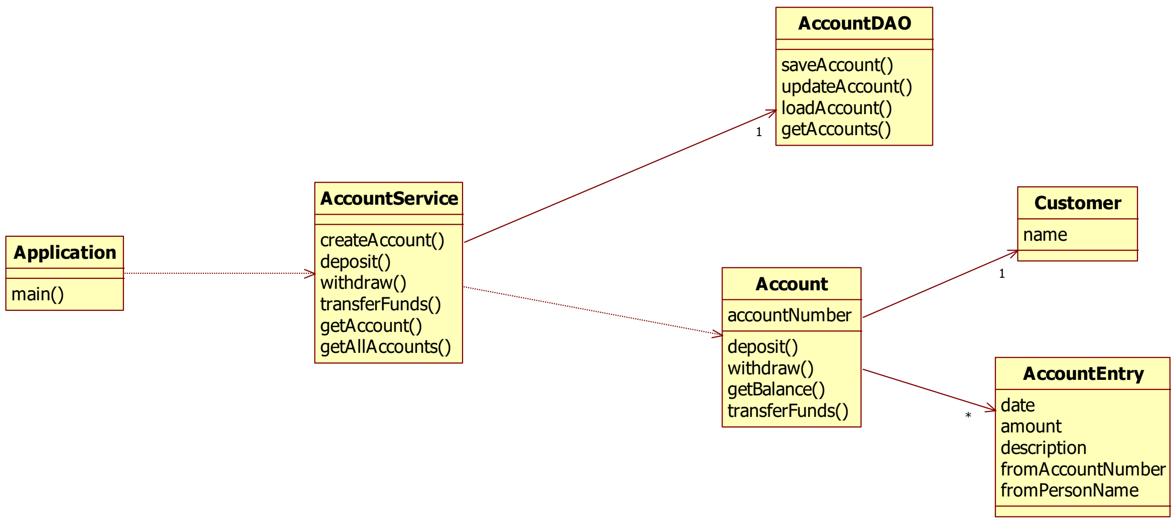
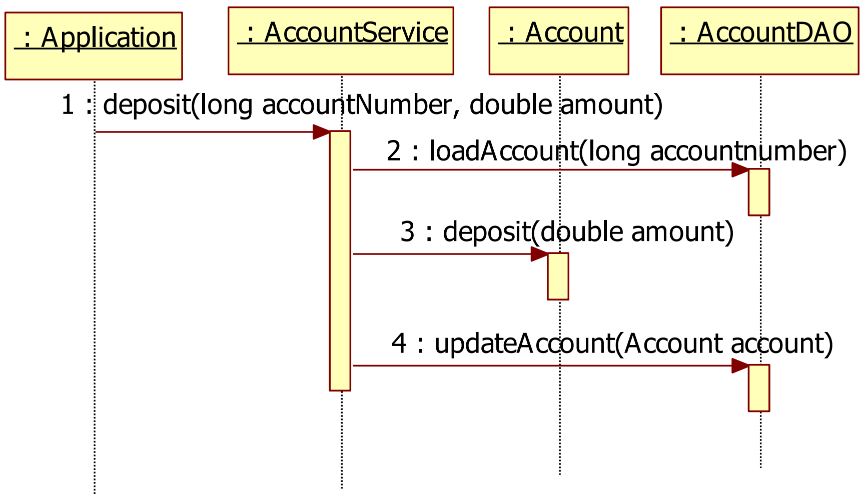
Lab 4

a. Given is the following bank application:





: Application
: AccountService
: Account
: AccountDAO
1 : deposit(long accountNumber, double amount)
2 : loadAccount(long accountnumber)
3 : deposit(double amount)
4 : updateAccount(Account account)

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

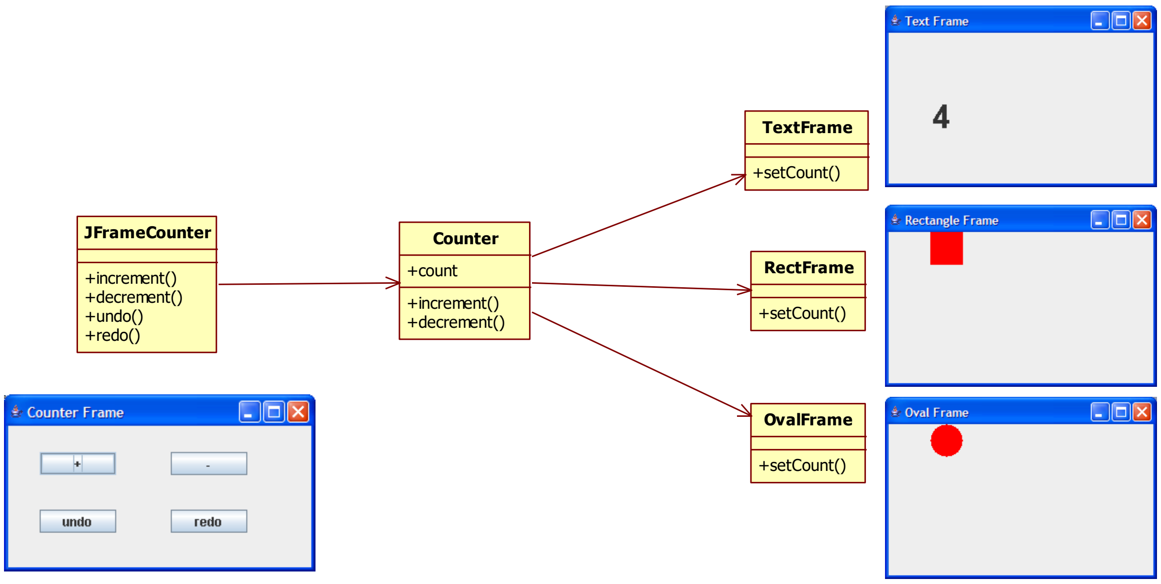
- Add a Logger class that logs every change to an Account. (The Logger should do a simple System.out.println() to the console)
- Add a SMSSender that sends a SMS at every change to an Account. (The SMSSender should do a simple System.out.println() to the console)
- Add an EmailSender that sends an email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

a. Draw the modified class diagram with the observer pattern (using the pull model) applied.

b. 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

c. Implement the observer pattern using the pull model in the given code.

In the previous lab the following counter application was given:



The problem with this 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.

d. Draw the class diagram of a better design (using the push model) so that it will be much easier to add different views of the counter value. So your diagram should

e. Draw the sequence diagram (using the push model) that shows the following scenario:
   1. The user clicks the increment button
   2. The user clicks the decrement button

f. Implement your new design in the given code in Java. Your solution should only contain the observer pattern (using the push model) and not the command pattern.

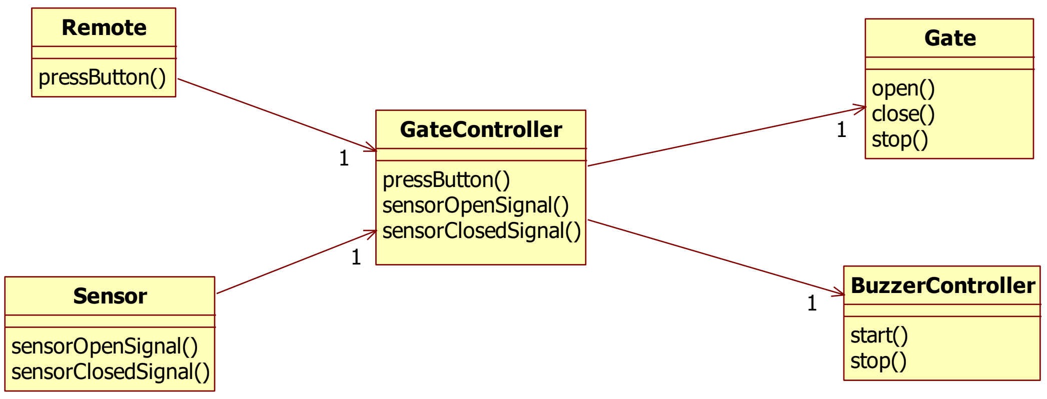
g. Now modify the solution of part e such that your solution contains both the command and the observer pattern in the same application

h. Suppose you have to design the controller software for an automatic gate. The gate controller application has the following requirements:

We have a remote control with only one button.
If we press the button on the remote control, and the gate is open, then the gate should be closing.
If we press the button on the remote control, and the gate is closed, then the gate should be opening.
If we press the button on the remote control, and the gate is opening, then the gate should be closing.
If we press the button on the remote control, and the gate is closing, then the gate should be opening.

We also have a gate sensor that signals our application if the gate is completely closed or opened.
Because we want to make the automatic gate as safe as possible, we also have an audio buzzer that makes noise when the gate is opening or closing. When the gate is completely open or closed, the buzzer is idle.

Your first design might look as follows:



Now you receive a new requirement from marketing. When the gate is busy with opening or closing, you also want to do the following:

1. When the gate is closing, a light next to the gate flashes in a red color.
2. When the gate is opening, a light next to the gate flashes in an orange color.
3. When the gate is completely open or closed, the light is off.
4. It should be easy to add other classes that do something when the state of the gate changes.

Implement your new design in Java using the PropertyChangeListener. Use System.out.println() to show the state of the gate and the behavior of the buzzer and the light.

What to hand in?

1. A jpeg picture of part a, b, d and e
2. A zip file containing the project of part c, f and g and h