Assignment 2: Abstract Classes and Programming with Lists

Due: Tuesday, Nov 8 at 11:00pm via InstructAssist

Want help finding a homework partner? Fill in this form. We'll try to pair people up through Sunday afternoon. After that, you will have to find your own partner if you want one.

See Setting up Partners for instructions on configuring your homework partner in InstructAssist.

Note to those with prior Java experience: One goal of 2102 is to help everyone learn when different iteration constructs (`for`, `while`, etc) are needed for a particular problem. Style grading will check whether you are using appropriate constructs. This week, we will cover the per-element style `for` loop, not the `for` loop that uses a variable to index into elements. For full points, do not use index-based `for` loops on this assignment. Use a per-element style loop instead.

1 Problem Description and Context

This week, we extend your initial classes for athletic competitions so you can practice programming with lists in Java. We also add a new kind of event so you can practice working with abstract classes.

We will have covered the material for questions 1 and 2 by the end of Tuesday. We will do lists by Friday.

1. Add a class `MarathonResult` to your project. Like a `CyclingResult`, a `MarathonResult` has a finishing time and position. However, the `pointsEarned` for a marathon is simply the finishing time (position is irrelevant). `MarathonResult` should also be an `IEvent`.

   Create abstract classes as needed to share appropriate details between `CyclingResult` and `MarathonResult`. You may use whatever names you wish for these classes.

   Do NOT add a `MarathonResult` field to your `Athlete` class. We are simply creating the ability to have a marathon, but the athletes in this assignment will not participate in one. Larger projects often have classes that don't get used in all scenarios.

2. Add a name field (type String) to the `Athlete` class

3. Create a class `Competition`, which contains a `LinkedList` of `Athletes`. It should also contain an integer indicating the (non-negative) number of `BiathlonRounds` to be used in this competition.

4. Modify your `BiathlonResult` class to now contain a list of `BiathlonRounds` (rather than a fixed three rounds). We assume the rounds are in order (the first round went first, the second went second, etc).

5. Modify `pointsEarned` in `BiathlonResult` to now total the points across all rounds in the list.

6. Modify `bestRound` in the `BiathlonResult` class to return best round in the list.

   We will check your test cases for this method against several broken solutions to this problem. Pay particular attention to creating a thorough set of tests for this method.

7. Within a single competition, all of the athletes should have completed the same number of rounds in the biathlon. Write a method in the `Competition` class called `BiathlonDNF` (for "did not finish"), which produces a `LinkedList` of the `Athlete`s in the competition whose list of `BiathlonRound`s is less than the number of rounds stored in the `Competition` class.

   The Athletes should in the same order in the returned list as they were in the list within the `Competition`.

8. Write a method in the `Competition` class called `scoreForAthlete`, which takes the name of an athlete and returns the `totalScore` that the athlete earned in the competition. You may assume that no two athletes have the same name. You may also assume that the athlete name given is in the competition (we'll talk about how to handle error cases later in the course).

9. Write a method in the `Competition` class called `countCyclingImproved`, which takes another `Competition` as input and returns the number (integer) of athletes whose `pointsEarned` in cycling was lower in "this" competition than in the given competition. You may assume that both competitions have the same athletes, but the athletes may appear in different orders within the athlete lists in both competitions.

   We will check your test cases for this method against several broken solutions to this problem. Pay particular attention to creating a thorough set of tests for this method.

10. Look back on your solutions to `scoreForAthlete` and `countCyclingImproved`. In hindsight, do you see any helper methods that you should have written that could have been shared over those two problems, or are you happy with how you organized the code?

    Put your answer (a couple of sentences) in a comment at the bottom of your `Competition` class. You do not need to write any code or rewrite either method for this question. We're just asking you to reflect on your code and tell us what changes you might have made were you to do this pair of problems again.

2 Support Files

Here are three files that may be helpful. You can download these directly into your project directory for this assignment.

- a basic Main.java file, which Java needs to compile and run your program.
- a skeletal Examples.java file showing you the imports that you need to include JUnit and the shape of a test case. Remove the sample test case as you add your

own.

- a checking stub file that attempts to create objects from the expected classes and call the expected methods within those classes. *Including this file when you compile will check that you have the class and method names that our grading tools expect*, which saves you from losing points.

  If you get a compilation error involving this file on a class or method that you defined, **do not edit this file – edit your files instead!**. As you are working, you may wish to comment out sections of the file that check methods you haven't written yet (that's fine). The final work you turn in should, however, compile against the entire contents of this file.

  You are welcome to leave this file in the directory when you submit your work.

3 What to Turn In

Submit a single zip file (not tar, rar, 7zip, etc) containing all of your `.java` files that contain your classes, interfaces, and examples for this assignment. Do not submit the `.class` files.

Make sure your tests are in files with `Examples` or `Tests` as part of the filename (a single `Examples.java` suffices). Your may put all of your other classes and interfaces either into a single file or into separate ones (as you prefer). If your have separate `src` and `test` subdirectories, you may retain that structure in your zip file.

Make sure that your test files include **ONLY** calls to methods that are listed in the assignment handout. If your tests include calls to other/helper methods or references to fields, your tests will fail to compile against our solution and we won't be able to grade your work.

4 Grading and Expectations

Follow the General Formatting Guidelines on assignments. Here is an example of a well-formatted version of the animals programs.

This assignment will earn points towards the following course themes:

- Java Programming (whether you used the constructs properly)
- Testing (the correctness and thoroughness of your tests)
- Program Design (appropriate use of classes and interfaces, documentation, clean code)

Here are some details on what we will look for in grading this assignment:

- Did you create classes with the fields required in the problem?
- Do your methods produce the answers expected based on the problem statements?
- Is each method (in both classes and interfaces) documented with a brief purpose statement?

- Is your code neatly indented and presented in a clean, readable manner?
- Are your test cases correct relative to the problem statement?
- Did you include at least one test for each required method?
- Are your test cases thorough for the methods marked for thoroughness checking?

5 Updates and Clarifications

As clarifying questions arise on the forum, we will post answers to them here.