Software Design Patterns

CAT 2: Design and Responsibilities Assignment Patterns

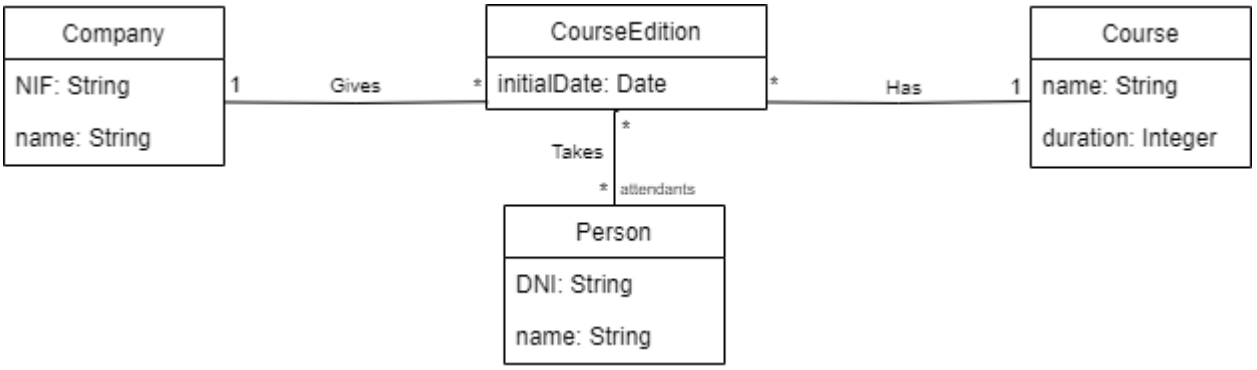
(evaluated out of 50 points)

Question 1 (10 points)

Statement

We are designing a system to manage the editions of courses taught by different companies. Companies have a NIF that identifies them and their name is also recorded. The courses have a name that identifies them and a duration. Course editions are made for a specific course and are taught by a company. Course editions have a start date and their attendees. Below is the static diagram of the system.

Static diagram



Integrity constrains

- A company is identified by its NIF.
- A course is identified by its name.
- A person is identified by his name.

- An edition of a course is identified by the name of the course and the NIF of the company that teaches it.
- The duration of a course must be a positive value.

We want to add different functionalities to the system. These functionalities will be provided through the following operations:

- Creation of an edition of a course. This operation must allow you to register an edition of a course for a course and a company. It must also initialize the attribute initialDate. We can assume that when the course is created we still have no attendees.
- Add an assistant to a course edition. This operation allows you to add a person as an assistant to an edition of a course.
- Get editions of any course with a start date after a date. This operation must obtain the name of all courses that have editions after the date d and it will be invoked from the presentation layer.

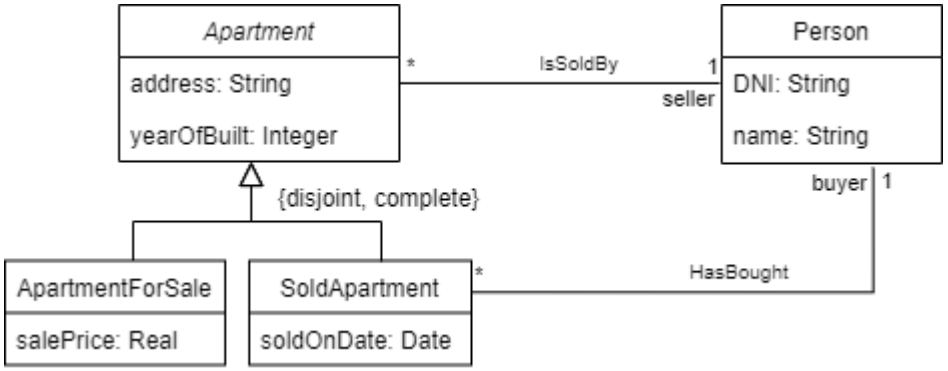
Questions:

a) (5 points) Justifies which responsibilities assignment patterns should be applied to define these operations.
b) (5 points) Provide the class diagram with these new operations. You must fully define the operation signature (operation name, parameters and, if necessary, operation return). There is no need to provide the pseudocode or sequence diagram of the operations.

Question 2 (15 points)

An estate administration has asked us to design a system to manage the current sales of flats. The address of the flats is recorded, which identifies them, the year of construction and the seller. Flats can be for sale or sold. For the flats for sale we know the selling price and for the sold flats we know the buyer and the date of purchase. The system records the people who own or buy the flats. For these people, we know the ID that identifies them and their name. Below is the static diagram of the system.

Static diagram



Integrity constrains

- An apartment is identified by its address.
- A person is identified by his name.
- salePrice>0
- The buyer and seller of a flat cannot be the same.

We want to design the operation *sellApartment(p:Person,saleDate:Date)* of class *Apartment*. This operation does the following:

- Throw the exception *apartmentAlreadySold* if the apartment was already sold or the exception *alreadyOwner* If the person *p* (which is the one who buys the apartment) is the seller. Otherwise,
- The apartment becomes sold and the person is assigned as the buyer *p*.
- The attribute soldOnDate gets value saleDate .

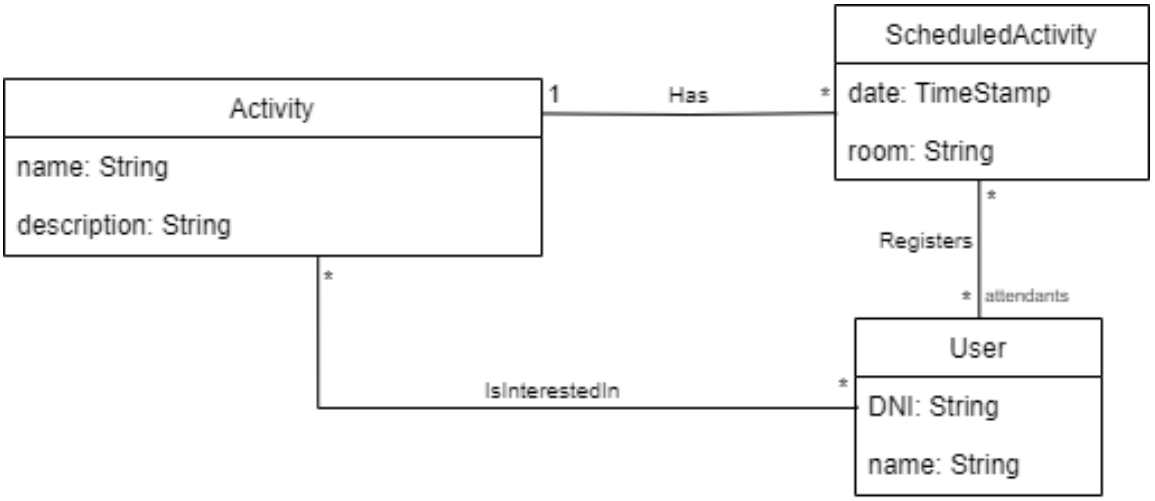
Questions:

a) (3 points) Justify which design pattern you would apply.
b) (6 points) Draw the sequence diagram of the operation *sellApartment(p:Person,saleDate:Date)* of class *Apartment*.
c) (6 points) Provide the class diagram with the application of the pattern identified in section a) with all the operations you need to define for the operation *sellApartment(p:Person,saleDate:Date)* (you must fully define the signature of all operations with their name, parameters and, if necessary, return of the operation).

Note: To throw exceptions, you can indicate this with a note in the sequence diagram.

Question 3 (15 points)

The app to manage the information of a fitness club that was presented at the CAT1 represents the information of the activities (pilates, cycling, etc...), the planned activities, which are activities that have been planned on a date, specific time and room (for example, the club has planned two pilates activities on September 30, one from 12 to 1:30 in the morning for 15 people and another from 17 to 18 in the afternoon for 20 people, in the Toning room), and users with their favorite activities and their registrations for planned activities. Below is a fragment of the class diagram of this app.



Integrity constrains

- An activity is identified by its name.
- A planned activity is identified by the activity and the date.
- A user is identified by his dni.
- There cannot be overlapping scheduled activities in the same room.

We want the app to enroll users in planned activities when these are added to an activity the user is interested in, minimizing the coupling between the User class and the Activity class.

Questions:

a) (3 points) Justify which design pattern you would apply.
b) (6 points) Make the pseudocode of the operation *addScheduledActivity(sa:ScheduledAcivity)* of class *Activity* and of all the operations that are invoked from this operation. You must also include the pseudocode of all classes that are involved in the definition of the operation.You don't have to worry if when you register a user for a scheduled activity, they already have another overlapping scheduled activity.

c) (6 points) Provide the class diagram with the application of the pattern identified in section a) with all the operations you need to define for the operation *addScheduledActivity(sa:ScheduledAcivity)* of class *Activity* (you must fully define the signature of all operations with their name, parameters and, if necessary, return of the operation).

Question 4 (10 points)

The app from the previous question must record the price of the monthly fee currently charged to users. All users of the system must pay the same monthly to use the services.

Questions:

a) (2 points) Justify which design pattern you would apply to record this fee.
b) (4 points) Provides the class diagram resulting from applying this pattern (only the application part of the pattern).
c) (4 points) Write the pseudocode of the class diagram that you propose in the previous section.