1. MDA-EFSM model for the Gas Pump Components:

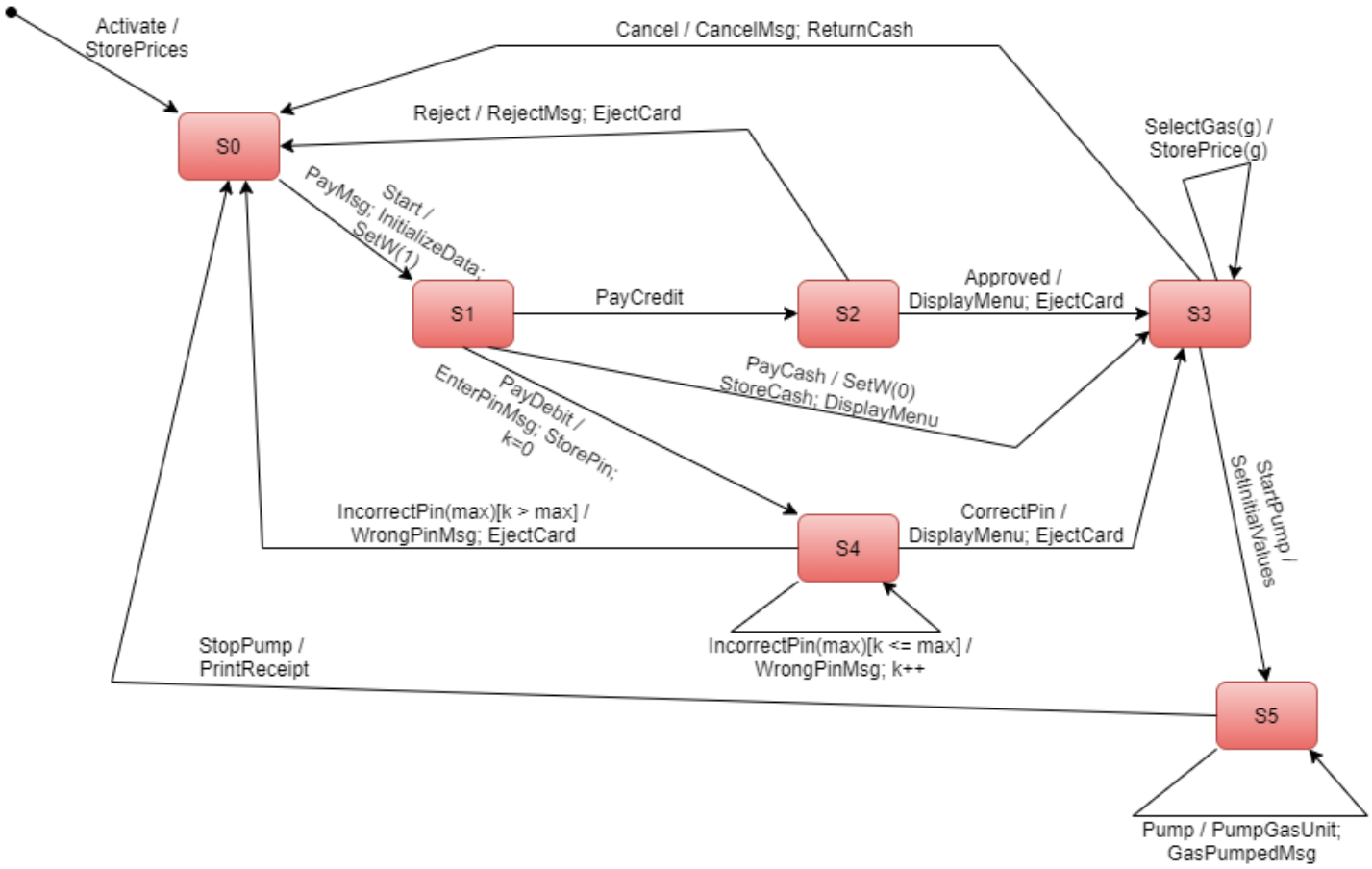
List of meta events for the MDA-EFSM:

Activate()
Start()
PayCredit()
PayCash()
PayDebit()
Reject()
Cancel()
Approved()
StartPump()
Pump()
StopPump()
SelectGas(int g)
CorrectPin()
IncorrectPin(int max)

List of meta actions for the MDA-EFSM:

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

State Diagram of the MDA-EFSM:



Operations of the Input Processor (GasPump-1)

```
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

Operations of the Input Processor (GasPump-2)

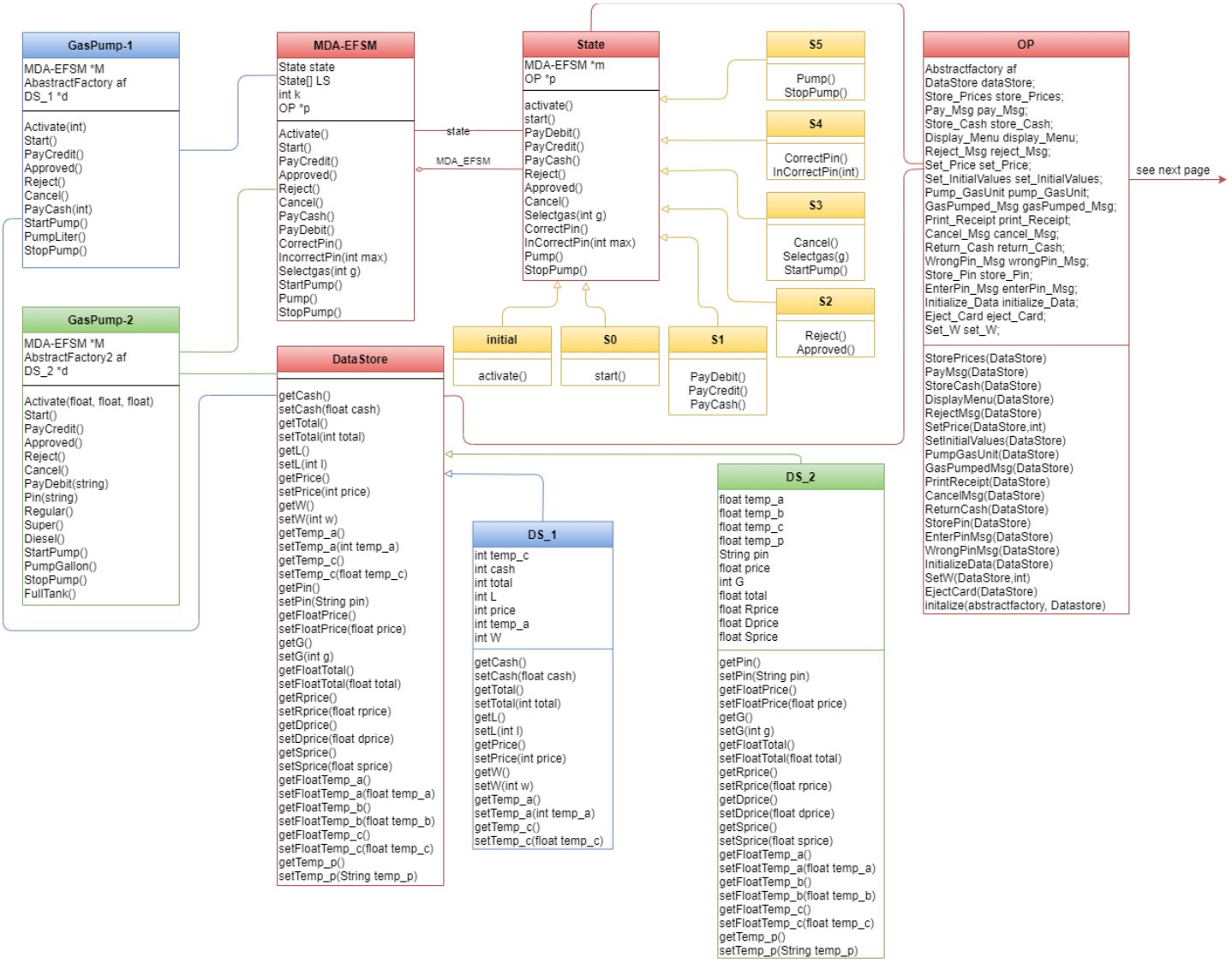
```
Activate(float a, float b)
 {
        if ((a>0)&&(b>0)&&(c>0))
        {
                d->temp_a=a;
                d->temp_b=b;
                d->temp_c=c;
                m->Activate()
        }
}

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();
}
```

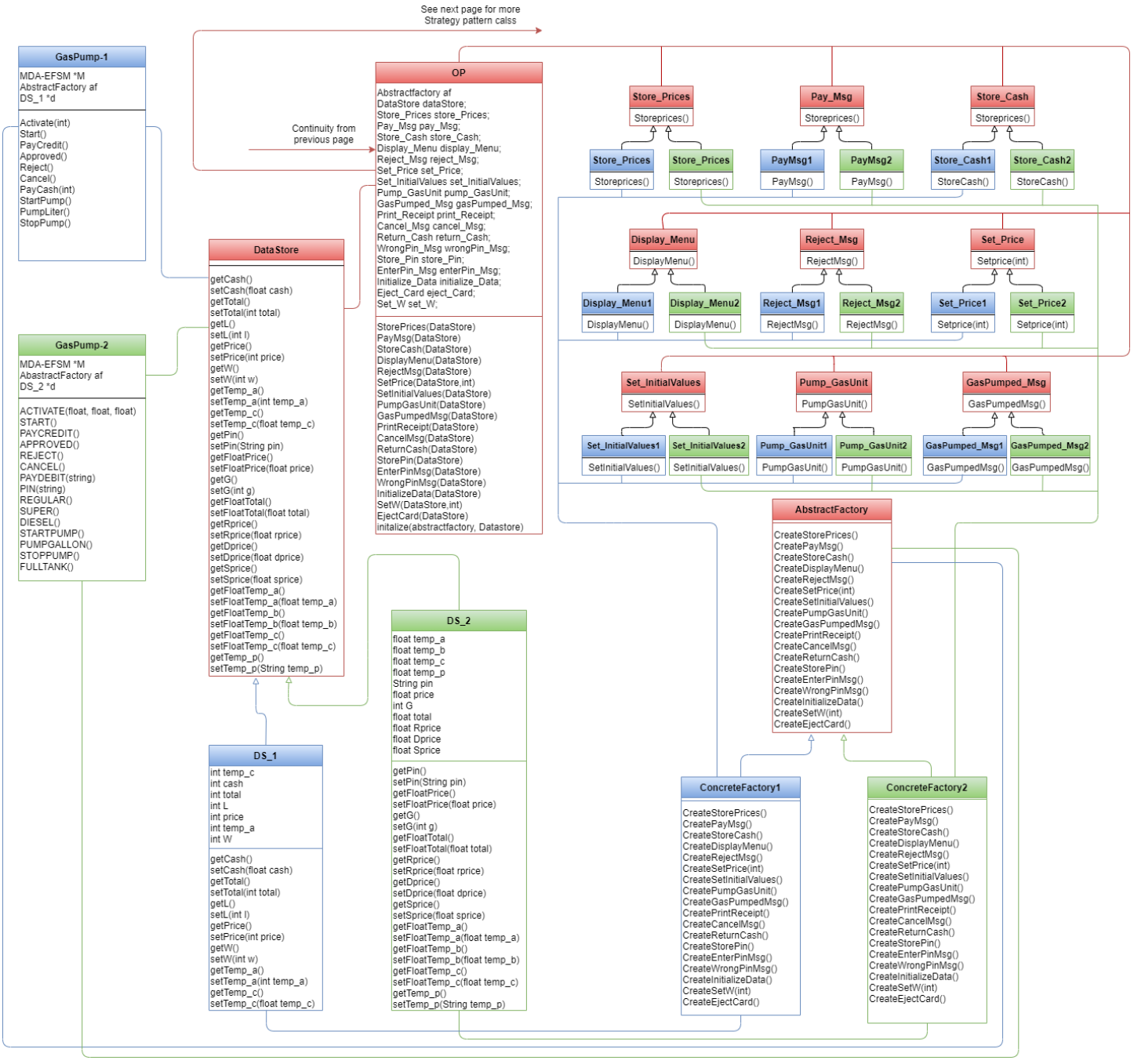
Notice:
pin: contains the pin in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object
SelectGas(g): Regular: g=1; Super: g=2; Diesel: g=3

2. Class Diagram(s) of the MDA of the Gas Pump Components:

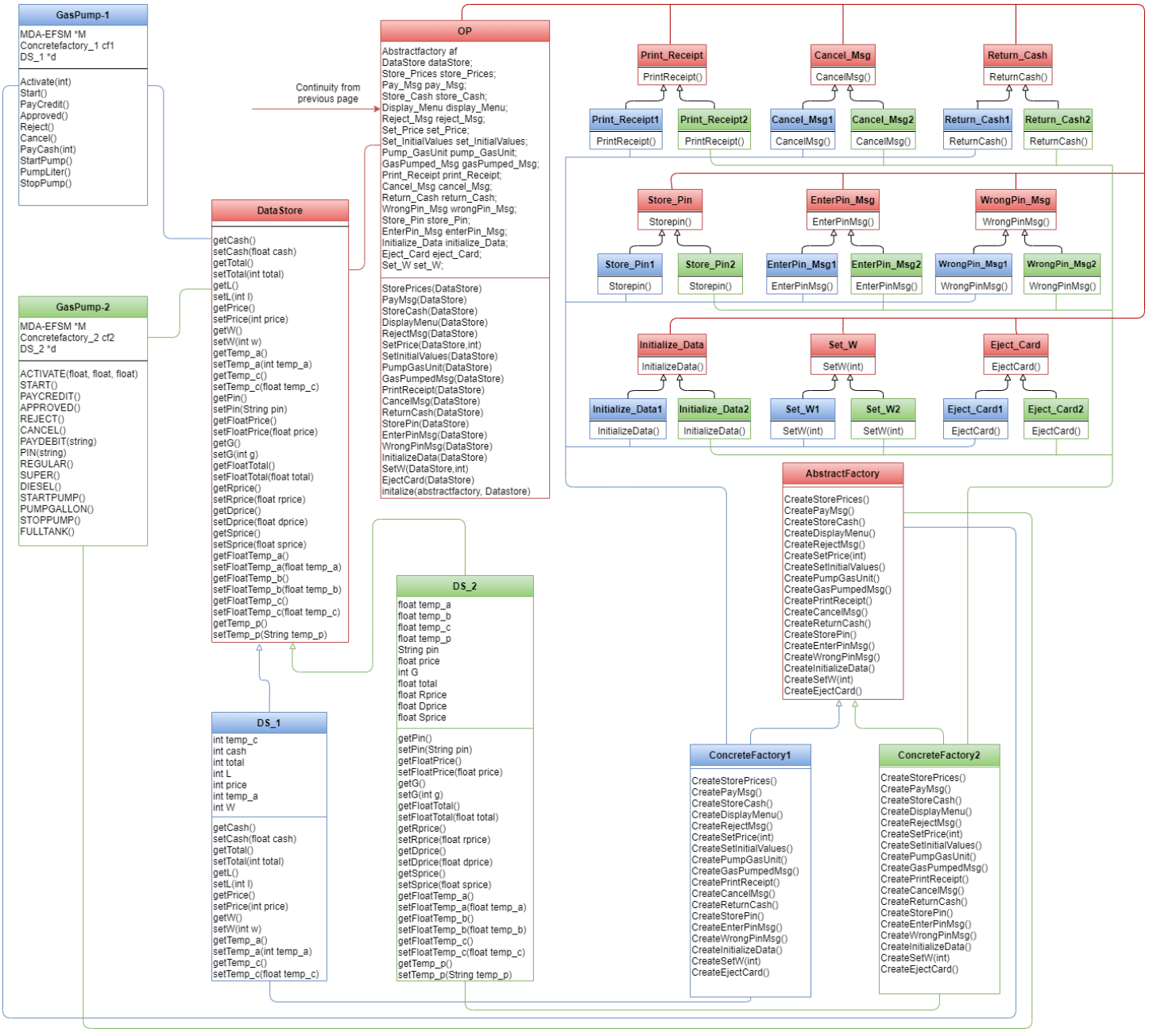
State Pattern



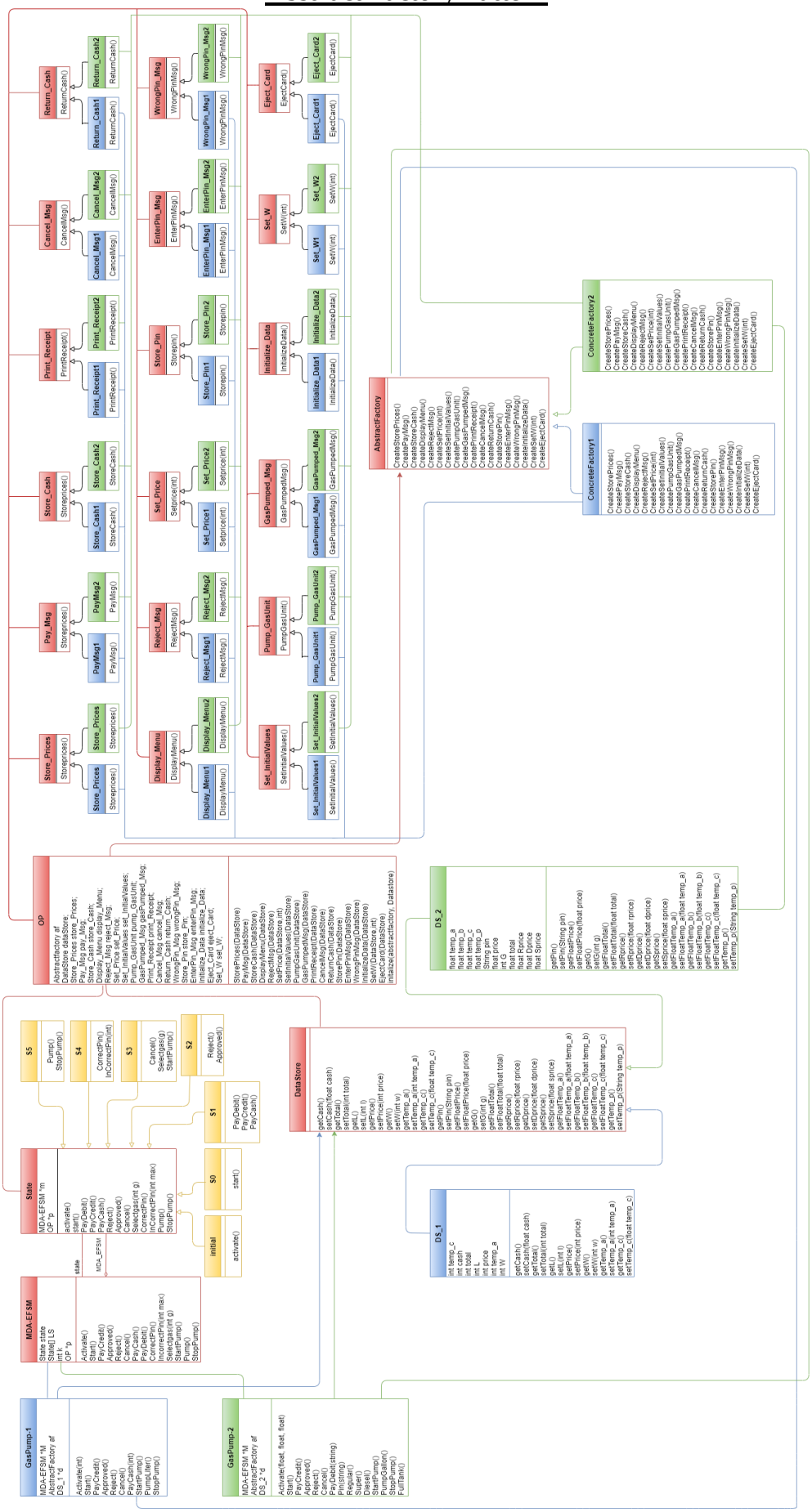
Strategy Pattern



More Strategy Patterns:



Abstract Factory Pattern



3. Purpose and Responsibility of Each class:

Gaspump_1
This class will call all the operation of GasPump-1

Class Variables:
DS_1 d1 //object of Data store 1
AbstractFactory af; // object of Abstract Factory
MDA_EFSM m; //object of MDA_EFSM

Operations:
- **GasPump_1() :**
  Creates an object of DS_1, ConcreteFactory1(), MDAEFSM() and return its address. It also calls the initialize() function of MDA_EFSM to set the datastore and abstract factory
- **Activate(int a) :**
  Check the activate method's input parameter and if its positive and greater than zero than it calls the EFSM meta event activate() otherwise shows an error message.
  It stores the price of Gas.
- **Start()** :
  Call EFSM-model's meta event start().
- **PayCredit() :**
  Call EFSM-model's meta event PayCredit().
- **Reject() :**
  Call EFSM-model's meta event Reject().
- **Cancel() :**
  Call EFSM-model's meta event Cancel().
- **Approved() :**
  Call EFSM-model's meta event Approved().
- **PayCash(float c) :**
  Check the PayCash method's input parameter and if its positive and greater than zero than it calls the EFSM meta event PayCash() otherwise shows an error message.
  It also stores the cash price.
- **StartPump() :**
  Call EFSM-model's meta event StartPump().
- **PumpLiter()** :
  Checks the payment method if the payment method is of card type then it will call EFSM model's meta event pump(). If the payment method is of cash type, then it will first check if the cash is enough to pay for 1 unit of selected gas type then it will call EFSM model's meta event pump() otherwise it will call EFSM model's meta event StopPump()
- **StopPump() :**
  Call EFSM-model's meta event StopPump().
- **print_Operations()** :
  It just prints all the operations available for gaspump_1.

Gaspump_2
This class call all the operation of GasPump-2.

Class Variables:
DS_2 d2 //object of Data store 2
AbstractFactory af; // object of Abstract Factory
MDA_EFSM m; //object of MDA_EFSM

Operations:
- **GasPump_2() :**
  Creates an object of DS_2, ConcreteFactory2(), MDAEFSM() and return its address. It also calls the initialize() function

of MDA_EFSM to set the datastore and abstract factory

- **Activate(float a,float b, float c) :**
  Check the activate method's input parameter and if its positive and greater than zero than it calls the EFSM meta event activate() otherwise shows an error message.
  It stores price of Diesel, Regular and Super Gas.
- **Start()** :
  Call EFSM-model's meta event start().
- **PayCredit() :**
  Call EFSM-model's meta event PayCredit().
- **Reject() :**
  Call EFSM-model's meta event Reject().
- **Cancel() :**
  Call EFSM-model's meta event Cancel().
- **Approved() :**
  Call EFSM-model's meta event Approved().
- **PayDebit(String p) :**
  Call the EFSM meta event PayDebit().
  It stores the pin of debit card.
- **Pin(String x) :**
  Match the value input parameter with the stored pin value if both of them matches then it will Call the EFSM meta event CorrectPin() else it will call the EFSM meta event InCorrectPin()
- **Diesel()** :
  Call EFSM-model's meta event selectGas() with 3 as a parameter.
- **Regular()** :
  Call EFSM-model's meta event selectGas() with 1 as a parameter.
- **Super() :**
  Call EFSM-model's meta event selectGas() with 2 as a parameter.
- **StartPump() :**
  Check the value of price if its positive and greater than zero than it calls the EFSM meta event StartPump()
- **PumpGallon()** :
  Call EFSM-model's meta event Pump().
- **StopPump() :**
  Call EFSM-model's meta event StopPump().
- **Fulltank() :**
  Call EFSM-model's meta event StopPump().
- **print_Operations()** :
  It just prints all the operations available for gaspump_1.

MDA_EFSM

This MDA-EFSM class connect with decentralized State design pattern.

Class Variables:

```
OP p            // object of OP
Int k           // number of attempts to insert the pin for debit card payment type
State LS[]      // Array to represent all the State objects Available for the system
                // LS[0] = object of initial state class
                // LS[1] = object of S0 state class
                // LS[2] = object of S1 state class
                // LS[3] = object of S2 state class
                // LS[4] = object of S3 state class
                // LS[5] = object of S4 state class
                // LS[6] = object of S5 state 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 GasPump_1, GasPump_2 and State class

Operations:
- **Activate() :**
  It calls the Activate() method of state class.
- **Start**() :
  It calls the Start() method of state class.
- **payCredit**() :
  It calls the PayCredit() method of state class.
- **payCash**() :
  It calls the PayCash() method of state class.
- **payDebit**() :
  It calls the PayDebit() method of state class.
- **CorrectPin**() :
  It calls the Correct() method of state class.
- **InCorrectPin**(int i) :
  It calls the InCorrect() method of state class with i as a input parameter which represents the maximum number of attempts allowed to Enter the pin for authentication purpose when user select Debit card payment type.
- **Reject**() :
  It calls the Reject() method of state class
- **Cancel**() :
  It calls the Cancel() method of state class.
- **approved**() :
  It calls the Approved() method of state class.
- **StartPump**() :
  It calls the StartPump() method of state class.
- **Pump**() :
  It calls the Pump() method of state class
- **StopPump**() :
  It calls the StopPump() method of state class.
- **SelectGas**(int i) :
  It calls the SelectGas() method of state class with i as a input parameter which represent the type of gas selected by the user.(1- Regular Gas, 2-Super Gas, 3-Diesel gas)
- **change_state(int a):**
  This method will set the state pointer S to a value, a value will be 0-initial , 1-S0, 2-S1, 3-S2, 4-S3, 5-S4, 6-S5 states
- **setk**() :
  Set the value of k which represents the number of attempts to enter the pin when selected Debit card as a Payment type.
- **getk**() :
  Returns the value of k which represents the number of attempts to enter the pin when selected Debit card as a Payment type.
- **initialize(AbstractFactory af, DataStore d)**
  This method is called at the point of initialization of Gas Pump, 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 method takes object of abstract factory and data store which was passed as per the version of Gas Pump.
- **getObject() :**
  This method will return current MDA_EFSM object

Abstractfactory Class
- This class is an abstract factory class that will help to generate required objects for different types of Gas Pumps
- All methods are abstract here
- This class is connected to GasPump_1, GasPump_2 and OP
- the child classes of this class are connected to Data Store child class as well

Operations:
public abstract Store_Prices CreateStorePrices();
public abstract Pay_Msg CreatePayMsg();
public abstract Store_Cash CreateStoreCash();
public abstract Display_Menu CreateDisplayMenu();
public abstract Reject_Msg CreateRejectMsg();
public abstract Set_Price CreateSetPrice();
public abstract Set_InitialValues CreateSetInitialValues();
public abstract Pump_GasUnit CreatePumpGasUnit();
public abstract GasPumped_Msg CreateGasPumpedMsg();
public abstract Print_Receipt CreatePrintReceipt();
public abstract Cancel_Msg CreateCancelMsg();
public abstract Return_Cash CreateReturnCash();
public abstract Store_Pin CreateStorePin();
public abstract EnterPin_Msg CreateEnterPinMsg();
public abstract WrongPin_Msg CreateWrongPinMsg();
public abstract Initialize_Data CreateInitializeData();
public abstract Set_W CreateSetW();
public abstract Eject_Card CreateEjectCard();

Concretefactory_1
This class uses abstract class AbstractFactory to return Gas pump specific components.
This class is providing driver object for gaspump1 and represent action strategy with data class for gaspump_1.

Operations
- **CreateStorePrices()**
  Create and returns object of storePrices_1 class that will store data as per user input for GasPump_1
- **CreatePayMsg()**
  Create and returns object of Pay_Msg1 class that will display different payment methods available for GasPump_1
- **CreateStoreCash()**
  Create and returns object of Store_Cash1 class that will store cash for GasPump_1
- **CreateDisplayMenu()**
  Create and returns object of Display_Menu1 class to display menu available for GasPump_1
- **CreateRejectMsg()**
  Create and returns object of Reject_Msg1 class to display rejected message of credit card transaction for GasPump_1
- **CreateSetPrice()**
  This will not return anything as we don't have any option to select gas type for GasPump_1
- **CreateSetInitialValues()**
  Creates object of Set_InitialValues1 class that will set the total price and number of liters pumped to 0 for GasPump_1
- **CreatePumpGasUnit()**
  Create and returns object of Pump_GasUnit1 class that returns total unit of gas pumped for GasPump_1
- **CreateGasPumpedMsg()**
  Creates and return object of GasPumped_Msg1 class that displays number of liters gas pumped
- **CreatePrintReceipt()**
  Create and return object of class Print_Receipt1 that will display the total bill of the transaction for GasPump_1
- **CreateCancelMsg()**
  Create and returns the object of Cancel_Msg1 class that will cancel the current transaction and displays the same for • GasPump_1
- **CreateReturnCash()**
  Create and returns the object of class Return_Cash1 that will Return the cash in case when user will select the cancel option for GasPump_1
- **CreateWrongPinMsg()**
  This will not return anything as we don't have debit card payment option for GasPump_1
- **CreateStorePin()**

This will not return anything as we don't have debit card payment option for GasPump_1

- **CreateEnterPinMsg()**
  This will not return anything as we don't have debit card payment option for GasPump_1
- **CreateInitializeData()**
  This will not return anything as we don't have any option to select gas type for GasPump_1
- **CreateSetW()**
  Creates and return object of Set_W1 class that is set to 1 for card payments and set to 0 for Cash payment
- **CreateEjectCard()**
  Creates and return object of Eject_Card1 class that will Display Eject Card message for any card transaction for GasPump_1

Concretefactory_2

This class uses abstract class AbstractFactory to return Gas pump specific components.
This class is providing driver object for gaspump1 and represent action strategy with data class for gaspump_2.

Operations

- **CreateStorePrices()**
  Create and returns object of storePrices_2 class that will store data as per user input for GasPump_2
- **CreatePayMsg()**
  Create and returns object of Pay_Msg2 class that will display different payment methods available for GasPump_2
- **CreateStoreCash()**
  this will not return anything as we don't have option of cash payment for GasPump_2
- **CreateDisplayMenu()**
  Create and returns object of Display_Menu2 class that will display different kinds of gas types available for GasPump_2 i.e., Diesel, Regular and Super gas
- **CreateRejectMsg()**
  Create and returns object of Reject_Msg2 class to display rejected message of credit card transaction for GasPump_2
- **CreateSetPrice()**
  Create and returns object of Set_Price2 class that will set the value of price based on the gas type selected by the user for GasPump_2
- **CreateSetInitialValues()**
  Creates object of Set_InitialValues2 class that will set the total price and number of Gallons pumped to 0 for GasPump_2
- **CreatePumpGasUnit()**
  Create and returns object of Pump_GasUnit2 class that returns total unit of gas pumped for GasPump_2
- **CreateGasPumpedMsg()**
  Creates and return object of GasPumped_Msg2 class that displays number of gallons gas pumped for GasPump_2
- **CreatePrintReceipt()**
  Create and return object of class Print_Receipt2 class that will display the total bill of the transaction along with number of gallons gas pumped for GasPump_2
- **CreateCancelMsg()**
  Create and returns the object of Cancel_Msg2 class that will cancel the current transaction and displays the same for GasPump_2
- **CreateReturnCash()**
  this will not return anything as we don't have cash payment option for GasPump_2
- **CreateWrongPinMsg()**
  Creates and return object of class WrongPin_Msg2 to display the wrong pin message in case when user selects debit card payment option for GasPump_2
- **CreateStorePin()**
  Creates and return object of Store_Pin2 class that will store the pin in case of debit card payment for GasPump_2
- **CreateEnterPinMsg()**
  Creates and return object of class EnterPin_Msg2 that will ask the user to enter the pin for authentication for GasPump_2
- **CreateInitializeData()**
  Creates and return object of class Initialize_Data2 that will set the value of price to 0

- **CreateSetW()**
  //this will not return anything as we don't have cash payment option for GasPump_2
- **CreateEjectCard()**
  Creates and return object of Eject_Card2 class that will Display Eject Card message for any card transaction for GasPump_2

DataStore:
DataStore class is used to store data. It is a parent DataStore class.
It will have list of all methods that are defined in both data stores.
It provides just the null definition but will not have an impact.
This class is connected to GasPump_1, GasPump_2 and OP
The child DataStore Classes are connected to Child abstract factory classes according to their version.

Operations:
- **getCash()**
  This method is just parent definition for child class, it does nothing here, but it will get value of cash variable, return type float
- **setCash(float cash)**
  This method is just parent definition for child class, it does nothing here, but it will set value of cash variables, type float
- **getTotal()**
  This method is just parent definition for child class, it does nothing here, but it will get value of total variables, return type int
- **setTotal(int total)**
  this method is just parent definition for child class, it does nothing here, but it will set value of total variables, type int
- **getL()**
  this method is just parent definition for child class, it does nothing here, but it will get value of L variable(i.e, number ot liters pumped), return type int
- **setL(int l)**
  this method is just parent definition for child class, it does nothing here, but it will set value of L variableL variable(i.e, number to liters pumped), type 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
- **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
- **getW()**
  this method is just parent definition for child class, it does nothing here, but it will get value of W variable(i.e, to check the payment type is cash or card), return type int
- **setW(int w)**
  this method is just parent definition for child class, it does nothing here, but it will set value of W variable(i.e, to check the payment type is cash or card), type int
- **getTemp_a()**
  This method is just parent definition for child class, it does nothing here, but it will get value of temp_a variables, return type int
- **setTemp_a(int temp_a)**
  this method is just parent definition for child class, it does nothing here, but it will set value of temp_a variables, type int
- **getTemp_c()**
  this method is just parent definition for child class, it does nothing here, but it will get value of temp_c variables, return type float
- **setTemp_c(float temp_a)**
  this method is just parent definition for child class, it does nothing here, but it will set value of temp_c variables, type float
- **getPin()**
  this method is just parent definition for child class, it does nothing here, but it will get value of Pin variables, return

type String

- **setPin(String pin)**
  this method is just parent definition for child class, it does nothing here, but it will set value of Pin variables, type String
- **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
- **setFloatPrice(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
- **getG()**
  this method is just parent definition for child class, it does nothing here, but it will get value of G variable(i.e, number to Gallons pumped), return type int
- **setG(int g)**
  this method is just parent definition for child class, it does nothing here, but it will set value of G variable(i.e, number ot Gallons pumped), type int
- **getFloatTotal()**
  this method is just parent definition for child class, it does nothing here, but it will get value of total variables, return type float
- **setFloatTotal(float total)**
  this method is just parent definition for child class, it does nothing here, but it will set value of total variables, type float
- **getRprice()**
  this method is just parent definition for child class, it does nothing here, but it will get value of RPrice variable, return type float
- **setRprice(float rprice)**
  this method is just parent definition for child class, it does nothing here, but it will set value of RPrice variables, type float
- **getDprice()**
  this method is just parent definition for child class, it does nothing here, but it will get value of DPrice variables, return type float
- **setDprice(float dprice)**
  this method is just parent definition for child class, it does nothing here, but it will set value of DPrice variables, type float
- **getSprice()**
  this method is just parent definition for child class, it does nothing here, but it will get value of SPrice variables, return type float
- **setSprice(float sprice)**
  this method is just parent definition for child class, it does nothing here, but it will set value of SPrice variables, type float
- **getFloatTemp_a()**
  this method is just parent definition for child class, it does nothing here, but it will get value of temp_a variables, return type float
- **setFloatTemp_a(float temp_a)**
  this method is just parent definition for child class, it does nothing here, but it will set value of temp_a variables, type float
- **getFloatTemp_b()**
  this method is just parent definition for child class, it does nothing here, but it will get value of temp_b variables, return type float
- **setFloatTemp_b(float temp_b)**
  this method is just parent definition for child class, it does nothing here, but it will set value of temp_b variables, type float
- **getFloatTemp_c()**
  this method is just parent definition for child class, it does nothing here, but it will get value of temp_c variables, return type int
- **setFloatTemp_c(float temp_c)**

this method is just parent definition for child class, it does nothing here, but it will set value of temp_c variables, type float
- **String 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 String
- **setTemp_p(String 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 String

DS_1:
This class contains data variable for GasPump_1 and the Getter Setter method for each.

Class Variables:
float cash
total
int L
int price
int W

//temporary variables
int temp_a
float temp_c

Operations:
- **getCash()**
get value of cash variable, return type float
- **setCash(float cash)**
set value of cash variables, type float
- **getTotal()**
get value of total variable, return type int
- **setTotal(int total)**
set value of total variables, type int
- **getL()**
get value of L variable, return type int
- **setL(int l)**
set value of L variables, type int
- **getPrice()**
get value of price variable, return type int
- **setPrice(int price)**
set value of price variables, type int
- **getW()**
get value of W variable, return type int
- **setW(int w)**
set value of W variables, type int
- **getTemp_a()**
get value of temp_a variable, return type int
- **setTemp_a(int temp_a)**
set value of temp_a variables, type int
- **getTemp_c()**
get value of temp_c variable, return type float
- **setTemp_c(float temp_c)**
set value of temp_c variables, type float

DS_2:
This class contains data variable for GasPump_2 and the Getter Setter methods for each.

Class Variables:
String pin
float price
int G
float total
float Rprice
float Dprice
float Sprice

//temporary variables
float temp_a
float temp_b
float temp_c
String temp_p

Operations:
• **getPin()**
get value of pin variable, return type String
• **setPin(String pin)**
set value of pin variables, type String
• **getFloatPrice()**
get value of price variable, return type float
• **setFloatPrice(float price)**
set value of price variables, type float
• **getG()**
get value of G variable, return type int
• **setG(int g)**
set value of G variables, type int
• **getFloatTotal()**
get value of total variable, return type float
• **setFloatTotal(float total)**
set value of total variables, type float
• **getRprice()**
get value of Rprice variable, return type float
• **setRprice(float rprice)**
set value of Rprice variables, type float
• **getDprice()**
get value of Dprice variable, return type float
• **setDprice(float dprice)**
set value of dprice variables, type float
• **getSprice()**
get value of Sprice variable, return type float
• **setSprice(float sprice)**
set value of Sprice variables, type float
• **getFloatTemp_a()**
get value of temp_a variable, return type float
• **setFloatTemp_a(float temp_a)**
set value of temp_a variables, type float
• **getFloatTemp_b()**
get value of temp_b variable, return type float
• **setFloatTemp_b(float temp_b)**
set value of temp_b variables, type float
• **getFloatTemp_c()**
get value of temp_c variable, return type float

• **setFloatTemp_c(float temp_c)**
set value of temp_c variables, type float
• **getTemp_p()**
get value of temp_p variable, return type String
• **setTemp_p(String temp_p)**
set value of temp_p variables, type String

State
Generalized parent class to represent states for the GasPump system
It has each state as child class and Follows "State Pattern" strategy with Decentralized approach
This Class is connected with MDA_EFSM and OP Class
This do not have any action to perform just used for generalization
Each method that are in each state is defined here with just a print statement as body
Here constructor will get MDA EFSM object for the actions in state class

Initial
This is a child class of State Class that implements Activate() method.
• **Activate() :**
this method calls StorePrices() meta action from Output Processer(i.e., OP) and change state to 1.

S0
This is a child class of State Class that implements Start() method.
• **Start() :**
this method calls PayMsg(), InitializeData(), SetW() meta actions from Output Processer(i.e., OP) and change state to 2.

S1
This is a child class of State Class that implements PayCredit(), PayDebit() and PayCash() methods.
• **PayCredit() :**
this method simply change state to 3.
• **PayDebit() :**
this method calls StoreCash(), DisplayMenu(), SetW() meta actions from Output Processer(i.e., OP) and change state to 4.
• **PayCash() :**
this method calls EnterPinMsg() and StorePin() meta actions from Output Processer(i.e., OP), reset the value of number of attempts to enter pin(k) to 0 and change state to 5.

S2
This is a child class of State Class that implements Reject() and Approved() methods.
• **Reject() :**
This method calls RejectMsg() and EjectCard() meta actions from Output Processer(i.e., OP) and change state to 1.
• **Approved()** :
this method calls DisplayMenu and EjectCard() meta actions from Output Processer(i.e., OP) and change state to 4.

S3
This is a child class of State Class that implements SelectGas(int g), Cancel() and StartPump() methods.
• **SelectGas(int g)** :
This method calls SetPrice(g) meta action from Output Processer(i.e., OP) and will remain in the same state.
• **Cancel()** :
This method calls CancelMsg() and ReturnCash meta action from Output Processer(i.e., OP) and change state to 1.
• **StartPump()** :
This method calls SetInitialvalues() meta action from Output Processer(i.e., OP) and change state to 6.

S4
This is a child class of State Class that implements CorrectPin() and InCorrectPin(int max) methods.
• **CorrectPin() :**

This method calls DisplayMenu() and EjectCard() meta actions from Output Processer(i.e., OP) and change state to 4.
• **InCorrectPin(int max)** :
this method first check the value of k (i.e, number of attempts to enter pin) if the value is less than or equals to the max variable then it will call WrongPinMSg() meta action from Output Processer(i.e., OP) and will remain in the same state otherwise it will call WrongPinMSg() and EjectCard() meta actions from Output Processer(i.e., OP) and change state to 1

S5
This is a child class of State Class that implements Pump() and StartPump(int max) methods
• **Pump()** :
This method calls PumpGasUnit and GasPumpedMsg meta action from Output Processer(i.e., OP) and will remain in the same state.
• **stopPump()** :
This method calls PrintReceipt() meta action and change state to 1.

OP
This class acts as the "Client" class in the strategy design pattern
Each Action in this class calls the platform specific implementation of the action.
Implemented meta action based on strategy pattern.

Operations:
• 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
• 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 cash
• WrongPinMsg() // displays a wrong pin message
• StorePin() // store price from the temporary data store
• EnterPinMsg()// displays Enter pin message
• InitializeData() // it reset the value of price
• SetW(int i) // it set the payment type(0 for cash payment or 1 for card payment)
• EjectCard() // displays Eject card message

Store_Prices
This class is the parent abstract class for Store Prices action strategy.
The classes implementing this class will be having definition of the class
This class have one abstract method to perform the action that will take a datastore object as an argument
• StorePrices(DataStore d)
Abstract method which will be override by subclass to store prices for the different gas types depending on the what types of gases are available on different GasPumps.

Store_Prices1
This is the child class that will implement the StorePrices action of parent class for GasPump_1. This stores price of types of gasoline in DS_1 class for future use.
• StorePrices(DataStore d)
This will set the value of price using DS_1 class as we have only one type of gas available for GasPump_1

Store_Prices2

This is the child class that will implement the StorePrices action of parent class for GasPump_2. This stores price of types of gasoline in DS_2 class for future use.
• StorePrices(DataStore d)
This will store the prices for diesel, regular and super gasoline in DS_2 class.

Pay_Msg
This class is the parent abstract class for Payment Message action strategy.
The classes implementing this class will be having definition of the class
This class have one abstract method to perform the action that will take a datastore object as an argument
• PayMsg(DataStore d)
abstract method which will be override by subclass to display different Payment options depending on the GasPumps.

Pay_Msg1
This is the child class that will implement the PayMsg action of parent class for GasPump_1.
• PayMsg (DataStore d)
This will display the two payment methods that are Pay cash and Pay by credit card

Pay_Msg2
This is the child class that will implement the PayMsg action of parent class for GasPump_2.
• PayMsg(DataStore d)
This will display the two payment methods that are Pay by debit card and Pay by credit card

Store_Cash
This class is the parent abstract class for Store Cash action strategy.
The classes implementing this class will be having definition of the class
This class have one abstract method to perform the action that will take a datastore object as an argument
• StoreCash(DataStore d)
abstract method which will be override by subclass to store the cash when cash Payment option available on the GasPumps.

Store_Cash1
This is the child class that will implement the StoreCash action of parent class for GasPump_1.
• StoreCash(DataStore d)
This will store the cash amount in cash variable of DS_2 class.

Store_Cash2
This is the child class that will implement the StoreCash action of parent class for GasPump_2.
• StoreCash (DataStore d)
It will display error message that you can not store cash you there is no cash payment option

Display_Menu
This class is the parent abstract class for Display Menu action strategy.
The classes implementing this class will be having definition of the class
This class have one abstract method to perform the action that will take a datastore object as an argument
• DisplayMenu(DataStore d)
Abstract method which will be override by subclass to display different menu message depending on the GasPumps.

Display_Menu1
This is the child class that will implement the DisplayMenu action of parent class for GasPump_1.
• DisplayMenu(DataStore d)
It will just display money received message as we have only multiple options to select gas type

Display_Menu2
This is the child class that will implement the DisplayMenu action of parent class for GasPump_2.

• DisplayMenu(DataStore d)
It uses DS_2 class and fetch stored data for showing detail like price of Diesel, regular, super gasoline as a menu for selecting the gasoline.

Reject_Msg
This class is the parent abstract class for Reject message action strategy.
The classes implementing this class will be having definition of the class
This class have one abstract method to perform the action that will take a datastore object as an argument
• RejectMsg(DataStore d)
abstract method which will be override by subclass to display different Reject Message depending on the payment options available for GasPumps.

Reject_Msg1
This is the child class that will implement the RejectMsg action of parent class for GasPump_1.
• RejectMsg(DataStore d)
This will display Credit card rejected and cancelling transaction message.

Reject_Msg2
This is the child class that will implement the RejectMsg action of parent class for GasPump_2.
• RejectMsg(DataStore d)
This will display Credit card rejected and cancelling transaction message.

Set_Price
This class is the parent abstract class for Set Price action strategy.
The classes implementing this class will be having definition of the class
This class have one abstract method to perform the action that will take a datastore object and type of gas selected by user as an argument
• SetPrice(DataStore d, int g)
Abstract method which will be override by subclass depending on the type of gasolines present on different GasPumps.

Set_Price1
This is the child class that will implement the SetPrice action of parent class for GasPump_1.
• SetPrice(DataStore d, int g)
this function will not do anything as we don't have option for selecting gas type for GasPump_1and the price is already set

Set_Price2
This is the child class that will implement the SetPrice action of parent class for GasPump_2.
• SetPrice(DataStore d, int g)
This will set the price variable of DS_2 class based on the value of g i.e, the type of gasoline selected by the user(set price=Dprice if user select diesel, set price=Rprice if user select Regular gase or set price=Sprice if user select Super gas)

Set_InitialValues
This class is the parent abstract class for Setting Initial Values action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• SetInitialValues(DataStore d)
Abstract method which will be override by subclass to initialize different variables depending on the GasPumps.

Set_InitialValues1
This is the child class that will implement the SetInitialValues action of parent class for GasPump_1.
• SetInitialValues (DataStore d)
This will set the value of Liters pumped to zero and total bill amount to zero as well.

Set_InitialValues2
This is the child class that will implement the SetInitialValues action of parent class for GasPump_2.

• SetInitialValues (DataStore d)
This will set the value of Gallons pumped to zero and total bill amount to zero as well.

Pump_GasUnit
This class is the parent abstract class for Pump Gas Unit action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• PumpGasUnit(DataStore d)
Abstract method which will be override by subclass to update the total units of gasoline pumped and to update the total amount depending on the GasPumps.

Pump_GasUnit1
This is the child class that will implement the PumpGasUnit action of parent class for GasPump_1.
• PumpGasUnit(DataStore d)
It will increase the value of L by 1 (i.e., number of liters pumped by 1) and also add the price of gas to the total amount.

Pump_GasUnit2
This is the child class that will implement the PumpGasUnit action of parent class for GasPump_2.
• PumpGasUnit (DataStore d)
It will increase the value of G by 1 (i.e., number of Gallons pumped by 1) and also add the price of 1 gallon of selected gas to the total amount.

GasPumped_Msg
This class is the parent abstract class for Gas Pumped message action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• GasPumpedMsg(DataStore d)
Abstract method which will be override by subclass to display number of units of gas pumped of different kinds depending on the GasPumps.

GasPumped_Msg1
This is the child class that will implement the GasPumpedMsg action of parent class for GasPump_1 .
• GasPumpedMsg (DataStore d)
This method showing detail message about total no. of Liters pumped by fetching data from DS_1 class .

GasPumped_Msg 2
This is the child class that will implement the GasPumpedMsg action of parent class for GasPump_2.
• GasPumpedMsg (DataStore d)
This method showing detail message about total no. of gallon pumped by fetching data from DS_2 class .

Print_Receipt
This class is the parent abstract class for Print Receipt action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• PrintReceipt(DataStore d)
Abstract method which will be override by subclass to display different transaction specific details depending on the GasPumps.

Print_Receipt1
This is the child class that will implement the PrintReceipt action of parent class for GasPump_1.
• PrintReceipt (DataStore d)
This Class will display the receipt showing total liters of gasoline pumped along with the rate per liter and the total amount

Print_Receipt 2
This is the child class that will implement the PrintReceipt action of parent class for GasPump_2.

• PrintReceipt (DataStore d)
This Class will display the receipt showing total gallons of gasoline pumped along with the rate per Gallon and the total amount

Cancel_Msg
This class is the parent abstract class for cancel message action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• CancelMsg(DataStore d)
Abstract method which will be override by subclass to display different Cancel messages depending on the GasPumps. But here in this case both the messages will be same.

Cancel_Msg1
This is the child class that will implement the CancelMsg action of parent class for GasPump_1.
• CancelMsg (DataStore d)
//It will display cancellation message for GasPump_1

Cancel_Msg2
This is the child class that will implement the CancelMsg action of parent class for GasPump_2.
• CancelMsg (DataStore d)
//It will display cancellation message for GasPump_2

Return_Cash
This class is abstract class for Return Cash action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• ReturnCash(DataStore d)
Abstract method which will be override by subclass to return cash depending on the GasPumps.

Return_Cash1
This is the child class that will implement the ReturnCash action of parent class for GasPump_1.
• ReturnCash (DataStore d)
It will return the cash just in case user will cancel the transaction. It will display the returning amount.

Return_Cash2
This is the child class that will implement the ReturnCash action of parent class for GasPump_2.
• ReturnCash (DataStore d)
GasPump2 uses credit card and Debit card payment methods so it will not return the cash and show just warning message.

WrongPin_Msg
This class is the parent abstract class for Wrong pin message action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• WrongPinMsg(DataStore d)
Abstract method which will be override by subclass to display different messages depending on the GasPumps.

WrongPin_Msg1
This is the child class that will implement the WrongPinMsg action of parent class for GasPump_1.
• WrongPinMsg (DataStore d)
It will not display anything just the warning message as GasPump_1 has no option for debit card payment.

WrongPin_Msg2
This is the child class that will implement the WrongPinMsg action of parent class for GasPump_2.
• WrongPinMsg (DataStore d)
It will display the Wrong Pin message just in case user selected debit card payment option and enters the wrong pin.

Store_Pin
This class is the parent abstract class for Store Pin action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• StorePin(DataStore d)
Abstract method which will be override by subclass to store pin depending on the GasPumps.

Store_Pin1
This is the child class that will implement the StorePin action of parent class for GasPump_1.
• StorePin(DataStore d)
It will not store anything just the warning message as GasPump_1 has no option for debit card payment.

Store_Pin2
This is the child class that will implement the StorePin action of parent class for GasPump_2.
• StorePin(DataStore d)
It will store the Pin into DS_2 class just in case user selected debit card payment option.

EnterPin_Msg
This class is the parent abstract class for Enter Pin message action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• EnterPinMsg(DataStore d)
Abstract method which will be override by subclass to display different Enter pin message depending on the GasPumps.

EnterPin_Msg1
This is the child class that will implement the EnterPinMsg action of parent class for GasPump_1.
• EnterPinMsg (DataStore d)
This Class will not display anything or just warning message as We don't have debit card payment option for GasPump_1

EnterPin_Msg2
This is the child class that will implement the EnterPinMsg action of parent class for GasPump_2.
• EnterPinMsg (DataStore d)
This Class will display the Enter Pin Message for GAsPump_2 when user will select debit card payment option

Initialize_Data
This class is the parent abstract class for Initialize Data action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• InitializeData(DataStore d)
Abstract method which will be override by subclass to initialize different variables available depending on the GasPumps.

Initialize_Data1
This is the child class that will implement the InitializeData action of parent class for GasPump_1.
• InitializeData (DataStore d)
This Class will not display anything as We don't have any option for selecting gasoline type for GasPump_1

Initialize_Data2
This is the child class that will implement the InitializeData action of parent class for GasPump_2.
• InitializeData (DataStore d)
This Class will will reset the value of price to 0 for DS_2 class

Set_W
This class is the parent abstract class for Set W action strategy.
The classes implementing this class will be having definition of the class

This class have one method to perform the action that will take a datastore object and type of payment as an argument
• SetW(DataStore d, int g)
Abstract method which will be override by subclass to initialize different variables available depending on the GasPumps.

Set_W1
This is the child class that will implement the SetW action of parent class for GasPump_1.
• SetW (DataStore d, int g)
It will set the value of W in DS_1 to 1 when the payment type is Card payment else set the value to 0 in case payment type is Cash Payment.

Set_W2
This is the child class that will implement the SetW action of parent class for GasPump_2.
• SetW (DataStore d, int g)
It will not set anything as we don't have cash option of payment for GasPump_2

Eject_Card
This class is the parent abstract class for Eject Card action strategy.
The classes implementing this class will be having definition of the class
This class have one method to perform the action that will take a datastore object as an argument
• EjectCard(DataStore d)
Abstract method which will be override by subclass to display different Eject Card messages depending on the GasPumps. But here in this case both the messages will be same.

Eject_Card1
This is the child class that will implement the EjectCard action of parent class for GasPump_1.
• EjectCard (DataStore d)
This will display the Eject Card Message to the user

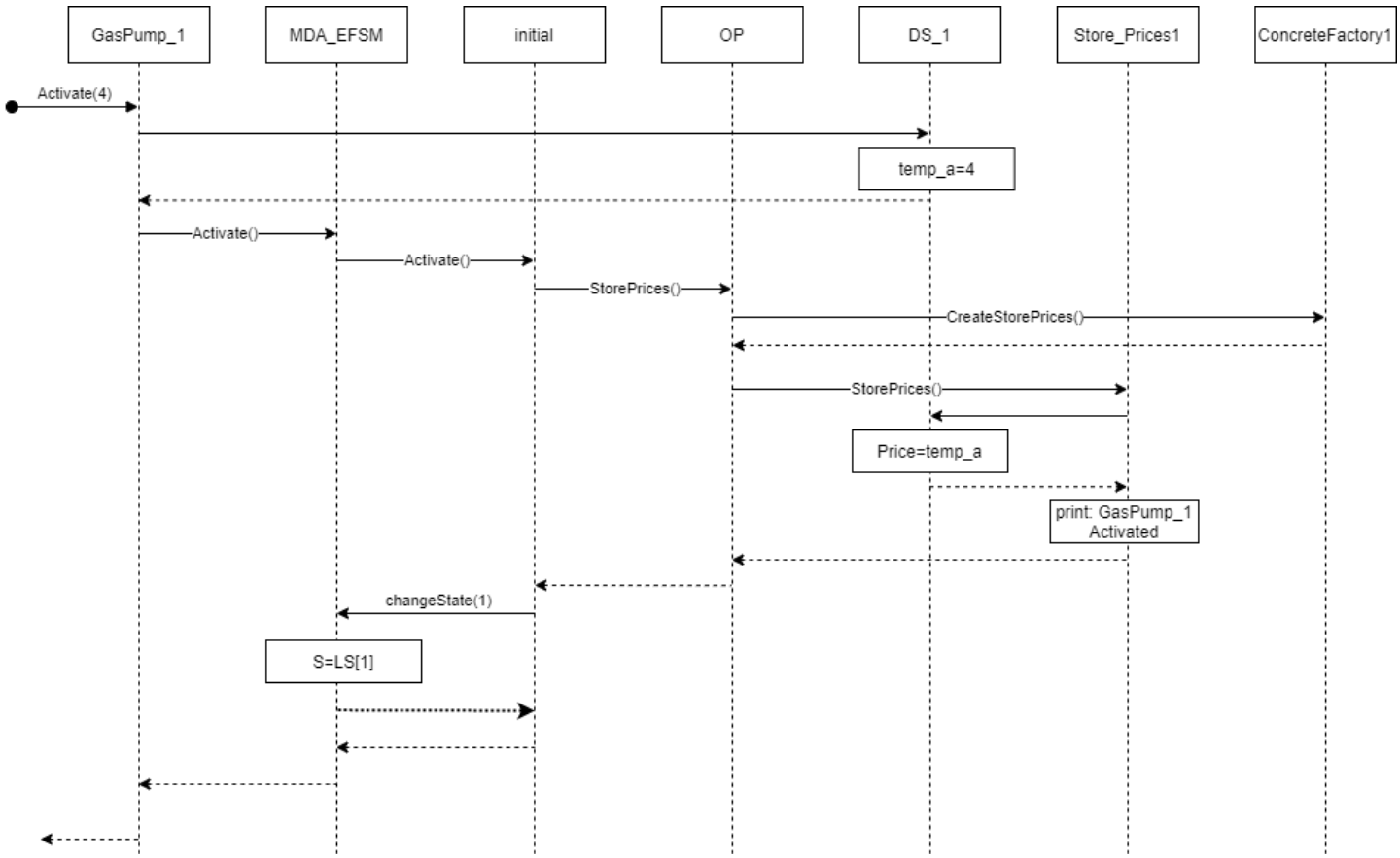
Eject_Card2
This is the child class that will implement the EjectCard action of parent class for GasPump_2.
• EjectCard (DataStore d)
This will display the Eject Card Message to the user

Dynamics

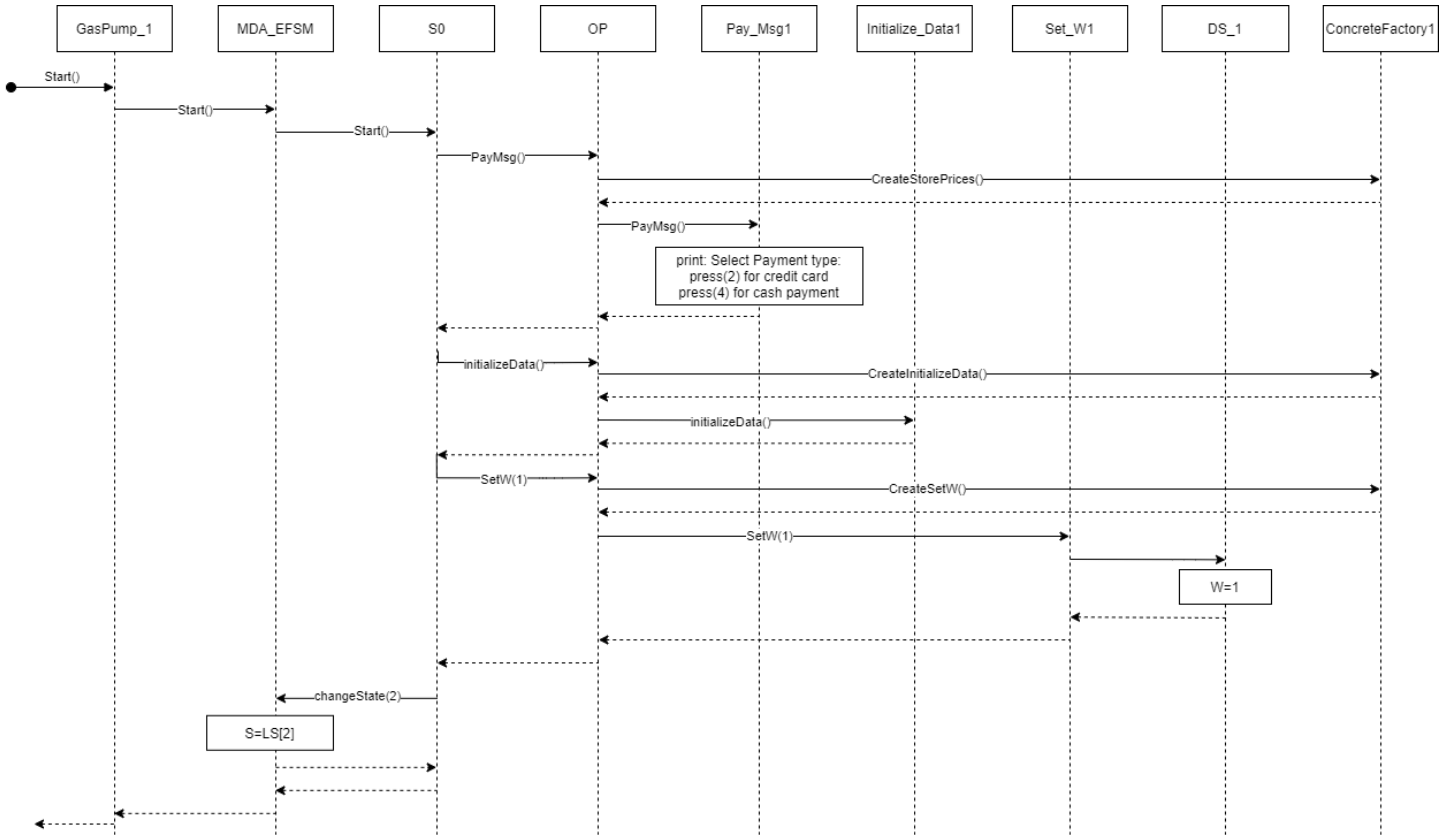
sequence diagrams for two Scenarios:

Scenario-I should show how one liter of gas is disposed in GasPump-1, i.e., the following sequence of operations is issued: Activate(4), Start(), PayCash(5), StartPump(), PumpLiter(), PumpLiter()

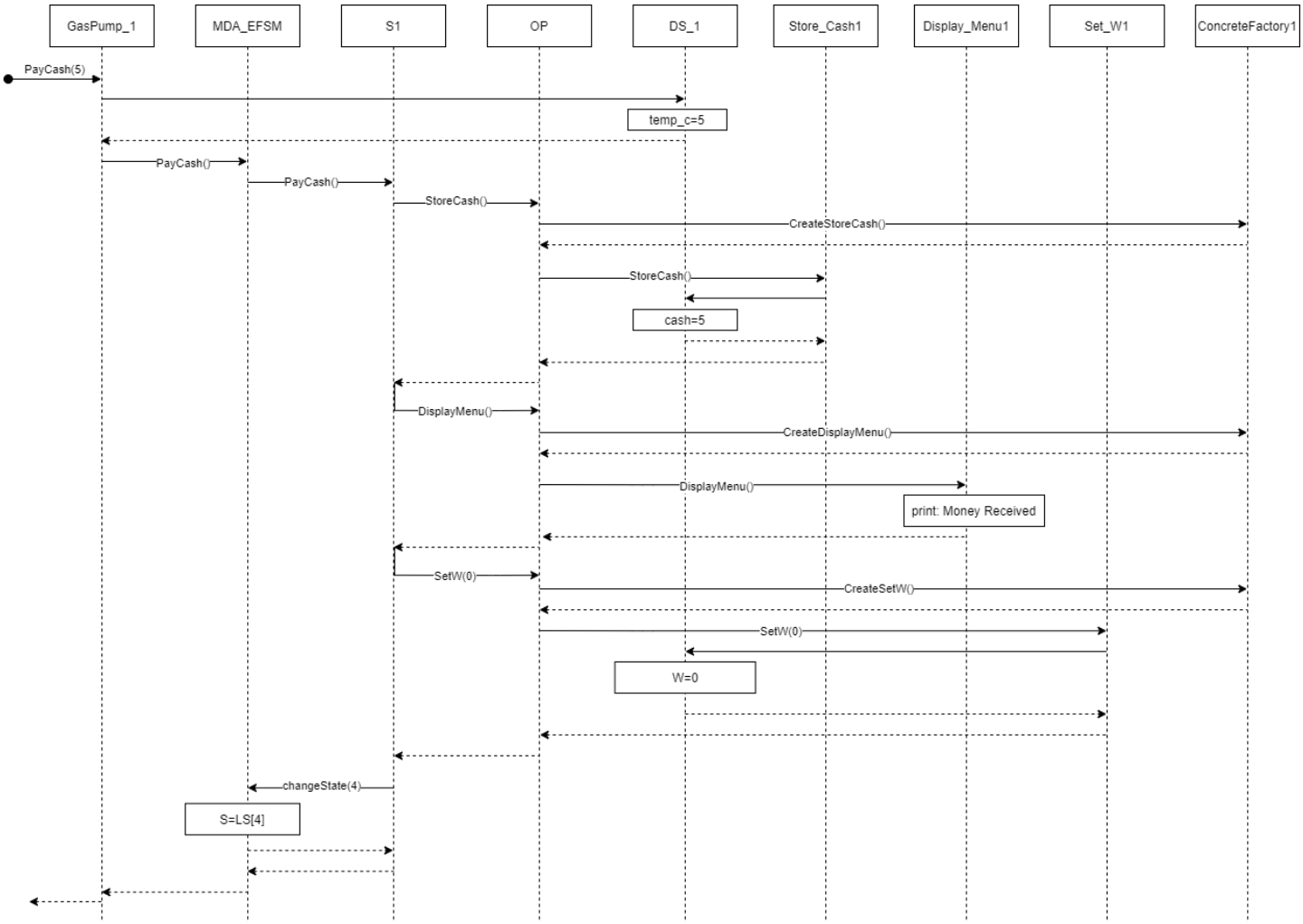
- **Activate(4)**



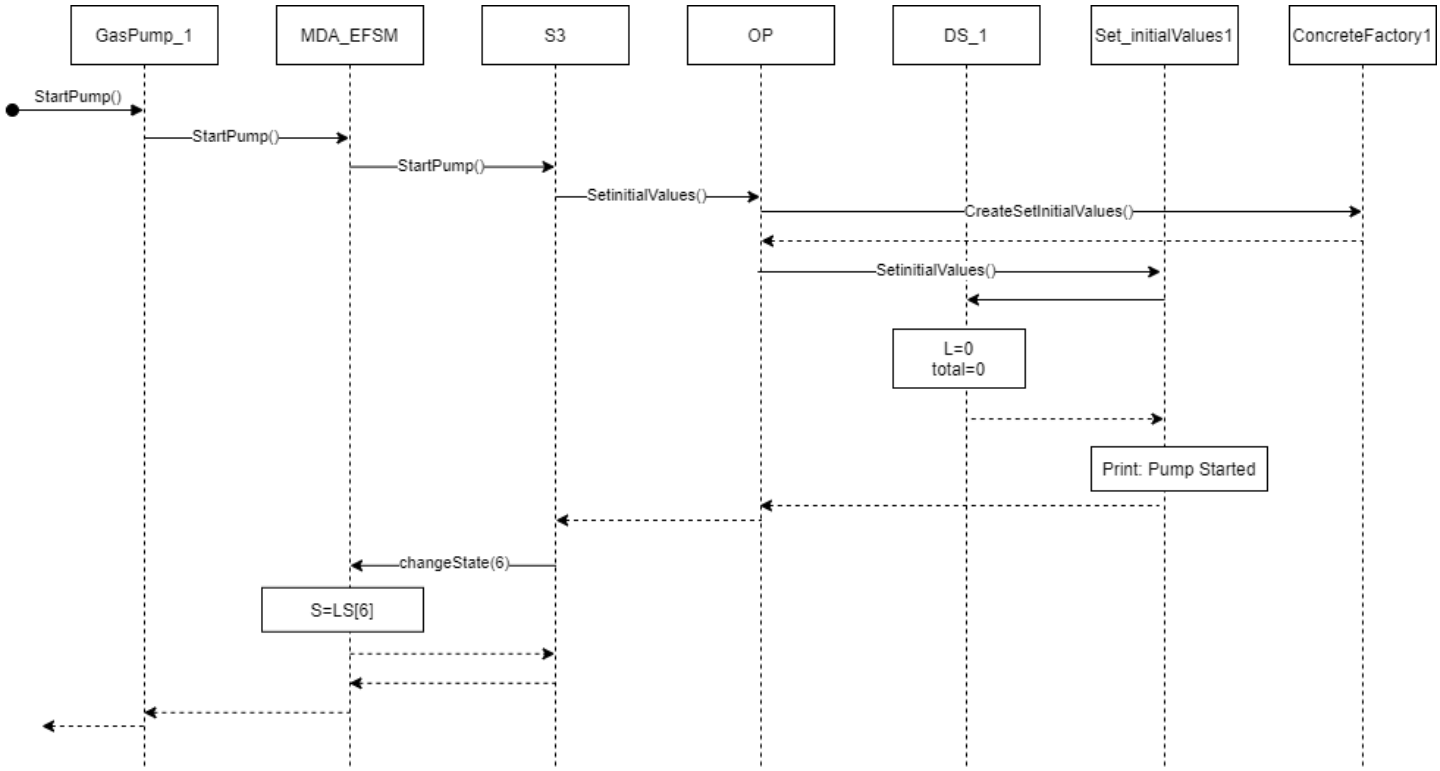
- **Start()**


GasPump_1
MDA_EFSM
S0
OP
Pay_Msg1
Initialize_Data1
Set_W1
DS_1
ConcreteFactory1
Start()
Start()
Start()
PayMsg()
CreateStorePrices()
PayMsg()
print: Select Payment type:
press(2) for credit card
press(4) for cash payment
initializeData()
CreateInitializeData()
initializeData()
SetW(1)
CreateSetW()
SetW(1)
W=1
changeState(2)
S=LS[2]

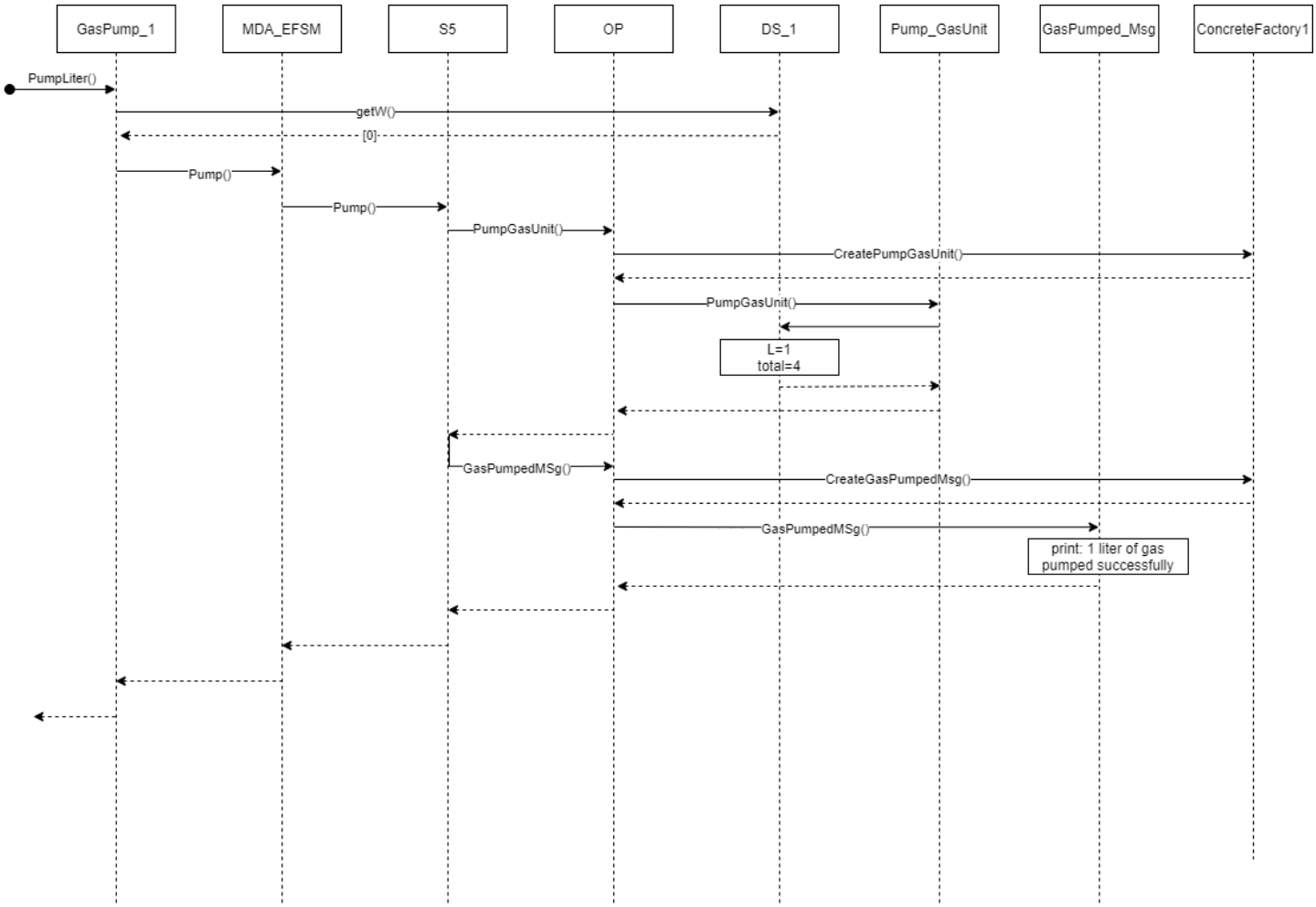
- **PayCash(5)**


GasPump_1
MDA_EFSM
S1
OP
DS_1
Store_Cash1
Display_Menu1
Set_W1
ConcreteFactory1
PayCash(5)
temp_c=5
PayCash()
PayCash()
StoreCash()
CreateStoreCash()
StoreCash()
cash=5
DisplayMenu()
CreateDisplayMenu()
DisplayMenu()
print: Money Received
SetW(0)
CreateSetW()
SetW(0)
W=0
changeState(4)
S=LS[4]

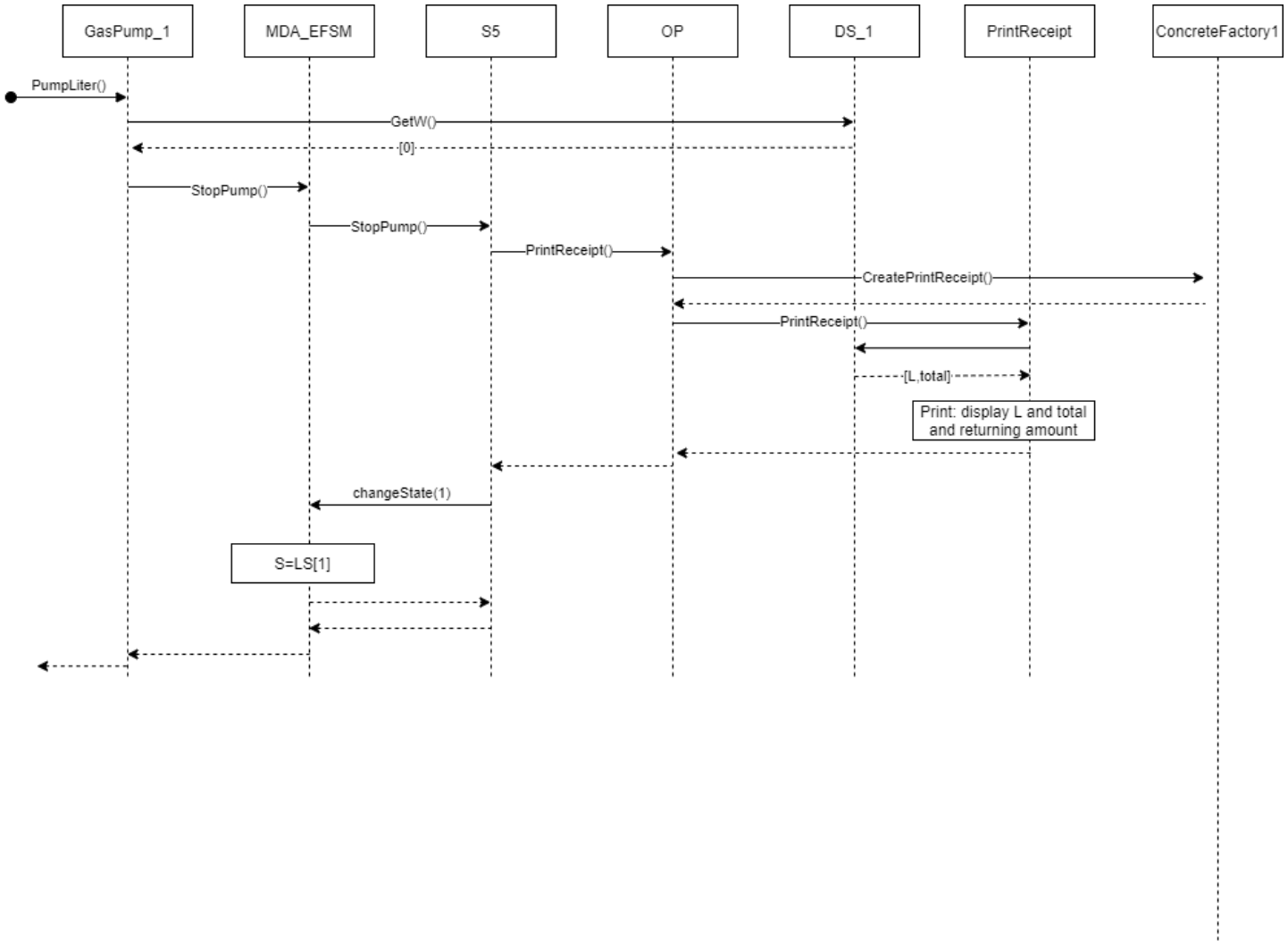
- **StartPump()**


GasPump_1
MDA_EFSM
S3
OP
DS_1
Set_initialValues1
ConcreteFactory1
StartPump()
StartPump()
StartPump()
SetinitialValues()
CreateSetInitialValues()
SetinitialValues()
L=0
total=0
Print: Pump Started
changeState(6)
S=LS[6]

- **PumpLiter()**

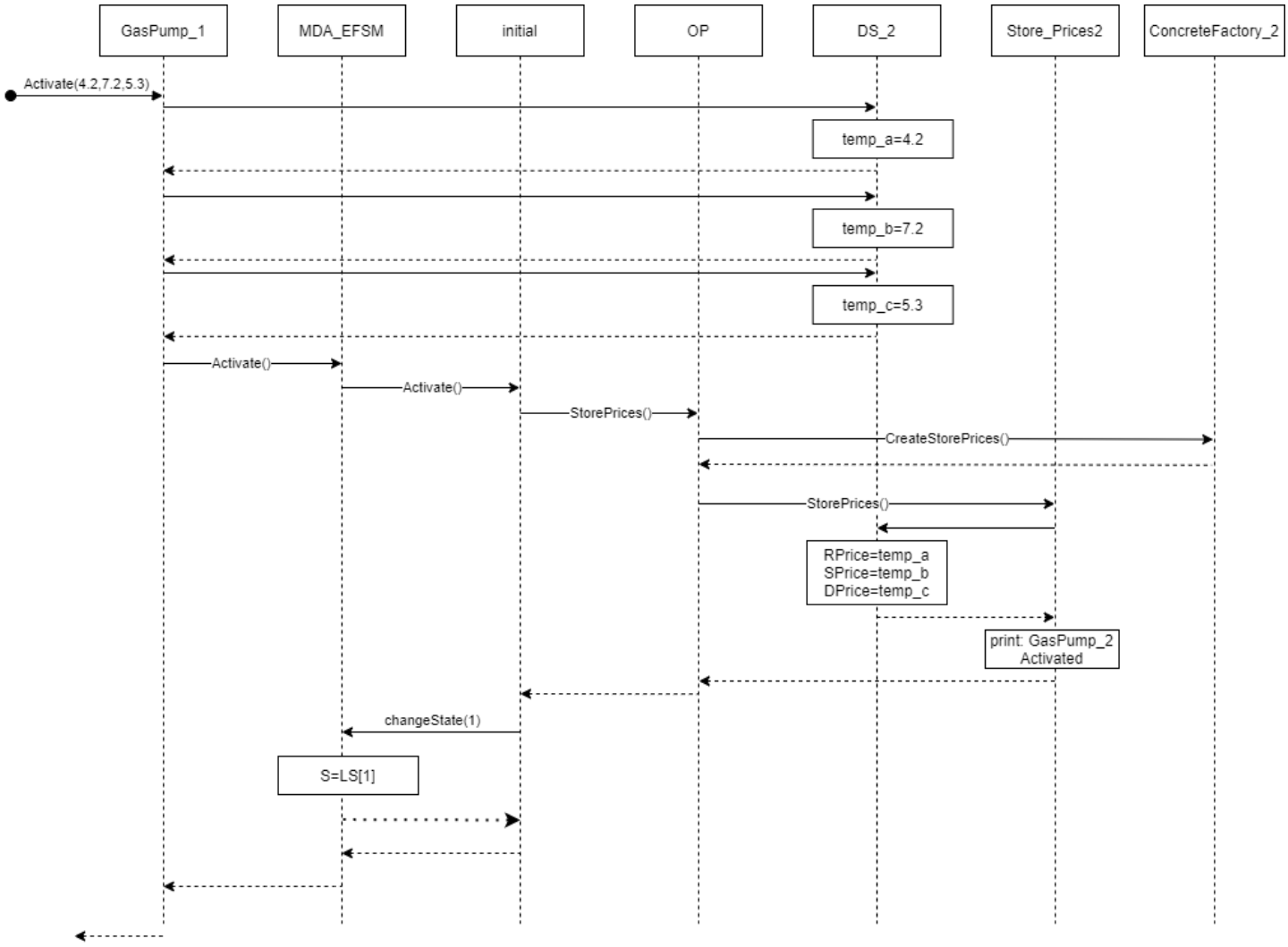

GasPump_1
MDA_EFSM
S5
OP
DS_1
Pump_GasUnit
GasPumped_Msg
ConcreteFactory1
PumpLiter()
getW()
[0]
Pump()
Pump()
PumpGasUnit()
CreatePumpGasUnit()
PumpGasUnit()
L=1
total=4
GasPumpedMSg()
CreateGasPumpedMsg()
GasPumpedMSg()
print: 1 liter of gas
pumped successfully

- **PumpLiter()**

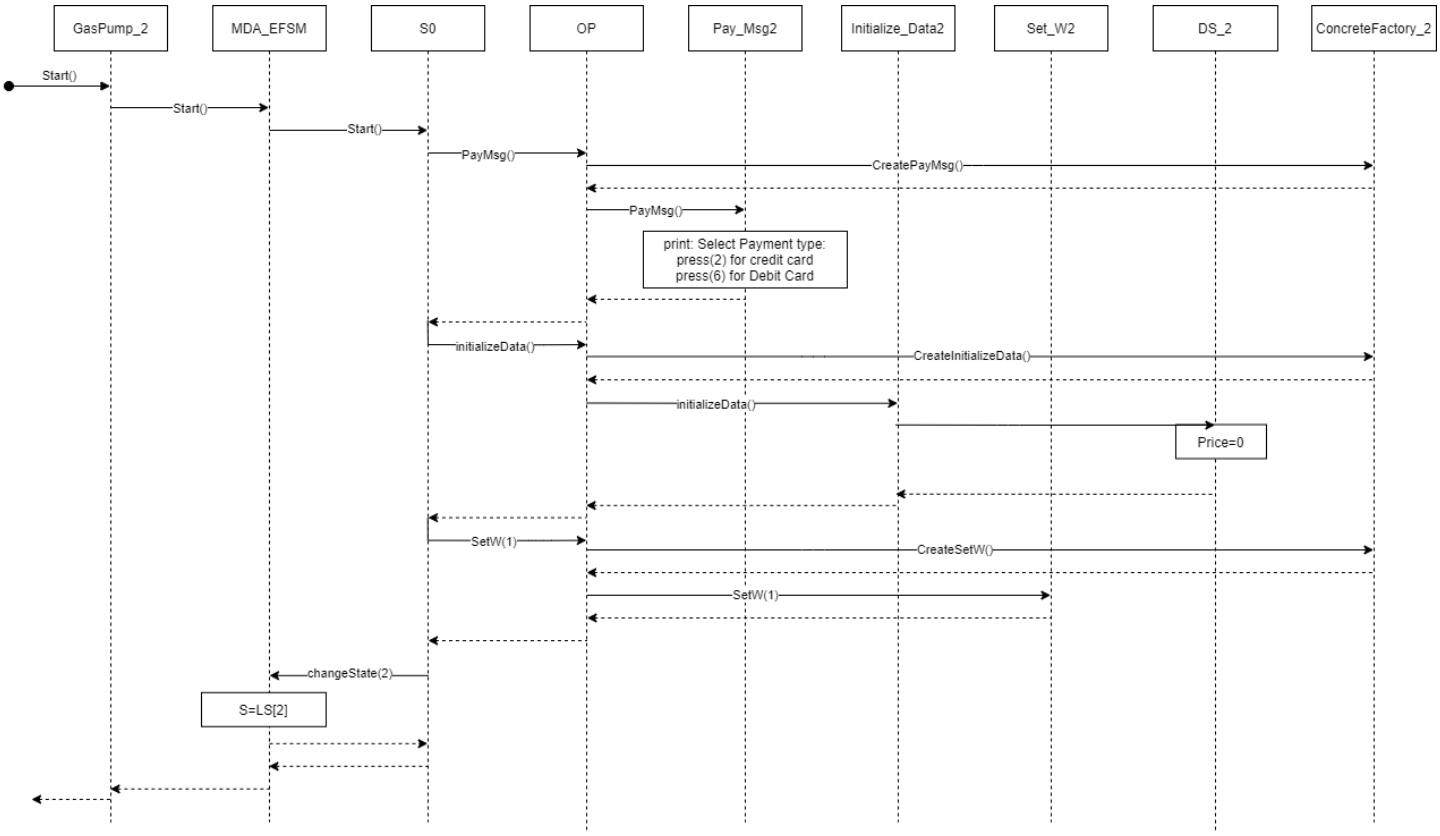

GasPump_1
MDA_EFSM
S5
OP
DS_1
PrintReceipt
ConcreteFactory1
PumpLiter()
GetW()
[0]
StopPump()
StopPump()
PrintReceipt()
CreatePrintReceipt()
PrintReceipt()
[L,total]
Print: display L and total and returning amount
changeState(1)
S=LS[1]

Scenario-II should show how one gallon of Super gas is disposed in GasPump-2, i.e., the following sequence of operations is issued: Activate(4.2, 7.2, 5.3), Start(), PayDebit("abc"), Pin("cba"), Pin("abc"), Super(), StartPump(), PumpGallon(), FullTank()

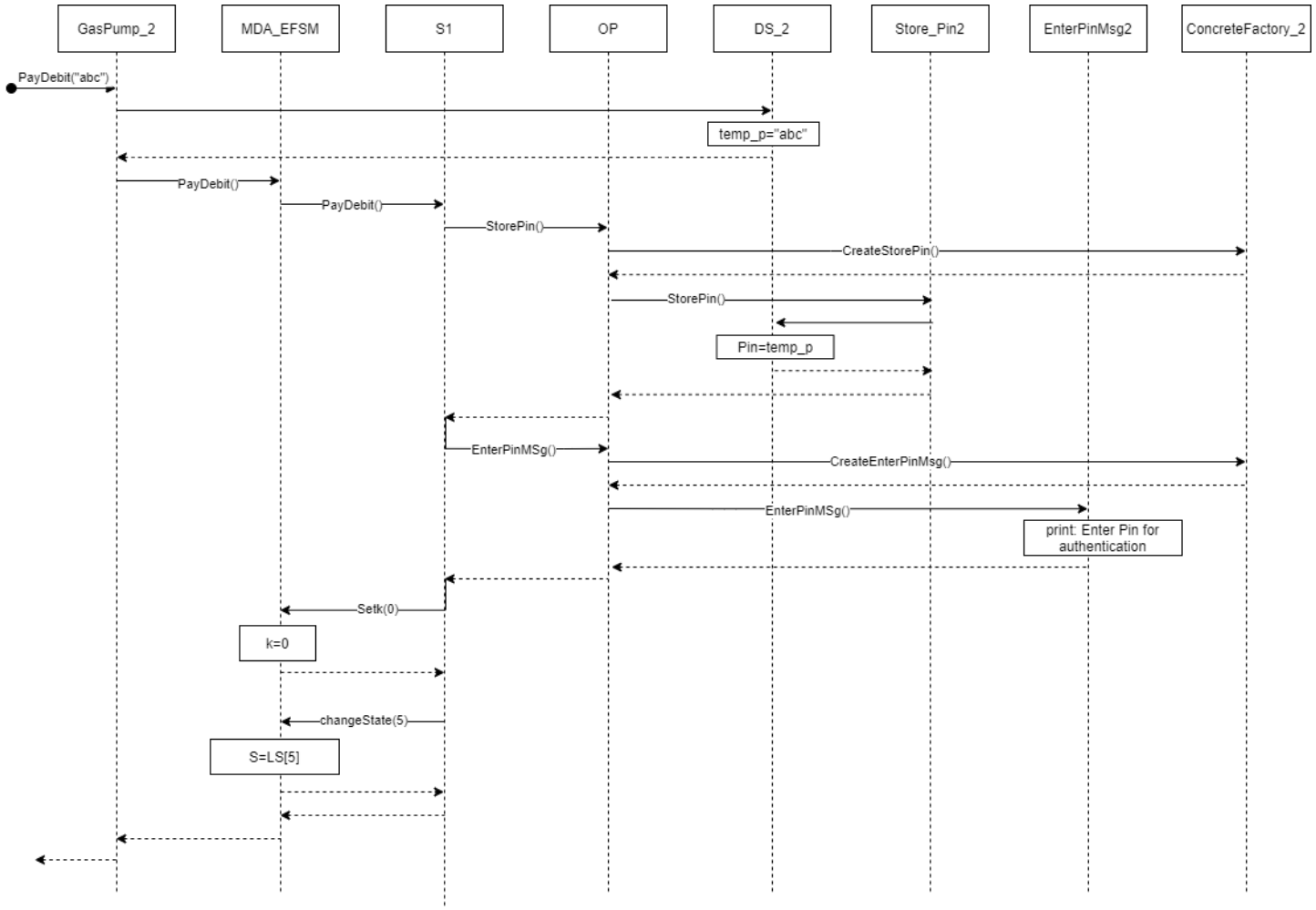
- **Activate(4.2, 7.2, 5.3)**



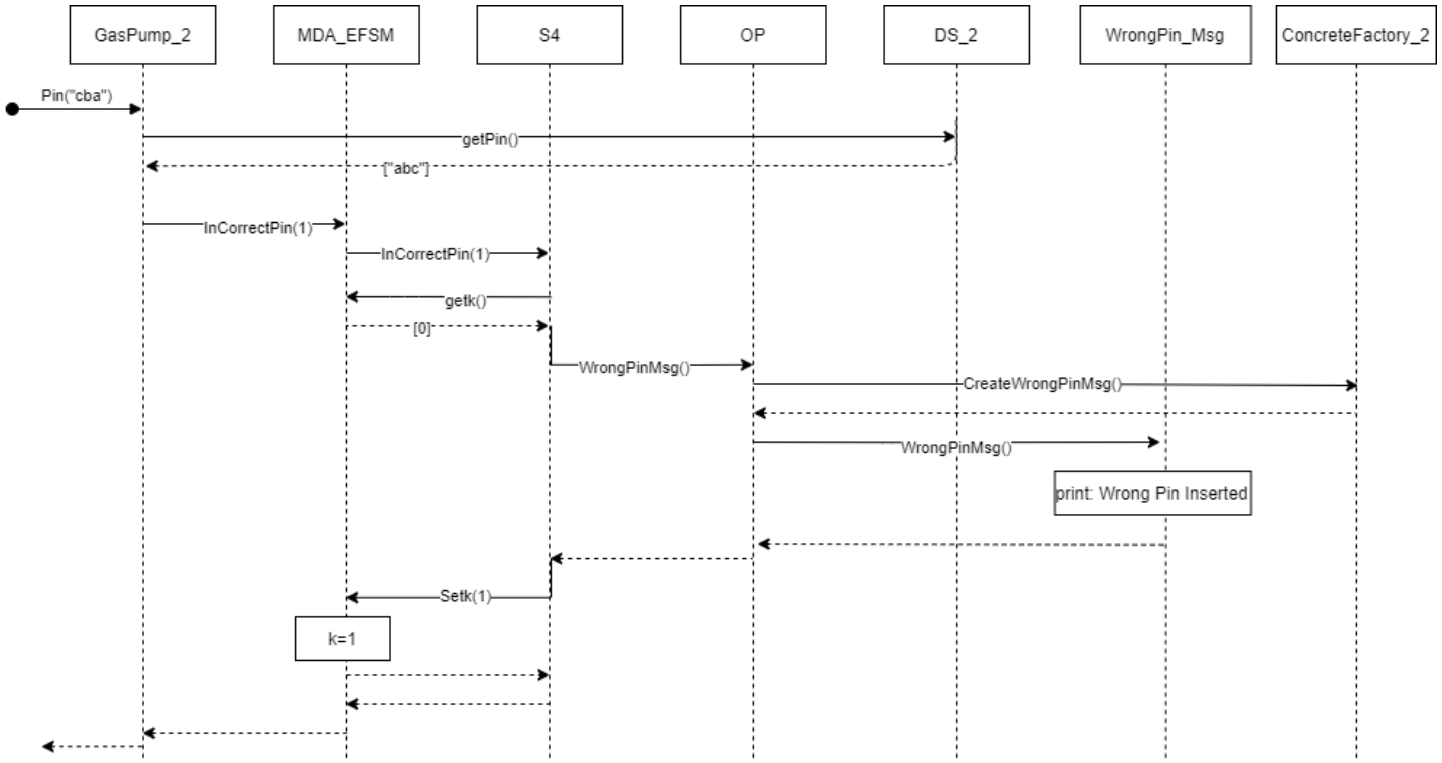
- **Start()**


GasPump_2
MDA_EFSM
S0
OP
Pay_Msg2
Initialize_Data2
Set_W2
DS_2
ConcreteFactory_2
Start()
Start()
Start()
PayMsg()
CreatePayMsg()
PayMsg()
print: Select Payment type:
press(2) for credit card
press(6) for Debit Card
initializeData()
CreateInitializeData()
initializeData()
Price=0
SetW(1)
CreateSetW()
SetW(1)
changeState(2)
S=LS[2]

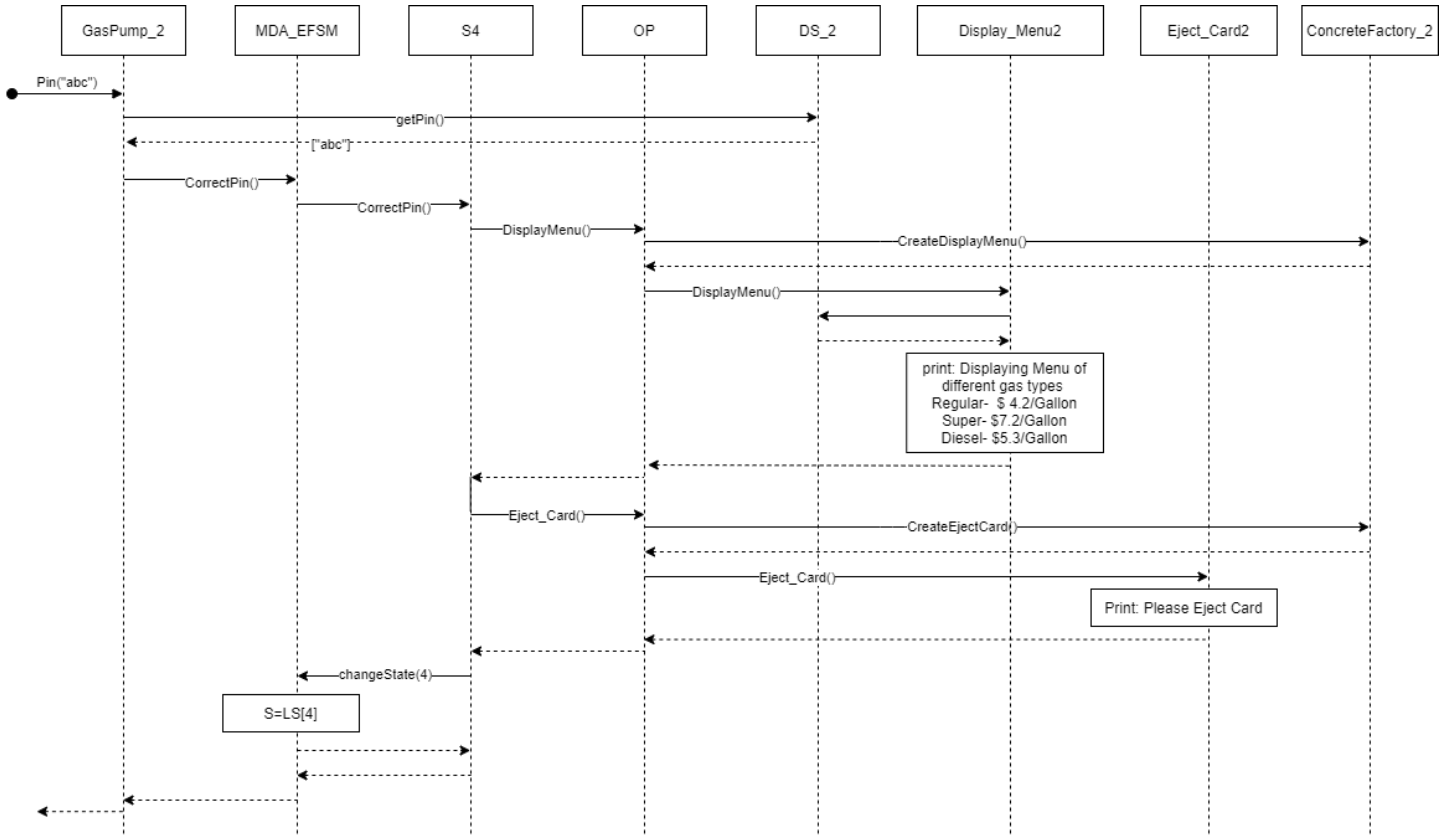
- **PayDebit(“abc”)**



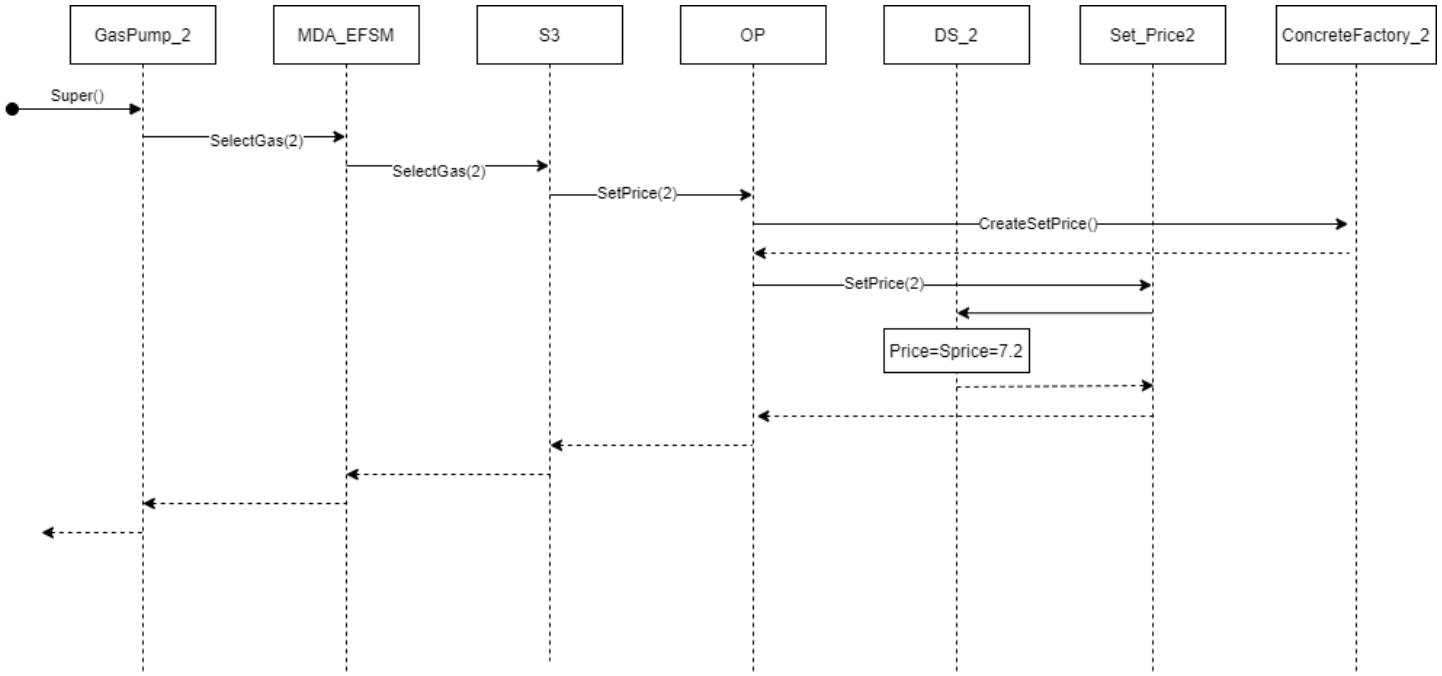
- **Pin(“cba”)**


GasPump_2
MDA_EFSM
S4
OP
DS_2
WrongPin_Msg
ConcreteFactory_2
Pin("cba")
getPin()
["abc"]
InCorrectPin(1)
InCorrectPin(1)
getk()
[0]
WrongPinMsg()
CreateWrongPinMsg()
WrongPinMsg()
print: Wrong Pin Inserted
Setk(1)
k=1

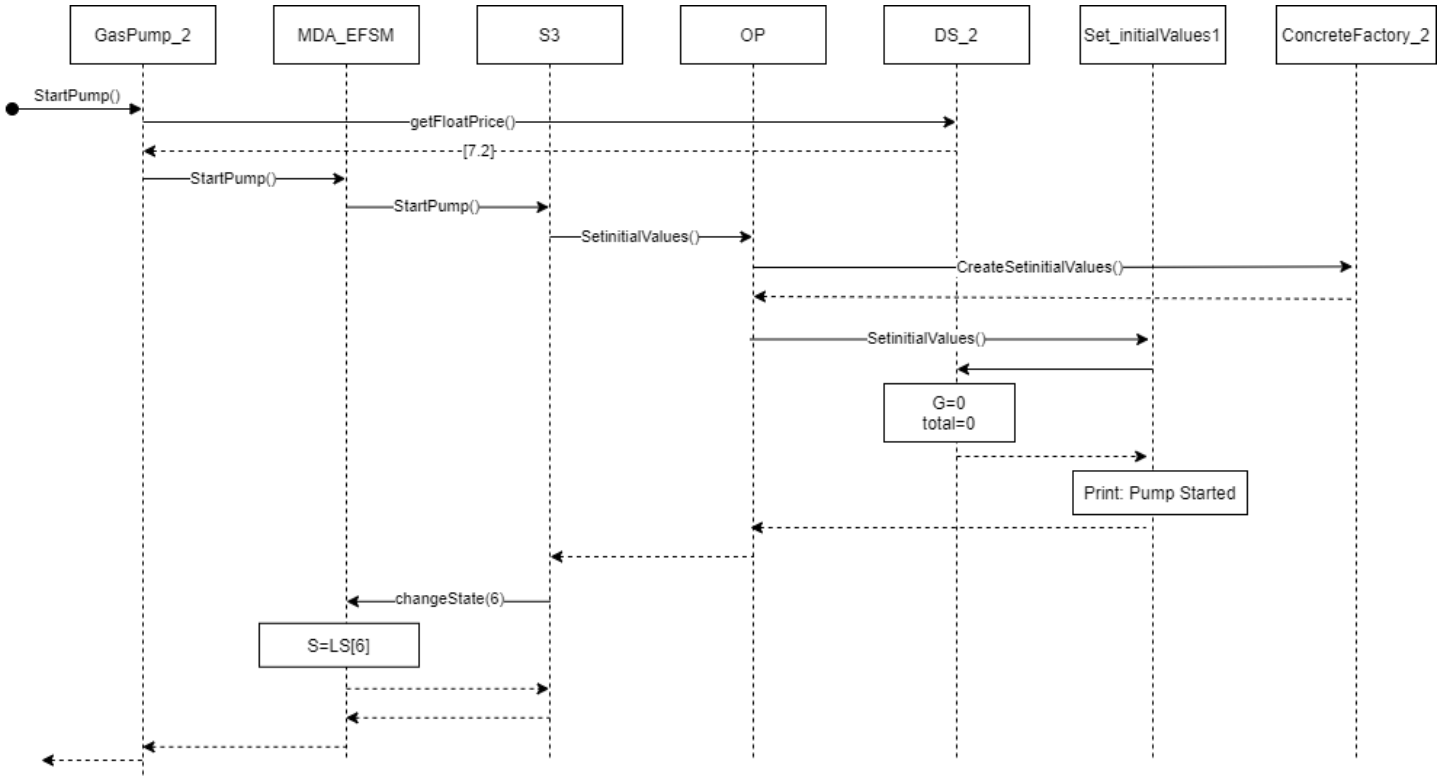
- **Pin(“abc”)**


GasPump_2
MDA_EFSM
S4
OP
DS_2
Display_Menu2
Eject_Card2
ConcreteFactory_2
Pin("abc")
getPin()
["abc"]
CorrectPin()
CorrectPin()
DisplayMenu()
CreateDisplayMenu()
DisplayMenu()
print: Displaying Menu of different gas types
Regular- $ 4.2/Gallon
Super- $7.2/Gallon
Diesel- $5.3/Gallon
Eject_Card()
CreateEjectCard()
Eject_Card()
Print: Please Eject Card
changeState(4)
S=LS[4]

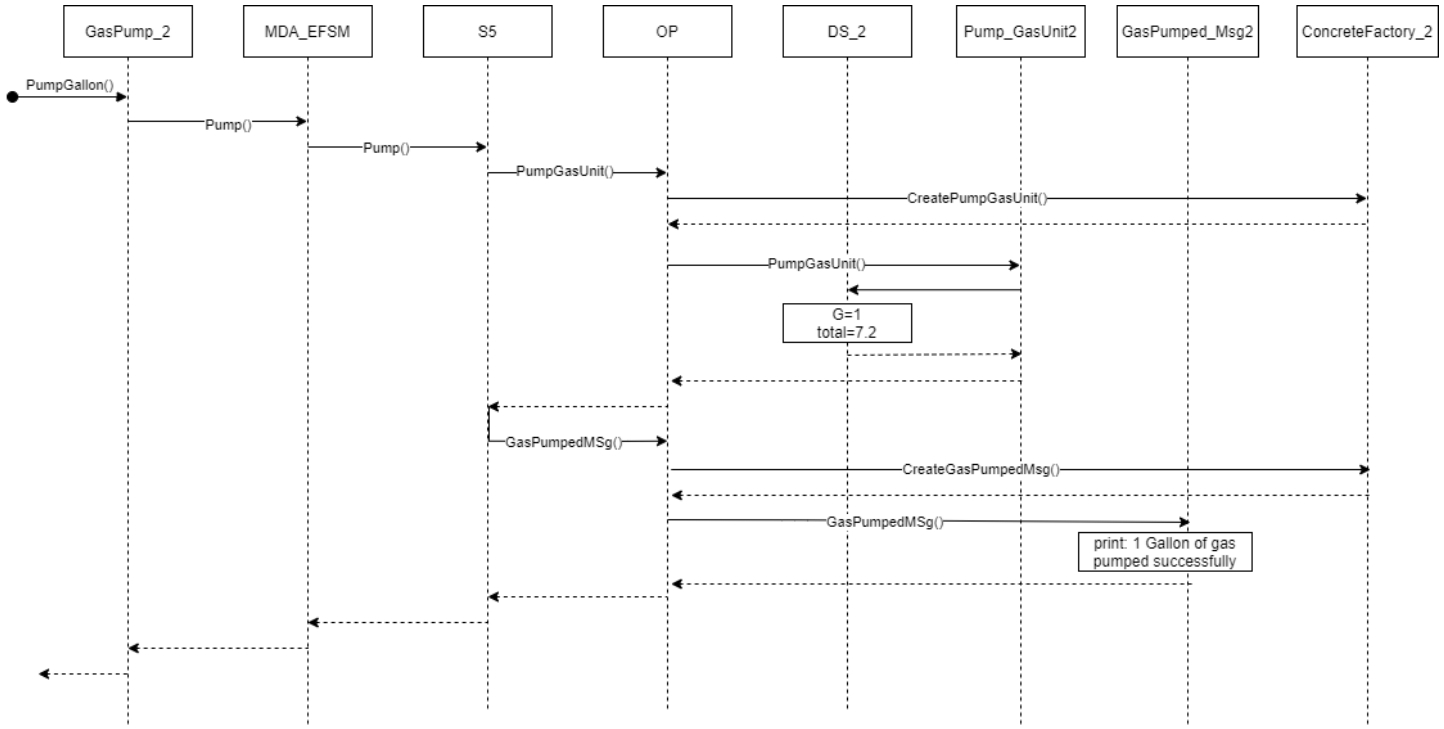
- **Super()**


GasPump_2
MDA_EFSM
S3
OP
DS_2
Set_Price2
ConcreteFactory_2
Super()
SelectGas(2)
SelectGas(2)
SetPrice(2)
CreateSetPrice()
SetPrice(2)
Price=Sprice=7.2

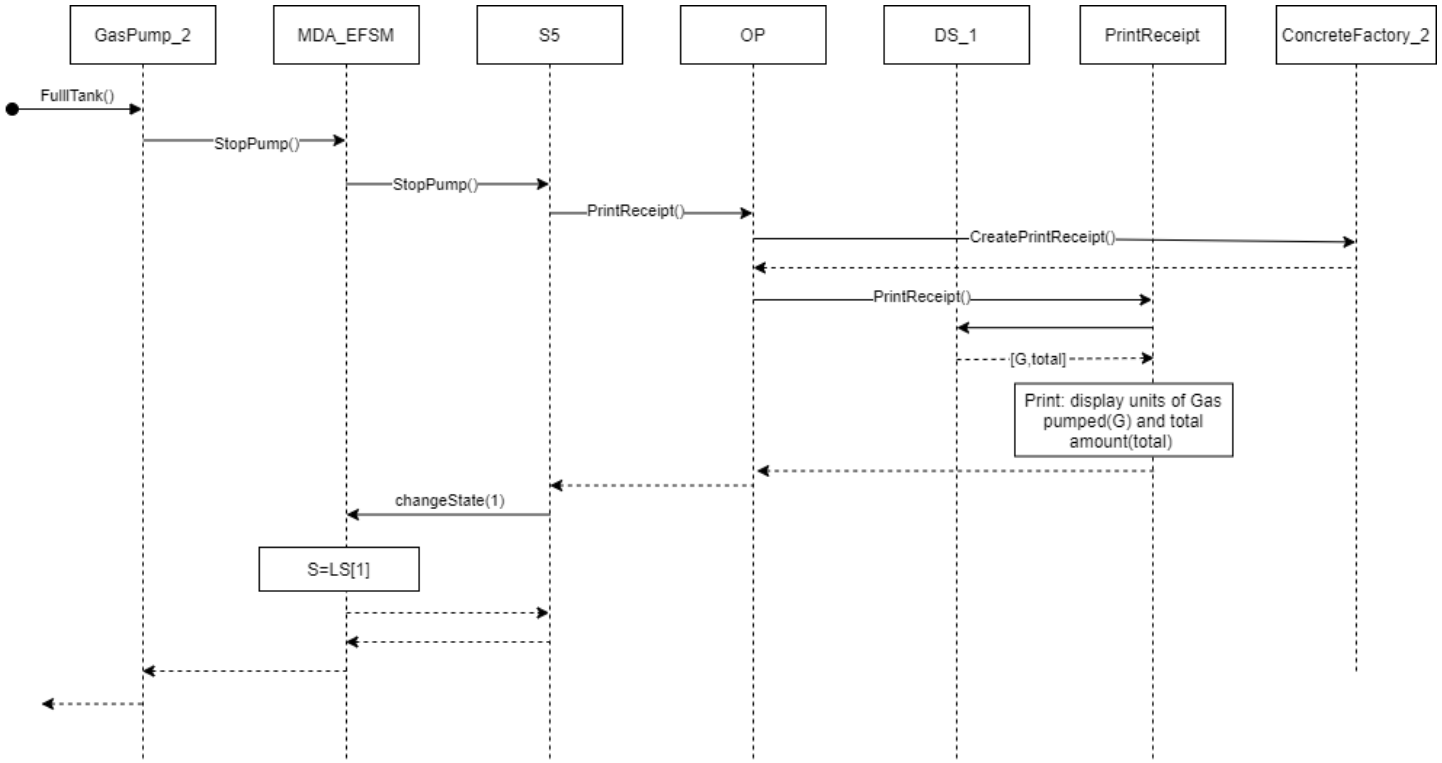
- **StartPump()**


GasPump_2
MDA_EFSM
S3
OP
DS_2
Set_initialValues1
ConcreteFactory_2
StartPump()
getFloatPrice()
[7.2]
StartPump()
StartPump()
SetinitialValues()
CreateSetinitialValues()
SetinitialValues()
G=0
total=0
Print: Pump Started
changeState(6)
S=LS[6]

- **PumpGallon()**


GasPump_2
MDA_EFSM
S5
OP
DS_2
Pump_GasUnit2
GasPumped_Msg2
ConcreteFactory_2
PumpGallon()
Pump()
Pump()
PumpGasUnit()
CreatePumpGasUnit()
PumpGasUnit()
G=1
total=7.2
GasPumpedMSg()
CreateGasPumpedMsg()
GasPumpedMSg()
print: 1 Gallon of gas
pumped successfully

- **FullTank()**


GasPump_2
MDA_EFSM
S5
OP
DS_1
PrintReceipt
ConcreteFactory_2
FulllTank()
StopPump()
StopPump()
PrintReceipt()
CreatePrintReceipt()
PrintReceipt()
[G,total]
Print: display units of Gas pumped(G) and total amount(total)
changeState(1)
S=LS[1]

Source Code and Pattern:

State Patterns –
Source code of State Patterns can be found in **'SSA_Project/src/State'** package. It contains code of all the states which are used for maintaining the state transition of MDA-EFSM.

There are **8** (including parent class) states in total which are as follow-
'State' class – **Parent** of all state class.
Concrete states classes – includes 'initial', 'S0', 'S1', 'S2', 'S3', 'S4', 'S5' classes**.**
Client Class- MDA-EFSM

Description of all state classes are already mentioned above.
Source code of each individual state classes can be found in following way–
State class - SSA_Project/src/State/State.java
S0 class - SSA_Project/src/State/S0.java
S1 class - SSA_Project/src/State/S1.java
S2 class - SSA_Project/src/State/S2.java
S3 class - SSA_Project/src/State/S3.java
S4 class - SSA_Project/src/State/S4.java
S5 class - SSA_Project/src/State/S5.java
S6 class - SSA_Project/src/State/S6.java

Strategy Patterns –
Source code of Strategy Pattern can be found in **''/SSA_Project/src/*'** packages except OP, State, MDA-EFSM, DataStore and GasPump package.
It groups actions of different GasPumps.
Description of all strategy pattern classes are already mentioned above.

Different types of Strategy Pattern Classes are as follow –
Strategy Abstract Classes includes Cancel_Msg, Display_Menu, Eject_Card, EnterPin_Msg, GasPumped_Msg, Initialize_Data, Pay_Msg, Print_Receipt, Reject_Msg, Pump_GasUnit, Return_Cash, SetInitValues, Set_Price, Store_Cash, StorePrices, WrongPin_Msg.

Strategy Concrete Classes includes Cancel_Msg1, Display_Menu1, Eject_Card1, EnterPin_Msg1, GasPumped_Msg1, Initialize_Data1, Pay_Msg1, Print_Receipt1, Reject_Msg1, Pump_GasUnit1, Return_Cash1, SetInitValues1, Set_Price1, Store_Cash1, StorePrices1, WrongPin_Msg1, Cancel_Msg2, Display_Menu2, Eject_Card2, EnterPin_Msg2, GasPumped_Msg2, Initialize_Data2, Pay_Msg2, Print_Receipt2, Reject_Msg2, Pump_GasUnit2, Return_Cash2, SetInitValues2, Set_Price2, Store_Cash2, StorePrices2, WrongPin_Msg2

Client Class – OP

Abstract Factory Patterns –
Source code of Abstract Pattern and Concrete Factory Patterns can be found in -**''/SSA_Project/src/AbstractFactory/'** package. Description of all Abstract Factory classes are already mentioned above.

There are 3 Abstract Factory classes which are as follows –
Abstract Factory Class – AbstractFactory class
Concrete Factory Classes – includes ConcreteFactory1 and ConcreteFactory2 classes.
Client Classes – GasPump_1, GasPump_2, OP

Source code of each individual Abstract Factory class and Concrete Factory classes can be found in following way–
AbstractFactory class - SSA_Project/src/AbstractFactory/AbstractFactory.java
ConcreteFactory1 class - - SSA_Project/src/AbstractFactory/ConcreteFactory1.java

ConcreteFactory2 class – - SSA_Project/src/AbstractFactory/ ConcreteFactory2.java

Concrete products created by concrete factories –

For GasPump1:
DS_1 Cancel_Msg1, Display_Menu1, Eject_Card1, EnterPin_Msg1, GasPumped_Msg1, Initialize_Data1, Pay_Msg1, Print_Receipt1, Reject_Msg1, Pump_GasUnit1, Return_Cash1, SetInitValues1, Set_Price1, Store_Cash1, StorePrices1, WrongPin_Msg1

For GasPump2:
DataStore2, Cancel_Msg2, Display_Menu2, Eject_Card2, EnterPin_Msg2, GasPumped_Msg2, Initialize_Data2, Pay_Msg2, Print_Receipt2, Reject_Msg2, Pump_GasUnit2, Return_Cash2, SetInitValues2, Set_Price2, Store_Cash2, StorePrices2, WrongPin_Msg2.