Useful methods of OutputStream

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public void write(int)throws IOException | is used to write a byte to the current output stream. |
| 2) public void write(byte[])throws IOException | is used to write an array of byte to the current output stream. |
| 3) public void flush()throws IOException | flushes the current output stream. |
| 4) public void close()throws IOException | is used to close the current output stream. |

### OutputStream Hierarchy

Java output stream hierarchy

### Useful methods of InputStream

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public abstract int read()throws IOException | reads the next byte of data from the input stream. It returns -1 at the end of the file. |
| 2) public int available()throws IOException | returns an estimate of the number of bytes that can be read from the current input stream. |
| 3) public void close()throws IOException | is used to close the current input stream. |

InputStream Hierarchy

Java input stream hierarchy

# Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a [file](https://www.javatpoint.com/java-file-class).

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use [FileWriter](https://www.javatpoint.com/java-filterwriter-class) than FileOutputStream.

## FileOutputStream class declaration

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

1. **public** **class** FileOutputStream **extends** OutputStream

## FileOutputStream class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| protected void finalize() | It is used to clean up the connection with the file output stream. |
| void write(byte[] ary) | It is used to write **ary.length** bytes from the byte [array](https://www.javatpoint.com/array-in-java) to the file output stream. |
| void write(byte[] ary, int off, int len) | It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream. |
| void write(int b) | It is used to write the specified byte to the file output stream. |
| FileChannel getChannel() | It is used to return the file channel object associated with the file output stream. |
| FileDescriptor getFD() | It is used to return the file descriptor associated with the stream. |
| void close() | It is used to closes the file output stream. |

## Java FileOutputStream Example 1: write byte

**import** java.io.FileOutputStream;

**public** **class** FileOutputStreamExample {

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

**try**{

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

             fout.write(65);

             fout.close();

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

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

      }

}

Output:

Play Video

Success...

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

testout.txt

A

## Java FileOutputStream example 2: write string

**import** java.io.FileOutputStream;

**public** **class** FileOutputStreamExample {

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

**try**{

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

             String s="Welcome to javaTpoint.";

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

             fout.write(b);

             fout.close();

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

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

}

}

Output:

Success...

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

testout.txt

Welcome to javaTpoint.

# Java FileInputStream Class

Java FileInputStream class obtains input bytes from a [file](https://www.javatpoint.com/java-file-class). It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use [FileReader](https://www.javatpoint.com/java-filereader-class) class.

## Java FileInputStream class declaration

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

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

## Java FileInputStream class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int available() | It is used to return the estimated number of bytes that can be read from the input stream. |
| int read() | It is used to read the byte of data from the input stream. |
| int read(byte[] b) | It is used to read up to **b.length** bytes of data from the input stream. |
| int read(byte[] b, int off, int len) | It is used to read up to **len** bytes of data from the input stream. |
| long skip(long x) | It is used to skip over and discards x bytes of data from the input stream. |
| FileChannel getChannel() | It is used to return the unique FileChannel object associated with the file input stream. |
| FileDescriptor getFD() | It is used to return the [FileDescriptor](https://www.javatpoint.com/java-filedescriptor-class) object. |
| protected void finalize() | It is used to ensure that the close method is call when there is no more reference to the file input stream. |
| void close() | It is used to closes the [stream](https://www.javatpoint.com/java-8-stream). |

## Java FileInputStream example 1: read single character

**import** java.io.FileInputStream;

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

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

**try**{

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

**int** i=fin.read();

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

            fin.close();

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

         }

        }

**Note:** Before running the code, a text file named as **"testout.txt"**is required to be created. In this file, we are having following content:

Welcome to 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

**package** com.javatpoint;

**import** java.io.FileInputStream;

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

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

**try**{

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

**int** i=0;

**while**((i=fin.read())!=-1){

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

            }

            fin.close();

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

         }

        }

Output:

Welcome to javaTpoint

# Java BufferedOutputStream Class

Java BufferedOutputStream [class](https://www.javatpoint.com/object-and-class-in-java) is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

For adding the buffer in an OutputStream, use the BufferedOutputStream class. Let's see the syntax for adding the buffer in an OutputStream:

1. OutputStream os= **new** BufferedOutputStream(**new** FileOutputStream("D:\\IO Package\\testout.txt"));

## Java BufferedOutputStream class declaration

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

1. **public** **class** BufferedOutputStream **extends** FilterOutputStream

## Java BufferedOutputStream class constructors

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| BufferedOutputStream(OutputStream os) | It creates the new buffered output stream which is used for writing the data to the specified output stream. |
| BufferedOutputStream(OutputStream os, int size) | It creates the new buffered output stream which is used for writing the data to the specified output stream with a specified buffer size. |

## Java BufferedOutputStream class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| void write(int b) | It writes the specified byte to the buffered output stream. |
| void write(byte[] b, int off, int len) | It write the bytes from the specified byte-input stream into a specified byte [array](https://www.javatpoint.com/array-in-java), starting with the given offset |
| void flush() | It flushes the buffered output stream. |

## Example of BufferedOutputStream class:

In this example, we are writing the textual information in the BufferedOutputStream object which is connected to the [FileOutputStream](https://www.javatpoint.com/java-fileoutputstream-class) [object](https://www.javatpoint.com/object-and-class-in-java). The flush() flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.

Play Video

**package** com.javatpoint;

**import** java.io.\*;

**public** **class** BufferedOutputStreamExample{

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

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

BufferedOutputStream bout=**new** BufferedOutputStream(fout);

String s="Welcome to javaTpoint.";

**byte** b[]=s.getBytes();

bout.write(b);

bout.flush();

bout.close();

fout.close();

System.out.println("success");

}

}

Output:

Success

testout.txt

Welcome to javaTpoint.

# Java BufferedInputStream Class

Java BufferedInputStream [class](https://www.javatpoint.com/object-and-class-in-java) is used to read information from [stream](https://www.javatpoint.com/java-8-stream). It internally uses buffer mechanism to make the performance fast.

The important points about BufferedInputStream are:

* When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
* When a BufferedInputStream is created, an internal buffer [array](https://www.javatpoint.com/array-in-java) is created.

## Java BufferedInputStream class declaration

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

**public** **class** BufferedInputStream **extends** FilterInputStream

## Java BufferedInputStream class constructors

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| BufferedInputStream(InputStream IS) | It creates the BufferedInputStream and saves it argument, the input stream IS, for later use. |
| BufferedInputStream(InputStream IS, int size) | It creates the BufferedInputStream with a specified buffer size and saves it argument, the input stream IS, for later use. |

## Java BufferedInputStream class methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| int available() | It returns an estimate number of bytes that can be read from the input stream without blocking by the next invocation method for the input stream. |
| int read() | It read the next byte of data from the input stream. |
| int read(byte[] b, int off, int ln) | It read the bytes from the specified byte-input stream into a specified byte array, starting with the given offset. |
| void close() | It closes the input stream and releases any of the system resources associated with the stream. |
| void reset() | It repositions the stream at a position the mark method was last called on this input stream. |
| void mark(int readlimit) | It sees the general contract of the mark method for the input stream. |
| long skip(long x) | It skips over and discards x bytes of data from the input stream. |
| boolean markSupported() | It tests for the input stream to support the mark and reset methods. |

### Example of Java BufferedInputStream

Let's see the simple example to read data of [file](https://www.javatpoint.com/java-file-class) using BufferedInputStream:

**import** java.io.\*;

**public** **class** BufferedInputStreamExample{

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

**try**{

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

BufferedInputStream bin=**new** BufferedInputStream(fin);

**int** i;

**while**((i=bin.read())!=-1){

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

}

bin.close();

fin.close();

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

}

}

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

javaTpoint

Output:

javaTpoint

# 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.

Play Video

**import** java.io.\*;

**public** **class** OutputExample {

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

FileOutputStream file = **new** FileOutputStream(D:\\testout.txt);

DataOutputStream data = **new** DataOutputStream(file);

data.writeInt(65);

data.flush();

data.close();

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

}

}

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:

**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.

**import** java.io.\*;

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

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

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

DataInputStream inst = **new** DataInputStream(input);

**int** count = input.available();

**byte**[] ary = **new** **byte**[count];

inst.read(ary);

**for** (**byte** bt : ary) {

**char** k = (**char**) bt;

System.out.print(k+"-");

}

}

}

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

**import** java.io.\*;

**public** **class** FilterExample {

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

File data = **new** File("D:\\testout.txt");

FileOutputStream file = **new** FileOutputStream(data);

FilterOutputStream filter = **new** FilterOutputStream(file);

String s="Welcome to javaTpoint.";

**byte** b[]=s.getBytes();

filter.write(b);

filter.flush();

filter.close();

file.close();

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

}

}

Output:

Success...

testout.txtPlay Video

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

**import** java.io.\*;

**public** **class** FilterExample {

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

File data = **new** File("D:\\testout.txt");

FileInputStream  file = **new** FileInputStream(data);

FilterInputStream filter = **new** BufferedInputStream(file);

**int** k =0;

**while**((k=filter.read())!=-1){

System.out.print((**char**)k);

}

file.close();

filter.close();

}

}

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

Welcome to javatpoint

Output:

Welcome to javatpoint

# 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

**import** java.io.\*;

**public** **class** WriterExample {

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

**try** {

Writer w = **new** FileWriter("output.txt");

String content = "I love my country";

w.write(content);

w.close();

System.out.println("Done");

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

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

**import** java.io.\*;

**public** **class** ReaderExample {

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

**try** {

Reader reader = **new** FileReader("file.txt");

**int** data = reader.read();

**while** (data != -1) {

System.out.print((**char**) data);

data = reader.read();

}

reader.close();

} **catch** (Exception ex) {

System.out.println(ex.getMessage());

}

}

}

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.

**import** java.io.FileWriter;

**public** **class** FileWriterExample {

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

**try**{

FileWriter fw=**new** FileWriter("D:\\testout.txt");

fw.write("Welcome to javaTpoint.");

fw.close();

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

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

}

}

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.

**import** java.io.FileReader;

**public** **class** FileReaderExample {

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

FileReader fr=**new** FileReader("D:\\testout.txt");

**int** i;

**while**((i=fr.read())!=-1)

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

fr.close();

}

}

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

Welcome to javaTpoint.

Output:

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:

**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.

**import** java.io.FileReader;

**public** **class** FileReaderExample {

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

FileReader fr=**new** FileReader("D:\\testout.txt");

**int** i;

**while**((i=fr.read())!=-1)

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

fr.close();

}

}

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

Welcome to javaTpoint.

Output:

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.

**import** java.io.\*;

**public** **class** BufferedReaderExample {

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

FileReader fr=**new** FileReader("D:\\testout.txt");

BufferedReader br=**new** BufferedReader(fr);

**int** i;

**while**((i=br.read())!=-1){

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

}

br.close();

fr.close();

}

}

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

Play Video

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.

**import** java.io.\*;

**public** **class** BufferedReaderExample{

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

InputStreamReader r=**new** InputStreamReader(System.in);

BufferedReader br=**new** BufferedReader(r);

System.out.println("Enter your name");

String name=br.readLine();

System.out.println("Welcome "+name);

}

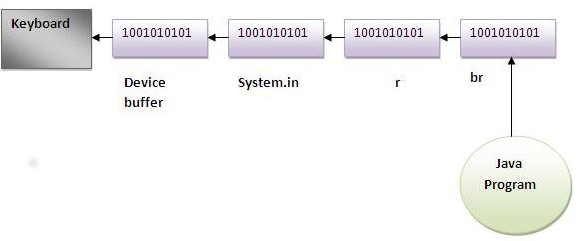
}

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.

**import** java.io.\*;

**public** **class** BufferedReaderExample{

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

InputStreamReader r=**new** InputStreamReader(System.in);

BufferedReader br=**new** BufferedReader(r);

String name="";

**while**(!name.equals("stop")){

System.out.println("Enter data: ");

name=br.readLine();

System.out.println("data is: "+name);

}

br.close();

r.close();

}

}

Output:

Enter data: Nakul

data is: Nakul

Enter data: 12

data is: 12

Enter data: stop

data is: stop

# 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:

**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.

**package** com.javatpoint;

**import** java.io.CharArrayWriter;

**import** java.io.FileWriter;

**public** **class** CharArrayWriterExample {

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

CharArrayWriter out=**new** CharArrayWriter();

out.write("Welcome to javaTpoint");

FileWriter f1=**new** FileWriter("D:\\a.txt");

FileWriter f2=**new** FileWriter("D:\\b.txt");

FileWriter f3=**new** FileWriter("D:\\c.txt");

FileWriter f4=**new** FileWriter("D:\\d.txt");

out.writeTo(f1);

out.writeTo(f2);

out.writeTo(f3);

out.writeTo(f4);

f1.close();

f2.close();

f3.close();

f4.close();

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

}

}

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 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.

**import** java.io.CharArrayReader;

**public** **class** CharArrayExample{

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

**char**[] ary = { 'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't' };

CharArrayReader reader = **new** CharArrayReader(ary);

**int** k = 0;

// Read until the end of a file

**while** ((k = reader.read()) != -1) {

**char** ch = (**char**) k;

System.out.print(ch + " : ");

System.out.println(k);

}

}

}

Output

j : 106

a : 97

v : 118

a : 97

t : 116

p : 112

o : 111

i : 105

n : 110

t : 116

# 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:

**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.

**import** java.io.FileOutputStream;

**import** java.io.PrintStream;

**public** **class** PrintStreamTest{

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

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

PrintStream pout=**new** PrintStream(fout);

pout.println(2016);

pout.println("Hello Java");

pout.println("Welcome to Java");

pout.close();

fout.close();

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

}

}

Output

Play Video

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.

**class** PrintStreamTest{

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

**int** a=19;

System.out.printf("%d",a); //Note: out is the object of printstream

}

}

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.

**import** java.io.File;

**import** java.io.PrintWriter;

**public** **class** PrintWriterExample {

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

//Data to write on Console using PrintWriter

PrintWriter writer = **new** PrintWriter(System.out);

writer.write("Javatpoint provides tutorials of all technology.");

writer.flush();

writer.close();

//Data to write in File using PrintWriter

PrintWriter writer1 =**null**;

writer1 = **new** PrintWriter(**new** File("D:\\testout.txt"));

writer1.write("Like Java, Spring, Hibernate, Android, PHP etc.");

writer1.flush();

writer1.close();

}

}

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

**public** **class** OutputStreamWriterExample {

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

**try** {

            OutputStream outputStream = **new** FileOutputStream("output.txt");

            Writer outputStreamWriter = **new** OutputStreamWriter(outputStream);

            outputStreamWriter.write("Hello World");

            outputStreamWriter.close();

        } **catch** (Exception e) {

            e.getMessage();

        }

    }

}

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**

**public** **class** InputStreamReaderExample {

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

**try**  {

            InputStream stream = **new** FileInputStream("file.txt");

            Reader reader = **new** InputStreamReader(stream);

**int** data = reader.read();

**while** (data != -1) {

                System.out.print((**char**) data);

                data = reader.read();

            }

        } **catch** (Exception e) {

            e.printStackTrace();

        }

    }

}

Output:

I love my country

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

reads Character by character from the file

# Java File Class

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

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

### Fields

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

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

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

### Useful Methods

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

## Java File Example 1

**import** java.io.\*;

**public** **class** FileDemo {

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

**try** {

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

**if** (file.createNewFile()) {

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

            } **else** {

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

            }

        } **catch** (IOException e) {

            e.printStackTrace();

        }

    }

}

Output:

New File is created!

## Java File Example 2

**import** java.io.\*;

**public** **class** FileDemo2 {

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

        String path = "";

**boolean** bool = **false**;

**try** {

            // createing  new files

            File file  = **new** File("testFile1.txt");

            file.createNewFile();

            System.out.println(file);

            // createing new canonical from file object

            File file2 = file.getCanonicalFile();

            // returns true if the file exists

            System.out.println(file2);

            bool = file2.exists();

            // returns absolute pathname

            path = file2.getAbsolutePath();

            System.out.println(bool);

            // if file exists

**if** (bool) {

                // prints

                System.out.print(path + " Exists? " + bool);

            }

        } **catch** (Exception e) {

            // if any error occurs

            e.printStackTrace();

        }

    }

}

Output:

Play Video

testFile1.txt

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

true

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

## Java File Example 3

**import** java.io.\*;

**public** **class** FileExample {

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

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

    String filenames[]=f.list();

**for**(String filename:filenames){

        System.out.println(filename);

    }

}

}

Output:

"info.properties"

"info.properties".rtf

.DS\_Store

.localized

Alok news

apache-tomcat-9.0.0.M19

apache-tomcat-9.0.0.M19.tar

bestreturn\_org.rtf

BIODATA.pages

BIODATA.pdf

BIODATA.png

struts2jars.zip

workspace

## Java File Example 4

**import** java.io.\*;

**public** **class** FileExample {

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

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

    File files[]=dir.listFiles();

**for**(File file:files){

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

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

    }

}

}

Output:

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

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

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

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

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

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

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

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

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

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

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

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

# StringTokenizer in Java

[StringTokenizer](https://www.javatpoint.com/string-tokenizer-in-java#StringTokenizer)

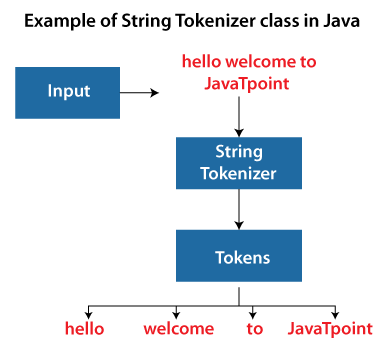
[Methods of StringTokenizer](https://www.javatpoint.com/string-tokenizer-in-java#Methods)

[Example of StringTokenizer](https://www.javatpoint.com/string-tokenizer-in-java#Example)

The **java.util.StringTokenizer** class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

In the StringTokenizer class, the delimiters can be provided at the time of creation or one by one to the tokens.



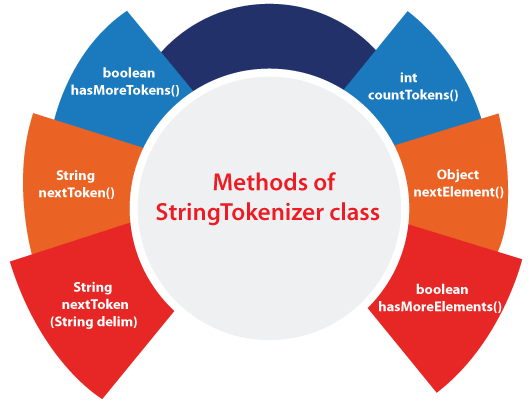
### Constructors of the StringTokenizer Class

There are 3 constructors defined in the StringTokenizer class.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| StringTokenizer(String str) | It creates StringTokenizer with specified string. |
| StringTokenizer(String str, String delim) | It creates StringTokenizer with specified string and delimiter. |
| StringTokenizer(String str, String delim, boolean returnValue) | It creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens. |

### Methods of the StringTokenizer Class

The six useful methods of the StringTokenizer class are as follows:



|  |  |
| --- | --- |
| **Methods** | **Description** |
| boolean hasMoreTokens() | It checks if there is more tokens available. |
| String nextToken() | It returns the next token from the StringTokenizer object. |
| String nextToken(String delim) | It returns the next token based on the delimiter. |
| boolean hasMoreElements() | It is the same as hasMoreTokens() method. |
| Object nextElement() | It is the same as nextToken() but its return type is Object. |
| int countTokens() | It returns the total number of tokens. |

### Example of StringTokenizer Class

Let's see an example of the StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

**Simple.java**

**import** java.util.StringTokenizer;

**public** **class** Simple{

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

   StringTokenizer st = **new** StringTokenizer("my name is khan"," ");

**while** (st.hasMoreTokens()) {

         System.out.println(st.nextToken());

     }

   }

}

**Output:**

my

name

is

khan

The above Java code, demonstrates the use of StringTokenizer class and its methods hasMoreTokens() and nextToken().

### Example of nextToken(String delim) method of the StringTokenizer class

**Test.java**

**import** java.util.\*;

**public** **class** Test {

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

       StringTokenizer st = **new** StringTokenizer("my,name,is,khan");

      // printing next token

      System.out.println("Next token is : " + st.nextToken(","));

   }

}

**Output:**

Next token is : my

### Example of hasMoreTokens() method of the StringTokenizer class

This method returns true if more tokens are available in the tokenizer String otherwise returns false.

**StringTokenizer1.java**

**import** java.util.StringTokenizer;

**public** **class** StringTokenizer1

{

 /\* Driver Code \*/

**public** **static** **void** main(String args[])

 {

   /\* StringTokenizer object \*/

   StringTokenizer st = **new** StringTokenizer("Demonstrating methods from StringTokenizer class"," ");

     /\* Checks if the String has any more tokens \*/

**while** (st.hasMoreTokens())

     {

         System.out.println(st.nextToken());

     }

 }    }

**Output:**

Demonstrating

methods

from

StringTokenizer

class

The above Java program shows the use of two methods hasMoreTokens() and nextToken() of StringTokenizer class.

### Example of hasMoreElements() method of the StringTokenizer class

This method returns the same value as hasMoreTokens() method of StringTokenizer class. The only difference is this class can implement the Enumeration interface.

**StringTokenizer2.java**

**import** java.util.StringTokenizer;

**public** **class** StringTokenizer2

{

**public** **static** **void** main(String args[])

 {

   StringTokenizer st = **new** StringTokenizer("Hello everyone I am a Java developer"," ");

**while** (st.hasMoreElements())

     {

         System.out.println(st.nextToken());

     }

 }

}

**Output:**

Hello

everyone

I

am

a

Java

developer

The above code demonstrates the use of hasMoreElements() method.

### Example of nextElement() method of the StringTokenizer class

nextElement() returns the next token object in the tokenizer String. It can implement Enumeration interface.

**StringTokenizer3.java**

**import** java.util.StringTokenizer;

**public** **class** StringTokenizer3

{

 /\* Driver Code \*/

**public** **static** **void** main(String args[])

 {

   /\* StringTokenizer object \*/

   StringTokenizer st = **new** StringTokenizer("Hello Everyone Have a nice day"," ");

     /\* Checks if the String has any more tokens \*/

**while** (st.hasMoreTokens())

     {

         /\* Prints the elements from the String \*/

         System.out.println(st.nextElement());

     }

 }

}

**Output:**

Hello

Everyone

Have

a

nice

day

The above code demonstrates the use of nextElement() method.

### Example of countTokens() method of the StringTokenizer class

This method calculates the number of tokens present in the tokenizer String.

**StringTokenizer4.java**

**import** java.util.StringTokenizer;

**public** **class** StringTokenizer3

{

 /\* Driver Code \*/

**public** **static** **void** main(String args[])

 {

   /\* StringTokenizer object \*/

   StringTokenizer st = **new** StringTokenizer("Hello Everyone Have a nice day"," ");

         /\* Prints the number of tokens present in the String \*/

         System.out.println("Total number of Tokens: "+st.countTokens());

 }

}

**Output:**

Total number of Tokens: 6

The above Java code demonstrates the countTokens() method of StringTokenizer() class.