Project Report CS0586

SOFTWARE SYSTEM ARCHITECTURE

Ashutosh

A20377427

aashutosh@hawk.iit.edu

Contents:

1. Introduction

The document report describes the project relevant to the coursework of CS586 that is Software System Architecture. The report gives a detailed mechanism of the Gas -Pump using MDA-EFSM model. The project has different patterns that are used in Software Engineering and using different pattern model there is an implementation of a gas pump model.

Overview

The project has a gas pump model with two separate designs. Each of the designs have specified functional requirements. The gas pump is designed using three different strategies:

- State Design Pattern
- Abstract Factory Design Pattern
- Strategy design Pattern

The gas pump components are as follow:

- GasPump-1
- GasPump-2

The GasPump-1 component supports the following operations:

Activate (float a, float b)

Start() //start the transaction
PayCredit() // pay for gas by a credit card
Reject() // credit card is rejected

Cancel() // cancel the transaction
Approved() // credit card is approved
Super() // Super gas is selected
Regular() // Regular gas is selected
StartPump() // start pumping gas
PumpGallon() // one gallon of gas is disposed
StopPump() // stop pumping gas

The GasPump-2 component supports the following operations:

Activate (int a, int b, int c)

Start() //start the transaction

PayCash(int c) // pay for gas by cash, where c represents prepaid cash

Cancel() // cancel the transaction

Premium() // Premium gas is selected

Regular() // Regular gas is selected

Super() // Super gas is selected

StartPump() // start pumping gas

PumpLiter() // one liter of gas is disposed

Stop() // stop pumping gas

Receipt() // Receipt is requested

NoReceipt() // No receipt

2. Pseudo-code of all Input Processor for GasPump 1, GasPump2

Gas Pump 1

Operations of the Input Processor:

```
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->PayType(1);
}
Reject() {
 m->Reject();
}
Cancel() {
 m->Cancel();
}
Approved() {
 m->Approved();
}
Super() {
 m->SelectGas(2)
```

```
}
Regular() {
 m->SelectGas(1)
}
StartPump() {
 m->StartPump();
}
PumpGallon() {
 m->Pump();
StopPump() {
 m->StopPump();
 m->Receipt();
}
```

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

Gas Pump-2

Operations of the Input Processor

```
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->PayType(2)
 }
}
Cancel() {
 m->Cancel();
}
Super() {
 m->SelectGas(2);
}
Premium() {
 m->SelectGas(3);
}
Regular() {
 m->SelectGas(1);
}
StartPump() {
```

```
 m->StartPump();
}
PumpLiter() {
 if (d->cash<(d->L+1)*d->price)
 m->StopPump();
 else m->Pump()
}
Stop() {
 m->StopPump();
}
Receipt() {
 m->Receipt();
}
NoReceipt() {
 m->NoReceipt();
}
```

Where

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

3. **Model Driven Architecture of GasPump Components**
   A general architecture of the Gaspump components is shown in the figure below.

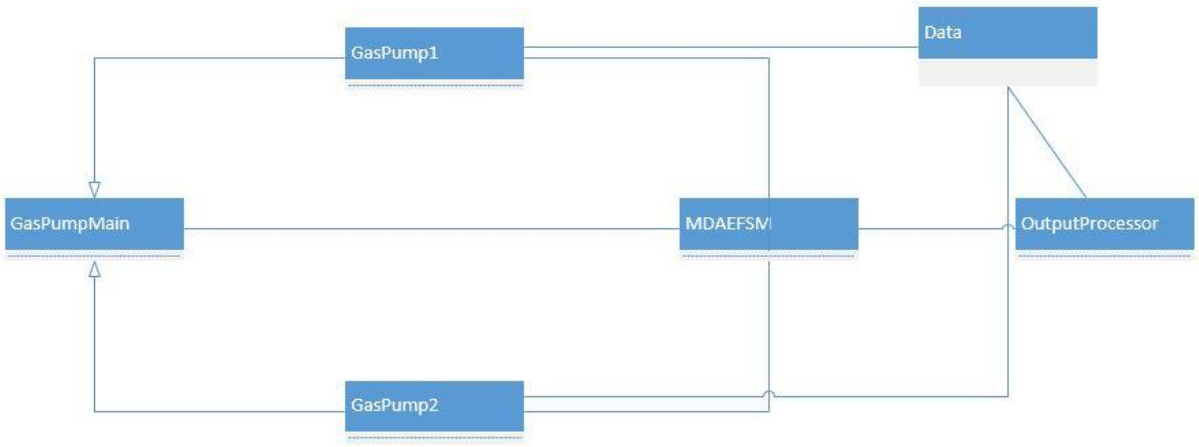


Figure 3.1 General Architecture

MDA-EFSM Events:
Activate()
Start()
PayCash(int t)
PayCredit
Reject()
Cancel()
Approved()
StartPump()
Pump()
StopPump()
SelectGas(int g)
Receipt()
NoReceipt()

MDA-EFSM Actions:

StoreData // 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
ReadyMsg // displays the ready for pumping message
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
StopMsg // stop pump message and receipt? msg (optionally)
PrintReceipt // print a receipt
CancelMsg // displays a cancellation message
ReturnCash // returns the remaining cash

State diagram of MDA-EFSM is as follow:

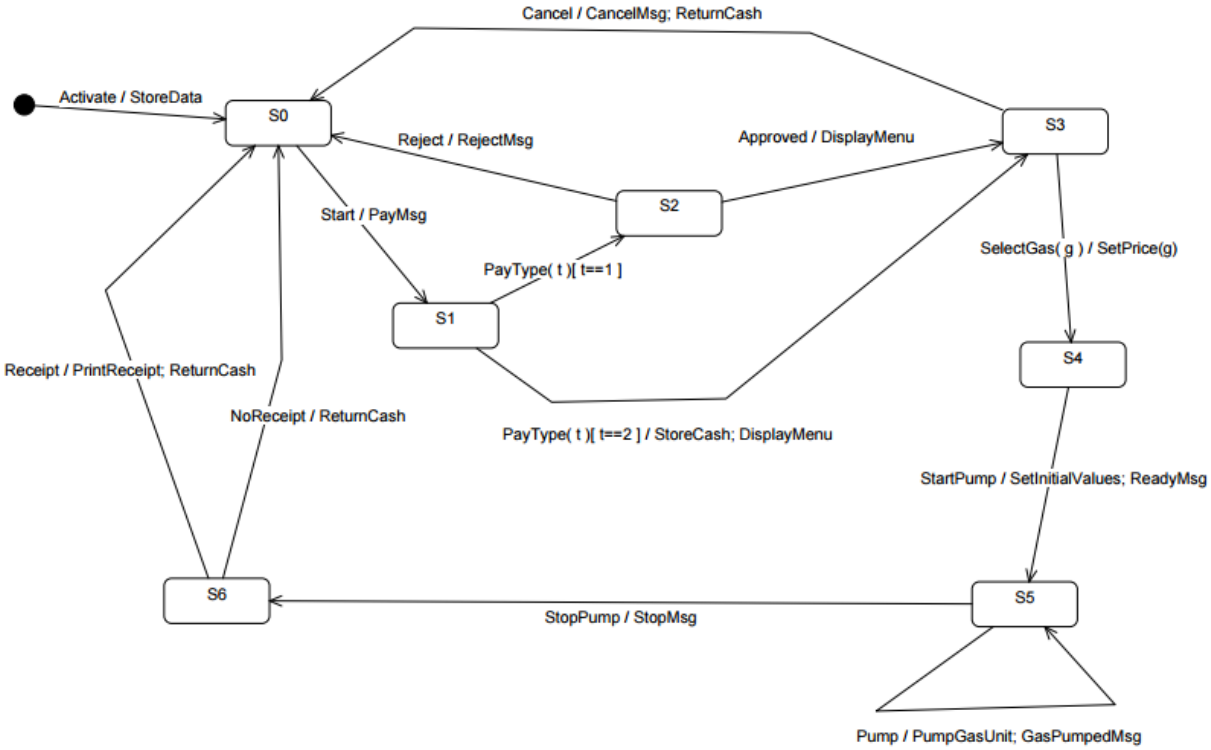


Figure 3.2 State Diagram of MDAEFSM

4. ClassDiagram

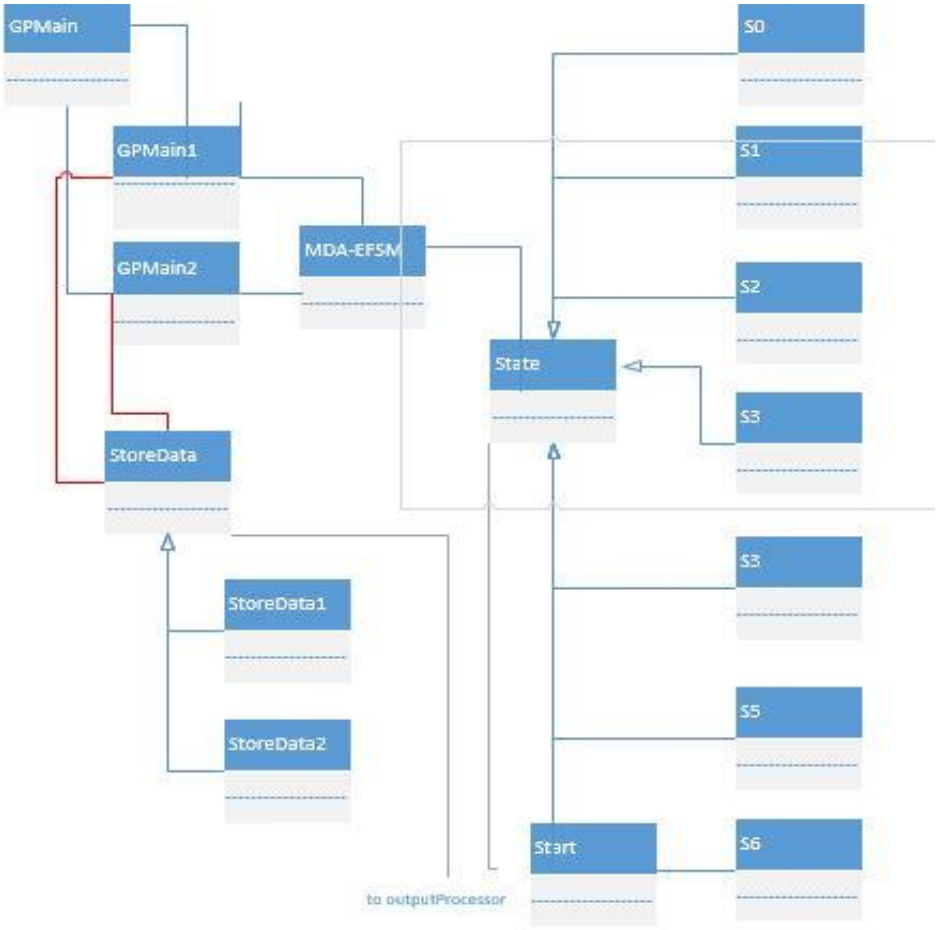


Figure 4.1 Overall class diagram(i)

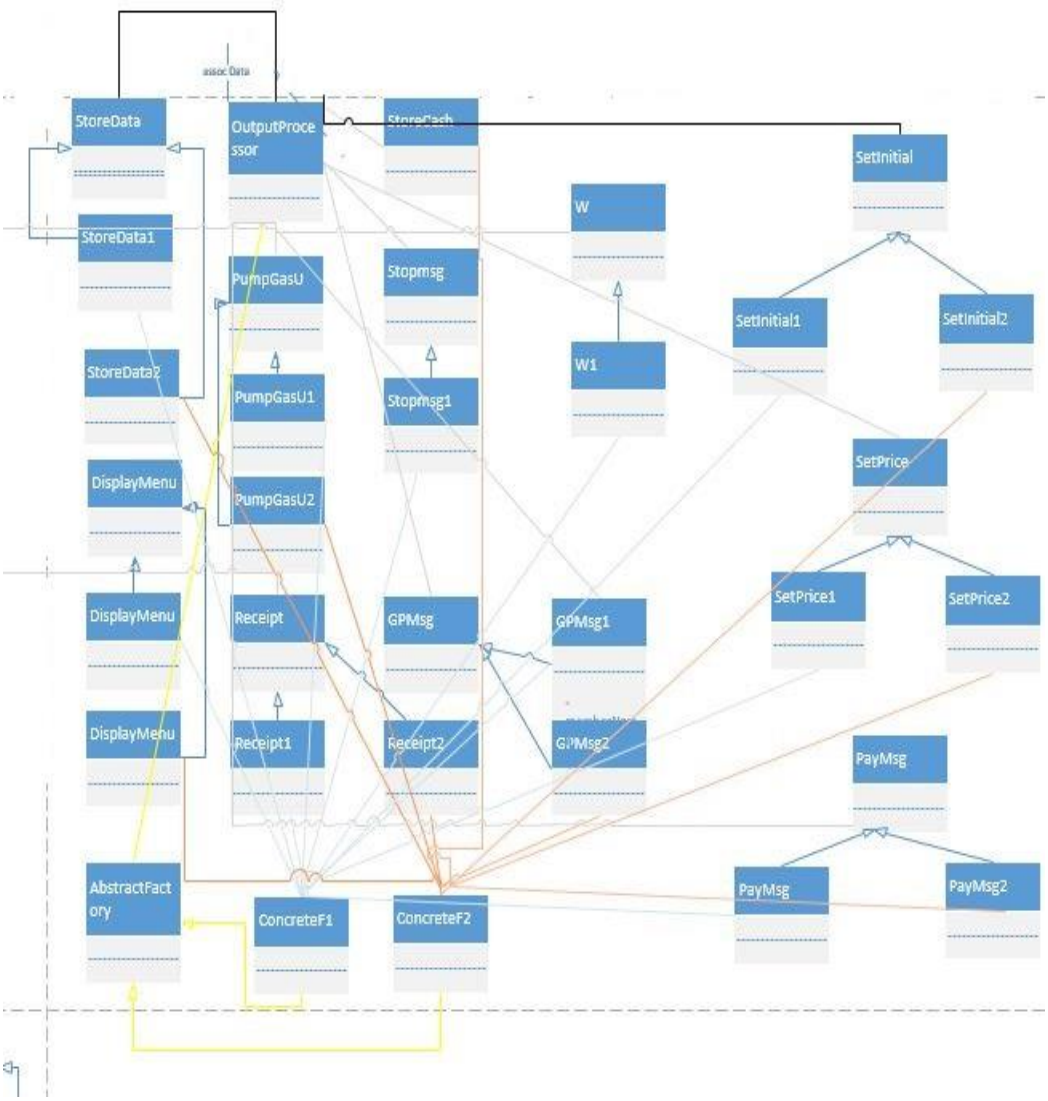


Figure 4.2 Overall class diagram(ii)

There are three sub division of the project which are :

1. Input Processor
2. MDA-EFSM
3. Output Processor

4.1 Input Processor

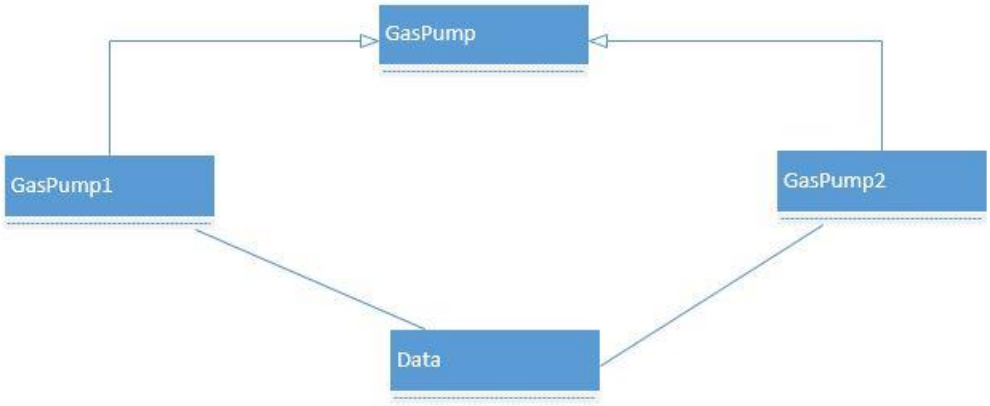


Figure 4.3 Input processor

GasPumpMain:

In gas pump main user can have two option of selecting the gas pump. A concretefactory with respect to the gas pump selected will be created along with the appropriate objects. All the operations are explained in the code for gaspump1 and gaspump2.

MDA-EFSM

This class is responsible for the change in state. It has a current state and list of states. This class represents the common functionality. The different states are as follow:

S0 - This state comes after start. It asks for payment method available and shows error for invalid state.
S1 - It has payCash() and payCredit() methods. For other methods it gives error for invalid state.
S2 - It has reject() and approve() methods. For other methods it gives error for invalid state.
S3 - It has payCash() and payCredit() methods. For other methods it gives error for invalide state.
S4 – It has startPump() method. For other methods it shows error for invalid state.
S5 - It has startPump () and pumpUnit() methods. For other methods it gives error for invalide state.
S6 – It has receipt() and noReceipt() method. For other methods it shows error for invalid state.
Start()- It is a start state. It gives error for invalide state.

Data Class

The operation for the data class is explained in the source code. The methods and attributes for the data class is shown in the class diagram:

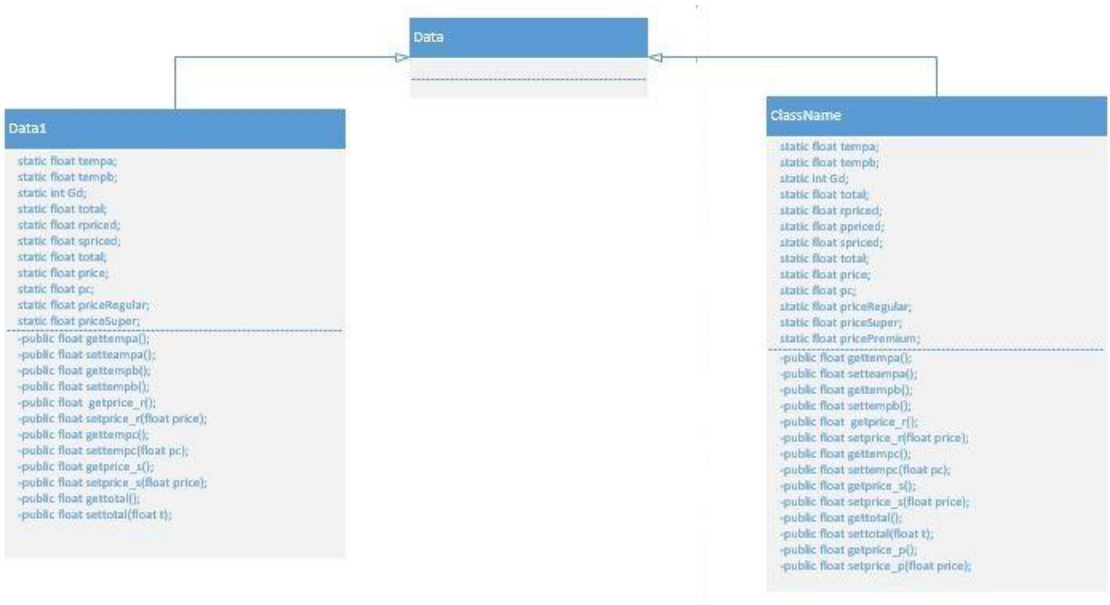


Figure 4.4 Data class diagram

MDA-EFSM

This class contains MDA and the states. The operation for the MDAEFSM class is explained in the source code. The methods and attributes for the data class is shown in the class diagram:

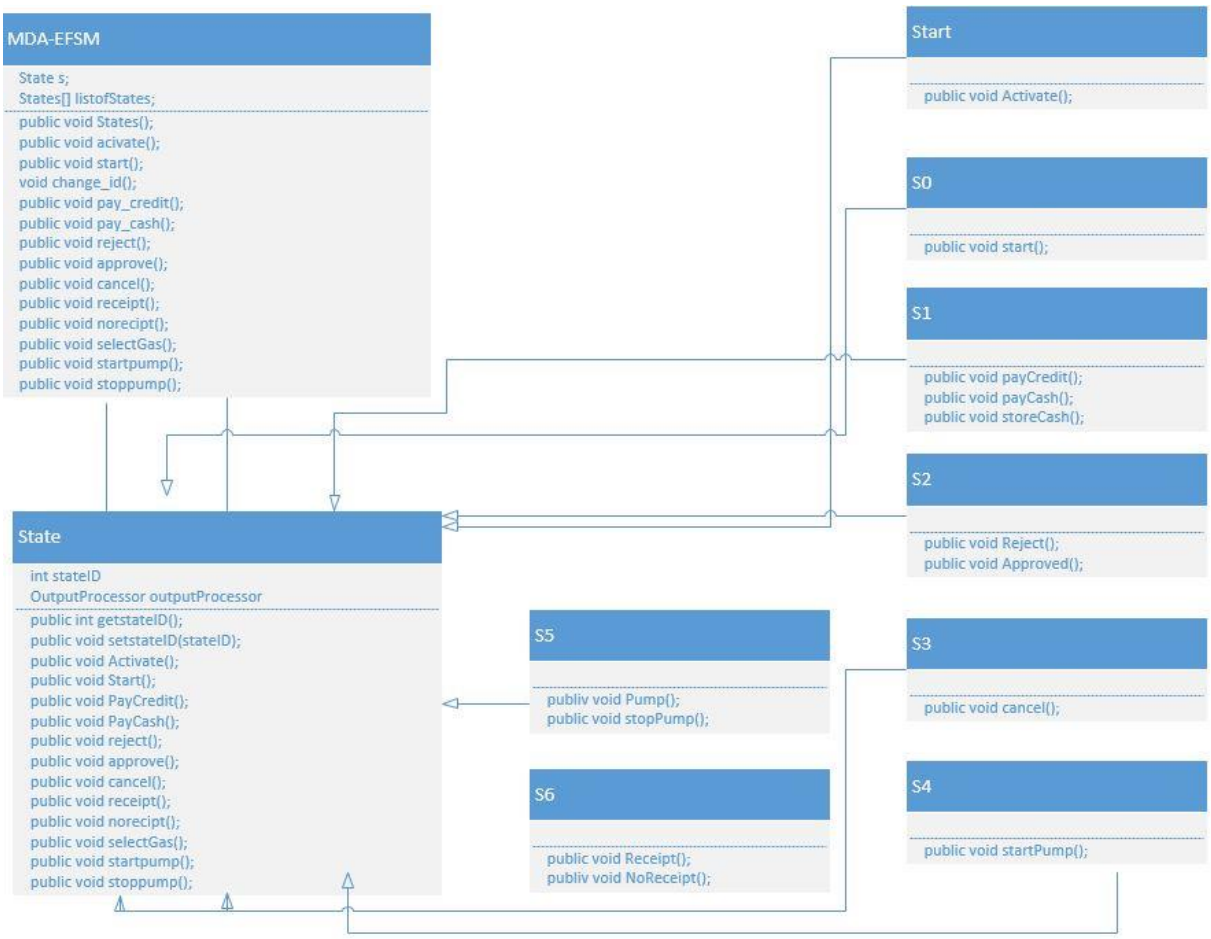


Figure 4.5 MDAEFSM class diagram

Output Processor

The operation for the Output Processor is explained in the source code. The methods and attributes for the data class is shown in the class diagram:

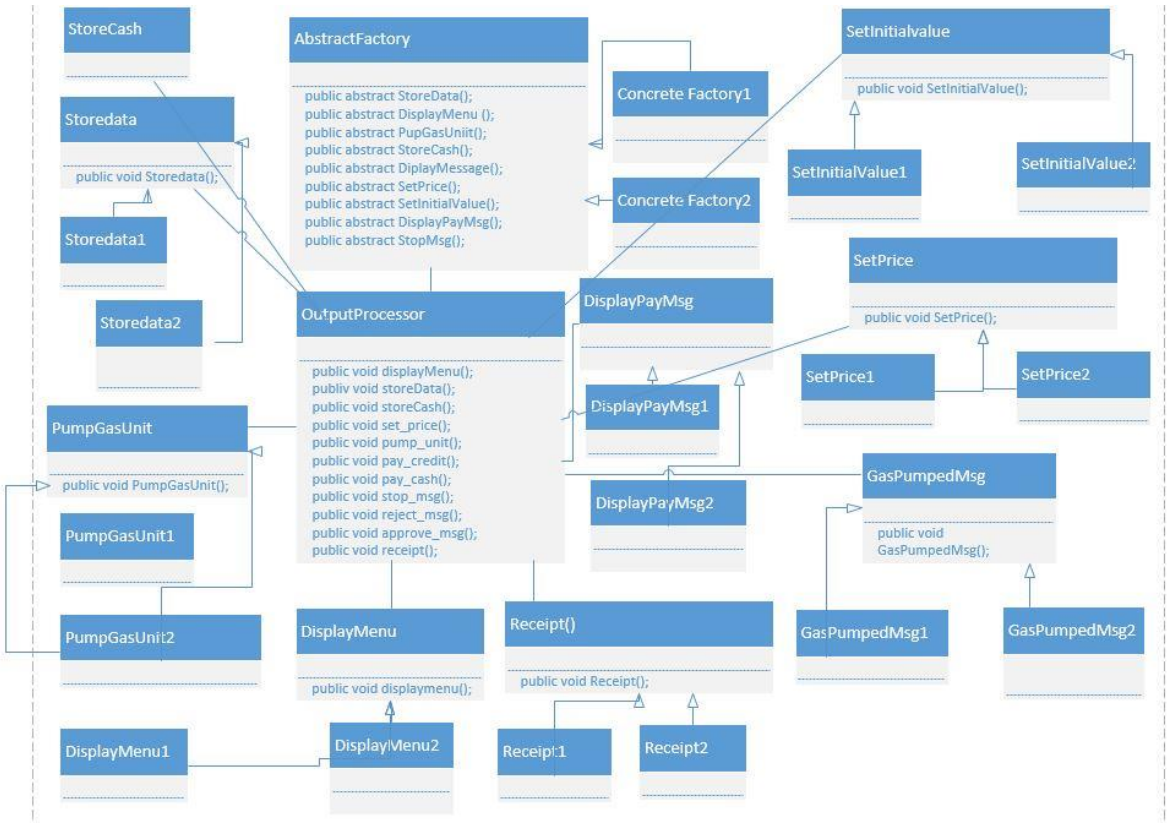


Figure 4.6 Abstract class diagram

Concrete factory 1

Figure 4.7 Concrete class1

Concrete factory2

Figure 4.7 Concrete class 2

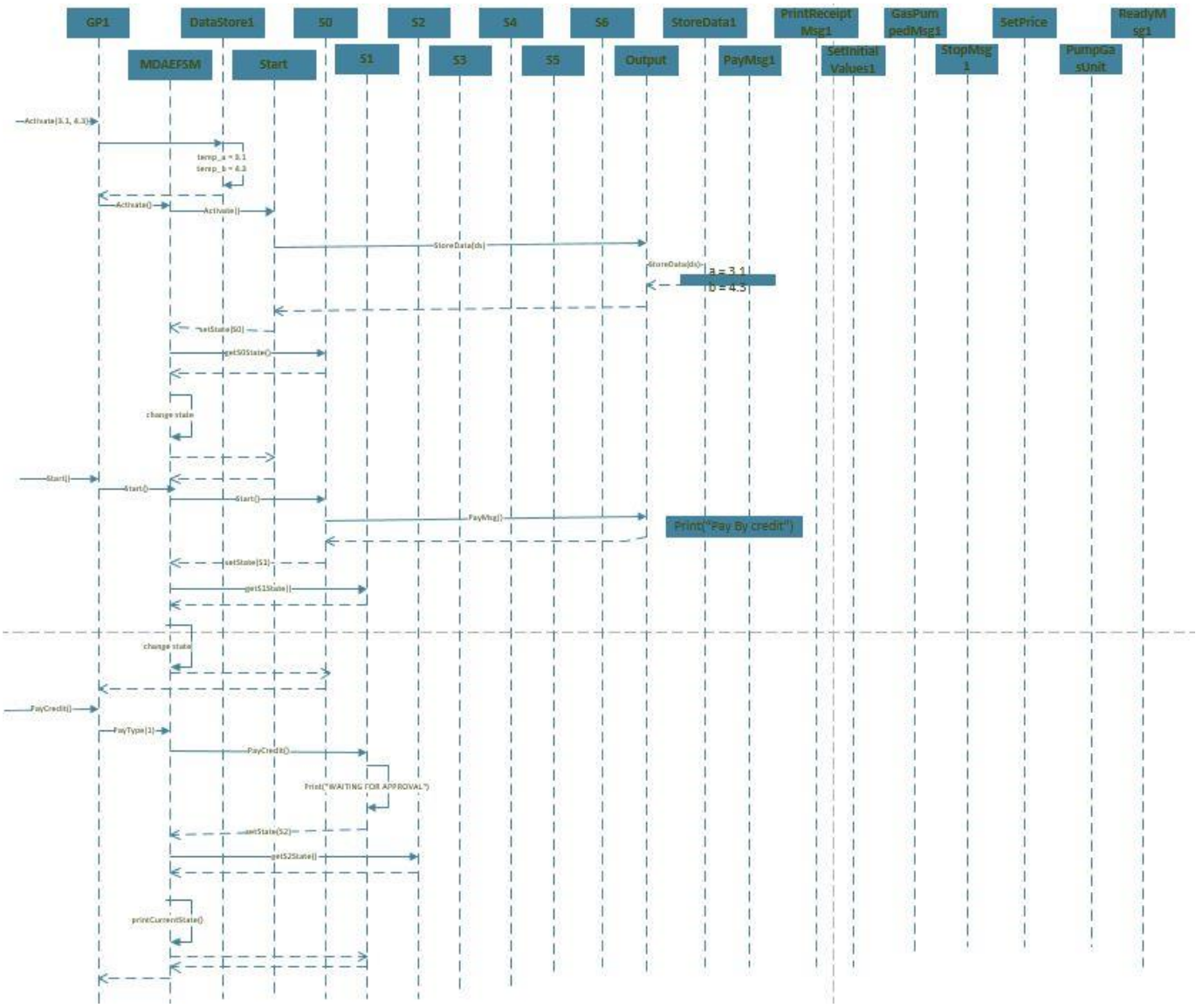
Sequence diagram

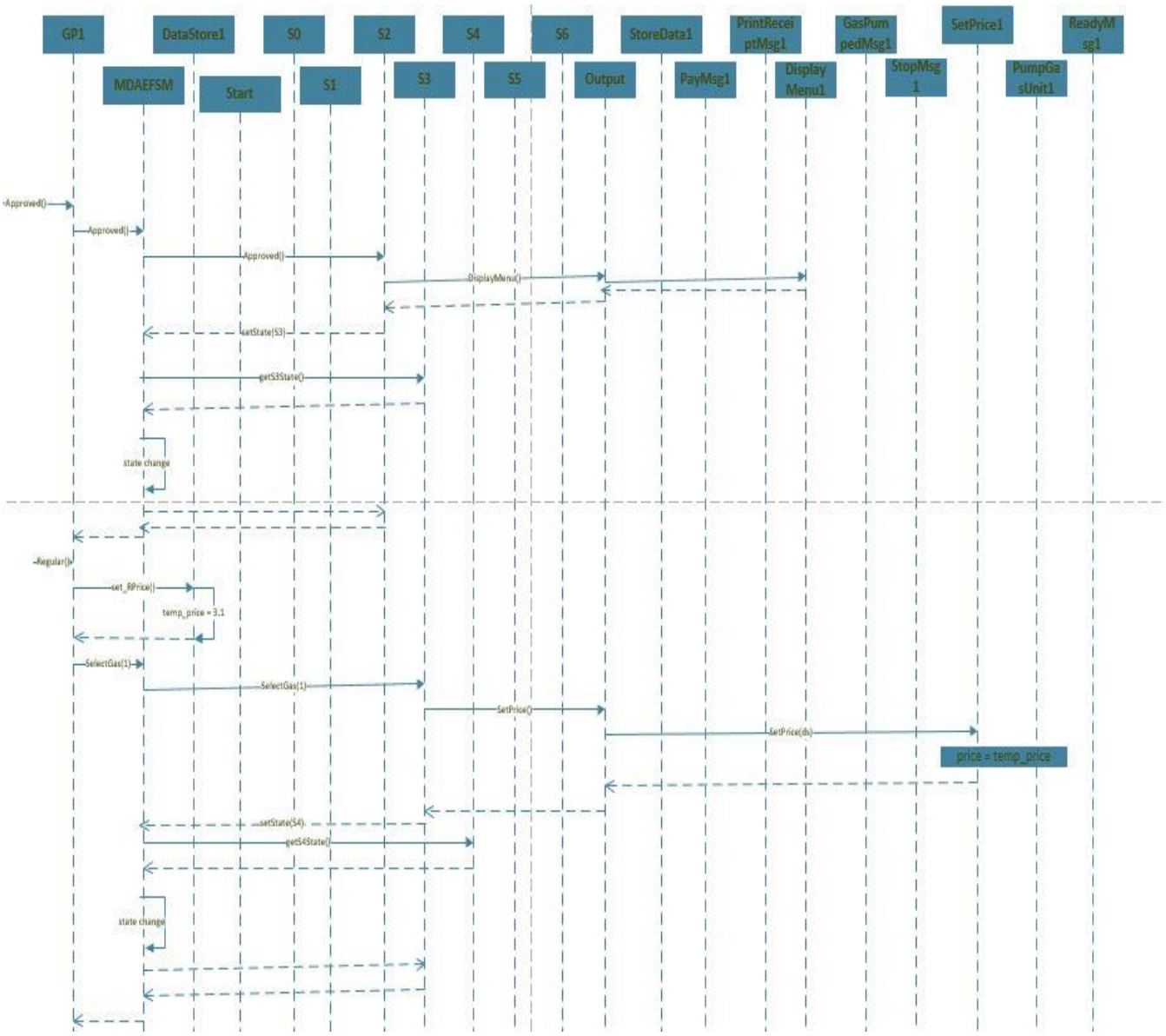
Sequence diagram are created for the two scenario's:

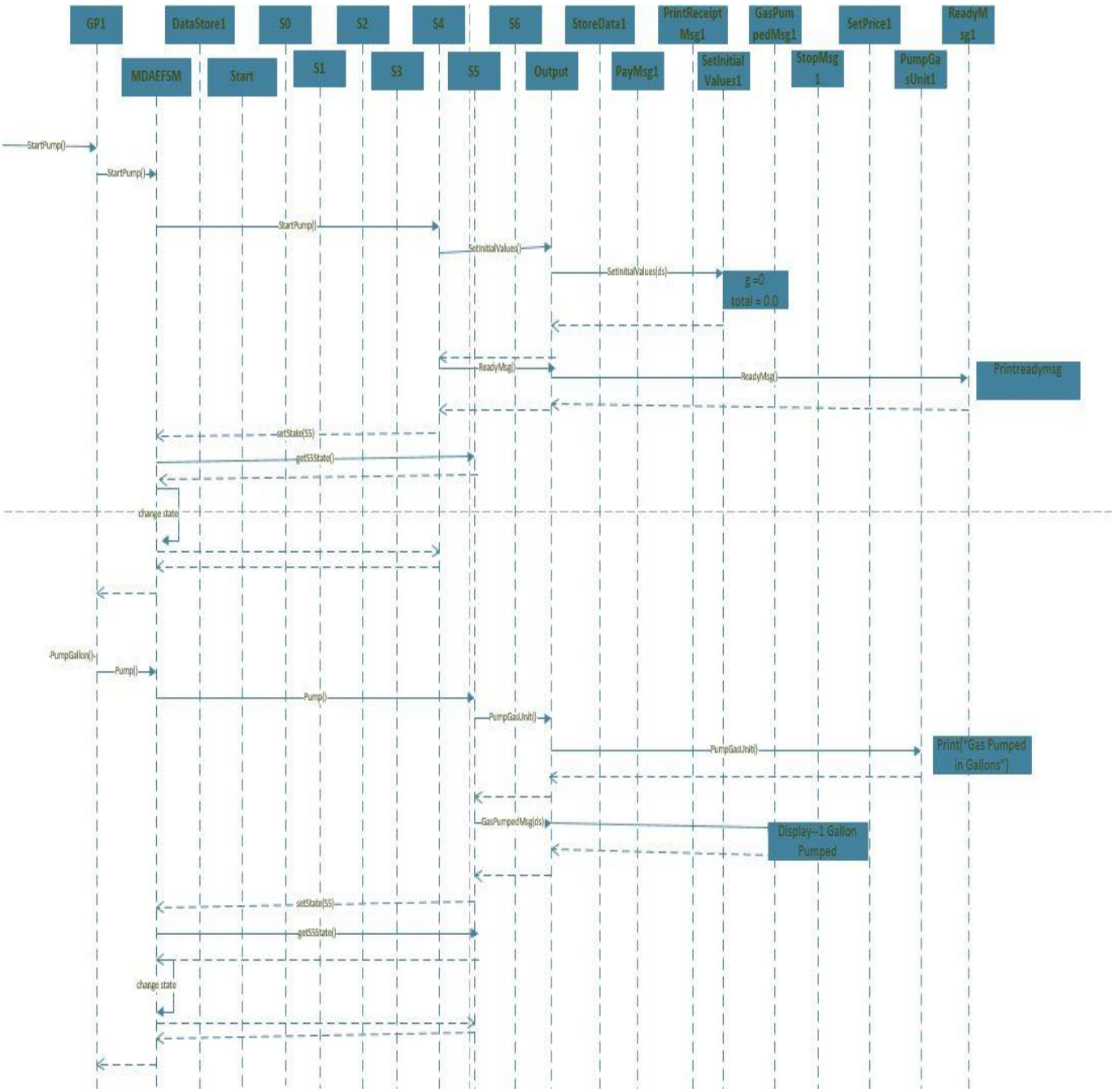
a. Scenario-I should show how one gallon of Regular gas is disposed in GasPump-1, i.e., the following sequence of operations is issued: Activate(3.1, 4.3), Start(), PayCredit(), Approved(), Regular(), StartPump(), PumpGallon(), StopPump()

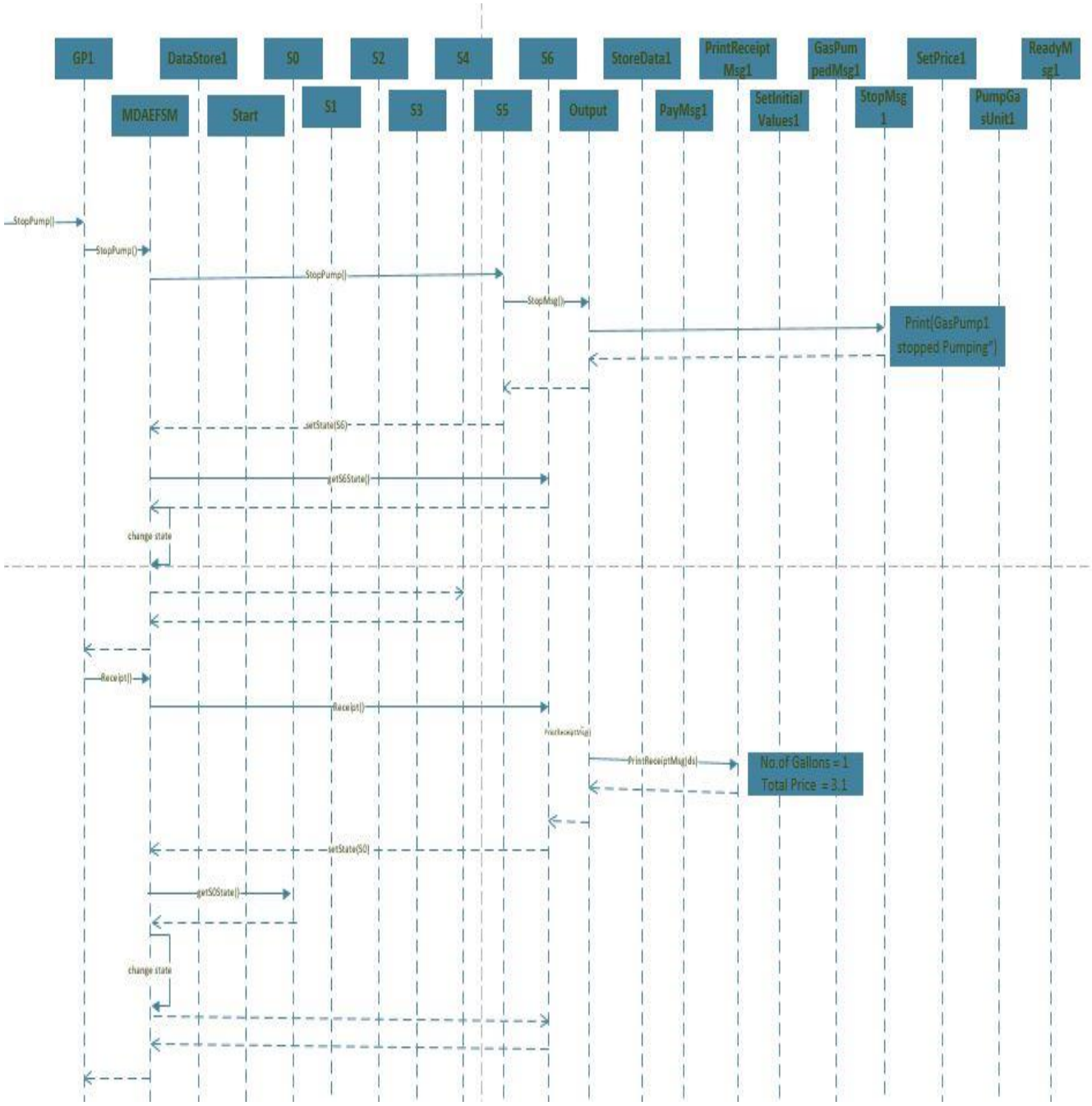
b. Scenario-II should show how one liter of Premium gas is disposed in GasPump-2, i.e., the following sequence of operations is issued: Activate(3, 4, 5), Start(), PayCash(6), Premium(), StartPump(), PumpLiter(), PumpLiter(), NoReceipt()

Scenario 1 for gas pump 1

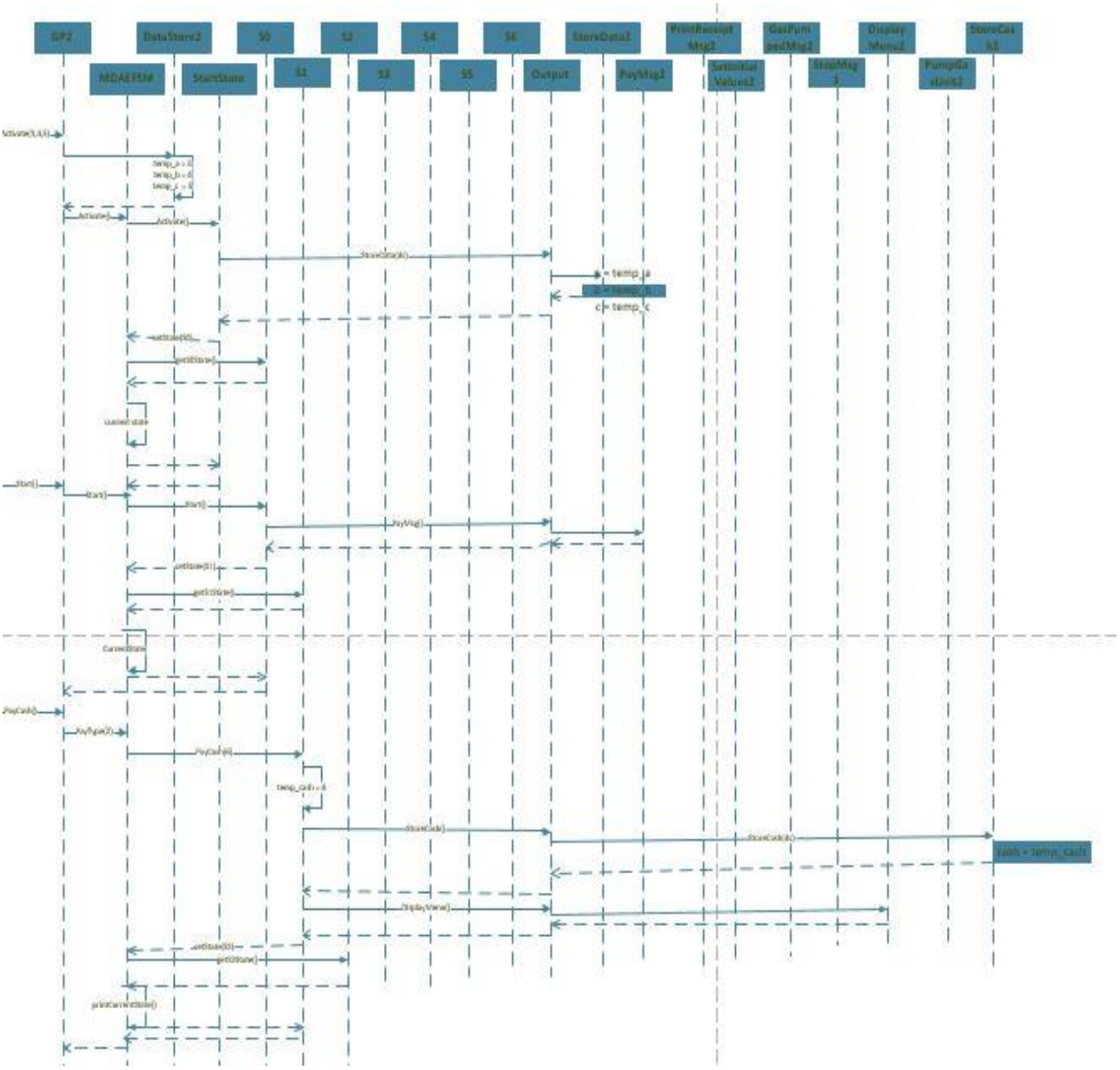

GP1
DataStore1
S0
S2
S4
S6
StoreData1
PrintReceiptMsg1
GasPumpedMsg1
SetPrice
ReadyMsg1
MDAEFSM
Start
S1
S3
S5
Output
PayMsg1
SetInitialValues1
StopMsg1
PumpGasUnit
Activate(3.1, 4.3)
temp_a = 3.1
temp_b = 4.3
Activate()
Activate()
StoreData(ds)
storeData(ds)
a = 3.1
b = 4.3
setState(S0)
getS0State()
change state
Start()
Start()
Start()
PayMsg()
Print("Pay By credit")
setState(S1)
getS1State()
change state
PayCredit()
PayType(1)
PayCredit()
Print("WAITING FOR APPROVAL")
setState(S2)
getS2State()
printCurrentState()

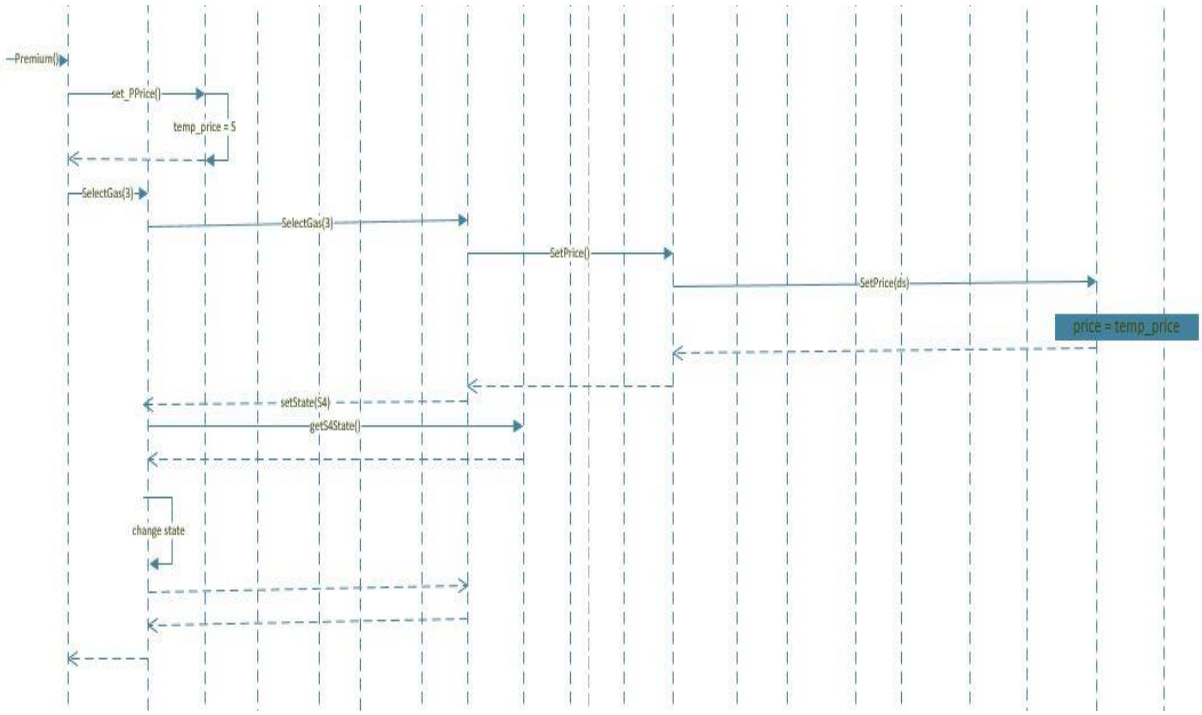

GP1
DataStore1
S0
S2
S4
S6
StoreData1
PrintReceiptMsg1
GasPumpedMsg1
SetPrice1
ReadyMsg1
MDAEFSM
Start
S1
S3
S5
Output
PayMsg1
DisplayMenu1
StopMsg1
PumpGasUnit1
Approved()
Approved()
Approved()
DisplayMenu()
setState(S3)
getS3State()
state change
Regular()
set_RPrice()
temp_price = 3.1
SelectGas(1)
SelectGas(1)
SetPrice()
SetPrice(ds)
price = temp_price
setState(S4)
getS4State()
state change

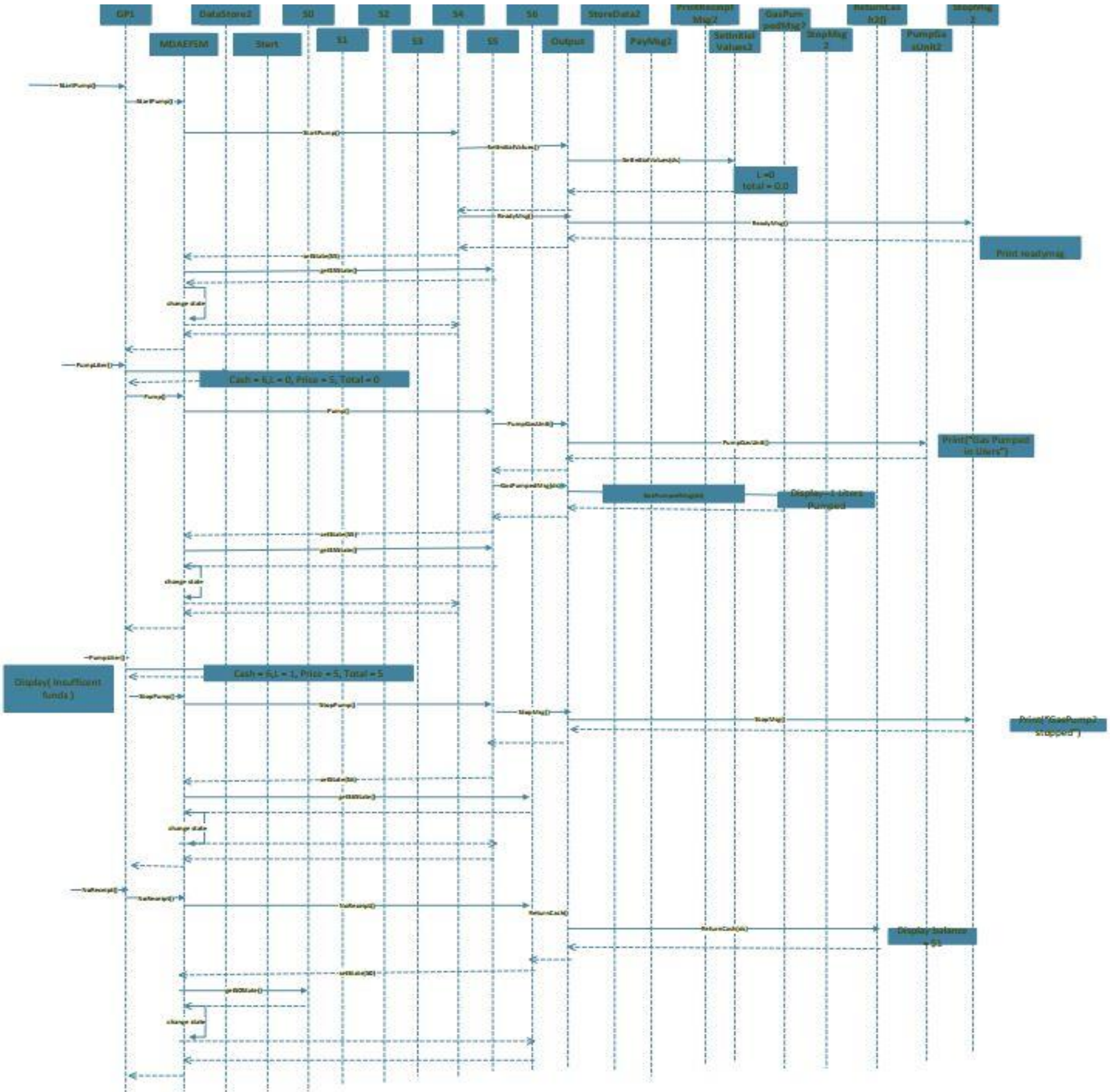

GP1
DataStore1
S0
S2
S4
S6
StoreData1
PrintReceipt Msg1
GasPum pedMsg1
SetPrice1
ReadyM sg1
MDAEFSM
Start
S1
S3
S5
Output
PayMsg1
SetInitial Values1
StopMsg 1
PumpGa sUnit1
StartPump()
StartPump()
StartPump()
SetInitialValues()
SetInitialValues(ds)
g =0
total = 0.0
ReadyMsg()
ReadyMsg()
Printreadymsg
setState(S5)
getSSState()
change state
PumpGallon()
Pump()
Pump()
PumpGasUnit()
PumpGasUnit()
Print("Gas Pumped in Gallons")
GasPumpedMsg(ds)
Display--1 Gallon Pumped
setState(S5)
getSSState()
change state


GP1
DataStore1
S0
S2
S4
S6
StoreData1
PrintReceipt Msg1
GasPum pedMsg1
SetPrice1
ReadyM sg1
MDAEFSM
Start
S1
S3
S5
Output
PayMsg1
SetInitial Values1
StopMsg 1
PumpGa sUnit1
StopPump()
StopPump()
StopPump()
StopMsg()
Print(GasPump1 stopped Pumping")
setState(S6)
getS6State()
change state
Receipt()
Receipt()
PrintReceiptMsg()
PrintReceiptMsg(ds)
No.of Gallons = 1
Total Price = 3.1
setState(S0)
getS0State()
change state

Scenario 2 for Gas Pump-2:




Premium()
set_PPrice()
temp_price = 5
SelectGas(3)
SelectGas(3)
SetPrice()
SetPrice(ds)
price = temp_price
setState(S4)
getS4State()
change state


GP1
DataStore2
S0
S2
S4
S6
StoreData2
MDAEFSM
Start
S1
S3
S5
Output
PayMsg2
SetInitial Values2
StopMsg 2
L = 0
total = 0.0
Print readymsg
Cash = 6, L = 0, Price = 5, Total = 0
Display(insufficient funds)
Cash = 6, L = 1, Price = 5, Total = 5

Source Code with design Pattern

GasPumpMain.java

```
import java.util.Scanner;

import Datastore.DataStore2;
import Datastore.DataStore;
import java.util.Scanner;
import Factory.AbstractFactory;
import Factory.CFactory1;
import Factory.CFactory2;
import MDEFSM.MDEFSM;
import OutputProcesser.DisplayMenu;
import OutputProcesser.DisplayMenu1;
import OutputProcesser.DisplayMenu2;



public class GasPumpMain {

	GasPump1 gp1;
	GasPump2 gp2;

	public static void main(String []args) {
		System.out.println("Enter your choice: \n");
		boolean c = true;
```

```
        while(c) {
                System.out.
                println("\nWelcome to GasPump\n");
                System.out.println("Select GasPump from the choice given:\n ");
                System.out.println("1.  GasPump-1 \n");
                System.out.println("2. GasPump-2 \n");
                System.out.println("3. Exit \n");

                Scanner sc = new Scanner(System.in);
                System.out.println("Enter your choice : \n");
                String gp = sc.next();
                if (gp!="3") {
                        c=selectGasPump(gp);                    }

                else {
                        System.out.println("\n Gas Pump Stopped");
                        c = false;

                }
        }
}

static boolean selectGasPump (String gp) {

        switch(gp) {

        case "1":
```

```
                GasPump1 gp1 = new GasPump1();
                gp1.startGasPump();
                break;
            case "2":
                GasPump2 gp2 = new GasPump2();
                gp2.startGasPump();
                break;
            case "3":
                System.out.println("\n ~~~~~~~~~~GasPump Stopped~~~~~~~~~~~~~~");
                return false;
            default:
                System.out.println("Invalid choice \n");
                break;



            }
            return true;
        }
}
```

GasPump1.java

```
import Factory.AbstractFactory;
import java.util.Scanner;
import Datastore.DataStore1;
import Datastore.DataStore2;
import Datastore.DataStore;
```

```
import Factory.CFactory1;
import MDEFSM.MDEFSM;
import MDEFSM.MDEFSM;
import Datastore.DataStore;


public class GasPump1 {
	float a,b;
	DataStore ds;
	AbstractFactory af;
	MDEFSM mde;
	DataStore1 db;
	float c;

	GasPump1(){
		af = new CFactory1();
		mde = new MDEFSM();
		DataStore.af=af;
	}


	void startGasPump(){
		System.out.println("~~~~~~~~~~~~~~~~~GasPump-1~~~~~~~~~~~~~~~~\n");

		System.out.println("The GasPump-1 component supports the following operations: \n");
		System.out.println("1. Activate (float a, float b) \n"); // It activates the gas pump where a is the price og gas per litres.
```

```
System.out.println("2. Start()  \n"); //It initiates the the transaction.
System.out.println("3. PayCredit()  \n"); // PayCredit takes the credit from the user.
System.out.println("4. Reject()  \n"); //It cancels the transaction
System.out.println("5. Cancel()  \n");   //Here regular gas is selected
System.out.println("6. Approved()  \n"); // User selects super gas
System.out.println("7. Regular() \n"); //Premium gas is selected by user
System.out.println("8. Super()  \n"); // It starts the pumping of the gas
System.out.println("9. StartPump()  \n"); //one litre will be pumped
System.out.println("10. PumpGallon()  \n"); // it stops the pumping of the gas
System.out.println("11. StopPump() \n"); //Receipt will be printed
System.out.println("12. ExitPump()"); //Exits pump#2


System.out.println("Execution for GasPump 1 starts here \n");

boolean cp = true;
while(cp) {
        System.out.println("\n Operations :\n 1.Activate  "
                                        +"2.Start  "
                                        +"3.PayCredit  "
                                        +"4.Reject  "
                                        +"5.Cancel  "
                                        +"6.Approved  "
                                        +"7.Regular  "
                                        +"8.Super  "
                                        +"9.StartPump  "
                                        +"10.PumpGallon  "
```

```
                                        +"11.StopPump  "
                                        +"12.Exit \n ");

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter your choice: \n");
        int op = sc.nextInt();

        if (op == 12) {
                System.out.println("GasPump-1 Exit \n");
                break;
        }
        switch(op)
        {
        case 1:
                float a,b;
                System.out.println("\n Operation: Activate(a,b) \n");
                System.out.println("Enter price for regular gas : ");

                try {
                        a=sc.nextFloat();

                }
                catch(Exception e) {
                        a=0;
                        System.out.print("Gas Price must be in float for GasPump-
1");
                }
```

```
                System.out.println("Enter Price for Super Gas : ");
                try {
                        b=sc.nextFloat();

                }
                catch(Exception e) {
                        b=0;
                        System.out.print("Gas Price must be in float for GasPump-
1");
                }

                Activate(a,b);
                break;

        case 2:
                System.out.println("\n Operation:  start() \n  ");
                start();
                break;
        case 3:
                System.out.println("\n Operation:  PayCredit() \n  ");
                PayCredit();
                break;
        case 4:
                System.out.println("\n Operation:  Reject() \n  ");
                Reject();
                break;
        case 5:
```

```
            System.out.println("\n Operation:  Cancel() \n  ");
            Cancel();
            break;
case 6:
            System.out.println("\n Operation:  Approved() \n  ");
            Approved();
            break;
case 7:
            System.out.println("\n Operation:  Regular() \n  ");
    Regular();
    break;
case 8:
            System.out.println("\n Operation:  Super() \n  ");
    Super();
    break;
case 9:
            System.out.println("\n Operation:  StartPump() \n  ");
            StartPump();
            break;
case 10:
            PumpGallon();
            break;
case 11:
            System.out.println("\n Operation:  StopPump() \n  ");
            StopPump();
            break;
case 12:
```

```
                    cp = false;
                    break;
                default:
                    System.out.println("Invalid Choice");
                break;



                }
        }
}

public void Activate(float a, float b) {
        if(a > 0 && b>0){
                db=(DataStore1) af.getdata();
                db.tempa=a;
                db.tempb=b;
                mde.activate();
        }
        else {
                System.out.println("Gas price must be greater then zero");
        }//Gas price can never be zero or less.
}
public void start() {
        mde.start();
}
public void Reject() {
        mde.reject();
```

```
}
public void PayCredit() {
        mde.pay_credit();
}

public void Cancel() {
        mde.cancel_msg();
}

public void Super() {
        mde.select_gas(2);
}
public void Approved() {
        db=(DataStore1) af.getdata();
        db.tempw=1;
        mde.approved();
}
public void Regular() {
        mde.select_gas(1);
}
public void StartPump() {
        System.out.println("\n~~~~~~~~~~Gas Pump 1 begins~~~~~~~~~~~~");
        db=(DataStore1) af.getdata();
        db.tempG=0;
        mde.start_pump();
}
public void PumpGallon() {
```

```
                System.out.println("\n~~~~~~~~~~~~~~Pumping starts from GP1~~~~~~~~~~~~~~");
                db=(DataStore1) af.getdata();
                mde.pump_unite();
        }
        public void StopPump() {
                mde.stop_pump();
                mde.receipt();
        }
}
```

GasPump2.java

```
import Datastore.DataStore2;
import Datastore.DataStore;
import java.util.Scanner;
import Factory.AbstractFactory;
import Factory.CFactory1;
import Factory.CFactory2;
import MDEFSM.MDEFSM;
import OutputProcesser.DisplayMenu;
import OutputProcesser.DisplayMenu1;
import OutputProcesser.DisplayMenu2;
```

```
public class GasPump2 {



        float b;

        float c;

        int a;

        int abc;

        DataStore ds;

        AbstractFactory af;

        MDEFSM mde;

        int debug;

        DataStore2 db;

        GasPump2() {

                af = new CFactory2();

                mde =new MDEFSM();

                DataStore.af=af;

        }


        void startGasPump() {

                System.out.println("~~~~~~~~~~~~~~~~~~GasPump-2~~~~~~~~~~~~~~~~~~\n");


                System.out.println("The GasPump-2 component supports the following operations: \n");

                System.out.println("1. Activate (float a, float b, float c) \n"); // It activates the gas pump where a is the price og gas per litres.

                System.out.println("2. Start()  \n"); //It initiates the the transaction.

                System.out.println("3. PayCash(float c)  \n"); // PayCash takes the cash c from the user.
```

```
System.out.println("4. Cancel()  \n"); //It cancels the transaction
System.out.println("5. Regular()  \n");   //Here regular gas is selected
System.out.println("6. Super()  \n"); // User selects super gas
System.out.println("7. Premium() \n"); //Premium gas is selected by user
System.out.println("8. StartPump()  \n"); // It starts the pumping of the gas
System.out.println("9. PumpLiter()  \n"); //one litre will be pumped
System.out.println("10. StopPump()  \n"); // it stops the pumping of the gas
System.out.println("11. Receipt()   \n"); //Receipt will be printed
System.out.println("12. NoReceipt()  \n"); //No Receipt will be printed
System.out.println("13. ExitPump()"); //Exits pump#2

System.out.println("Execution for GasPump-2 starts from here: \n");

boolean cp = true;
while(cp) {
        System.out.println("\n Operations :\n 1.Activate  "
                                                +"2.Start  "
                                                +"3.PayCash  "
                                                +"4.Cancel  "
                                                + "5.Regular  "
                                                +"6.Super  "
                                                +"7.Premium  "
                                                +"8.StartPump  "
                                                +"9.PumpLiter  "
                                                +"10.StopPump  "
                                                +"11.Receipt  "
                                                +"12.No Receipt  "
```

```
                                                    +"13.Exit \n ");
                    Scanner sc = new Scanner(System.in);
                    System.out.println("Please Enter your choice : \n");
                    int op = sc.nextInt();

                    if(op == 13) {
                            System.out.println("GasPump-3 Exit \n"); //If user selects exit then
it should exit the system.
                            break;
                    }
                    switch(op)
                    {
                            case 1:
                                    float a, b, c  ;
                                    System.out.println("\n Operation:  Activate(a,b) \n  ");
                                    System.out.println("Enter Price of Regular Gas : ");
                                    try {
                                            a=sc.nextFloat();

                                    }
                                    // GasPrice should be in float for different gases available at
the PumpStation.
                                    catch(Exception e) {
                                            a=0;
                                            System.out.print("Gas  Price  must  be  in  float  for
GasPump-2");
                                    }
                                    System.out.println("Enter Price of Premium Gas : ");
```

```
                try {
                        b=sc.nextFloat();

                }
                catch(Exception e) {
                        b=0;
                        System.out.print("Gas Price must be in float for
GasPump-2");
                }
                System.out.println("Enter Price of Super Gas : ");
                try {
                        c=sc.nextFloat();

                }
                catch(Exception e) {
                        c=0;
                        System.out.print("Gas Price must be in float for
GasPump-2");
                }


                Activate(a,b,c); //GasPump gets activated
                break;
            case 2:
                System.out.println("\n Operation:  start() \n  ");
                //start gets called.
                start();
                break;
```

```
                    case 3:
                        System.out.println("\n Operation:  PayCash(c) \n  ");
                        System.out.println("\n Please feed the cash amount : ");
                        //User will enter the cah amount.
                    Scanner pc =new Scanner(System.in);
                    float d = pc.nextFloat();
                    if(d>0) {
                         PayCash(d);
                    }
                    else {
                         System.out.println("!!!  Plese  Enter  Valid  Cash
Payment !!!!");
                    }//cash should always be greater then 0 for successful gas pump
operations
                    break;
                    case 4:
                        System.out.println("\n Operation:  Cancel() \n  ");
                        Cancel();
                        break;
                    case 5:
                        System.out.println("\n Operation: Regular()  \n");
                        Regular();
                        break;
                    case 6:
                        System.out.println("\n Operation:  Super() \n  ");
                        Super();
                        break;
                    case 7:
```

```
            System.out.println("\n Operation:  Premium() \n  ");
            Premium();
            break;
        case 8:
            System.out.println("\n Operation:  StartPump() \n  ");
            StartPump();
       break;
        case 9:
            System.out.println("\n Operation:  PumpLiter() \n  ");
            PumpLiter();
            break;
        case 10:
            System.out.println("\n Operation:  StopPump() \n  ");
            StopPump();
            break;
        case 11:
            System.out.println("\n Operation:  Receipt() \n  ");
            Receipt();
            break;
        case 12:
            System.out.println("\n Operation:  NoReceipt() \n  ");
            NoReceipt();
            break;
        case 13:
            cp = false;
            break;
        default:
```

```
                                System.out.println("Invalid Choice Selected");
                        break;
                }
        }
}

public void Activate(float a, float b, float c) {
        if(a > 0 && b>0 && c>0){
                db=(DataStore2) af.getdata();
                db.tempa=a;
                db.tempb=b;
                db.tempc=c;
                mde.activate();
        }
        else {
                System.out.println("Gas Price must be greater than 0");
        } //Gasprice can never be less then or equal to zero.
}

public void Cancel() {
        mde.cancel_msg();
}
public void start() {
        mde.start();
}
public void PayCash(float c) {
        db=(DataStore2) af.getdata();
```

```
            db.tempw=0;
            db.tempd=c;
            mde.pay_cash();
    }
    public void Premium() {
            mde.select_gas(3);
    }
    public void Regular() {
            mde.select_gas(1);
    }
    public void Super() {
            mde.select_gas(2);
    }
    public void Debug(){

    }


    public void StartPump() {
            System.out.println("\n~~~~~~~~~~GasPump Begins~~~~~~~~~~~~~~~");
            db=(DataStore2) af.getdata();
            db.tempL=0;
            mde.start_pump();
    }
    public void PumpLiter() {
            System.out.println("\n~~~~~~~~~GasPump2 pumps~~~~~~~~~~~~");
               float cash = db.gettemp_d();
```

```
                float price = db.getpcf();
                int liter = db.getL() + 1;
                //System.out.println(cash + "DEBUG price and liter" + price + " " + liter);
                if ( cash < (price * liter)){
                        mde.stop_pump();
                        }else
                                mde.pump_unite();

            //}
        }
        public void StopPump() {
                mde.stop_pump();
        }

        public void Receipt() {
                mde.receipt();
        }
        public void NoReceipt() {
                mde.no_receipt();
        }
}
```

MDAEFSM

```
package MDEFSM;

import States.S0;
```

```
import States.S1;
import States.S2;
import States.S3;
import States.S4;
import States.S5;
import States.S6;
import States.States;
import States.start;


public class MDEFSM {

	States[] state;
	int id = 0;
	//Includes all the states with the constructor

	public MDEFSM() {
		//Initialize all the states.
		state=new States[8];
		state[0]=new start();
		state[1]=new S0();
		state[2]=new S1();
		state[3]=new S2();
		state[4]=new S3();
		state[5]=new S4();
		state[6]=new S5();
		state[7]=new S6();
```

```
        //MDEFSM MDAEFSM = new MDAEFSM();
        //Initialize MDAEFSM
        //state[8]=new S7();]
}

//activates the gas pump system
public void activate() {
        state[id].activate();
        if(state[id].get_ID()==0) {
                id=1;
        }
}
//it starts the gas pump and changes the state
public void start() {
        state[id].start();
        if(state[id].get_ID()==1) {
                id=2;
        }
}
//change ID's
public void change_Id(int id) {
        this.id=id;
}
public void debug_op(){

}
public void pay_credit() {
```

```
            state[id].payCredit();
            if(state[id].get_ID()==2){
                  id=3;
            }
}

public void debug_pay_credit(){

}
public void pay_cash() {
            state[id].payCash();
            if(state[id].get_ID()==2) {
                  id=4;
            }
}
public void debug_pay_cash(){

}
public void reject() {
            state[id].reject();
            if(state[id].get_ID()==3){
                  id=1;
            }
}
public void approved() {
            state[id].approved();
            if(state[id].get_ID()==3) {
```

```
                id=4;
        }
}
public void cancel_msg() {
        state[id].cancel();
        if(state[id].get_ID()==4) {
                id=1;
        }
}
public void select_gas(int i) {
        state[id].selectGas(i);
        if(state[id].get_ID()==4) {
                id=5;
        }
}
public void start_pump() {
        state[id].startPump();
        if(state[id].get_ID()==5) {
                id=6;
        }
}
public void pump_unite() {
        state[id].pumpUnit();
        if(state[id].get_ID()==6) {
                id=6;
        }
}
```

```
	//Changes state to stopPump
	public void stop_pump() {
		state[id].stopPump();
		if(state[id].get_ID()==6) {
			id=7;
		}
	}
	public void receipt() {
		state[id].receipt();
		if(state[id].get_ID()==7) {
			id=1;
		}
	}
	public void no_receipt() {
		state[id].noReceipt();
		if(state[id].get_ID()==7) {
			id=1;
		}
	}

}
```

Output Processor

DisplayMenu

```
package OutputProcesser;
```

```
public abstract class DisplayMenu{

	public abstract void displayMenu();

}
```

DisplayMenu1

```
package OutputProcesser;

public class DisplayMenu1 extends DisplayMenu{

	@Override
	//Displays menu for gasPump1 for available options
	public void displayMenu() {

		System.out.println(" \n Operations Available : \n 7. Super Gas \n 8. Regular Gas");
	}

}
```

DisplayMenu2

```
package OutputProcesser;



public class DisplayMenu2 extends DisplayMenu{

	@Override
```

```
	//Displays menu for gas pump 2
	public void displayMenu() {

		System.out.println(" \n Operations Available : \n 5. Regular \n 6. Super \n 7. Premium");

}
}
```

GasPumpedMsg

```
package OutputProcesser;

import Datastore.DataStore;

public abstract class GasPumpedMsg {
public abstract void gasPumpedMsg(DataStore db);
}
```

Gp1Msg

```
package OutputProcesser;

import Datastore.DataStore;

public class GpMsg1 extends GasPumpedMsg{

	@Override
	public void gasPumpedMsg(DataStore db) {
```

```
                System.out.println("\n"+db.getG()+" Gallon Gas Pumped  "  );
        }
}
```

Gp2Msg

```
package OutputProcesser;

import Datastore.DataStore;

public class GpMsg2 extends GasPumpedMsg {

        @Override
        public void gasPumpedMsg(DataStore db) {
                System.out.println("\n"+db.getL()+" Liter Gas Pumped  " );
        }

}
```

OP

```
package OutputProcesser;
import Datastore.DataStore;
import Factory.AbstractFactory;

public class OP {
        AbstractFactory af;
        StoreData ds1;
        DataStore db1;
        PayMsg pm;
        GasPumpedMsg gm;
```

```
SetW sw;
DisplayMenu dm;
/*StoreCash sc;*/
SetPrice sp;
SetInitialValue sv;
PumpGasUnit pu;
StopMsg sm;
Receipt rcp;

//Sets initial value
public void setini_val() {
        af=DataStore.af;
        sv=af.getsv();
   db1=af.getdata();
   sv.setIniVal(db1);
}
//Stores data
public void storeData() {
        af=DataStore.af;
        db1=af.getdata();
        ds1=af.getdatastore();
        ds1.datastore(db1);
}

public OP() {
        af=DataStore.af;
}
```

```
    //Display's menu
    public void display_menu() {
            af=DataStore.af;
            dm=af.getdm();
            dm.displayMenu();
    }

    //print receipts
    public void print_receipt() {
            af=DataStore.af;
            rcp=af.getrcp();
            db1=af.getdata();
            rcp.printReceipt(db1);
    }
    /*public void store_cash() {
            af=DataStore.af;
sc=af.getsc();
db1=af.getdata();
sc.storeCash(db1);
    }*/
    public void set_price(int n) {
            af=DataStore.af;
            sp=af.getpsp();
            db1=af.getdata();
            sp.setPrice(n, db1);
    }
    //display reject message
```

```
public void reject_msg() {
        System.out.println("Invalid Card! Card Rejected!!");
}
//pumps one unit
public void pump_unit() {
        af=DataStore.af;
        pu=af.getpu();
        db1=af.getdata();
        pu.pumpGasUnit(db1);
}
/*
 * Print Receipt
 */
public void set_w() {
        af=DataStore.af;
        db1=af.getdata();
        sw=af.get_w();
        sw.setW(db1);
}
//display pay_msg
public void pay_Msg() {
        af=DataStore.af;
   pm=af.getpaymsg();
   pm.payMsg();
}
public void pay_credit() {
        System.out.println("Approval Pending!!");
```

```
        }


        public void gp_msg() {
           af=DataStore.af;
           gm=af.getGpMsg();
           db1=af.getdata();
           gm.gasPumpedMsg(db1);;
        }
        public void stop_msg() {
           af=DataStore.af;
           sm=af.getsm();
           sm.stopMsg();
        }
        public void ready_msg() {
                System.out.println("\n Pumping. . . . ");
        }
        public void cancel_msg() {
                System.out.println("\n Transaction is cancelled!!");
        }
}
```

PayMsg

```
package OutputProcesser;

public abstract class PayMsg {
```

```
	public abstract void payMsg();



}
```

PayMsg1

```
package OutputProcesser;

public class PayMsg1 extends PayMsg {

	//Display's the available methods for gaspump1
	public void payMsg() {
		System.out.println("Only Credit Card!");
	}

}
```

PayMsg2

```
package OutputProcesser;

public class PayMsg2 extends PayMsg{
	@Override
	//Displays the payment options
	public void payMsg() {
		System.out.println("Cash Only! ");
	}
}
```

PumpGasUnit

```
package OutputProcesser;

import Datastore.DataStore;

public abstract class PumpGasUnit {

	public abstract void pumpGasUnit(DataStore db);

}
```

PumpGasUnit1

```
package OutputProcesser;

import Datastore.DataStore;

public class PumpGasUnit1 extends PumpGasUnit {
	@Override
		//Calculates total about and number of gallons pumped for gas pump1
		public void pumpGasUnit(DataStore db) {
		db.setG(db.getG()+1);
		db.settotal(db.getpcf()*db.getG());
	}
}
```

PumpGasUnit2

```
package OutputProcesser;

import Datastore.DataStore;

public class PumpGasUnit2 extends PumpGasUnit{

	@Override
	//calculate toal number of litres and value pf l for gas pump 2.
	public void pumpGasUnit(DataStore db) {
	  db.setL(db.getL()+1);
	  //System.out.println(db.getL() + " AND " + db.getpcf());
	  db.settotal(db.getpcf()*db.getL());
	}

}
```

Receipt

```
package OutputProcesser;

import Datastore.DataStore;
import Datastore.DataStore;
import Datastore.DataStore1;


public abstract class Receipt {

	public abstract void printReceipt(DataStore db);
```

```
}
```

Receipt1

```
package OutputProcesser;


import Datastore.DataStore;
public class Receipt1 extends Receipt{


	public void printReceipt(DataStore db) {
		System.out.println("\n ~~~~~Receipt GasPump1~~~~~ \n Your total is "
+db.gettotalf());
	}
}
//tells the total amount and print it
```

Receipt2

```
package OutputProcesser;
import Datastore.DataStore;



	public class Receipt2 extends Receipt{


		public void printReceipt(DataStore db) {
			float total = db.getpcf() * db.getL();
			float bal = db.gettemp_d() - total;
			System.out.println("\n ~~~~~Receipt GasPump2~~~~~ \n Your total is
"+total);
			System.out.println("\n Remaining balance is " +bal);
```

```
                }
}
```

SetInitialValue

```
package OutputProcesser;

import Datastore.DataStore;
//sets initial value for both the pumps
public abstract class SetInitialValue {

public abstract void setIniVal (DataStore db);

}
```

SetInitialValue1

```
package OutputProcesser;

import Datastore.DataStore;

public class SetInitialValue1 extends SetInitialValue {
        public void setIniVal(DataStore db) {

                db.setG(db.gettemp_G());
        }
}
```

SetInitialValue2

```
package OutputProcesser;
```

```
import Datastore.DataStore;

public class SetInitialValue2 extends SetInitialValue {

	public void setIniVal(DataStore db) {
		db.setL(db.gettemp_L());
	}
}
```

setPrice

```
package OutputProcesser;

import Datastore.DataStore;
import Datastore.DataStore;
import Datastore.DataStore1;



public abstract class SetPrice {
	public abstract void setPrice(int n,DataStore db);

}
```

setPrice1

```
package OutputProcesser;
```

```
import Datastore.DataStore;
public class SetPrice1 extends SetPrice {
	@Override
	public void setPrice(int n, DataStore db) {
		if(n==1) {
			db.setpc(db.getprice_r());//sets price for regular
		}
		else if(n==2) {
			db.setpc(db.getprice_s());//sets price for supreme
		}
	}
}
```

setPrice2

```
package OutputProcesser;

import Datastore.DataStore;

public class SetPrice2 extends SetPrice {

	@Override
	public void setPrice(int n, DataStore db) {
		if(n==1) {
			//Test-System.out.println("this is " + db.getprice_r());
			db.setpc(db.getprice_r()); //Sets price for regular gas, gaspump2
		}
		else if(n==2) {
			//System.out.println("this is " + db.getprice_s());
```

```
                db.setpc(db.getprice_s());//Sets price for super gas, gaspump2
            }
            else if(n==3) {
                //System.out.println("this is " + db.getprice_p());
                db.setpc(db.getprice_p());//Sets price for premium gas, gaspump2
            }
        }

}
```

stopMsg

```
package OutputProcesser;

public abstract class StopMsg {

    public abstract void stopMsg();

}
```

stopMsg1

```
package OutputProcesser;

public class StopMsg1 extends StopMsg {
    @Override
    public void stopMsg() {
        System.out.println("\n Pumping Stopped! Add Cash!  ");
    }
}
```

storeCash

```
package OutputProcesser;

import Datastore.DataStore;

public class StoreCash {
	//stores cash for gas pump-2
	public void storeCash(DataStore db) {
		db.setc(db.gettemp_d());
	}
}
```

storeData

```
package OutputProcesser;

import Datastore.DataStore;

public abstract class StoreData {

	public abstract void datastore(DataStore db1);
}
```

storeData1

```
package OutputProcesser;

import Datastore.DataStore;

public class StoreData1 extends StoreData {
```

```
	public void datastore(DataStore db1) {
		float temp1=db1.gettemp_2a();
		float temp2=db1.gettemp_2b();
		db1.setprice_r(temp1);
		db1.setprice_s(temp2);;
	}
}
```

storeData2

```
package OutputProcesser;

import Datastore.DataStore;

public class StoreData2 extends StoreData {

	//stores data
	public void datastore(DataStore db1) {
		float temp1=db1.gettemp_2a();
		float temp2=db1.gettemp_2b();
		float temp3=db1.gettemp_2c();
		db1.setprice_r(temp1);
		db1.setprice_s(temp2);
		db1.setprice_p(temp3);
	}
}
```

Data

Datastore

```
package Datastore;

import Factory.AbstractFactory;

public abstract class DataStore {

	public static AbstractFactory af;

	public int gettemp_a() {
		return 0;
	};

	public float gettemp_2a() {
		return 0;
	};

	public float gettemp_2b() {
		return 0;
	};

	public float gettemp_2c() {
		return 0;
	};

	public void setprice(int price) {
```

```
}

public void setprice_p(float price){

}

public void setprice_r(float price) {
}

public void setprice_s(float price) {
}

public int getprice() {
        return 0;
}

public float getprice_r() {
        return 0;
}

public float getprice_s() {
        return 0;
}

public float getprice_p(){
        return 0;
}
```

```
public int gettw() {
        return 0;
}

public int getw() {
        return 0;
}

public void setw(int w) {
}

public int gettemp_c() {
        return 0;
}

public float gettemp_d() {
        return 0;
}

public int getc() {
        return 0;
}

public void setc(int c) {
}
```

```
public float getd() {
        return 0;
}

public void setc(float c) {
}

public int getpc() {
        return 0;
}

public float getpcf() {
        return 0;
}

public void setpc(int pc) {
}

public void setpc(float pc) {
}

public int gettemp_G() {
        return 0;
}

public int gettemp_L() {
        return 0;
```

```
}

public int getG() {
        return 0;
}

public int getL() {
        return 0;
}

public void setG(int c) {
}

public void setL(int c) {
}

public int gettotal() {
        return 0;
}

public float gettotalf() {
        return 0;
}

public void settotal(int t) {
}
```

```
        public void settotal(float t) {
        }
}
```

Datastore1

```
package Datastore;

public class DataStore1 extends DataStore {

        public float rpriced;
        public float spriced;
        public float tempa;
        public float tempb;
        public int tempw;
        public float ppriced;
        public int tempq;
        public int wd;
        public int tempc;
        public int cd;
        public int dc;
        public float price_peg;
        public int tempG;
        public int Gd;
   public float total;
   public float pricec;
   public float price_reg;
```

```
        public float gettemp_2a() {
                return tempa;
        }
        public float gettemp_2b() {
                return tempb;
        }

        public void setprice_r(float price) {
                rpriced=price;
        }
        public void setprice_s(float price) {
                spriced=price;
        }
    public int gettw()
    {
    return tempw;
    }
    public int getw()
    {return wd;}
    public void setw(int w)
    {
        wd=w;    }
    public int gettempc(){
        return tempc;
        }
    public int getc(){
        return cd;
```

```
        }
    public void setc(int c){
        cd=c;
    }
    public float getpcf(){
        return pricec;}
    public void setpc(float pc){
        pricec=pc;
        }
    public int gettemp_G()
    {return tempG;
    }
    public int getG(){
        return Gd;
        }
    public void setG(int d){
        Gd=d;
        }
    public float getprice_r(){
        return rpriced;
        }
    public float getprice_s(){
        return spriced;
        }
    public float gettotalf()
    {return total;}
    public void settotal(float t){
```

```
        total=t;
        }
}
```

Datastore2

```
package Datastore;

public class DataStore2 extends DataStore {

        public float rpriced;
        public float spriced;
        public float tempa;
        public float tempb;
        public int tempw;
        public int tama;
        public float tempo;
        public int wd;
        public int tempL;
        public int Ld;
    public float total;
        public float tempd;
        public float cd;
        public float rssb;
    public float pricec;
    public float ppriced;
    public float tempc;
```

```
	public float gettemp_2a() {
		return tempa;
	}
	public float gettemp_2b() {
		return tempb;
	}
	public float gettemp_2c() {
		return tempc;
	}
	public float getprice_p(){
		return ppriced;
	}
	public float getprice_r(){
	return rpriced;
}
public float getprice_s(){
	return spriced;
}
public float gettotalf(){
	return total;
}
public void settotal(float t){
	total=t;
}
	public void setprice_r(float price) {
		rpriced=price;
	}
```

```
        public void setprice_s(float price) {
                spriced=price;
        }

        public void setprice_p(float price) {
                ppriced=price;
        }

public float gettemp_d(){
        return tempd;
}
public float getd(){
        return cd;
}
public void setc(float c){
        cd=c;
}
public float getpcf(){
        //System.out.println("this is getpcf" + pricec);
        return pricec;
}
public void setpc(float pc){
        //System.out.
        //println("setpc " + pc);
        pricec=pc;
        //System.out.
        //println("setpc after init " + pricec);
```

```
}
public int gettemp_L(){
        return tempL;
}
public int getL(){
        return Ld;
}
public void setL(int d){
        Ld=d;
}
}
```

States

Start

```
package States;

public class start extends States{
        public final int ID=0;
        @Override
        public void activate() {
          //System.out.println("In state start");
                op.storeData();
        }
        @Override
        public int get_ID() {
                return ID;
```

```
}
@Override
public void start() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }
@Override
public void payCredit() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }
@Override
public void reject() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
}
@Override
public void approved() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
}
@Override
public void cancel() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
}
@Override
public void payCash() {
        System.out.println("\n  This is an invalid State!"
```

```
                        +"\n Try again with an Appropiate operation... ");
    }

    @Override
    public void startPump() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
    }
    @Override
    public void selectGas(int n) {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
    }
    @Override
    public void pumpUnit() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
    }
    @Override
    public void stopPump() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
    }
    @Override
    public void receipt() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
```

```
	}
	@Override
	public void noReceipt() {
		System.out.println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");
	}
}
```

S0

```
package States;

public class S0 extends States{

	public final int ID=1;
	@Override
	public void activate() {

		System.out.
		println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");
	}

	@Override
	public int get_ID() {
		return ID;
	}
```

```
@Override
public void start() {

        op.pay_Msg();
}

@Override
public void payCredit() {

        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");

}

@Override
public void reject() {

        System.out.println("\n   This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");

}

@Override
public void approved() {

        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
```

```
}

@Override
public void cancel() {

        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");

}

@Override
public void payCash() {

        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");

}


@Override
public void startPump() {

        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");

}
```

```
@Override
public void selectGas(int n) {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");

}

@Override
public void pumpUnit() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");

}

@Override
public void stopPump() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");

}

@Override
public void receipt() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
```

```
	}

	@Override
	public void noReceipt() {
		System.out.println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");

	}
}
```

S1

```
package States;

public class S1 extends States{
	public final int ID=2;
	@Override
	public void activate() {
		// TODO Auto-generated method stub
		System.out.println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");

	}

	@Override
	public int get_ID() {
		return ID;
	}
```

```
@Override
public void start() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");

}

@Override
public void payCredit() {

        op.pay_credit();

}

@Override
public void reject() {

        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }

@Override
public void approved() {
        System.out
        .println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }
```

```
@Override
public void cancel() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }



@Override
public void startPump() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
        }

@Override
public void selectGas(int n) {

        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
        }

@Override
public void pumpUnit() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
}

@Override
```

```
    public void stopPump() {
        System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void receipt() {

    }

    @Override
    public void noReceipt() {
        System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void payCash() {

    }
    //@Override
    //public void debug_receipt_state{

    //}

}
```

S2

```
package States;
```

```
public class S2 extends States{
	final public int ID=3;
	@Override
	public void activate() {
		System.out.println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");   }

	@Override
	public int get_ID() {
		return ID;
	}

	@Override
	public void start() {
		System.out.println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");
		}

	@Override
	public void payCredit() {
		System.out.println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");   }

	@Override
	public void reject() {
```

```
  op.reject_msg();
}

@Override
public void approved() {

op.set_w();
op.display_menu();
}

@Override
public void cancel() {

        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
}

@Override
public void payCash() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }




@Override
public void startPump() {
        System.out.println("\n  This is an invalid State!"
```

```
                        +"\n Try again with an Appropiate operation... ");   }

@Override
public void selectGas(int n) {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }

@Override
public void pumpUnit() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }

@Override
public void stopPump() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }

@Override
public void receipt() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
}

@Override
public void noReceipt() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
```

```
	}
}
```

S3

```
package States;


public class S3 extends States{
	public final int ID=4;
	@Override
	public void activate() {
		System.out.println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");
	}

	@Override
	public int get_ID() {
		return ID;
	}

	@Override
	public void start() {		System.out.println("\n  This is an invalid State!"
			+"\n Try again with an Appropiate operation... ");
	}

	@Override
	public void payCredit() {			System.out.println("\n  This is an invalid State!"
			+"\n Try again with an Appropiate operation... ");   }
```

```
    @Override
    public void reject() {              System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
    }

    @Override
    public void approved() {                    System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void cancel() {
op.cancel_msg();
    }

    @Override
    public void payCash() {
            System.out.println("\n  This is an invalid State!"
                                +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void startPump() {
            System.out.println("\n  This is an invalid State!"
                                +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void selectGas(int n) {
            op.set_price(n);
```

```
    }

    @Override
    public void pumpUnit() {
        System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void stopPump() {
        System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void receipt() {
        System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void noReceipt() {
        System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

}
```

S4

```
package States;

public class S4 extends States{
```

```
public final int ID=5;
@Override
public void activate() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }

@Override
public int get_ID() {
        return ID;
}

@Override
public void start() {           System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void payCredit() {               System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void reject() {          System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void approved() {                System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }
```

```
@Override
public void cancel() {            System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void payCash() {                  System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void selectGas(int n) {           System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void startPump() {
        op.setini_val();
        op.ready_msg();

}

@Override
public void pumpUnit() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }

@Override
public void stopPump() {                 System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");
```

```
	}

	@Override
	public void receipt() {		System.out.println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");
	}

	@Override
	public void noReceipt() {		System.out.println("\n  This is an invalid State!"
				+"\n Try again with an Appropiate operation... ");   }

}
```

S5

```
package States;

public class S5 extends States{
	public final int ID=6;
	@Override
	public void activate() {
		System.out.println("\n  This is an invalid State!"
					+"\n Try again with an Appropiate operation... ");   }

	@Override
	public int get_ID() {
		return ID;
	}
```

```
@Override
public void start() {
        System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");
}

@Override
public void payCredit() {                System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void reject() {             System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void approved() {                System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void cancel() {             System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void payCash() {                System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
```

```
    public void selectGas(int n) {          System.out.println("\n  This is an invalid State!"
                    +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void startPump() {               System.out.println("\n  This is an invalid State!"
                    +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void pumpUnit() {
    op.pump_unit();
    op.gp_msg();
    }

    @Override
    public void stopPump() {
    op.stop_msg();
    }

    @Override
    public void receipt() {
            System.out.println("\n  This is an invalid State!"
                            +"\n Try again with an Appropiate operation... ");   }

    @Override
    public void noReceipt() {               System.out.println("\n  This is an invalid State!"
                    +"\n Try again with an Appropiate operation... ");   }
}
```

S6

```
package States;

public class S6 extends States{
public final int ID=7;

        @Override
        public void activate() {
                System.out.println("\n  This is an invalid State!"
                                +"\n Try again with an Appropiate operation... ");   }

        @Override
        public int get_ID() {

                return ID;
        }

        @Override
        public void start() {
                System.out.println("\n  This is an invalid State!"
                                +"\n Try again with an Appropiate operation... ");   }

        @Override
        public void payCredit() {                   System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }
```

```
@Override
public void reject() {          System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void approved() {              System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void cancel() {          System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void payCash() {              System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void selectGas(int n) {          System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void startPump() {              System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }

@Override
public void pumpUnit() {              System.out.println("\n  This is an invalid State!"
                +"\n Try again with an Appropiate operation... ");   }
```

```
        @Override
        public void stopPump() {                System.out.println("\n  This is an invalid State!"
                        +"\n Try again with an Appropiate operation... ");   }

        @Override
        public void receipt() {
        op.print_receipt();

        }

        @Override
        public void noReceipt() {
         System.out.println("No receipt!");

        }
}
```

Abstract Factory

```
package Factory;


import Datastore.DataStore;
import OutputProcesser.PayMsg;
import OutputProcesser.StoreData;
import OutputProcesser.DisplayMenu;
```

```
import OutputProcesser.GasPumpedMsg;
import OutputProcesser.PumpGasUnit;
import OutputProcesser.PayMsg;
import OutputProcesser.Receipt;
import OutputProcesser.StoreCash;
import OutputProcesser.StopMsg;
import OutputProcesser.StoreCash;
import OutputProcesser.SetPrice;
import OutputProcesser.SetInitialValue;
import OutputProcesser.SetW;



public abstract class AbstractFactory {
public abstract DataStore getdata();
public abstract GasPumpedMsg getGpMsg();
public abstract SetInitialValue getsv();
public abstract PumpGasUnit getpu();

        public abstract PayMsg getpaymsg();
        public abstract StoreData getdatastore();
   public abstract Receipt getrcp();

   public abstract SetW get_w();
   public abstract StoreCash getsc();
   public abstract SetPrice getpsp();
   public abstract DisplayMenu getdm();
```

```
    public abstract StopMsg getsm();



}
```

Concrete Factory1

```
package Factory;

import Datastore.DataStore;
import Datastore.DataStore1;
import Datastore.DataStore2;
import OutputProcesser.DisplayMenu;
import OutputProcesser.DisplayMenu1;
import OutputProcesser.SetW;
import OutputProcesser.SetW1;
import OutputProcesser.StopMsg1;
import OutputProcesser.StoreData;
import OutputProcesser.StoreData1;
import OutputProcesser.GasPumpedMsg;
import OutputProcesser.GpMsg1;
import OutputProcesser.PumpGasUnit;
import OutputProcesser.PumpGasUnit1;
import OutputProcesser.PayMsg;
import OutputProcesser.Receipt;
import OutputProcesser.Receipt1;
import OutputProcesser.StoreCash;
import OutputProcesser.StopMsg;
import OutputProcesser.SetPrice;
```

```
import OutputProcesser.SetPrice1;
import OutputProcesser.SetInitialValue;
import OutputProcesser.SetInitialValue1;
import OutputProcesser.PayMsg1;



public class CFactory1 extends AbstractFactory{
        DataStore1 db1;
          StoreData1 ds1;
          PayMsg1 paymsg1;
          GpMsg1 gmMsg1;
           SetW1 sw1;
           StoreCash sc2;
           float a1;
           float b1;


                SetPrice1 sp1;
                SetInitialValue1 sv1;
                DisplayMenu1 dm1;
                PumpGasUnit1 pu1;
                StopMsg1 sm1;
                Receipt1 rcp1;
                /*
                 * Concrete Factory for GasPump1
                 */
```

```
        public CFactory1() {
                db1 =new DataStore1();
           ds1=new StoreData1();
           paymsg1=new PayMsg1();
           gmMsg1 = new GpMsg1();
           sw1=new SetW1();
           sc2=new StoreCash();
           sp1=new SetPrice1();
           sv1=new SetInitialValue1();
           dm1=new DisplayMenu1();
           pu1=new PumpGasUnit1();
           sm1=new StopMsg1();
           rcp1=new Receipt1();
    }

        @Override
        public DataStore getdata() {
                return db1;
        }



        @Override
        public StoreData getdatastore() {
                return ds1;
        }
```

```
@Override
public PayMsg getpaymsg() {
        return paymsg1;
}

@Override
public SetW get_w() {
        return sw1;
}


@Override
public StoreCash getsc() {
        return sc2;
}

@Override
public SetPrice getpsp() {
        return sp1;
}

@Override
public SetInitialValue getsv() {
        return sv1;
}
@Override
public DisplayMenu getdm() {
```

```
        return dm1;
    }


    @Override
    public PumpGasUnit getpu() {
        return pu1;
    }

    @Override
    public StopMsg getsm() {
        return sm1;
    }

    @Override
    public Receipt getrcp() {
        return rcp1;
    }

    @Override
    public GasPumpedMsg getGpMsg() {
        return gmMsg1;
    }

}
```

Concrete Factory2

```
package Factory;
```

```
import Datastore.DataStore2;
import Datastore.DataStore1;
import Datastore.DataStore;
import OutputProcesser.DisplayMenu;
import OutputProcesser.DisplayMenu2;
import OutputProcesser.DisplayMenu1;
import OutputProcesser.SetW;
import OutputProcesser.SetW1;
import OutputProcesser.StopMsg1;
import OutputProcesser.StoreData;
import OutputProcesser.StoreData2;
import OutputProcesser.GasPumpedMsg;
import OutputProcesser.GpMsg2;
import OutputProcesser.PumpGasUnit;
import OutputProcesser.PumpGasUnit2;
import OutputProcesser.PumpGasUnit1;
import OutputProcesser.PayMsg;
import OutputProcesser.Receipt;
import OutputProcesser.Receipt1;
import OutputProcesser.Receipt2;
import OutputProcesser.Receipt1;
import OutputProcesser.StoreCash;
import OutputProcesser.StopMsg;
import OutputProcesser.SetPrice;
import OutputProcesser.SetPrice2;
import OutputProcesser.SetInitialValue;
```

```
import OutputProcesser.SetInitialValue2;
import OutputProcesser.PayMsg2;
import OutputProcesser.PayMsg1;

public class CFactory2 extends AbstractFactory {

        DataStore2 db2;
   StoreData2 ds2;
   PayMsg2 paymsg2;
   GpMsg2 gmMsg2;
        StoreCash sc2;
        SetPrice2 sp2;
        SetInitialValue2 sv2;
   SetW1 sw1;
        DisplayMenu2 dm2;
        PumpGasUnit2 pu2;
        StopMsg1 sm1;
        Receipt2 rcp1;


        public CFactory2() {
                db2 =new DataStore2();
           ds2=new StoreData2();
           paymsg2=new PayMsg2();
           gmMsg2 = new GpMsg2();
           sw1=new SetW1();
           sc2=new StoreCash();
```

```
        sp2=new SetPrice2();
        sv2=new SetInitialValue2();
        dm2=new DisplayMenu2();
        pu2=new PumpGasUnit2();
        sm1=new StopMsg1();
        rcp1=new Receipt2();
    }

    @Override
    public DataStore getdata() {
        return db2;
    }

    @Override
    public StoreData getdatastore() {
        return ds2;
    }

    @Override
    public PayMsg getpaymsg() {
        return paymsg2;
    }

    @Override
    public SetW get_w() {
        return sw1;
    }
```

```
@Override
public DisplayMenu getdm() {
        return dm2;
}

@Override
public StoreCash getsc() {
        return sc2;
}

@Override
public SetPrice getpsp() {
        return sp2;
}

@Override
public SetInitialValue getsv() {
        return sv2;
}

@Override
public PumpGasUnit getpu() {
        return pu2;
}

@Override
```

```
        public StopMsg getsm() {
                                return sm1;
        }

        @Override
        public Receipt getrcp() {
                return rcp1;
        }

        @Override
        public GasPumpedMsg getGpMsg() {
                return gmMsg2;
        }
}
```