Software Systems Architecture (CS-586)

Project

- Sivasenthil Namachivayan

- A20391478

1. MDA-EFSM model for the GasPump component

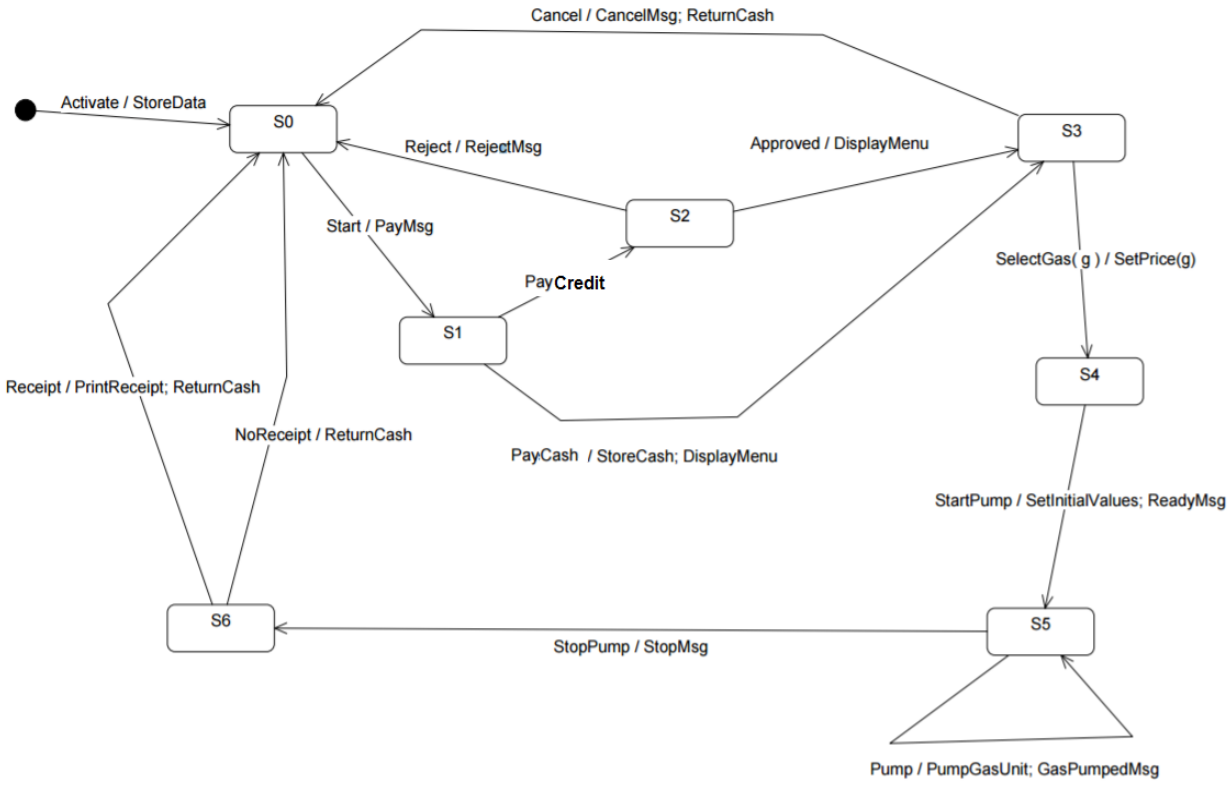
i. A list of meta events for the MDA-EFSM

1. Activate()
2. Start()
3. PayCash
4. PayCredit
5. Reject()
6. Cancel()
7. Approved()
8. StartPump()
9. Pump()
10. StopPump()
11. SelectGas(int g)
12. Receipt()
13. NoReceipt()

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

1. StoreData // stores price(s) for the gas from the temporary data store
2. PayMsg // displays a type of payment method
3. StoreCash // stores cash from the temporary data store
4. DisplayMenu // display a menu with a list of selections
5. RejectMsg // displays credit card not approved message – Generic Message
6. SetPrice(int g) // set the price for the gas identified by g identifier
7. ReadyMsg // displays the ready for pumping message – Generic Message
8. SetInitialValues // set G (or L) and total to 0
9. PumpGasUnit // disposes unit of gas and counts # of units disposed
10. GasPumpedMsg // displays the amount of disposed gas
11. StopMsg // stop pump message and receipt? msg (optionally) – Generic Message
12. PrintReceipt // print a receipt
13. CancelMsg // displays a cancellation message – Generic message
14. ReturnCash // returns the remaining cash

iii. A state diagram of the MDA-EFSM



iv. Pseudo-code of all operations of Input Processors of GasPump-1 and GasPump-2

Operations of the Input Processor (GasPump-1)

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

```
        m->Reject();
}
Cancel() {
        m->Cancel();
}
Approved(){
         m->Approved();
}
Regular()
{
        m->SelectGas(1)
}
Super() {
        m->SelectGas(2)
}
StartPump() {
        m->StartPump();
}
PumpGallon() {
        m->Pump();
}
StopPump() {
        m->StopPump();
        m->Receipt();
}
```

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

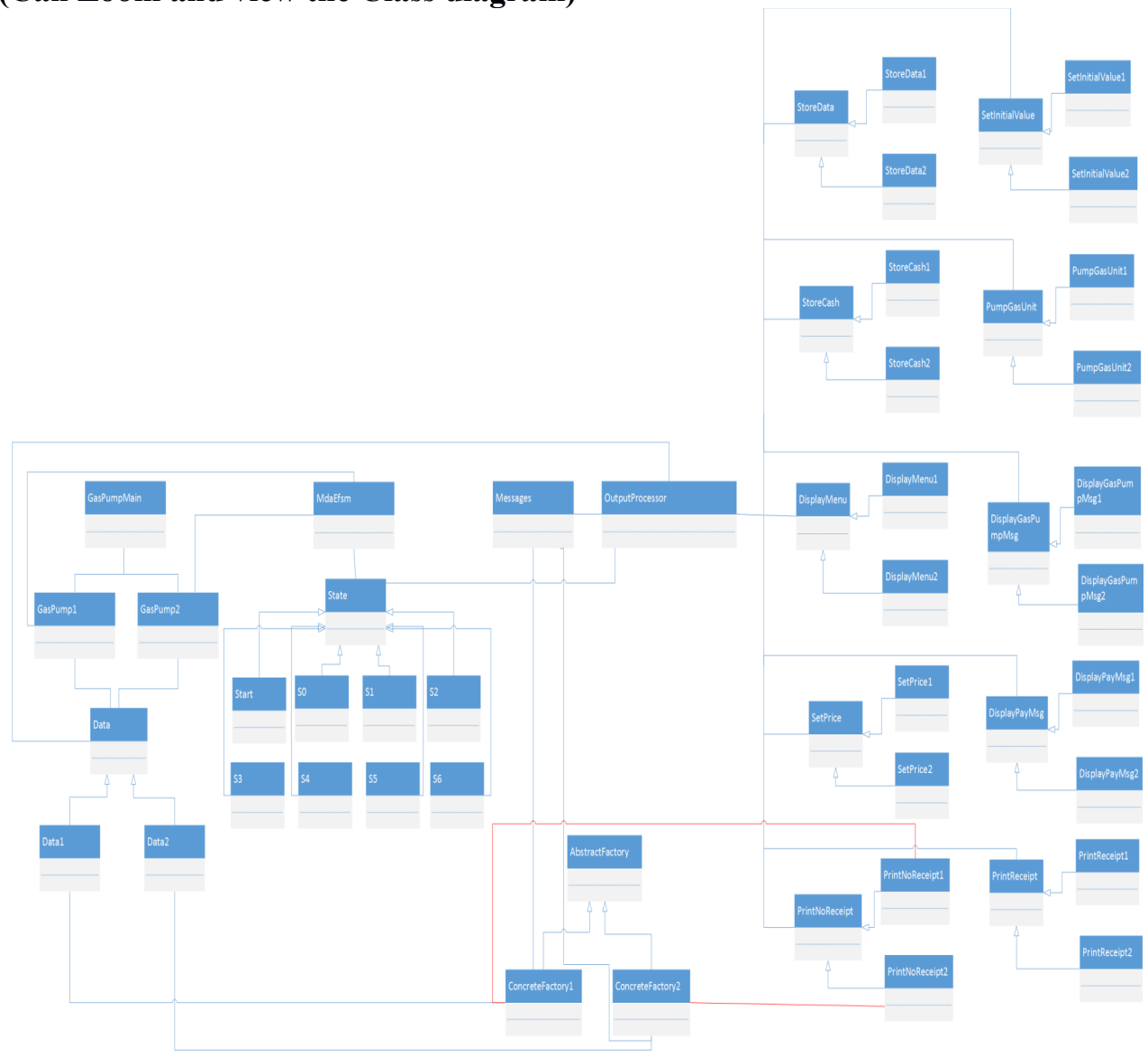
Operations of the Input Processor (GasPump-2)

```
Activate(int a, int b, int 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(float c) {
         if (c>0) {
                d→temp_cash=c;
```

```
                m->PayCash
        }
}
Cancel() {
        m->Cancel();
}
Regular() {
        m->SelectGas(1);
}
Super() {
        m->SelectGas(2);
}
Premium() {
        m->SelectGas(3);
}
StartPump() {
        m->StartPump();
}
PumpLiter() {
        if (d->cash<(d->L+1)*d->price)
                m->StopPump();
        else
                m->Pump()
}
Stop() {
        m->StopPump();
}
Receipt() {
        m->Receipt();
        Display return cash;
}
NoReceipt() {
         m->NoReceipt();
        Display return cash;
}
```

Notice:
cash: contains the value of cash deposited
price: contains the price of the selected gas
L: contains the number of liters already pumped cash , L, price are in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

2. Class diagram(s) of the MDA of the GasPump components (Can Zoom and view the Class diagram)



Note:

1.The Association relation will be there between ConcreteFactory1 to all the classes namely StoreCash1, DisplayMenu1, SetPrice1, StoreData1, SetInitialValues1, PumpGasUnit1, Displagaspumpedmsg1, DisplayPaMsg1 and PrintReceipt1 will be like the way ConcreteFactory1 is associated with PrintNoReceipt1 in the above class diagram. (Mentioned in red line)

2. The Association relation will be there between ConcreteFactory2 to all the classes namely StoreCash2, DisplayMenu2, SetPrice2, StoreData2, SetInitialValues2, PumpGasUnit2, Displagaspumpedmsg2, DisplayPayMsg2 and PrintReceipt2 like the way ConcreteFactory2 is associated with PrintNoReceipt2 in the above class diagram. (Mentioned in red line)

3.The Operations and the methods are in the source code and commented.

3. Class diagram description

GasPumpMain:

Main Program which invokes operations on GasPump1(GP1) and GasPump2(GP2) based on user selection. This takes the inputs from the user and perform the set of actions and invoke those operations on the Gaspumps. (The set of actions are listed in the Part 1 of this product description). The concreteFactory is created along with the objects that will be required for Mdaefsm.

GasPump1 & GasPump2:

These classes contains the operations related to the properties of GasPump1 and Gaspump2. These pumps contain the methods related to them.

Data1 & Data2:

These classes contain the variables dealt by the classes Gaspump1 and GasPump2 respectively.

ConcreteFactory1 & ConcreteFactory2:

These concrete classes return the objects of strategy classes Data of the respective Gaspump1 and GasPump2

Mdaefsm:

This Context Class invokes the operation in the respective state classes when a corresponding operation from the GasPump class(GasPump1 & GasPump2) is invoked. This will be performing all the state changes and it contains the list of classes which follows in state.
[0]-S0, [1]-S1, [2]-S2, [3]-S3, [4]-S4, [5]-S5, [6]-S6, [7]-Start

State:

This is an abstract class for all the states (Start, S0, S1, S2, S3, S4, S5, S6)

Start, S0, S1, S2, S3, S4, S5 & S6:

These classes are inherited from the Class State and perform the operations which is received from Mdaefsm and make the state transition when ever it is needed.

Output Processor:

This class gets the object from the concrete factory and performs the associated operation in the strategy pattern classes with respect to GP1 and GP2.

StoreData1 & StoreData2:

Store the prices for the gas from the temporary Data1 and Data2 for the GasPump1 and GasPump2 respectively.

StoreCash1 & StoreCash2:

Stores the cash from the temporary data store Data1 and Data2 for the GasPump1 and GasPump2 respectively.

DisplayMenu1 & DisplayMenu2:

Display a menu with a list of selections available in GasPump1 and GasPump2 respectively.

SetPrice1(int g) & SetPrice2(int g):

Set the price for the gas identified by g identifier for the GasPump1 and GasPump2 respectively.

SetInitialValues1 & SetInitialValues2:

Set the Initial values of G- Gallon and L-Liter to "0" and also the total to "0" for the respective values available in the GasPump1 and GasPump2 respectively.

PumpGasUnit1 & PumpGasUnit2:

Disposes the unit of gas as per the user and count the number of units disposed in Gallon or Liter as per the GasPump1 and GasPump2 respectively.

DisplayGasPumpedMsg1 & DisplayGasPumpedMsg2:

Display the amount of disposed gas of the GasPump1 and GasPump2 respectively.

DisplayPayMsg1 & DisplayPayMsg2:

Display the pay message of the GasPump1 and GasPump2 respectively.

PrintReceipt1 & PrintReceipt2:

The receipt will be printed as per the gas pumped by the user and provide the return cash value if while invoking the PrintReceipt2 only.

PrintNoReceipt1 & PrintNoReceipt2:

The receipt will not be printed for the user but the return cash will be provided.

Messages:

This class contains the generic messages for both the gas pumps which is involved in the operation.

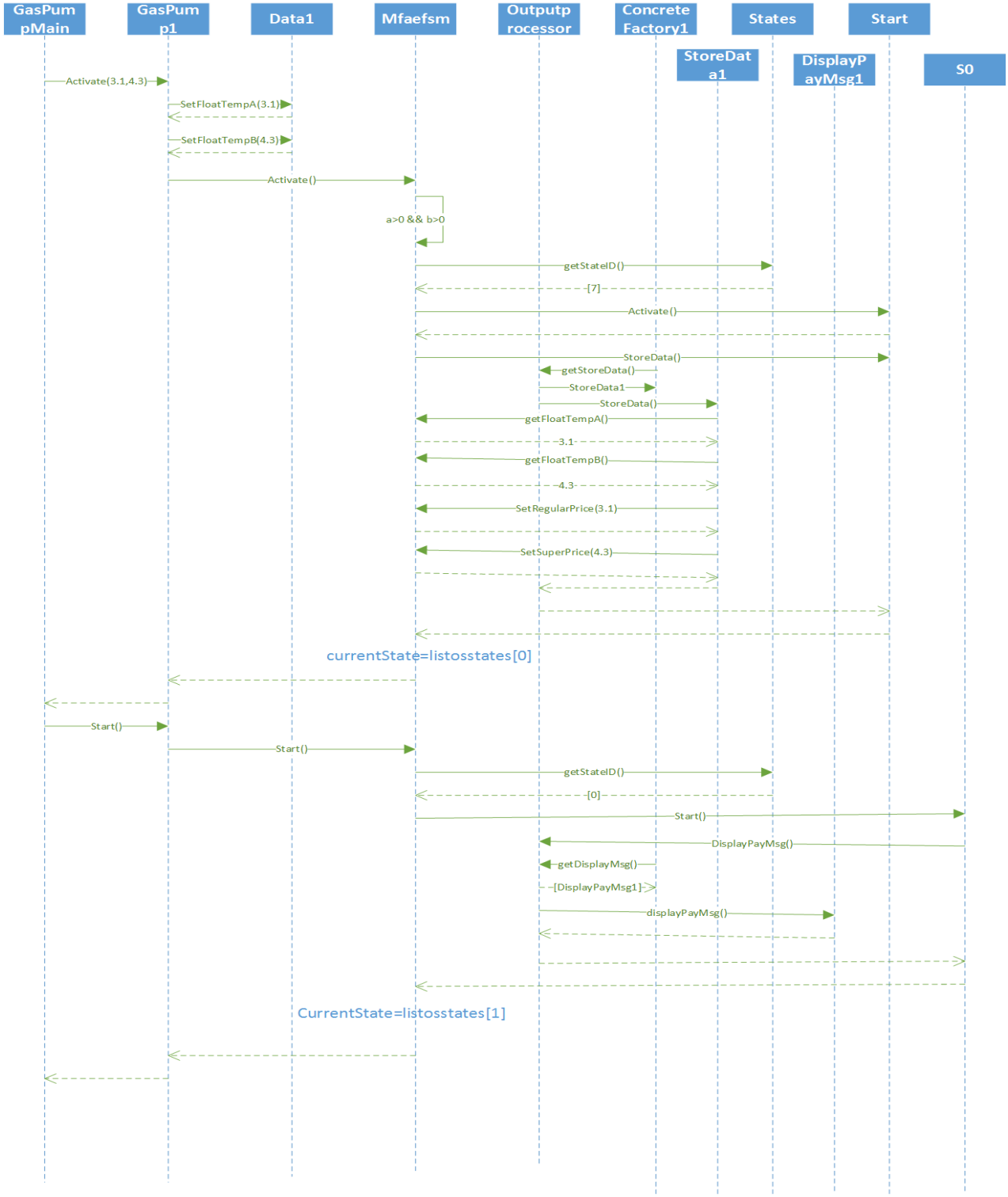
Data Class:

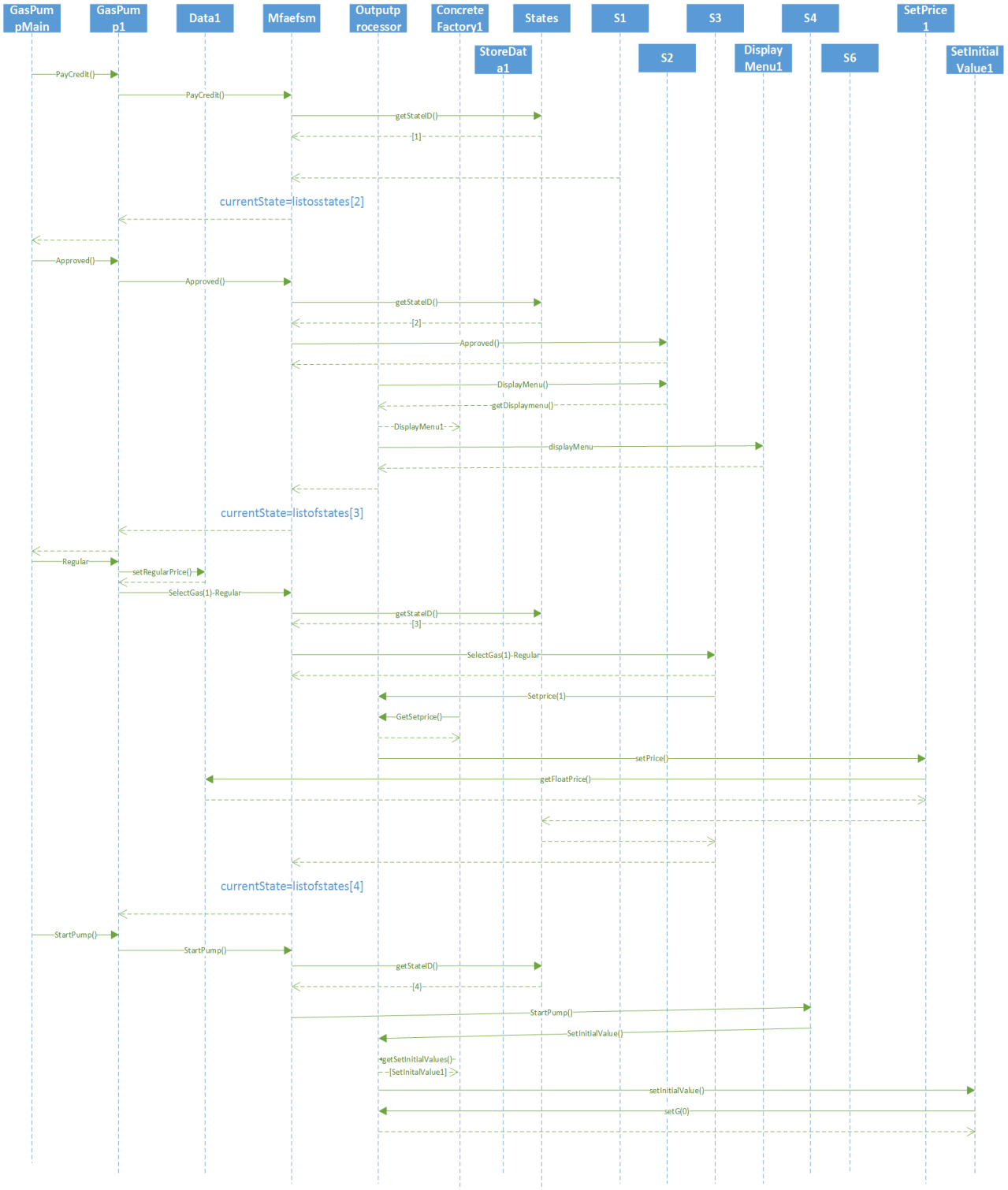
The attributes and the operations are listed and explained in the Source code then and there.

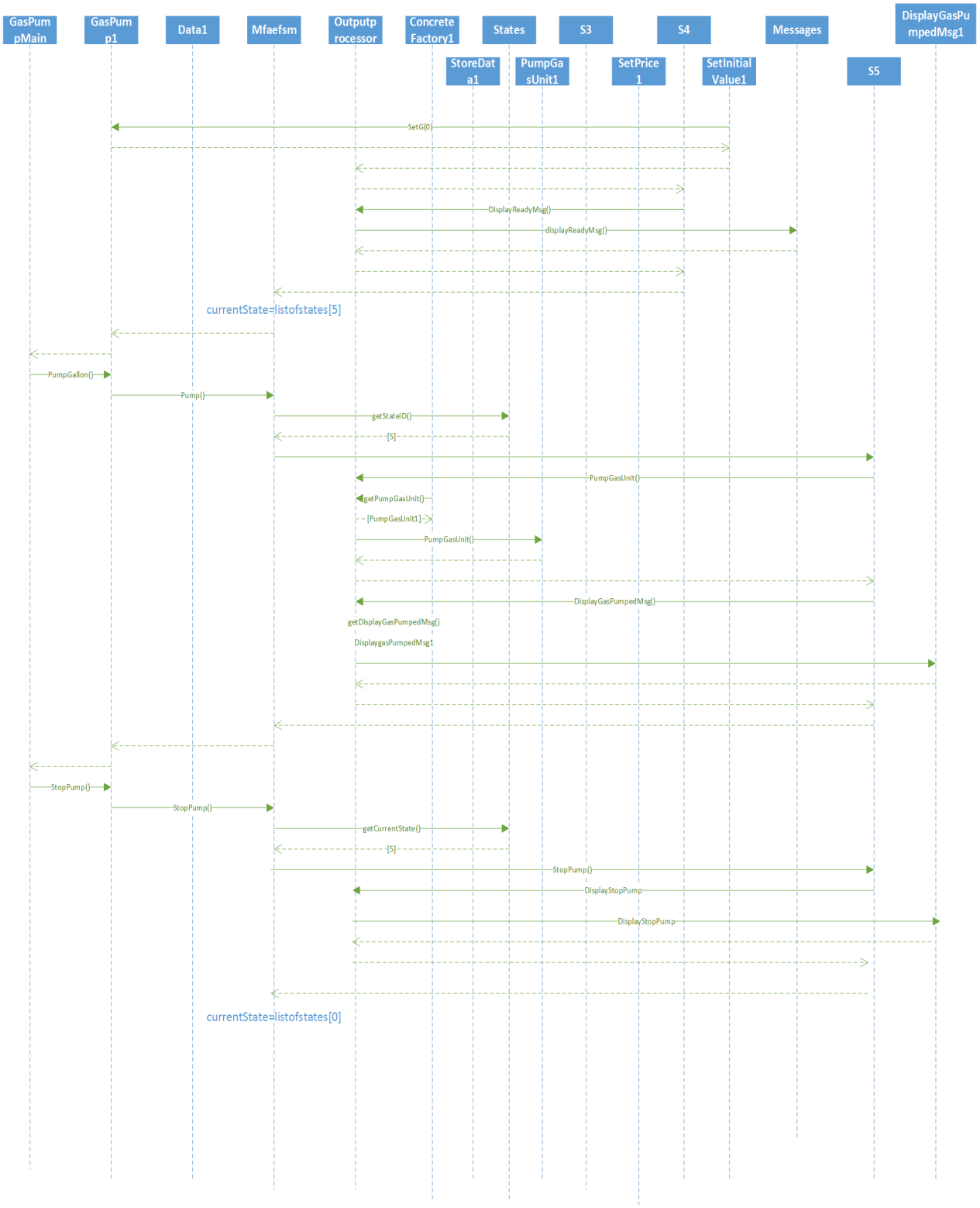
The OO Design patterns are used in the design and it is clearly briefed in the Section 5.

4. Sequence diagram

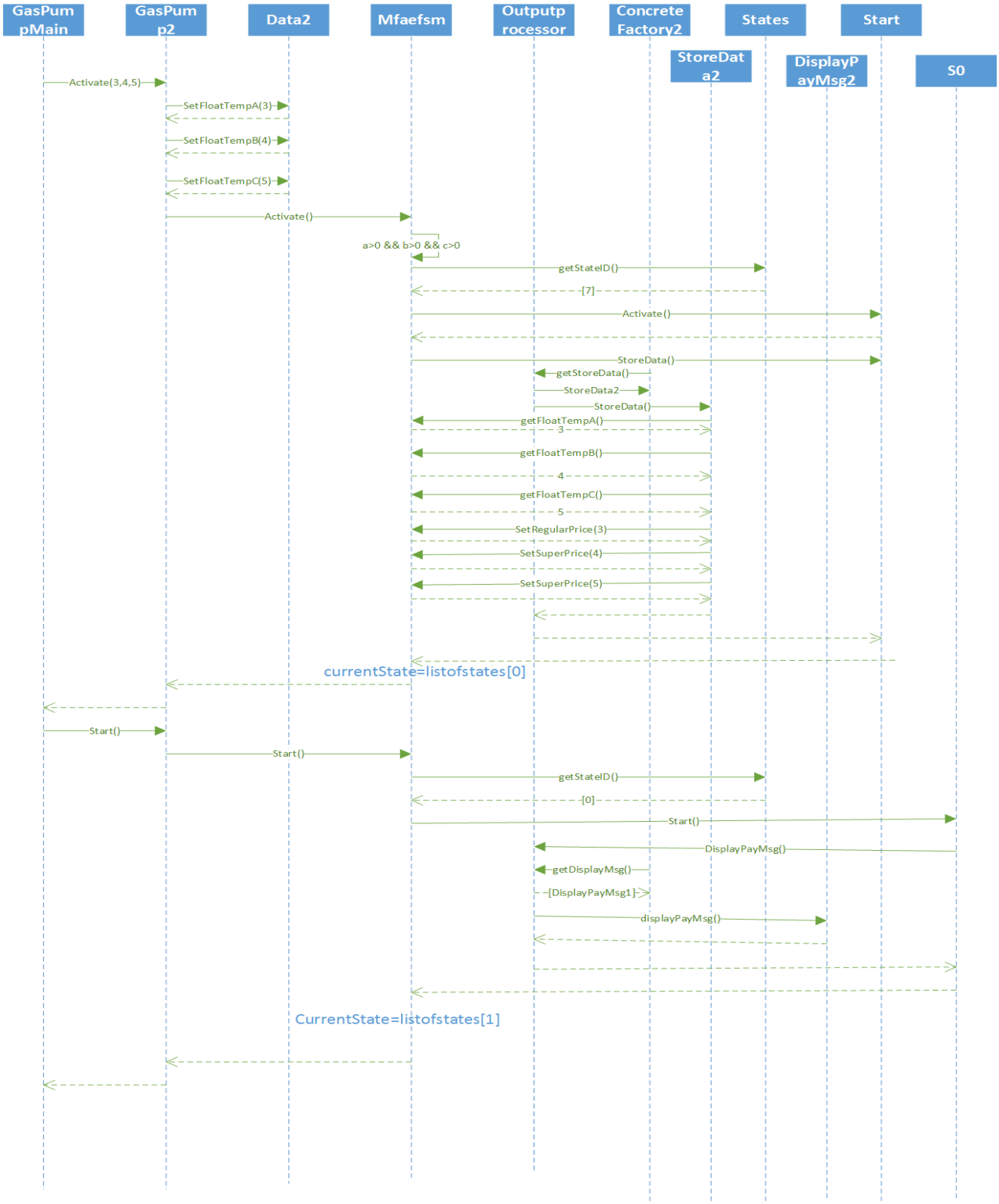
Scenario-I should show how one gallon of Regular gas is disposed in GasPump-1, i.e., the following sequence of operations is issued: Activate(3.1, 4.3), Start(), PayCredit(), Approved(), Regular(), StartPump(), PumpGallon(), StopPump()

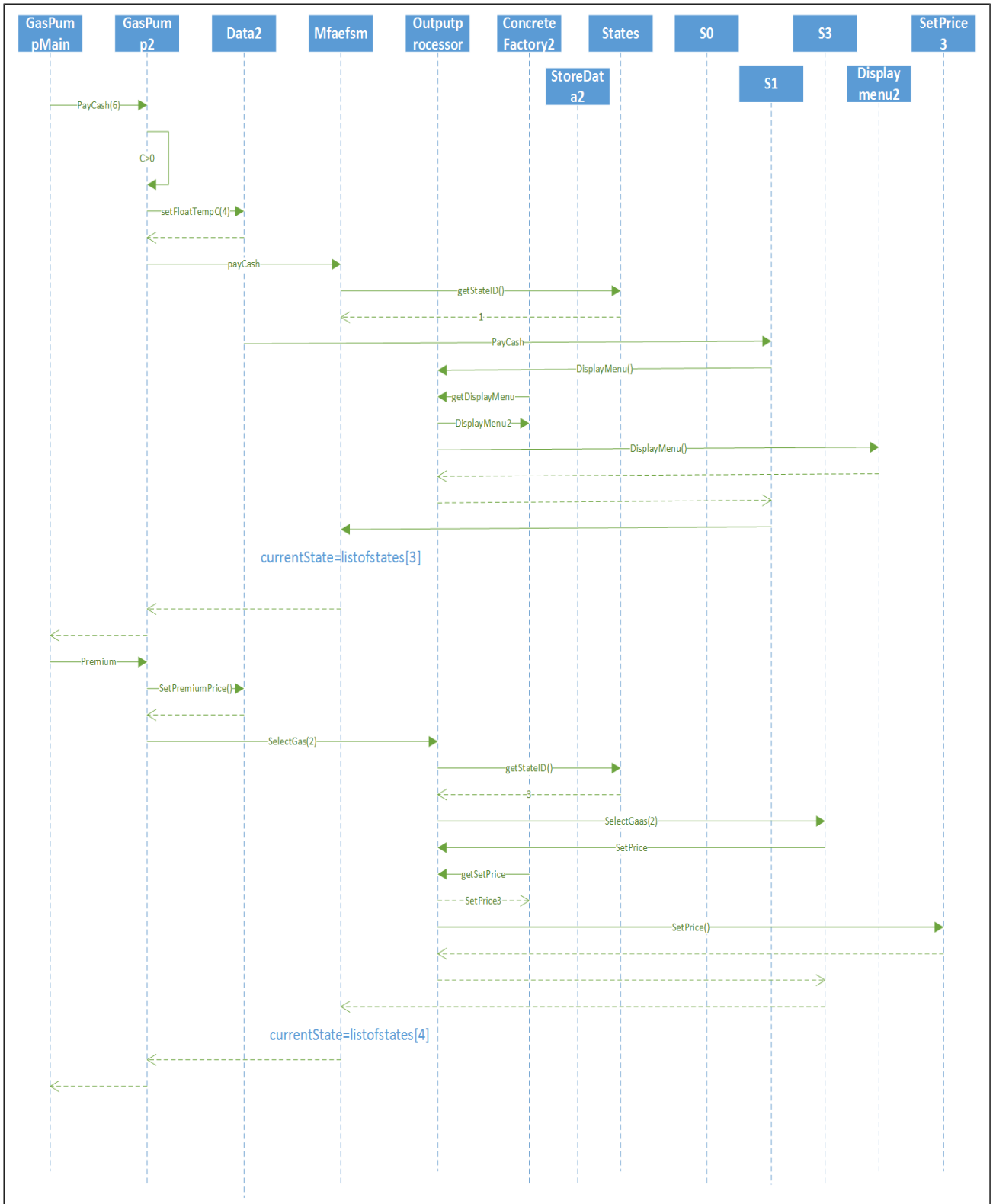


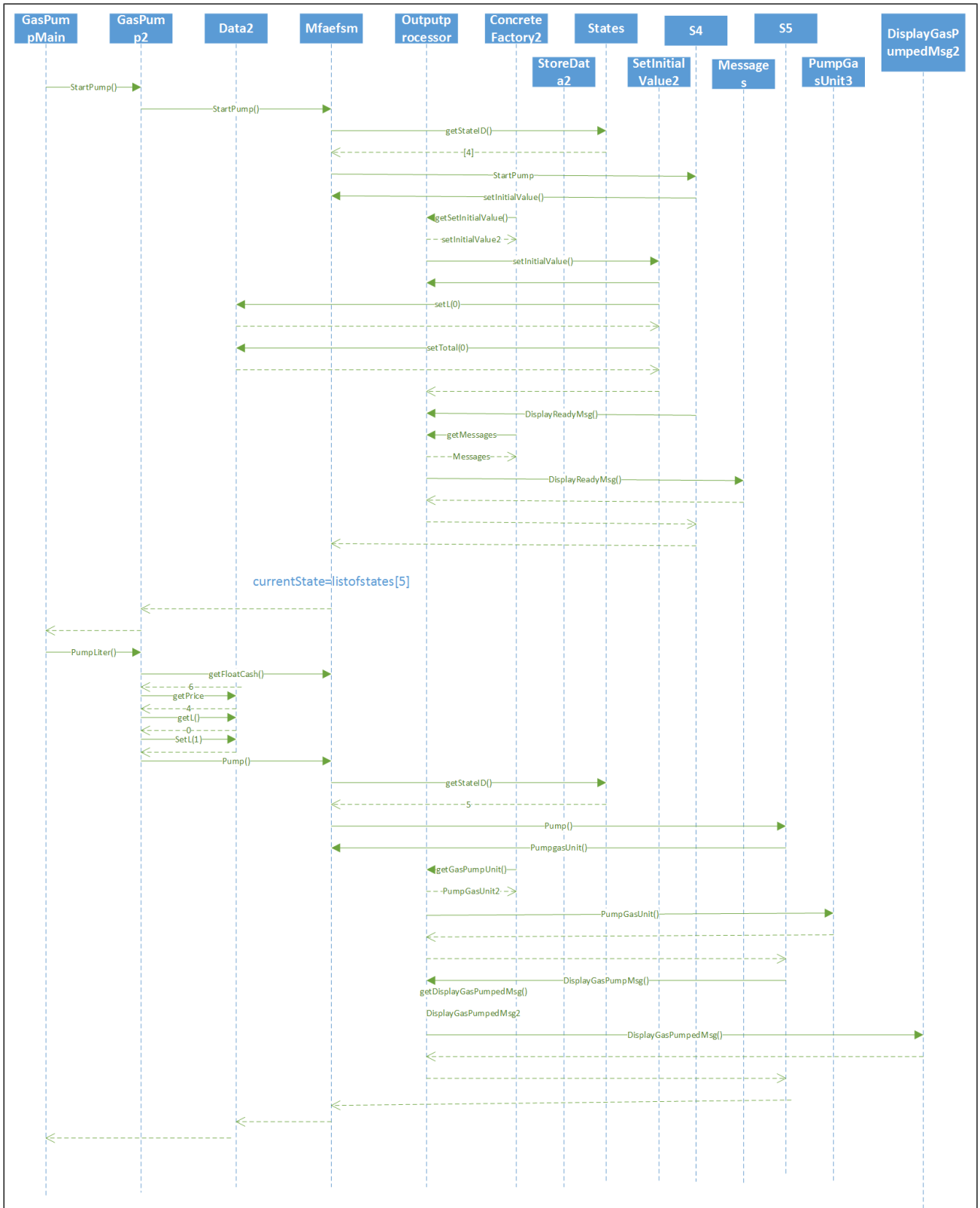

GasPumpMain
GasPump1
Data1
Mfaefsm
Outputprocessor
ConcreteFactory1
States
StoreData1
S1
S2
S3
S4
S6
DisplayMenu1
SetPrice1
SetInitialValue1
PayCredit()
PayCredit()
getStateID()
[1]
currentState=listosstates[2]
Approved()
Approved()
getStateID()
[2]
Approved()
DisplayMenu()
getDisplaymenu()
DisplayMenu1
displayMenu
currentState=listofstates[3]
Regular
setRegularPrice()
SelectGas(1)-Regular
getStateID()
[3]
SelectGas(1)-Regular
Setprice(1)
GetSetprice()
setPrice()
getFloatPrice()
currentState=listofstates[4]
StartPump()
StartPump()
getStateID()
[4]
StartPump()
SetInitialValue()
getSetInitialValues()
[SetInitalValue1]
setInitialValue()
setG(0)

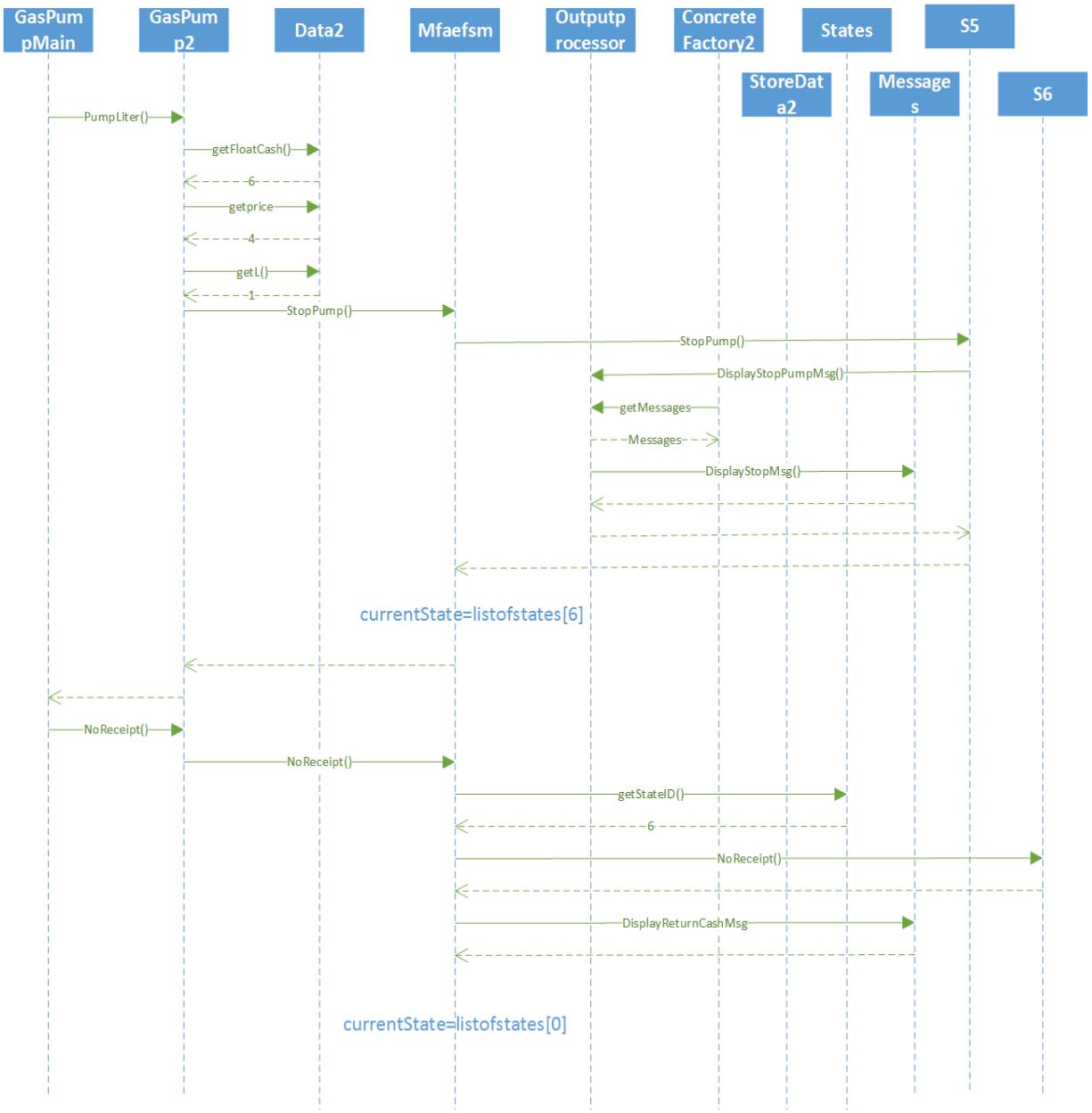

GasPumpMain
GasPump1
Data1
Mfaefsm
Outputprocessor
ConcreteFactory1
States
S3
S4
Messages
DisplayGasPumpedMsg1
StoreData1
PumpGasUnit1
SetPrice1
SetInitialValue1
S5
SetG(0)
DisplayReadyMsg()
displayReadyMsg()
currentState=listofstates[5]
PumpGallon()
Pump()
getStateID()
[5]
PumpGasUnit()
getPumpGasUnit()
[PumpGasUnit1]
PumpGasUnit()
DisplayGasPumpedMsg()
getDisplayGasPumpedMsg()
DisplaygasPumpedMsg1
StopPump()
StopPump()
getCurrentState()
[5]
StopPump()
DisplayStopPump
DisplayStopPump
currentState=listofstates[0]

Scenario-II should show how one liter of Premium gas is disposed in GasPump-2, i.e., the following sequence of operations is issued: Activate(3, 4, 5), Start(), PayCash(6), Premium(), StartPump(), PumpLiter(), PumpLiter(), NoReceipt()



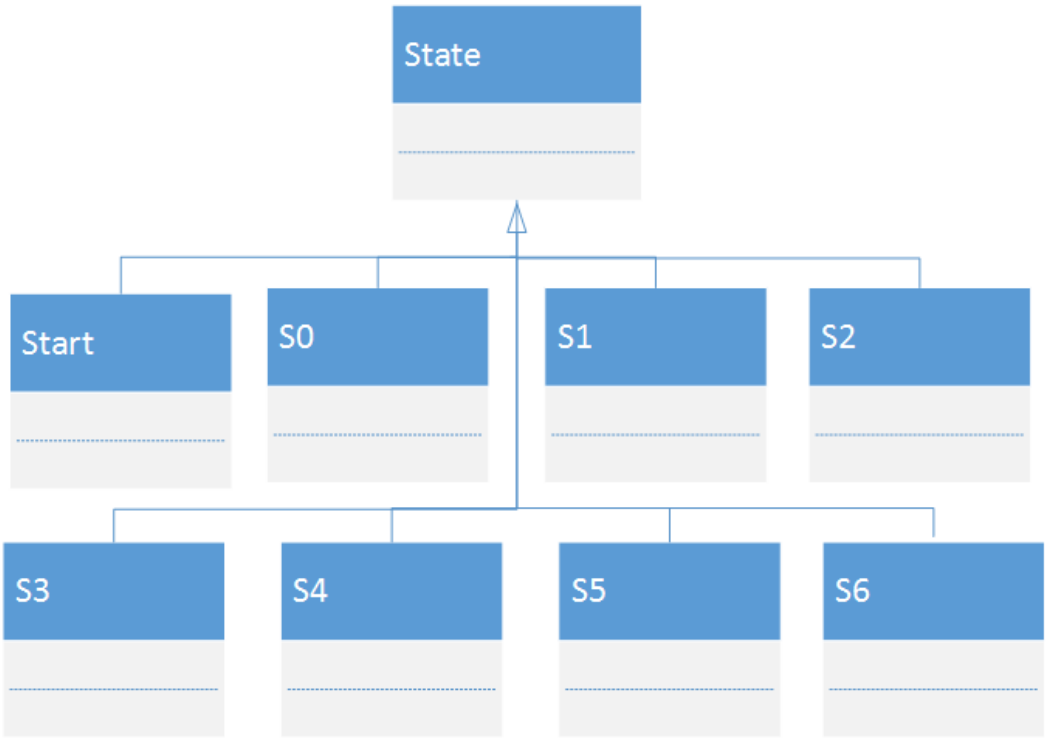

GasPumpMain
GasPump2
Data2
Mfaefsm
Outputprocessor
ConcreteFactory2
States
S0
S3
SetPrice3
StoreData2
S1
Displaymenu2
PayCash(6)
C>0
setFloatTempC(4)
payCash
getStateID()
1
PayCash
DisplayMenu()
getDisplayMenu
DisplayMenu2
DisplayMenu()
currentState=listofstates[3]
Premium
SetPremiumPrice()
SelectGas(2)
getStateID()
3
SelectGaas(2)
SetPrice
getSetPrice
SetPrice3
SetPrice()
currentState=listofstates[4]


GasPumpMain
GasPump2
Data2
Mfaefsm
Outputprocessor
ConcreteFactory2
States
S4
S5
DisplayGasPumpedMsg2
StoreData2
SetInitialValue2
Messages
PumpGasUnit3
StartPump()
StartPump()
getStateID()
[4]
StartPump
setInitialValue()
getSetInitialValue()
setInitialValue2
setInitialValue()
setL(0)
setTotal(0)
DisplayReadyMsg()
getMessages
Messages
DisplayReadyMsg()
currentState=listofstates[5]
PumpLiter()
getFloatCash()
6
getPrice
4
getL()
0
SetL(1)
Pump()
getStateID()
5
Pump()
PumpgasUnit()
getGasPumpUnit()
PumpGasUnit2
PumpGasUnit()
DisplayGasPumpMsg()
getDisplayGasPumpedMsg()
DisplayGasPumpedMsg2
DisplayGasPumpedMsg()


GasPumpMain
GasPump2
Data2
Mfaefsm
Outputprocessor
ConcreteFactory2
States
S5
StoreData2
Messages
S6
PumpLiter()
getFloatCash()
6
getprice
4
getL()
1
StopPump()
StopPump()
DisplayStopPumpMsg()
getMessages
Messages
DisplayStopMsg()
currentState=listofstates[6]
NoReceipt()
NoReceipt()
getStateID()
6
NoReceipt()
DisplayReturnCashMsg
currentState=listofstates[0]

5. The detailed use of 3 patterns in the design

i. State pattern

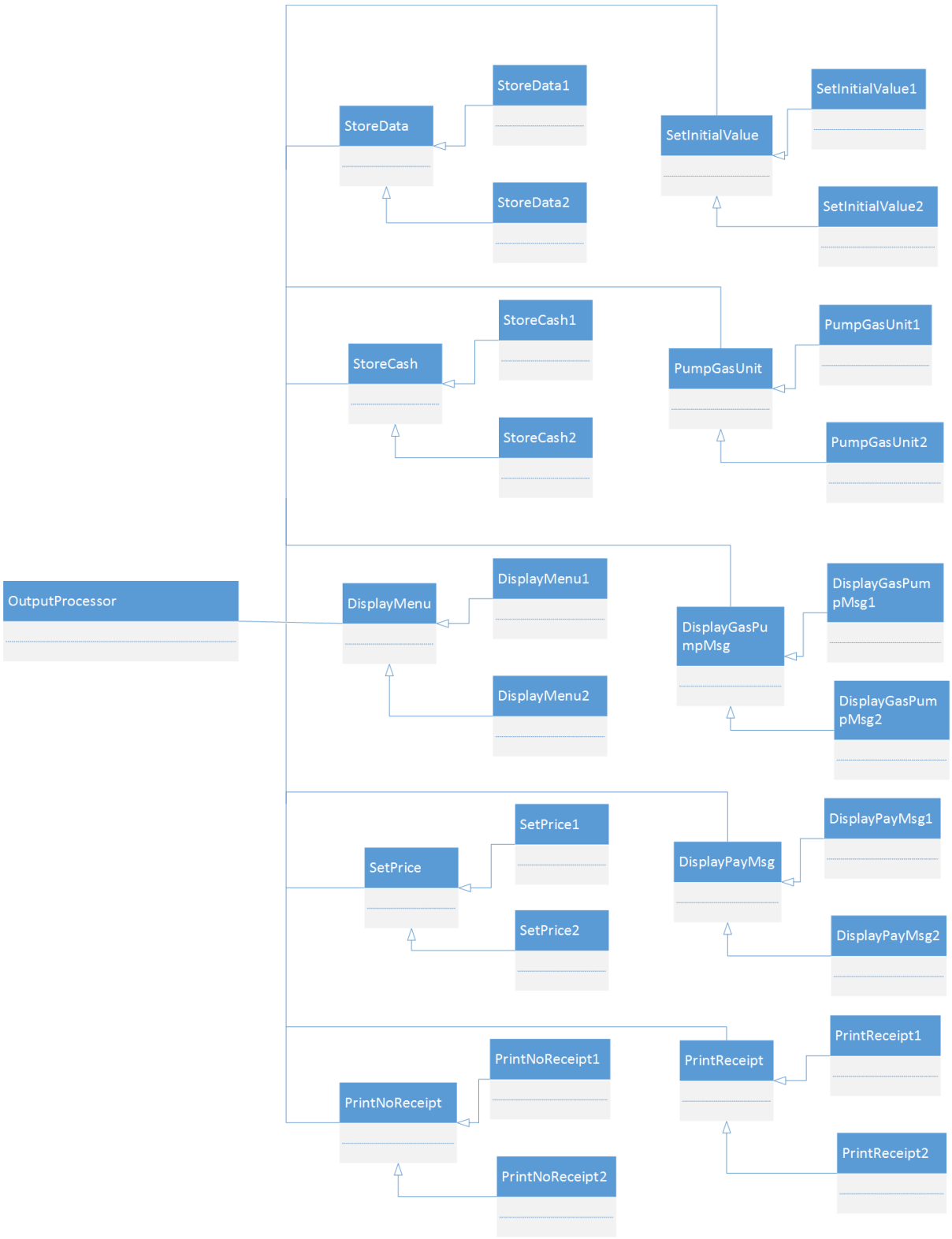


This pattern is implemented in the class States.java

The States class is extended by sub-classes which follow the state pattern

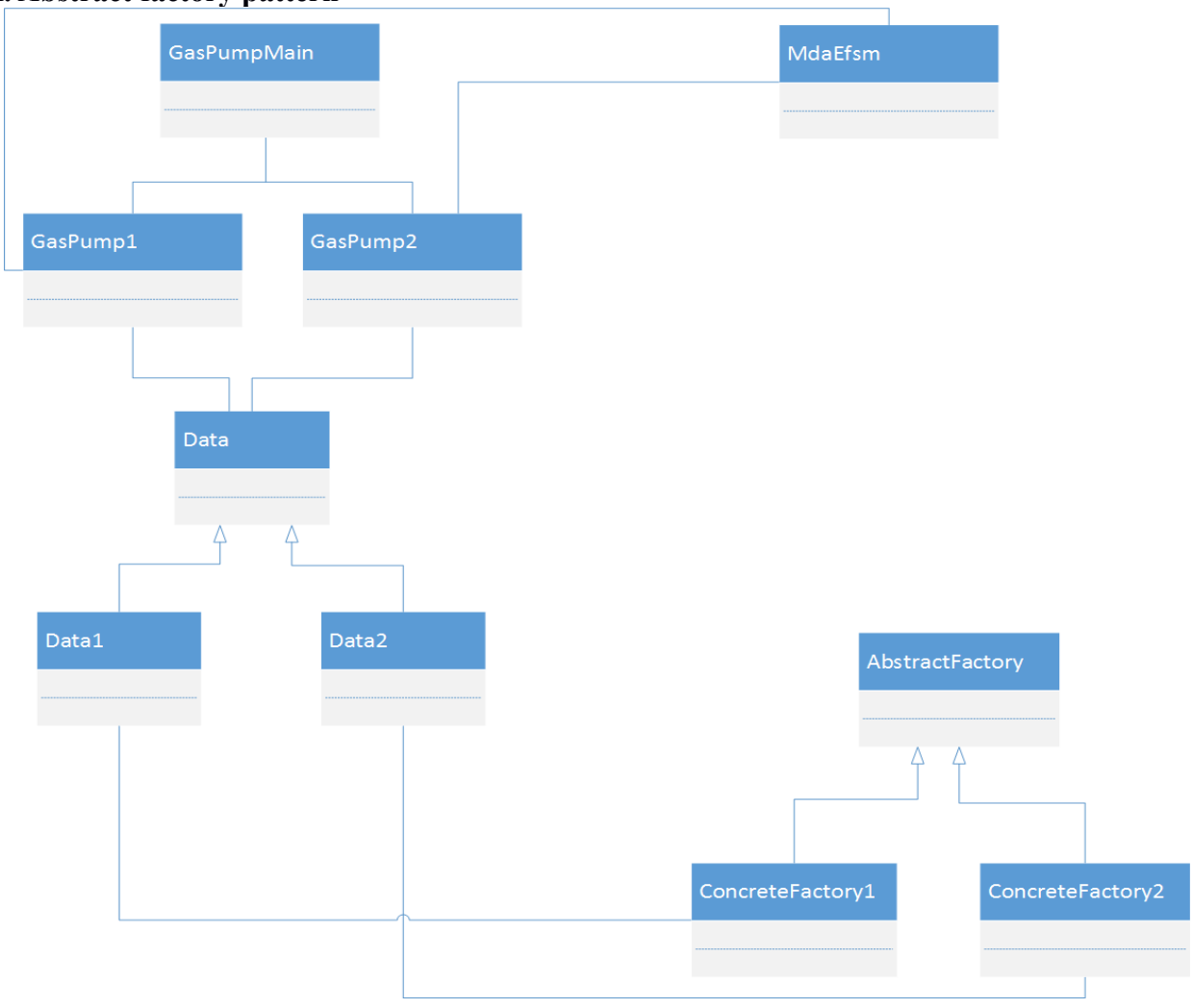
- Start
- S0
- S1
- S2
- S3
- S4
- S5
- S6

ii. Strategy pattern



The class OutputProcessor.java uses the Strategy Pattern in which each GasPump instance will have different actions which call the corresponding method that will be selected via the passed references.
The classes used for the actions are listed below:
• StoreData
• DisplayMenu
• PumpGasUnit
• PrintReceipt
• PrintNoReceipt
• StoreCash
• DisplayPumpedMsg
• SetInitialValues
• SetPrice
• DisplayPayMsg

iii. Abstract factory pattern



The abstract factory pattern is implemented using the class AbstractFactory.java which is then extended by the classes ConcreteFactory1 and ConcreteFactory2.
Methods that return the objects in abstractfactory are listed below:

a) public abstract StoreData getStoreData();
b) public abstract DisplayMenu getDisplayMenu();
c) public abstract PumpGasUnit getPumpGasUnit();
d) public abstract PrintReceipt getPrintReceipt();
e) public abstract PrintNoReceipt getPrintNoReceipt();
f) public abstract StoreCash getStoreCash();
g) public abstract DisplayGasPumpedMsg getDisplayGasPumpedMsg();
h) public abstract SetInitialValues getSetInitialValues();
i) public abstract SetPrice getSetPrice();
j) public abstract Messages getMessages();
k) public abstract DisplayPayMsg getDisplayPayMsg();

6. Source Code:

GasPumpMain.Java

```
package cs586;

/**
*
* @author Sivasenthil Namachivayan A20391478
* This is the main starting class of the project and has the main function of the project.
* The user will be provided a menu to choose the instance of the GasPump that must be run from
the      2 GasPumps.
* Inputs for various arguments will be taken from the user in this class.
*/

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class GasPumpMain {

        public static void main(String[] args) {

      try{ int choice=0;
        BufferedReader buf=new BufferedReader(new InputStreamReader(System.in));

        System.out.println("****CS-586 Software Systems Architecture**** \n Done by Sivasenthil
                                Namachivayan (A20391478) \n");

                                                                    // User can select an instance of
GasPump
        System.out.println("Select the Pump you want to use :");
        System.out.println("1. GasPump-1 :");
        System.out.println("2. GasPump-2 :");

        choice=(int)Float.parseFloat(buf.readLine());
                                // give the choice for the user to choose the gas pump
        System.out.println("You choosed the Gaspump:"+choice);

        switch(choice)
        {
           case 1:
           {
              GasPump1 gasPump1 = new GasPump1();      //create the gaspump1 object

              S0 s0 = new S0();          //Instantiate all the states of the State Pattern
              S1 s1 = new S1();
```

```
        S2 s2 = new S2();
        S3 s3 = new S3();
        S4 s4 = new S4();
        S5 s5 = new S5();
        S6 s6 = new S6();
        Start s7 = new Start();

        MdaEfsm mdaEfsm= new MdaEfsm();          //instantiate MDA
        OutputProcessor outputProcessor = new OutputProcessor();  //instantiate
                                                                  OutputProcessor
        ConcreteFactory1 concreteFactory1 = new ConcreteFactory1();       //instantiate correct
                                                                  ConcreteFactory
        Data data;
        data = concreteFactory1.getData();      //get data of ConcreteFactory
        gasPump1.setMdaEfsm(mdaEfsm);     //set GasPump1 to access MDA
        gasPump1.setFactory(concreteFactory1);       //set GasPump1 to use ConcreteFactory1
        gasPump1.setData(data);
        s0.setOutputProcessor(outputProcessor); //create connections between the states and
IDs
        s0.setStateId(0);

        s1.setOutputProcessor(outputProcessor);
        s1.setStateId(1);

        s2.setOutputProcessor(outputProcessor);
        s2.setStateId(2);

        s3.setOutputProcessor(outputProcessor);
        s3.setStateId(3);

        s4.setOutputProcessor(outputProcessor);
        s4.setStateId(4);

        s5.setOutputProcessor(outputProcessor);
        s5.setStateId(5);

        s6.setOutputProcessor(outputProcessor);
        s6.setStateId(6);

        s7.setOutputProcessor(outputProcessor);
        s7.setStateId(7);

        mdaEfsm.setListOfStates(s0,s1,s2,s3,s4,s5,s6,s7);              //set all states to MDA

        outputProcessor.setData(data);
```

```
outputProcessor.setAbstractFactory(concreteFactory1);        //set ConcreteFactory1

String input=null;
int i;
while(true)
{
   System.out.println("Enter the number for your Operation in sequence:  \n 1.Activate
                        2.Start 3.PayCredit 4.Reject 5.Cancel 6.Approved 7.SuperGas
                        8.RegularGas 9.StartPump 10.PumpGallon 11.StopPump");
   input=buf.readLine();
   i=Integer.parseInt(input);

   switch(i)
   {
      case 1:
                        //Accept inputs for regulargas and supergas and activate them
      System.out.println("Enter the price($) for Regular Gas");
         float a = Float.parseFloat(buf.readLine());
      System.out.println("Enter the price($) for Super Gas");
         float b = Float.parseFloat(buf.readLine());
       gasPump1.Activate(a,b);
       break;

      case 2:
            gasPump1.Start();            //start
          break;

      case 3:
          gasPump1.PayCredit();        //paycredit
          System.out.println("You have Inserted the Credit Card- Waiting for
                              approval\n");
        break;

      case 4:
          gasPump1.Reject();           //Reject
        break;


     case 5:
          gasPump1.Cancel();           //Cancel transaction
        break;

      case 6:
          gasPump1.Approved();         //Card approved
        break;
```

```
            case 7:
                gasPump1.Super();               //super gas
               break;

            case 8:
                gasPump1.Regular();             //regular gas
               break;

            case 9:
                gasPump1.StartPump();           //startpump
                break;

            case 10:
                gasPump1.PumpGallon();      //pumpGallon
                break;

            case 11:
                gasPump1.StopPump();        //stopPump
                break;


            default:
               System.out.println("You have entered the incorrect choice of the operation,
                                        please try again with the correct choice");
         }
      }
   }
case 2:
{
   GasPump2 gasPump2 = new GasPump2();                  //create the gaspump 2 object
   S0 s0 = new S0();          //instantiate all the states of the State pattern
   S1 s1 = new S1();
   S2 s2 = new S2();
   S3 s3 = new S3();
   S4 s4 = new S4();
   S5 s5 = new S5();
   S6 s6 = new S6();
   Start s7 = new Start();

   MdaEfsm mdaEfsm= new MdaEfsm();                              //instantiate MDA
   OutputProcessor outputProcessor = new OutputProcessor();  //instantiate
                                                              OutputProcessor
   ConcreteFactory2 concreteFactory2 = new ConcreteFactory2();       //instantiate
                                                              ConcreteFactory2
```

```
Data data;
data = concreteFactory2.getData();      //get Data2 from ConcreteFactory2

gasPump2.setMdaEfsm(mdaEfsm);     //set gasPump MDA pointer
gasPump2.setFactory(concreteFactory2);       //set gasPump CconcreteFactory2 pointer
gasPump2.setData(data);          //set Data2 to GasPump2

s0.setOutputProcessor(outputProcessor); //instantiate all states of State OO Pattern
s0.setStateId(0);

s1.setOutputProcessor(outputProcessor);
s1.setStateId(1);

s2.setOutputProcessor(outputProcessor);
s2.setStateId(2);

s3.setOutputProcessor(outputProcessor);
s3.setStateId(3);

s4.setOutputProcessor(outputProcessor);
s4.setStateId(4);

s5.setOutputProcessor(outputProcessor);
s5.setStateId(5);

s6.setOutputProcessor(outputProcessor);
s6.setStateId(6);

s7.setOutputProcessor(outputProcessor);
s7.setStateId(7);

mdaEfsm.setListOfStates(s0,s1,s2,s3,s4,s5,s6,s7);   //set all State pattern states to MDA

outputProcessor.setData(data);           //set OutputProcessor data as Data 2
outputProcessor.setAbstractFactory(concreteFactory2);        //set OutputProcessor to
                                                             use ConcreteFactory2
String input=null;
int i;
while(true)
{
   System.out.println("Enter the number for your Operation in sequence::  \n 1:Activate
                       2.Start 3.PayCash 4.Cancel 5.PremiumGas 6.RegularGas
                       7.SuperGas 8.StartPump 9.PumpLiter 10.StopPump 11.Receipt
                       12.NoReceipt");
   input=buf.readLine();
```

```
i=Integer.parseInt(input);

switch(i)
{
   case 1:  //take input of regular gas, super gas and premium gas prices and activate
                  them
         System.out.println("Enter the price($) for Regular Gas");
      float a = Float.parseFloat(buf.readLine());
      System.out.println("Enter the price($) for Premium Gas");
      float b = Float.parseFloat(buf.readLine());
      System.out.println("Enter the price($) for Super Gas");
      float C = Float.parseFloat(buf.readLine());
      gasPump2.Activate(a,b,C);
         break;

   case 2:
         gasPump2.Start();      //start

   case 3:                      //take cash from user and PayCash
         System.out.println("Enter the Cash($) you want to set for this Gaspump
                              Usage :\t");
         int c = (int)Float.parseFloat(buf.readLine());
         gasPump2.PayCash(c);
      break;

   case 4:                      //Cancel transaction
         gasPump2.Cancel();
      break;

   case 5:                      //set premium gas
      gasPump2.Premium();
      break;
   case 6:                      //set regular Gas
         gasPump2.Regular();
      break;

   case 7:                      //set Super Gas
         gasPump2.Super1();
      break;


  case 8:               //startPump
         gasPump2.StartPump();
      break;
```

```
                    case 9:             //pump gas in liters
                        gasPump2.PumpLiter();
                       break;

                    case 10:            //stopPump
                        gasPump2.StopPump();
                        break;

                    case 11:            //Receipt
                        gasPump2.Receipt();
                        break;

                   case 12:             //No receipt to be printed
                        gasPump2.NoReceipt();
                        break;

                    default:
                       System.out.println("You have entered the incorrect choice of the operation,
                                           please try again with the correct choice");

                }
            }
        }
    }
    catch(Exception exc)            //catch exceptions if any

    {
    System.out.println(exc);        //print the exception

    }


      }

}
```

GasPump1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  GasPump1 class consists of functions that will call methods in MdaEfsm class using references of
objects being passed
*/
```

```
public class GasPump1 {

	MdaEfsm mdaEfsm;
	AbstractFactory abstractFactory;
	Data data;

	//set reference to MdaEFSM object
	public void setMdaEfsm(MdaEfsm mdaEfsm) {
		this.mdaEfsm = mdaEfsm;
	}

	//Set reference to CF object
	public void setFactory(ConcreteFactory1 concreteFactory1) {
		this.abstractFactory = concreteFactory1;
	}

	//Set reference to Data object
	public void setData(Data data) {
		this.data = data;
	}

	//verify and set data and forward to MdaEfsm
	public void Activate(float a,float b) {

		if(a>0 && b>0){
			data.setFloatTempA(a);
			data.setFloatTempB(b);

			mdaEfsm.Activate();
		}
	}

	//forward to MdaEfsm
	public void Start() {
		mdaEfsm.Start();
	}

	//forward to MdaEfsm
	public void PayCredit() {
		mdaEfsm.PayCredit();
	}

	//forward to MdaEfsm
	public void Reject() {
		mdaEfsm.Reject();
```

```
	}

	//forward to MdaEfsm
	public void Cancel() {
		mdaEfsm.Cancel();
	}

	//forward to MdaEfsm
	public void Approved() {
		mdaEfsm.Approved();
	}

	//set price as super gas price and forward to MdaEfsm
	public void Super() {

		((Data1)data).setSuperPrice();
		mdaEfsm.SelectGas(2);
	}
	//set price as regular price and forward to MdaEfsm
	public void Regular(){
		((Data1)data).setRegularPrice();
		mdaEfsm.SelectGas(1);
	}
	//forward to MdaEfsm
	public void StartPump() {
		mdaEfsm.StartPump();
	}
	//forward to MdaEfsm
	public void Receipt(){
		mdaEfsm.Receipt();
	}
	//increment count and forward to MdaEfsm
	public void PumpGallon() {
		data.setG((((Data1)data).getG() + 1));
		mdaEfsm.Pump();
	}
	//forward to MdaEfsm and also ask for receipt to be printed
	public void StopPump() {
		mdaEfsm.StopPump();
		mdaEfsm.Receipt();

	}


}
```

GasPump2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  GasPump2 class consists of functions that will call methods in MdaEfsm class using references of objects being passed
*/

public class GasPump2 {

	MdaEfsm mdaEfsm;
	AbstractFactory abstractFactory;
	Data data;

	//set reference to MdaEFSM object
	public void setMdaEfsm(MdaEfsm mdaEfsm) {
		this.mdaEfsm = mdaEfsm;
	}

	//Set reference to CF object
	public void setFactory(ConcreteFactory2 concreteFactory2) {
		this.abstractFactory = concreteFactory2;
	}

	//Set reference to Data object
	public void setData(Data data) {
		this.data = data;
	}

	//verify and store data forward to MdaEfsm
	public void Activate(float a,float b, float C) {

		if(a>0 && b>0){
			((Data2)data).setFloatTempA(a);
			((Data2)data).setFloatTempB(b);
			((Data2)data).setFloatTempC(C);
			mdaEfsm.Activate();
		}

	}

	//forward to MdaEfsm
	public void Start() {
		mdaEfsm.Start();
```

```
	}

	//verify and store data and forward to MdaEfsm
	public void PayCash(int c) {
		if(c>0){
			((Data2)data).setFloatTempC(c);

			mdaEfsm.PayCash();
		}
	}

	//forward to MdaEfsm
	public void Cancel() {
		mdaEfsm.Cancel();
	}

	//set data and forward to MdaEfsm to select gas
	public void Premium() {
		((Data2)data).setPremiumPrice();
		mdaEfsm.SelectGas(2);
	}

	//set data and forward to MdaEfsm to select gas
	public void Regular(){
		((Data2)data).setRegularPrice();
		mdaEfsm.SelectGas(1);
	}

	//set data and forward to MdaEfsm to select gas
	public void Super1(){
		((Data2)data).setSuperPrice1();
		mdaEfsm.SelectGas(3);
	}

	//forward to MdaEfsm
	public void StartPump() {
		mdaEfsm.StartPump();
	}

	//increment counter of liters pumped and forward to MdaEfsm
	public void PumpLiter() {
		if(((Data2)data).getFloatCash() < ( ((Data2)data).getG() + 1) *
((Data2)data).getPrice()){
				mdaEfsm.StopPump();
			}
```

```
                        else{
                                ((Data2)data).setG( ( ((Data2)data).getG() + 1) );
                                mdaEfsm.Pump();
                        }
        }

        //forward to MdaEfsm
        public void StopPump() {
                mdaEfsm.StopPump();


        }

        //forward to MdaEfsm
        public void Receipt(){
                mdaEfsm.Receipt();
        }

        //forward to MdaEfsm
        public void NoReceipt(){
                mdaEfsm.NoReceipt();
        }

}
```

Mdaefsm.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  MdaEfsm class is responsible for changing the state of the system (centralized pattern)
*/
public class MdaEfsm {

        States currentState;
        States[] listOfStates = new States[8];

        public void setState(States states){
                currentState = states;

        }

        //list of states which get passed as parameters and this function stores the states in a list
        public void setListOfStates(States a,States b,States c,States d,States e,States f,States g,States
h){
                listOfStates[0] = a;
```

```
		listOfStates[1] = b;
		listOfStates[2] = c;
		listOfStates[3] = d;
		listOfStates[4] = e;
		listOfStates[5] = f;
		listOfStates[6] = g;
		listOfStates[7] = h;
		this.currentState = listOfStates[7];
	}

	//forwards to same function name of current state if the current state is correct
	public void Activate(){

		int currState = currentState.getStateId();
		switch(currState)
		{
			case 0: break;
			case 1: break;
			case 2: break;
			case 3: break;
			case 4: break;
			case 5: break;
			case 6: break;

			case 7:
					currentState.Activate();
					currentState.StoreData();
					currentState = listOfStates[0];
					break;
		};
	}

	//forwards to same function name of current state if the current state is correct
	public void Start(){
		int currState = currentState.getStateId();
		switch(currState){

			case 0:

				currentState.Start();
				currentState = listOfStates[1];
				break;

			case 1: break;
			case 2: break;
```

```
                case 3: break;
                case 4: break;
                case 5: break;
                case 6: break;

                case 7: break;

        };
}

//forwards to same function name of current state if the current state is correct
public void PayCredit(){
        int currState = currentState.getStateId();
        switch(currState){

                case 0: break;
                case 1:
                                currentState.PayCredit();
                                currentState = listOfStates[2];
                                break;
                case 2: break;
                case 3: break;
                case 4: break;
                case 5: break;
                case 6: break;
                case 7: break;


        };
}

//forwards to same function name of current state if the current state is correct
public void PayCash(){
        int currState = currentState.getStateId();
        switch(currState){

                case 0: break;
                case 1:
                        currentState.PayCash();
                        currentState.StoreCash();
                        currentState.setT(0);
                        currentState = listOfStates[3];
                        break;
                case 2: break;
                case 3: break;
```

```
                case 4: break;
                case 5: break;
                case 6: break;
                case 7: break;


        };
}

//forwards to same function name of current state if the current state is correct
public void Reject(){
        int currState = currentState.getStateId();
        switch(currState){

                case 0: break;
                case 1: break;
                case 2:
                        currentState.Reject();
                        currentState = listOfStates[0];
                        break;
                case 3: break;
                case 4: break;
                case 5: break;
                case 6: break;
                case 7: break;


        };
}

//forwards to same function name of current state if the current state is correct
public void Cancel(){
        int currState = currentState.getStateId();
        switch(currState){

                case 0: break;
                case 1: break;
                case 2: break;
                case 3:
                        currentState.Cancel();
                        currentState = listOfStates[0];
                        break;

                case 4: break;
                case 5: break;
```

```
                case 6: break;
                case 7: break;


        };

    }

    //forwards to same function name of current state if the current state is correct
    public void Approved(){
        int currState = currentState.getStateId();
        switch(currState){

                case 0: break;
                case 1: break;
                case 2:
                    currentState.Approved();
                    currentState.setT(1);
                    currentState = listOfStates[3];
                    break;
                case 3: break;
                case 4: break;
                case 5: break;
                case 6: break;
                case 7: break;


        };

    }

    //forwards to same function name of current state if the current state is correct
    public void StartPump(){
        int currState = currentState.getStateId();
        switch(currState){

                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4:
                    //to5

                            currentState.StartPump();
```

```
                                        currentState = listOfStates[5];
                                    break;
                        case 5: break;
                        case 6: break;
                        case 7: break;


                };

        }

        //forwards to same function name of current state if the current state is correct
        public void Pump(){
                int currState = currentState.getStateId();
                switch(currState){

                        case 0: break;
                        case 1: break;
                        case 2: break;
                        case 3: break;
                        case 4: break;
                        case 5:
                                //to5

                                        currentState.Pump();
                                break;
                        case 6: break;
                        case 7: break;


                };

        }

        //forwards to same function name of current state if the current state is correct
        public void StopPump(){
                int currState = currentState.getStateId();
                switch(currState){

                        case 0: break;
                        case 1: break;
                        case 2: break;
                        case 3: break;
                        case 4: break;
                        case 5:
```

```
                        //to6
                        currentState.StopPump();
                        currentState = listOfStates[6];
                        break;
                case 6: break;
                case 7: break;


        };
}

//forwards to same function name of current state if the current state is correct
public void SelectGas(int g){
        int currState = currentState.getStateId();
        switch(currState){

                case 0: break;
                case 1: break;
                case 2: break;
                case 3:
                        currentState.SelectGas(g);
                        //currentState.SetPrice(g);
                        currentState = listOfStates[4];
                        break;
                case 4: break;
                case 5: break;
                case 6: break;
                case 7: break;


        };

}

//forwards to same function name of current state if the current state is correct
public void Receipt(){
        int currState = currentState.getStateId();
        switch(currState){

                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4: break;
                case 5: break;
```

```
                        case 6:
                                //to0
                                currentState.Receipt();
                                currentState = listOfStates[0];
                                break;
                        case 7: break;


                };

        }

        //forwards to same function name of current state if the current state is correct
        public void NoReceipt(){
                int currState = currentState.getStateId();
                switch(currState){

                        case 0: break;
                        case 1: break;
                        case 2: break;
                        case 3: break;
                        case 4: break;
                        case 5: break;
                        case 6:
                                //to0
                                currentState.NoReceipt();
                                currentState = listOfStates[0];

                                break;
                        case 7: break;


                };

        }

        //this is only for testing purposes.. to check the current state of the machine
        public int getCurrentStateId() {
                return currentState.getStateId();
        }

}
```

OutputProcessor.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class processes and requests relevant concrete factory objects from the abstract factory and
calls them
*
*/
public class OutputProcessor {

	AbstractFactory abstractFactory;
	Data data;
	Messages messages;

	//set the pointer abstract factory to the ConcreteFactory object argument
	public void setAbstractFactory(AbstractFactory abstractFactory) {
		this.abstractFactory = abstractFactory;
	}
	//set the pointer to the passed data object argument
	public void setData(Data data) {
		this.data = data;
	}

	//request and return the correct concrete object and call it
	public void StoreData(){
		StoreData storeData;

		storeData = abstractFactory.getStoreData();
		storeData.setData(data);
		storeData.storeData();
	}

	//request and return the correct concrete object and call it
	public void StoreCash(){
		StoreCash storeCash = abstractFactory.getStoreCash();
		storeCash.setCash(data);
		storeCash.storeCash();
	}

	//request and return the correct concrete object and call it
	public void DisplayMenu(){
		DisplayMenu displayMenu;
		displayMenu = abstractFactory.getDisplayMenu();
```

```
            displayMenu.displayMenu();
        }

        //request and return the correct concrete object and call it
        public void DisplayPayMsg(){
            DisplayPayMsg displayPayMsg;
            displayPayMsg = abstractFactory.getDisplayPayMsg();
            displayPayMsg.displayPayMsg();
        }

        //request and return the correct concrete object and call it
        public void PumpGasUnit(){
            PumpGasUnit pumpGasUnit;
            pumpGasUnit = abstractFactory.getPumpGasUnit();
            pumpGasUnit.pumpGasUnit();
        }

        //request and return the correct concrete object and call it
        public void PrintReceipt(){
            PrintReceipt printReceipt;
            printReceipt = abstractFactory.getPrintReceipt();
            printReceipt.printReceipt(data);
        }

        //request and return the correct concrete object and call it
            public void PrintNoReceipt(){
                PrintNoReceipt printNoReceipt;
                printNoReceipt = abstractFactory.getPrintNoReceipt();
                printNoReceipt.printNoReceipt(data);
            }
        //request and return the correct concrete object and call it
        public void DisplayRejectMsg(){
            messages = abstractFactory.getMessages();
            messages.displayRejectMsg();
        }

        //request and return the correct concrete object and call it
        public void SetPrice(int g){
            SetPrice setPrice = abstractFactory.getSetPrice();
            setPrice.setPrice(data);
        }

        //request and return the correct concrete object and call it
        public void DisplayReadyMsg(){
            messages = abstractFactory.getMessages();
```

```
                messages.displayReadyMsg();
        }

        //request and return the correct concrete object and call it
        public void SetInitialValues(){
                this.data.setG(0);
                this.data.setTotal(0);
        }

        //request and return the correct concrete object and call it
        public void DisplayGasPumpedMsg(){
                DisplayGasPumpedMsg displayGasPumpedMsg =
abstractFactory.getDisplayGasPumpedMsg();
                displayGasPumpedMsg.displayGasPumpedMsg(data);
        }

        //request and return the correct concrete object and call it
        public void DisplayCancelMsg(){
                messages = abstractFactory.getMessages();
                messages.displayCancelMsg();
        }

        //request and return the correct concrete object and call it
        public void DisplayStopMsg() {
                messages = abstractFactory.getMessages();
                messages.displayStopMsg();
        }


}
```

Data.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class is extended by data 1 & 2  for GasPump 1 & 2 respectively and consists of all methods
used.
*
*/

public abstract class Data {

        public float getfloatTempA() {
                return 0;
```

```
	}
	public void setfloatTempA(float tempA) {}

	public float getfloatTempC() {
		return 0;
	}
	public void setfloatTempC(float tempC) {}

	public int getG() {
		return 0;
	}
	public void setG(int g) {}

	public int getIntPrice() {
		return 0;
	}
	public void setIntPrice(int price) {}

	public int getIntCash() {
		return 0;
	}
	public void setIntCash(int cash) {}

	public float getFloatTempA() {
		return 0;
	}
	public void setFloatTempA(float tempA) {}

	public float getFloatTempB() {
		return 0;
	}
	public void setFloatTempB(float tempB) {
	}

	public float getPriceRegular() {
		return 0;
	}
	public void setPriceRegular(float priceRegular) {
	}

	public float getPriceSuperPremium() {
		return 0;
	}
	public void setPriceSuperPremium(float priceSuperPremium) {
	}
```

```
	public float getPriceSuper() {
		return 0;
	}
	public void setPriceSuper(float priceSuper) {
	}

	public float getPriceSuper1() {
		return 0;
	}
	public void setPriceSuper1(float priceSuper1) {
	}

	public float getFloatTempC() {
		return 0;
	}
	public void setFloatTempC(float tempC) {
	}


	public float getFloatCash() {
		return 0;
	}
	public void setFloatCash(float cash) {}

	public void setTotal(int total){}
	public float getTotal(){
		return 0;
	}

}
```

Data1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class implements data elements for GasPump1 and includes methods for manipulation
*
*/
public class Data1 extends Data {

	static float tempA;                                    //stores user input
	static float tempB;                                    //stores user input
	static int t;                                          //cash(0) or credit(1)
```

```
        static int g;                                   //units of gas pumped
        static float total;                             //total for printing receipt
        static float price;                             //price of gas selected for
pumping
        static float priceRegular;                  //stores price of regular gas
        static float priceSuper;                    //stores price of super gas

        public float getFloatTempA() {
                return tempA;
        }
        public void setFloatTempA(float tempA) {
                Data1.tempA = tempA;
        }
        public float getFloatTempB() {
                return tempB;
        }
        public void setFloatTempB(float tempB) {
                Data1.tempB = tempB;
        }
        public int getT() {
                return t;
        }
        public void setT(int t) {
                Data1.t = t;
        }
        public int getG() {
                return g;
        }
        public void setG(int g) {
                Data1.g = g;
        }
        public float getTotal(){
                Data1.total = (Data1.g * Data1.price);          //calculates total from current g
and price and returns
                return Data1.total;
        }
        public float getPriceRegular() {
                return priceRegular;
        }
        public void setPriceRegular(float priceRegular) {
                Data1.priceRegular = priceRegular;
        }
        public float getPriceSuperPremium() {
                return priceSuper;
```

```
	}
	public void setPriceSuperPremium(float priceSuperPremium) {
		Data1.priceSuper = priceSuperPremium;

	}
	public void setTotal(float total) {
		Data1.total = total;
	}
	public float getPrice() {
		return price;
	}
	public void setRegularPrice() {
		Data1.price = priceRegular;
	}
	public void setSuperPrice() {
		Data1.price = priceSuper;
	}

}
```

Data2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class implements data elements for GasPump3 and includes methods for manipulation
*
*/
public class Data2 extends Data {

	static float tempA;				//stores user input
	static float tempB;				//stores user input
	static float tempC;				//stores user input
	static int g;					//units of gas pumped
	static int t;					//cash(0) or credit(1)
	static float total;				//total for printing receipt
	static float cash;				//stores cash paid by user
	static float price;				//stores price of gas which is selected for
pumping


	static float priceRegular;		//stores price of regular gas
	static float pricePremium;		//stores price of premium gas
	static float priceSuper1;		//stores price of super gas
```

```
	public float getFloatTempA() {
		return tempA;
	}
	public void setFloatTempA(float tempA) {
		Data2.tempA = tempA;
	}
	public float getFloatTempB() {
		return tempB;
	}
	public void setFloatTempB(float tempB) {
		Data2.tempB = tempB;
	}
	public float getFloatTempC() {
		return tempC;
	}
	public void setFloatTempC(float tempC) {
		Data2.tempC = tempC;
	}
	public int getG() {
		return g;
	}
	public void setG(int l) {
		Data2.g = l;
	}
	public float getTotal(){
		Data2.total = (Data2.g * Data2.price);		//calculates total from current g
and price and returns
		return Data2.total;
	}
	public void setTotal(float total){
		Data2.total= total;
	}
	public float getPriceRegular() {
		return priceRegular;
	}
	public void setPriceRegular(float priceRegular) {
		Data2.priceRegular = priceRegular;

	}
	public float getFloatCash(){
		return cash;
	}
	public float getPriceSuperPremium() {
		return pricePremium;
	}
```

```
	public void setPriceSuperPremium(float priceSuperPremium) {
		Data2.pricePremium = priceSuperPremium;

	}
	public void setFloatCash(float cash) {
		Data2.cash = cash;
	}
	public float getPrice() {
		return price;
	}

	public float getPriceSuper1() {
		return priceSuper1;
	}
	public void setPriceSuper1(float priceSuper1) {
		Data2.priceSuper1 = priceSuper1;

	}

	public void setPremiumPrice() {
		Data2.price = pricePremium;
	}
	public void setRegularPrice() {
		Data2.price = priceRegular;
	}
	public void setSuperPrice1() {
		Data2.price = priceSuper1;

	}

}
```

States.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class is used by all state objects and includes methods that are going be called by the state objects
*
*/
public class States {
	int stateId;
	OutputProcessor outputProcessor;    //pointer to OP
```

```
	public int getStateId(){
		return stateId;
	}
	public void setStateId(int stateId) {
		this.stateId = stateId;
	}

	public void setOutputProcessor(OutputProcessor outputProcessor) {
		this.outputProcessor = outputProcessor;
	}

	public void Activate(){};	//Start
	public void Start(){};			//S0
	public void PayCredit(){};		//S1
	public void PayCash(){};		//S1
	public void Reject(){};				//S2
	public void Cancel(){};				//S3
	public void Approved(){};		//S2
	public void StartPump(){};		//S4
	public void Pump(){};				//S5
	public void StopPump(){};		//S5
	public void SelectGas(int G){};	//S3
	public void Receipt(){};		//S6
	public void NoReceipt(){}		//S6
	public void StoreData(){}		//S1
	public void StoreCash(){}		//S1
	public void setT(int i) {}		//S1,S2
	public void SetPrice(int g) {}//S3
	public void ReturnCash() {} //S6
}
```

Start.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class extends States and implements methods for Start state
*
*/
public class Start extends States {

	public void Activate(){
	}
	//forward to outputprocessor
```

```
	public void StoreData(){
		outputProcessor.StoreData();
	}
}
```

S0.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class extends States and implements methods for S0 state
*
*/
public class S0 extends States{
	//forward to outputprocessor
	public void Start(){
		outputProcessor.DisplayPayMsg();
	}
}
```

S1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class extends States and implements methods for S1 state
*
*/
public class S1 extends States {
	public void PayCredit(){
	}

	//forward to outputprocessor
	public void PayCash(){
		//outputProcessor.SetT(0);
		outputProcessor.DisplayMenu();

	}

	//forward to outputprocessor
	public void StoreCash(){
		outputProcessor.StoreCash();

	}
}
```

S2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class extends States and implements methods for S2 state
*
*/
public class S2 extends States {

	//forward to outputprocessor
	public void Approved(){
		//outputProcessor.SetT(1);
		outputProcessor.DisplayMenu();
	}

	//forward to outputprocessor
	public void Reject(){
		outputProcessor.DisplayRejectMsg();
	}
}
```

S3.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class extends States and implements methods for S3 state
*
*/
public class S3 extends States {

	//forward to outputprocessor
	public void Cancel(){
		outputProcessor.DisplayCancelMsg();
		outputProcessor.PrintNoReceipt();
	}

	//forward to outputprocessor
	public void SelectGas(int g){
		outputProcessor.SetPrice(g);
	}
}
```

S4.java

```
package cs586;
```

```
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class extends States and implements methods for S4 state
*
*/
public class S4 extends States {

	//forward to outputprocessor
	public void StartPump(){
		outputProcessor.SetInitialValues();
		outputProcessor.DisplayReadyMsg();
	}

}
```

S5.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class extends States and implements methods for S5 state
*
*/
public class S5 extends States {

	//forward to outputprocessor
	public void Pump(){
		outputProcessor.PumpGasUnit();
		outputProcessor.DisplayGasPumpedMsg();
	}

	//forward to outputprocessor
	public void StopPump(){
		outputProcessor.DisplayStopMsg();
	}
}
```

S6.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class extends States and implements methods for S6 state
*
```

```
*/
public class S6 extends States {

	//forward to outputprocessor
	public void Receipt(){
		outputProcessor.PrintReceipt();
	}

	public void NoReceipt(){
		System.out.println("Transaction over, No receipt will be printed");
		outputProcessor.PrintNoReceipt();
	}
}
```

AbstractFactory.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  This class includes all the methods that will be implemented by the ConcreteFactory inheriting it.
*
*/
public abstract class AbstractFactory {
	public abstract StoreData getStoreData();
	public abstract DisplayMenu getDisplayMenu();
	public abstract PumpGasUnit getPumpGasUnit();
	public abstract PrintReceipt getPrintReceipt();
	public abstract PrintNoReceipt getPrintNoReceipt();
	public abstract StoreCash getStoreCash();
	public abstract DisplayGasPumpedMsg getDisplayGasPumpedMsg();
	public abstract SetInitialValues getSetInitialValues();
	public abstract SetPrice getSetPrice();
	public abstract Messages getMessages();
	public abstract DisplayPayMsg getDisplayPayMsg();
}
```

ConcreteFactory1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Includes all the methods from AbstractFactory which will be used by GasPump1 and its methods
to get specific concrete objects
*
*/
```

```
public class ConcreteFactory1 extends AbstractFactory {

	/*private static final ReturnCash ReturnCash2 = null;*/
	//get object of type Data1
	public Data1 getData() {
		Data1 data1 = new Data1();

		return data1;
	}

	//get object of type StoreData1
	@Override
	public StoreData getStoreData() {
		StoreData1 storedata1 = new StoreData1();

		return storedata1;
	}

	//get object of type DisplayMenu1
	@Override
	public DisplayMenu getDisplayMenu() {
		DisplayMenu1 displayMenu1 = new DisplayMenu1();

		return displayMenu1;
	}

	//get object of type PumpGasUnit1
	@Override
	public PumpGasUnit getPumpGasUnit() {
		PumpGasUnit1 pumpGasUnit1 = new PumpGasUnit1();

		return pumpGasUnit1;
	}

	//get object of type PrintReceipt1
	@Override
	public PrintReceipt getPrintReceipt() {
		PrintReceipt1 printReceipt1 = new PrintReceipt1();

		return printReceipt1;
	}
	//get object of type PrintNoReceipt1
	@Override
	public PrintNoReceipt getPrintNoReceipt() {
		PrintNoReceipt1 printNoReceipt1 = new PrintNoReceipt1();
```

```
		return printNoReceipt1;
	}

	//will return null because cash is not used in GP1
	@Override
	public StoreCash getStoreCash() {
		return null;
	}

	//get object of type GasPumpedMsg1
	@Override
	public DisplayGasPumpedMsg getDisplayGasPumpedMsg() {
		DisplayGasPumpedMsg1 displayGasPumpedMsg1 = new
DisplayGasPumpedMsg1();

		return displayGasPumpedMsg1;
	}

	//get object of type SetInitialValues1
	@Override
	public SetInitialValues getSetInitialValues() {
		SetInitialValues1 setInitialValues1 = new SetInitialValues1();

		return setInitialValues1;

	}

	//get object of type SetPrice1
	@Override
	public SetPrice getSetPrice() {
		SetPrice1 setPrice1 = new SetPrice1();

		return setPrice1;
	}

	//get object of type Messages
	@Override
	public Messages getMessages() {
		// TODO Auto-generated method stub
		Messages messages= new Messages();
		return messages;
	}
```

```
	@Override
	public DisplayPayMsg getDisplayPayMsg() {
		// TODO Auto-generated method stub
		DisplayPayMsg1 displayPayMsg1 = new DisplayPayMsg1();
		return displayPayMsg1;
	}

}
```

ConcreteFactory2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Includes all the methods from AbstractFactory which will be used by GasPump2 and its methods to get specific concrete objects
*
*/
public class ConcreteFactory2 extends AbstractFactory {


	//get object of type Data2
	public Data getData() {
		Data2 data2 = new Data2();

		return data2;
	}

	//get object of type StoreData2
	@Override
	public StoreData getStoreData() {
		StoreData2 storedata2 = new StoreData2();

		return storedata2;
	}

	//get object of type DisplayMenu2
	@Override
	public DisplayMenu getDisplayMenu() {
		DisplayMenu2 displayMenu2= new DisplayMenu2();

		return displayMenu2;
	}

	//get object of type PumpGasUnit2
	@Override
```

```
	public PumpGasUnit getPumpGasUnit() {
		PumpGasUnit2 pumpGasUnit2 = new PumpGasUnit2();

		return pumpGasUnit2;
	}

	//get object of type PrintReceipt2
	@Override
	public PrintReceipt getPrintReceipt() {
		PrintReceipt2 printReceipt2 = new PrintReceipt2();

		return printReceipt2;
	}

	//get object of type PrintNoReceipt2
	@Override
	public PrintNoReceipt getPrintNoReceipt() {
		PrintNoReceipt2 printNoReceipt2 = new PrintNoReceipt2();

		return printNoReceipt2;
	}

	//get object of type StoreCash2
	@Override
	public StoreCash getStoreCash() {
		StoreCash2 storeCash2 = new StoreCash2();

		return storeCash2;
	}

	//get object of type DisplayGasPumpedMsg2
	@Override
	public DisplayGasPumpedMsg getDisplayGasPumpedMsg() {
		DisplayGasPumpedMsg2 displayGasPumpedMsg2 = new
DisplayGasPumpedMsg2();

		return displayGasPumpedMsg2;
	}

	//get object of type SetInitialValues2
	@Override
	public SetInitialValues getSetInitialValues() {
		SetInitialValues2 setInitialValues2= new SetInitialValues2();

		return setInitialValues2;
```

```
	}

	//get object of type SetPrice2
	@Override
	public SetPrice getSetPrice() {
		SetPrice2 setPrice2 = new SetPrice2();

		return setPrice2;
	}

	//get object of type Messages
	@Override
	public Messages getMessages() {
		// TODO Auto-generated method stub
		Messages messages= new Messages();
		return messages;
	}

	//get object of type DisplayPayMsg2
	@Override
	public DisplayPayMsg getDisplayPayMsg() {
		// TODO Auto-generated method stub
		DisplayPayMsg2 displayPayMsg2 = new DisplayPayMsg2();
		return displayPayMsg2;
	}


}
```

Messages.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Includes methods that will print generic messages for all gas pumps
*
*/
public class Messages {
	//reject message
	public void displayRejectMsg() {
		System.out.println("\n Credit Card has been Rejected");
	}

	//ready message
	public void displayReadyMsg() {
```

```
		System.out.println("\n Ready to Start Pumping Gas");
	}

	//stop message
	public void displayStopMsg() {
		System.out.println("Done with pumping the Selected Gas");
	}

	//cancel message
	public void displayCancelMsg() {
		System.out.println("\n Transaction Cancelled");
	}

}
```

StoreData.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation
*
*/
public abstract class StoreData {

	Data data;

	public abstract void storeData();

	public void setData(Data data) {
		this.data = data;
	}

}
```

StoreData1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends abstract class StoreData and implements it for GasPump1
*
*/
public class StoreData1 extends StoreData {

	//stores user input values for premium and regular gas
```

```
	@Override
	public void storeData() {
		float a = data.getFloatTempA();
		float b = data.getFloatTempB();
		data.setPriceRegular(a);
		data.setPriceSuperPremium(b);
	}

}
```

StoreData2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends abstract class StoreData and implements it for GasPump2
*
*/
public class StoreData2 extends StoreData {

	//stores user input values for premium, super  and regular gas
	@Override
	public void storeData() {
		float a = data.getFloatTempA();
		float b = data.getFloatTempB();
		float c = data.getFloatTempC();
		data.setPriceRegular(a);
		data.setPriceSuperPremium(b);
		data.setPriceSuper1(c);
	}

}
```

DisplayMenu.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation
*
*/
public abstract class DisplayMenu {

	public abstract void displayMenu();
}
```

DisplayMenu1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends display menu and implements DisplayMenu for GasPump1
*
*/
public class DisplayMenu1 extends DisplayMenu {
	//display choice of gas selected in GasPump1
	@Override
	public void displayMenu() {
		System.out.println("Select your preferred Gas: Regular Gas (OR)  Super Gas\n" );
	}

}
```

DisplayMenu2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends display menu and implements DisplayMenu for GasPump2
*
*/
public class DisplayMenu2 extends DisplayMenu {
	//display choice of gas
	@Override
	public void displayMenu() {
		System.out.println("Select your preferred Gas: Regular Gas   (OR)  Premium Gas
(OR)  Super Gas \n" );
	}

}
```

PumpGasUnit.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation
*
*/
public abstract class PumpGasUnit {
	public abstract void pumpGasUnit();
}
```

PumpGasUnit1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends PumpGasUnit and implements PumpGasUnit for GasPump1
*
*/
public class PumpGasUnit1 extends PumpGasUnit {
	//mention that a unit of gas has been pumped
	@Override
	public void pumpGasUnit() {
		System.out.println("GasPump1:One Gallon is Pumped " );
	}

}
```

PumpGasUnit2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends PumpGasUnit and implements PumpGasUnit for GasPump2
*
*/
public class PumpGasUnit2 extends PumpGasUnit {
	//mention that a unit of gas has been pumped
	@Override
	public void pumpGasUnit() {
		System.out.println("GasPump2: One Liter is Pumped " );
	}

}
```

DisplayPayMsg.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation
*
*/
public abstract class DisplayPayMsg {
		public abstract void displayPayMsg();
}
```

DisplayPayMsg1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends abstract class and implements DisplayPayMsg for GasPump1
*
*/
public class DisplayPayMsg1 extends DisplayPayMsg{

	@Override
	public void displayPayMsg() {
		// TODO Auto-generated method stub
		System.out.println("Pay with Credit Card\n");
	}

}
```

DisplayPayMsg2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends abstract class and implements DisplayPayMsg for GasPump2
*
*/
public class DisplayPayMsg2 extends DisplayPayMsg{

	@Override
	public void displayPayMsg() {
		// TODO Auto-generated method stub
		System.out.println("Pay with cash($)\n");
	}
}
```

SetPrice.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation
*/
public abstract class SetPrice {
	public abstract void setPrice(Data data);
}
```

SetPrice1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends SetPrice and implements SetPrice for GasPump1
*
*/
public class SetPrice1 extends SetPrice {

	@Override
	public void setPrice(Data data) {
		// TODO Auto-generated method stub
		System.out.println("The Price of the Gas you selected is "+ ((Data1)data).getPrice());
	}

}
```

SetPrice2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends SetPrice and implements SetPrice for GasPump2
*
*/
public class SetPrice2 extends SetPrice {

	@Override
	public void setPrice(Data data) {
		// TODO Auto-generated method stub
		System.out.println("The Price of the Gas you selected is "+ ((Data2)data).getPrice());
	}

}
```

SetInitialValues.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation
*/
public abstract class SetInitialValues {
	public abstract void setInitialValues(Data data);
}
```

SetInitialValues1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends abstract class and implements SetInitialValues for GasPump1
*
*/
public class SetInitialValues1 extends SetInitialValues {
	//set units of gas pumped and total cost to zero
	@Override
	public void setInitialValues(Data data) {
		data.setG(0);
		data.setTotal(0);
	}

}
```

SetInitialValues2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  extends abstract class and implements SetInitialValues for GasPump2
*
*/
public class SetInitialValues2 extends SetInitialValues{
	//set units of gas pumped and total cost to zero
	@Override
	public void setInitialValues(Data data) {
		data.setG(0);
		data.setTotal(0);
	}

}
```

DisplayGasPumpedMsg.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation
*/
public abstract class DisplayGasPumpedMsg {
	public abstract void displayGasPumpedMsg(Data data);
}
```

DisplayGasPumpedMsg1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Extends Abstract class and implements methods in DisplayGasPumpedMsg for GasPump1
*
*/
public class DisplayGasPumpedMsg1 extends DisplayGasPumpedMsg{
	//print the total number of gallons pumped
	@Override
	public void displayGasPumpedMsg(Data data) {
		System.out.println("Gas Pumped : "+ data.getG() +" gallon(s)\n");
	}
}
```

DisplayGasPumpedMsg2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan
*  Extends Abstract class and implements methods in DisplayGasPumpedMsg for GasPump2
*
*/
public class DisplayGasPumpedMsg2 extends DisplayGasPumpedMsg{
	//print the total number of liters pumped
	@Override
	public void displayGasPumpedMsg(Data data) {
		System.out.println("Gas Pumped : "+ data.getG() +" liter(s)\n");
	}

}
```

StoreCash.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation
*
*/
public abstract class StoreCash {
	Data data;
```

```
	public abstract void storeCash();

	public void setCash(Data data) {
		this.data = data;
	}
}
```

StoreCash2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Extends Abstract class and implements StoreCash for GasPump2
*
*/
public class StoreCash2 extends StoreCash{
	//store user given cash in data2
	@Override
	public void storeCash() {
		float c = data.getFloatTempC();
		data.setFloatCash(c);
	}

}
```

PrintReceipt.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation of
PrintReceipt
*
*/
public abstract class PrintReceipt {

	public abstract void printReceipt(Data data);
}
```

PrintReceipt1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Extends Abstract class and implements PrintReceipt for GasPump1
*
*/
public class PrintReceipt1 extends PrintReceipt{
```

```
	//print receipt by showing number of gallons pumped and total cost
	@Override
	public void printReceipt(Data data) {
		System.out.println("GasPump1 receipt: "+ data.getG() +"gallons for $"+
data.getTotal());
	}
}
```

PrintReceipt2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Extends Abstract class and implements PrintReceipt for GasPump2
*
*/
public class PrintReceipt2 extends PrintReceipt{
	//print receipt by showing number of liters pumped and total cost
	@Override
	public void printReceipt(Data data) {
		System.out.println("GasPump2 receipt: "+ data.getG() +"liters for $"+
data.getTotal());
		System.out.println("ReturnCash:"+(data.getFloatCash()-data.getTotal())+"$");
	}
}
```

PrintNoReceipt.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Abstract class whose methods will be called by concrete classes for implementation of
PrintNoReceipt
*
*/
public abstract class PrintNoReceipt {

	public abstract void printNoReceipt(Data data);

}
```

PrintNoReceipt1.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Extends Abstract class and implements PrintNoReceipt for GasPump1
*
```

```
*/
public class PrintNoReceipt1 extends PrintNoReceipt{
	//print receipt by showing number of liters pumped and total cost
	@Override
	public void printNoReceipt(Data data) {
		System.out.println("ReturnCash:"+(data.getFloatCash()-data.getTotal())+"$");
	}
}
```

PrintNoReceipt2.java

```
package cs586;
/**
*
* @author Sivasenthil Namachivayan A20391478
*  Extends Abstract class and implements PrintNoReceipt for GasPump2
*
*/
public class PrintNoReceipt2 extends PrintNoReceipt{
	//print receipt by showing number of liters pumped and total cost
	@Override
	public void printNoReceipt(Data data) {
		System.out.println("ReturnCash:"+(data.getFloatCash()-data.getTotal())+"$");
	}
}
```