Homework – Course Availability

Authors: Kevin , Ryan, Alex, Ariel, Nick, Connor, Andrew, Jack

Purpose

In this homework you will utilize recursion and Java's File I/O classes to search through directories and read files. You will also have to write, throw, and handle exceptions.

Background

You are a new student at FASET, and you are extremely confused about your schedule. Thankfully, your FASET leader is here to save the day! They give you the list of classes you must take and a folder containing the course availability for every major — but you have to check that these classes have seats remaining for you. Fortunately, you are an expert on the topics of recursion, file I/O and exceptions, and realize that you can use these three to implement a search algorithm that will check for you!

Solution Description

For this assignment, you will be writing `Course` and `CourseDataBase` classes, the latter in which you will implement a search function to check for the availability of each of your courses. In addition, you'll create your own checked and unchecked exceptions, `CourseNotFoundException` and `NotAMajorException`. You will write your code in the provided but empty `.java` files in the `src` folder of ***database_courses.*** **Do not move these folders around.** *(Note: Despite your .java files being in a special folder, do not include a package statement.)*

Course Database Structure

For this assignment, you are given a directory called **database_courses**, containing folders of several majors and the respective classes available for each of those majors.

Major directories will be named **NameOfTheMajor**; for example, *CS*, *BME*, etc. Subsequently, the course level subdirectories defined in each of the Major directories will be named **Level**. For example, 1000, etc. The course directories' level will only contain MajorCourseNumber.txt files that will contain the course availability status. For example, CS1331.txt, CS1332.txt. If the course is available, the file will contain "AVAILABLE", otherwise will contain "UNAVAILABLE".

An example of the structure of the database:

```
database_courses:
|____ BME
      |____ 1000
            |____ BME1000.txt
            |____ BME1300.txt

|____ CS
      |____ 1000
            |____ CS1301.txt
            |____ CS1331.txt
            |____ CS1332.txt
|____src
      |____ Course.java
      |____ CourseDataBase.java
      |____ CourseNotFoundException.java
      |____ NotAMajorException.java
```

`Course.java`

This file represents a `Course` object. This is a class that you will write inside the provided Course.java file. In it you will write various methods that implement sorting and searching functionality.

- Variables
  - `String name`
    - This represents the name of the course
    - For example, CS1331, CS1332 etc.
  - `int level`
    - This represents the level of the course
    - These will *only* take on the values of: *1000*, *2000*, *3000*, *4000*
  - `String major`
    - This represents the prefix of the course
    - These will take on the values of: *BME*, *CS*, *ISYE*, *MGT*
  - `boolean availability`
    - This represents whether there are seats available in the course
    - *true* — there are seats available
    - *false* — there aren't seats available
- Constructors
  - The Course class has exactly one 4-argument constructor that initializes its 4 instance variables accordingly.
  - The constructor must take its arguments in the following order: `name, level, major`, and `availability`. The types of the parameters must correspond to the types of the fields.
- Methods
  - Getters and setter for all the variables
  - private helper methods as needed

`CourseDataBase.java`

This file represents a `CourseDataBase` object. In it you will implement various methods that will recursively search for the availability of a course. `CourseDataBase` should be a class.

Hint: Helper methods are generally a key for recursion and can help to structure recursive functions (Remember encapsulation— will any helper methods be ever called from outside the class?)

- Fields: None
- Constructors: None
- Methods
  - Static method `loader`
    - Takes in a `File` object as a parameter
    - Calls the helper method `loadhelper`
    - Returns an ArrayList of `Course` objects representing each course you found in the database
  - Static method `loadHelper`
    - Takes in a `File` object and an `ArrayList` of `Course` objects
    - Implement a recursive search method that will instantiate a `Course` object representing every course in the provided database and initialize their instance variables accordingly.
    - Add the newly created `Course` object to the `ArrayList`
    - If there are no courses listed in a folder, move on to the next.
    - **Hint**: If the `File` is a directory, continue to recurse by calling `loadHelper`. In addition, look at the `File` API for methods to: get the number of files in a

directory, as well as a list of all the available paths, check whether the current file path is a directory or not, and get files inside of the directory.

- Static method `getFilePath`
  - Takes in a `File` object and a `Course` object and recursively finds and returns a `String` array representing the `FilePath` to get from the outermost directory in `File` to the course's .txt file
  - For example, for CS1331, it should return: `["CS", "1000", "CS1331.txt"]`
  - If the major isn't in the directory, throw `NotAMajorException`
  - If the major is in the directory but the course isn't, throw `CourseNotFoundException` with the message, "Course {Course.name} was not found in the database"

- Static method `scheduleAvailability`
  - Takes in a `File` object and an array of `Course` objects, representing the schedule of a student
  - For each of the `Course` objects in the array, search through the database of files and determine if that particular `Course` object is available.
  - Return true if and only if ***all*** of the Courses listed in `Schedule` are available
  - **Hint**: What methods have we written already that could help us check the availability of Courses listed in the database?
  - **Note:** It is not sufficient to rely on the `availability` field in the `Course` objects. You must search through the database.

`NotAMajorException.java`

Unchecked exception that has a no-args constructor that use that super class's constructor to have the message of "That is not a major we know!" Hint: Think about what makes an exception unchecked vs checked.

`CourseNotFoundException.java`

Checked exception that has a 1-args constructor that correctly initializes the message field (hint: use the super constructor)

Import Restrictions:

To prevent the trivialization of this assignment, you may import only the following:

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

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`

Tips & Tricks

As always, feel free to create a driver class to test functionality of the various classes and methods we have asked you to implement. You will also find the Java File API super handy! Take advantage of this documentation as you

will continue to use it in 1331 and beyond. Take advantage of Office Hours for this assignment and start early — recursive methods can be tricky to debug, and we're here to help.

Collaboration Statement

- To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:
- *I worked on the homework assignment alone, using only course materials.*
- or
- *In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].*
- Recall that comments are special lines in Java that begin with `//`.
- You will be deducted points if you do not include this statement.

Turn-In Procedure

Submission

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

- `Course.java`
- `CourseDataBase.java`
- `NotAMajorException.java`
- `CourseNotFoundException.java`

Make sure you see the message stating "HW05 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.
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 last submission: be sure to submit every file each time you resubmit.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. 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. forgetting Checkstyle, 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.

Checkstyle and Javadocs

You must run Checkstyle on your submission. **The checkstyle cap for this assignment is 30 points**. If you don't have Checkstyle yet, download it from Canvas. Place it in the same folder as the files you want Checkstyle. 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 points we would take off (limited to the amount we specified above). The Java source files we provide contain no Checkstyle errors. In future homework assignments we may be increasing this cap, so get into the habit of fixing these style errors early!

Along with this, you will also Javadoc your code.
Run the following to only check your Javadoc.

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

Run the following to check both Javadoc and Checkstyle.

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

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 Piazza for all official clarifications