Tutorial for file I/O: writing

In this tutorial, you'll practice opening and writing text into a file. You'll create the flagship product for your new company, Screaming Books, which reads a book file and converts all its text to uppercase.

Your program converts books downloaded from Project Gutenberg. Project Gutenberg is an online library of free eBooks, whose mission is to "to encourage the creation and distribution of eBooks." You can choose from a selection of over 60,000 books to download and read.

This program also tracks the work it completes by adding a message to the end of a log file every time it converts another book.

To get started, open this project into IntelliJ. You'll start your work in the file `BookConverter.java`.

Step One: Review the existing code

The starter code for this tutorial is a simpler version of the book reader you created in the previous tutorial. It asks the user for a book filename and then waits for the user to type and press **Enter**. The program then opens the book file, reads each line, and displays it on the screen. Finally, it prints a message to the user containing the number of lines displayed.

Run this program, and type a filename for a book, such as `data/sherlock-holmes.txt`. You see the book content in the **Run** window, followed by a message showing the number of lines displayed.

Step Two: Open a file for writing

Instead of displaying text to the user, you want your program to write uppercase lines to a new book file. In this step, you open a new file for the converted book, and write lines to the file.

Under the **Step 2:** comment, create a `File` object for writing to, and change the **try-with-resources** line to open a file to read from *and* a file to write to:

```
// Create a File object for the output file
File convertedFile = getConvertedFile(bookFile);
// Open both the input and output files.
try (Scanner fileInput = new Scanner(bookFile);
    PrintWriter writer = new PrintWriter(convertedFile)) {
```

The `getConvertedFile()` method already exists at the bottom of `BookConverter.java`. Feel free to take a look at it. This method creates a file similar in name to the book file, but with ".screaming" inserted before the file extension.

Java creates both resources, `fileInput` and `writer`, on entry into the `try` block. When the block exits, Java releases both resources.

Now that you have the output file open, you can write the converted contents of each line to that file.

Inside the loop, replace the line which calls `System.out.println()` with code that writes text to a `PrintWriter`:

```
// Write the text in uppercase to the output file.
writer.println(lineOfText.toUpperCase());
```

The last thing you must do to complete this step is update the message to the user. Since you're no longer displaying the book, update the message which follows the loop to inform the user of the conversion:

```
// Tell the user what happened.
String message = "Converted " + lineCount +
        " lines of file " + bookFile.getName() +
        " to " + convertedFile.getName() +
        " on " + new Date();
System.out.println(message);
```

Run the program and type in a valid book file path. You see something like this:

```
Enter path to the book file: data/sherlock-holmes.txt
Converted 12305 lines of file sherlock-holmes.txt to sherlock-holmes.screaming.txt
on Tue May 25 13:39:17 EDT 2021
```

Look in the `data` folder, where the original book file is. You now see a file with a similar name, but containing ".screaming" in its name, such as `sherlock-holmes.screaming.txt`. Open that file, and notice all its text is uppercase.

Step Three: Write a message to a log file

Next, you want your program to write a message logging its actions every time it runs. This way, you'll have a historical account of all the file conversions. To do this, you must open another file for write access.

When you opened the output file for write in the previous step, your work *overwrote* an existing file if there was one. Since you want to maintain history in a log file, you don't want this behavior. Instead you must open the file for *append*.

After the **Step 3:** comment, add the following code:

```
String auditPath = "BookConverter.log";
File logFile = new File(auditPath);
// Using a FileOutputStream with true passed into the constructor opens the file
for append.
try (PrintWriter log = new PrintWriter(new FileOutputStream(logFile, true))) {
    log.println(message);
} catch (FileNotFoundException e) {
    System.out.println("*** Unable to open log file: " +
```

```
logFile.getAbsolutePath());
}
```

This code creates a new `PrintWriter` for you to add lines to the file, but if the file previously existed, your new lines get added at the end of the file.

After you run your program a few times, look in the file `BookConverter.log`. You see new messages added each time you run it, without removing the messages that were already there:

```
Converted 12305 lines of file sherlock-holmes.txt to sherlock-holmes.screaming.txt
on Mon May 24 16:24:23 EDT 2021
Converted 6206 lines of file fairy-tales.txt to fairy-tales.screaming.txt on Mon
May 24 16:50:31 EDT 2021
Converted 2933 lines of file jekyll-and-hyde.txt to jekyll-and-hyde.screaming.txt
on Tue May 25 10:46:18 EDT 2021
```

Next steps

Inspecting existing code is a great way to learn to program. In this tutorial, the method `getConvertedFile()` was already there for you when you started. Its job is to take a filename, and create a new name with the string ".screaming" inserted in the filename.

It uses `File.getAbsolutePath()`, `String.lastIndexOf()`, and two overloads of `String.substring()`. Examine this method until you understand how it works.