L13 Homework: Gym Equipment

Problem Description

Hello! Please make sure to read all parts of this document carefully.

In this assignment, you will be applying your knowledge of interfaces and Comparable to write one interface and some classes that can be used to simulate the management of equipment in a gym.

Solution Description

You will write one interface, Adjustable, and five classes: FreeWeight, Barbell, Dumbbell, WeightMachine, and Gym. Each will be written in its own file (named accordingly).

- You must use constructor chaining when it's applicable:
  - If a class has N constructors, it will use "this" constructor chaining N-1 times
  - A class will use "super" constructor chaining at most once. It may only be used in the constructor that has all the appropriate parameters. You will use it if and only if, in that constructor, the implicit "super" constructor chaining is not appropriate
- Visibility for required elements:
  - All classes, methods, and constructors must be public
  - All variables must be private
- You may add to aid you (if you want)
  - Methods, if they are private
  - Class constants (static final variables), if they are private and their type is a primitive type, a wrapper class, or String.
- You may not add any other classes or constructors

You will also be provided with an additional class, FreeWeightConcrete (in an appropriately named file). This class is used for autograding purposes, as we need a way to test the class that we cannot directly instantiate. **You need to familiarize yourself with this class, as it will help you understand autograder error messages related to the type which it is replacing**. This file is **error-free, and error messages related to it are actually errors about the related type**. You must not change it in any way, and you must submit it along with your other files in your submission.

Adjustable

- It represents objects that can be, in some way, adjusted in their weight. This is an interface. It will have:
    - `boolean adjustWeight(int)` abstract method. The parameter is the desired adjustment, and the return value is to indicate whether it could be applied successfully.

FreeWeight

- This class represents gym equipment that is a "free weight." You must not be able to create an instance of this class. `FreeWeight` should implement the `Comparable` interface with the proper type parameter such that it will allow comparisons between any kind of "free weight" equipment and another. It will have:
    - The following instance variables: `freeWeightID` (`String`), `category` (`int`), `weight` (`int`). None of them will be modified after
    - 3-arg constructor that takes the instance variables and sets them.
    - A `compareTo` method to allow comparing any `FreeWeight` object with another (even though `FreeWeight` is abstract, you can compare between any of the concrete classes that inherit from it by comparing all the `FreeWeight`). The ordering of `FreeWeight` established by this class is the following:
        - Order first by `category`, in ascending order
        - Order second by `weight`, in ascending order
        - If two `FreeWeight` have the same category and weight, they are equal (for `FreeWeight`'s `compareTo` – we will override it in the child classes so that each class can further order elements that are, for this definition, equal)
        - Note: by "order first, order second, etc." we mean that your `compareTo` should depend on the first element of the sequence that is mismatching, if any. The same applies to the other `compareTo` descriptions.
    - A `toString` method with the following format: "[`freeWeightID`]: [`weight`] lb." (without square brackets)
    - Getters for `freeWeightID` and `weight`.

Barbell

- This class represents barbells, a concrete class that extends from `FreeWeight` and implements `Adjustable`. It will have:
    - The following instance variables: `loadCapacity` (`int`), `loadedWeight` (`int`)
    - 3-arg constructor that takes the `freeWeightID`, `weight`, and `loadCapacity`. It will set the `loadedWeight` to `0` and the `category` to `1`.
    - A `compareTo` method to allow comparing the class to other `FreeWeight` (not just `Barbell` instances). This is an override to FreeWeight's compareTo, which further orders elements that the previous definition established as equal. The ordering of `FreeWeight` established by this class is the following:
        - Order first by `category`, in ascending order
            - If it has the same category, the instance is guaranteed to be a Barbell, and you can downcast the parameter for the comparisons that cannot be done against a general FreeWeight
        - Order second by `weight`, in ascending order
        - Order third by `loadCapacity`, in ascending order
        - Order fourth by `loadedWeight`, in descending order

        - Order fifth by `freeWeightID`, in ascending lexicographical order (`String`'s `compareTo`).
        - Tip: using the super implementation of `compareTo` to handle only cases that have a return value of `0` in the super implementation is recommended
    - A `boolean adjustWeight(int)` method to adjust the `loadedWeight` by adding the parameter. If the new value is less than zero, or the new value exceeds `loadCapacity`, it should return `false` and not modify the instance variable. Otherwise, it should return `true` after updating the instance variable.
    - A `toString method` with the following format:
        - "[`freeWeightID`]: [`weight`] lb. barbell" (without square brackets)
        - You must use the super implementation of the toString method.
    - Getters for the instance variables.

<u>Dumbbell</u>

- This class represents dumbbells, a concrete class that extends from `FreeWeight`. It will have:
    - The following instance variables: `gripType` (`String`).
    - 3-arg constructor that takes the `freeWeightID`, `weight`, and `gripType`. It will set the `category` to 2.
    - A `compareTo method` to allow comparing the class to other `FreeWeight` (not just Dumbbell instances). This is an override to FreeWeight's compareTo, which further orders elements that the previous definition established as equal. The ordering of `FreeWeight` established by this class is the following:
        - Order first by `category`, in ascending order
            - If it has the same category, the instance is guaranteed to be a Dumbbell, and you can downcast the parameter for the comparisons that cannot be done against a general FreeWeight
        - Order second by `weight`, in ascending order
        - Order third by gripType, in ascending lexicographical order
        - Order fourth by `freeWeightID`, in ascending lexicographical order.
        - Tip: using the super implementation of `compareTo` to handle only cases that have a return value of `0` in the super implementation is recommended
    - A `toString method` with the following format:
        - "[`freeWeightID`]: [`weight`] lb. dumbbell with [gripType] grip" (without square brackets)
        - You must use the super implementation of the toString method.
    - A getter for the instance variable.

<u>WeightMachine</u>

- It represents weight machines. This is a concrete class that implements Adjustable and Comparable. Comparable should allow weight machines to be compared only against other weight machines. It will have:
    - The following instance variables: `weightMachineID` (`String`), `weightIncrement` (`int`), `maxWeight` (`int`), `currentWeight` (`int`)
    - A constructor that takes in `weightMachineID`, `weightIncrement`, and `maxWeight` and sets those. `currentWeight` should be set to `0`.
    - A `compareTo method` to allow comparing the class to other `WeightMachine`. The ordering of `WeightMachine` established is the following:

        - Order first by `maxWeight`, in ascending order
        - Order second by `currentWeight`, in descending order
        - Order third by `weightMachineID`, in ascending lexicographical order
    - A `boolean adjustWeight(int)` method to adjust the currentWeight by adding the parameter. If the new value is less than zero, or the new value exceeds maxWeight, it should return false and not modify the instance variable. The same should happen if the parameter is not a multiple of `weightIncrement`. Otherwise, it should return true after updating the instance variable.
    - A `toString method` with the following format:
        - "[`weightMachineID`]: [`currentWeight`] lb. weight machine" (without square brackets)
    - Getters for the instance variables.

Gym

- It represents Gyms, which are described by the equipment they have. This is a concrete class. It will have:
    - The following instance variables: `freeWeights` (`FreeWeight[]`), `weightMachines` (`WeightMachine[]`).
        - Note that it's possible to have an array with a component type that is an abstract class. We will explore this more in L15, but the idea is that in `freeWeights` one can assign a Barbell or a Dumbbell to any of its elements.
    - A 2-arg constructor that receives the two variables (in the same order). It will assign the received array to the instances and will also sort the array.
        - Don't write your own sorting implementation in this HW. Stick to Java's built-in sorting.
    - A 0-arg constructor that creates a gym with no equipment (empty arrays). Make sure to instantiate new arrays of length 0 instead of keeping a default copy.
    - A `void browseGymEquipment()` method. It will print the String representation of all the equipment in the gym in order, each in its own line. The "free weight" equipment will go before the weight machines
    - A `void addEquipment(FreeWeight)` method that adds a new `FreeWeight` (which could be a `Barbell` or a `Dumbbell`) to `freeWeights`. This method should ensure that the new array is in the natural ordering of the elements. Don't modify the old array.
        - The easiest way to implement this is in O(n log n), and that is a valid solution. We invite you to think if there are any implementations with lower runtime complexity, and if so, implement them (but we recommend focusing on the basic implementation - you can do this after being done with your submissions).
    - A `void addEquipment(WeightMachine)` method that adds a new `WeightMachine` to `weightMachines`. This is an overload of the previous `addEquipment` method, and like it, it should ensure the new array is sorted, and the same note regarding the runtime complexity applies.
    - A `FreeWeight getFreeWeight(String)` method that returns a `FreeWeight` (could be a `Barbell` or a `Dumbbell`) if the parameter matches any of the IDs of the `FreeWeight` in the gym, and `null` otherwise.
    - A `WeightMachine getWeightMachine(String)` method that behaves the same as the one above but for weight machines.
    - An `int getEquipmentCount()` method that returns the equipment count (sum of the number of "free weights" and weight machines)

Testing Your Solution

You will notice that none of the classes have a `main` method. That is because not all classes have to be Java programs, and in this case, none are. To help test your implementations, you can create more classes in which you can create instances of your concrete classes, invoke methods to have them interact with each other, and print to console information that helps verify the correctness of your solution.

Rubric

[40] Type Review (it is very important for your submission that these tests pass. All of these are visible in every submission)

- [10] All types compile, all type headers are correct, all static variables are correct
- [5] Check Adjustable
- [5x5] Check constructors (1), methods (3), variables (1) presence in each class

[12x5] Implementations in each of the student-written classes

We reserve the right to adjust the rubric, but this is typically only done for correcting mistakes.

Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import **java.util.Arrays**.

Feature Restriction

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

- **var** (the reserved keyword)

Checkstyle

The Checkstyle deduction limit for this assignment is 20 points, counting Javadoc errors. Refer to the course guide for the use of Checkstyle and review the autograder Checkstyle report on your submissions.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, we require that you place a collaboration statement as a comment at the top of at least one .java file that you submit. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

If the autograder encounters issues processing your collaboration statement, begin it with "Collaboration Statement: ".

Allowed Collaboration

When completing homeworks for CS 1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- approved:
    - "Hey, I'm really confused on how we are supposed to implement this part of the homework.
    - What strategies/resources did you use to solve it?"
- disapproved:
    - "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Upload the files listed below to the corresponding assignment on Gradescope:

- Adjustable.java
- Barbell.java

- Dumbbell.java
- FreeWeight.java
- FreeWeightConcrete.java (provided but must also be submitted)
- Gym.java
- WeightMachine.java

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds 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.

Important Notes

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit .class files or .jar 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 Piazza for a note containing all official clarifications