CS 586 PROJECT PHASE-2

1. MDA-EFSM model for the Vending Machine components

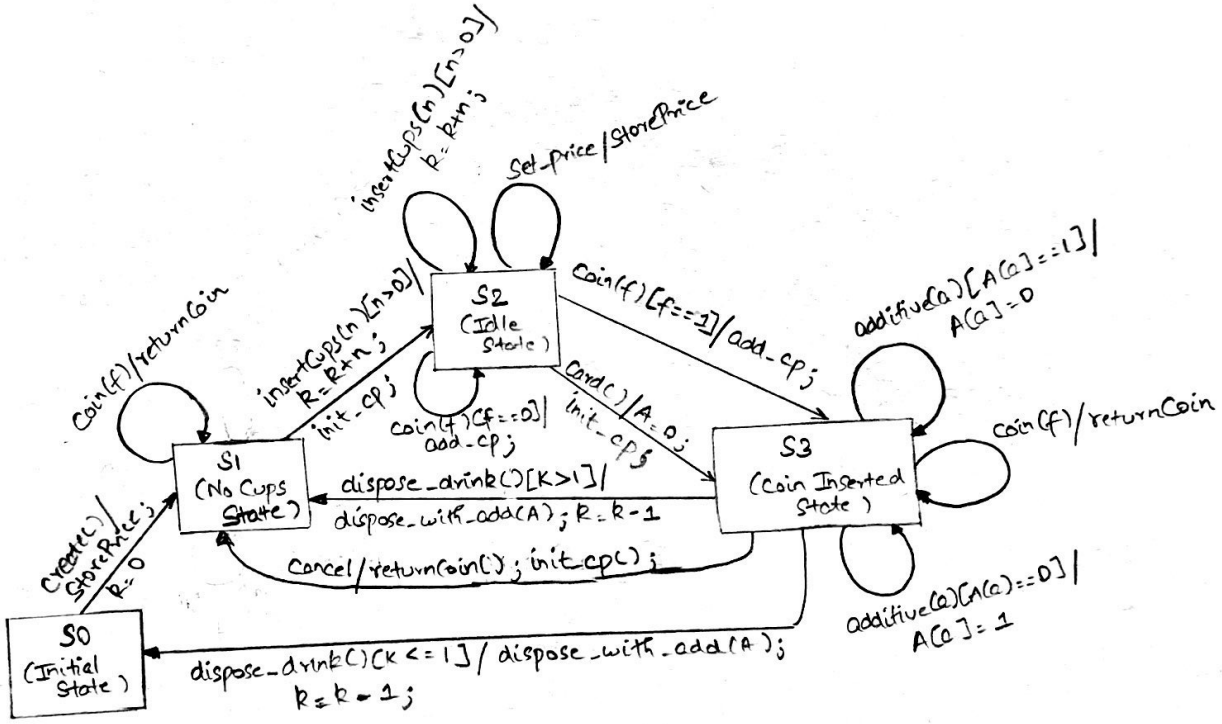
The MDA-EFSM model consists of meta events and the meta actions to take place.

a) The meta events in the MDA-EFSM include:

1. create()
2. insert_cups(int n) // n represents # of cups
3. coin(int f) // f=1: sufficient funds inserted for a drink
   // f=0: not sufficient funds for a drink
4. card()
5. cancel()
6. set_price()
7. dispose_drink(int d) // d represents a drink id
8. additive(int a) // a represents additive id

b) The meta actions in the MDA-EFSM include:

1. StorePrice()
2. init_cp() // zero Cumulative Fund cf
3. add_cp() // increase Cumulative Fund cf
4. returnCoin() // return coins inserted for a drink
5. dispose_with_add(int A[]) // dispose a drink with d id dispose marked additives in A list, // where additive with i id is disposed when A[i]=1



STATE DIAGRAM OF MDA-EFSM

d)

Input Process of Vending-Machine-1 where,
m: pointer to the MDA-EFSM
d: pointer to the data store DS-1
In the data store:
cf: represents a cumulative fund
price: represents a price for a drink

```
create(int p) {
        d->temp_p=p;
        m->create();
    }
coin(int v) {
        d->temp_v=v;
        if (d->cf+v>=d->price) m->coin(1);
        else m->coin(0);
    }
card(float x) {
        if (x>=d->price) m->card();
    }
sugar() {
        m->additive(1);
    }
tea() {
        m->dispose_drink(1);
    }
chocolate() {
        m->dispose_drink(2);
    }
insert_cups(int n) {
        m->insert_cups(n);
    }
set_price(int p) {
        d->temp_p=p;
        m->set_price()
    }
cancel() {
        m->cancel();
    }
```

Input Process of Vending-Machine-2 where,

m: pointer to the MDA-EFSM

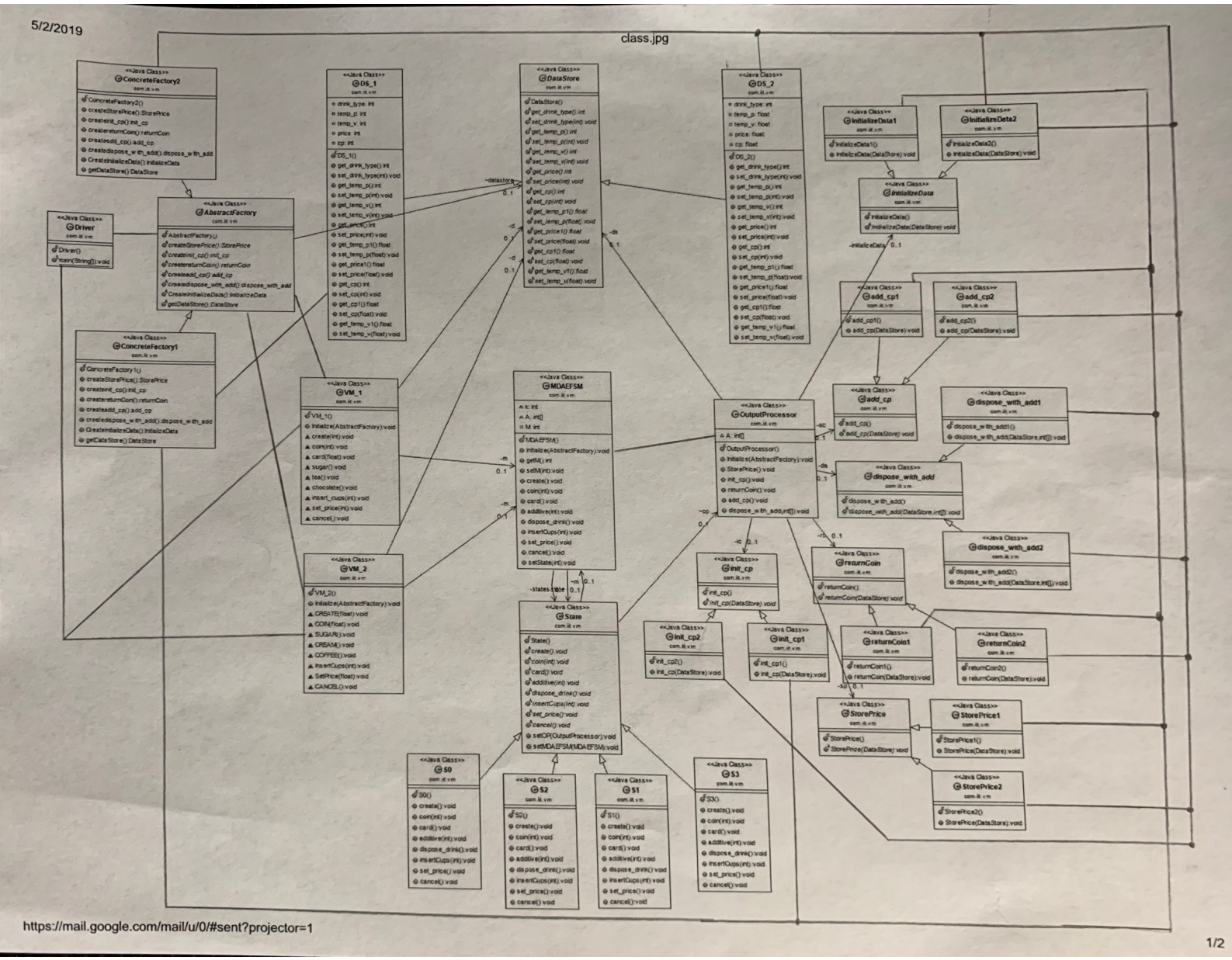
d: pointer to the data store DS-2

In the data store:

cf: represents a cumulative fund

price: represents a price for a drink

```
CREATE(float p) {
        d->temp_p=p;
        m->create();
}
COIN(float v) {
        d->temp_v=v;
        if (d->cf+v>=d->price) m->coin(1);
        else m->coin(0);
}
SUGAR() {
        m->additive(2);
}
CREAM() {
        m->additive(1);
}
COFFEE() {
        m->dispose_drink(1);
}
InsertCups(int n) {
        m->insert_cups(n);
}
SetPrice(float p) {
        d->temp_p=p;
        m->set_price()
}
CANCEL() {
        m->cancel();
}
```



3. Description and purpose of each Class and its Operations

Class VM_1:

This class holds the operations and functions of Vending Machine 1. The VM operations are directed to the MDA-EFSM.

Responsibilities of each operation in VM_1:

This class contains the core functional methods that are required to be executed by the Vending Machine 1.

- create()- used to create a new VM_1. This method invokes the create() of MDA_EFSM
- coin()- When coin is inserted into VM_1, it invokes the coin() of MDA_EFSM and does a primary checking if the entered coin value is less than or equal to the price of the drink.
- card()-Same work as coin and does the same primary check and invokes the card() in MDA_EFSM
- sugar()-To add sugar to the drink. It changes the variable in MDA_EFSM
- tea()-To dispose a type of drink and invokes the dispose_drink() in MDA_EFSM
- chocolate()-To dispose a type of drink and invokes the dispose_drink() in MDA_EFSM
- insert_cups()-To insert the number of cups and invokes the insertCups() in MDA_EFSM
- set_price()-To change the price and invokes the set_price() in MDA_EFSM
- cancel()-To cancel the operation and invokes the cancel() in MDA_EFSM

Class VM_2:

This class holds the operations and functions of Vending Machine 2. The VM operations are directed to the MDA-EFSM.

Responsibilities of each operation in VM_2:

This class contains the core functional methods that are required to be executed by the Vending Machine 2.

- CREATE()- used to create a new VM_2. This method invokes the create() of MDA_EFSM
- COIN()- When coin is inserted into VM_2, it invokes the coin() of MDA_EFSM and does a primary checking if the entered coin value is less than or equal to the price of the drink.
- SUGAR()-To add sugar to the drink. It changes the variable in MDA_EFSM
- CREAM()-To add cream to the drink. It changes the variable in MDA_EFSM
- COFFEE()-To dispose a type of drink and invokes the dispose_drink() in MDA_EFSM
- InsertCups()-To insert the number of cups and invokes the insertCups() in MDA_EFSM
- SetPrice()-To change the price and invokes the set_price() in MDA_EFSM
- CANCEL()-To cancel the operation and invokes the cancel() in MDA_EFSM

Class AbstractFactory:

This is an abstract class and is used to implement the abstract-factory pattern. It contains the class reference to all meta actions. Since, we implemented strategy pattern, the meta actions represent an individual class.

Class ConcreteFactory1:

This class inherits the AbstractFactory and is used by VM_1.

Responsibilities of each operation in ConcreteFactory1:

All the methods are used to create a new object and return the pointer. i.e. create object for StorePrice1, init_cp1, returnCoin1, add_cp1, dispose_with_add1, InitializeData1, DS_1

Class ConcreteFactory2:

This class inherits the AbstractFactory and is used by VM_2.

Responsibilities of each operation in ConcreteFactory2:

All the methods are used to create a new object and return the pointer. i.e. create object for StorePrice2, init_cp2, returnCoin2, add_cp2, dispose_with_add2, InitializeData2, DS_2

Class DataStore:

This is an abstract class and is used to store data. It contains the getter and setter methods to all the variables.

Class DS_1:

This class inherits the DataStore and is used by VM_1.

Responsibilities of each operation in DS_1:

All the methods are used to set or get the values of the variables which are used by VM_1 for its operations.

Class DS_2:

This class inherits the DataStore and is used by VM_2.

Responsibilities of each operation in DS_2:

All the methods are used to set or get the values of the variables which are used by VM_2 for its operations.

Class MDAEFSM:

This is a model independent class and contains a list of meta events which are independent of any platform. It contains an integer array to represent the list of additives (A[]) and an integer to represent the number of cups(k).

Responsibilities of each operation in MDAEFSM:

The main responsibility of the class is to invoke the functions that are required by the VM. However, it calls the state object to invoke the methods. After each action the state may be changed. The main objective is that the MDAEFSM remains the same regardless of any type of Vending Machine used.

Class State:

This is an abstract class and has abstract method of all the meta events listed by the MDAEFSM. It is used to implement the State pattern

Class S0:

This class inherits the State and represents the Initial State.

Responsibilities of each operation in S0:

In this state only the required methods invoke the needed meta actions. Other methods provide dummy operations only. S0 supports create()

Class S1:

This class inherits the State and represents the No Cups State.

Responsibilities of each operation in S1:

In this state only the required methods invoke the needed meta actions. Other methods provide dummy operations only. S0 supports insertCups()

Class S2:

This class inherits the State and represents the Idle State.

Responsibilities of each operation in S2:

In this state only the required methods invoke the needed meta actions. Other methods provide dummy operations only. S0 supports insertCups(), set_price(), coin(), card()

Class S3:

This class inherits the State and represents the Coin Inserted State.

Responsibilities of each operation in S3:

In this state only the required methods invoke the needed meta actions. Other methods provide dummy operations only. S0 supports dispose_drink(), set_price(), coin(), additive(), cancel()

Class OutputProcessor:

This is class that contains a list of meta actions that take place once the meta events are invoked. Since the strategy pattern is enforced each meta action is an independent class. The classes include add_cp, init_cp, dispose_with_add, returnCoin, StorePrice, InitializeData

Responsibilities of each operation in OutputProcessor:

Since the strategy pattern is followed, each method calls a meta action that is specific to the strategy class. i.e. the methods StorePrice(), init_cp(), returnCoin(), add_cp(), dispose_with_add() in the OutputProcessor calls the method of the class StorePrice, init_cp, returnCoin, add_cp, dispose_with_add respectively.

Class add_cp:
It is an abstract class and contains a constructor with the DataStore variable passed.
Class add_cp1:
This class inherits the add_cp and is used by VM_1.
Responsibilities of each operation in add_cp1:
The method is used to get the current price add the value specified. It also updates DS_1 about the current price increase
Class add_cp2:
This class inherits the add_cp and is used by VM_2.
Responsibilities of each operation in add_cp2:
The method is used to get the current price add the value specified. It also updates DS_2 about the current price increase

Class init_cp:
It is an abstract class and contains a constructor with the DataStore variable passed.
Class init_cp1:
This class inherits the init_cp and is used by VM_1.
Responsibilities of each operation in init_cp1:
The method is used to set the current price to zero.
Class init_cp2:
This class inherits the init_cp and is used by VM_2.
Responsibilities of each operation in add_cp2:
The method is used to set the current price to zero.

Class StorePrice:
It is an abstract class and contains a constructor with the DataStore variable passed.
Class StorePrice1:
This class inherits the StorePrice and is used by VM_1.
Responsibilities of each operation in StorePrice1:
The method is used to set the current price and also updates DS_1 about the current price increase
Class StorePrice2:
This class inherits the StorePrice and is used by VM_2.
Responsibilities of each operation in StorePrice2:
The method is used to set the current price and also updates DS_2 about the current price increase

Class returnCoin:
It is an abstract class and contains a constructor with the DataStore variable passed.
Class returnCoin1:
This class inherits the returnCoin and is used by VM_1.
Responsibilities of each operation in returnCoin1:

The method is used to set the current price to zero and also returns the amount and updates the DS_1 about the current price change.

Class returnCoin2:

This class inherits the returnCoin and is used by VM_2.

Responsibilities of each operation in returnCoin2:

The method is used to set the current price to zero and also returns the amount and updates the DS_2 about the current price change.

Class dispose_with_add:

It is an abstract class and contains a constructor with the DataStore variable passed.

Class dispose_with_add 1:

This class inherits the dispose_with_add and is used by VM_1.

Responsibilities of each operation in dispose_with_add1:

The method gets the drink type selected and checks if any additives are present and dispose them accordingly.

Class dispose_with_add 2:

This class inherits the dispose_with_add and is used by VM_2.

Responsibilities of each operation in dispose_with_add2:

The method gets the drink type selected and checks if any additives are present and dispose them accordingly.

Class InitializeData:

It is an abstract class and contains a constructor with the DataStore variable passed.

Class InitializeData 1:

This class inherits the InitializeData and is used by VM_1.

Responsibilities of each operation in InitializeData1:

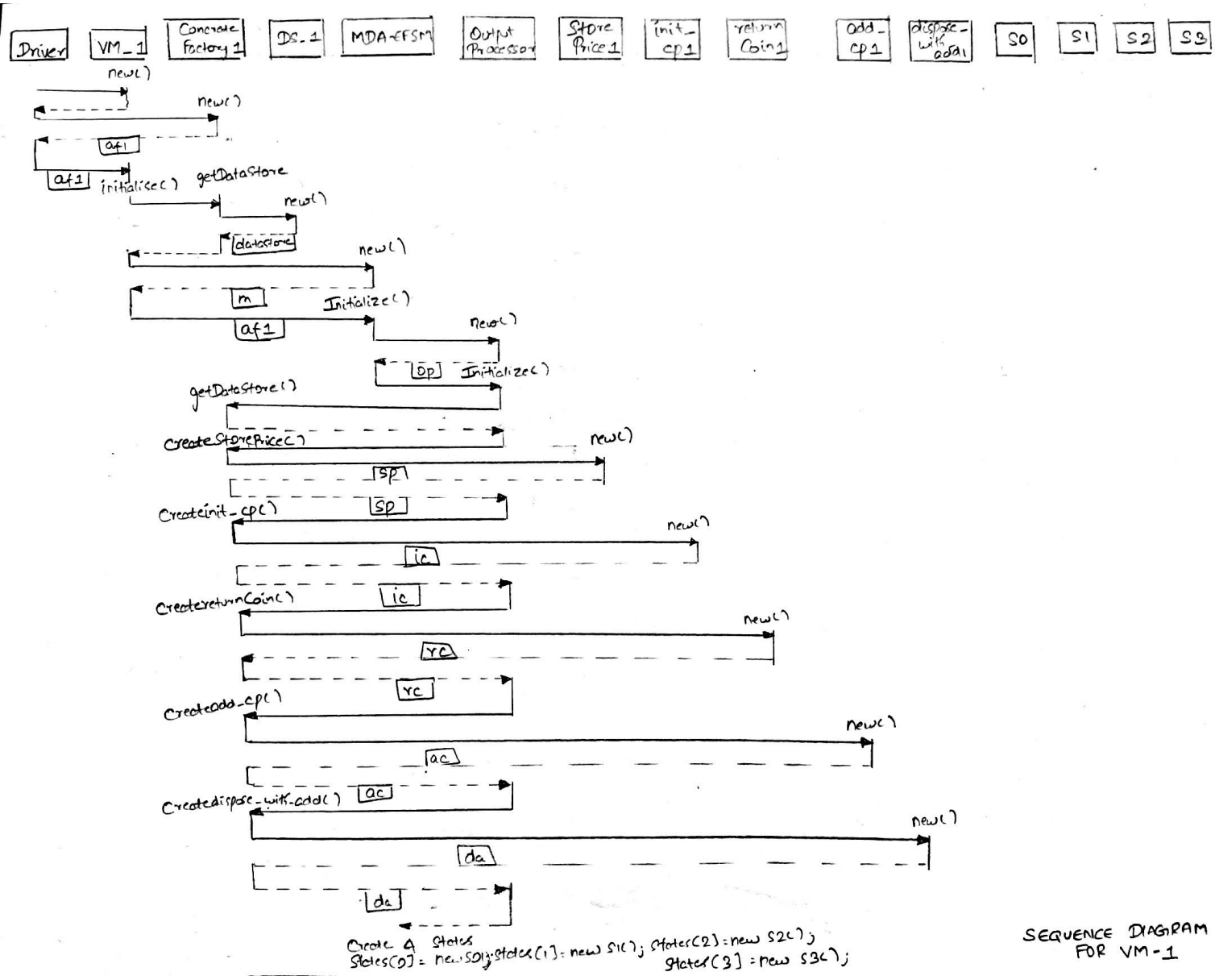
The method is used to set the initial values of the drink_type and current price as zero.

Class InitializeData 2:

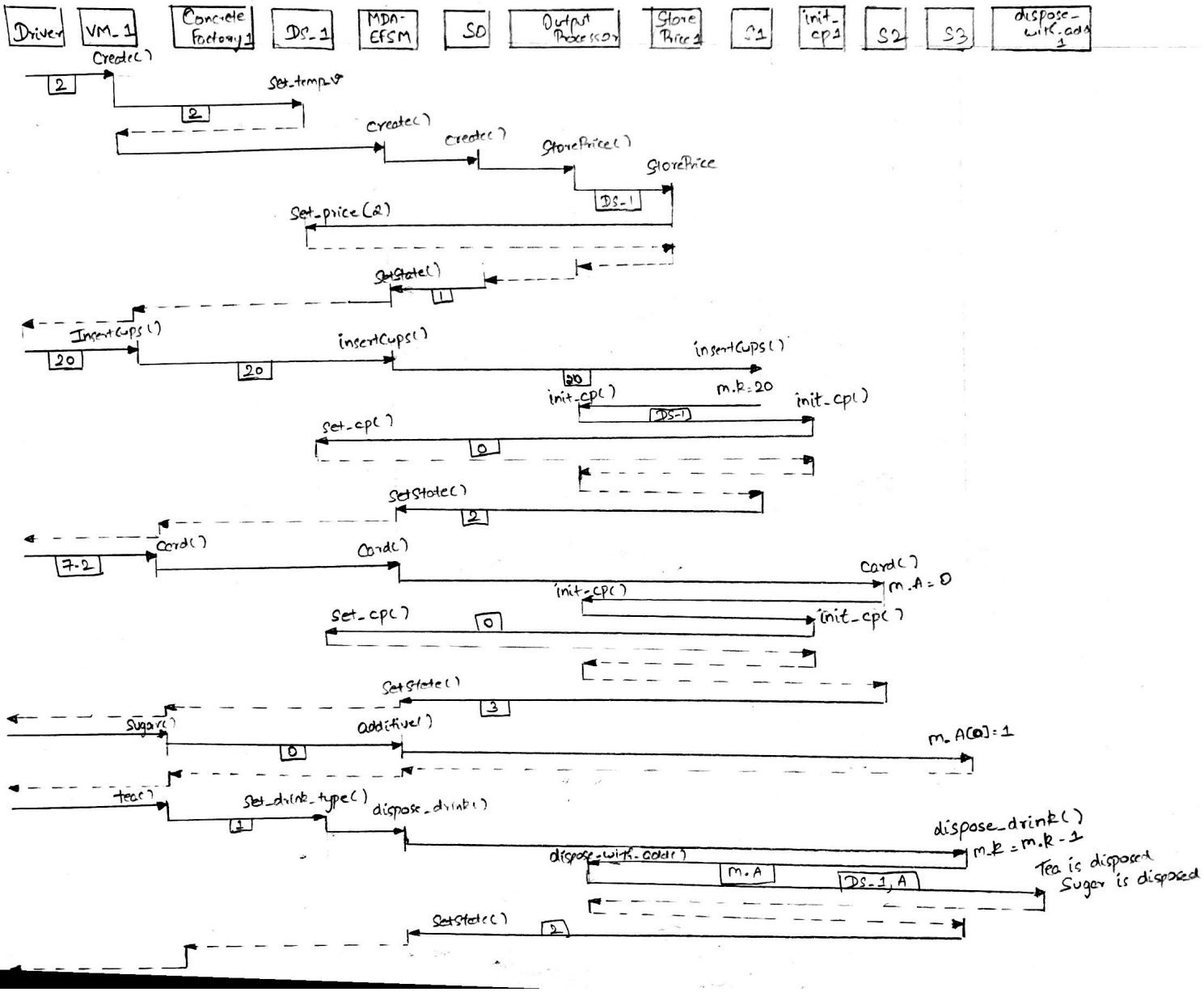
This class inherits the InitializeData and is used by VM_2.

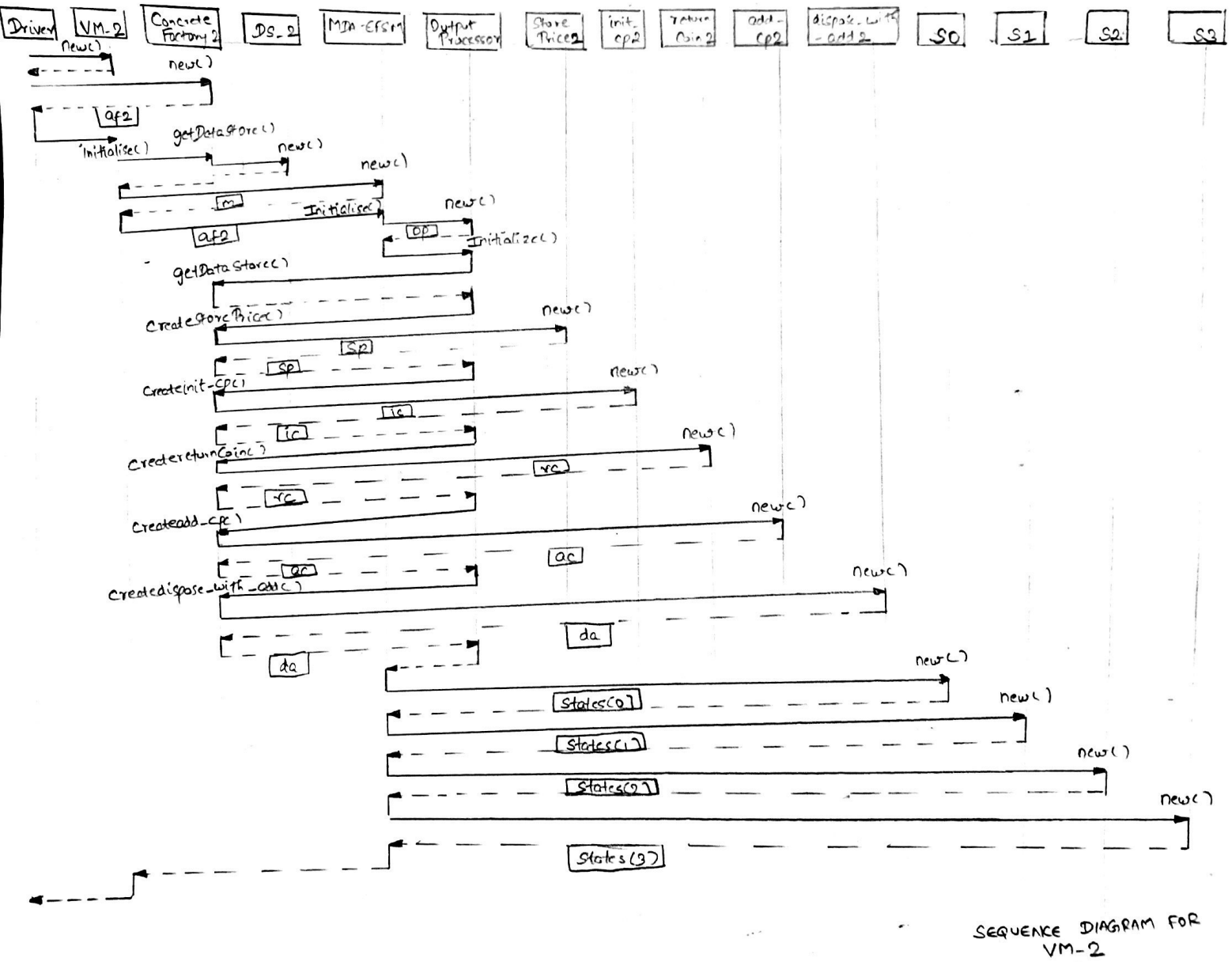
Responsibilities of each operation in InitializeData2:

The method is used to set the initial values of the drink_type and current price as zero.



SEQUENCE DIAGRAM FOR VM-1





SEQUENCE DIAGRAM FOR VM-2