Java File Handling

The File class from the java.io package, allows us to work with files.

To use the File class, create an object of the class, and specify the filename or directory name:

import java.io.File; // Import the File class

File myObj = new File("filename.txt"); // Specify the filename

The File class has many useful methods for creating and getting information about files. For example:

|  |  |  |
| --- | --- | --- |
| **Method** | **Type** | **Description** |
| canRead() | Boolean | Tests whether the file is readable or not |
| canWrite() | Boolean | Tests whether the file is writable or not |
| createNewFile() | Boolean | Creates an empty file |
| delete() | Boolean | Deletes a file |
| exists() | Boolean | Tests whether the file exists |
| getName() | String | Returns the name of the file |
| getAbsolutePath() | String | Returns the absolute pathname of the file |
| length() | Long | Returns the size of the file in bytes |
| list() | String[] | Returns an array of the files in the directory |
| mkdir() | Boolean | Creates a directory |

Create file with folder

File f = new File("files\\sample.txt");  
f.getParentFile().mkdir();  
f.createNewFile();

## Create a File

import java.io.File; // Import the File class

import java.io.IOException; // Import the IOException class to handle errors

public class CreateFile {

public static void main(String[] args) {

try {

File myObj = new File("filename.txt");

if (myObj.createNewFile()) {

System.out.println("File created: " + myObj.getName());

} else {

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

}

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

To create a file in a specific directory (requires permission), specify the path of the file and use double backslashes to escape the "\" character (for Windows). On Mac and Linux you can just write the path, like: /Users/name/filename.txt

### Example

File myObj = new File("C:\\Users\\MyName\\filename.txt");

## Stream

A series of data is referred to as ****a stream****. In [Java](https://www.javatpoint.com/java-tutorial), ****Stream**** is classified into two types, i.e., ****Byte Stream**** and ****Character Stream****.

**Byte streams**

Handle raw binary data, such as images, audio, video, and files. They are lower level streams that work directly with bytes, offering more control and flexibility. Byte streams are defined by the abstract classes InputStream and OutputStream

**Character streams**

Handle text data, such as text files. They are implemented by the Reader and Writer classes and their subclasses.

## Write To a File

In the following example, we use the FileWriter class together with its write() method to write some text to the file we created in the example above. Note that when you are done writing to the file, you should close it with the close() method:

### Example

import java.io.FileWriter;

import java.io.\*; // Import the FileWriter class

import java.io.IOException; // Import the IOException class to handle errors

public class WriteToFile {

public static void main(String[] args) {

try {

FileWriter myWriter = new FileWriter("filename.txt");

myWriter.write("Files in Java might be tricky, but it is fun enough!");

myWriter.close();

System.out.println("Successfully wrote to the file.");

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

Read a File

In the previous chapter, you learned how to create and write to a file.

In the following example, we use the Scanner class to read the contents of the text file we created in the previous chapter:

Example

import java.io.File; // Import the File class

import java.io.FileNotFoundException; // Import this class to handle errors

import java.util.Scanner; // Import the Scanner class to read text files

public class ReadFile {

public static void main(String[] args) {

try {

File myObj = new File("filename.txt");

Scanner myReader = new Scanner(myObj);

while (myReader.hasNextLine()) {

String data = myReader.nextLine();

System.out.println(data);

}

myReader.close();

} catch (FileNotFoundException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

The output will be:

Files in Java might be tricky, but it is fun enough!

Get File Information

To get more information about a file, use any of the File methods:

import java.io.File; // Import the File class

public class GetFileInfo {   
 public static void main(String[] args) {

File myObj = new File("filename.txt");

if (myObj.exists()) {

System.out.println("File name: " + myObj.getName());

System.out.println("Absolute path: " + myObj.getAbsolutePath());

System.out.println("Writeable: " + myObj.canWrite());

System.out.println("Readable " + myObj.canRead());

System.out.println("File size in bytes " + myObj.length());

} else {

System.out.println("The file does not exist.");

}

}

}

The output will be:

File name: filename.txt  
Absolute path: C:\Users\MyName\filename.txt  
Writeable: true  
Readable: true  
File size in bytes: 0

Delete a File

To delete a file in Java, use the delete() method:

import java.io.File; // Import the File class

public class DeleteFile {

public static void main(String[] args) {

File myObj = new File("filename.txt");

if (myObj.delete()) {

System.out.println("Deleted the file: " + myObj.getName());

} else {

System.out.println("Failed to delete the file.");

}

}

}

The output will be:

Deleted the file: filename.txt

Delete a Folder

You can also delete a folder. However, it must be empty:

import java.io.File;

public class DeleteFolder {

public static void main(String[] args) {

File myObj = new File("C:\\Users\\MyName\\Test");

if (myObj.delete()) {

System.out.println("Deleted the folder: " + myObj.getName());

} else {

System.out.println("Failed to delete the folder.");

}

}

}

The output will be:

Deleted the folder: Test

1. How to create read write a file?
2. How to delete and read other file infos with file Methods
3. How to make a folder?