**Reading and Writing Files in Java**

*By*[*Scott Robinson*](https://twitter.com/ScottWRobinson)*•*[*2 Comments*](https://stackabuse.com/reading-and-writing-files-in-java/#disqus_thread)

**Introduction**

In this article, we'll be diving into *Reading and Writing Files in Java*.

When programming, whether you're creating a mobile app, a web application, or just writing scripts, you often have the need to read or write data to a file. This data could be cache data, data you retrieved for a dataset, an image, or just about anything else you can think of.

In this tutorial, we are going to show the most common ways you can read and write to files in Java.

Java provides several API (also known as [Java I/O](https://docs.oracle.com/javase/tutorial/essential/io/)) to read and write files since its initial releases. With subsequent releases, Java I/O has been improved, simplified and enhanced to support new features.

Before we get in to some actual examples, it would help to understand the classes available to you that will handle the reading and writing of data to files. In the following sections we'll provide a brief overview of the Java I/O classes and explain what they do, then we'll take a look at Java NIO Streams, and finally we'll show some examples of reading and writing data to files.

**I/O Streams**

There are two types of Streams you can use to interact with files:

1. Character Streams
2. Byte Streams

For each of the above stream types, there are several supporting classes shipped with Java, which we'll take a quick look at below.

**Character Streams**

Character Streams are used to read or write the characters data type. Let's look at the most commonly used classes. All of these classes are defined under java.io package.

Here are some classes you should know that can be used to **read** character data:

* [Reader](https://docs.oracle.com/javase/8/docs/api/java/io/Reader.html): An abstract class to read a character stream.
* [InputStreamReader](https://docs.oracle.com/javase/8/docs/api/java/io/InputStreamReader.html): Class used to read the byte stream and converts to character stream.
* [FileReader](https://docs.oracle.com/javase/8/docs/api/java/io/FileReader.html): A class to read the characters from a file.
* [BufferedReader](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html): This is a wrapper over the Reader class that supports buffering capabilities. In many cases this is most preferable class to read data because more data can been read from the file in one read() call, reducing the number of actual I/O operations with file system.

And here are some classes you can use to **write** character data to a file:

* [Writer](https://docs.oracle.com/javase/8/docs/api/java/io/Writer.html): This is an abstract class to write the character streams.
* [OutputStreamWriter](https://docs.oracle.com/javase/8/docs/api/java/io/OutputStreamWriter.html): This class is used to write character streams and also convert them to byte streams.
* [FileWriter](https://docs.oracle.com/javase/8/docs/api/java/io/FileWriter.html): A class to actually write characters to the file.
* [BufferedWriter](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedWriter.html): This is a wrapper over the Writer class, which also supports buffering capabilities. This is most preferable class to write data to a file since more data can be written to the file in one write() call. And like the BufferedReader, this reduces the number of total I/O operations with file system.

**Byte Streams**

Byte Streams are used to read or write byte data with files. This is different from before in the way they treat the data. Here you work with raw bytes, which could be characters, image data, unicode data (which takes 2 bytes to represent a character), etc.

In this section we'll take a look at the most commonly used classes. All of these classes are defined under java.io package.

Here are the classes used to **read** the byte data:

* [InputStream](https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html): An abstract class to read the byte streams.
* [FileInputStream](https://docs.oracle.com/javase/8/docs/api/java/io/FileInputStream.html): A class to simply read bytes from a file.
* [BufferedInputStream](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedInputStream.html): This is a wrapper over InputStream that supports buffering capabilities. As we saw in the character streams, this is a more efficient method than FileInputStream.

And here are the classes used to **write** the byte data:

* [OutputStream](https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html): An abstract class to write byte streams.
* [FileOutputStream](https://docs.oracle.com/javase/8/docs/api/java/io/FileOutputStream.html): A class to write raw bytes to the file.
* [ByteOutputStream](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedOutputStream.html): This class is a wrapper over OutputStream to support buffering capabilities. And again, as we saw in the character streams, this is a more efficient method than FileOutputStream thanks to the buffering.

**Java NIO Streams**

[Java NIO](https://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html) is a non-blocking I/O API which was introduced back in Java 4 and can be found in the java.nio package. In terms of performance, this is a big improvement in the API for I/O operations.

Buffers, Selectors, and Channels are the three primary components of Java NIO, although in this article we'll focus strictly on using the NIO classes for interacting with files, and not necessarily the concepts behind the API.

As this tutorial is about reading and writing files, we will discuss only the related classes in this short section:

* [Path](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Path.html): This is a hierarchical structure of an actual file location and is typically used to locate the file you want to interact with.
* [Paths](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Paths.html): This is a class that provides several utility methods to create a Path from a given string URI.
* [Files](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html): This is another utility class which has several methods to read and write files without blocking the execution on threads.

Using these few classes, you can easily interact with files in a more efficient way.

**The Difference Between Java I/O and NIO**

The main difference between these two packages is that the read() and write() methods of Java IO are a blocking calls. By this we mean that the thread calling one of these methods will be blocked until the data has been read or written to the file.

**Subscribe to our Newsletter**

Top of Form

Get occassional tutorials, guides, and jobs in your inbox. No spam ever. Unsubscribe at any time.

Newsletter Signup

Subscribe

Bottom of Form

On the other hand, in the case of NIO, the methods are non-blocking. This means that the calling threads can perform other tasks (like reading/writing data from another source or update the UI) while the read or write methods wait for their operation to complete. This can result in a significant performance increase if you're dealing with many IO requests or lots of data.

**Examples of Reading and Writing Text Files**

In the previous sections, we have discussed the different APIs provided by Java and now it's time to use these API classes in some code.

The example code below handles reading and writing text files using the various classes we detailed above. To simplify things, and provide a better comparison of the actual methods being used, the input and output are going to remain the same between the examples.

**Note**: To avoid any confusion on the file path, the example code will read and write from a file on in user's home directory. The user's home directory can be found using System.getProperty("user.home");, which is what we use in our examples.

**Reading and Writing with FileReader and FileWriter**

Let's start by using the FileReader and FileWriter classes:

String directory = System.getProperty("user.home");

String fileName = "sample.txt";

String absolutePath = directory + File.separator + fileName;

*// Write the content in file*

try(FileWriter fileWriter = new FileWriter(absolutePath)) {

String fileContent = "This is a sample text.";

fileWriter.write(fileContent);

fileWriter.close();

} catch (IOException e) {

*// Cxception handling*

}

*// Read the content from file*

try(FileReader fileReader = new FileReader(absolutePath)) {

int ch = fileReader.read();

while(ch != -1) {

System.out.print((char)ch);

fileReader.close();

}

} catch (FileNotFoundException e) {

*// Exception handling*

} catch (IOException e) {

*// Exception handling*

}

Both classes accept a String, representing the path to the file in their constructors. You can also pass a File object as well as a FileDescriptor.

The write() method writes a valid character sequence - either a String, a char[]. Additionally, it can write a single char represented as an int.

The read() method reads and returns character-by-character, allowing us to use the read data in a while loop for example.

Don't forget to close both of these classes after use!

**Reading and Writing with BufferedReader and BufferedWriter**

Using BufferedReader and BufferedWriter classes:

String directory = System.getProperty("user.home");

String fileName = "sample.txt";

String absolutePath = directory + File.separator + fileName;

*// Write the content in file*

try(BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(absolutePath))) {

String fileContent = "This is a sample text.";

bufferedWriter.write(fileContent);

} catch (IOException e) {

*// Exception handling*

}

*// Read the content from file*

try(BufferedReader bufferedReader = new BufferedReader(new FileReader(absolutePath))) {

String line = bufferedReader.readLine();

while(line != null) {

System.out.println(line);

line = bufferedReader.readLine();

}

} catch (FileNotFoundException e) {

*// Exception handling*

} catch (IOException e) {

*// Exception handling*

}

**Reading and Writing with FileInputStream and FileOutputStream**

Using FileInputStream and FileOutputStream classes:

String directory = System.getProperty("user.home");

String fileName = "sample.txt";

String absolutePath = directory + File.separator + fileName;

*// write the content in file*

try(FileOutputStream fileOutputStream = new FileOutputStream(absolutePath)) {

String fileContent = "This is a sample text.";

fileOutputStream.write(fileContent.getBytes());

} catch (FileNotFoundException e) {

*// exception handling*

} catch (IOException e) {

*// exception handling*

}

*// reading the content of file*

try(FileInputStream fileInputStream = new FileInputStream(absolutePath)) {

int ch = fileInputStream.read();

while(ch != -1) {

System.out.print((char)ch);

ch = fileInputStream.read();

}

} catch (FileNotFoundException e) {

*// exception handling*

} catch (IOException e) {

*// exception handling*

}

**Reading and Writing with BufferedInputStream and BufferedOutputStream**

Using BufferedInputStream and BufferedOutputStream classes:

String directory = System.getProperty("user.home");

String fileName = "sample.txt";

String absolutePath = directory + File.separator + fileName;

*// write the content in file*

try(BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(new FileOutputStream(absolutePath))) {

String fileContent = "This is a sample text.";

bufferedOutputStream.write(fileContent.getBytes());

} catch (IOException e) {

*// exception handling*

}

*// read the content from file*

try(BufferedInputStream bufferedInputStream = new BufferedInputStream(new FileInputStream(absolutePath))) {

int ch = bufferedInputStream.read();

while(ch != -1) {

System.out.print((char)ch);

ch = bufferedInputStream.read();

}

} catch (FileNotFoundException e) {

*// exception handling*

} catch (IOException e) {

*// exception handling*

}

**Reading and Writing with Java.nio Classes**

Using the java.nio classes:

String directory = System.getProperty("user.home");

String fileName = "sample.txt";

String content = "This is a sample text.";

Path path = Paths.get(directory, fileName);

try {

Files.write(path, content.getBytes(), StandardOpenOption.CREATE);

} catch (IOException e) {

*// exception handling*

}

try {

List<String> list = Files.readAllLines(path);

list.forEach(line -> System.out.println(line));

} catch (IOException e) {

*// exception handling*

}

Another way to retrieve the content via the Files class, which is more important if you're not reading text data, is to use the readAllBytes method to read the data in to a byte array:

try {

byte[] data = Files.readAllBytes(path);

System.out.println(new String(data));

} catch (IOException e) {

*// exception handling*

}

In case you are interested in using streams with java.nio, you can also use the below methods provided by the Files class, which work just like the streams we covered earlier in the article:

Files.newBufferedReader(path)

Files.newBufferedWriter(path, options)

Files.newInputStream(path, options)

Files.newOutputStream(path, options)

**Conclusion**

In this article we have covered the most common ways to read and write data to a file using both the Java I/O package and the newer Java NIO package. Whenever possible, we recommend to use the Java NIO classes for file operations due to its non-blocking API, and in addition the code is a bit more maintainable and readable.

[*java*](https://stackabuse.com/tag/java/)

[](https://stackabuse.com/author/scott/)

About [Scott Robinson](https://stackabuse.com/author/scott/)

[Twitter](https://twitter.com/ScottWRobinson#_blank)

**Subscribe to our Newsletter**

Top of Form

Get occassional tutorials, guides, and jobs in your inbox. No spam ever. Unsubscribe at any time.

Newsletter Signup

Subscribe

Bottom of Form

[Previous Post](https://stackabuse.com/how-to-format-dates-in-python/) [Next Post](https://stackabuse.com/numpy-tutorial-a-simple-example-based-guide/)

**Ad**

**Follow Us**

[Twitter](https://twitter.com/StackAbuse)

[Facebook](https://www.facebook.com/stackabuse)

[RSS](https://stackabuse.com/rss/)

**Newsletter**

Subscribe to our newsletter! Get occassional tutorials, guides, and reviews in your inbox.

Top of Form

Newsletter Signup

Subscribe

No spam ever. Unsubscribe at any time.

Bottom of Form

**Ad**

**Want a remote job?**

* [Full Stack Software Engineer  
  **Medsphere** *4 days ago*](https://hireremote.io/remote-job/2142-full-stack-software-engineer-at-medsphere)

[react-js](https://hireremote.io/remote-react-js-jobs)[javascript](https://hireremote.io/remote-javascript-jobs)[xamarin](https://hireremote.io/remote-xamarin-jobs)[full stack](https://hireremote.io/remote-full-stack-jobs)

* [Senior Java Developer who wants to make a dif...  
  **O'Reilly Auto Parts** *5 days ago*](https://hireremote.io/remote-job/2127-senior-java-developer-who-wants-to-make-a-difference-at-oreilly-auto-parts)

[java](https://hireremote.io/remote-java-jobs)[javascript](https://hireremote.io/remote-javascript-jobs)[spring-boot](https://hireremote.io/remote-spring-boot-jobs)[postgresql](https://hireremote.io/remote-postgresql-jobs)

* [Technical Support Engineer  
   *6 days ago*](https://hireremote.io/remote-job/2126-technical-support-engineer)

[java](https://hireremote.io/remote-java-jobs)[linux](https://hireremote.io/remote-linux-jobs)

[More jobs](https://hireremote.io/)

Jobs via [HireRemote.io](https://hireremote.io/)

**[Prepping for an interview?](https://stackabu.se/daily-coding-problem" \t "_blank)**

* Improve your skills by solving one coding problem every day
* Get the solutions the next morning via email
* Practice on **actual problems** asked by top companies, like:

[Daily Coding Problem](https://stackabu.se/daily-coding-problem)

**Ad**

**Recent Posts**

[How to Convert a Java Array to ArrayList](https://stackabuse.com/how-to-convert-a-java-array-to-arraylist/)

[Bucket Sort in Python](https://stackabuse.com/bucket-sort-in-python/)

[Difference Between ArrayList and LinkedList in Java - Code and Performance](https://stackabuse.com/difference-between-arraylist-and-linkedlist-in-java-code-and-performance/)

**Tags**

[algorithms](https://stackabuse.com/tag/algorithms/)[amqp](https://stackabuse.com/tag/amqp/)[angular](https://stackabuse.com/tag/angular/)[announcements](https://stackabuse.com/tag/announcements/)[apache](https://stackabuse.com/tag/apache/)[api](https://stackabuse.com/tag/api/)[arduino](https://stackabuse.com/tag/arduino/)[artificial intelligence](https://stackabuse.com/tag/artificial-intelligence/)[asynchronous](https://stackabuse.com/tag/asynchronous/)[aws](https://stackabuse.com/tag/aws/)

**Follow Us**

[Twitter](https://twitter.com/StackAbuse)

[Facebook](https://www.facebook.com/stackabuse)

[RSS](https://stackabuse.com/rss/)

Copyright © 2020, [Stack Abuse](https://stackabuse.com/). All Rights Reserved.

[Disclosure](https://stackabuse.com/disclosure) • [Privacy](https://stackabuse.com/privacy-policy)