[Basic I/O](https://docs.oracle.com/javase/tutorial/essential/io/index.html)

Most of the classes covered in the I/O Streams section are in the java.io package. Most of the classes covered in the File I/O section are in the java.nio.file package.

# [I/O Streams](https://docs.oracle.com/javase/tutorial/essential/io/streams.html)

* [Byte Streams](https://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html) handle I/O of raw binary data.
* [Character Streams](https://docs.oracle.com/javase/tutorial/essential/io/charstreams.html) handle I/O of character data, automatically handling translation to and from the local character set.
* [Buffered Streams](https://docs.oracle.com/javase/tutorial/essential/io/buffers.html) optimize input and output by reducing the number of calls to the native API.
* [Scanning and Formatting](https://docs.oracle.com/javase/tutorial/essential/io/scanfor.html) allows a program to read and write formatted text.
* [I/O from the Command Line](https://docs.oracle.com/javase/tutorial/essential/io/cl.html) describes the Standard Streams and the Console object.
* [Data Streams](https://docs.oracle.com/javase/tutorial/essential/io/datastreams.html) handle binary I/O of primitive data type and String values.
* [Object Streams](https://docs.oracle.com/javase/tutorial/essential/io/objectstreams.html) handle binary I/O of objects.

# [File I/O (Featuring NIO.2)](https://docs.oracle.com/javase/tutorial/essential/io/fileio.html)

* [What is a Path?](https://docs.oracle.com/javase/tutorial/essential/io/path.html) examines the concept of a path on a file system.
* [The Path Class](https://docs.oracle.com/javase/tutorial/essential/io/pathClass.html) introduces the cornerstone class of the java.nio.file package.
* [Path Operations](https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html) looks at methods in the Path class that deal with syntactic operations.
* [File Operations](https://docs.oracle.com/javase/tutorial/essential/io/fileOps.html) introduces concepts common to many of the file I/O methods.
* [Checking a File or Directory](https://docs.oracle.com/javase/tutorial/essential/io/check.html) shows how to check a file's existence and its level of accessibility.
* [Deleting a File or Directory](https://docs.oracle.com/javase/tutorial/essential/io/delete.html).
* [Copying a File or Directory](https://docs.oracle.com/javase/tutorial/essential/io/copy.html).
* [Moving a File or Directory](https://docs.oracle.com/javase/tutorial/essential/io/move.html).
* [Managing Metadata](https://docs.oracle.com/javase/tutorial/essential/io/fileAttr.html) explains how to read and set file attributes.
* [Reading, Writing and Creating Files](https://docs.oracle.com/javase/tutorial/essential/io/file.html) shows the stream and channel methods for reading and writing files.
* [Random Access Files](https://docs.oracle.com/javase/tutorial/essential/io/rafs.html) shows how to read or write files in a non-sequentially manner.
* [Creating and Reading Directories](https://docs.oracle.com/javase/tutorial/essential/io/dirs.html) covers API specific to directories, such as how to list a directory's contents.
* [Links, Symbolic or Otherwise](https://docs.oracle.com/javase/tutorial/essential/io/links.html) covers issues specific to symbolic and hard links.
* [Walking the File Tree](https://docs.oracle.com/javase/tutorial/essential/io/walk.html) demonstrates how to recursively visit each file and directory in a file tree.
* [Finding Files](https://docs.oracle.com/javase/tutorial/essential/io/find.html) shows how to search for files using pattern matching.
* [Watching a Directory for Changes](https://docs.oracle.com/javase/tutorial/essential/io/notification.html) shows how to use the watch service to detect files that are added, removed or updated in one or more directories.
* [Other Useful Methods](https://docs.oracle.com/javase/tutorial/essential/io/misc.html) covers important API that didn't fit elsewhere in the lesson.
* [Legacy File I/O Code](https://docs.oracle.com/javase/tutorial/essential/io/legacy.html) shows how to leverage Path functionality if you have older code using the java.io.File class. A table mapping java.io.File API to java.nio.file API is provided.

# The I/O Classes in Action

Many of the examples in the next trail, [Custom Networking](https://docs.oracle.com/javase/tutorial/networking/index.html) use the I/O streams described in this lesson to read from and write to network connections.

**Security consideration:** Some I/O operations are subject to approval by the current security manager. The example programs contained in these lessons are standalone applications, which by default have no security manager. To work in an applet, most of these examples would have to be modified. See [What Applets Can and Cannot Do](https://docs.oracle.com/javase/tutorial/deployment/applet/security.html) for information about the security restrictions placed on applets.

## I/O Streams

### Byte Streams

Programs use *byte streams* to perform input and output of 8-bit bytes. All byte stream classes are descended from [InputStream](https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html) and [OutputStream](https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html).

There are many byte stream classes. To demonstrate how byte streams work, we'll focus on the file I/O byte streams, [FileInputStream](https://docs.oracle.com/javase/8/docs/api/java/io/FileInputStream.html) and [FileOutputStream](https://docs.oracle.com/javase/8/docs/api/java/io/FileOutputStream.html). Other kinds of byte streams are used in much the same way; they differ mainly in the way they are constructed.

#### Using Byte Streams

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class CopyBytes {

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

FileInputStream in = null;

FileOutputStream out = null;

try {

in = new FileInputStream("xanadu.txt");

out = new FileOutputStream("outagain.txt");

int c;

while ((c = in.read()) != -1) {

out.write(c);

}

} finally {

if (in != null) {

in.close();

}

if (out != null) {

out.close();

}

}

}

}

#### Always Close Streams

Closing a stream when it's no longer needed is very important。

#### When Not to Use Byte Streams

CopyBytes seems like a normal program, but it actually represents a kind of low-level I/O that you should avoid. Since xanadu.txt contains character data, the best approach is to use [character streams](https://docs.oracle.com/javase/tutorial/essential/io/charstreams.html)。There are also streams for more complicated data types. Byte streams should only be used for the most primitive I/O.

So why talk about byte streams? Because all other stream types are built on byte streams.

### Character Streams

For most applications, I/O with character streams is no more complicated than I/O with byte streams. Input and output done with stream classes automatically translates to and from the local character set. A program that uses character streams in place of byte streams automatically adapts to the local character set and is ready for internationalization — all without extra effort by the programmer.

If internationalization isn't a priority, you can simply use the character stream classes without paying much attention to character set issues. Later, if internationalization becomes a priority, your program can be adapted without extensive recoding. See the [Internationalization](https://docs.oracle.com/javase/tutorial/i18n/index.html) trail for more information.

#### Using Character Streams

All character stream classes are descended from [Reader](https://docs.oracle.com/javase/8/docs/api/java/io/Reader.html) and [Writer](https://docs.oracle.com/javase/8/docs/api/java/io/Writer.html). As with byte streams, there are character stream classes that specialize in file I/O: [FileReader](https://docs.oracle.com/javase/8/docs/api/java/io/FileReader.html) and [FileWriter](https://docs.oracle.com/javase/8/docs/api/java/io/FileWriter.html).

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class CopyCharacters {

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

FileReader inputStream = null;

FileWriter outputStream = null;

try {

inputStream = new FileReader("xanadu.txt");

outputStream = new FileWriter("characteroutput.txt");

int c;

while ((c = inputStream.read()) != -1) {

outputStream.write(c);

}

} finally {

if (inputStream != null) {

inputStream.close();

}

if (outputStream != null) {

outputStream.close();

}

}

}

}

#### Character Streams that Use Byte Streams

Character streams are often "wrappers" for byte streams. The character stream uses the byte stream to perform the physical I/O, while the character stream handles translation between characters and bytes.

#### Line-Oriented I/O

Character I/O usually occurs in bigger units than single characters. One common unit is the line: a string of characters with a line terminator at the end. A line terminator can be a carriage-return/line-feed sequence ("\r\n"), a single carriage-return ("\r"), or a single line-feed ("\n"). Supporting all possible line terminators allows programs to read text files created on any of the widely used operating systems.

Let's modify the CopyCharacters example to use line-oriented I/O. To do this, we have to use two classes we haven't seen before, [BufferedReader](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html) and [PrintWriter](https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html). We'll explore these classes in greater depth in [Buffered I/O](https://docs.oracle.com/javase/tutorial/essential/io/buffers.html) and [Formatting](https://docs.oracle.com/javase/tutorial/essential/io/formatting.html). Right now, we're just interested in their support for line-oriented I/O.

import java.io.FileReader;

import java.io.FileWriter;

import java.io.BufferedReader;

import java.io.PrintWriter;

import java.io.IOException;

public class CopyLines {

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

BufferedReader inputStream = null;

PrintWriter outputStream = null;

try {

inputStream = new BufferedReader(new FileReader("xanadu.txt"));

outputStream = new PrintWriter(new FileWriter("characteroutput.txt"));

String l;

while ((l = inputStream.readLine()) != null) {

outputStream.println(l);

}

} finally {

if (inputStream != null) {

inputStream.close();

}

if (outputStream != null) {

outputStream.close();

}

}

}

}

### Buffered Streams

Most of the examples we've seen so far use *unbuffered* I/O. This means each read or write request is handled directly by the underlying OS. This can make a program much less efficient, since each such request often triggers disk access, network activity, or some other operation that is relatively expensive.

inputStream = new BufferedReader(new FileReader("xanadu.txt"));

outputStream = new BufferedWriter(new FileWriter("characteroutput.txt"));

There are four buffered stream classes used to wrap unbuffered streams: [BufferedInputStream](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedInputStream.html) and[BufferedOutputStream](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedOutputStream.html) create buffered byte streams, while [BufferedReader](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html) and [BufferedWriter](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedWriter.html) create buffered character streams.

#### Flushing Buffered Streams

It often makes sense to write out a buffer at critical points, without waiting for it to fill. This is known as *flushing* the buffer.

### Scanning and Formatting

The [scanner](https://docs.oracle.com/javase/tutorial/essential/io/scanning.html) API breaks input into individual tokens associated with bits of data. The [formatting](https://docs.oracle.com/javase/tutorial/essential/io/formatting.html) API assembles data into nicely formatted, human-readable form.

#### Scanning

##### Breaking Input into Tokens

##### Translating Individual Tokens

#### Formatting

Stream objects that implement formatting are instances of either [PrintWriter](https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html), a character stream class, or[PrintStream](https://docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html), a byte stream class.

#### The print and println Methods

#### The format Method

Here are some other conversions:

* x formats an integer as a hexadecimal value.
* s formats any value as a string.
* tB formats an integer as a locale-specific month name.
* d formats an integer value as a decimal value.
* f formats a floating point value as a decimal value.
* n outputs a platform-specific line terminator.
* There are many other co
* **Note:**
* Except for %% and %n, all format specifiers must match an argument. If they don't, an exception is thrown.
* In the Java programming language, the \n escape always generates the linefeed character (\u000A). Don't use \n unless you specifically want a linefeed character. To get the correct line separator for the local platform, use %n.

### I/O from the Command Line

A program is often run from the command line and interacts with the user in the command line environment. The Java platform supports this kind of interaction in two ways: through the Standard Streams and through the Console.

#### Standard Streams

he Java platform supports three Standard Streams: *Standard Input*, accessed through System.in; *Standard Output*, accessed through System.out; and *Standard Error*, accessed through System.err.

By contrast, System.in is a byte stream with no character stream features. To use Standard Input as a character stream, wrap System.in in InputStreamReader.

InputStreamReader cin = new InputStreamReader(System.in);

#### The Console

### Data Streams

Data streams support binary I/O of primitive data type values (boolean, char, byte, short, int, long, float, and double) as well as String values.

Then DataStreams opens an output stream. Since a DataOutputStream can only be created as a wrapper for an existing byte stream object, DataStreams provides a buffered file output byte stream.

out = new DataOutputStream(new BufferedOutputStream(

new FileOutputStream(dataFile)));

## File I/O (Featuring NIO.2)

The java.nio.file package and its related package, java.nio.file.attribute, provide comprehensive support for file I/O and for accessing the default file system.

### What Is a Path? (And Other File System Facts)

#### What Is a Path?

#### Relative or Absolute?

An absolute path always contains the root element and the complete directory list required to locate the file.

A relative path needs to be combined with another path in order to access a file.

#### Symbolic Links

A *symbolic link* is a special file that serves as a reference to another file. A symbolic link is usually transparent to the user. Reading or writing to a symbolic link is the same as reading or writing to any other file or directory.

### The Path Class

#### Path Operations

This section covers the following:

* [Creating a Path](https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html#create)
* [Retrieving Information About a Path](https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html#info)
* [Removing Redundancies from a Path](https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html#normal)
* [Converting a Path](https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html#convert)
* [Joining Two Paths](https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html#resolve)
* [Creating a Path Between Two Paths](https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html#relativize)
* [Comparing Two Paths](https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html#compare)

##### Creating a Path

You can easily create a Path object by using one of the following get methods from the [Paths](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Paths.html) (note the plural) helper class:

Path p1 = Paths.get("/tmp/foo");

Path p2 = Paths.get(args[0]);

Path p3 = Paths.get(URI.create("file:///Users/joe/FileTest.java"));

The Paths.get method is shorthand for the following code:

Path p4 = FileSystems.getDefault().getPath("/users/sally");

##### Retrieving Information about a Path

Here is the output for both Windows and the Solaris OS:

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Invoked** | **Returns in the Solaris OS** | **Returns in Microsoft Windows** | **Comment** |
| toString | /home/joe/foo | C:\home\joe\foo | Returns the string representation of the Path. If the path was created using Filesystems.getDefault().getPath(String) or Paths.get (the latter is a convenience method for getPath), the method performs minor syntactic cleanup. For example, in a UNIX operating system, it will correct the input string //home/joe/foo to /home/joe/foo. |
| getFileName | foo | foo | Returns the file name or the last element of the sequence of name elements. |
| getName(0) | home | home | Returns the path element corresponding to the specified index. The 0th element is the path element closest to the root. |
| getNameCount | 3 | 3 | Returns the number of elements in the path. |
| subpath(0,2) | home/joe | home\joe | Returns the subsequence of the Path (not including a root element) as specified by the beginning and ending indexes. |
| getParent | /home/joe | \home\joe | Returns the path of the parent directory. |
| getRoot | / | C:\ | Returns the root of the path. |

Here is the output for Windows and the Solaris OS:

|  |  |  |
| --- | --- | --- |
| **Method Invoked** | **Returns in the Solaris OS** | **Returns in Microsoft Windows** |
| toString | sally/bar | sally\bar |
| getFileName | bar | bar |
| getName(0) | sally | sally |
| getNameCount | 2 | 2 |
| subpath(0,1) | sally | sally |
| getParent | sally | sally |
| getRoot | null | null |

### Removing Redundancies From a Path

Many file systems use "." notation to denote the current directory and ".." to denote the parent directory.

The normalize method removes any redundant elements, which includes any "." or "*directory*/.." occurrences.

### Converting a Path

### Joining Two Paths

### Creating a Path Between Two Paths

### File Operations

Before proceeding to the remaining sections, you should familiarize yourself with the following common concepts:

* [Releasing System Resources](https://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#resources)
* [Catching Exceptions](https://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#exception)
* [Varargs](https://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#varargs)
* [Atomic Operations](https://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#atomic)
* [Method Chaining](https://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#chaining)
* [What Is a Glob?](https://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#glob)
* [Link Awareness](https://docs.oracle.com/javase/tutorial/essential/io/fileOps.html#linkaware)

#### Releasing System Resources

Many of the resources that are used in this API, such as streams or channels, implement or extend the[java.io.Closeable](https://docs.oracle.com/javase/8/docs/api/java/io/Closeable.html) interface. A requirement of a Closeable resource is that the close method must be invoked to release the resource when no longer required.

#### Catching Exceptions

#### Varargs

Path source = ...;

Path target = ...;

Files.move(source,

target,

REPLACE\_EXISTING,

ATOMIC\_MOVE);

#### Atomic Operations

This is important when you have multiple processes operating on the same area of the file system, and you need to guarantee that each process accesses a complete file.

#### Method Chaining（方法链）

#### What Is a Glob?

For more information about the glob sytnax, see the API specification for the [getPathMatcher](https://docs.oracle.com/javase/8/docs/api/java/nio/file/FileSystem.html#getPathMatcher-java.lang.String-) method in the FileSystem class.

#### Link Awareness

The Files class is "link aware." Every Files method either detects what to do when a symbolic link is encountered, or it provides an option enabling you to configure the behavior when a symbolic link is encountered.

### Checking a File or Directory

#### Verifying the Existence of a File or Directory

You can do so with the[exists(Path, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#exists-java.nio.file.Path-java.nio.file.LinkOption...-) and the [notExists(Path, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#notExists-java.nio.file.Path-java.nio.file.LinkOption...-) methods. Note that !Files.exists(path) is not equivalent to Files.notExists(path).

#### Checking File Accessibility

To verify that the program can access a file as needed, you can use the [isReadable(Path)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#isReadable-java.nio.file.Path-),[isWritable(Path)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#isWritable-java.nio.file.Path-), and [isExecutable(Path)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#isExecutable-java.nio.file.Path-) methods.

#### Checking Whether Two Paths Locate the Same File

When you have a file system that uses symbolic links, it is possible to have two different paths that locate the same file. The [isSameFile(Path, Path)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#isSameFile-java.nio.file.Path-java.nio.file.Path-) method compares two paths to determine if they locate the same file on the file system. For example:

Path p1 = ...;

Path p2 = ...;

if (Files.isSameFile(p1, p2)) {

// Logic when the paths locate the same file

}

### Deleting a File or Directory

The Files class provides two deletion methods.The [delete(Path)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#delete-java.nio.file.Path-) method deletes the file or throws an exception if the deletion fails. The [deleteIfExists(Path)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#deleteIfExists-java.nio.file.Path-) method also deletes the file, but if the file does not exist, no exception is thrown. Failing silently is useful when you have multiple threads deleting files and you don't want to throw an exception just because one thread did so first.

### Copying a File or Directory

You can copy a file or directory by using the [copy(Path, Path, CopyOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#copy-java.nio.file.Path-java.nio.file.Path-java.nio.file.CopyOption...-) method. The copy fails if the target file exists, unless the REPLACE\_EXISTING option is specified.

In addition to file copy, the Files class also defines methods that may be used to copy between a file and a stream. The [copy(InputStream, Path, CopyOptions...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#copy-java.io.InputStream-java.nio.file.Path-java.nio.file.CopyOption...-) method may be used to copy all bytes from an input stream to a file. The [copy(Path, OutputStream)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#copy-java.nio.file.Path-java.io.OutputStream-) method may be used to copy all bytes from a file to an output stream.

### Moving a File or Directory

You can move a file or directory by using the [move(Path, Path, CopyOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#move-java.nio.file.Path-java.nio.file.Path-java.nio.file.CopyOption...-) method. The move fails if the target file exists, unless the REPLACE\_EXISTING option is specified.

### Managing Metadata (File and File Store Attributes)

A file system's metadata is typically referred to as its *file attributes*. The Files class includes methods that can be used to obtain a single attribute of a file, or to set an attribute.

|  |  |
| --- | --- |
| **Methods** | **Comment** |
| [size(Path)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#size-java.nio.file.Path-) | Returns the size of the specified file in bytes. |
| [isDirectory(Path, LinkOption)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#isDirectory-java.nio.file.Path-java.nio.file.LinkOption...-) | Returns true if the specified Path locates a file that is a directory. |
| [isRegularFile(Path, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#isRegularFile-java.nio.file.Path-java.nio.file.LinkOption...-) | Returns true if the specified Path locates a file that is a regular file. |
| [isSymbolicLink(Path)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#isSymbolicLink-java.nio.file.Path-) | Returns true if the specified Path locates a file that is a symbolic link. |
| [isHidden(Path)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#isHidden-java.nio.file.Path-) | Returns true if the specified Path locates a file that is considered hidden by the file system. |
| [getLastModifiedTime(Path, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#getLastModifiedTime-java.nio.file.Path-java.nio.file.LinkOption...-) [setLastModifiedTime(Path, FileTime)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#setLastModifiedTime-java.nio.file.Path-java.nio.file.attribute.FileTime-) | Returns or sets the specified file's last modified time. |
| [getOwner(Path, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#getOwner-java.nio.file.Path-java.nio.file.LinkOption...-) [setOwner(Path, UserPrincipal)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#setOwner-java.nio.file.Path-java.nio.file.attribute.UserPrincipal-) | Returns or sets the owner of the file. |
| [getPosixFilePermissions(Path, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#getPosixFilePermissions-java.nio.file.Path-java.nio.file.LinkOption...-) [setPosixFilePermissions(Path, Set<PosixFilePermission>)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#setPosixFilePermissions-java.nio.file.Path-java.util.Set-) | Returns or sets a file's POSIX file permissions. |
| [getAttribute(Path, String, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#getAttribute-java.nio.file.Path-java.lang.String-java.nio.file.LinkOption...-) [setAttribute(Path, String, Object, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#setAttribute-java.nio.file.Path-java.lang.String-java.lang.Object-java.nio.file.LinkOption...-) | Returns or sets the value of a file attribute. |

If a program needs multiple file attributes around the same time, it can be inefficient to use methods that retrieve a single attribute. Repeatedly accessing the file system to retrieve a single attribute can adversely affect performance. For this reason, the Files class provides two readAttributes methods to fetch a file's attributes in one bulk operation.

|  |  |
| --- | --- |
| **Method** | **Comment** |
| [readAttributes(Path, String, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#readAttributes-java.nio.file.Path-java.lang.String-java.nio.file.LinkOption...-) | Reads a file's attributes as a bulk operation. The String parameter identifies the attributes to be read. |
| [readAttributes(Path, Class<A>, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#readAttributes-java.nio.file.Path-java.lang.Class-java.nio.file.LinkOption...-) | Reads a file's attributes as a bulk operation. The Class<A> parameter is the type of attributes requested and the method returns an object of that class. |

Before showing examples of the readAttributes methods, it should be mentioned that different file systems have different notions about which attributes should be tracked. For this reason, related file attributes are grouped together into views. A *view* maps to a particular file system implementation, such as POSIX or DOS, or to a common functionality, such as file ownership.

The supported views are as follows:

* [BasicFileAttributeView](https://docs.oracle.com/javase/8/docs/api/java/nio/file/attribute/BasicFileAttributeView.html) – Provides a view of basic attributes that are required to be supported by all file system implementations.
* [DosFileAttributeView](https://docs.oracle.com/javase/8/docs/api/java/nio/file/attribute/DosFileAttributeView.html) – Extends the basic attribute view with the standard four bits supported on file systems that support the DOS attributes.
* [PosixFileAttributeView](https://docs.oracle.com/javase/8/docs/api/java/nio/file/attribute/PosixFileAttributeView.html) – Extends the basic attribute view with attributes supported on file systems that support the POSIX family of standards, such as UNIX. These attributes include file owner, group owner, and the nine related access permissions.
* [FileOwnerAttributeView](https://docs.oracle.com/javase/8/docs/api/java/nio/file/attribute/FileOwnerAttributeView.html) – Supported by any file system implementation that supports the concept of a file owner.
* [AclFileAttributeView](https://docs.oracle.com/javase/8/docs/api/java/nio/file/attribute/AclFileAttributeView.html) – Supports reading or updating a file's Access Control Lists (ACL). The NFSv4 ACL model is supported. Any ACL model, such as the Windows ACL model, that has a well-defined mapping to the NFSv4 model might also be supported.

[UserDefinedFileAttributeView](https://docs.oracle.com/javase/8/docs/api/java/nio/file/attribute/UserDefinedFileAttributeView.html) – Enables support of metadata that is user defined. This view can be mapped to any extension mechanisms that a system supports. In the Solaris OS, for The remainder of this section covers the following topics:

* [Basic File Attributes](https://docs.oracle.com/javase/tutorial/essential/io/fileAttr.html#basic)
* [Setting Time Stamps](https://docs.oracle.com/javase/tutorial/essential/io/fileAttr.html#time)
* [DOS File Attributes](https://docs.oracle.com/javase/tutorial/essential/io/fileAttr.html#dos)
* [POSIX File Permissions](https://docs.oracle.com/javase/tutorial/essential/io/fileAttr.html#posix)
* [Setting a File or Group Owner](https://docs.oracle.com/javase/tutorial/essential/io/fileAttr.html#lookup)
* [User-Defined File Attributes](https://docs.oracle.com/javase/tutorial/essential/io/fileAttr.html#user)
* [File Store Attributes](https://docs.oracle.com/javase/tutorial/essential/io/fileAttr.html#store)

#### Basic File Attributes

#### Setting Time Stamps

The following code snippet sets the last modified time in milliseconds:

Path file = ...;

BasicFileAttributes attr =

Files.readAttributes(file, BasicFileAttributes.class);

long currentTime = System.currentTimeMillis();

FileTime ft = FileTime.fromMillis(currentTime);

Files.setLastModifiedTime(file, ft);

}

#### DOS File Attributes

However, you can set a DOS attribute using the [setAttribute(Path, String, Object, LinkOption...)](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#setAttribute-java.nio.file.Path-java.lang.String-java.lang.Object-java.nio.file.LinkOption...-) method, as follows:

Path file = ...;

Files.setAttribute(file, "dos:hidden", true);

#### POSIX File Permissions

The code uses the methods in the [PosixFileAttributes](https://docs.oracle.com/javase/8/docs/api/java/nio/file/attribute/PosixFileAttributes.html) class.

Path file = ...;

PosixFileAttributes attr =

Files.readAttributes(file, PosixFileAttributes.class);

System.out.format("%s %s %s%n",

attr.owner().getName(),

attr.group().getName(),

PosixFilePermissions.toString(attr.permissions()));

### Reading, Writing, and Creating Files