CS 586

Project Report

GasPump

Shraddha Mantri

A20378714

Contents:

Overview

The goal of the project is to implement two GasPumps using Model Driven Architecture. To implement two GasPump design following design patterns have been used:

- State Pattern
- Abstract Factory Pattern
- Strategy Pattern

The two Gas Pumps differ in following components:

- Unit of Gas Pumped
- Type of Payment
- Types of Gas
- DataTypes
- Displaying Menu
- Messages
- Receipt

The user has the option to choose the gas of his choice and perform the operations as per his convenience.

MDA-EFSM Events:

Activate()

Start()

PayType(int t) //credit: t=1; cash: t=2

Reject()

Cancel()

Approved()

StartPump()

Pump()

StopPump()

SelectGas(int g)

Receipt()

NoReceipt()

MDA-EFSM Actions:

StoreData // stores price(s) for the gas from the temporary data store

PayMsg // displays a type of payment method

StoreCash // stores cash from the temporary data store

DisplayMenu // display a menu with a list of selections

RejectMsg // displays credit card not approved message

SetPrice(int g) // set the price for the gas identified by g identifier

ReadyMsg // displays the ready for pumping message

SetInitialValues // set G (or L) and total to 0

PumpGasUnit // disposes unit of gas and counts # of units disposed

GasPumpedMsg // displays the amount of disposed gas

StopMsg // stop pump message and receipt? msg (optionally)

PrintReceipt // print a receipt CancelMsg // displays a cancellation message

ReturnCash // returns the remaining cash

Pseudo Code:

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

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

d: is a pointer to the Data Store object

Operations of the Input Processor (GasPump-2)

```
Activate(int a, int b, int c) {
        if ((a>0)&&(b>0)&&(c>0)) {
                d->temp_a=a;
                d->temp_b=b;
                d->temp_c=c
                m->Activate()
        }
}
Start() {
        m->Start();
}
PayCash(float c) {
        if (c>0) {
                d->temp_cash=c;
                m->PayType(2)
        }
}
Cancel() {
        m->Cancel();
}
Super() {
        m->SelectGas(2);
}
Premium() {
        m->SelectGas(3)
}
```

```
Regular() {
        m->SelectGas(1)
}
StartPump() {
        m->StartPump();
}
PumpLiter() {
        if (d->cashL+1)*d->price)
                m->StopPump();
        else m->Pump()
}
Stop() {
        m->StopPump();
}
Receipt() {
        m->Receipt();
}
NoReceipt() {
        m->NoReceipt();
}
```

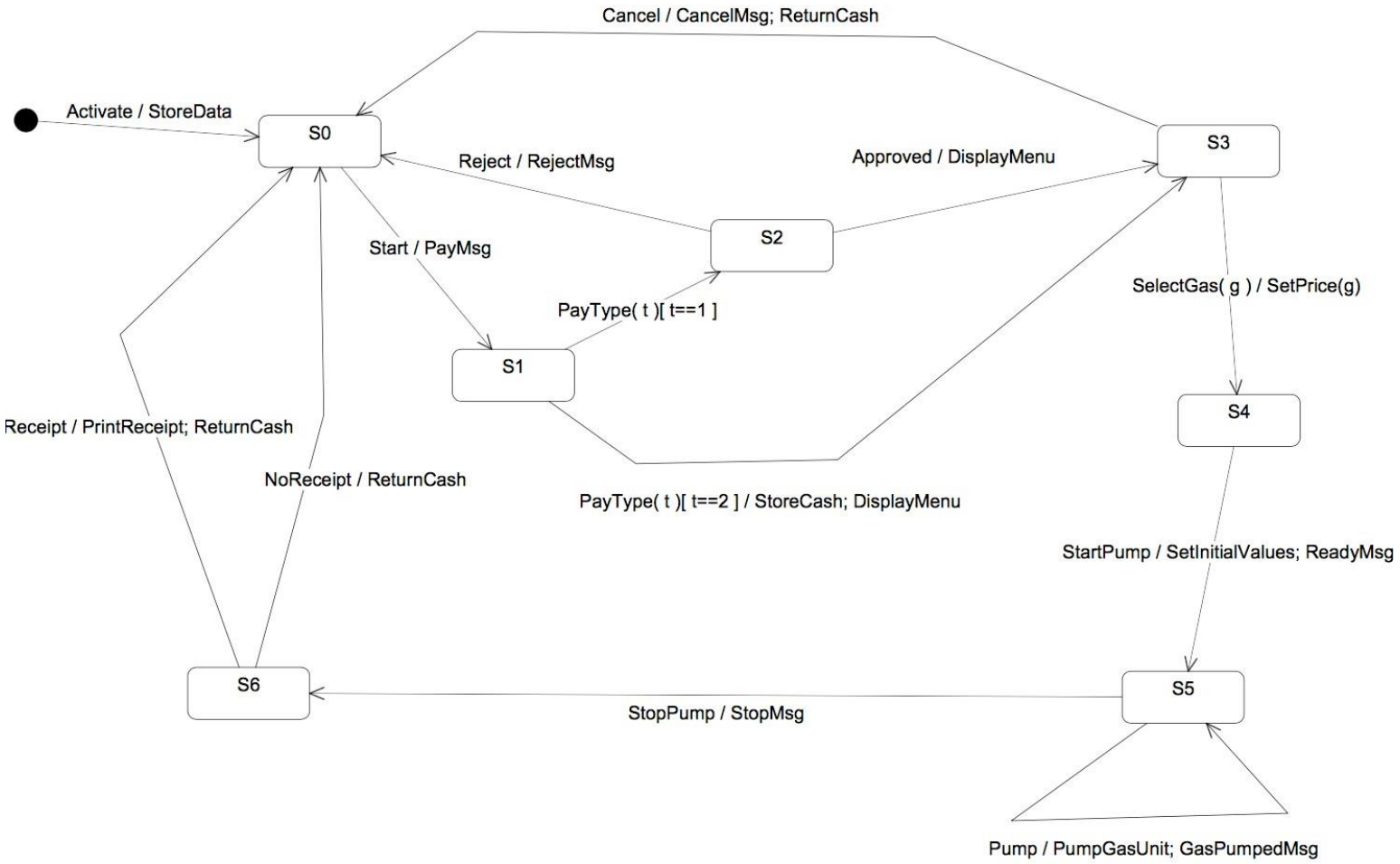
Notice: cash: contains the value of cash deposited

price: contains the price of the selected gas

L: contains the number of liters already pumped cash , L, price are in the data store

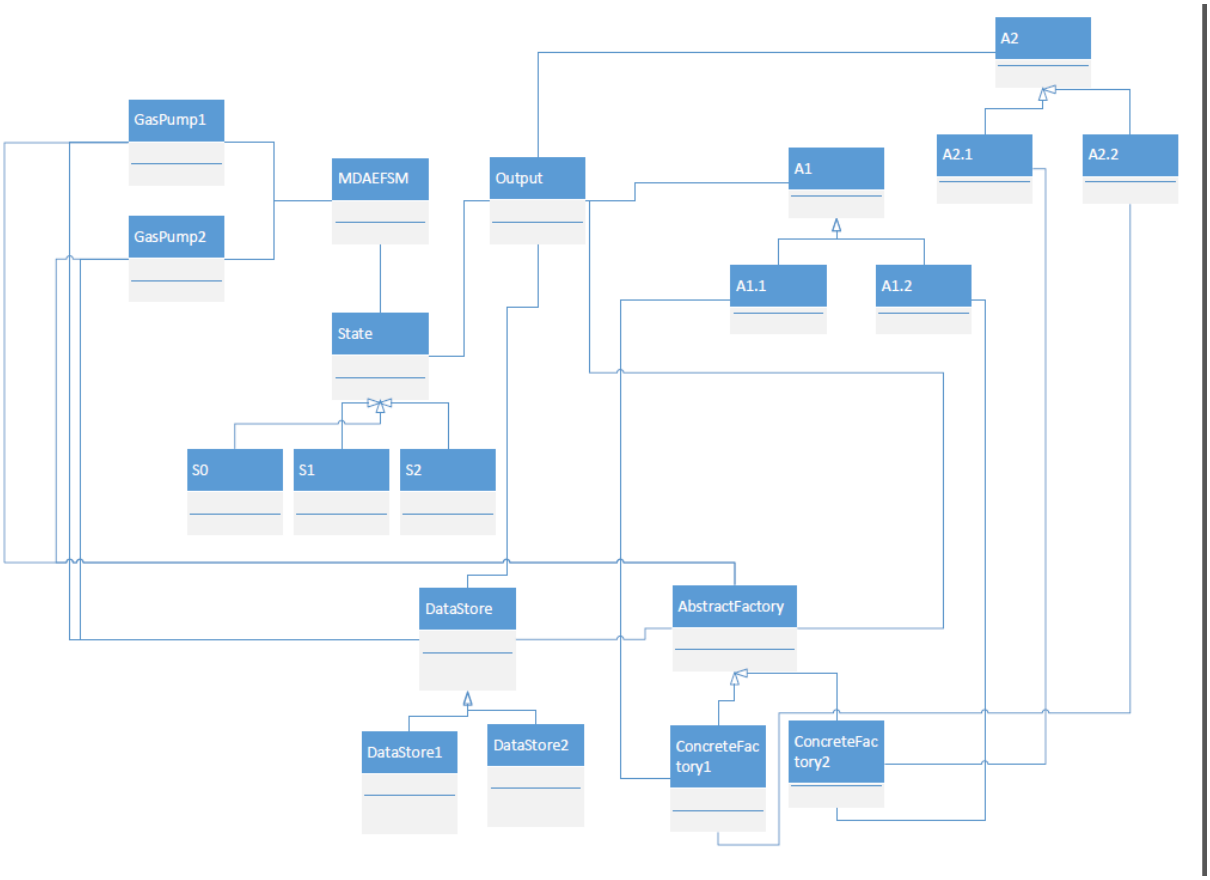
m: is a pointer to the MDA-EFSM object d: is a pointer to the Data Store object

MDAEFSM Diagram



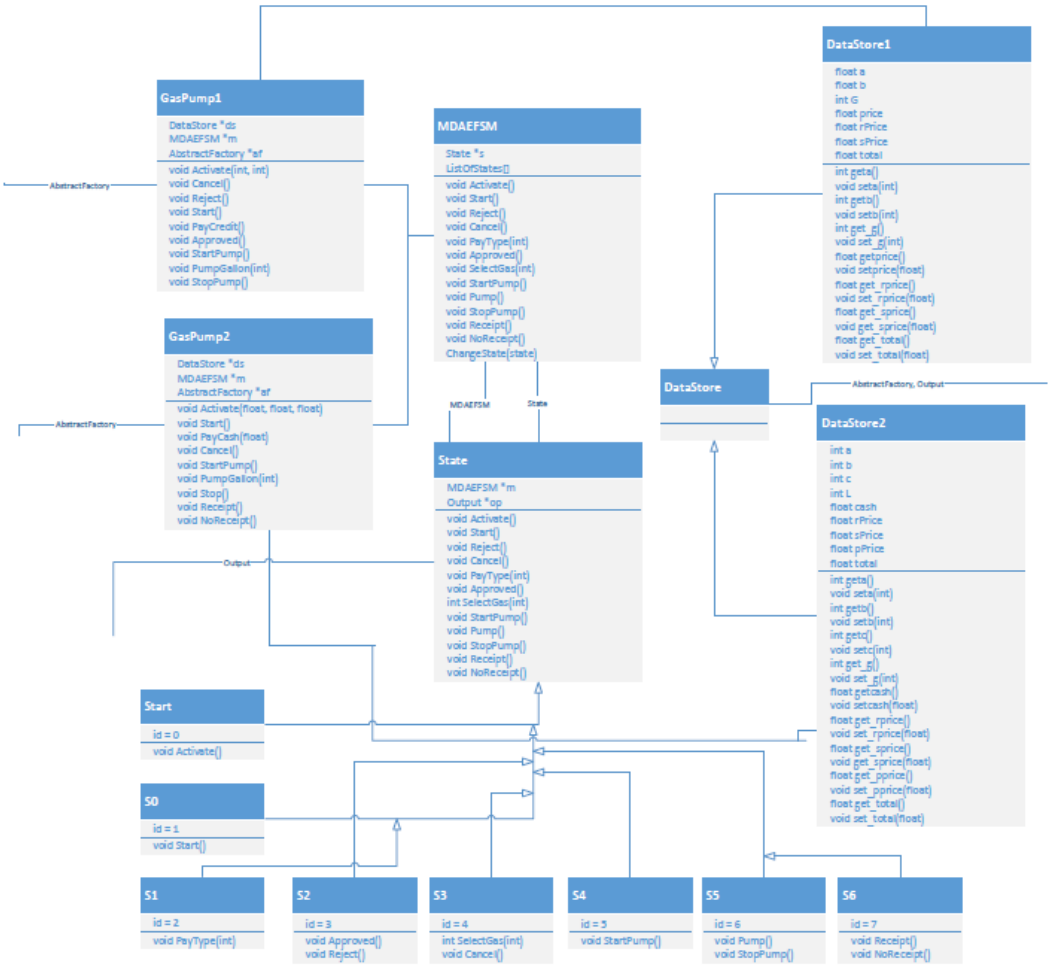
MDA-EFSM for Gas Pumps

High Level Design:



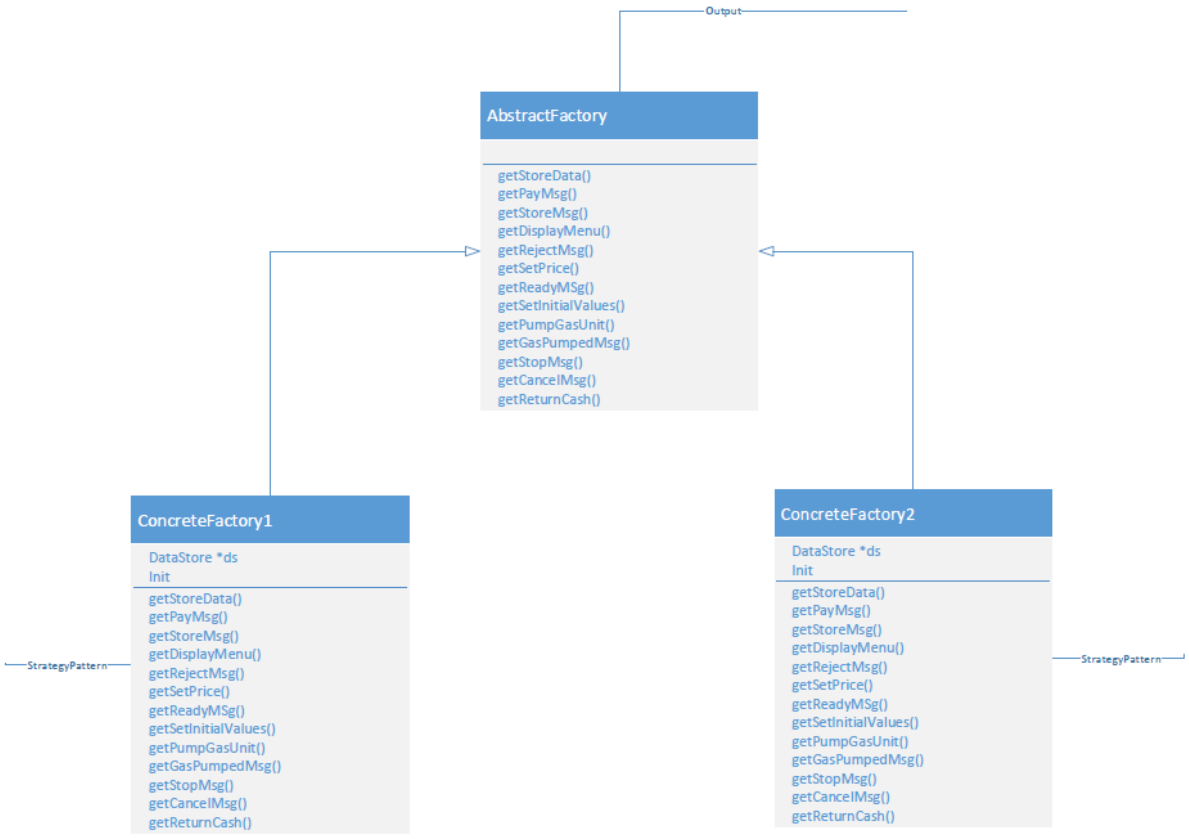
The high level design above, gives overview of how all the classes are interconnected and how three different design patterns are implemented to form a Gas Pump System.

State Design Pattern:



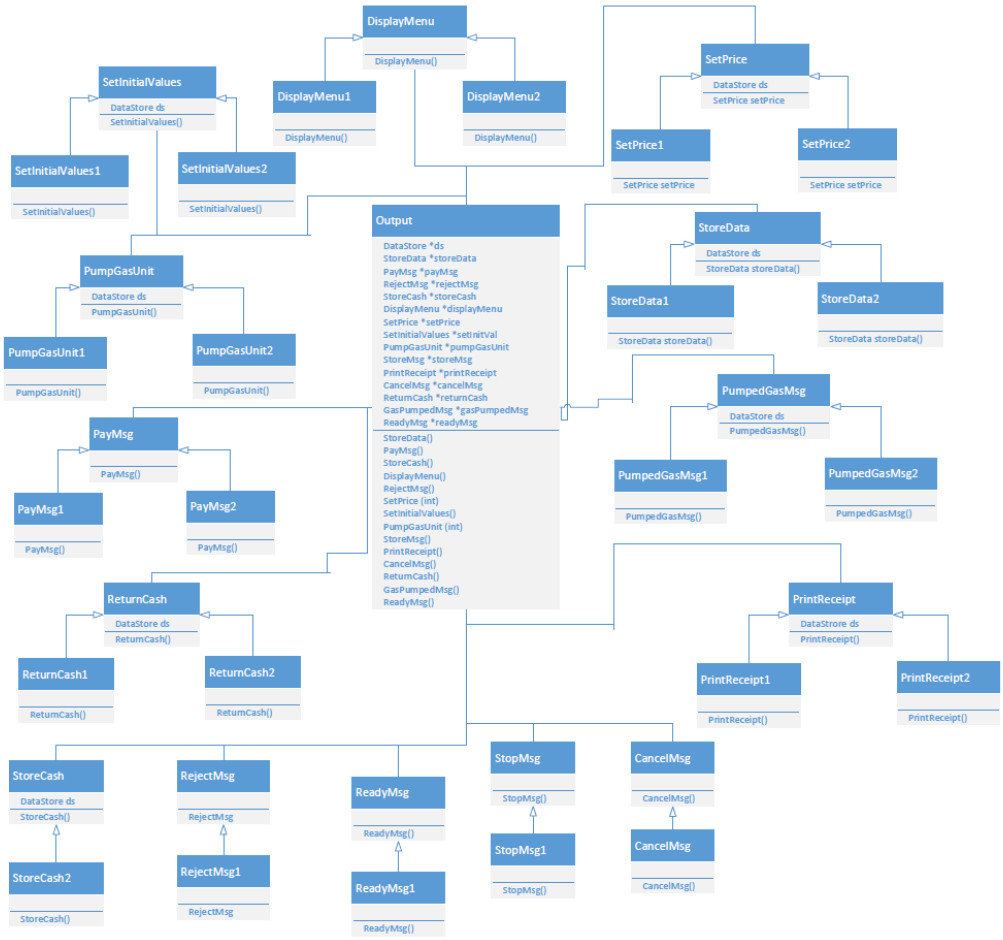
In above state design pattern, decentralized state pattern is implemented. State is an interface class that is inherited by different states from start to S6. Start to S6 are the different states in which different operations are perform and state change message is provided from one state to other directly. GasPump1 and GasPump2 acts as Input processor classes, DataStore class id associated with ConcreteFactory Class which is explained in diffrent diagram.

AbstractFactory Pattern:



In above pattern, abstract AbstractFactory class is created which is inherited by ConcreteFactory1 and ConcreteFactory2. Abstract Factory class help in initializing and choosing Strategy classes as per the requirement.

Strategy Pattern:



In above diagram, strategy pattern is shown. All the actions have abstract base class and actions are implemented according to the choice of GasPump. Strategy classes are invoked through ConcreteFactory class.

Implementation of GasPump:

MainDriver.java class:

This is the main class for our GassPump System. User is given the option to choose either of the classes. On selection of Gas Pump, respective menu of operations is displayed and performed as per the choice of user.

The responsibilities for the functions in **ConcreteFactory** Class are as follows:

Each of the functions like getStoreData, getDisplayMenu helps in choosing the right strategy at runtime. Thus ConcreteFactory class is responsible to choose the right strategy.

getStoreData() : ConcreteFactory creates the object of StoreData() (Strategy pattern implementation) according to the type of gas pump selected. StoreData stores the temporary variables in DataStore.

getDisplayMenu() : ConcreteFactory creates the object of DisplayMenu() (Strategy pattern implementation) according to the type of gas pump selected. DisplayMenu() dipalys the type of gases available for the selected Gas Pump.

getPayMsg() : ConcreteFactory creates the object of PayMsg() (Strategy pattern implementation) according to the type of gas pump selected. It gives the message for the type of payment selected.

getPrintReceipt() : ConcreteFactory creates the object of PrintReceipt() (Strategy pattern implementation) according to the type of gas pump selected. It prints the receipt along with number of gallons or liters of gas pumped along with its total price.

getSetPrice() : ConcreteFactory creates the object of SetPrice() (Strategy pattern implementation) according to the type of gas pump selected. It helps to know the price of selected gas.

getSetInitialValues() : ConcreteFactory creates the object of SetInitialValues() (Strategy pattern implementation) according to the type of gas pump selected. It helps to set the values or gallons or l liters and value of total according to the type of gas pump selected.

getPumpGasUnit() : ConcreteFactory creates the object of PumpGasUnit() (Strategy pattern implementation) according to the type of gas pump selected. Give the Unit of gas pumped. It may be gallons or liters.

getStopMsg() : ConcreteFactory creates the object of StopMsg() (Strategy pattern implementation) according to the type of gas pump selected. Displays stop message for both the gas pumps.

getRejectMsg() : ConcreteFactory creates the object of RejectMsg() (Strategy pattern implementation) according to the type of gas pump selected. RejectMsg is invoked in case the payment type by credit is rejected.

getCancelMsg() : ConcreteFactory creates the object of CancelMsg() (Strategy pattern implementation) according to the type of gas pump selected. CancelMsg is invoked in case the request is cancelled.

getReadyMsg() : ConcreteFactory creates the object of ReadyMsg() (Strategy pattern implementation) according to the type of gas pump selected. ReadyMsg is invoked when gas pump is ready to pump.

getGasPumpedMsg() : ConcreteFactory creates the object of GasPumpedMsg() (Strategy pattern implementation) according to the type of gas pump selected. GasPumpedMsg gives the total amount of gas pumped.

getStoreCash() : ConcreteFactory creates the object of StoreCash() (Strategy pattern implementation) according to the type of gas pump selected. It gets the stored cash amount entered by user when he selects the payment type as cash.

getReturnCash() : ConcreteFactory creates the object of ReturnCash() (Strategy pattern implementation) according to the type of gas pump selected. ReturnCash() calculates the amount that need to be returned.

State Pattern:

I have implemented decentralized state pattern. State class is associates with MDAEFSM class which acts as Context class for state pattern. State class is responsible to give current state and change the transition to next state.

State classes are described as below:

Start: Activate() is invoked in Start state. It activates the gas pump. Transition is changed to state S0.

S0: Start() is invoked in S0 state. It lets the user know the payment mode depending on the type of gas pump selected. Transition is changed to state S1.

S1: PayCash() or PayCredit() is invoked depending on the type of payment method. Transition is changed to S2.

S2: Approved() or Reject() is invoked. Transition is changed to S3 on approval and S0 on rejection.

S3: Cancel() is invoked in case of insufficient cash. Transition changes to S0. SelectGas() is invoked if request does not fail and on selection of gas, transition changes to S4.

S4: StartPump() is invoked in this state. SetInitialValues() and ReadyMsg() is called. Transition changes to S5.

S5: PumpGallon() or PumpLiter() is called depending on the type of gas pump chosen. Transition changes to S6 in case of StopPump(). It remains in State S5 till it is PumpGallon() or PumpLiter().

S6: PrintReceipt(), NoReceipt() and ReturnCash() is invoked depending on type of gas pump chosen and choice of user. Transition changes to S0.

GasPump1:

GasPump1 offers two types of gases: Regular and Super. It gives Payment mode as Credit. GasPump1 has list of events which invokes as required by user and lead to appropriate action. It is associated with MDAEFSM, DataStore and AbstractFactory.

Functions of GasPump1:

Activate(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Start(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

PayCredit(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Reject(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Cancel(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Approved(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

StartPump(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

PumpGallon(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

StopPump(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Regular(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Super(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Receipt(): GasPump1 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

GasPump2:

GasPump2 offers three types of gases: Regular, Super and Premium. It gives Payment mode as Cash. GasPump2 has list of events which invokes as required by user and lead to appropriate action. It is associated with MDAEFSM, DataStore and AbstractFactory.

Functions of GasPump2:

Activate(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Start(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

PayCash(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Reject(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Cancel(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Approved(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

StartPump(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

PumpLiter(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

StopPump(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Regular(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Super(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Premium(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Receipt(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

NoReceipt(): GasPump2 receives the call from MainDriver. It passes the call to MDAEFSM object, which calls it from State class and changes the state depending on the Currentstate.

Class MDAEFSM:

Depending on the CurrentState from state class, MDAEFSM changes the state. It associates with State class, GasPump1, GasPump2 class.

DataStore:

DataStore is an abstract class. DataStore1 stores the data for GasPump1 and is associated with GasPump1, Concretefactory1. It gets and sets the data related to GasPump1. DataStore2 stores the data for GasPump2 and is associated with GasPump2, Concretefactory2. It gets and sets the data related to GasPump2.

Output Class:

All the actions are invoked here depending on the right choice selected according to strategy pattern.

Source Code:

MainDriver.java

```
import java.util.InputMismatchException;
import java.util.Scanner;

import abstractfactory.ConcreteFactory1;
import abstractfactory.ConcreteFactory2;
import gaspump.GasPump1;
import gaspump.GasPump2;
import output.Output;
import statepattern.MDAEFSM;
import statepattern.State;

/**
 * @author shraddha
```

```
 *
 */

//Main Class

public class MainDriver{

	public static void main(String[] args){
		// TODO Auto-generated method stub
			Scanner scan = new Scanner(System.in);
			int input;
			int ch = 0;
			System.out.println("Select GasPump: ");
			System.out.println("1. GasPump1");
			System.out.println("2. GasPump2");
			input = scan.nextInt();

			if(input == 1){                                    //Gas Pump 1
				//Create required objects
				ConcreteFactory1 cf1 = new ConcreteFactory1();
				Output op = new Output(cf1, cf1.getDataStore());
				MDAEFSM mda = new MDAEFSM(op);
				GasPump1 gp1 = new GasPump1(mda, cf1.getDataStore());

				float a, b;
				//Menu of Operations for GasPump1
				System.out.println("	Gas Pump-1 Selected \n");
				System.out.println("	MENU of Operations");
				System.out.println("	0. Activate");
```

```
System.out.println("        1. Start");
System.out.println("        2. PayCredit");
System.out.println("        3. Reject");
System.out.println("        4. Cancel");
System.out.println("        5. Approved");
System.out.println("        6. Regular");
System.out.println("        7. Super");
System.out.println("        8. StartPump");
System.out.println("        9. PumpGallon");
System.out.println("        10. StopPump");
System.out.println("        99. Quit the Program");

while(true){

    if(ch != 5){
        System.out.println("Select Operation: ");
        System.out.println("0-Activate,1-Start,2-PayCredit,3-
Reject,4-Cancel,5-Approved, 8-StartPump, 9-PumpGallon, 10-StopPump, 99-Exit");
    }

    try{
        input = scan.nextInt();
    }
    catch(InputMismatchException ex){
        System.out.println("Invalid Input! Please choose a valid
option");
        continue;
    }
```

```
if (input == 99){ //Exit
        System.out.println("Exit");
        break;
}

ch = input;
switch(ch){
case 0: //Activate
        System.out.println("Operation: Activate ");
        System.out.println("Enter the price of Regular gas: ");
        a = scan.nextFloat();
        System.out.println("Enter the price of Super gas: ");
        b = scan.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: //Cancel
            System.out.println("Operation: Cancel");
            gp1.Cancel();
            break;

case 5: //Approved
            System.out.println("Operation: Approved");
            gp1.Approved();
            break;

case 6: //Gas Type: Regular
            System.out.println("Operation: Regular");
            gp1.Regular();
            break;

case 7: //Gas Type: Super
            System.out.println("Operation: Super");
            gp1.Super();
            break;

case 8: //StartPump
            System.out.println("Operation: StartPump");
            gp1.StartPump();
            break;

case 9: //PumpGallon
            System.out.println("Operation: PumpGallon");
```

```
                    gp1.PumpGallon();
                    break;

                case 10: //StopPump
                    System.out.println("Operation: StopPump");
                    gp1.StopPump();
                    break;

                case 11: //Receipt
                    System.out.println("Operation: Receipt");
                    gp1.Receipt();
                    break;



                default: //Invalid Choice
                    System.out.println("Invalid choice");
                    break;
                }
            }
        }


        else if(input == 2){                    //GasPump2
            //Create required objects
            ConcreteFactory2 cf2 = new ConcreteFactory2();
            Output op = new Output(cf2, cf2.getDataStore());
            MDAEFSM mda = new MDAEFSM(op);
            GasPump2 gp2 = new GasPump2(mda, cf2.getDataStore());
```

```
//Menu of operations for GasPump2
int a, b, c;
System.out.println("      Gas Pump-2 Selected");
System.out.println("      MENU of Operations");
System.out.println("      0. Activate(int)");
System.out.println("      1. Start");
System.out.println("      2. PayCash");
System.out.println("      3. Reject");
System.out.println("      4. Cancel");
System.out.println("      5. Approved");
System.out.println("      9. StartPump");
System.out.println("      10. PumpLiter");
System.out.println("      11. StopPump");
System.out.println("      12. Receipt");
System.out.println("      13. NoReceipt");
System.out.println("      99. Quit the Program");

while(true){
        System.out.println("Select Operation: ");
        System.out.println("0-Activate, 1-Start, 2-PayCash, 3-Reject, 4-
Cancel, 8-StartPump, 9-PumpLiter, 10-StopPump, 11-PrintReceipt, 12-NoReceipt, 99-Exit");
        try{
                input = scan.nextInt();
        }
        catch(InputMismatchException ex){
                System.out.println("Invalid Input! Please choose one of
the above options");
                continue;
        }
```

```
if (input == 99){ //Exit
        System.out.println("Exit");
        break;
}

ch = input;
switch(ch){
case 0: //Activate
        System.out.println("Activate: ");
        System.out.println("Enter the price of Regular gas: ");
        a = scan.nextInt();
        System.out.println("Enter the price of Super gas: ");
        b = scan.nextInt();
        System.out.println("Enter the price of Premium gas: ");
        c = scan.nextInt();
        gp2.Activate(a, b, c);
        break;

case 1: //Start
        System.out.println("Operation: Start");
        gp2.Start();
        break;

case 2: //PayCash
        System.out.println("Operation: PayCash");
        System.out.println("Enter Cash:");
        float cash = scan.nextFloat();
        gp2.PayCash(cash);
```

```
                break;

        case 3: //Reject
                System.out.println("Operation: Reject");
                gp2.Reject();
                break;

        case 4: //Cancel
                System.out.println("Operation: Cancel");
                gp2.Cancel();
                break;

        case 5: //Gas Type: Regular
                System.out.println("Operation: Regular");
                gp2.Regular();
                break;

        case 6: //Gas Type: Super
                System.out.println("Operation: Super");
                gp2.Super();
                break;

        case 7: //Gas Type: Premium
                System.out.println("Operation: Premium");
                gp2.Premium();
                break;

        case 8: //StartPump
                System.out.println("Operation: StartPump");
```

```
                    gp2.StartPump();
                    break;

                case 9: //PumpLiter
                    System.out.println("Operation: PumpLiter");
                    gp2.PumpLiter();
                    break;

                case 10: //StopPump
                    System.out.println("Operation: StopPump");
                    gp2.StopPump();
                    break;

                case 11: //Receipt
                    System.out.println("Operation: Receipt");
                    gp2.Receipt();
                    break;

                case 12: //NoReceipt
                    System.out.println("Operation: NoReceipt");
                    gp2.NoReceipt();

                default: //Invalid Choice
                    System.out.println("Invalid choice");
                    break;
                }
            }

}
```

```
        }

}
```

GasPump1.java

```
package gaspump;

import datastore.DataStore;
import datastore.DataStore1;
import statepattern.MDAEFSM;

/**
 * @author shraddha
 *
 */

//Input Class for GasPump1. All the events are described here.

public class GasPump1 {
        MDAEFSM mda = null;
        DataStore ds = null;

        public GasPump1(MDAEFSM mda, DataStore ds){
                this.mda = mda;
                this.ds = ds;
        }
```

```
public void Activate(float a, float b){
        if(a > 0 && b > 0){
                ((DataStore1)ds).a = a;
                ((DataStore1)ds).b = b;
        }
        mda.Activate();
}

public void Start(){
        mda.Start();;
}

public void PayCredit(){
        mda.PayType(1);
        System.out.println("Waiting for Approval");
}

public void Reject() {
        mda.Reject();
}

public void Cancel() {
        mda.Cancel();
}

public void Approved() {
        mda.Approved();
}
```

```
public void StartPump() {
		mda.StartPump();
}

public void PumpGallon() {
		mda.PumpGallon();
}

public void StopPump() {
		mda.StopPump();
		mda.Receipt();
}

public void Regular() {
		// TODO Auto-generated method stub
		((DataStore1) ds).set_rPrice();
		System.out.println("Regular Gas is Selected.");
		mda.SelectGas(1);
}

public void Super() {
		((DataStore1) ds).set_sPrice();
		System.out.println("Super Gas is Selected");
		mda.SelectGas(2);
}

public void Receipt() {
		System.out.println("*******************************");
		mda.Receipt();
```

```
        }



}
```

GasPump2.java

```
/**
 *
 */
package gaspump;

import datastore.DataStore;
import datastore.DataStore2;
import statepattern.MDAEFSM;

/**
 * @author shraddha
 *
 */

//Input Class for GasPump2. All the events for Gas Pump2 are described here.

public class GasPump2 {
        MDAEFSM mda = null;
        DataStore ds = null;
```

```
public GasPump2(MDAEFSM mda, DataStore ds){
	this.mda = mda;
	this.ds = ds;
}

public void Activate(int a, int b, int c){
	if(a > 0 && b > 0){
		((DataStore2)ds).a = a;
		((DataStore2)ds).b = b;
		((DataStore2)ds).c = c;
	}
	mda.Activate();
}

public void Start(){
	mda.Start();;
}

public void PayCash(float cash){
	if (cash > 0) {
		((DataStore2) ds).cash = cash;
		mda.PayType(2);
	}
}

public void Reject() {
	mda.Reject();
}
```

```
public void Cancel() {
	mda.Cancel();
}

public void Approved() {
	mda.Approved();
}

public void StartPump() {
	mda.StartPump();
}

public void PumpLiter() {
	if(((DataStore2)ds).cash < (((((DataStore2)ds).L) + 1) * ((DataStore2)ds).price) ){
		mda.StopPump();
	}
	else
		mda.PumpLiter();
}

public void StopPump() {
	mda.StopPump();
	//mda.Receipt();
}

public void Regular() {
	// TODO Auto-generated method stub
	((DataStore2) ds).set_rPrice();
	System.out.println("Regular Gas is selected.");
```

```
            mda.SelectGas(1);
        }

        public void Super() {
            ((DataStore2) ds).set_sPrice();
            System.out.println("Super Gas is selected.");
            mda.SelectGas(2);
        }

        public void Premium(){
            ((DataStore2) ds).set_pPrice();
            System.out.println("Premium Gas is selected.");
            mda.SelectGas(3);
        }

        public void Receipt() {
            // TODO Auto-generated method stub
            mda.Receipt();

        }

        public void NoReceipt() {
            // TODO Auto-generated method stub
            mda.NoReceipt();
        }

}
```

AbstractFactory.java

```
package abstractfactory;

import strategypattern.CancelMsg;
import strategypattern.DisplayMenu;
import strategypattern.GasPumpedMsg;
import strategypattern.PayMsg;
import strategypattern.PrintReceipt;
import strategypattern.PumpGasUnit;
import strategypattern.ReadyMsg;
import strategypattern.RejectMsg;
import strategypattern.ReturnCash;
import strategypattern.SetInitialValues;
import strategypattern.SetPrice;
import strategypattern.StopMsg;
import strategypattern.StoreCash;
import strategypattern.StoreData;

/**
 * @author shraddha
 *
 */

public interface AbstractFactory {
	public StoreData getStoreData();
	public DisplayMenu getDisplayMenu();
	public PayMsg getPayMsg();
	public SetPrice getSetPrice();
	public SetInitialValues getSetInitialValues();
```

```
        public PumpGasUnit getPumpGasUnit();
        public PrintReceipt getPrintReceipt();
        public StopMsg getStopMsg();
        public RejectMsg getRejectMsg();
        public CancelMsg getCancelMsg();
        public ReadyMsg getReadyMsg();
        public GasPumpedMsg getGasPumpedMsg();
        public StoreCash getStoreCash();
        public ReturnCash getReturnCash();
}
```

ConcreteFactory1.java

```
package abstractfactory;

/**
 * @author shraddha
 *
 */

import datastore.DataStore;
import datastore.DataStore1;
import strategypattern.*;


public class ConcreteFactory1 implements AbstractFactory {

                DataStore dataStore = new DataStore1();
                StoreData storeData = new StoreData1();
```

```
DisplayMenu displayMenu = new DisplayMenu1();
PayMsg payMsg = new PayMsg1();
SetPrice setPrice = new SetPrice1();
SetInitialValues setInitVal = new SetInitialValues1();
PumpGasUnit pumpGasUnit = new PumpGasUnit1();
PrintReceipt printReceipt = new PrintReceipt1();  //doubt setw
StopMsg stopMsg = new StopMsg1();
RejectMsg rejectMsg = new RejectMsg1();
CancelMsg cancelMsg = new CancelMsg1();
ReadyMsg readyMsg = new ReadyMsg1();
GasPumpedMsg gasPumpedMsg = new GasPumpedMsg1();
ReturnCash returncash = new ReturnCash1();

public DataStore getDataStore(){
        return this.dataStore;
}

@Override
public StoreData getStoreData() {
        // TODO Auto-generated method stub
        return this.storeData;
}

@Override
public DisplayMenu getDisplayMenu() {
        // TODO Auto-generated method stub
        return this.displayMenu;
}
```

```
@Override
public PayMsg getPayMsg() {
	// TODO Auto-generated method stub
	return this.payMsg;
}

@Override
public SetPrice getSetPrice() {
	// TODO Auto-generated method stub
	return this.setPrice;
}

@Override
public SetInitialValues getSetInitialValues() {
	// TODO Auto-generated method stub
	return this.setInitVal;
}

@Override
public PumpGasUnit getPumpGasUnit() {
	// TODO Auto-generated method stub
	return this.pumpGasUnit;
}

@Override
public PrintReceipt getPrintReceipt() {
	// TODO Auto-generated method stub
	return this.printReceipt;
}
```

```
@Override
public StopMsg getStopMsg() {
	// TODO Auto-generated method stub
	return this.stopMsg;
}

@Override
public RejectMsg getRejectMsg() {
	// TODO Auto-generated method stub
	return this.rejectMsg;
}

@Override
public CancelMsg getCancelMsg() {
	// TODO Auto-generated method stub
	return this.cancelMsg;
}

@Override
public ReadyMsg getReadyMsg() {
	// TODO Auto-generated method stub
	return this.readyMsg;
}

@Override
public GasPumpedMsg getGasPumpedMsg() {
	// TODO Auto-generated method stub
	return this.gasPumpedMsg;
```

```
	}

	@Override
	public StoreCash getStoreCash() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public ReturnCash getReturnCash() {
		// TODO Auto-generated method stub
		return this.returncash;
	}


}
```

ConcreteFactory2.java

```
/**
 *
 */
package abstractfactory;

import datastore.DataStore;
import datastore.DataStore2;
import strategypattern.*;
```

```
/**
 * @author shraddha
 *
 */
public class ConcreteFactory2 implements AbstractFactory{
	DataStore dataStore = new DataStore2();
	StoreData storeData = new StoreData2();
	DisplayMenu displayMenu = new DisplayMenu2();
	PayMsg payMsg = new PayMsg2();
	SetPrice setPrice = new SetPrice2();
	SetInitialValues setInitVal = new SetInitialValues2();
	PumpGasUnit pumpGasUnit = new PumpGasUnit2();
	PrintReceipt printReceipt = new PrintReceipt2();  //doubt setw
	StopMsg stopMsg = new StopMsg1();
	RejectMsg rejectMsg = new RejectMsg1();
	CancelMsg cancelMsg = new CancelMsg1();
	ReadyMsg readyMsg = new ReadyMsg1();
	StoreCash storeCash = new StoreCash2();
	GasPumpedMsg gasPumpedMsg = new GasPumpedMsg2();
	ReturnCash returnCash = new ReturnCash2();



	public DataStore getDataStore(){
		return this.dataStore;
	}

	@Override
```

```
public StoreData getStoreData() {
		// TODO Auto-generated method stub
		return this.storeData;
}

@Override
public DisplayMenu getDisplayMenu() {
		// TODO Auto-generated method stub
		return this.displayMenu;
}

@Override
public PayMsg getPayMsg() {
		// TODO Auto-generated method stub
		return this.payMsg;
}

@Override
public SetPrice getSetPrice() {
		// TODO Auto-generated method stub
		return this.setPrice;
}

@Override
public SetInitialValues getSetInitialValues() {
		// TODO Auto-generated method stub
		return this.setInitVal;
}
```

```
@Override
public PumpGasUnit getPumpGasUnit() {
	// TODO Auto-generated method stub
	return this.pumpGasUnit;
}

@Override
public PrintReceipt getPrintReceipt() {
	// TODO Auto-generated method stub
	return this.printReceipt;
}

@Override
public StopMsg getStopMsg() {
	// TODO Auto-generated method stub
	return this.stopMsg;
}

@Override
public RejectMsg getRejectMsg() {
	// TODO Auto-generated method stub
	return this.rejectMsg;
}

@Override
public CancelMsg getCancelMsg() {
	// TODO Auto-generated method stub
	return this.cancelMsg;
}
```

```
	@Override
	public ReadyMsg getReadyMsg() {
		// TODO Auto-generated method stub
		return this.readyMsg;
	}

	@Override
	public GasPumpedMsg getGasPumpedMsg() {
		// TODO Auto-generated method stub
		return this.gasPumpedMsg;
	}

	@Override
	public StoreCash getStoreCash() {
		// TODO Auto-generated method stub
		return this.storeCash;
	}

	@Override
	public ReturnCash getReturnCash() {
		// TODO Auto-generated method stub
		return this.returnCash;
	}
}
```

DataStore.java

```
package datastore;

/**
 * @author shraddha
 *
 */

//Abstract Class for DataStore
public abstract class DataStore {

}
```

DataStore1.java

```
package datastore;

/**
 * @author shraddha
 *
 */

//This class store data of GasPump1

public class DataStore1 extends DataStore{
		// TODO Auto-generated method stub
		public float a;//Temporary variable for regular gas price
		public float b; //Temporary variable for super gas price
		public float rPrice; //Variable for regular gas price
		public float sPrice; //Variable for super gas price
		public int g; //variable for number of gallons of gas
		public float price; //variable set according to selection of gas
		public float total; //store total amount

		public void seta(){
			this.rPrice = a;
		}

		public void setb(){
			this.sPrice = b;
		}

		public void set_rPrice(){
			this.price = a;
		}

		public void set_sPrice(){
			this.price = b;
		}

		public float getPrice(){
			return this.price;
		}

		public int get_g(){
			this.g = this.g + 1;
			return this.g;
		}
```

```
	public void set_g(int g){
		this.g = g;
	}

	public float get_total(){
		return total = price * g;
	}

	public void set_total(float total){
		this.total = total;
	}
}
```

DataStore2.java

```
/**
 *
 */
package datastore;

/**
 * @author shraddha
 *
 */

//This class stores data of GasPump2

public class DataStore2 extends DataStore{
	public int a; //Temporary variable for Regular gas price
	public int b; //Temporary variable for Super gas price
	public int c; //Temporary variable for Premium gas price
	public float cash; //variable to store cash amount
	public float temp_cash; // temporary variable to store cash
	public int L; //variable for number of liters of gas
```

```
public float total; //store total amount

public int rPrice; //variable for Regular gas price

public int sPrice; //variable for Super gas price

public int pPrice; //variable for Premium gas price
/////doubt about set w

public int price; //variable for gas price depending on the selection


public void seta(){

        this.rPrice = a;

}


public void setb(){

        this.sPrice = b;

}


public void setc(){

        this.pPrice = c;

}


public void set_rPrice(){

        this.price = a;

}


public void set_sPrice(){

        this.price = b;

}


public void set_pPrice(){

        this.price = c;
```

```
}

public float getPrice(){
        return this.price;
}

public int get_L(){
        this.L = this.L + 1;
        return this.L;
}

public void set_L(int L){
        this.L = L;
}

public float get_total(){
        return total = this.price * this.L;
}

public void set_total(float total){
        this.total = total;
}

public float getCash(){
        return this.cash;
}

public void setCash(float cash){
        this.cash = this.temp_cash;
```

```
	}
}
```

Output.java

```
package output;

import abstractfactory.*;
import strategypattern.*;
import datastore.DataStore;

/**
 * @author shraddha
 *
 */

//This is output class. All actions are described here.
//Output class associates with abstract factory, datastore and strategy classes.

public class Output {
	AbstractFactory af = null;
	DataStore ds = null;

	public Output(AbstractFactory af, DataStore ds){
		this.af = af;
		this.ds = ds;
	}

	public void StoreData(){
```

```
        StoreData storeData = af.getStoreData();
        storeData.StoreData(ds);
}

public void DisplayMenu(){
        DisplayMenu dispMenu = af.getDisplayMenu();
        dispMenu.DisplayMenu();
}

public void PayMsg(){
        PayMsg payMsg = af.getPayMsg();
        payMsg.PayMsg();
}

public void SetPrice(){
        SetPrice setPrice = af.getSetPrice();
        setPrice.SetPrice(ds);
}

public void SetInitialValues(){
        SetInitialValues setInitVal = af.getSetInitialValues();
        setInitVal.SetInitialValues(ds);
}

public void PumpGasUnit(){
        PumpGasUnit pumpGasUnit = af.getPumpGasUnit();
        pumpGasUnit.PumpGasUnit();
}
```

```
public void PrintReceipt(){
        PrintReceipt printReceipt = af.getPrintReceipt();
        printReceipt.PrintReceipt(ds);
}

public void StopMsg(){
        StopMsg stopMsg = af.getStopMsg();
        stopMsg.StopMsg();
}

public void RejectMsg(){
        RejectMsg rejectMsg = af.getRejectMsg();
        rejectMsg.RejectMsg();
}

public void CancelMsg(){
        CancelMsg cancelMsg = af.getCancelMsg();
        cancelMsg.CancelMsg();
}

public void ReadyMsg(){
        ReadyMsg readyMsg = af.getReadyMsg();
        readyMsg.ReadyMsg();
}

public void GasPumpedMsg(){
        GasPumpedMsg gasPumpedMsg = af.getGasPumpedMsg();
        gasPumpedMsg.GasPumpedMsg(ds);
}
```

```
	public void StoreCash(){
		StoreCash storeCash = af.getStoreCash();
		storeCash.StoreCash(ds);
	}

	public void ReturnCash() {
		ReturnCash returnCash = af.getReturnCash();
		returnCash.ReturnCash(ds);
	}


}
```

MDAEFSM.java

```
package statepattern;

import abstractfactory.AbstractFactory;
import output.Output;

/**
 * @author shraddha
 *
 */

//This is Context class for state pattern implementation
```

```
public class MDAEFSM {
	State s = new Start(this);
	State s0 = new S0(this);
	State s1 = new S1(this);
	State s2 = new S2(this);
	State s3 = new S3(this);
	State s4 = new S4(this);
	State s5 = new S5(this);
	State s6 = new S6(this);

	AbstractFactory af = null;
	Output op = null;
	State state = null;

	public MDAEFSM(Output op){
		state = s;
		this.af = af;
		this.op = op;
	}

	public void Activate(){
		state.Activate();
		printCurrState();
	}

	public void Start(){
		state.Start();
		printCurrState();
	}
```

```
public void PayType(int t){
        if(t == 1)
                state.PayCredit();
        else if(t == 2)
                state.PayCash();
        printCurrState();
}

public void PayCash(){
        state.PayCash();
        printCurrState();
}

public void Reject(){
        state.Reject();
        printCurrState();
}

public void Cancel(){
        state.Cancel();
        printCurrState();
}

public void Approved(){
        state.Approved();
        printCurrState();
}
```

```
public void SelectGas(int g){
        state.SelectGas(g);
        printCurrState();
}

public void StartPump(){
        state.StartPump();
        printCurrState();
}

public void PumpGallon(){
        state.PumpGallon();
        printCurrState();
}

public void PumpLiter(){
        state.PumpLiter();
        printCurrState();
}

public void StopPump(){
        state.StopPump();
        printCurrState();
}

public void Receipt(){
        state.Receipt();
        printCurrState();
}
```

```
public void NoReceipt(){
        state.NoReceipt();
        printCurrState();
}


public void setState(State state){
        this.state = state;
}

public State getStateStart(){
        return s;
}

public State getStateS0(){
        return s0;
}

public State getStateS1(){
        return s1;
}

public State getStateS2(){
        return s2;
}

public State getStateS3(){
        return s3;
```

```
	}

	public State getStateS4(){
		return s4;
	}

	public State getStateS5(){
		return s5;
	}

	public State getStateS6(){
		return s6;
	}

	public void printCurrState(){
		System.out.println("Current State: " + state.getClass().getName());
	}
}
```

State.java

```
/**
 *
 */
package statepattern;

/**
 * @author shraddha
 *
```

```
 */
public interface State {
        public void Activate();
        public void Start();
        public void PayCredit();
        public void Reject();
        public void Cancel();
        public void Approved();
        public void StartPump();
        public void PumpGallon();
        public void StopPump();
        public void Receipt();
        public void SelectGas(int g);
        public void PayCash();
        public void NoReceipt();
        public void PumpLiter();
}
```

Start.java

```
/**
 *
 */
package statepattern;

/**
 * @author shraddha
 *
```

```
 */

// Start state implements activation of gas pump
public class Start implements State{

	MDAEFSM state = null;

	public Start(MDAEFSM m){
		this.state = m;
	}

	@Override
	public void Activate() {
		state.op.StoreData();
		state.setState(state.getStateS0());
	}

	@Override
	public void Start() {
		// TODO Auto-generated method stub

	}

	@Override
	public void PayCredit() {
		// TODO Auto-generated method stub

	}
```

```
@Override
public void Reject() {
	// TODO Auto-generated method stub

}

@Override
public void Cancel() {
	// TODO Auto-generated method stub

}

@Override
public void Approved() {
	// TODO Auto-generated method stub

}

@Override
public void StartPump() {
	// TODO Auto-generated method stub

}

@Override
public void PumpGallon() {
	// TODO Auto-generated method stub

}
```

```
@Override
public void StopPump() {
	// TODO Auto-generated method stub

}

@Override
public void Receipt() {
	// TODO Auto-generated method stub

}

@Override
public void SelectGas(int g) {
	// TODO Auto-generated method stub

}

@Override
public void PayCash() {
	// TODO Auto-generated method stub

}

@Override
public void NoReceipt() {
	// TODO Auto-generated method stub
```

```
	}

	@Override
	public void PumpLiter() {
		// TODO Auto-generated method stub

	}


}
```

S0.java

```
/**
 *
 */
package statepattern;



/**
 * @author shraddha
 *
 */

// State S0 which implements start action

public class S0 implements State {
```

```
MDAEFSM state = null;

public S0(MDAEFSM state) {
	this.state = state;
}

@Override
public void Activate() {
	// TODO Auto-generated method stub

}

@Override
public void Start() {
	// TODO Auto-generated method stub
	state.op.PayMsg();
	state.setState(state.getStateS1());

}

@Override
public void PayCredit() {
	// TODO Auto-generated method stub

}

@Override
public void Reject() {
	// TODO Auto-generated method stub
```

```
}

@Override
public void Cancel() {
	// TODO Auto-generated method stub

}

@Override
public void Approved() {
	// TODO Auto-generated method stub

}

@Override
public void StartPump() {
	// TODO Auto-generated method stub

}

@Override
public void PumpGallon() {
	// TODO Auto-generated method stub

}

@Override
public void StopPump() {
```

```
        // TODO Auto-generated method stub

}

@Override
public void Receipt() {
        // TODO Auto-generated method stub

}

@Override
public void SelectGas(int g) {
        // TODO Auto-generated method stub

}

@Override
public void PayCash() {
        // TODO Auto-generated method stub

}

@Override
public void NoReceipt() {
        // TODO Auto-generated method stub

}

@Override
```

```
	public void PumpLiter() {
		// TODO Auto-generated method stub

	}




}
```

S1.java

```
/**
 *
 */
package statepattern;

/**
 * @author shraddha
 *
 */

// State S1 implements PayCredit or PayCash depending on PaYType option

public class S1 implements State {

	MDAEFSM state = null;

	public S1(MDAEFSM state) {
```

```
		this.state = state;
}

@Override
public void Activate() {
		// TODO Auto-generated method stub

}

@Override
public void Start() {
		// TODO Auto-generated method stub
}

@Override
public void PayCredit() {
		// TODO Auto-generated method stub
		state.setState(state.getStateS2());
}

@Override
public void Reject() {
		// TODO Auto-generated method stub

}

@Override
public void Cancel() {
		// TODO Auto-generated method stub
```

```
}

@Override
public void Approved() {
	// TODO Auto-generated method stub

}

@Override
public void StartPump() {
	// TODO Auto-generated method stub

}

@Override
public void PumpGallon() {
	// TODO Auto-generated method stub

}

@Override
public void StopPump() {
	// TODO Auto-generated method stub

}

@Override
public void Receipt() {
```

```
        // TODO Auto-generated method stub

}

@Override
public void SelectGas(int g) {
        // TODO Auto-generated method stub

}

@Override
public void PayCash() {
        // TODO Auto-generated method stub
        state.op.StoreCash();
        state.op.DisplayMenu();
        state.setState(state.getStateS3());
}

@Override
public void NoReceipt() {
        // TODO Auto-generated method stub

}

@Override
public void PumpLiter() {
        // TODO Auto-generated method stub

}
```

```
}
```

S2.java

```
/**
 *
 */
package statepattern;

/**
 * @author shraddha
 *
 */

//State S2 implements Reject or approved action.

public class S2 implements State {
	MDAEFSM state = null;

	public S2(MDAEFSM state) {
		this.state = state;
	}

	@Override
	public void Activate() {
		// TODO Auto-generated method stub
```

```
}

@Override
public void Start() {
	// TODO Auto-generated method stub

}

@Override
public void PayCredit() {
	// TODO Auto-generated method stub

}

@Override
public void Reject() {
	// TODO Auto-generated method stub
	state.op.RejectMsg();
	state.setState(state.getStateS0());
}

@Override
public void Cancel() {
	// TODO Auto-generated method stub

}

@Override
public void Approved() {
```

```
		// TODO Auto-generated method stub
		state.op.DisplayMenu();
		state.setState(state.getStateS3());

	}

	@Override
	public void StartPump() {
		// TODO Auto-generated method stub

	}

	@Override
	public void PumpGallon() {
		// TODO Auto-generated method stub

	}

	@Override
	public void StopPump() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Receipt() {
		// TODO Auto-generated method stub

	}
```

```
	@Override
	public void SelectGas(int g) {
		// TODO Auto-generated method stub

	}

	@Override
	public void PayCash() {
		// TODO Auto-generated method stub

	}

	@Override
	public void NoReceipt() {
		// TODO Auto-generated method stub

	}

	@Override
	public void PumpLiter() {
		// TODO Auto-generated method stub

	}


}
```

S3.java

```
/**
 *
 */
package statepattern;

/**
 * @author shraddha
 *
 */

//State S3 implements cacellation of request

public class S3 implements State {
	MDAEFSM state = null;

	public S3(MDAEFSM state) {
		this.state = state;
	}

	@Override
	public void Activate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Start() {
		// TODO Auto-generated method stub
```

```
}

@Override
public void PayCredit() {
	// TODO Auto-generated method stub

}

@Override
public void Reject() {
	// TODO Auto-generated method stub

}

@Override
public void Cancel() {
	// TODO Auto-generated method stub
	state.op.CancelMsg();
	state.op.ReturnCash();
	state.setState(state.getStateS0());
}
```

```
@Override
public void Approved() {
	// TODO Auto-generated method stub

}

@Override
public void StartPump() {
	// TODO Auto-generated method stub

}

@Override
public void PumpGallon() {
	// TODO Auto-generated method stub

}
```

```
@Override
public void StopPump() {
	// TODO Auto-generated method stub

}

@Override
public void Receipt() {
	// TODO Auto-generated method stub

}

@Override
public void SelectGas(int g) {
	// TODO Auto-generated method stub
	state.op.SetPrice();
	state.setState(state.getStateS4());
}
```

```
	@Override
	public void PayCash() {
		// TODO Auto-generated method stub

	}



	@Override
	public void NoReceipt() {
		// TODO Auto-generated method stub

	}



	@Override
	public void PumpLiter() {
		// TODO Auto-generated method stub

	}

}
```

S4.java

```
/**
```

```
 *
 */
package statepattern;

/**
 * @author shraddha
 *
 */
public class S4 implements State {
	MDAEFSM state = null;

	public S4(MDAEFSM state) {
		this.state = state;
	}

	@Override
	public void Activate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Start() {
		// TODO Auto-generated method stub

	}

	@Override
	public void PayCredit() {
```

```
        // TODO Auto-generated method stub

}

@Override
public void Reject() {
        // TODO Auto-generated method stub

}

@Override
public void Cancel() {
        // TODO Auto-generated method stub

}

@Override
public void Approved() {
        // TODO Auto-generated method stub

}

@Override
public void StartPump() {
        // TODO Auto-generated method stub
        state.op.SetInitialValues();
        state.op.ReadyMsg();
        state.setState(state.getStateS5());
}
```

```
@Override
public void PumpGallon() {
	// TODO Auto-generated method stub

}

@Override
public void StopPump() {
	// TODO Auto-generated method stub

}

@Override
public void Receipt() {
	// TODO Auto-generated method stub

}

@Override
public void SelectGas(int g) {
	// TODO Auto-generated method stub

}

@Override
public void PayCash() {
	// TODO Auto-generated method stub
```

```
	}

	@Override
	public void NoReceipt() {
		// TODO Auto-generated method stub

	}

	@Override
	public void PumpLiter() {
		// TODO Auto-generated method stub

	}


}
```

S5.java

```
/**
 *
 */
package statepattern;

/**
 * @author shraddha
 *
 */
public class S5 implements State {
```

```
MDAEFSM state = null;

public S5(MDAEFSM state) {
	this.state = state;
}

@Override
public void Activate() {
	// TODO Auto-generated method stub

}

@Override
public void Start() {
	// TODO Auto-generated method stub

}

@Override
public void PayCredit() {
	// TODO Auto-generated method stub

}

@Override
public void Reject() {
	// TODO Auto-generated method stub

}
```

```
@Override
public void Cancel() {
	// TODO Auto-generated method stub

}

@Override
public void Approved() {
	// TODO Auto-generated method stub

}

@Override
public void StartPump() {
	// TODO Auto-generated method stub

}

@Override
public void PumpGallon() {
	// TODO Auto-generated method stub
	state.op.PumpGasUnit();
	state.op.GasPumpedMsg();
	state.setState(state.getStateS5());
}

@Override
public void StopPump() {
```

```
        // TODO Auto-generated method stub
        state.op.StopMsg();
        state.setState(state.getStateS6());
}

@Override
public void Receipt() {
        // TODO Auto-generated method stub

}

@Override
public void SelectGas(int g) {
        // TODO Auto-generated method stub

}

@Override
public void PayCash() {
        // TODO Auto-generated method stub

}

@Override
public void NoReceipt() {
        // TODO Auto-generated method stub

}
```

```
	@Override
	public void PumpLiter() {
		// TODO Auto-generated method stub
		state.op.PumpGasUnit();
		state.op.GasPumpedMsg();
		state.setState(state.getStateS5());
	}



}
```

S6.java

```
/**
 *
 */
package statepattern;

/**
 * @author shraddha
 *
 */
public class S6 implements State {
	MDAEFSM state = null;

	public S6(MDAEFSM state) {
		this.state = state;
	}
```

```
@Override
public void Activate() {
	// TODO Auto-generated method stub

}

@Override
public void Start() {
	// TODO Auto-generated method stub

}

@Override
public void PayCredit() {
	// TODO Auto-generated method stub

}

@Override
public void Reject() {
	// TODO Auto-generated method stub

}

@Override
public void Cancel() {
	// TODO Auto-generated method stub
```

```
}

@Override
public void Approved() {
	// TODO Auto-generated method stub

}

@Override
public void StartPump() {
	// TODO Auto-generated method stub

}

@Override
public void PumpGallon() {
	// TODO Auto-generated method stub

}

@Override
public void StopPump() {
	// TODO Auto-generated method stub



}

@Override
public void Receipt() {
```

```
		// TODO Auto-generated method stub
		state.op.PrintReceipt();
		state.op.ReturnCash();
		state.setState(state.getStateS0());
}

@Override
public void SelectGas(int g) {
		// TODO Auto-generated method stub

}

@Override
public void PayCash() {
		// TODO Auto-generated method stub

}

@Override
public void NoReceipt() {
		// TODO Auto-generated method stub
		state.op.ReturnCash();
		state.setState(state.getStateS0());
}

@Override
public void PumpLiter() {
		// TODO Auto-generated method stub
```

```
        }


}
```

CancelMsg.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public abstract class CancelMsg {
        public abstract void CancelMsg();
}
```

CancelMsg1.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public class CancelMsg1 extends CancelMsg {
        public void CancelMsg(){
                System.out.println("Request Cancelled.");
        }
}
```

DisplayMenu.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public abstract class DisplayMenu {
	public abstract void DisplayMenu();
}
```

DisplayMenu1.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public class DisplayMenu1 extends DisplayMenu {
	public void DisplayMenu(){
		System.out.println("Select Gas Type");
		System.out.println("6. Regular");
		System.out.println("7. Super");
	}

}
```

DisplayMenu2.java

```
/**
 *
 */
package strategypattern;

/**
 * @author shraddha
 *
 */
public class DisplayMenu2 extends DisplayMenu {




	public void DisplayMenu(){
		System.out.println("Select Gas Type");
		System.out.println("5. Regular");
		System.out.println("6. Super");
		System.out.println("7. Premium");
	}

}
```

GasPumpedMsg.java

```
package strategypattern;

import datastore.DataStore;
```

```
/**
 * @author shraddha
 *
 */

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

GasPumpedMsg1.java

```
package strategypattern;

import datastore.DataStore;
import datastore.DataStore1;
import datastore.DataStore2;

/**
 * @author shraddha
 *
 */

public class GasPumpedMsg1 extends GasPumpedMsg {
	public void GasPumpedMsg(DataStore ds){
		System.out.println(((DataStore1)ds).get_g() + " Gallon(s) Pumped.");
	}
}
```

GasPumpedMsg2.java

```
package strategypattern;

import datastore.DataStore;
import datastore.DataStore1;
import datastore.DataStore2;

/**
 * @author shraddha
 *
 */

	public class GasPumpedMsg2 extends GasPumpedMsg {
		public void GasPumpedMsg(DataStore ds){
			System.out.println(((DataStore2)ds).get_L() + " liter(s) Pumped.");
		}

}
```

PayMsg.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public abstract class PayMsg {
	public abstract void PayMsg();
```

```
}
```

PayMsg1.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public class PayMsg1 extends PayMsg {
	public void PayMsg(){
		System.out.println("Pay by Credit");
	}

}
```

PayMsg2.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */
public class PayMsg2 extends PayMsg {

	public void PayMsg(){
		System.out.println("Pay by Cash");
	}
```

```
}
```

PrintReceipt.java

```
package strategypattern;

import datastore.DataStore;

/**
 * @author shraddha
 *
 */

public abstract class PrintReceipt {
        public abstract void PrintReceipt(DataStore ds);
}
```

PrintReceipt1.java

```
package strategypattern;

import datastore.DataStore;
import datastore.DataStore1;

/**
 * @author shraddha
 *
 */
```

```
public class PrintReceipt1 extends PrintReceipt {
	public void PrintReceipt(DataStore ds){
		int gallons = ((DataStore1)ds).get_g()- 1;
		float total = ((DataStore1)ds).get_total() - ((DataStore1)ds).price;
		System.out.println("Receipt: Gas Pump 1");
		System.out.println("Gas Pumped: " + gallons + "Gallon(s)");
		System.out.println("Total Amount: $" + total);
	}

}
```

PrintReceipt2.java

```
/**
 *
 */
package strategypattern;

import datastore.DataStore;
import datastore.DataStore1;
import datastore.DataStore2;

/**
 * @author shraddha
 *
 */
public class PrintReceipt2 extends PrintReceipt {
```

```
	public void PrintReceipt(DataStore ds){
		int liters = ((DataStore2)ds).get_L() - 1;
		float total = ((DataStore2)ds).get_total() - ((DataStore2)ds).price;
		System.out.println("Receipt: Gas Pump 2");
		System.out.println("Gas Pumped: " + liters + " Liter(s)");
		System.out.println("Total Amount: $" + total);
	}

}
```

PumpGasUnit.java

```
package strategypattern;

import datastore.DataStore;

/**
 * @author shraddha
 *
 */

public abstract class PumpGasUnit {
	public abstract void PumpGasUnit();
}
```

PumpGasUnit1.java

```
package strategypattern;

import datastore.DataStore;
```

```
import datastore.DataStore1;

/**
 * @author shraddha
 *
 */

public class PumpGasUnit1 extends PumpGasUnit {
	public void PumpGasUnit(){
		System.out.println("Gas Pump 1: Gallon(s) Pumped");
	}



}
```

PumpGasUnit2.java

```
/**
 *
 */
package strategypattern;

import datastore.DataStore;
import datastore.DataStore2;

/**
 * @author shraddha
 *
 */
```

```
public class PumpGasUnit2 extends PumpGasUnit {

	public void PumpGasUnit(){
		System.out.println("Gas Pump 2: Liters");
	}

}
```

ReadyMsg.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public abstract class ReadyMsg {
	public abstract void ReadyMsg();
}
```

ReadyMsg1.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public class ReadyMsg1 extends ReadyMsg {
```

```
	public void ReadyMsg(){
		System.out.println("Gas Pump is Ready.");
	}
}
```

RejectMsg.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public abstract class RejectMsg {
	public abstract void RejectMsg();
}
```

RejectMsg1.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public class RejectMsg1 extends RejectMsg {
	public void RejectMsg(){
		System.out.println("Card is Rejected.");
	}
}
```

ReturnCash.java

```
/**
 *
 */
package strategypattern;

import datastore.DataStore;

/**
 * @author shraddha
 *
 */
public abstract class ReturnCash {

        public abstract void ReturnCash(DataStore ds);
}
```

ReturnCash1.java

```
package strategypattern;

import datastore.DataStore;

/**
 * @author shraddha
 *
 */

public class ReturnCash1 extends ReturnCash {
```

```
	@Override
	public void ReturnCash(DataStore ds) {
		// TODO Auto-generated method stub

	}

}
```

ReturnCash2.java

```
/**
 *
 */
package strategypattern;

import datastore.DataStore;
import datastore.DataStore2;

/**
 * @author shraddha
 *
 */
public class ReturnCash2 extends ReturnCash{

	@Override
	public void ReturnCash(DataStore ds) {
		float total = ((DataStore2)ds).get_total() - ((DataStore2)ds).price;
		float balance = ((DataStore2)ds).getCash() - total;
		System.out.println("Returning: $" + balance + " to customer");
	}
```

```
}
```

SetInitialValues.java

```
package strategypattern;

import datastore.DataStore;

/**
 * @author shraddha
 *
 */

public abstract class SetInitialValues {
        public abstract void SetInitialValues(DataStore ds);
}
```

SetInitialValues1.java

```
package strategypattern;

import datastore.DataStore1;
import datastore.DataStore;

/**
 * @author shraddha
 *
 */
```

```
public class SetInitialValues1 extends SetInitialValues {


	public void SetInitialValues(DataStore ds){
		((DataStore1)ds).set_g(0);
		((DataStore1)ds).set_total(0);
	}

}
```

SetInitialValues2.java

```
/**
 *
 */
package strategypattern;

import datastore.DataStore;
import datastore.DataStore2;

/**
 * @author shraddha
 *
 */
public class SetInitialValues2 extends SetInitialValues {

	/* (non-Javadoc)
	 * @see strategypattern.SetInitialValues#SetInitialValues(datastore.DataStore)
```

```
	 */
	@Override
	public void SetInitialValues(DataStore ds) {
		// TODO Auto-generated method stub

	}

	public void SetInitialValues2(DataStore ds){
		((DataStore2)ds).set_L(0);
		((DataStore2)ds).set_total(0);;
	}
}
```

SetPrice.java

```
package strategypattern;

import datastore.DataStore;

/**
 * @author shraddha
 *
 */

public abstract class SetPrice {
	public abstract void SetPrice(DataStore ds);
}
```

SetPrice1.java

```
package strategypattern;

import datastore.DataStore;
import datastore.DataStore1;

/**
 * @author shraddha
 *
 */

public class SetPrice1 extends SetPrice {

	public void SetPrice(DataStore ds){
		System.out.println("Gas Pump 1: Gas Price: " + ((DataStore1) ds).getPrice());
	}

}
```

SetPrice2.java

```
/**
 *
 */
package strategypattern;

import datastore.DataStore;
import datastore.DataStore2;

/**
```

```
 * @author shraddha
 *
 */
public class SetPrice2 extends SetPrice {

	public void SetPrice(DataStore ds){
		System.out.println("Gas Pump 2: Gas Price: " + ((DataStore2) ds).price);
	}

}
```

StopMsg.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

public abstract class StopMsg {
	public abstract void StopMsg();
}
```

StopMsg1.java

```
package strategypattern;

/**
 * @author shraddha
 *
```

```
 */

public class StopMsg1 extends StopMsg {

	@Override
	public void StopMsg() {
		// TODO Auto-generated method stub
		System.out.println("Gas Pump stopped.");
	}

}
```

StoreCash.java

```
/**
 *
 */
package strategypattern;

import datastore.DataStore;

/**
 * @author shraddha
 *
 */
public abstract class StoreCash {
	public abstract void StoreCash(DataStore ds);
}
```

StoreCash2.java

```
/**
 *
 */
package strategypattern;

import datastore.DataStore;
import datastore.DataStore2;



/**
 * @author shraddha
 *
 */
public class StoreCash2 extends StoreCash{

	@Override
	public void StoreCash(DataStore ds) {
		System.out.println("Entered Cash: " + ((DataStore2)ds).getCash());
		// TODO Auto-generated method stub

	}

}
```

StoreData.java

```
package strategypattern;

import datastore.DataStore;
```

```
/**
 * @author shraddha
 *
 */

public abstract class StoreData {
	public abstract void StoreData(DataStore ds);

}
```

StoreData1.java

```
package strategypattern;

/**
 * @author shraddha
 *
 */

import datastore.DataStore;
import datastore.DataStore1;

public class StoreData1 extends StoreData{

	public void StoreData(DataStore ds){
		((DataStore1)ds).seta();
		((DataStore1)ds).setb();
	}
}
```

StoreData2.java

```
/**
 *
 */
package strategypattern;
import datastore.DataStore;
import datastore.DataStore2;

/**
 * @author shraddha
 *
 */
public class StoreData2 extends StoreData {

	public void StoreData(DataStore ds){
		((DataStore2)ds).seta();
		((DataStore2)ds).setb();
		((DataStore2)ds).setc();
	}
}
```