**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.

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:

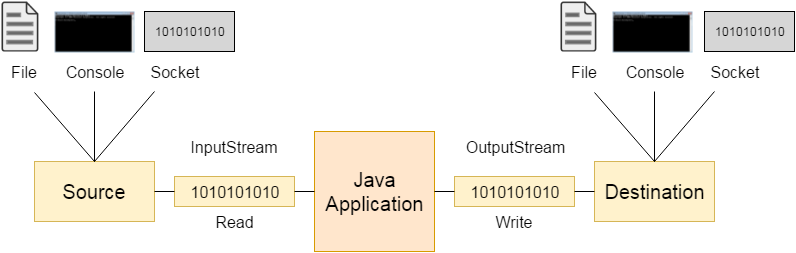
**OutputStream**

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.



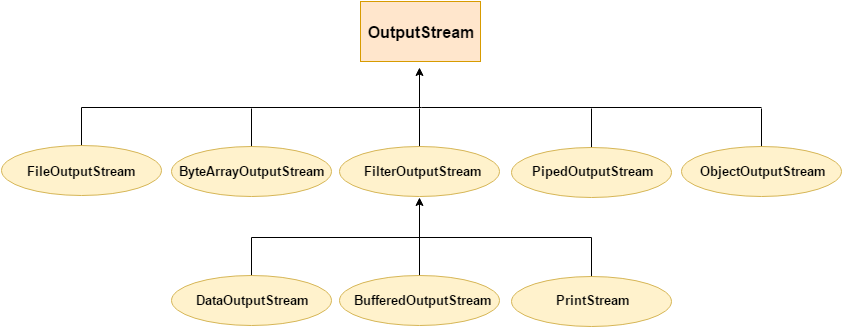
**OutputStream class**

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**



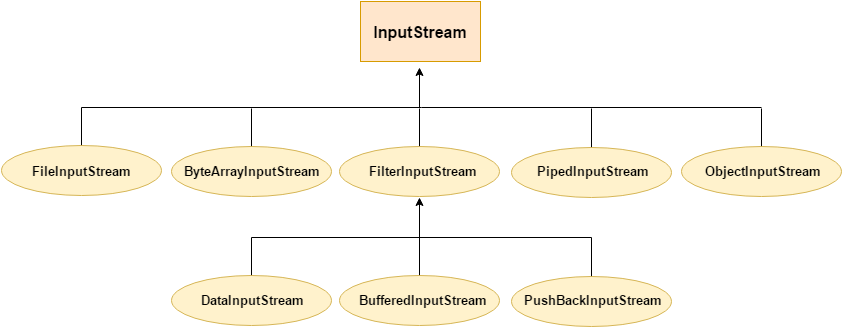
**InputStream class**

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 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](https://www.javatpoint.com/java-file-class).

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](https://www.javatpoint.com/java-filterwriter-class) 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](https://www.javatpoint.com/array-in-java) 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**

import java.io.FileOutputStream;

public class FileOutputStreamExample {

    public static void main(String args[]){

           try{

             FileOutputStream fout=new FileOutputStream("D:\\testout.txt");

             fout.write(65);

             fout.close();

             System.out.println("success...");

            }catch(Exception e){System.out.println(e);}

      }

}

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**

import java.io.FileOutputStream;

public class FileOutputStreamExample {

    public static void main(String args[]){

           try{

             FileOutputStream fout=new FileOutputStream("D:\\testout.txt");

             String s="Welcome to VIT.";

             byte b[]=s.getBytes();//converting string into byte array

             fout.write(b);

             fout.close();

             System.out.println("success...");

            }catch(Exception e){System.out.println(e);}

      }

}

Output:

Success...

The content of a text file **testout.txt** is set with the data **Welcome to VIT.**

testout.txt

Welcome to VIT.

**Java FileInputStream Class**

Java FileInputStream class obtains input bytes from a [file](https://www.javatpoint.com/java-file-class). 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](https://www.javatpoint.com/java-filereader-class) 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](https://www.javatpoint.com/java-filedescriptor-class) 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](https://www.javatpoint.com/java-8-stream). |

**Java FileInputStream example 1: read single character**

import java.io.FileInputStream;

public class DataStreamExample {

     public static void main(String args[]){

          try{

            FileInputStream fin=new FileInputStream("D:\\testout.txt");

            int i=fin.read();

            System.out.print((char)i);

            fin.close();

          }catch(Exception e){System.out.println(e);}

         }

        }

**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 VIT.

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**

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);}

         }

        }

Output:

Welcome to VIT

**Java File Class**

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

**Fields**

|  |  |  |  |
| --- | --- | --- | --- |
| **Modifier** | **Type** | **Field** | **Description** |
| static | String | pathSeparator | It is system-dependent path-separator character, represented as a [string](https://www.javatpoint.com/java-string) for convenience. |
| static | char | pathSeparatorChar | It is system-dependent path-separator character. |
| static | String | separator | It is system-dependent default name-separator character, represented as a string for convenience. |
| static | char | separatorChar | It is system-dependent default name-separator character. |

[**Constructors**](https://www.javatpoint.com/java-constructor)

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| File(File parent, String child) | It creates a new File instance from a parent abstract pathname and a child pathname string. |
| File(String pathname) | It creates a new File instance by converting the given pathname string into an abstract pathname. |
| File(String parent, String child) | It creates a new File instance from a parent pathname string and a child pathname string. |
| File(URI uri) | It creates a new File instance by converting the given file: URI into an abstract pathname. |

**Useful Methods**

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| static File | createTempFile(String prefix, String suffix) | It creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name. |
| Boolean | createNewFile() | It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. |
| Boolean | canWrite() | It tests whether the application can modify the file denoted by this abstract pathname.String[] |
| Boolean | canExecute() | It tests whether the application can execute the file denoted by this abstract pathname. |
| Boolean | canRead() | It tests whether the application can read the file denoted by this abstract pathname. |
| Boolean | isAbsolute() | It tests whether this abstract pathname is absolute. |
| Boolean | isDirectory() | It tests whether the file denoted by this abstract pathname is a directory. |
| Boolean | isFile() | It tests whether the file denoted by this abstract pathname is a normal file. |
| String | getName() | It returns the name of the file or directory denoted by this abstract pathname. |
| String | getParent() | It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory. |
| Path | toPath() | It returns a java.nio.file.Path object constructed from the this abstract path. |
| URI | toURI() | It constructs a file: URI that represents this abstract pathname. |
| File[] | listFiles() | It returns an [array](https://www.javatpoint.com/array-in-java) of abstract pathnames denoting the files in the directory denoted by this abstract pathname |
| Long | getFreeSpace() | It returns the number of unallocated bytes in the partition named by this abstract path name. |
| String[] | list(FilenameFilter filter) | It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter. |
| Boolean | mkdir() | It creates the directory named by this abstract pathname. |

**Java File Example 1**

import java.io.\*;

public class FileDemo {

    public static void main(String[] args) {

        try {

            File file = new File("javaFile123.txt");

            if (file.createNewFile()) {

                System.out.println("New File is created!");

            } else {

                System.out.println("File already exists.");

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

Output:

New File is created!

**Java File Example 2**

1. import java.io.\*;
2. public class FileDemo2 {
3. public static void main(String[] args) {
5. String path = "";
6. boolean bool = false;
7. try {
8. // createing  new files
9. File file  = new File("testFile1.txt");
10. file.createNewFile();
11. System.out.println(file);
12. // createing new canonical from file object
13. File file2 = file.getCanonicalFile();
14. // returns true if the file exists
15. System.out.println(file2);
16. bool = file2.exists();
17. // returns absolute pathname
18. path = file2.getAbsolutePath();
19. System.out.println(bool);
20. // if file exists
21. if (bool) {
22. // prints
23. System.out.print(path + " Exists? " + bool);
24. }
25. } catch (Exception e) {
26. // if any error occurs
27. e.printStackTrace();
28. }
29. }
30. }

Output:

testFile1.txt

/home/Work/Project/File/testFile1.txt

true

/home/Work/Project/File/testFile1.txt Exists? true

**Java File Example 3**

import java.io.\*;

public class FileExample {

public static void main(String[] args) {

    File f=new File("/Users/sonoojaiswal/Documents");

    String filenames[]=f.list();

    for(String filename:filenames){

        System.out.println(filename);

    }

}

}

Output:

"info.properties"

"info.properties".rtf

.DS\_Store

.localized

Alok news

apache-tomcat-9.0.0.M19

apache-tomcat-9.0.0.M19.tar

bestreturn\_org.rtf

BIODATA.pages

BIODATA.pdf

BIODATA.png

struts2jars.zip

workspace

**Java File Example 4**

import java.io.\*;

public class FileExample {

public static void main(String[] args) {

    File dir=new File("/Users/sonoojaiswal/Documents");

    File files[]=dir.listFiles();

    for(File file:files){

        System.out.println(file.getName()+" Can Write: "+file.canWrite()+"

        Is Hidden: "+file.isHidden()+" Length: "+file.length()+" bytes");

    }

}

}

Output:

"info.properties" Can Write: true Is Hidden: false Length: 15 bytes

"info.properties".rtf Can Write: true Is Hidden: false Length: 385 bytes

.DS\_Store Can Write: true Is Hidden: true Length: 36868 bytes

.localized Can Write: true Is Hidden: true Length: 0 bytes

Alok news Can Write: true Is Hidden: false Length: 850 bytes

apache-tomcat-9.0.0.M19 Can Write: true Is Hidden: false Length: 476 bytes

apache-tomcat-9.0.0.M19.tar Can Write: true Is Hidden: false Length: 13711360 bytes

bestreturn\_org.rtf Can Write: true Is Hidden: false Length: 389 bytes

BIODATA.pages Can Write: true Is Hidden: false Length: 707985 bytes

BIODATA.pdf Can Write: true Is Hidden: false Length: 69681 bytes

BIODATA.png Can Write: true Is Hidden: false Length: 282125 bytes

workspace Can Write: true Is Hidden: false Length: 1972 bytes