SEG 3103 Summer 2021

Assignment #2

This assignment is to be done individually or by teams of two students MAXIMUM (please read the policy about cheating). You are required to hand in a zip file containing your answers to the questions as well as your Elixir / ExUnit implementation.

Problem 1

The objective of this problem is to develop a white box test suite for Calculator, an Elixir module that calculates your final grade in either percentage (e.g. 82%), letter grade (e.g. A), or numeric grade (e..g. 9).

The grades project is setup as a web application, but for this assignment our focus is solely on the **Grades.Calculator** module.

Your goal is to design and run enough test-cases to achieve *100% branch coverage* of methods ***percentage_grade***, ***letter_grade***, and ***numeric_grade***.

Note that Elixir the syntax for branching is different than languages like Java. Consider the following Elixir snippet.

```
avg_homework =
  if Enum.count(homework) == 0 do
    0
  else
    Enum.sum(homework) / Enum.count(homework)
  end
```

This would be similar to the following if/else branch in a language like Java.

```
double avg_homework;
if Enum.count(homework) == 0 do
  avg_homework = 0;
else
  avg_homework = Enum.sum(homework) / Enum.count(homework);
end
```

Also consider the following Elixir cond statement

```
cond do
  mark > 0.895 -> "A+"
  mark > 0.845 -> "A"
  :else -> "B"
end
```

This would be similar to the following if/elseif/elseif branching in a language like Java.

```
if (mark > 0.895) {
  return "A+";
} else if (mark > 0.845) {
  return "A";
} else {
  return "B";
}
```

Question 1.1 (10%)

Draw the simplified control flow graph corresponding to each of the methods ***percentage_grade***, ***letter_grade***, and ***numeric_grade***.

Question 1.2 (20%)

Provide a white box test design for 100% branch coverage of the methods ***percentage_grade***, ***letter_grade***, and ***numeric_grade***. Your test suite will be evaluated on the number of its test cases (try to have the smallest possible number of test cases that will allow 100% branch coverage). Use the following template for your test case design:

Test Case Number	Test Data	Expected results	Conditions Covered	Branches Covered

Question 1.3 (15%)

Provide an implementation of your test suite using ExUnit.

Question 1.4 (5%)

What is the degree of statement coverage obtained? If you weren't able to achieve 100% coverage explain why.

Please be sure to attach screenshots of your coverage results.

Elixir's coverage tool is primitive, as it only provides statement level accuracy.

```
mix test --cover
```

How might you address the limitations of a testing tool that only provides statement level coverage?

Problem 2

The calculator code is a huge mess. Your objective in problem 2 is to methodically refactor the calculator without breaking its behaviour. Using our test cases from Problem 1 we should be able to change the internal structure of our code without changing the external behaviour (the definition of refactoring).

Please use your **seg3103_playground** repository to track your work as each question below should be captured by at least 1 commit, and every step of the refactoring should leave the code in a working manner.

Please document your refactoring and reference specific commits in your code.

Question 2.1 (10%)

Extract a helper method **avg** to clean up the duplicate code like

```
avg_homework =
  if Enum.count(homework) == 0 do
    0
  else
    Enum.sum(homework) / Enum.count(homework)
  end
```

Question 2.2 (10%)

Extract a helper method **failed_to_participate?** to clean up duplicate code like

```
avg_homework < 0.4 || avg_exams < 0.4 || num_labs(labs) < 3
```

Question 2.3 (10%)

Extract a helper method **calculate_grade** to clean up duplicate code like

```
0.2 * avg_labs + 0.3 * avg_homework + 0.2 * midterm + 0.3 * final
```

Question 2.4 (20%)

Provide at least 2 additional refactoring to the code. Your refactoring *should not* require additional testing, however if you encounter any bugs in the original code then please fix them separately (ensuring your tests continue to pass) before continuing to refactor.