CNG 443: Introduction to Object-Oriented Programming Languages and Systems

Assignment 1: Online Learning Platform

2019-2020 Fall

Important notes:

- Your code will be tested by Moss or similar software against cheating attempts. Any cases suspected of plagiarism will result in a loss of grade and might result in further disciplinary actions.
- Please, submit your code on ODTUclass before the due. Other submission methods will not be accepted.
- A penalty of $5 \times lateDay^2$ is applied for the late submission of the assignments for at most three days.
- The due date for this assignment is **8th November 2019**, **23.59**.

Learning Outcomes:

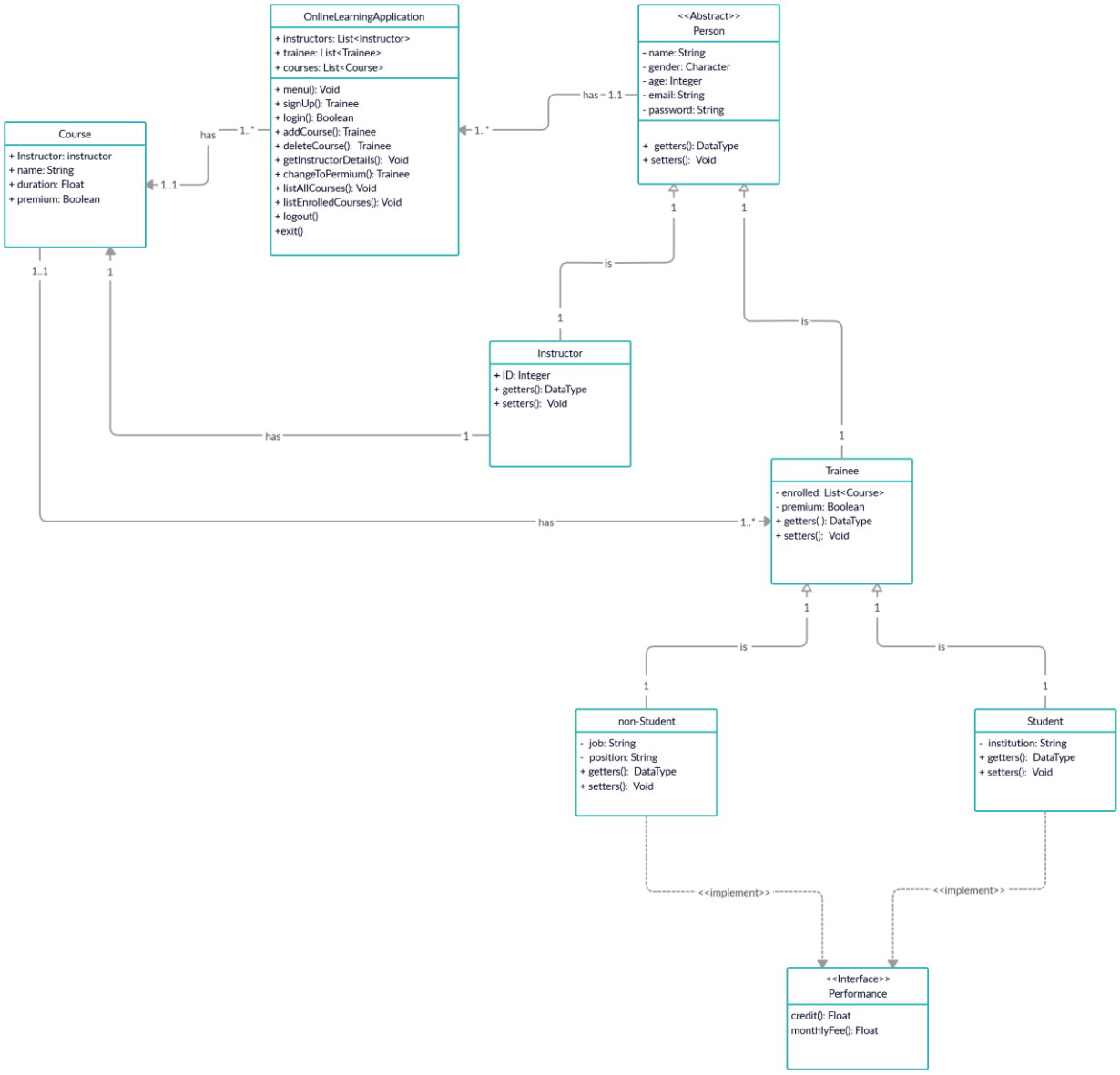
On successful completion of this assignment, a student will:

- Have used a UML class diagram to implement an application.
- Have practiced class hierarchy and the relevant design and implementation decisions.
- Have learned how to maintain different types of objects.
- Have practiced and used abstract classes, and interfaces.

Have learned how to create a package for an application.

Requirements

This assignment is about creating a small Java application to manage an online learning platform. It aims to maintain information about courses, trainees, instructors, and so on. The figure below shows a summary class diagram for this application.



The overall requirements are based on this class diagram which is also summarized below:

The main application called OnlineLearningApplication will be used to maintain information about this learning platform. This class will be the entry point to the application and will provide the overall interaction with the application. Since we still haven't yet covered Graphical User Interfaces (GUI) in this course, you have to implement this application as a command-line application. We also haven't covered how to make objects persistent or how to use and access a database. You have to maintain your objects within the application.

For further discussion about this, see the "Extra Requirements" section below. The required methods in OnlineLearningApplication are as follows:

- **menu()**: This function will maintain the menu that will be provided for the interaction with this application. You should design a menu (UI) for trainees only. It can be the main function of your JAVA program.
- **signUp()**: Using this function, a new trainee should be able to create an account. If the account is created successfully, the function should return a trainee object and show a message to the user demonstrating that the account is created successfully. If any of the attributes of a trainee remain null, the trainee object should be set to null, and the function should provide a message regarding the failure of the account registration. Technically, if the registration fails, a null trainee object should be returned.
- **login()**: Using the password provided during the registration, the trainee should be able to login to the account. Whether the login is successful or not, an appropriate message should be displayed to the user, and the function should return the corresponding Boolean value.
- **addCourse()**: After signing up, trainees should be able to enroll in one or more courses using the course name. Firstly, the course objects should be initialized and their names should be displayed to the trainees, if the trainee decides to see the list of courses (please check the description of **listAllCourses()** function). Secondly, the trainees will choose to add a course from the menu (after they log in) and enter the name of the course they wish to add. Finally, the name of the course, the trainee object and the course list will be given to this function as input parameters. Function decides whether the name exists in the list of courses. If the entered name by trainees belongs to a course in the list, **addCourse()** function will display a successful message, update the trainee object by adding the corresponding course, and in the end, it returns the trainee object.
- **deleteCourse()**: Trainees should be able to delete a chosen course. Hint: The procedure is almost the same as **addCourse()**.
- **getInstructorDetails()**: All the trainees should be able to search for the details of an instructor using his/her ID. The instructor objects should be defined and initialized in the compile-time. In other words, we do not enter the instructor's information in the runtime. Hence, there will be no menu (UI) for the instructor. After a trainee signs in, there should be an option to get an instructor details. By choosing this option, name of the instructors and their IDs will appear. **getInstructorDetails()** takes the ID entered by the trainee, and the list of instructors as well. If the ID belongs to an instructor, all his/her information such as full name, age, gender, and so on should be displayed to the trainee. If the ID does not belong to any of the instructors, a message should demonstrate the failure of the process to the user.
- **changeToPremium()**: Basically, there are two premium fields. One defined for the course which shows whether a course is free or not, and the other one belongs to the trainee class which illustrates

if a trainee has a premium account or not. Trainees should be able to change their account to premium so that they can see and add the premium courses as well as the free courses. Otherwise, they should not be able to observe or enroll in the premium courses. When the trainee logs in, there should be an option for changing the account to premium. For this purpose, a question should be displayed to the trainee (e.g. Would you like to change your account to premium? Yes, or No?). changeToPremium will take the current premium value, answer ("Yes" or "No"), and the trainee object. The current premium state should be checked. If it is already true, a message should be displayed, demonstrating that this trainee is already a premium member. In case the current premium status is false, if the answer is yes, the premium field for the trainee will be set to True and the trainee object will be returned. Otherwise, the premium field remains at the false state and a message will be displayed showing that the operation is canceled.

- **listAllCourses()**: After log in, the trainee should have an option to list all the courses. In addition to the list of courses, this function will take the premium status of the user as input. If the user is premium, the function will list all the courses, otherwise, it will only list the courses that premium status is false for them, which means the course is free.
- **listEnrolledCourses():** A trainee could have a list of enrolled courses. Therefore, this function, gets the trainee object as an input, and prints out all the courses with all the course enrolled by the trainee and their details such as name, duration, and so on.
- **logout()**: The user should be able to logout and return to the main menu (menu which as the exit, sign up and log in options).
- **exit():** Trainee should be able to exit from the main menu.
- **credit()**: Students enrolled in the premium courses should pay a fee. Therefore, this function gets the trainee object as an input, and count the number of premium courses for a trainee. Note that the implementation of credit for students and non-students will differ. For students, the number of premium courses they enrolled in should be multiplied to 0.8, and for the non-student it will be multiplied to 0.4. Finally, a float value will be returned
- **monthlyFee():** Let ***a*** be the multiplication result of the number of premium courses enrolled by the trainee to 10, and let ***b*** be the subtraction value of ***a*** and the output of **credit()** function. The function "**monthlyFee**" is supposed to calculate and return the value of $a - b$ for the students and $(a - b) * 2$ for the non-students.
- The required **getters** and **setters** mentioned in the UML class diagram should be implemented for the sake of encapsulation

NOTE: Please note that all the constant values such as coefficients should be defined as constant in your program.

Extra Requirements

Some extra requirements are listed below:

- In this course, we haven't yet covered how to use a database or how to make objects persistent; therefore, maintain objects such as courses, instructors, trainees etc. in collections, like ArrayList.
- Also, make sure that you have populated your collections with some initial data so that one can test your application easily. We expect you to create and initialize a list of course and instructor objects. Hence, when a trainee creates an account, in addition to observing the instructors' details, he/she will have access to a set of courses.
- In this course, we haven't yet covered Graphical User Interfaces (GUI), so please provide command-line based interaction. Note that we will have only one menu (UI) for the trainee, to be able to sign up, log in, add courses, etc.
- For each class, please decide what kind of constructors is required, what would be the access types of methods and fields. For each class, make sure that you have at least two different constructors. Unless necessary, do not use public fields. When you use private fields, make sure that you provide accessor and mutator methods.
- Pay attention to the overall design, layout, and presentation of your code.
- Make sure that all your methods and fields are commented properly including JavaDoc commands.
- Package all classes into a java-archive package called OnlineLearning.jar. With this package, one should be able to run your application by just typing "java -jar OnlineLearning.jar".

Assessment Criteria

This assignment will be marked as follow:

Aspect	Marks (Total 100)
All classes are implemented	10
All class hierarchies are implemented	10
All interfaces are implemented and used	10
For all classes constructors are properly implemented.	10
For all classes all required data fields are implemented	10
For all classes all required methods are implemented	10
All the methods implemented and works properly	30
Package Structure and Jar for Invoking the application	10

Grading Policy

In order to get full mark, your class should have a constructor with full parameters and JavaDoc comments. The following grading scheme will also be used for the requested methods.

Fully Working	0.2
Appropriate reuse of other code	0.2
Good coding style	0.2
Good Javadoc comments	0.4