Lab 1

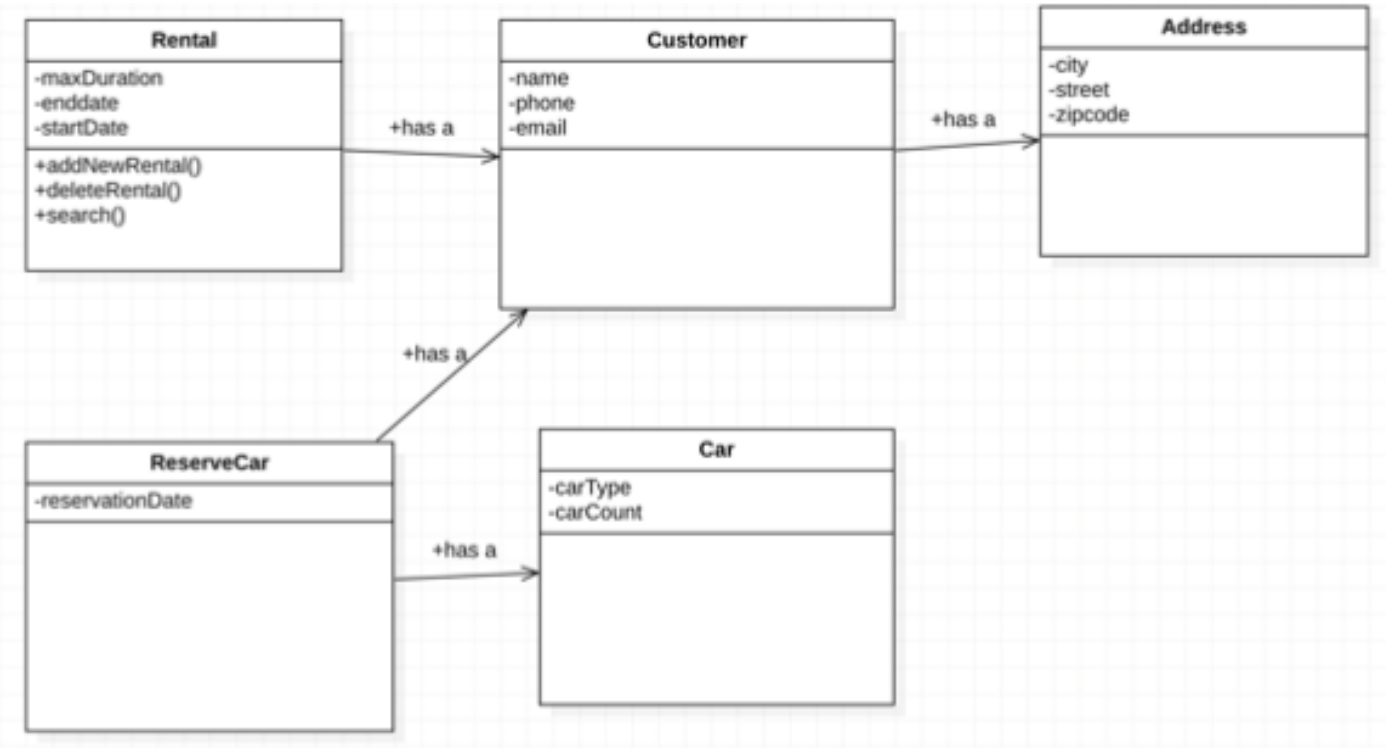
a.

We need to design the domain model of a car rental application. The car rental application has the following requirements:

- We should be able to add new rentals, delete rentals and search rentals.
- For every rental we need to store the name, phone, email, street, city and zip code of the customer. We also need to keep track of the start date of the rental, the maximum duration of the rental and the end date of the rental.
- The application should also support the functionality reserve cars. For every reservation you make, you need to keep track of the date you reserved it.
- The application should support the fact that we have multiple cars of the same type. So we might have 8 Ford F-150 Pick-up trucks available.

Draw the class diagram of the domain model. The domain model is also called the model of the business (logic) and only contains the domain (or business) classes. The domain model does **NOT** contain technical classes like service classes, DAO classes or other technical plumbing classes.

You can draw the UML diagrams of this lab either with pen and paper or use an UML tool. If you use pen and paper, make a picture of your solution and submit the pictures of your solutions. If you use a tool, just submit a screenshot of your solution.

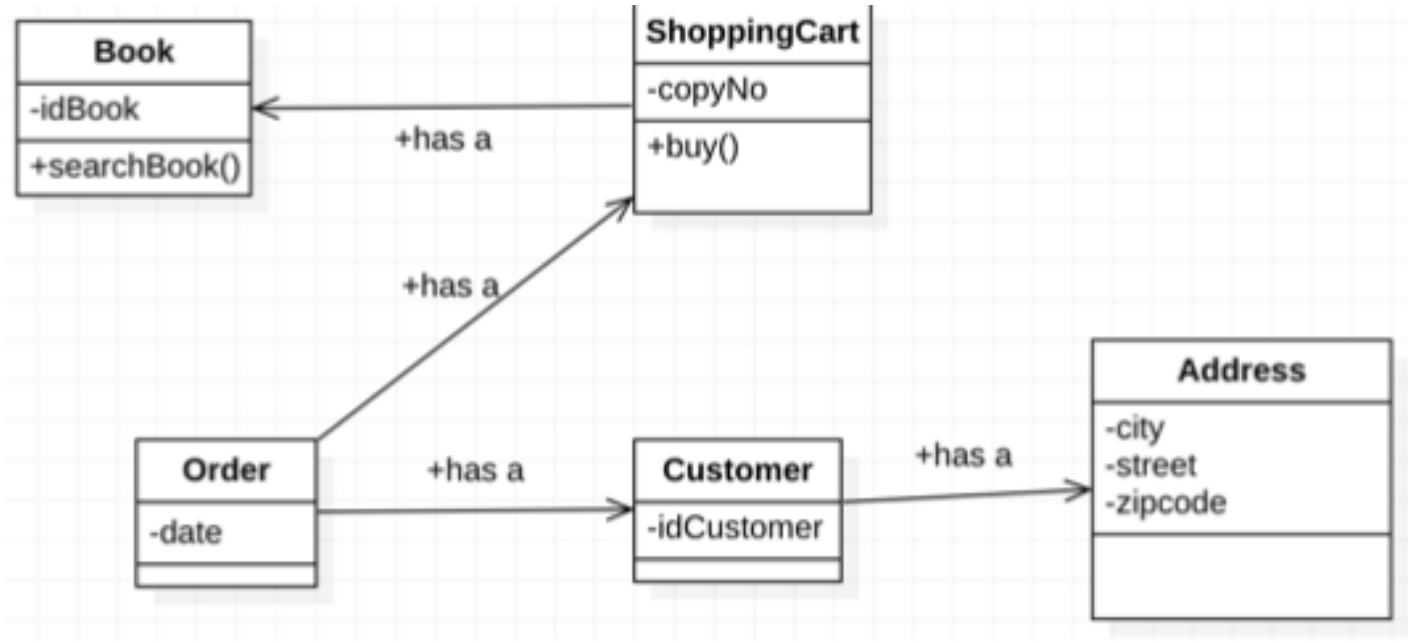


b.

We need to design the domain model of a webshop application for selling books. The book webshop application has the following requirements:

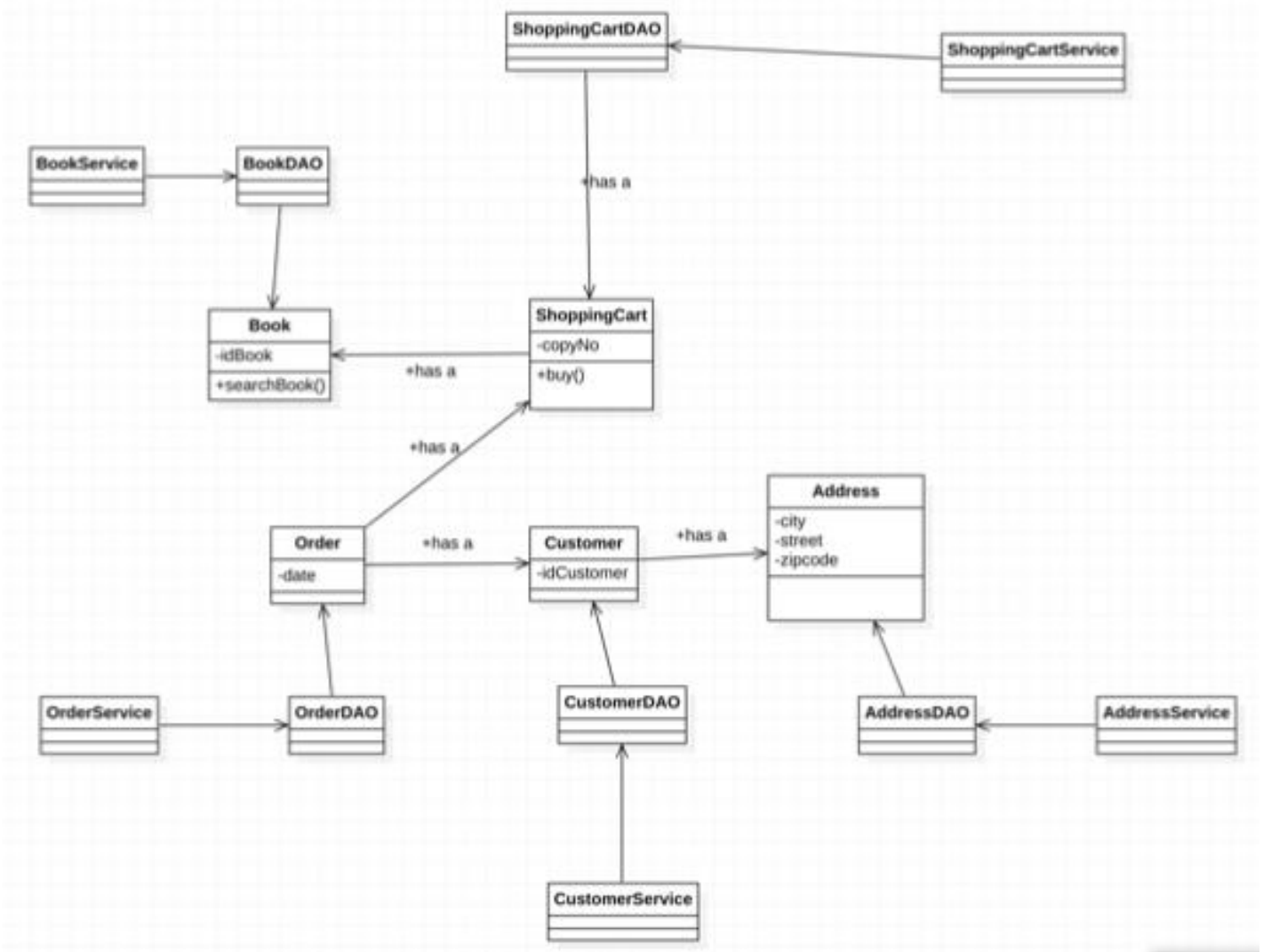
- We should be able to search books.
- We should be able to fill a shoppingcart with different books.
- We should be able to buy multiple copies of the same book.
- We should be able to order a content of the shoppingcart and add our customer data so it is shipped to the correct address

Draw the class diagram of the domain model



c.

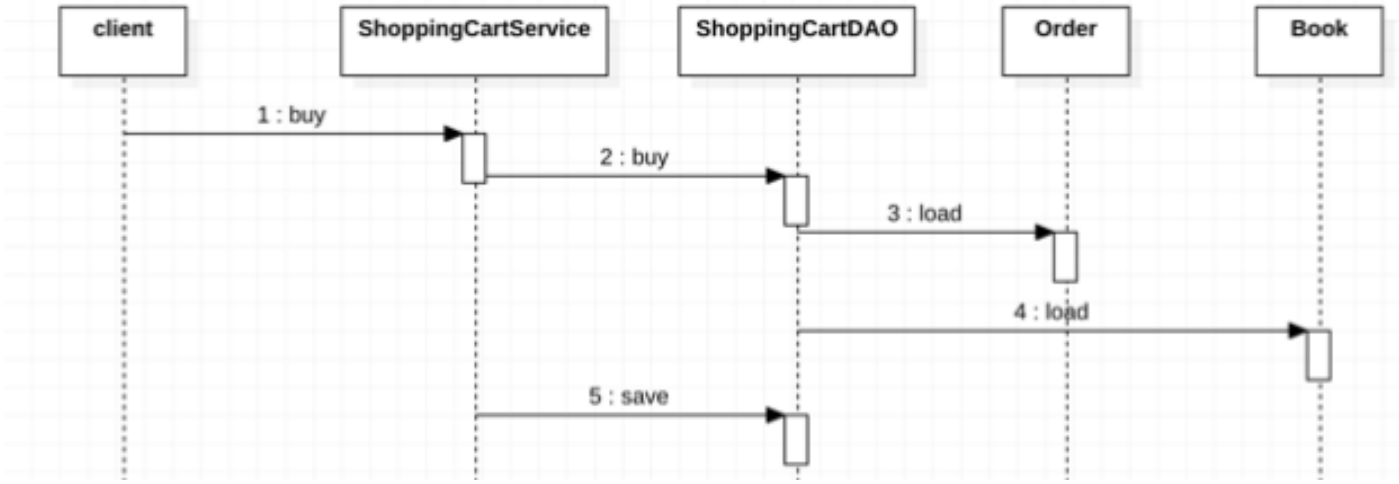
Draw the class diagram of the webshop of part b, but also add the necessary technical plumbing classes like service classes, DAO classes, etc. Assume that the shoppingcart is stored in the database.



d.

Based on the class diagram of part c, draw the sequence diagram of the scenario where we add a new product to the existing shoppingcart.

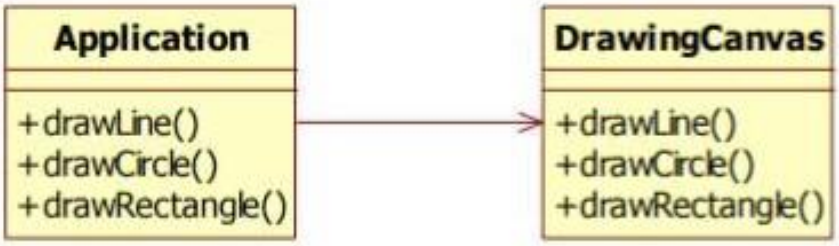
d)


client
ShoppingCartService
ShoppingCartDAO
Order
Book
1 : buy
2 : buy
3 : load
4 : load
5 : save

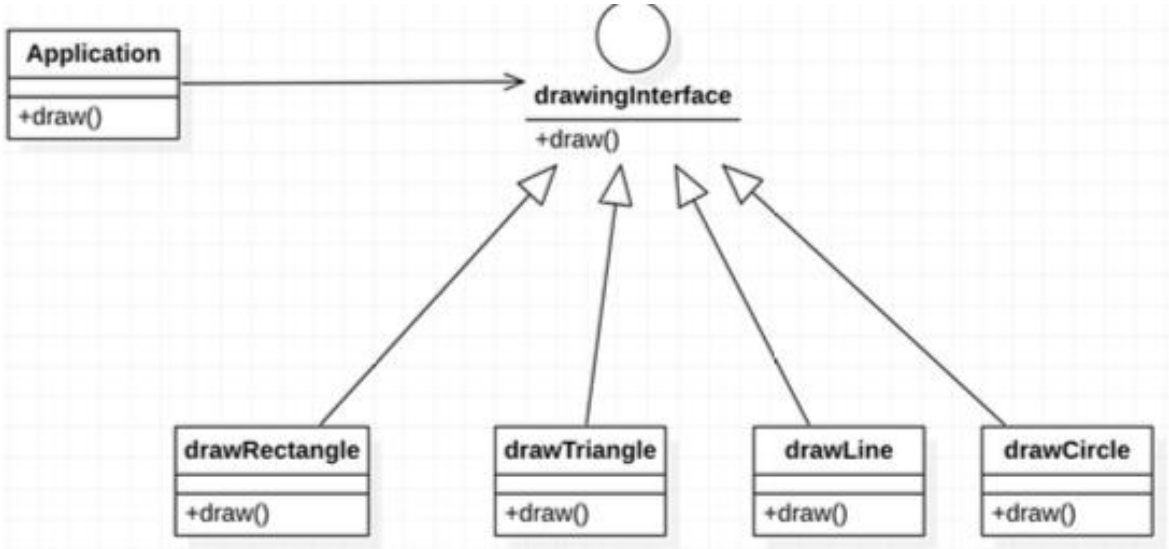
LAB2

a.

Suppose we have the following simple drawing program:

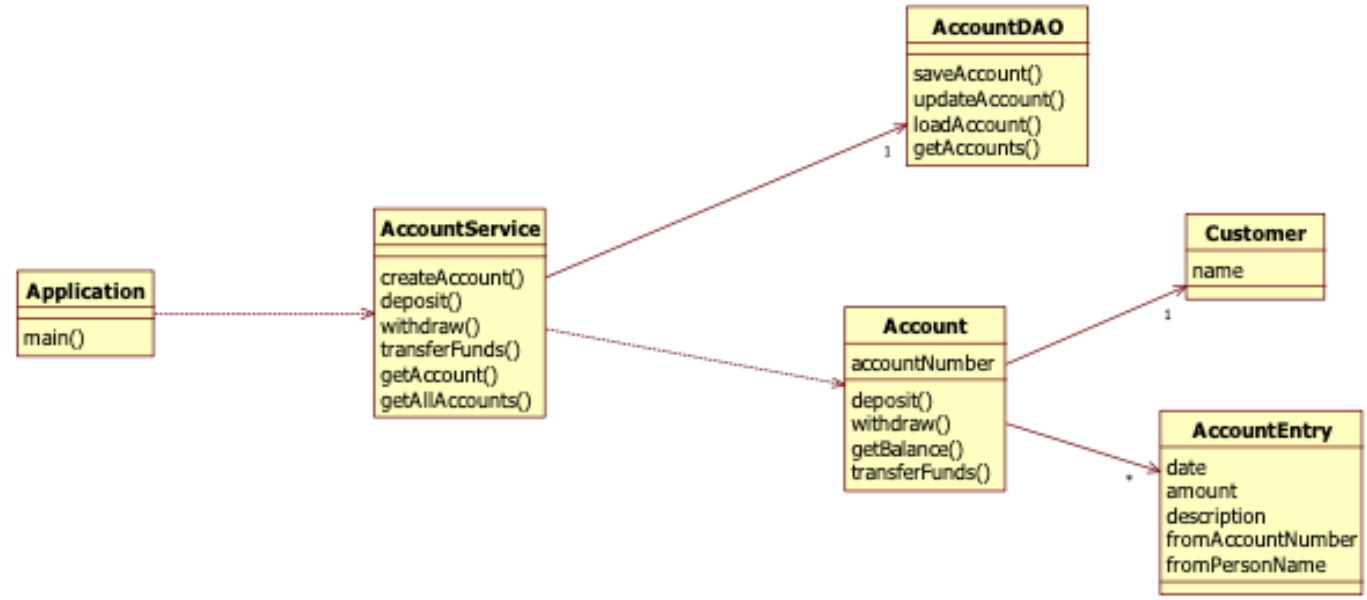


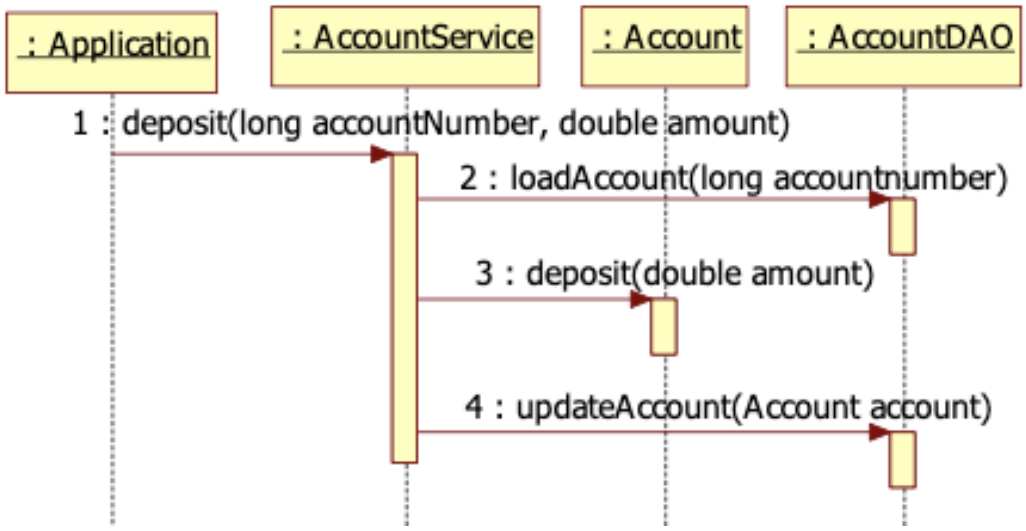
If you call drawLine() from the Application class, then the drawLine() of the DrawingCanvas gets called, and a line is drawn on the canvas. A disadvantage of the given drawing application is that it only draws lines, circles and rectangles, and if we also want to draw triangles, we have to change the code of the DrawingCanvas class. Redraw the class diagram such that it will be easier to add new shapes to the drawing program. Also draw a sequence diagram that shows how we can draw for example a line and 2 circles.



b.

Given is the following bank application:





Add functionality such that this bank supports Savings and Checkings accounts. For all accounts we want to add interest, and the interest is calculated with the algorithms given below. Use the strategy pattern.

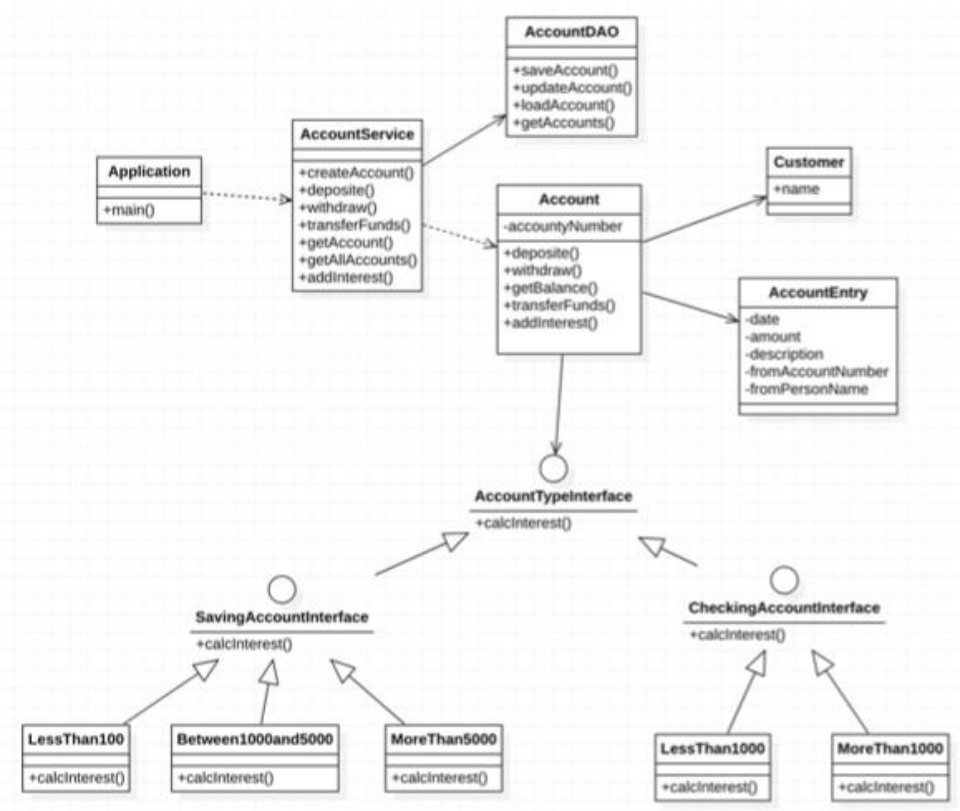
Interest for a savings account:

- If balance < 1000 then you get 1% interest
- If balance > 1000 and balance < 5000 then you get 2% interest
- If balance > 5000 then you get 4% interest

Interest for a checkings account:

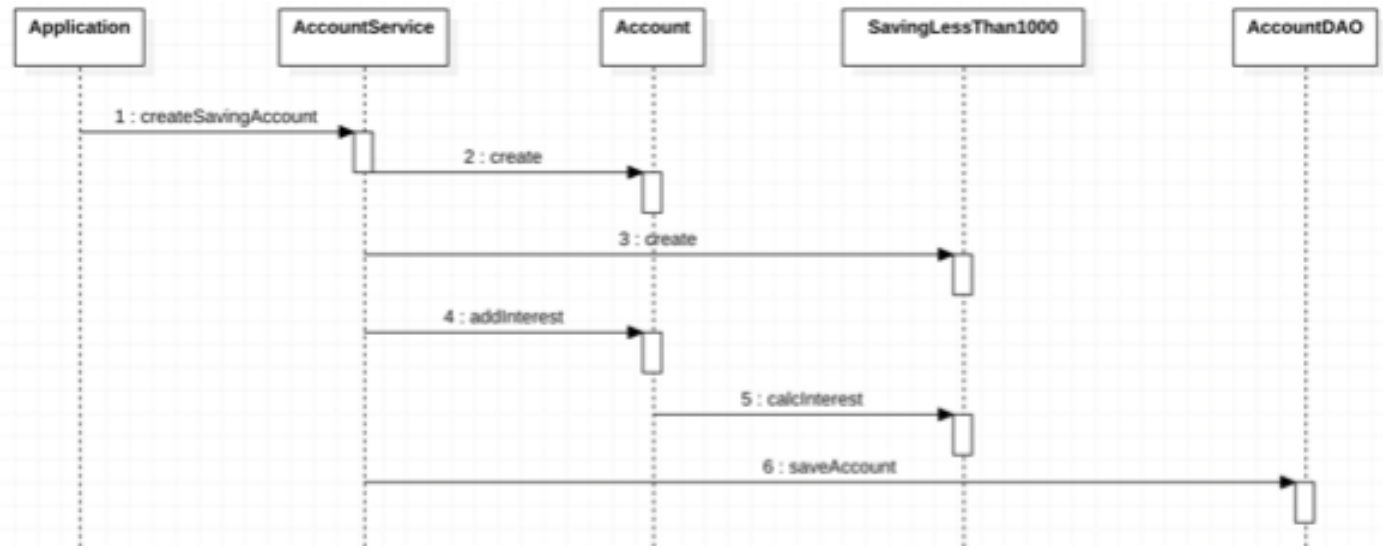
- If balance < 1000 then you get 1,5% interest
- If balance > 1000 then you get 2,5% interest

Draw the modified class diagram with the strategy pattern applied.



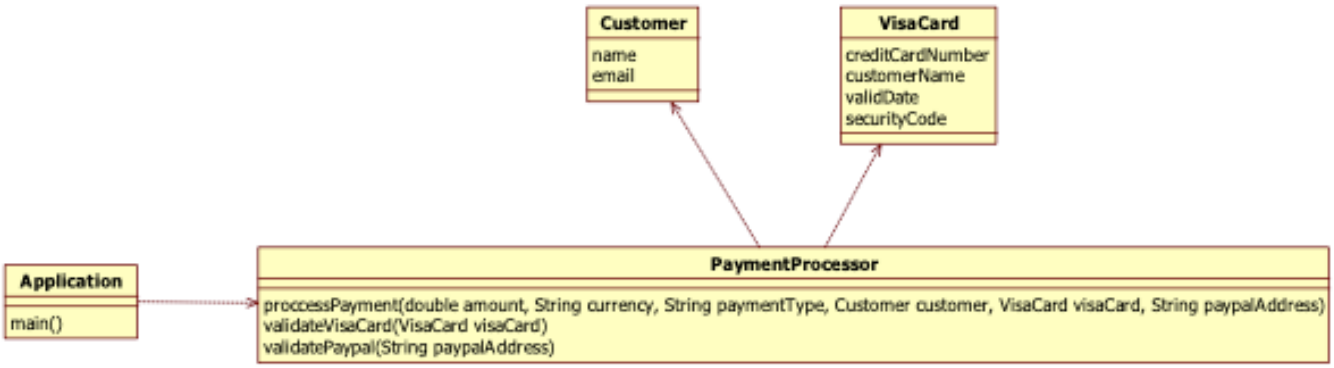
c.

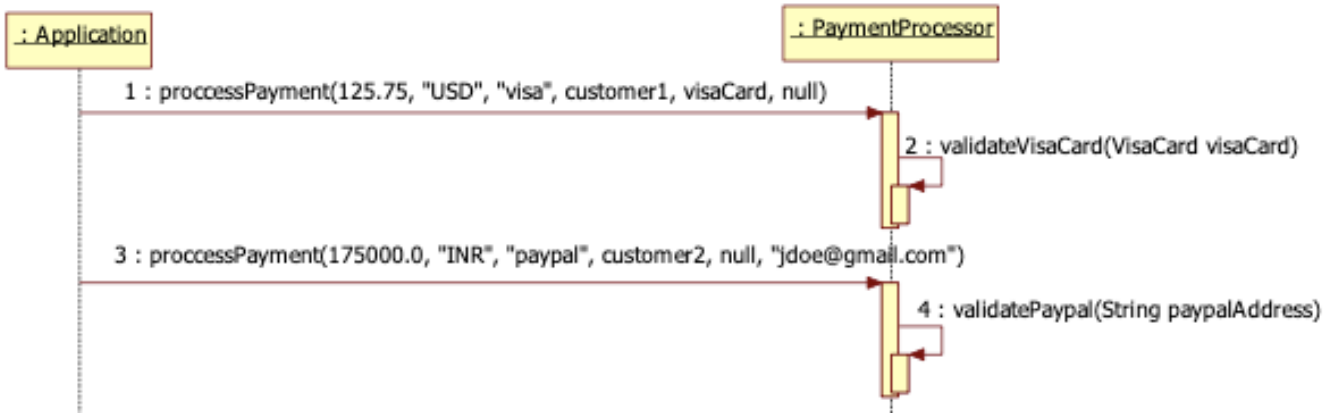
Draw a sequence diagram that shows how your new design works. On the sequence diagram you should show how the strategy pattern works. The AccountService should have a method addInterest(), and then the interest should be calculated and added to all accounts.



e.

Given is the following application:

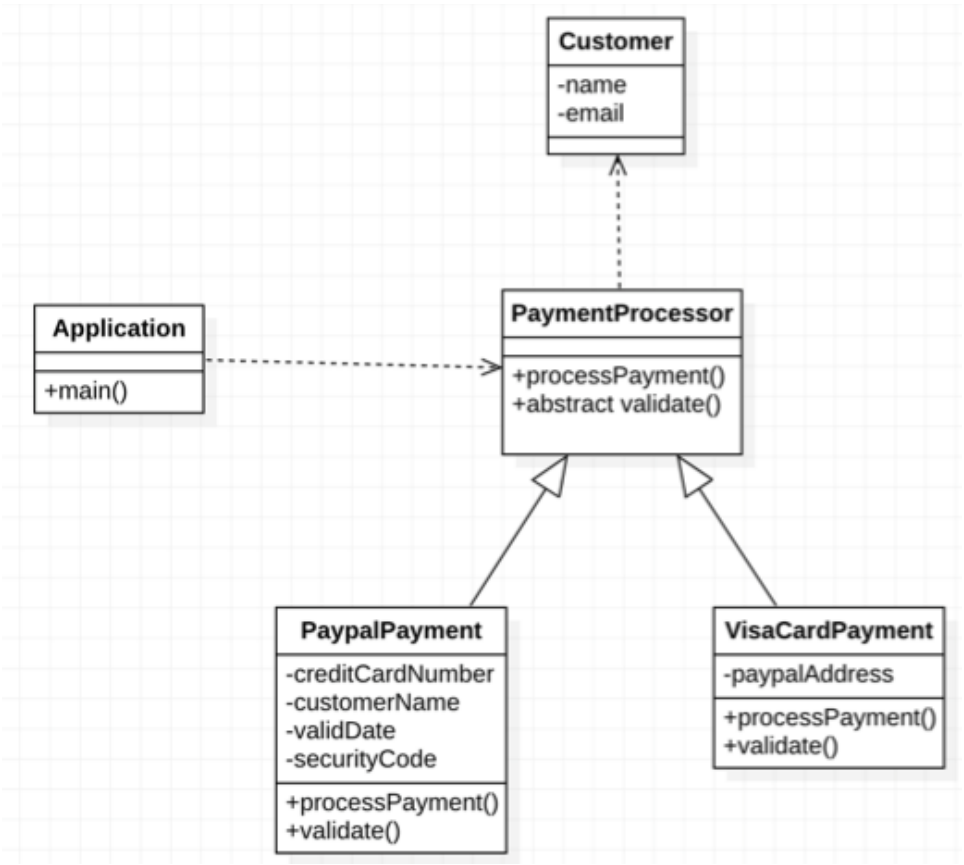




With the PaymentProcessor you can process payments. The given implementation supports both Visa Card payment and PayPal payments. The problem with the given implementation is the processPayment method is large and complex which makes it hard to add support for other payment methods like MasterCard payments.

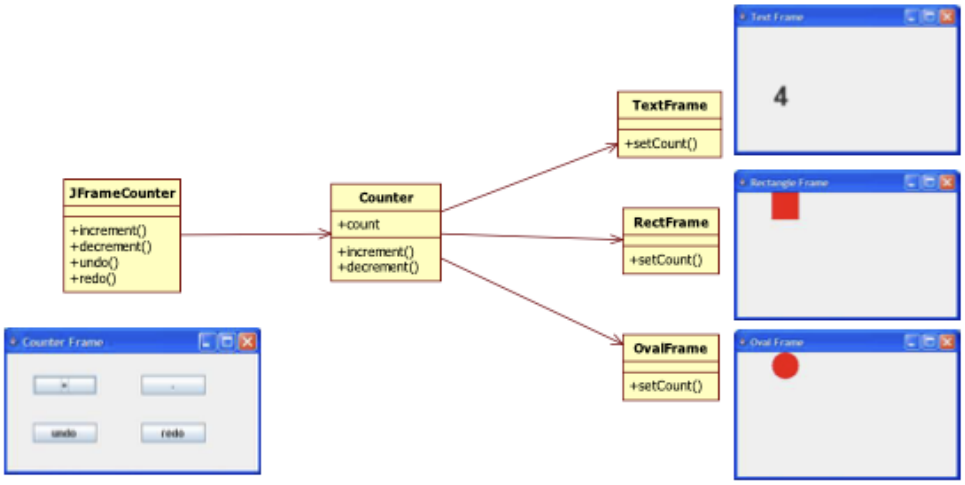
Draw the class diagram of a better design using the template method pattern.

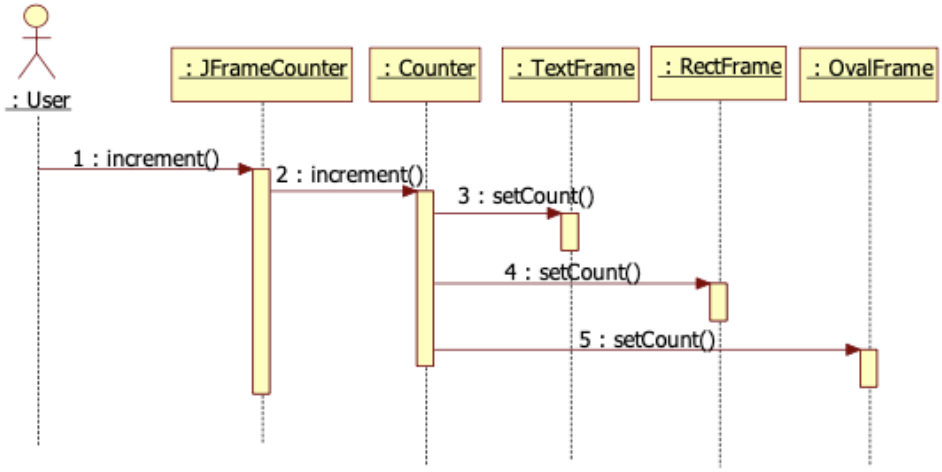
Draw the corresponding sequence diagram


Customer
-name
-email
Application
+main()
PaymentProcessor
+processPayment()
+abstract validate()
PaypalPayment
-creditCardNumber
-customerName
-validDate
-securityCode
+processPayment()
+validate()
VisaCardPayment
-paypalAddress
+processPayment()
+validate()

LAB3

a. Given is the following counter application:

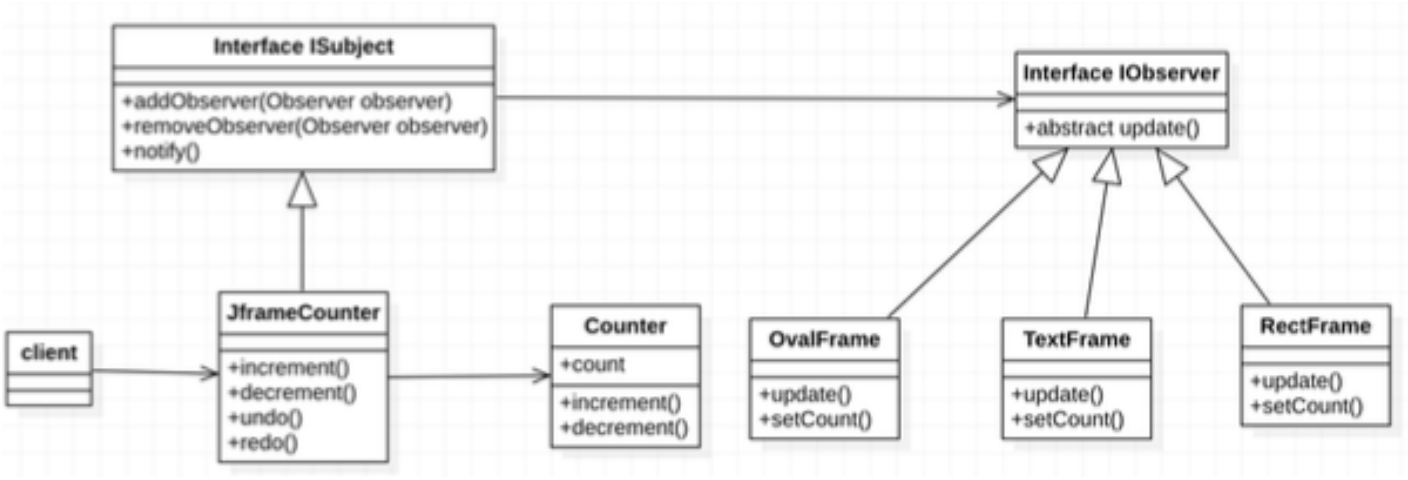




The undo and redo methods do not work yet. We will implement them in a later lab.

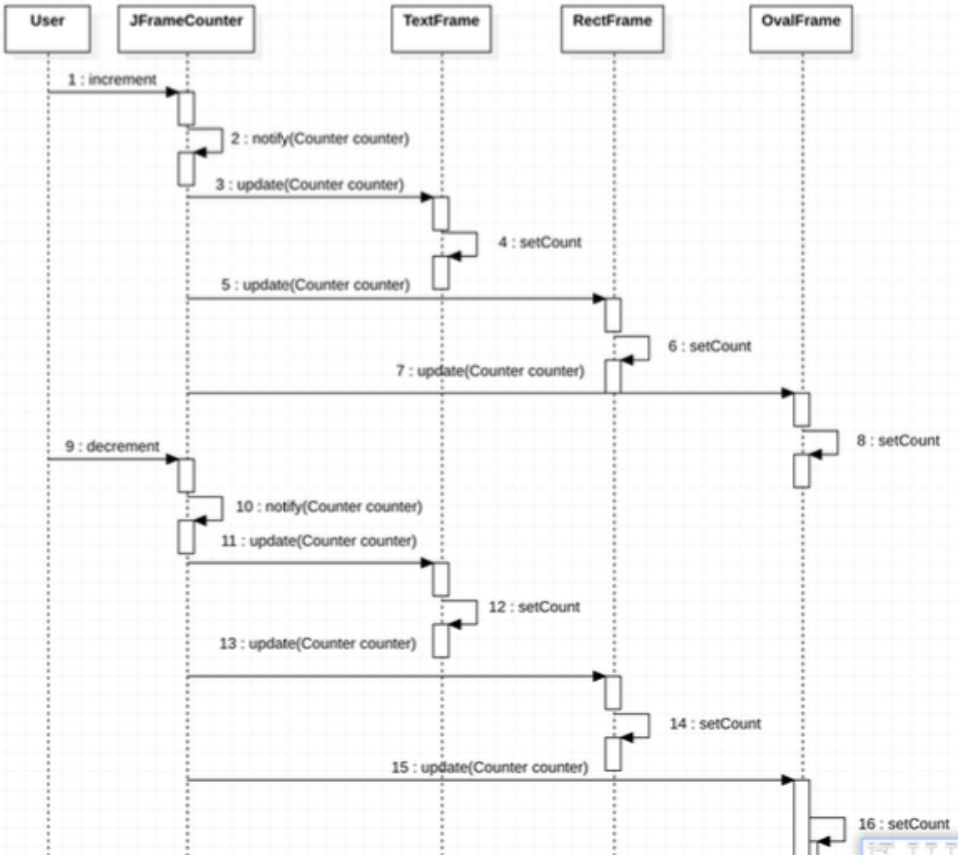
The problem with the given application is that the Counter class is tightly coupled with the UI classes TextFrame, RectFrame and OvalFrame. If we want to add another view of the counter, for example a binaryFrame that shows the value of the counter in binary, then we have to change the increment() and decrement() method in the Counter.

Draw the class diagram of a better design so that it will be much easier to add different views of the counter value.

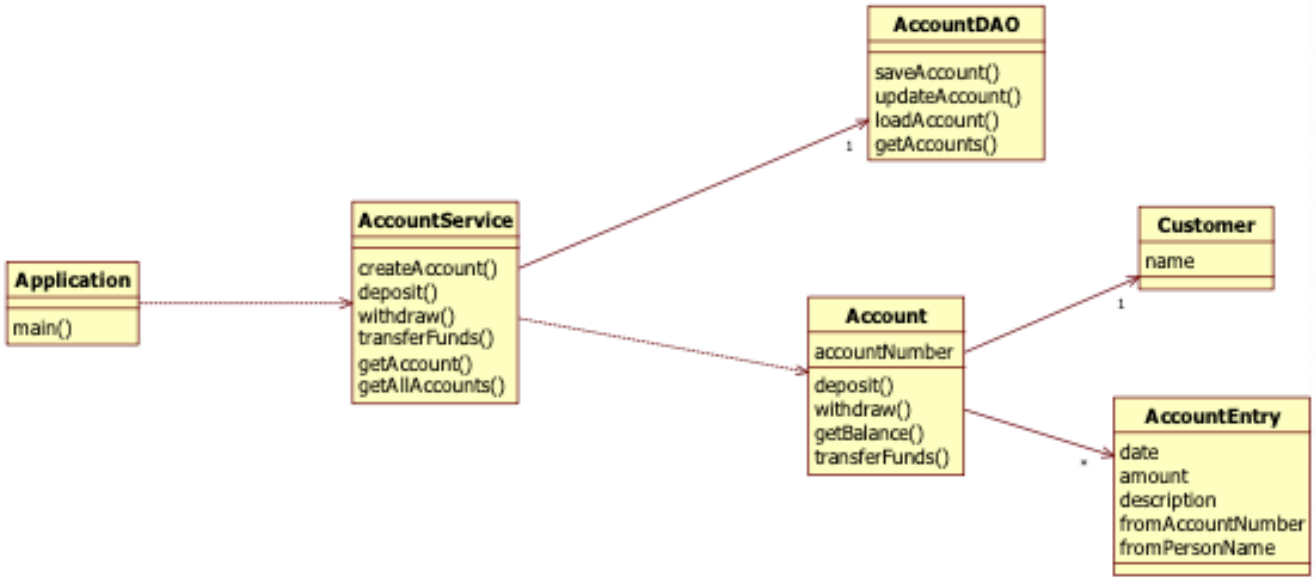


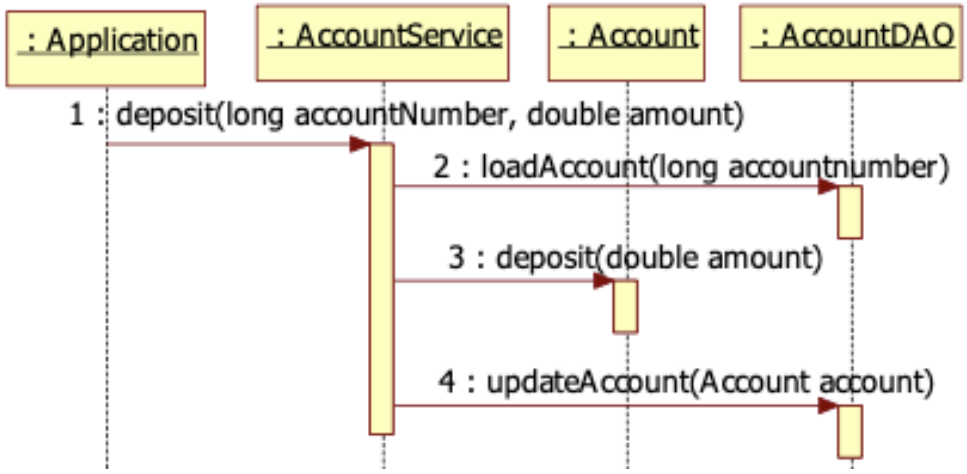
b. Draw the sequence diagram that shows the following scenario:

1. The user clicks the increment button
2. The user clicks the decrement button



d. Given is the following bank application:

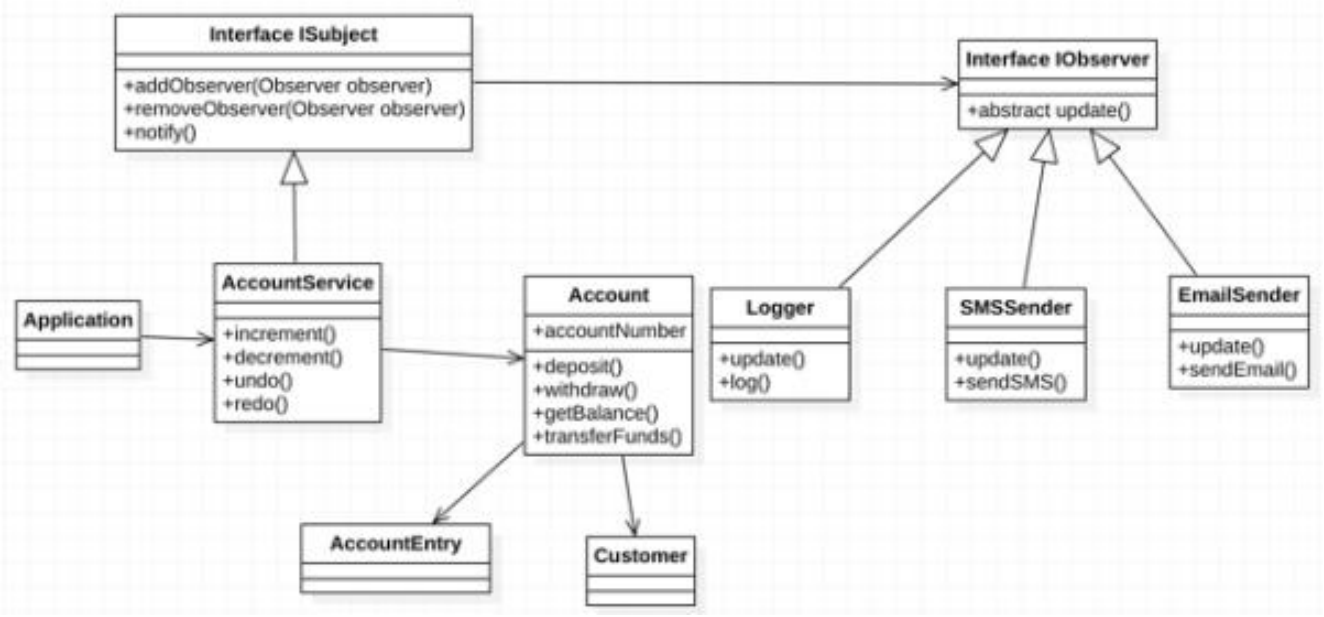




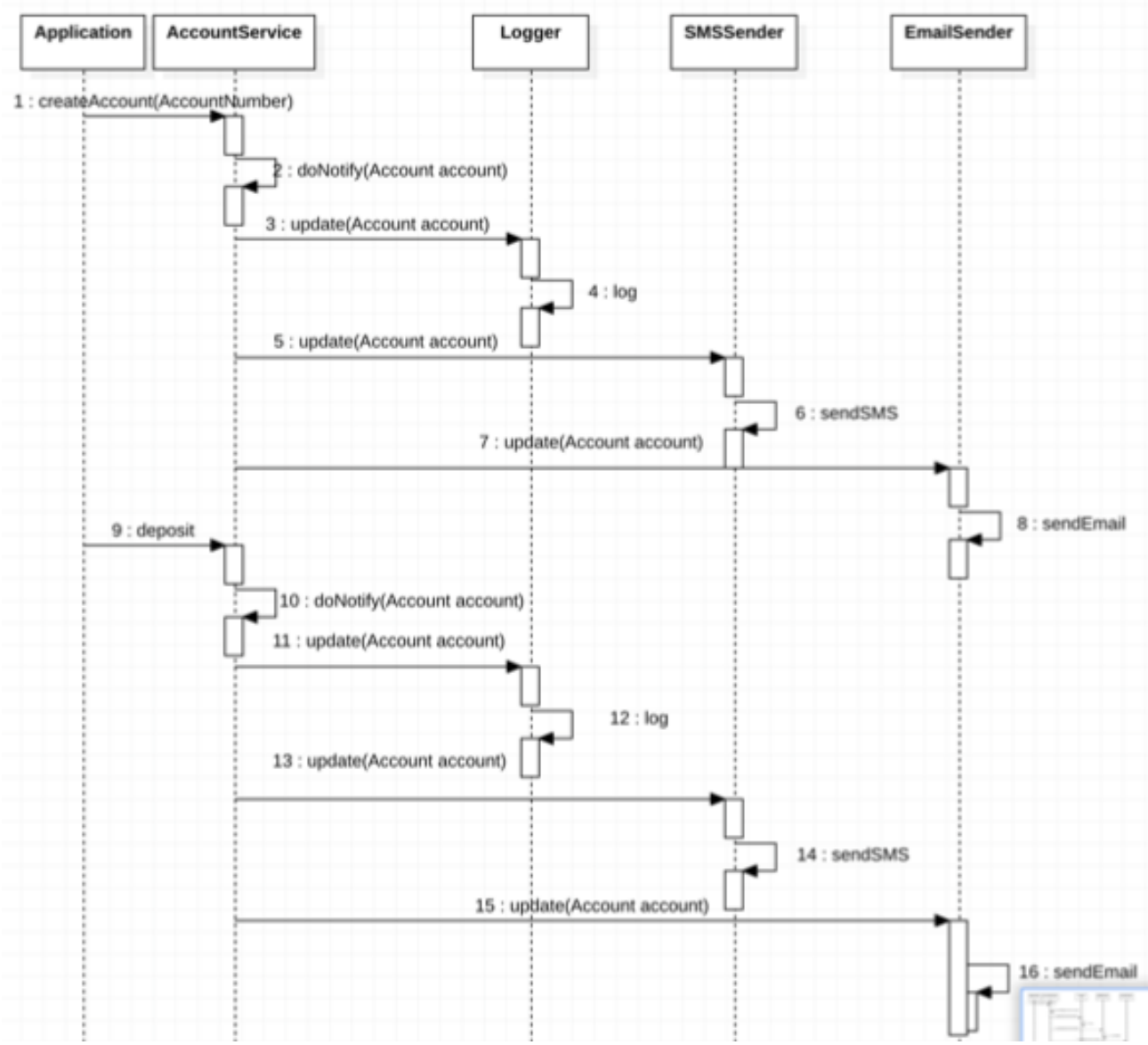
We want to add new functionality whenever the Account balance is changed. Implement the observer pattern in the given code. Add the following observers:

- Add a Logger class that logs every change to an Account. (The Logger should do a simple System.out.println() to the console)
- Add a SMSSender that sends a SMS at every change to an Account. (The SMSSender should do a simple System.out.println() to the console)
- Add an EmailSender that sends an email whenever a new Account is created. (The EmailSender should do a simple System.out.println() to the console)

e. Draw the modified class diagram with the observer pattern applied.

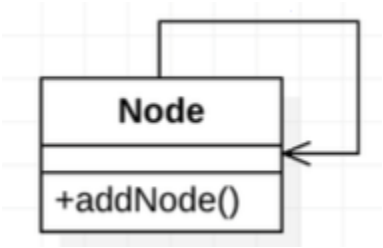


f. Draw a sequence diagram that shows how your new design works. On the sequence diagram show the following scenario:
   1. First create a new account
   2. Then deposit $80 on this new account

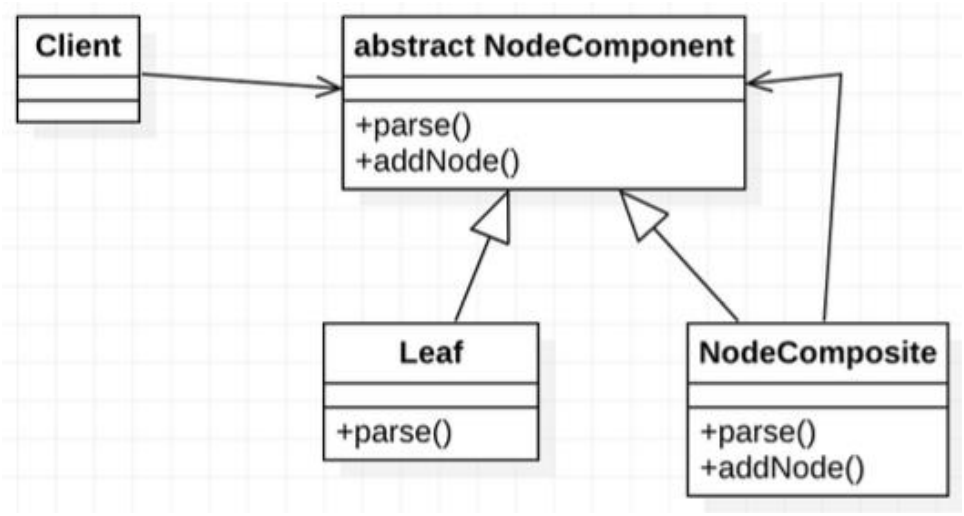


LAB4

d. Suppose you have to write an XML parser. Draw the class diagram of the domain model for this XML parser without using the composite pattern.



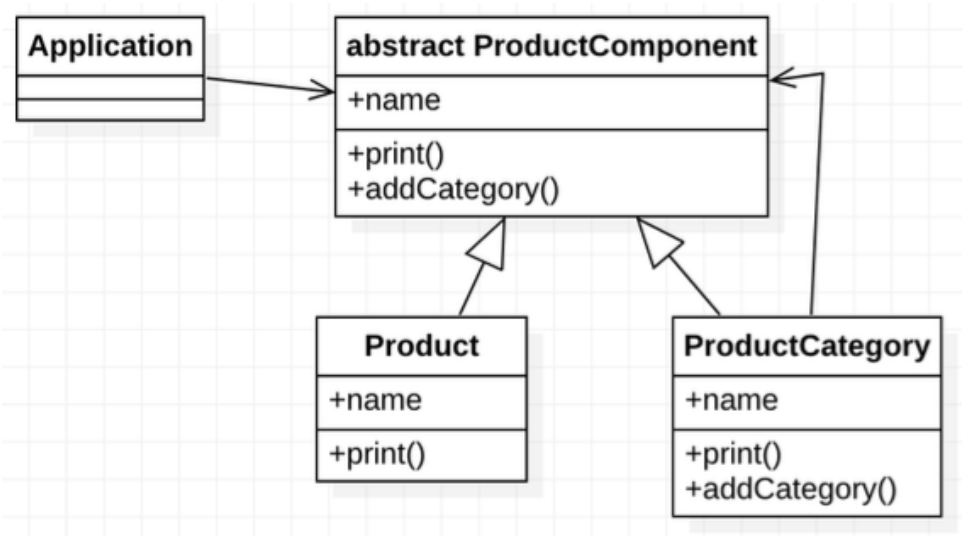
e. Draw the class diagram of the domain model for this XML parser using the composite pattern



f. Explain the advantage and/or disadvantage of the design of part c and part d.

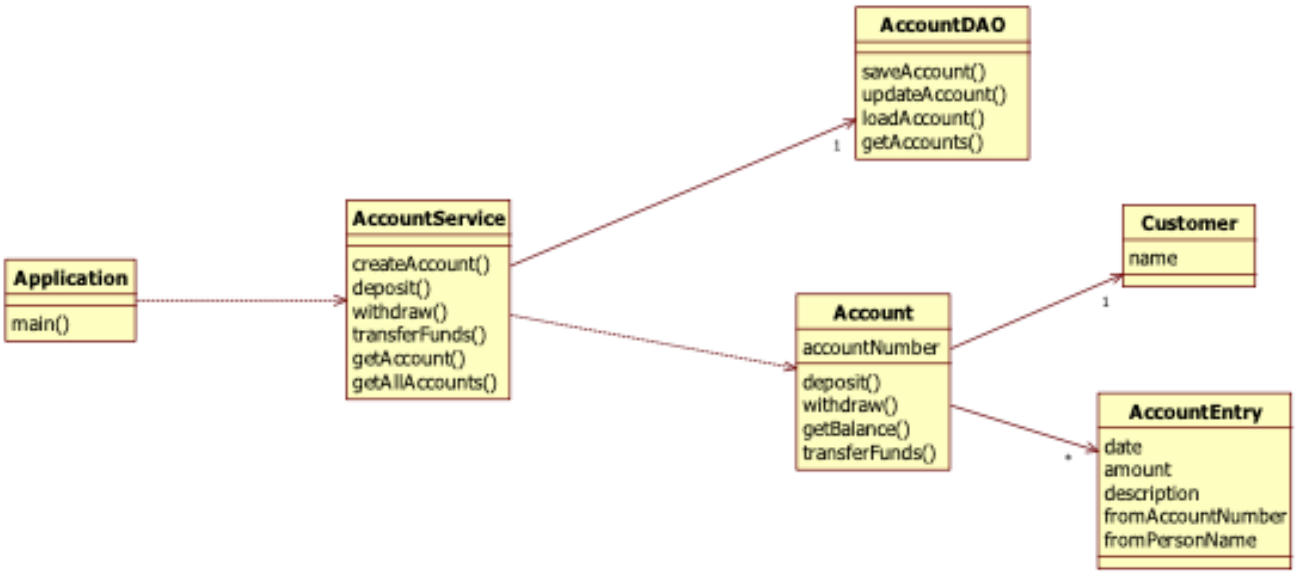
The advantage applying Composite pattern is I do not have to worry about the classes in the nested objects.

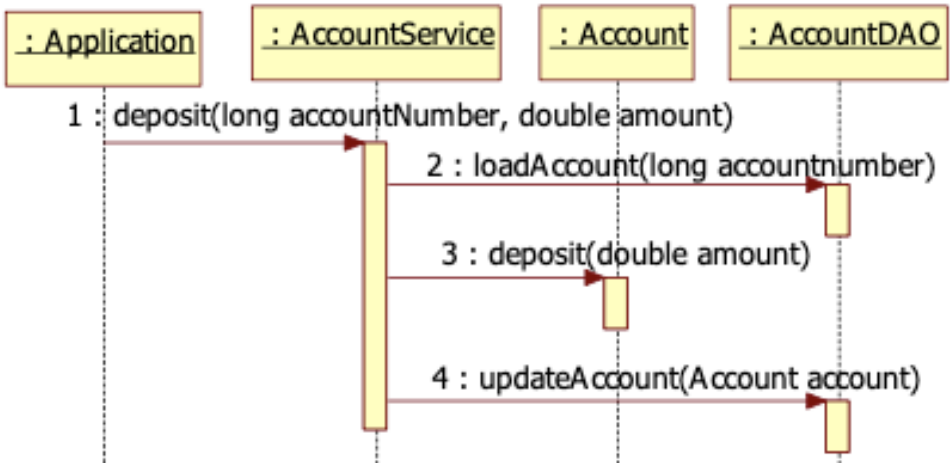
g. Suppose we have a webshop where the productcatalog consists of categories and products. Design this with the composite pattern and implement it in Java. Write code to test your application, for example print out the whole productcatalog with all categories and products



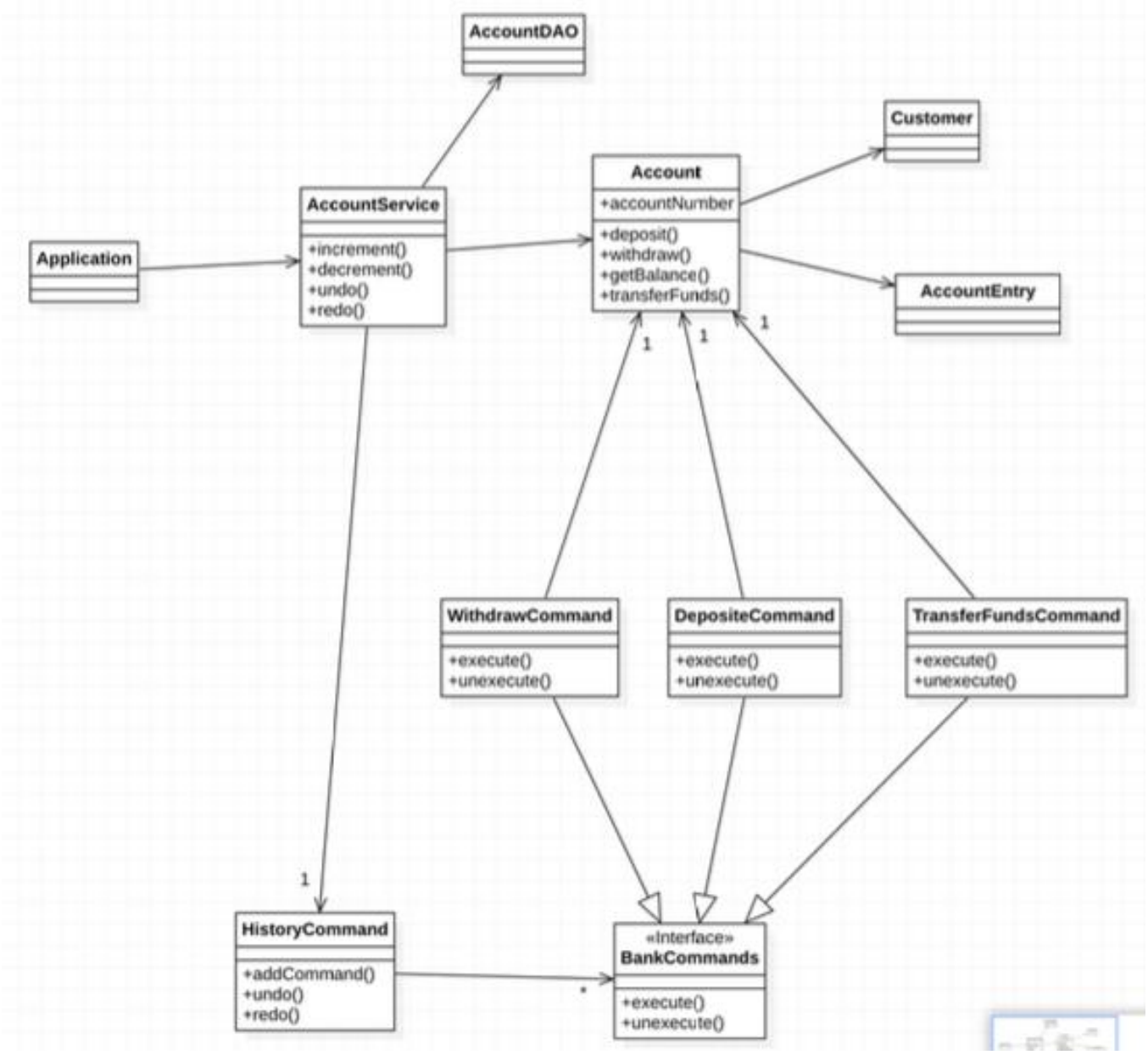
LAB5

a. Given is the following bank application:

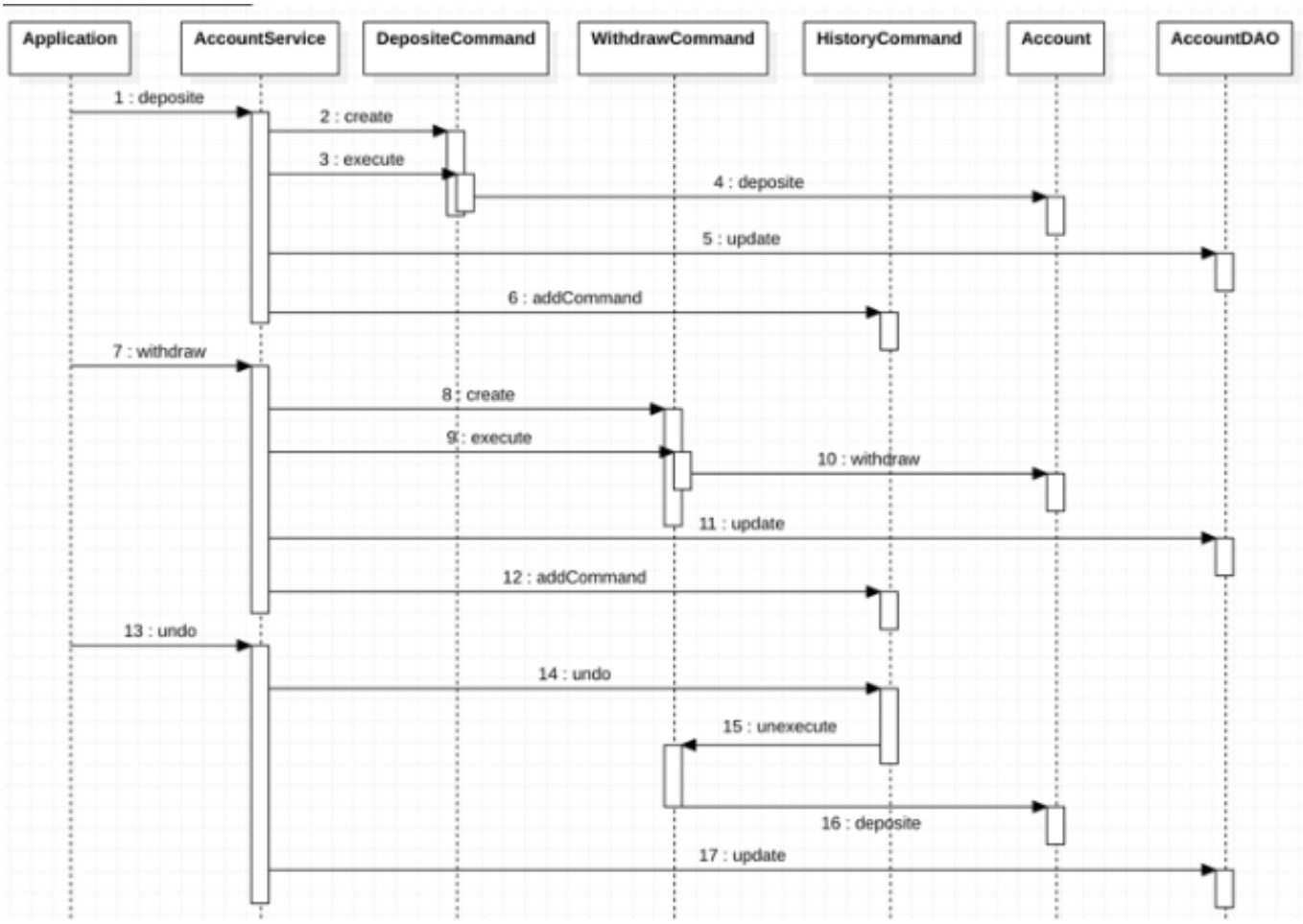




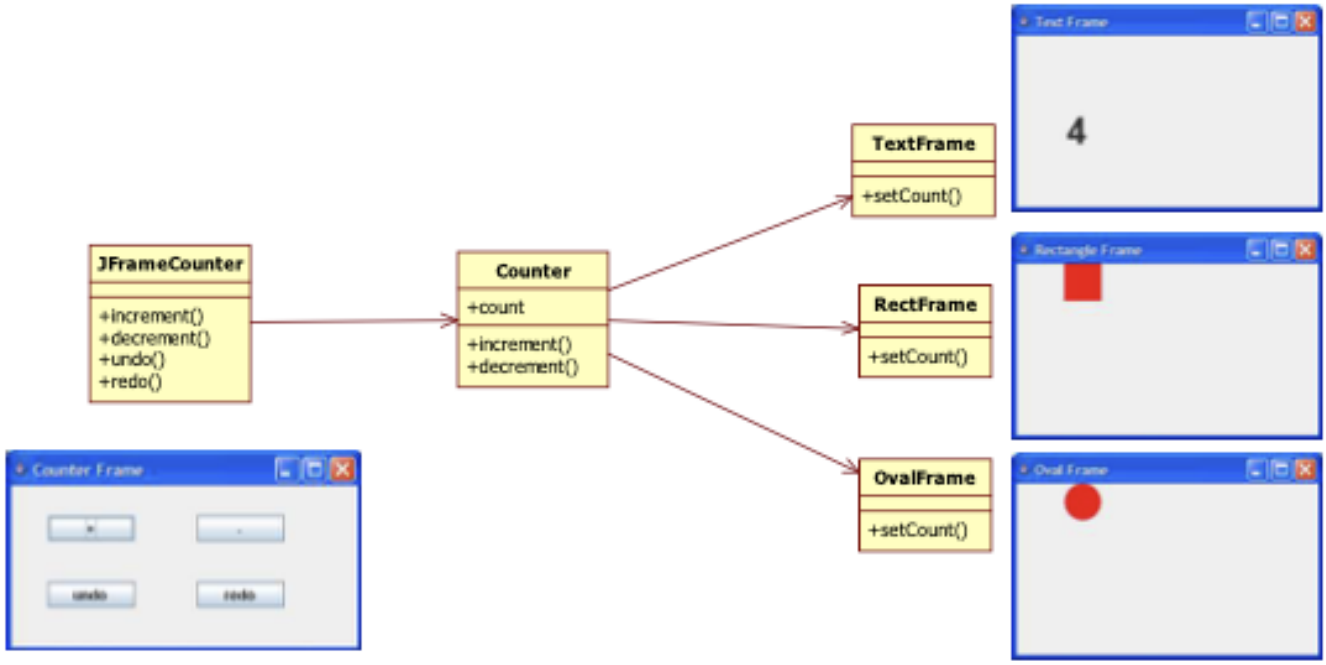
Now we want undo/redo functionality for the methods deposit(), withdraw() and transferFunds()
Draw the modified class diagram.



b. Draw a sequence diagram that shows how your new design works. On the sequence diagram show the following scenario:
   1. First deposit a certain amount
   2. Then withdraw a certain amount
   3. Then call undo

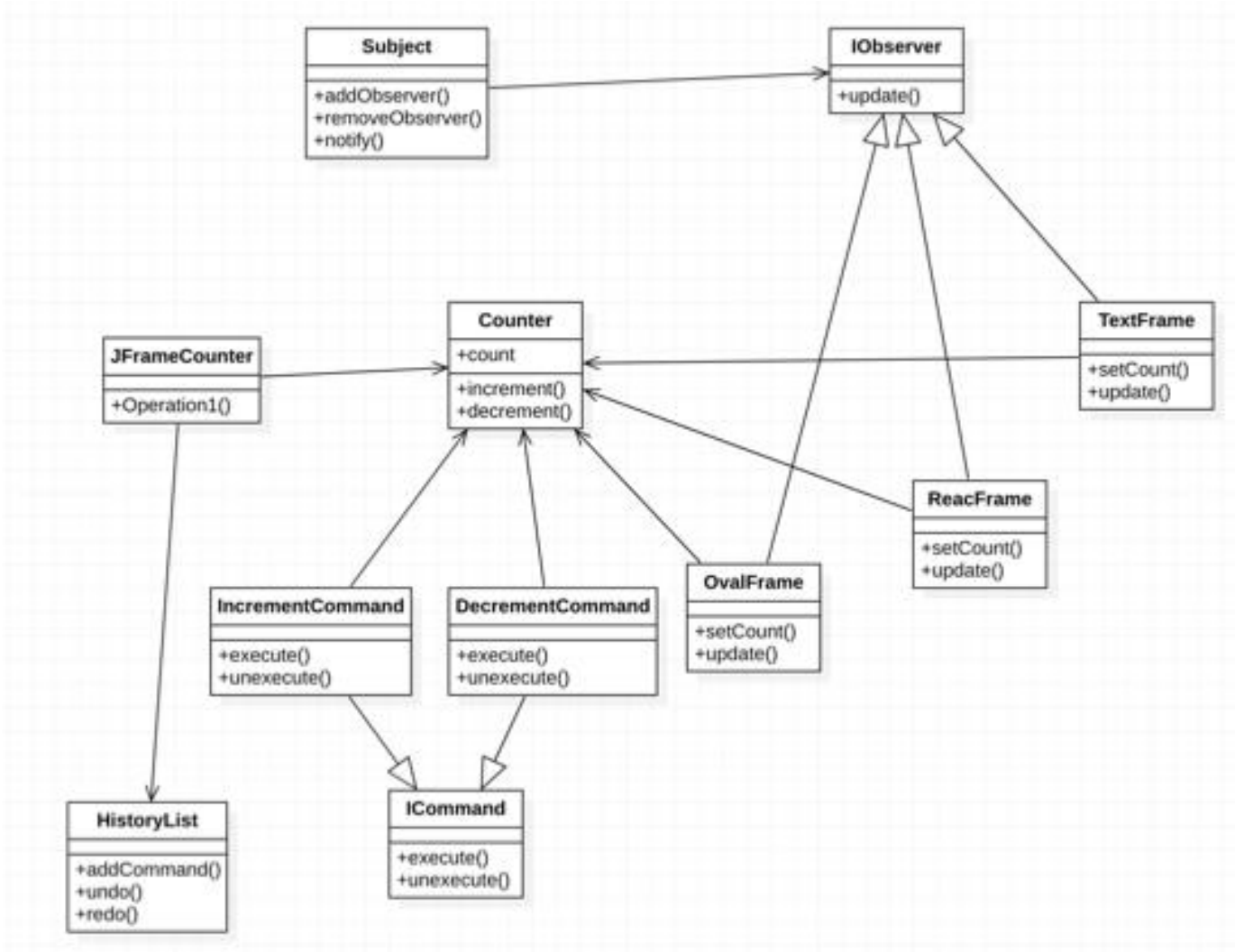


d. In lab 3 we applied the observer pattern to the following application:

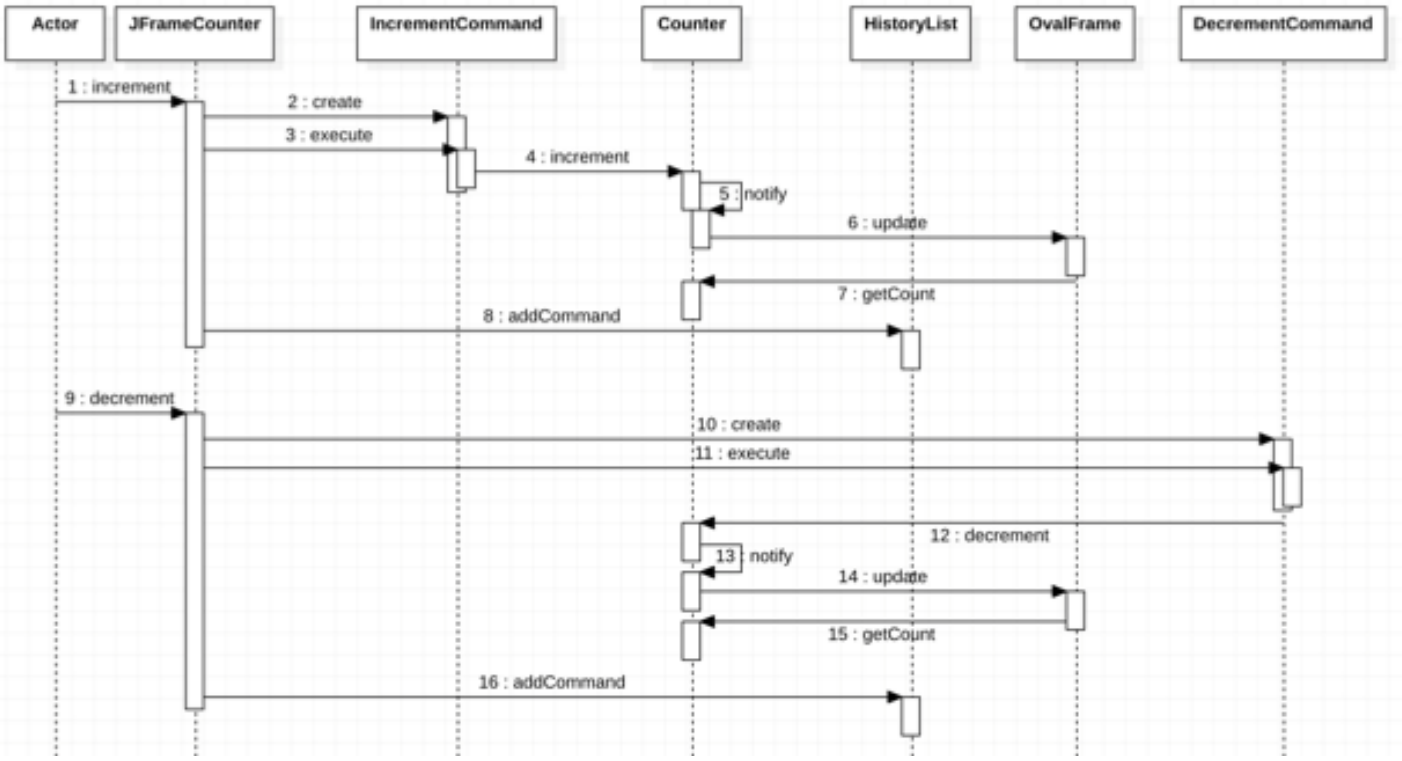


Now we want the undo/redo buttons to work.

Draw the class diagram so that the undo/redo buttons work correctly. Your class diagram needs to show both the observer pattern of lab 3 and the command pattern in one diagram.


Subject
+addObserver()
+removeObserver()
+notify()
IObserver
+update()
Counter
+count
+increment()
+decrement()
JFrameCounter
+Operation1()
TextFrame
+setCount()
+update()
ReacFrame
+setCount()
+update()
OvalFrame
+setCount()
+update()
IncrementCommand
+execute()
+unexecute()
DecrementCommand
+execute()
+unexecute()
ICommand
+execute()
+unexecute()
HistoryList
+addCommand()
+undo()
+redo()

e. Draw the sequence diagram that shows the following scenario:
   a. The user clicks the increment button
   b. The user clicks the decrement button
   c. The user clicks undo



LAB6

Given is the following simple Game application:

```
public class Game {
	private int totalPoints = 0;
	private int level = 1;

	public void play() {
		Random random = new Random();
		addPoints(random.nextInt(7));
		System.out.println("points="+totalPoints+" level="+level);
	}

	public int addPoints(int newPoints) {
		if (level == 1) {
			totalPoints = totalPoints + newPoints;
			if (totalPoints > 10) { // move to level 2
				level = 2;
				totalPoints = totalPoints + 1;//add 1 bonus point
			}
		} else if (level == 2) {
			totalPoints = totalPoints + 2 * newPoints;
			if (totalPoints > 20) { // move to level 3
				level = 3;
				totalPoints = totalPoints + 2; //add 2 bonus points
			}
		} else if (level == 3) {
			totalPoints = totalPoints + 3 * newPoints;
		}

		return totalPoints;
	}

}

public class Application {
	public static void main(String[] args) {
		Game game = new Game();
		game.play();
		game.play();
		game.play();
		game.play();
		game.play();
	}
}
```

Whenever we play the game, we get points. We can move up to a higher level of the game when we have a certain amount of points. In our current implementation we start with level 1, and if we have more than 10 points, we go to level 2. If we have more than 20 points in level 2, we move to level 3.

In level 1, our totalpoints is increased with the points we receive in each play. In level 2, our totalpoints is increased with 2 times the points we receive in each play. And in level 3, our totalpoints is increased with 3 times the points we receive in each play.
If you are upgraded to level 2, then you get 1 bonus point.
If you are upgraded to level 3, then you get 2 bonus points.

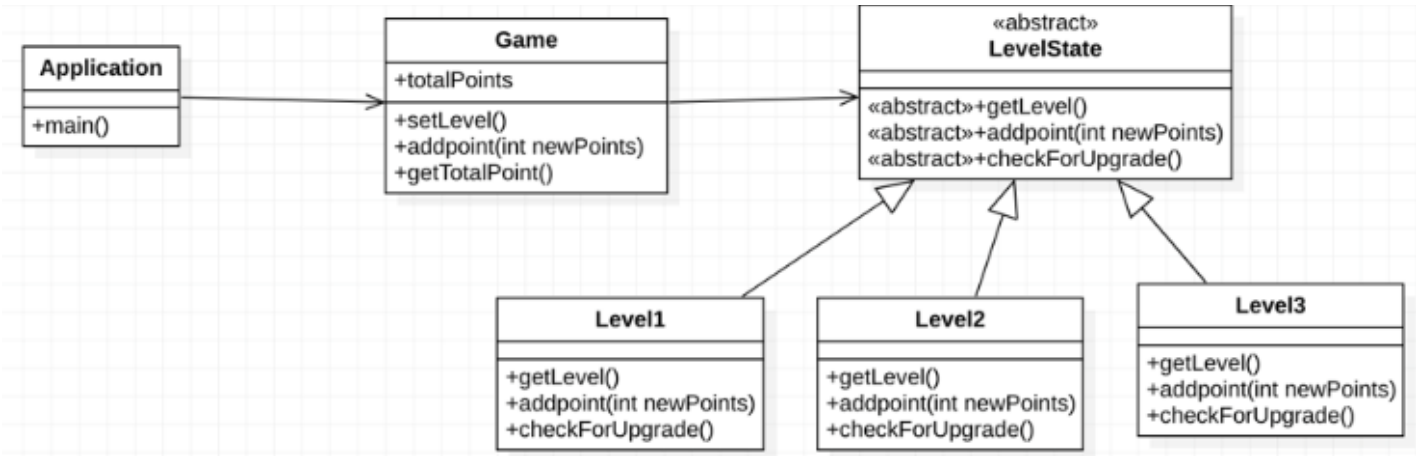
The problem with the given code is the addPoints() method. If we want to add another level, we have to change this method. If we want to change anything regarding point calculation rules, we always have to change this method. Just imagine how complex this method will be if we have 10 levels and very complex point calculations.
Redesign this application in such a way that

- It will be easier to add new levels
- It will be easier to change the point calculations
- The Game class is independent of the different levels. This means I should be able to add a new level without any change in the Game class.

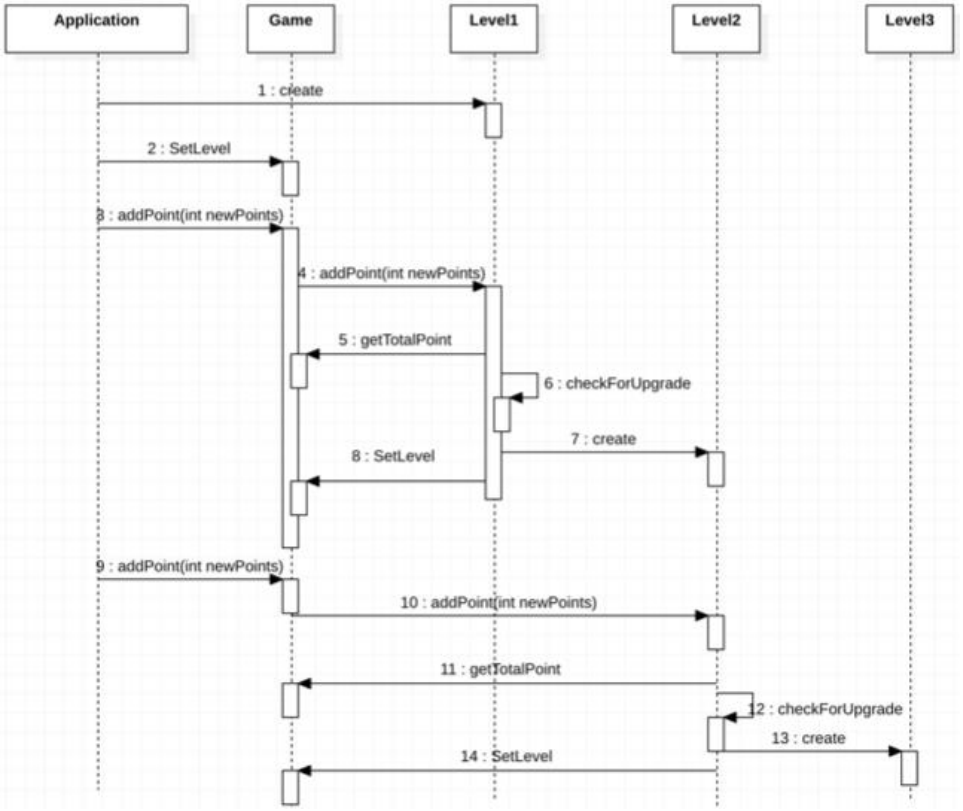
If you are done, add another level 2_5 with the following business rules:

- If you are in level 2, and you have more or equal to 15 points, then you go to level 2_5
- In level 2_5 you get 1 bonus point.
- If you are in level 2_5, and you have more or equal to 20 points, then you go to level 3

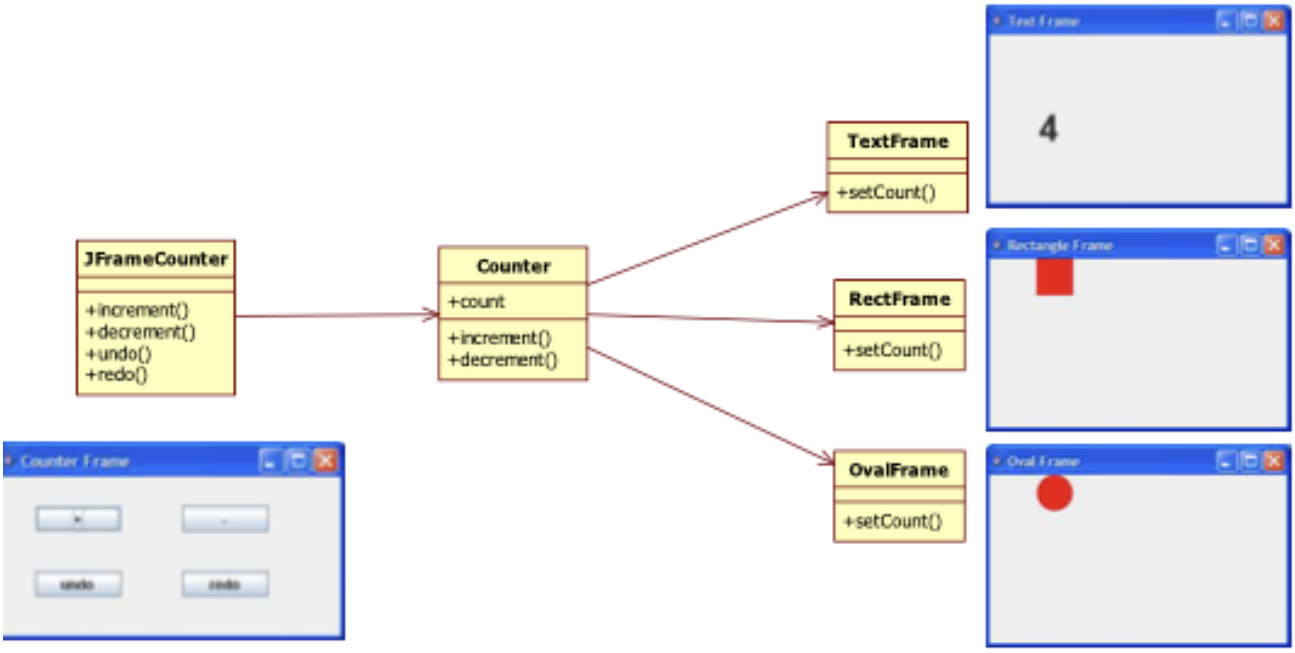
a. Draw the class diagram of your design.



b. Draw the sequence diagram that shows how your design works. Show in the sequence diagram how you move from level 1 to level 2.



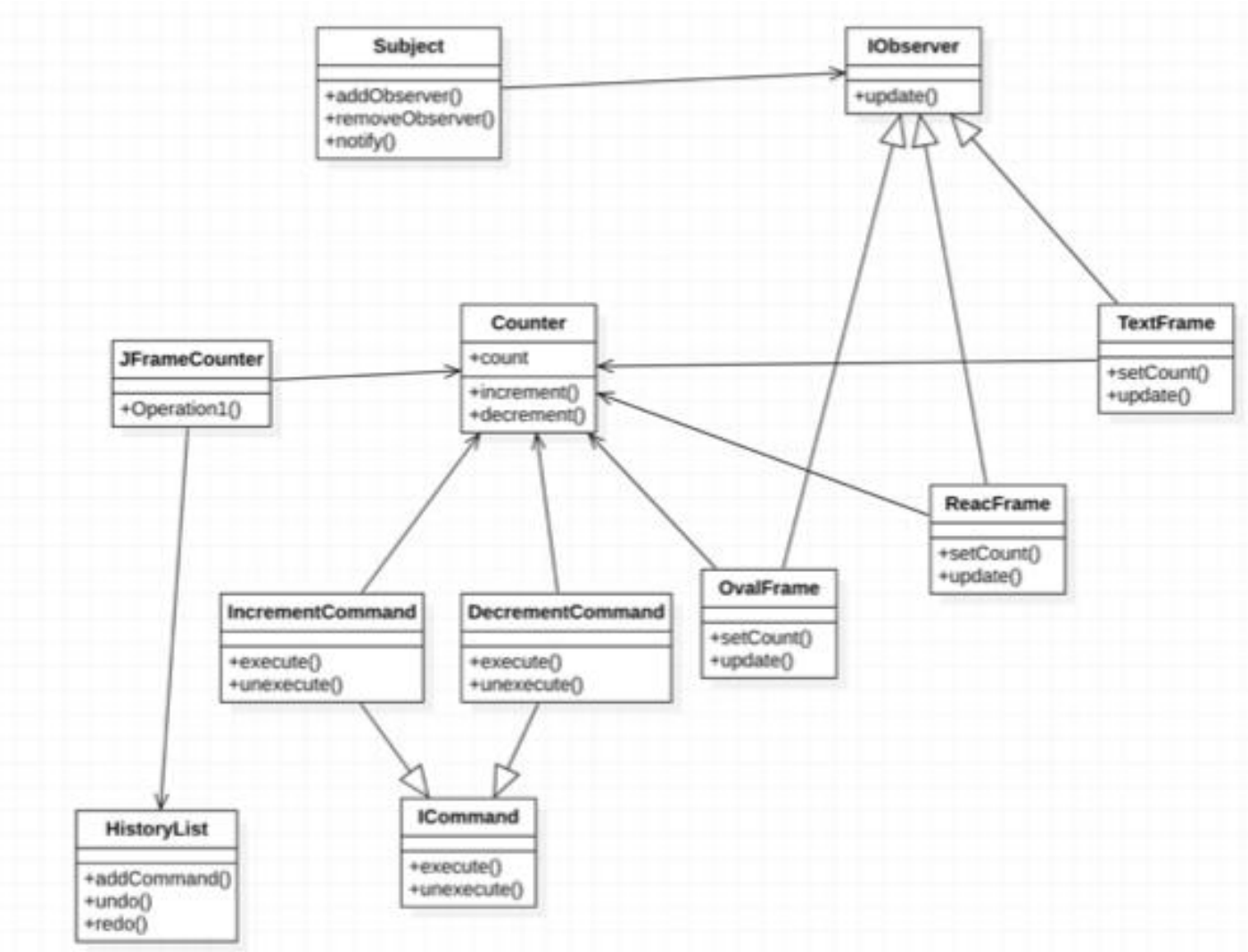
d. In lab 5 we applied the observer and command pattern to the following application:



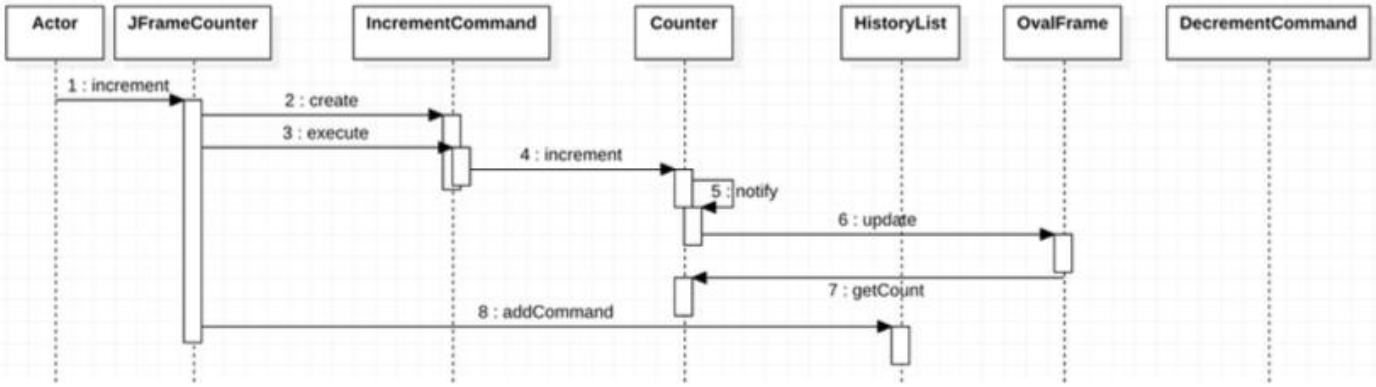
Now we want to add the following functionality to this application:

- When the Counter value is a single digit number, then every button action (increment and decrement) will add or subtract 1 point from the current teller Counter.
- When the Counter value is a double digit number, then every button action (increment and decrement) will add or subtract 2 points from the current Counter value.
- When the Counter value is a triple digit number, then every button action (increment and decrement) will add or subtract 3 points from the current Counter value.

Draw the class diagram. Your class diagram needs to show both the observer pattern of lab 3, the command pattern of lab 5 and the state pattern in one diagram.



e. Draw the sequence diagram that shows the following scenario:
   a. The user clicks the increment button
   b. The user clicks undo


Actor
JFrameCounter
IncrementCommand
Counter
HistoryList
OvalFrame
DecrementCommand
1 : increment
2 : create
3 : execute
4 : increment
5 : notify
6 : update
7 : getCount
8 : addCommand

LAB7

Write an application that handles camera information we receive from different camera's above or next to interstates and highways. We receive camera records with the attributes license plate, speed and camera id. Every camera has its own id. The cameras have built-in software to recognize license plates and to measure the speed of the cars. Our application has to handle these camera records in the following way:

We need to detect every car that is stolen. If we receive a camera record of a stolen car, we need to notify the police (using a system.out.println)

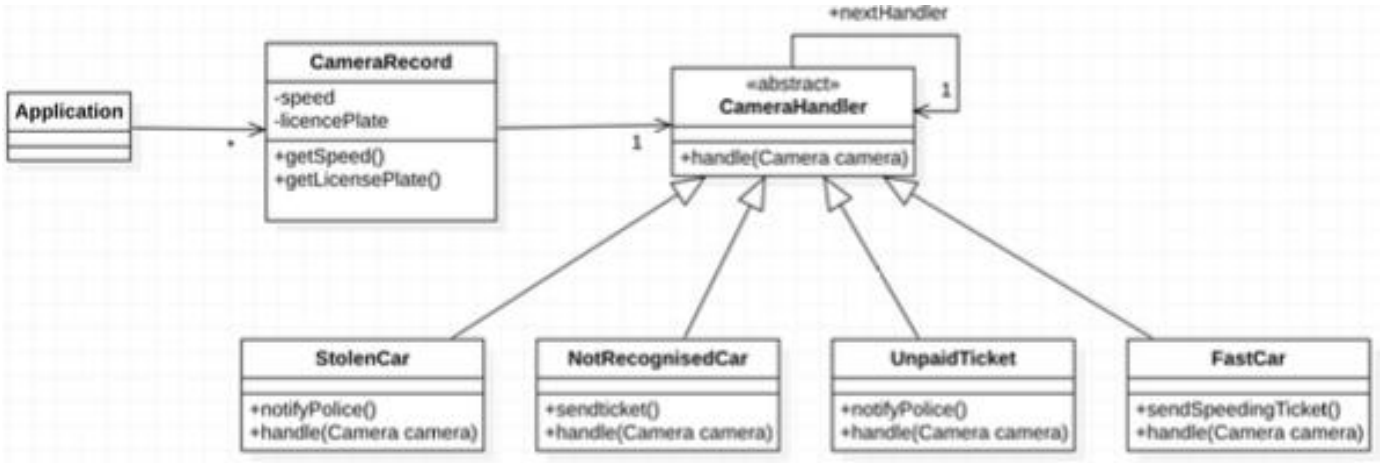
We need to detect every car that is speeding. If we receive a camera record of a car that drives too fast, we need to send the owner a speeding ticket (using a system.out.println)

We need to detect every car that is currently not registered. If we receive a camera record of a car that is currently not registered, we need to send the owner a ticket (using a system.out.println)

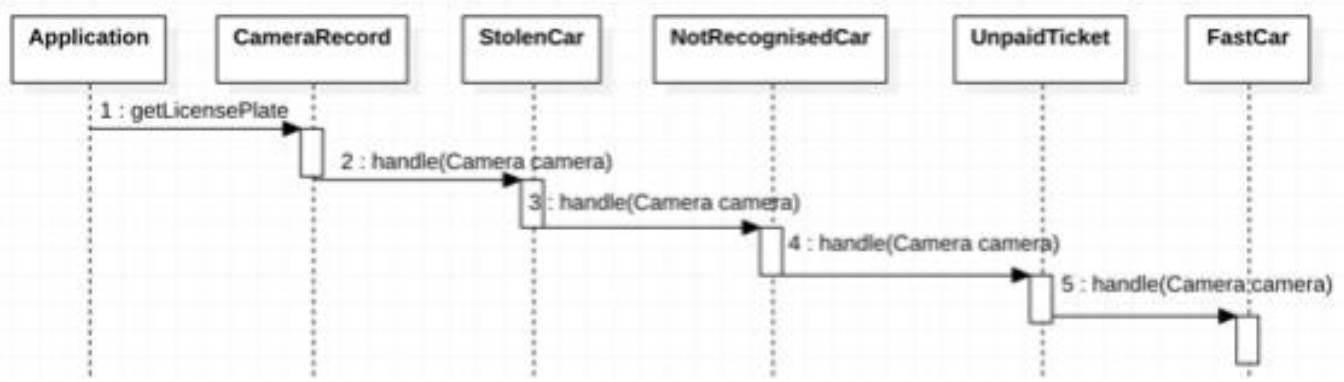
We also need to detect every car whose owner has one or more unpaid tickets. If we receive a camera record of a car whose owner has one or more unpaid tickets, we need to notify the police (using a system.out.println)

It should be easy to add other detection logic based on the camera records we receive in the system

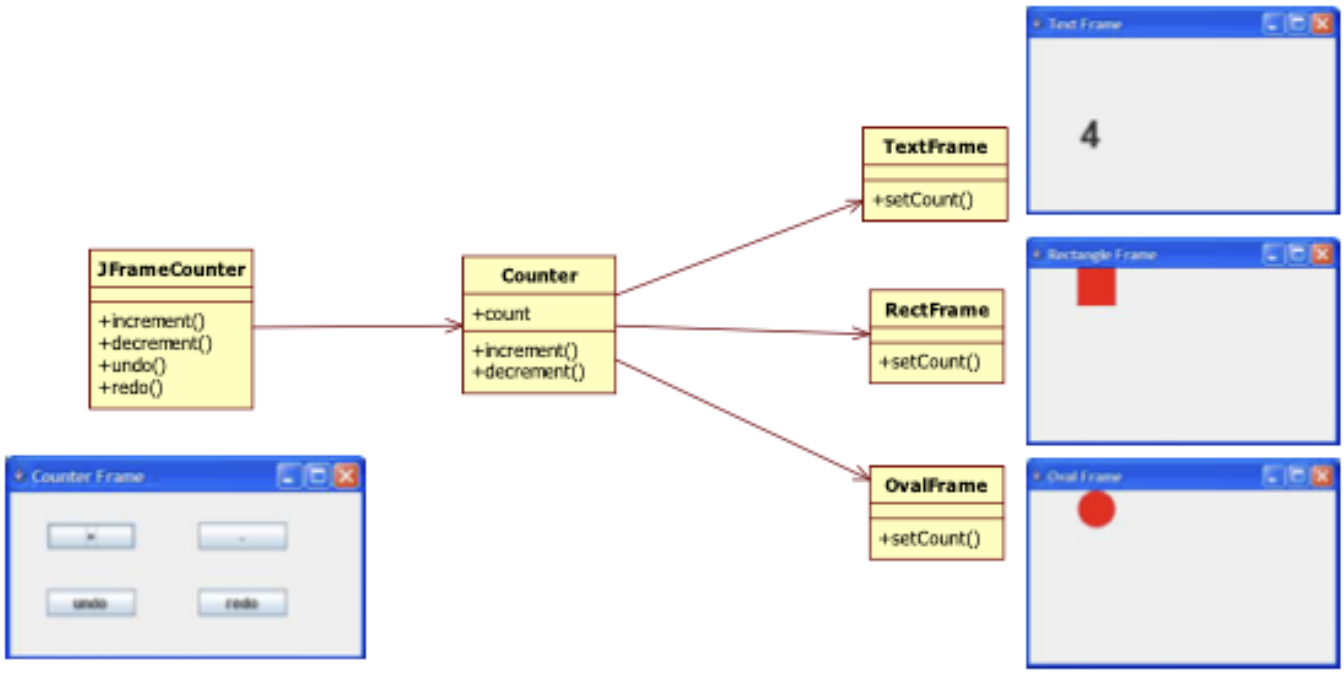
a. Draw the UML class diagram of your system



b. Draw the sequence diagram that show how your design works



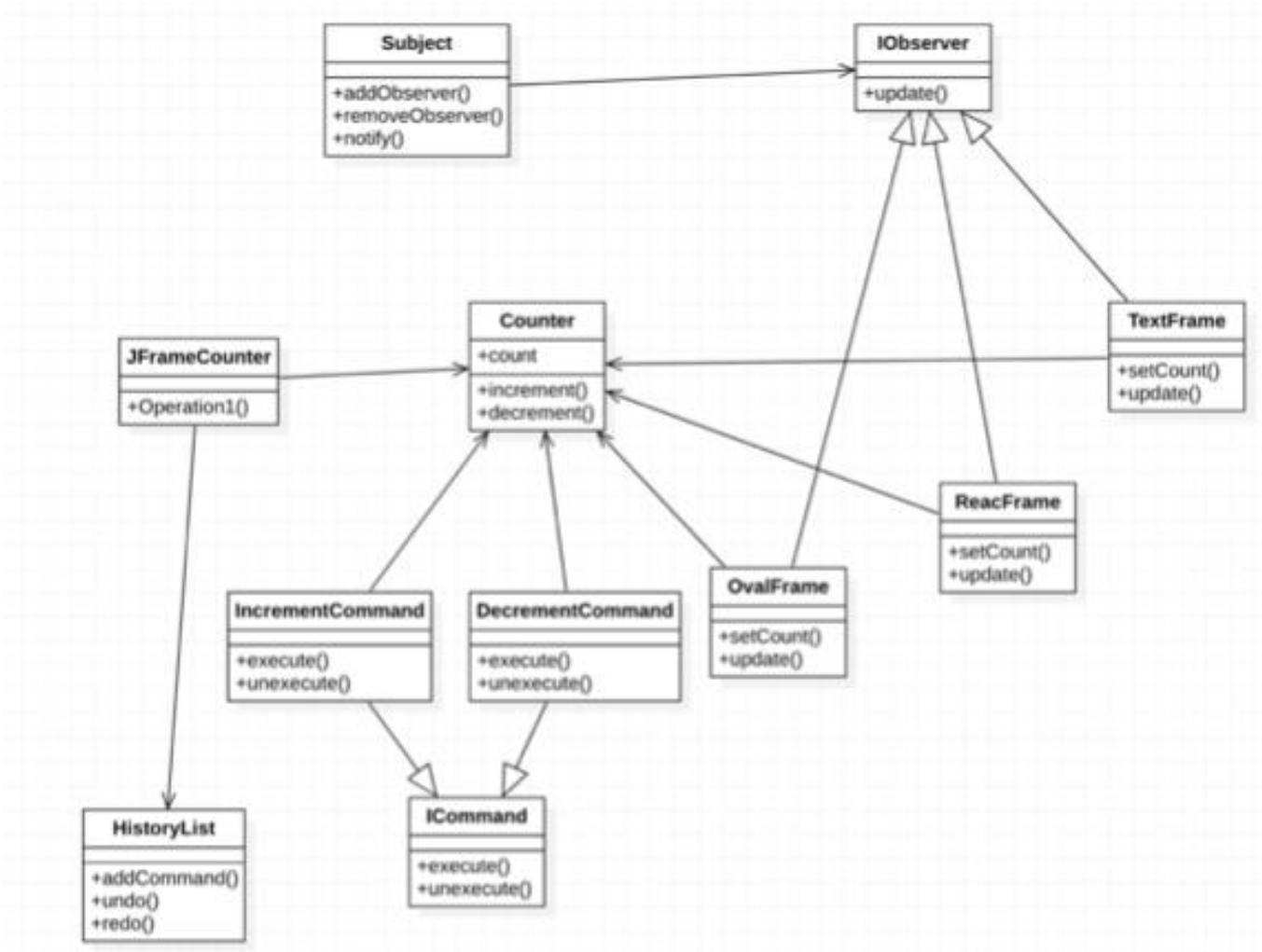
d. In lab 6 we applied different pattern to the counter application:



Now we get the following additional requirements:

- Whenever the counter value changes we want to do the following:
  - If the counter value is even and the counter value < 10 or if the counter value is even and the counter value =12 or 13 then we print in the console : "**Red**"
  - If the counter value is even and the counter value >= 10 but not 12 or 13 then we print in the console : "**Green**"
  - If the counter value is odd and the counter value < 15 or if the counter value is odd and the counter value =17 or 19 then we print in the console : "**Blue**"
  - If the counter value is odd and the counter value >= 15 but not 17 or 19 then we print in the console : "**Orange**"
- The counter value needs to be stored in the database. You don't need to do actual database access, but you can simulate that with a System.out.println()
- Anytime the counter value changes, we want to write that to a log file (you can simulate this with a System.out.println()
- We want a clear separation between UI, domain and database access

Draw the class diagram based on the solution of the previous lab. Your class diagram also needs to show the patterns we used in the previous labs.


Subject
+addObserver()
+removeObserver()
+notify()
IObserver
+update()
Counter
+count
+increment()
+decrement()
JFrameCounter
+Operation1()
TextFrame
+setCount()
+update()
ReacFrame
+setCount()
+update()
OvalFrame
+setCount()
+update()
IncrementCommand
+execute()
+unexecute()
DecrementCommand
+execute()
+unexecute()
ICommand
+execute()
+unexecute()
HistoryList
+addCommand()
+undo()
+redo()

e. Draw the sequence diagram that shows the following scenario:
   a. The user clicks the increment button
   b. The user clicks undo