Programming Exercise 06 – Generic Lineup

Authors: Nicolas, David, Eric
Topic: generics

Problem Description

In this assignment, you will create a lineup with a capacity of 4 elements. It should be able to store any kind of element so long as that kind of element is comparable (ensuring the lineup can be sorted).

Solution Description

You will need to complete and turn in one class: Lineup.java.

Notes:

1. All variables should be inaccessible from other classes and must require an instance to be accessed through, unless specified otherwise.
2. All methods should be accessible from everywhere and must require an instance to be accessed through, unless specified otherwise.
3. Use constructor and method chaining whenever possible! Ensure that your constructor chaining helps you reduce code reusage!
4. Reuse code when possible. Helper methods with appropriate visibility are optional.
5. Make sure to add all Javadoc comments to your methods and classes!

Lineup.java

This generic class represents a lineup of maximum size 4. It should **only** accept types that are comparable, and the type should only be comparable to objects of its same exact type (***hint:*** think about how the type parameter should be bounded to meet this requirement). Every lineup also maintains a sorted property, either ascending or descending.

Type Parameter:

- `T` – the type of the elements stored in this lineup

Variables:

- `ArrayList lineup` – the underlying data structure backing a lineup
  - This ArrayList should store the same type of elements as the elements in the lineup.
  - **Note:** This ArrayList must be limited to a *size* of 4 (i.e., it can store 4 or fewer elements at any given time). Do not add null elements to represent empty spaces.
- `boolean isAscending` – represents whether the lineup is in ascending order
  - If this value is false, the lineup must be in descending order.
  - **Note:** For either ordering, adjacent elements may be equal according to `compareTo`.

Constructors:

- A constructor that takes in four elements of type T: `first`, `second`, `third`, and `fourth`.
  - Initialize the `lineup` to have a capacity of 4. The `lineup` should then be filled from first to last (i.e., the first element of the backing ArrayList corresponds to `first`, the second to `second`, the third to `third`, and the last to `fourth`.
  - You may assume that the inputted elements are given in ascending order.
    - i.e., `first` <= `second` <= `third` <= `fourth`.
  - If **any** of the four input values are null, do not add any of the values to `lineup`.
  - Initialize the `isAscending` field to `true`.

Methods:

- `toString()`
  - This method should properly override the `toString` method from Object.
  - It should return a String in the following format: (single line, without curly braces, single space preceding/following each "->", no leading/trailing whitespace):

    ```
    "{first} -> {second} -> {third} -> {fourth}"
    ```

    - **Note:** Each set of curly braces represents an element's string representation.
    - You may assume that each element's string representation is on a single line with no leading/trailing whitespace.
  - If the `lineup` has less than 4 elements, represent the absence of a value with the string "null", which should only appear towards the end of the string representation.
- `add(T element)`
  - This method should add an element to the lineup to the proper position according to the value of `isAscending`.
    - For example, consider the following lineup of Integer objects in descending order with string representation "4 -> 2 -> 1 -> null". Calling `add(3)` should result in a lineup with string representation "4 -> 3 -> 2 -> 1".
  - Returns a `boolean` representing whether the element was added to the lineup.
  - If there is no space available, or the input element is null, do not add the element.
- `remove(T element)`
  - This method should remove the first occurrence of the input element according to that type's `equals` method.
  - Returns a `boolean` representing whether an element was removed from the lineup.
- `reverseLineup()`
  - This method should reverse the lineup's ordering.
    - For example, consider the following lineup of C objects in ascending order with string representation "1 -> 2 -> 3 -> null". Calling `reverseLineup()` should result in a lineup with string representation "3 -> 2 -> 1 -> null".
  - Returns nothing.
  - Update the value of the `isAscending` field.
    - **Note:** The value of this field should always be reversed, no matter the state of the lineup.
  - **Hint:** *The Collections API has a helpful method for this task*.

- `contains(T element)`
  - This method should return a `boolean` representing whether the lineup contains the specified element.
- `size()`
  - This method should return the size of the lineup.
    - The size of the lineup is the number of elements that are not null.
- `main()`

  You must follow the instructions for this main method as described below. Think of this main method as the program you will submit instead of a place to test your code. You may still share test cases in a Driver file with a different main method on the course discussion forum, but **no part of this main method should be shared or copied**.

  - Create two instances of Lineup.
  - Call each method at least once on either of these two instnaces.

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 45 points.** This means there is a maximum point deduction of 45. 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:

- `Lineup.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

- `java.util.ArrayList`
- `java.util.Collections`

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, it 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.