Homework 07 – Team Management: Recursion Mania

Authors: Kenneth, Nicolas, Harry, Vasilisa, Eric
Topics: Recursion and Merge Sort

Problem Description

Please make sure to read the document fully before starting!

You're now a member of a prestigious programming team that works on amazing projects. Now, your team leader hopes you, a strong programmer, will design a program to make team management easier! Therefore, you need to write methods that will help you manage your programming team using recursion. For some of them, you will need to use the given methods from *HWUtils.java*, *Group.java,* and *Member.java*, which will be provided.

Solution Description

For this assignment, you will create the following class: *Team.java.*

Notes:

1. *You will need to use the provided files: Group.java, HWUtils.java and Member.java. Review the Javadocs for those classes to understand their functionality.* ***Do not modify the provided files!***
2. *Whenever the term* ***sorted*** *is mentioned in this assignment, the order is defined as the way in which* ***compareTo()*** *is defined in the* ***Member*** *class.*
3. *All methods in this assignment must be recursive or call recursive helper methods.*
    a. ***INDICATE WHICH METHODS ARE RECURSIVE IN YOUR JAVADOC COMMENTS.***
    b. *If you decide to create additional helper methods, make sure that these helper methods are* ***inaccessible*** *outside the class.*
4. *Make sure to add all Javadoc comments to your methods.*
5. *Though we don't require you to write and submit a Driver, be sure to fully test your code locally!*

Team.java

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

Important: The primary objective of each method in this class must be performed recursively. This means that aside from `mergeSortMembers` and `mergeAllMembers`, unnecessarily merge sorting the array does not fulfill the requirement that each method must be written recursively or call a recursive helper method.

Variables:

- `members`
    - An array of Member instances that store all the members of your team. This variable should be **inaccessible** from other classes.

Constructor(s):

- A constructor that takes in an array of `Member` instances and makes a **deep copy** of the passed in array. Note that the passed in array may be unsorted.
    - If the passed in array is null or contains null elements, throw an `IllegalArgumentException` with an informative message.
    - The constructor does not need to be recursive.

Methods:

You will now implement methods to help you manage your team!

- `mergeSortMembers`
    - Sort the `members` array in non-decreasing order.
    - This method does not take any arguments nor return anything.
    - You **must implement the merge sort algorithm** so that this method runs in O(nlogn).
    - **Note:** You **must** use the `merge` method provided in HWUtils.java.
    - ***Hint:*** *You will **need** to write a recursive helper method to implement this sort.*
    - **Example:**
        - Original `members` array:

          [<Ken, BACKEND, 9>, <Erich, BACKEND, 8>, <Nathalie, FRONTEND, 7>,
          <Eman, FRONTEND, 6>, <Erich, FRONTEND, 8>, <Erich, BACKEND, 10>]
        - Sorted `members` array:

          [<Eman, FRONTEND, 6>, <Erich, FRONTEND, 8>, <Erich, BACKEND, 8>,
          <Erich, BACKEND, 10>, <Ken, BACKEND, 9>, <Nathalie, FRONTEND, 7>]

- `mergeAllMembers`
    - This method takes in a 2-D array of Member instances and does not return anything.
    - You may assume that the input array will not be null nor contain null elements. You may also assume that the 1-D arrays within the 2-D array do not contain null elements.
    - Merge all the 1-D arrays in the 2-D array with the 1-D `members` array.
        - **Note:** Do not assume that the original `members` array or 1-D arrays are already sorted.
        - ***Hint:*** *How can you reuse code to help you sort the unsorted arrays?*
    - **Note:** You **must** use the `merge` method provided in HWUtils.java.
    - ***Hint:*** *You may find it helpful to write a recursive helper method to help implement this method.*
    - **Example:**
        - Original `members` array:

          [<Pat, FRONTEND, 7>]
        - Input array:

          [
          [<Erich, BACKEND, 8>, <Ken, BACKEND, 9>],
          [<Eman, FRONTEND, 6>, <Klaus, FRONTEND, 4>],
          [<Harrison, ADMIN, 11>]
          ]

        - Sorted `members` array:
          [<Eman, FRONTEND, 6>, <Erich, BACKEND, 8>, <Harrison, ADMIN, 11>,
          <Ken, BACKEND, 9>, <Klaus, FRONTEND, 4>, <Pat, FRONTEND, 7>]

- `searchMember`
    - This method will take in a `Member` instance and returns the `Member` instance in the `members` array that has the same `name`, `subgroup`, and `hoursWorked`.
        - **Note:** If the member is found, you should return the member instance that is in the array instead of the passed in member instance.
    - If a member instance with the same `name`, `subgroup`, and `hoursWorked` is not found, throw a `NoSuchElementException` with an informative message.
    - If the `member` passed in is null, throw an `IllegalArgumentException` with an appropriate message that informs the user of the issue.
    - You may assume that the `members` array has **unique** member items, but you **may NOT assume that the list is already sorted**.
    - The sorting portion of this method should run in O(nlogn) time and should be performed recursively.
    - The searching portion of this method should run in O(log n) time and should also be performed recursively.
        - ***Hint: Which searching algorithm taught in lecture runs in O(log(n)) time?***
        - ***Hint:*** *You will* ***need*** *to write a recursive helper method to implement this search.*
    - The overall time complexity of this method should be O(nlogn + log n) == O(nlogn)
- `reverseMembers`
    - This method does not take any arguments and returns a **deep copy** of the `members` array in reversed sorted order (non-increasing order).
        - **Note:** The original array should **not** be modified.
        - **Note:** You may assume that the original array is already sorted.
    - The time complexity of this method should be O(n).
    - **Example:**
        - Original `members` array:

          [<Eman, Frontend, 6>, <Erich, Frontend, 8>, <Erich, Backend, 8>,
          <Erich, Backend, 10>, <Ken, Backend, 9>, <Nathalie, Frontend, 7>]

        - Output array:

          [<Nathalie, FRONTEND, 7>, <Ken, BACKEND, 9>, <Erich, BACKEND, 10>,
          <Erich, BACKEND, 8>, <Erich, FRONTEND, 8>, <Eman, FRONTEND, 6>]

- `selectLeaderboard`
    - This method will not take in anything and will return an `ArrayList` containing two Member instances.
    - You want to select leaders for the FRONTEND and the BACKEND subgroups so that the programming subgroups can be better managed.
    - You should **randomly** select two Member instances from the `members` array. That is, each Member instance of the FRONTEND subgroup should have an equal change of being selected among all Members of the FRONTEND subgroup, and each Member instance of the BACKEND subgroup should have an equal change of being selected

among all Members of the BACKEND subgroup. These will be added to the returned `ArrayList`.

- The returned `ArrayList` should satisfy the following properties:
  - `leaders.get(0)` returns a Member instance whose subgroup is `FRONTEND`.
  - `leaders.get(1)` returns a Member instance whose subgroup is `BACKEND`.
- This method **must** be recursive or call recursive helper methods and you **must** traverse the `members` array **only once**.
- **Hints:**
  - Create two ArrayLists, each to store members of a particular subgroup.
  - Write a recursive helper method that takes in those two subgroups and an additional parameter to help traverse through the `members` array **only once**.
  - After the recursive traversal through the `members` array, make the random selection of the leader for each subgroup.
- If no member can be select for both subgroups, throw a `NoSuchElementException` with the following message:

  `"Failed to select leaders for both subgroups."`
- If a member cannot be selected from only one subgroup, throw a `NoSuchElementException` with the following message:

  `"Failed to select a leader for the <subgroup> subgroup."`

- Getters and setters as necessary.

Checkstyle

You must run Checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of 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 points we would take off (limited by the checkstyle cap mentioned in the Rubric section). Note that you can ignore any Checkstyle errors in code we provide you, where applicable.

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.

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

Turn-In Procedure

Submission

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

- `Team.java`

DO NOT submit `Group.java`, `HWUtils.java`, and `Member.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 test 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 Piazza 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 homework. 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.

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.

Allowed Imports

- `java.util.ArrayList`
- `java.util.NoSuchElementException`
- `java.util.Random`

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. 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 Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW 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.

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.