Lab Assignment 1: **Grade Calculator Tool**

Aim – **To develop a text-based interactive program to simulate the assignment of marks, single-student mark look-up and computation of simple statistics for a computer science course of a given class size.**

The main objective in this assignment is to build a small prototype which would be useful in simulating and testing a partial set of requirements of a fully-featured tool that an instructor can use to maintain partial grades and compute final grades in a course. Secondly, to give you practice designing and coding simple classes and methods using loops, conditionals and console I/O in the Java language. Lastly, to give you an initial exposure to object-oriented techniques to solve computational problems. To attain these goals, you will organize the programming of this prototype around the following modular entities:

First, you will write a Java class named **Student** that allows the creation and maintenance of the marks, as well as of the final mark, for each student in the course.

Second, you will write a Java class named **GradeCalculator** which will drive the operational features of your tool and the text-based console I/O for receiving commands from and issuing results to a user.

Description

I -- User Interface and Program Flow

Upon starting, your tool will display on the console the following menu:

Grade Calculator (Version 0.1). Author: [Your first and last name]

1 – Simulate Course Marks

2 – View/Update Student Marks

3 – Run Mark Statistics

Select Option [1, 2 or 3] (9 to Quit):

Upon completing the processing of any of the operations 1, 2, or 3, your program will return to display the previous menu and wait for the user to select a menu option. If the user enters 9, the program terminates.

a) **Simulate Course Marks** – Selecting this option will run a simulation that creates all the Student objects for a course of size **N students**, assigns to each student randomly-generated marks for assignments 1 and 2, and the final exam. If Student objects from previous course marks simulations exist when running this simulation, their **status** attribute will set to false before creating the Student objects corresponding to the new simulation.

First, the program will prompt the user to enter the class size by displaying the message: **Enter course enrollment size:**

Next, the program will issue in sequence the following prompts to the user, asking for weight percentages within given ranges: **Enter weight assignment 1 (20-30):, Enter weight assignment 2 (20-30):, Enter weight final exam (40-60):**. If any of the weights is out of range, the program

will repeat the prompt for the respective weight. If the entered weights do not add up to 100, the program will display the message **<< Error: weights do not add up to 100% >>**, and will return to and display the selection menu.

b) **View/Update Student Marks** – Selecting this option will display the prompt **Enter Student Number:**

If the user enters an invalid student number, or the number corresponds to an inactive Student object (i.e., object's **status** attribute is set to **false**), the program will display the message: **[entered student number] is invalid,** and will return to and display the selection menu. Otherwise, it will display the message: **View or Update? (V/U):,** and will wait for the user to enter a selection. If the user enters V, the program will display all the course marks for the selected student. If the user selects U, the program will display the prompt **Mark Type? (A1, A2 or FE):** and will wait for a selection from the user. Once the user has input one of the possible options, the program will display the selected mark as: **[Mark Type] is [mark],** and will return to and display the selection menu. If the user enters an invalid mark type, the program will display the message: **[Mark Type] is an invalid mark type,** and will return to and display the selection menu.

Upon selection of option 2, the program checks whether Student objects have been created by running option 1 first. If that is not the case, it will display the message: **<< Error: empty class list >> Run option 1 first**. Afterward, it will return to and display the option selection menu.

c) **Run Mark Statistics** – Selecting this option will display on the console a single report, listing the marks for all the active Student objects (i.e., status is true), whose format is the following:

Student Number	**A1 ([weight] %)**	**A2 ([weight] %)**	**FE ([weight] %)**	**Final Mark**
**********	**********	**********	**********	**********
12345678	**55**	**85**	**70**	**75**
99999999	**70**	**90**	**75**	**80**

etc.

**********	**********	**********	**********	**********
AVERAGES	**63**	**85**	**72**	**76**

Highest final mark is [mark]

Lowest final mark is [mark]

After displaying the report, the program will issue the prompt: **Press any key to continue;** thereafter, it will return to and display the selection menu.

II – Classes

To implement your grade calculator program, you will write two Java classes with the following specifications:

1) **Student** class – This class must contain the following:

i) Attributes:

studentNumber: integer
status: boolean
assignment1Mark: integer
assignment2Mark: integer
finalExamMark: integer

ii) Methods:

a) A constructor method **Student** that sets the student number as the identifier for each student object. This method takes as a parameter an integer **studentNumber**, checks that this integer contains values between 10000000 and 99999999, and updates the **status** attribute with the validity of this requirement (true or false).

b) A method **updateMark** that takes as parameters: i) a string **markType** with possible values A1, A2 and FE, ii) an integer **mark** with a value between 0 and 100, and iii) an integer **weight** with a value between 10 and 50 representing the percentage assigned to the current mark. This method updates the respective mark for a student and returns a true or false value reflecting the validity of the mark update requirements.

c) A method **getMark** that takes a parameter **markType**, with possible values A1, A2 and FE and returns the requested mark. If the markType entered is invalid, the method returns -1.

d) A method **getStudentNumber** that returns the student number of a student object.

e) A method **setStudentNumber** that updates the **studentNumber** attribute using the same validation method and status update described for the constructor method.

f) A method **getStatus** that returns the status of a student object.

g) **Other methods that you feel are needed**

2) **GradeCalculator** class – This class must contain the following:

i) Attributes:

classSize: integer

classList: array of **Student** objects
assignment1Weight: integer
assignment2Weight: integer
finalExamWeight: integer

ii) Methods:

a) The **main** method will implement the program flow and user interactions previously described in section I (User Interface and Program Flow). For this purpose, it will call the methods listed below and any other methods you feel are needed.

b) A method **generateStudentMarks** that takes a student number and generates a **Student** object with its marks for assignments 1 and 2, and the final exam, randomly generated.

c) A method **computeStudentGrade** that takes a **Student** object and computes the student's final grade in the course.

d) A method **printClassReport** that takes an array of **Student** objects and displays the report specified in section I.

e) **Other methods that you feel are needed**

III – Testing

Produce a trace file, which contains the logging of all the user interaction cases explained in section I, points a) and b). This .txt file will also include a logging of a mark statistics report run, as described in section I, point c).

IV – Submission

Submit the source code for classes **Student** and **GradeCalculator,** as well as the trace file specified in section III, to onQ by **January 30, 2018, 1:30 p.m.** Include all these files in a single zip archive file named **[your student number]_Assignment1.zip**. For example, if your student number is 12345678 then you will name your submission file as 12345678_Assignment1.zip

IV – Marking Scheme

• Implementation of classes **Student** and **GradeCalculator**	12 points
• Generation of testing trace (.	6 points
• Code style and documentation	2 points
TOTAL:	20 points

V – Additional Guidelines for this Assignment

- Your program should use good indentation and white space, good variable names and some internal comments explaining what the code is doing. One comment at the top of the program should at least contain your name and NetID.

- Use the **Random** class, in the **java.util.Random** package, to create a random number generation object seeded with a fairly arbitrary number. For example, the following code accomplishes this:

  static Random generator = new Random(System.currentTimeMillis());

  System.currentTimeMillis() returns the current time in milliseconds, so it will yield a different seed value every time you run your program. You need to call the **nextInt()** method (see details in the Java API documentation) in the **generator** object to obtain a new random integer. This object could be assigned to an attribute of class **GradeCalculator**.