Homework 08 - Yelp

Oh no! You want to get lunch but there are so many restaurants to pick from and so many types of food to try. After receiving a database of all the foods and restaurants in the area from the Mayor, you decide to create a program to help you use the databse.

Purpose

The purpose of this assignment is to apply recursion, `File I/O`, `Exception`s and `ArrayList`s to a working Java Program. Additionally, you will be able to expand upon your knowledge of command line args to take in arguments.

Overview

Your task is to implement methods for working with the Yelp "database" (which is given to you as a set of nested folders). The structure is as follows: the outermost directory (or folder) is called `database_restaurants` and contains further folders of types of food (such as fastfood, asian, etc.), then within that, restaurants associated with that type of food. From there, each restaurant has multiple reviews that are `.txt` files.

You task is to create methods for working with the database. One method will recursively traverse the "database", find all the restaurants, and return them through an `ArrayList` of `Restaurant` objects. Another method will count the total number of reviews in all the restaurants. The final method will traverse the folders to find a specific review in a specific restaurant to read and return the rating that the review gave.

Be sure to download `database_restaurants.zip` from Canvas, place it in the same directory as your code, and unzip it. This will help you understand the file structure of the "database".

Restaurant Database File Structure

You are given a directory called `database_restaurants` containing folders of several different types of food. Those folders for food-type will contain either more folders of a more specific food-type or folders for restaurants. Folders for restaurants always have reviews inside as .txt files.

`Restaurant` directories will be named `restaurantName`. For example, restaurantMoes, restaurantMcdonalds. `Restaurant` directories contain only `.txt` files with reviews. Reviews are named with a prefix of review and then a suffix of the reviewers name (such as reviewJohn or reviewSarah).

An example of the folder structure is:

```
database_restaurants
|____ food_type1
        |____ restaurantName1
                |____ reviewNeethu
|____ food_type2
        |____ moreSpecificFoodType
                |___restaurantName2
                    |____ reviewMaddy
                    |____ reviewFelipe
```

Keep in mind there could be several nested types of food, for example you could have a path to a review of:

```
database_restaurants -> asian -> sushi -> restaurantSatto -> reviewSusan
```

Directions

Functionality: You will code the following functionality: * You will recurse through the nested folders and create an `ArrayList` of all the `Restaurants` found in the folder. * You will also implement a method that counts the total number of reviews in all the restaurants. * Finally, you will create a method to read a specific review for a `Restaurant` in the database. * If the restaurant isn't in the database, throw a `NotARestaurantException`. * If the review is not in the database, throw a `ReviewNotFoundException`.

Class Details

Restaurant.java

Instance Data

- String `name` - the restaurant name
- int `numReviews` - the number of reviews in a restaurant
- **Don't add any other fields**

Constructor

- 2-param constructor that creates a `Restaurant` object with the `name` and `numReviews` passed in

Methods

- getter for `name`
- getter for `numReviews`

NotARestaurantException.java

- **unchecked exception** that has a no-args constructor that uses the super class's constructor to have the message of "That is not a restaurant we know!"

ReviewNotFoundException.java

- **checked exception** that has a 1-args constructor that takes in a message of and when the exception is printed, it should have the message you pass in (hint: use the super constructor)

YelpDB.java

This class will hold a number of static methods for working with the database. **HINT** Helper methods are generally a key to recursion and can definitely help structure recursive functions (Remember encapsulation - will any helper methods need to be called from outside the class?)

Methods in YelpDB.java

- Method `main` (this is your `main` method)
  - Take in the `restaurantName` and `reviewName` using the command line argument
    - To pass arguments to your program via command line, run `YelpDB` with `java YelpDB restaurantName reviewName` instead of just `java YelpDB`. So for example, `java YelpDB Moes Shishir`
    - Make sure your commands are in that order - restaurantName, then reviewName.
    - These arguments could be used later in the main method for `readReview`

  - Create a `File` object for the `database_restaurants` directory. The `YelpDB` class should expect the "database" folder (called `database_restaurants`) to be in the same directory as itself, `YelpDB.java`. This `File` could be used later in the main method for testing code.
  - Be sure to **catch a `ReviewNotFoundException` or a `NotARestaurantException` from any methods you may call in the main method and print out the message.**
  - Use this method as a tester to test your code!
- Static Method `load`:
  - Take in a `File` object as the parameter
  - Call the helper method `loadHelper`
  - Return an `ArrayList` of `Restaurant` objects
- Static Method `loadHelper`:
  - Take in a `File` object and an `ArrayList` of `Restaurant` objects as the parameters
  - If the `File` is a directory and not a restaurant directory, **recursively** call the method on **all nested folders**
  - However if the directory is a restaurant (**remember** restaurant directories in the folders are named `restaurantName` where the suffix 'name' is the name of the specific restaurant)
    * create a `Restaurant` object with the restaurant name found and the **number of reviews in that restaurant (look at the File API for methods you could use to get the number of files in a folder)**
    * add the `Restaurant` you created to the `ArrayList`
  - Return the `ArrayList` of `Restaurant` objects
- Static Method `countNumReviews`:
  - Take in the `ArrayList` of `Restaurants`
  - Iterate through all the `Restuarants` in the `ArrayList` and find the cumulative sum of the number of reviews in all the `Restaurants`
  - Return the total number of reviews you found
- Static Method `readReview`:
  - Takes in a `File` object to search inside, an `ArrayList` of `Restaurants` representing the `Restaurants` in the database, a `String restaurantName` we are searching for, and a `String reviewName` find which review from that `restaurantName` we want to read. **Assume all inputs are the correct - no validation needed**
    * the restaurantName parameter will not start with "restaurant"; i.e. restaurantName would be "Mcdonalds", **not** "restaurantMcdonalds".
    * the reviewName parameter will just be the name of the person who wrote the review; i.e. reviewName would be "John" **not** "reviewJohn"
    * if that `Restaurant` is not in the database, throw a `NotARestaurantException` with the message "That is not a restaurant we know!". (**Hint:** How can you check if the `Restaurant` is in the database using an `ArrayList` method?)
  - If the `Restaurant` is in the database, find the `Restaurant` **recursively**
  - Read the appropriate review based on the `reviewName` using a Scanner
    * If the `reviewName` isn't in the `Restaurant`, be sure to throw `ReviewNotFoundException` with a message of "That is not a review!"
  - Catch any `FileNotFoundException` that could occur
  - Return the rating from the review you read!

-> **Keep in Mind when reading reviews** All reviews are written in the following format and are named `review{name}` where '{name}" is the name of the reviewer:

```
Name: John
Date: 02/02/20
Rating: 1.0
```

Review of Classes Involved:

- `YelpDB.java`
  - This class holds static methods for accessing the database
  - You will create a method to access the database and make an `ArrayList` of all the Restaurants found in it
  - You will create a method that counts the total number of reviews in an `ArrayList` of all the `Restaurants`
  - You will create a method to read a specific review from a specific `Restaurant` from the database
  - You will create the `main` method, which will be calling the static methods to get the data returned by them and use them for other methods
- `Restaurant.java`
  - This class stores information about the restaurants you find in the database
  - It holds the name of the restaurant and the number of reviews the `Restaurant` has
- `NotARestaurantException.java`
  - This class is a user created **unchecked** exception that you will use when checking if a `Restaurant` you want to read a review from is actually in the database
- `ReviewNotFoundException.java`
  - This class is a user created **checked** exception that you will use when checking if a review you want to read from a `Restaurant` is actually in the database

Allowed Imports

To prevent trivialization of the assignment, you may only import:
- `java.util.Scanner`
- `java.util.ArrayList`
- `java.io.File`
- `java.io.IOException`
- `java.io.FileNotFoundException`

Checkstyle and Javadocs

You must run checkstyle on your submission. The checkstyle cap for this assignment is **20** points. If you don't have checkstyle yet, download it from Canvas -> Files/Resources. Place it in the same folder as the files you want checkstyled. 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 above). The Java source files we provide contain no Checkstyle errors. 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
```

Feature Restrictions

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

Collaboration

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 `//`.

Turn-In Procedure

Submission

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

- `YelpDB.java`
- `Restaurant.java`
- `ReviewNotFoundException.java`
- `NotARestaurantException.java`

Make sure you see the message stating "HW08 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. Each test typically corresponds 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:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. 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.

Important Notes (Don't Skip)

- You may add extra private variables or methods to facilitate your program, but you may not alter the arguments in the constructor
- Take a look at the File API if you are stuck
- Check on Piazza for all official clarifications
- 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