Project Phase 2

1. MDA-EFSM model for the Vending Machine components

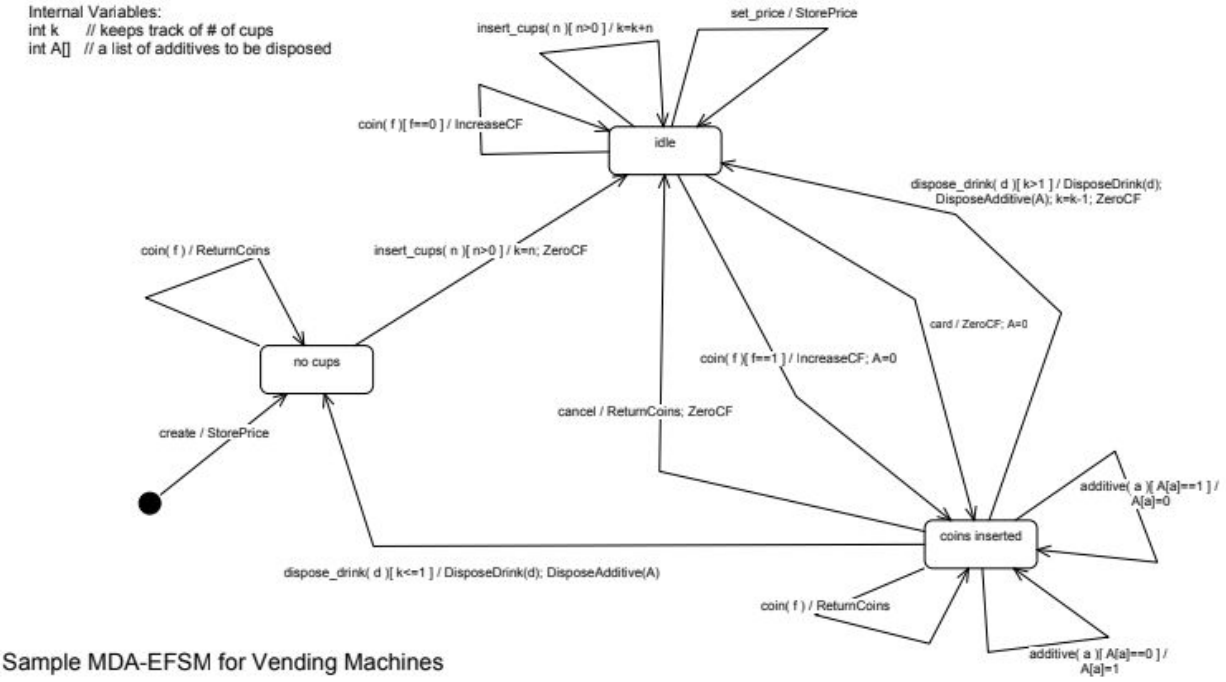
a. A list of meta events for the MDA-EFSM

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. A list of meta actions for the MDA-EFSM with their descriptions

1. **StorePrice()**
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. A **state diagram** of the MDA-EFSM



Sample MDA-EFSM for Vending Machines

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

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

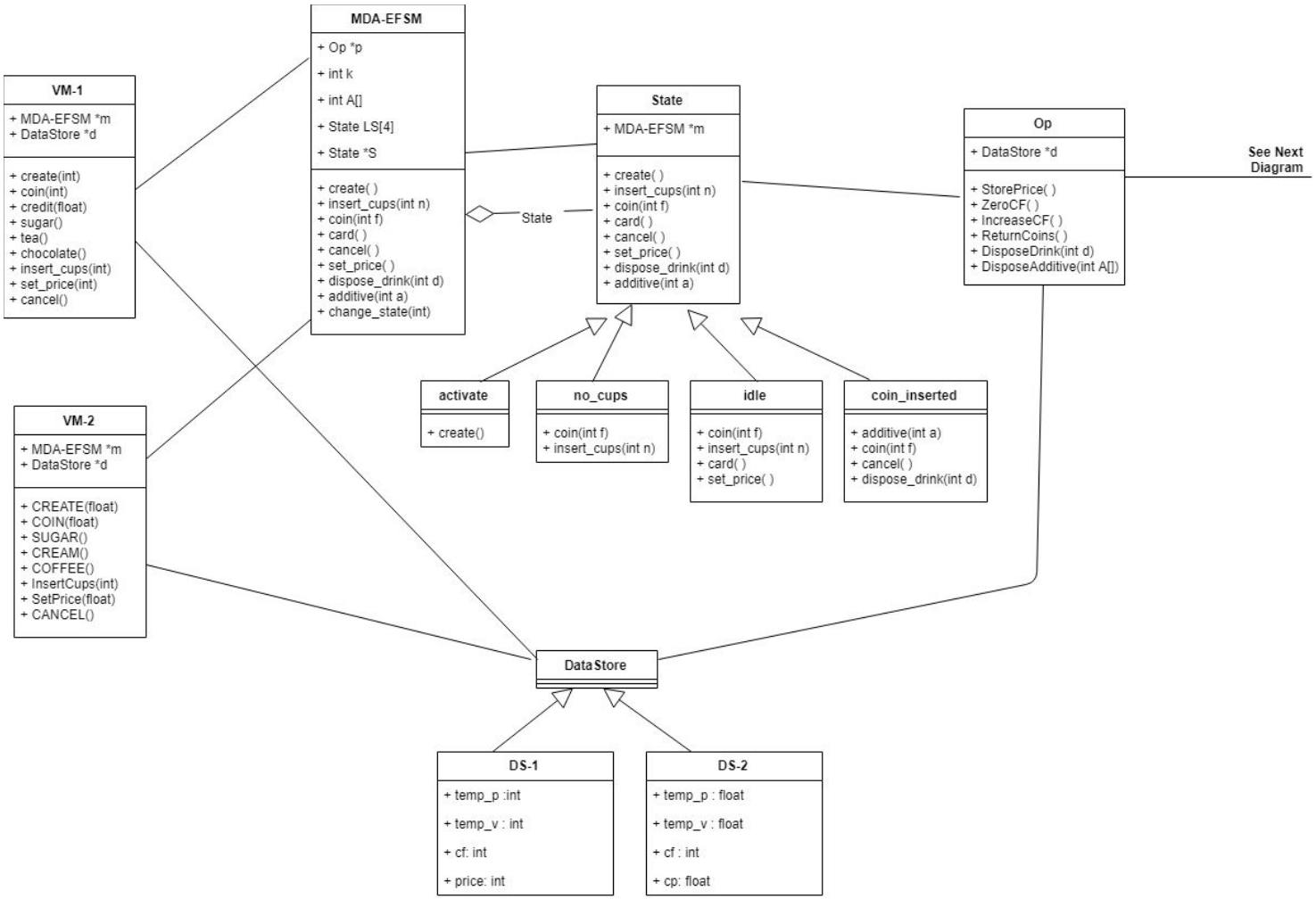
```
create(int p) {
        d->temp_p=p;
        m->create();
}
coin(int v) {
        d->temp_v=v;
        if (d->cf+v>=d->price) m->coin(1);
        else m->coin(0);
}
card(float x) {
        if (x>=d->price) m->card();
}
sugar() {
        m->additive(1);
}
tea() {
        m->dispose_drink(1);
}
chocolate()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

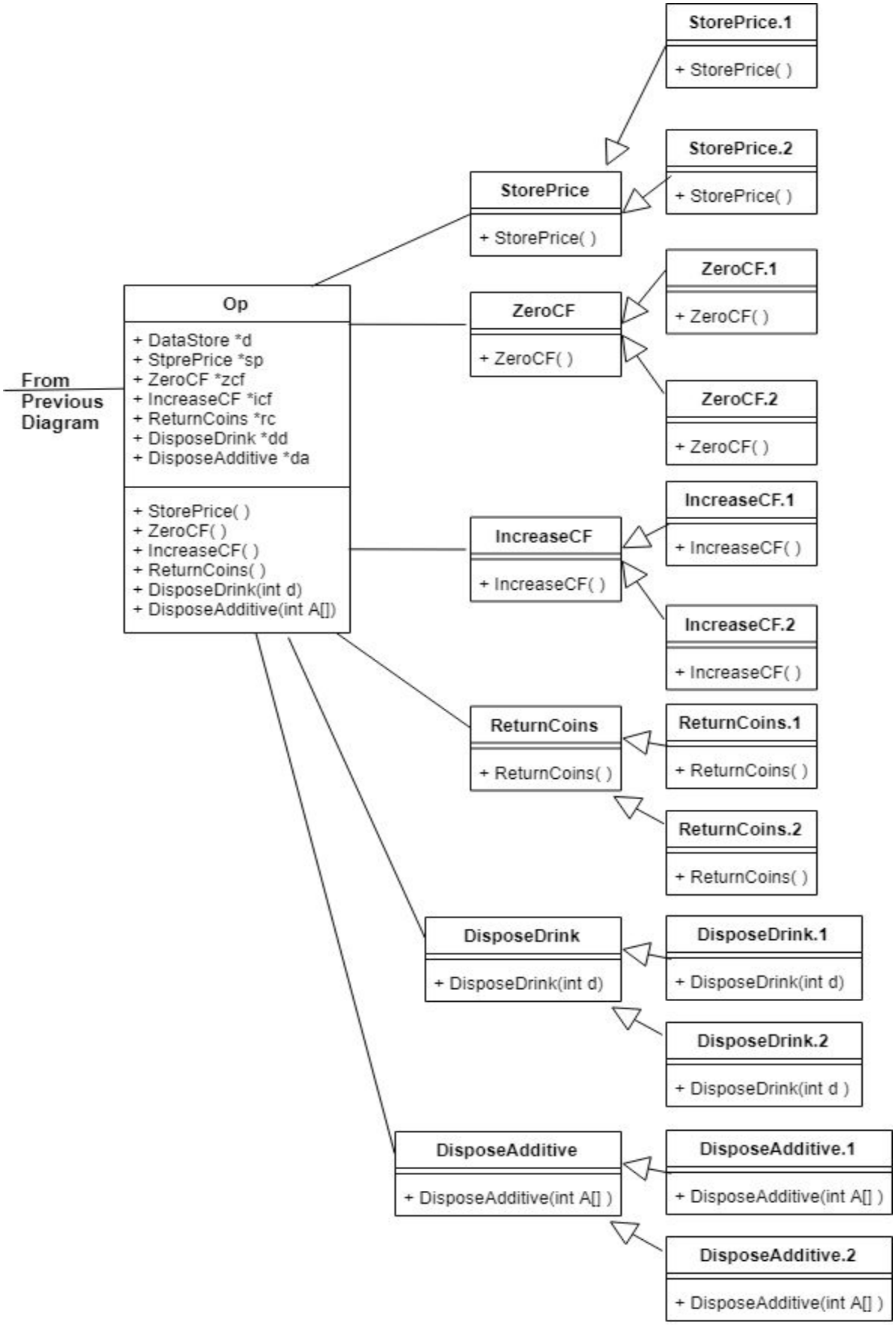
Vending-Machine-2

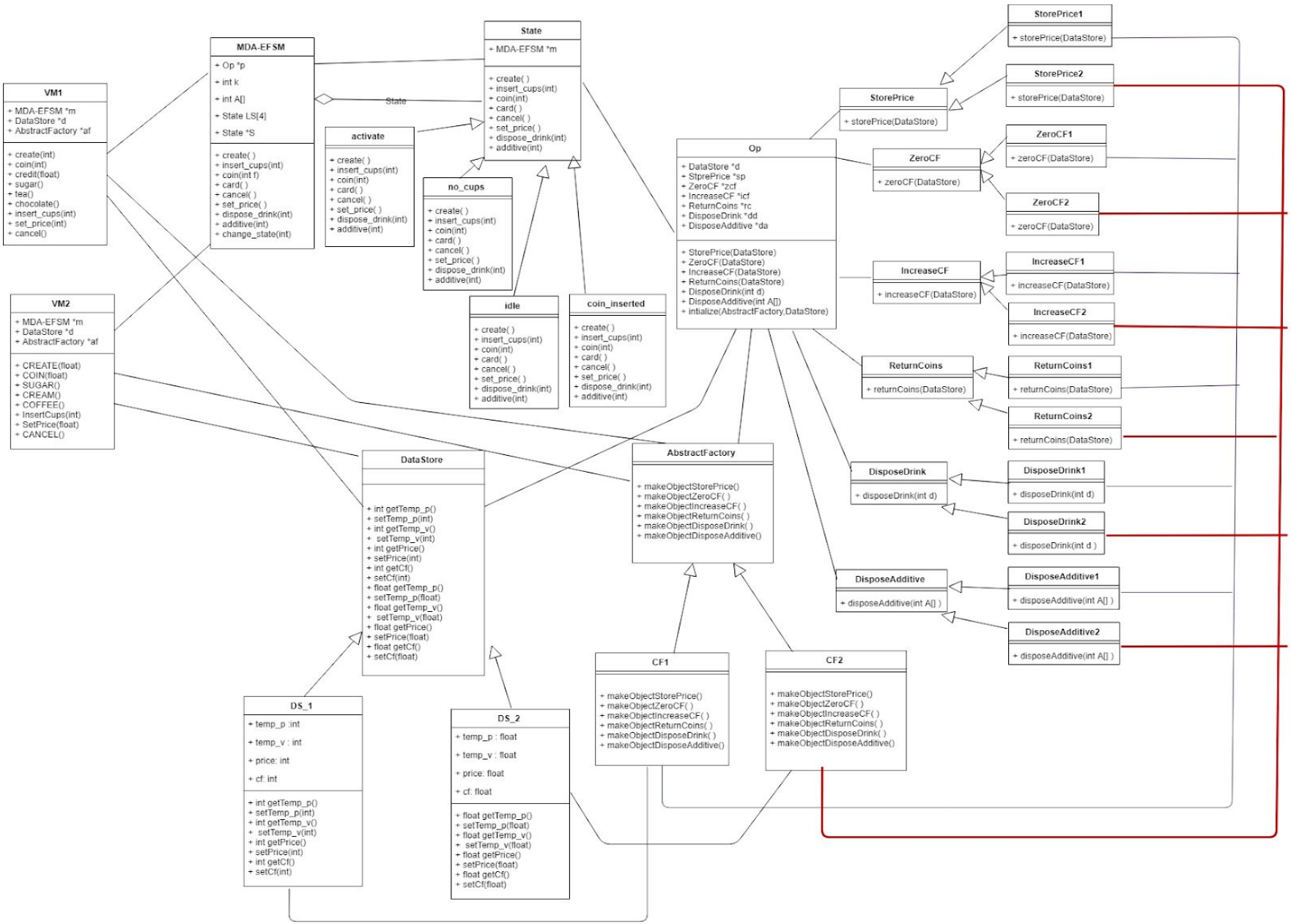
Where,
m: pointer to the MDA-EFSM
d: pointer to the data store DS-2
In the data store:
cf: represents a cumulative fund
price: represents a price for a drink

```
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();
}
```

2. Class diagram(s) of the MDA of the Vending Machine components. In your design, you MUST use the following OO design patterns:


MDA-EFSM
+ Op *p
+ int k
+ int A[]
+ State LS[4]
+ State *S
+ create( )
+ insert_cups(int n)
+ coin(int f)
+ card( )
+ cancel( )
+ set_price( )
+ dispose_drink(int d)
+ additive(int a)
+ change_state(int)
VM-1
+ MDA-EFSM *m
+ DataStore *d
+ create(int)
+ coin(int)
+ credit(float)
+ sugar()
+ tea()
+ chocolate()
+ insert_cups(int)
+ set_price(int)
+ cancel()
State
State
+ MDA-EFSM *m
+ create( )
+ insert_cups(int n)
+ coin(int f)
+ card( )
+ cancel( )
+ set_price( )
+ dispose_drink(int d)
+ additive(int a)
Op
+ DataStore *d
+ StorePrice( )
+ ZeroCF( )
+ IncreaseCF( )
+ ReturnCoins( )
+ DisposeDrink(int d)
+ DisposeAdditive(int A[])
See Next Diagram
activate
+ create()
no_cups
+ coin(int f)
+ insert_cups(int n)
idle
+ coin(int f)
+ insert_cups(int n)
+ card( )
+ set_price( )
coin_inserted
+ additive(int a)
+ coin(int f)
+ cancel( )
+ dispose_drink(int d)
VM-2
+ MDA-EFSM *m
+ DataStore *d
+ CREATE(float)
+ COIN(float)
+ SUGAR()
+ CREAM()
+ COFFEE()
+ InsertCups(int)
+ SetPrice(float)
+ CANCEL()
DataStore
DS-1
+ temp_p :int
+ temp_v : int
+ cf: int
+ price: int
DS-2
+ temp_p : float
+ temp_v : float
+ cf : int
+ cp: float


From Previous Diagram
Op
+ DataStore *d
+ StprePrice *sp
+ ZeroCF *zcf
+ IncreaseCF *icf
+ ReturnCoins *rc
+ DisposeDrink *dd
+ DisposeAdditive *da
+ StorePrice( )
+ ZeroCF( )
+ IncreaseCF( )
+ ReturnCoins( )
+ DisposeDrink(int d)
+ DisposeAdditive(int A[])
StorePrice
+ StorePrice( )
StorePrice.1
+ StorePrice( )
StorePrice.2
+ StorePrice( )
ZeroCF
+ ZeroCF( )
ZeroCF.1
+ ZeroCF( )
ZeroCF.2
+ ZeroCF( )
IncreaseCF
+ IncreaseCF( )
IncreaseCF.1
+ IncreaseCF( )
IncreaseCF.2
+ IncreaseCF( )
ReturnCoins
+ ReturnCoins( )
ReturnCoins.1
+ ReturnCoins( )
ReturnCoins.2
+ ReturnCoins( )
DisposeDrink
+ DisposeDrink(int d)
DisposeDrink.1
+ DisposeDrink(int d)
DisposeDrink.2
+ DisposeDrink(int d)
DisposeAdditive
+ DisposeAdditive(int A[])
DisposeAdditive.1
+ DisposeAdditive(int A[])
DisposeAdditive.2
+ DisposeAdditive(int A[])


VM1
+ MDA-EFSM *m
+ DataStore *d
+ AbstractFactory *af
+ create(int)
+ coin(int)
+ credit(float)
+ sugar()
+ tea()
+ chocolate()
+ insert_cups(int)
+ set_price(int)
+ cancel()
VM2
+ MDA-EFSM *m
+ DataStore *d
+ AbstractFactory *af
+ CREATE(float)
+ COIN(float)
+ SUGAR()
+ CREAM()
+ COFFEE()
+ InsertCups(int)
+ SetPrice(float)
+ CANCEL()
MDA-EFSM
+ Op *p
+ int k
+ int A[]
+ State LS[4]
+ State *S
+ create( )
+ insert_cups(int)
+ coin(int f)
+ card( )
+ cancel( )
+ set_price( )
+ dispose_drink(int)
+ additive(int)
+ change_state(int)
State
activate
+ create( )
+ insert_cups(int)
+ coin(int)
+ card( )
+ cancel( )
+ set_price( )
+ dispose_drink(int)
+ additive(int)
State
+ MDA-EFSM *m
+ create( )
+ insert_cups(int)
+ coin(int)
+ card( )
+ cancel( )
+ set_price( )
+ dispose_drink(int)
+ additive(int)
no_cups
+ create( )
+ insert_cups(int)
+ coin(int)
+ card( )
+ cancel( )
+ set_price( )
+ dispose_drink(int)
+ additive(int)
idle
+ create( )
+ insert_cups(int)
+ coin(int)
+ card( )
+ cancel( )
+ set_price( )
+ dispose_drink(int)
+ additive(int)
coin_inserted
+ create( )
+ insert_cups(int)
+ coin(int)
+ card( )
+ cancel( )
+ set_price( )
+ dispose_drink(int)
+ additive(int)
Op
+ DataStore *d
+ StprePrice *sp
+ ZeroCF *zcf
+ IncreaseCF *icf
+ ReturnCoins *rc
+ DisposeDrink *dd
+ DisposeAdditive *da
+ StorePrice(DataStore)
+ ZeroCF(DataStore)
+ IncreaseCF(DataStore)
+ ReturnCoins(DataStore)
+ DisposeDrink(int d)
+ DisposeAdditive(int A[])
+ intialize(AbstractFactory,DataStore)
StorePrice
+ storePrice(DataStore)
StorePrice1
+ storePrice(DataStore)
StorePrice2
+ storePrice(DataStore)
ZeroCF
+ zeroCF(DataStore)
ZeroCF1
+ zeroCF(DataStore)
ZeroCF2
+ zeroCF(DataStore)
IncreaseCF
+ increaseCF(DataStore)
IncreaseCF1
+ increaseCF(DataStore)
IncreaseCF2
+ increaseCF(DataStore)
ReturnCoins
+ returnCoins(DataStore)
ReturnCoins1
+ returnCoins(DataStore)
ReturnCoins2
+ returnCoins(DataStore)
DisposeDrink
+ disposeDrink(int d)
DisposeDrink1
+ disposeDrink(int d)
DisposeDrink2
+ disposeDrink(int d)
DisposeAdditive
+ disposeAdditive(int A[])
DisposeAdditive1
+ disposeAdditive(int A[])
DisposeAdditive2
+ disposeAdditive(int A[])
DataStore
+ int getTemp_p()
+ setTemp_p(int)
+ int getTemp_v()
+ setTemp_v(int)
+ int getPrice()
+ setPrice(int)
+ int getCf()
+ setCf(int)
+ float getTemp_p()
+ setTemp_p(float)
+ float getTemp_v()
+ setTemp_v(float)
+ float getPrice()
+ setPrice(float)
+ float getCf()
+ setCf(float)
AbstractFactory
+ makeObjectStorePrice()
+ makeObjectZeroCF( )
+ makeObjectIncreaseCF( )
+ makeObjectReturnCoins( )
+ makeObjectDisposeDrink( )
+ makeObjectDisposeAdditive()
DS_1
+ temp_p :int
+ temp_v : int
+ price: int
+ cf: int
+ int getTemp_p()
+ setTemp_p(int)
+ int getTemp_v()
+ setTemp_v(int)
+ int getPrice()
+ setPrice(int)
+ int getCf()
+ setCf(int)
DS_2
+ temp_p : float
+ temp_v : float
+ price: float
+ cf: float
+ float getTemp_p()
+ setTemp_p(float)
+ float getTemp_v()
+ setTemp_v(float)
+ float getPrice()
+ setPrice(float)
+ float getCf()
+ setCf(float)
CF1
+ makeObjectStorePrice()
+ makeObjectZeroCF( )
+ makeObjectIncreaseCF( )
+ makeObjectReturnCoins( )
+ makeObjectDisposeDrink( )
+ makeObjectDisposeAdditive()
CF2
+ makeObjectStorePrice()
+ makeObjectZeroCF( )
+ makeObjectIncreaseCF( )
+ makeObjectReturnCoins( )
+ makeObjectDisposeDrink( )
+ makeObjectDisposeAdditive()

3. For each class in the class diagram(s) you should:

a. Describe the purpose of the class, i.e., responsibilities

VM1:

This class will perform all the operations that should be done by vending machine 1
public void create(int p) : This will create the Virtual Machine with the value of p as beverage value
public void coin(int v) : This will check funds for the drink. If the funds are sufficient, it will send 1 otherwise zero
public void sugar(): here additive value 1 is sugar is selected
public void tea() : here drink type 1 is tea is selected
public void chocolate() : here drink type 2 is chocolate is selected
public void insert_cups(int n): adding cup to system, this will call insert_cups for MDA_EFSM
public void set_price(int price): setting new price for drink, the new price will be assigned to temp_p and Store Price will be called
public void cancel(): cancel the transaction

VM2:

This class will perform all the operations that should be done by vending machine 2
public void CREATE(float p) : this method will This will create the Vm machine with the value of p as beverage value
public void COIN(float v): his method will This will check funds for the drink. If the funds are sufficient, it will this method will send 1 otherwise zero
public void SUGAR() this method will set additive value to 1 as it is denoted by sugar
public void CREAM() : this method will set additive value to 2 as it is denoted by cream
public void COFFEE() this method will set drink value to 3 as it is denoted by coffee
public void InsertCups(int n) this method will adding n cups to vending machine
public void SetPrice(float price) this method sets the new price for the drink, the new price will be assigned to temp_p and Store Price will be called
public void CANCEL(): cancel the transaction

MDA_EFSM:

This is a MDA_EFSM class
This class have 5 variables
Op p : p pointer is the pointer class to Output Processor OP
int k: k represents the number of cups for that instance of Vending Machine
int A[]: Array to store value for different kind of additives

A[1] = 1 if sugar is selected, 0 if not selected
A[2] = 1 if cream is selected, 0 if not selected
A[3] = 1 if sugar-cream is selected, 0 if not selected

State LS[] : Array to represent all the State objects Available for the system
in this project
LS[0] = object of active state class
LS[1] = object of no_cups state class
LS[2] = object of idle state class
LS[3] = object of coin_inserted class
State S : pointer to the current state class

this class will be having meta events as methods of the class which
will be calling corresponding state classes and that state classes will call meta action in OP.

this class is connected to VM-1, VM-2 and State class

public void create() to call create method from S object
public void insert_cups(int n) calls insert_cups method for S object, takes n as number of cups
public void coin(int f) calls coin method of state class, takes f as 1 to show the fund is sufficient or 0 that shows non sufficient fund
public void card() calls card method of state class
public void cancel(): cancel the transaction
public void set_price() calls set_price method of corresponding state class
public void dispose_drink(int d) to call dispose_drink method of State class with argument as corresponding drink_type value
public void additive(int a) to call additive method of State class with argument as corresponding additive_type chosen
public void change_state(int a) this method will set the state pointer S to a value, a values will be 0-activate , 1-no_cups, 2-idle, 3-coin_inserted
public void initialize(AbstractFactory af, DataStore d)this method is called at the point of initialization of vending machines, this method will call the initialize method in OP class to initialize,the abstract factory that can get the needed output objects to complete the execution this methods takes object of abstract factory and data store which was passed as per the version of vending machine
public MDA_EFSM getObject() this method will return current MDA_EFSM object

AbstractFactory:

This class is an abstract factory class that will help to generate
required objects for different types of vending machine
All methods here are abstract
This class is connected to VM-1, VM-2 and OP
the child classes of this class are connected to Data Store child class as well

public abstract StorePrice makeObjectStorePrice();
public abstract ZeroCF makeObjectZeroCF();
public abstract IncreaseCF makeObjectIncreaseCF();
public abstract ReturnCoins makeObjectReturnCoins();
public abstract DisposeDrink makeObjectDisposeDrink();
public abstract DisposeAdditive makeObjectDisposeAdditive();

CF1:

Concrete Factory 1
Child of Abstract Factory
This will create required Output Class objects to use Vending Machine 1

public StorePrice makeObjectStorePrice() Get object of store price class 1 for vending machine 1
public ZeroCF makeObjectZeroCF() Get object of zero cumulative factor price class 1 for vending machine 1
public IncreaseCF makeObjectIncreaseCF() Get object of increase cumulative factor price class 1 for vending machine 1
public ReturnCoins makeObjectReturnCoins() Get object of return coins 1 for vending machine 1
public DisposeDrink makeObjectDisposeDrink() Get object of dispose drink class 1 for vending machine 1
public DisposeAdditive makeObjectDisposeAdditive() Get object of dispose Additive class 1 for vending machine 1

CF2:

Concrete Factory 2
Child of Abstract Factory
This will create required Output Class objects to use Vending Machine 2

public StorePrice makeObjectStorePrice() Get object of store price class 2 for vending machine 2
public ZeroCF makeObjectZeroCF() Get object of zero cumulative factor price class 2 for vending machine 2

public IncreaseCF makeObjectIncreaseCF() Get object of increase cumulative factor price class 2 for vending machine 2
public ReturnCoins makeObjectReturnCoins() Get object of return coins 2 for vending machine 2
public DisposeDrink makeObjectDisposeDrink() Get object of dispose drink class 2 for vending machine 2
public DisposeAdditive makeObjectDisposeAdditive() Get object of dispose Additive class 2 for vending machine 2

DataStore:

DataStore class is used to store data
This is a parent DataStore class
This will have list of all methods that are defined in both data stores
just the null definition. this will not have an impact
This class is connected to VM1, VM2 and OP
THe child DataStore Classes are connected to Child abstract factory classes according to the version

public int getTemp_p() this method is just parent definition for child class, it does nothing here but it will get value of temp_p variables, return type int
public void setTemp_p(int temp_p) this method is just parent definition for child class, it does nothing here but it will set value of temp_p variables, type int
public int getTemp_v() this method is just parent definition for child class, it does nothing here but it will get value of temp_v variables, return type int
public void setTemp_v(int temp_v) this method is just parent definition for child class, it does nothing here but it will set value of temp_v variables, type int
public int getPrice() this method is just parent definition for child class, it does nothing here but it will get value of price variables, return type int
public void setPrice(int price) this method is just parent definition for child class, it does nothing here but it will set value of price variables, type int
public int getCf() this method is just parent definition for child class, it does nothing here but it will get value of cf variables, return type int
public void setCf(int cf) this method is just parent definition for child class, it does nothing here but it will set value of price variables, type int
public float getfloatTemp_p() this method is just parent definition for child class, it does nothing here but it will get value of temp_p variables, return type float
public void setTemp_p(float temp_p) this method is just parent definition for child class, it does nothing here but it will set value of temp_p variables, type float
public float getfloatTemp_v() this method is just parent definition for child class, it does nothing here but it will get value of temp_v variables, return type float
public void setTemp_v(float temp_v) this method is just parent definition for child class, it does nothing here but it will set value of temp_v variables, type float

public float getfloatPrice() this method is just parent definition for child class, it does nothing here but it will get value of price variables, return type float
public void setPrice(float price) this method is just parent definition for child class, it does nothing here but it will set value of price variables, type float
public float getfloatCf() this method is just parent definition for child class, it does nothing here but it will set value of price variables, type float
public void setCf(float cf) this method is just parent definition for child class, it does nothing here but it will set value of price variables, type float

Data Store 1

The variable which are concerned to Vending Machine 1
This class contains data variable for VM 1 and getter setter method for each

public int getTemp_p() get value of temp_p variables, return type int
public void setTemp_p(int temp_p) set value of temp_p variables, type int
public int getTemp_v() get value of temp_v variables, return type int
public void setTemp_v(int temp_v) set value of temp_v variables, type int
public int getPrice() get value of price variables, return type int
public void setPrice(int price) set value of price variables, type int
public int getCf() get value of cf variables, return type int
public void setCf(int cf) set value of cf variables, type int

Data Store 2

The variable which are concerned to Vending Machine 2
This class contains data variable for VM 2 and getter setter method for each

public float getTemp_p() get value of temp_p variables, return type float
public void setTemp_p(float temp_p) set value of temp_p variables, type float
public float getTemp_v() get value of temp_v variables, return type float
public void setTemp_v(float temp_v) set value of temp_v variables, type float
public float getPrice() get value of price variables, return type float
public void setPrice(float price) set value of price variables, type float
public float getCf() get value of cf variables, return type float
public void setCf(float cf) set value of cf variables, type float

DisposeAdditive

DisposeAdditive Class
This class represents the meta action of Disposing Additive
This is a parent abstract class having one abstract method of disposing Additive
The child classes will be having different implementations or versions of this action

public abstract void disposeAdditive(int A[]); // abstract method having parameter of Additive value array

DisposeAdditive1 Class

* This class represents the meta action of Disposing Additive for Vending Machine 1
* This is a child class that will implement abstract methods of parent class having one abstract method of disposing Additive

public void disposeAdditive(int[] A)
method to dispose drink with additive, the argument is the additive array which have 3 values for each additive A[sugar = 1] = 0 if additive is not selected , 1 if additive is selected

DisposeAdditive2 Class

* This class represents the meta action of Disposing Additive for Vending Machine 2
* This is a child class that will implement abstract methods of parent class having one abstract method of disposing Additive

public void disposeAdditive(int[] A)
method to dispose drink with additive, the argument is the additive array which have 3 values for each additive A[sugar = 1] = 0 if additive is not selected , 1 if additive is selected
A[cream = 1] = 0 if additive is not selected , 1 if additive is selected
A[sugar-cream = 1] = 0 if additive is not selected , 1 if additive is selected

DisposeDrink

DisposeDrink Class
This class represents the meta action of Disposing Drink
This is a parent abstract class having one abstract method of disposing Additive
The child classes will be having different implementations or versions of this action
public abstract void disposeAdditive(int A[]); // abstract method having parameter of Additive value array

DisposeDrink1

DisposeDrink1 CLass
This class represents the meta action of Disposing drink for Vending Machine 1
This is a child class that will implement abstract methods of parent class having one abstract method of disposing drink

public void disposeDrink(int d)

method to dispose drink, the argument is the drink type which is an integer with possibility of 3 values for each drink
For vending machine 1

* drink_type or d = 1 , then drink is tea
* drink_type or d = 2 , then drink is chocolate

DisposeDrink2

DisposeDrink2 CLass
This class represents the meta action of Disposing drink for Vending Machine 2 This is a child class that will implement abstract methods of parent class having one abstract method of disposing drink

public void disposeDrink(int d)

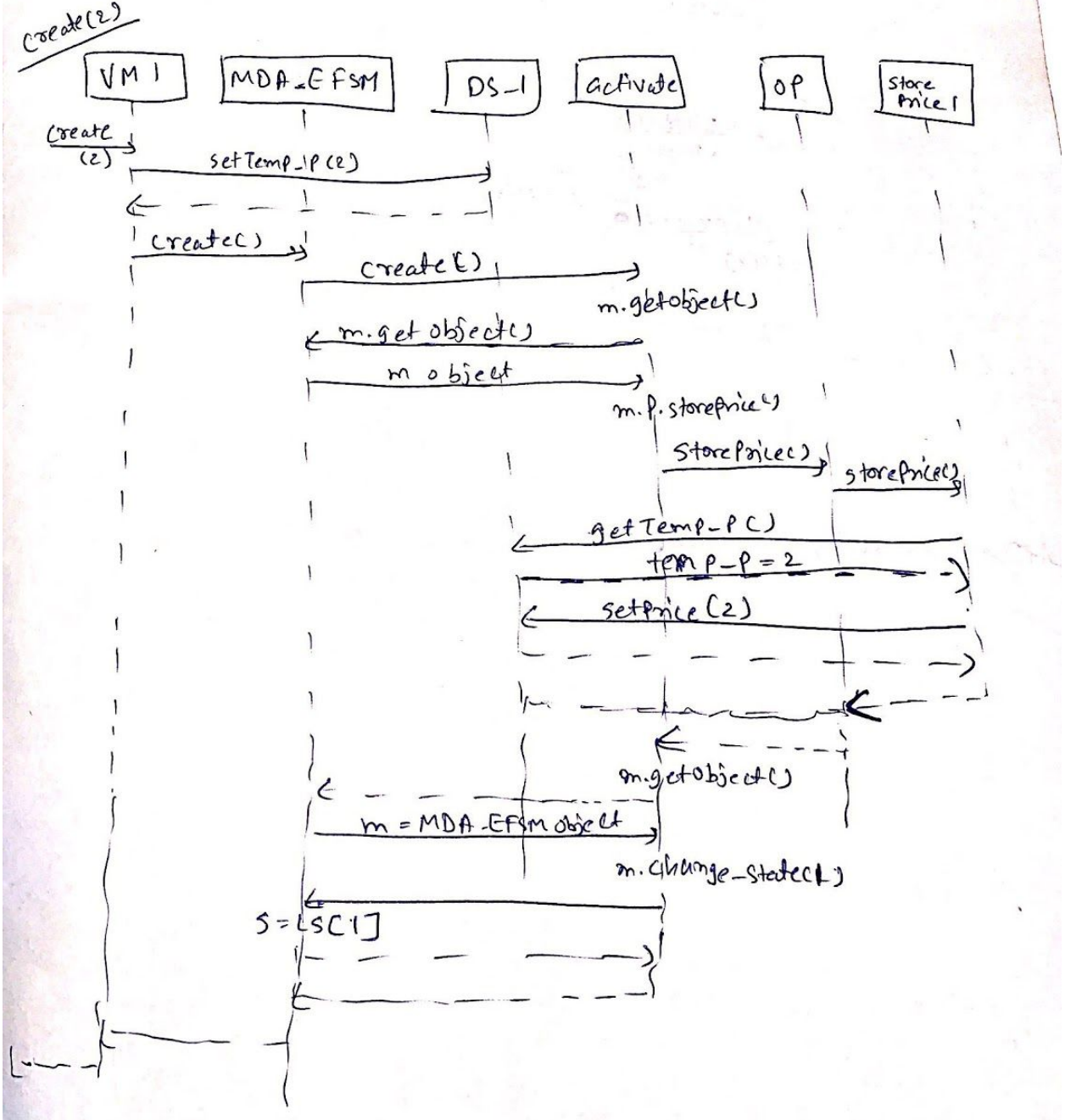
method to dispose drink, the argument is the drink type which is an integer with possibility of 3 values for each drink
For vending machine 2

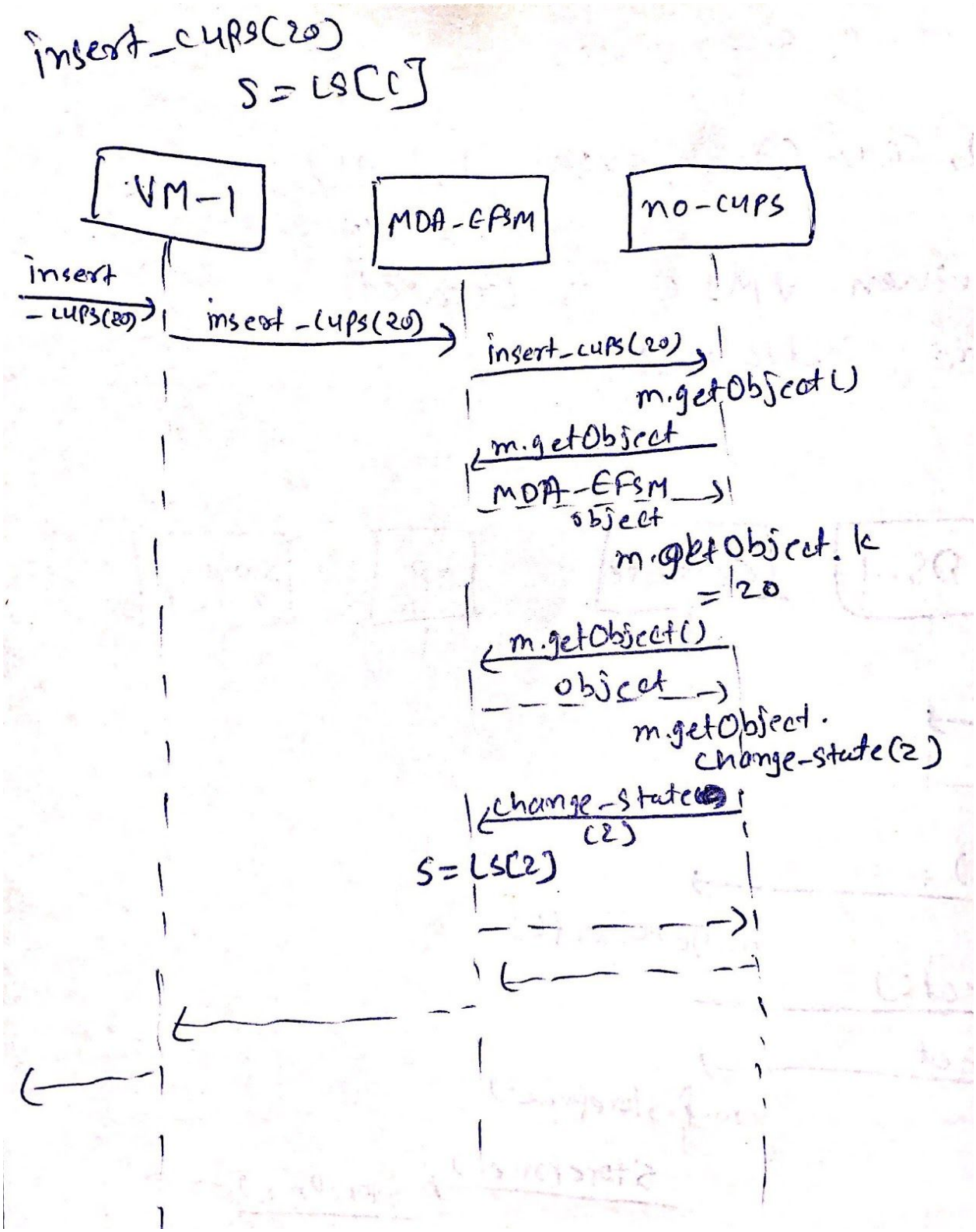
* drink_type or d = 3 , then drink is coffee

Sequence diagram for two scenarios

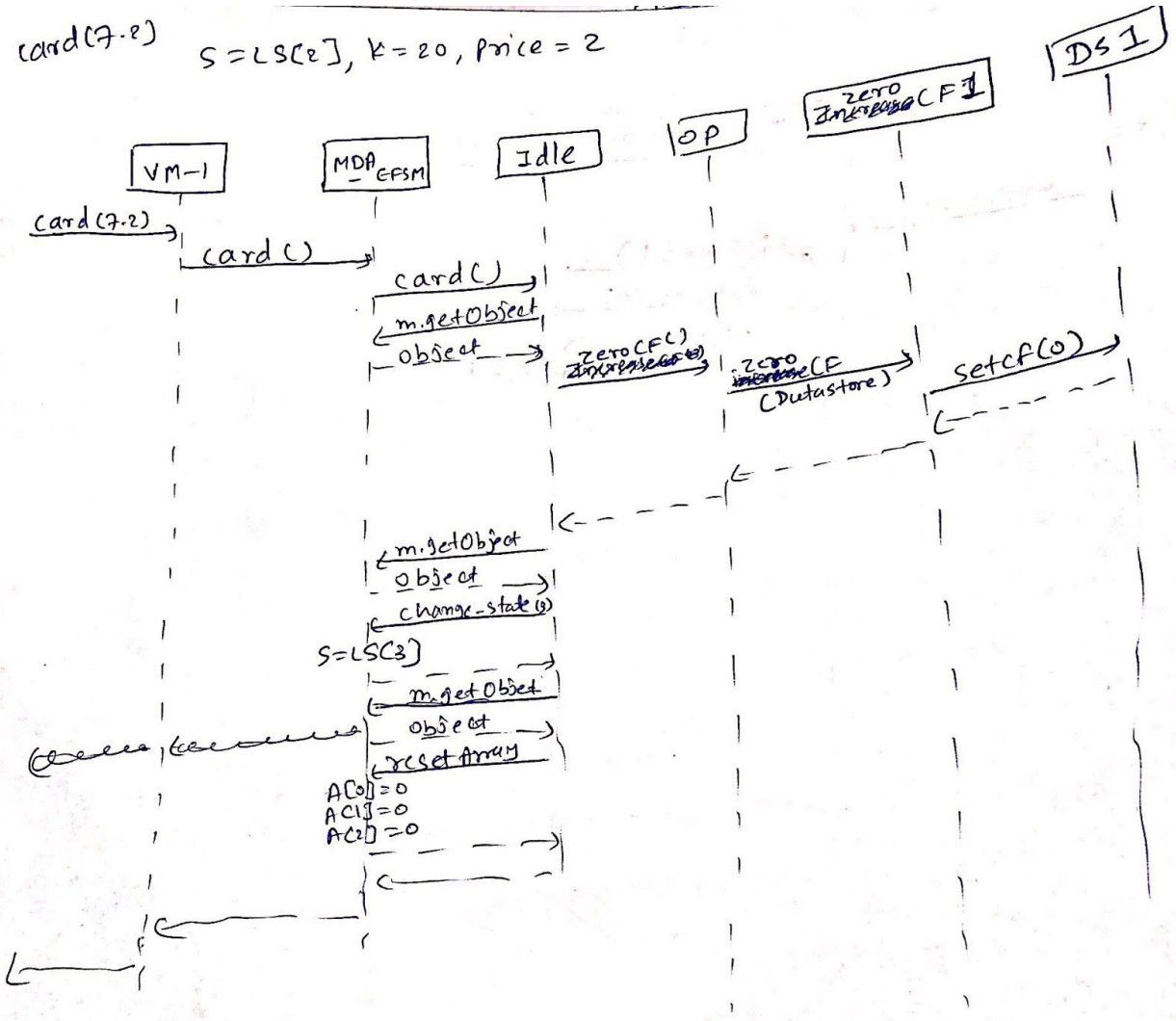
(i) create(2), insert_cups(20), card(7.2), sugar(1), tea()

→ S=LS[0] is initialized when VM1() is created.
assume all objects are initalized

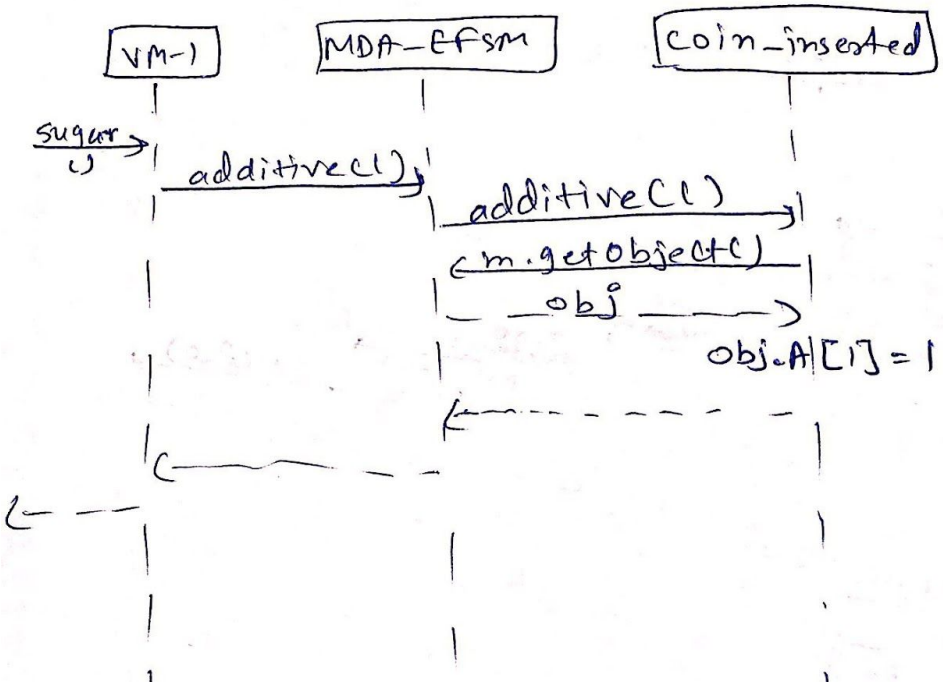


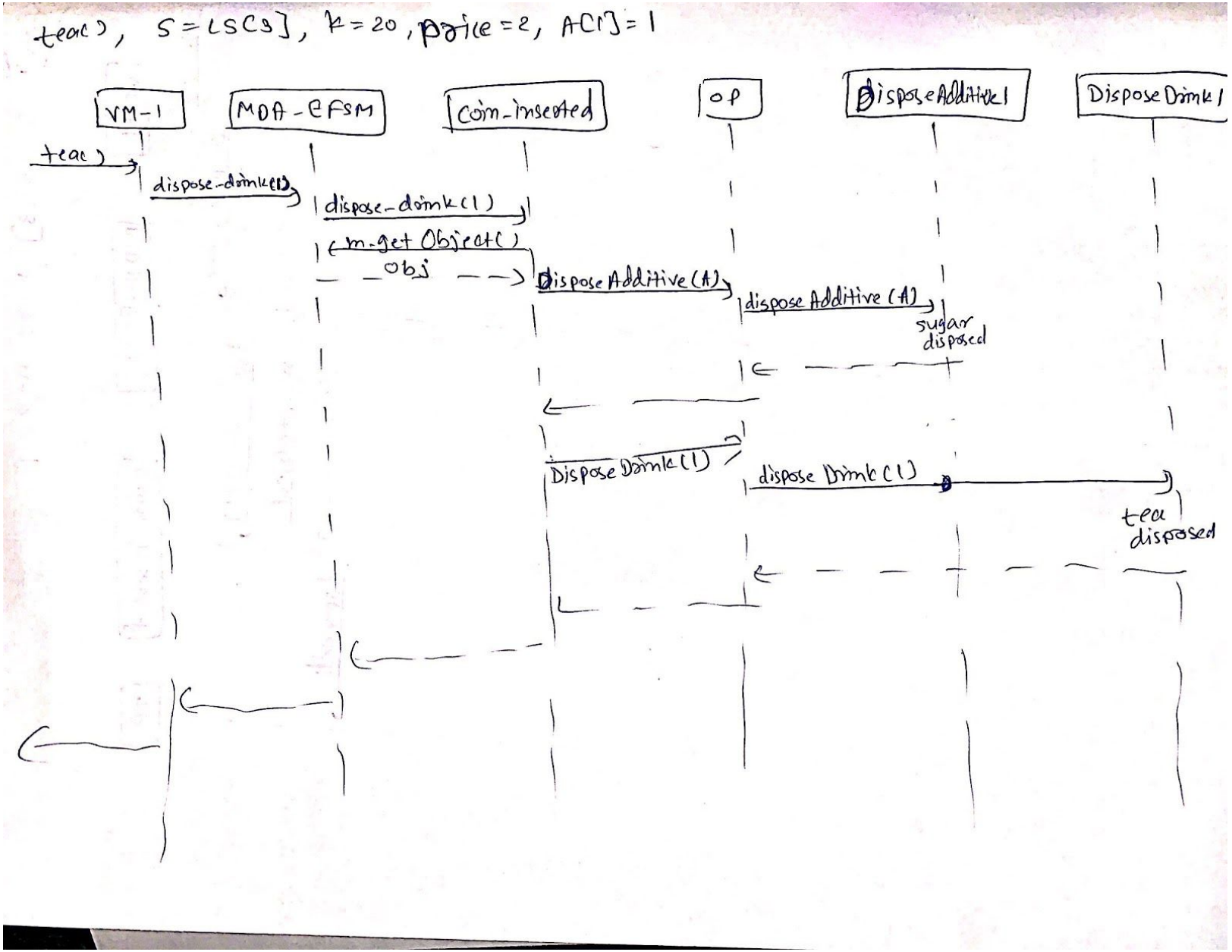

insert_cups(20)
S = LS[1]
VM-1
MDA-EFSM
no-cups
insert
-cups(20)
insert_cups(20)
insert_cups(20)
m.getObject()
m.getObject
MDA-EFSM
object
m.getObject.k
= 20
m.getObject()
object
m.getObject.
change-state(2)
change-state
(2)
S = LS[2]

card(7.2) S = LSC2], k = 20, price = 2



sugar() S=[sC3], k=20, n=2




tea(), S=[SC9], k=20, price=2, A[1]=1
VM-1
MDA-EFSM
Coin_inserted
op
DisposeAdditive1
DisposeDrink1
tea()
dispose-drink(1)
dispose-drink(1)
m-get Object()
obj
DisposeAdditive(A)
dispose Additive (A)
sugar disposed
Dispose Drink(1)
dispose Drink(1)
tea disposed