Homework 06 – Course Planning

Authors: Pratham, Emma, Lucas, Chelsea
Topics: File IO and Exceptions

Problem Description

Please make sure to read the document fully before starting!

Congratulations, you got admitted to Georgia Tech! As you begin your first year here, you find it difficult to choose which courses to take. To help you decide and improve your organization, you create a database of courses available to you.

Solution Description

Create files:

- `Course.java`
- `ComputerScience.java`
- `LabScience.java`
- `InvalidCourseException.java`
- `Classes.java`

You will be creating several fields and methods for each file. Based on the description given for each variable, you will have to decide whether these variables/methods should be static, and what visibility modifier applies. You should carefully follow the guidelines on these keywords as taught in lecture. In some cases, your program may still function with an incorrect keyword. Private helper methods are allowed and encouraged.

Course.java

This class represents a Course, and it is an abstract class.

~~**Variables:**~~

~~All variables should only be visible to subclasses of Course, unless specified otherwise:~~

- ~~`String courseName` – the name of the course, ex: "CS1331: Intro to OOP in Java"~~
- ~~`int id` – the course identification number, ex: 23547.~~
- ~~`String professorName` – the name of the professor teaching the course, ex: "Ricky Landry"~~

~~**Constructor(s):**~~

- ~~A constructor that takes in `courseName`, `id`, and `professorName`.~~
  - ~~If the passed value for `courseName` and `professorName` is an empty string or is null, throw an `IllegalArgumentException`.~~

- If the passed in value for `id` is not a 5-digit number or is negative, throw an `IllegalArgumentException`. Assume `id` will **not** have any leading zeros.

Methods:

All methods in this class should be public:

- `toString`
  - Should properly override Object's `toString` method and return:

    "`{courseName},{id},{professorName}`"
- `equals`
  - Should properly override Object's `equals` method
  - Two Course objects are equal if they have the same `courseName`, `id`, and `professorName`.
- Any **necessary** getter and setter methods (emphasis on **necessary** – don't write any that have no use)

ComputerScience.java

This class describes a computer science course and is a concrete implementation of Course.

Variables:

All variables must not be allowed to be directly modified outside the class in which they are declared:

- `String language` – the language in which the class is taught

Constructor(s):

- A constructor that takes in `courseName`, `id`, `professorName`, and `language`.
  - If the value passed for `language` is an empty string or is null, throw an `IllegalArgumentException`.
  - Reuse as much code as possible.

Methods:

All methods in this class should be public:

- `toString`
  - Should properly override Course's `toString` method and return:

    "`ComputerScience,{courseName},{id},{professorName},{language}`"
- `equals`
  - Should properly override Course's `equals` method
  - Two ComputerScience objects are equal if they have the same `courseName`, `id`, `professorName`, and `language`.
- Any **necessary** getter and setter methods (emphasis on **necessary** – don't write any that have no use)
- Reuse as much code as possible.

LabScience.java

This class represents a lab science and is a concrete implementation of Course.

Variables:

All variables must not be allowed to be directly modified outside the class in which they are declared:

- `boolean labCoatRequired` – a boolean representing whether a lab coat is needed

Constructor(s):

- A constructor that takes in `courseName`, `id`, `professorName`, and `labCoatRequired`.
  - Reuse as much code as possible.

Methods:

- `toString`
  - Should properly override Course's `toString` method and return:

    "`LabScience,{courseName},{id},{professorName},{labCoatRequired}`"
- `equals`
  - Should properly override Course's `equals` method
  - Two LabScience objects are equal if they have the same `courseName`, `id`, `professorName`, and `labCoatRequired`.
- Any **necessary** getter and setter methods (emphasis on **necessary** – don't write any that have no use)
- Reuse as much code as possible.

InvalidCourseException.java

This class describes an unchecked exception.

Constructor(s):

- A constructor that takes in a String representing the exception's message
- A no-args constructor that has the default message "Invalid course type!"

Classes.java

This class will hold various public static methods that will let you read and write to the database.

Methods:

All methods must have proper visibility:

- `outputCourses`
  - Takes in a String object representing the file name to read from
  - Throws a `FileNotFoundException` if the passed in file is null or doesn't exist
  - Returns an ArrayList of Course objects

- Each line of the file will contain information about different types of courses
  - File line tokens:

    `ComputerScience,courseName,id,professorName,language`

    or

    `LabScience,courseName,id,professorName,labCoatRequired`
  - Iterate through the file and create the appropriate Course object for each line
  - Add each Course object to ArrayList
  - If the type of course is not ComputerScience or LabScience, throw an `InvalidCourseException`
  - **NOTE:** You can assume that the id contained in the file will be in the correct format for a number (i.e. won't contain any characters).

- `writeCourses`
  - Takes in two parameters:
    - A String object representing the file name to write to
    - An ArrayList of Course objects
  - Returns a boolean value representing whether the write was successful
  - Iterate through the ArrayList and write each Course object to its own line
  - **Note:** if there is already information in the given file, make sure you don't delete the data. Add your courses to the existing file instead.
    - **IMPORTANT: java.io.FileWriter is NOT part of the allowed imports.** Consider other methods you can use from this class to accomplish this goal.
    - Make sure to catch any exceptions required of you! Upon catching, think about the best way to inform a user about what happened. This could be in the form of a print statement. Remember we only want to return false if NO writing occurred (regardless of what file was written to).

- `readCourses`
  - Takes in two parameters:
    - A String representing the file name to read from
    - A Course object
  - Throws a `FileNotFoundException` if the passed in file is null or doesn't exist
  - Returns an ArrayList of Integer objects
  - Iterate through the file:
    - Search for the inputted Course object
    - When you find this inputted Course object, add its line number onto the ArrayList
    - Lines will start counting from **1**
  - If the inputted Course object is not found, throw an `InvalidCourseException`
  - **EXAMPLE:** If the Course object is found on lines 4, 6, and 10, return an ArrayList containing Integers 4, 6, and 10.

- `main`
  - Create three `ComputerScience` and three LabScience objects
  - Write these objects into a file called "TestCourses.csv"
  - Create another `ComputerScience` object and add it to "TestCourses.csv" – do not overwrite existing data!
  - Read this csv file using `outputCourses` and print each object to a new line

- **NOTE:** This method will **NOT** be graded and are suggestions to help you test your code and is by no means comprehensive. You are encouraged and recommended to write your own test cases.

Reuse your code when possible. Certain methods can be almost completely reused using certain keywords. The tests on Gradescope are not comprehensive, so ensure that you create your own!

Checkstyle

You must run checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle cap for this assignment is 35 points.** This means there is a maximum point deduction of 35. If you don't have Checkstyle yet, download it from Canvas -> Modules -> CheckStyle Resources -> checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned in the Rubric section). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Course.java`
- `ComputerScience.java`
- `LabScience.java`
- `InvalidCourseException.java`
- `Classes.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any**

autograder test are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work. You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via Piazza for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import the following:

- `java.util.ArrayList`
- `java.util.Scanner`
- `java.io.File`
- `java.io.FileNotFoundException`
- `java.io.PrintWriter`

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW assignments broadly, that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs**

It is expected that everyone will follow the Student-Faculty Expectations document, and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. No inappropriate language is to be used, and any assignment, deemed by the professor, to contain inappropriate, offensive language or threats will get a zero. You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.