|  |  |
| --- | --- |
| Course | Advanced Software Design – CS525 |
| Assignment | Lab 3 |
| Week | 03 |
| Due | Mar 2, 2020 |
| Student name | Quan Hong Doan |
| Student ID | 986956 |

Online version:

<https://github.com/zithiat/asd/blob/master/labs%20%26%20assignments/Assignment_CS525_Week03_986956.docx>

**Problem 1**:

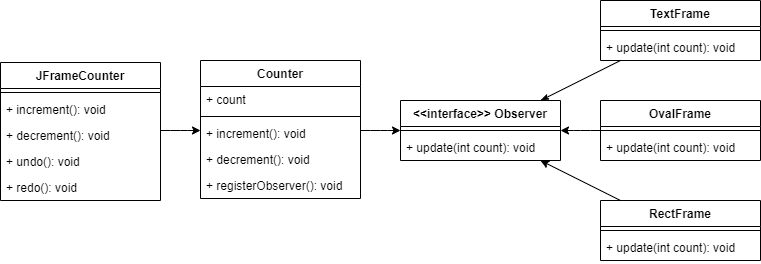
Problem 1 is the question a, b, c from the PDF file.

**Answer**:

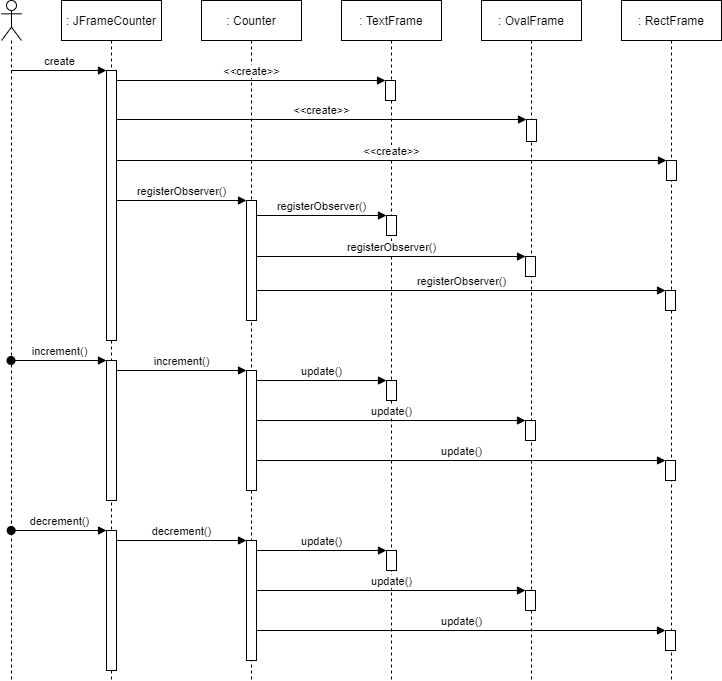
To reduce the risk of modifying the Counter class, we will create an interface as Observer with a single *update()* method. Then those frame classes like TextFrame, OvalFrame, RectFrame will implements from that Observer interface with the *update()* method.

Then in the Counter class, we will register all the Observer frame classes. When there’s any update from JFrameCounter for incrementing or decreasing the counter, we simply call updating for all observers.

**Modified class diagram**



**Sequence diagram** will be as below.



The modified code is online <https://github.com/zithiat/asd/tree/master/codes/codeforlabs/lab3/counter>

**JFrameCounter** modification:

**public** JFrameCounter() {

**try** {

jbInit();

counter = **new** Counter();

TextFrame textframe = **new** TextFrame();

textframe.setVisible(**true**);

RectFrame rectframe = **new** RectFrame();

rectframe.setVisible(**true**);

OvalFrame ovalframe = **new** OvalFrame();

ovalframe.setVisible(**true**);

counter.registerObserver(textframe);

counter.registerObserver(rectframe);

counter.registerObserver(ovalframe);

**Counter** modification:

**public** **class** Counter {

**private** **int** count = 0;

List<Observer> observerList;

**public** Counter() {

observerList = **new** ArrayList<Observer>();

}

**public** **void** registerObserver(Observer o) {

**this**.observerList.add(o);

}

**public** **void** increment() {

count++;

**this**.observerList.stream().forEach(e -> e.update(count));

}

**public** **void** decrement() {

count--;

**this**.observerList.stream().forEach(e -> e.update(count));

}

}

**Observer** interface:

**interface** Observer {

**public** **abstract** **void** update(**int** counter);

}

**TextFrame**, **OvalFrame**, and **RectFrame** modification:

// public void setCount (int cnt){

// count=cnt;

// repaint();

// }

@Override

**public** **void** update(**int** counter) {

**this**.count = counter;

repaint();

}

**Problem 2**:

Problem 2 is the combination of question d, e, f, and g.

**Answer**:

As we need to record different message for different actions, we will be using PULL mechanism. That means we will create few classes:

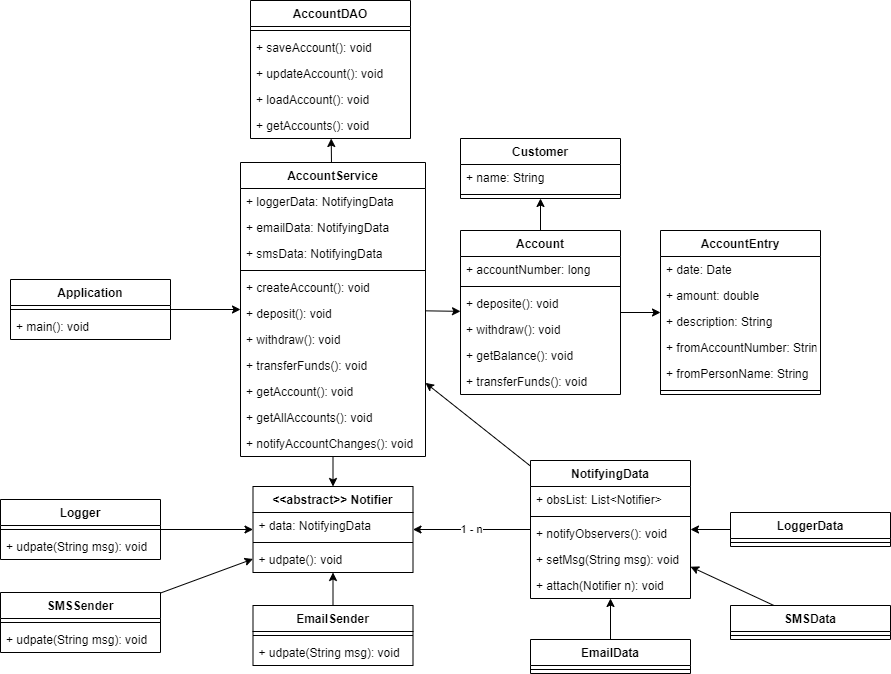
* Abstract Notifier class, which will be extended in Logger, SMSSender, and EmailSender.

When we initiate the notifier classes, we will attach a corresponding NotifyingData object with these.

* NotifyingData, which will notify to the observers, and add notifier into the notifier list.

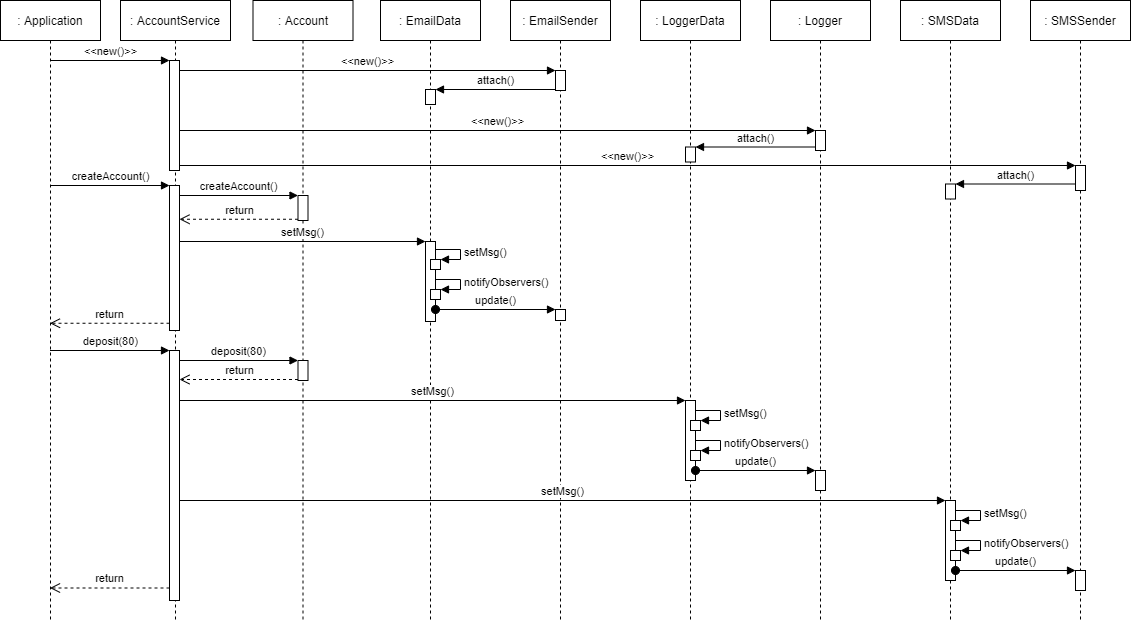
When there’re any messages, the NotifyingData will set the message and notify the observers. The observers will pull the data from the NotifyingData object and print out (in our example).

**Modified class diagram**:



**Sequence diagram** for:

* Account creation
* Deposit $80 on the newly created account



Implementation:

Notifier class, and Logger class (same for SMSSender and EmailSender)

**abstract** **class** Notifier {

NotifyingData data;

**abstract** **void** update();

}

**public** **class** Logger **extends** Notifier{

**public** Logger(NotifyingData nd) {

**this**.data = nd;

**this**.data.attach(**this**);

}

@Override

**public** **void** update() {

System.***out***.println("Logger: " + data.msg);

}

}

NotifyingData class, and its extended class – SMSData

**public** **class** NotifyingData {

**private** List<Notifier> obsList;

String msg;

**public** **void** notifyObservers() {

**this**.obsList.stream().forEach(e -> e.update());

}

**public** **void** setMsg(String msg) {

**this**.msg = msg;

}

**public** **void** attach(Notifier n) {

**if** (**null** == obsList)

obsList = **new** ArrayList<Notifier>();

**this**.obsList.add(n);

}

}

**public** **class** SMSData **extends** NotifyingData {

**public** SMSData() {

**this**.msg = "SMS: " + msg;

}

}

Our AccountService class will be as below. Online version: <https://github.com/zithiat/asd/tree/master/codes/codeforlabs/lab3/bank>

**public** **class** AccountService **implements** IAccountService {

…

**private** NotifyingData emailData = **new** EmailData();

**private** NotifyingData smsData = **new** SMSData();

**private** NotifyingData loggerData = **new** LoggerData();

**public** AccountService() {

…

// initiate the observers

**new** Logger(loggerData);

**new** EmailSender(emailData);

**new** SMSSender(smsData);

}

**public** Account createAccount(**…**) {

…

emailData.setMsg("New account created: " + accountNumber);

emailData.notifyObservers();

…

}

**public** **void** deposit(**…**) {

…

notifyChanges("Deposit " + amount + " to account " + accountNumber);

}

…

**public** **void** withdraw(**…**) {

…

notifyChanges ("Withdraw " + amount + " from account " + accountNumber);

}

**public** **void** transferFunds(**…**) {

…

notifyChanges ("Transfer funds " + amount + " from account " + fromAccountNumber + " to account " + toAccountNumber);

}

**private** **void** notifyChanges(String msg) {

loggerData.setMsg(msg);

smsData.setMsg(msg);

loggerData.notifyObservers();

smsData.notifyObservers();

}

}