CS 586 PROJECT PHASE 2

1. MDA-EFSM model for the Vending Machine components

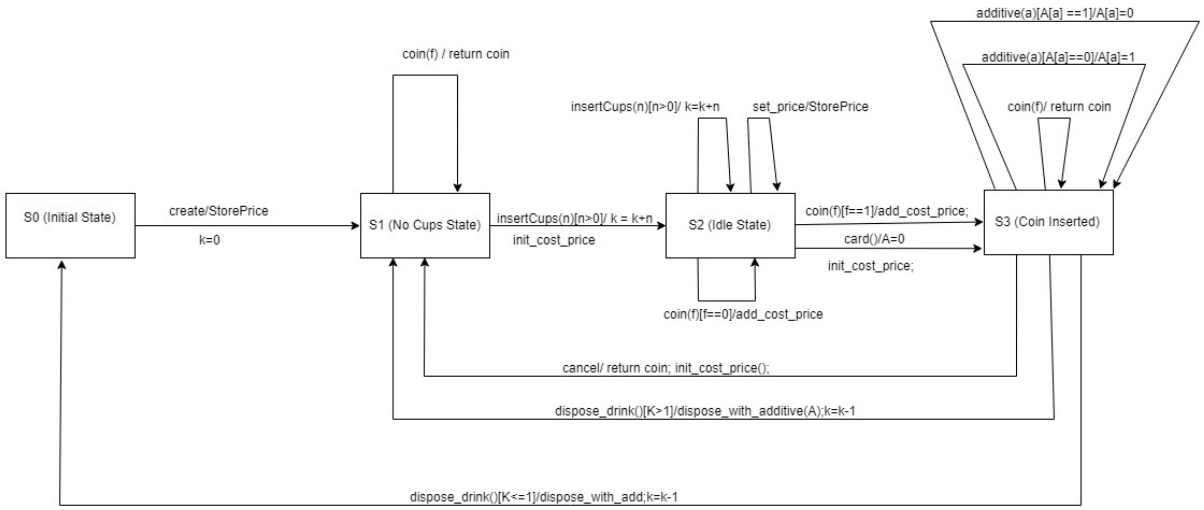
The MDA-EFSM model consists of meta events and the meta actions to take place.

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_cost_price() // zero Cumulative Fund cf
3. init_add_price() // 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



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();
}
```

Vending-Machine-2

Where,

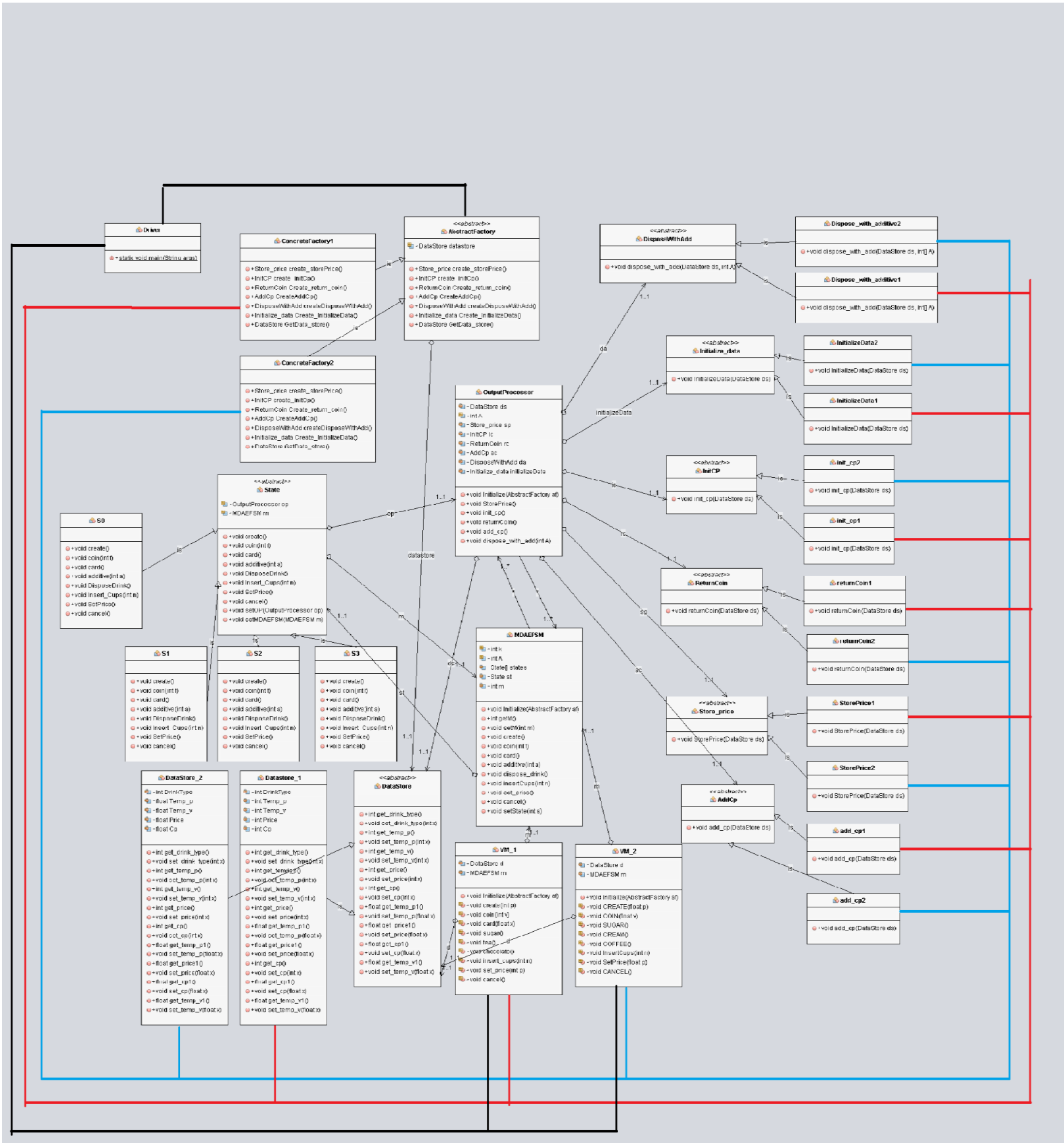
- m: pointer to the MDA-EFSM
- d: pointer to the data store DS-2

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();
}
```


Driver
+ static void main(String[] args)
ConcreteFactory1
+Store_price create_storePrice()
+InitCP create_initCp()
+ReturnCoin Create_return_coin()
+AddCp CreateAddCp()
+DisposeWithAdd createDisposeWithAdd()
+Initialize_data Create_InitializeData()
+DataStore GetData_store()
ConcreteFactory2
+Store_price create_storePrice()
+InitCP create_initCp()
+ReturnCoin Create_return_coin()
+AddCp CreateAddCp()
+DisposeWithAdd createDisposeWithAdd()
+Initialize_data Create_InitializeData()
+DataStore GetData_store()
<<abstract>>
AbstractFactory
- DataStore datastore
+Store_price create_storePrice()
+InitCP create_initCp()
+ReturnCoin Create_return_coin()
+AddCp CreateAddCp()
+DisposeWithAdd createDisposeWithAdd()
+Initialize_data Create_InitializeData()
+DataStore GetData_store()
<<abstract>>
DisposeWithAdd
+void dispose_with_add(DataStore ds, int A)
Dispose_with_additive2
+void dispose_with_add(DataStore ds, int[] A)
Dispose_with_additive1
+void dispose_with_add(DataStore ds, int[] A)
<<abstract>>
Initialize_data
+void InitializeData(DataStore ds)
InitializeData2
+void InitializeData(DataStore ds)
InitializeData1
+void InitializeData(DataStore ds)
OutputProcessor
- DataStore ds
- int A
- Store_price sp
- InitCP ic
- ReturnCoin rc
- AddCp ac
- DisposeWithAdd da
- Initialize_data initializeData
+void Initialize(AbstractFactory af)
+void StorePrice()
+void init_cp()
+void returnCoins()
+void add_cp()
+void dispose_with_add(int A)
<<abstract>>
InitCP
+void init_cp(DataStore ds)
init_cp2
+void init_cp(DataStore ds)
init_cp1
+void init_cp(DataStore ds)
<<abstract>>
ReturnCoin
+void returnCoin(DataStore ds)
returnCoin1
+void returnCoin(DataStore ds)
returnCoin2
+void returnCoin(DataStore ds)
<<abstract>>
State
- OutputProcessor op
- MDAEFSM m
+void create()
+void coin(int i)
+void card()
+void additive(int a)
+void DisposeDrink()
+void Insert_Cups(int n)
+void SetPrice()
+void cancel()
+void setOP(OutputProcessor op)
+void setMDAEFSM(MDAEFSM m)
S0
+void create()
+void coin(int i)
+void card()
+void additive(int a)
+void DisposeDrink()
+void Insert_Cups(int n)
+void SetPrice()
+void cancel()
S1
+void create()
+void coin(int i)
+void card()
+void additive(int a)
+void DisposeDrink()
+void Insert_Cups(int n)
+void SetPrice()
+void cancel()
S2
+void create()
+void coin(int i)
+void card()
+void additive(int a)
+void DisposeDrink()
+void Insert_Cups(int n)
+void SetPrice()
+void cancel()
S3
+void create()
+void coin(int i)
+void card()
+void additive(int a)
+void DisposeDrink()
+void Insert_Cups(int n)
+void SetPrice()
+void cancel()
MDAEFSM
- int k
- int A
- State[] states
- State st
- int m
+void Initialize(AbstractFactory af)
+int getM()
+void setM(int m)
+void create()
+void coin(int i)
+void card()
+void additive(int a)
+void dispose_drink()
+void insertCups(int n)
+void set_price()
+void cancel()
+void setState(int s)
<<abstract>>
Store_price
+void StorePrice(DataStore ds)
StorePrice1
+void StorePrice(DataStore ds)
StorePrice2
+void StorePrice(DataStore ds)
<<abstract>>
AddCp
+void add_cp(DataStore ds)
add_cp1
+void add_cp(DataStore ds)
add_cp2
+void add_cp(DataStore ds)
DataStore_2
- int DrinkType
- float Temp_p
- float Temp_v
- float Price
- float Cp
+int get_drink_type()
+void set_drink_type(int x)
+int get_temp_p()
+void set_temp_p(int x)
+int get_temp_v()
+void set_temp_v(int x)
+int get_price()
+void set_price(int x)
+int get_cp()
+void set_cp(int x)
+float get_temp_p1()
+void set_temp_p(float x)
+float get_price1()
+void set_price(float x)
+float get_cp1()
+void set_cp(float x)
+float get_temp_v1()
+void set_temp_v(float x)
Datastore_1
- int DrinkType
- int Temp_p
- int Temp_v
- int Price
- int Cp
+int get_drink_type()
+void set_drink_type(int x)
+int get_temp_p()
+void set_temp_p(int x)
+int get_temp_v()
+void set_temp_v(int x)
+int get_price()
+void set_price(int x)
+float get_temp_p1()
+void set_temp_p(float x)
+float get_price1()
+void set_price(float x)
+float get_cp1()
+void set_cp(float x)
+float get_temp_v1()
+void set_temp_v(float x)
<<abstract>>
DataStore
+int get_drink_type()
+void set_drink_type(int x)
+int get_temp_p()
+void set_temp_p(int x)
+int get_temp_v()
+int set_temp_v(int x)
+int get_price()
+void set_price(int x)
+float get_temp_p1()
+void set_temp_p(float x)
+float get_price1()
+void set_price(float x)
+float get_cp1()
+void set_cp(float x)
+float get_temp_v1()
+void set_temp_v(float x)
VM_1
- DataStore d
- MDAEFSM m
+void Initialize(AbstractFactory af)
+void create(int p)
+void coin(int v)
+void card(float x)
+void sugar()
+void tea()
+void hotchocolate()
+void insert_cups(int n)
+void set_price(int p)
+void cancel()
VM_2
- DataStore d
- MDAEFSM m
+void Initialize(AbstractFactory af)
+void CREATE(float p)
+void COIN(float v)
+void SUGAR()
+void CREAM()
+void COFFEE()
+void InsertCups(int n)
+void SetPrice(float p)
+void CANCEL()
1..1
is
op
sp
ic
rc
da
ac
ds
st
m
datastore
InitializeData

Class VM_1:

This class contains 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 contains 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_cost_price1, returnCoin1, init_add_price1, Dispose_With_additive1, InitializeData1, DataStore_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_cost_price2, returnCoin2, init_add_price2, Dispose_With_additive2, InitializeData2, DataStore_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 Datastore_1:

This class inherits the DataStore and is used by VM_1.

Responsibilities of each operation in Datastore_1: Setters or Getter methods to get or set the values of the variables which are used by VM_1 for its operations.

Class Datastore_2:

This class inherits the DataStore and is used by VM_2.

Responsibilities of each operation in DS_2: Setters or Getter methods to get or set 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 init_add_price, init_cost_price, Dipose_with_additive, 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_cost_price(), returnCoin(), init_add_price(), dispose_with_add() in the OutputProcessor calls the method of the class StorePrice, init_cost_price, returnCoin, init_add_price, dispose_with_add respectively.

Class init_add_price:

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

Class init_add_price1:

This class inherits the init_add_price and is used by VM_1.

Responsibilities of each operation in init_add_price1: Gets the current price add the value specified. It also updates DataStore_1 about the current price increase

Class init_add_price2:

This class inherits the init_add_price and is used by VM_2.

Responsibilities of each operation in init_add_price2: Gets the current price add the value specified. It also updates DataStore_2 about the current price increase

Class init_cost_price:

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

Class init_cost_price1: This class inherits the init_cost_price and is used by VM_1.

Responsibilities of each operation in init_cost_price1: Sets the current price to zero.

Class init_cost_price2:

This class inherits the init_cost_price and is used by VM_2.

Responsibilities of each operation in init_add_price2: 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: Set the current price and updates DataStore_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: Sets the current price and updates DataStore_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: Sets the current price to zero and returns the amount and updates the DataStore_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: Sets the current price to zero and returns the amount and updates the DataStore_2 about the current price change.

Class DisposeWithAdd:

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

Class dispose_with_additive1:

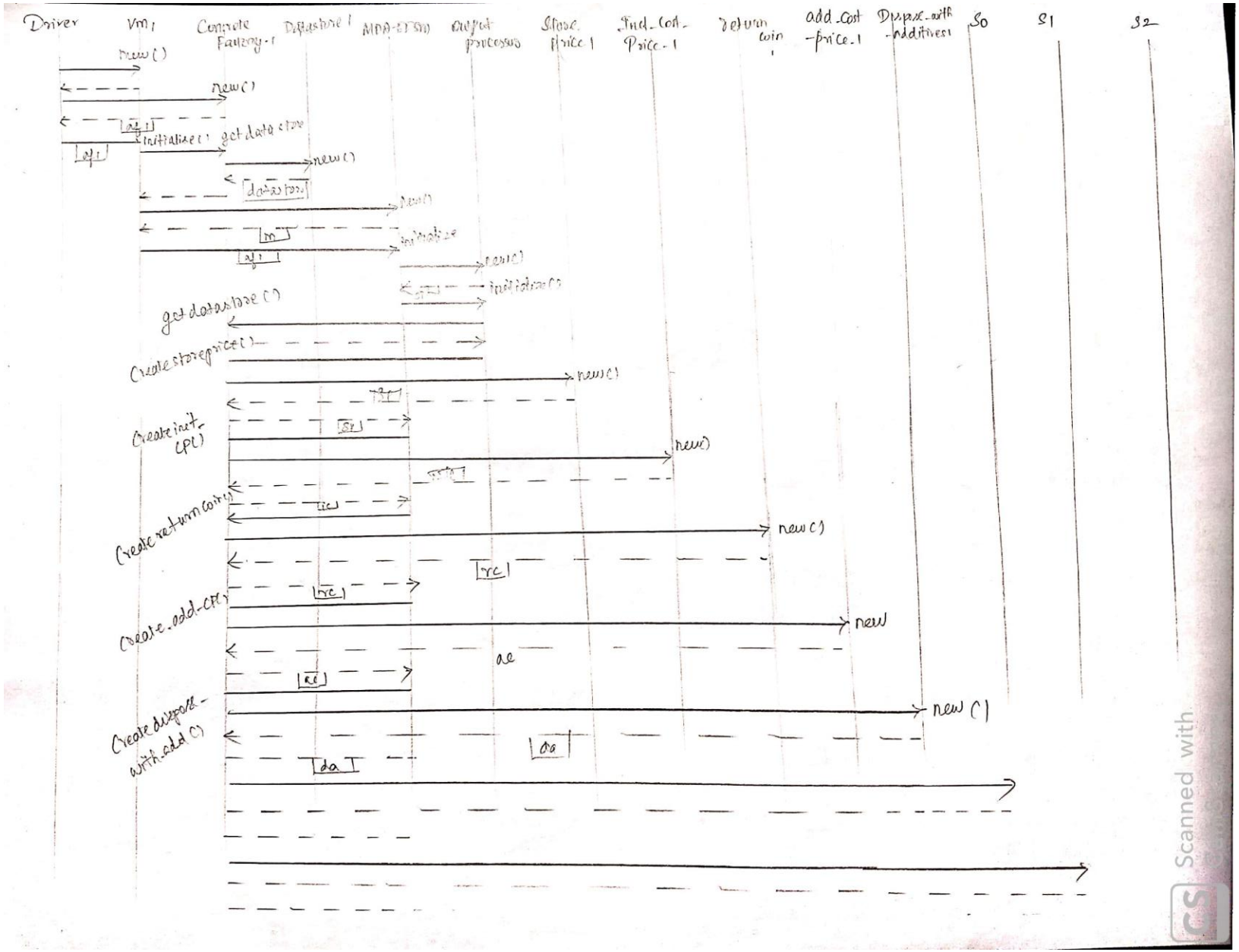
This class inherits the DisposeWithAdd and is used by VM_1.

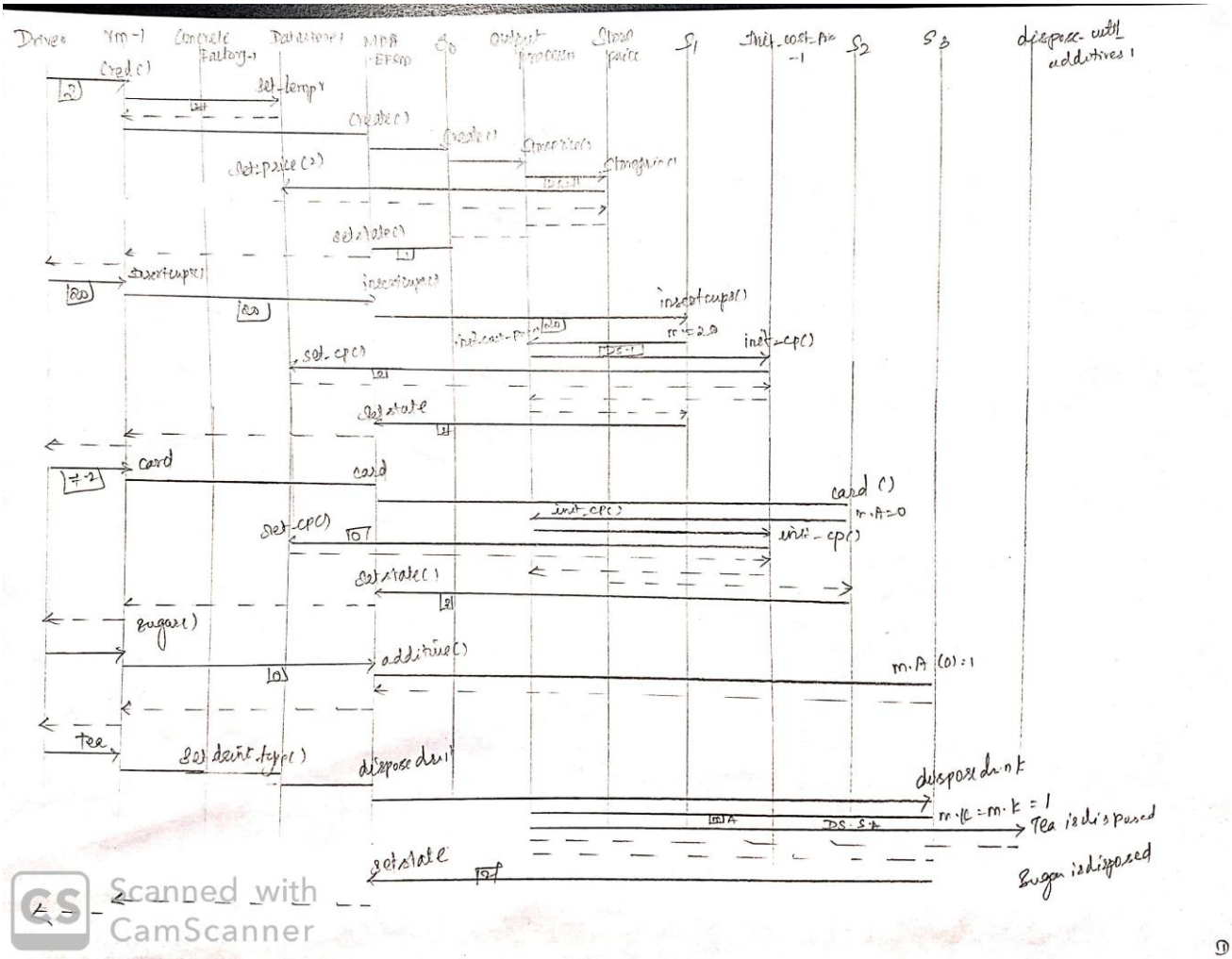
Responsibilities of each operation in dispose_with_additive1: Gets the drink type selected and checks if any additives are present and dispose them accordingly.

Class dispose_with_additive2:

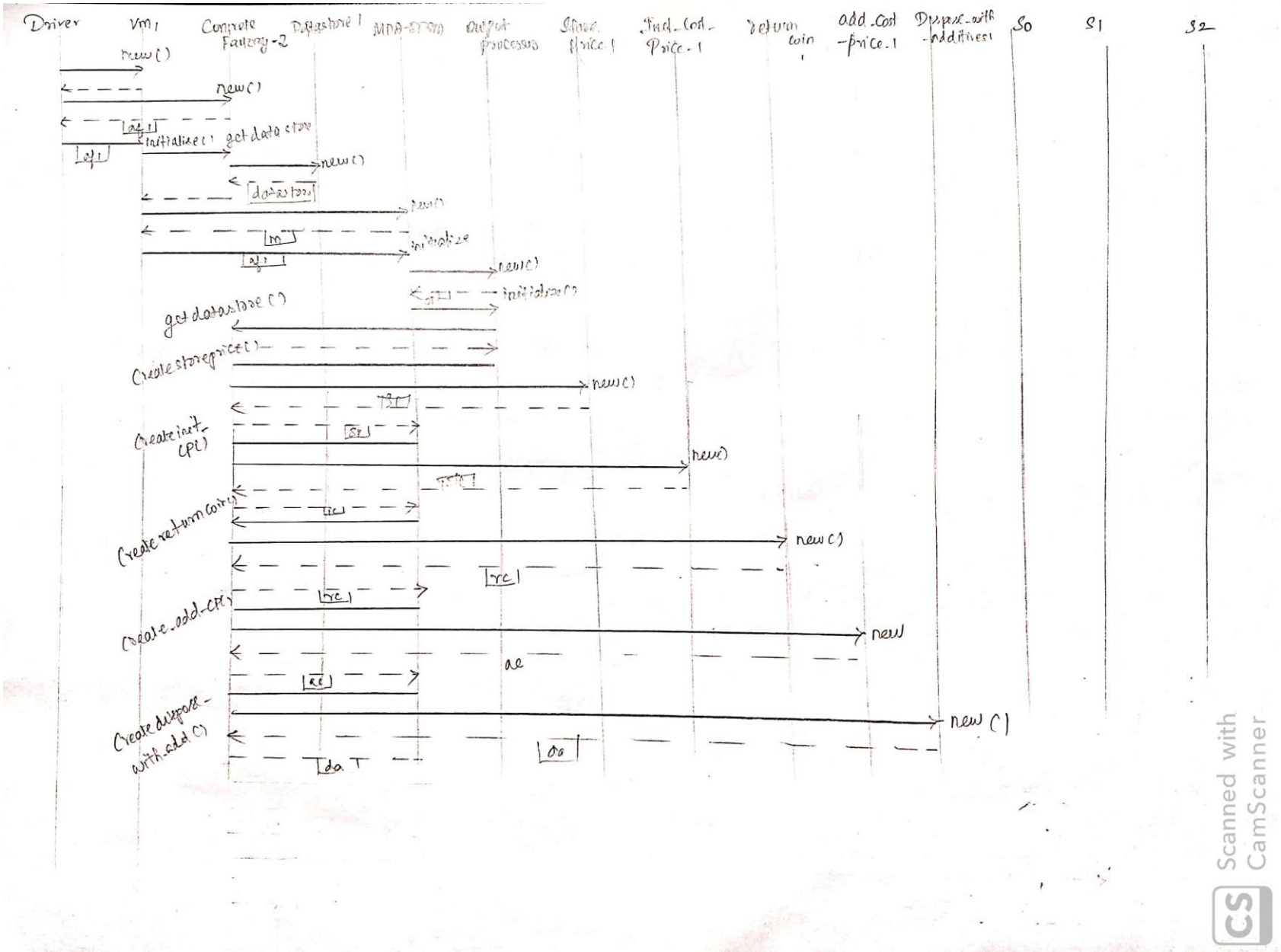
This class inherits the DiposeWithAdd and is used by VM_2.

Responsibilities of each operation in dispose_with_additive2: Gets the drink type selected and checks if any additives are present and dispose them accordingly.

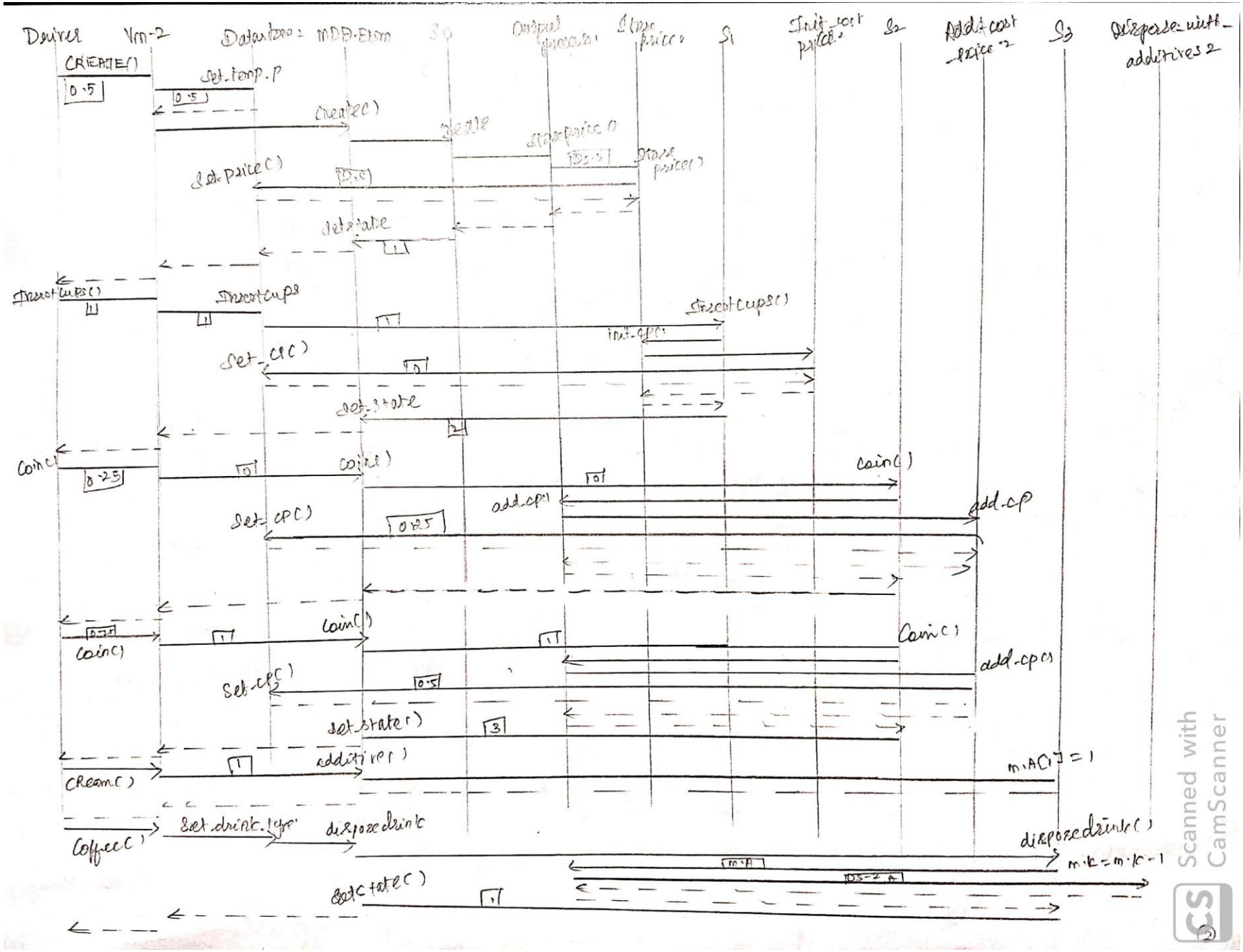

Driver
VM1
Concrete Factory-1
Diffusion1
MPA-2DSM
output processes
Store Price1
Ind_Cost_Price-1
Return coin1
add_cost_price-1
Dispose_with_additives1
S0
S1
S2
new()
initialize()
get data store
get database()
create store price()
create init_CP()
create return coin()
create add_cost()
create dispose_with_add()
as



Scanned with CamScanner



Scanned with CamScanner


Driver
Vm-2
Datastore : MDP-Elem
S0
Output processor
Store price
S1
Init_cost price
S2
Add+cost price 2
S3
Dispense_with_additives 2
CREATE()
0.5
set_temp.P
create()
create
storeprice 0
set_price()
store price()
set_state
Insertcups()
Insertcups
storecups()
init_cpc()
set_cpc()
set_state
Coin()
0.25
coin()
add_cp()
add_cp
Coin()
Coin()
add_cpc()
set_cpc()
0.5
set_state()
additive()
m.A[] = 1
Cream()
Coffee()
set_drink_type
dispose drink
dispose drink()
m.k = m.k - 1
m.A
DS-2.A
Get state()
Scanned with CamScanner