Software System Architecture

Project Report & Deliveries – A20560454

I: REPORT

1)

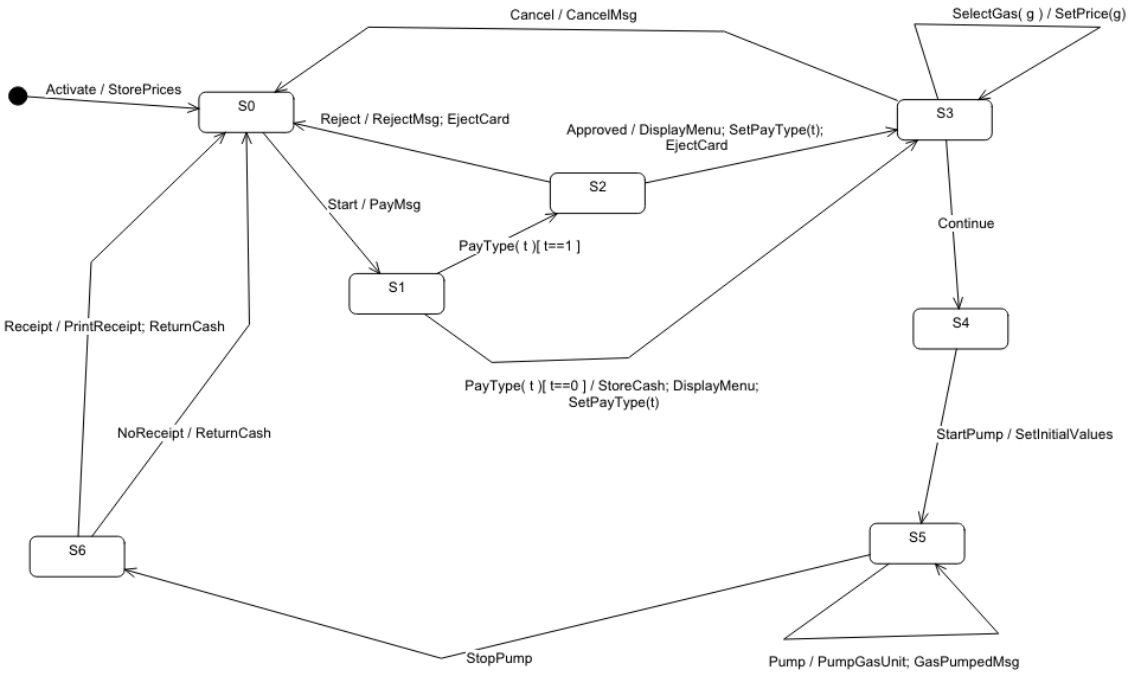
MDA-EFSM Meta Events

Activate()
Start()
PayType(int t) // Cash, t = 0; Credit, t = 1
Reject()
Cancel()
Approved()
StartPump()
SelectGasType(int g) // Regular, g = 1; Diesel, g = 2; Premium, g = 3
Pump()
StopPump()
Receipt()
NoReceipt()
Continue()

MDA-EFSM Meta Actions

StorePrices() // stores prices of gas from temp data store
PayMsg() // display type of payment method
StoreCash() // store cash from the temp data store
DisplayMenu() // display menu with list of selections
RejectMsg() // show credit card not approved message
SetPrice(int g) // store the price of the gas types from argument ‘g’
SetInitialValues() // set L or G and total to zero
PumpGasUnit() // dispose unit of gas (G/L) and count number of unit disposed
GasPumpedMsg() // displays the amount of disposed gas
PrintReceipt() // print a receipt
CancelMsg() // displays a cancellation message
ReturnCash() // returns the remaining cash
SetPayType(t) // Stores pay type t to variable w in the data store
EjectCard() // Card is ejected

State diagram/model of the MDA-EFSM



Pseudo-code of all operations of Input Processors of GP-1 and GP-2

a) Input Processors of GP-1

* m: is a pointer to the MDA-EFSM object
* d: is a pointer to the DataStore object

```
Activate(int a){
    If(a > 0){
            d -> temp_a = a;
            m -> Activate()
    }
}

Start(){
    m -> Start();
}

PayCredit(){
    m -> PayType(1);
}

Reject(){
    m -> Reject();
}

Cancel(){
    m -> Cancel();
}

Approved(){
    m -> Approved();
}
```

```
PayCash(int c){
    If(c > 0){
            d -> temp_c = c;
            m -> PayType(0);
    }
}

Cancel(){
    m -> Cancel();
}

StartPump(){
    m -> StartPump();
}

StopPump(){
    m -> StopPump();
    m-> Receipt();
}

Pump(){
    If ( d->w==1 ){
            m -> Pump();
    }
    Else {
            If( d->cash < d->price * (d-> L+1) ){
                    m -> StopPump();
                    m -> Receipt();
            }
            Else {
                    m -> Pump();
            }
    }
}
```

Notice:
cash: contains the value of cash deposited
price: contains the price of the gas L: contains the number of liters already pumped
w: pay type flag (cash: w=0; credit: 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

b) Input Processors of GP-2

* m: is a pointer to the MDA-EFSM object
* d: is a pointer to the DataStore object

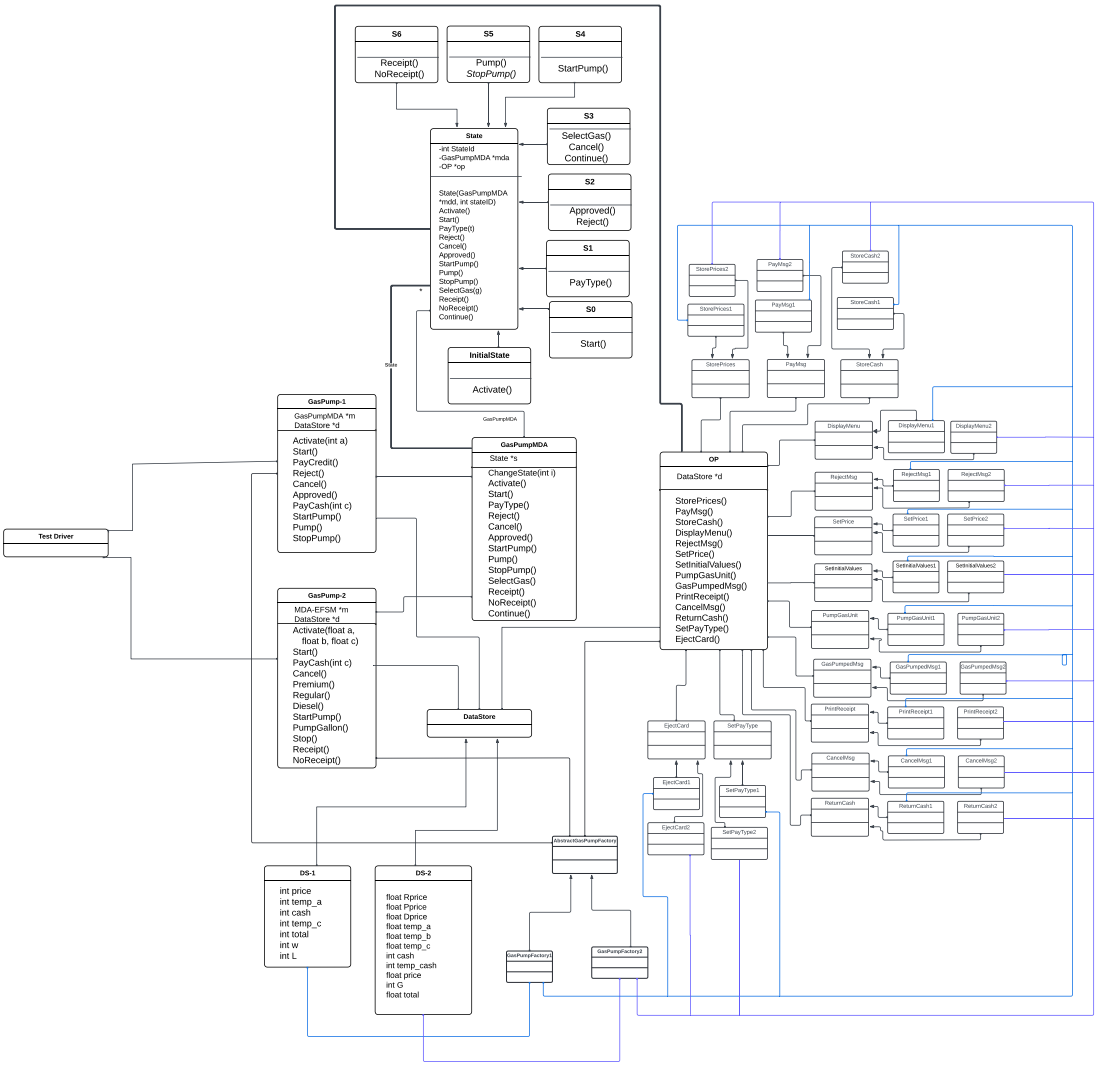
```
Activate(float a, float b, float c){
    If( (a > 0) && (b > 0) && (c > 0) ){
            d -> temp_a = a;
            d -> temp_b = b;
```

```
            d -> temp_c = c;

            m -> Activate()
    }
}

Start(){
    m -> Start();
}

PayCash(int c){
    If (c > 0) {
            d -> temp_c = c;
            m -> PayType(0);
    }
}

Cancel(){
    m -> Cancel();
}

Regular() {
    m -> SelectGasType(1)
    m -> Continue()
}

Diesel() {
    m -> SelectGasType(2)
    m -> Continue()
}

Premium() {
    m -> SelectGasType(3)
    m -> Continue()
}

StartPump(){
    m -> StartPump();
}

PumpGallon(){
    If( d->cash < d->price * (d->G+1) ){
            m -> StopPump();
    }
    Else{
            m -> Pump();
    }
}

Stop(){
    m -> StopPump();
}

Receipt(){
    m -> Receipt();
```

```
}

NoReceipt(){
    m -> NoReceipt();
}
```

Notice:
cash: contains the value of cash deposited
price: contains the price of the selected gas
G: contains the number of Gallons already pumped

2) Class diagram of the MDA of the GP components including:
- State,
- Strategy and
- Abstract Factory patterns in class diagrams

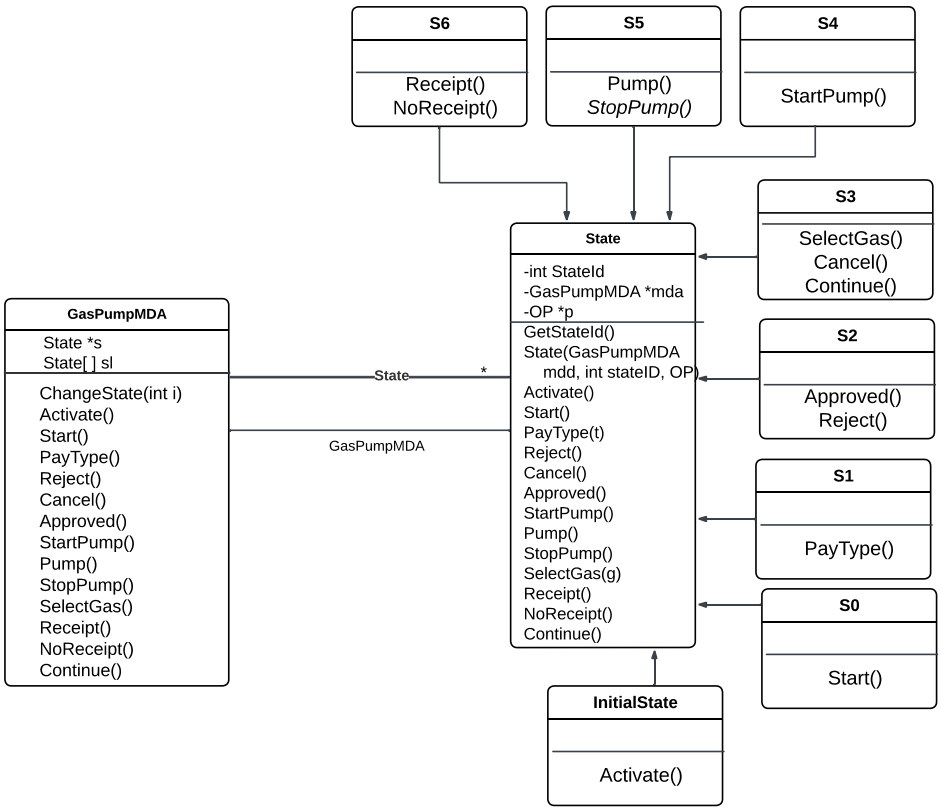


3) Purpose of the the classes with their operations

TestDriver:
This component allows users to select one of the gas pumps. Depending on the selection, either `gasPump1Factory` or `gasPump2Factory` is instantiated. These factories then create objects related to the chosen gas pump. The factories accept user input and forward it to the specific gas pump selected.

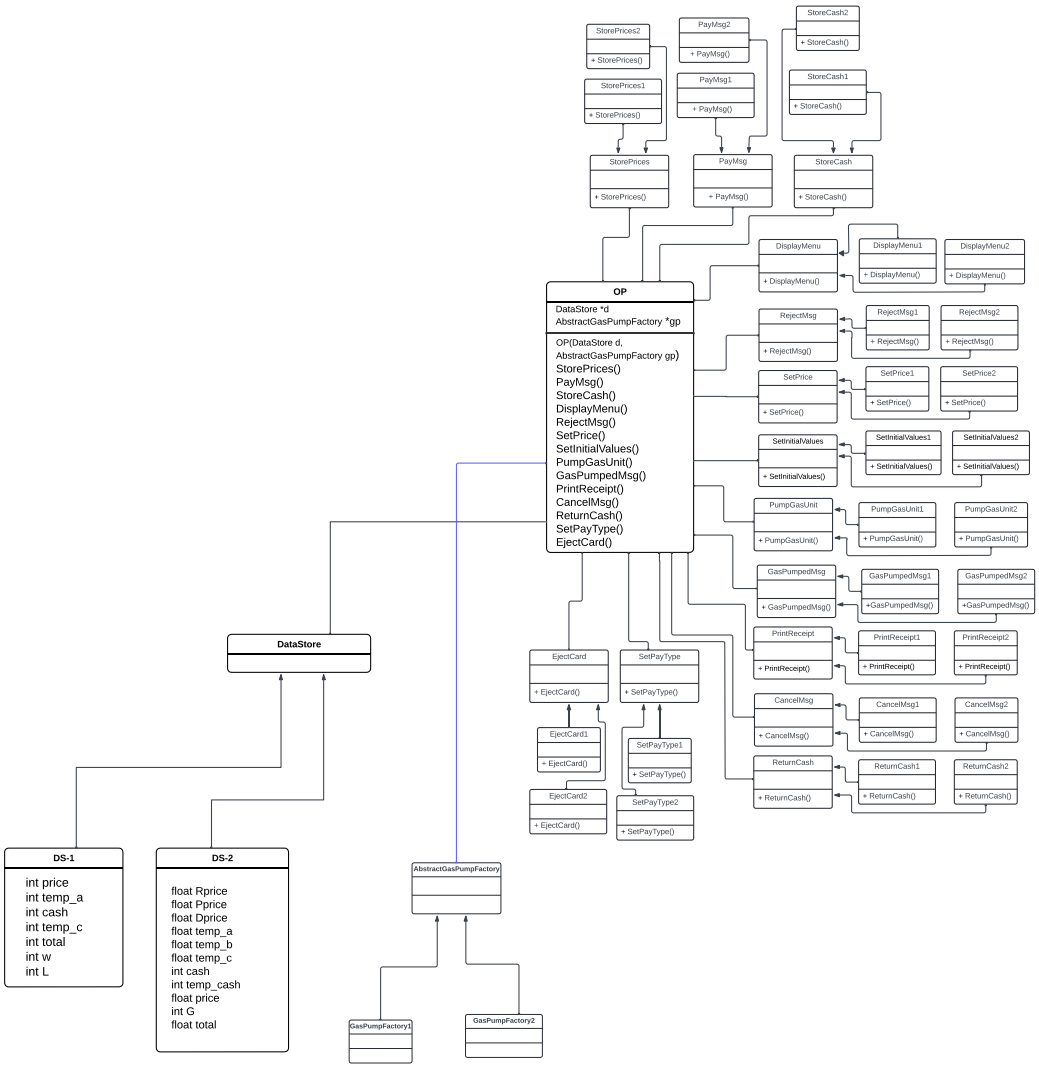
GasPump1: This class contains methods specific to the operations of GasPump1.
GasPump2: This class contains methods specific to the operations of GasPump2.

State Diagram (De-centralized)



Strategy Diagram

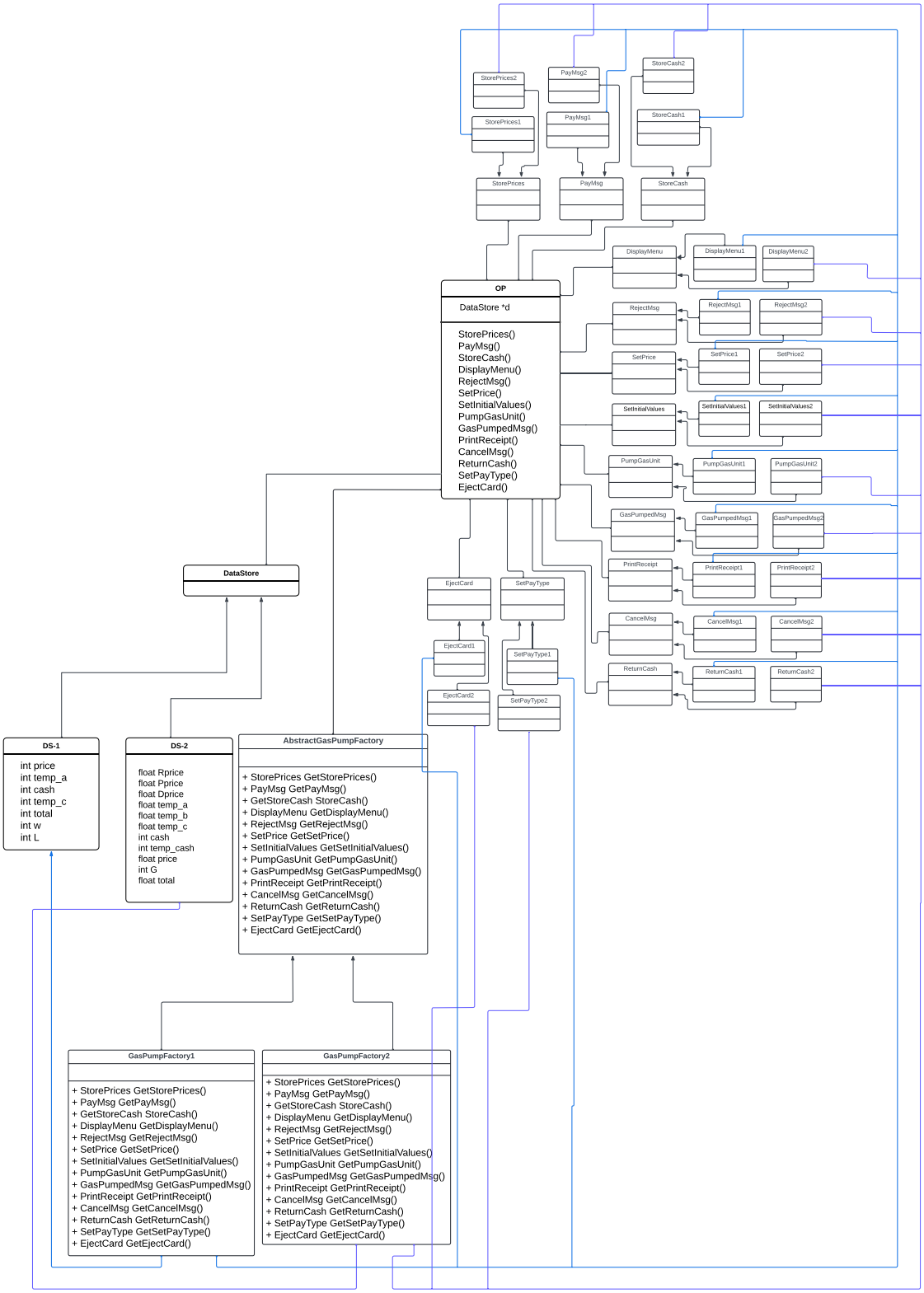
OP is OutputProcessor
The Operations of OutputProcessor class gets the object from the concrete factory(GasPumpFactory1,GasPumpFactory2) and dynamically performs the associated operation in the Strategy Pattern classes based on GasPump1 and GasPump2



Abstract Factory Diagram

The AbstractGasPumpFactory get the various class objects associated with GasPump1 and GasPump2 created.

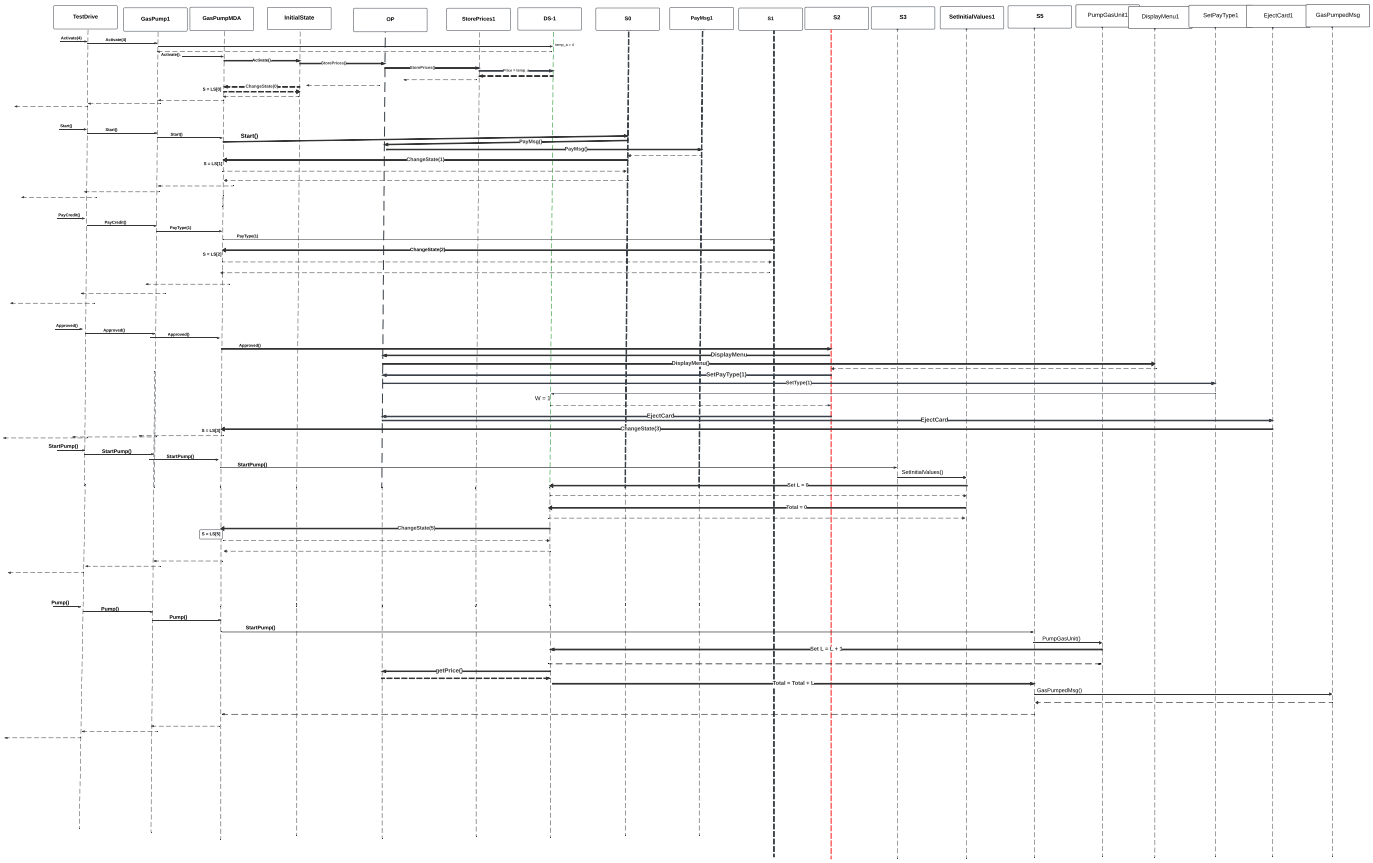
Concrete Factory: This component is responsible for returning instances of the strategy classes associated with the data stores for both GasPump1 and GasPump2.



4)

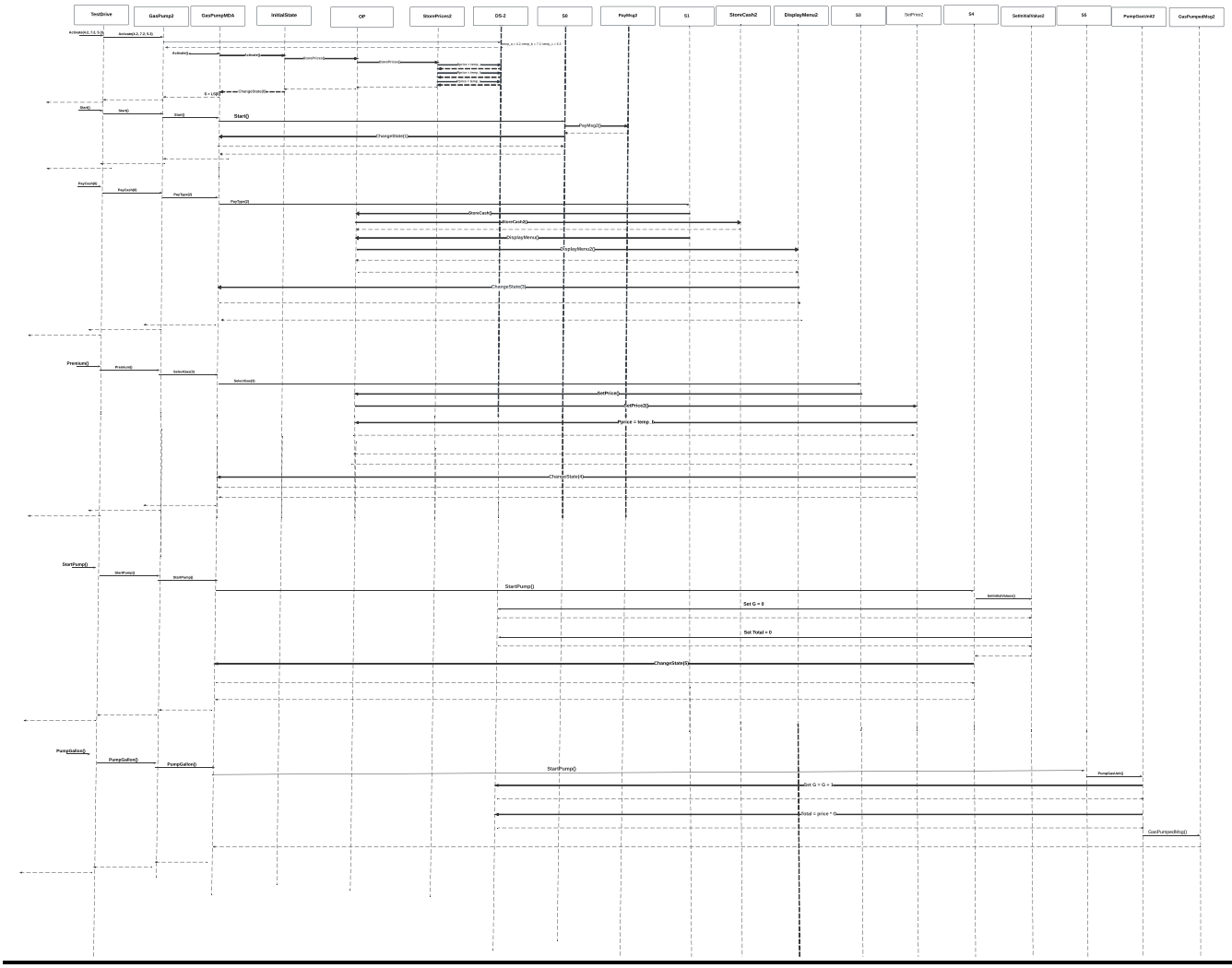
Sequence Diagram-GasPump1

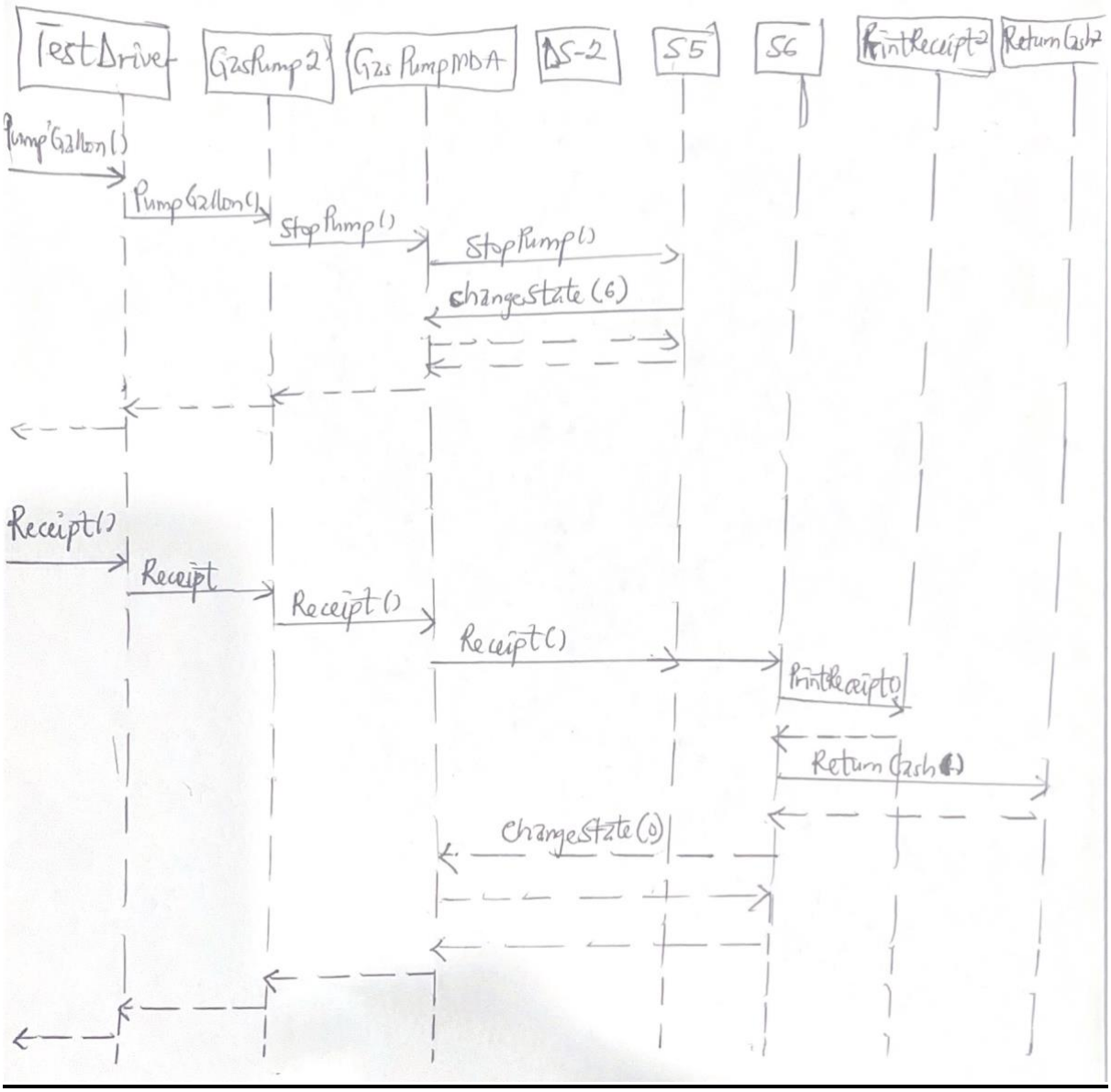
Activate(4), Start(), PayCredit(), Approved(), StartPump(), Pump(), StopPump()



Sequence Diagram-GasPump2

Activate(4.2, 7.2, 5.3), Start(), PayCash(10), Premium(), StartPump(), PumpGallon(), PumpGallon(), Receipt()


TestDrive
GasPump2
GasPumpMDA
InitialState
OP
StorePrices2
DS-2
S0
PayMsg2
S1
StoreCash2
DisplayMenu2
S3
SetPrice2
S4
SetInitialValue2
S5
PumpGasUnit2
GasPumpedMsg2
Activate(4.2, 7.2, 5.3)
Activate(3.2, 7.2, 5.3)
Activate()
StartUp()
StartUp()
Start()
Start()
PayGasMsg()
PayGasMsg()
PayTips()
PayTips()
StoreCash()
StoreCash()
DisplayMenu()
DisplayItems()
ChangeState()
PremiumSelected()
PremiumSelected()
SelectGrade()
SelectGrade()
SetPrice()
SetPrice()
Pprice = temp_b
ChangeState()
StartPump()
StartPump()
StartPump()
StartPump()
Set G = 0
SetInitialValues()
Set Total = 0
ChangeState()
PumpGallons()
PumpGallons()
StartPump()
PumpGasUnit()
Set G = G + 1
Total = price * G
GasPumpedMsg()



II: WELL-DOCUMENTED (COMMENTED) SOURCE CODE

Compressed and submitted to Blackboard

III: PROJECT EXECUTABLES

A runnable jar file uploaded to the blackboard. Steps to run the jar file as below.

1.Copy the .jar file to your desktop
2.Open “Command Prompt” and navigate to your desktop folder

(eg: cd.. cd..

cd users //select the appropriate user
cd Downloads
ls //to make sure that the given jar file is in the desktop)

3.Run the following command to make the jar file running as expected.

```
pkerbynn@pkErbynnMcPro Downloads % java -jar architecture-mda-project.jar
Select type of GasPump:
1. GasPump1
2. GasPump2
```

If the user selects GasPump1 ,menu displays as below:

```
pkerbynn@pkErbynnMcPro Downloads % java -jar architecture-mda-project.jar
Select type of GasPump:
1. GasPump1
2. GasPump2
1
GasPump-1
MENU of Operations:
------------------------
(0) Activate(int a)
(1) Start()
(2) PayCredit()
(3) Reject()
(4) Cancel()
(5) Approved()
(6) PayCash(int)
(7) StartPump()
(8) Pump()
(9) StopPump
(q). Quit the program
------------------------
```

Sample Output for GasPump1

```
Select Operation:
0-Activate,
1-Start,
2-PayCredit,
3-Reject,
4-Cancel,
5-Approved,
6-PayCash,
7-StartPump,
8-Pump,
9-StopPump,
q-quit
```

Then input 0 to activate shows

```
-------------------------
Select Operation:
0-Activate,
1-Start,
2-PayCredit,
3-Reject,
4-Cancel,
5-Approved,
6-PayCash,
7-StartPump,
8-Pump,
9-StopPump,
q-quit
0
  Operation:  Activate(int a)
  Enter value of the parameter a:
4
Prices stored successfully

Current price: 4
Current state: S0
```

Then in that order of output to the sequence;

Activate(4), Start(), PayCredit(), Approved(), StartPump(), Pump(), StopPump()

```
Prices stored successfully

Current price: 4
Current state: S0

Select Operation:
0-Activate,
1-Start,
2-PayCredit,
3-Reject,
4-Cancel,
5-Approved,
6-PayCash,
7-StartPump,
8-Pump,
9-StopPump,
q-quit
1
  Operation:  Start()
Thank you for choosing GasPump-1
Please select payment type:
Current state: S1
```

```
Select Operation:
0-Activate,
1-Start,
2-PayCredit,
3-Reject,
4-Cancel,
5-Approved,
6-PayCash,
7-StartPump,
8-Pump,
9-StopPump,
q-quit
2
  Operation:  PayCredit()
Current state: S2

Select Operation:
0-Activate,
1-Start,
2-PayCredit,
3-Reject,
4-Cancel,
5-Approved,
6-PayCash,
7-StartPump,
8-Pump,
9-StopPump,
q-quit
5
  Operation:  Approved()
CREDIT CARD APPROVED
```

```
Current state: S3

Select Operation:
0-Activate,
1-Start,
2-PayCredit,
3-Reject,
4-Cancel,
5-Approved,
6-PayCash,
7-StartPump,
8-Pump,
9-StopPump,
q-quit
7
  Operation:  StartPump()
Current state: S4
```

```
Select Operation:
0-Activate,
1-Start,
2-PayCredit,
3-Reject,
4-Cancel,
5-Approved,
6-PayCash,
7-StartPump,
8-Pump,
9-StopPump,
q-quit
8
  Operation: Pump()
Current later PRICE: 4
Current price: 4
Select Operation:
0-Activate,
1-Start,
2-PayCredit,
3-Reject,
4-Cancel,
5-Approved,
6-PayCash,
7-StartPump,
8-Pump,
9-StopPump,
q-quit
9
  Operation:  StopPump()
```

Final output of total per the sequence is expected to be **4**

```
q quit
9
  Operation:  StopPump()
Printing receipt ...
**********************************************************************
Total: $4.0
**********************************************************************
Transaction complete
```