Java Io

**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
3. The explanation of OutputStream and InputStream classes are given below:

### OutputStream

1. Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

### InputStream

1. Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.
2. Let's understand the working of Java OutputStream and InputStream by the figure given below.
3. Java IO

## 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

Java output stream 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 input stream 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:

## 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

1. **import** java.io.FileOutputStream;
2. **public** **class** FileOutputStreamExample {
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt");
6. fout.write(65);
7. fout.close();
8. System.out.println("success...");
9. }**catch**(Exception e){System.out.println(e);}

      }

}

Output:

Success...

## Java FileOutputStream example 2: write string

1. **import** java.io.FileOutputStream;
2. **public** **class** FileOutputStreamExample {
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt");
6. String s="Welcome to javaTpoint.";
7. **byte** b[]=s.getBytes();//converting string into byte array
8. fout.write(b);
9. fout.close();
10. System.out.println("success...");
11. }**catch**(Exception e){System.out.println(e);}
12. }
13. }

Output:

Success...

# 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

1. **import** java.io.FileInputStream;
2. **public** **class** DataStreamExample {
3. **public** **static** **void** main(String args[]){
4. **try**{
5. FileInputStream fin=**new** FileInputStream("D:\\testout.txt");
6. **int** i=fin.read();
7. System.out.print((**char**)i);
9. fin.close();
10. }**catch**(Exception e){System.out.println(e);}
11. }
12. }

…………………….

## Java FileInputStream example 2: read all characters

3. **import** java.io.FileInputStream;
4. **public** **class** DataStreamExample {
5. **public** **static** **void** main(String args[]){
6. **try**{
7. FileInputStream fin=**new** FileInputStream("D:\\testout.txt");
8. **int** i=0;
9. **while**((i=fin.read())!=-1){
10. System.out.print((**char**)i);
11. }
12. fin.close();
13. }**catch**(Exception e){System.out.println(e);}
14. }
15. }

# Java BufferedOutputStream Class

Java BufferedOutputStream [class](https://www.javatpoint.com/object-and-class-in-java) 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:

1. 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:

## 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](https://www.javatpoint.com/array-in-java), starting with the given offset |
| void flush() | It flushes the buffered output stream. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Java BufferedOutputStream Class**  Java BufferedOutputStream [class](https://www.javatpoint.com/object-and-class-in-java) 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:   1. 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:   1. **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](https://www.javatpoint.com/array-in-java), 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](https://www.javatpoint.com/java-fileoutputstream-class) [object](https://www.javatpoint.com/object-and-class-in-java). 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.    2. **import** java.io.\*; 3. **public** **class** BufferedOutputStreamExample{ 4. **public** **static** **void** main(String args[])**throws** Exception{ 5. FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt"); 6. BufferedOutputStream bout=**new** BufferedOutputStream(fout); 7. String s="Welcome to javaTpoint."; 8. **byte** b[]=s.getBytes(); 9. bout.write(b); 10. bout.flush(); 11. bout.close(); 12. fout.close(); 13. System.out.println("success"); 14. } 15. } |
| [Next →](https://www.javatpoint.com/java-sequenceinputstream-class)[← Prev](https://www.javatpoint.com/java-bufferedoutputstream-class)  Java BufferedInputStream Class  Java BufferedInputStream [class](https://www.javatpoint.com/object-and-class-in-java) is used to read information from [stream](https://www.javatpoint.com/java-8-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](https://www.javatpoint.com/array-in-java) is created.   Java BufferedInputStream class declaration  Let's see the declaration for Java.io.BufferedInputStream class: |

1. **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](https://www.javatpoint.com/java-file-class) using BufferedInputStream:

**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 SequenceInputStream Example

**import** java.io.\*;

**class** InputStreamExample {

**public** **static** **void** main(String args[])**throws** Exception{

   FileInputStream input1=**new** FileInputStream("D:\\testin.txt");

   FileInputStream input2=**new** FileInputStream("D:\\testout.txt");

   SequenceInputStream inst=**new** SequenceInputStream(input1, input2);

**int** j;

**while**((j=inst.read())!=-1){

    System.out.print((**char**)j);

   }

   inst.close();

   input1.close();

   input2.close();

  }

}

# Java ByteArrayOutputStream Class

Java ByteArrayOutputStream class is used to **write common data** into multiple files. In this stream, the data is written into a byte [array](https://www.javatpoint.com/array-in-java) which can be written to multiple streams later.

The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams.

The buffer of ByteArrayOutputStream automatically grows according to data.

## Java ByteArrayOutputStream class declaration

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

1. **public** **class** ByteArrayOutputStream **extends** OutputStream

## Java ByteArrayOutputStream class constructors

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| ByteArrayOutputStream() | Creates a new byte array output [stream](https://www.javatpoint.com/java-8-stream) with the initial capacity of 32 bytes, though its size increases if necessary. |
| ByteArrayOutputStream(int size) | Creates a new byte array output stream, with a buffer capacity of the specified size, in bytes. |

## Java ByteArrayOutputStream class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int size() | It is used to returns the current size of a buffer. |
| byte[] toByteArray() | It is used to create a newly allocated byte array. |
| String toString() | It is used for converting the content into a [string](https://www.javatpoint.com/java-string) decoding bytes using a platform default character set. |
| String toString(String charsetName) | It is used for converting the content into a string decoding bytes using a specified charsetName. |
| void write(int b) | It is used for writing the byte specified to the byte array output stream. |
| void write(byte[] b, int off, int len | It is used for writing **len** bytes from specified byte array starting from the offset **off** to the byte array output stream. |
| void writeTo(OutputStream out) | It is used for writing the complete content of a byte array output stream to the specified output stream. |
| void reset() | It is used to reset the count field of a byte array output stream to zero value. |
| void close() | It is used to close the ByteArrayOutputStream. |

## Example of Java ByteArrayOutputStream

Let's see a simple example of [java](https://www.javatpoint.com/java-tutorial) ByteArrayOutputStream class to write common data into 2 files: f1.txt and f2.txt.

**import** java.io.\*;

**public** **class** DataStreamExample {

**public** **static** **void** main(String args[])**throws** Exception{

      FileOutputStream fout1=**new** FileOutputStream("D:\\f1.txt");

      FileOutputStream fout2=**new** FileOutputStream("D:\\f2.txt");

      ByteArrayOutputStream bout=**new** ByteArrayOutputStream();

      bout.write(65);

      bout.writeTo(fout1);

      bout.writeTo(fout2);

      bout.flush();

      bout.close();//has no effect

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

     }

    }

