Theory: Writing files

16 minutes 9 / 9 problems solved

Start practicing

Now that we've learned how to create and manage files, let's discuss how to write text to a file. Java provides different ways for doing it, and in this lesson, we will consider two of the simplest ways: using the `java.io.FileWriter` and the `java.io.PrintWriter` classes.

§1. The FileWriter class

The class `FileWriter` has a set of constructors to write characters and strings to a specified file:

- `FileWriter(String fileName)`;
- `FileWriter(String fileName, boolean append)`;
- `FileWriter(File file)`;
- `FileWriter(File file, boolean append)`;

Two constructors take an additional parameter `append` that indicates whether to append (`true`) or overwrite (`false`) an existing file.

All these constructors can throw an `IOException` for several reasons:

- if the named file exists but it is a directory;
- if a file does not exist and cannot be created;
- if a file exists but cannot be opened.

In this lesson, sometimes we will skip the exception handling mechanism to simplify our examples.

Let's consider the following code:

```
1  File file = new File("/home/username/path/to/your/file.txt");
2  FileWriter writer = new FileWriter(file); // overwrites the file
3
4  writer.write("Hello");
5  writer.write("Java");
6
7  writer.close();
```

If the specified file does not exist, it will be created after executing this code. If the file already exists, this code overwrites the data.

The file will contain the text **HelloJava**.

If you want to append some new data, you should specify the second argument as true.

```
1    File file = new File("/home/username/path/to/your/file.txt");
2
FileWriter writer = new FileWriter(file, true); // appends text to the file
3
4    writer.write("Hello, World\n");
5    writer.close();
```

This code appends a new line to the file. Run it multiple times to see what happens. Note that here we are using Unix-like OS line breaks. There is a difference between line break characters on different platforms:

- `\n` Unix-like OS
- `\r\n` Windows OS

§2. Closing a FileWriter

3 required topics

- ✓ Formatted output — In project 1
- ✓ Exception handling — In project 1
- ✓ Files — In project

2 dependent topics

- Output streams — In project
- Input streams — In project

Table of contents:

It is important to close a `FileWriter` after using it to avoid a resource leak. This is done by invoking the close method:

```
writer.close();
```

Since Java 7, the convenient way to close an object of `FileWriter` is to use the **try-with-resources** statement.

```
File file = new File("/home/username/path/to/your/file.txt");

try (FileWriter writer = new FileWriter(file)) {
    writer.write("Hello, World");
} catch (IOException e) {
    System.out.printf("An exception occurred %s", e.getMessage());
}
```

It will close the writer automatically.

§3. The PrintWriter class

The `PrintWriter` class allows you to write formatted data to a file. It can output strings, primitive types and even an array of characters. The class provides several overloaded methods: `print`, `println` and `printf`.

```
File file = new File("/home/art/Documents/file.txt");
try (PrintWriter printWriter = new PrintWriter(file)) {
    printWriter.print("Hello"); // prints a string

 printWriter.println("Java"); // prints a string and then terminates the
line
    printWriter.println(123); // prints a number

 printWriter.printf("You have %d %s", 400, "gold coins"); // prints a fo
rmatted string
} catch (IOException e) {
    System.out.printf("An exception occurred %s", e.getMessage());
}
```

This example first creates an instance of `File` and, second, a `PrintWriter` in the **try-with-resources** statement to close it correctly. It writes `"Hello"` and `"Java"` on the same line, and then `123` on a new line. This example also calls the advanced `printf` method which can format a text using `%d`, `%s` and so on. Finally, the `PrintWriter` is closed.

The result contains:

```
HelloJava
123
You have 400 gold coins
```

The class has several constructors. Some of them are similar to FileWriter's constructors:

- `PrintWriter(String fileName)`;
- `PrintWriter(File file)`.

Others allow to pass `FileWriter` as a class that extends the `Writer` abstract class:

- `PrintWriter(Writer writer)`.

§4. Conclusion

`FileWriter` and `PrintWriter` both extend the `Writer` abstract class and have many similarities. However, `PrintWriter` is more of a high-level one and provides several useful methods. Among them are formatting methods and overloaded print methods for writing primitive types.

Report a typo

499 users liked this piece of theory. 9 didn't like it. **What about you?**

Start practicing

Comments (13) Hints (0) Useful links (0) Show discussion