CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

Write a File

Overview

1. Introduction
2. Write File using FileWriter class
3. Write File using writeString() method
4. Write File using BufferedWriter class
5. Write File using FileOutputStream class

Introduction

In this lecture, we will learn how to write a file in Java. We need to read and write files in many applications.

Java programming language supports both reading and writing text and binary files. But we will cover only to write text file here.

There are several ways to write a text file. Java provides several classes and methods to write a file. They are as:

- Using `FileWriter` Class
- Using `writeString()` method
- By using `BufferedWriter` Class
- By using `FileOutputStream` Class

Let's understand all ways one by one with the help of example programs.

Write File in Java using FileWriter class

The most basic way to write a file in Java is by using `FileWriter` class of `java.io` package.

`FileWriter` in Java is an output stream that writes data as characters into the text file using the platform's default character encoding and buffer size. It is useful when we want to:

- write text into a specific file.
- append text in a file.
- copy text from one file to another file.

`FileWriter` class is a subclass of `OutputStreamWriter` class that extends the `Writer` class.

Write File in Java using FileWriter class

There are the following steps to use `FileWriter` class to write a text file in Java. They are:

1. Create an object of `FileWriter` class associated with the text file we want to write.
2. Use `write()` method to write into a file inherited from Writer class.
3. Close the file.
4. Display a message “`File successfully written…`”.

Example 1: FileWriter

Let’s create a Java program to write a text file using `write()` method inherited from `Writer` class.

Example 1: FileWriter

```
import java.io.FileWriter;
import java.io.IOException;
public class WriteFileTester1 {
   public static void main(String[] args) throws IOException {
       // Create a FileWriter object to open the file.
       FileWriter fw = new FileWriter("./src/writefile1.txt");

       // To write text to the file, call write() method inherited from Writer class.
       fw.write("Welcome to UNO CSCI \n"); // Writing text to the file.
       fw.write("I love Java Programming");

       fw.close(); // Closing the file.
       System.out.println("Successfully written...");
   }
}
```

Example 1: FileWriter

Output:

```
Successfully written...
```

Explanation:

In this example program,

1. We have created a `FileWriter` object, specifying the name of a file as either `String` or `File` reference.

2. The `write()` method inherited from Writer class has been used to write lines of text. Here, we assume that you have the following data in `"writefile1.txt"` file:

```
Welcome to UNO CSCI
I love Java Programming
```

Note:

If you are writing multiple lines of text, must use newline. Without the newline, the next string would start on the same line, immediately after the previous one.

Example 2: FileWriter & Scanner

Let’s create a Java program to write a text file in Java using `FileWriter` and `Scanner` classes. We will prompt to write text into a file from the user. Look at the source code given below.

Example 2: FileWriter & Scanner

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
public class WriteFileTester2 {
   public static void main(String[] args) throws IOException {
       // Create a FileWriter object to open the file.
       FileWriter fw = new FileWriter("./src/writefile2.txt");

       // Create a Scanner class object to take input from the keyboard.
       Scanner sc = new Scanner(System.in);
       System.out.println("Write your content: ");
       String str = sc.nextLine();

       // To write text to the file, call write() method inherited from Writer class.
       fw.write(str); // Writing text to the file.

       fw.close(); // Closing the file.
       System.out.println("Successfully written...");
   }
}
```

Example 2: FileWriter & Scanner

Output:

```
Write your content:
Hello World
Successfully written...
```

writefile2.txt:

```
Hello World
```

Explanation:

In this example program,

1. We have created a `FileWriter` object, specifying the name of a file as either `String` or `File` reference.

2. Use `Scanner` to get text from keyboard as `String`

3. The `write()` method inherited from `Writer` class has been used to write lines of text. Here, we assume that you have the following data in `"writefile2.txt"` file:

Write File in Java using writeString() method

Java 11 has introduced `writeString()` method in the `Files` class. The `writeString()` method writes a sequence of characters such as `String` and `StringBuilder` to a text file.

The general syntax of this method is as follows:

```
Path writeString(Path path, CharSequence csq, Charset cs, OpenOption... options)
```

This method returns nothing. It can take four parameters. These are:

- `path` – It is a file path of data type Path.
- `csq` – It is a sequence of characters as string which we will enter into a file.
- `cs` – It is a UTF-8 Charset used to write the content to file.
- `options` – This parameter provides different options to enter the string in the file, such as append the string to a file, overwrite anything in the file with the current string, etc.

Note: Writing a text file using this method, the first two parameters are mandatory.

Write File in Java using writeString() method

The `writeString()` method can throw four types of exceptions.

They are `IllegalArgumentException, IOException, UnsupportedOperationException,` and `SecurityException`. Therefore, it is better to use when the file content is short.

Example 3: Files & writeString()

Let’s make a Java program to write contents in the text file in Java using `writeString()` method of `Files` class. Look at the following code.

Example 3: Files & writeString()

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
public class WriteFileTester3 {
   public static void main(String[] args) throws IOException {
       // Create a string literal to assign the content of the file.
       String text = "Hello World\nHappy Learning!";

       // Defining the name of the file with location.
       Path fileName = Path.of("./src/writefile3.text");

       // Call writeString() method for writing the content into the file.
       Files.writeString(fileName, text);

       // Call readString() method for reading the content of the file.
       String fileContent = Files.readString(fileName);

       // Displaying the content inside the file.
       System.out.println(fileContent);
   }
}
```

Example 3: Files & writeString()

Output:

```
Hello World
Happy Learning!
```

writefile3.txt:

```
Hello World
Happy Learning!
```

Explanation:

In this example program,

1. We use `writeString()` method of `Files` class to write content into a file.
2. We have used `Path` class to assign `filename` with path where the content of file has written.
3. `Files` class in Java also provides another method named `readString()` to read the content of any existing file. We have used it to check the content is properly written into the file or not.

Write File in Java using BufferedWriter class

- `BufferedWriter` class in Java write text to a character-output stream. It buffers the stream of characters before writing them to an underlying output stream. It has a default buffer size, but we can also assign a large buffer.

- `BufferedWriter` class is useful for writing characters, strings, and arrays. Writing a text file using `BufferedWriter` class is optional.

- We can also write a text file without using a buffer. But using a buffer for writing makes the writing process more efficient.

Example 4: BufferedWriter

So, let's create a Java program to write a text file using `BufferedWriter` class.

There are the following steps to write a text file using `FileWriter` and `BufferedWriter` classes:

1. Create an instance of `FileWriter` class associated with a text file we want to write.
2. Create an object of `BufferedWriter` class associated with `FileWriter` object.
3. Write to the buffer.
4. Flush the buffer.
5. Close the buffer.

Example 4: BufferedWriter

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
public class WriteFileTester4 {
   public static void main(String[] args) throws IOException {
       // Create an object of FileWriter class.
       FileWriter fw = new FileWriter("./src/writefile4.txt");

       // Create an object of BufferedWriter class and reference variable fw to its constructor.
       BufferedWriter bw = new BufferedWriter(fw);

       bw.write("This is an apple");
       bw.newLine(); // For line separator.
       bw.write("This is an orange");
       bw.flush(); // flushing the buffer.
       bw.close(); // Closing the stream.
       System.out.println("File written successfully.");
   }
}
```

Example 4: BufferedWriter

Output:

```
File written successfully.
```

writefile4.txt:

```
This is an apple
This is an orange.
```

Explanation:

The example program shows the use of `BufferedWriter` class to write into a file.

1. The `BufferedWriter` class first packages up many small requests in its own internal buffer.

2. Then when the buffer is full, or `flush()` method is invoked, `BufferedWriter` class writes the whole buffer into a text file in one go.

3. We have used `flush()` method to flush the buffer and `close()` method to close the file.

Write File Using FileOutputStream Class

- `FileOutputStream` class in Java provides methods for writing data to a text file or to a `FileDescriptor`. It stores the content or data as individual bytes.

- But, we can write both byte-oriented and character-oriented data via `FileOutputStream` class. For character-oriented data, you should use `FileWriter` class or `BufferedWriter` class rather than `FileOutputStream`.

Example 5: FileOutputStream

The following example program shows the use of `FileOutputStream` class to write data into a file.

Example 5: FileOutputStream

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
public class WriteFileTester5 {
   public static void main(String[] args) throws IOException {
       try {
           // Store the file path into a variable filepath of type String.
           String filepath = "./src/writefile5.txt";

           // Create FileOutputStream object, passing file path to its constructor.
           FileOutputStream fos = new FileOutputStream(filepath);

           String str = "Welcome to Java!";
           byte bytearray[ ] = str.getBytes(); // Converting string into byte array.
           fos.write(bytearray);
           fos.close(); // Closing file.
           System.out.println("Successfully written...");
       }
       catch(FileNotFoundException e){
           System.out.println(e);
       }
   }
}
```

Example 5: FileOutputStream

Output:

```
Successfully written...
```

writefile5.txt:

```
Welcome to Java!
```

Explanation:

In this example,

1. We used byte array to convert `String` content into a `byte[]` array.
2. The `write()` method will write it into the specified text file.
3. Thus, we can use `FileOutputStream` to write data into a text file easily.

END