CS601-01/CS601-02

Project 1: Implementing data structures to store and efficiently search hotel review data.

Due: Sep 15, 11:59pm

For this project, you will design and implement data structures that store hotel reviews and support efficient search. The dataset consists of JSON files obtained from the Expedia API. In this project, you will practice the following skills:

- Object-oriented design
- Using nested data structures from the Collections framework for efficient search
- Working with JSON files and GSON library
- Directory Traversal

You are required to use the provided starter code for this project. Do not change the directory structure, tests, or JSON files. You will, however, need to add your own packages and classes. Do not place all your code inside the `HotelReviewService` class, as that would be considered poor design. Your solution will be evaluated based on correctness of functionality, efficiency of search and data handling, and quality of design.

The `HotelReviewService` class serves as the main entry point for the project. Its main method accepts four command-line arguments that specify the paths to the hotels file and the reviews folder. The easiest option in IntelliJ is to run this class once, which will automatically create a run configuration; you can then open **Run > Edit Configurations** and add the arguments in the **Program Arguments** field.

```
-hotels pathTohotelFile -reviews pathToReviewsDirectory
```

The first parameter, `-hotels`, is a flag indicating that the command-line argument following it should be interpreted as the path to the JSON **file** containing general information about hotels (e.g., dataset/hotels/hotels.json).
The third parameter, `-reviews`, should be followed by a path to the directory containing JSON files with hotel reviews (e.g., dataset/reviews). You need to parse these JSON files and load the data into carefully designed data structures to enable efficient querying. Your code would normally be run with the following command line arguments:

```
-hotels dataset/hotels/hotels.json -reviews dataset/reviews
```
or
```
-reviews dataset/reviews -hotels dataset/hotels/hotels.json
```

The order of the `"-hotels"` and `"-reviews"` flags may be swapped. In later projects that build on top of this project, you will add more command line arguments. Your code should

handle multiple arguments in the **-key value** format. Arguments may appear in any order, but each **-key** (like "-hotels") is always followed by its **value** (like the file path). *Hint:* Consider storing these parameters and their values in a `HashMap` in a separate class whose responsibility is to parse command line arguments. You must handle the case where the user does not provide any arguments. In such a scenario, the program should print an error message and exit gracefully, rather than crashing.

Do not *hardcode* the names of files or folders in your source code, this will result in a deduction. Your solution must work with any directory containing JSON files in the same format.
You are responsible for thoroughly testing your solution. The tests provided by the instructor are not sufficient on their own. If you decide to add your own tests, write them in separate classes and do **not** modify the test cases provided by the instructor. You are **not** allowed to use any *external* libraries except for the **GSON** and **JUnit 5.**

Dataset: JSON Files

To learn about the JSON file format, refer to the W3Schools JSON Tutorial, and review the lecture notes and code examples on Json. To view the JSON file displayed nicely in IntelliJ, press Option + Command + L on a Mac and Ctrl + Alt + L on Windows.

The dataset folder of the project contains four subfolders: `hotelsTiny`, `reviewsTiny`, `hotels`, and `reviews`.

- The **hotels** subfolder contains a single JSON file with information about over a hundred hotels in the San Francisco Bay area. When parsing this file, you should extract and store only the following information about each hotel:
    - Name of the hotel
    - Hotel ID (stored as a **String**)
    - Latitude and longitude
    - Street address
    - City
- The **reviews** folder and its subfolders contain JSON files with hotel reviews. Each JSON file holds reviews for a specific hotel. You need to **recursively** traverse this directory to find all JSON files with reviews. For each review, you should store the following information:
    - Hotel ID (stored as a String)
    - Review ID (stored as a String)
    - Overall rating
    - Review title
    - Review Text

- User Nickname
- Date when the review was posted

You are required to parse JSON files using the GSON library; please refer to the examples shown in class.

The **hotelsTiny** and **reviewsTiny** subfolders each contain a single JSON file and can be used in the early stages of debugging and testing. They are also utilized in some of the instructor's tests.

Search

After loading the data from JSON files into your custom data structures (hint: they will be *maps*), your program should be able to **efficiently** process the following types of queries passed to the `processQuery` method in the `HotelReviewService` class:

- **findHotel hotelId**

  This query should display general information about the hotel with the given ID.

  Sample Query:

  ```
  findHotel 10323
  ```

  Returned string:

  ```
  hotelName = Hilton Garden Inn San Francisco/Oakland Bay Bridge
  hotelId = 10323
  latitude = 37.837773
  longitude = -122.298142
  address = 1800 Powell Street, Emeryville, CA, USA
  ```

  Use **System.lineSeparator** to add new line characters to the output.

  Do not hardcode "\n".

- **findReviews hotelId**

  This query should display all reviews for the hotel with the given hotel ID. Reviews should be **sorted by date (most recent first),** and if the dates are the same, by review id. Sorting should **not** be performed during query processing, it should be done while building the map with reviews.

  Sample Query (using hotelsTiny and reviewsTiny dataset):

  ```
  findReviews 25622
  ```

  Returned string:

  ```
  hotelId = 25622
  ```

```
reviewId = 57b717a44751ca0b791823b2
averageRating = 4.0
title = Room too small
reviewText = Great location, but the room is too small
userNickname = Xiaofeng
submissionDate = 2015-03-04
--------------------
hotelId = 25622
reviewId = 23d756a64672vr2gwegyhqw4
averageRating = 5.0
title = Great deal
reviewText = Loved the neighborhood, very lively
userNickname = Chris
submissionDate = 2014-09-05
--------------------
hotelId = 25622
reviewId = 92rlnlvnabuwbf256jsf20fj
averageRating = 3.0
title = Overpriced
reviewText = Good location, but very expensive
userNickname = Alicia
submissionDate = 2014-09-05
--------------------
```

- **findWord word**

The query should display all reviews that contain the specified word, where the word must be surrounded by whitespace or punctuation (not part of another word). Reviews must be sorted first by the number of times the word occurs in the review (higher counts first), then by the review date (most recent first), and if both frequency and date are the same, by reviewId in increasing order. The frequency count should be printed on a separate line before each review. This query must be implemented efficiently by using the map you built for each word when loading the data, and the reviews must be sorted and stored at that time as well; sorting should not be performed during query processing.

Sample query (using dataset/hotels and dataset/reviews):

findWord dog

Resulting string:

4

hotelId = 225819

reviewId = 574fa3a6021cc109df10d39c

averageRating = 3.0

title = Great concept poor execution.

reviewText = My normal go to when traveling to San Francisco is the Mark Hopkins. This time I wanted to try something new a bit more hip then old school. The staff is amazing, kind, knowledgable and overall good people. The room not so great, office chair super dirty as pictured below the vents were super dusty. The only reason I looked was because my allergies were so agitated it had to be something in the room. I travel with my hypoallergenic dog when we asked for a dog bed it came covered in dog hair. Mind you my dog doesn't shed and she refused to lay on that bed. Lastly No room service only breakfastI guess thats where it counts but was disappointing.

userNickname =

submissionDate = 2016-06-02

2

hotelId = 23838

reviewId = 56a1184aeb9755099412f0b7

averageRating = 5.0

title = Wonderful experience!

reviewText = Great stay! From our initial experience of arrival and dropping off our car to the valet (a very friendly gentlemen welcomed us and took care of the car quickly), to a pleasant gentlemen at the desk who answered all of our questions & checked us in quickly (we did have reservations for our 3 day stay), to other hotel staff who were all smiles and acknowledged us as we arrived and got our luggage to the elevator. Beautiful room, great views of the waterfront and arriving and departing flight at SFO.

Surprisingly quiet, giving the proximity to the airport. Clean, spacious, and well-appointed comfy room. Our experiences at Hangar Steak for both dinner and breakfast were wonderful. Expertly prepared steaks (tried the filet. Best I've EVER eaten). My wife had the free-range chicken with root veggies. She thought it was AMAZING. Local

produce was used, and it's SO fresh!! The breakfast buffet was also beyond awesome. Also, we appreciated having a dog relief station right out back for our service dog, complete with baggies. All staff members ROCK! Awesome folks, truly! Marriot just GETS the concept of hospitality. We'll be staying here again very soon!

userNickname = Shannon

submissionDate = 2016-01-21

2

hotelId = 200649

reviewId = 562548ec47aaef094a82737d

averageRating = 1.0

title = Don't book this hotel

reviewText = There were some dog wastes in front of our room door, and we did not know where were they come from. We asked front desk to help for cleaning up around 10 pm, but no one came to clean up until morning. The most awful thing is that they CHARGED us $75 for the dog wastes when we checked out. TERRIBLE!

userNickname =

submissionDate = 2015-10-19

1

hotelId = 4705

reviewId = 57a8c6e9e40c4a0b28fde3fa

averageRating = 5.0

title = Nice room, nice beds, nice staff

reviewText = Very nice front desk staff. Gave me upgrade room after I was placed in a room next to a dog whining.

userNickname = kirsten

submissionDate = 2016-08-08

1

hotelId = 5830

reviewId = 5793d5a53a803a0b288f2525

averageRating = 4.0

title = Great for me and my dog!

reviewText = Lovely, pet friendly hotel. The only complaint is the parking structure is the only place to park in the area, and they charge $25 for an overnight stay (I arrived at 11pm and left by 10am). I didn't ask what 2 nights would have cost...

Also, the tub/shower had nicks that weren't repaired, but being on the dog floor, I didn't mind.

Everyone was friendly and helpful.

userNickname = Susan

submissionDate = 2016-07-23

1

hotelId = 26945

reviewId = 5786d9852c54c70b28dbd121

averageRating = 3.0

title = La Quinta Inn near SFO

reviewText = Mid-level hotel in significant need of upgrading. Staff was friendly and helpful, and the free shuttle to/from SFO was very useful. However, room was tiny and cramped and the walls were paper thin. Had a barking dog in the next room and thank god it went to sleep late at night. Not the worst hotel experience, but not the best.

userNickname = Traveler

submissionDate = 2016-07-14

1

hotelId = 1606984

reviewId = 571980dbaf750809dae2c418

averageRating = 2.0

title = Up for a dance competition

reviewText = Hmnnn well the first room we didn't even get to the door as there was a loud barking dog next door , nope! Then the 2nd room only had one bed when we booked 2 queens for 4 of us. The third room was fine but really disappointed there wasn't even a cofee maker. The guy working was very nice. Trash outside when we first walked in was overflowing. Overall it was OK. Would pay a bit more for something a bit nicer with full breakfast and a coffee pot!

```
userNickname = dance mom
submissionDate = 2016-04-22
--------------------
1
hotelId = 7655
reviewId = 56afe64303bdc90980cd1c23
averageRating = 5.0
title = Great for family and including the pets
reviewText = Stay here yearly for the Cow Palace dog show and is very comfortable and
convenient. Close to nearby shopping and BJ's , Red Lobster and many more places to
eat. Breakfast is good and for the price a great deal.
userNickname =
submissionDate = 2016-02-01
--------------------
1
hotelId = 524164
reviewId = 55827028
averageRating = 4.0
title = Nice Hotel close to highway.
reviewText = Overall the hotel was very satisfactory. It was close to the airport (10
minutes away) and to At&t Park (20 minutes). It was a dog friendly hotel so we brought
our mutt. It has a nice walking trail by the water.
userNickname = RAMIL
submissionDate = 2013-08-06
--------------------
```

You must handle cases when the user enters an invalid command or an invalid hotel ID.

Efficiency

The search must be implemented as efficiently as possible. You should assume that the dataset is loaded once into your data structures, and that many search queries will be run after that; therefore, it is important to design your code so that searches are fast. You must create and store maps (for example, a variation of an inverted index) as instance variables while loading the data, rather than constructing them on the fly during each

search, since that would be inefficient and will result in a deduction. An inverted index is a data structure that maps words to the reviews that contain those words (and may also store their positions in the text). For example, with two hotel reviews, you could build an index where each word points to the reviews in which it occurs. Consider a simple example with two hotel reviews:

- **Review1**: "Great location, nice and cozy. Cannot find a better location, great."
- **Review2**: "1. Beds were uncomfortable. 2. Location is great."

Assuming we exclude numbers and commonly occurring words such as "a", "the", "is", "are", "were", "and", etc. (which you do *not* need to do for this project) and convert all words to lowercase, the inverted index might look like this:

- "great" -> [Review1, Review2]
- "nice" -> [Review1]
- "cozy" -> [Review1]
- "location" -> [Review1, Review2]
- "beds" -> [Review2]
- "uncomfortable" -> [Review2]

However, since your program must return reviews sorted by the frequency of the searched word within the review, you need to modify the inverted index to support this requirement.

Hint: (Look at this only after you have spent some time developing your own solution.)
One possible approach is to use a TreeSet to store "Review with Frequency" objects for each word. This allows you to efficiently sort and access reviews that contain a given word based on the frequency of the word, assuming you correctly implement Comparable or Comparator. For instance, for the key "great", you might get a TreeSet like this:

- "great" -> [(Review1, 2), (Review2, 1)]

In this setup, the TreeSet ensures that reviews are sorted by frequency (in descending order), by date (most recent first) if frequencies are the same, and by review ID if frequencies and dates are the same.
Please note that there are other ways to implement this functionality; we discussed a good alternative in class.

Style

Your code must follow style guidelines described in StyleGuidelines.pdf posted on our course website with this assignment. Please add JavaDoc-style comments (starting with /**) above each class and each public method. For example, if a method has parameters and a return value, your comment should briefly describe what the method does, explain

each parameter with @param, and describe the return value with @return, as shown in the example below:

```
/**
 * Calculates the average rating of a hotel based on all reviews.
 *
 * @param hotelId the id of the hotel whose reviews should be averaged
 *
 * @return the average rating as a double, or -1 if no reviews are found
 */
public double calculateAverageRating(String hotelId) {
    // implementation here
}
```

Policies / Use of AI Assistants

This assignment is to be done individually. You may not discuss implementation details of the project or work with other students, look at another student's code, or ask them for help. You may not copy code from other students, from the web, or from AI tools.

You may ask the instructor, the TA, and CS tutors from the CS tutoring center for help. In addition to regular office hours, the instructor has office hours by appointment.

Use of AI tools (ChatGPT, Copilot, etc.):

- You must first produce your own design (classes, methods, variables) and **commit/push it to Github** with a clear message: *"My own design without AI."*
- After that, you may use AI tools to improve your design or help debug specific methods, but not to write full method implementations for you. Always attempt to debug on your own first before consulting AI.
- If you use AI, you must describe how you used it in your **README** (what suggestions did it make to improve your original design? what prompts you tried, what you accepted or rejected, what you attempted on your own first).
- During your code review, you must be able to explain both your original design and how AI influenced your revised design.
- **Failure to document AI use clearly in your README will be treated as an academic integrity violation.**

Grading / Code reviews

Each student will be required to go through the code review for this project. Code reviews will be conducted via Zoom after the project deadline. The instructor/TAs will post the calendar link(s) that would allow each student to book a ~20-30 min slot either with the instructor or the TA. Depending on how the instructor and the TAs split the work, you may be asked to sign up for a slot with the specific person. You will need to have audio and

video on during the code review and be able to share the screen while opening your project 1 code in IntelliJ.

Please do **not** modify your code (locally or on GitHub) **after the project deadline and before your scheduled code review**. After your code review, you will be asked to make improvements based on feedback, and you must resubmit those changes as instructed. During the code review, we will evaluate the functionality, design, efficiency, and coding style of your code, time permitting. You will be asked to explain your code, answer questions about it, and make changes to it during the code review.

Project 1 grading rubric (100 pts)

1) Functionality (tests/correctness) — 45

Will grade the version submitted before the deadline.

Tests: 45 pts	
SearchByWordTest	16 pts
SearchHotelTest	10 pts
SearchReviewsTest	14 pts
SearchInvalidQueriesTest	5 pts

2) Design / OOP principles and Efficiency — 25

Efficient data structures (10 pts): 10 points will be awarded for implementing appropriate data structures (maps) that support efficient search. Full credit requires building and storing these structures once during data loading, rather than recreating or sorting them on the fly during queries.

Good OOP design (15 pts): Credit will be given for applying sound object-oriented design principles (e.g., separation of concerns, proper class structure, encapsulation, avoiding putting all logic in a single class).

Possible deductions:

- Not using command-line arguments correctly, including code crashing when no arguments are provided: –2 pts
- Hardcoding file paths: –2 pts
- Breaking encapsulation (e.g., non-private instance variables: –1 pt each; methods that expose or break encapsulation: –0.5 pt each)

3) Code review: understanding & ability to change code — 25
Individual live code review via Zoom. The instructor/TA will ask students about the code, give tasks to make small changes to their code; prompts pasted in chat. We also talk about improvements to the design.

Excellent	Explains and modifies code quickly & correctly	**25**
Adequate	Minor hints, mostly correct fixes	**18**
Partial	Struggles to modify some parts or explain logic	**10**
No ownership	Cannot explain or reproduce code	**0**

Integration with AI Policy
As part of the code review, you will also be asked to explain your **original design (before AI use)** and your **improved design (after consulting AI tools, if applicable)**. You must be able to justify why you accepted or rejected specific AI suggestions. If you cannot explain your design choices or your own code, this will be treated the same way as not being able to explain the functionality; your code review score will reflect it.

If **Code-review = 0**, the project is **0/100 unless** the student **rewrites the project** under the guidance of the instructor and passes another code review; if they pass, they get points for the project, but it is **capped at 60/100.**

Two projects with Code-review = 0 results in an **F for the course** and report to the Academic Integrity committee.

If **Code-review = Partial (10)**, the student must rewrite the parts they could not explain/modify, and pass another code review. Code review grade will remain at 10. Must have all subsequent code reviews with the instructor and encouraged to meet with the graduate advisor.

Three projects with Code-review = Partial will result in a C- cap for the course.

During and after the code review, we will also talk about improvements to the code design/efficiency, and the instructor/TA **will create Issues on Github for the pull request.**

Students will have a chance to fix design/efficiency issues within 3 days and get design/efficiency points back. Students can**not** get lost *functionality* points back.

4) Process & Style — 5

The code must follow the **style guidelines** posted by the instructor.

There are also specific requirements for how you must use Git/GitHub in this project. You will create a separate branch for your work, commit and push regularly (at least 8 commits are required, with descriptive commit messages), and open a Pull Request into main **before the project deadline**. Your Pull Request will then be reviewed by the instructor or TA during the code review. You may be asked to fix issues by making additional commits to your branch. Only after your Pull Request is approved may you merge it into main. Please see the detailed instructions on Canvas for the exact workflow to follow.

Issues with commit history: <= 2 commits total **or** ≥80% of changes in the final hour or most commits within a short window of ~30 minutes, then the code **review is capped at 10/25**. The second occurrence during the semester will result in **0/25**. The instructor may also request to look at the student's local history in IntelliJ.

Do not insert comments inside method bodies; your code should be clear and self-explanatory. Instead, include JavaDoc comments above each method.

Late policy

Each student has **two "late days"** to use at their discretion over the course of the semester. These may be applied to **any project**, just notify the instructor via email before the deadline. It is recommended that you save these late days for later in the semester, when your workload is likely to be heavier. Beyond these two late days, extensions will only be granted in cases of a (1) **verifiable medical emergency**, or (2) **official accommodations** approved through the SDS (Student Disability Services) office.