**University of Wolverhampton**

**Faculty of Science and Engineering**

**School of Mathematics and Computer Science**

# Final Module Assessment

|  |  |
| --- | --- |
| **Module** | 5CS019 Object Oriented Design and Programming |
| **Module Leader** | Hemanga Gautam |
| **Semester** | 1 |
| **Year** | 2020 |
|  |  |
| **Assessment** | Portfolio task |
| **% of module mark** | 80% |
| **Due Date** | You should aim to complete this work by \_\_\_\_\_\_\_\_. You are advised to stop working on this task at that point and move on to the next task. You should speak to your tutors during scheduled workshop sessions to get feedback on your work. The final portfolio submission deadline is \_\_\_\_\_\_\_\_\_. |
| **Hand-in – what?** | Zip or jar file containing program code for the tasks and other specified portfolio items. Other file formats that cannot easily be opened by your tutor will not be accepted. Program code should be Java source files only, and diagrams should be submitted as pdf documents. |
| **Hand-in- where?** | To be submitted via Canvas by 23:59 on the given date. |
| **Cheating, plagiarism, collusion** | Any evidence of these offences will not be tolerated. The work you submit must be your own work. |
| **Pass mark** | 40% is required overall to pass the module. Each portfolio task has an indicative weighting. However, the final grade will not be calculated entirely mechanically. A degree of academic judgement will be used to assess how well you have met the learning outcomes overall. You must attempt and submit all portfolio tasks. |
| **Method of retrieval** | Resit portfolio tasks will be in the summer resit period. These will not necessarily be the same as the original tasks, but variants of them. |
| **Collection of marked work** | Feedback returned through Canvas |

# Task Details

# Course Management System

Consider a course registration system for a college. Students are expected to register for the course of study of their choice. Additionally, an administrator at the college does occasionally add new courses to the list of available ones for students to have a wider selection of courses. It is possible for an administrator to cancel a course if it is no longer being offered. A cancelled course might be available at another time. However, when a course becomes unpopular, it is deleted altogether and cannot be reactivated at a later date. A deleted course is permanently removed from the list of courses at the college.

An example course of study is BSc in Computer Science. Courses are delivered on a semester basis, with each semester having 4 modules that students must study, a total of 8 modules altogether to complete a year of study.

An instructor is assigned up to 4 modules across the different levels of study. Students in levels 4 and 5 have no optional modules; however in level 6, students have two electives and 2 mandatory modules. To proceed between levels, a student must pass half of the modules for the current level. A pass mark constitutes a score of at least 40% on assessed work for the module.

**Users of the system**

**Students** – a student is a key stakeholder of this system as they choose courses, and enroll for modules on that course. At level 6, the student is also able to select from some optional courses during semesters 1 and 2. Students also are able to view which instructor(s) are teaching on the modules they are studying.

**Course administrator** – they can add new courses, add modules to a course, cancel a course, or delete a course altogether. They also are responsible for edits/amendments of course names, module names, etc. An administrator is able to generate a report/result slip for a student depicting modules done for the semester or year of study, and the marks and grades for each module. Based on this, there is decision appended to the result slip indicating whether a student will progress to next level of study or not.

**Instructor** – they are assigned modules to teach, can view what modules they are on, and the students registered on those modules. Instructors are able to add marks to each module they are responsible for teaching.

**Key features required of the system**

* Ability to amend courses and modules.
* Ability to amend instructor details or to remove instructors from modules.
* Ability to produce a result slip for each student indicating their grades on each module
* Persistence of data – all data is saved between program invocations

Additionally, there should be evidence of application of object oriented concepts, such as inheritance, encapsulation, object associations and polymorphism.

**Your task**

UML class diagram

You are to produce a UML class diagram of your solution. You are free to use any tool you like, e.g., MS Visio, StarUML. The construction of the class diagram can be iterative in nature, you can start with the more obvious classes such as Student, Module, Course, and then add to these as you go along.

Your classes in the class diagram should have comprehensive details (class name, attributes, and operations). For operations, you do not need to depict any getter and setter methods as these are rather obviously expected for most attributes. Relationships between classes should be clearly shown.

Application code

You are to implement your design using Java, making sure to exploit object oriented concepts in your implementation. In this task, these concepts are likely to be useful:

* Encapsulation and data hiding
* Inheritance and polymorphism

Other software development concepts that will be expected are:

* Data storage (in either text files or binary files)
* The use of collections

**The main areas to pay attention to are:**

* Design
* UML class diagram – with relevant details about each class
* Matching of class diagram and code
* Code
* functionality
* layout and style
* Use of OO concepts
* Program Output and results (outputs e.g., console output of current modules for the course, result slip)

**Detailed marking guide**

| **Grade** | **Design (UML Class Diagram)** | **Functionality and code style** | **Data Persistence** | **Evidence of Object Orientation** | **Results/output** |
| --- | --- | --- | --- | --- | --- |
| 0-19% | The design class diagram lacks some of the expected classes. Classes in the diagram do not have detail such as operations and attributes and the relationships do not reflect the described scenario. | The submitted code has compilation errors; functionality described in the brief is hardly provided.  Code is poorly structured with methods and attributes that do not seem intuitive.  Hardly any work done. | Application data, e.g., module details, staff details, etc. cannot be saved and retrieved. No output files. | OO concepts that lend themselves to this kind of problem aren’t utilized. No evidence of proper use of encapsulation, and class associations are hardly obvious. | No evidence provided in outputs produced by the submitted code. No output data in files or use of main method to show how code did work. |
| 20-29% | Level of detail in the class models is patchy, and the relationships among classes does not match the scenario provided.  There is some detail in some classes. | Some minimal functionality is achieved, but application in the whole does provide the features outlined in the brief.  Code style in parts is okay, but largely requires better structure. | An attempt on persistence but is inadequate; output files that do not keep requisite data. | Some evidence of OO, e.g., encapsulation but hardly any clarity on class associations, or inheritance | Some minimal results achieved, indicative of partially done work. |
| 30-39% | Requirements for 20-29% met.  Relationships among classes are close to what is outlined in the brief; some detail in classes | Requirements for 20-29% met.  Some functionality such as add module, cancel or delete module achieved. | Good attempt at saving some data into a file but requirements for 40% not met. | Requirements for 20-29% are met. | Requirements for 20-29% are met |
| 40-49% | Requirements for 30-39% met.  Relationships among classes are clear; multiplicity also considered; most classes have an acceptable level of detail | Requirements for 30-39% met.  Code structure is based on clear methods/functions;  Methods for most features are in place.  Code matches class diagram. | Basic data persistence achieved, but lacks ability to save edited data or retrieval and update and then save again. | A number of OO concepts evidently applied (e.g., encapsulation, inheritance, polymorphism) | Some results showing how application works, using main method to do inputs, process them and show console based results as well as writing to file |
| 50-59% | Requirements for 40 - 49% met.  Detailed class diagram, clear details and relationships. | Requirements for 40-49% met.  Code implementation matches class model; style is consistent and uses intuitive names for variables and methods.  Achieves many features, with some exception handling issues | Requirements for 40-49% met.  Data stored in files okay, retrieval and updates okay. | OO concepts represented well, though further refinements would have produced better code. | Results are showed using main method as well as output files. Console output also for result slip for a student |
| 60-69% | Requirements for 50-59% met.  Design model is comprehensive, and shows key methods and their return types, and parameters clearly | Requirements for 50-59% met.  Also, clear use of collections to store multiple objects. | Requirements for 50-59% met.  Use of I/O classes for data persistence is good. Clear output files | Requirements for 50-59% met.  The OO concepts are applied well in code and design. | Output/results showed in application execution in the main method as well as output files. Convincing behavior of application. |
| 70 - 84 | Requirements for 60-69% are met.  Design diagram has no issues | Code works, with no issues.  Collections used well.  Some aspects of exception handling could be better | Done well, no issues | Well done, no issues | Results are well presented  Application prints out a clear result slip |
| 85-100 | Design model has no issues, clear and well presented | Application handles exceptions well and provides useful messages to the user | Done well, no issues | Done well, no issues | Results are well presented.  Application prints out a clear result slip |

**You are also to accomplish the requirements outlined in the addendum.**