

# Java Program to Write into a File

Difficulty Level : Basic Last Updated : 09 Feb, 2022

In this article, we will see different ways of writing into a File using Java Programming language. Java FileWriter class in java is used to write character-oriented data to a file as this class is character-oriented class because of what it is used in file handling in java.

There are many ***ways to write into a file in Java*** as there are many classes and methods which can fulfill the goal as follows:

1. Using writeString() method
2. Using FileWriter Class
3. Using BufferedWriter Class
4. Using FileOutputStream Class

## Method 1: Using writeString() method

This method is supported by Java version 11. This method can take four parameters. These are file path, character sequence, charset, and options. The first two parameters are mandatory for this method to write into a file. It writes the characters as the content of the file. It returns the file path and can throw four types of exceptions. It is better to use when the content of the file is short.

**Example:** It shows the use of the *writeString()* method that is under the Files class to write data into a file. Another class, Path, is used to assign the filename with a path where the content will be written. Files class has another method named *readString()* to read the content of any existing file that is used in the code to check the content is properly written in the file.

---

```
// Java Program to Write Into a File
// using writeString() Method

// Importing required classes
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
```

```
// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
        throws IOException
    {
        // Assigning the content of the file
        String text
            = "Welcome to geekforgeeks\nHappy Learning!";

        // Defining the file name of the file
        Path fileName = Path.of(
            "/Users/mayanksolanki/Desktop/demo.docx");

        // Writing into the file
        Files.writeString(fileName, text);

        // Reading the content of the file
        String file_content = Files.readString(fileName);

        // Printing the content inside the file
        System.out.println(file_content);
    }
}
```

## Output

```
Welcome to geekforgeeks
Happy Learning!
```

```
((base) Mayanks-MacBook-Air:~ mayanksolanki$ cd /Users/mayanksolanki/Desktop/Job/
Coding/GFG/Folder/
((base) Mayanks-MacBook-Air:Folder mayanksolanki$ javac GFG.java
((base) Mayanks-MacBook-Air:Folder mayanksolanki$ java GFG
Welcome to geekforgeeks
Happy Learning!
((base) Mayanks-MacBook-Air:Folder mayanksolanki$
```

## Method 2: Using FileWriter Class

If the content of the file is short, then using the `FileWriter` class to write in the file is another better option. It also writes the stream of characters as the content of the file like `writeString()` method. The constructor of this class defines the default character encoding and the default buffer size in bytes.

The following below example illustrates the use of the `FileWriter` class to write content into a file. It requires creating the object of the `FileWriter` class with the filename to write into a file. Next, the `write()` method is used to write the value of the text variable in the file. If any error occurs at the time of writing the file, then an `IOException` will be thrown, and the error message will be printed from the catch block.

### Example:

---

```
// Java Program to Write into a File
// using FileWriterClass

// Importing required classes
import java.io.FileWriter;
import java.io.IOException;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Content to be assigned to a file
        // Custom input just for illustratinon purposes
        String text
            = "Computer Science Portal GeeksforGeeks";

        // Try block to check if exception occurs
        try {

            // Create a FileWriter object
            // to write in the file
            FileWriter fWriter = new FileWriter(
                "/Users/mayanksolanki/Desktop/demo.docx");

            // Writing into file
            // Note: The content taken above inside the
            // string
            fWriter.write(text);

            // Printing the contents of a file
```

```
System.out.println(text);

// Closing the file writing connection
fWriter.close();

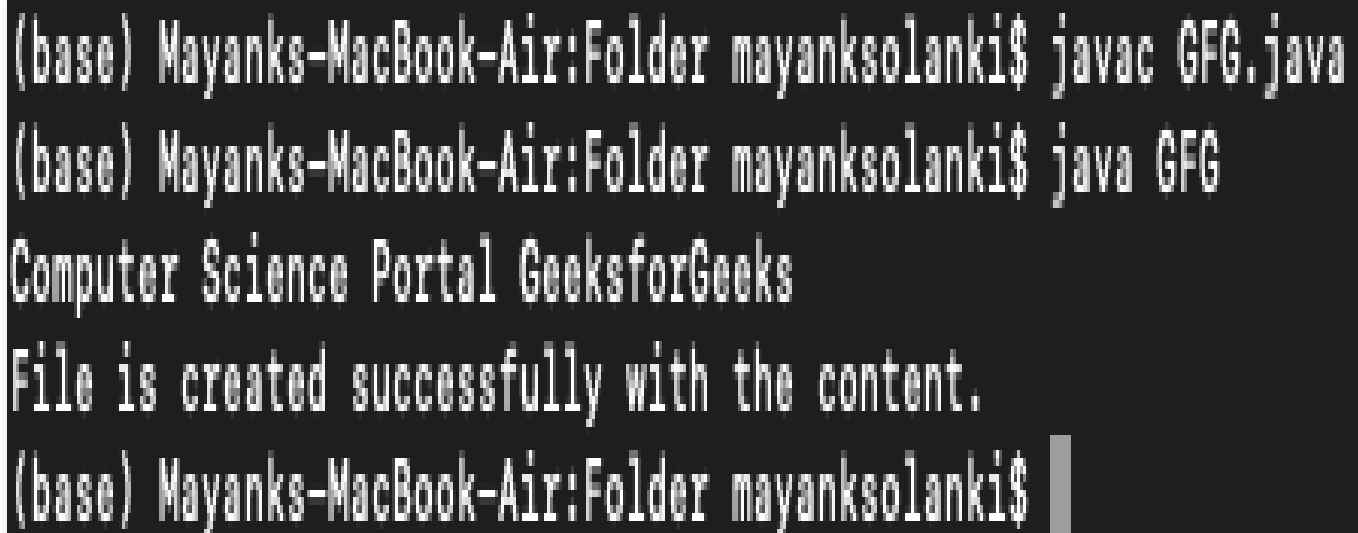
// Display message for successful execution of
// program on the console
System.out.println(
    "File is created successfully with the content.");
}

// Catch block to handle if exception occurs
catch (IOException e) {

    // Print the exception
    System.out.print(e.getMessage());
}
}
```

## Output

File is created successfully with the content.



```
(base) Mayanks-MacBook-Air:Folder mayanksolanki$ javac GFG.java
(base) Mayanks-MacBook-Air:Folder mayanksolanki$ java GFG
Computer Science Portal GeeksforGeeks
File is created successfully with the content.
(base) Mayanks-MacBook-Air:Folder mayanksolanki$
```

## Method 3: Using BufferedWriter Class

It is used to write text to a character-output stream. It has a default buffer size, but a large buffer size can be assigned. It is useful for writing characters, strings, and arrays. It is better to wrap this class with any writer class for writing data to a file if no prompt output is required.

## Example:

---

```
// Java Program to write into a File
// Using BufferedWriter Class

// Importing java input output libraries
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Assigning the file content
        // Note: Custom contents taken as input to
        // illustrate
        String text
            = "Computer Science Portal GeeksforGeks";

        // Try block to check for exceptions
        try {

            // Step 1: Create an object of BufferedWriter
            BufferedWriter f_writer
                = new BufferedWriter(new FileWriter(
                    "/Users/mayanksolanki/Desktop/demo.docx"));

            // Step 2: Write text(content) to file
            f_writer.write(text);

            // Step 3: Printing the content inside the file
            // on the terminal/CMD
            System.out.print(text);

            // Step 4: Display message showcasing
            // successful execution of the program
            System.out.print(
                "File is created successfully with the content.");

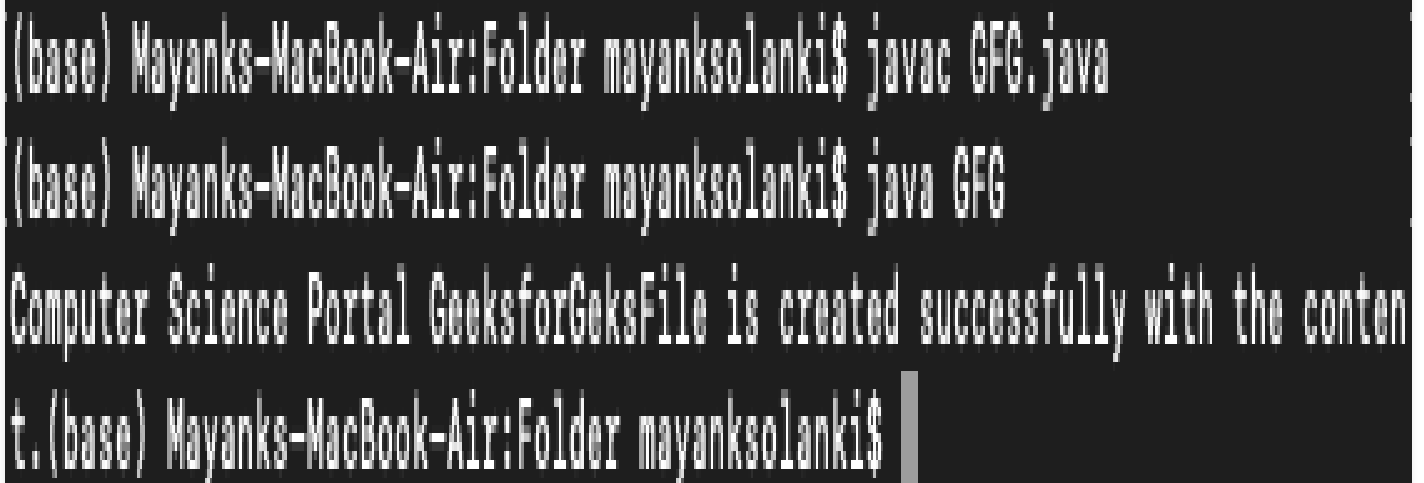
            // Step 5: Close the BufferedWriter object
            f_writer.close();
        }

        // Catch block to handle if exceptions occurs
        catch (IOException e) {
```

```
        // Print the exception on console
        // using getMessage() method
        System.out.print(e.getMessage());
    }
}
```

## Output

File is created successfully with the content.



```
(base) Mayanks-MacBook-Air:Folder mayanksolanki$ javac GFG.java
(base) Mayanks-MacBook-Air:Folder mayanksolanki$ java GFG
Computer Science Portal GeeksforGeksFile is created successfully with the content.
(base) Mayanks-MacBook-Air:Folder mayanksolanki$
```

The following example shows the use of `BufferedWriter` class to write into a file. It also requires creating the object of `BufferedWriter` class like `FileWriter` to write content into the file. But this class supports large content to write into the file by using a large buffer size.

## Method 4: Using `FileOutputStream` Class

It is used to write raw stream data to a file. `FileWriter` and `BufferedWriter` classes are used to write only the text to a file, but the binary data can be written by using the `FileOutputStream` class.

To write data into a file using `FileOutputStream` class is shown in the following example. It also requires creating the object of the class with the filename to write data into a file. Here, the string content is converted into the byte array that is written into the file by using the `write()` method.

## Example:

```
// Java Program to Write into a File
// using FileOutputStream Class

// Importing java input output classes
import java.io.FileOutputStream;
import java.io.IOException;

public class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Assign the file content
        String fileContent = "Welcome to geeksforgeeks";
        FileOutputStream outputStream = null;

        // Try block to check if exception occurs
        try {

            // Step 1: Create an object of FileOutputStream
            outputStream = new FileOutputStream("file.txt");

            // Step 2: Store byte content from string
            byte[] strToBytes = fileContent.getBytes();

            // Step 3: Write into the file
            outputStream.write(strToBytes);

            // Print the success message (Optional)
            System.out.print(
                "File is created successfully with the content.");
        }

        // Catch block to handle the exception
        catch (IOException e) {

            // Display the exception/s
            System.out.print(e.getMessage());
        }

        // finally keyword is used with in try catch block
        // and this code will always execute whether
        // exception occurred or not
        finally {

            // Step 4: Close the object
            if (outputStream != null) {

                // Note: Second try catch block ensures that
```

```
// the file is closed even if an error
// occurs
try {

    // Closing the file connections
    // if no exception has occurred
    outputStream.close();
}

catch (IOException e) {

    // Display exceptions if occurred
    System.out.print(e.getMessage());
}
}
}
```

## Output

File is created successfully with the content.