ICT3101 Lab 2 part 2

12Recap:

13. SpecFlow reuses common step definitions for Gherkin statements in its Features. In our example this includes "**Given** I have a calculator"; hence, we have two separate step definition classes bound to our new scenario. This will cause you an issue to run, why? See if you can resolve this with a SpecFlow technique called "Context Injection".

New:

14. Context Injection works by allowing us to share the context of a scenario throughout its test steps. What this means, in our case, is that the instance of Calculator can be passed between our step definition classes. Use the following code at the beginning of **each** of the step definition classes to achieve Context Injection (editing the constructor name accordingly).

```
//Context Injection for SpecFLow:
        private Calculator _calculator;
        public UsingCalculatorDivisionsSteps(Calculator calc)
        {
            this._calculator = calc;
        }
//--------------------------------
```

15. Now check that your SpecFlow tests run for your new Divisions feature.

16. Next, create a similar Feature for your Factorial function. Note that SpecFlow is meant to translate high-level user requirements and hence doesn't facilitate exception handling (not intentionally, at least). Ensure you have at least two scenarios, one for a normal factorial calculation and another for a zero factorial. Are there any issues you face here?

Reminder of BDD ~definitions:

- A **Behavior** is an operation with inputs, actions, and expected outcomes.
- A **Scenario** is the specification of a behavior using formal steps and examples.
- A **Feature** is a desired product functionality involving multiple behaviors specified as scenarios.

Main Behaviour

17. For the main assignment of this lab, we're going to create a new feature and some new functions. With reference to lecture 1, use BDD to develop functions to calculate the Mean Time Between Failures (MTBF) and Availability. Here's an incomplete sample of the Feature to get you started (you need to replace the question marks with appropriate numbers).

```
Feature: UsingCalculatorAvailability
  In order to calculate MTBF and Availability
  As someone who struggles with math
  I want to be able to use my calculator to do this

@Availability
Scenario: Calculating MTBF
  Given I have a calculator
  When I have entered ? and ? into the calculator and press MTBF
  Then the availability result should be "?"

@Availability
Scenario: Calculating Availability
  Given I have a calculator
  When I have entered ? and ? into the calculator and press Availability
  Then the availability result should be "?"
```

Fill out another scenario or two which you think are relevant, and create the necessary step definitions for all. Consider any issues you face; also, do you think these new functions are well tested under the SpecFlow framework (why yes, why no)?

18. For the final part of this lab we need to create a new feature to help us with the Basic Musa model calculations. The feature should start like this:

```
Feature: UsingCalculatorBasicReliability
        In order to calculate the Basic Musa model's failures/intensities
        As a Software Quality Metric enthusiast
        I want to use my calculator to do this
```

    a. First generate scenarios, and the resulting step definitions, that allow for the calculation of the **current failure intensity** ($\lambda(\tau)$).

    b. Next generate scenarios, and the resulting step definitions, that allow for the calculation of the **average number of expected failures** ($\mu(\tau)$).

[Note, to do this properly, add an additional numerical parameter into your calculator input.]

—The end of part 2 questions—