File Handling in Java

Difficulty Level : Medium • Last Updated : 03 Jan, 2022

In Java, with the help of File Class, we can work with files. This File Class is inside the java.io package. The File class can be used by creating an object of the class and then specifying the name of the file.

Why File Handling is Required?

- File Handling is an integral part of any programming language as file handling enables us to store the output of any particular program in a file and allows us to perform certain operations on it.
- In simple words, file handling means reading and writing data to a file.

Java

```
// Importing File Class
import java.io.File;

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

        // File name specified
        File obj = new File("myfile.txt");
          System.out.println("File Created!");
    }
}
```

Output

```
File Created!
```

Streams in Java

- In Java, a sequence of data is known as a stream.
- This concept is used to perform I/O operations on a file.
- There are two types of streams :

1. Input Stream:

The Java InputStream class is the superclass of all input streams. The input stream is used to read data from numerous input devices like the keyboard, network, etc. InputStream is an abstract class, and because of this, it is not useful by itself. However, its subclasses are used to read data.

There are several subclasses of the InputStream class, which are as follows:

1. AudioInputStream
2. ByteArrayInputStream
3. FileInputStream
4. FilterInputStream
5. StringBufferInputStream
6. ObjectInputStream

Creating an InputStream

```
// Creating an InputStream
InputStream obj = new FileInputStream();
```

Here, an input stream is created using FileInputStream.

> ***Note:*** *We can create an input stream from other subclasses as well as InputStream.*

Methods of InputStream

S No.	Method	Description

No.		
1	read()	Reads one byte of data from the input stream.
2	read(byte[] array)()	Reads byte from the stream and stores that byte in the specified array.
3	mark()	It marks the position in the input stream until the data has been read.
4	available()	Returns the number of bytes available in the input stream.
5	markSupported()	It checks if the mark() method and the reset() method is supported in the stream.
6	reset()	Returns the control to the point where the mark was set inside the stream.
7	skips()	Skips and removes a particular number of bytes from the input stream.
8	close()	Closes the input stream.

2. Output Stream:

The output stream is used to write data to numerous output devices like the monitor, file, etc. OutputStream is an abstract superclass that represents an output stream. OutputStream is an abstract class and because of this, it is not useful by itself. However, its subclasses are used to write data.

There are several subclasses of the OutputStream class which are as follows:

1. ByteArrayOutputStream
2. FileOutputStream
3. StringBufferOutputStream
4. ObjectOutputStream
5. DataOutputStream
6. PrintStream

```
OutputStream obj = new FileOutputStream();
```

Here, an output stream is created using FileOutputStream.

> ***Note:*** *We can create an output stream from other subclasses as well as OutputStream.*

Methods of OutputStream

S. No.	Method	Description
1.	write()	Writes the specified byte to the output stream.
2.	write(byte[] array)	Writes the bytes which are inside a specific array to the output stream.
3.	close()	Closes the output stream.
4.	flush()	Forces to write all the data present in an output stream to the destination.

Based on the data type, there are two types of streams :

1. Byte Stream:
This stream is used to read or write byte data. The byte stream is again subdivided into two types which are as follows:

- **Byte Input Stream:** Used to read byte data from different devices.
- **Byte Output Stream:** Used to write byte data to different devices.

2. Character Stream:
This stream is used to read or write character data. Character stream is again subdivided into 2 types which are as follows:

Owing to the fact that you know what a stream is, let's polish up File Handling in Java by further understanding the various methods that are useful for performing operations on the files like creating, reading, and writing files.

Java File Class Methods

The following table depicts several File Class methods:

Method Name	Description	Return Type
canRead()	It tests whether the file is readable or not.	Boolean
canWrite()	It tests whether the file is writable or not.	Boolean
createNewFile()	It creates an empty file.	Boolean
delete()	It deletes a file.	Boolean
exists()	It tests whether the file exists or not.	Boolean
length()	Returns the size of the file in bytes.	Long
getName()	Returns the name of the file.	String
list()	Returns an array of the files in the directory.	String[]
mkdir()	Creates a new directory.	Boolean
getAbsolutePath()	Returns the absolute pathname of the file.	String

Let us now get acquainted with the various file operations in Java.

File operations in Java

- **Read from a File**
- **Write to a File**
- **Delete a File**

Now let us study each of the above operations in detail.

1. Create a File

- In order to create a file in Java, you can use the createNewFile() method.
- If the file is successfully created, it will return a Boolean value true and false if the file already exists.

Following is a demonstration of how to create a file in Java :

Java

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

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

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

        try {
            File Obj = new File("myfile.txt");
            if (Obj.createNewFile()) {
                System.out.println("File created: "
                                   + Obj.getName());
            }
            else {
                System.out.println("File already exists.");
            }
        }
        catch (IOException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
```

```
An error has occurred.
```

2. Read from a File: We will use the Scanner class in order to read contents from a file. Following is a demonstration of how to read contents from a file in Java :

Java

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

// Import this class for handling errors
import java.io.FileNotFoundException;

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

public class GFG {
    public static void main(String[] args)
    {
        try {
            File Obj = new File("myfile.txt");
            Scanner Reader = new Scanner(Obj);
            while (Reader.hasNextLine()) {
                String data = Reader.nextLine();
                System.out.println(data);
            }
            Reader.close();
        }
        catch (FileNotFoundException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

Output

```
An error has occurred.
```


3. Write to a File: We use the FileWriter class along with its write() method in order to write some text to the file. Following is a demonstration of how to write text to a file in Java :

```java
// Import the FileWriter class
import java.io.FileWriter;

// Import the IOException class for handling errors
import java.io.IOException;

public class GFG {
    public static void main(String[] args)
    {
        try {
            FileWriter Writer
                = new FileWriter("myfile.txt");
            Writer.write(
                "Files in Java are seriously good!!");
            Writer.close();
            System.out.println("Successfully written.");
        }
        catch (IOException e) {
            System.out.println("An error has occurred.");
            e.printStackTrace();
        }
    }
}
```

Output

```
An error has occurred.
```

4. Delete a File: We use the delete() method in order to delete a file. Following is a demonstration of how to delete a file in Java :

Java

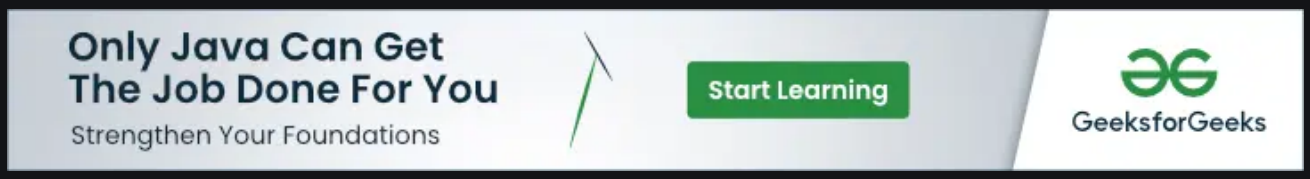
```java
// Import the File class
import java.io.File;

public class GFG {
    public static void main(String[] args)
    {
        File Obj = new File("myfile.txt");
        if (Obj.delete()) {
            System.out.println("The deleted file is : "
                               + myObj.getName());
        }
        else {
```

```
    }
}
```

Output

```
Failed in deleting the file.
```



Previous Next

RECOMMENDED ARTICLES Page : 1 2 3

27, Nov 16

02 File Handling in Java with CRUD operations
12, Apr 19

03 Different Ways to Copy Content From One File to Another File in Java
23, Oct 20

04 How to Convert a Kotlin Source File to a Java Source File in Android?
17, Nov 21

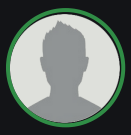
Another File
27, Jan 21

06 Comparison of Exception Handling in C++ and Java
10, Dec 10

07 Output of Java program | Set 12(Exception Handling)
22, May 17

08 Nested try blocks in Exception Handling in Java
21, Oct 18

Article Contributed By :



shreyasnaphad
@shreyasnaphad

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Article Tags : java-file-handling, Picked, Java

Practice Tags : Java

Improve Article | Report Issue

GeeksforGeeks

A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org