Exercise 1:

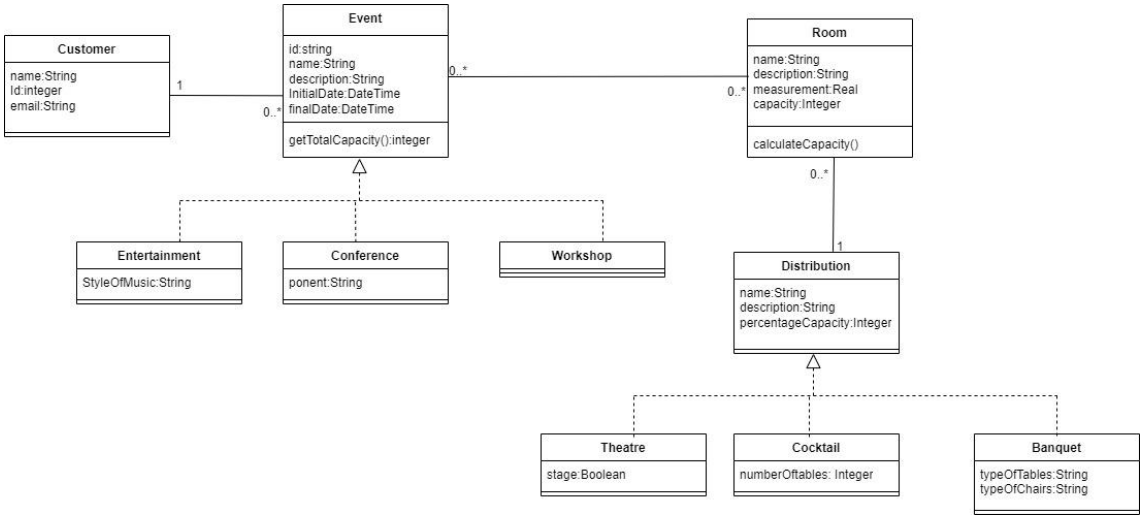
a)Law of Demeter principle is violated. This principle indicates that object operation should only use: The operations of the same object, the associated objects (or its attributes), the objects that the operation receives as a parameter, the objects that the operation create. We can see that it is violated because the Event class is accessing directly with the Distribution class and Room class.

b) This principle is violated. This principle told us that whenever possible we should avoid duplication of information and responsibilities. As we can see, There is a duplication of code in the operation to calculate the capacity. It is negative because repetition of the code can cause errors when starting it, in the same way that if there are errors in the code, we can modify it only once, instead of several times.

c) I also consider that this principle is violated. This is because modifying the rooms and layout can directly affect the Event class. The violation of the law of demeter tells us that there is a violation of the OCP. When accessing methods from the Room and Distribution classes, the class is open to modification but is not completely closed, since extending one of these classes can have consequences on Event.

Exercise 2:

a)



b)

To solve the problem, I have created a method to calculate the capacity within Room. This method has the capacity of the Room Class and can acquire the capacity percentage information from Distribution and then calculate the total capacity. Once this is done, the Event method will be able to access the result and display the final capacity.

```
public abstract class Event

{

private Integer id;

 private String name;
```

```
private String description;
private Datetime initialDate;
 private Datetime finalDate;
 private Customer customer;
 private List rooms;
public Integer getTotalCapacity()
{
method getTotalCapacity(): Integer{
        totalCapacity = 0
        foreach Room r in rooms:
           totalCapacity += r.calculateCapacity()
}
        return totalCapacity

}
}
Public class Room
{
private String name;
private String description;
private Real Measurement
private Integer capacity
method calculateCapacity(): Integer
{
       roomCapacity =capacity
       percentCapacity = distribution.getCapacityPercentage() / 100

       if
 {
distribution.hasStage():
          roomCapacity -= 10
```

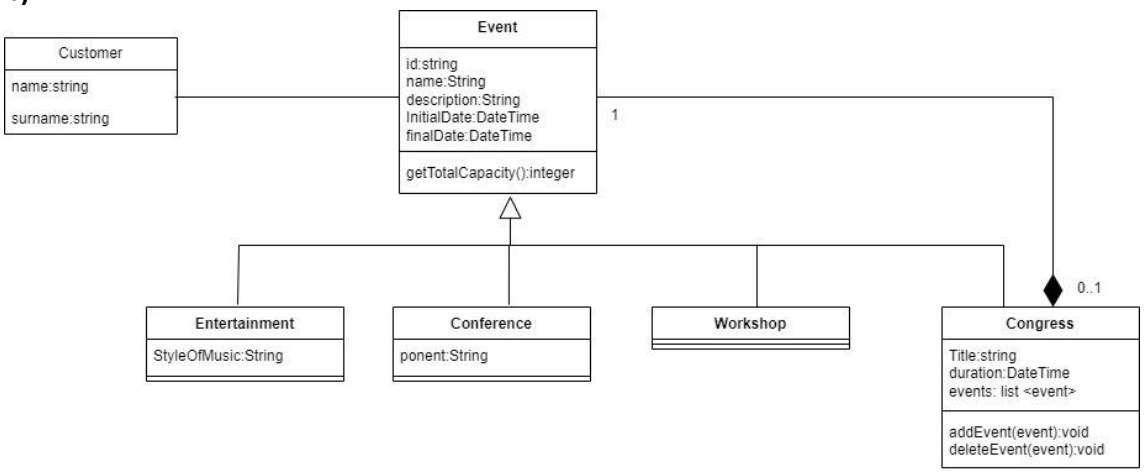
```
}
    return roomCapacity * percentCapacity
}
```

Exercise 3-

a)

I think that the composite pattern is the most suitable for this case. I consider it to be the best because it will allow us to manipulate groupings of elements and simple elements in a uniform way. It is a pattern that allows you to compose objects in a tree structure and work with these structures individually. This pattern is characterized by three large components: Component (the abstract class or interface), Leaf (individual element) and Composite (an element that contains sub-elements). In this case, we will need a container that groups several elements to be able to operate (Congress).

b)



Integrity Contraints:

A congress can not have more tan three conferences in the same event.

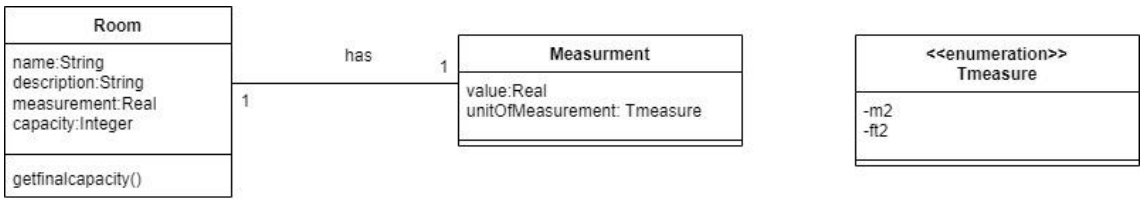
Exercise 4:

a) I consider that the quantity pattern would be appropriate for this situation. If customers are having difficulty understanding the distance of the rooms, it means that the solution for expressing the measurement is very poor. And as written in the learning resources, this pattern can be used in the following cases:

- to make explicit the unit of measure;
- to use different measurement units;
- to be able to convert between units in a simple way

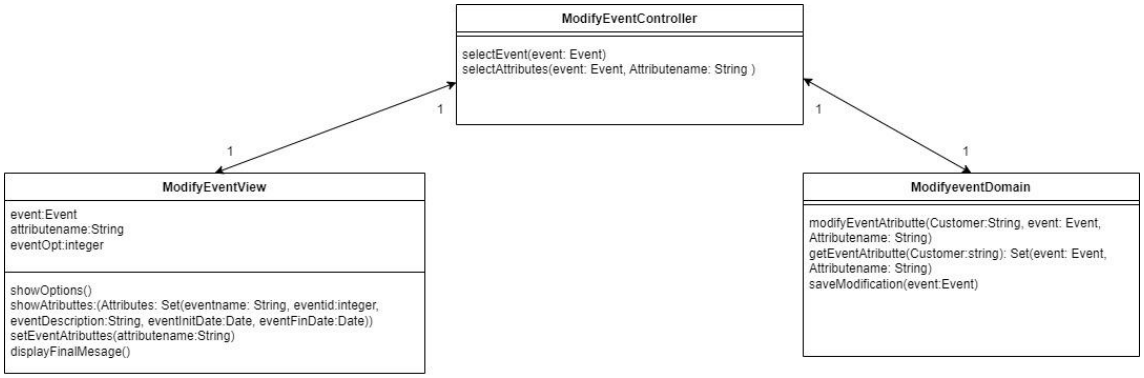
In this case, we can see that the unit of measurement would not be clear in the analysis, since it would be implicit in the attribute. Therefore, I would extract the measurement attribute and convert it into a class and then place an enum type attribute that can have the values of ft2 or m2.

b)



Exercise 5:

a)In this case, ModifyEvent represents the view, ModifyeEventController is the controller and ModifyEventDomain is the Model, since it is the class which has business logic.



ModifyEventView:

-ShowOptions():Shows the different functionalities available.

- showAtributtes:(Attributes: Set(eventname: String, eventid:integer, eventDescription:String, eventInitDate:Date, eventFinDate:Date)): Shows the different attributes that the event has through the screen.

- setEventAtributtes(attributename:String): Register the attribute we want to modify

-displayFinalMessage: Show success message: Successfully registered.

ModifyeEventController:

-selectEvent():Detects the event that the customer has chosen the option to modify some attribute of the event.

- selectAttributes(event: Event, Attributename: String): With this operation, the system detects the attribute within the event that the customer has selected and invokes the operation to modify it.

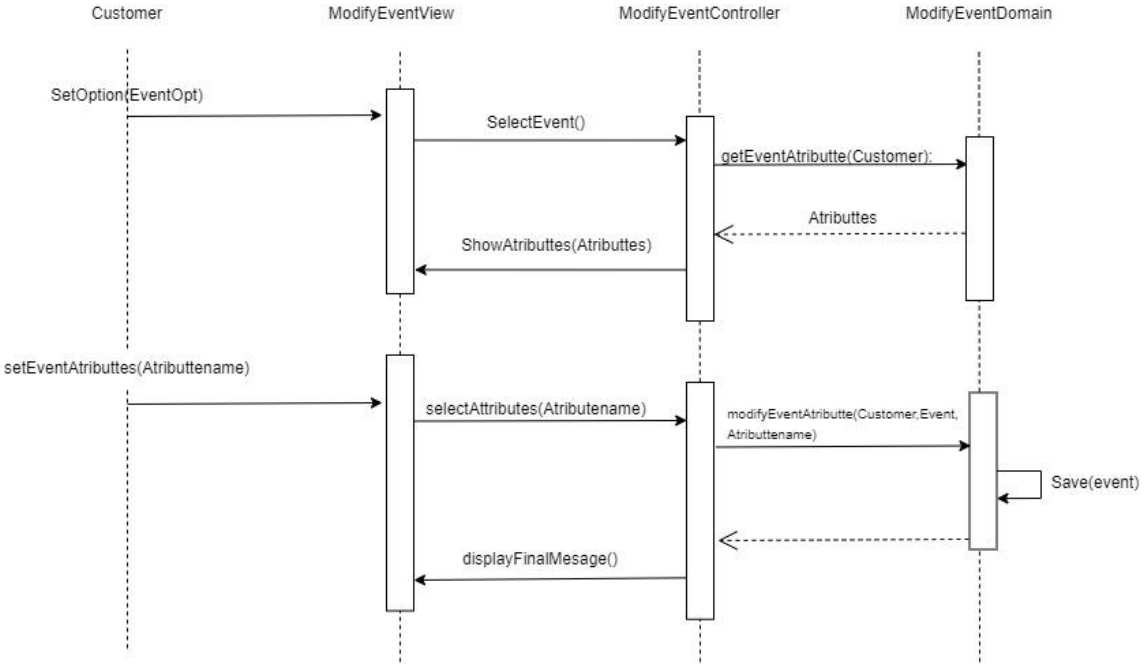
ModifyEventDomain:

- getEventAtributte(Customer:string): Set(event: Event, Attributename: String)With this function you obtain the event and the attributes to modify.

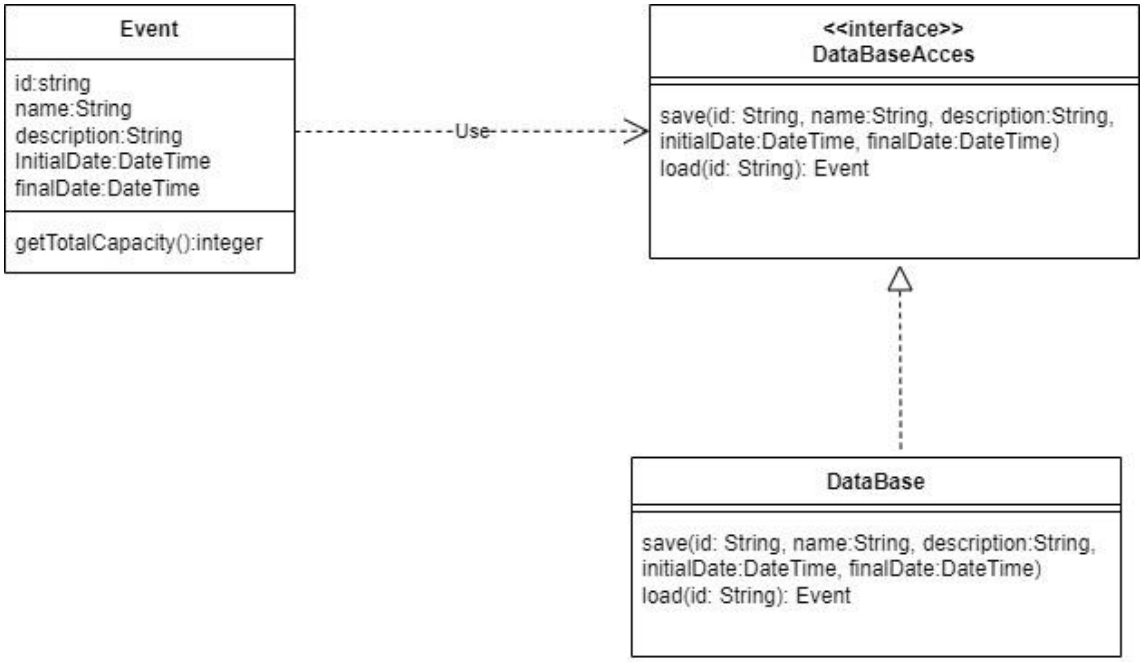
- modifyEventAtributte(Customer:String, event: Event, Attributename: String): Modifies the selected attribute of the evento

-save(event): update Event data which were modified.

b)



Exercise 6:



Exercise 7:

a) As the learning resources indicate, the classes in this layer respond to requests to read data, consult databases, etc. It is a layer that provides persistent data for business logic. In this case, the class that would enter here would be Database .

b) The domain layer is responsible for encapsulating the business logic, it is responsible for materializing concepts of our business and where it works. In this layer, the concepts and objects that are intrinsic to the application domain are defined, encapsulating its logic and behavior. Therefore, we can affirm that all the classes from exercise 2 are included in this layer, since they are necessary elements for the operation of our application. If these objects are not represented, there is nothing to solve. In the same way, ModifyEventDomain would also be within this layer, since it performs main functions of the application.

c) The responsibility of the classes in this layer would be to collect user input, call the domain layer, and display the results to the users. In this case, the classes that would be defined in this layer would be: ModifyEventView and ModifyEventController. They are the classes in managing the data received by the customer and showing the results obtained.