Software System Architecture-Design Document

GASPUMP

Borde, Rohan

A20375497 | RBORDE@HAWK.IIT.EDU

INTRODUCTION

The goal of this project is to design two different GasPump components using the Model-Driven Architecture (MDA) and then implement these GasPump components based on this design. Both GasPump components are state-based components and are used to control simple gas pumps.
Users can pay by cash, a credit card or a debit card. The gas pump may dispose different types of the gasoline. The price of the gasoline is provided when the gas pump is activated. The detailed behavior of GasPump components is specified using EFSM. The EFSM of Figure 1 shows the detail behavior of GasPump-1 and the EFSM of Figure 2 shows the detailed behavior of GasPump-2. Notice that there are several differences between GasPump components.

The goal of this project is to design two GasPump components using the Model-Driven Architecture (MDA) covered in the course. In the first part of the project, you should design an executable metamodel, referred to as MDA-EFSM, for GasPump components. This MDA-EFSM should capture the "generic behavior" of both GasPump components and should be de-coupled from data and implementation details. Notice that in your design there should be ONLY one MDA-EFSM for both GasPump components. In addition, in the Model-Driven Architecture coupling between components should be minimized and cohesion of components should be maximized (components with high cohesion and low coupling between components). The meta-model (MDA-EFSM) used in the Model-Driven architecture should be expressed as an EFSM (Extended Finite State Machine) model. Notice that the EFSMs shown in Figure 1 and Figure 2 are not acceptable as a meta-model (MDA-EFSM) for this model driven architecture.

The goal of the second part of the project is to design two GasPump components using the Model-Driven Architecture (MDA) and then implement these GasPump components based on this design using the OO programming language. This OO-oriented design should be based on the MDA-EFSM for both GasPump components that was identified in the first part of the project. You may use your own MDAEFSM (assuming that it was correct) or you can use the posted sample MDA-EFSM. In your design, you MUST use the following OO design patterns:

- state pattern
- strategy pattern
- abstract factory pattern

MDA-EFSM MODEL FOR THE GASPUMP COMPONENTS

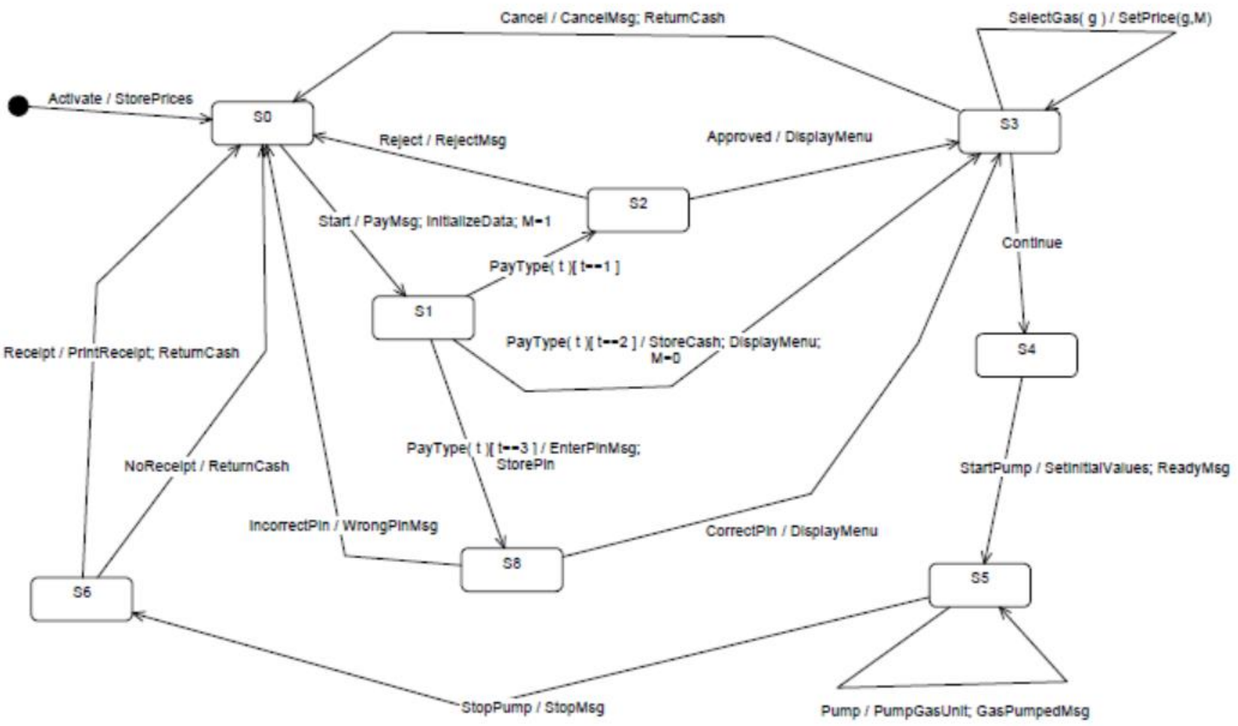
i. A list of meta events for the MDA-EFSM:

Activate()
Start()
PayType(int t) //credit: t=1; cash: t=2; debit: t=3
Reject()
Cancel()
Approved()
StartPump()
Pump()
StopPump()
SelectGas(int g) // Regular: g=1; Super: g=2; Premium: g=3; Diesel: g=4
Receipt()
NoReceipt()
CorrectPin()
IncorrectPin()
Continue()

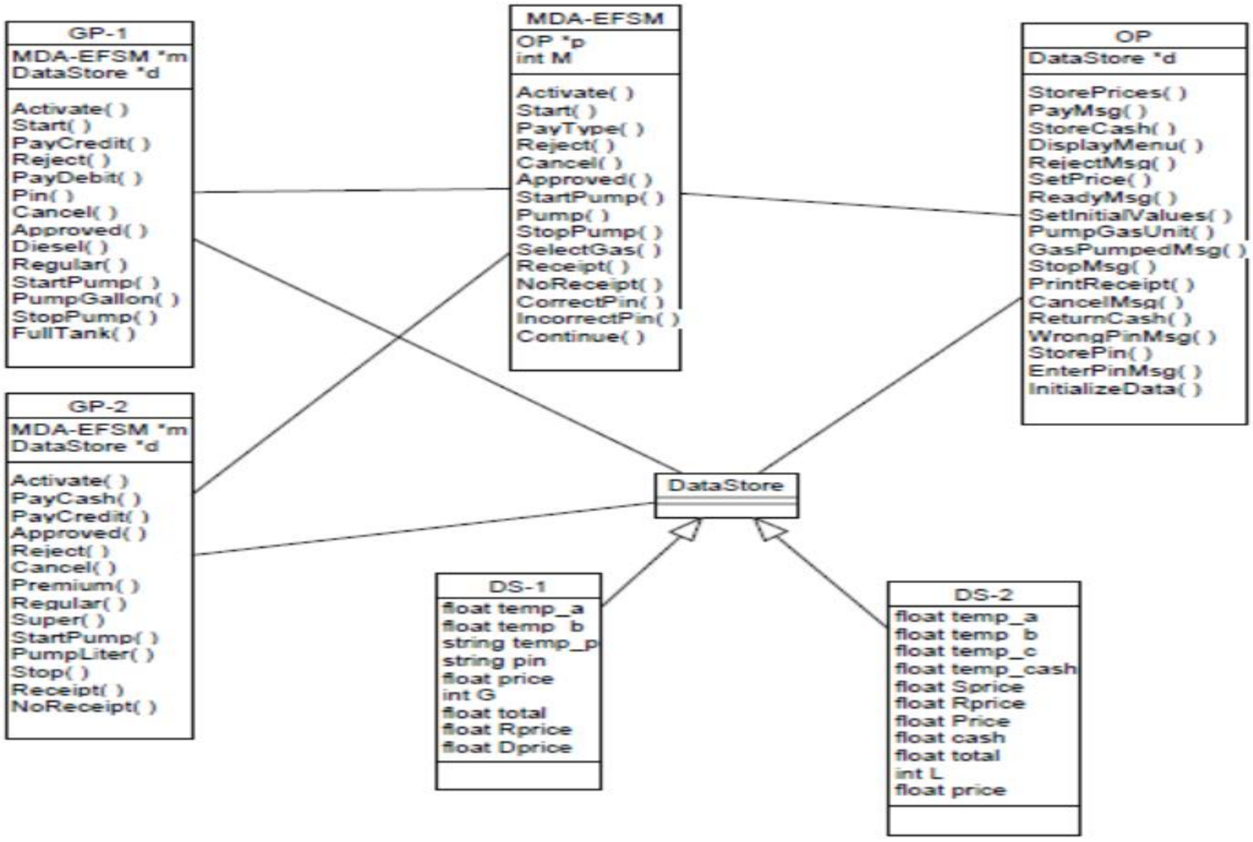
ii. A list of meta actions for the MDA-EFSM with their descriptions

StorePrices // stores price(s) for the gas from the temporary data store
PayMsg // displays a type of payment method
StoreCash // stores cash from the temporary data store
DisplayMenu // display a menu with a list of selections
RejectMsg // displays credit card not approved message
SetPrice(int g, int M) // set the price for the gas identified by g identifier as in SelectGas(int g); if M=1, the price may be increased
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
WrongPinMsg // displays incorrect pin message
StorePin // stores the pin from the temporary data store
EnterPinMsg // displays a message to enter pin
InitializeData // set the value of price and cash to 0

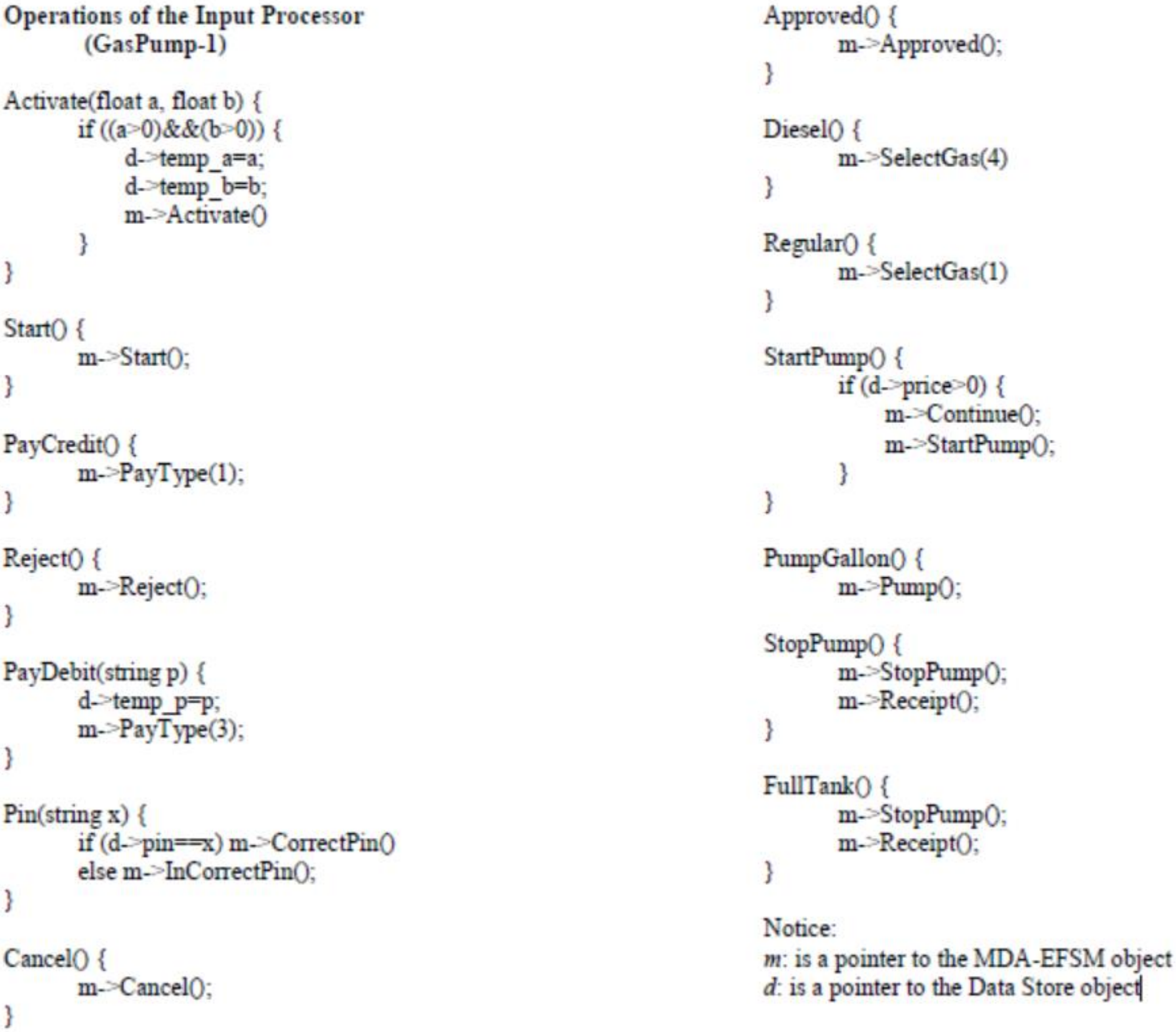
iii. A state diagram of the MDA-EFSM


Cancel / CancelMsg; ReturnCash
SelectGas(g) / SetPrice(g,M)
Activate / StorePrices
S0
S3
Reject / RejectMsg
Approved / DisplayMenu
S2
Start / PayMsg; InitializeData; M=1
Continue
PayType(t)[t==1]
S1
PayType(t)[t==2] / StoreCash; DisplayMenu; M=0
S4
Receipt / PrintReceipt; ReturnCash
PayType(t)[t==3] / EnterPinMsg; StorePin
StartPump / SetInitialValues; ReadyMsg
NoReceipt / ReturnCash
IncorrectPin / WrongPinMsg
CorrectPin / DisplayMenu
S8
S6
S5
StopPump / StopMsg
Pump / PumpGasUnit; GasPumpedMsg

MDA-EFSM for Gas Pumps



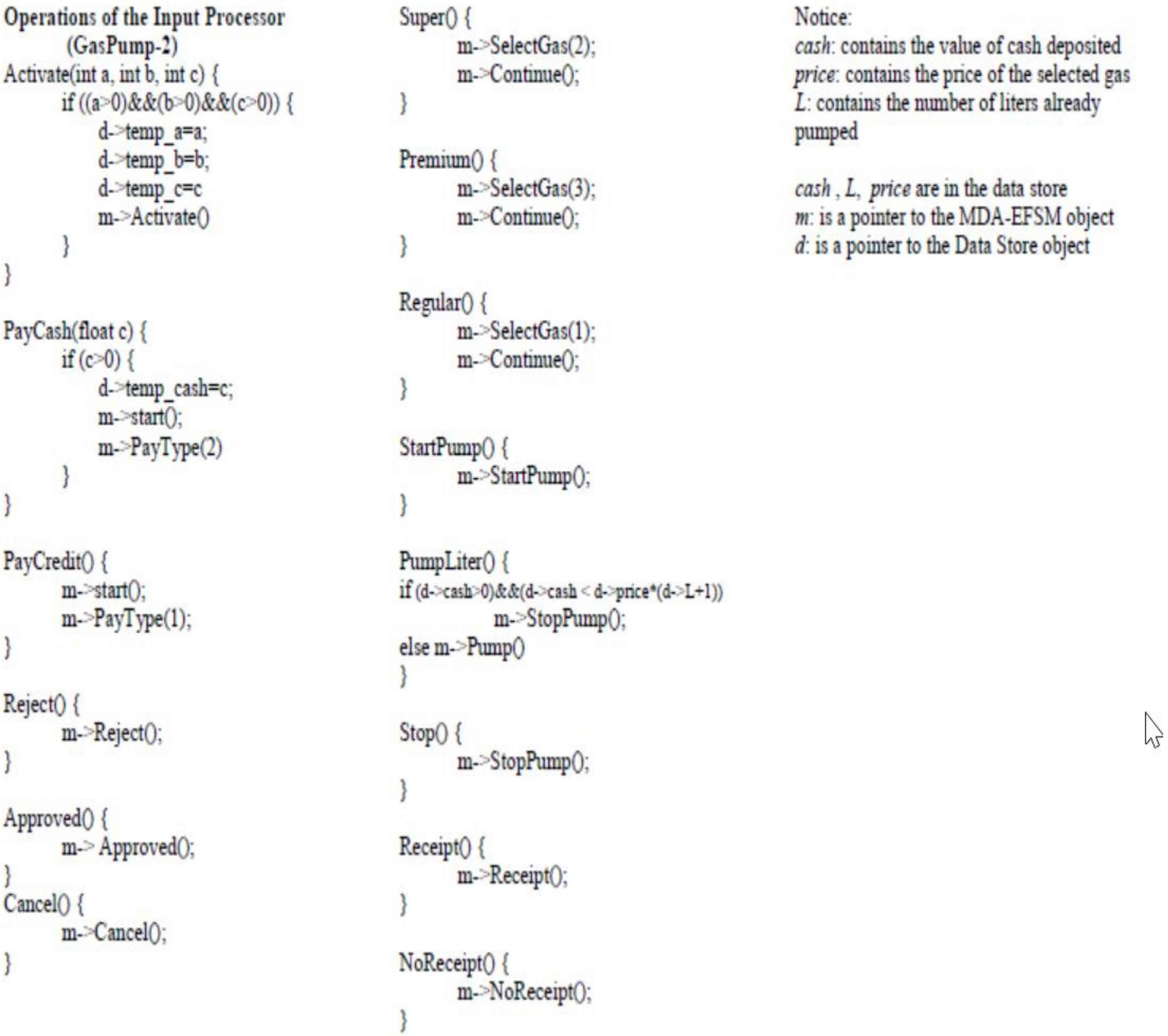
iv. Pseudo-code of all operations of Input Processors of GasPump-1 and GasPump-2



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->PayType(1);
}

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

PayDebit(string p) {
        d->temp_p=p;
        m->PayType(3);
}

Pin(string x) {
        if (d->pin==x) m->CorrectPin()
        else m->InCorrectPin();
}

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

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

Diesel() {
        m->SelectGas(4)
}

Regular() {
        m->SelectGas(1)
}

StartPump() {
        if (d->price>0) {
                m->Continue();
                m->StartPump();
        }
}

PumpGallon() {
        m->Pump();

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

FullTank() {
        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()
        }
}

PayCash(float c) {
        if (c>0) {
              d->temp_cash=c;
              m->start();
              m->PayType(2)
        }
}

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

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

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

```
Super() {
        m->SelectGas(2);
        m->Continue();
}

Premium() {
        m->SelectGas(3);
        m->Continue();
}

Regular() {
        m->SelectGas(1);
        m->Continue();
}

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

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

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

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

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

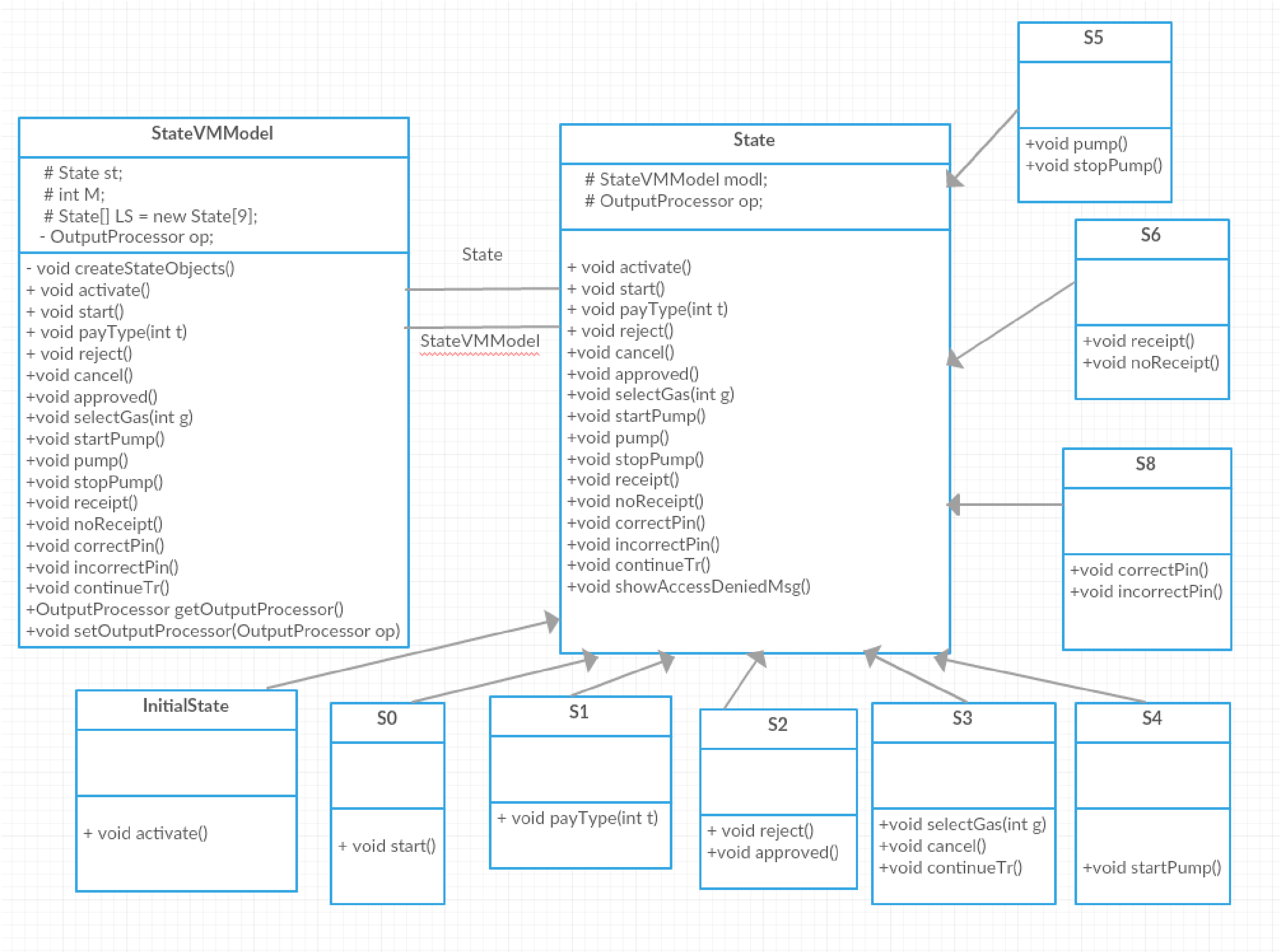
Notice:
cash: contains the value of cash deposited
price: contains the price of the selected gas
L: contains the number of liters already pumped

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

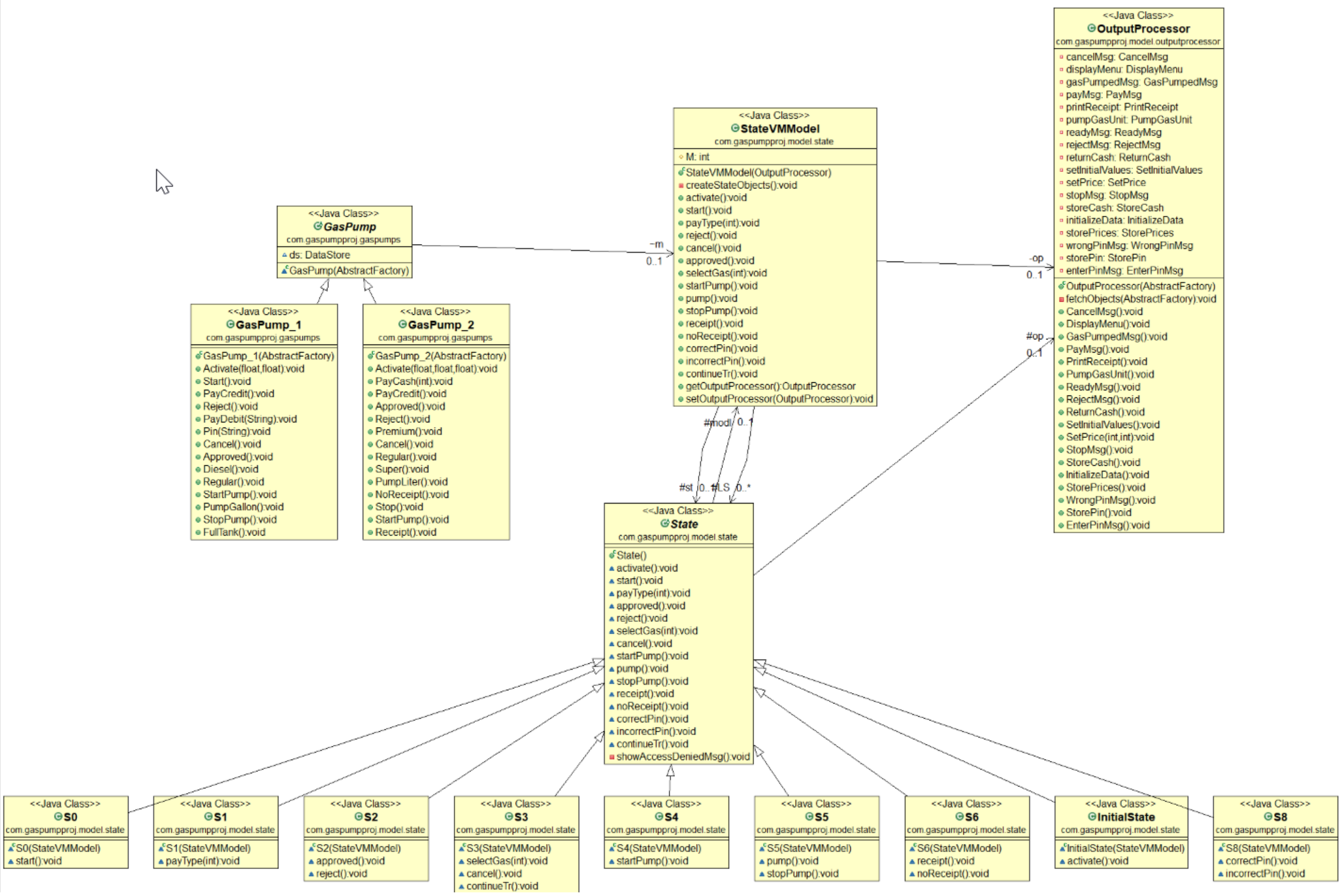
CLASS DIAGRAM(S) OF THE MDA OF THE GASPUMP COMPONENTS

i. State pattern

Classes related to this de-centralized state pattern are implemented in com.gaspumpproj.model.state package. Here, StateVMModel is the "VM" class. State is the abstract State superclass. There are state classes as InitialState, S0, S1, S2, S3, S4, S5, S6, S8 which are the subclasses of State class. These State classes are responsible for performing both actions and state transitions.

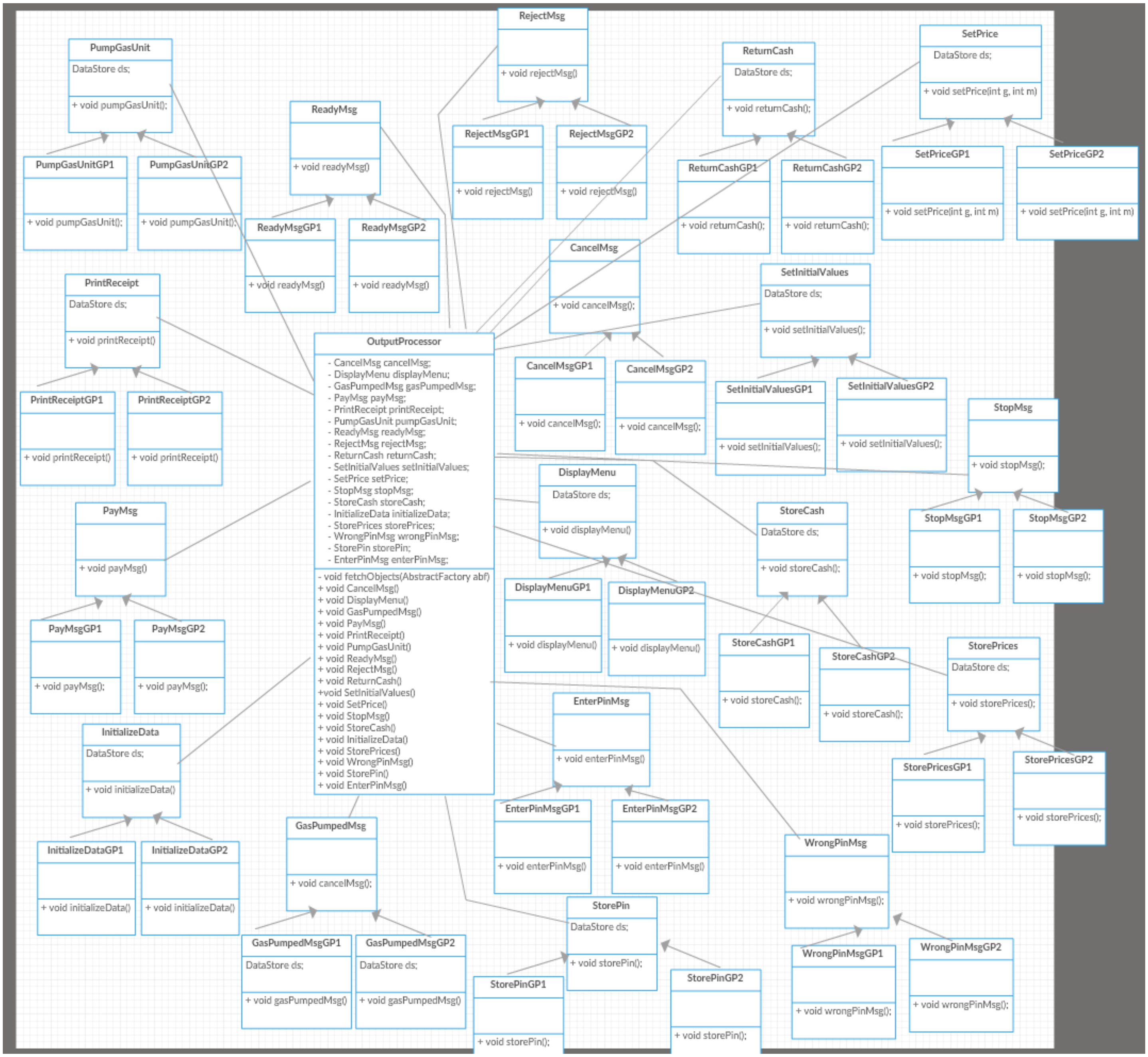


Following is the state diagram with output processor and Gas Pump class:


<<Java Class>>
OutputProcessor
com.gaspumpproj.model.outputprocessor
cancelMsg: CancelMsg
displayMenu: DisplayMenu
gasPumpedMsg: GasPumpedMsg
payMsg: PayMsg
printReceipt: PrintReceipt
pumpGasUnit: PumpGasUnit
readyMsg: ReadyMsg
rejectMsg: RejectMsg
returnCash: ReturnCash
setInitialValues: SetInitialValues
setPrice: SetPrice
stopMsg: StopMsg
storeCash: StoreCash
initializeData: InitializeData
storePrices: StorePrices
wrongPinMsg: WrongPinMsg
storePin: StorePin
enterPinMsg: EnterPinMsg
OutputProcessor(AbstractFactory)
fetchObjects(AbstractFactory):void
CancelMsg():void
DisplayMenu():void
GasPumpedMsg():void
PayMsg():void
PrintReceipt():void
PumpGasUnit():void
ReadyMsg():void
RejectMsg():void
ReturnCash():void
SetInitialValues():void
SetPrice(int,int):void
StopMsg():void
StoreCash():void
InitializeData():void
StorePrices():void
WrongPinMsg():void
StorePin():void
EnterPinMsg():void
<<Java Class>>
StateVMModel
com.gaspumpproj.model.state
M: int
StateVMModel(OutputProcessor)
createStateObjects():void
activate():void
start():void
payType(int):void
reject():void
cancel():void
approved():void
selectGas(int):void
startPump():void
pump():void
stopPump():void
receipt():void
noReceipt():void
correctPin():void
incorrectPin():void
continueTr():void
getOutputProcessor():OutputProcessor
setOutputProcessor(OutputProcessor):void
-op
0..1
#op
0..1
-m
0..1
<<Java Class>>
GasPump
com.gaspumpproj.gaspumps
ds: DataStore
GasPump(AbstractFactory)
<<Java Class>>
GasPump_1
com.gaspumpproj.gaspumps
GasPump_1(AbstractFactory)
Activate(float,float):void
Start():void
PayCredit():void
Reject():void
PayDebit(String):void
Pin(String):void
Cancel():void
Approved():void
Diesel():void
Regular():void
StartPump():void
PumpGallon():void
StopPump():void
FullTank():void
<<Java Class>>
GasPump_2
com.gaspumpproj.gaspumps
GasPump_2(AbstractFactory)
Activate(float,float,float):void
PayCash(int):void
PayCredit():void
Approved():void
Reject():void
Premium():void
Cancel():void
Regular():void
Super():void
PumpLiter():void
NoReceipt():void
Stop():void
StartPump():void
Receipt():void
#mod
0..1
#st
0..
#LS
0..*
<<Java Class>>
State
com.gaspumpproj.model.state
State()
activate():void
start():void
payType(int):void
approved():void
reject():void
selectGas(int):void
cancel():void
startPump():void
pump():void
stopPump():void
receipt():void
noReceipt():void
correctPin():void
incorrectPin():void
continueTr():void
showAccessDeniedMsg():void
<<Java Class>>
S0
com.gaspumpproj.model.state
S0(StateVMModel)
start():void
<<Java Class>>
S1
com.gaspumpproj.model.state
S1(StateVMModel)
payType(int):void
<<Java Class>>
S2
com.gaspumpproj.model.state
S2(StateVMModel)
approved():void
reject():void
<<Java Class>>
S3
com.gaspumpproj.model.state
S3(StateVMModel)
selectGas(int):void
cancel():void
continueTr():void
<<Java Class>>
S4
com.gaspumpproj.model.state
S4(StateVMModel)
startPump():void
<<Java Class>>
S5
com.gaspumpproj.model.state
S5(StateVMModel)
pump():void
stopPump():void
<<Java Class>>
S6
com.gaspumpproj.model.state
S6(StateVMModel)
receipt():void
noReceipt():void
<<Java Class>>
InitialState
com.gaspumpproj.model.state
InitialState(StateVMModel)
activate():void
<<Java Class>>
S8
com.gaspumpproj.model.state
S8(StateVMModel)
correctPin():void
incorrectPin():void

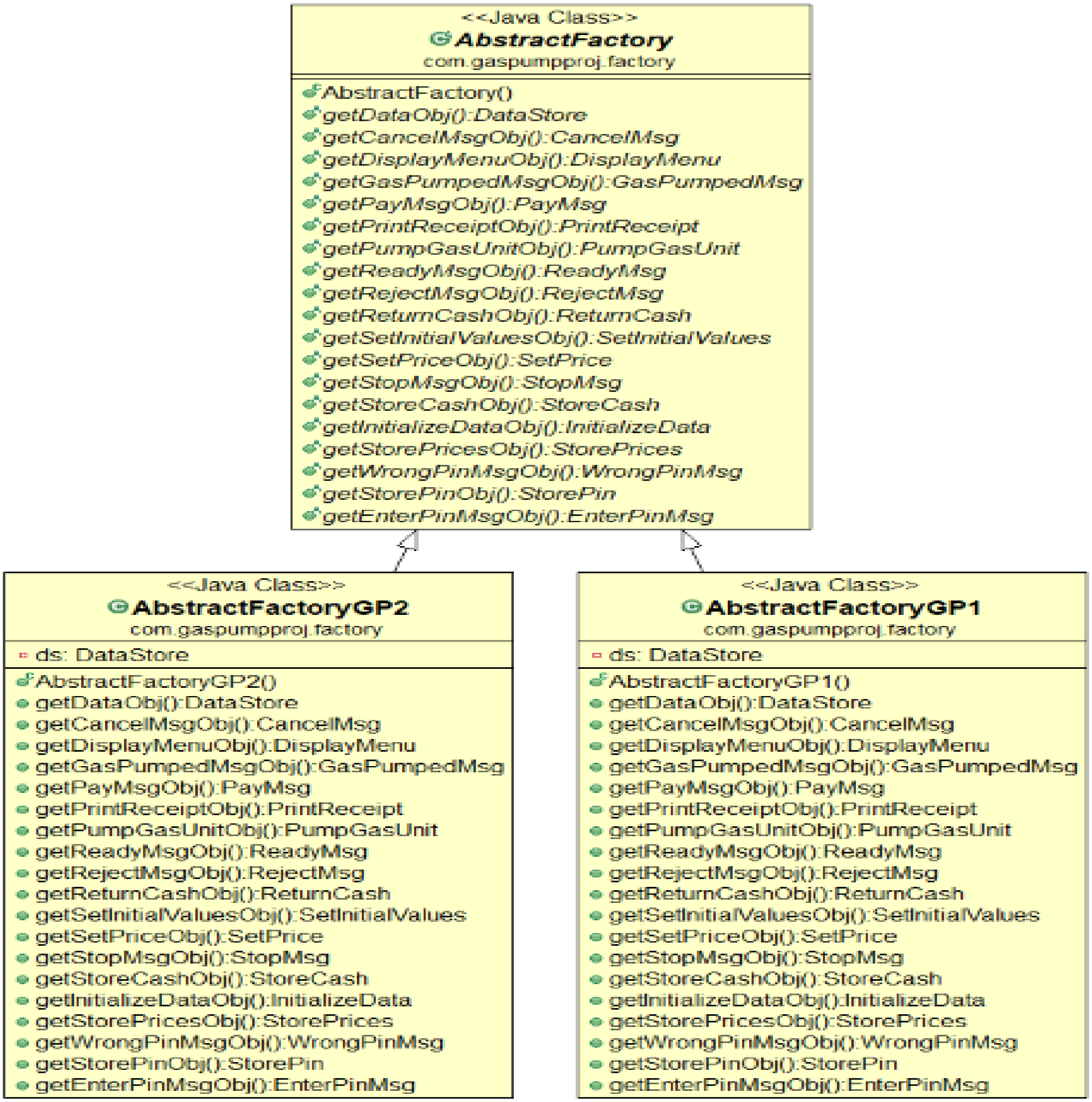
ii. Strategy pattern:

This pattern is Implemented in package "com.gaspumpproj.strategy.action" of the GasPumpProj. OutputProcessor is the client class that needs to be initialized with proper action strategies. Each package contains 3 classes. One super and two subclasses which corresponds to each action. One class is the abstract strategy that groups different implementations of a specific strategy. The other 2 classes are GasPump-1 and GasPump-2 specific implementation for the meta-actions of the model.

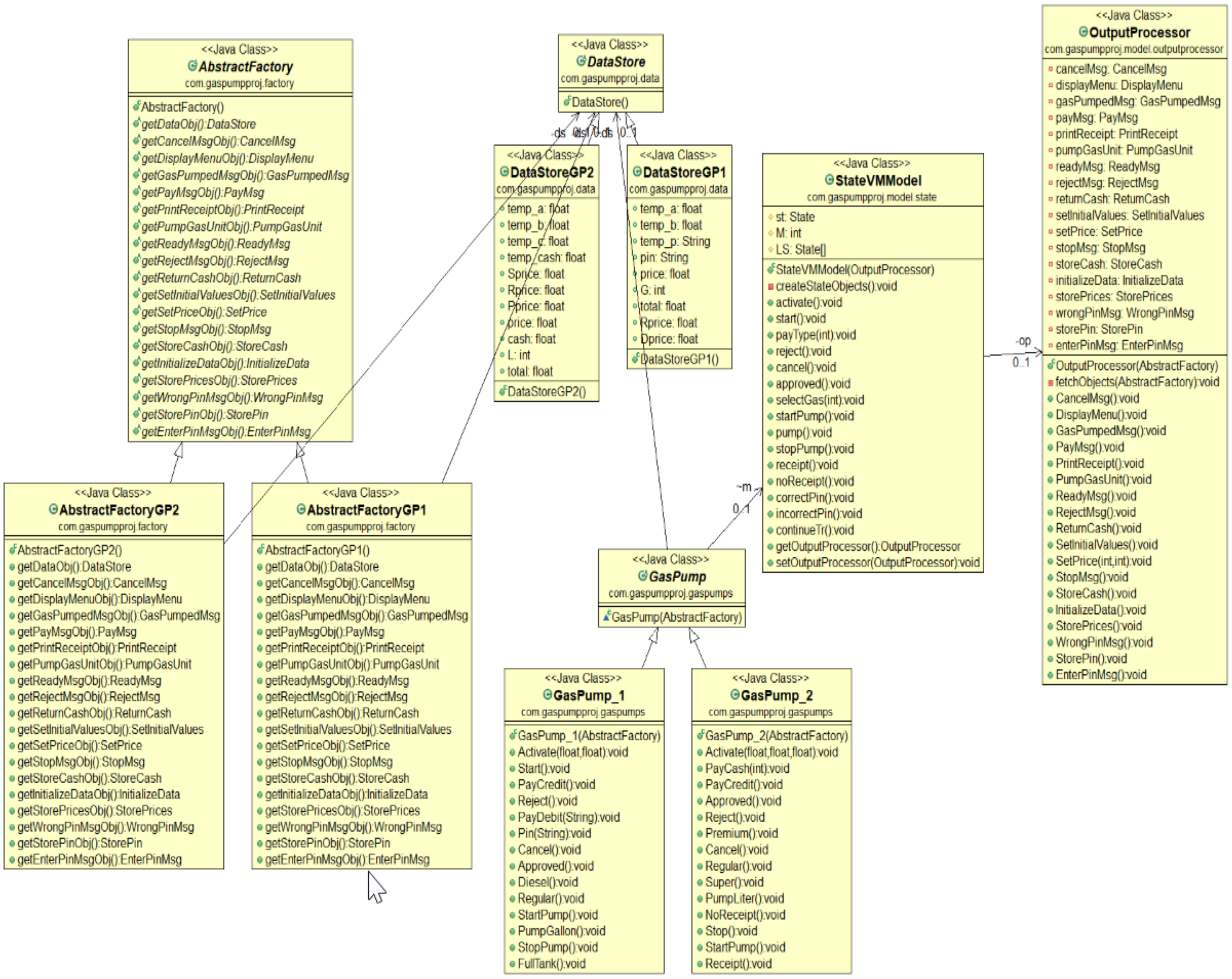


iii. Abstract factory pattern

This pattern is implemented in package "com.gaspumpproj.factory" of the GasPumpProj. AbstractFactory is the abstract superclass. AbstractFactoryGP1 is the factory that returns the necessary driver objects for GasPump1. AbstractFactoryGP2 is the factory that returns the necessary driver objects for GasPump2.


<<Java Class>>
AbstractFactory
com.gaspumpproj.factory
AbstractFactory()
getDataObj():DataStore
getCancelMsgObj():CancelMsg
getDisplayMenuObj():DisplayMenu
getGasPumpedMsgObj():GasPumpedMsg
getPayMsgObj():PayMsg
getPrintReceiptObj():PrintReceipt
getPumpGasUnitObj():PumpGasUnit
getReadyMsgObj():ReadyMsg
getRejectMsgObj():RejectMsg
getReturnCashObj():ReturnCash
getSetInitialValuesObj():SetInitialValues
getSetPriceObj():SetPrice
getStopMsgObj():StopMsg
getStoreCashObj():StoreCash
getInitializeDataObj():InitializeData
getStorePricesObj():StorePrices
getWrongPinMsgObj():WrongPinMsg
getStorePinObj():StorePin
getEnterPinMsgObj():EnterPinMsg
<<Java Class>>
AbstractFactoryGP2
com.gaspumpproj.factory
ds: DataStore
AbstractFactoryGP2()
getDataObj():DataStore
getCancelMsgObj():CancelMsg
getDisplayMenuObj():DisplayMenu
getGasPumpedMsgObj():GasPumpedMsg
getPayMsgObj():PayMsg
getPrintReceiptObj():PrintReceipt
getPumpGasUnitObj():PumpGasUnit
getReadyMsgObj():ReadyMsg
getRejectMsgObj():RejectMsg
getReturnCashObj():ReturnCash
getSetInitialValuesObj():SetInitialValues
getSetPriceObj():SetPrice
getStopMsgObj():StopMsg
getStoreCashObj():StoreCash
getInitializeDataObj():InitializeData
getStorePricesObj():StorePrices
getWrongPinMsgObj():WrongPinMsg
getStorePinObj():StorePin
getEnterPinMsgObj():EnterPinMsg
<<Java Class>>
AbstractFactoryGP1
com.gaspumpproj.factory
ds: DataStore
AbstractFactoryGP1()
getDataObj():DataStore
getCancelMsgObj():CancelMsg
getDisplayMenuObj():DisplayMenu
getGasPumpedMsgObj():GasPumpedMsg
getPayMsgObj():PayMsg
getPrintReceiptObj():PrintReceipt
getPumpGasUnitObj():PumpGasUnit
getReadyMsgObj():ReadyMsg
getRejectMsgObj():RejectMsg
getReturnCashObj():ReturnCash
getSetInitialValuesObj():SetInitialValues
getSetPriceObj():SetPrice
getStopMsgObj():StopMsg
getStoreCashObj():StoreCash
getInitializeDataObj():InitializeData
getStorePricesObj():StorePrices
getWrongPinMsgObj():WrongPinMsg
getStorePinObj():StorePin
getEnterPinMsgObj():EnterPinMsg

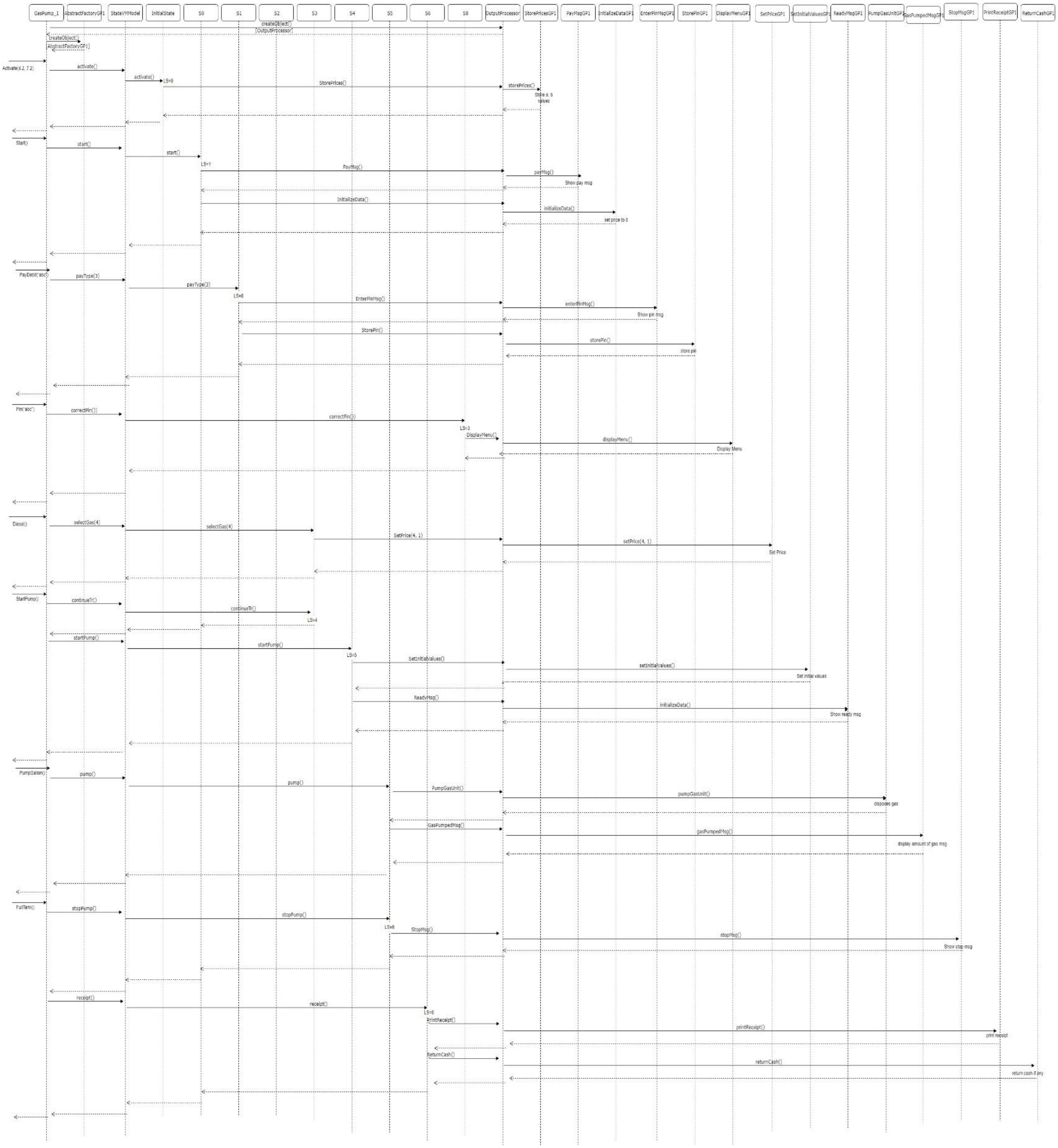
Following is the factory diagram with other classes:


<<Java Class>>
AbstractFactory
com.gaspumpproj.factory
AbstractFactory()
getDataObj():DataStore
getCancelMsgObj():CancelMsg
getDisplayMenuObj():DisplayMenu
getGasPumpedMsgObj():GasPumpedMsg
getPayMsgObj():PayMsg
getPrintReceiptObj():PrintReceipt
getPumpGasUnitObj():PumpGasUnit
getReadyMsgObj():ReadyMsg
getRejectMsgObj():RejectMsg
getReturnCashObj():ReturnCash
getSetInitialValuesObj():SetInitialValues
getSetPriceObj():SetPrice
getStopMsgObj():StopMsg
getStoreCashObj():StoreCash
getInitializeDataObj():InitializeData
getStorePricesObj():StorePrices
getWrongPinMsgObj():WrongPinMsg
getStorePinObj():StorePin
getEnterPinMsgObj():EnterPinMsg
<<Java Class>>
DataStore
com.gaspumpproj.data
DataStore()
-ds
0..1
-ds
0..1
~ds
0..1
<<Java Class>>
DataStoreGP2
com.gaspumpproj.data
temp_a: float
temp_b: float
temp_c: float
temp_cash: float
Sprice: float
Rprice: float
Pprice: float
price: float
cash: float
L: int
total: float
DataStoreGP2()
<<Java Class>>
DataStoreGP1
com.gaspumpproj.data
temp_a: float
temp_b: float
temp_p: String
pin: String
price: float
G: int
total: float
Rprice: float
Dprice: float
DataStoreGP1()
<<Java Class>>
StateVMModel
com.gaspumpproj.model.state
st: State
M: int
LS: State[]
StateVMModel(OutputProcessor)
createStateObjects():void
activate():void
start():void
payType(int):void
reject():void
cancel():void
approved():void
selectGas(int):void
startPump():void
pump():void
stopPump():void
receipt():void
noReceipt():void
correctPin():void
incorrectPin():void
continueTr():void
getOutputProcessor():OutputProcessor
setOutputProcessor(OutputProcessor):void
-op
0..1
<<Java Class>>
OutputProcessor
com.gaspumpproj.model.outputprocessor
cancelMsg: CancelMsg
displayMenu: DisplayMenu
gasPumpedMsg: GasPumpedMsg
payMsg: PayMsg
printReceipt: PrintReceipt
pumpGasUnit: PumpGasUnit
readyMsg: ReadyMsg
rejectMsg: RejectMsg
returnCash: ReturnCash
setInitialValues: SetInitialValues
setPrice: SetPrice
stopMsg: StopMsg
storeCash: StoreCash
initializeData: InitializeData
storePrices: StorePrices
wrongPinMsg: WrongPinMsg
storePin: StorePin
enterPinMsg: EnterPinMsg
OutputProcessor(AbstractFactory)
fetchObjects(AbstractFactory):void
CancelMsg():void
DisplayMenu():void
GasPumpedMsg():void
PayMsg():void
PrintReceipt():void
PumpGasUnit():void
ReadyMsg():void
RejectMsg():void
ReturnCash():void
SetInitialValues():void
SetPrice(int,int):void
StopMsg():void
StoreCash():void
InitializeData():void
StorePrices():void
WrongPinMsg():void
StorePin():void
EnterPinMsg():void
<<Java Class>>
AbstractFactoryGP2
com.gaspumpproj.factory
AbstractFactoryGP2()
getDataObj():DataStore
getCancelMsgObj():CancelMsg
getDisplayMenuObj():DisplayMenu
getGasPumpedMsgObj():GasPumpedMsg
getPayMsgObj():PayMsg
getPrintReceiptObj():PrintReceipt
getPumpGasUnitObj():PumpGasUnit
getReadyMsgObj():ReadyMsg
getRejectMsgObj():RejectMsg
getReturnCashObj():ReturnCash
getSetInitialValuesObj():SetInitialValues
getSetPriceObj():SetPrice
getStopMsgObj():StopMsg
getStoreCashObj():StoreCash
getInitializeDataObj():InitializeData
getStorePricesObj():StorePrices
getWrongPinMsgObj():WrongPinMsg
getStorePinObj():StorePin
getEnterPinMsgObj():EnterPinMsg
<<Java Class>>
AbstractFactoryGP1
com.gaspumpproj.factory
AbstractFactoryGP1()
getDataObj():DataStore
getCancelMsgObj():CancelMsg
getDisplayMenuObj():DisplayMenu
getGasPumpedMsgObj():GasPumpedMsg
getPayMsgObj():PayMsg
getPrintReceiptObj():PrintReceipt
getPumpGasUnitObj():PumpGasUnit
getReadyMsgObj():ReadyMsg
getRejectMsgObj():RejectMsg
getReturnCashObj():ReturnCash
getSetInitialValuesObj():SetInitialValues
getSetPriceObj():SetPrice
getStopMsgObj():StopMsg
getStoreCashObj():StoreCash
getInitializeDataObj():InitializeData
getStorePricesObj():StorePrices
getWrongPinMsgObj():WrongPinMsg
getStorePinObj():StorePin
getEnterPinMsgObj():EnterPinMsg
~m
0..1
<<Java Class>>
GasPump
com.gaspumpproj.gaspumps
GasPump(AbstractFactory)
<<Java Class>>
GasPump_1
com.gaspumpproj.gaspumps
GasPump_1(AbstractFactory)
Activate(float,float):void
Start():void
PayCredit():void
Reject():void
PayDebit(String):void
Pin(String):void
Cancel():void
Approved():void
Diesel():void
Regular():void
StartPump():void
PumpGallon():void
StopPump():void
FullTank():void
<<Java Class>>
GasPump_2
com.gaspumpproj.gaspumps
GasPump_2(AbstractFactory)
Activate(float,float,float):void
PayCash(int):void
PayCredit():void
Approved():void
Reject():void
Premium():void
Cancel():void
Regular():void
Super():void
PumpLiter():void
NoReceipt():void
Stop():void
StartPump():void
Receipt():void

SEQUENCE DIAGRAMS

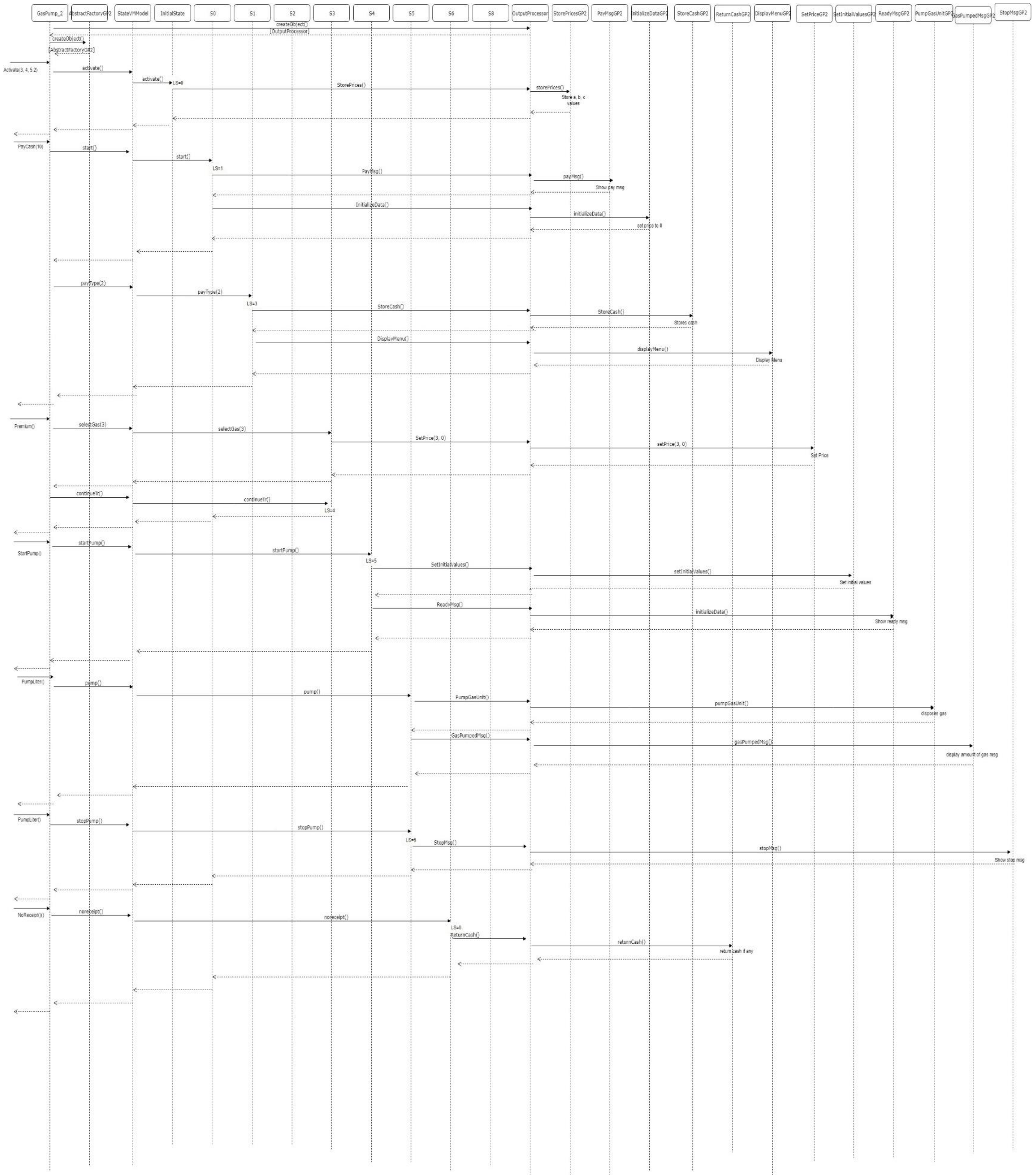
a. Scenario-I should show how one gallon of Diesel gas is disposed in GasPump-1, i.e., the following sequence of operations is issued: Activate(4.2, 7.2), Start(), PayDebit("abc"), Pin("abc"), Diesel(), StartPump(), PumpGallon(), FullTank()

Note: Please check Diagrams\seq_diag_prob1 folder for the diagram in HTML and zoomed formats.



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, 5.2), PayCash(10), Premium(), StartPump(), PumpLiter(), PumpLiter(), NoReceipt()

Note: Please check Diagrams\seq_diag_prob2 folder for the diagram in HTML and zoomed formats.



CODE, PURPOSE OF THE CLASS AND THE RESPONSIBILITY OF OPERATION

Note: Please check Code folder for the project code.

Classes, methods with description is documented in code itself.

```
package com.gaspumpproj.data;

/**
 * This is the superclass of data stores of gas pump 1 and gas pump 2. This is a abstract class.
 *
 * @author borde
 */

public abstract class DataStore {

}

package com.gaspumpproj.data;

/**
 * DataStoreGP1 is class related to data store values of Gas Pump 1. It is shared by system components.
 * Here, we have avoided to use getter setters for simplicity.
 *
 * @author borde
 *
 */
public class DataStoreGP1 extends DataStore {
    // Temporary
    public float temp_a;
    public float temp_b;
    public String temp_p;

    // Non temporary
        public String pin;
    public float price;
    public int G;
    public float total;
    public float Rprice;
    public float Dprice;


}

package com.gaspumpproj.data;

/**
 * DataStoreGP2 is class related to data store values of Gas Pump 2. It is shared by system components.
 * Here, we have avoided to use getter setters for simplicity.
 *
 * @author borde
 *
 */
public class DataStoreGP2 extends DataStore {
    // Temporary
    public float temp_a;
    public float temp_b;
    public float temp_c;
    public float temp_cash;

    // Non temporary
    public float Sprice;
    public float Rprice;
    public float Pprice;
    public float price;
    public float cash;
    public int L;
    public float total;

}

package com.gaspumpproj.driver;

import java.util.Scanner;

import com.gaspumpproj.factory.AbstractFactory;
import com.gaspumpproj.factory.AbstractFactoryGP1;
import com.gaspumpproj.factory.AbstractFactoryGP2;
import com.gaspumpproj.gaspumps.GasPump_1;
import com.gaspumpproj.gaspumps.GasPump_2;

/**
 * This is the driver class which where user provides the input information as well as data.
 *
 * @author borde
 *
 */
public class Driver {
        /*
         * Process on gas pump 1
         */
        static void processGaspump_1(Scanner reader) {
                AbstractFactory abf = new AbstractFactoryGP1();
                GasPump_1 gp1 = new GasPump_1(abf);
                float a, b;
                String pin, x;
                int c1;
                char ch;
                //Menu
                System.out.println("\t\t=====================================================");
                System.out.println("\t\t\t\t\tGasPump-1");
                System.out.println("\t\tMENU of Operations");
                System.out.println("\t\t0. Activate(float, float)");
                System.out.println("\t\t1. Start()");
                System.out.println("\t\t2. PayCredit()");
                System.out.println("\t\t3. Reject()");
                System.out.println("\t\t4. PayDebit(String)");
                System.out.println("\t\t5. Pin(String)");
                System.out.println("\t\t6. Cancel()");
                System.out.println("\t\t7. Approved()");
                System.out.println("\t\t8. Diesel()");
                System.out.println("\t\t9. Regular()");
                System.out.println("\t\ta. StartPump()");
                System.out.println("\t\tb. PumpGallon()");
                System.out.println("\t\tc. StopPump()");
                System.out.println("\t\td. FullTank()");
                System.out.println("\t\tPlease make a note of these operations");
                System.out.println("\t\tGasPump-1 Execution");
                System.out.println("\t\t=====================================================");

                ch = '1';
                while (ch !='q') {
                        System.out.println("-----------------------------------------------------------------------------
-----");
                        System.out.println(" Select Operation: ");
                        System.out.println("0-Activate, 1-Start, 2-PayCredit, 3-Reject, 4-PayDebit, 5-Pin, 6-Cancel,");
                        System.out.println("7-Approved, 8-Diesel, 9-Regular, a-StartPump, b-PumpGallon, c-StopPump, d-
FullTank, q-quit");
                        System.out.println("Enter your choice : ");
                        ch =(char) reader.next().charAt(0);
                        switch (ch) {
                                case '0': { //Activate()
                                                        System.out.println(" Operation: Activate()");
                                                        System.out.println(" Enter value a :");
                                                        a = reader.nextFloat();
                                                        System.out.println(" Enter value b :");
                                                        b = reader.nextFloat();
                                                        gp1.Activate(a,b);
                                                        break;
                                                }

                                case '1': { //Start
                                                        System.out.println(" Operation: Start()");
                                                        gp1.Start();
                                                        break;
                                                }
                                case '2': { //PayCredit
                                                        System.out.println(" Operation: PayCredit()");
                                                        gp1.PayCredit();
                                                        break;
                                                }
                                case '3': { //Reject
```

```
                                                        System.out.println(" Operation: Reject()");
                                                        gp1.Reject();
                                                        break;
                                                }
                                case '4': {//PayDebit
                                                        System.out.println(" Operation: PayDebit()");
                                                        System.out.println(" Enter pin string: ");
                                                        pin = reader.next();
                                                        gp1.PayDebit(pin);
                                                        break;
                                                }
                                case '5': {//Pin
                                                        System.out.println("Operation: Pin()");
                                                        System.out.println(" Enter pin value: ");
                                                        x = reader.next();
                                                        gp1.Pin(x);
                                                        break;
                                                }
                                case '6': {//Cancel
                                                        System.out.println(" Operation: Cancel()");
                                                        gp1.Cancel();
                                                        break;
                                                }
                                case '7': { //Approved
                                                        System.out.println(" Operation: Approved()");
                                                        gp1.Approved();
                                                        break;
                                                }
                                case '8': { //Diesel
                                                        System.out.println("Operation: Diesel()");
                                                        gp1.Diesel();
                                                        break;
                                                }
                                case '9': {//Regular
                                                        System.out.println(" Operation: Regular()");
                                                        gp1.Regular();
                                                        break;
                                                }
                                case 'a': {
                                                        //StartPump
                                                        System.out.println(" Operation: StartPump()");
                                                        gp1.StartPump();
                                                        break;
                                                }
                                case 'b': { //PumpGallon
                                                        System.out.println(" Operation: PumpGallon()");
                                                        gp1.PumpGallon();
                                                        break;
                                                }
                                case 'c': { //StopPump
                                                        System.out.println(" Operation: StopPump()");
                                                        gp1.StopPump();
                                                        break;
                                                }
                                case 'd': { //FullTank
                                                        System.out.println(" Operation: FullTank()");
                                                        gp1.FullTank();
                                                        break;
                                }
                                case 'q': break;
                        }// endswitch
                        System.out.println("-----------------------------------------------------------------------------
-----");
                } //endwhile
        }

        /*
         * Process on gas pump 2
         */
        static void processGaspump_2(Scanner reader) {
                AbstractFactory abf = new AbstractFactoryGP2();
                GasPump_2 gp2 = new GasPump_2(abf);
                float a, b, c;
                int c1;
                char ch;
                //Menu
                System.out.println("\t\t=====================================================");
                System.out.println("\t\t\t\t\tGasPump-2");
                System.out.println("\t\tMENU of Operations");
                System.out.println("\t\t0. Activate(float, float, float)");
                System.out.println("\t\tx. Start()");
                System.out.println("\t\t1. PayCash(int)");
                System.out.println("\t\t2. PayCredit()");
                System.out.println("\t\t3. Approved()");
                System.out.println("\t\t4. Reject()");
                System.out.println("\t\t5. Cancel()");
                System.out.println("\t\t6. Premium()");
                System.out.println("\t\t7. Regular()");
                System.out.println("\t\t8. Super()");
                System.out.println("\t\t9. StartPump()");
                System.out.println("\t\ta. PumpLiter()");
                System.out.println("\t\tb. Stop()");
                System.out.println("\t\tc. Receipt()");
                System.out.println("\t\td. NoReceipt()");
                System.out.println("\t\tq. Quit the program");
                System.out.println("\t\tPlease make a note of these operations");
                System.out.println("\t\tGasPump-2 Execution");
                System.out.println("\t\t=====================================================");

                ch = '1';
                while (ch !='q') {
                        System.out.println("------------------------------------------------------------------------------
-----");
                        System.out.println(" Select Operation: ");
                        System.out.println("0-Activate, 1-PayCash, 2-PayCredit, 3-Approved, 4-Reject, 5-Cancel, 6-
Premium,");
                        System.out.println("7-Regular, 8-Super, 9-StartPump, a-PumpLiter, b-Stop, c-Receipt, d-NoReceipt, 
q-quit");
                        System.out.println("Enter your choice : ");
                        ch =(char) reader.next().charAt(0);
                        switch (ch) {
                                case '0': { //Activate()
                                                        System.out.println(" Operation:");
                                                        System.out.println(" Enter value a :");
                                                        a = reader.nextFloat();
                                                        System.out.println(" Enter value b :");
                                                        b = reader.nextFloat();
                                                        System.out.println(" Enter value c :");
                                                        c = reader.nextFloat();
                                                        gp2.Activate(a,b,c);
                                                        break;
                                                }
                                case '1': { //PayCash
                                                        System.out.println(" Operation: PayCash(int c)");
                                                        System.out.println(" Enter value of the parameter c:");
                                                        c1 = reader.nextInt();
                                                        gp2.PayCash(c1);
                                                        break;
                                                }
                                case '2': { //PayCredit
                                                        System.out.println(" Operation: PayCredit()");
                                                        gp2.PayCredit();
                                                        break;
                                                }
                                case '3': { //Approved
                                                        System.out.println(" Operation: Approved()");
                                                        gp2.Approved();
                                                        break;
                                                }
                                case '4': {//Reject
                                                        System.out.println(" Operation: Reject()");
                                                        gp2.Reject();
                                                        break;
                                                }
                                case '5': {//Cancel
                                                        System.out.println("Operation: Cancel()");
                                                        gp2.Cancel();
                                                        break;
                                                }
                                case '6': {//Premium
                                                        System.out.println(" Operation: Premium()");
                                                        gp2.Premium();
```

```java
                            break;
                        }
                case '7': { //Regular
                            System.out.println(" Operation: Regular()");
                            gp2.Regular();
                            break;
                        }
                case '8': { //Super
                            System.out.println("Operation: Super()");
                            gp2.Super();
                            break;
                        }
                case '9': {//StartPump
                            System.out.println(" Operation: StartPump()");
                            gp2.StartPump();
                            break;
                        }
                case 'a': {
                            //PumpLiter
                            System.out.println(" Operation: PumpLiter()");
                            gp2.PumpLiter();
                            break;
                        }
                case 'b': { //Stop
                            System.out.println(" Operation: Stop()");
                            gp2.Stop();
                            break;
                        }
                case 'c': { //Receipt
                            System.out.println(" Operation: Receipt()");
                            gp2.Receipt();
                            break;
                        }
                case 'd': { //NoReceipt
                            System.out.println(" Operation: NoReceipt()");
                            gp2.NoReceipt();
                            break;
                }
                case 'q': break;
            }// endswitch
            System.out.println("------------------------------------------------------------------------------
-----");
        } //endwhile
    }

    public static void main(String[] args){
        Scanner reader=new Scanner(System.in);
        char ch = '1';
        // Selection of gas pump type:
        System.out.println("\t\t=====================================================");
        System.out.println("\t\t\t\t Select Gaspump Type: ");
        System.out.println("\t\t1. Gas Pump - 1 ");
        System.out.println("\t\t2. Gas Pump - 2 ");
        System.out.println("Enter Choice : ");
        ch =(char) reader.next().charAt(0);

        System.out.println("\t\t=====================================================");

        switch (ch) {
            case '1':
                processGaspump_1(reader);
                break;
            case '2':
                processGaspump_2(reader);
                break;
            default:
                System.out.println(" Wrong selection. Exiting application.");
        }


    }// main
}
package com.gaspumpproj.factory;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.strategy.action.cancelmsg.CancelMsg;
import com.gaspumpproj.strategy.action.displaymenu.DisplayMenu;
import com.gaspumpproj.strategy.action.enterpinmsg.EnterPinMsg;
import com.gaspumpproj.strategy.action.gaspumpedmsg.GasPumpedMsg;
import com.gaspumpproj.strategy.action.initializedata.InitializeData;
import com.gaspumpproj.strategy.action.paymsg.PayMsg;
import com.gaspumpproj.strategy.action.printreceipt.PrintReceipt;
import com.gaspumpproj.strategy.action.pumpgasunit.PumpGasUnit;
import com.gaspumpproj.strategy.action.readymsg.ReadyMsg;
import com.gaspumpproj.strategy.action.rejectmsg.RejectMsg;
import com.gaspumpproj.strategy.action.returncash.ReturnCash;
import com.gaspumpproj.strategy.action.setinitialvalues.SetInitialValues;
import com.gaspumpproj.strategy.action.setprice.SetPrice;
import com.gaspumpproj.strategy.action.stopmsg.StopMsg;
import com.gaspumpproj.strategy.action.storecash.StoreCash;
import com.gaspumpproj.strategy.action.storepin.StorePin;
import com.gaspumpproj.strategy.action.storeprices.StorePrices;
import com.gaspumpproj.strategy.action.wrongpinmsg.WrongPinMsg;

/**
 * Here, i used Abstract Factory design pattern.
 * This is the abstract class which is the super class of two abstract factories AbstractFactoryGP1 and
AbstractFactoryGP2.
 * It initializes with the proper action for the specific gas pump that is desired.
 * This is done thorough the Abstract Factory design pattern.

 * @author borde
 */
public abstract class AbstractFactory {

    public abstract DataStore getDataObj();

    public abstract CancelMsg getCancelMsgObj();

    public abstract DisplayMenu getDisplayMenuObj();

    public abstract GasPumpedMsg getGasPumpedMsgObj();

    public abstract PayMsg getPayMsgObj();

    public abstract PrintReceipt getPrintReceiptObj();

    public abstract PumpGasUnit getPumpGasUnitObj();

    public abstract ReadyMsg getReadyMsgObj();

    public abstract RejectMsg getRejectMsgObj();

    public abstract ReturnCash getReturnCashObj();

    public abstract SetInitialValues getSetInitialValuesObj();

    public abstract SetPrice getSetPriceObj();

    public abstract StopMsg getStopMsgObj();

    public abstract StoreCash getStoreCashObj();

    public abstract InitializeData getInitializeDataObj();

    public abstract StorePrices getStorePricesObj();

    public abstract WrongPinMsg getWrongPinMsgObj();

    public abstract StorePin getStorePinObj();

    public abstract EnterPinMsg getEnterPinMsgObj();



}

package com.gaspumpproj.factory;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;
import com.gaspumpproj.strategy.action.cancelmsg.CancelMsg;
import com.gaspumpproj.strategy.action.cancelmsg.CancelMsgGP1;
```

```java
import com.gaspumpproj.strategy.action.displaymenu.DisplayMenu;
import com.gaspumpproj.strategy.action.displaymenu.DisplayMenuGP1;
import com.gaspumpproj.strategy.action.enterpinmsg.EnterPinMsg;
import com.gaspumpproj.strategy.action.enterpinmsg.EnterPinMsgGP1;
import com.gaspumpproj.strategy.action.gaspumpedmsg.GasPumpedMsg;
import com.gaspumpproj.strategy.action.gaspumpedmsg.GasPumpedMsgGP1;
import com.gaspumpproj.strategy.action.initializedata.InitializeData;
import com.gaspumpproj.strategy.action.initializedata.InitializeDataGP1;
import com.gaspumpproj.strategy.action.paymsg.PayMsg;
import com.gaspumpproj.strategy.action.paymsg.PayMsgGP1;
import com.gaspumpproj.strategy.action.printreceipt.PrintReceipt;
import com.gaspumpproj.strategy.action.printreceipt.PrintReceiptGP1;
import com.gaspumpproj.strategy.action.pumpgasunit.PumpGasUnit;
import com.gaspumpproj.strategy.action.pumpgasunit.PumpGasUnitGP1;
import com.gaspumpproj.strategy.action.readymsg.ReadyMsg;
import com.gaspumpproj.strategy.action.readymsg.ReadyMsgGP1;
import com.gaspumpproj.strategy.action.rejectmsg.RejectMsg;
import com.gaspumpproj.strategy.action.rejectmsg.RejectMsgGP1;
import com.gaspumpproj.strategy.action.returncash.ReturnCash;
import com.gaspumpproj.strategy.action.returncash.ReturnCashGP1;
import com.gaspumpproj.strategy.action.setinitialvalues.SetInitialValues;
import com.gaspumpproj.strategy.action.setinitialvalues.SetInitialValuesGP1;
import com.gaspumpproj.strategy.action.setprice.SetPrice;
import com.gaspumpproj.strategy.action.setprice.SetPriceGP1;
import com.gaspumpproj.strategy.action.stopmsg.StopMsg;
import com.gaspumpproj.strategy.action.stopmsg.StopMsgGP1;
import com.gaspumpproj.strategy.action.storecash.StoreCash;
import com.gaspumpproj.strategy.action.storecash.StoreCashGP1;
import com.gaspumpproj.strategy.action.storepin.StorePin;
import com.gaspumpproj.strategy.action.storepin.StorePinGP1;
import com.gaspumpproj.strategy.action.storeprices.StorePrices;
import com.gaspumpproj.strategy.action.storeprices.StorePricesGP1;
import com.gaspumpproj.strategy.action.wrongpinmsg.WrongPinMsg;
import com.gaspumpproj.strategy.action.wrongpinmsg.WrongPinMsgGP1;

/**
 * This class is abstract factory which is related to Gas Pump 1.
 * Here, we instantiates the action strategies with the shared data structure for Gas Pump 1.
 * 
 * @author borde
 *
 */
public class AbstractFactoryGP1 extends AbstractFactory {
    private DataStore ds;
    
    public AbstractFactoryGP1() {
        // Create new data store which is related to GasPump1.
        this.ds  = new DataStoreGP1();
    }
    
    @Override
    public DataStore getDataObj() {
        // Returns the data store appropriate for GasPump1
        return this.ds;
    }

    @Override
    public CancelMsg getCancelMsgObj() {
        // Returns the CancelMsg class for GasPump1.
        return new CancelMsgGP1();
    }

    @Override
    public DisplayMenu getDisplayMenuObj() {
        // Returns the DisplayMenu class that performs the appropriate action for displaying the menu prompt for GasPump1.
        return new DisplayMenuGP1(this.ds);
    }

    @Override
    public GasPumpedMsg getGasPumpedMsgObj() {
        // Returns the GasPumpedMsg class for GasPump1.
        return new GasPumpedMsgGP1(this.ds);
    }

    @Override
    public PayMsg getPayMsgObj() {
        // Returns the Payment prompt message action appropriate for GasPump1
        return new PayMsgGP1();
    }

    @Override
    public PrintReceipt getPrintReceiptObj() {
        // Returns the PrintReceipt class which is responsible for printing the GasPump1
        return new PrintReceiptGP1(this.ds);
    }

    @Override
    public PumpGasUnit getPumpGasUnitObj() {
        // Returns the PumpGasUnit class which is responsible to "pumping" a unit of gas for GasPump1
        return new PumpGasUnitGP1(this.ds);
    }

    @Override
    public ReadyMsg getReadyMsgObj() {
        //  Returns the ReadyMsg class which is responsible for prompting the user to start pumping gas on GasPump1.
        return new ReadyMsgGP1();
    }

    @Override
    public RejectMsg getRejectMsgObj() {
        // Returns the RejectMsg class that displays the appropriate credit card rejection message for GasPump1
        return new RejectMsgGP1();
    }

    @Override
    public ReturnCash getReturnCashObj() {
        // Returns the ReturnCash action object appropriate for GasPump1
        return new ReturnCashGP1();
    }

    @Override
    public SetInitialValues getSetInitialValuesObj() {
        // Returns the SetInitialValues class which provides the appropriate action for initializing the shared data 
structure with GasPump1
        return new SetInitialValuesGP1(this.ds);
    }

    @Override
    public SetPrice getSetPriceObj() {
        //  Returns the SetPrice class that is responsible for setting the price of gasoline according to specifications 
of GasPump1 requirements.
        return new SetPriceGP1(this.ds);
    }

    @Override
    public StopMsg getStopMsgObj() {
        // Returns the StopMsg class which provides the appropriate action for displaying a message that the gas pump has 
been stopped according to GasPump1
        return new StopMsgGP1();
    }

    @Override
    public StoreCash getStoreCashObj() {
        // Returns the StoreCash action object appropriate for GasPump1
        return new StoreCashGP1();
    }

        @Override
        public InitializeData getInitializeDataObj() {
                // Returns the InitializeData action object appropriate for GasPump1
                return new InitializeDataGP1(this.ds);
        }

        @Override
        public StorePrices getStorePricesObj() {
                // Returns the StorePrices action object appropriate for GasPump1
                return new StorePricesGP1(this.ds);
        }

        @Override
        public WrongPinMsg getWrongPinMsgObj() {
                // Returns the WrongPinMsg action object appropriate for GasPump1
```

```java
                return new WrongPinMsgGP1();
        }

        @Override
        public StorePin getStorePinObj() {
                // Returns the StorePin action object appropriate for GasPump1
                return new StorePinGP1(this.ds);
        }

        @Override
        public EnterPinMsg getEnterPinMsgObj() {
                // Returns the EnterPinMsg action object appropriate for GasPump1
                return new EnterPinMsgGP1();
        }
}
package com.gaspumpproj.factory;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;
import com.gaspumpproj.strategy.action.cancelmsg.CancelMsg;
import com.gaspumpproj.strategy.action.cancelmsg.CancelMsgGP2;
import com.gaspumpproj.strategy.action.displaymenu.DisplayMenu;
import com.gaspumpproj.strategy.action.displaymenu.DisplayMenuGP2;
import com.gaspumpproj.strategy.action.enterpinmsg.EnterPinMsg;
import com.gaspumpproj.strategy.action.enterpinmsg.EnterPinMsgGP2;
import com.gaspumpproj.strategy.action.gaspumpedmsg.GasPumpedMsg;
import com.gaspumpproj.strategy.action.gaspumpedmsg.GasPumpedMsgGP2;
import com.gaspumpproj.strategy.action.initializedata.InitializeData;
import com.gaspumpproj.strategy.action.initializedata.InitializeDataGP2;
import com.gaspumpproj.strategy.action.paymsg.PayMsg;
import com.gaspumpproj.strategy.action.paymsg.PayMsgGP2;
import com.gaspumpproj.strategy.action.printreceipt.PrintReceipt;
import com.gaspumpproj.strategy.action.printreceipt.PrintReceiptGP2;
import com.gaspumpproj.strategy.action.pumpgasunit.PumpGasUnit;
import com.gaspumpproj.strategy.action.pumpgasunit.PumpGasUnitGP2;
import com.gaspumpproj.strategy.action.readymsg.ReadyMsg;
import com.gaspumpproj.strategy.action.readymsg.ReadyMsgGP2;
import com.gaspumpproj.strategy.action.rejectmsg.RejectMsg;
import com.gaspumpproj.strategy.action.rejectmsg.RejectMsgGP2;
import com.gaspumpproj.strategy.action.returncash.ReturnCash;
import com.gaspumpproj.strategy.action.returncash.ReturnCashGP2;
import com.gaspumpproj.strategy.action.setinitialvalues.SetInitialValues;
import com.gaspumpproj.strategy.action.setinitialvalues.SetInitialValuesGP2;
import com.gaspumpproj.strategy.action.setprice.SetPrice;
import com.gaspumpproj.strategy.action.setprice.SetPriceGP2;
import com.gaspumpproj.strategy.action.stopmsg.StopMsg;
import com.gaspumpproj.strategy.action.stopmsg.StopMsgGP2;
import com.gaspumpproj.strategy.action.storecash.StoreCash;
import com.gaspumpproj.strategy.action.storecash.StoreCashGP2;
import com.gaspumpproj.strategy.action.storepin.StorePin;
import com.gaspumpproj.strategy.action.storepin.StorePinGP2;
import com.gaspumpproj.strategy.action.storeprices.StorePrices;
import com.gaspumpproj.strategy.action.storeprices.StorePricesGP2;
import com.gaspumpproj.strategy.action.wrongpinmsg.WrongPinMsg;
import com.gaspumpproj.strategy.action.wrongpinmsg.WrongPinMsgGP2;

/**
 * This class is abstract factory which is related to Gas Pump 2.
 * Here, we instantiates the action strategies with the shared data structure for Gas Pump 2.
 *
 * @author borde
 *
 */
public class AbstractFactoryGP2 extends AbstractFactory {
    private DataStore ds;

    public AbstractFactoryGP2() {
        // Create new data store which is related to GasPump2.
        this.ds  = new DataStoreGP2();
    }

    @Override
    public DataStore getDataObj() {
        // Returns the data store appropriate for GasPump2
        return this.ds;
    }
    @Override
    public CancelMsg getCancelMsgObj() {
        // Returns the CancelMsg class for GasPump2.
        return new CancelMsgGP2();
    }

    @Override
    public DisplayMenu getDisplayMenuObj() {
        // Returns the DisplayMenu class that performs the appropriate action for displaying the menu prompt for GasPump2.
        return new DisplayMenuGP2(this.ds);
    }

    @Override
    public GasPumpedMsg getGasPumpedMsgObj() {
        // Returns the GasPumpedMsg class for GasPump2.
        return new GasPumpedMsgGP2(this.ds);
    }

    @Override
    public PayMsg getPayMsgObj() {
        // Returns the Payment prompt message action appropriate for GasPump2
        return new PayMsgGP2();
    }

    @Override
    public PrintReceipt getPrintReceiptObj() {
        // Returns the PrintReceipt class which is responsible for printing the GasPump2
        return new PrintReceiptGP2(this.ds);
    }

    @Override
    public PumpGasUnit getPumpGasUnitObj() {
        // Returns the PumpGasUnit class which is responsible to "pumping" a unit of gas for GasPump2
        return new PumpGasUnitGP2(this.ds);
    }

    @Override
    public ReadyMsg getReadyMsgObj() {
        //  Returns the ReadyMsg class which is responsible for prompting the user to start pumping gas on GasPump2.
        return new ReadyMsgGP2();
    }

    @Override
    public RejectMsg getRejectMsgObj() {
        // Returns the RejectMsg class that displays the appropriate credit card rejection message for GasPump2
        return new RejectMsgGP2();
    }

    @Override
    public ReturnCash getReturnCashObj() {
        // Returns the ReturnCash action object appropriate for GasPump2
        return new ReturnCashGP2(this.ds);
    }

    @Override
    public SetInitialValues getSetInitialValuesObj() {
        // Returns the SetInitialValues class which provides the appropriate action for initializing the shared data
structure with GasPump2
        return new SetInitialValuesGP2(this.ds);
    }

    @Override
    public SetPrice getSetPriceObj() {
        //  Returns the SetPrice class that is responsible for setting the price of gasoline according to specifications
of GasPump2 requirements.
        return new SetPriceGP2(this.ds);
    }

    @Override
    public StopMsg getStopMsgObj() {
        // Returns the StopMsg class which provides the appropriate action for displaying a message that the gas pump has
been stopped according to GasPump2
        return new StopMsgGP2();
    }
    @Override
    public StoreCash getStoreCashObj() {
        // Returns the StoreCash action object appropriate for GasPump2
        return new StoreCashGP2(this.ds);
    }

        @Override
        public InitializeData getInitializeDataObj() {
                // Returns the InitializeData action object appropriate for GasPump2
                return new InitializeDataGP2(this.ds);
        }

        @Override
        public StorePrices getStorePricesObj() {
                // Returns the StorePrices action object appropriate for GasPump2
                return new StorePricesGP2(this.ds);
        }

        @Override
        public WrongPinMsg getWrongPinMsgObj() {
                // Returns the WrongPinMsg action object appropriate for GasPump2
                return new WrongPinMsgGP2();
        }

        @Override
        public StorePin getStorePinObj() {
                // Returns the StorePin action object appropriate for GasPump2
                return new StorePinGP2(this.ds);
        }

        @Override
        public EnterPinMsg getEnterPinMsgObj() {
                // Returns the EnterPinMsg action object appropriate for GasPump2
                return new EnterPinMsgGP2();
        }
}
package com.gaspumpproj.gaspumps;

import com.gaspumpproj.data.DataStoreGP1;
import com.gaspumpproj.factory.AbstractFactory;

/**
 * This class is used as input processor for gas pump 1.
 * @author borde
 */
public class GasPump_1 extends GasPump {

        public GasPump_1(AbstractFactory af) {
            super(af);
        }

        /*
         * Operations of the Input Processor
         */
        // Validate input params and call activate event.
        public void Activate(float a, float b) {
                if (a > 0 && b > 0) {
            DataStoreGP1 d = (DataStoreGP1) this.ds;
            d.temp_a = a;
            d.temp_b = b;
            m.activate();
        } else {
            System.out.println("Invalid prices (price should be greater than 0).. Activation failed!");
        }

        }

        public void Start() {
                m.start();
        }

        public void PayCredit() {
                m.payType(1);
        }

        public void Reject() {
```

```java
                 m.reject();
        }

        public void PayDebit(String pin) {
                DataStoreGP1 d = (DataStoreGP1) this.ds;
                d.temp_p = pin;
                m.payType(3);
        }

        public void Pin(String x) {
                DataStoreGP1 d = (DataStoreGP1) this.ds;
                if ((d.pin).equals(x)) {
                        m.correctPin();
                } else {
                        m.incorrectPin();
                }
        }

        public void Cancel() {
                m.cancel();
        }

        public void Approved() {
                m.approved();
        }

        public void Diesel() {
                m.selectGas(4);
        }

        public void Regular() {
                m.selectGas(1);
        }

        public void StartPump() {
                DataStoreGP1 d = (DataStoreGP1) this.ds;
                if (d.price > 0) {
                        m.continueTr();
                        m.startPump();
                }
        }

        public void PumpGallon() {
                m.pump();
        }

        public void StopPump() {
                m.stopPump();
                m.receipt();
        }

        public void FullTank() {
                m.stopPump();
                m.receipt();
        }

}
package com.gaspumpproj.gaspumps;

import com.gaspumpproj.data.DataStoreGP2;
import com.gaspumpproj.factory.AbstractFactory;
/**
 * This class is used as input processor for gas pump 2.
 * @author borde
 */
public class GasPump_2 extends GasPump {

        public GasPump_2(AbstractFactory af) {
            super(af);
        }

        /*
         * Operations of the Input Processor
         */
        // Validate input params and call activate event.
        public void Activate(float a, float b, float c) {
                if (a > 0 && b > 0 && c > 0) {
            DataStoreGP2 d = (DataStoreGP2) this.ds;
            d.temp_a = a;
            d.temp_b = b;
            d.temp_c = c;
            m.activate();
        } else {
            System.out.println("Invalid prices (price should be greater than 0).. Activation failed!");
        }
        }

        public void PayCash(int cash) {
                if (cash > 0) {
                        DataStoreGP2 d = (DataStoreGP2) ds;
            d.temp_cash = cash;
            m.start();
            m.payType(2);
        } else {
            System.out.println("Invalid cash amount..");
        }
        }

        public void PayCredit() {
                m.start();
                m.payType(1);
        }

        public void Approved() {
                m.approved();
        }

        public void Reject() {
                m.reject();
        }

        public void Premium() {
                m.selectGas(3);
                m.continueTr();
        }

        public void Cancel() {
                m.cancel();
        }

        public void Regular() {
                m.selectGas(1);
                m.continueTr();
        }

        public void Super() {
                m.selectGas(2);
                m.continueTr();
        }

        public void PumpLiter() {
                DataStoreGP2 d = (DataStoreGP2) ds;
        if (d.cash>0 && (d.cash < d.price * (d.L + 1))) {
            m.stopPump();
        } else {
            m.pump();
        }
        }

        public void NoReceipt() {
                m.noReceipt();
        }

        public void Stop() {
                m.stopPump();
        }

        public void StartPump() {
                m.startPump();
        }

        public void Receipt() {
                m.receipt();
        }
}
package com.gaspumpproj.gaspumps;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.factory.AbstractFactory;
import com.gaspumpproj.model.outputprocessor.OutputProcessor;
import com.gaspumpproj.model.state.StateVMModel;


/**
 * This is the abstract class which is the super class of GasPump_1 and GasPump_2.
 * Here, we do initializations using abstract factory related to each Gas Pump.
 *
 * @author borde
 *
 */
public abstract class GasPump {
        DataStore ds;
    StateVMModel m;

    // Constructor to initialize the object creation and setting values.
    GasPump(AbstractFactory af) {
        OutputProcessor outp = new OutputProcessor(af);
        this.ds = af.getDataObj();
        this.m = new StateVMModel(outp);

        //        this.m.setOutputProcessor(outp);
    }

}
package com.gaspumpproj.model.outputprocessor;

import com.gaspumpproj.factory.AbstractFactory;
import com.gaspumpproj.strategy.action.cancelmsg.CancelMsg;
import com.gaspumpproj.strategy.action.displaymenu.DisplayMenu;
import com.gaspumpproj.strategy.action.enterpinmsg.EnterPinMsg;
import com.gaspumpproj.strategy.action.gaspumpedmsg.GasPumpedMsg;
import com.gaspumpproj.strategy.action.initializedata.InitializeData;
import com.gaspumpproj.strategy.action.paymsg.PayMsg;
import com.gaspumpproj.strategy.action.printreceipt.PrintReceipt;
import com.gaspumpproj.strategy.action.pumpgasunit.PumpGasUnit;
import com.gaspumpproj.strategy.action.readymsg.ReadyMsg;
import com.gaspumpproj.strategy.action.rejectmsg.RejectMsg;
import com.gaspumpproj.strategy.action.returncash.ReturnCash;
import com.gaspumpproj.strategy.action.setinitialvalues.SetInitialValues;
import com.gaspumpproj.strategy.action.setprice.SetPrice;
import com.gaspumpproj.strategy.action.stopmsg.StopMsg;
import com.gaspumpproj.strategy.action.storecash.StoreCash;
import com.gaspumpproj.strategy.action.storepin.StorePin;
import com.gaspumpproj.strategy.action.storeprices.StorePrices;
import com.gaspumpproj.strategy.action.wrongpinmsg.WrongPinMsg;


/**
 * This class is the general output processor for the gas pump system.
 * Each meta-action in this class calls the platform specific implementation of the action.
 * This class acts as the client class in the strategy design pattern.
 *
 * @author borde
 *
 */
public class OutputProcessor {
        private CancelMsg cancelMsg;
    private DisplayMenu displayMenu;
    private GasPumpedMsg gasPumpedMsg;
    private PayMsg payMsg;
    private PrintReceipt printReceipt;
    private PumpGasUnit pumpGasUnit;
    private ReadyMsg readyMsg;
    private RejectMsg rejectMsg;
    private ReturnCash returnCash;
    private SetInitialValues setInitialValues;
    private SetPrice setPrice;
    private StopMsg stopMsg;
```

```java
    private StoreCash storeCash;
    private InitializeData initializeData;
    private StorePrices storePrices;
    private WrongPinMsg wrongPinMsg;
    private StorePin storePin;
    private EnterPinMsg enterPinMsg;

    // Constructor for output processor class which initializes action class object values
    public OutputProcessor(AbstractFactory abstractfactory) {
        fetchObjects(abstractfactory);
    }

    private void fetchObjects(AbstractFactory abf) {
        this.cancelMsg = abf.getCancelMsgObj();
        this.displayMenu = abf.getDisplayMenuObj();
        this.gasPumpedMsg = abf.getGasPumpedMsgObj();
        this.payMsg = abf.getPayMsgObj();
        this.printReceipt = abf.getPrintReceiptObj();
        this.pumpGasUnit = abf.getPumpGasUnitObj();
        this.readyMsg = abf.getReadyMsgObj();
        this.rejectMsg = abf.getRejectMsgObj();
        this.returnCash = abf.getReturnCashObj();
        this.setInitialValues = abf.getSetInitialValuesObj();
        this.setPrice = abf.getSetPriceObj();
        this.stopMsg = abf.getStopMsgObj();
        this.storeCash = abf.getStoreCashObj();
        this.initializeData = abf.getInitializeDataObj();
        this.storePrices = abf.getStorePricesObj();
        this.wrongPinMsg =  abf.getWrongPinMsgObj();
        this.storePin =  abf.getStorePinObj();
        this.enterPinMsg =  abf.getEnterPinMsgObj();
    }
    /*
     * Following are the Meta actions which are implemented using strategy design pattern.
     */

    public void CancelMsg() {
        this.cancelMsg.cancelMsg();
    }

    public void DisplayMenu() {
        this.displayMenu.displayMenu();
    }

    public void GasPumpedMsg() {
        this.gasPumpedMsg.gasPumpedMsg();
    }

    public void PayMsg() {
        this.payMsg.payMsg();
    }

    public void PrintReceipt() {
        this.printReceipt.printReceipt();
    }

    public void PumpGasUnit() {
        this.pumpGasUnit.pumpGasUnit();
    }

    public void ReadyMsg() {
        this.readyMsg.readyMsg();
    }

    public void RejectMsg() {
        this.rejectMsg.rejectMsg();
    }

    public void ReturnCash() {
        this.returnCash.returnCash();
    }

    public void SetInitialValues() {
        this.setInitialValues.setInitialValues();
    }
    public void SetPrice(int g, int m) {
        this.setPrice.setPrice(g, m);
    }

    public void StopMsg() {
        this.stopMsg.stopMsg();
    }

    public void StoreCash() {
        this.storeCash.storeCash();
    }

    public void InitializeData() {
        this.initializeData.initializeData();
    }

    public void StorePrices(){
        this.storePrices.storePrices();
    }

    public void WrongPinMsg(){
        this.wrongPinMsg.wrongPinMsg();
    }

    public void StorePin(){
        this.storePin.storePin();
    }

    public void EnterPinMsg(){
        this.enterPinMsg.enterPinMsg();
    }
}
package com.gaspumpproj.model.state;

/**
 * MDA-EFSM: This class represents initial state.
 *
 * @author borde
 */
class InitialState extends State {

        InitialState(StateVMModel model) {
        this.modl = model;
        this.op = modl.getOutputProcessor();
    }

    @Override
    void activate() {
        // check current state
        if (modl.st == modl.LS[7]) {
        // Transition to State S0
        modl.st = modl.LS[0];
        // call the StorePrices() meta-action
        op.StorePrices();
        }
    }
}
package com.gaspumpproj.model.state;

/**
 * MDA-EFSM: This class represents S0 state.
 *
 * @author borde
 */
class S0 extends State {

        S0(StateVMModel model) {
        this.modl = model;
        this.op = modl.getOutputProcessor();
    }

    @Override
    void start() {
        if (modl.st == modl.LS[0]) {
        // Transition to State S1
        modl.st = modl.LS[1];
        //call the PayMsg(), InitializeData(), SetM() meta-action
        op.PayMsg();
        op.InitializeData();
        modl.M = 1;
        }
    }
}
package com.gaspumpproj.model.state;

/**
 * MDA-EFSM: This class represents S1 state.
 *
 * @author borde
 */
class S1 extends State {

        S1(StateVMModel model) {
        this.modl = model;
        this.op = modl.getOutputProcessor();
    }

        // credit: t=1;
    // cash: t=2;
    // debit: t=3;
    @Override
    void payType(int t) {
        if(modl.st == modl.LS[1]) {
                if (t == 1) {
                        // Transition to State S2
                        modl.st = modl.LS[2];
                } else if (t == 2) {
                        // Transition to State S3
                        modl.st = modl.LS[3];
                        op.StoreCash();
                        op.DisplayMenu();
                        modl.M = 0;
                } else if (t == 3) {
                        // Transition to State S8
                        modl.st = modl.LS[8];
                        op.EnterPinMsg();
                        op.StorePin();
                }
        }
    }
}
package com.gaspumpproj.model.state;

/**
 * MDA-EFSM: This class represents S2 state.
 *
 * @author borde
 */
class S2 extends State {

        S2(StateVMModel modl) {
        this.modl = modl;
        this.op = modl.getOutputProcessor();
    }

    @Override
    void approved() {
        if (modl.st == modl.LS[2]) {
        // Transition to State S3
        modl.st = modl.LS[3];
        // call DisplayMenu() meta-action
        op.DisplayMenu();
        }
    }

    @Override
    void reject() {
        if (modl.st == modl.LS[2]) {
        // Transition to State S0
            modl.st = modl.LS[0];
            // call RejectMsg() action
            op.RejectMsg();
        }
    }
}
package com.gaspumpproj.model.state;

/**
 * MDA-EFSM: This class represents S3 state.
 * 
 * @author borde
 */
class S3 extends State {

        S3(StateVMModel model) {
        this.modl = model;
        this.op = modl.getOutputProcessor();
    }

    @Override
    void selectGas(int g) {
        if (modl.st == modl.LS[3]) {
            op.SetPrice(g, modl.M);
        }
    }

    @Override
    void cancel() {
        if (modl.st == modl.LS[3]) {
        // Transition to State S0
        modl.st = modl.LS[0];
            op.CancelMsg();
            op.ReturnCash();
        }
    }

    @Override
    void continueTr() {
        if (modl.st == modl.LS[3]) {
        // Transition to State S4
        modl.st = modl.LS[4];
        }
    }
}
package com.gaspumpproj.model.state;

/**
 * MDA-EFSM: This class represents S4 state.
 * 
 * @author borde
 */
class S4 extends State {

        S4(StateVMModel model) {
        this.modl = model;
        this.op = modl.getOutputProcessor();
    }

    @Override
    void startPump() {
        if (modl.st == modl.LS[4]) {
        // Transition to State S5
        modl.st = modl.LS[5];
        //  call SetInitialValues() and ReadyMsg()
            op.SetInitialValues();
            op.ReadyMsg();
        }
    }
}
package com.gaspumpproj.model.state;

/**
 * MDA-EFSM: This class represents S5 state.
 * 
 * @author borde
 */
class S5 extends State {
        S5(StateVMModel model) {
        this.modl = model;
        this.op = modl.getOutputProcessor();
    }

    @Override
    void pump() {
        if (modl.st == modl.LS[5]) {
            // do self loop
        // call PumpGasUnit() and GasPumpedMsg()
            op.PumpGasUnit();
            op.GasPumpedMsg();
        }
    }

    @Override
    void stopPump() {
        if (modl.st == modl.LS[5]) {
        // Transition to State S6
        modl.st = modl.LS[6];
        // call StopMsg() action
        op.StopMsg();
        }
    }
}
package com.gaspumpproj.model.state;

/**
 * MDA-EFSM: This class represents S6 state.
 *
 * @author borde
 */
class S6 extends State {

        S6(StateVMModel model) {
        this.modl = model;
        this.op = modl.getOutputProcessor();
    }

    @Override
    void receipt() {
        if (modl.st == modl.LS[6]) {
        // Transition to State S0
        modl.st = modl.LS[0];
        // call PrintReceipt() and ReturnCash() actions
        op.PrintReceipt();
        op.ReturnCash();
        }
    }

    @Override
    void noReceipt() {
        if (modl.st == modl.LS[6]) {
        // Transition to State S0
        modl.st = modl.LS[0];
        // call ReturnCash() meta-actions
        op.ReturnCash();
        }
    }
}
package com.gaspumpproj.model.state;

/**
 * MDA-EFSM: This class represents S8 state.
 *
 * @author borde
 */
class S8 extends State {

        S8(StateVMModel model) {
        this.modl = model;
        this.op = modl.getOutputProcessor();
    }

    @Override
    void correctPin() {
        if (modl.st == modl.LS[8]) {
        // Transition to State S3
                        modl.st = modl.LS[3];
                        op.DisplayMenu();
        }
    }

    @Override
    void incorrectPin() {
        if (modl.st == modl.LS[8]) {
        // Transition to State S0
                        modl.st = modl.LS[0];
                        op.WrongPinMsg();
        }
    }
}
package com.gaspumpproj.model.state;

import com.gaspumpproj.model.outputprocessor.OutputProcessor;

/**
 * This class is the abstract class. It is the super class of all state classes.
 * In this class each operation should be abstract but currently i am considering each method implementation
 * does not do anything here. I have done it as to avoid implementation of methods which are not required in state
 * classes. To avoid empty methods in sub classes i have kept it in this class with "Do nothing methods"
 *
 * Each state subclass inherits these methods and overrides the appropriate ones.
   This means that methods that do not get overridden will show "Access Denied" message.
 *
 * @author borde
 *
 */
public abstract class State {
    protected StateVMModel modl;
    protected OutputProcessor op;

    void activate() {
        showAccessDeniedMsg();
    }
    void start(){
        showAccessDeniedMsg();
    }

    void payType(int t){
        showAccessDeniedMsg();
    }
    void approved(){
        showAccessDeniedMsg();
    }
    void reject(){
        showAccessDeniedMsg();
    }
    void selectGas(int g){
        showAccessDeniedMsg();
    }
    void cancel(){
        showAccessDeniedMsg();
    }
    void startPump(){
        showAccessDeniedMsg();
    }
    void pump(){
        showAccessDeniedMsg();
    }
    void stopPump(){
        showAccessDeniedMsg();
    }
    void receipt(){
        showAccessDeniedMsg();
    }
    void noReceipt(){
        showAccessDeniedMsg();
    }
    void correctPin() {
        showAccessDeniedMsg();
    }
    void incorrectPin() {
        showAccessDeniedMsg();
    }
    // continue
    void continueTr() {
        showAccessDeniedMsg();
    }

    private void showAccessDeniedMsg() {
        System.out.println("Access Denied: This operation not allowed here.");
    }
}
package com.gaspumpproj.model.state;

import com.gaspumpproj.model.outputprocessor.OutputProcessor;

/**
 * Here, I am using the De-centralized State Design Pattern.
 * Each state has it's own classes to perform actions as well as change of state.
 * State = 7 is the initial state.
 * Other wise all the state class names represents the state name.
 *
 * @author borde
 *
 */
public class StateVMModel {
    protected State st;
    protected int M;
    // There are 9 states in our state diagram.
    protected State[] LS = new State[9];
    private OutputProcessor op;

    public StateVMModel(OutputProcessor op) {
        this.op = op;
        createStateObjects();
        // State = 7 is the initial state.
        st = LS[7];
    }

    private void createStateObjects() {
         // instantiate each state
        LS[7] = new InitialState(this);
        LS[0] = new S0(this);
        LS[1] = new S1(this);
        LS[2] = new S2(this);
        LS[3] = new S3(this);
        LS[4] = new S4(this);
        LS[5] = new S5(this);
        LS[6] = new S6(this);
        LS[8] = new S8(this);
    }

    /*
     * Implementation of state event functions.
     * Here, we forward the call to the state particular state class to perform action and state transition.

     */

    public void activate() {
        st.activate();
    }

    public void start() {
        st.start();
    }

    // credit: t=1;
    // cash: t=2;
    // debit: t=3;
    public void payType(int t) {
        st.payType(t);
    }

    public void reject() {
        st.reject();
    }
    public void cancel() {
        st.cancel();
    }

    public void approved() {
        st.approved();
    }

    // Regular: g=1;
    // Super: g=2;
    // Premium: g=3;
    // Diesel: g=4
    public void selectGas(int g) {
        st.selectGas(g);
    }

    public void startPump() {
        st.startPump();
    }

    public void pump() {
        st.pump();
    }

    public void stopPump() {
        st.stopPump();
    }

    public void receipt() {
        st.receipt();
    }

    public void noReceipt() {
        st.noReceipt();
    }

    public void correctPin() {
        st.correctPin();
    }

    public void incorrectPin() {
        st.incorrectPin();
    }

    // continue
    public void continueTr() {
        st.continueTr();
    }

    /*
        OutputProcessor : Getters and Setters
         */
        public OutputProcessor getOutputProcessor() {
            return this.op;
        }

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

}
package com.gaspumpproj.strategy.action.cancelmsg;
/**
 * Here, I have used Strategy Pattern.
 * This is the super class of CancelMsgGP1 and CancelMsgGP2. Also, this class is abstract class.
 * This class represents to "CancelMsg" action which is used to displays a cancellation message.
 *
 * In this case, i have created two action strategy subclasses (CancelMsgGP1 and CancelMsgGP2)
 * each one separately for each gas pump class by considering future modifications.
 *
 * @author borde
 *
 */

public abstract class CancelMsg {
    public abstract void cancelMsg();
}
package com.gaspumpproj.strategy.action.cancelmsg;

/**
 * This is CancelMsg action class related to Gas pump 1.
 * @author borde
 *
 */

public class CancelMsgGP1 extends CancelMsg {

    @Override
    public void cancelMsg() {
        // Print a cancellation message
        System.out.println("Transaction has been cancelled for gas pump #1..");
    }
}
package com.gaspumpproj.strategy.action.cancelmsg;

/**
 * This is CancelMsg action class related to Gas pump 2.
 * @author borde
 *
 */

public class CancelMsgGP2 extends CancelMsg {

    @Override
    public void cancelMsg() {
        // Print a cancellation message
        System.out.println("Transaction has been cancelled for gas pump #2..");
    }
}

package com.gaspumpproj.strategy.action.displaymenu;

import com.gaspumpproj.data.DataStore;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of DisplayMenuGP1 and DisplayMenuGP2. Also, this class is abstract class.
 * This class represents to "DisplayMenu" action which is used to display a menu with a list of selections.
 *
 * @author borde
 *
 */
public abstract class DisplayMenu {
    DataStore ds;

    public abstract void displayMenu();

}
package com.gaspumpproj.strategy.action.displaymenu;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;

/**
 * This is DisplayMenu action class related to Gas pump 1.
 * @author borde
 *
 */
public class DisplayMenuGP1 extends DisplayMenu {

    public DisplayMenuGP1(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void displayMenu() {
        // display a menu with a list of selections
        DataStoreGP1 data = (DataStoreGP1) ds;

        System.out.println("Card has been approved. \n Please select gas type from following list: ");
        System.out.println(
```

```java
                "\n#8 Diesel (Rate: $" + data.Dprice + " per gallon) " +
                "\n#9 Regular (Rate: $" + data.Rprice + " per gallon) " +
                "Please select #6 to cancel");
    }
}
package com.gaspumpproj.strategy.action.displaymenu;


import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;

/**
 * This is DisplayMenu action class related to Gas pump 2.
 * @author borde
 *
 */
public class DisplayMenuGP2 extends DisplayMenu {

    public DisplayMenuGP2(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void displayMenu() {
        // display a menu with a list of selections
         DataStoreGP2 data = (DataStoreGP2) ds;
         System.out.println("Please select gas type from following list: ");
         System.out.println(
                 "\n#6 Premium (Rate: $" + data.Pprice + " per liter)" +
                 "\n#7 Regular (Rate: $" + data.Rprice + "per liter) " +
                 "\n#8 Super (Rate: $" + data.Sprice + "per liter) " +
                 "Please select #5 to cancel");

    }
}

package com.gaspumpproj.strategy.action.enterpinmsg;
/**
 * Here, I have used Strategy Pattern.
 * This is the super class of EnterPinMsgGP1 and EnterPinMsgGP2. Also, this class is abstract class.
 * This class represents to "EnterPinMsg" action which is used to display a message to enter pin.
 *
 * @author borde
 *
 */
public abstract class EnterPinMsg {

    public abstract void enterPinMsg();
}
package com.gaspumpproj.strategy.action.enterpinmsg;

/**
 * This is EnterPinMsg action class related to Gas pump 1.
 * @author borde
 *
 */
public class EnterPinMsgGP1 extends EnterPinMsg {

    @Override
    public void enterPinMsg() {
        // displays a message to enter pin
        System.out.println("This operation is related to Gas Pump #1..");
        System.out.println("Please enter pin..");
    }
}
package com.gaspumpproj.strategy.action.enterpinmsg;

/**
 * This is EnterPinMsg action class related to Gas pump 2.
 * This class can be used in future.
 *
 * @author borde
 *
 */
public class EnterPinMsgGP2 extends EnterPinMsg {
    @Override
    public void enterPinMsg() {
        // Do nothing
    }
}
package com.gaspumpproj.strategy.action.gaspumpedmsg;

import com.gaspumpproj.data.DataStore;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of GasPumpedMsgGP1 and GasPumpedMsgGP2. Also, this class is abstract class.
 * This class represents to "GasPumpedMsg" action which is used to display the amount of disposed gas.
 * 
 * @author borde
 *
 */
public abstract class GasPumpedMsg {
        DataStore ds;

    public abstract void gasPumpedMsg();
}
package com.gaspumpproj.strategy.action.gaspumpedmsg;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;


/**
 * This is GasPumpedMsg action class related to Gas pump 1.
 * @author borde
 *
 */
public class GasPumpedMsgGP1 extends GasPumpedMsg {

    public GasPumpedMsgGP1(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void gasPumpedMsg() {
        // displays the amount of disposed gas
        DataStoreGP1 data = (DataStoreGP1) ds;
        System.out.println("Total number of gallons pumped: " + data.G);
        System.out.println("Total price: " + data.total);
    }

}
package com.gaspumpproj.strategy.action.gaspumpedmsg;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;

/**
 * This is GasPumpedMsg action class related to Gas pump 2.
 * @author borde
 *
 */
public class GasPumpedMsgGP2 extends GasPumpedMsg {

    public GasPumpedMsgGP2(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void gasPumpedMsg() {

        // displays the amount of disposed gas
        DataStoreGP2 data = (DataStoreGP2) ds;
        System.out.println("Total number of liters pumped: " + data.L);
        System.out.println("Total price: " + data.total);
    }
}
package com.gaspumpproj.strategy.action.initializedata;

import com.gaspumpproj.data.DataStore;
```

```
/**
 * Here, I have used Strategy Pattern.
 * This is the super class of InitialDataGP1 and InitialDataGP2. Also, this class is abstract class.
 * This class represents to "InitialData" action which is used to set the value of price to 0.
 *
 * @author borde
 *
 */
public abstract class InitializeData {
        DataStore ds;

    public abstract void initializeData();
}
package com.gaspumpproj.strategy.action.initializedata;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;

/**
 * This is InitializeData action class related to Gas pump 1.
 * @author borde
 *
 */
public class InitializeDataGP1 extends InitializeData {

        public InitializeDataGP1(DataStore ds) {
            this.ds = ds;
        }

    @Override
    public void initializeData() {
        // set the value of price to 0
        DataStoreGP1 data = (DataStoreGP1) ds;
        data.price = 0;
    }
}
package com.gaspumpproj.strategy.action.initializedata;

import com.gaspumpproj.data.DataStore;

/**
 * This is InitializeData action class related to Gas pump 2.
 * @author borde
 *
 */
public class InitializeDataGP2 extends InitializeData {

        public InitializeDataGP2(DataStore ds) {
            this.ds = ds;
        }

    @Override
    public void initializeData() {
        // set the initial values in future
    }
}
package com.gaspumpproj.strategy.action.paymsg;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of PayMsgGP1 and PayMsgGP2. Also, this class is abstract class.
 * This class represents to "PayMsg" action which is used to display a type of payment method.
 *
 * @author borde
 *
 */
public abstract class PayMsg {

    public abstract void payMsg();
}
package com.gaspumpproj.strategy.action.paymsg;

/**
 * This is PayMsg action class related to Gas pump 1.
 * @author borde
```

```java
 *
 */
public class PayMsgGP1 extends PayMsg {

    @Override
    public void payMsg() {
        // displays a type of payment method
        System.out.println("Current selection is Gas pump #1");
        System.out.println("Please select payment type..");
    }
}
package com.gaspumpproj.strategy.action.paymsg;

/**
 * This is PayMsg action class related to Gas pump 2.
 * @author borde
 *
 */
public class PayMsgGP2 extends PayMsg {

    @Override
    public void payMsg() {
        // displays a type of payment method
        System.out.println("Current selection is Gas pump #2");
        System.out.println("Please select payment type..");
    }
}
package com.gaspumpproj.strategy.action.printreceipt;

import com.gaspumpproj.data.DataStore;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of PrintReceiptGP1 and PrintReceiptGP2. Also, this class is abstract class.
 * This class represents to "PrintReceipt" action which is used to print a receipt.
 *
 * @author borde
 *
 */
public abstract class PrintReceipt {
        DataStore ds;

    public abstract void printReceipt();
}
package com.gaspumpproj.strategy.action.printreceipt;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;

/**
 * This is PrintReceipt action class related to Gas pump 1.
 * @author borde
 *
 */
public class PrintReceiptGP1 extends PrintReceipt {

        public PrintReceiptGP1(DataStore ds) {
            this.ds = ds;
        }

    @Override
    public void printReceipt() {
        // print a receipt
        DataStoreGP1 data = (DataStoreGP1) ds;
        System.out.println("Printing receipt ...");
        System.out.println("==========================================================================");
        System.out.println(" Number of gallons: " + data.G);
        System.out.println(" Fuel price: $" + data.price + " per gallon");

        System.out.println("\n--------------------------------------------");
        System.out.println("Total amount : $" + data.total);
        System.out.println("===========================================================================");
        System.out.println("Thanks for using Gas Pump #1.\nTransaction completed successfully..");
    }
}
package com.gaspumpproj.strategy.action.printreceipt;
```

```java
import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;

/**
 * This is PrintReceipt action class related to Gas pump 2.
 * @author borde
 *
 */
public class PrintReceiptGP2 extends PrintReceipt {

        public PrintReceiptGP2(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void printReceipt() {
        // print a receipt

        DataStoreGP2 data = (DataStoreGP2) ds;
        System.out.println("Printing receipt ...");
        System.out.println("==========================================================================");
        System.out.println(" Number of liters: " + data.L);
        System.out.println(" Fuel price: $" + data.price + " per liter");
        System.out.println(" Cash inserted: $" + data.cash);
        System.out.println("\n--------------------------------------------");
        System.out.println("Total amount : $" + data.total);
        System.out.println("===========================================================================");
        System.out.println("Thanks for using Gas Pump #2.\nTransaction completed successfully..");
    }
}
package com.gaspumpproj.strategy.action.pumpgasunit;

import com.gaspumpproj.data.DataStore;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of PumpGasUnitGP1 and PumpGasUnitGP2. Also, this class is abstract class.
 * This class represents to "PumpGasUnit" action which is used to dispose unit of gas and counts # of units disposed.
 *
 * @author borde
 *
 */
public abstract class PumpGasUnit {
        DataStore ds;

    public abstract void pumpGasUnit();
}
package com.gaspumpproj.strategy.action.pumpgasunit;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;


/**
 * This is PumpGasUnit action class related to Gas pump 1.
 * @author borde
 *
 */
public class PumpGasUnitGP1 extends PumpGasUnit {

         public PumpGasUnitGP1(DataStore ds) {
                this.ds = ds;
            }

    @Override
    public void pumpGasUnit() {
        // disposes unit of gas and counts # of units disposed
        DataStoreGP1 data = (DataStoreGP1) ds;
        // Do processing
        data.G = data.G + 1;
        data.total = data.price * data.G;
    }
}
package com.gaspumpproj.strategy.action.pumpgasunit;
```

```
import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;


/**
 * This is PumpGasUnit action class related to Gas pump 2.
 * @author borde
 *
 */
public class PumpGasUnitGP2 extends PumpGasUnit {

        public PumpGasUnitGP2(DataStore ds) {
            this.ds = ds;
        }

    @Override
    public void pumpGasUnit() {

        // disposes unit of gas and counts # of units disposed
        DataStoreGP2 data = (DataStoreGP2) ds;

        // Do processing
        data.L = data.L + 1;
        data.total = data.price * data.L;
    }
}
package com.gaspumpproj.strategy.action.readymsg;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of ReadyMsgGP1 and ReadyMsgGP2. Also, this class is abstract class.
 * This class represents to "ReadyMsg" action which is used to displays the ready for pumping message.
 *
 * @author borde
 *
 */

public abstract class ReadyMsg {

        public abstract void readyMsg();

}

package com.gaspumpproj.strategy.action.readymsg;

/**
 * This is ReadyMsg action class related to Gas pump 1.
 * @author borde
 *
 */

public class ReadyMsgGP1 extends ReadyMsg {

    @Override
    public void readyMsg() {
        // Display Message
        System.out.println("All set. Ready to dispense fuel..");
        System.out.println("Press key #b to dispense 1 gallon fuel");
        System.out.println("Press key #c to stop");
    }
}
package com.gaspumpproj.strategy.action.readymsg;

/**
 * This is ReadyMsg action class related to Gas pump 2.
 * @author borde
 *
 */

public class ReadyMsgGP2 extends ReadyMsg {

    @Override
    public void readyMsg() {
        // Display Message
        System.out.println("All set. Ready to dispense fuel..");
        System.out.println("Press key #a to dispense 1 liter fuel");
        System.out.println("Press key #b to stop");
    }
}
package com.gaspumpproj.strategy.action.rejectmsg;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of RejectMsgGP1 and RejectMsgGP2. Also, this class is abstract class.
 * This class represents to "RejectMsg" action which is used to display credit card not approved message.
 *
 * In this case, i have created two action strategy subclasses (RejectMsgGP1 and RejectMsgGP2)
 * each one separately for each gas pump class by considering future modifications.
 *
 * @author borde
 *
 */
public abstract class RejectMsg {

    public abstract void rejectMsg();
}
package com.gaspumpproj.strategy.action.rejectmsg;

/**
 * This is RejectMsg action class related to Gas pump 1.
 * @author borde
 *
 */
public class RejectMsgGP1 extends RejectMsg {

    @Override
    public void rejectMsg() {
        // displays credit card not approved message
        System.out.println("Credit card has been rejected.");
        System.out.println("Transaction has been cancelled for gas pump #1..");
    }
}
package com.gaspumpproj.strategy.action.rejectmsg;

/**
 * This is CancelMsg action class related to Gas pump 2.
 * This class can be used in future.
 * Currently, it does not do anything as Gas pump 2 does not support card payment.
 *
 * @author borde
 *
 */

public class RejectMsgGP2 extends RejectMsg {

    @Override
    public void rejectMsg() {
        // Do nothing
    }
}
package com.gaspumpproj.strategy.action.returncash;

import com.gaspumpproj.data.DataStore;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of ReturnCashGP1 and ReturnCashGP2. Also, this class is abstract class.
 * This class represents to "ReturnCash" action which is used to returns the remaining cash.
 *
 * @author borde
 *
 */
public abstract class ReturnCash {
   DataStore ds;

   public abstract void returnCash();
}
package com.gaspumpproj.strategy.action.returncash;

/**
 * This is ReturnCash action class related to Gas pump 1.
 * This class is for future use as Gas Pump 1 does not support payment with cash.
 *
 * @author borde
 *
 */
public class ReturnCashGP1 extends ReturnCash {

    @Override
    public void returnCash() {
        // Do nothing
    }
}
package com.gaspumpproj.strategy.action.returncash;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;

/**
 * This is ReturnCash action class related to Gas pump 2.
 *
 * @author borde
 *
 */
public class ReturnCashGP2 extends ReturnCash {

        public ReturnCashGP2(DataStore ds) {
             this.ds = ds;
        }

    @Override
    public void returnCash() {
        // returns the remaining cash
        DataStoreGP2 data = (DataStoreGP2) ds;
        float creturn = data.cash - data.total;
        if (creturn > 0) {
            System.out.println("Calculated cash return: $" + creturn);
            System.out.println("Returning $" + creturn);
        } else {
            System.out.println("No cash to return");
        }
        data.cash = 0;
        System.out.println("Thanks for using gas pump #2. \nTransaction completed successfully..");
    }
}
package com.gaspumpproj.strategy.action.setinitialvalues;

import com.gaspumpproj.data.DataStore;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of SetInitialValuesGP1 and SetInitialValuesGP2. Also, this class is abstract class.
 * This class represents to "SetInitialValues" action which is used to set G (or L) and total to 0.
 *
 * @author borde
 *
 */
public abstract class SetInitialValues {
        DataStore ds;

    public abstract void setInitialValues();
}
package com.gaspumpproj.strategy.action.setinitialvalues;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;

/**
 * This is SetInitialValues action class related to Gas pump 1.
 * @author borde
 *
 */
public class SetInitialValuesGP1 extends SetInitialValues {

        public SetInitialValuesGP1(DataStore ds) {
            this.ds = ds;
        }

    @Override
    public void setInitialValues() {
        // set G and total to 0;
        DataStoreGP1 data = (DataStoreGP1) ds;
        data.G = 0;
        data.total = 0;
    }
}
package com.gaspumpproj.strategy.action.setinitialvalues;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;

/**
 * This is SetInitialValues action class related to Gas pump 2.
 * @author borde
 *
 */
public class SetInitialValuesGP2 extends SetInitialValues {

        public SetInitialValuesGP2(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void setInitialValues() {
        // set L and total to 0;
        DataStoreGP2 data = (DataStoreGP2) ds;
        data.L = 0;
        data.total = 0;
    }
}
package com.gaspumpproj.strategy.action.setprice;

import com.gaspumpproj.data.DataStore;


/**
 * Here, I have used Strategy Pattern.
 * This is the super class of SetPriceGP1 and SetPriceGP2. Also, this class is abstract class.
 * This class represents to "SetPrice" action which is used to set the price for the gas identified by g identifier as in SelectGas(int g); if M=1, the price may be increased.
 *
 * @author borde
 *
 */
public abstract class SetPrice {
    DataStore ds;

    // Regular: g=1;
    // Super: g=2;
    // Premium: g=3;
    // Diesel: g=4
    public abstract void setPrice(int g, int m);

}
package com.gaspumpproj.strategy.action.setprice;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;

/**
 * This is SetPrice action class related to Gas pump 1.
 *
 * @author borde
 *
 */
public class SetPriceGP1 extends SetPrice {

    public SetPriceGP1(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void setPrice(int g, int m) {
        // set the price for the gas identified by g identifier as in SelectGas(int g);
        // if M=1, the price may be increased
```

```java
        DataStoreGP1 data = (DataStoreGP1) ds;
        String gType = "";
        if (g == 1) {
        // Regular
        data.price = data.Rprice;
        gType = "Regular";
        } else if (g == 4) {
        // Diesel
        data.price = data.Dprice;
        gType = "Diesel";
        }
        System.out.println("Selected Gas Type : " + gType);
        System.out.println("Press key #a to start the pump..");
    }
}
package com.gaspumpproj.strategy.action.setprice;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;

/**
 * This is SetPrice action class related to Gas pump 2.
 * 
 * @author borde
 *
 */
public class SetPriceGP2 extends SetPrice {

    public SetPriceGP2(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void setPrice(int g, int m) {
        // set the price for the gas identified by g identifier as in SelectGas(int g);
        // if M=1, the price may be increased
        DataStoreGP2 data = (DataStoreGP2) ds;
        String gType = "";
        if (g == 1) {
        // Regular
        data.price = data.Rprice;
        gType = "Regular";
        } else if (g == 2) {
        // Super
        data.price = data.Sprice;
        gType = "Super";
        } else if (g == 3) {
        // Premium
        data.price = data.Pprice;
        gType = "Premium";
        }

        System.out.println("Selected Gas Type : " + gType);
        System.out.println("Press key #9 to start the pump..");

        if(m == 1) {
        data.price=(float) (1.1*data.price);
        System.out.println("Note: Price increased -> price=1.1*price.");
        }
    }
}
package com.gaspumpproj.strategy.action.stopmsg;
/**
 * Here, I have used Strategy Pattern.
 * This is the super class of StopMsgGP1 and StopMsgGP2. Also, this class is abstract class.
 * This class represents to "StopMsg" action which is used to stop pump message and receipt msg (optionally).
 * 
 * In this case, i have created two action strategy subclasses (StopMsgGP1 and StopMsgGP2)
 * each one separately for each gas pump class by considering future modifications.
 * 
 * @author borde
 *
 */
public abstract class StopMsg {

    public abstract void stopMsg();
```

```java
}
package com.gaspumpproj.strategy.action.stopmsg;

/**
 * This is StopMsg action class related to Gas pump 1.
 * @author borde
 *
 */
public class StopMsgGP1 extends StopMsg {

    @Override
    public void stopMsg() {
        // stop pump message and receipt msg (optionally)
        System.out.println("Gas pump #1 has been stopped..");
    }
}
package com.gaspumpproj.strategy.action.stopmsg;

/**
 * This is StopMsg action class related to Gas pump 2.
 *
 * @author borde
 */
public class StopMsgGP2 extends StopMsg {

    @Override
    public void stopMsg() {
        // stop pump message and receipt msg (optionally)
        System.out.println("Gas pump #1 has been stopped..");
    }
}
package com.gaspumpproj.strategy.action.storecash;

import com.gaspumpproj.data.DataStore;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of StoreCashGP1 and StoreCashGP2. Also, this class is abstract class.
 * This class represents to "StoreCash" action which is used to store cash from the temporary data store.
 *
 * @author borde
 *
 */
public abstract class StoreCash {
        DataStore ds;

    public abstract void storeCash();
}
package com.gaspumpproj.strategy.action.storecash;

/**
 * This is StoreCash action class related to Gas pump 1.
 * This is for the future use as Gas Pump 1 does not support payment with cash.
 *
 * @author borde
 *
 */
public class StoreCashGP1 extends StoreCash {

    @Override
    public void storeCash() {
        // Do nothing
    }
}
package com.gaspumpproj.strategy.action.storecash;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;

/**
 * This is StoreCash action class related to Gas pump 2.
 *
 * @author borde
 *
 */
public class StoreCashGP2 extends StoreCash {
    public StoreCashGP2(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void storeCash() {
        // stores cash from the temporary data store
        DataStoreGP2 data = (DataStoreGP2) ds;
        data.cash = data.temp_cash;
        System.out.println("Cash added to system $" + data.cash + " successfully..");
    }
}
package com.gaspumpproj.strategy.action.storepin;

import com.gaspumpproj.data.DataStore;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of StorePinGP1 and StorePinGP2. Also, this class is abstract class.
 * This class represents to "StorePin" action which is used to store the pin from the temporary data store.
 *
 * @author borde
 *
 */
public abstract class StorePin {
        DataStore ds;

    public abstract void storePin();
}
package com.gaspumpproj.strategy.action.storepin;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;

/**
 * This is StorePin action class related to Gas pump 1.
 *
 * @author borde
 *
 */
public class StorePinGP1 extends StorePin {

        public StorePinGP1(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void storePin() {
        // stores pin from the temporary data store
        DataStoreGP1 data = (DataStoreGP1) ds;
        data.pin = data.temp_p;
        System.out.println("Pin has been saved successfully..");
    }
}
package com.gaspumpproj.strategy.action.storepin;

import com.gaspumpproj.data.DataStore;

/**
 * This is StorePin action class related to Gas pump 2.
 * This class is for future use.
 *
 * @author borde
 *
 */
public class StorePinGP2 extends StorePin {

    public StorePinGP2(DataStore ds) {
        this.ds = ds;
    }

    @Override
    public void storePin() {
        // Do nothing
    }
```

```java
}
package com.gaspumpproj.strategy.action.storeprices;

import com.gaspumpproj.data.DataStore;

/**
 * Here, I have used Strategy Pattern.
 * This is the super class of StorePricesGP1 and StorePricesGP2. Also, this class is abstract class.
 * This class represents to "StorePrices" action which is used to stores price(s) for the gas from the temporary data
store.
 *
 * @author borde
 *
 */
public abstract class StorePrices {
        DataStore ds;

    public abstract void storePrices();
}
package com.gaspumpproj.strategy.action.storeprices;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP1;

/**
 * This is StorePrices action class related to Gas pump 1.
 *
 * @author borde
 *
 */
public class StorePricesGP1 extends StorePrices {

        public StorePricesGP1(DataStore ds) {
            this.ds = ds;
        }

    @Override
    public void storePrices() {
        // stores initial prices
        // stores price(s) for the gas from the temporary data store
        DataStoreGP1 data = (DataStoreGP1) ds;
        data.Rprice = data.temp_a;
        data.Dprice = data.temp_b;
        System.out.println("Gas Pump #1 activated successfully!!!");
    }
}
package com.gaspumpproj.strategy.action.storeprices;

import com.gaspumpproj.data.DataStore;
import com.gaspumpproj.data.DataStoreGP2;

/**
 * This is StorePrices action class related to Gas pump 2.
 *
 * @author borde
 *
 */
public class StorePricesGP2 extends StorePrices {

        public StorePricesGP2(DataStore ds) {
            this.ds = ds;
        }

    @Override
    public void storePrices() {
        // stores initial prices
        // stores price(s) for the gas from the temporary data store
        DataStoreGP2 data = (DataStoreGP2) ds;
        data.Rprice = data.temp_b;
        data.Pprice = data.temp_c;
        data.Sprice = data.temp_a;
        System.out.println("Gas Pump #2 activated successfully!!!");
    }
}
package com.gaspumpproj.strategy.action.wrongpinmsg;
/**
```

```java
 * Here, I have used Strategy Pattern.
 * This is the super class of WrongPinMsgGP1 and WrongPinMsgGP2. Also, this class is abstract class.
 * This class represents to "WrongPinMsg" action which is used to display incorrect pin message.
 *
 * @author borde
 *
 */
public abstract class WrongPinMsg {

    public abstract void wrongPinMsg();
}
package com.gaspumpproj.strategy.action.wrongpinmsg;

/**
 * This is WrongPinMsg action class related to Gas pump 1.
 * @author borde
 *
 */
public class WrongPinMsgGP1 extends WrongPinMsg {

    @Override
    public void wrongPinMsg() {
        // displays incorrect pin message
        System.out.println("Wrong pin entered in Gas Pump #1..");
    }
}
package com.gaspumpproj.strategy.action.wrongpinmsg;

/**
 * This is WrongPinMsg action class related to Gas pump 2.
 * This is for future use.
 *
 * @author borde
 */
public class WrongPinMsgGP2 extends WrongPinMsg {

    @Override
    public void wrongPinMsg() {
        // Do nothing
    }
}
```