PROBLEM #1

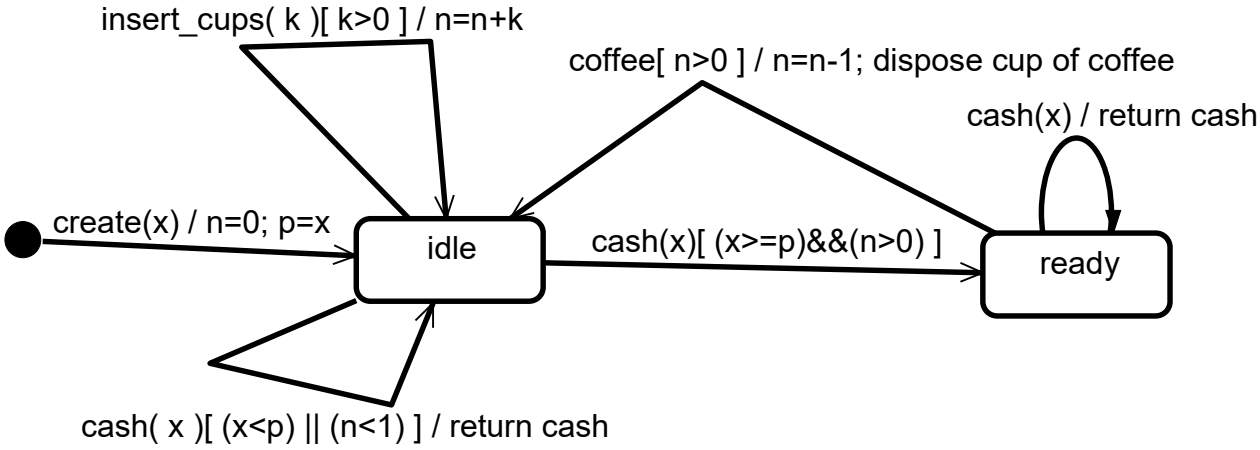
An EFSM (Extended Finite State Machine) of a component is shown below. The component supports the following operations: *create(float x), cash(float x), insert_cups(int k), coffee()*

Design the system using the **State design pattern**. You should use the **de-centralized** version of this pattern.

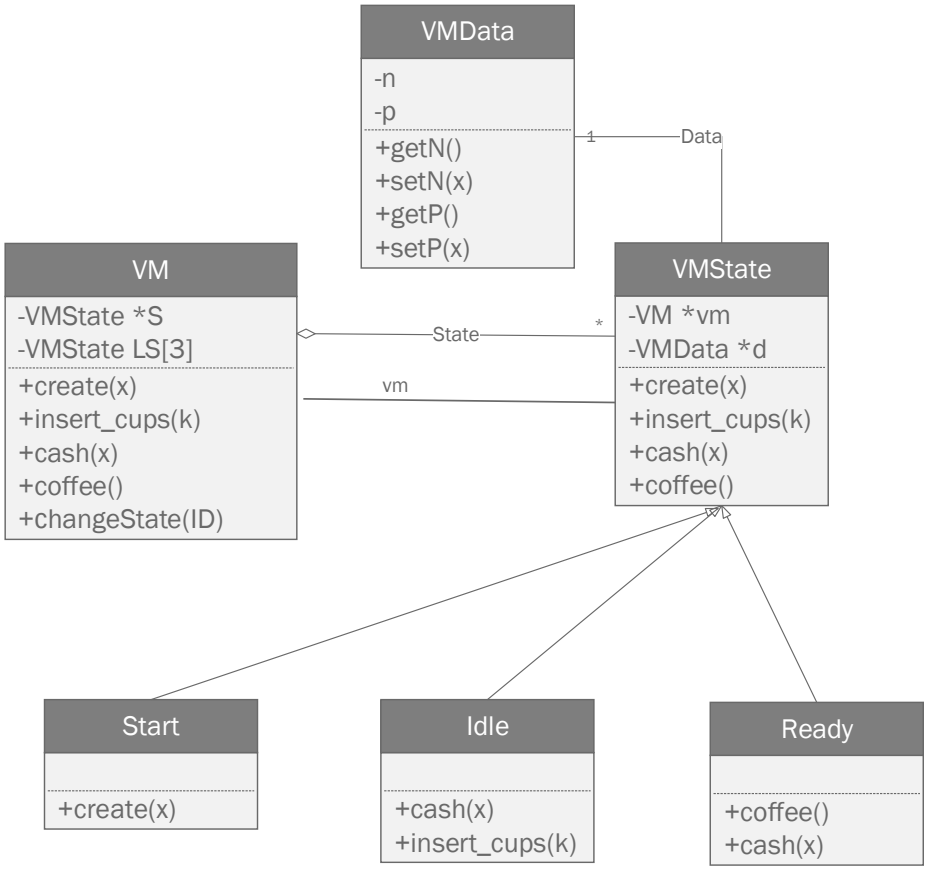
In your solution:

a. Provide a class diagram for the component. 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. Notice that the components in your design should be de-coupled as much as possible. In addition, components should have high cohesion.

b. Provide a **sequence diagram** for the following operation sequence:
create(2.5), *insert_cups(10)*, *cash(3)*, *coffee()*



De-centralized State Pattern



Class "VM"

S //points to current state object
LS[0] //points to "Start" Object
LS[1] //points to "Idle" Object
LS[2] //points to "Ready" Object

S = LS[0] // initialize state object to "Start"

Operations

changeState(ID){
S = LS[ID]
}

create(x){
S->create(x)
}

```
insert_cups(k){
    S->insert_cups(k)
}

cash(x){
    S->cash(x)
}

coffee(){
    S-> coffee()
}
```

Class "VMState"

Operations

create(), insert_cups(), cash() and coffee() are abstract operations

Class "Start"

Operations

```
create(x){
    d->setN(0)
    d->setP(x)
    vm->changeState(1)          //change VM state from "Start" to "Idle"
}
```

Class "Idle"

Operations

```
cash(x){
    IF ( x >= d->getP() ) && (d->getN() > 0 ) THEN
        vm ->changeState(2)     //change VM state from "Idle" to "Ready"
    ELSE IF ( x < d->getP() ) || (d->getN() < 1 ) THEN
        return cash
    ENDIF
}

insert_cups(k){
    IF  k > 0 THEN
        numberOfCups = d->getN()
        numberOfCups = numberOfCups + k
```

```
        d->setN(numberOfCups)
    ENDIF
}
```

Class "Ready"

Operations

```
coffee(x){
    IF  d->getN() > 0 THEN
        numberOfCups = d->getN()
        numberOfCups = numberOfCups - 1
        d->setN(numberOfCups)
        dispose cup of coffee
        vm->changeState(1)      //change VM state from "Ready" to "Idle"
    ENDIF
}

cash(x){
    return cash
}
```

Class "VMData"

```
n       // number of cups
p       // price


getN(){
   return n
}

setN(int x){
    n = x
}


getP(){
   return p
}

setP(int x){
    p = x
}
```

De-centralized Pattern – Sequence Diagram

create(2.5), insert_cups(10), cash(3), coffee()