CSCU9A3 Practical 1

Colin's Cycle Calculator: Part 1

To Discuss

At the beginning of the lab we shall discuss the following:

- IDEs : BlueJ, Eclipse and Netbeans
- Problem solving and group work
- Unit Testing
- Nice code

The Problem

Colin owns a small bicycle shop in Stirling, and has worked there for over 20 years. Over the years he has grown tired of the countless number of people who have asked him how long their particular cycle ride would take. He has written down a set of secret rules that he uses to estimate an answer and has come to you to turn his rules into a computer algorithm, as he'd like to offer this as a service on his shop website. Colin's rules are:

1. Firstly, gauge the cyclist's competency to work out their base average speed.
   - **Beginner: 10mph**
   - **Intermediate: 15mph**
   - **Advanced: 20mph**
2. Cycling alone or with someone else?
   - **Cycling together increases the cyclist's base average speed by 20%.**
3. The number of years experience is crucial to the average speed, as cyclists build up appropriate muscles over the years.
   - **Increase the average speed by 0.2mph for every year experience.**
4. Finally, the weather has an impact on the average speed during the cycle.
   - **For every degree Celsius lower than 10, decrease the average speed by 0.1mph.**
   - **For every degree Celsius over 20, decrease the average speed by 0.1mph.**
   - **For every set of 15mph winds, decrease the average speed by 1mph.**
   - **If it is raining, decrease the average speed by 2mph.**

Knowing Colin's top-secret information, it is over to you to devise a way to calculate the estimated number of hours that a particular cycle would take. You are given the following method heading:

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

Fill in the logic within this method and return Colin's estimated number of hours that the cycle will take.

Notes

- Duration (hours) = distance (miles) / speed (mph)
- To increase x by 20%: x = x*1.2

Getting Started

Do not be tempted to jump straight onto the computer and start coding. Use the techniques taught to you in lectures to break the problem down on paper and produce pseudo-code to solve the problems. Only then, once you are happy with your solution on paper, should you move onto coding.

A skeleton class has been created for you, along with an associated unit test. These can be found in

```
Groups On Wide > CSCU9A3 > Practicals > Practical 1
```

Copy these files to a new project directory and open them in your desired IDE. Run the tests to see that they all indeed fail. Now go make them pass! Remember: after each part you implement, run the tests to ensure that what you have done is correct so far. You should see the relevant test pass for each stage.

[Checkpoint]

When all the tests pass in `CycleCalculatorTest` you can be happy that your solution is correct. But before you call over a demonstrator, read through your code, tidy up any dead unused code, add comments where necessary and ensure that your indentations are correct. The checkpoint will be granted only with well-formatted code that works.

Arrays

It is vital that you understand arrays and can use them within your code. A method has been provided for you to fill in, which receives an array of durations (as `double` values) and you must return the total of all these durations added together (as a `double`).

```
public double getTotalDuration(double[] durations)
```

A unit test called `testGetTotalDuration` has been written for you, in `CycleCalculatorAdvancedTest`, to allow you to verify your solution.

Constants

Can you imagine if Colin decided to tweak some of his values in his rules? It would require you, as the programmer, to dive into the code and work out which numbers to change. This isn't particularly nice is it? When programming you can have **constants**, which are properties whose values do not change. These are typically declared, and values assigned, at the top of a class. In Java we would write:

```
private final int BEGINNER_SPEED = 10;
```

`final` is used to denote that the property value will never change, showing that the property is a constant. You'll notice that the property name was written in uppercase; this is common practice in many programming languages where constants are written in full uppercase with words separated by an underscore.

Go through your code and refactor any constants that exist and appreciate exactly why this makes your code a lot better (constants for low and high temperature thresholds are quite useful). Ask if you don't know. (Don't go crazy with this, just do enough to gain an appreciation then move on).

Time Format

Calling our `getDuration` method returns the time in a horrible format. For example 1.75 hours would be far nicer written as "1 hour and 45 mins". Create an appropriate method within CycleCalculator and get it to return a nicely formatted time as a String. The method signature would look something like:

```
public String getFormattedDuration(double time)
```

There is no unit test provided for this so add an additional test method to the CycleCalculatorTest class and write 3 different tests within this method. You should aim to try cases where the duration is exactly on an hour boundary and ones where it is not. Hint: You may find the Math.floor method and the modulus operator (%) useful here.

[Checkpoint]

Once you are sure your getFormattedDuration method works and that the tests you have set are valid (and are correctly passed), call over a demonstrator to check your code.

Java Reminders

Variables

```
int myInteger;
int myInitialisedInteger = 25;
double bigFloatingPointNumVar = 6.21;
String niceString = "Awesome";
```

Conditionals

```
if (<condition>) {
      //do things!
}
```

<condition> examples:

- `speed < 20`
- `cyclingAlone == false` (or `!cyclingAlone`)
- `isRaining == true` (or `isRaining`)
- `competence.equals("Beginner")`

  (notice for string comparisons we use '`.equals`' instead of '`==`')

Loops

For

```
for (initialise; condition; step) {
      // do stuff
}
```

Example: a loop that will run 10 times

```
for (int index = 0; index < 10; index++) {
      //do stuff
}
```

While

```
Initialise;
while (condition) {
      //do stuff
      Increment;
}
```

Example: a loop that will run 10 times

```
int index = 0;
while (index < 10) {
      //do stuff
      Index++;
}
```