University of Exeter

College of Engineering, Mathematics and Physical Sciences

ECM1410

Object-Oriented Programming

Continuous Assessment

Date Set: 23th January 2021
Date Due: 12th February 2021

This CA comprises 20% of the overall module assessment.

This is an **individual** exercise, and your attention is drawn to the guidelines on collaboration and plagiarism in the College Handbook (`https://student-harrison.emps.ex.ac.uk/`).

This assessment covers the use and implementation of a range of object-oriented concepts using the Java programming language that you are covering in ECM1410. The assignment is *summative*. Please ensure you read the entire document before you begin the assessment.

This coursework is devised for you to do in your own time, rather than in the workshop sessions, and will help you enhance your understanding of Object-Oriented Programming. Remember to follow the established deadline and submission guidelines in the module's ELE page, and to use your own words when writing your answers.

1 Coding (80 marks)

For this part of the assignment you must implement a software solution for a given problem. You must follow the object-oriented paradigm and use Java while addressing your solution. As the focus of this first assessment is not modelling OO (yet!), we have already specified for you the classes of the system with their respective attributes.

Problem Description University of Knowledge (UoK) is a small university. Given its recurrent lower budgets, UoK could not afford a big and integrated system such as an Enterprise Resource Planner (ERP). Rather, UoK has been developing several independent systems to reduce paperwork. A group of students decided they could help. They propose to use their coursework to gradually create such a system.

As a proof of concept, the university need a system able to load module descriptors and students. Then, they need to create modules and register students to the modules. Ultimately, the university wants to generate reports about students and their module performance.

Please, find below the minimum requirements for this version of this system:

- Module Descriptor:
  1. A module descriptor must have a *code*, a *name*, and a double array to store the *weights* of the continuous assessments.
  2. The code and the name can never be null.
  3. The code must be unique.
  4. The Continuous Assessment (CA) weights must sum up to 1, and must be non-negative.
- Student:
  1. A student must have an *ID*, a *name*, a *gender*, a *GPA*, and an array of *records* for each module they have been enrolled.
  2. The ID and name cannot be null.
  3. The ID must be unique.
  4. The gender should be represented by one of the following characters: 'F', 'M', or 'X'. The student can also prefer not to disclose this information, thus gender can be an empty field.
  5. The grade point average (GPA) is the average of student record final scores (the explanation of Student Record is below in the Official Transcript example).
  6. The system must be able to generate a transcript containing all student records, grouped by year and term. Implement the function `public String printTranscript()` using the following format:

```
                        University of Knowledge - Official Transcript
<new line>
<new line>
ID: <student ID>
Name: <student name>
GPA: <student GPA>
<new line>
| <module year> | <module term> | <module descriptor code> | <student record final score> |
| <module year> | <module term> | <module descriptor code> | <student record final score> |
...
| <module year> | <module term> | <module descriptor code> | <student record final score> |
<new line to separate different terms>
| <module year> | <module term> | <module descriptor code> | <student record final score> |
...
```

- Student record:
  1. A student record refers to a *student* in a *module*, and contains an array of *marks*, a *final score*, and a Boolean to indicate whether the student was *above the average*.
  2. The final score is a weighted average calculated based on the arrays of marks and the array of weights of the module descriptor of the respective module.
  3. Marks and the final score must range between 0 and 100.
  4. *Above the average* should be *true* if the student final score is greater than the average final scores in that particular module.
  5. A student can only have one record per module.
- Module:
  1. A module must refer to a *module descriptor* in a specific *year* and *term*. It contains an array of *student records*, and *average score*.
  2. The average score of a module is the mean of the student record final scores for that module.
  3. A module descriptor can only be offered once per year and term. In other words, at most, there will be only one instance of a module with the same descriptor, year, and term.
- University:
  1. The university has an array of *module descriptors*, an array of *students*, and an array of *modules*.
  2. The system must be able to initialise the array of module descriptors as shown in Table 1.
  3. The system must be able to initialise the array of students as shown in Table 2.
  4. The system must be able to initialise modules with their respective students and marks as shown in Table 3.
  5. In order to generate reports, the UoK (University.java) must implement the following functions:
     (a) `public int getTotalNumberStudents()`: return the number of students registered in the system.
     (b) `public Student getBestStudent()`: return the student with the highest GPA.
     (c) `public Module getBestModule()`: return the module with the highest average score.

Use the Tables 1, 2, and 3 to fill the information of Students, Modules and University. Use the provided skeleton/-code of classes to implement the above-mentioned system. You should NOT rename any of the given classes nor their attributes and methods, as we are going to use automated scripts to check your program. You can create auxiliary methods, attributes, or classes to fulfil the implementation of the requested functionalities.

2 Data

Use the following skeleton of classes and the initial data in the tables.

2.1 Classes

2.1.1 ModuleDescriptor.java

```
public class ModuleDescriptor {

   private String code;

   private String name;

   private double[] continuousAssignmentWeights;

}
```

2.1.2 Student.java

```
public class Student {

   private int id;

   private String name;

   private char gender;

   private double gpa;

   private StudentRecord[] records;

   public String printTranscript() {
      // do something
      return "";
   }
}
```

2.1.3 StudentRecord.java

```
public class StudentRecord {

   private Student student;

   private Module module;

   private double[] marks;

   private double finalScore;

   private Boolean isAboveAverage;

}
```

2.1.4 Module.java

```
public class Module {

   private int year;

   private byte term;

   private ModuleDescriptor module;

   private StudentRecord[] records;

   private double finalAverageGrade;

}
```

2.1.5 University.java

```
public class University {

   private ModuleDescriptor[] moduleDescriptors;

   private Student[] students;

   private Module[] modules;

   /**
    * @return The number of students registered in the system.
    */
   public int getTotalNumberStudents() {
      // TODO - needs to be implemented
      return null;
   }

   /**
    * @return The student with the highest GPA.
    */
   public Student getBestStudent() {
      // TODO - needs to be implemented
      return null;
   }

   /**
    * @return The module with the highest average score.
    */
   public Module getBestModule() {
      // TODO - needs to be implemented
      return null;
   }

   public static void main(String[] args) {
      // TODO - needs to be implemented
   }
}
```

2.2 Initial Data

Table 1: Use this table to initialize the module descriptors.

Module Name	Module Code	CA weights
Real World Mathematics	ECM0002	$[0.1, 0.3, 0.6]$
Programming	ECM1400	$[0.25, 0.25, 0.25, 0.25]$
Data Structures	ECM1406	$[0.25, 0.25, 0.5]$
Object-Oriented Programming	ECM1410	$[0.2, 0.3, 0.5]$
Information Systems	BEM2027	$[0.1, 0.3, 0.3, 0.3]$
Thermal Physics	PHY2023	$[0.4, 0.6]$

Table 2: Use this table to initialize the students.

ID	Name	Gender
1000	Ana	F
1001	Oliver	M
1002	Mary	F
1003	John	M
1004	Noah	M
1005	Chico	M
1006	Maria	F
1007	Mark	X
1008	Lia	F
1009	Rachel	F

Table 3: Use this table to initialize the modules and the student records.

Student Code	Module Code	Year	Term	Marks
1000	ECM1400	2019	1	[9, 10, 10, 10]
1001	ECM1400	2019	1	[8, 8, 8, 9]
1002	ECM1400	2019	1	[5, 5, 6, 5]
1003	ECM1400	2019	1	[6, 4, 7, 9]
1004	ECM1400	2019	1	[10, 9, 10, 9]
1005	PHY2023	2019	1	[9, 9]
1006	PHY2023	2019	1	[6, 9]
1007	PHY2023	2019	1	[5, 6]
1008	PHY2023	2019	1	[9, 7]
1009	PHY2023	2019	1	[8, 5]
1000	BEM2027	2019	2	[10, 10, 9.5, 10]
1001	BEM2027	2019	2	[7, 8.5, 8.2, 8]
1002	BEM2027	2019	2	[6.5, 7.0, 5.5, 8.5]
1003	BEM2027	2019	2	[5.5, 5, 6.5, 7]
1004	BEM2027	2019	2	[7, 5, 8, 6]
1005	ECM1400	2019	2	[9, 10, 10, 10]
1006	ECM1400	2019	2	[8, 8, 8, 9]
1007	ECM1400	2019	2	[5, 5, 6, 5]
1008	ECM1400	2019	2	[6, 4, 7, 9]
1009	ECM1400	2019	2	[10, 9, 8, 9]
1000	ECM1406	2020	1	[10, 10, 10]
1001	ECM1406	2020	1	[8, 7.5, 7.5]
1002	ECM1406	2020	1	[9, 7, 7]
1003	ECM1406	2020	1	[9, 8, 7]
1004	ECM1406	2020	1	[2, 7, 7]
1005	ECM1406	2020	1	[10, 10, 10]
1006	ECM1406	2020	1	[8, 7.5, 7.5]
1007	ECM1406	2020	1	[10, 10, 10]
1008	ECM1406	2020	1	[9, 8, 7]
1009	ECM1406	2020	1	[8, 9, 10]
1000	ECM1410	2020	1	[10, 9, 10]
1001	ECM1410	2020	1	[8.5, 9, 7.5]
1002	ECM1410	2020	1	[10, 10, 5.5]
1003	ECM1410	2020	1	[7, 7, 7]
1004	ECM1410	2020	1	[5, 6, 10]
1005	ECM0002	2020	2	[8, 9, 8]
1006	ECM0002	2020	2	[6.5, 9, 9.5]
1007	ECM0002	2020	2	[8.5, 10, 8.5]
1008	ECM0002	2020	2	[7.5, 8, 10]
1009	ECM0002	2020	2	[10, 6, 10]

3 Concepts & Basics

3.1 Explain the differences between a *compiled* and an *interpreted* programming language. (2 marks)

3.2 Why Java is *secure* and *platform-independent*? (2 marks)

3.3 Explain the differences and similarities between *variables* and *constants*. (2 marks)

3.4 What are the 8 primitive data types in Java? What are the differences between the types? For each type (separate lines), give a Java statement creating a variable and assigning a value. (6 marks)

3.5 What is casting? Explain the differences between *implicit* and *explicit* casting. (2 marks)

3.6 What is overflow? Please, give an example of overflow. (2 marks)

3.7 What are the four main features of Object-Oriented Programming? Please, give a definition of each key feature. (4 marks)

4 Submission

The CA requires electronic submission to the E-BART online system. Upload a compressed version of your files as a single file using the **zip** format by midday on the due date specified on the cover letter of this document. Note:

- The submitted file should follow this name convention (all lowercase): `ecm1410-`***`your_student_number`***`-ca1.zip`
- Source (*.java*) and bytecode (*.class*) files, after unzip, must be placed in the innermost folder of this structure (all lowercase):

  `ecm1410/`***`your_student_number`***`/ca1/src/`
- A PDF file with the enumerated answers 1–7 for question presented in Section 3. The PDF file must be named:

  `ecm1410/`***`your_student_number`***`/ca1/res/concepts_answers.pdf`
- A PDF file with a printout of all source files submitted, including **the line numbers** for each file independently. This PDF and any other file you may want to use in your implementation must be placed in:

  `ecm1410/`***`your_student_number`***`/ca1/res/`

5 Marking Criteria

This assessment will be marked using the following criteria:

- 80 marks for part 1 - coding.
- 20 marks for part 3 - concepts answers.

Table 4 details the criteria for each part of the submission, and it also includes the possible penalties.

Table 4: Marks scheme.

Criterion	**Description**	**Marks Available**
Questions	The degree to which the answers to the Concepts & Basics questions are correct and complete.	/20
Comments & annotations	The degree of quality and appropriateness of documentation comments, code comments and annotations.	/10
Java conventions	The degree of adherence to Java naming conventions and formatting. See lecture notes and e.g. `https://google.github.io/styleguide/javaguide.html`	/10
Operation	The degree to which the functionalities are correctly implemented.	/40
OO design	The degree to which the code is object-oriented, well structured and presented, with a coherent design and clear and appropriate management of object states, with well encapsulated objects, appropriate distribution of computational load across objects and appropriate use of types.	/20
Penalty	Failing to adhere submission filename conventions.	−5
Penalty	Disregard submission folder structure.	−5
Penalty	Non-submission of the printout lined code.	−5
Penalty	Inconsistency within the given input data.	−5
Feedback		