COM1003 Java Programming, Spring Semester

Problem sheet 2: A simple OO program

This is an individual assignment. You are <u>not</u> allowed to work in groups and your code must be all <u>your own</u> work. You may not use other peoples' code or ideas, e.g., taken from the Internet.
Any form of unfair means (plagiarism or collusion) will be treated as a serious academic offence and will be processed under our University Discipline Regulations. These might result in a loss of marks, a possible failure of the module or even expulsion from our University.
For further details, please visit the UG Student Handbook:
<u>https://sites.google.com/sheffield.ac.uk/comughandbook/general-information/assessment/unfair-means</u>

Before you begin, you should have completed the weeks 1-4 practical sheets and be up-to-date with the most recent self-assessment quizzes available in BlackBoard.
You may ask demonstrators for help in understanding the problem, but this is intended to be an <u>individual assignment</u>, so you should not ask for help with the programming task.

In case you were unable to submit Problem sheet 1, please discuss this as soon as possible with the module leader by sending an email to m.villa-uriol@sheffield.ac.uk.

This problem sheet is based around a simple set of classes that provides a simple model of what fitness activity trackers are able to capture nowadays. The idea is to represent different activity trackers and the different measurements that are able to record.

Note that fitness trackers will be typically storing a number of measurements such as: steps, distance, and heart rate. In real life, they are also able to store measurements of energy expenditure or other variables. In this problem sheet, you will have access to a range of classes representing fitness activity trackers. You will notice that some of the classes still present some level of repetition an inefficiency that we would like to avoid by all means in the future using public abstract classes and/or interfaces. In the following weeks we will see how we would have done that. Please, do think about how you would improve the code presented in this problem sheet.

In this Problem sheet, you will be using the classes `Steps`, `HeartRate` and `Distance`, that you created in Problem sheet 1. This means that each of the new classes in this Problem sheet will need to have the following line importing the code from Problem sheet 1:

```java
import uk.ac.sheffield.com1003.problemsheet1.*;
```

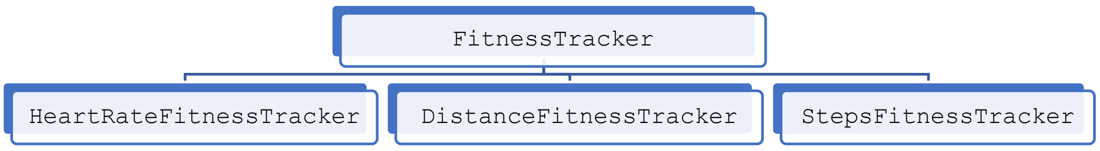
Similarly to what you did for Problem sheet 1, along with this handout, you can download from Blackboard the Java code for the following classes and place them in the `uk.ac.sheffield.com1003.problemsheet2` package:

- **`FitnessTracker.java`:** This class represents the most basic fitness tracker, only containing the name of the fitness activity tracker model. Note that the tracker does not store any measurement.
- **`StepsFitnessTracker.java`:** This class extends `FitnessTracker`, and represents a fitness tracker able to count steps only. Consequently, it only stores step measurements.
- **`HeartRateFitnessTracker.java`:** This class extends `FitnessTracker`, and is also able to capture the heart rate of the individual. Hence, it only stores heart rate.
- **`FitnessExperiment.java`:** This class represents an experiment where a number of fitness trackers from different brands were used simultaneously to track the physical activity of an individual for a period of time.

In task 2, you will need to program another class (you **have not** been provided with this class):

- **`DistanceFitnessTracker.java`:** This class extends `FitnessTracker`, and represents a fitness tracker able to record distance. This class should only store distance measurements.

Note that the class `FitnessTracker` is at the top of the inheritance hierarchy and that each of the classes inheriting from it, present different features.



To simplify the problem, our classes are only storing **one** type of measurement, either the total number of steps (class `StepsFitnessTracker`), or the total distance walked (class `DistanceFitnessTracker`), or the average heart rate (class `HeartRateFitnessTracker`). As opposed to what happens with commercial fitness trackers, none of our trackers is able to track simultaneously distance, heart rate and steps.
Each of these classes also have a method with a signature similar to:

```
public void addXYZ(XYZ newMeasurement)
```

For example:

- `StepsFitnessTracker` has `addSteps(Steps steps)`
- `HeartRateFitnessTracker` has `addHeartRage(HeartRate newHeartRate)`

These methods are meant to add more measurements to these objects, and they are meant to either keep the total distance, number of steps walked, or average heart rate. See the code provided and the comments in the code to understand how they work.

Task 1

Create an object of type `FitnessExperiment` that stores an array of `StepsFitnessTracker`, `DistanceFitnessTracker`, and `HeartRateFitnessTracker` objects. You could use code such as the following to initialise the array containing the fitness tracker measurements:

```
    FitnessTracker[] trackers = {
      new StepsFitnessTracker("steps", new Steps(230)),
      new StepsFitnessTracker("steps2", new Steps(150)),
      new StepsFitnessTracker("steps2", new Steps(150)),
      new HeartRateFitnessTracker("hr", new HeartRate(80)),
      new HeartRateFitnessTracker("hr", new HeartRate(80))
    };
```

In the `FitnessExperiment` class, complete the implementation of the methods named `getTotalSteps()` and `printExperimentDetails()`. The comments in the code specify how they should operate and provide you with additional hints to get you started.
Hint: You can use the method `getSteps()` in `StepsFitnessTracker`, but think how you will know the real type of each object in the array of fitness trackers (`trackers` in the example above).

Task 2

Write a new class called `DistanceFitnessTracker` that extends `FitnessTracker` (see the inheritance diagram). `DistanceFitnessTracker` will represent a fitness tracker able to record only distance. In this Problem Sheet, you will not be asked to convert between distance units (i.e. the default behaviour of the `Distance` class will be used, namely, kilometres).

Now, your array containing `FitnessTracker` objects could be initialised as follows:

```
FitnessTracker[] trackers = {
  new StepsFitnessTracker("steps", new Steps(230)),
  new StepsFitnessTracker("steps2", new Steps(150)),
  new StepsFitnessTracker("steps2", new Steps(150)),
  new HeartRateFitnessTracker("hr", new HeartRate(80)),
  new HeartRateFitnessTracker("hr", new HeartRate(80)),
  new DistanceFitnessTracker("dist1", new Distance(1000)),
  new DistanceFitnessTracker("dist2", new Distance(2430)),
  new DistanceFitnessTracker("dist2", new Distance(2430))
};
```

Similarly to how the constructors for `StepsFitnessTracker` and `HeartRateFitnessTracker` operate, the constructor of `DistanceFitnessTracker` objects should be initialised as follows:

```
public DistanceFitnessTracker(String modelName, Distance distance)
```

Implement the `DistanceFitnessTracker` class, and add to your `FitnessExperiment` class a method called `getTotalDistance()` that calculates the total distance walked by all the objects stored in your `FitnessExperiment`. Make sure that your `getTotalSteps()` in the class `FitnessExperiment` still works as expected. And last, make sure that you display appropriately both values (total number of steps and total distance walked in that experiment) using the `printExperimentDetails()` method.

Task 3

Write an `equals` method for the `StepsFitnessTracker` class that tests whether the one object is equal to another object that is passed as an argument (`Object otherObject`). The `Object` superclass provides an `equals` method that tests if two object references are equal. Your new `equals` method should override the `Object equals` method, and should test the following:

- Return true if the current (`this`) object, and `otherObject` are identical references (hint: you can use the `==` operator for this).
- Returns false if `otherObject` is null.
- Returns false if the current (`this`) object and `otherObject` have different classes (hint: you can use the `getClass()` method provided by the `Object` superclass, see the Java 8 API documentation for details).
- Casts `otherObject` to a `StepsFitnessTracker`, and tests for equality of each instance variable[1], if the instance variables are the same the method should return true.

[1] To test for equality of each instance variable, you might want to also use the `equals` methods, which means that similarly to what has been suggested in this problem sheet, you would also override the default behaviour of the `equals` methods for each of the classes `Distance`, `Steps`, and `HeartRate` in your Problem sheet 1 package (`uk.ac.sheffield.com1003.problemsheet1`).

Now write an `equals` method for the `DistanceFitnessTracker` and `HeartRateFitnessTracker` classes that:

- Use the superclass equals method, and return false if this method fails.
- Casts the parameter to either `HeartRateFitnessTracker` or `DistanceFitnessTracker`.
- Tests for equality of the subclass instance fields.

Please, use JUnit testing and write a test class named `TestFTEquals` to test that these new `equals` methods work properly. For inspiration, you can have a look at the test classes provided (`TestFitnessTracker`, `TestDistanceFitnessTracker`, `TestStepsFitnessTracker`, and `TestHeartRateFitnessTracker`) as they are already providing you some basic tests about how to test for equality of objects.

Task 4

Modify the class `FitnessExperiment` class so that it uses polymorphism to:

(i) display the properties of each object in a `FitnessExperiment`, and

(ii) identify any `StepsFitnessTracker`, `HeartRateFitnessTracker` and `DistanceFitnessTracker` objects that are equal. Here we are trying to identify those cases where a user mistake might have been made during the initialisation of the `FitnessExperiment` object (e.g. the data for the same tracker being introduced twice). Those situations should be flagged to the person leading the experiment to verify if the error was genuine or if the object was really the same object.

Testing your classes with provided basic tests

As done for Problem sheet 1, you have been provided with 5 additional test classes (`TestFitnessExperiment`, `TestFitnessTracker`, `TestDistanceFitnessTracker`, `TestStepsFitnessTracker`, and `TestHeartRateFitnessTracker`). They are meant to help you assess that your solution is 'correct' and that you are working in the right direction. By now we have already covered the topic of Unit testing in Java, so you can understand much better how the correctness of your submitted code will be assessed. Actually, some of the provided code will help you with Task 3.
Your code will need to successfully pass the tests provided in these 5 classes. Note that the battery of tests provided doesn't fully test your solution, so even if your solution passes these tests, you need to be sure that your solution is actually fully correct, as during marking, other tests will be run on your code.

Coding tips

Your code should be written using a good coding style, and you should note that ***readability*** is very important. The Java guidelines used by Google (https://google.github.io/styleguide/javaguide.html) are a good guide. In particular, you should:

- Use comments, but only when required (i.e. not excessively, but you should have a JavaDoc comment block at the head of each class).
- Use one code statement per line.
- Code your methods with as few lines as possible, and as many lines as needed. Ideally a complete method should be visible in a code editor without scrolling (see *Clean Cod*e for a longer discussion).
- Adhere to naming conventions; class names in *UpperCamelCase*, method names in *lowerCamelCase*, constant names in *CONSTANT_CASE*, and variable names in *lowerCamelCase*.

- Choose sensible, self-documenting, and descriptive names for all variables, methods, and classes.
- Use indentation consistently with either 2 or 4 whitespaces per indent (use whitespace because tabs can be interpreted differently on different operating systems).
- Before your code is checked, ask yourself whether the code is understandable by someone marking it.

Assessment and hand-in procedure

This problem sheet is worth 10% of your mark for the spring semester. How many marks you get depend on submitting a solution that:

- Implements the four tasks listed in this handout.
- Passes all the tests provided in the classes `TestFitnessExperiment`, `TestFitnessTracker`, `TestDistanceFitnessTracker`, `TestStepsFitnessTracker`, and `TestHeartRateFitnessTracker` (these classes should not be modified).

You must submit your code (problem sheet 2 classes along with the classes in your problem sheet 1) by the deadline (Friday 19th March, 3pm) <u>or earlier</u>.

Follow the guidance about how to perform your upload in Blackboard.

Start working on this problem sheet <u>as soon as possible</u> so that you can get help from the demonstrators or the lecturer during the labs.

Please, use the discussion board dedicated to 'Problem sheet 2 to ask your questions, without sharing your solution.

Late submissions will attract standard late penalties.

Maria-Cruz Villa-Uriol, February 2021