

Java I/O

Java I/O (Input and Output) is used to process the input and produce the output based on the input. Java uses the concept of 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 IO API.

Stream

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

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

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

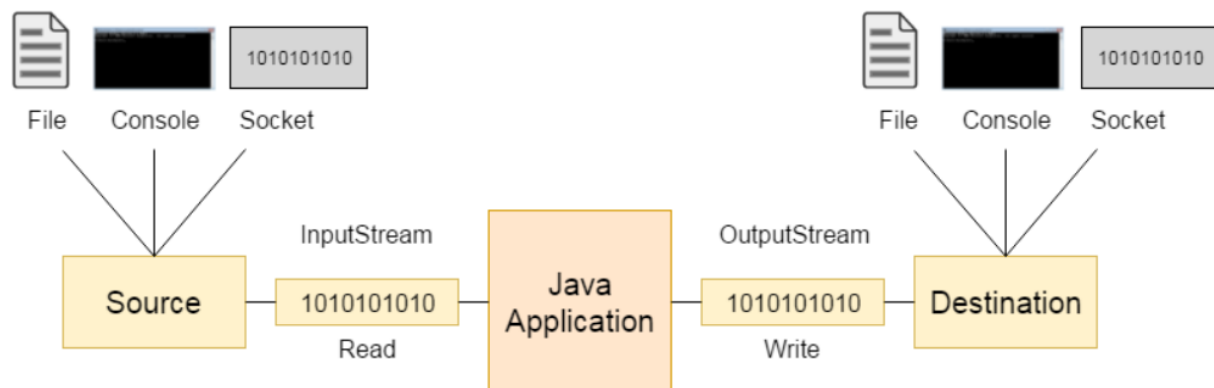
OutputStream

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 working of Java **OutputStream** and **InputStream** by the figure given below.



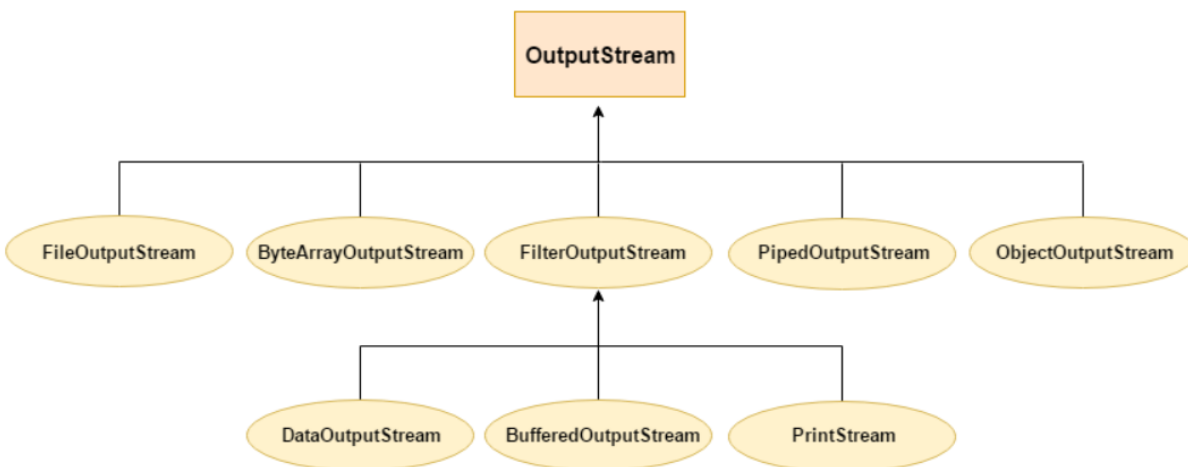
OutputStream class

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.

Commonly used methods of OutputStream class

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 :



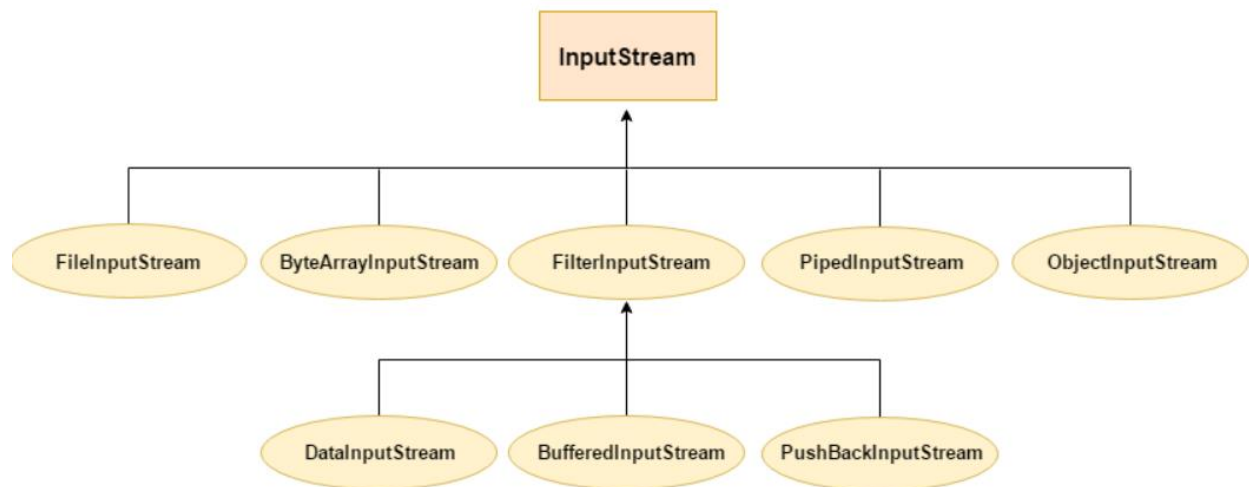
InputStream class

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

Commonly used methods of InputStream class

1. `public abstract int read()` throws `IOException`: reads the next byte of data from the input stream. It returns -1 at the end of 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 :



File Handling

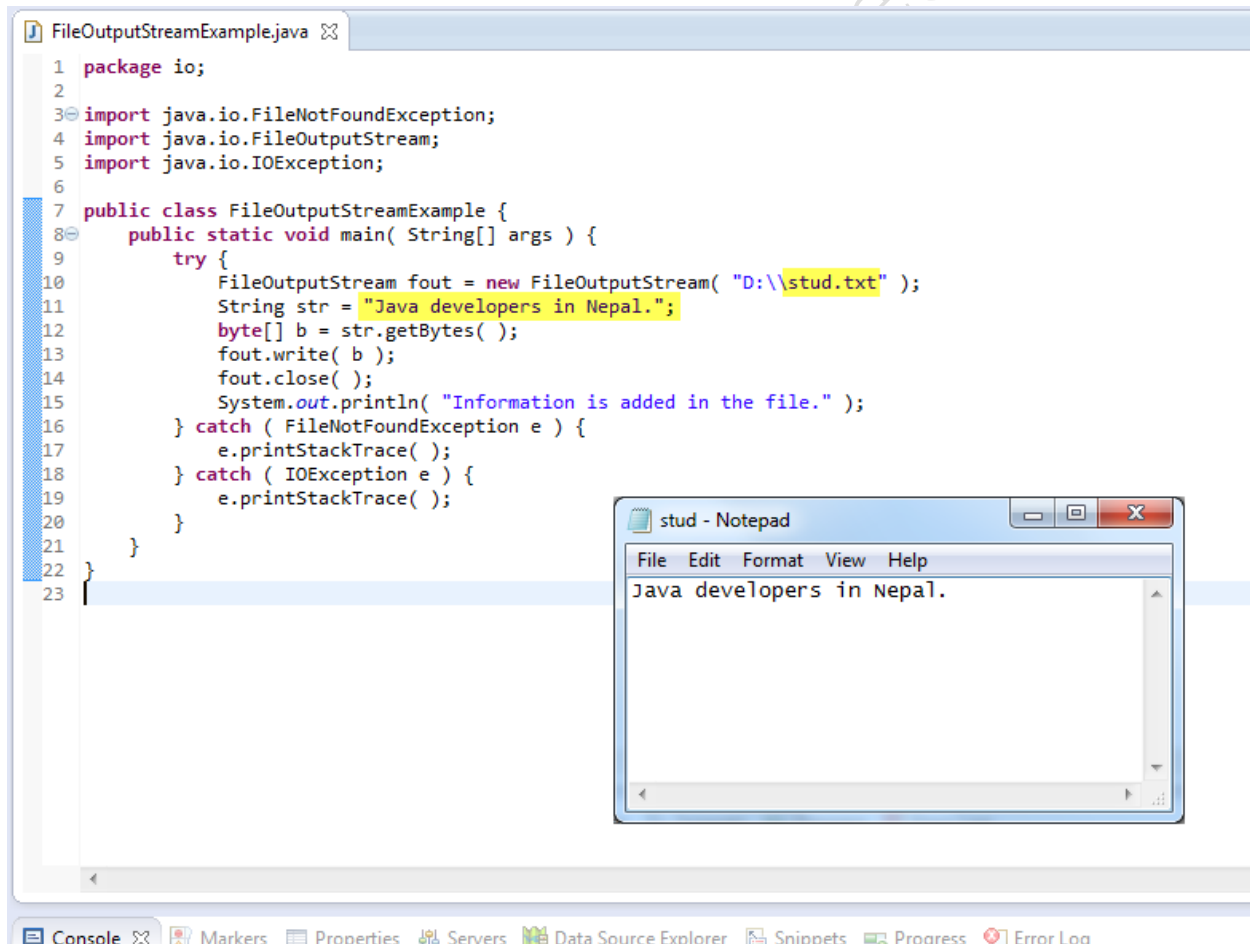
FileInputStream and FileOutputStream

In Java, FileInputStream and FileOutputStream classes are used to read and write data in file. In another words, they are used for file handling in java.

Java FileOutputStream class

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

If you have to write primitive values then use FileOutputStream. Instead, for character-oriented data, prefer FileWriter. But you can write byte-oriented as well as character-oriented data.



Java FileInputStream class

Java FileInputStream class obtains input bytes from a file. It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.

It should be used to read byte-oriented data for example to read image, audio, video etc.

```
FileStreamReadOneWriteToOther.java
1 package io;
2
3 import java.io.FileInputStream;
4 import java.io.FileOutputStream;
5
6 /**
7  * Copy content of one file to another.
8  * @author ojhay
9  */
10 public class FileStreamReadOneWriteToOther {
11     public static void main( String[] args ) {
12
13         try {
14             FileInputStream fin = new FileInputStream( "D:\\stud.txt" );
15             FileOutputStream fout = new FileOutputStream( "D:\\stud2.txt" );
16             int i = 0;
17             while ( ( i = fin.read( ) ) != -1 ) {
18                 System.out.print( ( char ) i );
19                 fout.write( ( char ) i );
20             }
21             fin.close( );
22             fout.close( );
23
24         } catch ( Exception e ) {
25             e.printStackTrace( );
26         }
27     }
28 }
29
```

Java BufferedOutputStream and BufferedInputStream

Java BufferedOutputStream class

Java BufferedOutputStream class uses an internal buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

```
BufferedOutputStreamExample.java
1 package io;
2
3 import java.io.BufferedOutputStream;
4 import java.io.FileOutputStream;
5
6 public class BufferedOutputStreamExample {
7     public static void main( String[] args ) {
8         try {
9             FileOutputStream fout = new FileOutputStream( "D:\\bstud.txt" );
10            BufferedOutputStream bout = new BufferedOutputStream( fout );
11
12            String str = "Java developers of Nepal.\nRunning Chapter: File Handling, BufferedOutputStream";
13            byte[] b = str.getBytes( );
14
15            bout.write( b );
16            bout.flush( );
17
18            bout.close( );
19            fout.close( );
20
21            System.out.println( "Information is added in the file." );
22
23        } catch ( Exception e ) {
24            e.printStackTrace( );
25        }
26    }
27 }
28
```

Java BufferedInputStream class

Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

```
BufferedInputStreamExample.java
1 package io;
2
3 import java.io.BufferedInputStream;
4 import java.io.FileInputStream;
5
6 public class BufferedInputStreamExample {
7     public static void main( String args[] ) {
8         try {
9
10             FileInputStream fin = new FileInputStream( "D:\\bstud.txt" );
11             BufferedInputStream bin = new BufferedInputStream( fin );
12
13             int i;
14             while ( ( i = bin.read() ) != -1 ) {
15                 System.out.println( ( char ) i );
16             }
17
18             bin.close();
19             fin.close();
20
21         } catch ( Exception e ) {
22             e.printStackTrace();
23         }
24     }
25 }
26
```

Java FileWriter and FileReader (File Handling in java)

Java FileWriter and FileReader classes are used to write and read data from text files. These are character-oriented classes, used for file handling in java.

Java has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.

```

1 package io;
2
3 import java.io.FileWriter;
4
5 public class FileWriterExample {
6     public static void main( String args[] ) {
7         try {
8
9             FileWriter fw = new FileWriter( "D:\\fstud.txt" );
10
11             fw.write( "Java developers of नेपाल.\nRunning Chapter: File Handling, FileWriter." );
12
13             fw.close( );
14
15         } catch ( Exception e ) {
16             e.printStackTrace( );
17
18             System.out.println( "Information is added in the file." );
19         }
20     }
21 }

```

Make sure you have saved source file as UTF-8

```

1 package io;
2
3 import java.io.FileReader;
4
5 public class FileReaderExample {
6     public static void main( String args[] ) throws Exception {
7
8         FileReader fr = new FileReader( "D:\\fstud.txt" );
9         int i;
10
11         while ( ( i = fr.read( ) ) != -1 ) {
12
13             System.out.print( ( char ) i );
14         }
15
16         fr.close( );
17     }
18 }
19

```

Make sure you have saved source file as UTF-8

Console Markers Properties Servers Data Source Explorer Snippets Progi

<terminated> FileReaderExample [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 20,
 Java developers of नेपाल.
 Running Chapter: File Handling, FileWriter.

Reading data from keyboard

There are many ways to read data from the keyboard. For example:

- InputStreamReader
- Console
- Scanner
- DataInputStream etc.

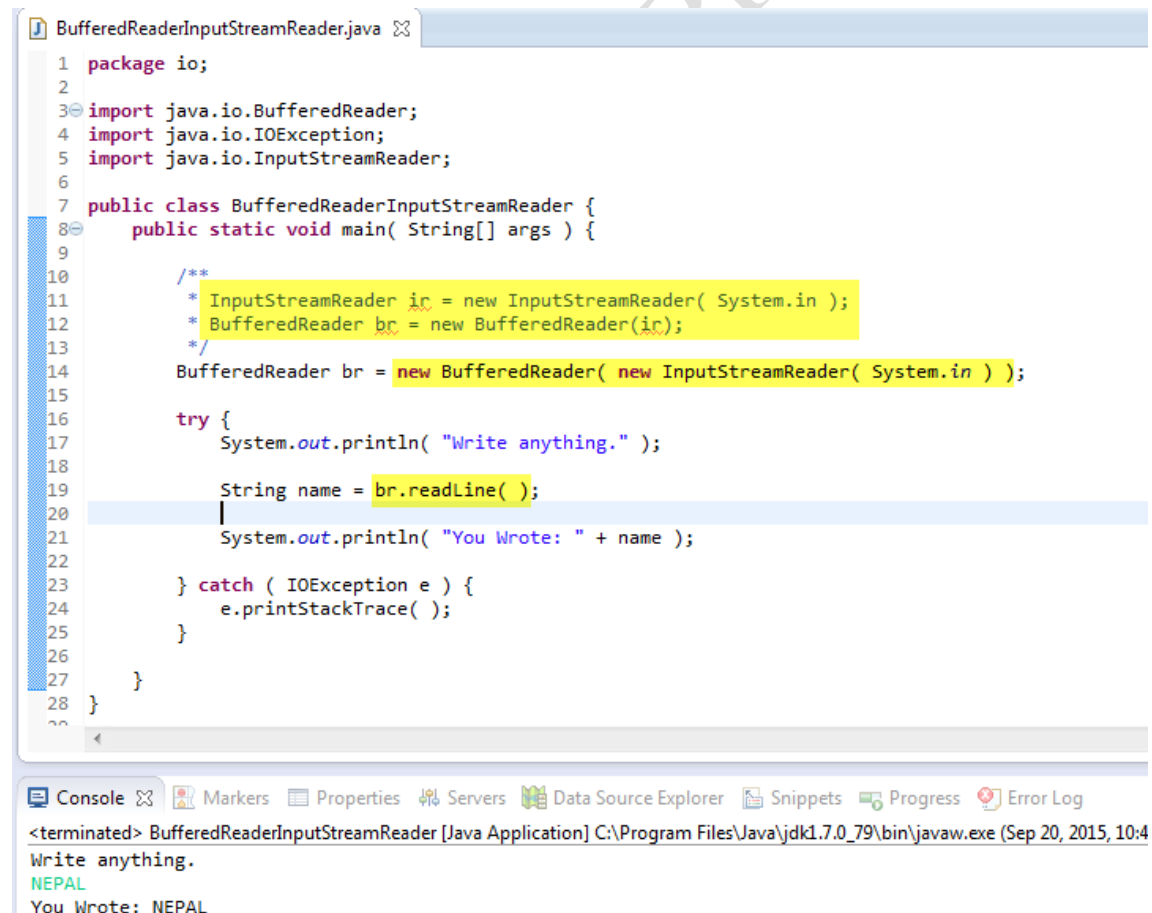
InputStreamReader class

InputStreamReader class can be used to read data from keyboard. It performs two tasks:

- connects to input stream of keyboard
- converts the byte-oriented stream into character-oriented stream

BufferedReader class

BufferedReader class can be used to read data line by line by readLine() method. reading data from keyboard by InputStreamReader and BufferdReader class



```

1 package io;
2
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.io.InputStreamReader;
6
7 public class BufferedReaderInputStreamReader {
8     public static void main( String[] args ) {
9
10         /**
11          * InputStreamReader ic = new InputStreamReader( System.in );
12          * BufferedReader br = new BufferedReader(ic);
13          */
14         BufferedReader br = new BufferedReader( new InputStreamReader( System.in ) );
15
16         try {
17             System.out.println( "Write anything." );
18
19             String name = br.readLine( );
20             System.out.println( "You Wrote: " + name );
21
22         } catch ( IOException e ) {
23             e.printStackTrace( );
24         }
25     }
26 }
27
28 }

```

Console Output:

```

<terminated> BufferedReaderInputStreamReader [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 20, 2015, 10:4
Write anything.
NEPAL
You Wrote: NEPAL

```

File Class:

Using the File class you can:

- Check if a file or directory exists.
- Create a directory if it does not exist.
- Read the length of a file.
- Rename or move a file.
- Delete a file.
- Check if path is file or directory.
- Read list of files in a directory.

```
FileExample.java
1 package io;
2
3 import java.io.File;
4
5 public class FileExample {
6
7     @SuppressWarnings( "unused" )
8     public static void main( String[] args ) {
9
10         /* Instantiate File */
11         File file = new File( "D:\\ab.txt" );
12
13         // Check if above path is Directory?
14         boolean isDirectory = file.isDirectory( );
15
16         /* File exist or not */
17         boolean isFileExists = file.exists( );
18
19         /* Create Directory */
20         File fDir = new File( "D:\\temp" );
21         boolean dirCreated = fDir.mkdir( );
22
23         /* List name of all files */
24         String[] fileNames = file.list( );
25
26         /* List of Files */
27         File[] files = file.listFiles( );
28
29     }
30 }
31
```