Assignment 1: Classes and Methods on Athletes

Due: Tuesday, Nov 1 at 11:00pm via InstructAssist

Want help finding a homework partner? Fill in this form. We'll pair students up daily until 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.

Be sure to follow the Expectations for Preparing Homework as you work on this assignment.

1 Problem Description and Context

Inspired by the recent Olympics, a local organization plans to create a multi-sport athletic competition. For now, the competition will require every athlete to compete in two events: a cycling race and a short biathlon (of multiple rounds, each with trail running and target shooting). Your job is to create the initial software to record athletes' results in these events, and to provide methods that do some basic performance comparisons between individual athletes.

Specifically, your tasks are as follows:

1. Create an `Athlete` class with two fields: one holds a `BiathlonResult` object and the other holds a `CyclingResult` object. [**Note:** updated 11/1 to get these fields in the same order as in the testing stub file.]

2. The `CyclingResult` class stores the time (in seconds) at which the athlete crossed the finish line and the athlete's position in the finishing order (1 for first, 2 for second, etc). Use a double for the time and an int for the position.

3. The `BiathlonResult` class, which holds three separate `BiathlonRound` objects, one for each of three rounds (e.g., `round1`, `round2`, and `round3`). Each `BiathlonRound` stores the number of targets hit (an integer from 0-5, inclusive) and the time (in seconds) of the athlete's run around the track in that round. Use a double for the time. Assume that only valid numbers of target hits will be created; you do not need to do any error checking at this point in the course.

4. Each of `CyclingResult` and `BiathlonResult` should implement an `IEvent` interface. This interface should require a method called `pointsEarned` which takes no additional inputs and returns a double representating an athlete's score on that event.

    - For most athletes, the points earned for a `CyclingResult` is the finishing time. However, the points for the first place finisher takes 10 seconds off the finishing time. Similarly, the second-place finisher gets 7 points off, and the third-place finisher gets 3 points off. All other finishers have their raw finishing time as the points.

    - The points earned for a `BiathlonResult` is the sum of the points for each round. For an individual round, the points is the finishing time (seconds) plus a penalty of 1 minute for each target missed (out of 5).

5. In the `BiathlonResult` class, include a method called `bestRound`, which takes no inputs and returns whichever of the three `BiathlonRounds` that earned the best (lowest) score. In case of a tie, return the earliest round (i.e., round1 over round2, round2 over round3, etc).

6. In the `Athlete` class, include a `totalScore` method that takes no inputs and sums the points across the Cycling and Biathlon results

7. In the `Athlete` class, include a `hasBeaten` method that takes another `Athlete` as input and returns a boolean indicating whether the athlete has a lower total score than the given (input) Athlete.

8. In the `Athlete` class, write TWO versions of a `betterCyclist` method (call them `betterCyclist1` and `betterCyclist2`). Each should take another `Athlete` as input, and return whichever of the two athletes has the lower score on the cycling event alone. Assume there are no ties (meaning we won't test for ties and neither should you, as the behavior in event of a tie is not specified).

   Why two versions? Because there are several ways of dividing the comparison of cycling scores across the classes. To help you understand this, we are asking you write two versions of this method. You won't lose points this week for the style you picked (that comes later); we will just grade you on whether the methods yield the right answer. We will, however, go over the alternatives after the assignment comes in (some are clearly better than others). Style will count towards grades later in the term.

   Added 10/27: Think about different helpers you might create when solving this problem, then think about which classes those helpers might go in. That's the kind of "difference" we are aiming for here. We are not looking for differences such as using "if" in one solution vs "and/or" in the other solution. For those with prior Java experience – don't just change control operators (if vs switch, for example). We are looking for different ways to break down the problem conceptually.

9. Create a test suite for your work. Put all of your tests and test data in a class called `Examples`.

   Note on testing Doubles: When you want to use `assertEquals` to compare doubles, you include a third argument which is the allowable difference between the two values for them to still be considered equal. For example:

   ```
   assertEquals(5.0, 4.995, .01)
   ```

   returns true. Doubles can be imprecise due to the way they are represented within the computer, hence the need for this third argument.

   Note on Testing `bestRound`: A subtlety to JUnit (that we will talk about only next week) affects how you write tests that compare objects. When writing tests for `bestRound`, name the objects for your rounds and use the names in the `assertEquals` test, as follows:

   ```
   public class Examples {
     BiathlonRound quickRound = new BiathlonRound(...);
     BiathlonResult goodResult = new BiathlonResult(... quickRound
   ```

```
...);
    ...
    @Test
    public void testQuickBest() {
      assertEquals(quickRound, goodResult.bestRound());
    }
  }
```

You should **NOT** make a new `BiathlonRound` for the expected answer in the `assertEquals`. Such a test would fail, even if the two rounds had the same contents (again, for reasons we will explain in detail in week 2).

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. 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 all of your tests are in separate files from your code, as explained in the Expectations for Preparing Homework.

4 Grading and Expectations

Follow the Expectations for Preparing Homework (which includes formatting and organizing your files). Here is an example of a well-formatted version of the animals programs, as well as an example of a well-documented examples class with tests.

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?
- Are your test cases thorough, covering different situations (based on input data) and exercising all of your code?

5 Updates and Clarifications

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