try{
49.           FileInputStream fin=new FileInputStream("D:\\testout.txt");
50.           int i=0;
51.           while((i=fin.read())!=-1){
52.            System.out.print((char)i);
53.           }
54.           fin.close();
55.          }catch(Exception e){System.out.println(e);}
56.         }
57.        }
```

Output:

```
Welcome to javaTpoint
```

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:

```
58. 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:

```
59. 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.

Java BufferedOutputStream class methods

Method	Description
void write(int b)	It writes the specified byte to the buffered output stream.
void write(byte[] b, int off, int len)	It write the bytes from the specified byte-input stream into a specified byte array, starting with the given offset
void flush()	It flushes the buffered output stream.

Example of BufferedOutputStream class:

In this example, we are writing the textual information in the BufferedOutputStream object which is connected to the FileOutputStream object. The flush() flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.

```
60. package com.javatpoint;
61. import java.io.*;
62. public class BufferedOutputStreamExample{
63. public static void main(String args[])throws Exception{
64.     FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
65.     BufferedOutputStream bout=new BufferedOutputStream(fout);
66.     String s="Welcome to javaTpoint.";
67.     byte b[]=s.getBytes();
68.     bout.write(b);
69.     bout.flush();
```

```
70.     bout.close();
71.     fout.close();
72.     System.out.println("success");
73. }
74. }
```

Output:

```
Success
```

testout.txt

```
Welcome to javaTpoint.
```

Java BufferedInputStream Class

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.

Java BufferedInputStream class declaration

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

```
75. 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.

Java BufferedInputStream class methods

Method	Description
int available()	It returns an estimate number of bytes that can be read from the input stream without blocking by the next invocation method for the input stream.
int read()	It read the next byte of data from the input stream.
int read(byte[] b, int off, int ln)	It read the bytes from the specified byte-input stream into a specified byte array, starting with the given offset.
void close()	It closes the input stream and releases any of the system resources associated with the stream.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readlimit)	It sees the general contract of the mark method for the input stream.
long skip(long x)	It skips over and discards x bytes of data from the input stream.
boolean markSupported()	It tests for the input stream to support the mark and reset methods.

Example of Java BufferedInputStream

Let's see the simple example to read data of file using BufferedInputStream:

```
76. package com.javatpoint;
77.
78. import java.io.*;
79. public class BufferedInputStreamExample{
80.  public static void main(String args[]){
81.   try{
82.     FileInputStream fin=new FileInputStream("D:\\testout.txt");
83.     BufferedInputStream bin=new BufferedInputStream(fin);
84.     int i;
85.     while((i=bin.read())!=-1){
86.      System.out.print((char)i);
87.     }
88.     bin.close();
89.     fin.close();
90.   }catch(Exception e){System.out.println(e);}
91.  }
92. }
```

Here, we are assuming that you have following data in **"testout.txt"** file:

```
javaTpoint
```

Output:

```
javaTpoint
```

Java FileWriter Class

Java FileWriter class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.

Unlike FileOutputStream class, you don't need to convert string into byte array because it provides method to write string directly.

Java FileWriter class declaration

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

```
93. public class FileWriter extends OutputStreamWriter
```

Constructors of FileWriter class

Constructor	Description
FileWriter(String file)	Creates a new file. It gets file name in string.
FileWriter(File file)	Creates a new file. It gets file name in File object.

Methods of FileWriter class

Method	Description
void write(String text)	It is used to write the string into FileWriter.
void write(char c)	It is used to write the char into FileWriter.
void write(char[] c)	It is used to write char array into FileWriter.
void flush()	It is used to flushes the data of FileWriter.
void close()	It is used to close the FileWriter.

Java FileWriter Example

In this example, we are writing the data in the file testout.txt using Java FileWriter class.

```
94. package com.javatpoint;
95. import java.io.FileWriter;
96. public class FileWriterExample {
97.     public static void main(String args[]){
98.         try{
99.           FileWriter fw=new FileWriter("D:\\testout.txt");
100.                 fw.write("Welcome to javaTpoint.");
101.                 fw.close();
```

```
102.                }catch(Exception e){System.out.println(e);}
103.                System.out.println("Success...");
104.            }
105.        }
```

Output:

```
Success...
```

testout.txt:

```
Welcome to javaTpoint.
```

Java FileReader Class

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

It is character-oriented class which is used for file handling in java.

Java FileReader class declaration

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

```
106.        public class FileReader extends InputStreamReader
```

Constructors of FileReader class

Constructor	Description
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Methods of FileReader class

Method	Description
int read()	It is used to return a character in ASCII form. It returns -1 at the end of file.
void close()	It is used to close the FileReader class.

Java FileReader Example

In this example, we are reading the data from the text file **testout.txt** using Java FileReader class.

```
107.    package com.javatpoint;
108.
109.    import java.io.FileReader;
110.    public class FileReaderExample {
111.       public static void main(String args[])throws Exception{
112.            FileReader fr=new FileReader("D:\\testout.txt");
113.            int i;
114.            while((i=fr.read())!=-1)
115.            System.out.print((char)i);
116.            fr.close();
117.       }
118.    }
```

Here, we are assuming that you have following data in "testout.txt" file:

```
Welcome to javaTpoint.
```

Output:

```
Welcome to javaTpoint.
```