CS 586

Software Systems Architecture

Spring 2019

PROJECT REPORT

Aditya Yaji

A20426486

1. MDA-EFSM model for the Vending Machine components

a. MDA-EFSM Events:

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. MDA-EFSM Actions:

1. StorePrice() // Stores initial price, p for a drink in a temporary variable.

2. ZeroCF() // zero Cumulative Fund, cf

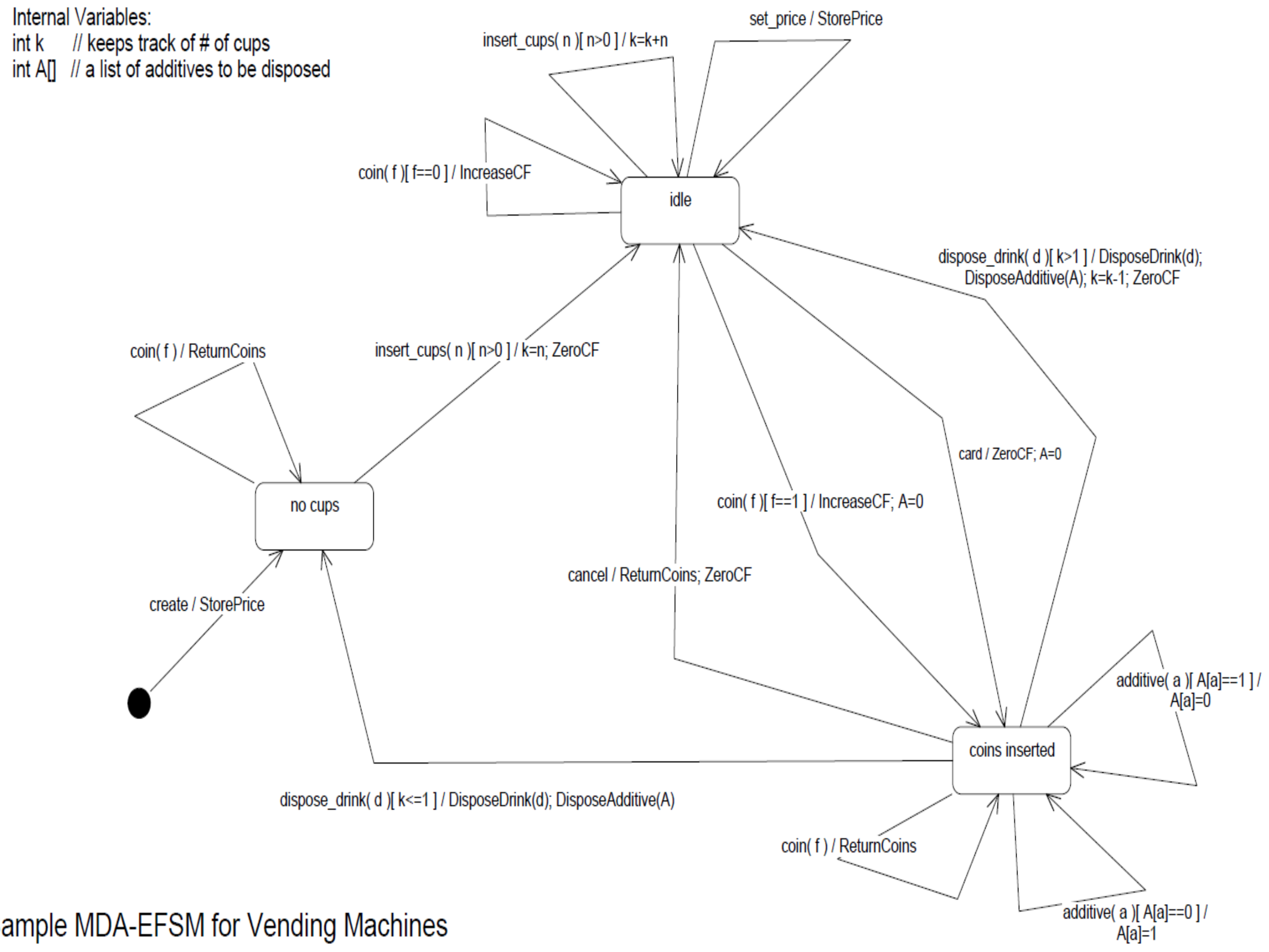
3. IncreaseCF() // increase Cumulative Fund, cf

4. ReturnCoins() // return coins inserted for a drink

5. DisposeDrink(int d) // dispose a drink with d id

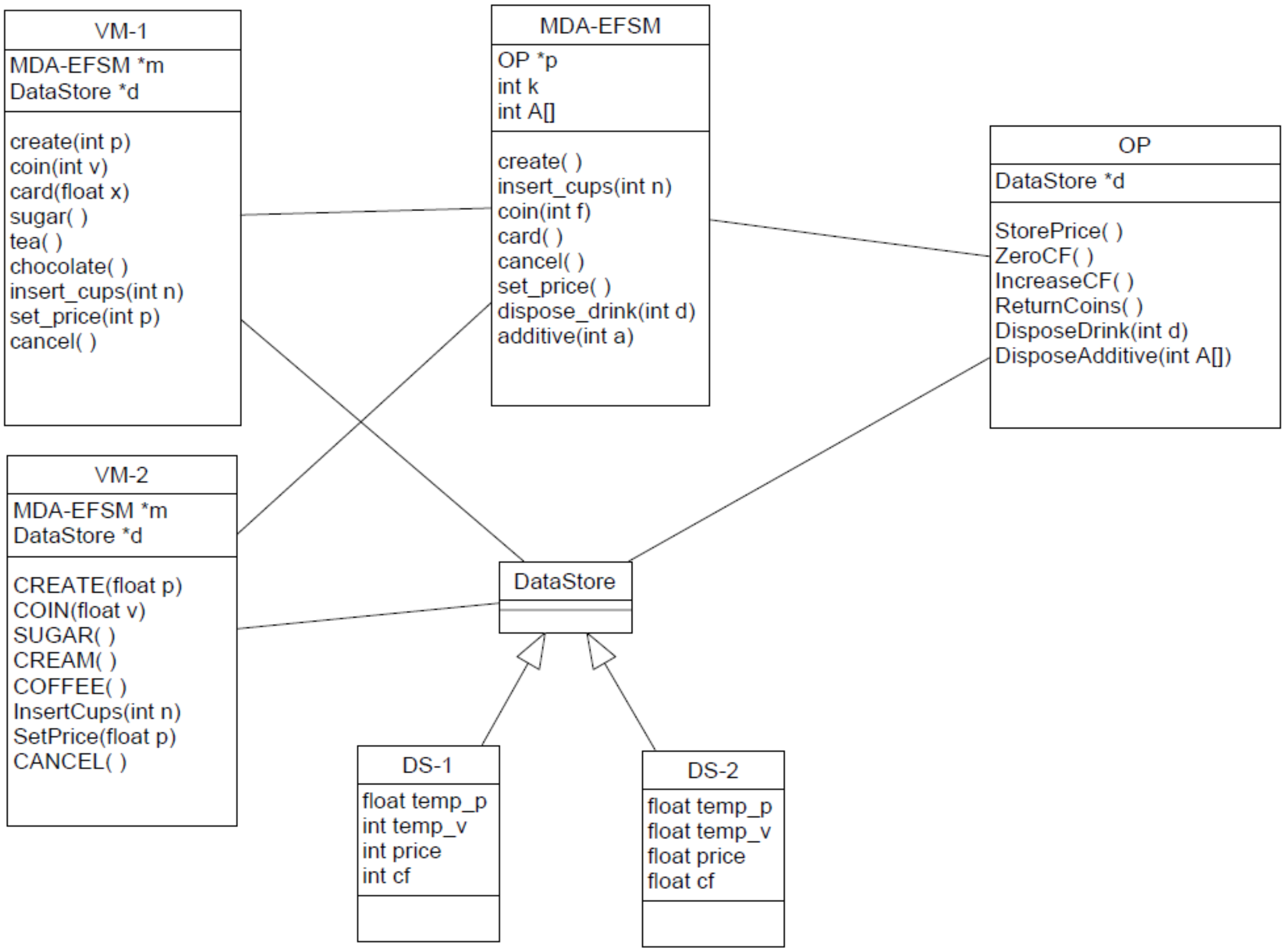
6. DisposeAdditive(int A[]) //dispose marked additives in A list,
// where additive with i id is disposed when A[i]=1

c. State diagram of the MDA-EFSM:


Internal Variables:
int k // keeps track of # of cups
int A[] // a list of additives to be disposed
set_price / StorePrice
insert_cups(n)[n>0] / k=k+n
coin(f)[f==0] / IncreaseCF
idle
dispose_drink(d)[k>1] / DisposeDrink(d); DisposeAdditive(A); k=k-1; ZeroCF
coin(f) / ReturnCoins
insert_cups(n)[n>0] / k=n; ZeroCF
card / ZeroCF; A=0
no cups
coin(f)[f==1] / IncreaseCF; A=0
cancel / ReturnCoins; ZeroCF
create / StorePrice
additive(a)[A[a]==1] / A[a]=0
coins inserted
dispose_drink(d)[k<=1] / DisposeDrink(d); DisposeAdditive(A)
coin(f) / ReturnCoins
additive(a)[A[a]==0] / A[a]=1

Sample MDA-EFSM for Vending Machines

d. Class diagram of the MDA-EFSM:



e. Pseudo-code of all operations of Input Processors of Vending Machines: VM-1 and VM-2:

Vending Machine-1(VM1):

```
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();
}
```

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

Vending Machine-2 (VM-2):

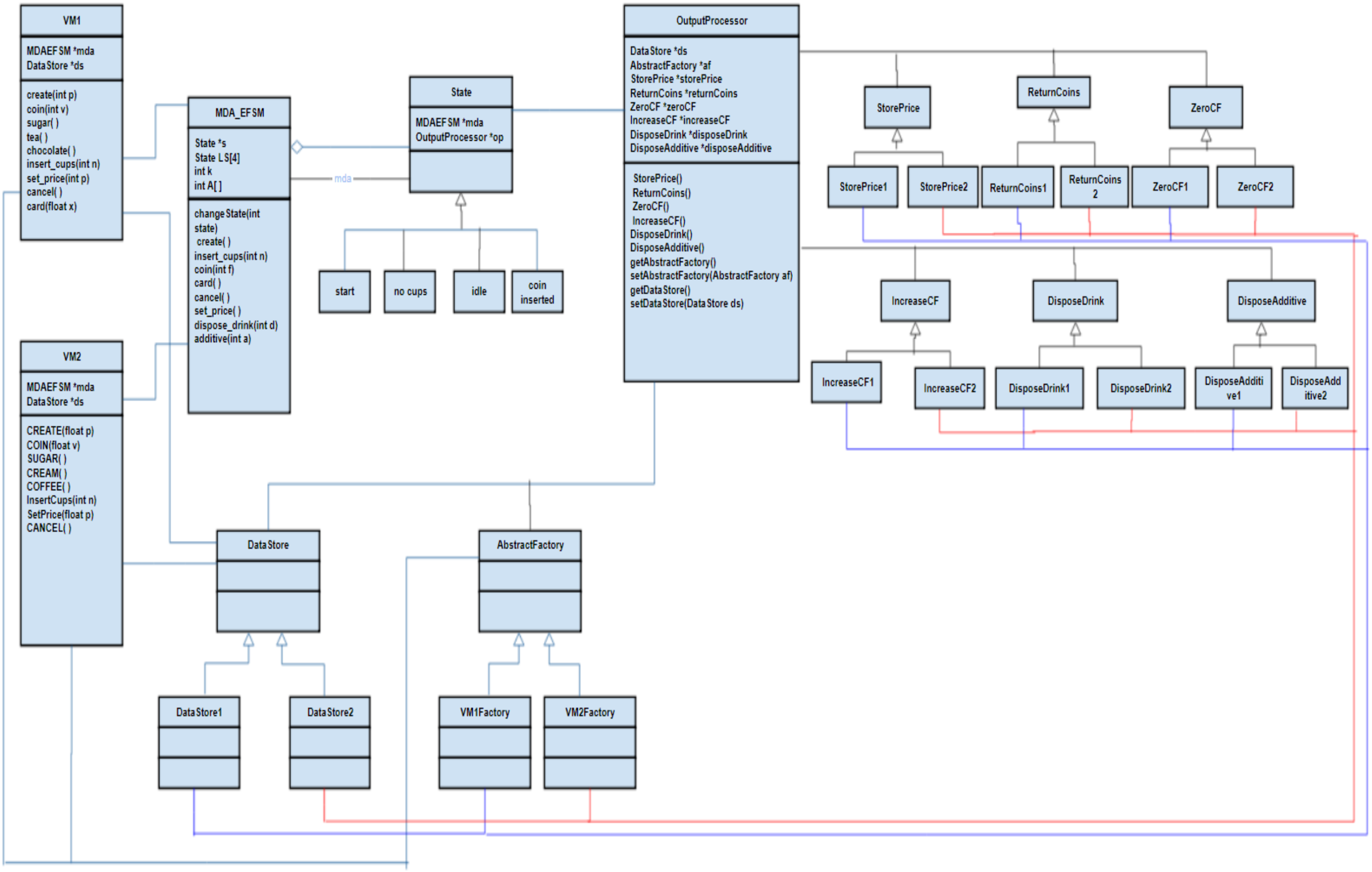
```
CREATE(float p) {
   d->temp_p=p;
   m->create();
}

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();
}
```

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

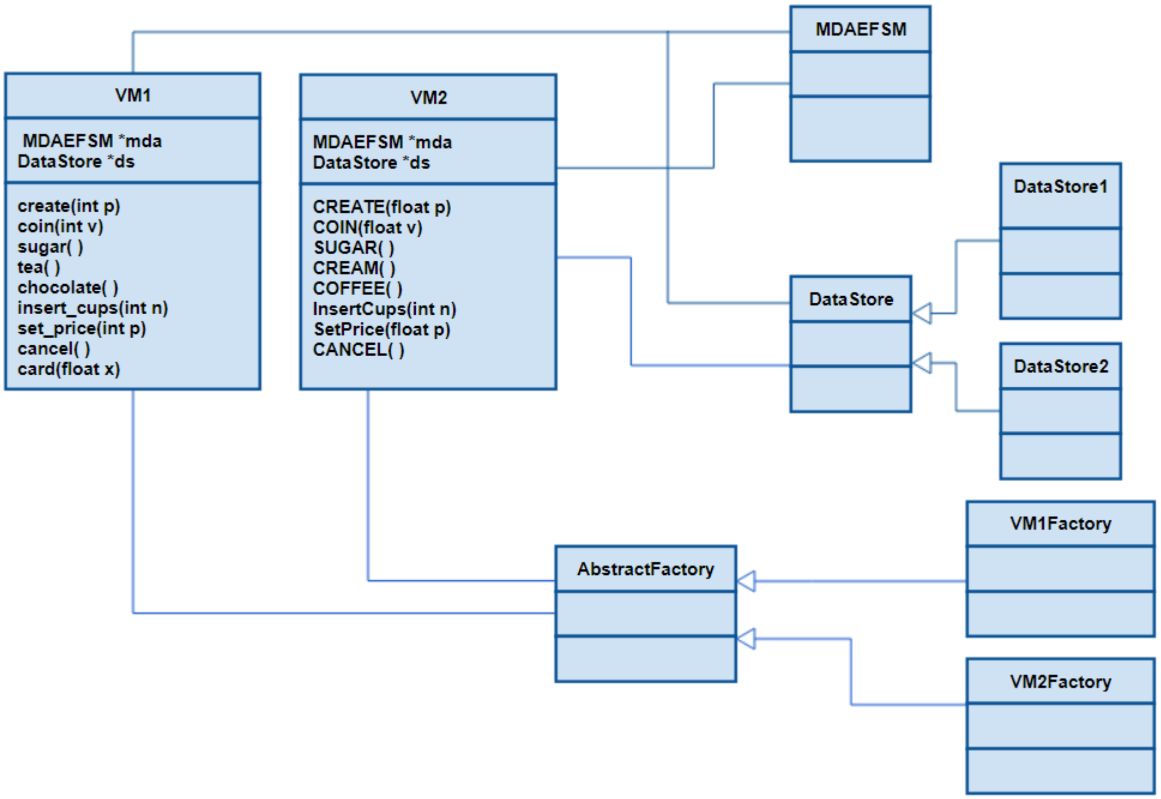
2. MDA-EFSM model for the Vending Machine components

a. Vending-Machine components full architecture:

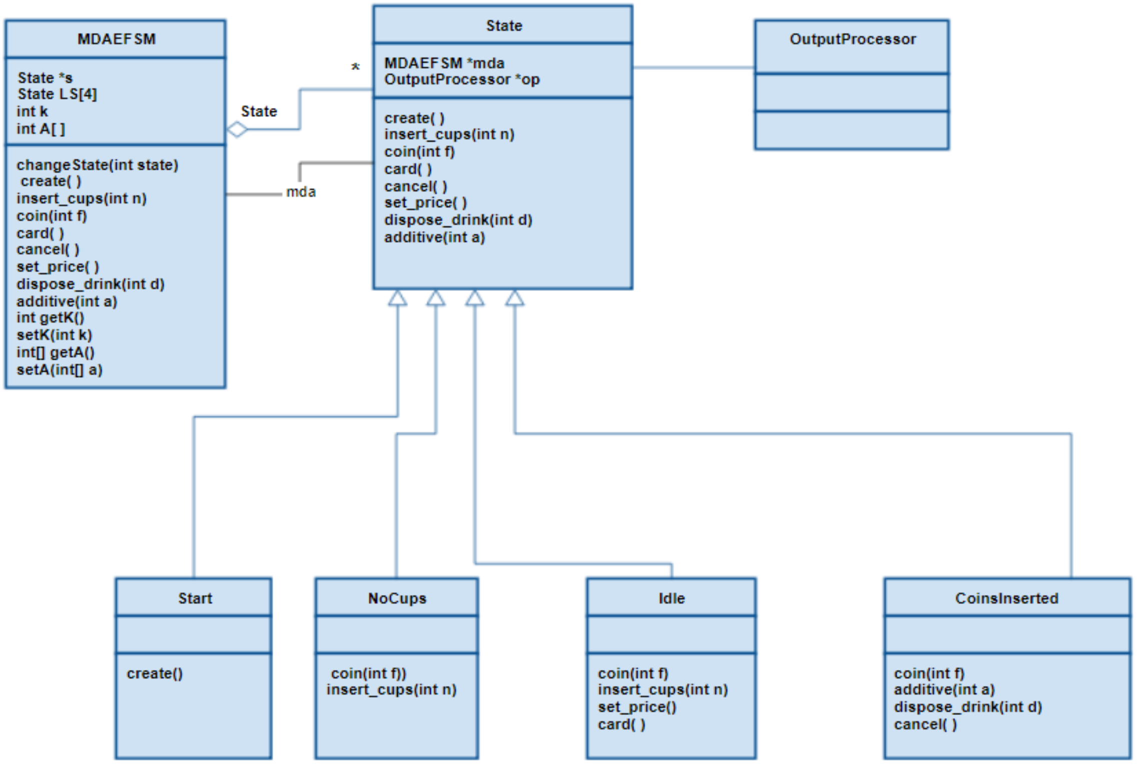


(Descriptive diagrams of each component/class is further show in the report)

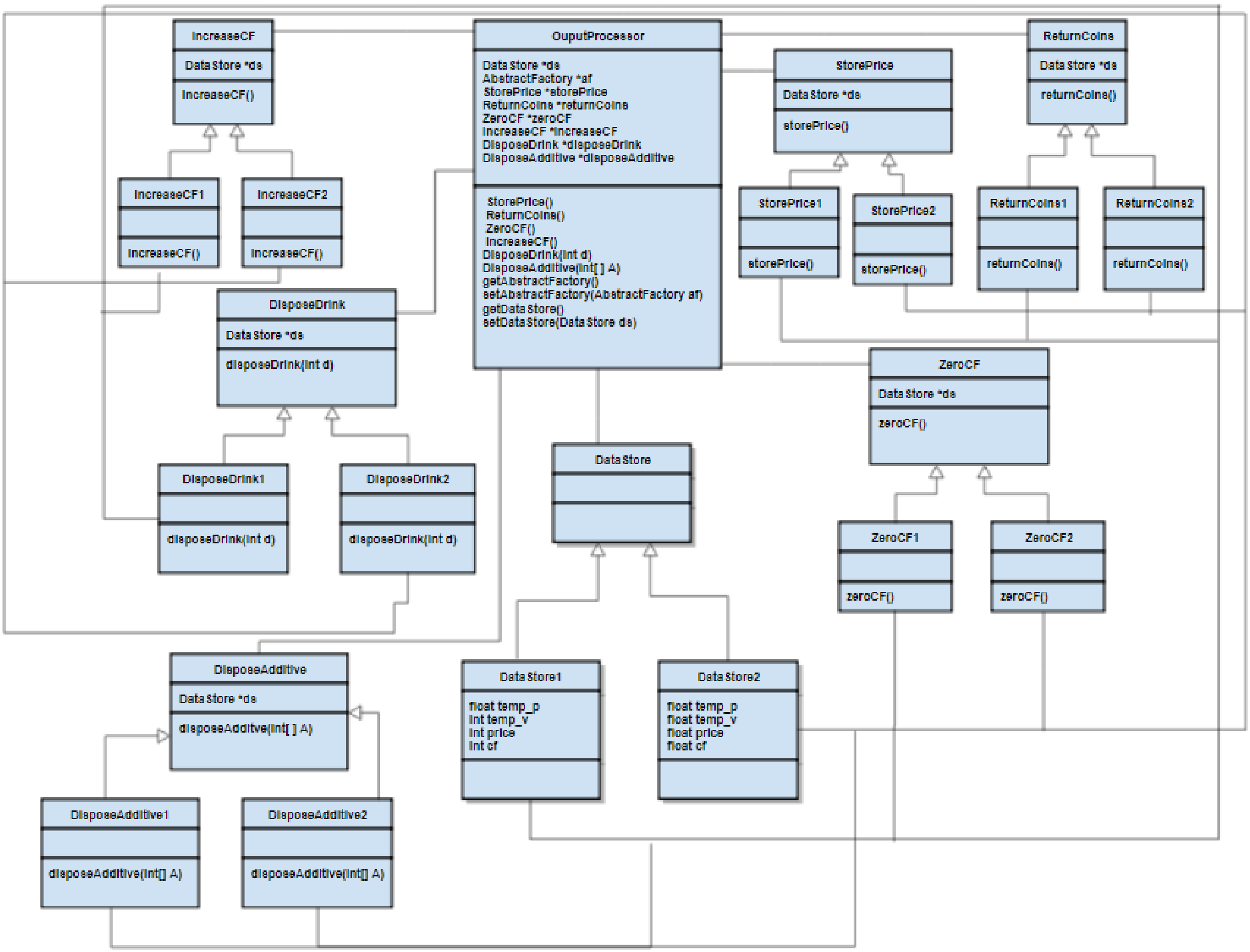
b. **Class diagram for input processor:** (rest description continued in next diagram)



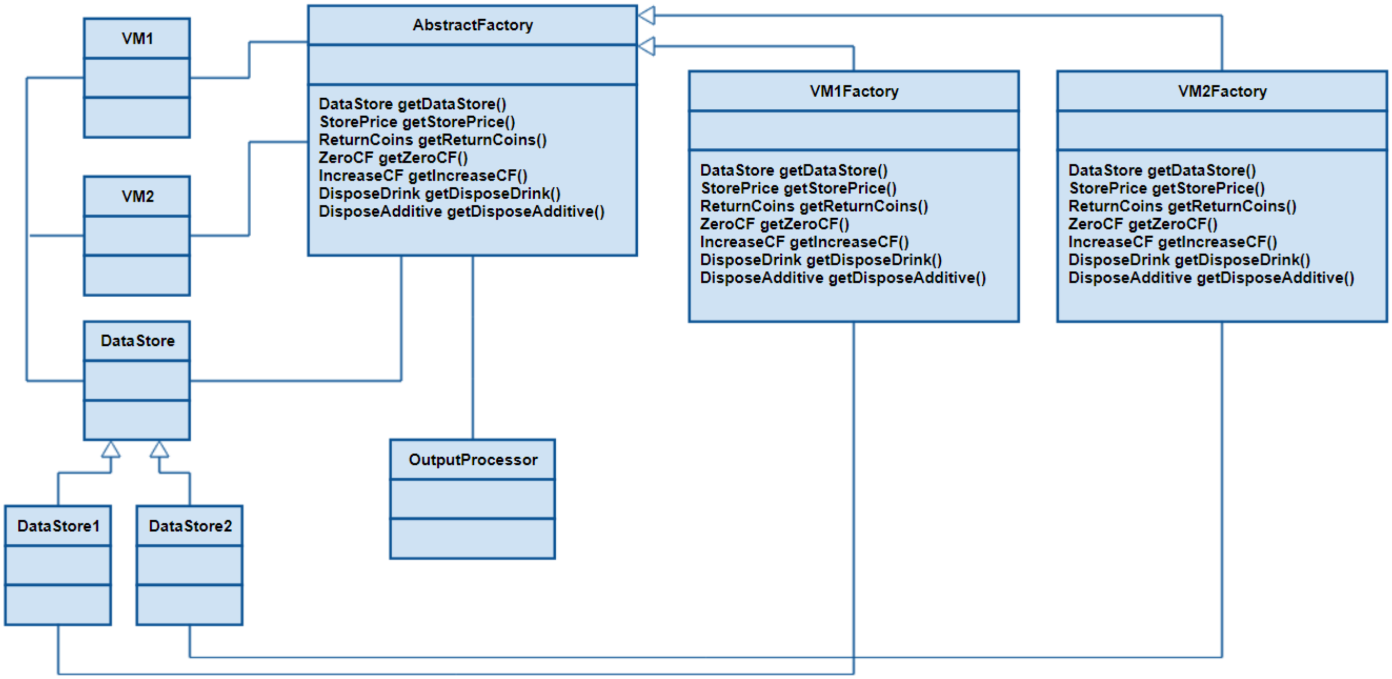
c. **Class diagram for MDA EFSM model (State Pattern):** (rest description continued in next diagram)



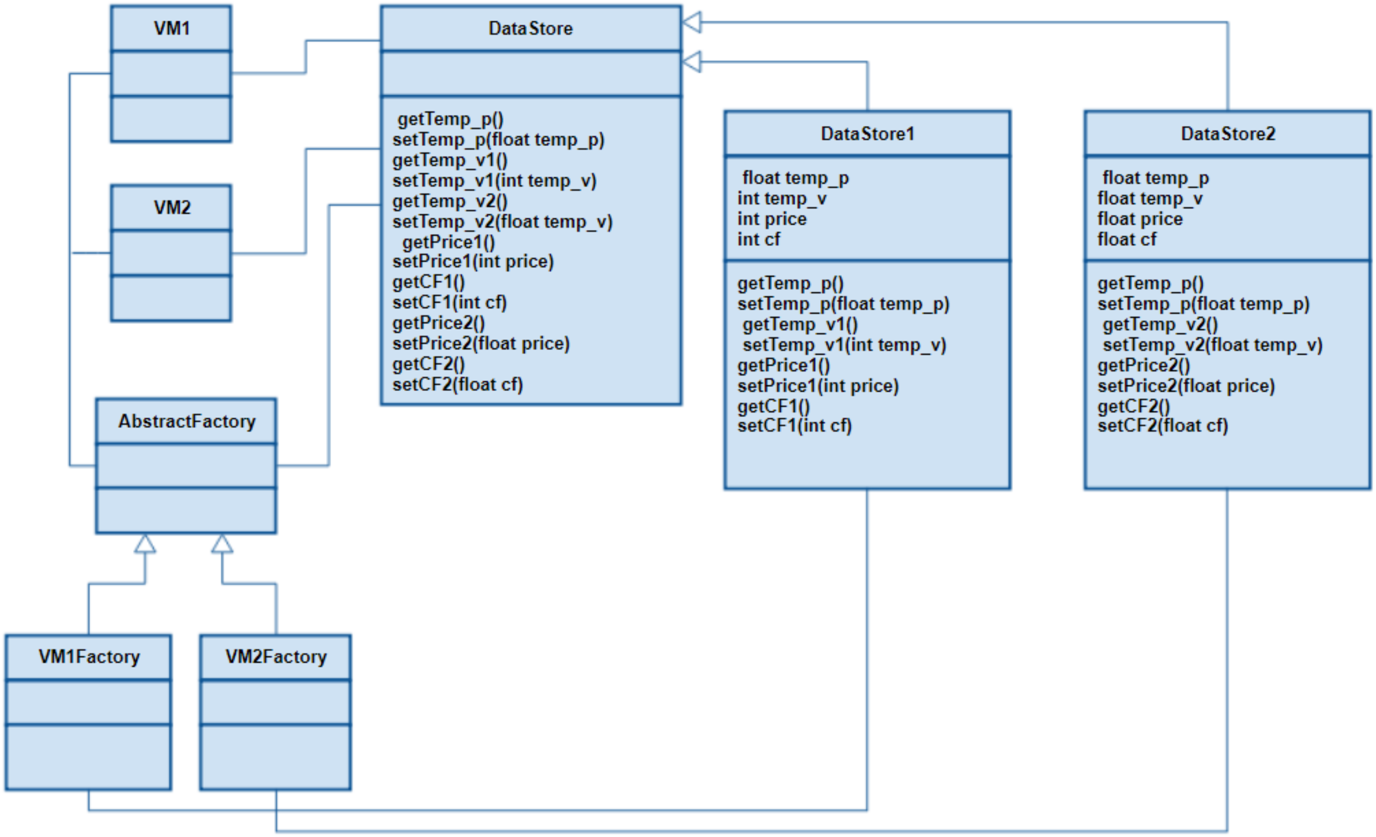
d. Class diagram for output processor (Strategy Pattern): (rest description continued in next diagram)



e. Class diagram for Abstract Factory Pattern: (rest description continued in next diagram)


VM1
AbstractFactory
DataStore getDataStore()
StorePrice getStorePrice()
ReturnCoins getReturnCoins()
ZeroCF getZeroCF()
IncreaseCF getIncreaseCF()
DisposeDrink getDisposeDrink()
DisposeAdditive getDisposeAdditive()
VM1Factory
DataStore getDataStore()
StorePrice getStorePrice()
ReturnCoins getReturnCoins()
ZeroCF getZeroCF()
IncreaseCF getIncreaseCF()
DisposeDrink getDisposeDrink()
DisposeAdditive getDisposeAdditive()
VM2Factory
DataStore getDataStore()
StorePrice getStorePrice()
ReturnCoins getReturnCoins()
ZeroCF getZeroCF()
IncreaseCF getIncreaseCF()
DisposeDrink getDisposeDrink()
DisposeAdditive getDisposeAdditive()
VM2
DataStore
OutputProcessor
DataStore1
DataStore2

f. Class diagram for data store:



3. Description of responsibilities in each class and the operations supported by those classes:

Class Driver	
Purpose	This class is used to run two Vending Machine components.
Operations	
main(String[] args)	This method is used to run two Vending Machine components.

Class VM1	
Purpose	This class represents VM1 and supports all the operations provided by VM1. This is a part of input processor.
Attributes	
MDAEFSM *mda	Pointer to MDAEFSM object.
DataStore *ds	Pointer to DataStore object.
Operations	
create(int p)	This method is used to start Vending Machine-1 where p represents initial price of the drink and stores in a temporary var called float temp_v.
coin(int v)	This method is used to get drink by inserting coins, v in VM.
card(float x)	This method is used to pay for drink by card, where x is the limit of card.
sugar()	This method is used to dispose sugar additive.
tea()	This method is used to dispose cup of tea.
chocolate()	This method is used to dispose cup of chocolate.
insert_cups(int n)	This method is used to insert n cups to the VM.
set_price(int p)	This method is used to set price, p for the drink.
cancel()	This method is used to cancel the operation.

Class VM2	
Purpose	This class represents VM2 and supports all the operations provided by VM2. This is a part of input processor.
Attributes	
MDAEFSM *mda	Pointer to MDAEFSM object.
DataStore *ds	Pointer to DataStore object.
Operations	
CREATE(float p)	This method is used to start Vending Machine-2 where p represents initial price of the drink and stores in a temporary var called float temp_v.
COIN(float v)	This method is used to get drink by inserting coins, v in VM.
SUGAR()	This method is used to dispose sugar additive.
CREAM()	This method is used to dispose cup of tea.
COFFEE()	This method is used to dispose cup of chocolate.
InsertCups(int n)	This method is used to insert n cups to the VM.
SetPrice(float p)	This method is used to set price, p for the drink.
CANCEL()	This method is used to cancel the operation.

Class MDAEFSM	
Purpose	This class represents the MDAEFSM. It supports the MDAEFSM events. This class is also a context class of State Pattern.
Attributes	
State[] LS	Stores the objects of different state classes.
State *s	Pointer to current state of MDAEFSM.
int k	Keeps track of # of cups.
int A[]	A list of additives to be disposed.
Operations	
ChangeState(int state)	This method is used to change state.
create()	This method is used to activate the Vending machine.
coin(int f)	This method is used to pay by coins/money for a drink where: f=0: insufficient coins for a drink, f=1: sufficient coins for a drink and f=2: returns back coins inserted as it is not in idle(good) state to process.
card()	This is the method to pay for drink using card.
additive(int a)	This is the method for disposing additive with drink where a id is used to select the type of additive. In VM1: a=1(sugar) and in VM2: a=1(cream) and a=2(sugar).
dispose_drink(int d)	This method is used to dispose cup of drink where d id represents the type of drink. In VM1: d=1(tea) and d=2(chocolate). In VM2: d=1(coffee).
insert_cups(int n)	This is the method for inserting n # of cups to the VM.
set_price()	This method is used to set the price for the drink in idle state.
cancel()	This is the method for cancelling the current operation of the VM.
getK()	Getter method to obtain # of cups in the machine.
setK(int k)	Setter method to set # of cups in the machine.
getA()	Getter method for a list of additives to be disposed.
setA(int[] A)	Setter method for a list of additives to be disposed.

Class State	
Purpose	This class is state class of State Pattern. It represents the state for MDAEFSM.
Attributes	
MDAEFSM *mda	Pointer to MDAEFSM object.
OutputProcessor *op	Pointer to OutputProcessor class object.
Abstract Operations	
create()	This method is used to activate the Vending machine.
coin(int f)	This method is used to pay by coins/money for a drink where: f=0: insufficient coins for a drink, f=1: sufficient coins for a drink and f=2: returns back coins inserted as it is not in idle(good) state to process.
card()	This is the method to pay for drink using card.
additive(int a)	This is the method for disposing additive with drink where a is used to select the type of additive. In VM1: a=1(sugar) and in VM2: a=1(cream) and a=2(sugar).

dispose_drink(int d)	This method is used to dispose cup of drink where d represents the type of drink. In VM1: d=1(tea) and d=2(chocolate). In VM2: d=1(coffee).
insert_cups(int n)	This is the method for inserting n # of cups to the VM.
set_price()	This method is used to set the price for the drink in idle state.
cancel()	This is the method for cancelling the current operation of the VM.
Operations	
getMDAEFSM()	This method is used to get MDAEFSM object.
setMDAEFSM(MDAEFSM mda)	This method is used to set MDAEFSM object.
getOp()	This method is used to get OutputProcessor object.
setOp(OutputProcessor op)	This method is used to set OutputProcessor object.

Class Start	
Purpose	This is a subclass of State class and represents Start state.
Operation	
create()	This method is used to start the VM process. It changes state from start to no cups state. It executes the storePrice() action and set # of cups to zero.

Class NoCups	
Purpose	This is a subclass of State class and represents no cups state.
Operation	
coin(int f)	This method is used to pay by coins/money for a drink where f=2: returns back coins inserted as it is not in idle(good) state to process. It executes returnCoins() action and remains in no cups state itself.
insert_cups(int n)	This is the method for inserting n # of cups to the VM. Initializes n value to k and executes zeroCF() action if n>0. It changes state from no cups to idle state.

Class Idle	
Purpose	This is a subclass of State class and represents idle state.
Operation	
coin(int f)	This method is used to pay by coins/money for a drink where: f=0: insufficient coins for a drink and f=1: sufficient coins for a drink. It executes IncreaseCF() action in both cases but sets additive array-A to zero if sufficient amount (f=1).
insert_cups(int n)	This is the method for inserting n # of cups to the VM and increments the # of cups added, k if n>0. This operation remains in idle state itself.
set_price()	This method is used to set the price for the drink in idle state. It executes storePrice() action. This operation remains in idle state itself.
card()	This is the method to pay for drink using card. It executes zeroCF() action and initializes additive array-A to zero. This operation causes change of state from idle to coin-inserted.

Class CoinsInserted	
Purpose	This is a subclass of State class and represents coins inserted state.
Operation	
coin(int f)	This method is used to pay by coins/money for a drink where f=2: returns back coins inserted as it is not in idle(good) state to process. It executes returnCoins() action and remains in coins inserted state itself.
cancel()	This is the method for cancelling the current operation of the VM. It executes returnCoins() and zeroCF() action. This operation does not change the state and remains in coins inserted state itself.
dispose_drink(int d)	This method is used to dispose cup of drink where d id represents the type of drink. In VM1: d=1(tea) and d=2(chocolate). In VM2: d=1(coffee). It keeps track on # of cups, if k<=1 then it goes to no-cups state else, it goes to idle state with decrement in cups and executing zeroCF() action. It executes both DisposeDrink(int d) and DisposeAdditive(int A[]) actions in both possibilities.
additive(int a)	This method is used to dispose additive where a id represents the type of additive.

Class OutputProcessor	
Purpose	This class represents Output Processor and used to execute actions.
Attributes	
DataStore *ds	Pointer to DataStore object.
AbstractFactory *af	Pointer to AbstractFactory object.
StorePrice *storePrice	Pointer to StorePrice object.
ReturnCoins * returnCoins	Pointer to ReturnCoins object.
ZeroCF *zeroCF	Pointer to ZeroCF object.
IncreaseCF *increaseCF	Pointer to IncreaseCF object.
DisposeDrink * disposeDrink	Pointer to DisposeDrink object.
DisposeAdditive *disposeAdditive	Pointer to DisposeAdditive object.
Operations	
storePrice()	This is for storePrice() action. It creates StorePrice object using AbstractFactory class and it executes the storePrice() method of StorePrice class.
returnCoins()	This is for returnCoins() action. It creates ReturnCoins object using AbstractFactory class and it executes the returnCoins() method of ReturnCoins class.
zeroCF()	This is for zeroCF() action. It creates ZeroCF object using AbstractFactory class and it executes the zeroCF() method of ZeroCF class.
increaseCF()	This is for increaseCF() action. It creates IncreaseCF object using AbstractFactory class and it executes the increaseCF() method of IncreaseCF class.
disposeDrink(int d)	This is for disposeDrink(int d) action. It creates DisposeDrink object using AbstractFactory class and it executes the disposeDrink(int d) method of DisposeDrink class.

disposeAdditive(int[] A)	This is for disposeAdditive(int[] A) action. It creates DisposeAdditive object using AbstractFactory class and it executes the disposeAdditive(int[] A) method of DisposeAdditive class.
getAbstractFactory()	Get the AbstractFactory object.
setAbstractFactory(AbstractFactory af)	Set the AbstractFactory object.
getDataStore()	Get the DataStore object.
setDataStore(DataStore ds)	Set the DataStore object.

Interface StorePrice	
Purpose	This is an interface to store initial price of a drink.
Abstract Operations	
storePrice()	This is an abstract method for storing initial price of a drink.
getDataStore()	This method is used to get the DataStore object
setDataStore(DataStore ds)	This method is used to set the DataStore object

Class StorePrice1	
Purpose	This class is subclass of StorePrice and is used to store initial price of a drink.
Operation	
storePrice()	This method is used for storing the initial price of a drink.

Class StorePrice2	
Purpose	This class is subclass of StorePrice and is used to store initial price of a drink.
Operation	
storePrice()	This method is used for storing the initial price of a drink.

Interface ReturnCoins	
Purpose	This is an interface to return back coins inserted when in not ready state (idle and coins inserted state).
Abstract Operations	
returnCoins()	This is an abstract method to return back coins inserted when in idle and coins inserted state.
getDataStore()	This method is used to get the DataStore object
setDataStore(DataStore ds)	This method is used to set the DataStore object

Class ReturnCoins1	
Purpose	This class is subclass of ReturnCoins and is used to return back coins inserted when in idle and coins inserted state.
Operation	
returnCoins()	This method is used to return back coins inserted when in idle and coins inserted state.

Class ReturnCoins2	
Purpose	This class is subclass of ReturnCoins and is used to return back coins inserted when in idle and coins inserted state.
Operation	
returnCoins()	This method is used to return back coins inserted when in idle and coins inserted state.

Interface ZeroCF	
Purpose	This is an interface used to initialize cumulative fund to zero.
Abstract Operations	
zeroCF()	This is an abstract method used to initialize cumulative fund to zero.
getDataStore()	This method is used to get the DataStore object
setDataStore(DataStore ds)	This method is used to set the DataStore object

Class ZeroCF1	
Purpose	This class is subclass of ZeroCF and is used to initialize cumulative fund to zero.
Operation	
zeroCF1()	This method is used to initialize cumulative fund to zero.

Class ZeroCF2	
Purpose	This class is subclass of ZeroCF and is used to initialize cumulative fund to zero.
Operation	
zeroCF1()	This method is used to initialize cumulative fund to zero.

Interface IncreaseCF	
Purpose	This is an interface used to increase cumulative fund value by adding to existing cumulative fund.
Abstract Operations	

increaseCF()	This is an abstract method used to increase cumulative fund value by adding to existing cumulative fund.
getDataStore()	This method is used to get the DataStore object
setDataStore(DataStore ds)	This method is used to set the DataStore object

Class IncreaseCF1	
Purpose	This class is subclass of IncreaseCF and is used to increase cumulative fund value by adding to existing cumulative fund.
Operation	
increaseCF1()	This method is used to increase cumulative fund value by adding to existing cumulative fund.

Class IncreaseCF2	
Purpose	This class is subclass of IncreaseCF and is used to increase cumulative fund value by adding to existing cumulative fund.
Operation	
increaseCF2()	This method is used to increase cumulative fund value by adding to existing cumulative fund.

Interface DisposeDrink	
Purpose	This is an interface used to dispose cup of drink.
Abstract Operations	
disposeDrink(int d)	This is an abstract method used to dispose cup of drink with d id.
getDataStore()	This method is used to get the DataStore object
setDataStore(DataStore ds)	This method is used to set the DataStore object

Class DisposeDrink1	
Purpose	This class is subclass of DisposeDrink and is used to dispose cup of drink.
Operation	
disposeDrink(int d)	This method is used to dispose a drink with d id.

Class DisposeDrink2	
Purpose	This class is subclass of DisposeDrink and is used to dispose cup of drink.
Operation	
disposeDrink(int d)	This method is used to dispose a drink with d id.

Interface DisposeAdditive	
Purpose	This is an interface used to dispose additive for a drink.
Abstract Operations	
disposeAdditive(int[] A)	This is an abstract method used to dispose additive for a drink.
getDataStore()	This method is used to get the DataStore object
setDataStore(DataStore ds)	This method is used to set the DataStore object

Class DisposeAdditive1	
Purpose	This class is subclass of DisposeAdditive and is used to dispose additive for a drink.
Operation	
disposeAdditive(int[] A)	This method is used to dispose marked additives in A list where additive with i id is disposed when A[i]=1.

Class DisposeAdditive2	
Purpose	This class is subclass of DisposeDrink and used to dispose additive for a drink.
Operation	
disposeAdditive(int[] A)	This method is used to dispose marked additives in A list where additive with i id is disposed when A[i]=1.

Interface AbstractFactory	
Purpose	This interface is used to create DataStore and actions objects. It is a part of Abstract Factory design pattern.
Abstract Operations	
DataStore getDataStore()	This is an abstract method to create and return DataStore object.
StorePrice getStorePrice()	This is an abstract method to create and return StorePrice object.
ReturnCoins getReturnCoins()	This is an abstract method to create and return ReturnCoins object.
ZeroCF getZeroCF()	This is an abstract method to create and return ZeroCF object.
IncreaseCF getIncreaseCF()	This is an abstract method to create and return IncreaseCF object.
DisposeDrink getDisposeDrink()	This is an abstract method to create and return DisposeDrink object.
DisposeAdditive getDisposeAdditive()	This is an abstract method to create and return DisposeAdditve object.

Class VM1Factory	
Purpose	This class is used to create the data store and actions objects for VM1. This class is concrete factory class for VM1 and this is a part of Abstract factory design pattern.
Operations	

DataStore getDataStore()	This is a method to create and return DataStore1 object.
StorePrice getStorePrice()	This is a method to create and return StorePrice1 object.
ReturnCoins getReturnCoins()	This is a method to create and return ReturnCoins1 object.
ZeroCF getZeroCF()	This is a method to create and return ZeroCF1 object.
IncreaseCF getIncreaseCF()	This is a method to create and return IncreaseCF1 object.
DisposeDrink getDisposeDrink()	This is a method to create and return DisposeDrink1 object.
DisposeAdditive getDisposeAdditive()	This is a method to create and return DisposeAdditive1 object.

Class VM2Factory	
Purpose	This class is used to create the data store and actions objects for VM2. This class is concrete factory class for VM2 and this is a part of Abstract factory design pattern.
Operations	
DataStore getDataStore()	This is a method to create and return DataStore2 object.
StorePrice getStorePrice()	This is a method to create and return StorePrice2 object.
ReturnCoins getReturnCoins()	This is a method to create and return ReturnCoins2 object.
ZeroCF getZeroCF()	This is a method to create and return ZeroCF2 object.
IncreaseCF getIncreaseCF()	This is a method to create and return IncreaseCF2 object.
DisposeDrink getDisposeDrink()	This is a method to create and return DisposeDrink2 object.
DisposeAdditive getDisposeAdditive()	This is a method to create and return DisposeAdditive2 object.

Class DataStore	
Purpose	This is an abstract class and is used to store data.
Abstract Operations	
getTemp_p()	This is an abstract method to get the price value stored in temporary variable temp_p from DataStore1 and DataStore2.
setTemp_p(float temp_p)	This is an abstract method to set the value of temporary variable temp_p in DataStore1 and DataStore2.
getTemp_v1()	This is an abstract method to get the integer coin value stored in temporary variable temp_v from DataStore1.
setTemp_v1(int temp_v)	This is an abstract method to set the value of temporary variable temp_v in DataStore1.
getPrice1()	This is an abstract method to get the integer value of price from DataStore1.
setPrice1(int price)	This is an abstract method to set the integer value of price in DataStore1.
getCF1()	This is an abstract method to get the value of cumulative fund, cf inserted for a drink from DataStore1.
setCF1(int cf)	This is an abstract method to set the value of cumulative fund, cf inserted for a drink in DataStore1.
getTemp_v2()	This is an abstract method to get the float coin value stored in temporary variable temp_v from DataStore2.

setTemp_v2(float temp_v)	This is an abstract method to set the float value of temporary variable temp_v in DataStore2.
getPrice2()	This is an abstract method to get the float value of price from DataStore2.
setPrice2(float price)	This is an abstract method to set the float value of price in DataStore2.
getCF2()	This is an abstract method to get the value of cumulative fund, cf inserted for a drink from DataStore2.
setCF2(int cf)	This is an abstract method to set the value of cumulative fund, cf inserted for a drink in DataStore2.

Class DataStore1	
Purpose	This class is used to store data for VM1.
Attributes	
float temp_p	This is a temporary variable which stores the value of price, p for a drink.
int temp_v	This is a temporary variable which stores the value of coin inserted, v for a drink.
int price	This is a variable which stores the value of price for a drink.
int cf	This variable stores the cumulative fund inserted for a drink.
Operations	
getTemp_p()	This is a method to get the price value stored in temporary variable temp_p.
setTemp_p(float temp_p)	This is a method to set the value of temporary variable temp_p.
getTemp_v1()	This is a method to get the integer coin value stored in temporary variable temp_v.
setTemp_v1(int temp_v)	This is a method to set the value of temporary variable temp_v.
getPrice1()	This is a method to get the integer value of variable price.
setPrice1(int price)	This is a method to set the integer value of variable price.
getCF1()	This is a method to get the integer value of cumulative fund, cf inserted for a drink.
setCF1(int cf)	This is a method to set the value of cumulative fund, cf inserted for a drink.

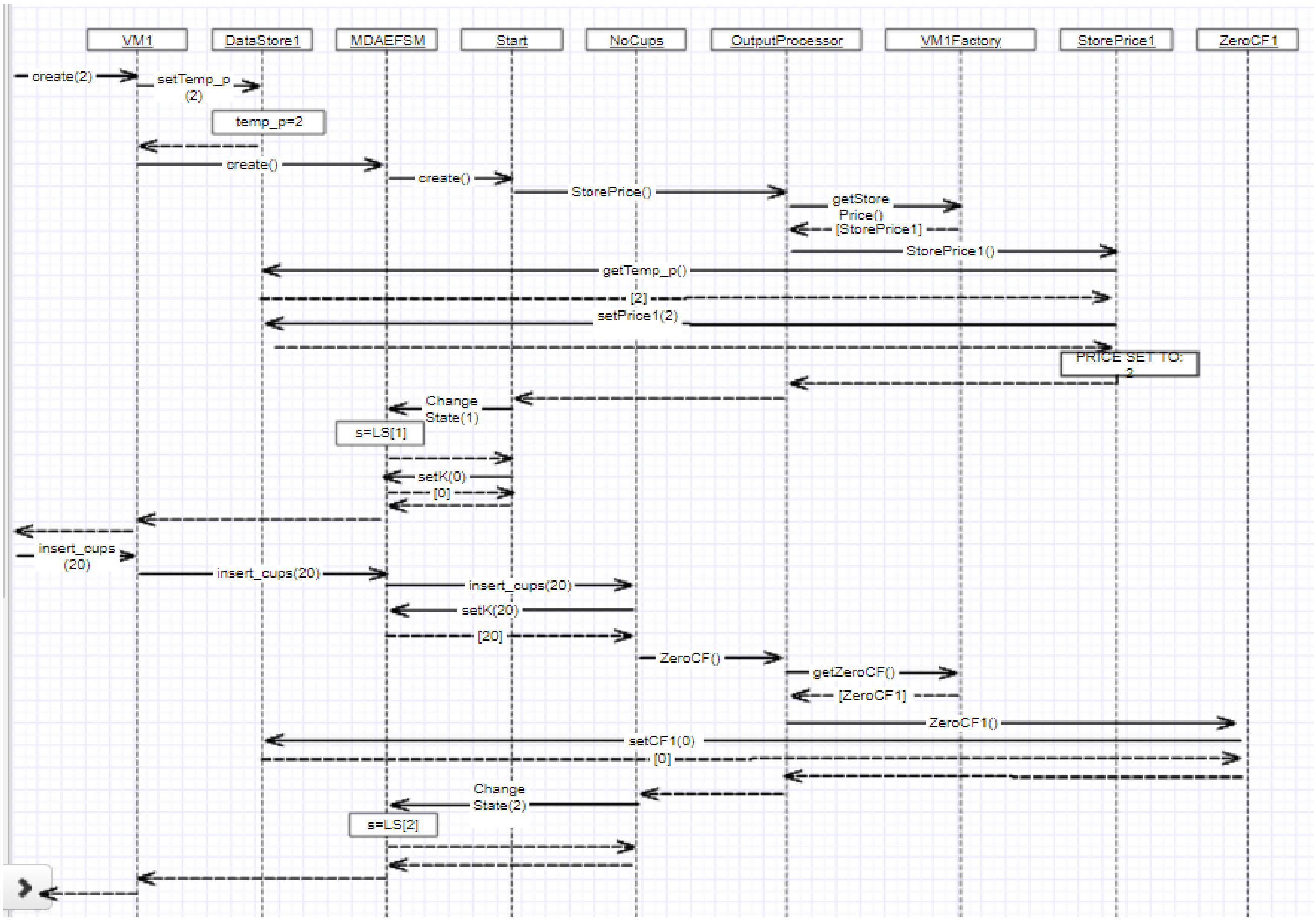
Class DataStore2	
Purpose	This class is used to store data for VM2.
Attributes	
float temp_p	This is a temporary variable which stores the value of price, p for a drink.
float temp_v	This is a temporary variable which stores the value of coin inserted, v for a drink.

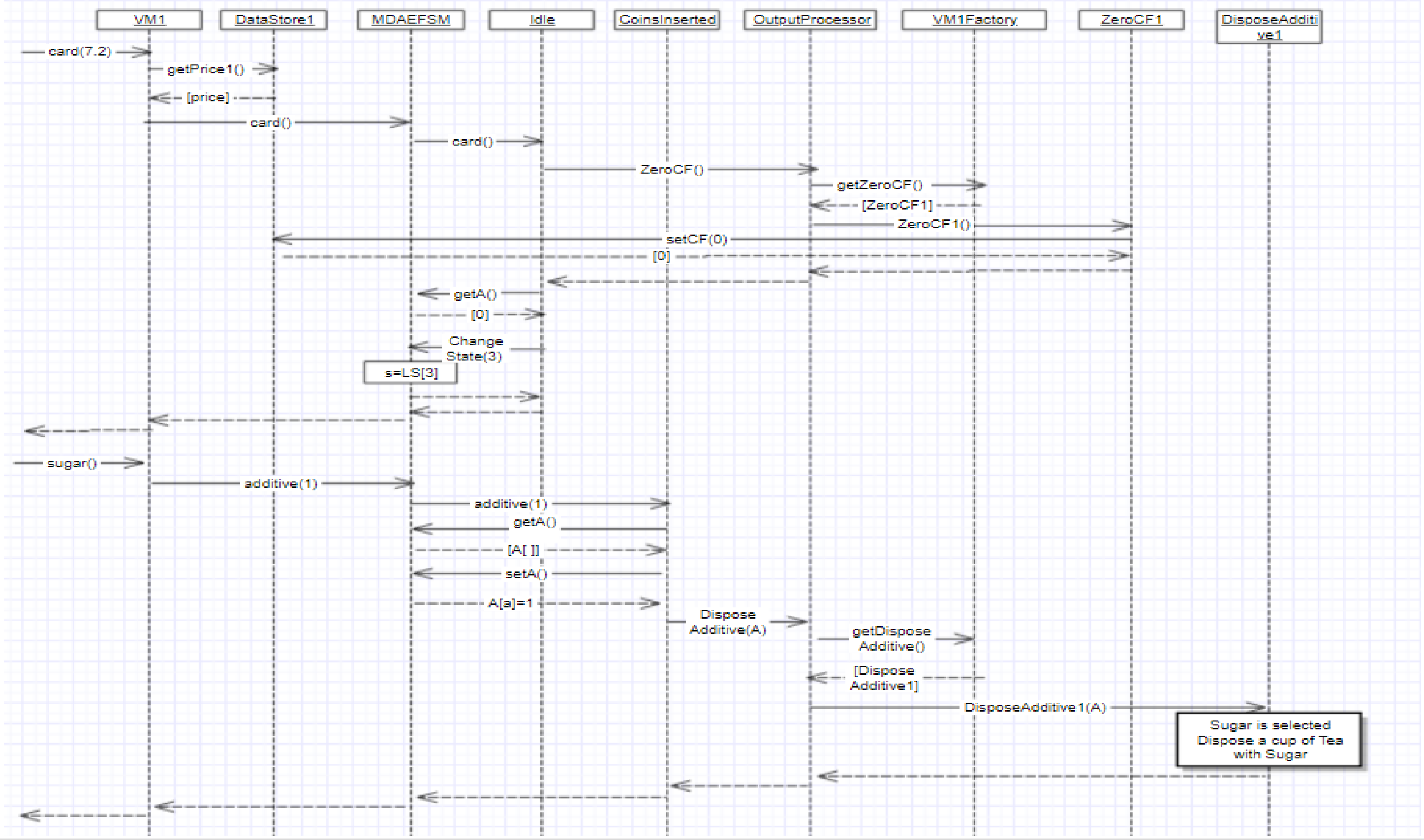
float price	This is a variable which stores the value of price for a drink.
float cf	This variable stores the cumulative fund inserted for a drink.
Operations	
getTemp_p()	This is a method to get the price value stored in temporary variable temp_p.
setTemp_p(float temp_p)	This is a method to set the value of temporary variable temp_p.
getTemp_v2()	This is a method to get the float coin value stored in temporary variable temp_v.
setTemp_v2(float temp_v)	This is a method to set the value of temporary variable temp_v.
getPrice2()	This is a method to get the float value of variable price.
setPrice2(float price)	This is a method to set the float value of variable price.
getCF2()	This is a method to get the float value of cumulative fund, cf inserted for a drink.
setCF2(float cf)	This is a method to set the value of cumulative fund, cf inserted for a drink.

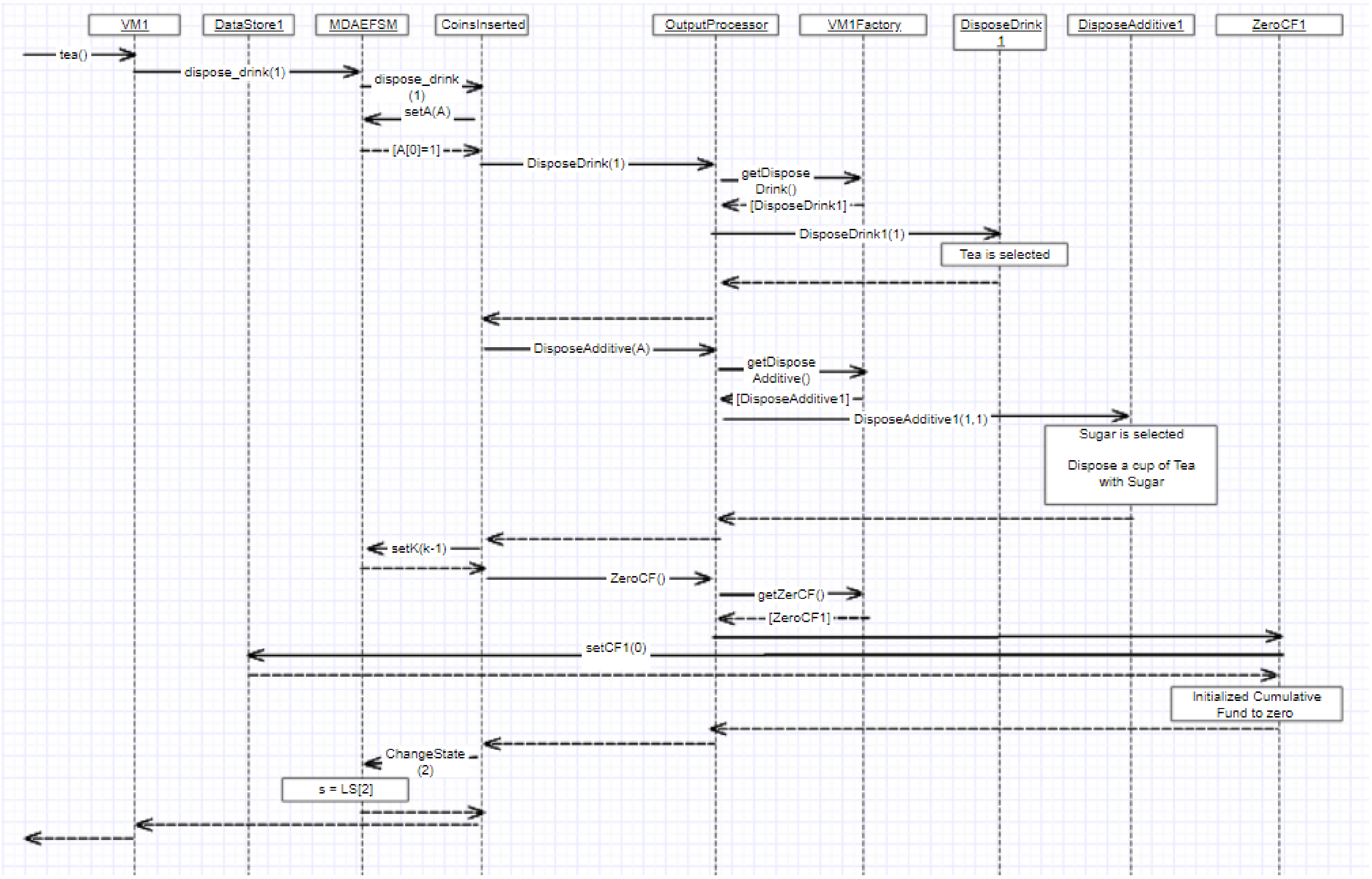
4. Provide two sequence diagrams for two Scenarios:

a. Scenario-I should show as to how the cup of tea is disposed in the Vending Machine VM-1 component, i.e., the following sequence of operations is issued:

create(2), insert_cups(20), card(7.2), sugar(), tea()

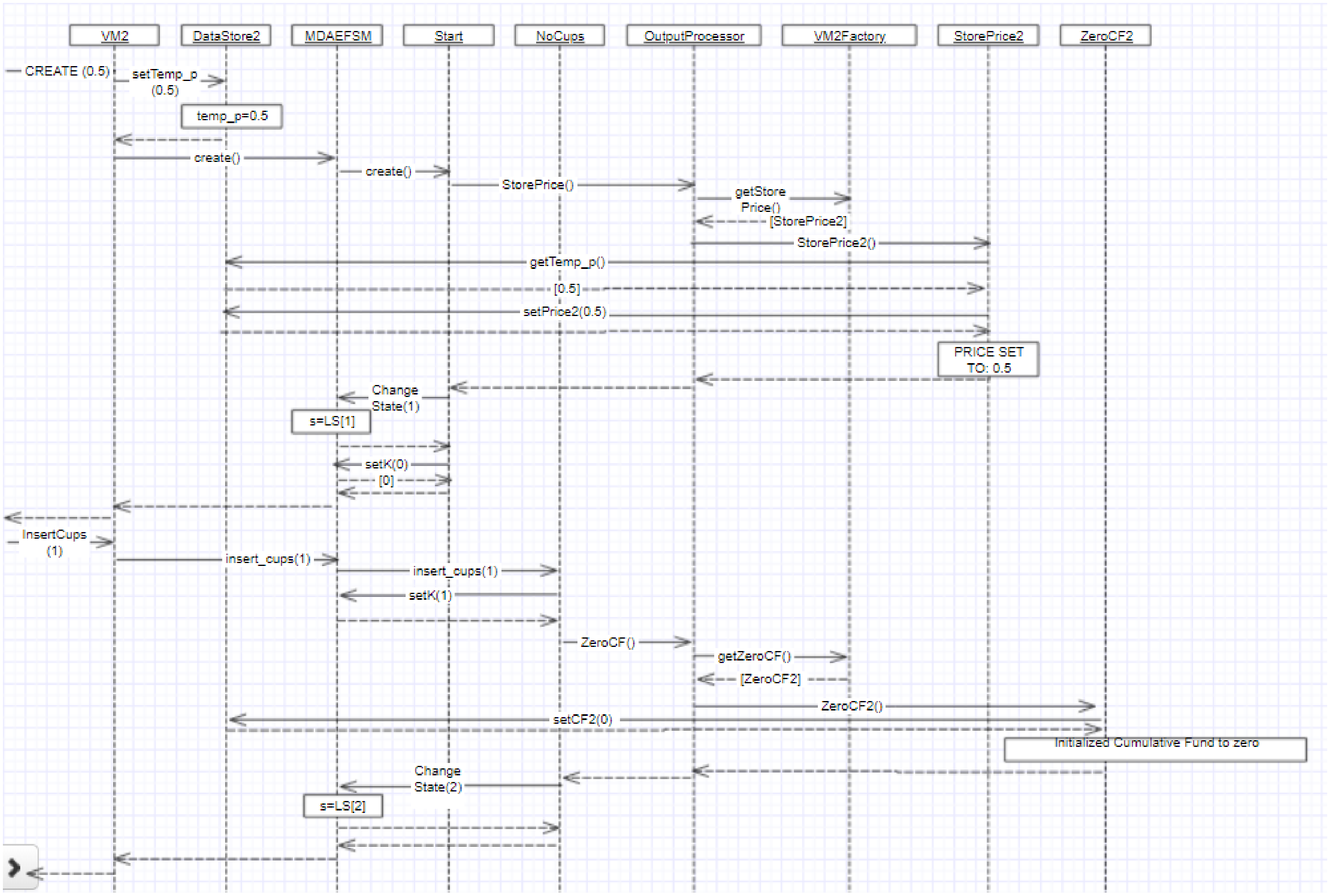

VM1
DataStore1
MDAEFSM
Start
NoCups
OutputProcessor
VM1Factory
StorePrice1
ZeroCF1
create(2)
setTemp_p (2)
temp_p=2
create()
create()
StorePrice()
getStore Price()
[StorePrice1]
StorePrice1()
getTemp_p()
[2]
setPrice1(2)
PRICE SET TO: 2
Change State(1)
s=LS[1]
setK(0)
[0]
insert_cups (20)
insert_cups(20)
insert_cups(20)
setK(20)
[20]
ZeroCF()
getZeroCF()
[ZeroCF1]
ZeroCF1()
setCF1(0)
[0]
Change State(2)
s=LS[2]

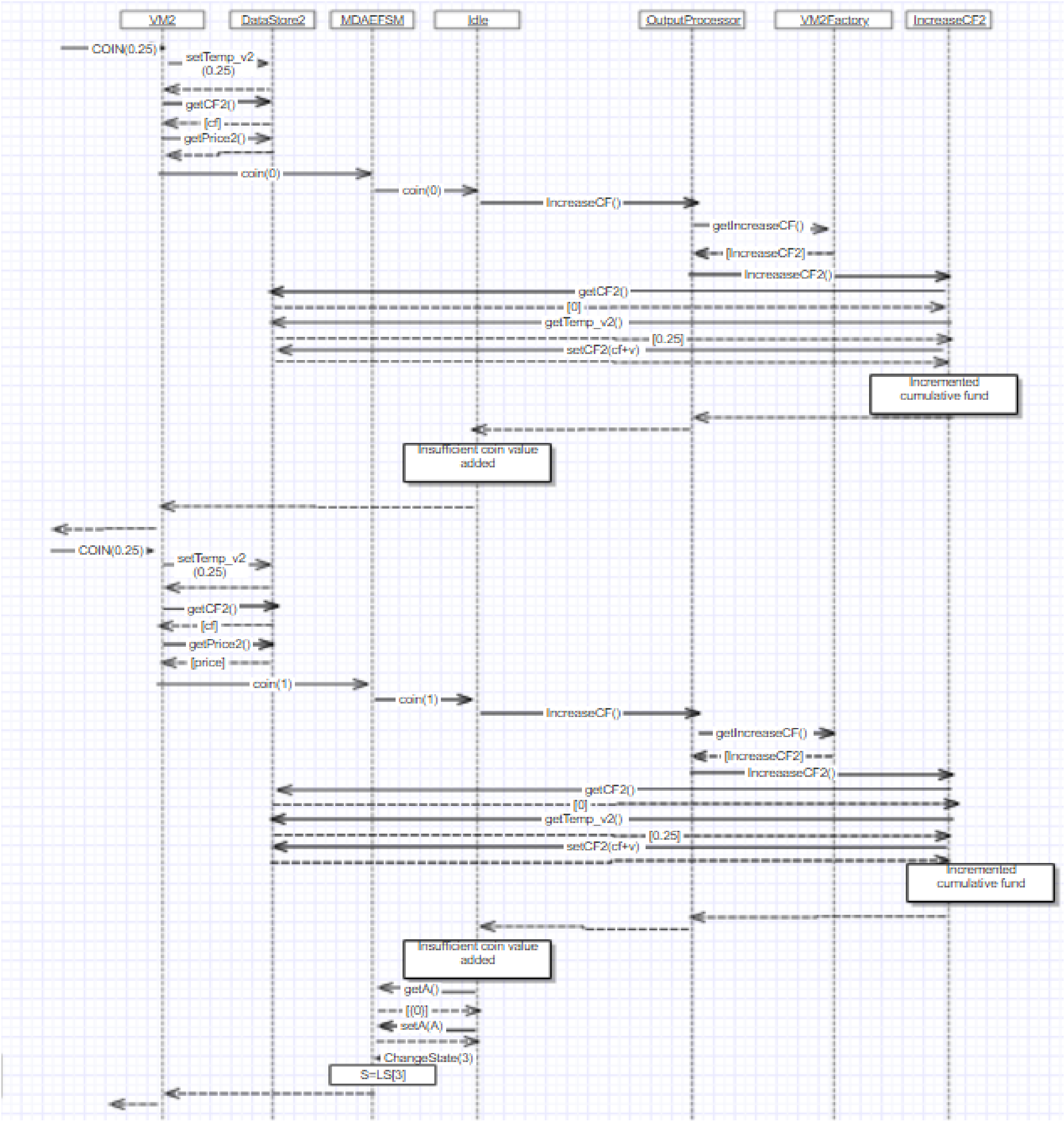

VM1
DataStore1
MDAEFSM
Idle
CoinsInserted
OutputProcessor
VM1Factory
ZeroCF1
DisposeAdditive1
card(7.2)
getPrice1()
[price]
card()
card()
ZeroCF()
getZeroCF()
[ZeroCF1]
ZeroCF1()
setCF(0)
[0]
getA()
[0]
Change State(3)
s=LS[3]
sugar()
additive(1)
additive(1)
getA()
[A[ ]]
setA()
A[a]=1
Dispose Additive(A)
getDispose Additive()
[Dispose Additive1]
DisposeAdditive1(A)
Sugar is selected Dispose a cup of Tea with Sugar

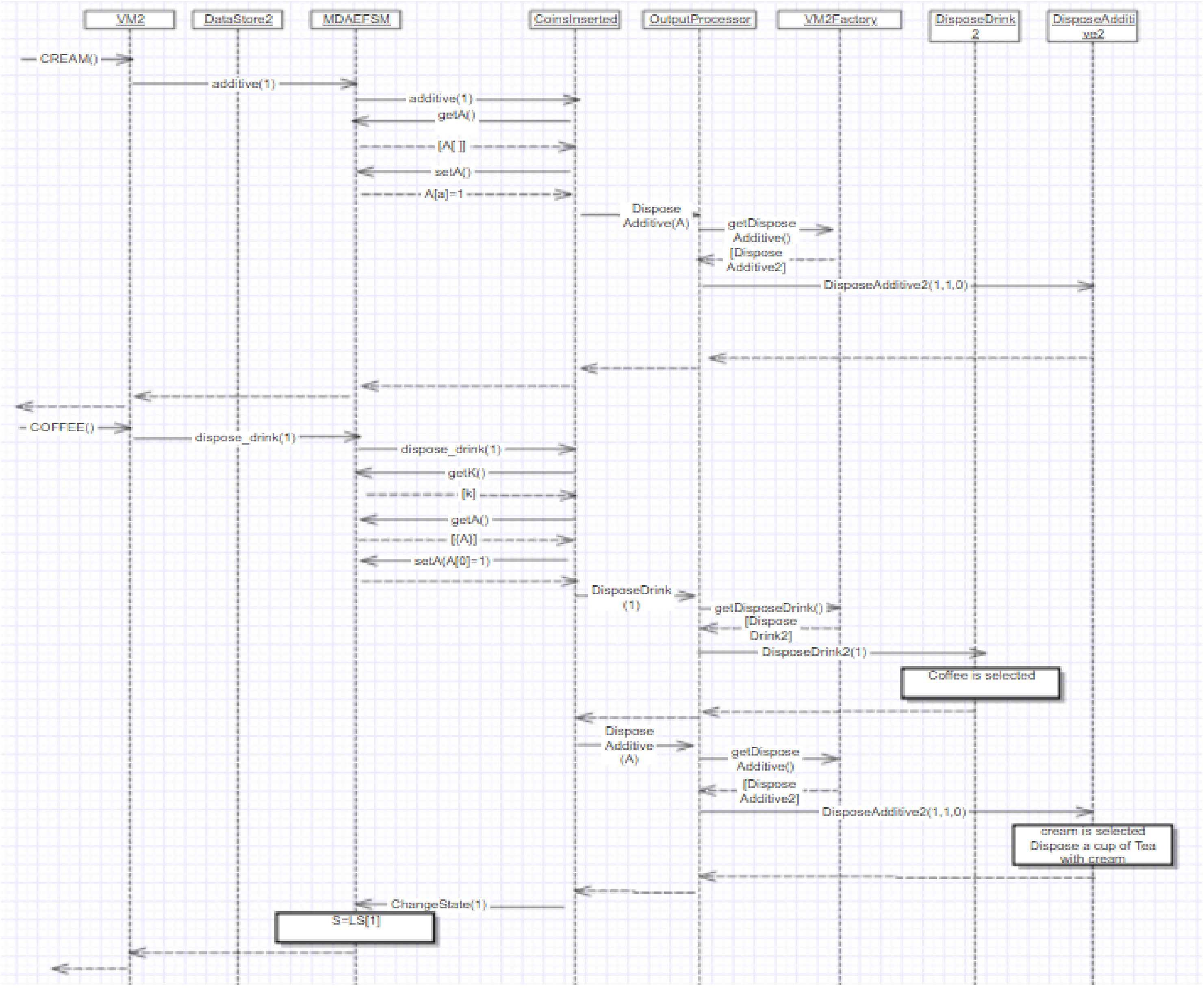


b. Scenario-II should show as to how a cup of coffee is disposed in the Vending Machine VM-2 component, i.e., the following sequence of operations is issued:

CREATE(0.5), InsertCups(1), COIN(0.25), COIN(0.25), CREAM(), COFFEE()


VM2
DataStore2
MDAEFSM
Start
NoCups
OutputProcessor
VM2Factory
StorePrice2
ZeroCF2
CREATE (0.5)
setTemp_p (0.5)
temp_p=0.5
create()
create()
StorePrice()
getStore Price()
[StorePrice2]
StorePrice2()
getTemp_p()
[0.5]
setPrice2(0.5)
PRICE SET TO: 0.5
Change State(1)
s=LS[1]
setK(0)
[0]
InsertCups (1)
insert_cups(1)
insert_cups(1)
setK(1)
ZeroCF()
getZeroCF()
[ZeroCF2]
ZeroCF2()
setCF2(0)
Initialized Cumulative Fund to zero
Change State(2)
s=LS[2]


VM2
DataStore2
MDAEFSM
Idle
OutputProcessor
VM2Factory
IncreaseCF2
COIN(0.25)
setTemp_v2
(0.25)
getCF2()
[cf]
getPrice2()
coin(0)
coin(0)
IncreaseCF()
getIncreaseCF()
[IncreaseCF2]
IncreaseCF2()
getCF2()
[0]
getTemp_v2()
[0.25]
setCF2(cf+v)
Incremented
cumulative fund
Insufficient coin value
added
COIN(0.25)
setTemp_v2
(0.25)
getCF2()
[cf]
getPrice2()
[price]
coin(1)
coin(1)
IncreaseCF()
getIncreaseCF()
[IncreaseCF2]
IncreaseCF2()
getCF2()
[0]
getTemp_v2()
[0.25]
setCF2(cf+v)
Incremented
cumulative fund
Insufficient coin value
added
getA()
[(0)]
setA(A)
ChangeState(3)
S=LS[3]


VM2
DataStore2
MDAEFSM
CoinsInserted
OutputProcessor
VM2Factory
DisposeDrink 2
DisposeAdditive2
CREAM()
additive(1)
additive(1)
getA()
[A[ ]]
setA()
A[a]=1
Dispose Additive(A)
getDispose Additive()
[Dispose Additive2]
DisposeAdditive2(1,1,0)
COFFEE()
dispose_drink(1)
dispose_drink(1)
getK()
[k]
getA()
[(A)]
setA(A[0]=1)
DisposeDrink (1)
getDisposeDrink()
[Dispose Drink2]
DisposeDrink2(1)
Coffee is selected
Dispose Additive (A)
getDispose Additive()
[Dispose Additive2]
DisposeAdditive2(1,1,0)
cream is selected
Dispose a cup of Tea
with cream
ChangeState(1)
S=LS[1]