

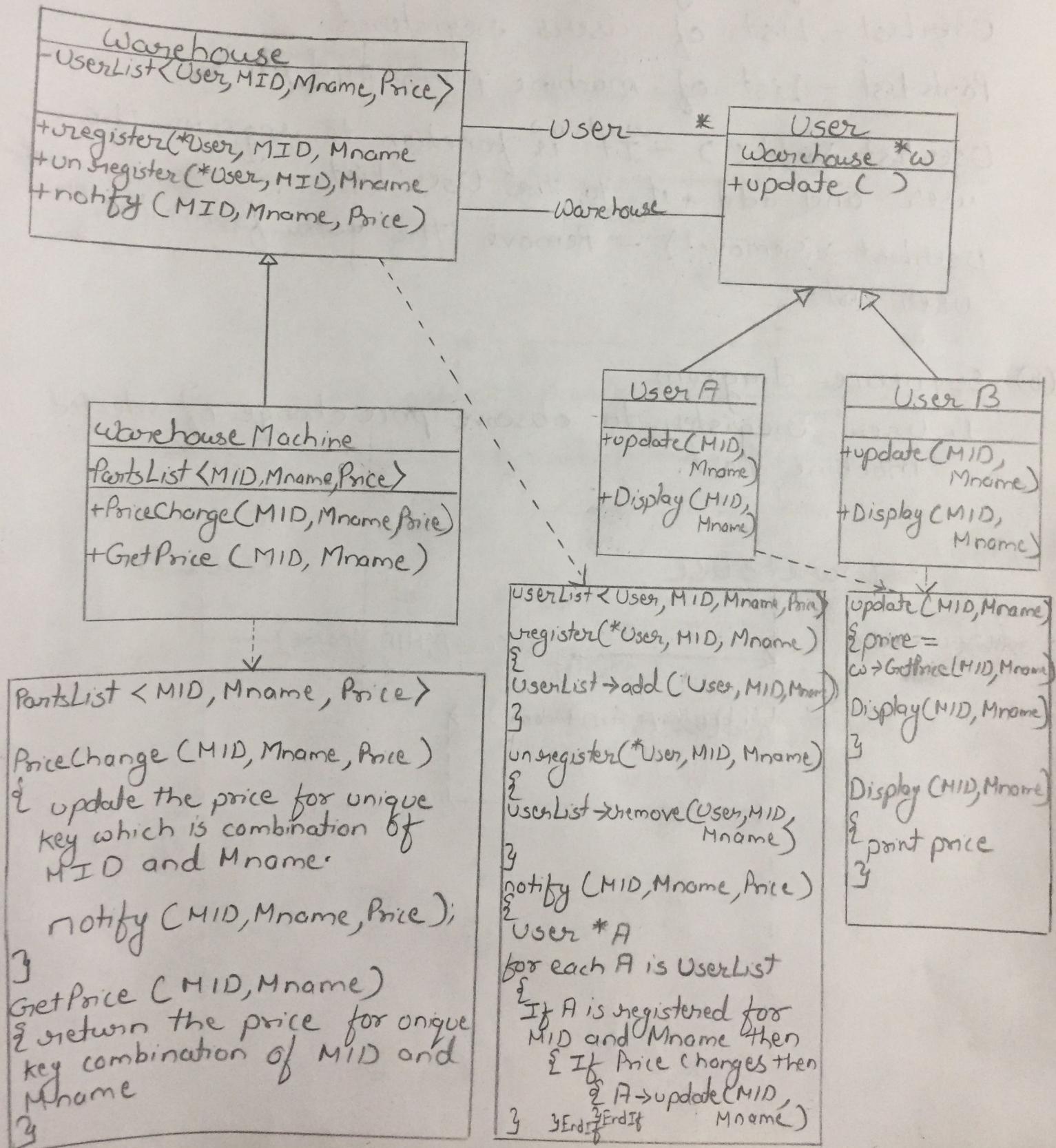
Name - Abhinav Pimpalgaonkar  
 Student Id - A20387324

(1)

## Assignment - 1

### Problem 1

Observer Pattern for Warehouse - Class diagram



### \* Key Points

MID - Id number for machine Part

Mname - Machine Part Name

Price - Price for machines parts which is uniquely identified by MID and Mname

UserList - List of users registered

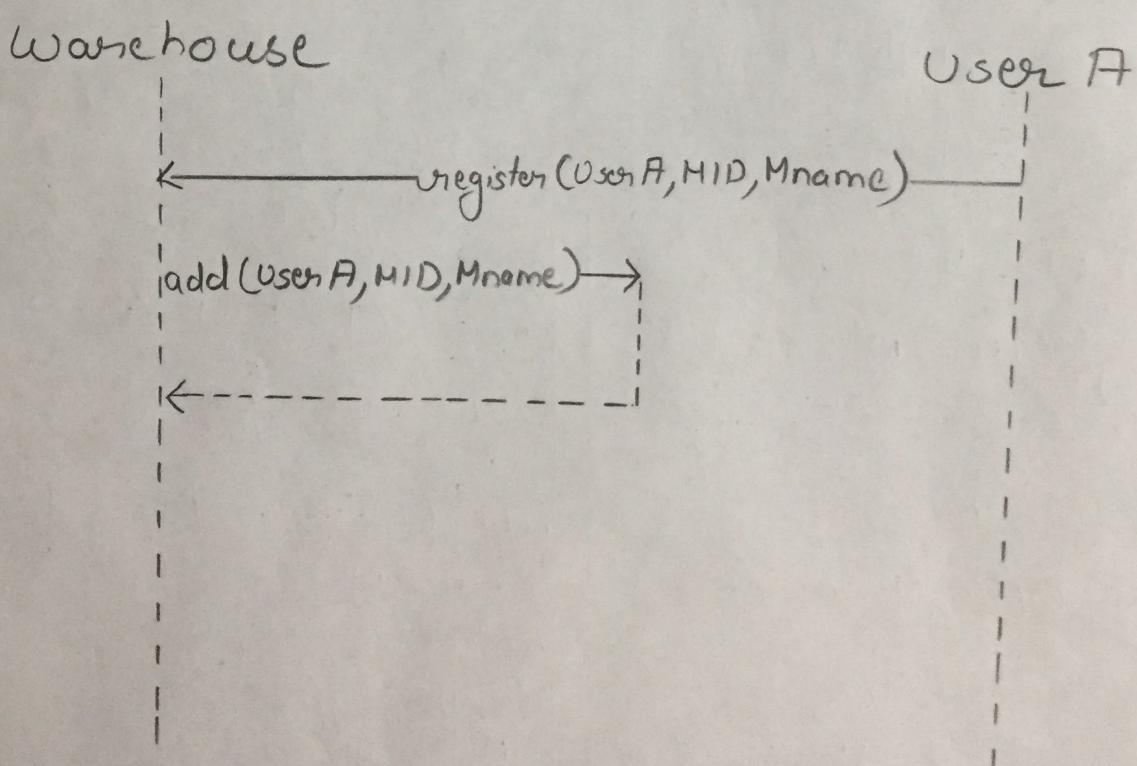
PartsList - List of machine parts and price

UserList → add() - It is function to register the user and add it to the User List

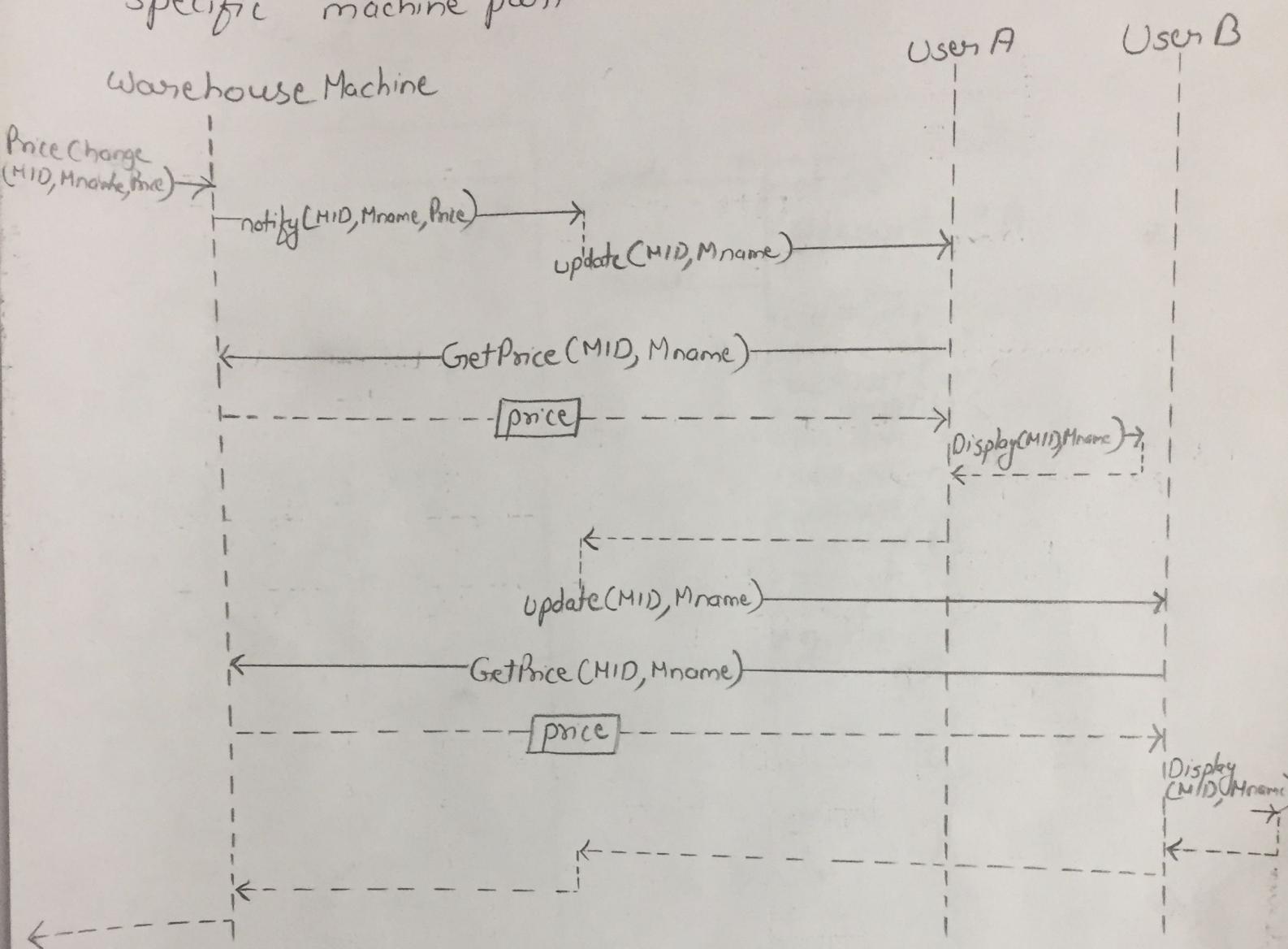
UserList → remove() - Remove the user from the user List

### (b) Sequence diagram

1. User register to observe price change of selected machine part.

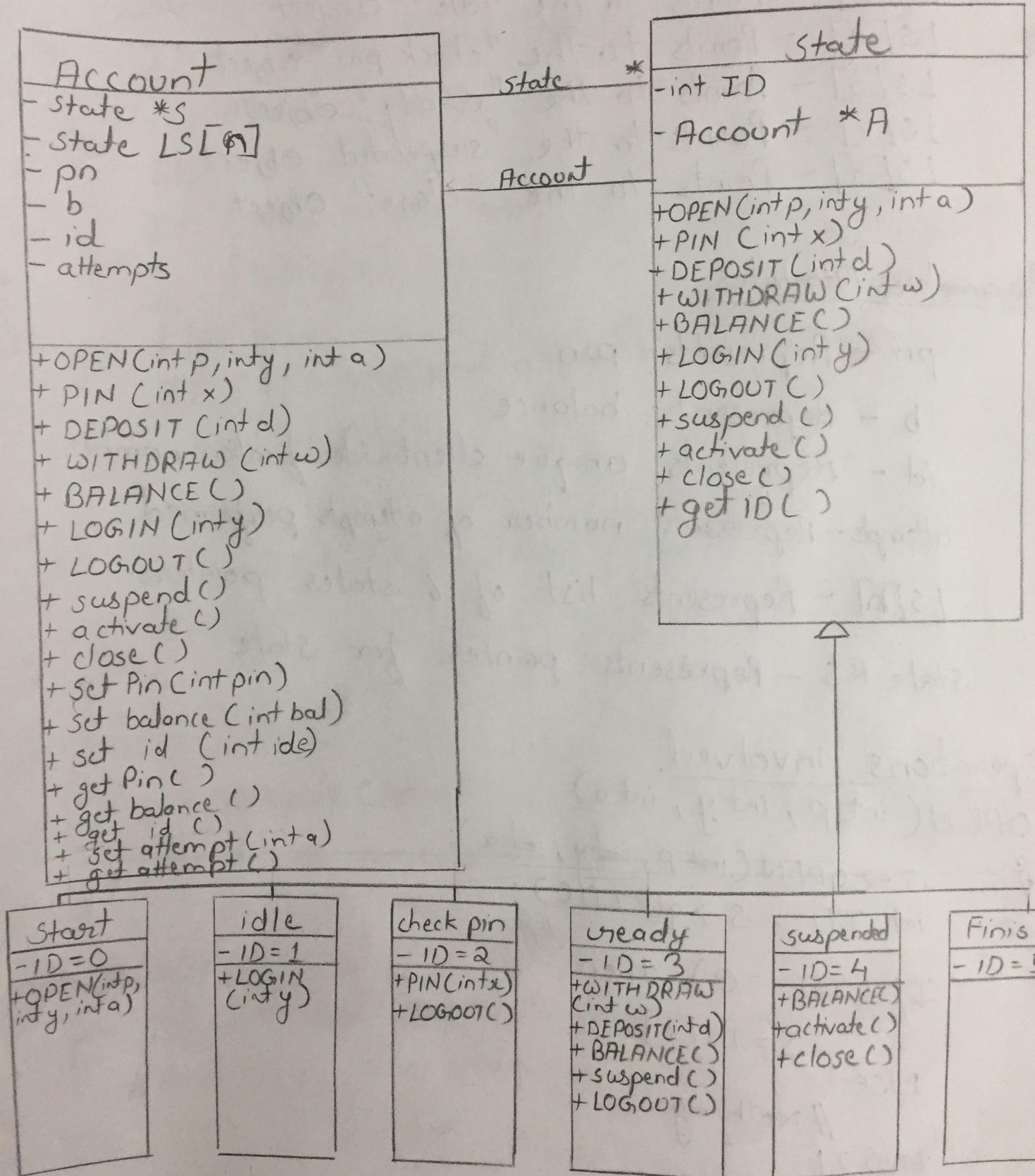


2. System notification to user about price change of specific machine part



Problem 2.

(a) Design the system using State Design Pattern.

1. Centralized Version

## Account Class

### List of states

- LS[0] - Points to the "start" Object
- LS[1] - Points to the "idle" Object
- LS[2] - Points to the "check pin" Object
- LS[3] - Points to the "ready" Object
- LS[4] - Points to the "suspended" Object
- LS[5] - Points to the "close" Object

### Parameters Attributes

- pn - Represents pin
- b - Represents balance
- id - Represents unique client identification number
- attempts - Represents number of attempts performed
- LS[] - Represents list of 6 states pointers.
- state\*s - Represents pointer for state

### Operations involved

```

> OPEN(int p, int y, int a)
  {
    s → OPEN(int p, int y, int a);
    int ID = s → getID();
    If (ID == 0) then
      s = LS[1]
    Else
      // Nothing
  END If

```

2) PIN (int x)

{  
 S → PIN (~~int~~ x); int temp = pn; int temp2 = attempts;  
 int ID = S → getID();  
 If (ID == 2) Then  
 IF (x != temp and temp2 < 2) then  
 // Remain in the same state  
 Else  
 IF (x == temp) then  
 S = LS[3];  
 Else  
 IF (x != temp and attempts == 2) then  
 S = LS[1];  
 Else  
 S = LS[1]; // For LOGOUT  
 End IF  
 }

3) DEPOSIT (int d)

{  
 S → DEPOSIT (~~int~~ d);  
 // Remain in same state

4) WITHDRAW (int w)

{  
 S → WITHDRAW (~~int~~ w);  
 // Remain in same state

}

(7)

## ⑤ BALANCE()

{  $S \rightarrow \text{BALANCE}();$   
     // Remain in same state  
     }

## ⑥ LOGIN (int y)

{  $S \rightarrow \text{LOGIN}(y);$  int a=y;  
     int ID =  $S \rightarrow \text{getID}();$   
     IF (ID == 1) Then  
         IF (a != id)  
             // Remain in same state  
         Else  
              $S = LS[2];$   
     END IF  
     }

## ⑦ LOGOUT ()

{  $S \rightarrow \text{LOGOUT}();$   
     int ID =  $S \rightarrow \text{getID}();$   
     IF (ID == 2 OR ID == 3)  
          $S = LS[1];$   
     END IF  
     }

(8)

⑧ suspend( )

{  $s \rightarrow \text{suspend}()$ ,  
 int  $\tau_d ID = s \rightarrow \text{getID}()$   
 IF ( $ID == 3$ ) then  
 $s = LS[4]$

END IF

}

⑨ activate( )

{  $s \rightarrow \text{activate}()$ ,  
 int  $ID = s \rightarrow \text{getID}()$   
 IF ( $ID == 4$ ) then  
 $s = LS[3]$

END IF

}

⑩ close( )

{  $s \rightarrow \text{close}()$ ,  
 int  $ID = s \rightarrow \text{getID}()$   
 IF ( $ID == 4$ )  
 $s = LS[5]$

END IF

}

(9)

(11) setPin (int pin)  
 {  
 pin = pin  
 }

(12) setBalance (int bal)  
 {  
 b = bal  
 }

(13) setId (int id)  
 {  
 id = id  
 }

(14) getPin ()  
 {  
 return pin  
 }

(15) getBalance ()  
 {  
 return b  
 }

(16) getId ()  
 {  
 return id  
 }

(17) setAttempts (int a)  
 {  
 attempts = a  
 }

(18) getAttempts ()  
 {  
 return attempts  
 }

### Class State

List of states  
 ID = 0 Start

ID = 1 idle

ID = 2 check pin

ID = 3 ready

ID = 4 suspended

ID = 5 finish

### Attributes

id - indicates the state id

**Note** - There are two functions.

- 1) get id() - This is in the class Account. Used for unique identity of Person
- 2) get ID() - This is in the class state. Used for change of states.

Operations involved

▷ OPEN (int p, int y, int a)

▷ get ID ()

{ return ID;

}

Class start

ID = 0

Operations

▷ OPEN (int p, int y, int a)

{ A → setPin (int p)

A → set balance (a)

A → set id (y)

}

Class idle

ID = 1

Operations

▷ Login (int y)

{ int b = A → get id ()

IF (b != y) then

print "incorrect user id"

Else

A → set attempt (0)

}

## class check pin

ID = 2

### Operations

1) PIN (int x)

{ int c = A → getPin()

int d = A → getAttempt()

IF (c != x and attempt(d < 2)) then

    Print "Incorrect Pin"

    d = d + 1

    A → setAttempt(d)

}

Else

    IF (c == x) ~~and~~ then

        Display Menu

Else

    int e = A → getAttempt()

    IF (c != x and e == 2) then

        Print "Incorrect Pin; too many attempts"

End IF

}

LOGOUT ()

{ int e = A → getAttempt()

    e = e + 4

}

class ready

ID = 3

Operations

1) WITHDRAW (int w)

{ int f = A → getbalance () }

IF (f &lt; -5000) then

{ Print "Cannot withdraw; Account needs to be suspended"

A → suspend ()

Else

{ f = f - w

A → setbalance (f)

A → setbalance (f)

}

2) DEPOSIT (int d)

{ int g = A → getbalance () }

g = g + d

A → setbalance (g)

}

3) BALANCE ()

{ int h = A → getbalance () }

Print h

}

4) Suspend()  
 { Print Message "Account suspended; Do you want to activate the account or close the account  
 Take two input from user - activate and close  
 17 - Activate  
 08 - Close  
 If (1) then  
 A → activate()  
 Else  
 A → close()  
 End IF

{ } suspend()

{ }

}

5) Logout  
 { Print Message "Enter 0 for logout"  
 IF(0) then  
 A → LOGIN // Display Nothing  
 End IF

{ }

Class suspended.

ID = H

Operations.

1) BALANCE()  
 { int i = A → get balance()  
 Display i

{ }

2) close()

{ }

{ }

③ activate()

27

## Class Finish

$ID = 5$   
No operations.

Sequence Diagram for the following sequence

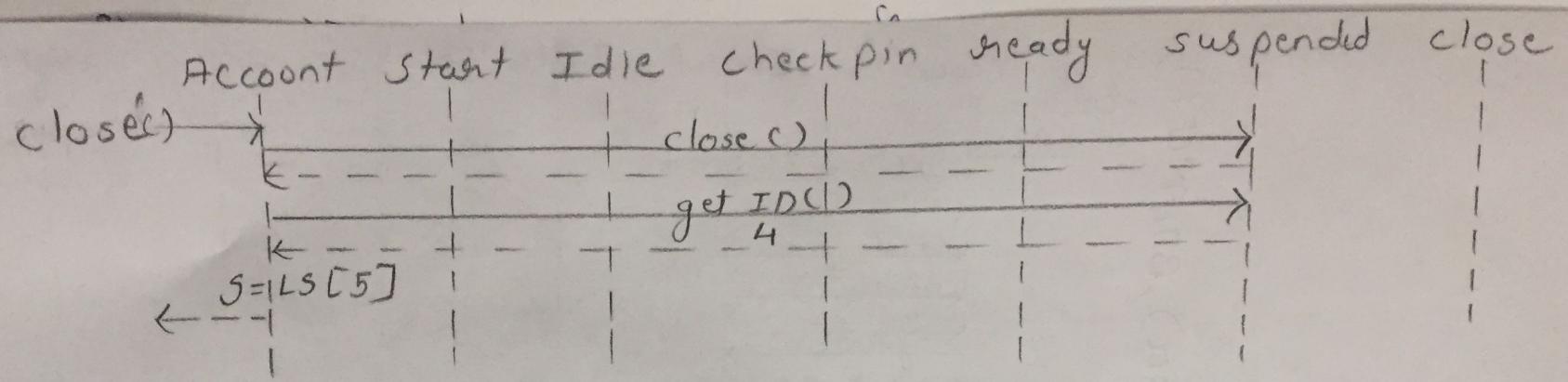
OPEN(123, 111, 1000), PIN(111), Login(111), Deposit(200), Balance(), Suspend(), Close()

```

sequenceDiagram
    participant Account
    participant ATM
    Note left of Account: OPEN(123, 111, 1000)  
PIN(111, 0)  
setPin(P)  
setBalance(1000)  
setAttempts(0)  
getID()
    Note right of Account: LOGIN(111)  
Login(111)  
getID()  
setAttempt(0)  
getTDC()
    Note left of ATM: PIN(111)  
getPin()  
getAttempt()
    Note right of ATM: Temp:p2  
111=Temp  
getTDC()
    Note left of Account: DEPOSIT(200)
    Note right of Account: getBalance()  
setBalance(1200)
    Note right of ATM: Balance()  
getBalance()
    Note left of ATM: suspend()
    Note right of ATM: suspend()
    Note left of Account: close()
    Note right of Account: getTDC()
    Note right of ATM: 1000+200
  
```

The sequence diagram illustrates the interaction between an **Account** object and an **ATM** object. The process begins with the **Account** object performing its own operations: `OPEN(123, 111, 1000)`, `PIN(111, 0)`, `setPin(P)`, `setBalance(1000)`, `setAttempts(0)`, and `getID()`. Subsequently, the **Account** sends a `LOGIN(111)` message to the **ATM**. The **ATM** responds with `Login(111)`, `getID()`, `setAttempt(0)`, and `getTDC()`. The **ATM** then sends a `PIN(111)` message to the **Account**, which returns `getPin()` and `getAttempt()`. The **ATM** also sends `Temp:p2` and `111=Temp` to the **Account**, followed by `getTDC()`. The **Account** performs a `DEPOSIT(200)` operation, which involves `getBalance()` and `setBalance(1200)`. The **ATM** then sends `Balance()` and `getBalance()` messages to the **Account**. Finally, both the **Account** and **ATM** send `suspend()` messages to each other, and the **Account** concludes with a `close()` operation.

Continued on back

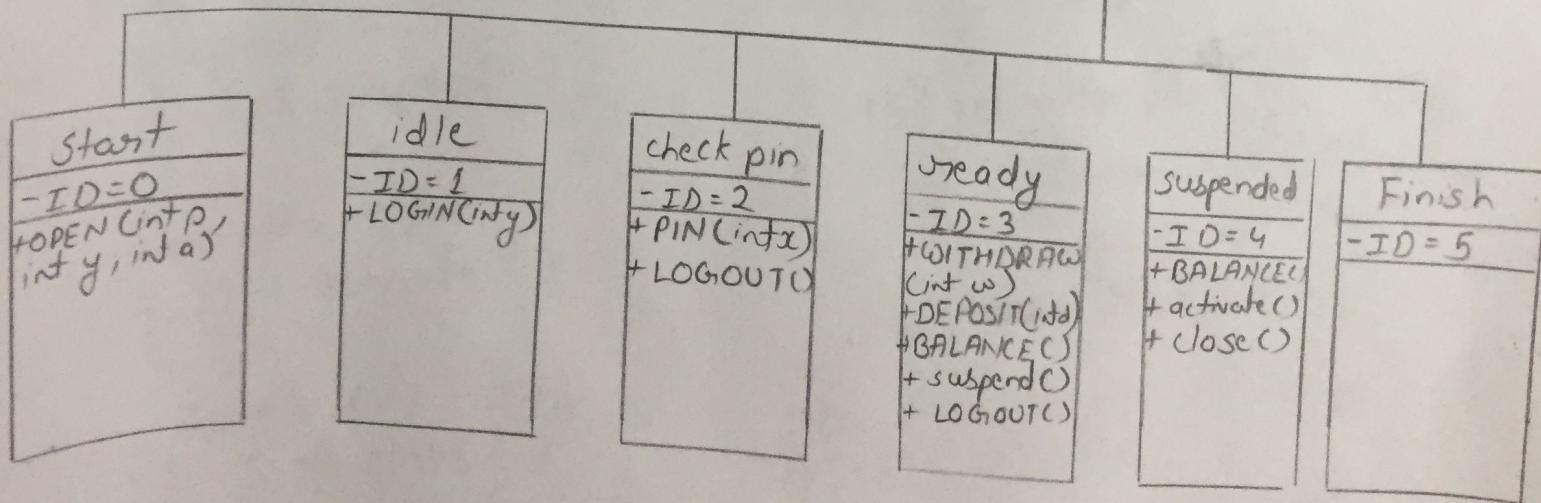
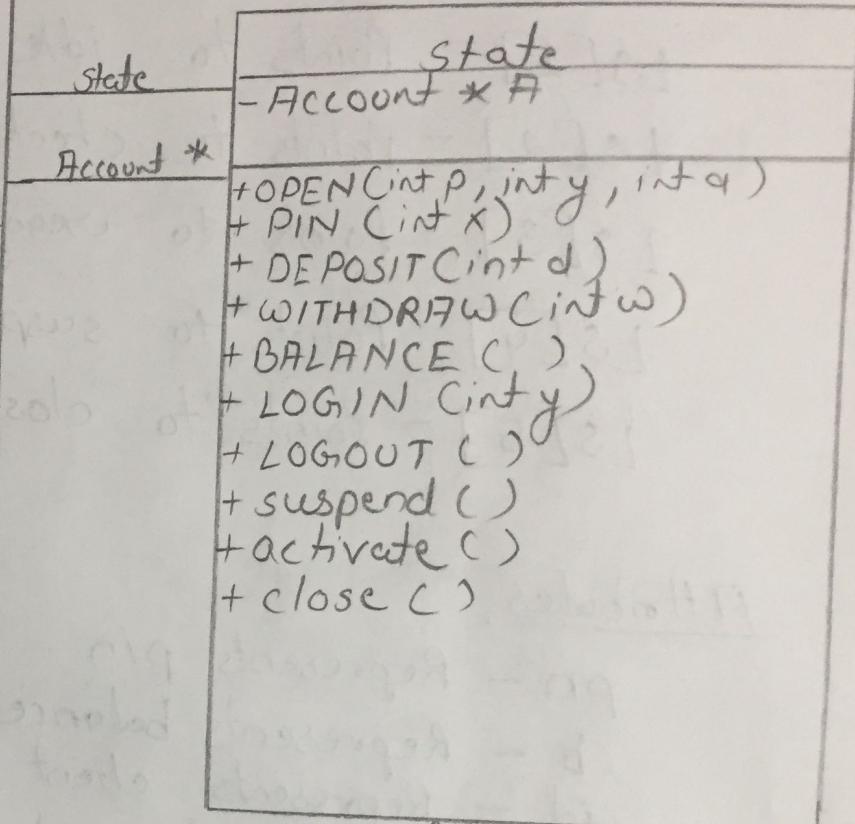
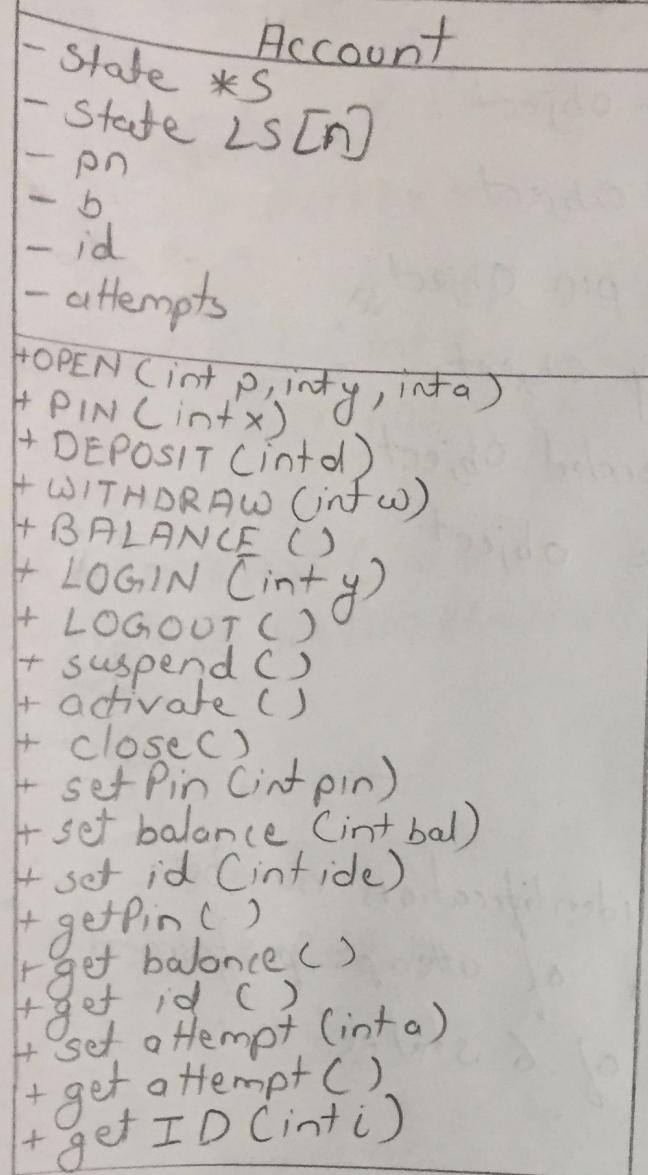


Note

get ID() - Returns states Id

get id() - Returns unique id issued for a person

# Decentralised State Pattern



## Class Account.

### List of State

$LS[0]$  - Points to start object

$LS[1]$  - Points to idle object

$LS[2]$  - Points to check pin object

$LS[3]$  - Points to ready object

$LS[4]$  - Points to suspended object

$LS[5]$  - Points to close object

### Attributes

$pn$  - Represents pin

$b$  - Represents balance

$id$  - Represents client identification number

$attempts$  - Represents number of attempts performed

$LS[n]$  - Points to list of 6 states.

- 1) Class Account Operations
- 1) OPEN (int p, int y, int a)
    - { S → OPEN (p, y, a)
    - }
  - 2) PIN (int x)
    - { S → PIN (x)
    - }
  - 3) DEPOSIT (int d)
    - { S → DEPOSIT (d)
    - }
  - 4) WITHDRAW (int w)
    - { S → WITHDRAW (w)
    - }
  - 5) BALANCE ()
    - { S → BALANCE ()
  - 6) LOGIN (int y)
    - { S → LOGIN (y)
    - }
  - 7) LOGOUT ()
    - { S → LOGOUT ()
    - }
  - 8) suspend ()
    - { S → suspend ()
    - }
  - 9) activate ()
    - { S → activate ()
    - }
  - 10) close ()
    - { S → close ()
    - }
- 11) setPin (int pin)
    - { pn = pin
    - }
  - 12) ~~getPin ()~~
    - { return pn
    - }
  - 13) setBalance (int bal)
    - { b = bal
    - }
  - 14) setId (int ide)
    - { id = ide
    - }
  - 15) getBalance ()
    - { return b
    - }
  - 16) getId ()
    - { return id
    - }
  - 17) setAttempt (int a)
    - { attempts = a
    - }
  - 18) getAttempt ()
    - { return attempts
    - }
  - 19) getID (int i)
    - { S = LS[i]
    - }

Class "State"

All the functions are abstract functions

Class "Start"

ID = 0

Operations

- 1) OPEN (int p, int y, int a)
  - 2 A → setPoint (p)
  - A → setId (y)
  - A → setBalance (a)
  - A → get ID [1]
- 3

Class idle

ID = 1

Operations

- 1) LOGIN (int y)
  - 2 int a = A → get id ()
    - IF (y != a) then
 

Print "Invalid Id"
    - else
      - 2 A → set attempt (0)
      - A → get ID [2]
  - 3

Class check pin  
 $ID = 2$

1) Operations

1) PIN (int &i)

{ int a = ~~get~~ A → get Pin()  
 int b = A → get attempt

If ( $x \neq a$  and  $b < 2$ ) then

{ Print "Incorrect Pin"

    A → set attempt (b+1)

}

else

If ( $x = a$  and  $b = 2$ ) then

{ Print "Incorrect pin and too many attempts "

    A → get ID (1)

    A → set attempt (0)

}

else

{ Display the menu of option

    A → get ID (3)

    3 End IF

2) LOGOUT ()

{ A → get ID (1)

}