SANCHEZ Pablo, THERET Guillaume, RATOUIT Pablo, ROUX Gwendal

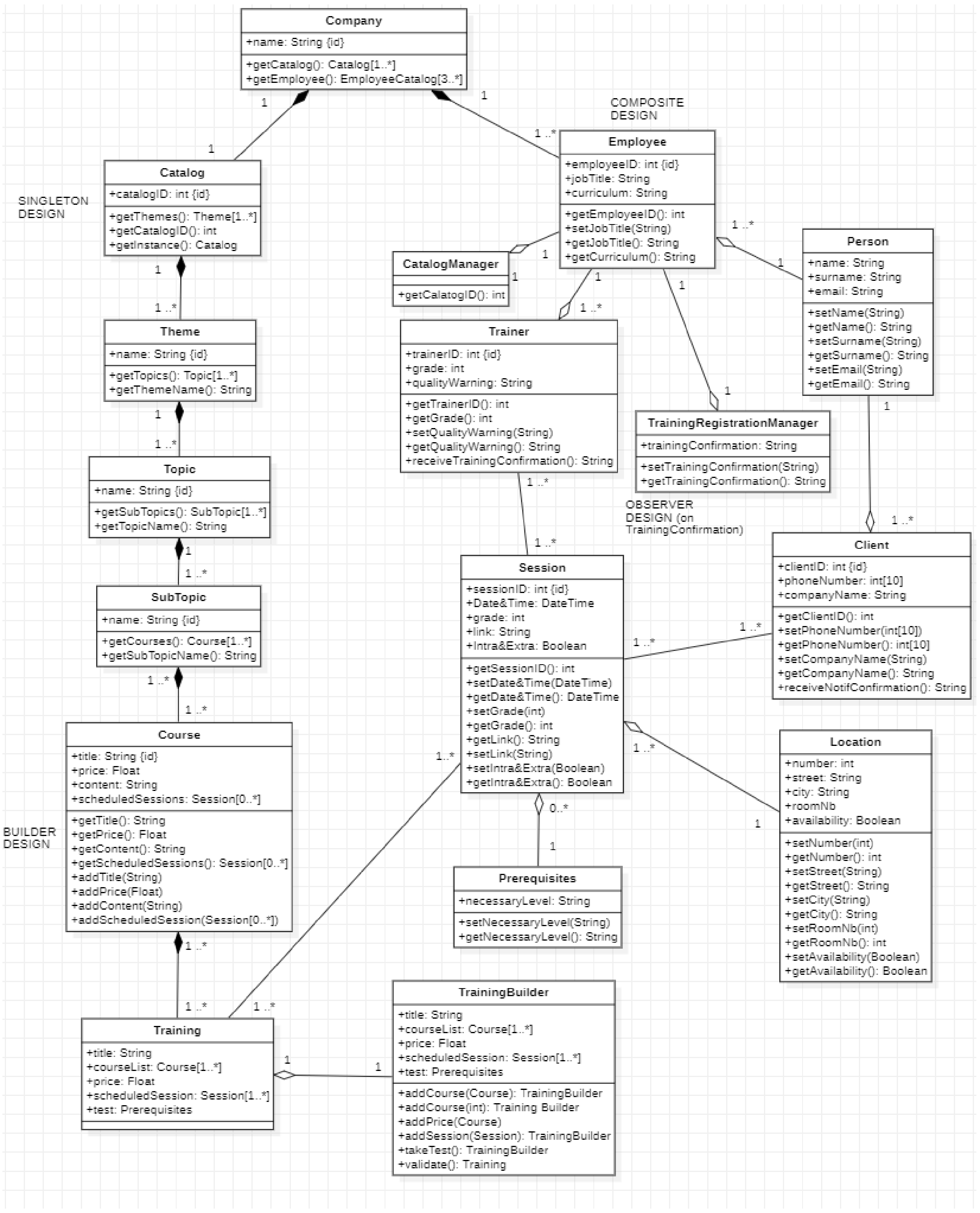
M1 SE2

Software Design and Modelling

Project

UML Diagrams:

Class diagram:



First, we have the company class, to define the unique object of the company from the scenario. This company is made of several employees, as well as one catalog.

The catalog is made of several themes. Each theme is divided into topics. Topics are made of several subtopics, and each subtopic divided into courses. Each course is passed through one or several training, that as built with TrainingBuilder. Trainings happen during Sessions, and each session has prerequisites and a unique location.

The class person defines a human being by its full name and email. A person can be a client, in which case he can participate to sessions, or an employee. An employee can be a CatalogManager, having direct access to the catalog, a TrainingRegistrationManager, sending updates on trainings statuts, or a Trainer for a Session.

4 Design Patterns:

We decided to implement different Design Patterns within the project that suited the situations:

SINGLETON - The singleton is used for a class whose instance is unique. The class method creates an instance if it doesn't already exist or returns the instance. We use it for the Catalog class, since the Catalog described in the subject contains the various themes and topics available. In fact, there will never be several instances of this class, as there is only one catalog for the company.

In our case, the SINGLETON design is used in the Catalog class.

OBSERVER – The Observer is a behavioral design pattern that allows methods to be executed according to certain predefined events. We've assigned it to the TrainingRegistrationManager class, since its role is to create a session once enough customers have registered. In addition, it can also cancel a session due to a no-show. To do this, it observes instances of the Session class and acts accordingly. For its part, the Session class notifies the observer if a client registers for the session or withdraws.

In our case, the BUILDER design is used in the Course class.

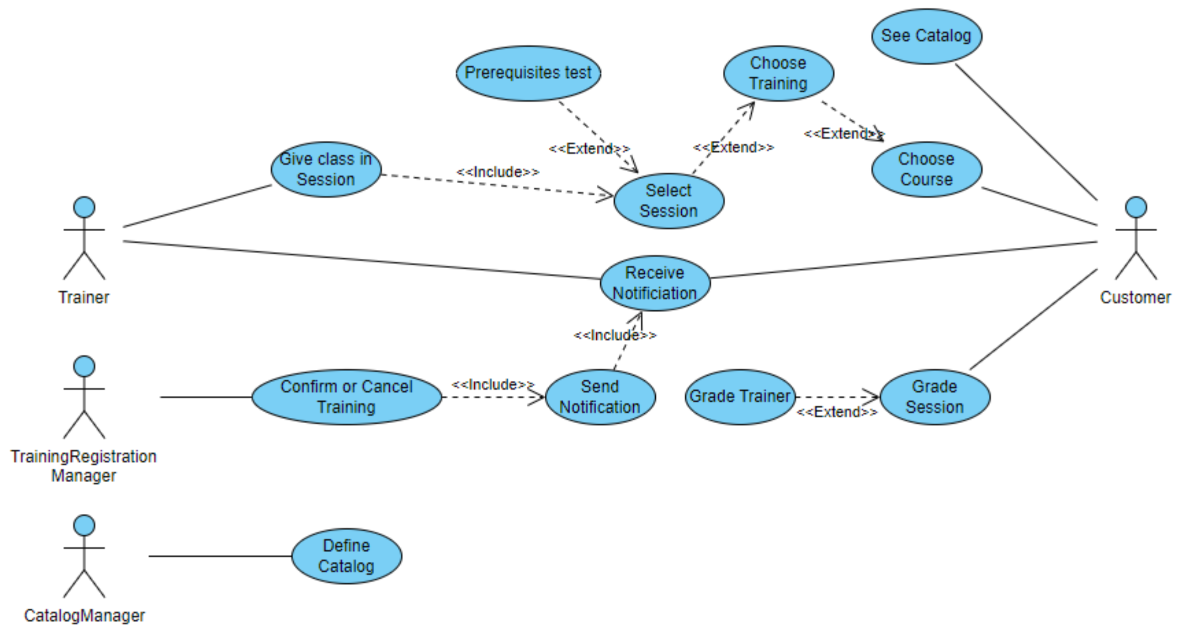
BUILDER – The Builder design pattern is a creation pattern that defines a way to build an instance of the Course class. If a class has many arguments, the Builder allows you to create an instance and modify only the useful parameters using functions rather than declarations. As Courses are customizable by the customer and depend on the level of participants, it's easier to use this design pattern.

In our case, the OBSERVER design is used in the TrainingRegistrationManager, Trainer, and Client classes, with the methods setTrainingConfirmation, getTrainingConfirmation, and receiveTrainingConfirmation, as well as attribute trainingConfirmation.

COMPOSITE - The Composite design pattern is used to define classes that are special cases of a more general class. Each class implements the same interface and has the same internal structure but may have methods that perform custom tasks. We've applied it to the Employee class, as the subject describes various contact persons, who will therefore be classes composed from Employee.

In our case, the COMPOSITE design is used in the Employee, CatalogManager, Trainer, and TrainingREgistrationManager classes.

Use Case diagram:



In our use case diagram, we understand that CatalogManager has a small role, he defines what is inside the catalog, meaning that he has small interaction with the rest.

Most of the interactions are between Trainer and Customer. The Customer can read the catalog, and choose a course. If a course is chosen, then the Customer chooses trainings to achiev, and defines sessions to participate in. Each session implies for the customer to pass a prerequisites test. After a session, a Customer is expected to grade the session, implying a change in the mean grade of the trainer. If a session is selected by the Customer, a Trainer is assigned to give the class, and to be graded.

Finally, the TrainingRegistrationManager has to confirm of cancel a training session, sending a notification to both Trainer and Customer, who read it.

Code :

Available here : GitHub - Bllopp/Software_Design_Project

GRASP and SOLID:

Both Software Designs are respected in our code.