Goal:

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.

Description of the Project:

There are two *GasPump* components: a)*GasPump-1* b)*GasPump-2*

The **GasPump-1** component supports the following operations:

Activate (float a, float b) // the gas pump is activated where a is the price of the Regular gas
// and b is the price of Super gas per gallon

Start() //start the transaction

PayCredit() // pay for gas by a credit card

Reject() // credit card is rejected

Cancel() // cancel the transaction

Approved() // credit card is approved

Super() // Super gas is selected

Regular() // Regular gas is selected

StartPump() // start pumping gas

PumpGallon() // one gallon of gas is disposed

StopPump() // stop pumping gas

The **GasPump-2** component supports the following operations:

Activate (int a, int b, int c) // the gas pump is activated where a is the price of Regular gas, b is
//the price of Premium gas and c is the price of Super gas per liter

Start() //start the transaction

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

Cancel() // cancel the transaction

Premium() // Premium gas is selected

Regular() // Regular gas is selected

Super() // Super gas is selected

StartPump() // start pumping gas

PumpLiter() // one liter of gas is disposed

Stop() // stop pumping gas

Receipt() // Receipt is requested

NoReceipt() // No receipt

Both GasPump components are state-based components and are used to control simple gas pumps.

Users can pay by cash or a credit 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.

Aspects that vary between two GasPump components:

a. Types of gasoline disposed

b. Types of payment

c. Display menu(s)

d. Messages

e. Receipts

f. Operation names and signatures

g. Data types

The design makes use of the following patterns:

a. State pattern
b. Strategy pattern
c. Abstract factory pattern

All these patterns will be discussed in detail during the course of this report.

MDA-EFSM model for the Gas Components:

A list of events for the MDA-EFSM :

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

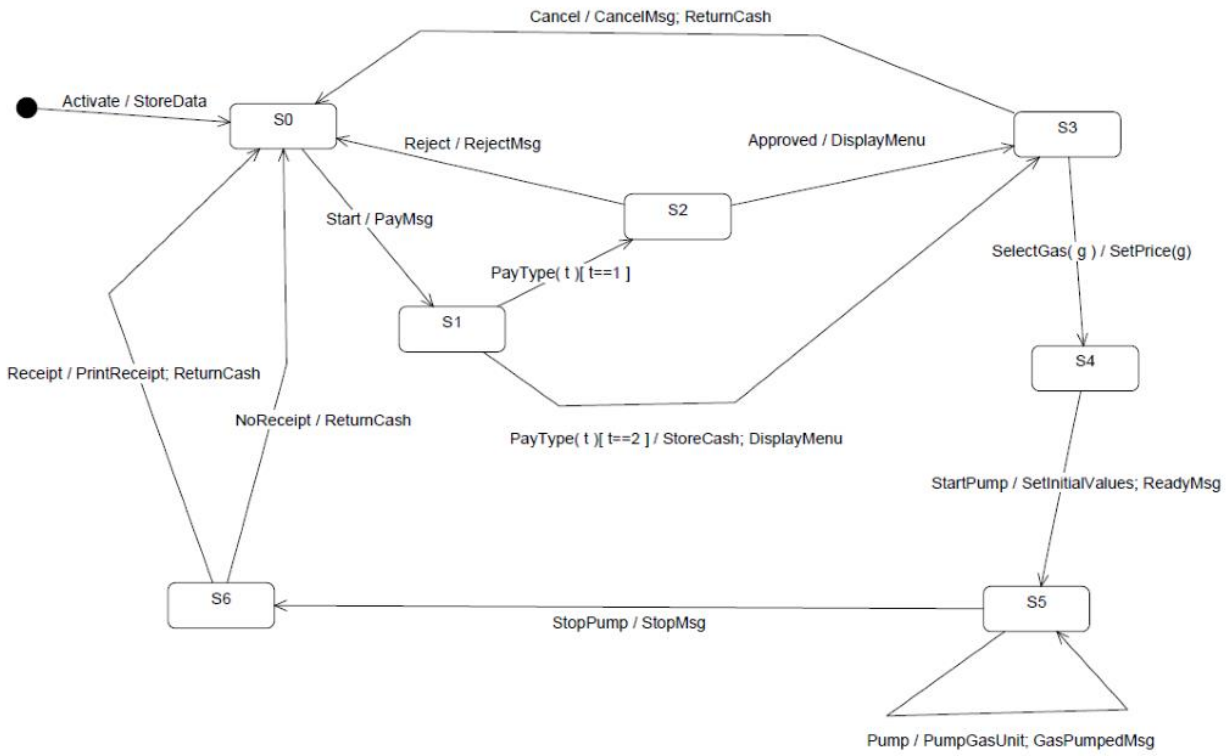
StopMsg // stop pump message and receipt? msg (optionally)

PrintReceipt // print a receipt

CancelMsg // displays a cancellation message

ReturnCash // returns the remaining cash

A State diagram for the MDA EFSM is as shown



MDA-EFSM for Gas Pumps

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->cash<(d->L+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

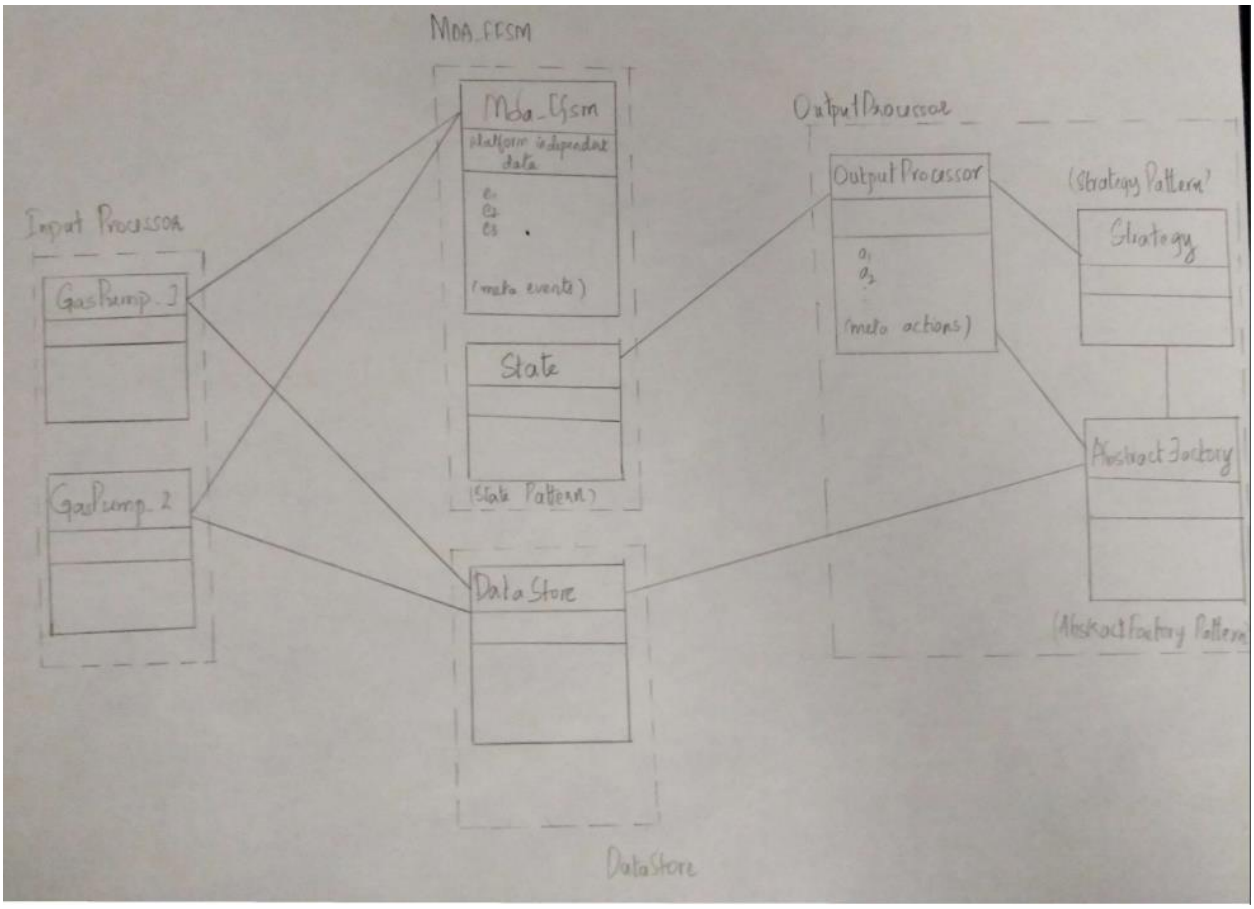
L: contains the number of liters already pumped

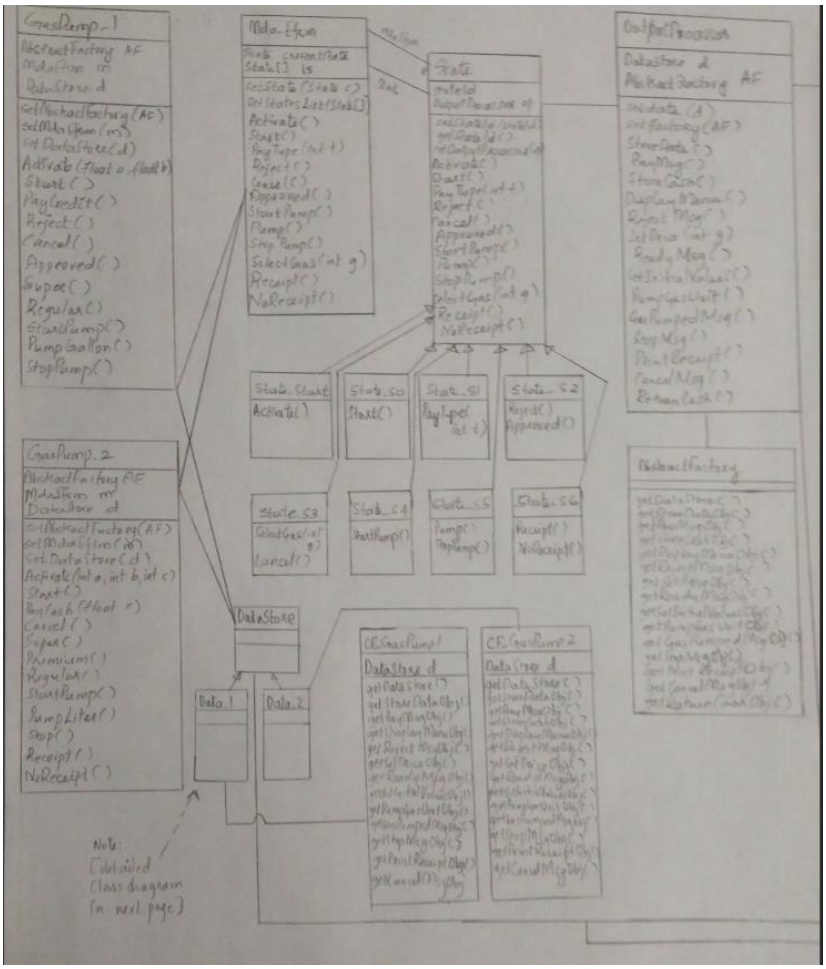
cash , L, price are in the data store

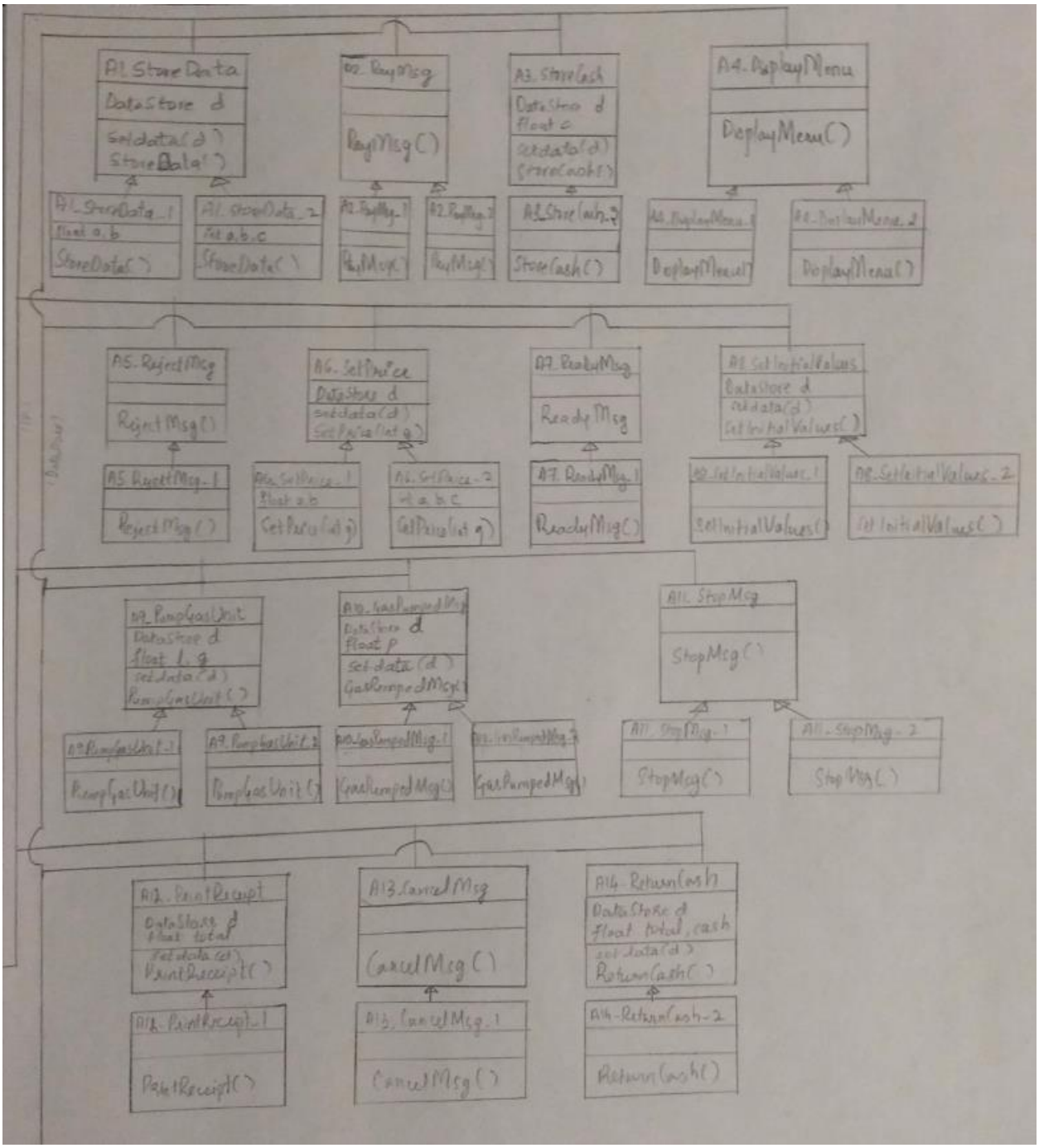
m: is a pointer to the MDA-EFSM object

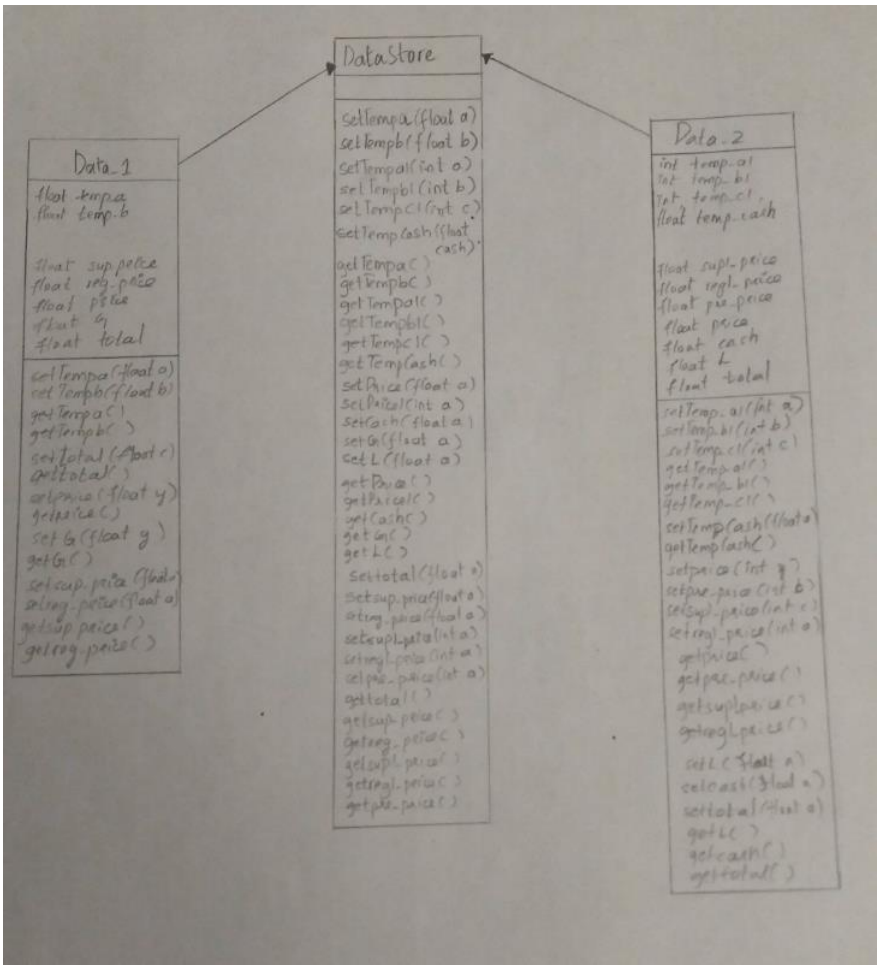
d: is a pointer to the Data Store object

Class Diagram:

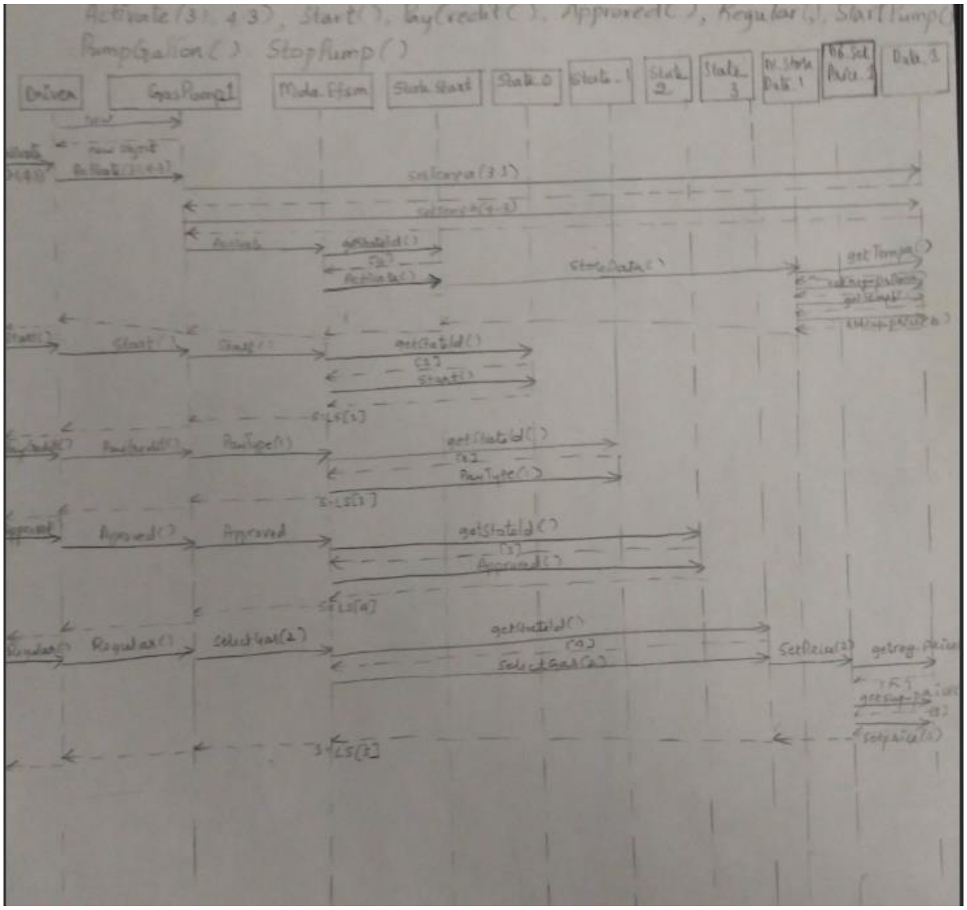

MDA_FFSM
Mda_Ffsm
platform is dependant data
(meta events)
State
(State Pattern)
DataStore
DataStore
Input Processor
GasPump_1
GasPump_2
OutputProcessor
Output Processor
(meta actions)
(Strategy Pattern)
Strategy
Abstract Factory
(Abstract Factory Pattern)

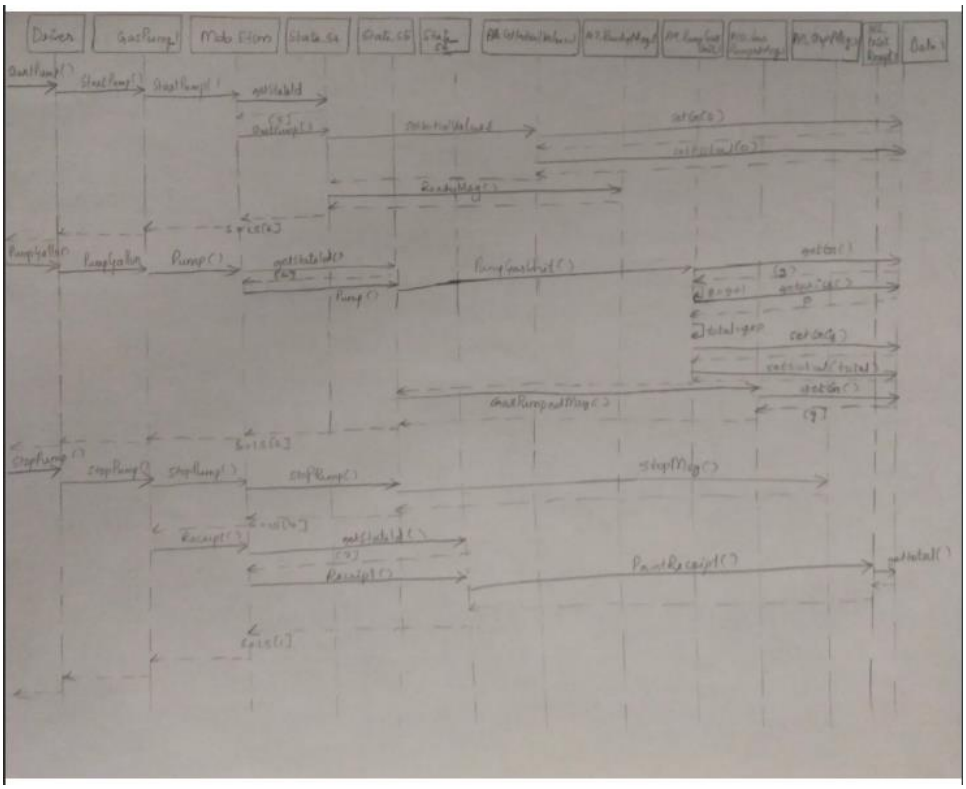

GasPump_1
Media_Item
State
OutputProcessor
State_Start
State_S0
State_S1
State_S2
GasPump_2
AbstractFactory
DataStore
Data_1
Data_2
CF_GasPump_1
CF_GasPump_2
Note:
Detailed
Class diagram
(in next page)

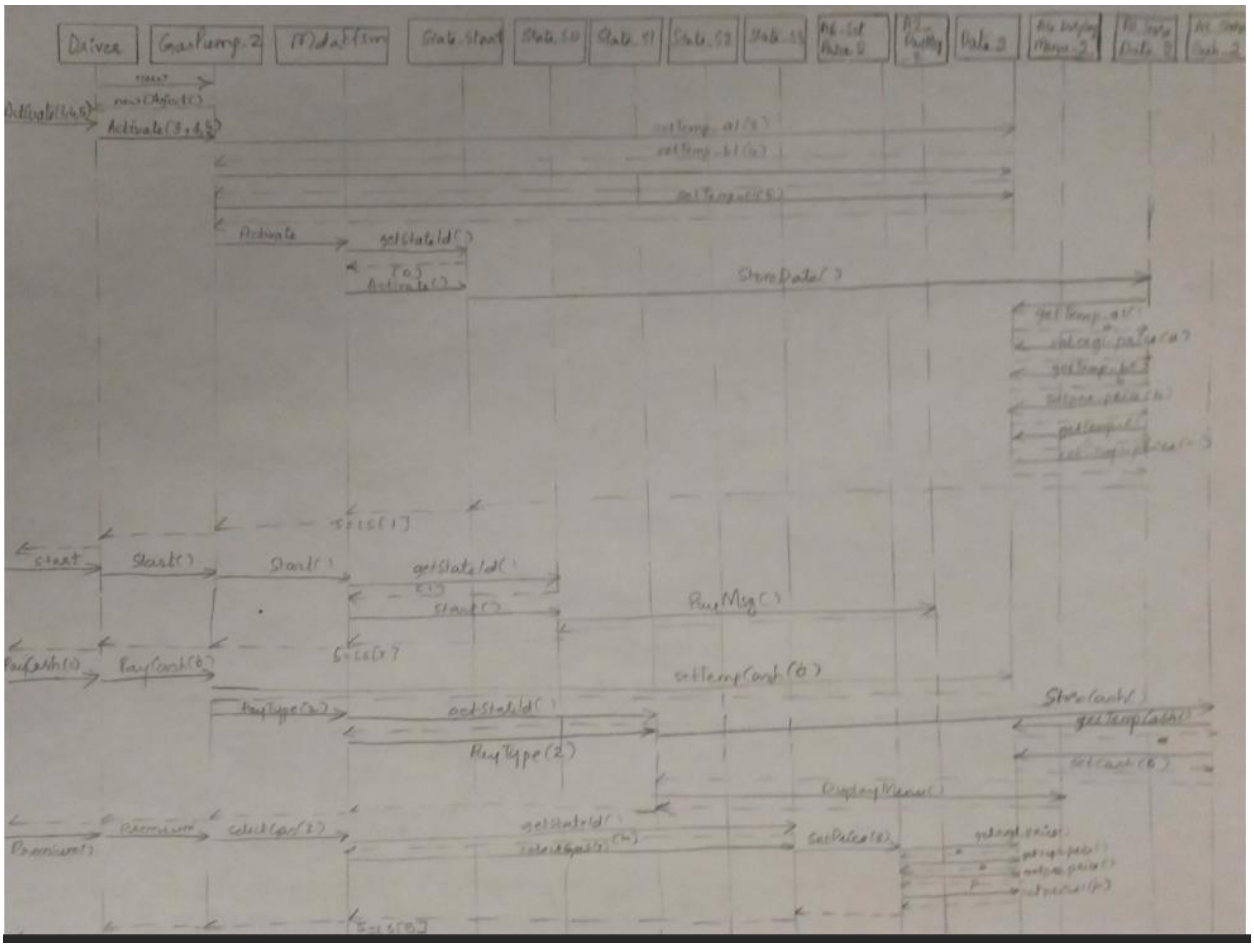

A1. StoreData
DataStore d
SetData(d)
StoreData()
A2. PayMsg
PayMsg()
A3. StoreCash
DataStore d
Float c
SetData(d)
StoreCash()
A4. DisplayMenu
DisplayMenu()
A5. RejectMsg
RejectMsg()
A6. SetPrice
DataStore d
SetData(d)
SetPrice(int g)
A7. ReadyMsg
ReadyMsg()
A8. SetInitialValues
DataStore d
SetData(d)
SetInitialValues()
A9. PumpGasUnit
DataStore d
float l, g
SetData(d)
PumpGasUnit()
A10. GasPumpedMsg
DataStore d
Float p
SetData(d)
GasPumpedMsg()
A11. StopMsg
StopMsg()
A12. PrintReceipt
DataStore d
float total
SetData(d)
PrintReceipt()
A13. CancelMsg
CancelMsg()
A14. ReturnCash
DataStore d
float total, cash
SetData(d)
ReturnCash()

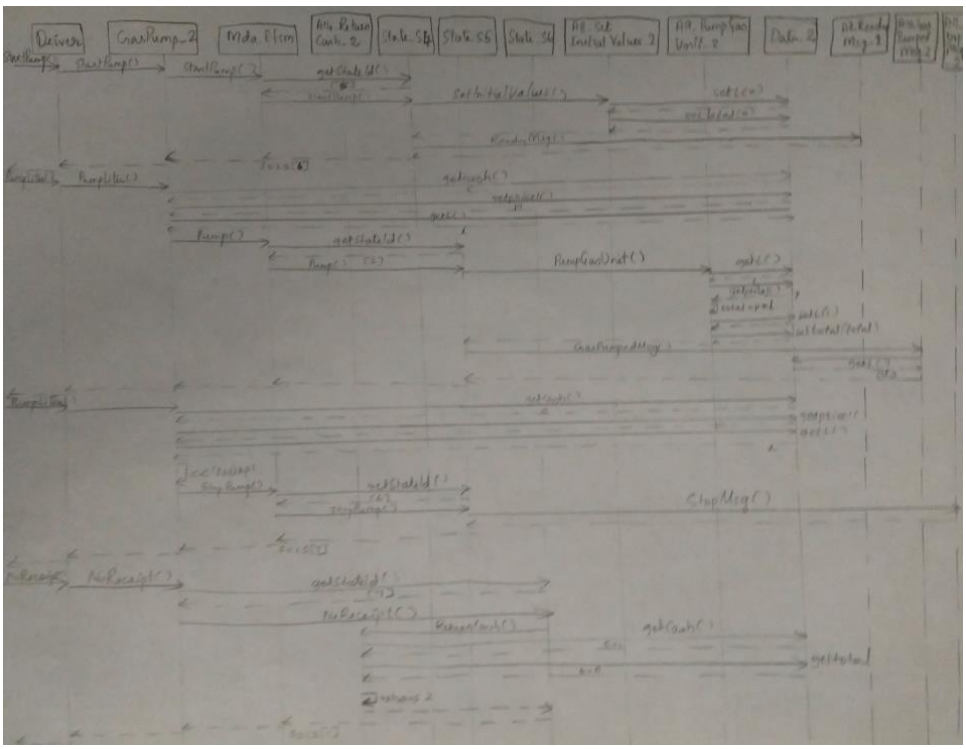


Sequence Diagrams:


PumpGallon() StopPump()
Driver
GasPump1
State_0
getStateId()
Start()
Approved()




Driver
GasPump_2
Mda_Platform
State_Start
State_S0
State_S1
State_S2
State_S3
Data_3
Activate
getStateId()
StoreData()
start
Start()
getStateId()
PumpMsg()
PayCash(0)
setTempCash(0)
PayType(2)
getStateId()
StoreCash()
getTempCash()
setCash(0)
DisplayMenu()
SetPrice(6)


Driver
GasPump_2
Mda_Platform
State_S5
State_S6
State_S4
Data_2
StartPump()
getStateId()
SetInitialValues()
ReadyMsg()
PumpStart()
getWeight()
getPrice()
Pump()
getStateId()
PumpGradeSet()
GasPumpedMsg()
StopPump()
getStateId()
StopMsg()
PaySlip()
getStateId()
PaySlip()
PaymentCash()
getCash()
getTotal()

Purpose of classes and responsibility of each operation supported by each class:

Class: AbstractFactory

Purpose: Abstract AbstractFactory class for creating objects for Strategy classes.

Operations: all abstract methods

Class: CF_GasPump1

Purpose: Create object References for strategy classes(containing implementation for meta actions) for GasPump_1 operations.

Operations: getStoreDataObj() //returns obj reference for StoreData strategy

getPayMsgObj //returns obj reference for PayMsg strategy

getDisplayMenuOj //returns obj reference for DisplayMenu strategy

getRejectMsgObj //returns obj reference for RejectMsg strategy

getSetPriceObj //returns obj reference for SetPrice strategy

getReadyMsgObj //returns obj reference for eadyMsg strategy

getSetInitialValuesObj //returns obj reference for SetInitialValues strategy

getPumpGasUnitObj //returns obj reference for PumpGasUnit strategy

getGasPumpedMsgObj //returns obj reference for GasPumpedMsg strategy

getStopMsgObj //returns obj reference for StopMsg strategy

getPrintReceiptObj //returns obj reference for PrintReceipt strategy

getCancelMsgObj //returns obj reference for CancelMsg strategy

getReturnCashObj //returns obj reference for ReturnCash strategy

Class: CF_GasPump2

Purpose: Create object References for strategy classes(containing implementation for meta actions) for GasPump_2 operations .

Operations: getStoreDataObj() //returns obj reference for StoreData strategy

getPayMsgObj //returns obj reference for PayMsg strategy

getStoreCash //returns obj reference for StoreCash strategy

getDisplayMenuOj //returns obj reference for DisplayMenu strategy

getRejectMsgObj //returns obj reference for RejectMsg strategy

getSetPriceObj //returns obj reference for SetPrice strategy

getReadyMsgObj //returns obj reference for eadyMsg strategy

getSetInitialValuesObj //returns obj reference for SetInitialValues strategy

getPumpGasUnitObj //returns obj reference for PumpGasUnit strategy

getGasPumpedMsgObj //returns obj reference for GasPumpedMsg strategy

getStopMsgObj //returns obj reference for StopMsg strategy

getPrintReceiptObj //returns obj reference for PrintReceipt strategy

getCancelMsgObj //returns obj reference for CancelMsg strategy

getReturnCashObj //returns obj reference for ReturnCash strategy

Class: DataStore

Purpose: Abstract class for DataStore for GasPump-1 and GasPump-2

Operations: all abstract methods

Class: Data_1

Purpose: Storage for GasPump-1 operations

Operations: Below operations Store gas prices temporarily while activating gaspump.

setTempa(float a)

setTempb(float b)

getTempa()

getTempb()

Below operations Store total amount of the transaction

settotal(floast c)

gettotal()

Below operations Storeprice for gas chosen

setprice(float y)

getprice()

Below operations Store Gallon units when PumpGallon is chosen

setG(float y)

getG()

Below operations Store the prices in respective variables

setsup_price(float y)

setreg_price(float y)

getsup_price()

getreg_price()

Class: Data_2

Purpose: Storage for GasPump-2 operations

Operations: Below operations Store gas prices temporarily while activating gaspump.

setTemp_a1(int a)

setTemp_b1(int b)

setTemp_c1(int c)

getTemp_a1()

getTemp_b1()

getTemp_c1()

Below operations Store cash entered and total amount of the transaction

setcash(float a)

settotal(float c)

getcash()

gettotal()

Below operations Storeprice for gas chosen

setprice1(int y)

getprice1()

Below operations Store Gallon units when PumpGallon is chosen

setL(float y)

getL()

Below operations Store the prices in respective variables

setsup1_price(int y)

setreg1_price(int y)

setpre_price(int y)

getsup1_price()

getreg1_price()

getpre_price()

Class: GasPump_1

Purpose: Input Processor class for GasPump-1 to which invokes different operations of the gas pump

Operations: Activate() -Store gas prices temporarily, calls Activate of MdaEfsm

Start() -calls Start of MdaEfsm

PayCredit() -calls PayType(1) of MdaEfsm

Reject() -calls Reject() of MdaEfsm

Cancel() -calls Cancel() of MdaEfsm

Approved() -calls Approved() of MdaEfsm

Super() -calls SelectGas(2) of MdaEfsm

Regular() -calls SelectGas(1) of MdaEfsm

StartPump() -calls StartPump() of MdaEfsm

PumpGallon() -calls Pump() of MdaEfsm

StopPump() calls StopPump() & Receipt of MdaEfsm

Class: GasPump_2

Purpose: Input Processor class for GasPump-2 to which invokes different operations of the gas pump

Operations: Activate() -Store gas prices temporarily, calls Activate of MdaEfsm

Start() -calls Start of MdaEfsm

PayCash() -Store cash temporarily ,calls PayType(2) of MdaEfsm

Cancel() -calls Cancel() of MdaEfsm

Super() -calls SelectGas(2) of MdaEfsm

Regular() -calls SelectGas(1) of MdaEfsm

Premium() -calls SelectGas(3) of MdaEfsm

StartPump() -calls StartPump() of MdaEfsm

PumpLiter() -calls Pump() of MdaEfsm if enough cash is paid

Stop() -calls StopPump() & Receipt of MdaEfsm

Receipt -calls Receipt() of MdaEfsm

NoReceipt -calls NoReceipt() of MdaEfsm

Class: MdaEfsm

Purpose: Context class for State classes that invoke meta actions in Op.

Operations: setState -This method is used to set the current state of the MDA-EFSM

setStatesList -defines the states for the state classes

Activate() -calls the Activate method in the respective State class

Start() -calls the Start method in the respective State class

PayType(int t) -calls the PayType method in the respective State class

Reject() -calls the Reject method in the respective State class

Cancel() -calls the Cancel method in the respective State class

Approved() -calls the Approved method in the respective State class

StartPump() -calls the StartPump method in the respective State class

Pump() -calls the Pump method in the respective State class

StopPump() -calls the StopPump method in the respective State class

SelectGas(int g) -calls the SelectGas method in the respective State class

Receipt() -calls the Receipt method in the respective State class

NoReceipt() -calls the NoReceipt method in the respective State class

Class: OutputProcessor

Purpose: Implements Meta Actions and creates references to respective classes(using Concrete Factory object) in the strategy classes that implements the different

strategies for the meta actions in the project

Operations: StoreData -calls StoreData of respective Strategy class

PayMsg -calls PayMsg of respective Strategy class

StoreCash -calls StoreCash of respective Strategy class

DisplayMenu -calls DisplayMenu of respective Strategy class

RejectMsg -calls RejectMsg of respective Strategy class

SetPrice(int g) -calls SetPrice of respective Strategy class

ReadyMsg -calls ReadyMsg of respective Strategy class

SetInitialValues -calls SetInitialValues of respective Strategy class

PumpGasUnit -calls PumpGasUnit of respective Strategy class

GasPumpedMsg -calls GasPumpedMsg of respective Strategy class

StopMsg -calls StopMsg of respective Strategy class

PrintReceipt -calls PrintReceipt of respective Strategy class

CancelMsg -calls CancelMsg of respective Strategy class

ReturnCash -calls ReturnCash of respective Strategy class

Class: State

Purpose: class with abstract meta events which serves as super class for all state classes

Operations: Below are all abstract methods

Activate()

Start()

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

Reject()

Cancel()

Approved()

StartPump()

Pump()

StopPump()

SelectGas(int g)

Receipt()

NoReceipt()

Class: State_Start

Purpose: Activating gas Pump, Starting point of State classes

Operations: Activate -calls StoreData of outputProcessor

Class: State_S0

Purpose: To implement Start() event

Operations: Start() -calls calls PayMsg of outputProcessor

Class: State_S1

Purpose: To implement PayType(t) event

Operations:

Class: State_S2

Purpose: To implement Approved() & Reject() events

Operations: -calls calls DisplayMenu(),RejectMsg() of outputProcessor

Class:State_S3

Purpose: To implement SelectGas(int g) & Cancel events

Operations: -calls calls SetPrice(g)&CancelMsg of outputProcessor

Class:State_S4

Purpose: To implement StartPump event

Operations: -calls calls SetInItialValues&ReadyMsg of outputProcessor

Class: State_S5

Purpose: To implement Pump()&StopPump() events

Operations: -calls calls PumpGasUnit(),GasPumpedMsg&StopMsg of outputProcessor

Class: State_S6

Purpose: To implement Receipt() & NoReceipt() events

Operations: -calls calls PrintReceipt() & ReturnCash() of outputProcessor

Class: A1_StoreData

Purpose: Abstract Class for StoreData() strategies

Operations: abstract StoreData()

Class: A1_StoreData_1

Purpose: Store gas price data for Gaspump-1

Operations: StoreData() -store super & regular price data

Class: A1_StoreData_2

Purpose: Store gas price data for Gaspump-2

Operations: StoreData() -store super,premium & regular price data

Class: A2_PayMsg

Purpose: abstract class for PayMsg() strategies

Operations: abstract PayMsg()

Class: A2_PayMsg_1

Purpose: Dispaly payment type msg for Gaspump-1

Operations: PayMsg() -display credit pay msg

Class: A2_PayMsg_2

Purpose: Dispaly payment type msg for Gaspump-2

Operations: PayMsg() -display credit pay msg

Class: A3_StoreCash

Purpose: to store cash for Gaspump-2

Operations: abstract StoreCash()

Class: A3_StoreCash_2

Purpose: stores cash entered for Gaspump-2

Operations: StoreCash() -stores cash from temp cash variable.

Class: A4_DisplayMenu

Purpose: Dispaly Menu for gasPumps

Operations:asbtract DisplayMenu()

Class: A4_DisplayMenu_1

Purpose: Dispaly Menu for Gaspump-1

Operations: DisplayMenu() -display list of selections

Class: A4_DisplayMenu_2

Purpose: Dispaly Menu for Gaspump-2

Operations: DisplayMenu() --display list of selections

Class: A5_RejectMsg

Purpose: Display a Reject message when credit card is rejected

Operations: abstract RejectMsg()

Class: A5_RejectMsg_1

Purpose: Displays Reject message when credit card is rejected for Gaspump-1

Operations: RejectMsg() -dispaly credit card rejected message

Class: A6_SetPrice

Purpose: Set price for the gas based on the gas chosen

Operations: abstract SetPrice(g)

Class: A6_SetPrice_1

Purpose: Set price for the gas based on the super / regular

Operations: SetPrice(g) -setprice based on g

Class: A6_SetPrice_2

Purpose: Set price for the gas based on the super/premium/regular

Operations:SetPrice(g) -setprice based on g

Class: A7_ReadyMsg

Purpose: Display a message when ready for pumping

Operations: abstract ReadyMsg()

Class: A7_ReadyMsg_1

Purpose: Display a message when ready for pumping

Operations: ReadyMsg -displays ready for pumping msg

Class: A8_SetInitialValues

Purpose: Sets the initial value of total and units pumped to zero

Operations: abstract SetInitialValues()

Class: A8_SetInitialValues_1

Purpose: Sets the initial value of total and units pumped in gallons to zero

Operations: SetInitialValues() -sets gallon units and total to zero

Class: A8_SetInitialValues_2

Purpose: Sets the initial value of total and units pumped in liters to zero

Operations: SetInitialValues() -sets liter units and total to zero

Class: A9_PumpGasUnit

Purpose: disposes unit of gas and counts # of units disposed

Operations: abstract PumpGasUnit

Class: A9_PumpGasUnit_1

Purpose: disposes unit of gas and counts # of units disposed in gallons

Operations: PumpGasUnit() - calculates disposed unit of gas and counts # of units disposed in gallons

Class: A9_PumpGasUnit_2

Purpose: disposes unit of gas and counts # of units disposed in liters

Operations: PumpGasUnit() -calculates disposed unit of gas and counts # of units disposed in liters

Class: A10_GasPumpedMsg

Purpose: displays the amount of disposed gas

Operations: abstract GasPumpedMsg()

Class: A10_GasPumpedMsg_1

Purpose: displays the amount of disposed gas in gallons

Operations: GasPumpedMsg() -display amount of disposed gas in gallons

Class: A10_GasPumpedMsg_2

Purpose: displays the amount of disposed gas in liters

Operations: GasPumpedMsg() -displays the amount of disposed gas in liters

Class: A11_StopMsg

Purpose: stop pump message and receipt? msg (optionally)

Operations: abstract StopMsg()

Class: A11_StopMsg_1

Purpose: stop pump message

Operations: StopMsg() -prints stopped message and receipt.

Class: A11_StopMsg_2

Purpose: stop pump message and receipt? msg

Operations: StopMsg() -prints stopped message and asks if receipts needs to be printed

Class: A12_PrintReceipt

Purpose: print a receipt

Operations: abstract PrintReceipt()

Class: A12_PrintReceipt_1

Purpose: print a receipt

Operations: PrintReceipt() -prints a receipt

Class: A13_CancelMsg

Purpose: displays a cancellation message

Operations: abstract CancelMsg()

Class: A13_CancelMsg_1

Purpose: displays a cancellation message

Operations: CancelMsg() -display cancelled message

Class: A14_ReturnCash

Purpose: returns the remaining cash

Operations:abstract ReturnCash()

Class: A14_ReturnCash_1

Purpose: returns the remaining cash

Operations: ReturnCash() -returns remaining cash after deducting from total

Source Code with Patterns:

Abstract Factory Pattern:

```
package AbstractFactory;

import DataStore.DataStore;
import Strategy.*;


/**
 * Created by Sharel on 4/19/2017.
 */
/*Abstract AbstractFactory class for creating objects for Strategy classes*/
public abstract class AbstractFactory {
    public abstract  DataStore getDataStore();

    public abstract A1_StoreData getStoreDataObj();
    public abstract A2_PayMsg getPayMsgObj();
    public abstract A3_StoreCash getStoreCash();
    public abstract A4_DisplayMenu getDisplayMenuOj();
    public abstract A5_RejectMsg getRejectMsgObj();
    public abstract A6_SetPrice getSetPriceObj();
    public abstract A7_ReadyMsg getReadyMsgObj();
    public abstract A8_SetInitialValues getSetInitialValuesObj();
    public abstract A9_PumpGasUnit getPumpGasUnitObj();
```

```
    public abstract A10_GasPumpedMsg getGasPumpdMsgObj();
    public abstract A11_StopMsg getStopMsgObj();
    public abstract A12_PrintReceipt getPrintReceiptObj();
    public abstract A13_CancelMsg getCancelMsgObj();
    public abstract A14_ReturnCash getReturnCashObj();
}
```

Concrete Classes of AF:

```
package AbstractFactory;

import DataStore.*;
import Strategy.*;


/**
 * Created by Sharel on 4/19/2017.
 */
/*Create respective object References for strategy classes for GasPump_1
operations*/
public class CF_GasPump1 extends AbstractFactory {
    DataStore ds;

    @Override
    public DataStore getDataStore() {
        ds = new Data_1();
        return ds;
    }

    @Override
    public A1_StoreData getStoreDataObj() {
        A1_StoreData a1_storeData = new A1_StoreData_1();
        a1_storeData.setdata(ds);
        return a1_storeData;
    }

    @Override
    public A2_PayMsg getPayMsgObj() {
        A2_PayMsg a2_payMsg = new A2_PayMsg_1();
        return a2_payMsg;
    }

    @Override
    public A3_StoreCash getStoreCash() {
       return null;
    }

    @Override
    public A4_DisplayMenu getDisplayMenuOj() {
        A4_DisplayMenu a4_displayMenu= new A4_DisplayMenu_1();
        return a4_displayMenu;
```

```
}

@Override
public A5_RejectMsg getRejectMsgObj() {
    A5_RejectMsg a5_rejectMsg=new A5_RejectMsg_1();
    return a5_rejectMsg;
}

@Override
public A6_SetPrice getSetPriceObj() {
A6_SetPrice a6_setPrice = new A6_SetPrice_1();
a6_setPrice.setdata(ds);
    return a6_setPrice;
}

@Override
public A7_ReadyMsg getReadyMsgObj() {
    A7_ReadyMsg a7_readyMsg=new A7_ReadyMsg_1();
    return a7_readyMsg;
}

@Override
public A8_SetInitialValues getSetInitialValuesObj() {
    A8_SetInitialValues a8_setInitialValues=new A8_SetInitialValues_1();
    a8_setInitialValues.setdata(ds);
    return a8_setInitialValues;
}

@Override
public A9_PumpGasUnit getPumpGasUnitObj() {
    A9_PumpGasUnit a9_pumpGasUnit=new A9_PumpGasUnit_1();
    a9_pumpGasUnit.setdata(ds);
    return a9_pumpGasUnit;
}

@Override
public A10_GasPumpedMsg getGasPumpedMsgObj() {
    A10_GasPumpedMsg a10_gasPumpedMsg = new A10_GasPumpedMsg_1();
    a10_gasPumpedMsg.setdata(ds);
    return a10_gasPumpedMsg;
}

@Override
public A11_StopMsg getStopMsgObj() {
    A11_StopMsg a11_stopMsg= new A11_StopMsg_1();
    return a11_stopMsg;
}

@Override
public A12_PrintReceipt getPrintReceiptObj() {
    A12_PrintReceipt a12_printReceipt=new A12_PrintReceipt_1();
    a12_printReceipt.setdata(ds);
    return a12_printReceipt;
```

```
    }

    @Override
    public A13_CancelMsg getCancelMsgObj() {
        A13_CancelMsg a13_cancelMsg=new A13_CancelMsg_1();
        return a13_cancelMsg;
    }

    @Override
    public A14_ReturnCash getReturnCashObj() {
        A14_ReturnCash a14_returnCash = new A14_ReturnCash_2();
        a14_returnCash.setdata(ds);
        return a14_returnCash;
    }
}
```

```
package AbstractFactory;

import DataStore.*;
import Strategy.*;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Create respective object References for strategy classes for GasPump_2
operations*/

public class CF_GasPump2 extends AbstractFactory {
    DataStore ds;

    @Override
    public DataStore getDataStore() {
        ds = new Data_2();
        return ds;
    }

    @Override
    public A1_StoreData getStoreDataObj() {
        A1_StoreData a1_storeData = new A1_StoreData_2();
        a1_storeData.setdata(ds);
        return a1_storeData;
    }

    @Override
    public A2_PayMsg getPayMsgObj() {
        A2_PayMsg a2_payMsg = new A2_PayMsg_2();
        return a2_payMsg;
    }

    @Override
```

```
public A3_StoreCash getStoreCash() {
    A3_StoreCash a3_storeCash=new A3_StoreCash_2();
    a3_storeCash.setdata(ds);
    return a3_storeCash;
}

@Override
public A4_DisplayMenu getDisplayMenuOj() {
    A4_DisplayMenu a4_displayMenu= new A4_DisplayMenu_2();
    return a4_displayMenu;
}

@Override
public A5_RejectMsg getRejectMsgObj() {
    A5_RejectMsg a5_rejectMsg=new A5_RejectMsg_1();
    return a5_rejectMsg;
}

@Override
public A6_SetPrice getSetPriceObj() {
    A6_SetPrice a6_setPrice = new A6_SetPrice_2();
    a6_setPrice.setdata(ds);
    return a6_setPrice;
}

@Override
public A7_ReadyMsg getReadyMsgObj() {
    A7_ReadyMsg a7_readyMsg=new A7_ReadyMsg_1();
    return a7_readyMsg;
}

@Override
public A8_SetInitialValues getSetInitialValuesObj() {
    A8_SetInitialValues a8_setInitialValues=new A8_SetInitialValues_2();
    a8_setInitialValues.setdata(ds);
    return a8_setInitialValues;
}

@Override
public A9_PumpGasUnit getPumpGasUnitObj() {
    A9_PumpGasUnit a9_pumpGasUnit=new A9_PumpGasUnit_2();
    a9_pumpGasUnit.setdata(ds);
    return a9_pumpGasUnit;
}

@Override
public A10_GasPumpedMsg getGasPumpedMsgObj() {
    A10_GasPumpedMsg a10_gasPumpedMsg = new A10_GasPumpedMsg_2();
    a10_gasPumpedMsg.setdata(ds);
    return a10_gasPumpedMsg;
}

@Override
```

```
    public A11_StopMsg getStopMsgObj() {
        A11_StopMsg a11_stopMsg= new A11_StopMsg_2();
        return a11_stopMsg;
    }

    @Override
    public A12_PrintReceipt getPrintReceiptObj() {
        A12_PrintReceipt a12_printReceipt=new A12_PrintReceipt_1();
        a12_printReceipt.setdata(ds);
        return a12_printReceipt;
    }

    @Override
    public A13_CancelMsg getCancelMsgObj() {
        A13_CancelMsg a13_cancelMsg=new A13_CancelMsg_1();
        return a13_cancelMsg;
    }

    @Override
    public A14_ReturnCash getReturnCashObj() {
        A14_ReturnCash a14_returnCash = new A14_ReturnCash_2();
        a14_returnCash.setdata(ds);
        return a14_returnCash;
    }
}
```

DataStore class:

```
package DataStore;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract class for DataStore for GasPump-1 and GasPump-2*/

public abstract class DataStore {
    public void setTempa(float a) {

    }
    public float getTempa() {
        return 0;
    }

    public void setTempb(float b) {}
    public float getTempb() {
        return 0;
```

```
    }

    public void setTemp_a1(int a) {}
    public void setTemp_b1(int b) {}

    public void setTemp_c1(int c) {}


    public int getTemp_a1() {
        return 0;
    }
    public int getTemp_b1() {
        return 0;
    }
    public int getTemp_c1() {
        return 0;
    }

   public float getTempCash() {
        return 0;
    }

    public void setTempCash(float cash) {

    }

    public void setprice(float a) {

    }
    public void setprice1(int a) {

    }
    public void setcash(float a) {

    }
    public void setG(float a) {

    }
    public float getprice() {
        return 0;
    }
    public int getprice1() {
        return 0;
    }

    public float getcash() {
        return 0;
    }
    public float getG() {
        return 0;
    }
```

```
    public float gettotal() {
        return 0;
    }
    public void settotal(float a) {

    }

    public void setsup1_price(int a) {

    }
    public int getsup1_price()

    {
        return 0;
    }

    public void setsup_price(float a) {

    }
    public float getsup_price() {
        return 0;
    }

    public void setreg_price(float a) {

    }

    public float getreg_price() {
        return 0;
    }
    public void setpre_price(int a) {

    }
    public void setreg1_price(int a) {

    }
    public int getreg1_price() {
        return 0;
    }
    public int getpre_price() {
        return 0;
    }
    public float getL() {
        return 0;
    }
    public void setL(float a) {

    }
}
```

```
package DataStore;

/**
 * Created by Sharel on 4/19/2017.
 */
public class Data_1 extends DataStore{



    float temp_a;
    float temp_b;


    static float price;
    static float cash;
    static float total;
    static float G;
    static float reg_price;
    static float sup_price;

    //Store gas prices temporarily
    public void setTempa(float a)
    {
        temp_a =a;
    }
    public float getTempa(){return temp_a;}
    public void setTempb(float b)
                {
        temp_b=b;
    }
    public float getTempb(){return temp_b;}

    //Store ttotal amount of the transaction
    public void settotal(float c)
    {
        total =c;
    }
    public float gettotal()
    {
        return total;
    }

    //Storeprice for gas chosen
    public void setprice(float y)
    {
        price=y;
    }
    public float getprice()
    {
        return price;
    }
    public void setcash(float a)
    {
```

```
        cash =a;
    }
    public float getcash()
    {
        return cash;
    }

    //Store Gallon units when PumpGallon is chosen
    public void setG(float y)
    {
        G=y;
    }
    public float getG()
    {
        return G;
    }

    //Store the prices in respective variables
    public void setsup_price(float y)
    {
        sup_price=y;
    }
    public float getsup_price()
    {
        return sup_price;
    }
    public void setreg_price(float y)
    {
        reg_price=y;
    }
    public float getreg_price()
    {
        return reg_price;
    }
}



package DataStore;

/**
 * Created by Sharel on 4/19/2017.
 */
public class Data_2 extends DataStore {

    int temp_a1;
    int temp_b1;
    int temp_c1;
    float temp_cash;

    static float L;
    static int reg1_price;
    static int pre_price;
    static int sup1_price;
```

```
    static float cash;
    static float total;
    static int price;


    //Price for the gas selected
    public int getprice1()
    {
        return price;
    }
    public void setprice1(int y)
    {
        price=y;
    }

    //Store temp the prices for gases
    public void setTemp_a1(int a)
    {
        temp_a1 =a;
    }
    public void setTemp_b1(int b)
    {
        temp_b1=b;
    }

    public void setTemp_c1(int c)
    {
        temp_c1=c;
    }



    public int getTemp_a1(){return temp_a1;}
    public int getTemp_b1(){return  temp_b1;}
    public int getTemp_c1(){return  temp_c1;}

    //Get the litre unit when PumpLiter is chosen

    public void setL(float a)
    {
        L=a;
    }
    public float getL()
    {
        return L;
    }

    //Store prices for different gas types
    public void setpre_price(int a)
    {
        pre_price=a;
    }
    public int getpre_price()
    {
```

```
        return pre_price;
    }
    public void setreg1_price(int a)
    {
        reg1_price=a;
    }
    public int getreg1_price()
    {
        return reg1_price;
    }

    public void setsup1_price(int a)
    {
           sup1_price=a;
    }
    public int getsup1_price()
    {
        return sup1_price;
    }

    //Store the cash entered and Transaction total amount
    public void setcash(float a)
    {
        cash=a;
    }
    public float getcash()
    {
        return cash;
    }
    public float gettotal()
    {
        return total;
    }
    public void settotal(float a)
    {
        total=a;
    }

    //Store Cash in temp variables

    public void setTempCash(float cash){

        temp_cash = cash;
    }

    public float getTempCash(){
        return temp_cash;
    }

}
```

InputProcessor classes:

```
package InputProcessor;

import AbstractFactory.*;
import DataStore.*;
import MdaEfsm.*;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Input Processor class for GasPump-1 to which invokes different operations of the
gas pump*/
public class GasPump_1 {
    AbstractFactory abstractFactory;
    MdaEfsm mdaEfsm;
    DataStore dataStore;

    //Creating objects for AF, MDaEfsm And DataStore
    public void setAbstractFactory(AbstractFactory abstractFactory) {
        this.abstractFactory = abstractFactory;
    }

    public void setMdaEfsm(MdaEfsm mdaEfsm) {
        this.mdaEfsm = mdaEfsm;
    }

    public void setDataStore(DataStore dataStore) {
        this.dataStore = dataStore;
    }

    public  void  Activate(float a, float b) {
        if ((a>0)&&(b>0)) {
            dataStore.setTempa(a);
            dataStore.setTempb(b);
            mdaEfsm.Activate();

        }
    }
    public  void  Start() {
    mdaEfsm.Start();
    }
    public  void  PayCredit() {
        mdaEfsm.PayType(1);
    }

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

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

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

    public  void Super() {
        mdaEfsm.SelectGas(2);
    }
public void Regular() {
        mdaEfsm.SelectGas(1);
    }

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

public void PumpGallon() {
        mdaEfsm.Pump();

}

public void StopPump() {
        mdaEfsm.StopPump();
        mdaEfsm.Receipt();

        }


}
```

```
package InputProcessor;

import AbstractFactory.AbstractFactory;
import DataStore.DataStore;
import MdaEfsm.MdaEfsm;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Input Processor class for GasPump-2 to which invokes different operations of the
gas pump*/
public class GasPump_2 {
    AbstractFactory abstractFactory;
    MdaEfsm mdaEfsm;
    DataStore dataStore;

    //Creating objects for AF, MDaEfsm And DataStore

    public void setAbstractFactory(AbstractFactory abstractFactory) {
        this.abstractFactory = abstractFactory;
```

```
    }

    public void setMdaEfsm(MdaEfsm mdaEfsm) {
        this.mdaEfsm = mdaEfsm;
    }

    public void setDataStore(DataStore dataStore) {
        this.dataStore = dataStore;
    }

    public  void  Activate(int a, int b, int c) {
        if ((a>0)&&(b>0)&&(c>0)) {
            dataStore.setTemp_a1(a);
            dataStore.setTemp_b1(b);
            dataStore.setTemp_c1(c);

            mdaEfsm.Activate();
        }
    }
    public  void  Start() {
        mdaEfsm.Start();
    }
    public  void  PayCash(float c) {
        if (c>0) {
            dataStore.setTempCash(c);
            mdaEfsm.PayType(2);

        }
    }

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



    public  void Super() {
        mdaEfsm.SelectGas(2);
    }
    public void Regular() {
        mdaEfsm.SelectGas(1);
    }
    public void Premium() {
        mdaEfsm.SelectGas(3);
    }

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

public void PumpLiter() {

    float cash = dataStore.getcash();
    float price =dataStore.getprice1();
```

```
    float L = dataStore.getL();
    boolean res = (cash<(L+1)*price);
   // System.out.println("\nCash:"+cash
+"\tprice:"+price+"\tLiter:"+L+"\tResult:"+res);


        if(cash<(L+1)*price){
            mdaEfsm.StopPump();
        }
        else {
            mdaEfsm.Pump();
        }
     }


   public void Stop() {
      mdaEfsm.StopPump();
    }
public  void Receipt() {
       mdaEfsm.Receipt();
    }
public void NoReceipt() {
mdaEfsm.NoReceipt();
    }


}
```

MDA-EFSM class:

```
package MdaEfsm;
import State.*;

/**
 * Created by Sharel on 4/17/2017.
 */
public class MdaEfsm {

    private State currentState;
    State[] ls = new State[8];

//below method is used to set the current state of the MDA-EFSM
    public void setState(State s)
    {
        currentState = s;
    }

    //to define the states for the state classes
    public void setStatesList( State[] x)
    {
        ls=x;
```

```
    }

    /*Meta Events implementation to refer to respective state classes
implementing the functionalities in the State design pattern*/

    public void Activate() {

        int id = currentState.getStateId();
        System.out.println("Current state is in :"+id);

        switch (id) {
            case 0: {
                currentState.Activate();
                currentState = ls[1];       //ready for next state
                break;
            }
            case 1: break;
            case 2: break;
            case 3: break;
            case 4: break;
            case 5: break;
            case 6: break;
            case 7: {
                System.out.println("Already activated!");
                break;
            }

        }
    }

    public void Start(){
        int id = currentState.getStateId();
        switch(id)
        {
            case 0: break;
            case 1: {
                currentState.Start();
                currentState = ls[2];
                break;
            }
            case 2: break;
            case 3: break;
            case 4: break;
            case 5: break;
            case 6: break;
            case 7:
            {
                System.out.println("Incorrect state.");
                break;
            }
        }
    }

    public void PayType(int t){
```

```
        int id = currentState.getStateId();
        switch(id)
        {
            case 0: break;
            case 1: break;
            case 2: {
                if(t==1){
                    currentState.PayType(t);
                    currentState = ls[3];
                }
                else{
                    currentState.PayType(t);
                    currentState = ls[4];
                }
                break;
            }
            case 3: break;
            case 4: break;
            case 5: break;
            case 6: break;
            case 7:
            {
                System.out.println("Paytype");
                break;
            }
        }

    }

    public void  Reject(){
        int id = currentState.getStateId();
        switch(id)
        {
            case 0: break;
            case 1: break;
            case 2: break;
            case 3:
                    currentState.Reject();
                    currentState = ls[1];  //change state to 0 or 1??
                break;

            case 4: break;
            case 5: break;
            case 6: break;
            case 7:
            {
                System.out.println("Reject!");
                break;
            }
        }
    }
    public  void Cancel(){
        int id = currentState.getStateId();
        switch(id)
        {
```

```
            case 0: break;
            case 1: break;
            case 2: break;
            case 3: break;
            case 4:
                currentState.Cancel();
                currentState = ls[1];  //change state to 0 or 1??
                break;

            case 5: break;
            case 6: break;
            case 7:
            {
                System.out.println("Cancel!");
                break;
            }
        }

    }
    public  void  Approved(){
        int id = currentState.getStateId();
        switch(id)
        {
            case 0: break;
            case 1: break;
            case 2: break;
            case 3:
                currentState.Approved();
                currentState = ls[4];  //change state to 0 or 1??
                break;
            case 4: break;
            case 5: break;
            case 6: break;
            case 7:
            {
                System.out.println("Approved!");
                break;
            }
        }

    }
    public  void  StartPump(){
        int id = currentState.getStateId();
        switch(id)
        {
            case 0: break;
            case 1: break;
            case 2: break;
            case 3: break;
            case 4: break;
            case 5:
                currentState.StartPump();
                currentState = ls[6];  //change state to 0 or 1??
                break;
```

```
            case 6: break;
            case 7:
            {
                System.out.println("StartPump!");
                break;
            }
        }
    }
    public  void  Pump(){

        int id = currentState.getStateId();
        switch(id)
        {
            case 0: break;
            case 1: break;
            case 2: break;
            case 3: break;
            case 4: break;
            case 5: break;
            case 6: {
                System.out.println("\n\n The gas is being pumped....");
                currentState.Pump();
                currentState = ls[6];  //change state to 0 or 1??
                break;
            }

            case 7:
            {
                System.out.println("Pump!");
                break;
            }
        }
    }
    public  void  StopPump(){
        int id = currentState.getStateId();
        switch(id)
        {
            case 0: break;
            case 1: break;
            case 2: break;
            case 3: break;
            case 4: break;
            case 5: break;
            case 6:
                currentState.StopPump();
                currentState = ls[7];  //change state to 0 or 1??
                break;

            case 7:
            {
                System.out.println("StopPump!");
                break;
```

```
            }
        }

    }
    public  void  SelectGas(int g){
        int id = currentState.getStateId();
        switch(id)
        {
            case 0: break;
            case 1: break;
            case 2: break;
            case 4: currentState.SelectGas(g);
                    currentState = ls[5];  //change state to 0 or 1??
                    break;
            case 3: break;
            case 5: break;
            case 6: break;
            case 7:

                break;

            case 8:
            {
                System.out.println("SelectGas");
                break;
            }
        }

    }
    public  void  Receipt(){
        int id = currentState.getStateId();
        switch(id)
        {
            case 0: break;
            case 1: break;
            case 2: break;
            case 4: break;
            case 3: break;
            case 5: break;
            case 6: break;
            case 7:currentState.Receipt();
                   currentState = ls[1];  //change state to 0 or 1??

                break;



        }
    }

    public  void  NoReceipt(){
        int id = currentState.getStateId();
        switch(id)
        {
```

```
            case 0: break;
            case 1: break;
            case 2: break;
            case 4: break;
            case 3: break;
            case 5: break;
            case 6: break;
            case 7:currentState.NoReceipt();
                currentState = ls[1];  //change state to 0 or 1??
                break;

        }
    }
}
```

OutputProcessor:

```
package OutputProcessor;
import DataStore.*;
import AbstractFactory.*;
import Strategy.*;

/**
 * Created by Sharel on 4/17/2017.
 */
public class OutputProcessor {

    DataStore ds;
    AbstractFactory AF;

    //setting up the objects for AF and DataStore classes
    public void setdata(DataStore x){
        ds=x;
    }
    public void setfactory(AbstractFactory x)
    {
        AF=x;
    }

    /*Meta Actions implementation here creates references to respective
classes(using Concrete Factory object)
     *in the strategy classes that implements the different
     *strategies for the meta actions in the project*/

    public void StoreData(){
        A1_StoreData a1_storeData;
        a1_storeData = AF.getStoreDataObj();
        a1_storeData.StoreData();
    }

    public void PayMsg(){
```

```
        A2_PayMsg a2_payMsg;
        a2_payMsg = AF.getPayMsgObj();
        a2_payMsg.PayMsg();
    }

    public void StoreCash(){
        A3_StoreCash a3_storeCash;
        a3_storeCash = AF.getStoreCash();
        a3_storeCash.StoreCash();
    }

    public void DisplayMenu(){
        A4_DisplayMenu a4_displayMenu;
        a4_displayMenu = AF.getDisplayMenuOj();
        a4_displayMenu.DisplayMenu();
    }

    public void RejectMsg(){
        A5_RejectMsg a5_rejectMsg;
        a5_rejectMsg = AF.getRejectMsgObj();
        a5_rejectMsg.RejectMsg();
    }

    public void SetPrice(int g){
        A6_SetPrice a6_setPrice;
        a6_setPrice = AF.getSetPriceObj();
        a6_setPrice.SetPrice(g);
    }

    public void ReadyMsg(){
        A7_ReadyMsg a7_readyMsg;
        a7_readyMsg = AF.getReadyMsgObj();
        a7_readyMsg.ReadyMsg();
    }

    public void SetInItialValues(){
        A8_SetInitialValues a8_setInitialValues;
        a8_setInitialValues = AF.getSetInitialValuesObj();
        a8_setInitialValues.SetInitialValues();
    }

    public void PumpGasUnit(){
        A9_PumpGasUnit a9_pumpGasUnit;
        a9_pumpGasUnit = AF.getPumpGasUnitObj();
        System.out.println("\nIn OP..");
        a9_pumpGasUnit.PumpGasUnit();
    }

    public void GasPumpedMsg( ){
        A10_GasPumpedMsg a10_gasPumpedMsg;
        a10_gasPumpedMsg = AF.getGasPumpedMsgObj();
        a10_gasPumpedMsg.GasPumpedMsg();
    }
```

```
    public void StopMsg(){
        A11_StopMsg a11_stopMsg;
        a11_stopMsg=AF.getStopMsgObj();
        a11_stopMsg.StopMsg();
    }

    public void PrintReceipt(){
        A12_PrintReceipt a12_printReceipt;
        a12_printReceipt = AF.getPrintReceiptObj();
        a12_printReceipt.PrintReceipt();
    }

    public void CancelMsg(){
        A13_CancelMsg a13_cancelMsg;
        a13_cancelMsg = AF.getCancelMsgObj();
        a13_cancelMsg.CancelMsg();
    }

    public void ReturnCash(){
        A14_ReturnCash a14_returnCash;
        a14_returnCash = AF.getReturnCashObj();
        a14_returnCash.ReturnCash();
    }
}
```

State Design Pattern:

```
package State;

import OutputProcessor.*;

/**
 * Created by Sharel on 4/17/2017.
 */

/*State class with abstract meta events which serves as super class for all state
classes*/
public abstract class State {
    int stateId;
    OutputProcessor outputProcessor;

//to retrieve/store the state at any given point in time
    public int getStateId() {
        return stateId;
    }

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

```
    //object reference to OutputProcessor
    public void setOutputProcessor(OutputProcessor outputProcessor) {
        this.outputProcessor = outputProcessor;
    }

    //Meta Events - abstract
    public abstract void Activate();
    public abstract void Start();
    public abstract void  PayType(int t); //credit: t=1; cash: t=2
    public abstract void Reject();
    public abstract void Cancel();
    public abstract void  Approved();
    public abstract void  StartPump();
    public abstract void  Pump();
    public abstract void  StopPump();
    public abstract void  SelectGas(int g);
    public abstract void  Receipt();
    public abstract void  NoReceipt();


}
```

```
package State;

/**
 * Created by Sharel on 4/19/2017.
 */

/*State class that directs outputProcessor to implement
*Activate() meta Event with meta action StoreData();
* this is the starting point of the state classes*/

public class State_Start extends State {
    @Override
    public void Activate() {
        outputProcessor.StoreData();
        System.out.println("\n \n ******Activating GasPump******");
    }

    @Override
    public void Start() {

    }

    @Override
    public void PayType(int t) {

    }

    @Override
    public void Reject() {
```

```
    }

    @Override
    public void Cancel() {

    }

    @Override
    public void Approved() {

    }

    @Override
    public void StartPump() {

    }

    @Override
    public void Pump() {

    }

    @Override
    public void StopPump() {

    }

    @Override
    public void SelectGas(int g) {

    }

    @Override
    public void Receipt() {

    }

    @Override
    public void NoReceipt() {

    }
}



package State;

/**
 * Created by Sharel on 4/19/2017.
 */
```

```
/*State class that directs outputProcessor to implement Start() meta Event with
meta action PayMsg(); */

public class State_S0 extends State {
    @Override
    public void Activate() {

    }

    @Override
    public void Start() {
        outputProcessor.PayMsg();
    }

    @Override
    public void PayType(int t) {

    }

    @Override
    public void Reject() {

    }

    @Override
    public void Cancel() {

    }

    @Override
    public void Approved() {

    }

    @Override
    public void StartPump() {

    }

    @Override
    public void Pump() {

    }

    @Override
    public void StopPump() {

    }
```

```
    @Override
    public void SelectGas(int g) {

    }

    @Override
    public void Receipt() {

    }

    @Override
    public void NoReceipt() {

    }
}
```

```
package State;

/**
 * Created by Sharel on 4/19/2017.
 */

/*State class that directs outputProcessor to implement
*PayType() meta Event with meta action StoreCash()
* and DisplayMenu when payment option is cash */

public class State_S1 extends State{
    @Override
    public void Activate() {

    }

    @Override
    public void Start() {

    }

    @Override
    public void PayType(int t) {
        if(t==1){
            System.out.println("\n\n\t\tYou chose to pay using credit. Follow the
instructions to continue...\n");
            //changes state to next
        }
        else if(t==2){
            System.out.println("\n\n\t\tYou chose to pay using Cash. Follow the
instructions to continue...\n");
             outputProcessor.StoreCash();
             outputProcessor.DisplayMenu();
        }
```

```
    }

    @Override
    public void Reject() {

    }

    @Override
    public void Cancel() {

    }

    @Override
    public void Approved() {

    }

    @Override
    public void StartPump() {

    }

    @Override
    public void Pump() {

    }

    @Override
    public void StopPump() {

    }

    @Override
    public void SelectGas(int g) {

    }

    @Override
    public void Receipt() {

    }

    @Override
    public void NoReceipt() {

    }
}
```

```
package State;

/**
 * Created by Sharel on 4/19/2017.
 */

/*State class that directs outputProcessor to implement
*Reject()&Approved() meta Event with meta action RejectMsg()
* and DisplayMenu() when payment option is credit */

public class State_S2 extends State{
    @Override
    public void Activate() {
            }

    @Override
    public void Start() {

    }

    @Override
    public void PayType(int t) {

    }

    @Override
    public void Reject() {
        outputProcessor.RejectMsg();
    }

    @Override
    public void Cancel() {

    }

    @Override
    public void Approved() {
        outputProcessor.DisplayMenu();
    }

    @Override
    public void StartPump() {

    }

    @Override
    public void Pump() {

    }

    @Override
```

```
    public void StopPump() {

    }

    @Override
    public void SelectGas(int g) {

    }

    @Override
    public void Receipt() {

    }

    @Override
    public void NoReceipt() {

    }
}
```

```
package State;

/**
 * Created by Sharel on 4/19/2017.
 */

/*State class that directs outputProcessor to implement
*Cancel()& SelectGas(g) meta Event with meta action CancelMsg()
* and SetPrice(g) */

public class State_S3 extends State {
    @Override
    public void Activate() {

    }

    @Override
    public void Start() {

    }

    @Override
    public void PayType(int t) {

    }

    @Override
    public void Reject() {

    }
```

```
    @Override
    public void Cancel() {
        outputProcessor.CancelMsg();

    }

    @Override
    public void Approved() {

    }

    @Override
    public void StartPump() {

    }

    @Override
    public void Pump() {

    }

    @Override
    public void StopPump() {

    }

    @Override
    public void SelectGas(int g) {
        outputProcessor.SetPrice(g);
        System.out.println("\nFollow the instructions to continue using
GasPump..");

    }

    @Override
    public void Receipt() {

    }

    @Override
    public void NoReceipt() {

    }
}



package State;

/**
```

```
 * Created by Sharel on 4/19/2017.
 */

/*State class that directs outputProcessor to implement
*StartPump meta Event with meta action SetInItialValues()
 * and ReadyMsg()*/
public class State_S4 extends State {
    @Override
    public void Activate() {

    }

    @Override
    public void Start() {

    }

    @Override
    public void PayType(int t) {

    }

    @Override
    public void Reject() {

    }

    @Override
    public void Cancel() {

    }

    @Override
    public void Approved() {

    }

    @Override
    public void StartPump() {

        outputProcessor.SetInItialValues();
        outputProcessor.ReadyMsg();
    }

    @Override
    public void Pump() {

    }

    @Override
    public void StopPump() {
```

```
    }

    @Override
    public void SelectGas(int g) {

    }

    @Override
    public void Receipt() {

    }

    @Override
    public void NoReceipt() {

    }
}
```

```
package State;

/**
 * Created by Sharel on 4/19/2017.
 */

/*State class that directs outputProcessor to implement
*Pump & StopPump meta Event with meta action PumpGasUnit(),GasPumpedMsg
 * and StopMsg() respectively*/

public class State_S5 extends State{
    @Override
    public void Activate() {

    }

    @Override
    public void Start() {

    }

    @Override
    public void PayType(int t) {

    }

    @Override
    public void Reject() {

    }
```

```
    @Override
    public void Cancel() {

    }

    @Override
    public void Approved() {

    }

    @Override
    public void StartPump() {

    }

    @Override
    public void Pump() {
        outputProcessor.PumpGasUnit();
        outputProcessor.GasPumpedMsg();
    }

    @Override
    public void StopPump() {
        outputProcessor.StopMsg();
    }

    @Override
    public void SelectGas(int g) {

    }

    @Override
    public void Receipt() {

    }

    @Override
    public void NoReceipt() {

    }
}



package State;

/**
 * Created by Sharel on 4/19/2017.
 */
```

```
/*State class that directs outputProcessor to implements
*Receipt & NoReceipt meta Event with meta
* action PrintReceipt,ReturnCash respectively*/

public class State_S6 extends State{
    @Override
    public void Activate() {

    }

    @Override
    public void Start() {

    }

    @Override
    public void PayType(int t) {

    }

    @Override
    public void Reject() {

    }

    @Override
    public void Cancel() {

    }

    @Override
    public void Approved() {

    }

    @Override
    public void StartPump() {

    }

    @Override
    public void Pump() {

    }

    @Override
    public void StopPump() {

    }

    @Override
```

```
    public void SelectGas(int g) {

    }

    @Override
    public void Receipt() {
        outputProcessor.PrintReceipt();
        outputProcessor.ReturnCash();

    }


    @Override
    public void NoReceipt() {
        System.out.println("No Receipt will be printed for this transaction!");
        outputProcessor.ReturnCash();

    }
}
```

Strategy Design Pattern:

```
package Strategy;

import DataStore.*;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing StoreData() strategy of the meta
action*/
public abstract class A1_StoreData {

   DataStore ds;

   public void setdata(DataStore x){
      ds=x;
   }
    public abstract void StoreData();

}


package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */
```

```
/*Concrete Strategy class implementing StoreData() strategy
* for storing gas prices of GasPump-1 for super & regular
* into temp variables of DataStore**/
public class A1_StoreData_1 extends A1_StoreData {


    public void StoreData() {
        float a, b;
        a=ds.getTempa();
        ds.setreg_price(a);
        b=ds.getTempb();
        ds.setsup_price(b);


    }

}


package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing StoreData() strategy
* for storing gas prices of GasPump-2 for super,premium & regular
* into temp variables of DataStore
* */
public class A1_StoreData_2 extends A1_StoreData {


    public void StoreData() {
        int a, b,c;
        a=ds.getTemp_a1();
        ds.setreg1_price(a);
        b=ds.getTemp_b1();
        ds.setpre_price(b);
        c=ds.getTemp_c1();
        ds.setsup1_price(c);
    }
}


package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing PayMsg() strategy of the meta action*/
```

```
public abstract class A2_PayMsg {


    public abstract void PayMsg();
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing PayMsg() strategy
* for displaying message while payment mode is chosen as credit*/

public class A2_PayMsg_1 extends A2_PayMsg{

    @Override
    public void PayMsg() {
        System.out.println("Paid using Credit.");
    }
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing PayMsg() strategy
* for displaying message while payment mode is chosen as cash*/

public class A2_PayMsg_2 extends A2_PayMsg{
    @Override
    public void PayMsg() {
        System.out.println("Paid using Cash.");
    }
}
```

```
package Strategy;

import DataStore.DataStore;

/**
 * Created by Sharel on 4/19/2017.
 */
/*Abstract Strategy class for implementing StoreCash() strategy of the meta
```

```
action*/
public abstract class A3_StoreCash {

    DataStore ds;
    public void setdata(DataStore x){
        ds=x;
    }
    public abstract void StoreCash();
}



package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing StoreCash() strategy
* by storing the cash from temp variable
* while payment mode is chosen as cash*/

public class A3_StoreCash_2 extends A3_StoreCash {
    @Override
    public void StoreCash() {
        float c =ds.getTempCash();
        ds.setcash(c);
    }
}


package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing DisplayMenu() strategy of the meta
action*/
public abstract class A4_DisplayMenu {

    public abstract void DisplayMenu();
}



package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing DisplayMenu() strategy
```

```
* for displaying list of selections
* while payment mode is chosen as credit*/

public class A4_DisplayMenu_1 extends A4_DisplayMenu {
    @Override
    public void DisplayMenu() {


        System.out.println("\n ***********DISPLAY MENU*************");
        System.out.println("\n Select option 6 for Super Fuel and then select
option 8 to Start the Pump");
        System.out.println("\n Select option 7 for Regular Fuel option 8 to Start
the Pump");
    }
}


package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing DisplayMenu() strategy
* for displaying list of selections
* while payment mode is chosen as cash*/

public class A4_DisplayMenu_2 extends A4_DisplayMenu{

    @Override
    public void DisplayMenu() {


        System.out.println("\n ***********DISPLAY MENU*************");
        System.out.println("\n Select option 4 for Super Fuel and then select
option 7 to Start the Pump");
        System.out.println("\n Select option 5 for Premium Fuel option 7 to Start
the Pump");
        System.out.println("\n Select option 6 to Regular Fuel option 7 to Start
the Pump");
    }
}



package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing RejectMsg() strategy of the meta
action*/
```

```
public abstract class A5_RejectMsg {

    public abstract void RejectMsg();
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing RejectMsg() strategy
* for displaying rejection message
* while credit card is rejected*/

public class A5_RejectMsg_1 extends A5_RejectMsg {
    @Override
    public void RejectMsg() {
        System.out.println("Credit card rejected!");
    }
}
```

```
package Strategy;

import DataStore.DataStore;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing SetPrice(g) strategy of the meta
action*/

public abstract class A6_SetPrice {
    DataStore ds;
    public void setdata(DataStore x){
        ds=x;
    }
    public abstract void SetPrice(int g);
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing SetPrice() strategy
* for setting the price to the price of gas chosen during selections
```

```
* while payment mode is chosen as credit*/

public class A6_SetPrice_1 extends A6_SetPrice {
    @Override
    public void SetPrice(int g) {
        float a=ds.getreg_price();
        float b=ds.getsup_price();
        if( g== 1)
            ds.setprice(a);
        else if (g == 2)
            ds.setprice(b);

    }
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */


/*Concrete Strategy class implementing SetPrice() strategy
* for setting the price to the price of gas chosen during selections
* while payment mode is chosen as cash*/

public class A6_SetPrice_2 extends A6_SetPrice {
    @Override
    public void SetPrice(int g) {
        int a=ds.getreg1_price();
        int b=ds.getpre_price();
        int c=ds.getsup1_price();
        if( g== 1)
            ds.setprice1(a);
        else if (g == 2)
            ds.setprice1(c);
        else if(g==3){
            ds.setprice1(b);
        }
    }
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing ReadyMsg() strategy of the meta
action*/
```

```
public abstract class A7_ReadyMsg {
    public abstract void ReadyMsg();
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing ReadyMsg() strategy
* for displaying the message when it is ready for pumping
* while payment mode is chosen as credit/cash*/

public class A7_ReadyMsg_1 extends A7_ReadyMsg {
    @Override
    public void ReadyMsg() {
        System.out.println("Ready for pumping.");
    }
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */
public class A7_ReadyMsg_2 extends A7_ReadyMsg {
    @Override
    public void ReadyMsg() {
        System.out.println("Ready for pumping.");
    }
}
```

```
package Strategy;

import DataStore.DataStore;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing SetInitialValues() strategy of the meta
action*/

public abstract class A8_SetInitialValues {
    DataStore ds;
    public void setdata(DataStore x){
        ds=x;
    }
    public abstract void SetInitialValues();
```

```
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing SetInitialValues() strategy
* for setting the total to 0 and units of gallons to 0 before Pumping
* while payment mode is chosen as credit*/


public  class A8_SetInitialValues_1 extends A8_SetInitialValues{
    @Override
    public void SetInitialValues() {
        ds.setG(0);
        ds.settotal(0);
    }
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/22/2017.
 */

/*Concrete Strategy class implementing SetInitialValues() strategy
* for setting the total to 0 and units of liter to 0 before Pumping
* while payment mode is chosen as cash*/

public class A8_SetInitialValues_2 extends A8_SetInitialValues{
    @Override
    public void SetInitialValues() {
        ds.setL(0);
        ds.settotal(0);
    }
}
```

```
package Strategy;

import DataStore.DataStore;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing PumpGasUnit() strategy of the meta
action*/

public abstract class A9_PumpGasUnit  {
```

```
    DataStore ds;
    float l,g;
    public void setdata(DataStore dOb){
        ds=dOb;
    }

    public abstract void PumpGasUnit();
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing PumpGasUnit() strategy
*to pump one unit of gas and calculate total based on units disposed
* while payment mode is chosen as credit*/

public class A9_PumpGasUnit_1 extends A9_PumpGasUnit {
    @Override
    public void PumpGasUnit() {
        //dispose units of gas
         g=ds.getG();
        g=g+1;
        float total;
        float price = ds.getprice();
        total =price *g;
        ds.setG(g);
        ds.settotal(total);
        System.out.println("Amount for gas disposed: "+total);
    }
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/22/2017.
 */
/*Concrete Strategy class implementing PumpGasUnit() strategy
*to pump one unit of gas and calculate total based on units disposed
* while payment mode is chosen as cash*/

public class A9_PumpGasUnit_2 extends A9_PumpGasUnit {
    @Override
    public void PumpGasUnit() {

        System.out.println("\nIn PumpGasUnit");
         l=ds.getL();
        l=l+1;
        float total;
        float price = ds.getprice1();
```

```
        total =price*l;
        ds.setL(l);
        ds.settotal(total);
        System.out.println("Amount for gas disposed: "+total);
    }

}
```

```
package Strategy;

import DataStore.DataStore;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing GasPumpedMsg() strategy of the meta
action*/

public abstract class A10_GasPumpedMsg {
    DataStore ds;
    public void setdata(DataStore dOb){
        ds=dOb;
    }
    public abstract void GasPumpedMsg();

}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing GasPumpedMsg() strategy
*to display units of gas  disposed
* while payment mode is chosen as credit*/

public class A10_GasPumpedMsg_1 extends A10_GasPumpedMsg {
    @Override
    public void GasPumpedMsg() {
        float g = ds.getG();
        System.out.println("Amount of Gas pumped in units: " +g);

    }
}
```

```
package Strategy;
```

```
/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing GasPumpedMsg() strategy
*to display units of gas  disposed
* while payment mode is chosen as cash*/

public class A10_GasPumpedMsg_2 extends A10_GasPumpedMsg {
    @Override
    public void GasPumpedMsg() {
        float l = ds.getL();
        System.out.println("The Gas pump has successfully pumped units :" +l);

    }
}



package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing StopMsg() strategy of the meta action*/

public abstract class A11_StopMsg {
    public abstract void StopMsg();
}


package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing StopMsg() strategy
*to display the message when gas has stopped pumping
* while payment mode is chosen as credit*/

public class A11_StopMsg_1 extends A11_StopMsg {
    @Override
    public void StopMsg() {
        System.out.println("Gas Pump has been stopped.");
    }
}


package Strategy;

/**
```

```
 * Created by Sharel on 4/23/2017.
 */

/*Concrete Strategy class implementing StopMsg() strategy
*to display the message when gas has stopped pumping
* while payment mode is chosen as cash*/

public class A11_StopMsg_2 extends A11_StopMsg {
    @Override
    public void StopMsg() {
        System.out.println("\nGas Pump has been stopped.\n");
        System.out.println("\nChoose to print the Receipt?");
    }
}
```

```
package Strategy;

import DataStore.DataStore;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing PrintReceipt() strategy of the meta
action*/

public abstract  class A12_PrintReceipt {
    DataStore ds;
    public void setdata(DataStore dOb){
        ds=dOb;
    }
    public abstract void PrintReceipt();
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing PrintReceipt() strategy
*to print Receipt and display total amount for gas disposed
* while payment mode is chosen as credit/cash*/

public class A12_PrintReceipt_1 extends A12_PrintReceipt{
    @Override
    public void PrintReceipt() {
        System.out.println("\n\tPrinting receipt...\n");
```

```
        float total = ds.gettotal();
        System.out.printf("\n\tThe total amount for the gas that has been pumped
is: "+total);

    }
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing CancelMsg() strategy of the meta
action*/

public abstract class A13_CancelMsg {
    public abstract void CancelMsg();
}
```

```
package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */


/*Concrete Strategy class implementing CancelMsg() strategy
*to print cancellation message when the operation has been cancelled
* while payment mode is chosen as credit/cash*/

public class A13_CancelMsg_1 extends A13_CancelMsg{
    @Override
    public void CancelMsg() {
        System.out.println("Operation cancelled successfully.");
    }
}
```

```
package Strategy;

import DataStore.DataStore;
```

```
import javax.xml.crypto.Data;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Abstract Strategy class for implementing ReturnCash() strategy of the meta
action*/
public abstract class A14_ReturnCash {
    DataStore dataStore;
    public void setdata(DataStore ds){
        dataStore=ds;

    }
    public abstract void ReturnCash();
}


package Strategy;

/**
 * Created by Sharel on 4/19/2017.
 */

/*Concrete Strategy class implementing ReturnCash() strategy
*to return the remaining cash, if any  and display a info message
* while payment mode is chosen as cash*/

public class A14_ReturnCash_2 extends A14_ReturnCash {
    @Override
    public void ReturnCash() {

        float cash = dataStore.getcash();
        float total = dataStore.gettotal();

        if(cash!=0 ) {
            if (cash > total) {
                float ret = cash - total;
                System.out.println("\nReturning remaining cash:\n " + ret);
            } else {
                System.out.println("\nNothing to return!");
            }
        }

    }
}
```

Driver:

```
package Driver;
```

```
import AbstractFactory.CF_GasPump1;
import AbstractFactory.CF_GasPump2;
import DataStore.DataStore;
import InputProcessor.GasPump_1;
import InputProcessor.GasPump_2;
import MdaEfsm.MdaEfsm;
import OutputProcessor.OutputProcessor;
import State.*;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

/**
 * Created by Sharel on 4/22/2017.
 */
public class Driver {
    static Scanner input=new Scanner(System.in);
    static BufferedReader buf=new BufferedReader(new
InputStreamReader(System.in));

    public static void main(String[] args) throws NumberFormatException,
IOException
    {
        System.out.println("\n\t\t\t!@!@!@Welcome to the Gas Pump
Station!@!@!@!@\t\t");
        System.out.println("\n*****Choose the suitable Gas Pump from the list
below*****\t\t");
        System.out.println("\n\t 1. GasPump-1 Pay by credit \t");
        System.out.println("\n\t 2. GasPump-2 Pay by Cash \t");
        System.out.println("\n Key in your choice \t\t ");
        int choice = input.nextInt();

        switch (choice){
            case 1:
            {
                MdaEfsm mdaEfsm =new MdaEfsm();
                DataStore dataStore;
                OutputProcessor outputProcessor=new OutputProcessor();
                GasPump_1 gasPump_1=new GasPump_1();
                CF_GasPump1 CF_gasPump1 = new CF_GasPump1();

                //State class references
                State_Start state_start=new State_Start();
                State_S0 state_s0=new State_S0();
                State_S1 state_s1=new State_S1();
                State_S2 state_s2=new State_S2();
                State_S3 state_s3 = new State_S3();
                State_S4 state_s4 = new State_S4();
                State_S5 state_s5 = new State_S5();
                State_S6 state_s6 = new State_S6();

                //initialisation
```

```
                    dataStore= CF_gasPump1.getDataStore();
                    gasPump_1.setAbstractFactory(CF_gasPump1);
                    gasPump_1.setDataStore(dataStore);
                    gasPump_1.setMdaEfsm(mdaEfsm);

                state_start.setOutputProcessor(outputProcessor);
                state_start.setStateId(0);

                state_s0.setOutputProcessor(outputProcessor);
                state_s0.setStateId(1);

                state_s1.setOutputProcessor(outputProcessor);
                state_s1.setStateId(2);

                state_s2.setOutputProcessor(outputProcessor);
                state_s2.setStateId(3);

                state_s3.setOutputProcessor(outputProcessor);
                state_s3.setStateId(4);

                state_s4.setOutputProcessor(outputProcessor);
                state_s4.setStateId(5);

                state_s5.setOutputProcessor(outputProcessor);
                state_s5.setStateId(6);

                state_s6.setOutputProcessor(outputProcessor);
                state_s6.setStateId(7);



                //Setting up concrete factory
                outputProcessor.setdata(dataStore);
                outputProcessor.setfactory(CF_gasPump1);

                mdaEfsm.setState(state_start);

                //Setting up the states
                State[] stateList=
{state_start,state_s0,state_s1,state_s2,state_s3,state_s4,state_s5,state_s6};
                mdaEfsm.setStatesList(stateList);

                String input=null;
                int ch;

                while(true){
                    System.out.println("\n\n~~~~~~~~~~~~~~~~~~~Choose from the
below options to utilize GasPump-1 facilities~~~~~~~~\t\t");
                    System.out.println("\n\t\t 0.\t Activate(Regular,Super) ");
                    System.out.println("\n\t\t 1.\t Start ");
                    System.out.println("\n\t\t 2.\t PayCredit ");
                    System.out.println("\n\t\t 3.\t Reject");
                    System.out.println("\n\t\t 4.\t Cancel");
```

```
                    System.out.println("\n\t\t 5.\t Approved");
                    System.out.println("\n\t\t 6.\t Super");
                    System.out.println("\n\t\t 7.\t Regular");
                    System.out.println("\n\t\t 8.\t StartPump");
                    System.out.println("\n\t\t 9.\t PumpGallon");
                    System.out.println("\n\t\t 10.\t StopPump");
                    System.out.println("\n\t\t Press any key to exit \n\n");
                    input=buf.readLine();

                    ch=Integer.parseInt(input);

                    switch(ch)
                    {
                        case 0: System.out.println(" \n\n Enter the value of
Regular(a) to activate");
                            float a=Float.parseFloat(buf.readLine());
                            System.out.println("\n\n Enter the value of Super(b)
to activate");
                            float b=Float.parseFloat(buf.readLine());
                            gasPump_1.Activate(a,b);      //calls method activate
in GasPump1
                            break;

                        case 1: gasPump_1.Start();
                            break;

                        case 2: gasPump_1.PayCredit();
                            break;

                        case 3: gasPump_1.Reject();
                            break;

                        case 4: gasPump_1.Cancel();
                            break;

                        case 5: gasPump_1.Approved();
                            break;

                        case 6:gasPump_1.Super();
                            break;

                        case 7:gasPump_1.Regular();
                            break;

                        case 8: gasPump_1.StartPump();
                            break;
                        case 9: gasPump_1.PumpGallon();
                            break;
                        case 10: gasPump_1.StopPump();
                            break;
                        default:
                            System.out.println("\n Please enter a correct option
from the list");
```

```
            }
        }


    }

    case 2:
        MdaEfsm mdaEfsm =new MdaEfsm();
        DataStore dataStore;
        OutputProcessor outputProcessor=new OutputProcessor();
        GasPump_2 gasPump_2=new GasPump_2();
        CF_GasPump2 CF_gasPump2 = new CF_GasPump2();

        //State class references
        State_Start state_start=new State_Start();
        State_S0 state_s0=new State_S0();
        State_S1 state_s1=new State_S1();
        State_S2 state_s2=new State_S2();
        State_S3 state_s3 = new State_S3();
        State_S4 state_s4 = new State_S4();
        State_S5 state_s5 = new State_S5();
        State_S6 state_s6 = new State_S6();

        //initialisation
        dataStore= CF_gasPump2.getDataStore();
        gasPump_2.setAbstractFactory(CF_gasPump2);
        gasPump_2.setDataStore(dataStore);
        gasPump_2.setMdaEfsm(mdaEfsm);

        state_start.setOutputProcessor(outputProcessor);
        state_start.setStateId(0);

        state_s0.setOutputProcessor(outputProcessor);
        state_s0.setStateId(1);

        state_s1.setOutputProcessor(outputProcessor);
        state_s1.setStateId(2);

        state_s2.setOutputProcessor(outputProcessor);
        state_s2.setStateId(3);

        state_s3.setOutputProcessor(outputProcessor);
        state_s3.setStateId(4);

        state_s4.setOutputProcessor(outputProcessor);
        state_s4.setStateId(5);

        state_s5.setOutputProcessor(outputProcessor);
        state_s5.setStateId(6);

        state_s6.setOutputProcessor(outputProcessor);
        state_s6.setStateId(7);
```

```
                //Setting up concrete factory
                outputProcessor.setdata(dataStore);
                outputProcessor.setfactory(CF_gasPump2);

                mdaEfsm.setState(state_start);

                //Setting up the states
                State[] stateList=
{state_start,state_s0,state_s1,state_s2,state_s3,state_s4,state_s5,state_s6};
                mdaEfsm.setStatesList(stateList);

                String input=null;
                int ch;

                while(true){
                    System.out.println("\n\n~~~~~~~~~~~~~~~~~~~Choose from the
below options to utilize GasPump-2 facilities~~~~~~~~\t\t");
                    System.out.println("\n\t\t 0.\t
Activate(Regular,Premium,Super) ");
                    System.out.println("\n\t\t 1.\t Start");
                    System.out.println("\n\t\t 2.\t PayCash");
                    System.out.println("\n\t\t 3.\t Cancel");
                    System.out.println("\n\t\t 4.\t Super");
                    System.out.println("\n\t\t 5.\t Premium");
                    System.out.println("\n\t\t 6.\t Regular");
                    System.out.println("\n\t\t 7.\t StartPump");
                    System.out.println("\n\t\t 8.\t PumpLiter");
                    System.out.println("\n\t\t 9.\t Stop");
                    System.out.println("\n\t\t 10.\tReceipt");
                    System.out.println("\n\t\t 11.\tNoReceipt");
                    System.out.println("\n\t\t Press any key to exit \n\n");
                    input=buf.readLine();
                    ch=Integer.parseInt(input);

                    switch(ch)
                    {
                        case 0: System.out.println(" \n\n Enter the value of
Regular(a) to activate\n");
                                int a=Integer.parseInt(buf.readLine());
                                System.out.println("\n\n Enter the value of Premium(b)
to activate\n");
                                int b=Integer.parseInt(buf.readLine());
                                System.out.println("\n\nEnter the value of Super(c) to
activate\n");
                                int c = Integer.parseInt(buf.readLine());
                                gasPump_2.Activate(a,b,c);
                                break;

                        case 1: gasPump_2.Start();
                            break;
```

```
                        case 2:
                            System.out.println("\n Enter the amount to pay:\n");
                            float amt=Float.parseFloat(buf.readLine());
                            gasPump_2.PayCash(amt);
                            break;

                        case 3: gasPump_2.Cancel();
                            break;

                        case 4: gasPump_2.Super();
                               break;

                        case 5: gasPump_2.Premium();
                            break;

                        case 6: gasPump_2.Regular();
                            break;

                        case 7: gasPump_2.StartPump();
                            break;

                        case 8:gasPump_2.PumpLiter();
                            break;

                        case 9: gasPump_2.Stop();
                            break;

                        case 10:gasPump_2.Receipt();
                            break;

                        case 11:gasPump_2.NoReceipt();
                            break;
                        default:
                            System.out.println("\nPlease enter a correct option
from the list");
                    }
                }
            }
        }
}
```