CSCU9A3 Practical 2

Cycle Ride Calculator – Objectified!

Intro

Remember how we had this giant method:

```
public double getDuration(int numMiles, String competency, int
numYearsExperience, boolean cyclingAlone, int temp, int windSpeed,
boolean isRaining)
```

There are rather too many parameters. How could we group some of this information together to result in fewer parameters, but the same information being passed?

Work together and come up with ideas.

Hints

Consider if we had the following information: student name, registration number, and username. We could group this information together as a Student class. It would look like the following:

Student
name registration number username

The information is then encapsulated within the class, so instead of passing the name, registration number and username separately, we could just pass a Student object instead.

Weather

Once you have some ideas, copy the code from:

```
Groups On Wide > CSCU9A3 > Practicals > Practical 2
```

Run the tests and ensure that they pass. Now take a look at the code and explore what has changed since last week. Much better!

As you can see we have some parameters that would naturally lend themselves to being encapsulated by a weather class. Make it so!

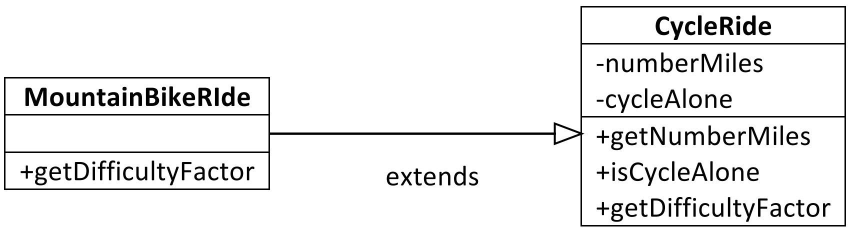
[Checkpoint]

This is a good place to check your progress. Show a demonstrator your lovely new weather class, and the `getDuration` method that now uses it. Also, remember to check that your tests all still pass.

Further Work

Inheritance

Imagine that there could be different types of `CycleRide`, for example you could go mountain biking or road racing. Each would have different difficulty factors. For example a road route would have no difficulty factor, whereas a mountain bike route would have a relatively high difficulty factor. ***Instead of storing the difficulty factor as a variable*** on `CycleRide`, design different subclasses of `CycleRide` that can return a value suitable for that type of ride? Diagrammatically this might be,



The method to return the difficulty factor should return 1 for a generic `CycleRide`, a higher number (say 1.5) for a mountain bike ride, and a smaller number for a road race. Implement `MountainBikeRide`, along with another example of a `CycleRide`. This will most likely require you to refer to notes, examples, the Internet, books. Do what it takes to get it all working. Ask for help if you really get stuck!

Once it is compiling, choose how the difficulty factor affects the overall cycle time, and implement it. Ensure that the tests pass for an easy ride, since the difficulty factor is 1 and should not affect the overall time, and then that the tests all fail when you use a different type of `CycleRide`. Note that you would also probably add more new features (methods and attributes) that would be specific to your new types of `CycleRide` that would differentiate them further from the base `CycleRide`.

The general rule with inheritance is that the base class contains the core functionality and the sub-classes extend this functionality to provide their own specific behaviour.

[Checkpoint]

Demonstrate both your working Weather and your new CycleRide classes.