Homework 07 – Sushi Rolls

Authors: Ruchi, Harry, Xinyi, Eric
Topics: recursion, merge sort

Problem Description

Recently, a few TAs on the writing team went to a revolving sushi restaurant together and they suddenly thought of a brilliant theme for the upcoming homework: sushi! In this assignment, you will be tasked with writing methods to recursively solve some puzzles.

Solution Description

For this assignment, you will implement recursive methods in Restaurant.java. Some methods will require you to use the methods in the provided *RecursionUtils.java*, *SushiRoll.java*, and *Color.java* files.

Notes:

1. All methods should be accessible from everywhere and must not require an instance to be accessed through, unless specified otherwise.
2. You will be able to use provided methods from *RecursionUtils.java*, *SushiRoll.java*, and *Color.java*. Review the Javadocs for these classes and be careful not to modify these files.
3. Helper methods with appropriate visibility are optional.
4. Make sure to add Javadoc comments to your methods and classes!
5. **All methods in this assignment must use recursion or call recursive helper methods.**

Restaurant.java

This class will hold all the recursive methods you will be implementing.

Methods:

- `mergeSortRolls`
  - Takes in a `SushiRoll[]` and returns a `SushiRoll[]` containing the sushi rolls of the input array sorted in ascending lexicographical order by name.
    - Assume the input array will not be null nor contain null elements.
  - The implementation to sort the input array **must** be recursive merge sort.
  - This method should have a time complexity of $O(n \log n)$.
  - **No loops** are permitted to be written in this method nor in a helper method you create.
    - ***Hint***: Use the `copyOfRange` method in *RecursionUtils.java* to create subarrays.
  - When merging two arrays, you **must** use the `merge` method from *RecursionUtils.java*.
  - Example (each "{name}" represents a SushiRoll object; the color is omitted):
    - **Input**: `["Tobiko", "Avocado", "Unagi", "Dragon", "Maki"]`
    - **Output**: `["Avocado", "Dragon", "Maki", "Tobiko", "Unagi"]`

- `mergeOrders`
  - Takes in a `SushiRoll[][]` of orders, where you may assume each order contains sushi rolls sorted in ascending lexicographical order by name.
    - **Note**: An *order* is represented by a `SushiRoll[]`.
    - Assume the input array will not be null nor contain null elements, and that the array at each index will also not contain null elements.
  - Returns a `SushiRoll[]` containing all the sushi rolls from each order, sorted in ascending lexicographical order by name.
  - If the input array contains no orders, or if all orders are empty, return an empty array.
  - When merging two arrays, you **must** use the `merge` method from *RecursionUtils.java*.
  - **No loops** are permitted to be written in this method nor in a helper method you create.
    - ***Hint***: Write a helper method that takes in more parameters or use a provided helper method.
  - Example (each "{name}" represents a SushiRoll object; the color is omitted):
    - **Input**: `[["Avocado","Dragon"], ["Tobiko","Unagi"], ["Maki"]]`
    - **Output**: `["Avocado", "Dragon", "Maki", "Tobiko", "Unagi"]`
- `platesOfColor`
  - Takes in a `SushiRoll[]` that is sorted in ascending lexicographical order by name, and a `Color` representing the desired color of plate.
    - Assume the input array will not be null nor contain null elements.
  - Returns a `SushiRoll[]` containing only the sushi rolls on plates of the specified color.
  - If the input array does not contain any sushi rolls on plates of the desired color, return an empty array.
  - The returned array **must** be kept in ascending lexicographical order by name.
    - You may **not** call `mergeSortRolls` nor use any sorting algorithms.
  - **No loops** are permitted to be written in this method nor in a helper method you create.
    - ***Hint***: Write a helper method that takes in more parameters or use provided helper methods.
  - Example (each "{name} {color}" represents a SushiRoll object):
    - **Input**:
      ```
      ["Avocado BLUE", "Dragon BLUE", "Maki RED",
       "Rainbow GREEN", "Unagi RED"], "RED"
      ```
    - **Output**: `["Maki RED", "Unagi RED"]`
- `totalPrice`
  - Takes in a `SushiRoll[]` and returns a `double` representing the total order price.
    - Assume the input array will not be null nor contain null elements.
  - Remember that the price of each sushi roll depends on the color of the plate.
  - **No loops** are permitted to be written in this method nor in a helper method you create.
  - Example (each "{name} {color}" represents a SushiRoll object):
    - **Input**:
      ```
      ["Avocado BLUE", "Dragon BLUE", "Maki RED",
       "Rainbow GREEN", "Unagi RED"]
      ```
    - **Output**: `15.0`

- `flip`
  - Takes in a `SushiRoll[]` that is sorted in ascending lexicographical order by name.
    - Assume the input array will not be null nor contain null elements.
  - Returns nothing.
  - The input array should be modified "in-place" so that it becomes sorted in **descending** lexicographical order by name.
    - **Note**: "In place" means that the data in the original array should not ever be copied to another data structure (e.g., another array).
  - **No loops** are permitted to be used in this method nor in a helper method that is called by this method.
    - ***Hint***: Write a helper method that takes in more parameters.
  - Example (each "{name}" represents a SushiRoll object; the color is omitted):
    - **Input**: `["Avocado", "Dragon", "Maki", "Tobiko", "Unagi"]`
    - **Output**: `["Unagi", "Tobiko", "Maki", "Dragon", "Avocado"]`

Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 40 points.** This means there is a maximum point deduction of 40. If you don't have Checkstyle yet, download it from Canvas → Modules → Checkstyle Resources → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. 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 number of points we would take off (limited by the Checkstyle cap). In future assignments 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
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

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

- `Restaurant.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the assignment. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond 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. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g., 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.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

To prevent trivialization of the assignment, you may not import any classes for this assignment.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

The only code you may share are test cases written in a Driver class. If you choose to share your Driver class, they should be posted to the assignment discussion thread on the course discussion forum. We encourage you to write test cases and share them with your classmates, but we will not verify their correctness (i.e., use them at your own risk).

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 Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.