Homework 05 – Course Planning

Topics: file IO, 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

You will need to write and submit five classes: Course.java, ComputerScience.java, LabScience.java, InvalidCourseException.java, and Classes.java.

Notes:

1. All variables should be inaccessible from other classes and must require an instance to be accessed through, unless specified otherwise.
2. All methods should be accessible from everywhere and must require an instance to be accessed through, unless specified otherwise.
3. When throwing exceptions, provide descriptive and specific messages.
4. Private helper methods are optional, and **reuse as much code as possible!**

Course.java

This abstract class represents a course in general.

Variables:

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

- `String courseName` – the name of the course
- `int id` – the course identification number
- `String professorName` – the name of the professor teaching the course

Constructors:

- A constructor that takes in `courseName`, `id`, and `professorName`.
    - If `courseName` or `professorName` is an empty string or null, or If `id` is not a five-digit number or is negative, throw an `IllegalArgumentException`.

Methods:

- `toString`
    - This method should properly override Object's `toString` method.
    - It should return a String in the following format (no space following commas):

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

`"{courseName},{id},{professorName}"`

- `equals`
  - This method should properly override Object's `equals` method.
  - Two Course objects are equal if they have the same `courseName`, `id`, and `professorName`.
- **Necessary** getter and setter methods.

ComputerScience.java

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

Variables:

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

Constructors:

- A constructor that takes in `courseName`, `id`, `professorName`, and `language`.
  - If `language` is an empty string or null, throw an `IllegalArgumentException`.

Methods:

- `toString`
  - This method should properly override Course's `toString` method.
  - It should return a String in the following format (no space following commas):

    `"ComputerScience,{courseName},{id},{professorName},{language}"`
- `equals`
  - This method should properly override Course's `equals` method.
  - Two ComputerScience objects are equal if they have the same `courseName`, `id`, `professorName`, and `language`.
- **Necessary** getter and setter methods.

LabScience.java

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

Variables:

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

Constructors:

- A constructor that takes in `courseName`, `id`, `professorName`, and `labCoatRequired`.

Methods:

- `toString`
  - This method should properly override Course's `toString` method.

  - It should return a String in the following format (no space following commas):

    `"LabScience,{courseName},{id},{professorName},{labCoatRequired}"`

- `equals`
  - This method should properly override Course's `equals` method.
  - Two LabScience objects are equal if they have the same `courseName`, `id`, `professorName`, and `labCoatRequired`.
- **Necessary** getter and setter methods.

InvalidCourseException.java

This class describes an unchecked exception that indicates an attempt to create an invalid course.

Constructors:

- A constructor that takes in a String representing the exception's `message`.
- A no-argument constructor that defaults the message to "Invalid course type!".

Classes.java

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

Methods:

- `outputCourses`
  - Takes in a String representing the name of the file to read from.
    - Throws a `FileNotFoundException` if the file name is null or a file with the specified file name does not exist.
  - Returns an ArrayList of Courses.
  - Each line of the file contains information about different types of courses.
    - File line tokens will be in the following format, without the curly braces:

      `{CourseType},{courseName},{id},{professorName},{language/labCoatRequired}`

    - **Note:** You can assume that the `id` contained in the file will be in the correct format for a number (i.e, will only contain digits), and that `labCoatRequired` can be parsed into a boolean.
    - **Note:** There can be any number of lines in the file, but lines will not be empty.
  - Iterate through the file and create the appropriate Course object from each token.
  - Add each course to the ArrayList.
  - If the course type is not `ComputerScience` or `LabScience`, throw an `InvalidCourseException`.
- `writeCourses`
  - Takes in a String representing the file name to write to and an ArrayList of Courses.
  - Returns a boolean value representing whether the write was successful.
  - Iterate through the ArrayList and write each Course object to its own line.

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

  - **Note:** If the file already contains information, be careful not to overwrite the existing data. Instead, add the courses to the end of the existing file.
    - **IMPORTANT: java.io.FileWriter is NOT an allowed import.** Use the recommended read-modify-write pattern to implement this method.
    - Be sure to catch any exceptions that this method may throw. Remember that we need to return false if NO writing occurred.

- `readCourses`
  - Takes in a String representing the file name to read from and a Course object.
    - Throws a `FileNotFoundException` if the file name is null or a file with the specified file name does not exist.
  - Returns an ArrayList of Integers.
  - Iterate through the file and search for the inputted Course object.
    - For each line that the inputted Course object is found on, add its line number to the ArrayList.
    - Lines should 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 named "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. It is by no means comprehensive. We recommend that you write your own test cases.

Be sure to reuse as much as your code as possible. A large portion of some methods can be implemented by reusing code. The suggested tests and those on Gradescope are by no means comprehensive, so we strongly recommend that you create your own!

Checkstyle

You must run Checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-uide.pdf under the Checkstyle Resources module on 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). In future assignments 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 the class forum 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 assignment (submit early and often). 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

- `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 autograder. 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 assignment 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.

You **MAY NOT** use code generation tools to complete this assignment. This includes generative AI tools like ChatGPT.

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,

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

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.

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.