Homework 8

Sylvia Necula

March 17, 2016

1 Introduction

Please read the entire document and **do not forget to put the collaboration statement at the top of your submission!**

2 Problem Description

In this week's homework, you will be creating your own version of Yelp! Yelp is a website for food lovers that allows you to view and add restaurant recommendations. Users have the ability to add reviews and search for restaurants using various criteria such as price range and cuisine. This homework is mainly focusing on file input and output and exceptions, but good style about everything you have learned up to this point still applies!

3 Solution Description

The Yelp application is made up of the Yelp database, reviews, and restaurants. Information about the restaurants and reviews stored in the database are contained in the YelpDB.txt file and must be loaded every time the database is initialized. Likewise, when the program ends, any information about restaurants and reviews in the database must be stored in the text file. There are also two exceptions that may occur when loading or saving information to the text file, in addition to java's FileNotFoundException.
A Driver file is included to allow you to test the functionality of your code. Remember that any instance data you choose to create for your classes must be set to private so you should also include the appropriate methods for other classes to obtain instance data. Additionally, all search methods and equals methods you write should ignore the case of the Strings being compared.

3.1 Driver.java and YelpDB.java

Weve provided you with a driver file that can be used to test functionality once all of the other classes are properly implemented. Feel free to add your own code as well for testing purposes, but we will not be grading you on your testing code.
The YelpDB.txt file is where all of the information about restaurants and reviews should be loaded and saved to every time the program is run. This file has a very specific format which must be adhered to in order for all of the information to be loaded properly. The format is outlined below.

```
2  //the number of restaurants in the database
Antico Pizza //name of first restaurant
Pizza //cuisine of first restaurant
4.0 //average rating
$ //price range
1 //number of reviews for the first restaurant
4 //review rating
3/12/16 //review date
Waffle House //name of second restaurant
Breakfast & Brunch //repeats
4.5
$
2
5
3/8/16
4
3/10/16
```

As long as you follow the formatting for the file, you can decide how to save and load information about each restaurant and review in the database. See the YelpDB.java class for more information about what exceptions you need to handle when saving or loading.

3.2 Review.java

- **Instance Data:** Every review must have a rating (whole number of stars from 1 to 5), a date on which the review was written that is inputted in the format "mm/dd/yy, and the restaurant it was written for. All instance data must be private; therefore, you should make the appropriate methods to allow other classes to get access to and manipulate this data.
- **Constructor:** When a review is created, all of the above attributes are required.
- `toString`: Additionally, the Review class should have a method that returns all of the instance data as a String. You can format this however you want as long as the user understands all of the information contained in the review.
- `equals`: Two reviews are considered equivalent if they have the same date, rating, and were written for the same restaurant.

3.3 Restaurant.java

- **Instance Data:** Every restaurant has a name, a type of cuisine that it serves, an array of reviews, a price range denoted as a number of dollar signs from 1 to 4, and an average rating. The average rating is a decimal number calculated based on the ratings from the reviews written about the restaurant.
- **Constructor:** When a restaurant is created, it must have a name, a cuisine, and a price range. Its list of reviews should be an empty array of length 10. Initially, a restaurants average rating is 0 because a restaurant doesnt have any reviews when it is first created.
- `addReview`: Users should be able to add a review to a restaurant once they have created the review. If the restaurants review array does not have enough space to hold the new review, you should extend the length of array to twice the size it was previously. The average rating

for the restaurant should be updated every time a new review is added. Remember after adding a review to recalculate the restaurant's new average rating and update it appropriately.

- `deleteReview`: Reviews for a restaurant should be able to be deleted. In order to do so, you must specify which review they want to delete. If a review is deleted, all other reviews that were added to the array after it need to be shifted down such that there are no null gaps in between restaurants. Every time a restaurant is deleted, the average rating should be updated. If a restaurant only has a single review and it is deleted, the restaurants average rating should return to the initial value of 0.

- `toString`: Whenever a restaurant is printed, its name, cuisine, average rating, and price range should all be printed on one line. The average rating should be printed to two decimal points. If there are any reviews written for that restaurant, they should be printed on subsequent lines.

- `equals`: Two restaurants are considered equivalent if they have the same name.

3.4 YelpDB.java

- **Instance Data:** The Yelp database stores an array with the restaurants that can be reviewed by users.

- **Constructor:** When creating the database, the restaurant array should be empty and of length 20.

- `load`: The load method should create a new Scanner that takes in a filename. The method should throw a FileNotFoundException if the filename is null or does not correspond to a valid file in the directory. If the file is found, it should first read in the number of restaurants and then the data for each restaurant and reviews. If at any point, an exception is thrown that corresponds to the format of the file or if a duplicate restaurant is found, this means that the file is corrupt. You must catch these exceptions and then have the method throw a CorruptDatabaseException with an appropriate error message for each situation.

- `save`: The save method should create a new PrintWriter that prints to a file called YelpDB.txt. It should first print the number of restaurants that are in the database, followed by the data about each restaurant and its reviews just like the provided text file you are loading from.

- `addToDatabase`: A restaurant should only be added to the database if it doesnt already exist, meaning there isnt a restaurant with the same name already in the database. If this does occur, a DuplicateRestaurantException should be thrown. If there is no space left in the array, the array should double in size to accommodate the new restaurant and then add the restaurant.

- `search`: Yelpers (users of your application) should be able to search for a restaurant by name in the database. This method should print out the restaurants name and other information such as cuisine and price range if it is found in the database. You should inform the user if their search does not produce any results because the restaurant does not exist in the database.

- `search`: Yelpers can search based on a cuisine and a price range. Like the other searches, this method prints out all of the restaurants that fit the criteria. If there are no restaurants that have the same cuisine and price range that the user searched for, then a message should be printed that tells the user that no restaurants matched their search.

- `search`: A search with no filters prints out the restaurant with the highest average rating. If there are multiple restaurants with the highest rating, they should all be printed out. A message should be printed if there are no restaurants in the database and thus there are no search results.

3.5 DuplicateRestaurantException.java and CorruptDatabaseException.java

You will be writing two of your own exceptions for this assignment. The DuplicateRestaurantException is an unchecked exception and occurs when someone tries to add a restaurant to the database that has the same name as a restaurant that is already in the database. The CorruptDatabaseException is a checked exception and occurs when trying to load information about the database from a text file and the file has been modified in such a way that the information cannot be read (i.e. the file contains duplicate restaurants, the file is missing information, the information in the file is not formatted correctly).

- **Instance Data:** The exceptions has a String message that gives details about why the exception was thrown.

- **Constructors:** The classes should have two constructors, one with no parameters, and a second that takes in a String error message. Remember to use constructor chaining so that you arent reusing any code.

- `getMessage`: The getMessage method should override Exception's getMessage and just return the error message associated with the exception.

If any of these exceptions are thrown, they must be properly handled so that your program does not crash.

4 Tips

1. You are allowed and encouraged to create helper methods to help keep your methods short and easy to understand.

5 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **100** points. Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar *.java
Audit done. Errors (potential points off):
0
```

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. The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **100** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

6 Javadocs

For this assignment you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online documentation for them is very detailed and helpful.
You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to have are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 13.31
 */
public class Dog {

   /**
    * Creates an awesome dog (NOT a dawg!)
    */
   public Dog(){
      ...
   }

   /**
    * This method takes in two ints and returns their sum
    * @param a first number
    * @param b second number
    * @return sum of a and b
    */
   public int add(int a, int b){
      ...
   }
}
```

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.

2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class start with a brief description of the role of the class in your program.

3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method

has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

6.1 Javadoc and Checkstyle

You can use the Checkstyle jar mentioned in the following section to test your javadocs for completeness. Simply add -j to the checkstyle command, like this:

```
$ java -jar checkstyle-6.2.2.jar -j *.java
Audit done. Errors (potential points off):
0
```

The javadoc cap for this assignment is **20** points.

7 Turn-in Procedure

Submit your `Driver.java, Restaurant.java, Review.java, and YelpDB.java` files on T-Square as an attachment. Do not submit any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double- check that you have submitted and not just saved a draft.

Keep in mind, unless you are told otherwise, non-compiling submissions on ALL homeworks will be an automatic 0. You may not be warned of this in the future, and so you should consider this as your one and only formal warning. This policy applies to (but is not limited to) the following:

1. Forgetting to submit a file
2. One file out of many not compiling, thus causing the entire project not to compile
3. One single missing semi-colon you accidentally removed when fixing Checkstyle stuff
4. Files that compile in an IDE but not in the command line
5. Typos
6. etc...

We know it's a little heavy-handed, but we do this to keep everyone on an even playing field and keep our grading process going smoothly. So please please make sure your code compiles before submitting. We'd hate to see all your hard work go to waste!

8 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.

2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.

3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.

4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps insure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is 8PM Thursday. Do not wait until the last minute!