COMP503/ENSE502/ENSE602 _ Assignment 1
Sem 2 2020

Technical School App

Due date: Friday, 28th August 2020 at 5:00 pm

This assignment has 100 marks and is worth 10% of your final grade.

Brief

For this assignment, you will **individually** develop a **Technical School App** (eight classes) in Java to determine whether or not a student is certified, their transcript is checked against a certification criterion. For this assignment, you will develop classes to store data for modules, students, and their transcripts. You will demonstrate the functionality of your code on sample transcripts.

Methodology and Marking Scheme

You have been provided with an outline of eight classes that contain the core functionalities of the technical school app: **Grade**, **ModuleType**, **Level**, **Module**, **Student, Result, TechnicalSchool,** and **StudentEvaluation**. The relationships between classes are described below using a UML Class Diagram (Figure 1). Complete the assignment by using the given UML class diagram. Please use the following steps to develop the eight classes with the provided instance variables and methods using appropriate access modifiers (as shown in the diagram). Please take the time to test your methods as you progress.

Note: No other methods/instance variables are required. But feel free if you would like to add more methods!

COMP503/ENSE502/ENSE602 _ Assignment 1 Sem 2 2020

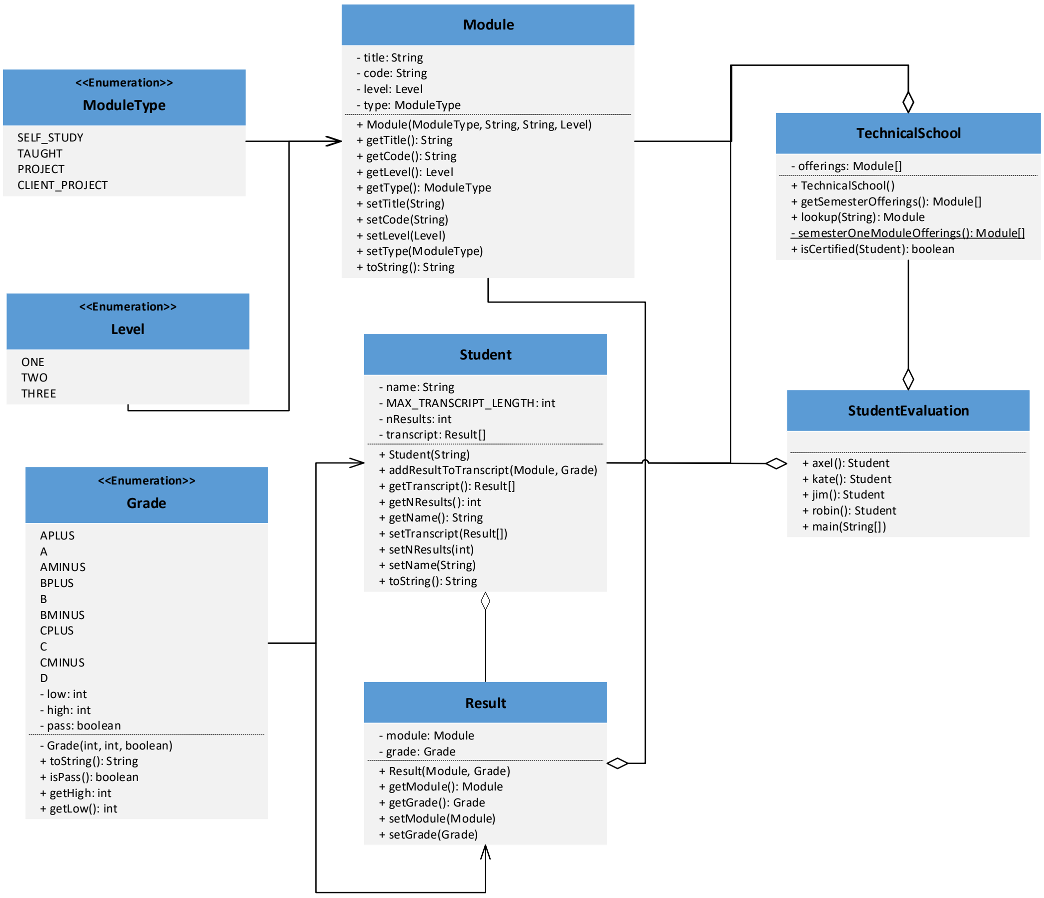

Figure 1: UML Diagram of The Technical School App

Step 1: Creating enumerated types

1. Create an enumerated type **Grade**, which maintains:
   - A list of Grade values with associated boundaries for each mark:
     - 100% ≥ A+ (**APLUS**) ≥ 90%
     - 90% > **A** ≥ 85%
     - 85% > A- (**AMINUS**) ≥ 80%
     - 80% > B+(**BPLUS**) ≥ 75%
     - 75% > **B** ≥ 70%
     - 70% > B- (**BMINUS**) ≥ 65%
     - 65% > C+ (**CPLUS**) ≥ 60%
     - 60% > **C** ≥ 55%
     - 55% > C- (**CMINUS**) ≥ 50%
     - 50% > **D**
   - Encapsulated instance variables (for each Grade):
     - Two integer variables indicating the range of each letter grade
     - A Boolean variable to indicate if the grade is a pass (greater than or equal to 50%).

   NOTE: the variables are immutable!

  - `isPass()` method, which returns true if the Grade is a pass and false otherwise.
  - a constructor with input for all instance variables to initialize Grade objects.
2. Create an enumerated type **ModuleType**, which maintains four values corresponding to the type of modules available: **SELF_STUDY**, **TAUGHT**, **PROJECT,** and **CLIENT_PROJECT**.
3. Create an enumerated type Level containing three-level values: **ONE**, **TWO**, **THREE**.

Step 2: The Module Class

Create the **Module** class which:

- maintains instance variables for type, title, code, and level. All instance variables are private with get and set methods.
- has one constructor with input for all instance variables to initialize Module object.
- overrides the toString method to return a beautiful text representation.

Step 3: The Technical School class

The **TechnicalSchool** class maintains the Semester 1 Module Offerings T based on the following table:

Module Type	Title	Code	Level
Taught	Programming	PROG101	1
Taught	Statistics	STAT102	1
Taught	Database Design	DATA222	2
Taught	Object-Oriented Programming	PROG202	2
Taught	Information Systems	INSY313	3
Taught	Web Services	WEBS332	3
Self-Study	Technology Applications	TECH103	1
Self-Study	Theory of Computation	THEO111	1
Self-Study	Model Checking	MODC233	2
Self-Study	Topology	TOPG233	2
Self-Study	Logic	LOGI321	3
Project	Web App Dev	PROJ399	3
Client Project	Great Code Company	EXTO396	3

Table1: Semester 1 Module Offerings

The **TechnicalSchool** class:

- maintains a private instance variable `Module[] offerings` which stores the Semester 1 Modules based on Table 1.
- has a method `private static Module[] semesterOneModuleOfferings()`, which returns a primitive array populated by 13 Module objects, corresponding to each row of the table.
- create a default constructor `TechnicalSchool()` instantiates the `offerings` variable with appropriate Module objects comes from `semesterOneModuleOfferings()` (i.e. `this.offerings = TechnicalSchool.semesterOneModuleOfferings()`)
- has a get method for `offerings` variable.
- has a `public Module lookup(String code)` method to search the offerings array and return a module with the matching code.

Step 4: The Result class

The **Result** class:

- stores Module and Grade objects.
- has an appropriate constructor to initialize both instance variables.
- has get and set methods.
- has a toString method to represent the result object.

Step 5: The Student class

The **Student** class:

- maintains an array of results, called a transcript.
- declares
  - a student name,
  - a private final static integer variable that sets the maximum size of the array (e.g., *MAX_TRANSCRIPT_LENGTH* = 20)
  - an instance variable called numberOfResults, which maintains the number of results available for the student.

Complete the **Student** class with the following methods:

- a constructor initializes name (using input parameter) and transcript as a new array
- `public void addResultToTranscript(Module module, Grade grade)`, which creates a Result object and adds it to the end of the transcript and updates numberOfResults. If the transcript is already full, do not add the result.
- `public Result[] getTranscript()` which returns an array of Result objects. The returned array should not contain any null entries; e.g., it is of length numberOfResults.

Step 6: Certification Algorithm

The certification algorithm is a method (i.e., isCertified method) in the **TechnicalSchool** class. It takes a Student object as an input and examines his transcript to determine if he is certified, according to the following criteria:

- at least three modules passed at level 1, either taught or self-study **AND**
- at least three modules passed at level 2, **more than** one must be self-study **AND**
- at least four modules passed at level 3, **at least** two must be taught **AND**
- at least one project module passed (either of project or client project).

Complete the method and returns true if the student satisfies **ALL of** the above four criteria and false otherwise.

Step 7: The StudentEvaluation Class

Create a **StudentEvaluation** class which contains:

- the main method to run your application
- four static methods. Each method returns a Student object populated with results from the Transcript Tables below. To test and demonstrate your implementation of the certification algorithm, you need to print out the result for students Robin, Kate, Axel, and Jim. You also need to provide more examples of students to show the functionality of your code.

Transcript for Robin	**Transcript for Kate**	**Transcript for Axel**	**Transcript for Jim**
PROG101 C	PROG101 A+	PROG101 B+	PROG101 A
DATA222 C	STAT102 A-	STAT102 C	STAT102 B+
INSY313 C+	TECH103 B+	DATA222 A	DATA222 C+
WEBS332 C+	MODC233 A	INSY313 A-	PROG202 C
TECH103 C+	TOPG233 C	WEBS332 A	INSY313 C
MODC233 C-	DATA222 A	TECH103 D	WEBS332 C+
TOPG233 C-	INSY313 B+	MODC233 B	TECH103 C-
PROJ399 A+	WEBS332 A-	TOPG233 B	THEO111 D
false	PROJ399 B	PROJ399 C-	MODC233 A+
	EXTO396 B	EXTO396 C	TOPG233 A
	true	false	LOGI321 B
			PROJ399 B-
			EXTO396 A+
			true

Javadoc Commenting

1. Your classes must have commenting of the form:

```
/**
 * Comment describing the class.
 * @author yourname studentnumber
 **/
```

2. All methods must be commented with appropriate Javadocs metatags. For example:

```
/**
        * A comment to describe the method
        * @param a first parameter description
        * @param b second parameter description
        * @return a description of the returned result
        * @author studentnumber
        * */
```

COMP503/ENSE502/ENSE602 _ Assignment 1 Sem 2 2020

Marking Scheme

Criteria:	Weight:	Grade A Grade Range: 100 ≥ x ≥ 80%	Grade B Grade Range: 80 > x ≥ 65%	Grade C Grade Range: 65 > x ≥ 50%	Grade D Grade Range: 50 > x ≥ 0%
The functionality of Module class	**20%**	OOP paradigm consistently used for the implementation of all Module functionality.	Inconsistent use of OOP paradigm but the correct implementation of Module functionality	Incorrect Module functionality and poor use of OOP paradigm	Absent Module class or code does not compile
The functionality of TechnicalSchool class	**30%**	OOP paradigm consistently used for the implementation of all TechnicalSchool functionality.	Inconsistent use of OOP paradigm but the correct implementation of TechnicalSchool functionality	Some basic functionality of TechnicalSchool	Absent functionality of TechnicalSchool or code does not compile.
The functionality of Student class	**20%**	OOP paradigm consistently used for the implementation of all Student functionality.	Inconsistent use of OOP paradigm but the correct implementation of Student functionality	Some basic functionality of Student	Absent functionality of the Student or code does not compile.
Functionality of StudentEvaluation class	**10%**	OOP paradigm consistently used for implementation of all evaluation functionality.	Inconsistent use of OOP paradigm but the correct implementation of evaluation functionality	Some basic functionality of Evaluation	Absent functionality of Evaluation or code does not compile.
Code Quality: -Whitespace -Naming -Reuse -Modularity -Encapsulation	**15%**	Whitespace is comprehensively consistent. All naming is sensible and meaningful. Code reuse is present. The code is modular. The code is well encapsulated.	Whitespace is comprehensively consistent. The majority of naming is sensible. The code is modular. The code is encapsulated.	Whitespace is comprehensively consistent. The code has some modularity. The code has some encapsulation.	Whitespace is inconsistent, and hence code is difficult to read.
Documentation Standards: -Algorithms Commented -Javadoc	**5%**	The entire codebase has comprehensive Javadoc commenting. Algorithms are well commented.	The majority of the codebase features Javadoc commenting. The majority of algorithms are commented.	Some Javadoc comments present. Some algorithms are commented.	No Javadoc comments present.

COMP503/ENSE502/ENSE602 _ Assignment 1 Sem 2 2020

Authenticity

Remember!

- It is unacceptable to hand in any code which has previously been submitted for assessment (for any paper, including Programming 2) or available online
- **All work submitted must be unique and your own**!

Submission Instructions

Export your Java project from Eclipse as an archive **.zip** file before the deadline.
Please ensure your submission matches the following format:
lastname-firstname-studentid.zip (Replace the underlined text with your personal details).

An extension will only be considered with a Special Consideration Form approved by the School Registrar. These forms are available at AUT Blackboard.

You will receive your marked assignment via Blackboard. Please look over your entire assignment to make sure that it has been marked correctly. If you have any concerns, you must raise them with the lecturer. You have **one week** to raise any concerns regarding your mark. After that time, your mark cannot be changed.

Do not go to the lecturer because you do not like your mark. Only go if you feel something has been mismarked.