CST3110 – Testing and Verification
Coursework Project 1

This document lays out the coursework project details. Contact me if you have any questions about this document (`a.popescu@mdx.ac.uk`).

Read the entire document before starting work.

1 Project development and submission

Please follow the instructions below to the letter.

Environment: All work must be done in Eclipse. The submitted work consists of an Eclipse project of type `Java Project`. Select this type when creating your Java project.

Naming: Your Eclipse project must have a name in the format `M00123456-NG`, that is, your student number (`M00123456`) and your initials (`NG`). Give this name when creating your Java project.

Files to get before starting: Download the file `starter.zip` from the My Learning page, under "Project One". Right-click on your Eclipse project in the Package Explorer and select `Import > General > Archive File`, then choose the file you just downloaded.

Read, and make sure you understand, all the code included before starting work.

Before submitting: *You may not submit code that does not compile.* Syntax errors are not acceptable; if you see red marks on your code, you are not ready for submission!

File to submit: A submission is a `.zip` file created by right-clicking on your project and then selecting `Export > General > Archive file`. Name the `.zip` file the same as your Eclipse project plus the `.zip` extension (eg., `M00123456-NG.zip`). Double-check the contents of the `.zip` file before uploading.

Method: Submission is done electronically via the CST3110 My Learning page, under the folder "Coursework Project 1", in "Project 1 submissions". Upload the `.zip` file to submit here.

Deadline: 23:55, Sunday, 15th of December 2019 (at the end of learning week 11).

20-point scale	General scale
1	79% - 100%
2	76% - 78%
3	73% - 75%
4	70% - 72%
5	67% - 69%
6	65% - 66%
7	62% - 64%
8	60% - 61%
9	57% - 59%
10	55% - 56%
11	52% - 54%
12	50% - 51%
13	47% - 49%
14	45% - 46%
15	42% - 44%
16	40% - 41%
17	35% - 39%
18	30% - 34%
19	0% - 29%
20	Non-participation

Grade	Class of Honours Degree
1 2 3 4	FIRST CLASS
5 6 7 8	UPPER SECOND
9 10 11 12	LOWER SECOND
13 14 15 16	THIRD
17	FAIL – MARGINAL Compensation allowed
18	FAIL – Compensation allowed
19	FAIL – Compensation not allowed
20	FAIL- Incorporating failure to participate in assessment necessary to achieve all learning outcomes. Compensation not allowed

Figure 1: Conversion (left) and classification (right) on the 20-point scale.

2 Introduction

Middlesex University (MDX) maintains a computer system for storing, converting and classifying student grades.

2.1 The Middlesex 20-point scale

MDX does not use a percentage-scale for module grades. Instead it uses a 20-point scale that starts with 1 (a first-class result) down to 20 (failure by non-participation). Conversion from percentage-based grades to the 20-point scale is required in order to enter student grades into the MDX system.

Guidelines for conversion exist in MDX's academic regulations documents. Figure 1 (left) shows the correspondence between the two scales.

Classification of a grade on the 20-point scale into first class, upper second class and so on, is done according to Figure 1 (right).

Class/Borderline	Class of Qualification			
	3 Pass	2.2 Pass	2.1 Merit	1st Distinction
1st/Distinction (1-4)				50%
2.1/Merit or better (5-8)			50%	
2.2/Pass or better (9-12)		50%		
3/Pass or better (13-16)	100%		25% MAX	25% MAX

Figure 2: Profile classification at MDX.

2.2 Honours degree classification

An undergraduate student completing their study is awarded a degree classified as first class, upper second, etc. The rules for degree classification at MDX are presented here. From now on, a grade is always in the 20-point scale.

A *profile* is a list of grades. A *level 6 profile* only contains the grades obtained in the *third* year of study. A *level 5 profile* contains the grades obtained in the *second and third* years. Assuming four modules per year, level 6 profiles have four grades and level 5 profiles have eight grades.[1]

When a student *completes* the study for their degree, the degree class is decided as follows.

1. Two profiles are created using the student grades: the level 5 profile and the level 6 profile.
2. Each profile is classified as first class, upper second class, lower second class or third class as indicated in Figure 2.

   Specifically, if 50% of grades in the profile are first-class then the profile is first class. Otherwise, if 50% of grades are upper second or above, then the profile is upper second class. Otherwise, if 50% of grades are lower second or above, then the profile is lower second class. Otherwise, the profile is classified as third class.
3. Each profile is also marked as *clear* or *borderline* according to the bottom row of Figure 2.

   Namely, if the profile is classified as first or upper second class then it is a clear profile if the third class grades in the profile are no more than 25% of the total, otherwise the profile is borderline. Lower second and third class profiles are always clear.
4. The following rules are then applied to decide which degree classification is awarded.
   (a) If both profiles have the same classification then that classification is awarded.
   (b) Otherwise, if the level 6 profile is better and that profile is clear, and no more than one class above the level 5 profile, then the level 6 profile classification is awarded.
   (c) Otherwise, if the level 5 profile is better and that profile is clear, and no more than one class above the level 6 profile, then the level 5 profile classification is awarded.
   (d) Otherwise, a procedure called *discretion* is applied.

[1] For simplicity, we assume that we only deal with full-time undergraduates on a three-year degree (with four modules per year) who never repeat a year, fail a module, defer, interrupt or otherwise deviate from the standard pattern.

3 Requirements

3.1 The class `Grade`

The class `Grade` holds a grade in the MDX 20-point scale. The requirements are:

- The constructor must throw `IllegalArgumentException` if its input is outside 1–20.
- The static method `fromPercentage` creates a `Grade` object. If the input is within 0–100 then the grade returned is determined according to Figure 1 (left). If the argument is the number -1 then the grade 20 (non-participation) is returned. The method must throw an `IllegalArgumentException` if its argument is not within 0-100, nor -1.
- The method `classify` returns the `Classification` (first, upper second, etc) of the current grade object (as stored in the `points` field) according to Figure 1 (right). We use a single fail classification for simplicity.

3.2 The class `Profile`

This class holds a profile. All requirements below are as explained in Section 2.2.

- The constructor takes a list of grades to be inserted in the profile. It must throw an `IllegalArgumentException` if there are any fail grades, or if the list is empty or **`null`**.
- The method `isClear` must return **`true`** if the current profile is clear, and **`false`** if the profile is borderline.
- The `classify` method must return the classification of the current profile.

3.3 The class `Degree`

This class represents a degree to be awarded to a student.

- The constructor takes two `Lists` of `Grades` for years two and three respectively. It must throw an `IllegalArgumentException` whenever either list given is **`null`**, does not contain *four* grades, or contains a fail grade.
- The method `classify` must return the correct degree classification, as in Section 2.2.

4 Project tasks

Task 1: For each class below, design and implement (with `jUnit`) the following tests.

- Class `Grade`:
  - TWO tests for inputs below and above the valid range for the constructor.
  - ONE test for a valid input, checking that `getPoints` returns the right value.
  - FIVE tests for `classify`, using `Classification`s as equivalence classes.
  - TWO tests for inputs below and above the valid range for `fromPercentage`.
  - TWENTY tests for `fromPercentage`, using each point in the 20-point scale as an equivalence class.
- Class `Profile`:
  - THREE tests for the constructor, one for each distinct way input can be invalid.
  - SIX tests, one for each possible combination of `Classification` and truth value (whether the profile is clear or not) as an equivalence class.
- Class `Degree`:
  - THREE tests for the constructor, one for each distinct way input can be invalid.
  - FIVE tests, using `Classification`s as equivalence classes.

Task 2: Implement the functionality required to pass the tests, with requirements as described in Section 3.

You are encouraged to write additional tests if it helps with development. These tests will not be assessed directly, but will contribute towards coverage (see next task).

Task 3: Achieve *maximum branch coverage* of the methods listed below. If necessary, do this by adding new tests. If it is impossible to achieve maximum coverage by adding tests, consider whether your implementation can be improved (e.g., by eliminating dead code).

- Class `Grade`: constructor, `classify`, `fromPercentage`.
- Class `Profile`: constructor, `classify`, `isClear`.
- Class `Degree`: constructor, `classify`.

Here, coverage of a method (say, `Grade.classify`) will be measured by adding up the coverage reports on all tests *for that method* produced in tasks 1 (above) and 3 (this one).

New tests produced as part of this task should be written into a new source file per class (e.g., `GradeTestCoverage.java`, `ProfileTestCoverage.java`, etc).

5 Guidelines

Do

- override `Object`-derived methods such as `equals` or `toString`, if necessary;
- add standard Java methods, such as `compareTo`, if necessary;
- add **`private`** methods and fields, if necessary;
- use a wide range of `jUnit` assertions and features such as parameterised tests;
- modify the bodies of the methods indicated in the starter code;
- write code that is clear, reabable and consistently indented.

Do not

- change any **`public`** interfaces (method names, accessibility modifiers, types);
- add or remove any **`public`** methods;
- modify the class `Classification`;
- submit code that does not compile;
- email me your submission.

6 Assessment

1. The coursework is for individual work, not teamwork.
2. **Late submissions will not be allowed**. Continuous re-submission is possible, so you can submit a draft version early and keep submitting improved versions until the deadline.
3. Grading:

Class	Total	Implementation	Tests	Coverage
`Grade`	30%	10%	15%	5%
`Profile`	40%	20%	15%	5%
`Degree`	30%	10%	10%	10%
Total	100%	40%	40%	20%

4. Implementations will be assessed using the following criteria:
   (a) Implementation correctness.
   (b) Code clarity and readability.
5. Test suites will be assessed using the following criteria:
   (a) Agreement between test input selection and assertions with requirements.
   (b) Test code clarity and readability.

   Note: *tests that fail may still contribute positively to your grade.*