Homework 4

1 Introduction

This assignment will cover classes, encapsulation and constructors.

2 Problem Description

It's Olympics season! There is a lot going on, so we need to keep track of lots of athletes, sports, and medals. Use your Java skills to do that so we can decide the winners of each event!

3 Solution Description

For this homework you will need to create the following classes:

`Sport`
Create an enum called `Sport` to represent the different event types at the Olympics. The `Sport` enum should have the following constants: `BOBSLEIGH`, `CROSS_COUNTRY_SKIING`, `CURLING`, `FIGURE_SKATING`, and `LUGE`.

You may be wondering why we're listing an enum under classes. It turns out that all enums are actually classes, and they can have more complex members like a class can. To learn more about this, see:
https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

`Athlete`
Create a class called `Athlete` to model the competitors at the Olympics. `Athlete` objects should have the following private instance data: a `String name`, a `String country`, a `Sport sport`, an `int goldMedals`, an `int silverMedals`, an `int bronzeMedals`, and a `boolean injured`.

`Athlete` should have two constructors: The first constructor should take a name, a country, a sport, number of gold medals, number of silver medals, and number of bronze medals in that order. Every `Athlete` should start uninjured. The second constructor should take only a name, country, and sport. It should set all medal values to 0. This constructor must call the first constructor. Do not simply set each instance variable again.

`Athlete` should have a getter method for each of its instance variables (`getName()`, `getCountry()`, etc.). The getter for the `injured` variable should be called `isInjured()`. Additionally, `Athlete` should have a setter for the `injured` variable called `setInjured(boolean injured)`.

An athlete should also be able to win some medals! `Athlete` should have the following methods:
`public void winGoldMedal()` - increments goldMedals by 1
`public void winSilverMedal()` - increments silverMedals by 1
`public void winBronzeMedal()` - increments bronzeMedals by 1
`public int getTotalMedals()` - returns the total number of medals ever won by the athlete.

`Event`
The last class for this program will model an Olympic event. An `Event` object should have the following private instance data: an `Athlete[] competitors`, an `Athlete goldMedalist`, an `Athlete silverMedalist`, an `Athlete bronzeMedalist`, and a `boolean eventPlayed`.

The `Event` class should have just one constructor, which takes in a `Athlete` array. This array is guaranteed to not have null elements. The gold, silver, and bronze medalists should should all start out as null, and `eventPlayed` should start out as false.

In this `Event` class we also want to model the unpredictability of `Athlete`s being injured during an event. The `Event` class should have the following method, which uses the `competitors` array to randomly decide if each athlete gets injured based on the probability parameter:
`public boolean injure(double probability)`

For one call of the injure method, each athlete in the competition should be tested (random value generated) against the passed-in probability. The chance that each athlete is injured should be equal to the probability passed into the method. If an athlete gets injured, set its injured variable to true. Finally, return true if any previously uninjured athlete was injured, otherwise return false.

Now we need to decide the winners of each event! Create the following method:
`public void play()`

This method should decide the fate of the event. We are going to use a simple (somewhat unrealistic) heuristic. Whichever athlete already has the most medals of a type is the person most likely to win that medal. For example, when deciding the silver medalist, you should choose the (winner) athlete with the most silver medals. If there is a tie, you should randomly choose the winner among the tied athletes.

As a very first step in this method, you should call the `Event`'s injure method to see if any athlete has an injury and cant finish the event. Any athlete who gets injured cannot be selected as a medalist. Next, the `play` method should decide the medalists based on which athlete has the most medals of the type being decided, as discussed above. First, pick a gold medalist from the `competitors` array, setting the event's `goldMedalist` to the selected competitor, and making sure to increment the medalist's gold medal count by calling `winGoldMedal()`, another method in the `Athlete` class. Repeat this process to select a silver medalist and bronze medalist keeping in mind that no athlete may receive more than one medal for an event. If there are no athletes left to assign as medalists, the variables should be left null. Once this process has been completed, `eventPlayed` should be set to true.

We did it! Now we have a working model of the Olympics, and we can create all kinds of athletes and events.

Make sure to test your code! That can be as simple as writing a main() method. Create some `Athlete` and `Event` objects, call their methods, and make sure everything works as expected. Nobody writes perfect code on their first try.

4 Helpful Hints

- Test! Test! Test! It is very important that you test your code thoroughly for this assignment because runtime errors have a heavy penalty (-25).

5 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **15** points.
Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar *.java
Audit done. Errors (potential points off):
0
```

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 points we would take off.

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **15** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

6 Turn-in Procedure

Non-compiling or missing submissions will receive a zero. NO exceptions

Submit `Sport.java`, `Athlete.java`, and `Event.java` to T-Square. Do not submit any compiled bytecode (`.class` files) or the Checkstyle jar file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code through Checkstyle!

6.1 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps insure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling 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. The real due date is midnight. Do not wait until the last minute!