CPSC250L Lab 4
Text I/O and `String` Formatting

Fall 2016

1 Introduction

The focus for this lab is file input and output as well as string formatting. Check the JavaDoc for `File`, `Scanner`, `PrintWriter`, and `String.format`.

2 Exercises

2.1 Line Numbers

In this exercise, you will read in a file, prepend line numbers to each line, and then output it to another file.

Exercise 1

Create a class named `LineNumbers` and implement the following method in it.

- `public static void process(File input, File output)`

  This method receives a text file. The method writes all lines from the input file to the output file with each line preceded by its line number formatted as follows:

```
### | Original Line
```

The line number should be right aligned in three spaces, be followed by a space, then a pipe, another space, and lastly the original line. For example:

Input File:

```
This is a line of text
Yet another line
You only live twice
⋮
This is the last line
```

Output File:

```
  1 | This is a line of text
  2 | Yet another line
  3 | You only live twice
⋮
123 | This is the last line
```

Test your code against `LineNumbersTest.java`. When all tests pass proceed to the next exercise.

Exercise 1 Complete

Run:

```
git add .
git commit -m "Completed exercise 1"
git push origin master
```

2.2 Cookie Jar

Exercise 2

Create a class named `CookieJar` and implement the following method in it.

- `public static void cashingIn(File input, File output)`

  This method receives an input file with your count of coins given as pairs of integer and type of coin, e.g., `1 penny`, `42 quarters`, `9 nickels`. The method reads number/coin-type pairs from the input file and write the total dollar amount to the output file. For example, given the input file `2 quarters 4 dimes 1 penny 3 nickels 3 pennies`, the method prints `You have $1.09 in the jar` to the output file. The dollar amount should be formatted in 2 decimal places and with commas (when appropriate). In cases when the total amount in the jar is zero the method writes

  Code:

  ```
  You have no money in the jar
  ```

  to the output file. The input file may contain coin pairs in one line or across different lines, may contain several counts of the same type of coin at different times, and may list the type of coin in plural or singular form, e.g., `pennies` for many 1 cent coins or `penny` for one coin. Pennies are worth 1 cent each, nickels are worth 5 cents each, dimes are worth 10 cents each, and quarters are worth 25 cents each.

Test your code against `CookieJarTest.java`. When all tests pass, proceed to the next exercise.

Exercise 2 Complete

Run:

```
git add .
git commit -m "Completed exercise 2"
git push origin master
```

2.3 Album and Comparable

In this exercise, we implement a `Comparable` type called `Album`.

Exercise 3

Open the `Album` class provided in the Git repository and implement the following methods.

1. `public Album(String _name, String _author, int _length)`

   This constructor receives two `String`s representing the album name and band name respectively, as well as an `int` which represents the `Album`'s runtime in seconds. You should set the fields in `Album` to these values.

2. `public int compareTo(Album other)`

   This method receives another `Album` object and returns a negative value if this `Album` is *less than* `other`, 0 if the `Album`s are the *same*, and a positive value if this `Album` is *greater than* `other`. The comparison is described as follows.

   (a) If `author` does not equal `other.author`, return the lexicographical comparison of `author` and `other.author`.

   (b) Else if `name` does not equal `other.name`, return the lexicographical comparison of `name` and `other.name`.

   (c) Otherwise, return the difference between `length` and `other.length`.

3. `public String getName()`

   This method returns the `Album`'s `name`.

4. `public String getAuthor()`

   This method returns the `Album`'s `author`.

5. `public int getLength()`

   This method returns the `Album`'s `length`.

Test your code against `AlbumTest.java`.

Exercise 3 Complete

Run:

```
git add .
git commit -m "Completed exercise 3"
git push origin master
```

2.4 Discography

For this exercise, we will read a list of `Album`s from a file, sort them, and then output the sorted list to another file.

Exercise 4

Your local library has compiled a list of all CD albums available in their collection. The

information is written in a comma-separated value (CSV) text file, where each line has the name of an album, its author and the length of every song in that album (in minutes and seconds). For example, the entry for the CD "Best of Elmo" is "Best of Elmo,Sesame Street,2:29,1:30,2:09,1:46,1:55,2:02,1:42,2:40,
1:56,1:30,2:03,1:14,2:28,2:47".

Given an input text file with a group of CD data (each album in one line) the library wants all albums sorted and formatted for easy reading.

The output is formatted as follows.

- The first column lists authors, which are left aligned in as many spaces as the length of the longest author in the input file. Authors are sorted alphabetically.
- The second column lists album names, which are left aligned in as many spaces as the longest album name in the input file. Albums from the same author will be sorted alphabetically by name.
- The third column list the total running time of albums, which are in the form H:MM:SS, where H is hours (0-9), minutes (0-59), and seconds (0-59). Minutes and seconds are displayed in 2 spaces (with a leading zero if needed).

Each column is divided by a `|` character (with a space before and after).

For example input and output look at the `sample-discography-input1.txt` and `sample-discography-output1.txt` included with the Git repository.

Create a `Discography` class and implement the following method.

- `public static void writeReport(File input, File output)`

  This method should read in each `Album` from the input files into an `ArrayList` of `Album` objects. You should then use `Collections.sort` to sort your `ArrayList`. Then output each album's information to the output file using the above formatting guidelines.

  Hint: If you want, you can implement a `toString()` in `Album` that returns the `Album`'s data in the expected format.

Test your code against `DiscographyTest.java`.

Exercise 4 Complete

Run:

```
git add .
git commit -m "Completed exercise 4"
git push origin master
```

---

3 Common Mistakes

The following are warnings about and solutions to common mistakes for this lab.

1. Be sure to close your `Scanner`s and `PrintWriter`s!
2. When iterating through a `File`'s contents, ensure that you do not go past the end of the file.
3. Pay close attention to the format of your output.
4. Ensure that you handle any exceptions that your code may throw.
5. In the `Discography` exercise, use the `Album` class!