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

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.

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. |

FLUSH METHOD

In Java, the flush() method in the context of an output stream is used to ensure that any buffered data is actually written out. Many output streams in Java, like BufferedOutputStream or PrintWriter, use internal buffers to optimize performance by reducing the number of I/O operations. When data is written to these streams, it might not be immediately sent to the destination (like a file, network socket, etc.) but instead stored in a buffer.

The flush() method forces any data that might still be in the buffer to be written out. This is particularly important when you want to ensure that all data has been sent before performing another operation, such as closing the stream or switching to a different task.

Here’s an example of its usage:

FileOutputStream fileOut = new FileOutputStream("output.txt");

BufferedOutputStream bufferedOut = new BufferedOutputStream(fileOut);

bufferedOut.write("Hello, World!".getBytes());

bufferedOut.flush(); // Ensures "Hello, World!" is written to the file immediately

bufferedOut.close();

In this example, flush() ensures that "Hello, World!" is written to output.txt before closing the stream. If you don't call flush() before closing the stream, the remaining buffered data might be lost if close() doesn't automatically flush it (though most implementations of close() do call flush() internally).  
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:

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

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);}
10. }
11. }

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

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...

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

testout.txt

Welcome to javaTpoint.

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.

1. **package** com.javatpoint;
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. }

Output:

Success

testout.txt

Welcome to javaTpoint.

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:

1. **package** com.javatpoint;
3. **import** java.io.\*;
4. **public** **class** BufferedInputStreamExample{
5. **public** **static** **void** main(String args[]){
6. **try**{
7. FileInputStream fin=**new** FileInputStream("D:\\testout.txt");
8. BufferedInputStream bin=**new** BufferedInputStream(fin);
9. **int** i;
10. **while**((i=bin.read())!=-1){
11. System.out.print((**char**)i);
12. }
13. bin.close();
14. fin.close();
15. }**catch**(Exception e){System.out.println(e);}
16. }
17. }

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

javaTpoint

Output:

javaTpoint

Java SequenceInputStream Class

[Java](https://www.javatpoint.com/java-tutorial) SequenceInputStream [class](https://www.javatpoint.com/object-class) is used to read data from multiple [streams](https://www.javatpoint.com/java-8-stream). It reads data sequentially (one by one).

Java SequenceInputStream Class declaration

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

1. **public** **class** SequenceInputStream **extends** InputStream

Constructors of SequenceInputStream class

|  |  |
| --- | --- |
| [**Constructor**](https://www.javatpoint.com/java-constructor) | **Description** |
| SequenceInputStream(InputStream s1, InputStream s2) | creates a new input stream by reading the data of two input stream in order, first s1 and then s2. |
| SequenceInputStream(Enumeration e) | creates a new input stream by reading the data of an enumeration whose type is InputStream. |

Methods of SequenceInputStream class

|  |  |
| --- | --- |
| **Method** | **Description** |
| int read() | It is used to read the next byte of data from the input stream. |
| int read(byte[] ary, int off, int len) | It is used to read len bytes of data from the input stream into the [array](https://www.javatpoint.com/array-in-java) of bytes. |
| int available() | It is used to return the maximum number of byte that can be read from an input stream. |
| void close() | It is used to close the input stream. |

Java SequenceInputStream Example

In this example, we are printing the data of two files testin.txt and testout.txt.

1. **package** com.javatpoint;
3. **import** java.io.\*;
4. **class** InputStreamExample {
5. **public** **static** **void** main(String args[])**throws** Exception{
6. FileInputStream input1=**new** FileInputStream("D:\\testin.txt");
7. FileInputStream input2=**new** FileInputStream("D:\\testout.txt");
8. SequenceInputStream inst=**new** SequenceInputStream(input1, input2);
9. **int** j;
10. **while**((j=inst.read())!=-1){
11. System.out.print((**char**)j);
12. }
13. inst.close();
14. input1.close();
15. input2.close();
16. }
17. }

Here, we are assuming that you have two files: testin.txt and testout.txt which have following information:

testin.txt:

Welcome to Java IO Programming.

testout.txt:

It is the example of Java SequenceInputStream class.

After executing the program, you will get following output:

Output:

Welcome to Java IO Programming. It is the example of Java SequenceInputStream class.

Example that reads the data from two files and writes into another file

In this example, we are writing the data of two files **testin1.txt** and **testin2.txt** into another file named **testout.txt.**

1. **package** com.javatpoint;
3. **import** java.io.\*;
4. **class** Input1{
5. **public** **static** **void** main(String args[])**throws** Exception{
6. FileInputStream fin1=**new** FileInputStream("D:\\testin1.txt");
7. FileInputStream fin2=**new** FileInputStream("D:\\testin2.txt");
8. FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt");
9. SequenceInputStream sis=**new** SequenceInputStream(fin1,fin2);
10. **int** i;
11. **while**((i=sis.read())!=-1)
12. {
13. fout.write(i);
14. }
15. sis.close();
16. fout.close();
17. fin1.close();
18. fin2.close();
19. System.out.println("Success..");
20. }
21. }

Output:

Succeess...

testout.txt:

1. Welcome to Java IO Programming. It is the example of Java SequenceInputStream **class**.

SequenceInputStream example that reads data using enumeration

If we need to read the data from more than two files, we need to use [Enumeration](https://www.javatpoint.com/enum-in-java). Enumeration object can be obtained by calling elements() method of the Vector class. Let's see the simple example where we are reading the data from 4 files: a.txt, b.txt, c.txt and d.txt.

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **import** java.util.\*;
4. **class** Input2{
5. **public** **static** **void** main(String args[])**throws** IOException{
6. //creating the FileInputStream objects for all the files
7. FileInputStream fin=**new** FileInputStream("D:\\a.txt");
8. FileInputStream fin2=**new** FileInputStream("D:\\b.txt");
9. FileInputStream fin3=**new** FileInputStream("D:\\c.txt");
10. FileInputStream fin4=**new** FileInputStream("D:\\d.txt");
11. //creating Vector object to all the stream
12. Vector v=**new** Vector();
13. v.add(fin);
14. v.add(fin2);
15. v.add(fin3);
16. v.add(fin4);
17. //creating enumeration object by calling the elements method
18. Enumeration e=v.elements();
19. //passing the enumeration object in the constructor
20. SequenceInputStream bin=**new** SequenceInputStream(e);
21. **int** i=0;
22. **while**((i=bin.read())!=-1){
23. System.out.print((**char**)i);
24. }
25. bin.close();
26. fin.close();
27. fin2.close();
28. }
29. }

The a.txt, b.txt, c.txt and d.txt have following information:

a.txt:

Welcome

b.txt:

to

c.txt:

java

d.txt:

programming

Output:

Welcometojavaprogramming

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.

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **public** **class** DataStreamExample {
4. **public** **static** **void** main(String args[])**throws** Exception{
5. FileOutputStream fout1=**new** FileOutputStream("D:\\f1.txt");
6. FileOutputStream fout2=**new** FileOutputStream("D:\\f2.txt");
8. ByteArrayOutputStream bout=**new** ByteArrayOutputStream();
9. bout.write(65);
10. bout.writeTo(fout1);
11. bout.writeTo(fout2);
13. bout.flush();
14. bout.close();//has no effect
15. System.out.println("Success...");
16. }
17. }

Output:

Success...

f1.txt:

A

f2.txt:

A

# Java ByteArrayInputStream Class

The ByteArrayInputStream is composed of two words: ByteArray and InputStream. As the name suggests, it can be used to read byte [array](https://www.javatpoint.com/array-in-java) as input stream.

Java ByteArrayInputStream [class](https://www.javatpoint.com/object-and-class-in-java) contains an internal buffer which is used to **read byte array** as stream. In this stream, the data is read from a byte array.

The buffer of ByteArrayInputStream automatically grows according to data.

## Java ByteArrayInputStream class declaration

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

1. **public** **class** ByteArrayInputStream **extends** InputStream

## Java ByteArrayInputStream class constructors

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| ByteArrayInputStream(byte[] ary) | Creates a new byte array input stream which uses **ary** as its buffer array. |
| ByteArrayInputStream(byte[] ary, int offset, int len) | Creates a new byte array input stream which uses **ary** as its buffer array that can read up to specified **len** bytes of data from an array. |

## Java ByteArrayInputStream class methods

|  |  |
| --- | --- |
| **Methods** | **Description** |
| int available() | It is used to return the number of remaining bytes that can be read from the input stream. |
| int read() | It is used to read the next byte of data from the input stream. |
| int read(byte[] ary, int off, int len) | It is used to read up to len bytes of data from an array of bytes in the input stream. |
| boolean markSupported() | It is used to test the input stream for mark and reset method. |
| long skip(long x) | It is used to skip the x bytes of input from the input stream. |
| void mark(int readAheadLimit) | It is used to set the current marked position in the stream. |
| void reset() | It is used to reset the buffer of a byte array. |
| void close() | It is used for closing a ByteArrayInputStream. |

## Example of Java ByteArrayInputStream

Let's see a simple example of [java](https://www.javatpoint.com/java-tutorial) ByteArrayInputStream class to read byte array as input stream.

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **public** **class** ReadExample {
4. **public** **static** **void** main(String[] args) **throws** IOException {
5. **byte**[] buf = { 35, 36, 37, 38 };
6. // Create the new byte array input stream
7. ByteArrayInputStream byt = **new** ByteArrayInputStream(buf);
8. **int** k = 0;
9. **while** ((k = byt.read()) != -1) {
10. //Conversion of a byte into character
11. **char** ch = (**char**) k;
12. System.out.println("ASCII value of Character is:" + k + "; Special character is: " + ch);
13. }
14. }
15. }

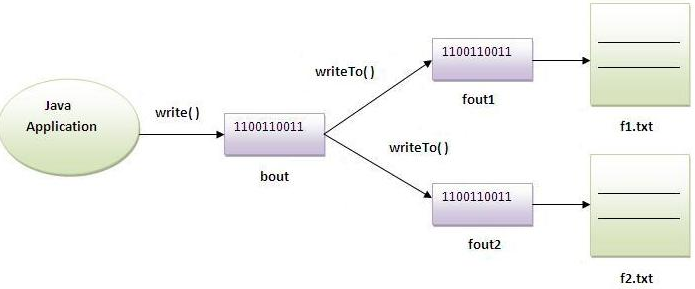
Output:

ASCII value of Character is:35; Special character is: #

ASCII value of Character is:36; Special character is: $

ASCII value of Character is:37; Special character is: %

ASCII value of Character is:38; Special character is: &



# Java DataOutputStream Class

Java DataOutputStream [class](https://www.javatpoint.com/object-and-class-in-java) allows an application to write primitive [Java](https://www.javatpoint.com/java-tutorial) 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:

1. **public** **class** DataOutputStream **extends** FilterOutputStream **implements** DataOutput

## Java DataOutputStream class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int size() | It is used to return the number of bytes written to the data output stream. |
| void write(int b) | It is used to write the specified byte to the underlying output stream. |
| void write(byte[] b, int off, int len) | It is used to write len bytes of data to the output stream. |
| void writeBoolean(boolean v) | It is used to write Boolean to the output stream as a 1-byte value. |
| void writeChar(int v) | It is used to write char to the output stream as a 2-byte value. |
| void writeChars(String s) | It is used to write [string](https://www.javatpoint.com/java-string) to the output stream as a sequence of characters. |
| void writeByte(int v) | It is used to write a byte to the output stream as a 1-byte value. |
| void writeBytes(String s) | It is used to write string to the output stream as a sequence of bytes. |
| void writeInt(int v) | It is used to write an int to the output stream |
| void writeShort(int v) | It is used to write a short to the output stream. |
| void writeShort(int v) | It is used to write a short to the output stream. |
| void writeLong(long v) | It is used to write a long to the output stream. |
| void writeUTF(String str) | It is used to write a string to the output stream using UTF-8 encoding in portable manner. |
| void flush() | It is used to flushes the data output stream. |

### Example of DataOutputStream class

In this example, we are writing the data to a text file **testout.txt** using DataOutputStream class.

1. **package** com.javatpoint;
3. **import** java.io.\*;
4. **public** **class** OutputExample {
5. **public** **static** **void** main(String[] args) **throws** IOException {
6. FileOutputStream file = **new** FileOutputStream(D:\\testout.txt);
7. DataOutputStream data = **new** DataOutputStream(file);
8. data.writeInt(65);
9. data.flush();
10. data.close();
11. System.out.println("Succcess...");
12. }
13. }

Output:

Succcess...

testout.txt:

A