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 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 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 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 single stream only ? * How can we access multiple files by single stream ? * How can we improve the performance of Input and Output operation ? * How many ways can we read data from the keyboard? * What is 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 working of Java OutputStream and InputStream by the figure given below.

Java IO

OutputStream class

|  |  |
| --- | --- |
| **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 class is an abstract class. It is the super class 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

OutputStream Hierarchy

Java output stream hierarchy

InputStream class

InputStream class is an abstract class. It is the super class 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 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.

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 FileOutStream.

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 ued 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. |

**Buffer** is temporary placeholder in memory (ram/disk) on which data can be dumped and then processing can be done.

**Stream**is a sequence of data elements. like when you typing something in your computer then when you press the keys it will form a data stream and then it goes to the processor for processing.

**Stream can be processed through Buffer.**

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 java.";
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 java.

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:

1. **public** **class** FileInputStream **extends** InputStream

|  |  |
| --- | --- |
| **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 class methods

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. }

**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

1. **package** com.javatpoint;
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. }

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:

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

|  |  |
| --- | --- |
| **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 constructors

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.

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 java.";
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 java.

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:

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. |

|  |  |
| --- | --- |
| **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. |

Java BufferedInputStream class methods

Example of Java BufferedInputStream

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

Java world

Output:

Java world

Ways to read input from console in Java

In Java, there are three different ways for reading input from the user in the command line environment(console).

**1.Using Buffered Reader Class**

This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

**Advantages**

* The input is buffered for efficient reading.

**Drawback:**

* The wrapping code is hard to remember.

**Program:**

|  |
| --- |
| // Java program to demonstrate BufferedReader  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  public class Test  {      public static void main(String[] args) throws IOException      {          //Enter data using BufferReader          BufferedReader reader =                     new BufferedReader(new InputStreamReader(System.in));            // Reading data using readLine          String name = reader.readLine();            // Printing the read line          System.out.println(name);      }  } |

Input:

Dhaka

Output:

Dhaka

Note: To read other types, we use functions like Integer.parseInt(), Double.parseDouble(). To read multiple values, we use split().

**2. Using Scanner Class**

This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however it is also can be used to read input from the user in the command line.

Advantages:

* Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
* Regular expressions can be used to find tokens.

|  |
| --- |
| // Java program to demonstrate working of Scanner in Java  import java.util.Scanner;    class GetInputFromUser  {      public static void main(String args[])      {          // Using Scanner for Getting Input from User          Scanner in = new Scanner(System.in);            String s = in.nextLine();          System.out.println("You entered string "+s);            int a = in.nextInt();          System.out.println("You entered integer "+a);            float b = in.nextFloat();          System.out.println("You entered float "+b);      }  } |

Input:

Dhaka

12

3.4

Output:

You entered integer 12

Enter a float

You entered float 3.4

**3. Using Console Class**

It has been becoming a preferred way for reading user’s input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).  
**Advantages:**

* Reading password without echoing the entered characters.
* Reading methods are synchronized.
* Format string syntax can be used.

**Drawback:**

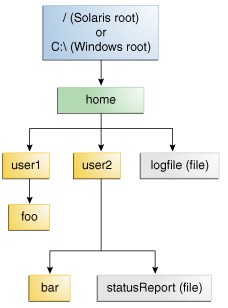
* Does not work in non-interactive environment (such as in an IDE).

|  |
| --- |
| // Java program to demonstrate working of System.console()  // Note that this program does not work on IDEs as  // System.console() may require console  public class Sample  {      public static void main(String[] args)      {          // Using Console to input data from user          String name = System.console().readLine();            System.out.println(name);      }  } |

## What Is a Path?

## A Path instance contains the information used to specify the location of a file or directory. At the time it is defined, a Path is provided with a series of one or more names. A root element or a file name might be included, but neither are required. A Path might consist of just a single directory or file name.

The following figure shows a sample directory tree containing a single root node. Microsoft Windows supports multiple root nodes. Each root node maps to a volume, such as C:\ or D:\. The Solaris OS supports a single root node, which is denoted by the slash character, /.



Sample Directory Structure

A file is identified by its path through the file system, beginning from the root node. For example, the statusReport file in the previous figure is described by the following notation in the Solaris OS:

/home/sally/statusReport

In Microsoft Windows, statusReport is described by the following notation:

C:\home\sally\statusReport

The character used to separate the directory names (also called the delimiter) is specific to the file system: The Solaris OS uses the forward slash (/), and Microsoft Windows uses the backslash slash (\).

**Relative Paths**

A relative path assumes that the file is on the current server. Using relative paths allows you to construct your site offline and fully test it before uploading it.

For example:

php/webct/itr/index.php

.

**Absolute Paths**

An absolute path refers to a file on the Internet using its full URL. Absolute paths tell the browser precisely where to go.

For example:

<http://www.uvsc.edu/disted/php/webct/itr/index.php>

# 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:

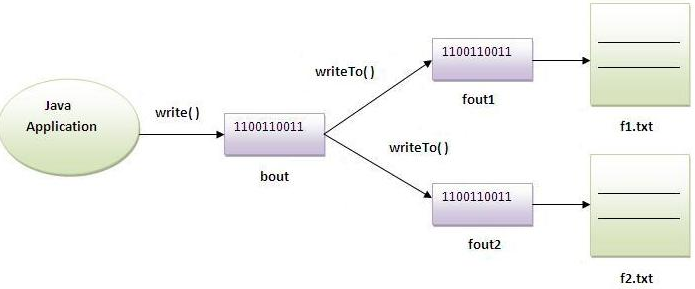
Success...

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

# Java DataInputStream Class

Java DataInputStream [class](https://www.javatpoint.com/object-and-class-in-java) allows an application to read primitive data from the input 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 DataInputStream class declaration

Let's see the declaration for java.io.DataInputStream class:

1. **public** **class** DataInputStream **extends** FilterInputStream **implements** DataInput

## Java DataInputStream class Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int read(byte[] b) | It is used to read the number of bytes from the input stream. |
| int read(byte[] b, int off, int len) | It is used to read **len** bytes of data from the input stream. |
| int readInt() | It is used to read input bytes and return an int value. |
| byte readByte() | It is used to read and return the one input byte. |
| char readChar() | It is used to read two input bytes and returns a char value. |
| double readDouble() | It is used to read eight input bytes and returns a double value. |
| boolean readBoolean() | It is used to read one input byte and return true if byte is non zero, false if byte is zero. |
| int skipBytes(int x) | It is used to skip over x bytes of data from the input stream. |
| String readUTF() | It is used to read a [string](https://www.javatpoint.com/java-string) that has been encoded using the UTF-8 format. |
| void readFully(byte[] b) | It is used to read bytes from the input stream and store them into the buffer [array](https://www.javatpoint.com/array-in-java). |
| void readFully(byte[] b, int off, int len) | It is used to read **len** bytes from the input stream. |

## Example of DataInputStream class

In this example, we are reading the data from the file testout.txt file.

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **public** **class** DataStreamExample {
4. **public** **static** **void** main(String[] args) **throws** IOException {
5. InputStream input = **new** FileInputStream("D:\\testout.txt");
6. DataInputStream inst = **new** DataInputStream(input);
7. **int** count = input.available();
8. **byte**[] ary = **new** **byte**[count];
9. inst.read(ary);
10. **for** (**byte** bt : ary) {
11. **char** k = (**char**) bt;
12. System.out.print(k+"-");
13. }
14. }
15. }

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

JAVA

Output:

J-A-V-A

# Java FilterOutputStream Class

Java FilterOutputStream class implements the OutputStream [class](https://www.javatpoint.com/object-and-class-in-java). It provides different sub classes such as [BufferedOutputStream](https://www.javatpoint.com/java-bufferedoutputstream-class)and [DataOutputStream](https://www.javatpoint.com/java-dataoutputstream-class) to provide additional functionality. So it is less used individually.

### Java FilterOutputStream class declaration

Let's see the declaration for java.io.FilterOutputStream class:

1. **public** **class** FilterOutputStream **extends** OutputStream

### Java FilterOutputStream class Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| void write(int b) | It is used to write the specified byte to the output stream. |
| void write(byte[] ary) | It is used to write ary.length byte to the output stream. |
| void write(byte[] b, int off, int len) | It is used to write len bytes from the offset off to the output stream. |
| void flush() | It is used to flushes the output stream. |
| void close() | It is used to close the output stream. |

### Example of FilterOutputStream class

1. **import** java.io.\*;
2. **public** **class** FilterExample {
3. **public** **static** **void** main(String[] args) **throws** IOException {
4. File data = **new** File("D:\\testout.txt");
5. FileOutputStream file = **new** FileOutputStream(data);
6. FilterOutputStream filter = **new** FilterOutputStream(file);
7. String s="Welcome to javaTpoint.";
8. **byte** b[]=s.getBytes();
9. filter.write(b);
10. filter.flush();
11. filter.close();
12. file.close();
13. System.out.println("Success...");
14. }
15. }

Output:

Success...

testout.txt

Welcome to javaTpoint.

# Java FilterInputStream Class

Java FilterInputStream class implements the InputStream. It contains different sub classes as [BufferedInputStream](https://www.javatpoint.com/java-bufferedinputstream-class), [DataInputStream](https://www.javatpoint.com/java-datainputstream-class) for providing additional functionality. So it is less used individually.

### Java FilterInputStream class declaration

Let's see the declaration for java.io.FilterInputStream class

1. **public** **class** FilterInputStream **extends** InputStream

### Java FilterInputStream class Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int available() | It is used to return an estimate number of 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[] b) | It is used to read up to byte.length bytes of data from the input stream. |
| long skip(long n) | It is used to skip over and discards n bytes of data from the input stream. |
| boolean markSupported() | It is used to test if the input stream support mark and reset method. |
| void mark(int readlimit) | It is used to mark the current position in the input stream. |
| void reset() | It is used to reset the input stream. |
| void close() | It is used to close the input stream. |

### Example of FilterInputStream class

1. **import** java.io.\*;
2. **public** **class** FilterExample {
3. **public** **static** **void** main(String[] args) **throws** IOException {
4. File data = **new** File("D:\\testout.txt");
5. FileInputStream  file = **new** FileInputStream(data);
6. FilterInputStream filter = **new** BufferedInputStream(file);
7. **int** k =0;
8. **while**((k=filter.read())!=-1){
9. System.out.print((**char**)k);
10. }
11. file.close();
12. filter.close();
13. }
14. }

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

Welcome to javatpoint

Output:

Welcome to javatpoint

# Java - ObjectStreamClass

ObjectStreamClass act as a [Serialization](https://www.javatpoint.com/serialization-in-java) descriptor for class. This [class](https://www.javatpoint.com/object-and-class-in-java) contains the name and serialVersionUID of the class.

### Fields

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Field** | **Description** |
| static ObjectStreamField[] | NO\_FIELDS | serialPersistentFields value indicating no serializable fields |

### Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| Class<?> | forClass() | It returns the class in the local VM that this version is mapped to. |
| ObjectStreamField | getField(String name) | It gets the field of this class by name. |
| ObjectStreamField[] | getFields() | It returns an [array](https://www.javatpoint.com/array-in-java) of the fields of this serialization class. |
| String | getName() | It returns the name of the class described by this descriptor. |
| long | getSerialVersionUID() | It returns the serialVersionUID for this class. |
| Static ObjectStreamClass | lookup(Class<?> cl) | It finds the descriptor for a class that can be serialized. |
| Static ObjectStreamClass | lookupAny(Class<?> cl) | It returns the descriptor for any class, regardless of whether it implements Serializable. |
| String | toString() | It returns a string describing this ObjectStreamClass. |

## Example

1. **import** java.io.ObjectStreamClass;
2. **import** java.util.Calendar;
4. **public** **class** ObjectStreamClassExample {
5. **public** **static** **void** main(String[] args) {
7. // create a new object stream class for Integers
8. ObjectStreamClass osc = ObjectStreamClass.lookup(SmartPhone.**class**);
10. // get the value field from ObjectStreamClass for integers
11. System.out.println("" + osc.getField("price"));
13. // create a new object stream class for Calendar
14. ObjectStreamClass osc2 = ObjectStreamClass.lookup(String.**class**);
16. // get the Class instance for osc2
17. System.out.println("" + osc2.getField("hash"));
19. }
20. }

Output:

I price

null

# Java ObjectStreamField class

A description of a Serializable field from a [Serializable](https://www.javatpoint.com/serialization-in-java) class. An [array](https://www.javatpoint.com/array-in-java) of ObjectStreamFields is used to declare the Serializable fields of a class.

The java.io.ObjectStreamClass.getField(String name) method gets the field of this class by name.

### Constructor

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| ObjectStreamField(String name, Class<?> type) | It creates a Serializable field with the specified type. |
| ObjectStreamField(String name, Class<?> type, boolean unshared) | It creates an ObjectStreamField representing a serializable field with the given name and type. |

### Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| int | compareTo(Object obj) | It compares this field with another ObjectStreamField. |
| String | getName() | It gets the name of this field. |
| int | GetOffset() | Offset of field within instance data. |
| Class<?> | getType() | It get the type of the field. |
| char | getTypeCode() | It returns character encoding of field type. |
| String | getTypeString() | It return the [JVM](https://www.javatpoint.com/internal-details-of-jvm) type signature. |
| boolean | isPrimitive() | It return true if this field has a primitive type. |
| boolean | isUnshared() | It returns boolean value indicating whether or not the serializable field represented by this ObjectStreamField instance is unshared. |
| protected void | setOffset(int offset) | Offset within instance data. |
| String | toString() | It return a [string](https://www.javatpoint.com/java-string) that describes this field. |

**public char getTypeCode()**

Returns character encoding of field type. The encoding is as follows:

|  |  |
| --- | --- |
| B | byte |
| C | char |
| D | double |
| F | float |
| I | int |
| J | long |
| L | class or interface |
| S | short |
| Z | boolean |
| [ | array |

**Returns:**

the typecode of the serializable field

## Example:

1. **import** java.io.ObjectStreamClass;
2. **import** java.util.Calendar;
4. **public** **class** ObjectStreamClassExample {
5. **public** **static** **void** main(String[] args) {
7. // create a new object stream class for Integers
8. ObjectStreamClass osc = ObjectStreamClass.lookup(String.**class**);
10. // get the value field from ObjectStreamClass for integers
11. System.out.println("" + osc.getField("value"));
13. // create a new object stream class for Calendar
14. ObjectStreamClass osc2 = ObjectStreamClass.lookup(Calendar.**class**);
16. // get the Class instance for osc2
17. System.out.println("" + osc2.getField("isTimeSet"));
19. }
20. }

Output:

I value

Z isTimeSet

# Java Console Class

The Java Console class is be used to get input from console. It provides methods to read texts and passwords.

If you read password using Console class, it will not be displayed to the user.

The java.io.Console class is attached with system console internally. The Console class is introduced since 1.5.

Let's see a simple example to read text from console.

1. String text=System.console().readLine();
2. System.out.println("Text is: "+text);

## Java Console class declaration

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

1. **public** **final** **class** Console **extends** Object **implements** Flushable

## Java Console class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| Reader reader() | It is used to retrieve the reader [object](https://www.javatpoint.com/object-and-class-in-java) associated with the console |
| String readLine() | It is used to read a single line of text from the console. |
| String readLine(String fmt, Object... args) | It provides a formatted prompt then reads the single line of text from the console. |
| char[] readPassword() | It is used to read password that is not being displayed on the console. |
| char[] readPassword(String fmt, Object... args) | It provides a formatted prompt then reads the password that is not being displayed on the console. |
| Console format(String fmt, Object... args) | It is used to write a formatted [string](https://www.javatpoint.com/java-string) to the console output stream. |
| Console printf(String format, Object... args) | It is used to write a string to the console output stream. |
| PrintWriter writer() | It is used to retrieve the [PrintWriter](https://www.javatpoint.com/java-printwriter-class) object associated with the console. |
| void flush() | It is used to flushes the console. |

## How to get the object of Console

System class provides a static method console() that returns the [singleton](https://www.javatpoint.com/singleton-design-pattern-in-java) instance of Console class.

1. **public** **static** Console console(){}

Let's see the code to get the instance of Console class.

1. Console c=System.console();

## Java Console Example

1. **import** java.io.Console;
2. **class** ReadStringTest{
3. **public** **static** **void** main(String args[]){
4. Console c=System.console();
5. System.out.println("Enter your name: ");
6. String n=c.readLine();
7. System.out.println("Welcome "+n);
8. }
9. }

Output

Enter your name: Nakul Jain

Welcome Nakul Jain

## Java Console Example to read password

1. **import** java.io.Console;
2. **class** ReadPasswordTest{
3. **public** **static** **void** main(String args[]){
4. Console c=System.console();
5. System.out.println("Enter password: ");
6. **char**[] ch=c.readPassword();
7. String pass=String.valueOf(ch);//converting char array into string
8. System.out.println("Password is: "+pass);
9. }
10. }

Output

Enter password:

Password is: 123

# Java FilePermission Class

Java FilePermission class contains the permission related to a directory or [file](https://www.javatpoint.com/java-file-class). All the permissions are related with path. The path can be of two types:

1) **D:\\IO\\**-: It indicates that the permission is associated with all sub directories and files recursively.

2) **D:\\IO\\\***: It indicates that the permission is associated with all directory and files within this directory excluding sub directories.

## Java FilePermission class declaration

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

1. **public** **final** **class** FilePermission **extends** Permission **implements** Serializable

## Methods of FilePermission class

|  |  |
| --- | --- |
| **Method** | **Description** |
| ByteArrayOutputStream() | Creates a new byte [array](https://www.javatpoint.com/array-in-java) output 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 FilePermission class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int hashCode() | It is used to return the hash code value of an [object](https://www.javatpoint.com/object-and-class-in-java). |
| String getActions() | It is used to return the "canonical string representation" of an action. |
| boolean equals(Object obj) | It is used to check the two FilePermission objects for equality. |
| boolean implies(Permission p) | It is used to check the FilePermission object for the specified permission. |
| PermissionCollection newPermissionCollection() | It is used to return the new PermissonCollection object for storing the FilePermission object. |

## Java FilePermission Example

Let's see the simple example in which permission of a directory path is granted with read permission and a file of this directory is granted for write permission.

1. **package** com.javatpoint;
3. **import** java.io.\*;
4. **import** java.security.PermissionCollection;
5. **public** **class** FilePermissionExample{
6. **public** **static** **void** main(String[] args) **throws** IOException {
7. String srg = "D:\\IO Package\\java.txt";
8. FilePermission file1 = **new** FilePermission("D:\\IO Package\\-", "read");
9. PermissionCollection permission = file1.newPermissionCollection();
10. permission.add(file1);
11. FilePermission file2 = **new** FilePermission(srg, "write");
12. permission.add(file2);
13. **if**(permission.implies(**new** FilePermission(srg, "read,write"))) {
14. System.out.println("Read, Write permission is granted for the path "+srg );
15. }**else** {
16. System.out.println("No Read, Write permission is granted for the path "+srg);            }
17. }
18. }

Output

Read, Write permission is granted for the path D:\IO Package\java.txt

File Permissions in Java

Java provides a number of method calls to check and change the permission of a file, such as a read-only file can be changed to have permissions to write. File permissions are required to be changed when the user want to restrict the operations permissible on a file. For example, a file permission can be changed from write to read-only because the user no longer want to edit the file.

**Checking the current file permissions**

A file can be in any combination of following permissible permissions:

* **Executable:** Tests whether the application can execute the file denoted by this abstract path name.  
  Syntax:
* **public boolean canExecute()**
* **Returns:** true if and only if the abstract path name

exists and the application is allowed to execute the file

* **Readable:**Tests whether the application can read the file denoted by this abstract path name.  
  Syntax:
* **public boolean canRead()**
* **Returns:** true if and only if the file specified by this

abstract path name exists and can be read by the application; false otherwise

* **Writable:** Tests whether the application can modify the file denoted by this abstract path name.  
  Syntax:
* **public boolean canWrite()**
* **Returns:** true if and only if the file system actually
* contains a file denoted by this abstract path name and

the application is allowed to write to the file; false otherwise.

For example, a file can be readable and writable but not executable. Here’s Java program to get the current permissions associated with a file.

|  |
| --- |
| // Java program to check the current file permissions.  import java.io.\*;    public class Test  {      public static void main(String[] args)      {          // creating a file instance          File file = new File("C:\\Users\\Mayank\\Desktop\\1.txt");            // check if the file exists          boolean exists = file.exists();          if(exists == true)          {              // printing the permissions associated with the file              System.out.println("Executable: " + file.canExecute());              System.out.println("Readable: " + file.canRead());              System.out.println("Writable: "+ file.canWrite());          }          else          {              System.out.println("File not found.");          }      }  } |

**Output**

Executable: true

Readable: true

Writable: true

**Changing file permissions**

A file can have any combinations of the following permissions:

* Executable
* Readable
* Writable

Here are methods to change the permissions associated with a file:

* **setExecutable**A convenience method to set the owner’s execute permission for this abstract path name.
* **public boolean setExecutable(boolean executable)**
* **Description:**
* **Parameters:** executable - If true, sets the access
* permission to allow execute operations;
* if false to disallow execute operations

**Returns:** true if and only if the operation succeeded.

The operation will fail if the user does not have permission to change the access permissions of this abstract path name. If executable is false and the underlying file system does not implement an execute permission, then the operation will fail.

* **setReadable:** A convenience method to set the owner’s read permission for this abstract path name.
* **public boolean setReadable(boolean readable)**
* **Parameters:** readable - If true, sets the access permission to
* allow read operations; if false to disallow read operations

**Returns:** true if and only if the operation succeeded.

The operation will fail if the user does not have permission to change the access permissions of this abstract path name. If readable is false and the underlying file system does not implement a read permission, then the operation will fail.

* **setWritable :** A convenience method to set the owner’s write permission for this abstract path name.
* **public boolean setWritable(boolean writable)**
* **Parameters:** writable - If true, sets the access permission
* to allow write operations; if false to disallow write operations

**Returns:** true if and only if the operation succeeded.

The operation will fail if the user does not have permission to change the access permissions of this abstract path name.

|  |
| --- |
| // Java program to change the file permissions  import java.io.\*;    public class Test  {      public static void main(String[] args)      {          // creating a new file instance          File file = new File("C:\\Users\\Mayank\\Desktop\\1.txt");            // check if file exists          boolean exists = file.exists();          if(exists == true)          {              // changing the file permissions              file.setExecutable(true);              file.setReadable(true);              file.setWritable(false);              System.out.println("File permissions changed.");                // printing the permissions associated with the file currently              System.out.println("Executable: " + file.canExecute());              System.out.println("Readable: " + file.canRead());              System.out.println("Writable: "+ file.canWrite());            }          else          {              System.out.println("File not found.");          }      }  } |

**Output**

File permissions changed.

Executable: true

Readable: true

Writable: false

# Java Writer

It is an [abstract](https://www.javatpoint.com/abstract-class-in-java) class for writing to character streams. The methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses will override some of the methods defined here to provide higher efficiency, functionality or both.

### Fields

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Field** | **Description** |
| protected Object | lock | The object used to synchronize operations on this stream. |

### Constructor

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Constructor** | **Description** |
| protected | Writer() | It creates a new character-stream writer whose critical sections will synchronize on the writer itself. |
| protected | Writer(Object lock) | It creates a new character-stream writer whose critical sections will synchronize on the given [object](https://www.javatpoint.com/object-and-class-in-java). |

### Methods

|  |  |  |
| --- | --- | --- |
| Modifier and Type | **Method** | **Description** |
| Writer | append(char c) | It appends the specified character to this writer. |
| Writer | append(CharSequence csq) | It appends the specified character sequence to this writer |
| Writer | append(CharSequence csq, int start, int end) | It appends a subsequence of the specified character sequence to this writer. |
| abstract void | close() | It closes the stream, flushing it first. |
| abstract void | flush() | It flushes the stream. |
| void | write(char[] cbuf) | It writes an [array](https://www.javatpoint.com/array-in-java) of characters. |
| abstract void | write(char[] cbuf, int off, int len) | It writes a portion of an array of characters. |
| void | write(int c) | It writes a single character. |
| void | write(String str) | It writes a [string](https://www.javatpoint.com/java-string). |
| void | write(String str, int off, int len) | It writes a portion of a string. |

## Java Writer Example

1. **import** java.io.\*;
2. **public** **class** WriterExample {
3. **public** **static** **void** main(String[] args) {
4. **try** {
5. Writer w = **new** FileWriter("output.txt");
6. String content = "I love my country";
7. w.write(content);
8. w.close();
9. System.out.println("Done");
10. } **catch** (IOException e) {
11. e.printStackTrace();
12. }
13. }
14. }

Output:

Done

output.txt:

I love my country

# Java Reader

[Java](https://www.javatpoint.com/java-tutorial) Reader is an [abstract class](https://www.javatpoint.com/abstract-class-in-java) for reading character [streams](https://www.javatpoint.com/java-8-stream). The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will [override](https://www.javatpoint.com/method-overriding-in-java) some of the methods to provide higher efficiency, additional functionality, or both.

Some of the implementation [class](https://www.javatpoint.com/object-class) are [BufferedReader](https://www.javatpoint.com/java-bufferedreader-class), [CharArrayReader](https://www.javatpoint.com/java-chararrayreader-class), [FilterReader](https://www.javatpoint.com/java-filterreader-class), [InputStreamReader](https://www.javatpoint.com/Input-from-keyboard-by-InputStreamReader), PipedReader, [StringReader](https://www.javatpoint.com/java-stringreader-class)

### Fields

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Field** | **Description** |
| protected Object | lock | The object used to synchronize operations on this stream. |

### Constructor

|  |  |  |
| --- | --- | --- |
| [**Modifie**](https://www.javatpoint.com/access-modifiers)**r** | [**Constructor**](https://www.javatpoint.com/java-constructor) | **Description** |
| protected | Reader() | It creates a new character-stream reader whose critical sections will synchronize on the reader itself. |
| protected | Reader(Object lock) | It creates a new character-stream reader whose critical sections will synchronize on the given object. |

### Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| abstract void | close() | It closes the stream and releases any system resources associated with it. |
| void | mark(int readAheadLimit) | It marks the present position in the stream. |
| boolean | markSupported() | It tells whether this stream supports the mark() operation. |
| int | read() | It reads a single character. |
| int | read(char[] cbuf) | It reads characters into an [array](https://www.javatpoint.com/array-in-java). |
| abstract int | read(char[] cbuf, int off, int len) | It reads characters into a portion of an array. |
| int | read(CharBuffer target) | It attempts to read characters into the specified character buffer. |
| boolean | ready() | It tells whether this stream is ready to be read. |
| void | reset() | It resets the stream. |
| long | skip(long n) | It skips characters. |

## Example

1. **import** java.io.\*;
2. **public** **class** ReaderExample {
3. **public** **static** **void** main(String[] args) {
4. **try** {
5. Reader reader = **new** FileReader("file.txt");
6. **int** data = reader.read();
7. **while** (data != -1) {
8. System.out.print((**char**) data);
9. data = reader.read();
10. }
11. reader.close();
12. } **catch** (Exception ex) {
13. System.out.println(ex.getMessage());
14. }
15. }
16. }

file.txt:

I love my country

Output:

I love my country

# Java FileWriter Class

Java FileWriter class is used to write character-oriented data to a [file](https://www.javatpoint.com/java-file-class). It is character-oriented class which is used for file handling in [java](https://www.javatpoint.com/java-tutorial).

Unlike FileOutputStream class, you don't need to convert string into byte [array](https://www.javatpoint.com/array-in-java) because it provides method to write string directly.

## Java FileWriter class declaration

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

1. **public** **class** FileWriter **extends** OutputStreamWriter

## Constructors of FileWriter class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| FileWriter(String file) | Creates a new file. It gets file name in [string](https://www.javatpoint.com/java-string). |
| FileWriter(File file) | Creates a new file. It gets file name in File [object](https://www.javatpoint.com/object-and-class-in-java). |

## 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.

1. **package** com.javatpoint;
2. **import** java.io.FileWriter;
3. **public** **class** FileWriterExample {
4. **public** **static** **void** main(String args[]){
5. **try**{
6. FileWriter fw=**new** FileWriter("D:\\testout.txt");
7. fw.write("Welcome to javaTpoint.");
8. fw.close();
9. }**catch**(Exception e){System.out.println(e);}
10. System.out.println("Success...");
11. }
12. }

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

It is character-oriented class which is used for [file](https://www.javatpoint.com/java-file-class) handling in [java](https://www.javatpoint.com/java-tutorial).

## Java FileReader class declaration

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

1. **public** **class** FileReader **extends** InputStreamReader

## Constructors of FileReader class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| FileReader(String file) | It gets filename in [string](https://www.javatpoint.com/java-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](https://www.javatpoint.com/java-file-class) 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.

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

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

Welcome to javaTpoint.

Output:

Welcome to javaTpoint.

# Java BufferedWriter Class

Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits [Writer](https://www.javatpoint.com/java-writer-class) class. The buffering characters are used for providing the efficient writing of single [arrays](https://www.javatpoint.com/array-in-java), characters, and [strings](https://www.javatpoint.com/java-string).

## Class declaration

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

1. **public** **class** BufferedWriter **extends** Writer

## Class constructors

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| BufferedWriter(Writer wrt) | It is used to create a buffered character output stream that uses the default size for an output buffer. |
| BufferedWriter(Writer wrt, int size) | It is used to create a buffered character output stream that uses the specified size for an output buffer. |

## Class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| void newLine() | It is used to add a new line by writing a line separator. |
| void write(int c) | It is used to write a single character. |
| void write(char[] cbuf, int off, int len) | It is used to write a portion of an array of characters. |
| void write(String s, int off, int len) | It is used to write a portion of a string. |
| void flush() | It is used to flushes the input stream. |
| void close() | It is used to closes the input stream |

## Example of Java BufferedWriter

Let's see the simple example of writing the data to a text file **testout.txt** using Java BufferedWriter.

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **public** **class** BufferedWriterExample {
4. **public** **static** **void** main(String[] args) **throws** Exception {
5. FileWriter writer = **new** FileWriter("D:\\testout.txt");
6. BufferedWriter buffer = **new** BufferedWriter(writer);
7. buffer.write("Welcome to javaTpoint.");
8. buffer.close();
9. System.out.println("Success");
10. }
11. }

Output:

success

testout.txt:

Welcome to javaTpoint.

# Java BufferedReader Class

Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits [Reader](https://www.javatpoint.com/java-reader-class) [class](https://www.javatpoint.com/object-and-class-in-java).

## Java BufferedReader class declaration

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

1. **public** **class** BufferedReader **extends** Reader

## Java BufferedReader class constructors

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| BufferedReader(Reader rd) | It is used to create a buffered character input stream that uses the default size for an input buffer. |
| BufferedReader(Reader rd, int size) | It is used to create a buffered character input stream that uses the specified size for an input buffer. |

## Java BufferedReader class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int read() | It is used for reading a single character. |
| int read(char[] cbuf, int off, int len) | It is used for reading characters into a portion of an [array](https://www.javatpoint.com/array-in-java). |
| boolean markSupported() | It is used to test the input stream support for the mark and reset method. |
| String readLine() | It is used for reading a line of text. |
| boolean ready() | It is used to test whether the input stream is ready to be read. |
| long skip(long n) | It is used for skipping the characters. |
| void reset() | It repositions the [stream](https://www.javatpoint.com/java-8-stream) at a position the mark method was last called on this input stream. |
| void mark(int readAheadLimit) | It is used for marking the present position in a stream. |
| void close() | It closes the input stream and releases any of the system resources associated with the stream. |

## Java BufferedReader Example

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

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **public** **class** BufferedReaderExample {
4. **public** **static** **void** main(String args[])**throws** Exception{
5. FileReader fr=**new** FileReader("D:\\testout.txt");
6. BufferedReader br=**new** BufferedReader(fr);
8. **int** i;
9. **while**((i=br.read())!=-1){
10. System.out.print((**char**)i);
11. }
12. br.close();
13. fr.close();
14. }
15. }

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

Welcome to javaTpoint.

Output:

Welcome to javaTpoint.

## Reading data from console by InputStreamReader and BufferedReader

In this example, we are connecting the BufferedReader stream with the [InputStreamReader](https://www.javatpoint.com/Input-from-keyboard-by-InputStreamReader) stream for reading the line by line data from the keyboard.

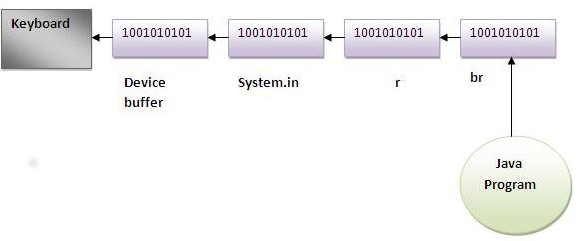
1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **public** **class** BufferedReaderExample{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. InputStreamReader r=**new** InputStreamReader(System.in);
6. BufferedReader br=**new** BufferedReader(r);
7. System.out.println("Enter your name");
8. String name=br.readLine();
9. System.out.println("Welcome "+name);
10. }
11. }

Output:

Enter your name

Nakul Jain

Welcome Nakul Jain



## Another example of reading data from console until user writes stop

In this example, we are reading and printing the data until the user prints stop.

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **public** **class** BufferedReaderExample{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. InputStreamReader r=**new** InputStreamReader(System.in);
6. BufferedReader br=**new** BufferedReader(r);
7. String name="";
8. **while**(!name.equals("stop")){
9. System.out.println("Enter data: ");
10. name=br.readLine();
11. System.out.println("data is: "+name);
12. }
13. br.close();
14. r.close();
15. }
16. }

Output:

Enter data: Nakul

data is: Nakul

Enter data: 12

data is: 12

Enter data: stop

data is: stop

# Java CharArrayReader Class

The CharArrayReader is composed of two words: CharArray and Reader. The CharArrayReader class is used to read character [array](https://www.javatpoint.com/array-in-java)as a reader (stream). It inherits [Reader](https://www.javatpoint.com/java-reader-class) class.

## Java CharArrayReader class declaration

Let's see the declaration for Java.io.CharArrayReader [class](https://www.javatpoint.com/object-and-class-in-java):

1. **public** **class** CharArrayReader **extends** Reader

## Java CharArrayReader class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int read() | It is used to read a single character |
| int read(char[] b, int off, int len) | It is used to read characters into the portion of an array. |
| boolean ready() | It is used to tell whether the stream is ready to read. |
| boolean markSupported() | It is used to tell whether the stream supports mark() operation. |
| long skip(long n) | It is used to skip the character in the input stream. |
| void mark(int readAheadLimit) | It is used to mark the present position in the stream. |
| void reset() | It is used to reset the stream to a most recent mark. |
| void close() | It is used to closes the stream. |

## Example of CharArrayReader Class:

Let's see the simple example to read a character using Java CharArrayReader class.

1. **package** com.javatpoint;
3. **import** java.io.CharArrayReader;
4. **public** **class** CharArrayExample{
5. **public** **static** **void** main(String[] ag) **throws** Exception {
6. **char**[] ary = { 'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't' };
7. CharArrayReader reader = **new** CharArrayReader(ary);
8. **int** k = 0;
9. // Read until the end of a file
10. **while** ((k = reader.read()) != -1) {
11. **char** ch = (**char**) k;
12. System.out.print(ch + " : ");
13. System.out.println(k);
14. }
15. }
16. }

Output

j : 106

a : 97

v : 118

a : 97

t : 116

p : 112

o : 111

i : 105

n : 110

t : 116

# Java CharArrayWriter Class

The CharArrayWriter class can be used to write common data to multiple files. This class inherits [Writer](https://www.javatpoint.com/java-writer-class) class. Its buffer automatically grows when data is written in this stream. Calling the close() method on this [object](https://www.javatpoint.com/object-and-class-in-java) has no effect.

## Java CharArrayWriter class declaration

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

1. **public** **class** CharArrayWriter **extends** Writer

## Java CharArrayWriter class Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int size() | It is used to return the current size of the buffer. |
| char[] toCharArray() | It is used to return the copy of an input data. |
| String toString() | It is used for converting an input data to a [string](https://www.javatpoint.com/java-string). |
| CharArrayWriter append(char c) | It is used to append the specified character to the writer. |
| CharArrayWriter append(CharSequence csq) | It is used to append the specified character sequence to the writer. |
| CharArrayWriter append(CharSequence csq, int start, int end) | It is used to append the subsequence of a specified character to the writer. |
| void write(int c) | It is used to write a character to the buffer. |
| void write(char[] c, int off, int len) | It is used to write a character to the buffer. |
| void write(String str, int off, int len) | It is used to write a portion of string to the buffer. |
| void writeTo(Writer out) | It is used to write the content of buffer to different character stream. |
| void flush() | It is used to flush the stream. |
| void reset() | It is used to reset the buffer. |
| void close() | It is used to close the stream. |

### Example of CharArrayWriter Class:

In this example, we are writing a common data to 4 files a.txt, b.txt, c.txt and d.txt.

1. **package** com.javatpoint;
3. **import** java.io.CharArrayWriter;
4. **import** java.io.FileWriter;
5. **public** **class** CharArrayWriterExample {
6. **public** **static** **void** main(String args[])**throws** Exception{
7. CharArrayWriter out=**new** CharArrayWriter();
8. out.write("Welcome to javaTpoint");
9. FileWriter f1=**new** FileWriter("D:\\a.txt");
10. FileWriter f2=**new** FileWriter("D:\\b.txt");
11. FileWriter f3=**new** FileWriter("D:\\c.txt");
12. FileWriter f4=**new** FileWriter("D:\\d.txt");
13. out.writeTo(f1);
14. out.writeTo(f2);
15. out.writeTo(f3);
16. out.writeTo(f4);
17. f1.close();
18. f2.close();
19. f3.close();
20. f4.close();
21. System.out.println("Success...");
22. }
23. }

Output

Success...

After executing the program, you can see that all files have common data: Welcome to javaTpoint.

a.txt:

Welcome to javaTpoint

b.txt:

Welcome to javaTpoint

c.txt:

Welcome to javaTpoint

d.txt:

Welcome to javaTpoint

# Java PrintStream Class

The PrintStream class provides methods to write data to another stream. The PrintStream [class](https://www.javatpoint.com/object-and-class-in-java) automatically flushes the data so there is no need to call flush() method. Moreover, its methods don't throw IOException.

## Class declaration

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

1. **public** **class** PrintStream **extends** FilterOutputStream **implements** Closeable. Appendable

## Methods of PrintStream class

|  |  |
| --- | --- |
| **Method** | **Description** |
| void print(boolean b) | It prints the specified boolean value. |
| void print(char c) | It prints the specified char value. |
| void print(char[] c) | It prints the specified character [array](https://www.javatpoint.com/array-in-java) values. |
| void print(int i) | It prints the specified int value. |
| void print(long l) | It prints the specified long value. |
| void print(float f) | It prints the specified float value. |
| void print(double d) | It prints the specified double value. |
| void print(String s) | It prints the specified [string](https://www.javatpoint.com/java-string) value. |
| void print(Object obj) | It prints the specified object value. |
| void println(boolean b) | It prints the specified boolean value and terminates the line. |
| void println(char c) | It prints the specified char value and terminates the line. |
| void println(char[] c) | It prints the specified character array values and terminates the line. |
| void println(int i) | It prints the specified int value and terminates the line. |
| void println(long l) | It prints the specified long value and terminates the line. |
| void println(float f) | It prints the specified float value and terminates the line. |
| void println(double d) | It prints the specified double value and terminates the line. |
| void println(String s) | It prints the specified string value and terminates the line. |
| void println(Object obj) | It prints the specified object value and terminates the line. |
| void println() | It terminates the line only. |
| void printf(Object format, Object... args) | It writes the formatted string to the current stream. |
| void printf(Locale l, Object format, Object... args) | It writes the formatted string to the current stream. |
| void format(Object format, Object... args) | It writes the formatted string to the current stream using specified format. |
| void format(Locale l, Object format, Object... args) | It writes the formatted string to the current stream using specified format. |

## Example of java PrintStream class

In this example, we are simply printing integer and string value.

1. **package** com.javatpoint;
3. **import** java.io.FileOutputStream;
4. **import** java.io.PrintStream;
5. **public** **class** PrintStreamTest{
6. **public** **static** **void** main(String args[])**throws** Exception{
7. FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt ");
8. PrintStream pout=**new** PrintStream(fout);
9. pout.println(2016);
10. pout.println("Hello Java");
11. pout.println("Welcome to Java");
12. pout.close();
13. fout.close();
14. System.out.println("Success?");
15. }
16. }

Output

Success...

The content of a text file **testout.txt** is set with the below data

2016

Hello Java

Welcome to Java

## Example of printf() method using java PrintStream class:

Let's see the simple example of printing integer value by format specifier using **printf()** method of **java.io.PrintStream** class.

1. **class** PrintStreamTest{
2. **public** **static** **void** main(String args[]){
3. **int** a=19;
4. System.out.printf("%d",a); //Note: out is the object of printstream
5. }
6. }

Output

19

# Java PrintWriter class

Java PrintWriter class is the implementation of [Writer](https://www.javatpoint.com/java-writer-class) class. It is used to print the formatted representation of [objects](https://www.javatpoint.com/object-and-class-in-java) to the text-output stream.

## Class declaration

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

1. **public** **class** PrintWriter **extends** Writer

## Methods of PrintWriter class

|  |  |
| --- | --- |
| **Method** | **Description** |
| void println(boolean x) | It is used to print the boolean value. |
| void println(char[] x) | It is used to print an [array](https://www.javatpoint.com/array-in-java) of characters. |
| void println(int x) | It is used to print an integer. |
| PrintWriter append(char c) | It is used to append the specified character to the writer. |
| PrintWriter append(CharSequence ch) | It is used to append the specified character sequence to the writer. |
| PrintWriter append(CharSequence ch, int start, int end) | It is used to append a subsequence of specified character to the writer. |
| boolean checkError() | It is used to flushes the stream and check its error state. |
| protected void setError() | It is used to indicate that an error occurs. |
| protected void clearError() | It is used to clear the error state of a stream. |
| PrintWriter format(String format, Object... args) | It is used to write a formatted [string](https://www.javatpoint.com/java-string) to the writer using specified arguments and format string. |
| void print(Object obj) | It is used to print an object. |
| void flush() | It is used to flushes the stream. |
| void close() | It is used to close the stream. |

## Java PrintWriter Example

Let's see the simple example of writing the data on a **console** and in a **text file testout.txt** using Java PrintWriter class.

1. **package** com.javatpoint;
3. **import** java.io.File;
4. **import** java.io.PrintWriter;
5. **public** **class** PrintWriterExample {
6. **public** **static** **void** main(String[] args) **throws** Exception {
7. //Data to write on Console using PrintWriter
8. PrintWriter writer = **new** PrintWriter(System.out);
9. writer.write("Javatpoint provides tutorials of all technology.");
10. writer.flush();
11. writer.close();
12. //Data to write in File using PrintWriter
13. PrintWriter writer1 =**null**;
14. writer1 = **new** PrintWriter(**new** File("D:\\testout.txt"));
15. writer1.write("Like Java, Spring, Hibernate, Android, PHP etc.");
16. writer1.flush();
17. writer1.close();
18. }
19. }

Outpt

Javatpoint provides tutorials of all technology.

The content of a text file **testout.txt** is set with the data **Like Java, Spring, Hibernate, Android, PHP etc.**

# Java OutputStreamWriter

OutputStreamWriter is a [class](https://www.javatpoint.com/object-and-class-in-java) which is used to convert character stream to byte stream, the characters are encoded into byte using a specified charset. write() method calls the encoding converter which converts the character into bytes. The resulting bytes are then accumulated in a buffer before being written into the underlying output stream. The characters passed to write() methods are not buffered. We optimize the performance of OutputStreamWriter by using it with in a BufferedWriter so that to avoid frequent converter invocation.

### Constructor

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| OutputStreamWriter(OutputStream out) | It creates an OutputStreamWriter that uses the default character encoding. |
| OutputStreamWriter(OutputStream out, Charset cs) | It creates an OutputStreamWriter that uses the given charset. |
| OutputStreamWriter(OutputStream out, CharsetEncoder enc) | It creates an OutputStreamWriter that uses the given charset encoder. |
| OutputStreamWriter(OutputStream out, String charsetName) | It creates an OutputStreamWriter that uses the named charset. |

### Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| void | close() | It closes the stream, flushing it first. |
| void | flush() | It flushes the stream. |
| String | getEncoding() | It returns the name of the character encoding being used by this stream. |
| void | write(char[] cbuf, int off, int len) | It writes a portion of an [array](https://www.javatpoint.com/array-in-java) of characters. |
| void | write(int c) | It writes a single character. |
| void | write(String str, int off, int len) | It writes a portion of a [string](https://www.javatpoint.com/java-string). |

## Example

1. **public** **class** OutputStreamWriterExample {
2. **public** **static** **void** main(String[] args) {
4. **try** {
5. OutputStream outputStream = **new** FileOutputStream("output.txt");
6. Writer outputStreamWriter = **new** OutputStreamWriter(outputStream);
8. outputStreamWriter.write("Hello World");
10. outputStreamWriter.close();
11. } **catch** (Exception e) {
12. e.getMessage();
13. }
14. }
15. }

Output:

output.txt file will contains text "Hello World"

# Java InputStreamReader

An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

### Constructor

|  |  |
| --- | --- |
| **Constructor name** | **Description** |
| InputStreamReader(InputStream in) | It creates an InputStreamReader that uses the default charset. |
| InputStreamReader(InputStream in, Charset cs) | It creates an InputStreamReader that uses the given charset. |
| InputStreamReader(InputStream in, CharsetDecoder dec) | It creates an InputStreamReader that uses the given charset decoder. |
| InputStreamReader(InputStream in, String charsetName) | It creates an InputStreamReader that uses the named charset. |

### Method

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| void | close() | It closes the stream and releases any system resources associated with it. |
| String | getEncoding() | It returns the name of the character encoding being used by this stream. |
| int | read() | It reads a single character. |
| int | read(char[] cbuf, int offset, int length) | It reads characters into a portion of an [array](https://www.javatpoint.com/array-in-java). |
| boolean | ready() | It tells whether this stream is ready to be read. |

## Example

1. **public** **class** InputStreamReaderExample {
2. **public** **static** **void** main(String[] args) {
3. **try**  {
4. InputStream stream = **new** FileInputStream("file.txt");
5. Reader reader = **new** InputStreamReader(stream);
6. **int** data = reader.read();
7. **while** (data != -1) {
8. System.out.print((**char**) data);
9. data = reader.read();
10. }
11. } **catch** (Exception e) {
12. e.printStackTrace();
13. }
14. }
15. }

Output:

I love my country

The file.txt contains text "I love my country" the InputStreamReader

reads Character by character from the file

# Java PushbackInputStream Class

Java PushbackInputStream [class](https://www.javatpoint.com/object-and-clas-in-java) overrides InputStream and provides extra functionality to another input stream. It can unread a byte which is already read and push back one byte.

### Class declaration

Let's see the declaration for java.io.PushbackInputStream class:

1. **public** **class** PushbackInputStream **extends** FilterInputStream

### Class Methods

It is used to test if the input stream support mark and reset method.

|  |  |
| --- | --- |
| **Method** | **Description** |
| int available() | It is used to return the number of 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. |
| boolean markSupported() |  |
| void mark(int readlimit) | It is used to mark the current position in the input stream. |
| long skip(long x) | It is used to skip over and discard x bytes of data. |
| void unread(int b) | It is used to pushes back the byte by copying it to the pushback buffer. |
| void unread(byte[] b) | It is used to pushes back the [array](https://www.javatpoint.com/array-in-java) of byte by copying it to the pushback buffer. |
| void reset() | It is used to reset the input stream. |
| void close() | It is used to close the input stream. |

### Example of PushbackInputStream class

1. **import** java.io.\*;
2. **public** **class** InputStreamExample {
3. **public** **static** **void** main(String[] args)**throws** Exception{
4. String srg = "1##2#34###12";
5. **byte** ary[] = srg.getBytes();
6. ByteArrayInputStream array = **new** ByteArrayInputStream(ary);
7. PushbackInputStream push = **new** PushbackInputStream(array);
8. **int** i;
9. **while**( (i = push.read())!= -1) {
10. **if**(i == '#') {
11. **int** j;
12. **if**( (j = push.read()) == '#'){
13. System.out.print("\*\*");
14. }**else** {
15. push.unread(j);
16. System.out.print((**char**)i);
17. }
18. }**else** {
19. System.out.print((**char**)i);
20. }
21. }
22. }
23. }

Output:

1. 1\*\*2#34\*\*#12

# Java PushbackReader Class

[Java](https://www.javatpoint.com/java-tutorial) PushbackReader [class](https://www.javatpoint.com/object-class) is a character [stream](https://www.javatpoint.com/java-8-stream) reader. It is used to pushes back a character into stream and [overrides](https://www.javatpoint.com/method-overriding-in-java) the [FilterReader](https://www.javatpoint.com/java-filterreader-class) class.

### Class declaration

Let' s see the declaration for java.io.PushbackReader class:

1. **public** **class** PushbackReader **extends** FilterReader

### Class Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int read() | It is used to read a single character. |
| void mark(int readAheadLimit) | It is used to mark the present position in a stream. |
| boolean ready() | It is used to tell whether the stream is ready to be read. |
| boolean markSupported() | It is used to tell whether the stream supports mark() operation. |
| long skip(long n) | It is used to skip the character. |
| void unread (int c) | It is used to pushes back the character by copying it to the pushback buffer. |
| void unread (char[] cbuf) | It is used to pushes back an array of character by copying it to the pushback buffer. |
| void reset() | It is used to reset the stream. |
| void close() | It is used to close the stream. |

### Example of PushbackReader class

1. **import** java.io.\*;
2. **public** **class** ReaderExample{
3. **public** **static** **void** main(String[] args) **throws** Exception {
4. **char** ary[] = {'1','-','-','2','-','3','4','-','-','-','5','6'};
5. CharArrayReader reader = **new** CharArrayReader(ary);
6. PushbackReader push = **new** PushbackReader(reader);
7. **int** i;
8. **while**( (i = push.read())!= -1) {
9. **if**(i == '-') {
10. **int** j;
11. **if**( (j = push.read()) == '-'){
12. System.out.print("#\*");
13. }**else** {
14. push.unread(j); // push back single character
15. System.out.print((**char**)i);
16. }
17. }**else** {
18. System.out.print((**char**)i);
19. }
20. }
21. }
22. }

Output

1. 1#\*2-34#\*-56

# Java StringWriter Class

Java StringWriter class is a character stream that collects output from string buffer, which can be used to construct a [string](https://www.javatpoint.com/java-string). The StringWriter class inherits the [Writer](https://www.javatpoint.com/java-writer-class) class.

In StringWriter class, system resources like [network](https://www.javatpoint.com/java-networking) [sockets](https://www.javatpoint.com/socket-programming) and [files](https://www.javatpoint.com/java-file-class) are not used, therefore closing the StringWriter is not necessary.

## Java StringWriter class declaration

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

1. **public** **class** StringWriter **extends** Writer

## Methods of StringWriter class

|  |  |
| --- | --- |
| **Method** | **Description** |
| void write(int c) | It is used to write the single character. |
| void write(String str) | It is used to write the string. |
| void write(String str, int off, int len) | It is used to write the portion of a string. |
| void write(char[] cbuf, int off, int len) | It is used to write the portion of an [array](https://www.javatpoint.com/array-in-java) of characters. |
| String toString() | It is used to return the buffer current value as a string. |
| StringBuffer getBuffer() | It is used t return the string buffer. |
| StringWriter append(char c) | It is used to append the specified character to the writer. |
| StringWriter append(CharSequence csq) | It is used to append the specified character sequence to the writer. |
| StringWriter append(CharSequence csq, int start, int end) | It is used to append the subsequence of specified character sequence to the writer. |
| void flush() | It is used to flush the stream. |
| void close() | It is used to close the stream. |

## Java StringWriter Example

Let's see the simple example of StringWriter using BufferedReader to read file data from the stream.

1. **import** java.io.\*;
2. **public** **class** StringWriterExample {
3. **public** **static** **void** main(String[] args) **throws** IOException {
4. **char**[] ary = **new** **char**[512];
5. StringWriter writer = **new** StringWriter();
6. FileInputStream input = **null**;
7. BufferedReader buffer = **null**;
8. input = **new** FileInputStream("D://testout.txt");
9. buffer = **new** BufferedReader(**new** InputStreamReader(input, "UTF-8"));
10. **int** x;
11. **while** ((x = buffer.read(ary)) != -1) {
12. writer.write(ary, 0, x);
13. }
14. System.out.println(writer.toString());
15. writer.close();
16. buffer.close();
17. }
18. }

testout.txt:

Javatpoint provides tutorial in Java, Spring, Hibernate, Android, PHP etc.

Output:

Javatpoint provides tutorial in Java, Spring, Hibernate, Android, PHP etc.

# Java StringReader Class

[Java](https://www.javatpoint.com/java-tutorial) StringReader [class](https://www.javatpoint.com/object-class) is a character [stream](https://www.javatpoint.com/java-8-stream) with [string](https://www.javatpoint.com/java-string) as a source. It takes an input string and changes it into character stream. It inherits [Reader class](https://www.javatpoint.com/java-reader-class).

In StringReader class, system resources like network sockets and files are not used, therefore closing the StringReader is not necessary.

## Java StringReader class declaration

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

1. **public** **class** StringReader **extends** Reader

## Methods of StringReader class

|  |  |
| --- | --- |
| **Method** | **Description** |
| int read() | It is used to read a single character. |
| int read(char[] cbuf, int off, int len) | It is used to read a character into a portion of an [array](https://www.javatpoint.com/array-in-java). |
| boolean ready() | It is used to tell whether the stream is ready to be read. |
| boolean markSupported() | It is used to tell whether the stream support mark() operation. |
| long skip(long ns) | It is used to skip the specified number of character in a stream |
| void mark(int readAheadLimit) | It is used to mark the mark the present position in a stream. |
| void reset() | It is used to reset the stream. |
| void close() | It is used to close the stream. |

## Java StringReader Example

1. **import** java.io.StringReader;
2. **public** **class** StringReaderExample {
3. **public** **static** **void** main(String[] args) **throws** Exception {
4. String srg = "Hello Java!! \nWelcome to Javatpoint.";
5. StringReader reader = **new** StringReader(srg);
6. **int** k=0;
7. **while**((k=reader.read())!=-1){
8. System.out.print((**char**)k);
9. }
10. }
11. }

Output:

Hello Java!!

Welcome to Javatpoint.

# Java - PipedWriter

The PipedWriter class is used to write [java](https://www.javatpoint.com/java-tutorial) pipe as a stream of characters. This class is used generally for writing text. Generally PipedWriter is connected to a [PipedReader](https://www.javatpoint.com/java-pipedreader-class) and used by different [threads](https://www.javatpoint.com/creating-thread).

### Constructor

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| PipedWriter() | It creates a piped writer that is not yet connected to a piped reader. |
| PipedWriter(PipedReader snk) | It creates a piped writer connected to the specified piped reader. |

### Method

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Method** |
| void | close() | It closes this piped output stream and releases any system resources associated with this stream. |
| void | connect(PipedReader snk) | It connects this piped writer to a receiver. |
| void | flush() | It flushes this output stream and forces any buffered output characters to be written out. |
| void | write(char[] cbuf, int off, int len) | It writes len characters from the specified character [array](https://www.javatpoint.com/array-in-java) starting at offset off to this piped output stream. |
| void | write(int c) | It writes the specified char to the piped output stream. |

## Example

1. **import** java.io.PipedReader;
2. **import** java.io.PipedWriter;
4. **public** **class** PipeReaderExample2 {
5. **public** **static** **void** main(String[] args) {
6. **try** {
8. **final** PipedReader read = **new** PipedReader();
9. **final** PipedWriter write = **new** PipedWriter(read);
11. Thread readerThread = **new** Thread(**new** Runnable() {
12. **public** **void** run() {
13. **try** {
14. **int** data = read.read();
15. **while** (data != -1) {
16. System.out.print((**char**) data);
17. data = read.read();
18. }
19. } **catch** (Exception ex) {
20. }
21. }
22. });
24. Thread writerThread = **new** Thread(**new** Runnable() {
25. **public** **void** run() {
26. **try** {
27. write.write("I love my country\n".toCharArray());
28. } **catch** (Exception ex) {
29. }
30. }
31. });
33. readerThread.start();
34. writerThread.start();
36. } **catch** (Exception ex) {
37. System.out.println(ex.getMessage());
38. }
40. }
41. }

Output:

I love my country

# Java - PipedReader

The PipedReader class is used to read the contents of a pipe as a stream of characters. This [class](https://www.javatpoint.com/object-and-class-in-java) is used generally to read text.

PipedReader class must be connected to the same [PipedWriter](https://www.javatpoint.com/java-pipedwriter-class) and are used by different [threads](https://www.javatpoint.com/creating-thread).

### Constructor

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| PipedReader(int pipeSize) | It creates a PipedReader so that it is not yet connected and uses the specified pipe size for the pipe's buffer. |
| PipedReader(PipedWriter src) | It creates a PipedReader so that it is connected to the piped writer src. |
| PipedReader(PipedWriter src, int pipeSize) | It creates a PipedReader so that it is connected to the piped writer src and uses the specified pipe size for the pipe's buffer. |
| PipedReader() | It creates a PipedReader so that it is not yet connected. |

### Method

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Method** |
| void | close() | It closes this piped stream and releases any system resources associated with the stream. |
| void | connect(PipedWriter src) | It causes this piped reader to be connected to the piped writer src. |
| int | read() | It reads the next character of data from this piped stream. |
| int | read(char[] cbuf, int off, int len) | It reads up to len characters of data from this piped stream into an [array](https://www.javatpoint.com/array-in-java) of characters. |
| boolean | ready() | It tells whether this stream is ready to be read. |

## Example

1. **import** java.io.PipedReader;
2. **import** java.io.PipedWriter;
4. **public** **class** PipeReaderExample2 {
5. **public** **static** **void** main(String[] args) {
6. **try** {
8. **final** PipedReader read = **new** PipedReader();
9. **final** PipedWriter write = **new** PipedWriter(read);
11. Thread readerThread = **new** Thread(**new** Runnable() {
12. **public** **void** run() {
13. **try** {
14. **int** data = read.read();
15. **while** (data != -1) {
16. System.out.print((**char**) data);
17. data = read.read();
18. }
19. } **catch** (Exception ex) {
20. }
21. }
22. });
24. Thread writerThread = **new** Thread(**new** Runnable() {
25. **public** **void** run() {
26. **try** {
27. write.write("I love my country\n".toCharArray());
28. } **catch** (Exception ex) {
29. }
30. }
31. });
33. readerThread.start();
34. writerThread.start();
36. } **catch** (Exception ex) {
37. System.out.println(ex.getMessage());
38. }
40. }
41. }

Output:

I love my country

# Java FilterWriter

Java FilterWriter class is an abstract [class](https://www.javatpoint.com/object-and-class-in-java) which is used to write filtered character streams.

The sub class of the FilterWriter should override some of its methods and it may provide additional methods and fields also.

### Fields

|  |  |  |  |
| --- | --- | --- | --- |
| **Modifier** | **Type** | **Field** | **Description** |
| protected | Writer | out | The underlying character-output stream. |

### Constructors

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Constructor** | **Description** |
| protected | FilterWriter(Writer out) | It creates InputStream class Object |

### Methods

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| void | close() | It closes the stream, flushing it first. |
| void | flush() | It flushes the stream. |
| void | write(char[] cbuf, int off, int len) | It writes a portion of an [array](https://www.javatpoint.com/array-in-java) of characters. |
| void | write(int c) | It writes a single character. |
| void | write(String str, int off, int len) | It writes a portion of a [string](https://www.javatpoint.com/java-string). |

## FilterWriter Example

1. **import** java.io.\*;
2. **class** CustomFilterWriter **extends** FilterWriter {
3. CustomFilterWriter(Writer out) {
4. **super**(out);
5. }
6. **public** **void** write(String str) **throws** IOException {
7. **super**.write(str.toLowerCase());
8. }
9. }
10. **public** **class** FilterWriterExample {
11. **public** **static** **void** main(String[] args) {
12. **try** {
13. FileWriter fw = **new** FileWriter("Record.txt");
14. CustomFilterWriter filterWriter = **new** CustomFilterWriter(fw);
15. filterWriter.write("I LOVE MY COUNTRY");
16. filterWriter.close();
17. FileReader fr = **new** FileReader("record.txt");
18. BufferedReader bufferedReader = **new** BufferedReader(fr);
19. **int** k;
20. **while** ((k = bufferedReader.read()) != -1) {
21. System.out.print((**char**) k);
22. }
23. bufferedReader.close();
24. } **catch** (IOException e) {
25. e.printStackTrace();
26. }
27. }
28. }

Output:

i love my country

While running the current program if the current working directory does not contain the file, a new file is created and CustomFileWriter will write the text "I LOVE MY COUNTRY" in lowercase to the file.

# Java FilterReader

Java FilterReader is used to perform filtering operation on [reader](https://www.javatpoint.com/java-reader-class) stream. It is an abstract class for reading filtered character streams.

The FilterReader provides default methods that passes all requests to the contained stream. Subclasses of FilterReader should override some of its methods and may also provide additional methods and fields.

### Field

|  |  |  |  |
| --- | --- | --- | --- |
| **Modifier** | **Type** | **Field** | **Description** |
| protected | Reader | in | The underlying character-input stream. |

### Constructors

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Constructor** | **Description** |
| protected | FilterReader(Reader in) | It creates a new filtered reader. |

### Method

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| void | close() | It closes the stream and releases any system resources associated with it. |
| void | mark(int readAheadLimit) | It marks the present position in the stream. |
| boolean | markSupported() | It tells whether this stream supports the mark() operation. |
| boolean | ready() | It tells whether this stream is ready to be read. |
| int | read() | It reads a single character. |
| int | read(char[] cbuf, int off, int len) | It reads characters into a portion of an [array](https://www.javatpoint.com/array-in-java). |
| void | reset() | It resets the stream. |
| long | skip(long n) | It skips characters. |

## Example

In this example, we are using "javaFile123.txt" file which contains "India is my country" text in it. Here, we are converting whitespace with question mark '?'.

1. **import** java.io.\*;
2. **class** CustomFilterReader **extends** FilterReader {
3. CustomFilterReader(Reader in) {
4. **super**(in);
5. }
6. **public** **int** read() **throws** IOException {
7. **int** x = **super**.read();
8. **if** ((**char**) x == ' ')
9. **return** ((**int**) '?');
10. **else**
11. **return** x;
12. }
13. }
14. **public** **class** FilterReaderExample {
15. **public** **static** **void** main(String[] args) {
16. **try**  {
17. Reader reader = **new** FileReader("javaFile123.txt");
18. CustomFilterReader fr = **new** CustomFilterReader(reader);
19. **int** i;
20. **while** ((i = fr.read()) != -1) {
21. System.out.print((**char**) i);
22. }
23. fr.close();
24. reader.close();
25. } **catch** (Exception e) {
26. e.getMessage();
27. }
28. }
29. }

Output:

India?is?my?country

# 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

1. **import** java.io.\*;
2. **public** **class** FileDemo {
3. **public** **static** **void** main(String[] args) {
5. **try** {
6. File file = **new** File("javaFile123.txt");
7. **if** (file.createNewFile()) {
8. System.out.println("New File is created!");
9. } **else** {
10. System.out.println("File already exists.");
11. }
12. } **catch** (IOException e) {
13. e.printStackTrace();
14. }
16. }
17. }

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

1. **import** java.io.\*;
2. **public** **class** FileExample {
3. **public** **static** **void** main(String[] args) {
4. File f=**new** File("/Users/sonoojaiswal/Documents");
5. String filenames[]=f.list();
6. **for**(String filename:filenames){
7. System.out.println(filename);
8. }
9. }
10. }

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

1. **import** java.io.\*;
2. **public** **class** FileExample {
3. **public** **static** **void** main(String[] args) {
4. File dir=**new** File("/Users/sonoojaiswal/Documents");
5. File files[]=dir.listFiles();
6. **for**(File file:files){
7. System.out.println(file.getName()+" Can Write: "+file.canWrite()+"
8. Is Hidden: "+file.isHidden()+" Length: "+file.length()+" bytes");
9. }
10. }
11. }

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

# Java FileDescriptor

FileDescriptor class serves as an handle to the underlying machine-specific structure representing an open file, an open [socket](https://www.javatpoint.com/socket-programming), or another source or sink of bytes. The handle can be err, in or out.

The FileDescriptor class is used to create a [FileInputStream](https://www.javatpoint.com/java-fileinputstream-class) or [FileOutputStream](https://www.javatpoint.com/java-fileoutputstream-class) to contain it.

### Field

|  |  |  |  |
| --- | --- | --- | --- |
| **Modifier** | **Type** | **Field** | **Description** |
| static | FileDescriptor | err | A handle to the standard error stream. |
| static | FileDescriptor | in | A handle to the standard input stream. |
| static | FileDescriptor | out | A handle to the standard output stream. |

### Constructors

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| FileDescriptor() | Constructs an (invalid) FileDescriptor [object](https://www.javatpoint.com/object-and-class-in-java). |

### Method

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| void | sync() | It force all system buffers to synchronize with the underlying device. |
| boolean | valid() | It tests if this file descriptor object is valid. |

## Java FileDescriptor Example

1. **import** java.io.\*;
2. **public** **class** FileDescriptorExample {
3. **public** **static** **void** main(String[] args) {
4. FileDescriptor fd = **null**;
5. **byte**[] b = { 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58 };
6. **try**  {
7. FileOutputStream fos = **new** FileOutputStream("Record.txt");
8. FileInputStream fis = **new** FileInputStream("Record.txt");
9. fd = fos.getFD();
10. fos.write(b);
11. fos.flush();
12. fd.sync();// confirms data to be written to the disk
13. **int** value = 0;
14. // for every available bytes
15. **while** ((value = fis.read()) != -1) {
16. **char** c = (**char**) value;// converts bytes to char
17. System.out.print(c);
18. }
19. System.out.println("\nSync() successfully executed!!");
20. } **catch** (Exception e) {
21. e.printStackTrace();
22. }
23. }
24. }

Output:

0123456789:

Sync() successfully executed!!

Record.txt:

0123456789:

# Java - RandomAccessFile

This [class](https://www.javatpoint.com/object-class) is used for reading and writing to random access file. A random access file behaves like a large [array](https://www.javatpoint.com/array-in-java) of bytes. There is a cursor implied to the array called file [pointer](https://www.javatpoint.com/c-pointers), by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is [thrown](https://www.javatpoint.com/throw-keyword). It is a type of IOException.

### Constructor

|  |  |
| --- | --- |
| [**Constructor**](https://www.javatpoint.com/java-constructor) | **Description** |
| RandomAccessFile(File file, [String](https://www.javatpoint.com/java-string)mode) | Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument. |
| RandomAccessFile(String name, String mode) | Creates a random access file stream to read from, and optionally to write to, a file with the specified name. |

### Method

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Method** |
| void | close() | It closes this random access file stream and releases any system resources associated with the stream. |
| FileChannel | getChannel() | It returns the unique [FileChannel](https://www.javatpoint.com/data-transfer-between-channels) object associated with this file. |
| int | readInt() | It reads a signed 32-bit integer from this file. |
| String | readUTF() | It reads in a string from this file. |
| void | seek(long pos) | It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. |
| void | writeDouble(double v) | It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first. |
| void | writeFloat(float v) | It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first. |
| void | write(int b) | It writes the specified byte to this file. |
| int | read() | It reads a byte of data from this file. |
| long | length() | It returns the length of this file. |
| void | seek(long pos) | It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. |

## Example

1. **import** java.io.IOException;
2. **import** java.io.RandomAccessFile;
4. **public** **class** RandomAccessFileExample {
5. **static** **final** String FILEPATH ="myFile.TXT";
6. **public** **static** **void** main(String[] args) {
7. **try** {
8. System.out.println(**new** String(readFromFile(FILEPATH, 0, 18)));
9. writeToFile(FILEPATH, "I love my country and my people", 31);
10. } **catch** (IOException e) {
11. e.printStackTrace();
12. }
13. }
14. **private** **static** **byte**[] readFromFile(String filePath, **int** position, **int** size)
15. **throws** IOException {
16. RandomAccessFile file = **new** RandomAccessFile(filePath, "r");
17. file.seek(position);
18. **byte**[] bytes = **new** **byte**[size];
19. file.read(bytes);
20. file.close();
21. **return** bytes;
22. }
23. **private** **static** **void** writeToFile(String filePath, String data, **int** position)
24. **throws** IOException {
25. RandomAccessFile file = **new** RandomAccessFile(filePath, "rw");
26. file.seek(position);
27. file.write(data.getBytes());
28. file.close();
29. }
30. }

The myFile.TXT contains text "This class is used for reading and writing to random access file."

after running the program it will contains

This class is used for reading I love my country and my peoplele.

# Java Scanner Class

Java Scanner class comes under the java.util package. Java has various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using a regular expression.

Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

## Java Scanner Class Declaration

1. **public** **final** **class** Scanner
2. **extends** Object
3. **implements** Iterator<String>

## Java Scanner Class Constructors

|  |  |  |
| --- | --- | --- |
| **SN** | **Constructor** | **Description** |
| 1) | Scanner(File source) | It constructs a new Scanner that produces values scanned from the specified file. |
| 2) | Scanner(File source, String charsetName) | It constructs a new Scanner that produces values scanned from the specified file. |
| 3) | Scanner(InputStream source) | It constructs a new Scanner that produces values scanned from the specified input stream. |
| 4) | Scanner(InputStream source, String charsetName) | It constructs a new Scanner that produces values scanned from the specified input stream. |
| 5) | Scanner(Readable source) | It constructs a new Scanner that produces values scanned from the specified source. |
| 6) | Scanner(String source) | It constructs a new Scanner that produces values scanned from the specified string. |
| 7) | Scanner(ReadableByteChannel source) | It constructs a new Scanner that produces values scanned from the specified channel. |
| 8) | Scanner(ReadableByteChannel source, String charsetName) | It constructs a new Scanner that produces values scanned from the specified channel. |
| 9) | Scanner(Path source) | It constructs a new Scanner that produces values scanned from the specified file. |
| 10) | Scanner(Path source, String charsetName) | It constructs a new Scanner that produces values scanned from the specified file. |

## Java Scanner Class Methods

The following are the list of Scanner methods:

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **Modifier & Type** | **Method** | **Description** |
| 1) | void | [close()](https://www.javatpoint.com/post/java-scanner-close-method) | It is used to close this scanner. |
| 2) | pattern | [delimiter()](https://www.javatpoint.com/post/java-scanner-delimiter-method) | It is used to get the Pattern which the Scanner class is currently using to match delimiters. |
| 3) | Stream<MatchResult> | findAll() | It is used to find a stream of match results that match the provided pattern string. |
| 4) | String | [findInLine()](https://www.javatpoint.com/post/java-scanner-findinline-method) | It is used to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters. |
| 5) | string | [findWithinHorizon()](https://www.javatpoint.com/post/java-scanner-findwithinhorizon-method) | It is used to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters. |
| 6) | boolean | [hasNext()](https://www.javatpoint.com/post/java-scanner-hasnext-method) | It returns true if this scanner has another token in its input. |
| 7) | boolean | [hasNextBigDecimal()](https://www.javatpoint.com/post/java-scanner-hasnextbigdecimal-method) | It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextBigDecimal() method or not. |
| 8) | boolean | [hasNextBigInteger()](https://www.javatpoint.com/post/java-scanner-hasnextbiginteger-method) | It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextBigDecimal() method or not. |
| 9) | boolean | [hasNextBoolean()](https://www.javatpoint.com/post/java-scanner-hasnextboolean-method) | It is used to check if the next token in this scanner's input can be interpreted as a Boolean using the nextBoolean() method or not. |
| 10) | boolean | [hasNextByte()](https://www.javatpoint.com/post/java-scanner-hasnextbyte-method) | It is used to check if the next token in this scanner's input can be interpreted as a Byte using the nextBigDecimal() method or not. |
| 11) | boolean | [hasNextDouble()](https://www.javatpoint.com/post/java-scanner-hasnextdouble-method) | It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextByte() method or not. |
| 12) | boolean | [hasNextFloat()](https://www.javatpoint.com/post/java-scanner-hasnextfloat-method) | It is used to check if the next token in this scanner's input can be interpreted as a Float using the nextFloat() method or not. |
| 13) | boolean | [hasNextInt()](https://www.javatpoint.com/post/java-scanner-hasnextint-method) | It is used to check if the next token in this scanner's input can be interpreted as an int using the nextInt() method or not. |
| 14) | boolean | [hasNextLine()](https://www.javatpoint.com/post/java-scanner-hasnextline-method) | It is used to check if there is another line in the input of this scanner or not. |
| 15) | boolean | [hasNextLong()](https://www.javatpoint.com/post/java-scanner-hasnextlong-method) | It is used to check if the next token in this scanner's input can be interpreted as a Long using the nextLong() method or not. |
| 16) | boolean | [hasNextShort()](https://www.javatpoint.com/post/java-scanner-hasnextshort-method) | It is used to check if the next token in this scanner's input can be interpreted as a Short using the nextShort() method or not. |
| 17) | IOException | [ioException()](https://www.javatpoint.com/post/java-scanner-ioexception-method) | It is used to get the IOException last thrown by this Scanner's readable. |
| 18) | Locale | [locale()](https://www.javatpoint.com/post/java-scanner-locale-method) | It is used to get a Locale of the Scanner class. |
| 19) | MatchResult | [match()](https://www.javatpoint.com/post/java-scanner-match-method) | It is used to get the match result of the last scanning operation performed by this scanner. |
| 20) | String | [next()](https://www.javatpoint.com/post/java-scanner-next-method) | It is used to get the next complete token from the scanner which is in use. |
| 21) | BigDecimal | [nextBigDecimal()](https://www.javatpoint.com/post/java-scanner-nextbigdecimal-method) | It scans the next token of the input as a BigDecimal. |
| 22) | BigInteger | [nextBigInteger()](https://www.javatpoint.com/post/java-scanner-nextbiginteger-method) | It scans the next token of the input as a BigInteger. |
| 23) | boolean | [nextBoolean()](https://www.javatpoint.com/post/java-scanner-nextboolean-method) | It scans the next token of the input into a boolean value and returns that value. |
| 24) | byte | [nextByte()](https://www.javatpoint.com/post/java-scanner-nextbyte-method) | It scans the next token of the input as a byte. |
| 25) | double | [nextDouble()](https://www.javatpoint.com/post/java-scanner-nextdouble-method) | It scans the next token of the input as a double. |
| 26) | float | [nextFloat()](https://www.javatpoint.com/post/java-scanner-nextfloat-method) | It scans the next token of the input as a float. |
| 27) | int | [nextInt()](https://www.javatpoint.com/post/java-scanner-nextint-method) | It scans the next token of the input as an Int. |
| 28) | String | [nextLine()](https://www.javatpoint.com/post/java-scanner-nextline-method) | It is used to get the input string that was skipped of the Scanner object. |
| 29) | long | [nextLong()](https://www.javatpoint.com/post/java-scanner-nextlong-method) | It scans the next token of the input as a long. |
| 30) | short | [nextShort()](https://www.javatpoint.com/post/java-scanner-nextshort-method) | It scans the next token of the input as a short. |
| 31) | int | [radix()](https://www.javatpoint.com/post/java-scanner-radix-method) | It is used to get the default radix of the Scanner use. |
| 32) | void | [remove()](https://www.javatpoint.com/post/java-scanner-remove-method) | It is used when remove operation is not supported by this implementation of Iterator. |
| 33) | Scanner | [reset()](https://www.javatpoint.com/post/java-scanner-reset-method) | It is used to reset the Scanner which is in use. |
| 34) | Scanner | [skip()](https://www.javatpoint.com/post/java-scanner-skip-method) | It skips input that matches the specified pattern, ignoring delimiters |
| 35) | Stream<String> | [tokens()](https://www.javatpoint.com/post/java-scanner-tokens-method) | It is used to get a stream of delimiter-separated tokens from the Scanner object which is in use. |
| 36) | String | [toString()](https://www.javatpoint.com/post/java-scanner-tostring-method) | It is used to get the string representation of Scanner using. |
| 37) | Scanner | [useDelimiter()](https://www.javatpoint.com/post/java-scanner-usedelimiter-method) | It is used to set the delimiting pattern of the Scanner which is in use to the specified pattern. |
| 38) | Scanner | [useLocale()](https://www.javatpoint.com/post/java-scanner-uselocale-method) | It is used to sets this scanner's locale object to the specified locale. |
| 39) | Scanner | [useRadix()](https://www.javatpoint.com/post/java-scanner-useradix-method) | It is used to set the default radix of the Scanner which is in use to the specified radix. |

## Example 1

1. **import** java.util.\*;
2. **public** **class** ScannerClassExample1 {
3. **public** **static** **void** main(String args[]){
4. String s = "Hello, This is JavaTpoint.";
5. //Create scanner Object and pass string in it
6. Scanner scan = **new** Scanner(s);
7. //Check if the scanner has a token
8. System.out.println("Boolean Result: " + scan.hasNext());
9. //Print the string
10. System.out.println("String: " +scan.nextLine());
11. scan.close();
12. System.out.println("--------Enter Your Details-------- ");
13. Scanner in = **new** Scanner(System.in);
14. System.out.print("Enter your name: ");
15. String name = in.next();
16. System.out.println("Name: " + name);
17. System.out.print("Enter your age: ");
18. **int** i = in.nextInt();
19. System.out.println("Age: " + i);
20. System.out.print("Enter your salary: ");
21. **double** d = in.nextDouble();
22. System.out.println("Salary: " + d);
23. in.close();
24. }
25. }

**Output:**

Boolean Result: true

String: Hello, This is JavaTpoint.

-------Enter Your Details---------

Enter your name: Abhishek

Name: Abhishek

Enter your age: 23

Age: 23

Enter your salary: 25000

Salary: 25000.0

## Example 2

1. **import** java.util.\*;
2. **public** **class** ScannerClassExample2 {
3. **public** **static** **void** main(String args[]){
4. String str = "Hello/This is JavaTpoint/My name is Abhishek.";
5. //Create scanner with the specified String Object
6. Scanner scanner = **new** Scanner(str);
7. System.out.println("Boolean Result: "+scanner.hasNextBoolean());
8. //Change the delimiter of this scanner
9. scanner.useDelimiter("/");
10. //Printing the tokenized Strings
11. System.out.println("---Tokenizes String---");
12. **while**(scanner.hasNext()){
13. System.out.println(scanner.next());
14. }
15. //Display the new delimiter
16. System.out.println("Delimiter used: " +scanner.delimiter());
17. scanner.close();
18. }
19. }

**Output:**

Boolean Result: false

---Tokenizes String---

Hello

This is JavaTpoint

My name is Abhishek.

Delimiter used: /

# java.io.PrintStream class

The PrintStream class provides methods to write data to another stream. The PrintStream class automatically flushes the data so there is no need to call flush() method. Moreover, its methods don't throw IOException.

### Commonly used methods of PrintStream class

|  |
| --- |
| There are many methods in PrintStream class. Let's see commonly used methods of PrintStream class:   * **public void print(boolean b):** it prints the specified boolean value. * **public void print(char c):** it prints the specified char value. * **public void print(char[] c):** it prints the specified character array values. * **public void print(int i):** it prints the specified int value. * **public void print(long l):** it prints the specified long value. * **public void print(float f):** it prints the specified float value. * **public void print(double d):** it prints the specified double value. * **public void print(String s):** it prints the specified string value. * **public void print(Object obj):** it prints the specified object value. * **public void println(boolean b):** it prints the specified boolean value and terminates the line. * **public void println(char c):** it prints the specified char value and terminates the line. * **public void println(char[] c):** it prints the specified character array values and terminates the line. * **public void println(int i):** it prints the specified int value and terminates the line. * **public void println(long l):** it prints the specified long value and terminates the line. * **public void println(float f):** it prints the specified float value and terminates the line. * **public void println(double d):** it prints the specified double value and terminates the line. * **public void println(String s):** it prints the specified string value and terminates the line./li> * **public void println(Object obj):** it prints the specified object value and terminates the line. * **public void println():** it terminates the line only. * **public void printf(Object format, Object... args):** it writes the formatted string to the current stream. * **public void printf(Locale l, Object format, Object... args):** it writes the formatted string to the current stream. * **public void format(Object format, Object... args):** it writes the formatted string to the current stream using specified format. * **public void format(Locale l, Object format, Object... args):** it writes the formatted string to the current stream using specified format. |

### Example of java.io.PrintStream class

|  |
| --- |
| In this example, we are simply printing integer and string values. |

1. **import** java.io.\*;
2. **class** PrintStreamTest{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. FileOutputStream fout=**new** FileOutputStream("mfile.txt");
5. PrintStream pout=**new** PrintStream(fout);
6. pout.println(1900);
7. pout.println("Hello Java");
8. pout.println("Welcome to Java");
9. pout.close();
10. fout.close();
11. }
12. }

[download this PrintStream example](https://www.javatpoint.com/src/io/printstream1.zip)

### Example of printf() method of java.io.PrintStream class:

|  |
| --- |
| Let's see the simple example of printing integer value by format specifier. |

1. **class** PrintStreamTest{
2. **public** **static** **void** main(String args[]){
3. **int** a=10;
4. System.out.printf("%d",a);//Note, out is the object of PrintStream class
6. }
7. }

Output:10

# Compressing and Decompressing File

The DeflaterOutputStream and InflaterInputStream classes provide mechanism to compress and decompress the data in the **deflate compression format**.

## DeflaterOutputStream class

The DeflaterOutputStream class is used to compress the data in the deflate compression format. It provides facility to the other compression filters, such as GZIPOutputStream.

### Example of Compressing file using DeflaterOutputStream class

In this example, we are reading data of a file and compressing it into another file using DeflaterOutputStream class. You can compress any file, here we are compressing the Deflater.java file

1. **import** java.io.\*;
2. **import** java.util.zip.\*;
3. **class** Compress{
4. **public** **static** **void** main(String args[]){
5. **try**{
6. FileInputStream fin=**new** FileInputStream("Deflater.java");
7. FileOutputStream fout=**new** FileOutputStream("def.txt");
8. DeflaterOutputStream out=**new** DeflaterOutputStream(fout);
10. **int** i;
11. **while**((i=fin.read())!=-1){
12. out.write((**byte**)i);
13. out.flush();
14. }
15. fin.close();
16. out.close();
17. }**catch**(Exception e){System.out.println(e);}
18. System.out.println("rest of the code");
19. }
20. }

## InflaterInputStream class

The InflaterInputStream class is used to decompress the file in the deflate compression format. It provides facility to the other decompression filters, such as GZIPInputStream class.

### Example of decompressing file using InflaterInputStream class

In this example, we are decompressing the compressed file def.txt into D.java .

1. **import** java.io.\*;
2. **import** java.util.zip.\*;
3. **class** DeCompress{
4. **public** **static** **void** main(String args[]){
5. **try**{
6. FileInputStream fin=**new** FileInputStream("def.txt");
7. InflaterInputStream in=**new** InflaterInputStream(fin);
8. FileOutputStream fout=**new** FileOutputStream("D.java");
10. **int** i;
11. **while**((i=in.read())!=-1){
12. fout.write((**byte**)i);
13. fout.flush();
14. }
15. fin.close();
16. fout.close();
17. in.close();
18. }**catch**(Exception e){System.out.println(e);}
19. System.out.println("rest of the code");
20. }
21. }