Homework 5 - AthleteList

Introduction

In this assignment, you'll practice

- implementing interfaces,
- using generics,
- using exceptions,
- and implementing a simple data structure.

Problem Description

The 2018 Winter Olympics have come and gone! Unsatisfied with their 4th place finish, Team USA has hired you, a young, ambitious Georgia Tech CS grad, to revamp their system for managing their athletes! You've decided to utilize your vast knowledge of Java collections and use a custom ArrayList implementation.

Background Information

Up to this point in the course, we've used arrays as our primary data structure. It's time to learn about another one, ArrayLists!

Our main gripe with arrays is that they are fixed size. ArrayLists solve that problem – they resize as elements are added to them. Behind the scenes, an ArrayList stores the elements added to it in a backing array. You'll similarly use an array in your implementation of AthleteList as your backing data structure. If the array is at capacity and an element needs to be added, then the contents of the backing array are copied into a new, larger array. This new array becomes the backing array.

Something else to be noted in our discussion of ArrayLists is the distinction we make between **capacity** and **size**. Capacity is the maximum number of elements the backing array can hold. This is an implementation detail that's hidden from the user. Size is the number of elements that have been added to an ArrayList, and is known to the user through a `size()` method. Size is typically tracked by the ArrayList. You'll encounter both of these terms in your implementation.

Solution Description

You will be provided the following files. **You may only modify `train()` in Athlete!** The file names are clickable, and take you to the Java files.

- `AthleteListInterface` (/spring2018/hw5/AthleteListInterface.java) – your `AthleteList` class will need to implement this interface. All details regarding the abstract methods you will override has been provided in the Javadocs, and have been repeated below for convenience. `AthleteListInterface` contains the following abstract methods:
    - `void add(Athlete t)` – This method adds an `Athlete` to the `AthleteList` . If the `AthleteList` is full, you'll need to resize it to twice the current capacity prior to adding the element. If the element passed in is null, throw an `IllegalArgumentException` .

  - `Athlete remove(int index)` – Removes the athlete at the passed-in index and returns it. You should shift every element to the right of the removed element one spot to the left, so that there are no gaps in the middle of the `AthleteList` . See the interface for an example. If `index` is less than zero or greater than or equal to the number of elements remaining in the `AthleteList` , you should throw an `IndexOutOfBoundsException` .
  - `Athlete get(int index)` – Returns the element at the specified index. If `index` is less than zero or greater than or equal to the number of elements remaining in the `AthleteList` , you should throw an `IndexOutOfBoundsException` .
  - `void clear()` – Empties the `AthleteList` of all elements and resets it back to its original capacity.
  - `int size()` – Returns the number of elements currently in the `AthleteList` .
  - `boolean isEmpty()` – Returns whether the `AthleteList` is empty or not.
  - `void train(int index)` – Calls the `train()` method of the `Athlete` at the passed-in index. More details on that below. As with `get` and `remove` , if `index` is out of bounds, throw an `IndexOutOfBoundsException` .
  - `void rest(int index)` – Calls the `rest()` method of the `Athlete` at the passed-in index. If `index` is out of bounds, throw an `IndexOutOfBoundsException` .
  - `Athlete[] asArray()` – This method is for our grading (and your testing!) purposes. Return an array with the same elements as the backing array, but with `size()` elements, that is, no `null` elements from right-padding a backing array that is bigger than the number of elements currently stored in the `AthleteList` .

- `Athlete` (/spring2018/hw5/Athlete.java) – A class used to model an athlete. An `Athlete` has the following instance variables and instance methods:
  - `name` – A String representing the name of the `Athlete` .
  - `energy` – An int representing the amount of energy the `Athlete` has left. This depletes as the `Athlete` trains.
  - `strength` – An int representing the strength of the `Athlete` . This increases as the `Athlete` trains.
  - `public Athlete(String name, int energy, int strength)` – Public constructor with parameters for each of the instance variables.
  - `public Athlete(String name)` – Public constructor that only takes a `String` for the name and sets energy to 15 and strength to 5.
  - Getters for each of the instance variables.
  - `public void train()` – Decreases energy by the current value of strength, and then increases strength by one. **You will modify this method as such:** If energy is less than strength, throw an `OvertrainedAthleteException` , which you will write.
  - `public void rest()` – Increases energy by 5.

You will need to write the following classes:

- `AthleteList` – your Athlete management solution. This class will implement `AthleteListInterface` , and should be of any generic type **which is an `Athlete` or a subclass of `Athlete` .** Use an array as your backing data structure. Write one no-arg constructor, which initializes the backing array. When you initialize an `AthleteList` , use the `INITIAL_CAPACITY` constant defined in `AthleteListInterface` . **I recommend you keep track of the number of elements currently in the AthleteList.**

- `OvertrainedAthleteException` – A custom **unchecked** exception which will be thrown when a tired `Athlete` 's `train()` is called. The constructor for `OvertrainedAthleteException` should accept a String as a parameter and invoke the constructor in the super class that also accepts a String as a parameter. This parameter is the message that will be displayed when an `OvertrainedAthleteException` is thrown. You will also need to modify the `train()` method in `Athlete` as described above.

Solution Constraints

- As always, you shouldn't use anything that will oversimplify the assignment (like `ArrayList`).
- You don't need to import anything for this assignment. If you find yourself needing to import something in the course of this assignment, rethink how you're approaching the problem.
- **You may not use the Arrays class nor System.copyof for this homework!** Doing so will be considered trivializing the assignment and you will lose most, if not all, points.

Tips and Considerations

- Read the directions and the documentation in `AthleteListInterface` carefully! There's a lot to digest in this homework, make you understand what's expected of each method.
- Writing a Main class with a main method to test your code would be very useful.
- `asArray()` will be very useful in debugging!

Grading

- [10] `AthleteList` constructor
- [10] `add()` correctly adds an element to `AthleteList`
- [5] `add()` correctly resizes the backing array when necessary
- [10] `remove()` correctly removes an element from `AthleteList`
- [5] `remove()` correctly shifts elements after removal
- [10] `get()`
- [10] `clear()`
- [5] `size()`
- [5] `isEmpty()`
- [5] `AthleteList` 's `train()`
- [5] `AthleteList` 's `rest()`
- [5] `asArray()`
- [10] `OvertrainedAthleteException`
- [5] `Athlete` 's `train()` correctly throws `OvertrainedAthleteException`

Checkstyle

For each of your homwork assignments we will run checkstyle and deduct one point for every checkstyle error.

For this homework **there is no checkstyle cap**, meaning you can lose **all** points on this assignment due to style errors.

- If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can!

- You can run checkstyle on your code by using the jar file found on the course website that includes xml configuration file specifying our checks. To check the style of your coed run `java -jar checkstyle-6.2.2.jar *.java`.
- To check your Javadocs run `java -jar checkstyle-6.2.2.jar -j *.java`.
- Note that the command for checking code and the command for checking Javadocs are different. You will have to run both commands to fully test for style errors.
- Javadoc errors are the same as checkstyle errors, as in each one is worth a single point and they are counted towards the checkstyle cap.
- **You will be responsible for running checkstyle on *ALL* of your code.**
- Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the customization tips (http://cs1331.gatech.edu/customization-tips.html) page for more information.

Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework specification 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:

OKAY: “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?”

BY NO MEANS OKAY: “Hey… the homework is due in like 20 minutes… Can I see your code? I *promise* won’t copy it directly!”

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

- Submit your modified `Athlete.java` and your `AthleteList.java` and `OvertrainedAthleteException.java` files to Canvas.

Verify the Success of Your Submission to Canvas

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that your program runs with no syntax or runtime errors. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

- After submitting the files to Canvas, return to the Assignment menu option and this homework. It should show the submitted files.
- Download copies of your submitted files from the Canvas Assignment page placing them in a new folder.

- Re-run and test the files you downloaded from Canvas to make sure it’s what you expect.
- This procedure helps guard against a few things:
  - It helps insure that you turn in the correct files.
  - It helps you realize if you omit a file or files. Missing files will not be given any credit, and **non-compiling/non-running homework solutions will receive few to zero points**. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. Do not wait until the last minute! (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
  - Helps find syntax errors or runtime errors that you may have added after you last tested your code.