CS601-01/CS601-02

Lab1: Implementing a data structure to store hotel data.

Part 1 due: Sep 11, 11:59pm

Part 2 due: Sep 15, 11:59pm

For this lab, you will design and implement a data structure to store information about hotels in San Francisco. The data is in JSON files that were obtained from the expedia API. The goal of this assignment is to familiarize you with data structures in Java, as well as with JSON files, File input/output and JUnit.

You are required to use the provided **starter code** for this lab**; do not modify signatures of any methods; do not modify JUnit tests**. Passing the provided tests is 80% of your grade for this assignment (see below for information on tests).
Click on this link https://classroom.github.com/a/sKfWqU-Y and follow instructions. Github Classroom will create a private repo for you with the starter code for lab 1. The starter code is the same for both parts of the lab. Clone this repository to create a local copy on your computer. See instructions for lab 0 on how to do it. Your solution should be submitted to your lab1 private repository on github. Please note that solutions submitted via email, or canvas, on in a different repo, will **not** get any credit.

I expect the structure of your submission to be exactly as specified. Not using exactly the directory structure I provide will result in deductions. You are not allowed to use any external libraries except for the **JSON-Simple library** mentioned in this specification. You'd also need **JUnit** to run the tests.

For **part 1 of lab 1** you need to write the following:

- Fill in code in class **Address** (a class that stores an address of a hotel: street address, city, state, longitude and latitude) - see the starter code.
- Fill in all the code in class **Hotel** (a class that stores general information about a single hotel: hotel id, hotel name and address). Hotel class should implement a `Comparable` interface so that hotels could be compared based on the name (alphabetically). You may use compareTo method of class String. toString method should return a string with the hotel info in the following format:

```
hotelName: hotelID
streetAddress
city, state
```

Example:

Travelodge Central San Francisco: 40682
1707 Market St
San Francisco, CA

- Fill in all the code in class **Review** (a class that stores information about a single hotel review and has the following instance variables: review id, hotel id, review title, review text, username, date, rating and whether the user can recommend it to others). This class should implement a `Comparable` interface so that reviews can be compared based on (a) the date (a review is "less" than another review, if it was submitted more **recently**), (b) if the dates are the same, based on user nicknames (alphabetically), (c) If the user nicknames are also the same, based on the review id. toString() method should return a string in the following format:

```
Review by username on date
Rating: rating
reviewTitle
textOfReview
```

Example:

Review by Ben on Tue Aug 16 18:38:29 PDT 2016
Rating: 2
Very bad experience
Awaken by noises from top floor at 5AM. Lots of mosquitos too.

Fill in the code in class **HotelData.** For **part 1** of the lab, you do **not** need to write loadReviews or printToFile, but **you need to implement all the other methods in HotelData**. This class should contain TreeMaps that will store all of the hotel information including hotel reviews submitted by users:

-The first TreeMap (I will refer to it as ***hotelsMap***) will map each hotel id to the corresponding Hotel. That will allow us to quickly get the reference to the Hotel object given the hotel Id.
-The second TreeMap (I will refer to is as ***reviewsMap***) should allow us to quickly find all the reviews for a particular hotel (given the hotel id). The key in the second TreeMap will be a **hotelId,** and the value is a **TreeSet of Review-s**.
Read comments above each method carefully - some require you to catch or throw exceptions.

addHotel method should create a Hotel object with the given parameters, and add its reference to the hotelsMap.

addReview method should create a Review object with the given parameters and add its reference to the appropriate place in reviewsMap.

loadHotelInfo method should take the name of the json file that has info about the hotels (like "hotels.json"), and use JSON Simple library to parse this json file, create Hotel objects and add Hotels to **hotelsMap** described above. Before you start working on this method, make sure you (a) understand the structure of **hotels.json** (for instance, for each hotel, the "f" attribute stores the name, while "ci" stores the city where the hotel is located etc.) and (b) add JSON Simple

library to your build path (see below) and learn how to parse json files using JSON Simple library. You can look at the following example (JsonSimpleReadExample.java) that demonstrates how to read from a simple json file using this library.

toString(int hotelId) method takes the id of the hotel and returns a string representing this hotel's info in a certain format (see comment above the method).

getHotels() returns the list of all hotelIds, sorted alphabetically.

For part 2 of the lab, you will fill in code for *all the methods in the starter code.*

Do **not** change the signatures of any methods in the starter code class (otherwise your code will fail the tests), but you may add additional methods.

Directory Structure

The lab1 project on github has the following subdirectories:

input folder contains hotel data in json files. **hotels.json** contains names, ids and addresses of many hotels in San Francisco and several nearby cities. It has many attributes for each hotel. You need to read this file using the JSON Simple library and save all the hotel info in your TreeMap in HoteData class. We are only interested in the hotel id, the hotel name, and the address of the hotel including latitude and longitude.

The **input** folder also has a **reviews** subfolder that contains json files with reviews (each of these json files contains reviews for one hotel). Some of these json files are in the **nested subfolders. For part 2 of lab1 (not for part 1!)** you need to **recursively** traverse a given directory to find all the review files in the directory and it's subfolders (that can also contain subfolders etc.) For each hotel review, you need to store the review id, the hotel id, the overall rating, the name of the user who posted the review, the title of the review, the date of the review and finally, the text of the review. Do **not** change anything in this directory.

Do **not** hardcode the names of files or folders in your code. Your code should work on any directory that has json files in the same format.

src directory contains the source code in **hotelapp** package.

lib directory contains third-party libraries - for this lab, you will be using **JSONSimple:** https://code.google.com/archive/p/json-simple/. The jar file for the library is already placed in the lib folder. The jar files for JUnit are also in the lib subfolder. You need to add all jar files in the **lib** folder to the build path of your project. (by editing File->Project Structure->Dependencies).

test directory contains test files that will be used to test your lab. Right click on it in IntelliJ, and select Mark Directory As -> Test Sources Root. You need to run **HotelDataTest**. For lab 1 part 2 you need to pass all of these tests. For lab 1 part 1, you need to pass only the following tests:

- HotelDataTest.testInvalidDate
- HotelDataTest.testSimpleAddHotel
- HotelDataTest.testSimpleAddHotelReview
- HotelDataTest.testThreeReviewsSameHotel
- HotelDataTest.testInvalidRating
- HotelDataTest.testGetHotels
- HotelDataTest.testAddInvalidReview

You need to pass all tests in HotelDataTest for lab1 **part 2.**

Passing the tests is 80% of the total grade for each part of the lab.

JSON Files

To learn about the JSON file format, refer to the w3schools.com tutorial:

https://www.w3schools.com/js/js_json_syntax.asp

I like the following website for viewing JSON files: http://codebeautify.org/jsonviewer

The Javadoc provided in the skeleton code provides specific descriptions of the expected behavior and output of each method. Fill in the code as needed, and add other methods as necessary. You do not need to change the Driver class, it's been provided to you. Proper encapsulation is important: make sure your instance variables are private, etc.

Notes:

- You may not make any modifications to the API or the test cases that will be provided to you.
- You are not allowed to use any other external libraries for this lab except for JSONSimple and JUnit.

Make sure your code compiles and runs before your final submission - projects that don't compile or don't run will get a 0. Passing the tests is 80% of the grade for each part of the lab.

This assignment is to be done individually. You may **not** work with other people, look at their code or ask them for help. You may not copy any part of this code from the web. You may ask the instructor, the TAs and CS tutors from the CS tutoring center for help.