HOMEWORK ASSIGNMENT #1

PROBLEM #1 (40 points)
A system keeps track of prices of machine parts in a warehouse. Prices of machine parts in the warehouse may change. Interested users may register with the system to watch price changes of a specific machine part in the warehouse. Each machine part is uniquely identified by its name and ID number. At any time different types of users may register or un-register to watch the price of a specific machine part in the warehouse. Each time when the price of a machine part changes, the system notifies all users registered to watch the price change of this machine part.

a. Develop a class diagram for the system using the **Observer** design pattern. For each class list all operations with parameters and briefly specify them using pseudo-code. In addition, for each class provide its attributes. Make the necessary assumptions for your design. Notice that the components in your design should be de-coupled as much as possible. In addition, components should have high cohesion.

b. Provide two sequence diagrams showing:
    - How the user registers to observe a price change of a selected machine part.
    - How the system notifies users about price change of a specific machine part.

PROBLEM #2 (60 points)

The *ACCOUNT* component supports the following operations:

```
OPEN (int p, int y, int a)     // open an account where p is a pin, y is an user's
                                  identification #, and a is a balance
PIN (int x)                    // provides pin #
DEPOSIT (int d);               // deposit amount d
WITHDRAW (int w);              // withdraw amount w
BALANCE ();                    // display the current balance
LOGIN(int y)                   // login where y is a client's identification #
LOGOUT()                       // logout from the account
suspend()                      // suspends an account
activate()                     // activates a suspended account
close()                        // an account is closed
```

The *ACCOUNT* component is a state-based component and supports three types of transactions: withdrawal, deposit, and balance inquiry. Before any transaction can be performed, operation *OPEN*(*p*, *y*, *a*) must be issued, where *y* is a client's identification #, *p* is a pin used to get permission to perform transactions and *a* is an initial balance in the account. It is assumed that *OPEN()* operation is issued only once for a given account. Before any transaction can be performed, operation *LOGIN(y)* must be issued (where *y* is a client's identification #) followed by *PIN(x)* operation. The *PIN(x)* operation must contain the valid pin # that must be the same as the pin # provided in *OPEN*(*p*, *y*, *a*) operation. There is a limit on the number of attempts with an invalid pin. The account can be overdrawn. If the account is overdrawn then the withdrawal transaction cannot be performed. The account may become suspended by *suspend()* operation. If the account is suspended, withdrawal, deposit, and logout transactions cannot be performed. A suspended account can be activated by *activate()* operation. In addition, a suspended account can be closed by *close()* operation. The detailed behavior of the *ACCOUNT* component is specified using EFSM. The EFSM of Figure 1 shows the detail behavior of the *ACCOUNT* component.

Design the system using the **State design pattern**. Provide two solutions:

- **de-centralized** version of the State pattern
- **centralized** version of the State pattern

Notice that the components in your design should be de-coupled as much as possible. In addition, components should have high cohesion.

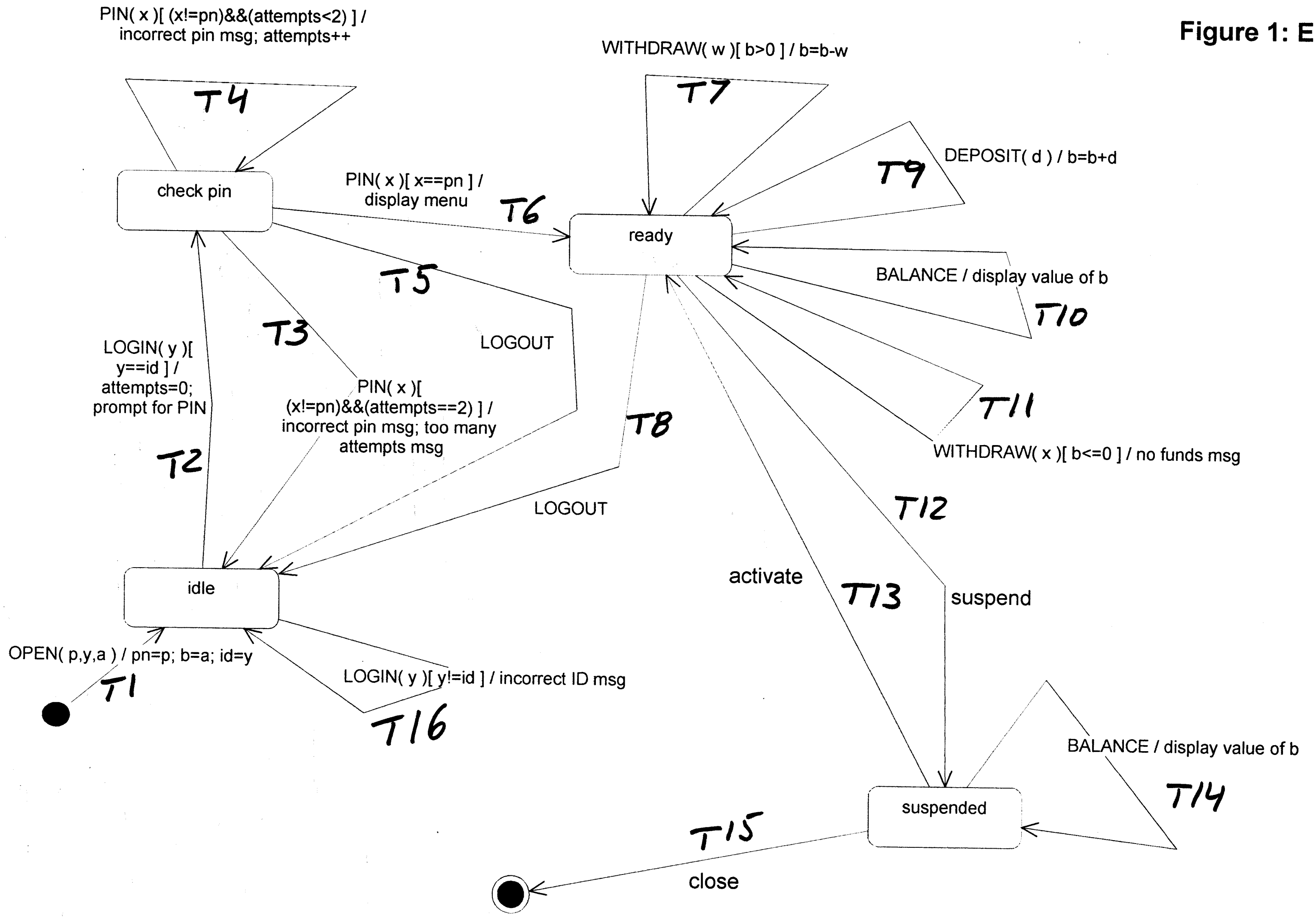
For each solution:

a. Provide a class diagram for the system. For each class list all operations with parameters and specify them using **pseudo-code**. In addition, for each class provide its attributes and data structures. Make the necessary assumptions for your design.

b. Provide a **sequence diagram** for the following operation sequence:
OPEN(123,111,1000), LOGIN(111), PIN(123), DEPOSIT(200), BALANCE(), suspend(), close()

Notice when the EFSM model is "executed" on this sequence of operations, the following sequence of transitions is traversed/executed: T_1, T_2, T_6, T_9, T_{10}, T_{12}, T_{15}

Figure 1: EFSM of ACCOUNT


PIN(x)[(x!=pn)&&(attempts<2)] / incorrect pin msg; attempts++
T4
check pin
PIN(x)[x==pn] / display menu
T6
WITHDRAW(w)[b>0] / b=b-w
T7
DEPOSIT(d) / b=b+d
T9
ready
BALANCE / display value of b
T10
T5
LOGOUT
T3
LOGIN(y)[y==id] / attempts=0; prompt for PIN
PIN(x)[(x!=pn)&&(attempts==2)] / incorrect pin msg; too many attempts msg
T11
WITHDRAW(x)[b<=0] / no funds msg
T8
T2
LOGOUT
T12
idle
activate
T13
suspend
OPEN(p,y,a) / pn=p; b=a; id=y
T1
LOGIN(y)[y!=id] / incorrect ID msg
T16
BALANCE / display value of b
T14
suspended
T15
close