Question 1 [15 points] {15 minutes}

We need to design a photo application that allows us to:

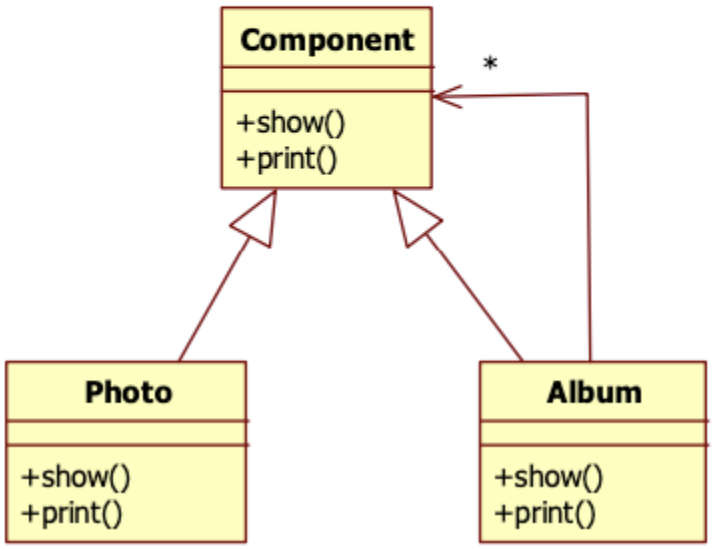
- Create different photo albums that contain photos
- Photo albums can also contain sub albums
- We should be able to show the whole album, or to show an individual photo
- We should be able to print the whole album, or to print an individual photo

a. What pattern would you apply in your design?
b. Draw the class diagram of your design.

RESULT 1

a. Composite

b.



Question 2 [15 points] {15 minutes}

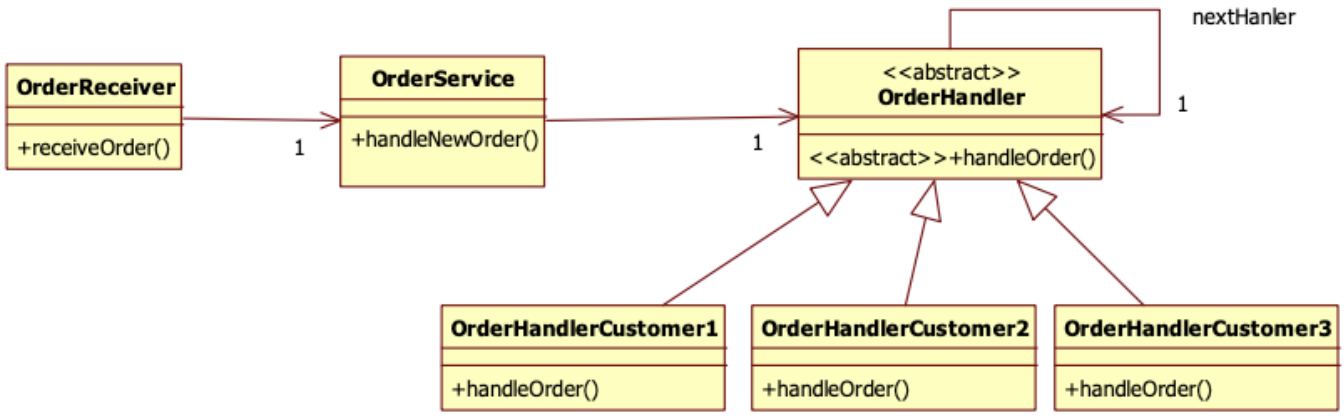
Suppose a certain device can send 45 different messages to an application that need to handle these messages. Every message contains a certain number between 1 and 45 that corresponds to the type of message. The receiver object that receives these messages has to make sure the correct logic is executed based on the message number. For every message with a different number, some different logic needs to be executed.
It is important that the design of the application allows us to easily add more messages (for example 100 different messages) with the least amount of code change.

a. What pattern would you apply in your design?
b. Draw the class diagram of your design.

RESULT 2

a. Chain of Responsibility

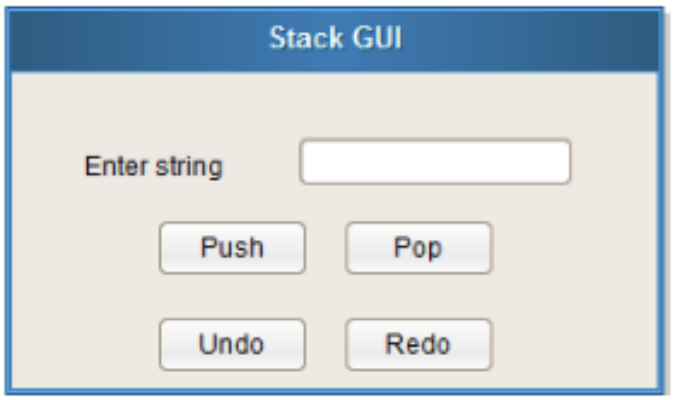
b.

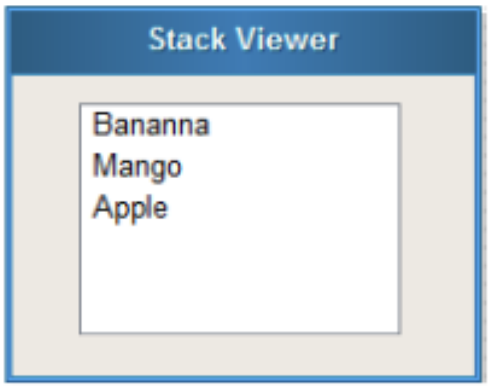


Question 3 [35 points] {40 minutes}

Suppose we have to design a simple Stack application:
In case you don't know what a Stack is: A Stack is a list of objects that work according the LIFO principle, which is Last In, First Out.
The 2 important methods are push(x) which adds the object x to the top of the stack, and pop() which takes the last added object off the stack.
The Stack application has the following requirements:

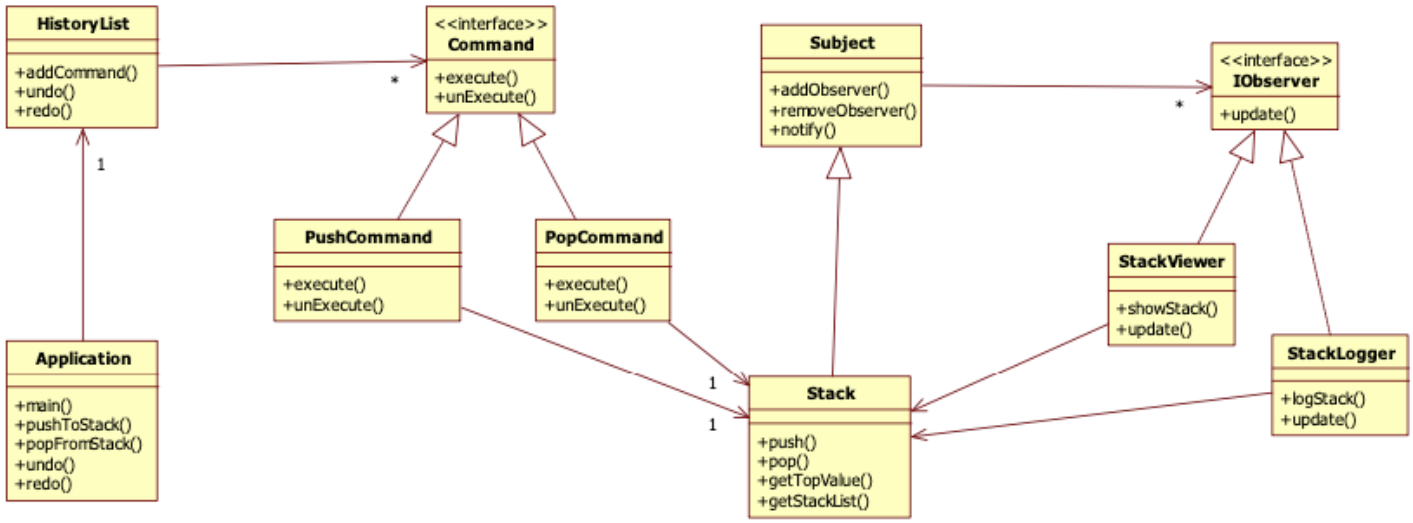
- The Stack contains String objects.
- We have a GUI class which allows us to enter a String. Then this GUI class has 4 buttons: A **Push** button, a **Pop** button, an **Undo** button and a **Redo** button.
- We have a StackViewerGUI class which shows the content of the stack.
- It should be easy to add more GUI classes to this application that show the content of the stack in a different way.
- Every time we change the stack, the content of the stack is written to a logfile.
- The stack should be reusable in other applications, so it should not depend on other classes.

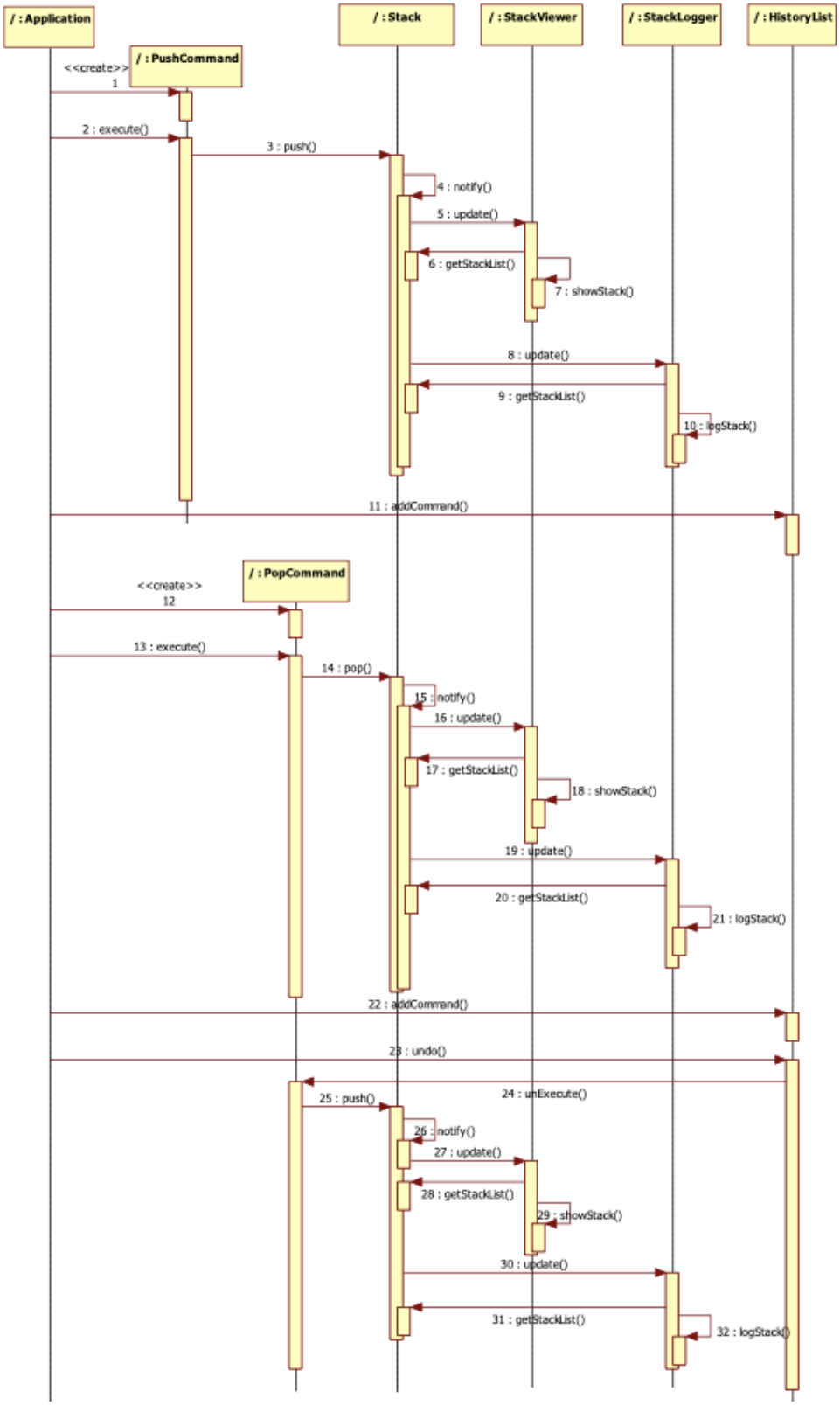




a. Draw a **class diagram** of your design of the stack application. Make sure you apply the principles and best practices we studied in this course. Make sure your class diagram contains all the important information to communicate your design.
b. Draw a **sequence diagram** showing the following sequence:
   1. First we push a particular String on the stack
   2. Then we call pop() on the stack
   3. Then we click the undo button.

RESULT 3


HistoryList
+addCommand()
+undo()
+redo()
<<interface>>
Command
+execute()
+unExecute()
Subject
+addObserver()
+removeObserver()
+notify()
<<interface>>
IObserver
+update()
PushCommand
+execute()
+unExecute()
PopCommand
+execute()
+unExecute()
StackViewer
+showStack()
+update()
Application
+main()
+pushToStack()
+popFromStack()
+undo()
+redo()
Stack
+push()
+pop()
+getTopValue()
+getStackList()
StackLogger
+logStack()
+update()
1
*
*
1
1
1


/ : Application
/ : PushCommand
/ : Stack
/ : StackViewer
/ : StackLogger
/ : HistoryList
<<create>>
1
2 : execute()
3 : push()
4 : notify()
5 : update()
6 : getStackList()
7 : showStack()
8 : update()
9 : getStackList()
10 : logStack()
11 : addCommand()
/ : PopCommand
<<create>>
12
13 : execute()
14 : pop()
15 : notify()
16 : update()
17 : getStackList()
18 : showStack()
19 : update()
20 : getStackList()
21 : logStack()
22 : addCommand()
23 : undo()
24 : unExecute()
25 : push()
26 : notify()
27 : update()
28 : getStackList()
29 : showStack()
30 : update()
31 : getStackList()
32 : logStack()

QUESTION 4

Suppose you have to design a frequent flyer miles application that keeps track of the number of miles a certain customer has.

The application has the following requirements:

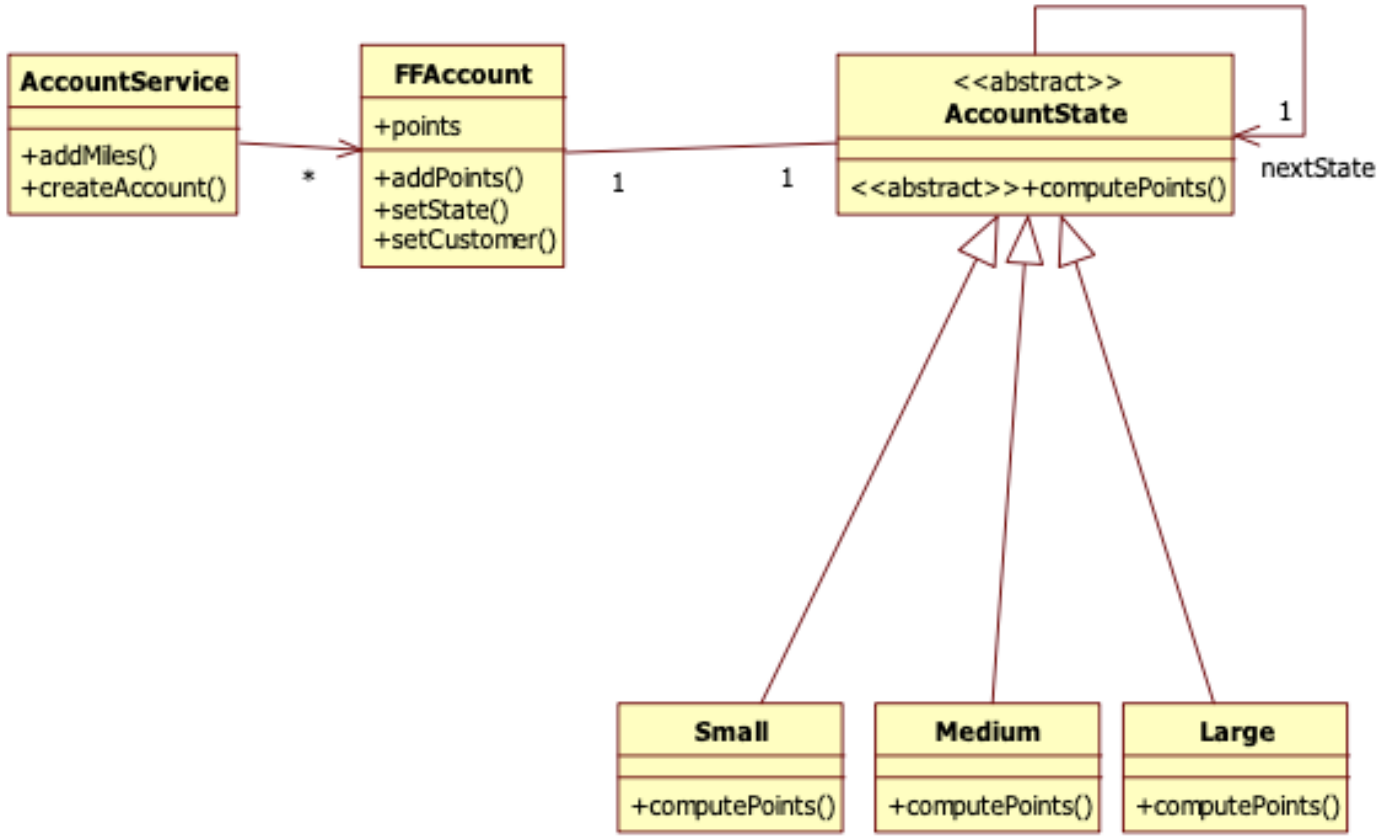
- There are 3 types of accounts, "**small**" , "**medium**" and "**large**". The
- Everyone starts with a "small" account.
- When you have a **small** account and have more than 100.000 miles, you are upgraded to a "**medium**" account.
- When you have a **medium** account and have more than 200.000 miles, you are upgraded to a "**large**" account.
- Small accounts receive the same number of miles as the actual miles of their flights.
- Medium accounts receive 2 times the number of miles they actually fly.
- Large accounts receive 3 times the number of miles they actually fly.
- It should be easy to add new types of accounts. For example, if want to add an "**extra-large**" or an "**extra-small**" account, then this should be very easy to implement.

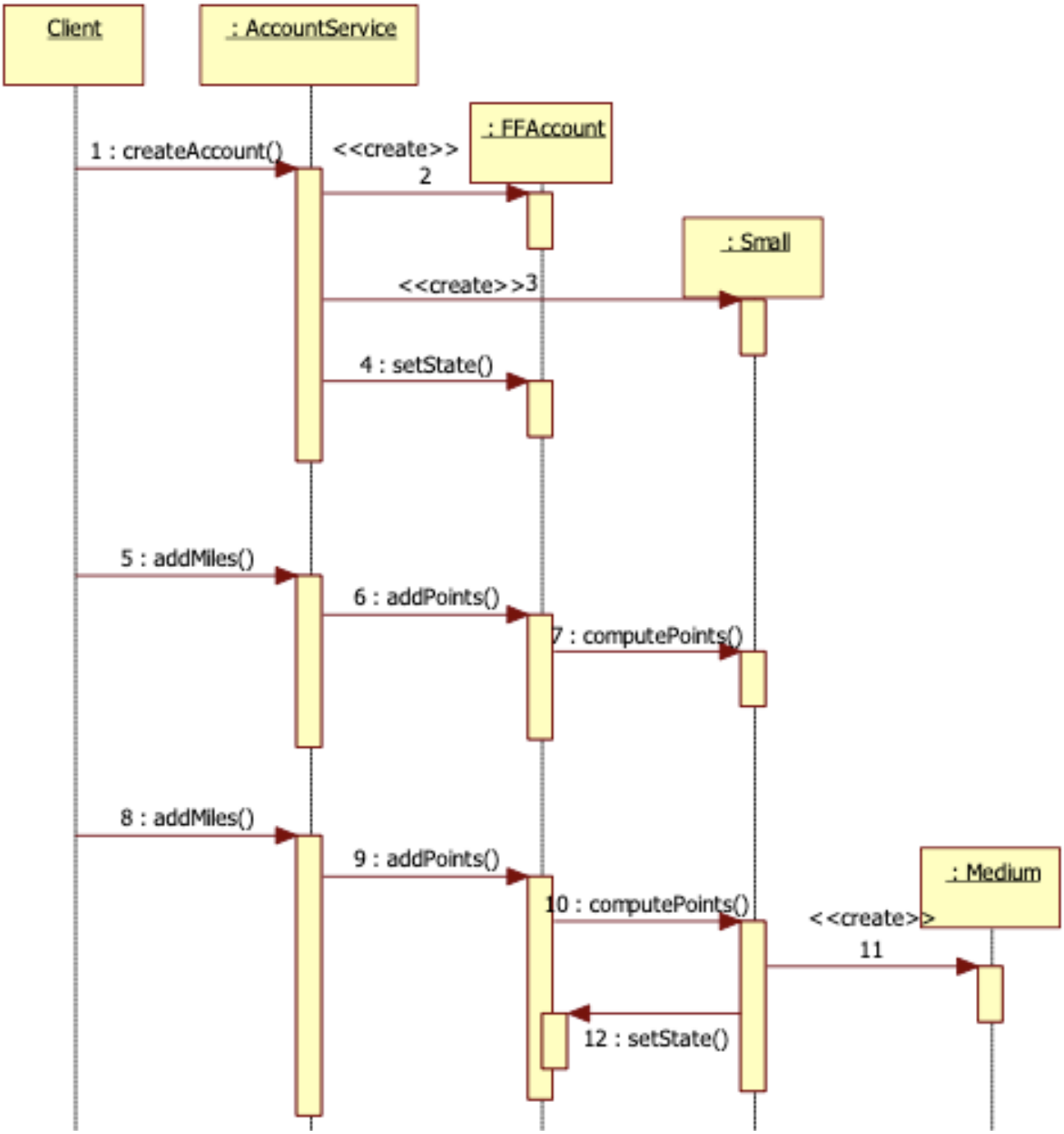
a. Draw the class diagram of the Frequent Flyer Miles application. Make sure you apply the principles and best practices we studied in this course. Make sure you add all necessary UML elements (attributes, methods, multiplicity, etc) to communicate the important parts of your design.

b. Draw a sequence diagram that shows the behavior of your application. The sequence diagram should show the following sequence of actions:
    a. First we create a new **small** account.
    b. Then call the method addMiles() again on the account, but we have NOT enough miles to get an upgrade
    c. Then call the method addMiles() again on the account, but now we DO have enough miles to switch from **medium** to **large**

Complete the partial given sequence diagram:

RESULT 4


AccountService
+addMiles()
+createAccount()
*
FFAccount
+points
+addPoints()
+setState()
+setCustomer()
1
1
<<abstract>>
AccountState
<<abstract>>+computePoints()
1
nextState
Small
+computePoints()
Medium
+computePoints()
Large
+computePoints()


Client
: AccountService
: FFAccount
: Small
: Medium
1 : createAccount()
<<create>>
2
<<create>>3
4 : setState()
5 : addMiles()
6 : addPoints()
7 : computePoints()
8 : addMiles()
9 : addPoints()
10 : computePoints()
<<create>>
11
12 : setState()

Question 4 [5 points] {10 minutes}

Describe how we can relate the **Chain of Responsibility** pattern to one or more principles of SCI. Your answer should be about half a page, but should not exceed one page (handwritten). The number of points you get for this questions depends how well you explain the relationship between the **Chain of Responsibility** pattern and one or more principles of SCI.