**Java I/O** (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

**1) System.out:**standard output stream

**2) System.in:**standard input stream

**3) System.err:**standard error stream

Let's see the code to print **output and an error** message to the console.

1. System.out.println("simple message");
2. System.err.println("error message");

Let's see the code to get **input** from console.

1. **int** i=System.in.read();//returns ASCII code of 1st character
2. System.out.println((**char**)i);//will print the character

## OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

### **OutputStream**

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

### **InputStream**

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

Let's understand the working of Java OutputStream and InputStream by the figure given below.

Java IO

## OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

### **Useful methods of OutputStream**

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

### **OutputStream Hierarchy**

Java output stream hierarchy

## InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

### **Useful methods of InputStream**

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

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

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

## FileOutputStream class methods

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. }

Java FileOutputStream example 2: write string

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

# **Java FileInputStream Class**

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

## Java FileInputStream class declaration

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

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

## Java FileInputStream class methods

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

Java FileInputStream example 1: read single character

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

Java FileInputStream example 2: read all characters

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. }

# **Java BufferedOutputStream Class**

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

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

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

## Java BufferedOutputStream class declaration

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

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

## Java BufferedOutputStream class constructors

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

## Java BufferedOutputStream class methods

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

Example of BufferedOutputStream class:

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

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **public** **class** BufferedOutputStreamExample{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. FileOutputStream fout=**new** FileOutputStream("D:\\testout.txt");
6. BufferedOutputStream bout=**new** BufferedOutputStream(fout);
7. String s="Welcome to javaTpoint.";
8. **byte** b[]=s.getBytes();
9. bout.write(b);
10. bout.flush();
11. bout.close();
12. fout.close();
13. System.out.println("success");
14. }
15. }

# **Java BufferedInputStream Class**

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

The important points about BufferedInputStream are:

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

## Java BufferedInputStream class declaration

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

1. **public** **class** BufferedInputStream **extends** FilterInputStream

## Java BufferedInputStream class constructors

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

## Java BufferedInputStream class methods

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

### **Example of Java BufferedInputStream**

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

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

# **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

### **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. }

# **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. }

# **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. }

# **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. }

# **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. }

# **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. }

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

# **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. }

# **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

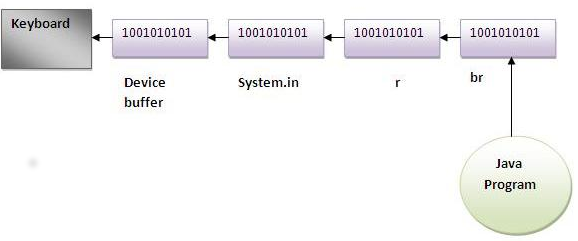
In this example, we are reading the data from the text file **testout.txt** using Java BufferedReader class.

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. }

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. }



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. }

# **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. }

# **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. }