Name: Daniel Jesús del Río Cerpa

Exercise 1

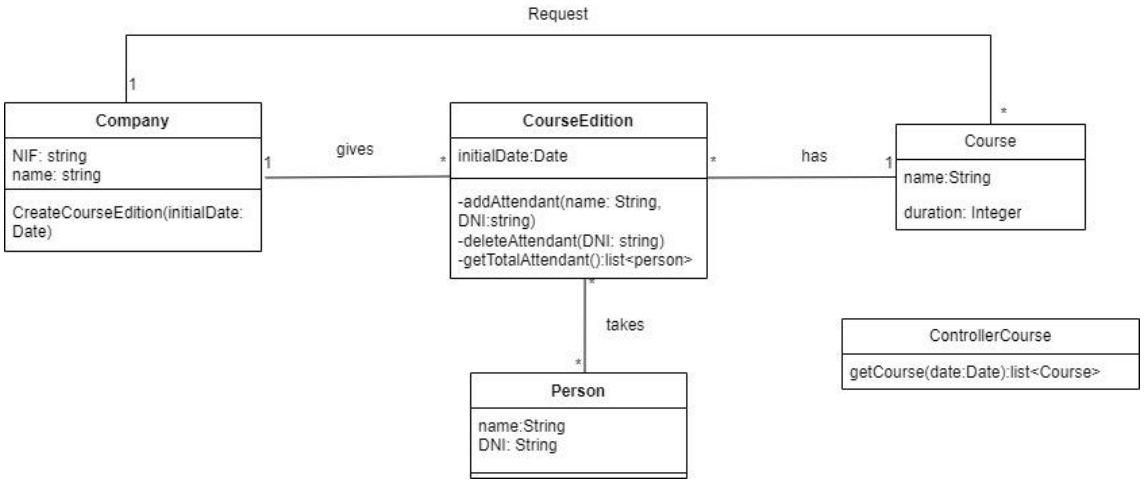
a)

First case: I consider that the most appropriate pattern would be the "**Creator**", since it is a pattern in charge of creating instances and initializing objects. In this case, I would assign this function to the Company class, since it is the company that manages and organizes the editions of the courses and, therefore, Company must create these instances based on the need.

Second case: In this case, I would use the "**Expert"** pattern on the CourseEdition class, since it is the class that would have the necessary information to manage the list of attendees, therefore, we assign it the responsibility of processing that data.

Third case: I think the "**Controller**" pattern would be optimal, we create a separate class to manage this event. There are different types of controllers, but in this case, I think that the facade controller would be the most optimal, since I consider that there are not many different events.

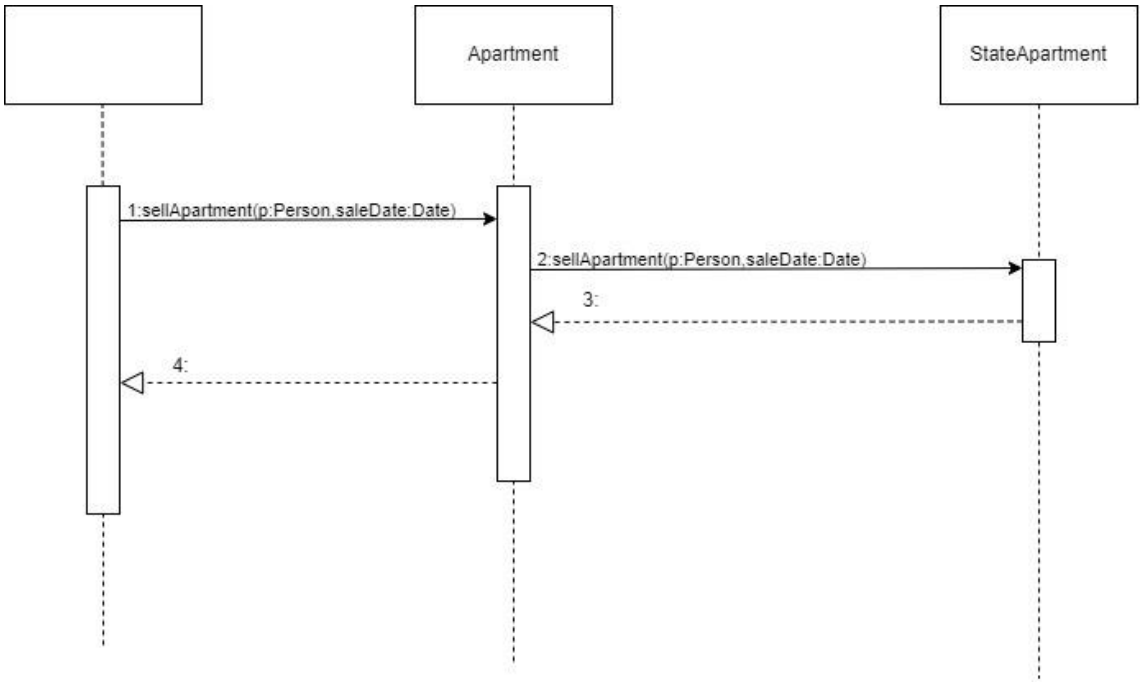
b)

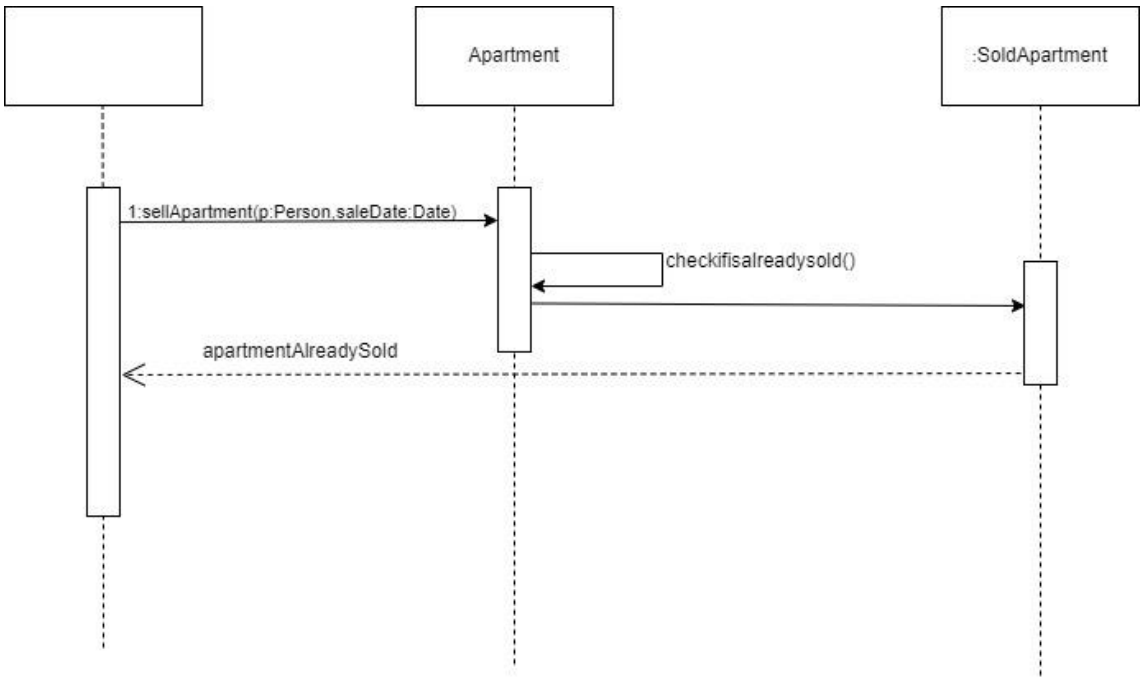


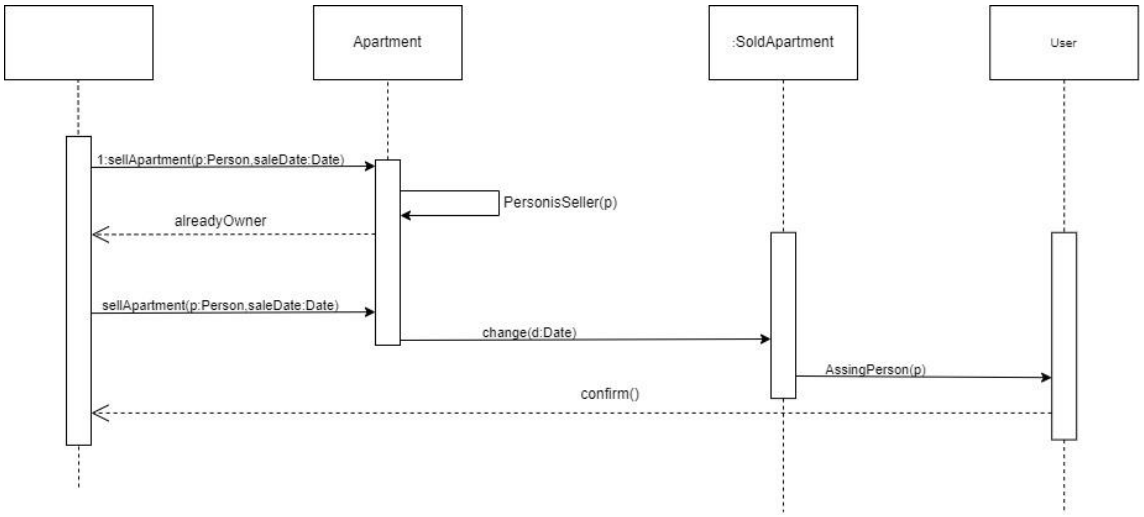
Exercise 2

a) I consider that the optimal design pattern would be the "**State**" pattern, since it is a case in which we find conditional blocks, where the state of the object can change the operation and, therefore, change its behavior. In this case, the apartment may be sold or for sale.

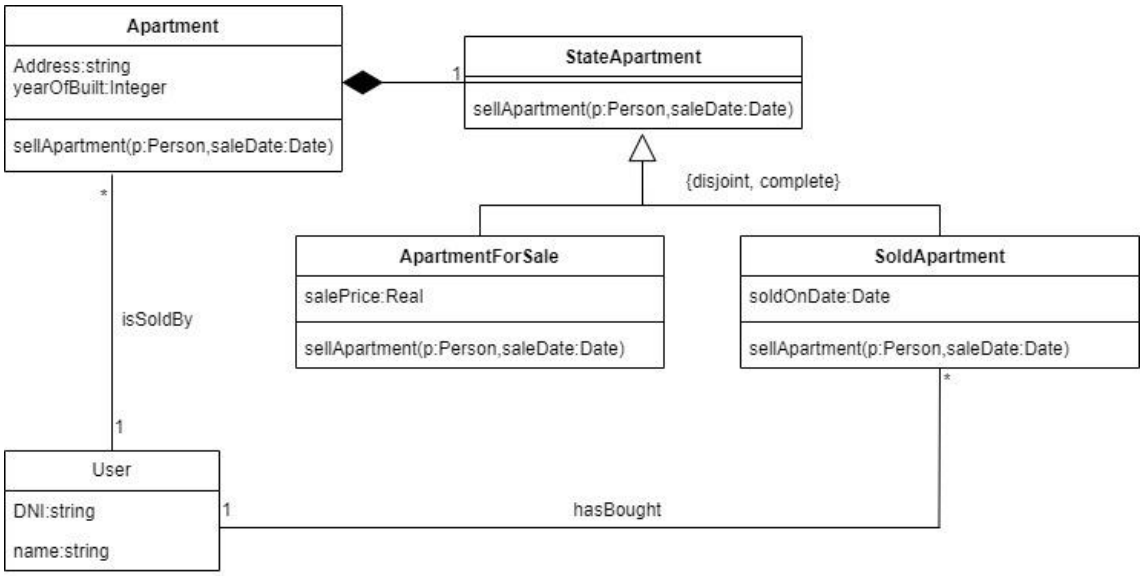
b)


Apartment
StateApartment
1:sellApartment(p:Person,saleDate:Date)
2:sellApartment(p:Person,saleDate:Date)
3:
4:


Apartment
:SoldApartment
1:sellApartment(p:Person,saleDate:Date)
checkifisalreadysold()
apartmentAlreadySold



c)



Exercise 3

a) I consider that the most optimal pattern would be the Observer. This pattern allows defining a behavioural mechanism to notify several objects of an event that is occurring in the object it observes. Thus, as different users will want to sign up for different Schedule Activities, we could implement the "**Observer**" pattern through an interface that collects these different states.

b)

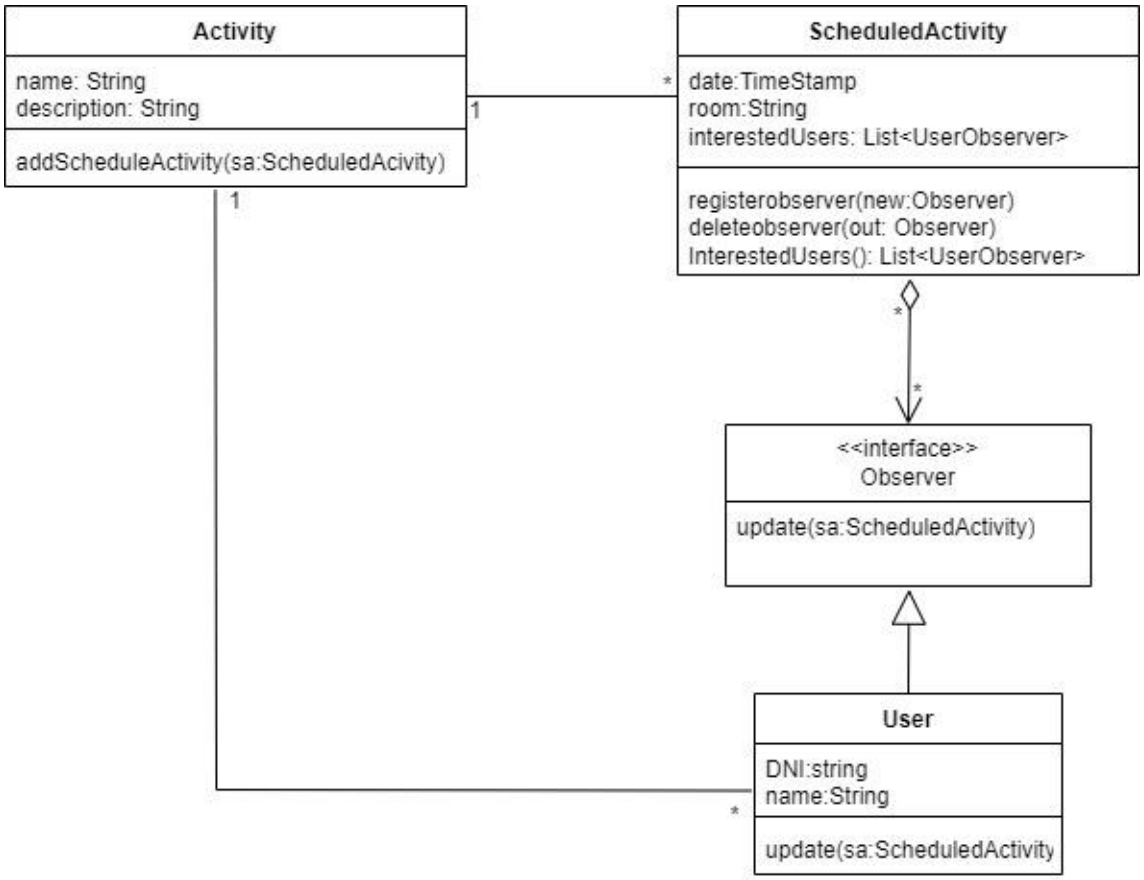
```
public interface Observer {

update(sa:ScheduleActivity)

}
```

```
public class ScheduleActivity {
protected List<Observer> interestedUsers;
public void registerObserver(Observer o) {
        this.interestedUsers.RegisterObserver(o);
}
public void deleteObserver(Observer o) {
        this.interestedUsers.deleteObserver(o);
}
Class Activity{
Public void addScheduleActivity(sa: ScheduleActivity){
        This.scheduleActivity.add(sa)
        }
}
class User Implements observers{
        Public void update(sa: ScheduleActivity){
        This.update(sa)
        }
}
```

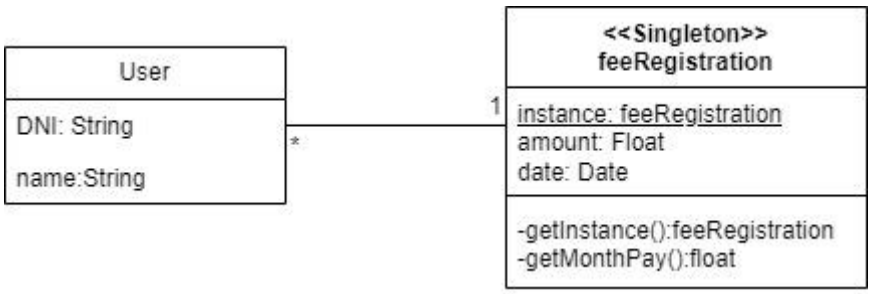
c)



Exercise 4

a) I consider that the most appropriate pattern is the "**Singleton**". It is a design pattern that allows us to ensure that a class has a single instance, providing a global access point. It is a pattern that is normally used to control access to a shared resource (databases, files, etc.). Therefore, if we use this pattern we can avoid duplication and create a global access point.

b)



c)

```
Public Class feeRegistration{

        Private feeRegistration instance=null;
```

```
        Private float amount;
        Private date Date;
        Public getInstance{
                if (instance == null) {
                instance = new feeRegistration ();
    }
return instance;
}
        Public getMonthPay(){
                This.amount;
                Return amount;
        }
}
```