CS586 SOFTWARE SYSTEM ARCHITECTURE

PROJECT REPORT

Yogasree Segar

A20448385

Aim

To design a gas pump model with abstract factory pattern, state pattern and strategy pattern to get a good design of the model.

MDA_EFSM ACTIONS:

StorePrices // stores price(s) for the gas from the temporary data store

PayMsg // displays a type of payment method

StoreCash // stores cash from the temporary data store

DisplayMenu // display a menu with a list of selections

RejectMsg // displays credit card not approved message

SetPrice(int g) // set the price for the gas identified by g identifier as in SelectGas(int g)

SetInitialValues // set G (or L) and total to 0;

PumpGasUnit // disposes unit of gas and counts # of units disposed

GasPumpedMsg // displays the amount of disposed gas

PrintReceipt // print a receipt

CancelMsg // displays a cancellation message

ReturnCash // returns the remaining cash

WrongPinMsg // displays incorrect pin message

StorePin // stores the pin from the temporary data store

EnterPinMsg // displays a message to enter pin

InitializeData // set the value of price to 0 for GP-2; do nothing for GP-1

EjectCard() // card is ejected

SetW(int w) // set value for cash flag

MDA_EFSM EVENTS:

Activate()

Start()

PayCredit()

PayCash()

PayDebit()

Reject()

Cancel()

Approved()

StartPump()

Pump()

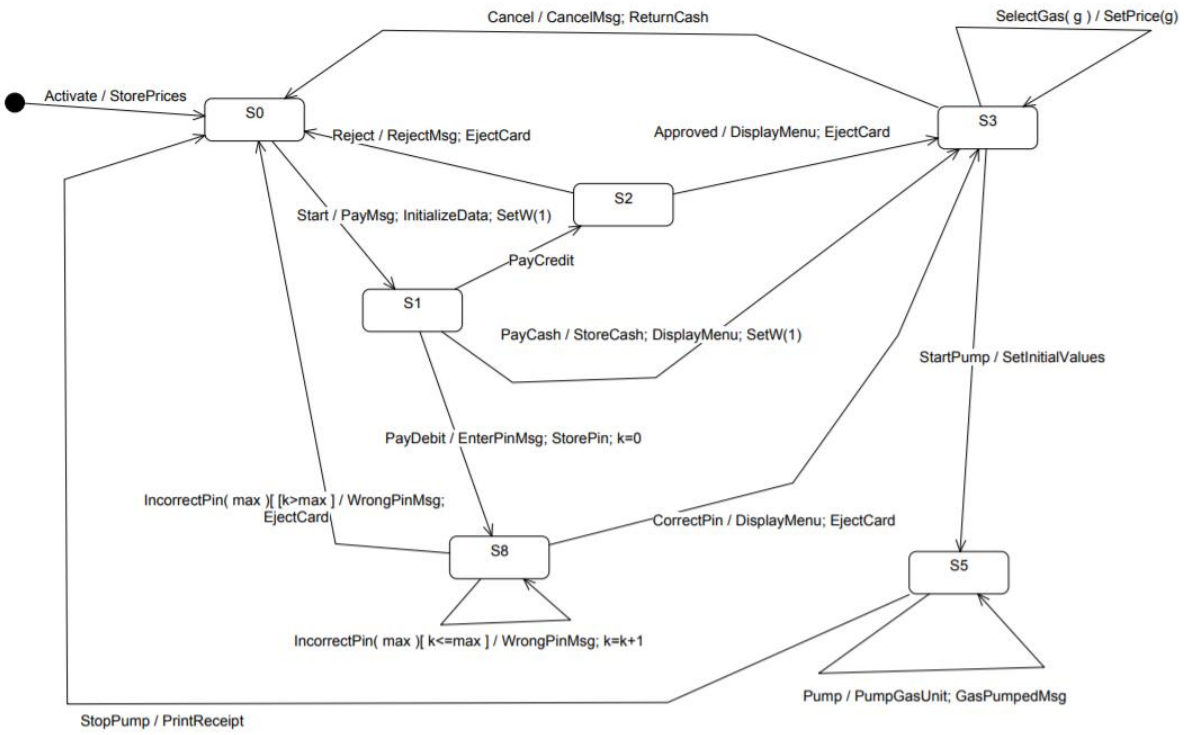
StopPump()

SelectGas(int g)

CorrectPin()

IncorrectPin(int max)

STATE DIAGRAM



MDA-EFSM for Gas Pumps

PSEUDOCODE FOR GP1 AND GP2

GP1

```
Activate(int a) {
 if (a>0) {
 d->temp_a=a;
 m->Activate()
 }
}
Start() {
 m->Start();
}
PayCash(float c) {
 if (c>0) {
 d->temp_c=c;
 m->PayCash()
 }
}
PayCredit() {
 m->PayCredit();
}
Reject() {
 m->Reject();
}
Approved() {
 m-> Approved();
}
Cancel() {
 m->Cancel();
}
StartPump() {
```

```
 m->StartPump();
}
PumpLiter() {
if (d->w==1) m->Pump()
else if (d->cash>0)&&(d->cash < d->price*(d->L+1))
 m->StopPump();
else m->Pump()
}
StopPump() {
 m->StopPump();
}
```

Notice:

cash: contains the value of cash deposited

price: contains the price of the selected gas L: contains the number of liters already pumped

w: cash flag (cash: w=0; otherwise: w=1)

cash,

L, price, w are in the data store

m: is a pointer to the MDA-EFSM object d: is a pointer to the Data Store object

GP2

```
Activate(float a, float b) {
 if ((a>0)&&(b>0)&&(c>0)) {
 d->temp_a=a;
 d->temp_b=b;
 d->temp_c=c;
 m->Activate()
 }
}
Start() {
```

```
 m->Start();
}
PayCredit() {
 m->PayCredit();
}
Reject() {
 m->Reject();
}
PayDebit(string p) {
 d->temp_p=p;
 m->PayDebit();
}
Pin(string x) {
 if (d->pin==x) m->CorrectPin()
 else m->InCorrectPin(1);
}
Cancel() {
 m->Cancel();
}
Approved() {
 m->Approved();
}
Diesel() {
 m->SelectGas(3)
}
Regular() {
 m->SelectGas(1)
}
```

```
Super() {
 m->SelectGas(2)
}
StartPump() {
 if (d->price>0) m->StartPump();
}
PumpGallon() {
 m->Pump();
StopPump() {
 m->StopPump();
}
FullTank() {
 m->StopPump();
}
```

Notice:

pin: contains the pin in the data store m: is a pointer to the MDA-EFSM object d: is a pointer to the Data Store object

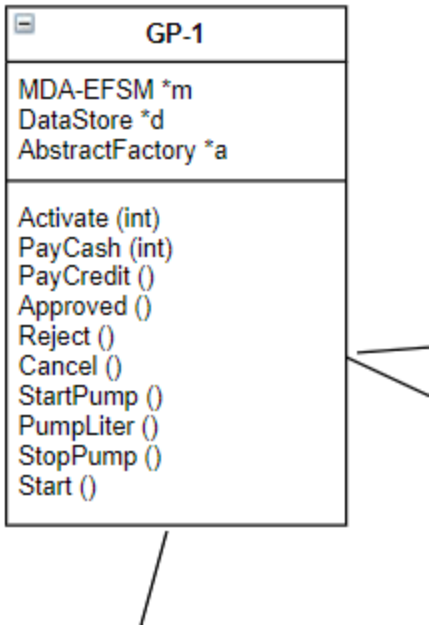
SelectGas(g): Regular: g=1; Super: g=2; Diesel: g=3

Reference: Referred to the contents provided in the class

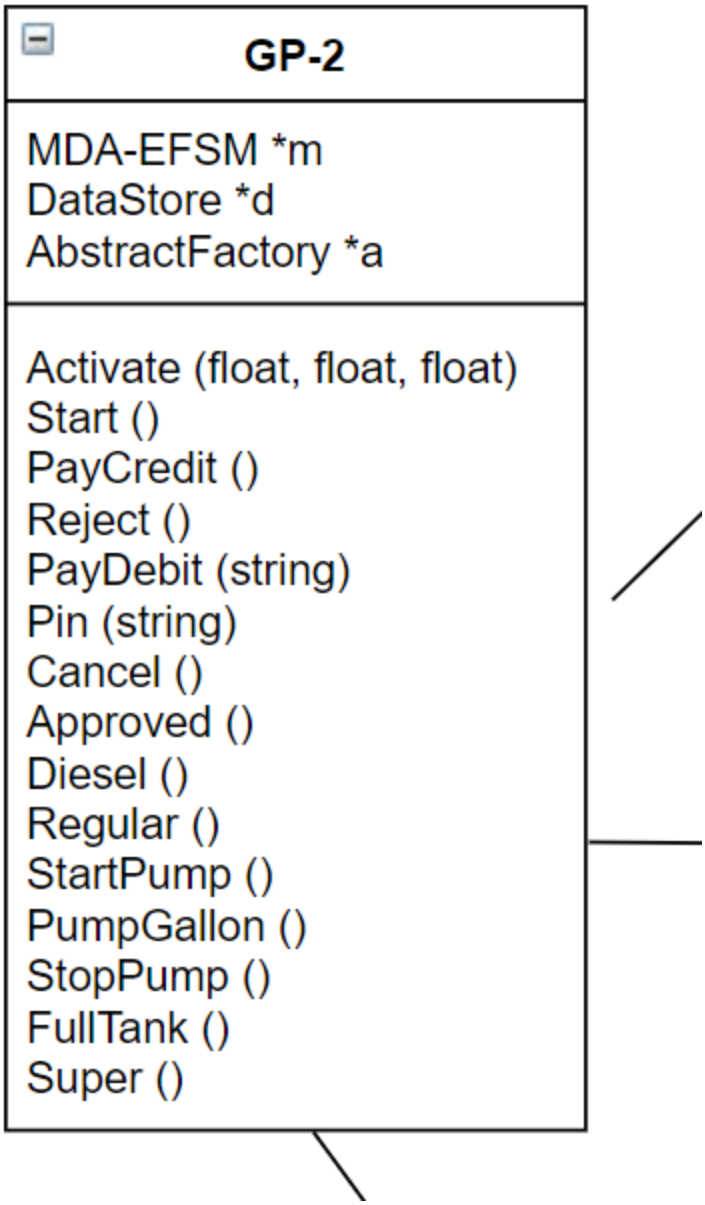
RESPONSIBILITIES OF EACH CLASS

Class GP1

GasPump1 with its own set of functions and parameters. It dispenses specific fuel in liters. It has its own parameter settings and functions.

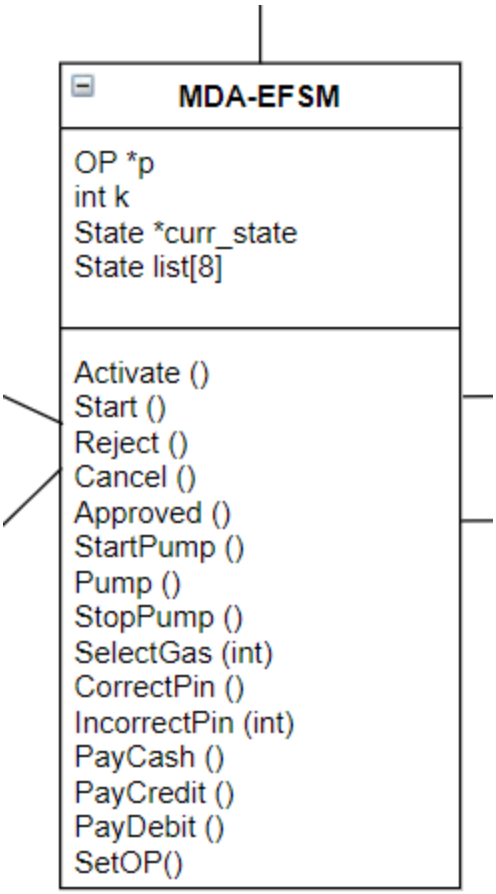


Class GP2 – another model of gaspump that dispenses specific fuel in gallons. This too has its own parameters and functions.

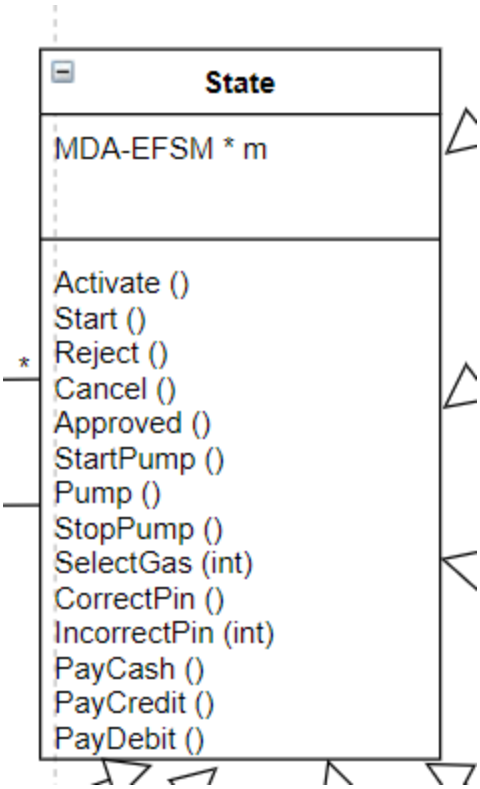


Class MDA_EFSM:

MDA_EFSM class to which the gaspumps, OP are connected. It is responsible to get the meta events and do the meta actions.

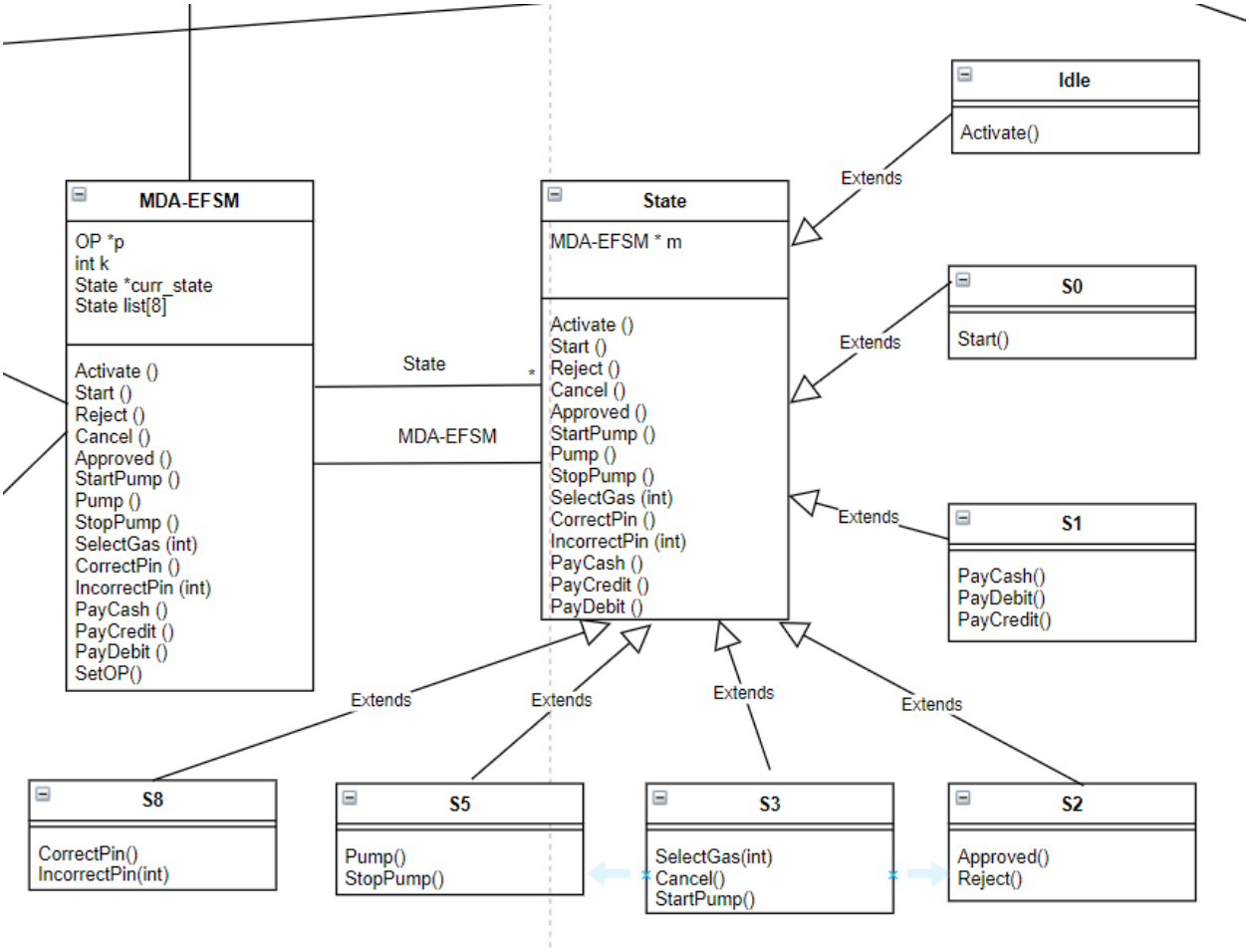


State Class – The super state class to which other classes are connected for de centralized pattern. It includes all the methods of MDA_EFSM in the abstract forms. The MDA_EFSM and the state classes are connected to each other.



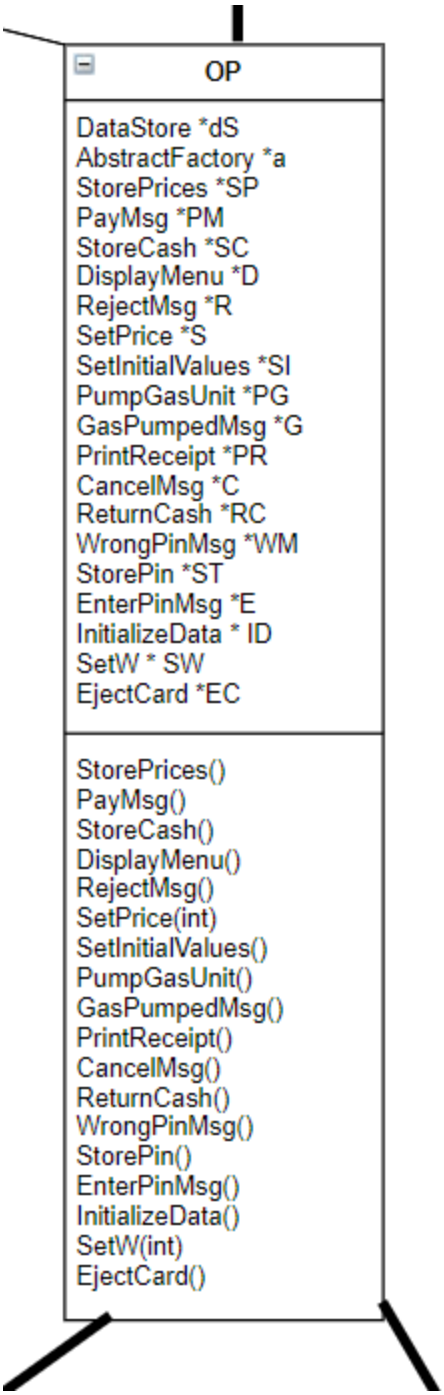
States:

There are 7 state classes in this model that perform each of the abstract methods in the state super class.



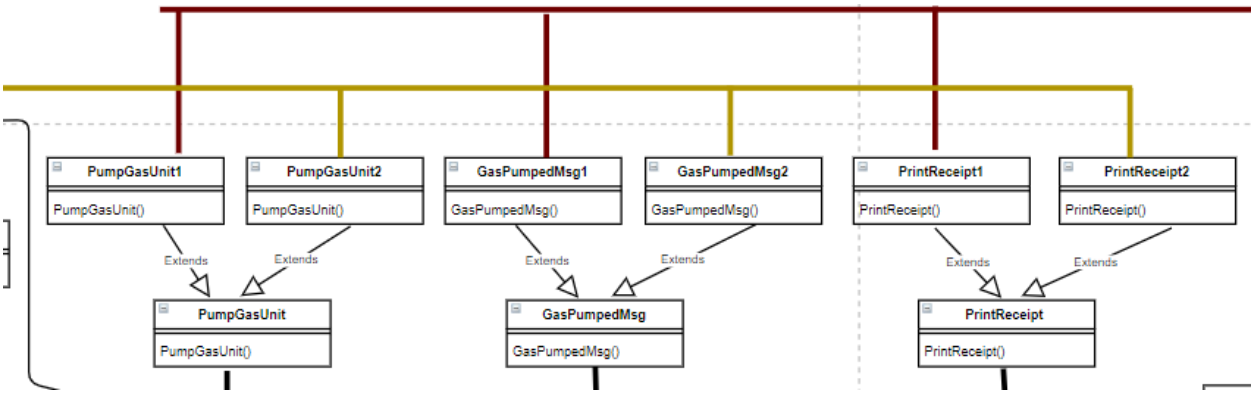
Class OP

This class performs all the meta actions. This class has a strategy pattern implemented on it as the meta actions performs variety of actions according to the gas pump models



Strategy Classes:

All the individual actions of the OP are implemented as separate classes. Further these classes can be again extended into multiple classes and each of them can be connected to the gas pump models according to their requirements.



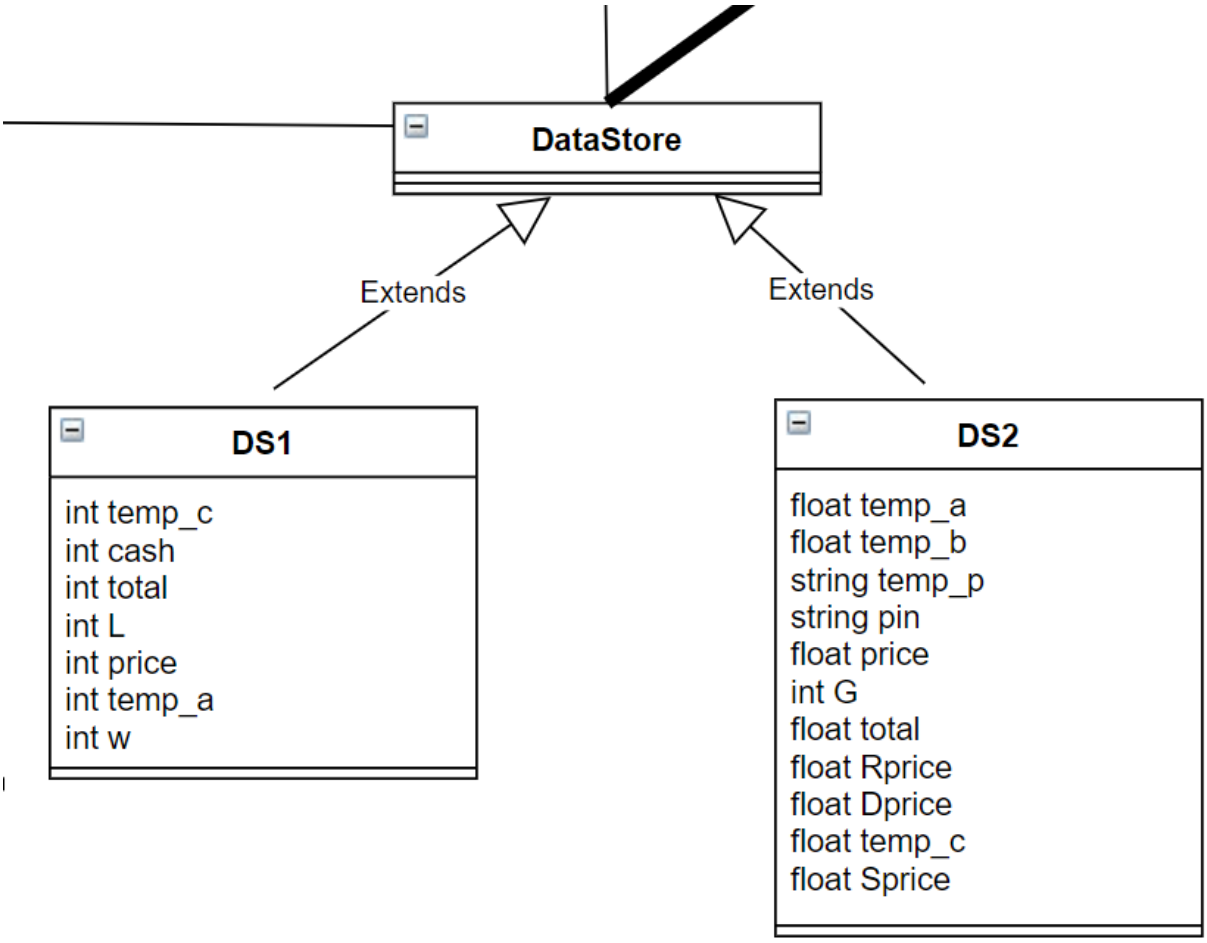
(Sample Strategy classes)

(Entire Strategy implementation is included at the end)

Abstract Factory Classes: Abstract Factory classes are created connecting to OP. It includes the abstract functions of all the functions of the OP. Two separate abstract factories are then created and extended to the super abstract factory class. The first class is connected to all the first class of the strategy patterns and to thegaspump 1. Similarly abstract factory 2 is connected to gas pump 2 and all the second classes of the strategy patterns.

The brown line connects to GP1 and green to GP2

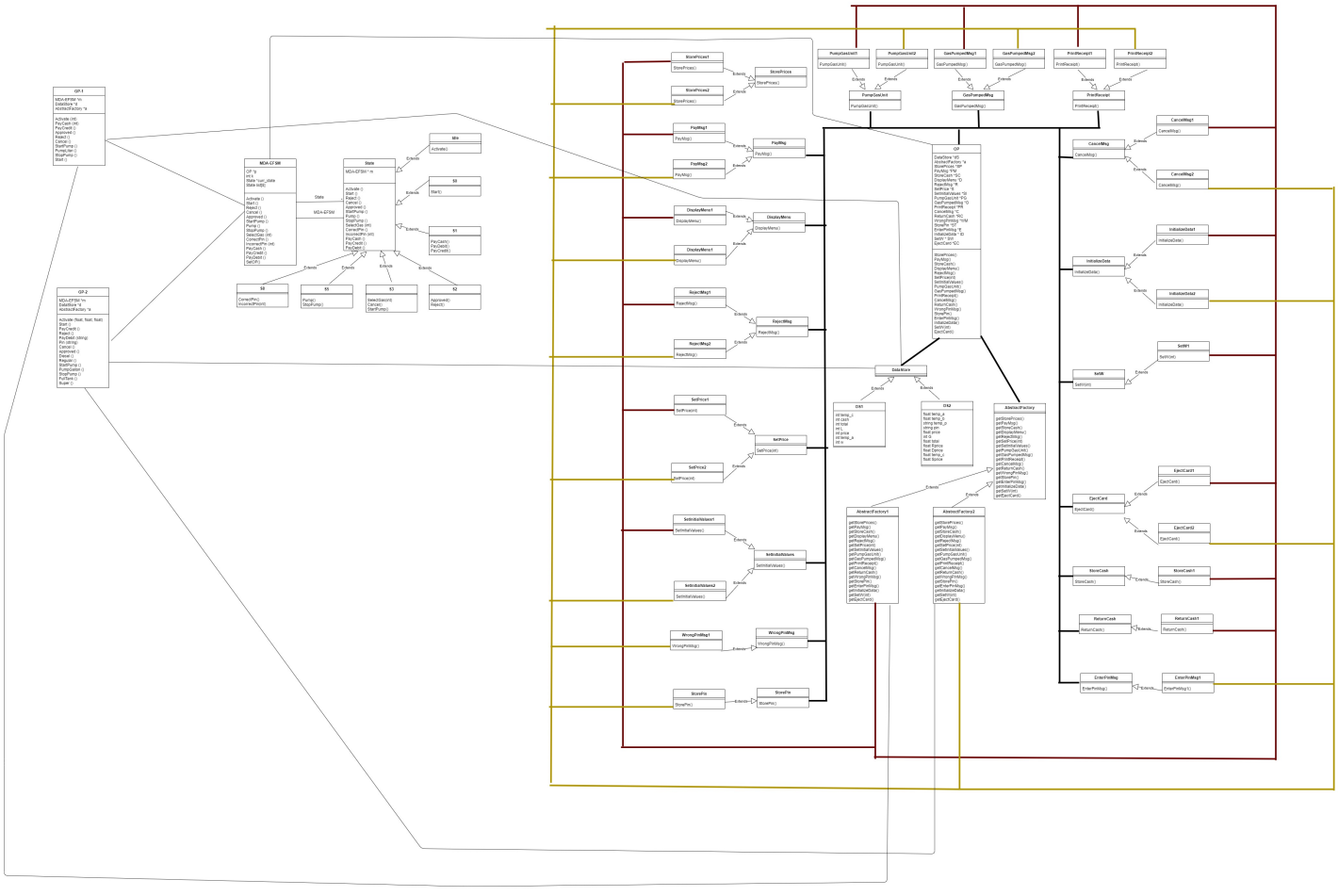
DataStore Class: The datastore class acts as a storage of data. It stores both the temporary variables and the permanent variables needed for the class functions. There is a super class to which 2 sub classes are extended from. DS1 for GP1 and DS2 for GP2. This class is connected to the gas pump models, OP and to the Abstract Factory through OP.

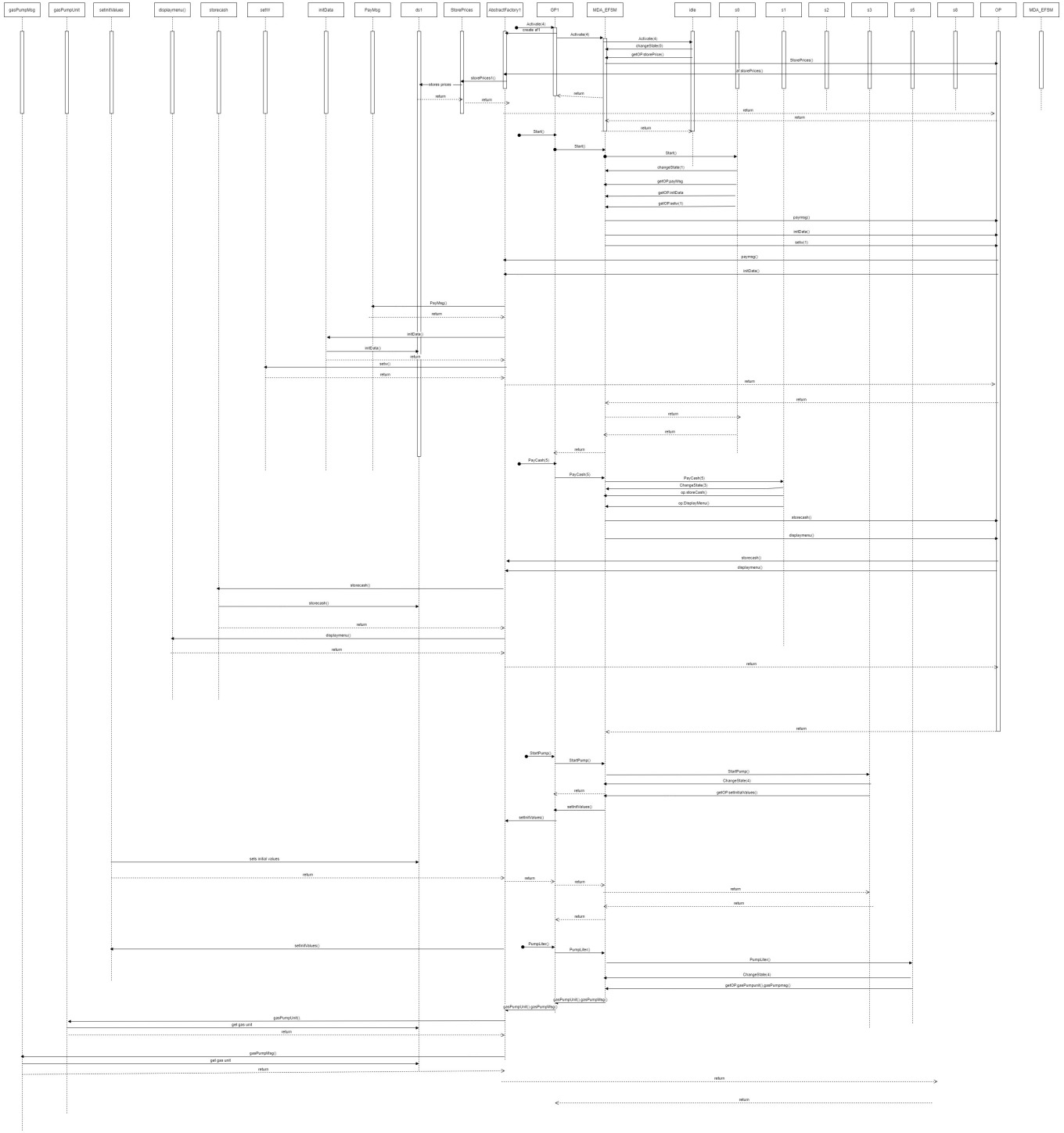


Note:

The class diagram attached is for the model given. However the code is written with as many classes as possible for easy future enhancements. (eg Strategy pattern applied to gaspumps, all the strategy classes are created and implemented but not added in the class diagram).

Class Diagram




gasPumpMsg
gasPumpUnit
setInitValues
displaymenu()
storecash
setW
initData
PayMsg
ds1
StorePrices
AbstractFactory1
GP1
MDA_EFSM
idle
s0
s1
s2
s3
s5
s8
OP
MDA_EFSM
Activate(4)
create gP1
Activate(4)
Activate(4)
changeState(0)
getOP.storePrices()
StorePrices()
storePrices()
storePrices()
stores prices
return
return
return
return
return
Start()
Start()
Start()
changeState(1)
getOP.payMsg
getOP.initData
getOP.setw(1)
paymsg()
initData()
setw(1)
paymsg()
initData()
PayMsg()
return
initData()
initData()
return
setw()
return
return
return
return
return
return
PayCash(5)
PayCash(5)
PayCash(5)
ChangeState(3)
op.storeCash()
op.DisplayMenu()
storecash()
displaymenu()
storecash()
displaymenu()
storecash()
storecash()
return
displaymenu()
return
return
return
StartPump()
StartPump()
StartPump()
ChangeState(4)
return
getOP.setInitialValues()
setInitValues()
setInitValues()
sets initial values
return
return
return
return
return
return
PumpLiter()
PumpLiter()
setInitValues()
PumpLiter()
ChangeState(4)
getOP.gasPumpunit().gasPumpmsg()
gasPumpUnit().gasPumpMsg()
gasPumpUnit().gasPumpMsg()
gasPumpUnit()
get gas unit
return
gasPumpMsg()
get gas unit
return
return
return