NAME: Daniel Jesús del Río Cerpa

Software Design Patterns

Classroom 1

Bachelor´s degree in techniques of software development.

1 a)

I consider that the principle of Interface Segregation (ISP) is being violated. If we pay attention to the learning resources of the subject, it tells us the following:

“Clients would not have to depend on operations they do not use.”

According to this principle, we must disintegrate the interfaces in a detailed way, in such a way that clients implement only the necessary methods that they are going to use, since if we use a very thick interface, we can decompose clients that don’t use certain methods.

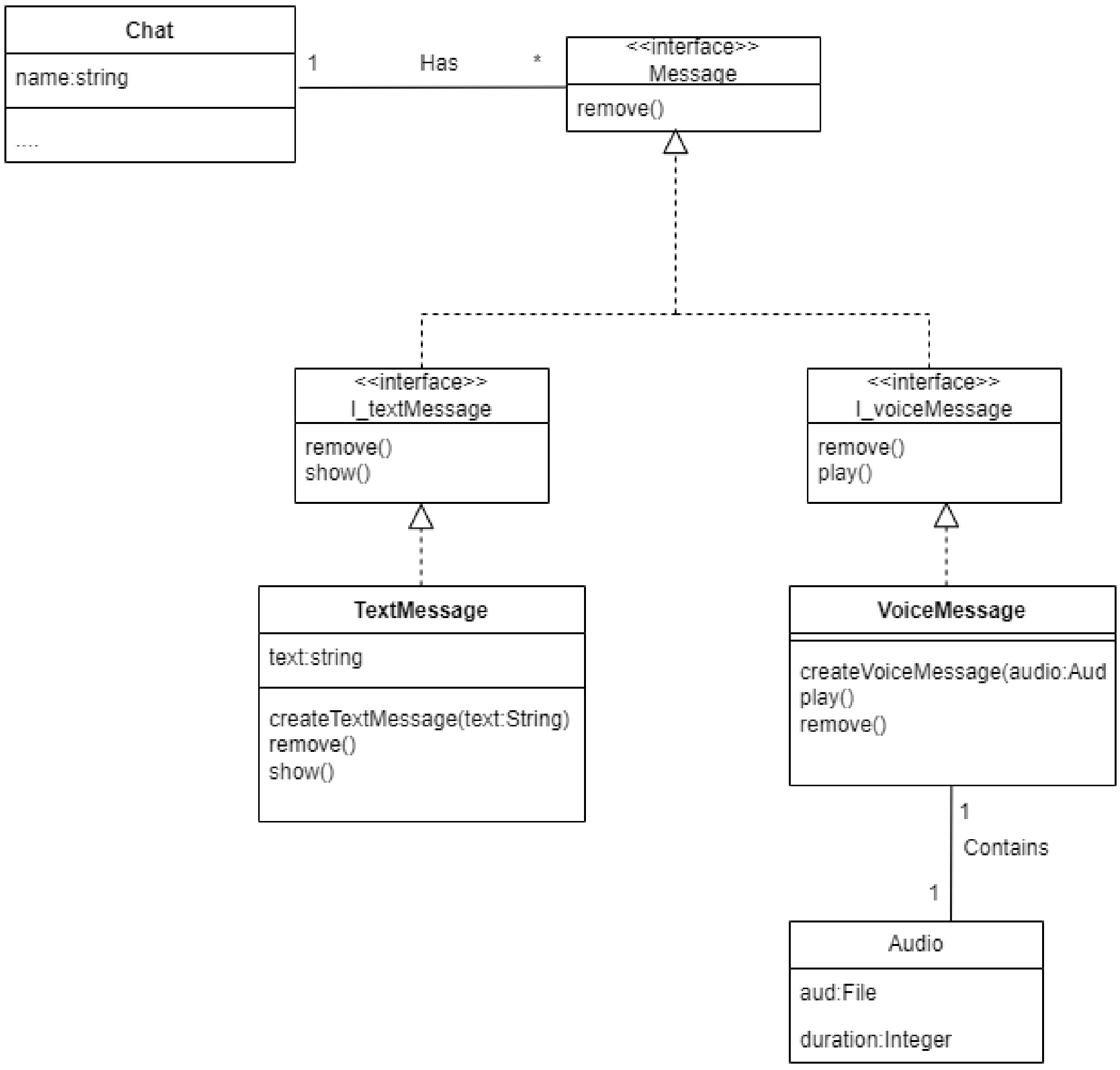
If we look at the diagram, the play() and show() actions are not used in both cases. Play() is used only with the VoiceMessage class and the show() action only with the TextMessage class. As the Integrity Contraints indicate:

TextMessage::play(): does nothing

VoiceMessage::show(): does nothing

Ergo, The principle of interface segregation is being violated, since the interface is implementing methods that are not going to be executed by all classes. The solution that I propose later consists of creating two interfaces for voice messages and text messages, which will be interfaces that inherit the remove() action from the main interface. Next, I present my model (1b).

1b)

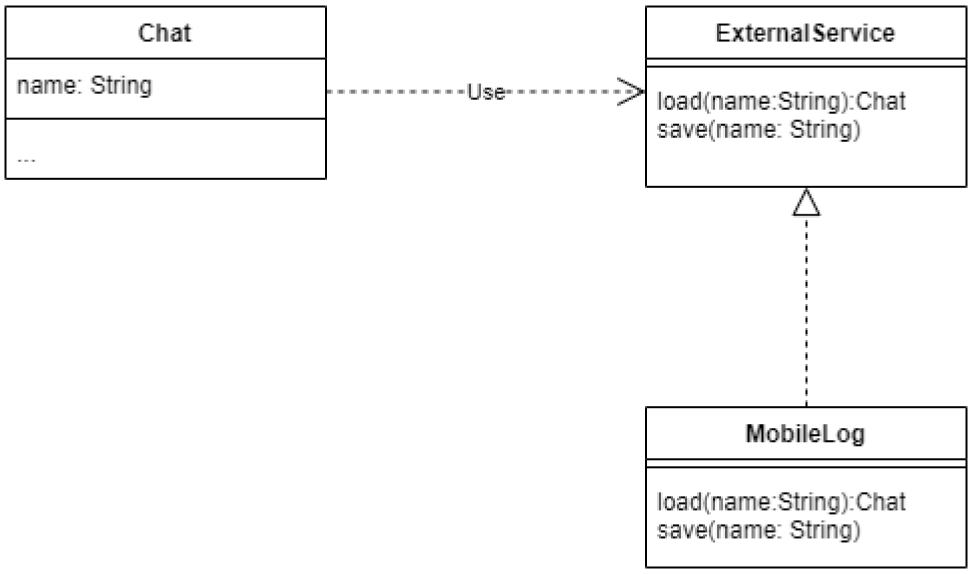

Chat
name:string
....
1
Has
*
<<interface>>
Message
remove()
<<interface>>
I_textMessage
remove()
show()
<<interface>>
I_voiceMessage
remove()
play()
TextMessage
text:string
createTextMessage(text:String)
remove()
show()
VoiceMessage
createVoiceMessage(audio:Aud
play()
remove()
1
Contains
1
Audio
aud:File
duration:Integer

2 a)

In this case, I consider that the Dependency Inversion Principle is being violated. This principle tells us that high-level classes should not depend on low-level classes. But both must depend on abstractions. Abstractions should not depend on details, but details should depend on abstractions. In this way, we prevent this coupling from having an impact at the level of abstraction at which it occurs.

In this case the Chat class depends on the MobileLog class, which is a low-level class. As the statement indicates, if something happens to Mobilelog, this will have an impact on Chat, since the loss of the mobile phone means the loss of all chats. In this situation, I have used the External file class as a high-level class with the load and save actions. This class is used by chat to save and load messages, and I make the MobileLog class acquire those functions later, although mobileLog is still a low-level class.

2b)

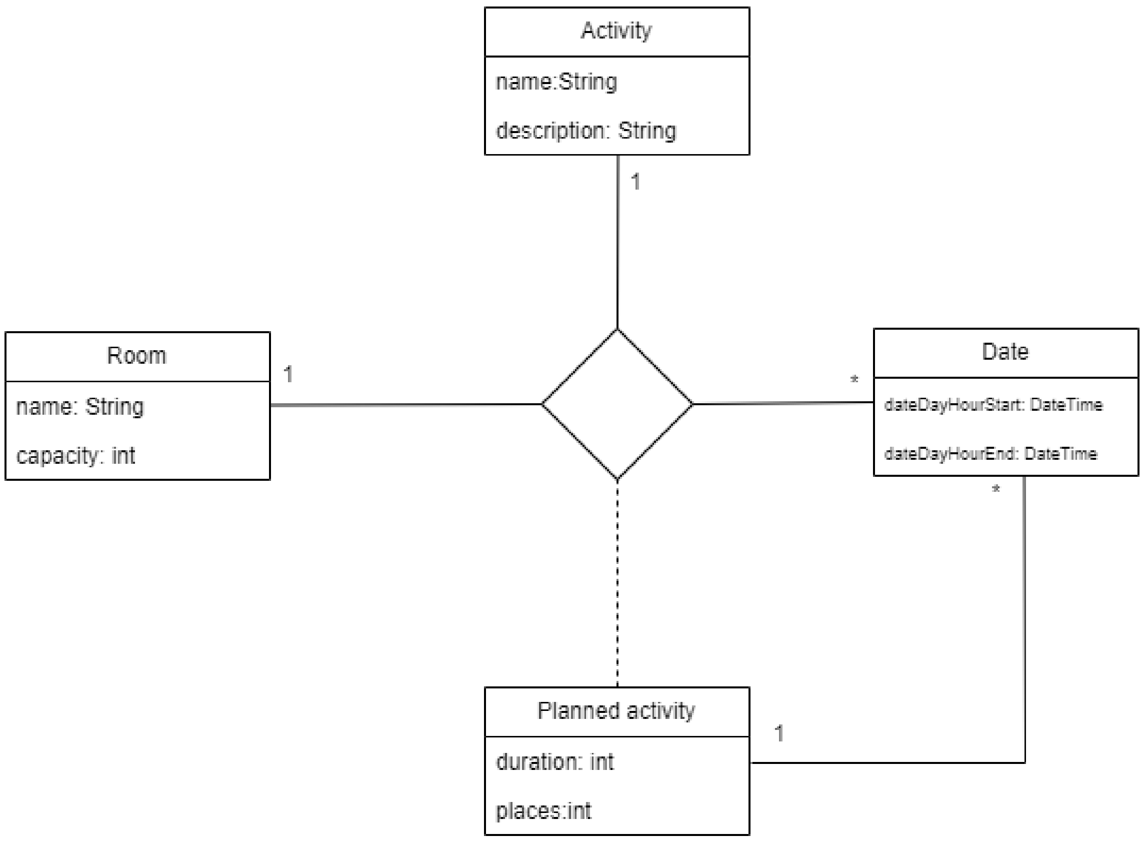


3 a)

I consider that the best option is the historical association pattern. If we start the approach from an association between “Room” and “activity”, we realize that we can extend the association with a “Date” type class, which will save the date and time where the class takes place. Therefore, we have a ternary association that would allow us to see the history of classes that are taught in the rooms on a series of dates. Therefore, we can recover the information that the association has had over time.

On the other hand, these activities have a period in which they are taught, they have a schedule that determines them in terms of time and quantity. Therefore, I add the associative class “Planned activity”, which will have an associated class type “Date”, since they have a certain period to be taught.

3b)



Integrity Contraints:

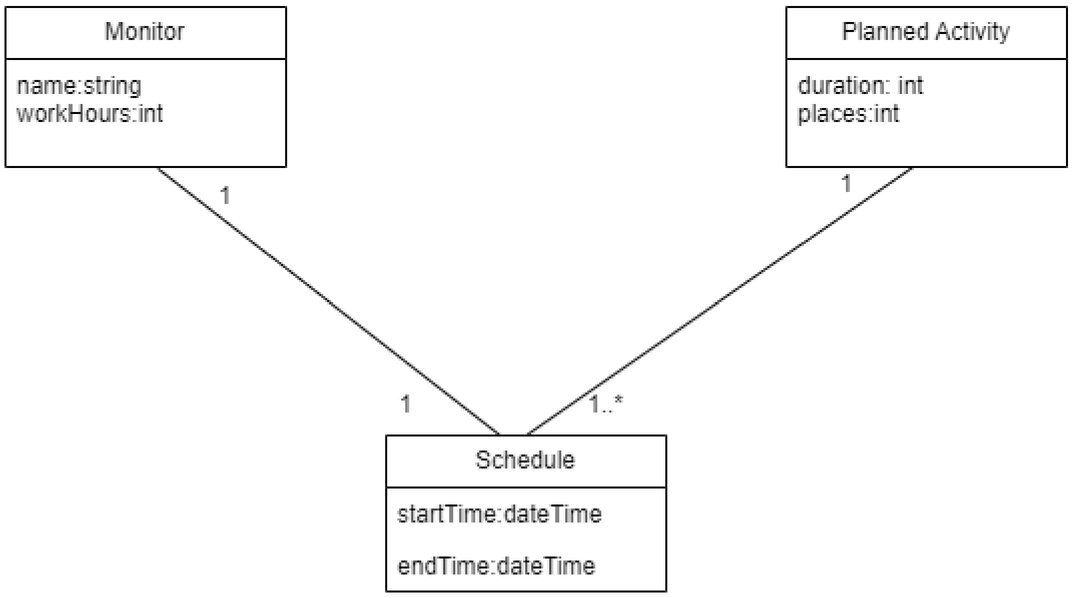
.

- There cannot be more than one planned activity for the same date and time.
- There cannot be a scheduled activity of the same type for the same date and time.

4 a)

In this problem, I consider that the best solution is the pattern of Range. If we pay attention to the statement, the purpose of the diagram is to know the semantics of a range. If we propose the exercise, the basis would be a binary association, an association between “Monitor” and “Planned Activity”. However, these planned activities are carried out by the Monitor during his day, in a range of time, and the planned activities are also carried out within a period of time. Therefore, putting time as attribute in each class is going to complicate the design of the diagram, so the optimal thing would be to create a “Schedule” type class in which the start date and end date attributes are present, in such a way that the instance towards that class has the start and end values. Thus, we can save records of the interaction between the instructor and the planned classes that he teaches on his schedule.

4b)



Integrity Contraints:

- The Schedule is always from Monday to Friday
- The monitor's working hours are always continuous.