CS586

# PROJECT DETAIL REPORT

Yudong Wu
A20405374

CS586
A20405374
Yudong Wu

# Project Report

Part1:

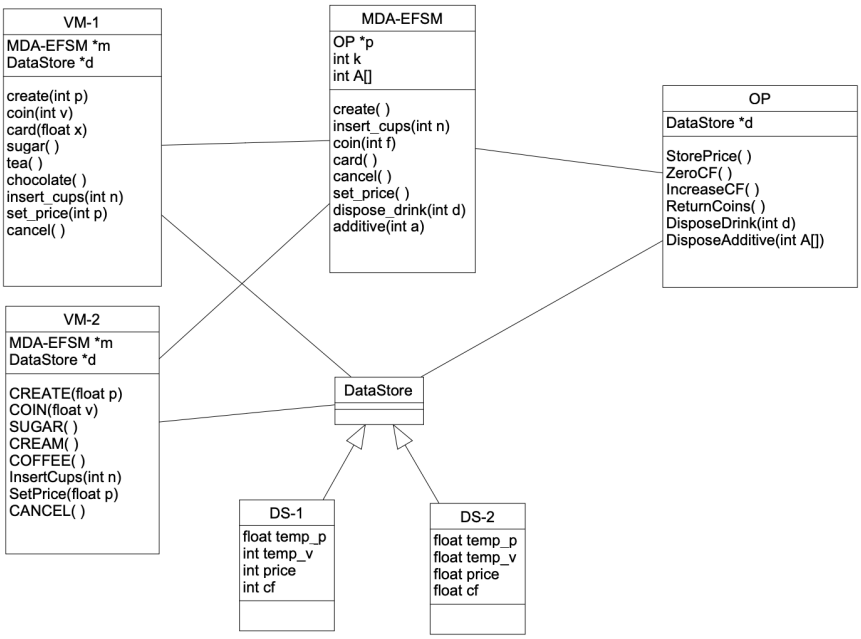
**MDA-EFSM Events:**

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

**MDA-EFSM Actions:**

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]=true

**Class Diagram for MDA_EFSM:**



**Pseudo-code:**

**Vending-Machine-1**

```
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**

```
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();
}
```

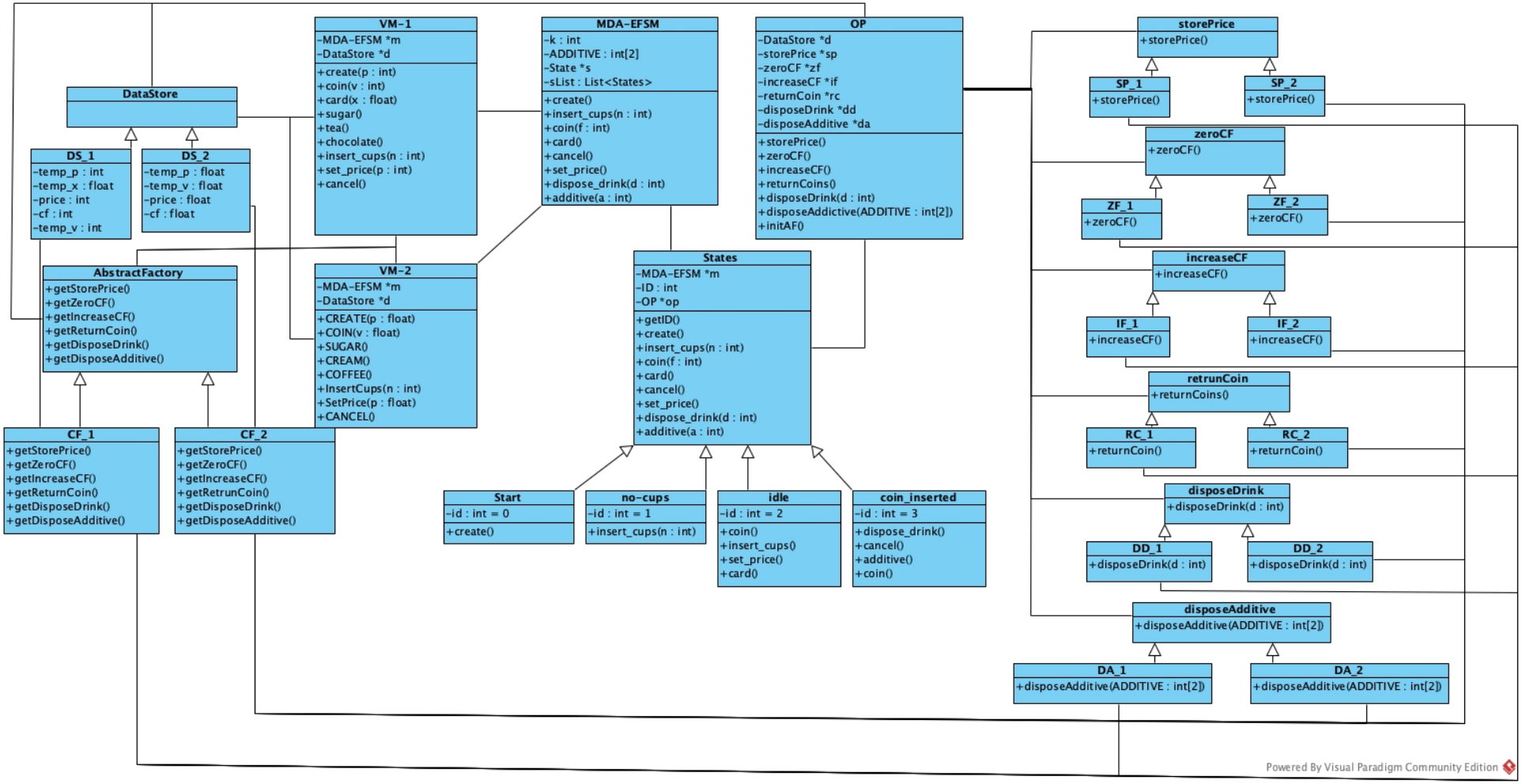
where,
*m:* pointer to the MDA-EFSM
d: pointer to the data store DS-2

In the data store:
*cf*: represents a cumulative fund
*price*: represents a price for a drink

Part2:

**Class Diagram:**

**See next page**


VM-1
-MDA-EFSM *m
-DataStore *d
+create(p : int)
+coin(v : int)
+card(x : float)
+sugar()
+tea()
+chocolate()
+insert_cups(n : int)
+set_price(p : int)
+cancel()
MDA-EFSM
-k : int
-ADDITIVE : int[2]
-State *s
-sList : List<States>
+create()
+insert_cups(n : int)
+coin(f : int)
+card()
+cancel()
+set_price()
+dispose_drink(d : int)
+additive(a : int)
OP
-DataStore *d
-storePrice *sp
-zeroCF *zf
-increaseCF *if
-returnCoin *rc
-disposeDrink *dd
-disposeAdditive *da
+storePrice()
+zeroCF()
+increaseCF()
+returnCoins()
+disposeDrink(d : int)
+disposeAddictive(ADDITIVE : int[2])
+initAF()
DataStore
DS_1
-temp_p : int
-temp_x : float
-price : int
-cf : int
-temp_v : int
DS_2
-temp_p : float
-temp_v : float
-price : float
-cf : float
AbstractFactory
+getStorePrice()
+getZeroCF()
+getIncreaseCF()
+getReturnCoin()
+getDisposeDrink()
+getDisposeAdditive()
VM-2
-MDA-EFSM *m
-DataStore *d
+CREATE(p : float)
+COIN(v : float)
+SUGAR()
+CREAM()
+COFFEE()
+InsertCups(n : int)
+SetPrice(p : float)
+CANCEL()
States
-MDA-EFSM *m
-ID : int
-OP *op
+getID()
+create()
+insert_cups(n : int)
+coin(f : int)
+card()
+cancel()
+set_price()
+dispose_drink(d : int)
+additive(a : int)
CF_1
+getStorePrice()
+getZeroCF()
+getIncreaseCF()
+getReturnCoin()
+getDisposeDrink()
+getDisposeAdditive()
CF_2
+getStorePrice()
+getZeroCF()
+getIncreaseCF()
+getRetrunCoin()
+getDisposeDrink()
+getDisposeAdditive()
Start
-id : int = 0
+create()
no-cups
-id : int = 1
+insert_cups(n : int)
idle
-id : int = 2
+coin()
+insert_cups()
+set_price()
+card()
coin_inserted
-id : int = 3
+dispose_drink()
+cancel()
+additive()
+coin()
storePrice
+storePrice()
SP_1
+storePrice()
SP_2
+storePrice()
zeroCF
+zeroCF()
ZF_1
+zeroCF()
ZF_2
+zeroCF()
increaseCF
+increaseCF()
IF_1
+increaseCF()
IF_2
+increaseCF()
retrunCoin
+returnCoins()
RC_1
+returnCoin()
RC_2
+returnCoin()
disposeDrink
+disposeDrink(d : int)
DD_1
+disposeDrink(d : int)
DD_2
+disposeDrink(d : int)
disposeAdditive
+disposeAdditive(ADDITIVE : int[2])
DA_1
+disposeAdditive(ADDITIVE : int[2])
DA_2
+disposeAdditive(ADDITIVE : int[2])
Powered By Visual Paradigm Community Edition

**DataStore Class:**
This is an abstract class for extend use.

**DS_1 Class:**
This is actual class to store data structure for VM_1;
it includes get/set functions for *"int temp_p, int temp_v, float tmep_x, int price, int CF"*
**DS_2 Class:**
This is actual class to store data structure for VM_2;
It includes get/set functions for *"float temp_fp, float temp_fv, float price, float CF"*

**VM_1 Class:**
This class contain functions that supported by VM_1;
MDA-EFSM *mda //pointer to MDA-EFSM
Create(int p){
    Create new Concreate Factory for VM_1;
    Call initialize(Concreate Factory CF_1) function in OP;
    Store p as temp_p in DS_1;
}
Coin(int v){
    Store v as temp_v in DS_1;
    If CF + v > price this is a valid coin insert function, call coin(1) function in MDA-EFSM;
    Else call coin(0) function in MDA-EFSM;
}
Card(float x){
    Store x as temp_x in DS_1;
    If price < x, Call card() function in MDA-EFSM;
    Else show not enough fund message;
}
Sugar(){
    Call sugar operation in MDA-EFSM;
}
Tea(){
    Call tea operation in MDA-EFSM;
}
Chocolate(){
    Call chocolate operation in MDA-EFSM;
}
Insert_cups(int n){
    Transfer n of cups to MDA-EFSM;
}
Set_price(int p){
    Store p as temp_p in DS_1;
    Call set_price() in MDA-EFSM.
}
Cancal(){
    Call cancel operation in MDA-EFSM.
}

**VM_2 Class:**

This class contain functions that supported by VM_2;

MDA-EFSM *mda

Create(float p){

    Create new ConcreateFactory2;

    Call initialize(ConcreateFactory CF_2) function in OP;

    Store p as temp_p in DS_2;

}

Coin(float v){

    Store v as temp_v in DS_2;

    If CF + v > price then this is valid coin, call coin(1) function in MDA-EFSM;

    Else call(0) in MDA-EFSM;

}

Cream(){

    Call cream() operation in MDA-EFSM;

}

Coffee(){

    Call coffee() operation in MDA-EFSM;

}

Insert_cups(int n){

    Transfer n of cups to MDA-EFSM;

}

Set_price(float p){

    Store p as temp_p in DS_2;

    Call set_price() in MDA-EFSM.

}

Cancal(){

    Call cancel operation in MDA-EFSM.

}

**MDA-EFSM Class:**

Because we use Centralized solution, the MDA-EFSM class is responsible for switching states.

The sList in MDA-EFSM store pointers of states;

OP *op;  State *state;

Int CUPS //store cup number  Boolean[] ADDITIVE {false, false}

States SList[4]: 0-start, 1-no-cups, 2-idle, 3-coin_Inserted;

Create(){

    Call create() function in state;

    Get current stateID, if is *start*, Change state to *no-cups*;

}

Insert_cups(int n){

    Update CUPS in MDA-EFSM;

    Call insert_cups function in state;

    If state = no_cups & n > 0, getStateID, if current state is *no_cups*, change state to *idle*;

}

coin(int I ){

    Call coin(i) function in state;

    I = 1, getStateID if current state is *idle*, change state to *coin_inserted*;

}
Card(){
	Call card() function in state;
}
Cancel(){
	Call cancel() function in state;
	If current state is *coin_inserted* change state to *idle*;
}
Set_price(){
	Call set_price() function In state;
}
Dispose_drink(int I){
	Call dispose_drink(i) function In state;
	getStateID, if current state is *coin_inserted* & CUPS > 0 change state to *idle*;
	if current state is *coin_inserted* & CUPS == 0 change state to *no_cups;*
}
Additive(int i){
	Change status of ADDITIVE[i];
}

**States Class:**
States class is an abstract class;
Int ID		//stateID
getID(){
	return current state ID;
}

**Start Class:**
Class for Start state
Create(){
	Call store_price() function in OP;
}

**no-cups Class:**
class for no_cups state
insert_cups(){
	call zeroCF() function in OP;
}
Coin(){
	Call returnCoin() function in OP;
}

**Idle Class:**
Class for Idle state
Coin(){
	Call inceaseCF() function In OP;
}
Insert_cups(){

```
        Do nothing here
}
Set_price(){
        Call storePrice() function in OP;
}
Card(){
        Do nothing here;
}
```

**Coin inserted Class:**

```
Class for coin_inserted state
Dispose_drink(){
        Call disposeDrink() function in OP;
        Call disposeAdditive(ADDITIVE[]) function in OP;
        Call zeroCF() function in OP;
}
Cancel(){
        Call returnCoin() zeroCF() function in OP;
}
Additive(){
        Do nothing here;
}
Coin(){
        Call returnCoin() function in OP;
}
```

**OP Class:**

```
this is actual action class which write data to DataStore Class;
StorePrice* sp          ZeroCF* ZF      IncreaseCF* IF          ReturnCoin* RC          DisposeDrink* DD
DisposeAdditive* DA
initAF(ConcreateFactory CF){
        call getStorePrice() function in AbstractFacory;
        set sp;
        call getZeroCF() function in AbstractFacory;
        set zf;
        call getIncrease() function in AbstractFacory;
        set if;
        call getReturnCoin() function in AbstractFacory;
        set rc;
        call getDisposeDrink() function in AbstractFacory;
        set dd;
        call getDisposeAdditive() function in AbstractFacory;
        set da;
}
storePrice(){
        call storePrice() in sp;
}
zeroCF(){
```

```
        call zeroCF() function in zf;
}
increaseCF(){
        call increaseCF() function in if;
}
returnCoin(){
        call returnCoin() function in rc;
}
disposeDrink(){
        call disposeDrink() function in dd;
}
disposeAdditive(){
        call disposeAdditive() function in da;
}
```

**StorePrice; zeroCF; increaseCF; returnCoin; disposeDrink; disposeAdditive are abstract classes;**

**SP_1 Class:**

```
This is VM_1 storePrice strategy class;
storePrice(){
        read temp_p in DS-1;
        set Price as temp_p;
}
```

**ZF_1 Class:**

```
This is VM_1 zeroCF strategy class;
zeroCF(){
        set CF as 0 in DS_1;
}
```

**IF_1 Class:**

```
this is VM_1 increaseCF strategy class;
increaseCF(){
        read temp_v, CF in DS_1;
        set new CF as temp_v + CF;
}
```

**RC_1 Class:**

```
this is VM_1 returnCoin strategy class;
returnCoin(){
        read temp_v in DS_1;
        return temp_v to user;
}
```

**DD_1 Class:**

```
This is VM_1 disposeDrink strategy class;
1 as tea, 2 as chocolate
disposeDrink(int I ){
        if I == 1 dispose tea;
```

```
        if I == 2 dispose chocolate;
}
```

**DA_1 Class:**

```
This is VM_1 disposeAdditive strategy class;
disposeAdditive(ADDITIVE[]){
        if additive[0] is true dispose sugar;
}
```

**SP_2 Class:**

```
This is VM_2 storePrice strategy class;
storePrice(){
        read temp_p in DS-2;
        set Price as temp_p;
}
```

**ZF_2 Class:**

```
This is VM_2 zeroCF strategy class;
zeroCF(){
        set CF as 0 in DS_2;
}
```

**IF_2 Class:**

```
this is VM_2 increaseCF strategy class;
increaseCF(){
        read temp_v, CF in DS_2;
        set new CF as temp_v + CF;
}
```

**RC_2 Class:**

```
this is VM_2 returnCoin strategy class;
returnCoin(){
        read temp_v in DS_2;
        return temp_v to user;
}
```

**DD_2 Class:**

```
This is VM_2 disposeDrink strategy class;
3 as coffee
disposeDrink(int I ){
        if I == 3 dispose coffee;
}
```

**DA_2 Class:**

```
This is VM_2 disposeAdditive strategy class;
disposeAdditive(ADDITIVE[]){
        if additive[1] is true dispose cream;
}
```

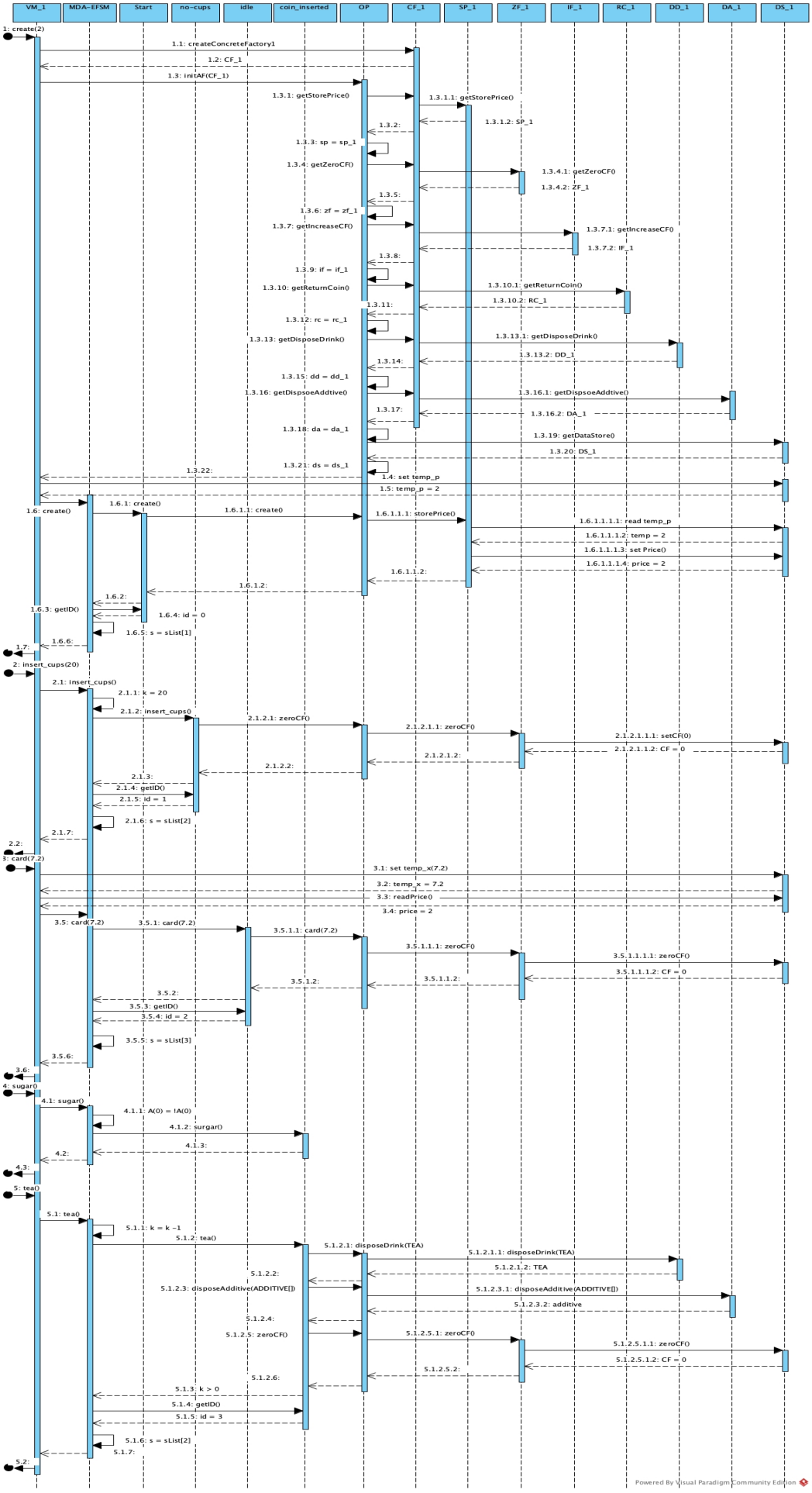
**AbstractFactory Class:**
This is an abstract class;

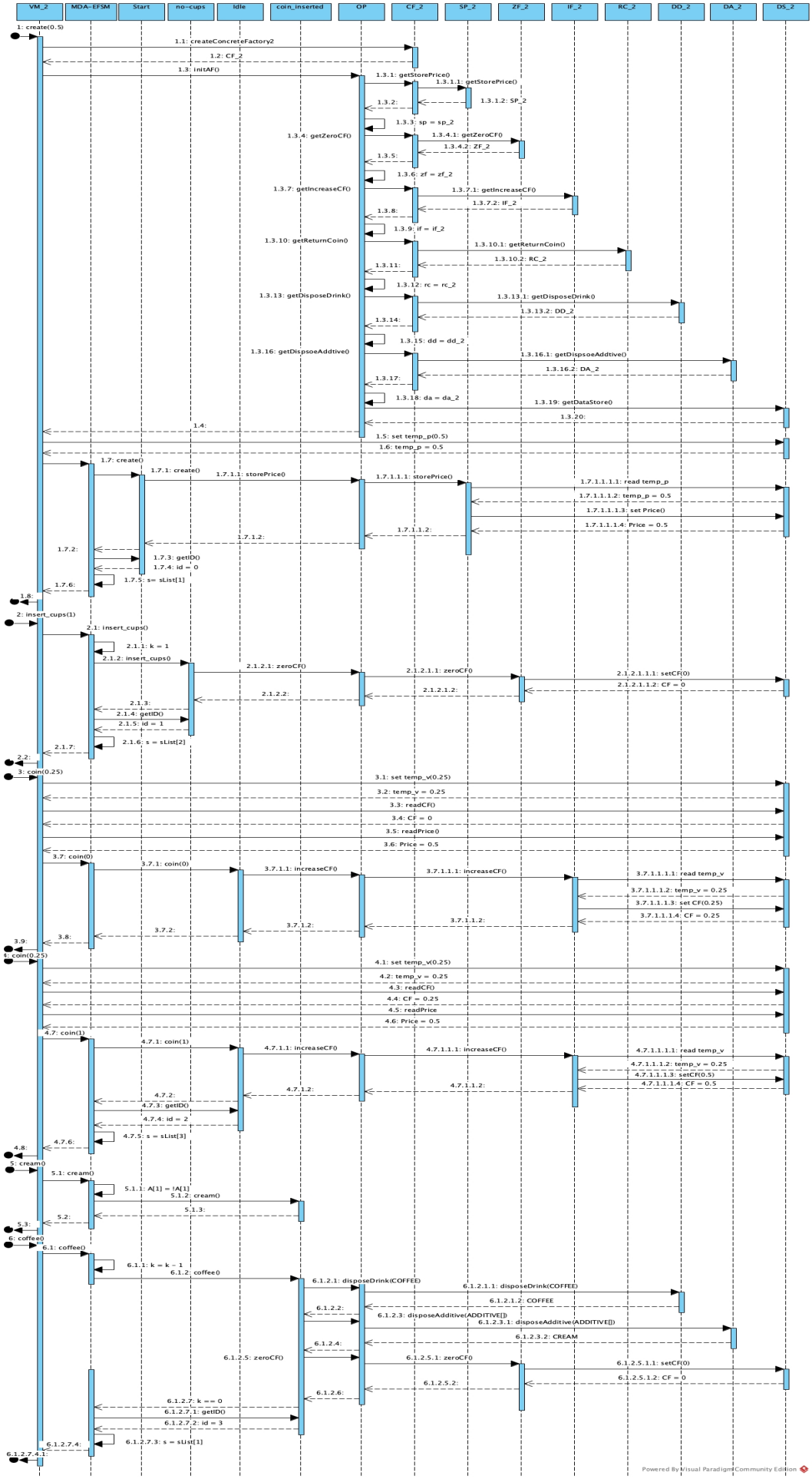
**CF_1 Class:**
This is concrete fctory class for VM_1;
getStorePrice(){
	create new SP_1 class;
}
getZeroCF(){
	create new ZF_1 class;
}
getIncreaseCF(){
	create new IF_1 class;
}
getReturnCoin(){
	create new RC_1 class;
}
getDisposeDrink(){
	create new DD_1 class;
}
getDisposeAdditive(){
	create new DA_1 class;
}
getDataStore(){
	create new DS_1 class;
}

**CF_2 Class:**
This is concrete factory class for VM_2;
getStorePrice(){
	create new SP_2 class;
}
getZeroCF(){
	create new ZF_2 class;
}
getIncreaseCF(){
	create new IF_2 class;
}
getReturnCoin(){
	create new RC_2 class;
}
getDisposeDrink(){
	create new DD_2 class;
}
getDisposeAdditive(){
	create new DA_2 class;
}

```
getDataStore(){
        create new DS_2 class;
}
```

**Sequence Diagram:**
(see next page)


VM_1
MDA-EFSM
Start
no-cups
idle
coin_inserted
OP
CF_1
SP_1
ZF_1
IF_1
RC_1
DD_1
DA_1
DS_1
1: create(2)
1.1: createConcreteFactory1
1.2: CF_1
1.3: initAF(CF_1)
1.3.1: getStorePrice()
1.3.1.1: getStorePrice()
1.3.1.2: SP_1
1.3.2:
1.3.3: sp = sp_1
1.3.4: getZeroCF()
1.3.4.1: getZeroCF()
1.3.4.2: ZF_1
1.3.5:
1.3.6: zf = zf_1
1.3.7: getIncreaseCF()
1.3.7.1: getIncreaseCF()
1.3.7.2: IF_1
1.3.8:
1.3.9: if = if_1
1.3.10: getReturnCoin()
1.3.10.1: getReturnCoin()
1.3.10.2: RC_1
1.3.11:
1.3.12: rc = rc_1
1.3.13: getDisposeDrink()
1.3.13.1: getDisposeDrink()
1.3.13.2: DD_1
1.3.14:
1.3.15: dd = dd_1
1.3.16: getDispsoeAddtive()
1.3.16.1: getDispsoeAddtive()
1.3.16.2: DA_1
1.3.17:
1.3.18: da = da_1
1.3.19: getDataStore()
1.3.20: DS_1
1.3.21: ds = ds_1
1.3.22:
1.4: set temp_p
1.5: temp_p = 2
1.6: create()
1.6.1: create()
1.6.1.1: create()
1.6.1.1.1: storePrice()
1.6.1.1.1.1: read temp_p
1.6.1.1.1.2: temp = 2
1.6.1.1.1.3: set Price()
1.6.1.1.1.4: price = 2
1.6.1.1.2:
1.6.1.2:
1.6.2:
1.6.3: getID()
1.6.4: id = 0
1.6.5: s = sList[1]
1.6.6:
1.7:
2: insert_cups(20)
2.1: insert_cups()
2.1.1: k = 20
2.1.2: insert_cups()
2.1.2.1: zeroCF()
2.1.2.1.1: zeroCF()
2.1.2.1.1.1: setCF(0)
2.1.2.1.1.2: CF = 0
2.1.2.1.2:
2.1.2.2:
2.1.3:
2.1.4: getID()
2.1.5: id = 1
2.1.6: s = sList[2]
2.1.7:
2.2:
3: card(7.2)
3.1: set temp_x(7.2)
3.2: temp_x = 7.2
3.3: readPrice()
3.4: price = 2
3.5: card(7.2)
3.5.1: card(7.2)
3.5.1.1: card(7.2)
3.5.1.1.1: zeroCF()
3.5.1.1.1.1: zeroCF()
3.5.1.1.1.2: CF = 0
3.5.1.1.2:
3.5.1.2:
3.5.2:
3.5.3: getID()
3.5.4: id = 2
3.5.5: s = sList[3]
3.5.6:
3.6:
4: sugar()
4.1: sugar()
4.1.1: A(0) = !A(0)
4.1.2: surgar()
4.1.3:
4.2:
4.3:
5: tea()
5.1: tea()
5.1.1: k = k -1
5.1.2: tea()
5.1.2.1: disposeDrink(TEA)
5.1.2.1.1: disposeDrink(TEA)
5.1.2.1.2: TEA
5.1.2.2:
5.1.2.3: disposeAdditive(ADDITIVE[])
5.1.2.3.1: disposeAdditive(ADDITIVE[])
5.1.2.3.2: additive
5.1.2.4:
5.1.2.5: zeroCF()
5.1.2.5.1: zeroCF()
5.1.2.5.1.1: zeroCF()
5.1.2.5.1.2: CF = 0
5.1.2.5.2:
5.1.2.6:
5.1.3: k > 0
5.1.4: getID()
5.1.5: id = 3
5.1.6: s = sList[2]
5.1.7:
5.2:
Powered By Visual Paradigm Community Edition


VM_2
MDA-EFSM
Start
no-cups
Idle
coin_inserted
OP
CF_2
SP_2
ZF_2
IF_2
RC_2
DD_2
DA_2
DS_2
1: create(0.5)
1.1: createConcreteFactory2
1.2: CF_2
1.3: initAF()
1.3.1: getStorePrice()
1.3.1.1: getStorePrice()
1.3.1.2: SP_2
1.3.2:
1.3.3: sp = sp_2
1.3.4: getZeroCF()
1.3.4.1: getZeroCF()
1.3.4.2: ZF_2
1.3.5:
1.3.6: zf = zf_2
1.3.7: getIncreaseCF()
1.3.7.1: getIncreaseCF()
1.3.7.2: IF_2
1.3.8:
1.3.9: if = if_2
1.3.10: getReturnCoin()
1.3.10.1: getReturnCoin()
1.3.10.2: RC_2
1.3.11:
1.3.12: rc = rc_2
1.3.13: getDisposeDrink()
1.3.13.1: getDisposeDrink()
1.3.13.2: DD_2
1.3.14:
1.3.15: dd = dd_2
1.3.16: getDispsoeAddtive()
1.3.16.1: getDispsoeAddtive()
1.3.16.2: DA_2
1.3.17:
1.3.18: da = da_2
1.3.19: getDataStore()
1.3.20:
1.4:
1.5: set temp_p(0.5)
1.6: temp_p = 0.5
1.7: create()
1.7.1: create()
1.7.1.1: storePrice()
1.7.1.1.1: storePrice()
1.7.1.1.1.1: read temp_p
1.7.1.1.1.2: temp_p = 0.5
1.7.1.1.1.3: set Price()
1.7.1.1.1.4: Price = 0.5
1.7.1.1.2:
1.7.1.2:
1.7.2:
1.7.3: getID()
1.7.4: id = 0
1.7.5: s= sList[1]
1.7.6:
1.8:
2: insert_cups(1)
2.1: insert_cups()
2.1.1: k = 1
2.1.2: insert_cups()
2.1.2.1: zeroCF()
2.1.2.1.1: zeroCF()
2.1.2.1.1.1: setCF(0)
2.1.2.1.1.2: CF = 0
2.1.2.1.2:
2.1.2.2:
2.1.3:
2.1.4: getID()
2.1.5: id = 1
2.1.6: s = sList[2]
2.1.7:
2.2:
3: coin(0.25)
3.1: set temp_v(0.25)
3.2: temp_v = 0.25
3.3: readCF()
3.4: CF = 0
3.5: readPrice()
3.6: Price = 0.5
3.7: coin(0)
3.7.1: coin(0)
3.7.1.1: increaseCF()
3.7.1.1.1: increaseCF()
3.7.1.1.1.1: read temp_v
3.7.1.1.1.2: temp_v = 0.25
3.7.1.1.1.3: set CF(0.25)
3.7.1.1.1.4: CF = 0.25
3.7.1.1.2:
3.7.1.2:
3.7.2:
3.8:
3.9:
4: coin(0.25)
4.1: set temp_v(0.25)
4.2: temp_v = 0.25
4.3: readCF()
4.4: CF = 0.25
4.5: readPrice
4.6: Price = 0.5
4.7: coin(1)
4.7.1: coin(1)
4.7.1.1: increaseCF()
4.7.1.1.1: increaseCF()
4.7.1.1.1.1: read temp_v
4.7.1.1.1.2: temp_v = 0.25
4.7.1.1.1.3: setCF(0.5)
4.7.1.1.1.4: CF = 0.5
4.7.1.1.2:
4.7.1.2:
4.7.2:
4.7.3: getID()
4.7.4: id = 2
4.7.5: s = sList[3]
4.7.6:
4.8:
5: cream()
5.1: cream()
5.1.1: A[1] = !A[1]
5.1.2: cream()
5.1.3:
5.2:
5.3:
6: coffee()
6.1: coffee()
6.1.1: k = k - 1
6.1.2: coffee()
6.1.2.1: disposeDrink(COFFEE)
6.1.2.1.1: disposeDrink(COFFEE)
6.1.2.1.2: COFFEE
6.1.2.2:
6.1.2.3: disposeAdditive(ADDITIVE[])
6.1.2.3.1: disposeAdditive(ADDITIVE[])
6.1.2.3.2: CREAM
6.1.2.4:
6.1.2.5: zeroCF()
6.1.2.5.1: zeroCF()
6.1.2.5.1.1: setCF(0)
6.1.2.5.1.2: CF = 0
6.1.2.5.2:
6.1.2.6:
6.1.2.7: k == 0
6.1.2.7.1: getID()
6.1.2.7.2: id = 3
6.1.2.7.3: s = sList[1]
6.1.2.7.4:
6.1.2.7.4.1:
Powered By Visual Paradigm Community Edition