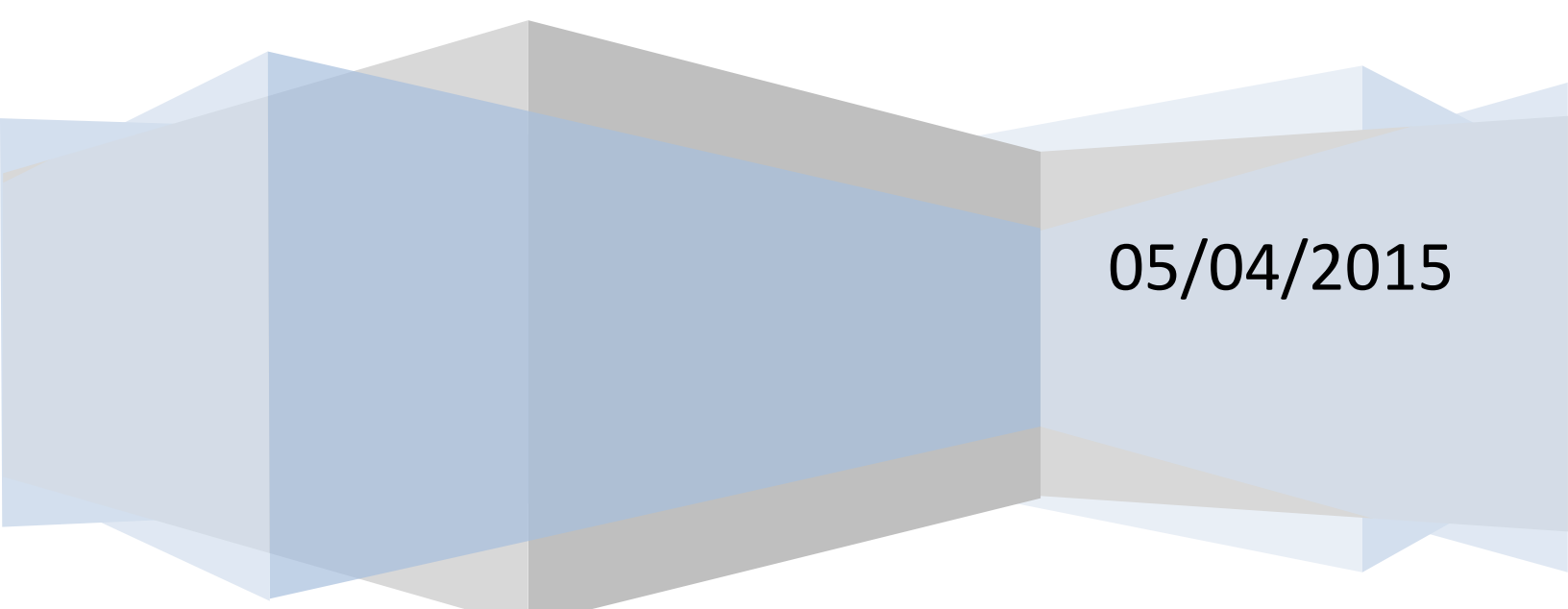


# Project Report

Shouvik Dutta (A20346105)



05/04/2015

**MDA-EFSM Events:**

Activate()  
Start()  
PayCredit()  
PayCash()  
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  
SetW(int k) // set value for credit/cash flag  
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) 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

## Operations of the Input Processor

### (GasPump-1)

```
Activate(int a) {
    if (a>0) {
        d->temp_a=a;
        m->Activate()
    }
}
Start() {
    m->Start();
}
PayCredit() {
    m->PayCredit();
}
Reject() {
    m->Reject();
}
Cancel() {
    m->Cancel();
}
Approved() {
    m->Approved();
}
PayCash(int c) {
    if (c>0) {
        d->temp_c=c;
        m->PayCash();
    }
}

StartPump() {
    m->SelectGas(1);
    m->StartPump();
}
PumpGallon() {
    if (d->w==1) m->Pump();
    else if (d->w==0) {
        if (d->cash<(d->G+1)*d->price) {
            m->StopPump();
            m->Receipt()
        }
        else m->Pump();
    }
}
```

```

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

```

Notice:

cash: contains the value of cash deposited

price: contains the price of the selected gas

G: contains the number of gallons already pumped

w: cash/credit flag

cash , G, price, w are in the data store

m: is a pointer to the MDA-EFSM object

d: is a pointer to the Data Store object

### **(GasPump-2)**

```

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

### **(GasPump-3)**

```
Activate(float a, float b) {  
  if ((a>0)&&(b>0)) {  
    d->temp_a=a;  
    d->temp_b=b;  
    m->Activate()  
  }  
}  
Start() {  
  m->Start();  
}  
PayCash(float c) {  
  if (c>0) {  
    d->temp_c=c;  
    m->PayCash()  
  }  
}  
Cancel() {  
  m->Cancel();  
}  
Premium() {  
  m->SelectGas(2);  
}  
Regular() {  
  m->SelectGas(1);  
}  
StartPump() {
```

```

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

```

Notice:

cash: contains the value of cash deposited

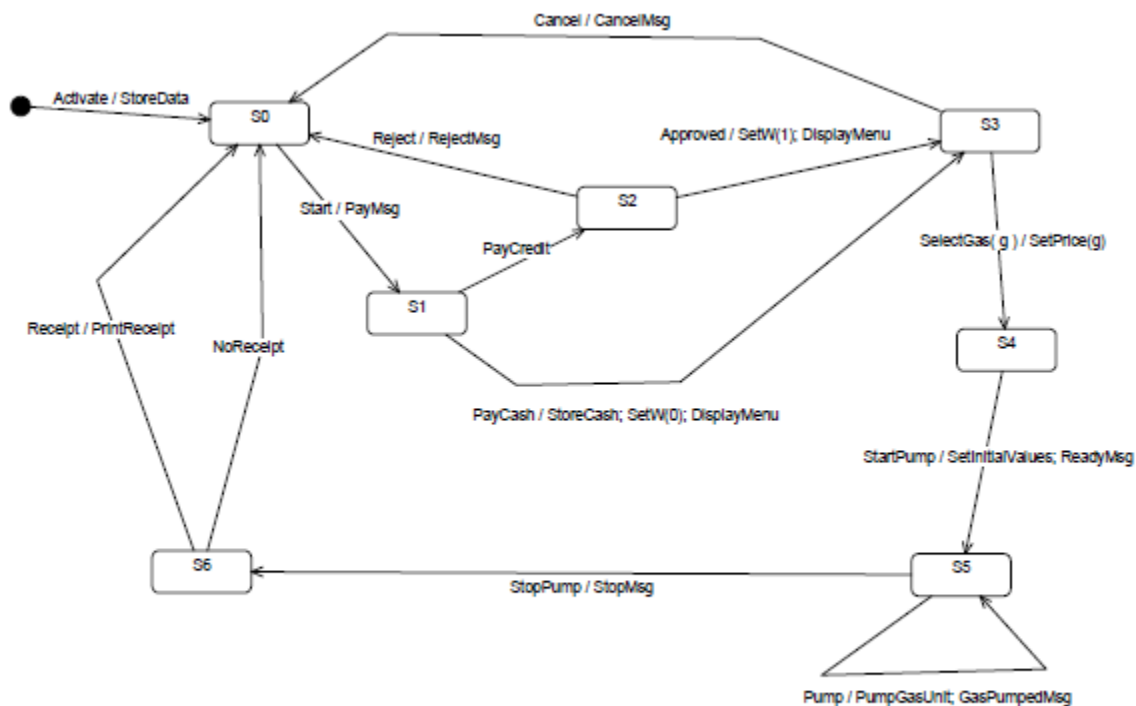
price: contains the price of the selected gas

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



MDA-EFSM for Gas Pumps

**Class Structure:****GasPump Classes:**

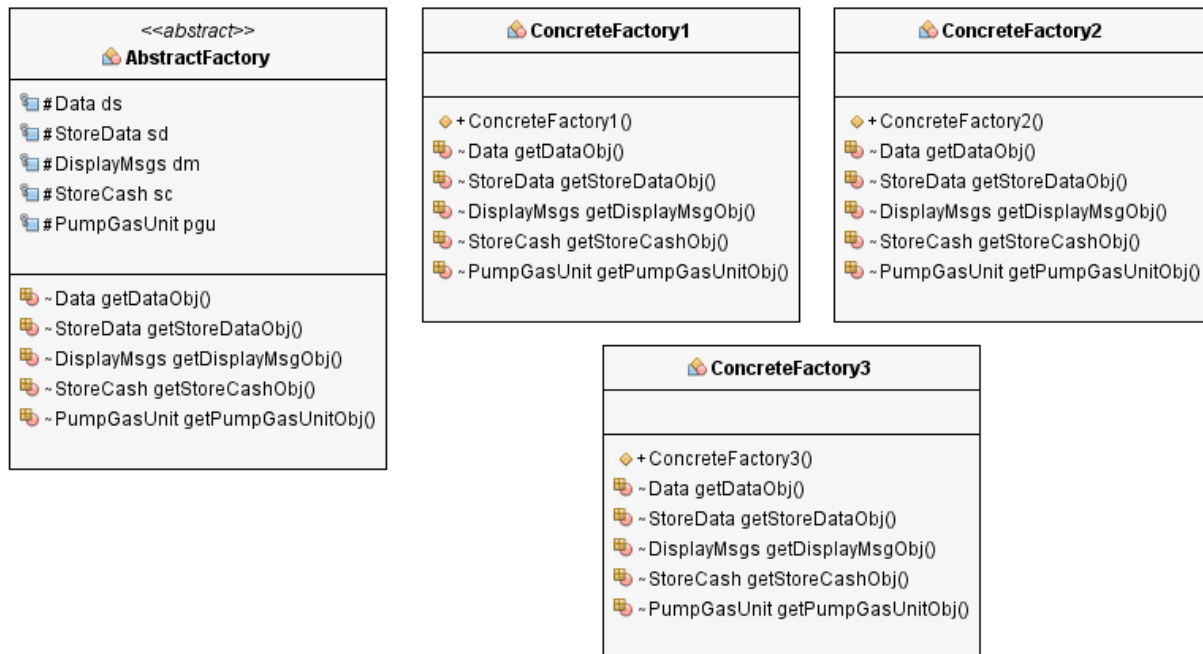
GasPump1	GasPump2	GasPump3
<ul style="list-style-type: none"> <li>- MDA_EFSM m</li> <li>- Data d</li> <li>- AbstractFactory af</li> </ul>	<ul style="list-style-type: none"> <li>- MDA_EFSM m</li> <li>- Data d</li> <li>- AbstractFactory af</li> </ul>	<ul style="list-style-type: none"> <li>- MDA_EFSM m</li> <li>- Data d</li> <li>- AbstractFactory af</li> </ul>
<ul style="list-style-type: none"> <li>+ GasPump1(MDA_EFSM mobj)</li> <li>~void Activate(int a)</li> <li>~void Start()</li> <li>~void PayCredit()</li> <li>~void PayCash(int c)</li> <li>~void Reject()</li> <li>~void Cancel()</li> <li>~void Approved()</li> <li>~void StartPump()</li> <li>~void PumpGallon()</li> <li>~void StopPump()</li> </ul>	<ul style="list-style-type: none"> <li>+ GasPump2(MDA_EFSM mobj)</li> <li>+void Activate(float a, float b)</li> <li>+void Start()</li> <li>~void PayCredit()</li> <li>~void Reject()</li> <li>~void Cancel()</li> <li>~void Approved()</li> <li>~void Super()</li> <li>~void Regular()</li> <li>~void StartPump()</li> <li>~void PumpGallon()</li> <li>~void StopPump()</li> </ul>	<ul style="list-style-type: none"> <li>+ GasPump3(MDA_EFSM mobj)</li> <li>+void Activate(float a, float b)</li> <li>+void Start()</li> <li>~void PayCash(float c)</li> <li>~void Cancel()</li> <li>~void Premium()</li> <li>~void Regular()</li> <li>~void StartPump()</li> <li>~void PumpLiter()</li> <li>~void StopPump()</li> <li>~void Receipt()</li> <li>~void NoReceipt()</li> </ul>

**GasPump1 Class:** This class is used to provide all the functionality which are specific to GasPump1.

**GasPump2 Class:** This class is used to provide all the functionality which are specific to GasPump2.

**GasPump3 Class:** This class is used to provide all the functionality which are specific to GasPump3.

**Abstract Factory and Concrete Factory Classes:** (These Classes are involved and help in implementing Abstract Factory Pattern)



**AbstractFactory Class:** This is an abstract class which exposes functions and member variables specific to a particular GasPump. The sole purpose of this class is to help create various Concrete Factories based on the Abstract Factory Pattern.

**ConcreteFactory1:** This class extends AbstractFactory Class and implements all the functions which are specific to GasPump1's requirement.

**ConcreteFactory2:** This class extends AbstractFactory Class and implements all the functions which are specific to GasPump2's requirement.

**ConcreteFactory3:** This class extends AbstractFactory Class and implements all the functions which are specific to GasPump3's requirement.



**State related Classes:** (These Classes are involved and helps in implementing State Pattern using decentralized approach)



**State Class:** This is an abstract class which implements the State pattern in our project. It exposes functions and also has a pointer to the Output Processor object. As I have taken the decentralized approach, we also have a pointer to MDA\_EFSM object which is used for the purpose of changing its state.

**Start Class:** This class implements the Activate function of the State class. This Activate function internally calls the StoreData action function on OutputProcessor object and changes state to S0.

**S0 Class:** This class implements the Start function of the State class. This Start function internally calls the PayMsg action function on OutputProcessor object and changes state to S1.

**S1 Class:** This class implements the PayCash and PayCredit function of the State class. The PayCash function internally calls the StoreCash, SetW(0) and DisplayMenu action functions on OutputProcessor object and changes state to S3. The PayCredit function internally changes the state to State S2.

**S2 Class:** This class implements the Approved and Reject function of the State class. The Approved function internally calls the SetW(1) and DisplayMenu action functions on OutputProcessor object and changes state to S3. The Reject function internally calls the RejectMsg action functions on OutputProcessor object and changes the state to State S0.

**S3 Class:** This class implements the SelectGas(g) and Cancel function of the State class. The SelectGas(g) function internally calls the SetPrice(g) function on OutputProcessor object and

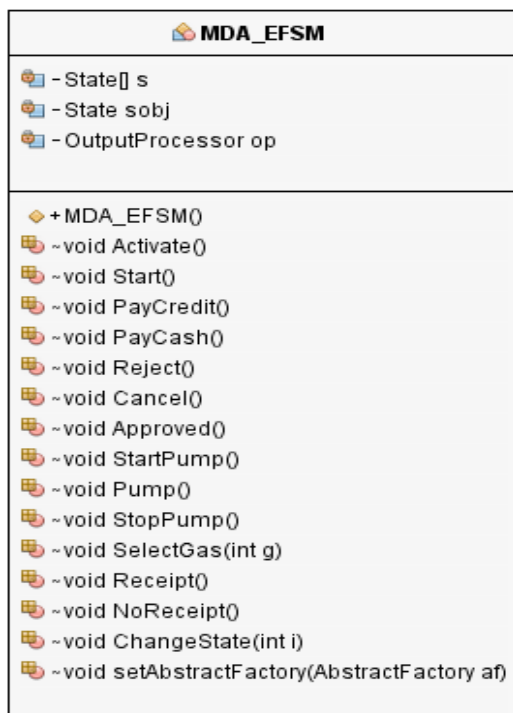
changes state to S4. The Cancel function internally calls the CancelMsg action function on OutputProcessor object and changes the state to State S0.

**S4 Class:** This class implements the StartPump function of the State class. The StartPump function internally calls the SetInitialValues and ReadyMsg functions on OutputProcessor object and changes state to S5.

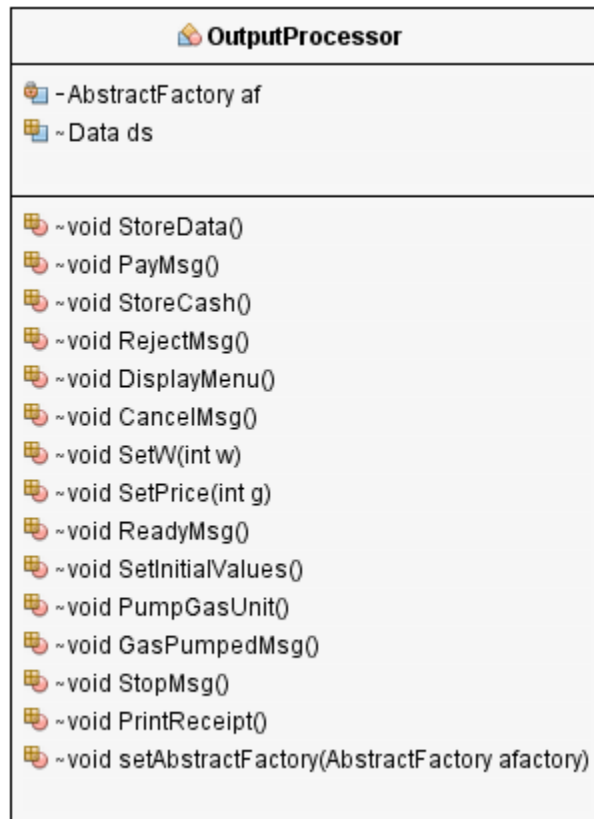
**S5 Class:** This class implements the Pump and StopPump function of the State class. The Pump function internally calls the PumpGasUnit and GasPumpedMsg functions on OutputProcessor object. The StopPump function internally calls the StopMsg functions on OutputProcessor object and changes the state to State S6.

**S6 Class:** This class implements the Pump and StopPump function of the State class. The Pump function internally calls the Receipt and NoReceipt functions on OutputProcessor object. The Receipt function internally calls the PrintReceipt function on OutputProcessor object and changes the state to State S0 where as the NoReceipt function directly changes the state to State S0.

### MDA-EFSM Class:

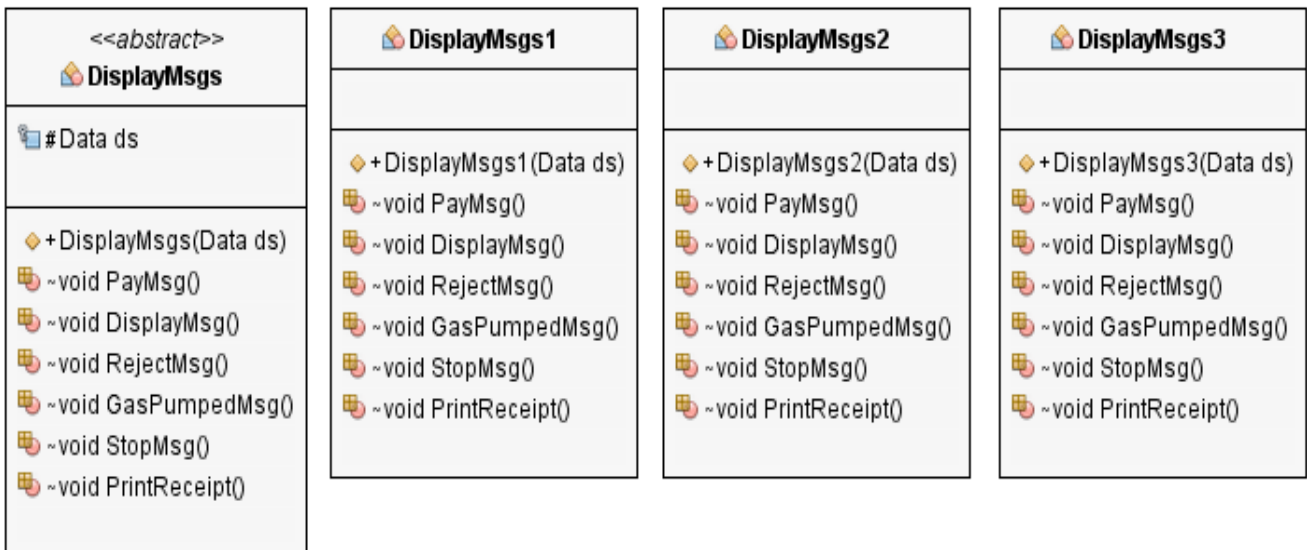


**MDA\_EFSM Class:** This class contains all the platform independent events which are being called by various flavors of GasPump(Input Processor) and based on the current state this class perform an action of the OutputProcessor. As I used decentralized approach MDA\_EFSM class also has a changeState method which is used by the state classes to change the current State.

**OutputProcessor Class:**

**OutputProcessor Class:** This class provides implementation for the actions of the MDA-EFSM design. Unlike MDA-EFSM this class has platform specific actions functions.

**Abstract Class DisplayMsgs and all Concrete DisplayMsgs Class:** (These Classes are involved in implementation of Strategy Pattern)



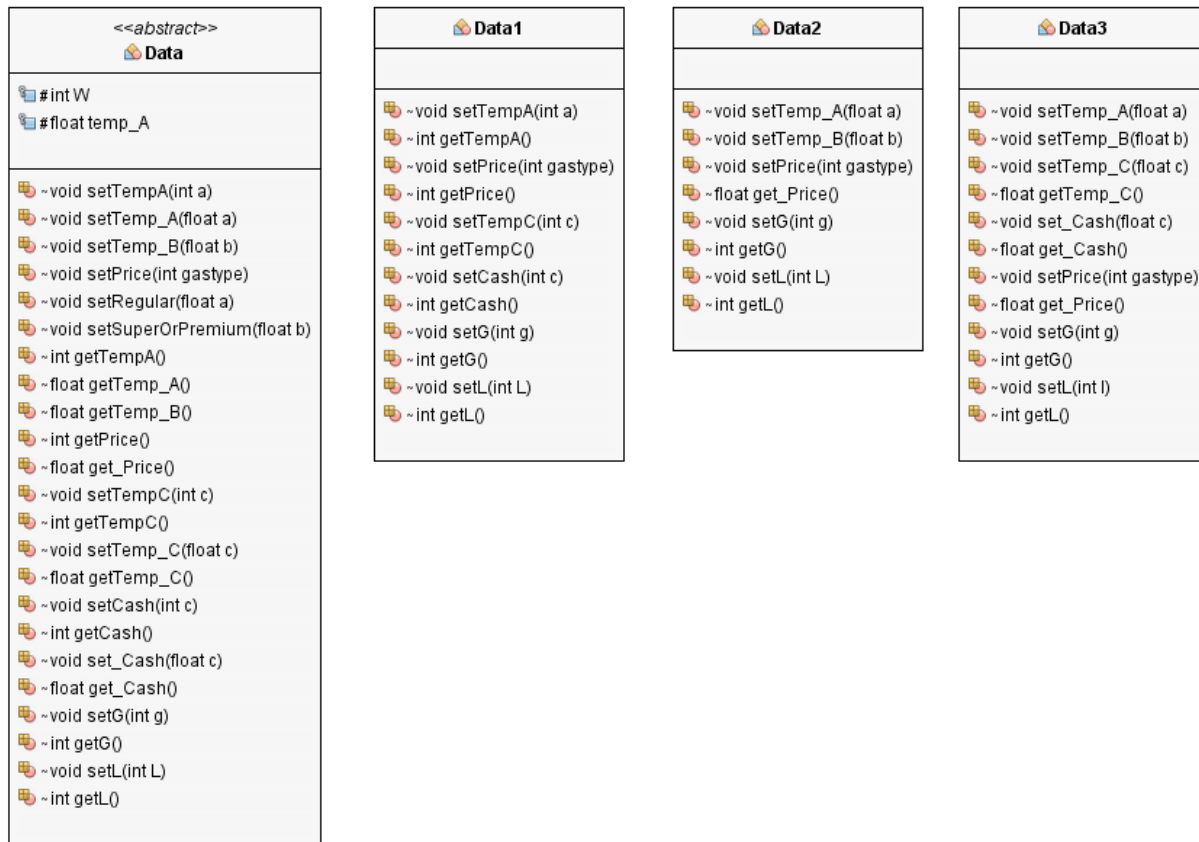
DisplayMsgs Class: This is an abstract class which exposes functions specific to a particular GasPump. The functions that it exposes are various flavors of action function being called using Strategy Pattern by OutputProcessor based on the type of GasPump.

DisplayMsgs1 Class: This class extends DisplayMsgs Class and implements all the functions like DisplayMsgs, PayMsg etc. which are specific to GasPump1's requirement.

DisplayMsgs2 Class: This class extends DisplayMsgs Class and implements all the functions like DisplayMsgs, PayMsg etc. which are specific to GasPump2's requirement.

DisplayMsgs3 Class: This class extends DisplayMsgs Class and implements all the functions like DisplayMsgs, PayMsg etc. which are specific to GasPump3's requirement.

**Abstract Class Data and all Concrete Data Class:** (These Classes are involved in implementation of Strategy Pattern)



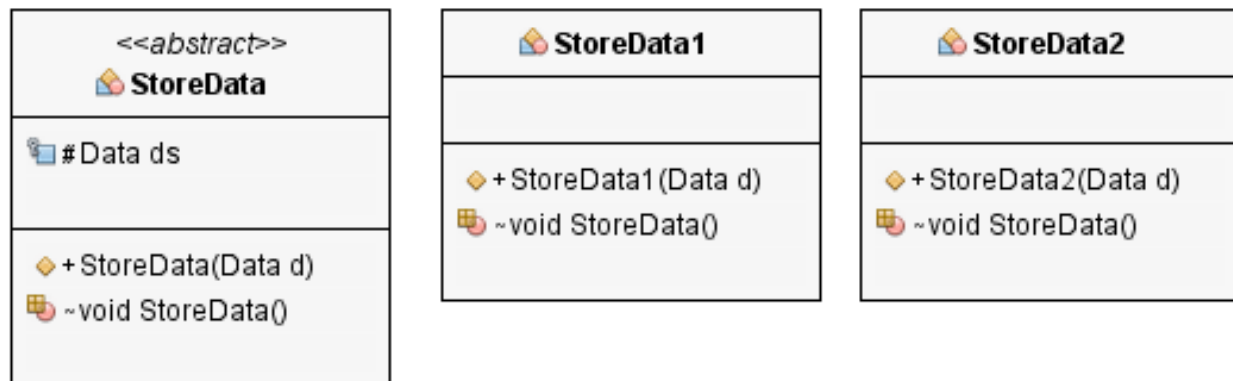
**Data Class:** This is an abstract class which exposes functions and member variables specific to a particular GasPump. The functions that it exposes are all setters and getters which are being used as per a specific GasPump's requirement. As per the strategy pattern the child classes provide their own implementation of setters and getters.

**Data1 Class:** This class extends Data Class and implements the setter and getter functions for the member variables specific to GasPump1's requirement only.

**Data2 Class:** This class extends Data Class and implements the setter and getter functions for the member variables specific to GasPump2's requirement only.

**Data3 Class:** This class extends Data Class and implements the setter and getter functions for the member variables specific to GasPump3's requirement only.

**Abstract Class StoreData and all Concrete StoreData Class:** (These Classes are involved in implementation of Strategy Pattern)

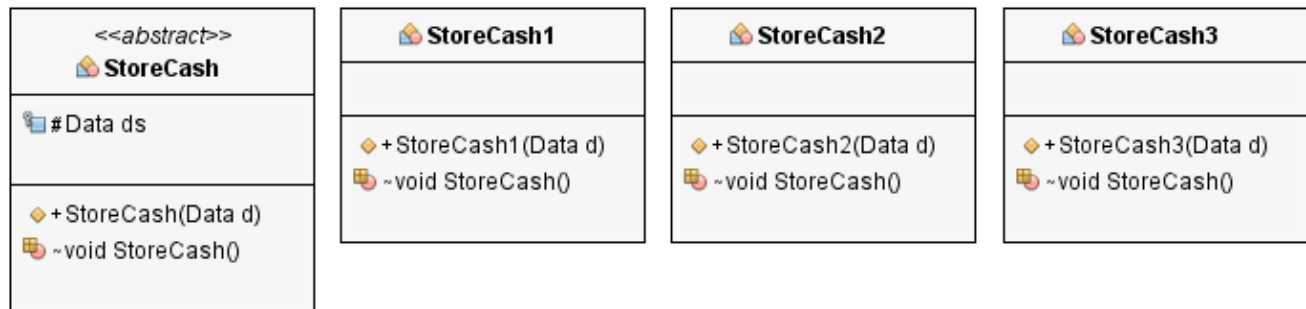


**StoreData Class:** This is an abstract class which exposes StoreData function specific to a particular GasPump. The function that it exposes provides two flavors of StoreData action function being called using Strategy Pattern by OutputProcessor based on the type of GasPump.

**StoreData1 Class:** This class extends StoreData Class and implements StoreData function which stores only one data into a variable named price. This class comes handy when we have only a single variety of gas eg. Regular in our GasPump.

**StoreData2 Class:** This class extends StoreData Class and implements StoreData function which stores two data related to price into a variables like SPrice and RPrice. This class comes handy when we have two variety of gas eg. Super, Regular or Premium and Regular in our GasPump.

**Abstract Class StoreCash and all Concrete StoreData Class:** (These Classes are involved in implementation of Strategy Pattern)



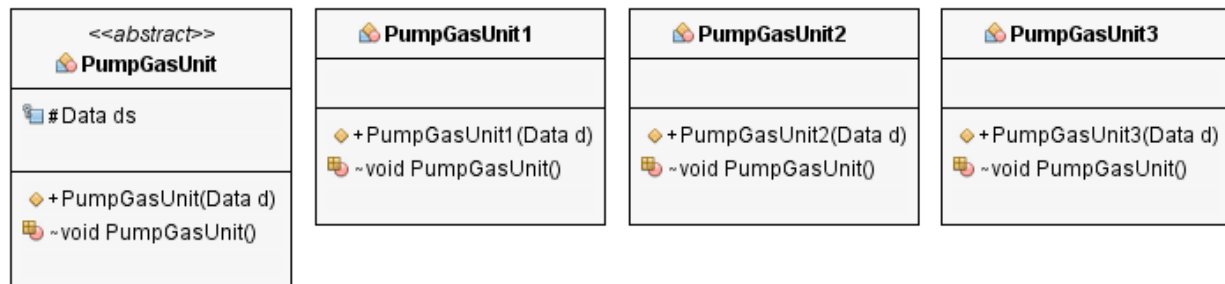
**StoreCash Class:** This is an abstract class which exposes StoreCash function specific to a particular GasPump. The function that it exposes provides various flavours of StoreCash action function being called using Strategy Pattern by OutputProcessor based on the type of GasPump.

**StoreCash1 Class:** This class extends StoreCash Class and implements StoreCash function which stores cash entered by User. This class comes handy when a user has selected payment method as Pay by cash.

**StoreCash2 Class:** This class extends StoreCash Class and provides an empty implementation to StoreCash function as GasPump2 doesn't provide Pay by cash option.

**StoreCash3 Class:** This class extends StoreCash Class and implements StoreCash function which stores cash entered by User. This class is called everytime by the user as selected payment method is always Pay by cash for GasPump3.

**Abstract Class PumpGasUnit and all Concrete PumpGasUnit Class:** (These Classes are involved in implementation of Strategy Pattern)



**PumpGasUnit Class:** This is an abstract class which exposes PumpGas function specific to a particular GasPump. The function that it exposes provides various flavours of PumpGasUnit action function being called using Strategy Pattern by OutputProcessor based on the type of GasPump.

**PumpGasUnit1 Class:** This class extends PumpGasUnit Class and implements PumpGasUnit function specific to GasPump1.

**PumpGasUnit2 Class:** This class extends PumpGasUnit Class and implements PumpGasUnit function specific to GasPump2.

**PumpGasUnit3 Class:** This class extends PumpGasUnit Class and implements PumpGasUnit function specific to GasPump3.



## Source Code:

### **GasPump1.java**

```
public class GasPump1 {
    private MDA_EFSM m;
    private Data d;
    private AbstractFactory af;

    public GasPump1(MDA_EFSM mobj) {
        m = mobj;
        af = new ConcreteFactory1();
        d=af.getDataObj();
        m.setAbstractFactory(af);
    }
    void Activate(int a){
        if(a>0){
            d.setTempA(a); //Storing value of a in TempA
            m.Activate(); //Calling activate function of mda-efsm
        }
        else
            System.out.print("Price should be a positive number");
    }
    void Start(){
        m.Start();//Calling Start function of mda-efsm
    }
    void PayCredit(){
        m.PayCredit();//Calling PayCredit function of mda-efsm
    }
    void PayCash(int c){
        if(c>0){
            d.setTempC(c); //Storing value of c in TempC
            m.PayCash();//Calling PayCash function of mda-efsm
        }
        else System.out.print("Cash amount should be a positive number");
    }
    void Reject(){
        m.Reject();//Calling Reject function of mda-efsm
    }
    void Cancel(){
        m.Cancel();//Calling Cancel function of mda-efsm
    }
    void Approved(){
        m.Approved();//Calling Approved function of mda-efsm
    }
    void StartPump(){
```

```

        m.SelectGas(1);//Calling SelectGas function of mda-efsm
        m.StartPump();//Calling StartPump function of mda-efsm
    }
    void PumpGallon(){
        if(d.W == 1)
            m.Pump();//Calling Pump function of mda-efsm
        else{
            int cash = d.getCash();//Getting Cash value from DataStore
            int G = d.getG();//Getting # of Gallons from DataStore
            int price = d.getPrice();//Getting price per Gallon from DataStore
            if(cash < (G + 1)*price){
                m.StopPump();//Calling StopPump function of mda-efsm
                m.Receipt();//Calling Receipt function of mda-efsm
            }
            else
                m.Pump();//Calling Pump function of mda-efsm
        }
    }
}
void StopPump(){
    m.StopPump();//Calling StopPump function of mda-efsm
    m.Receipt();//Calling Receipt function of mda-efsm
}
}

```

## GasPump2.java

```

public class GasPump2 {
    private MDA_EFSM m;
    private Data d;
    private AbstractFactory af;

    public GasPump2(MDA_EFSM mobj) {
        m = mobj;
        af = new ConcreteFactory2();
        d=af.getDataObj();
        m.setAbstractFactory(af);
    }
    public void Activate(float a, float b){
        if((a>0)&&(b>0)){
            d.setTemp_A(a);//Storing value of a in TempA
            d.setTemp_B(b);//Storing value of b in TempB
            m.Activate();//Calling Activate function of mda-efsm
        }
        else
            System.out.println("Price should be a positive number");
    }
}

```

```
public void Start(){
    m.Start();//Calling Start function of mda-efsm
}
void PayCredit(){
    m.PayCredit();//Calling PayCredit function of mda-efsm
}
void Reject(){
    m.Reject();//Calling Reject function of mda-efsm
}
void Cancel(){
    m.Cancel();//Calling Cancel function of mda-efsm
}
void Approved(){
    m.Approved();//Calling Approved function of mda-efsm
}
void Super(){
    m.SelectGas(2);//Calling SelectGas function of mda-efsm
}
void Regular(){
    m.SelectGas(1);//Calling SelectGas function of mda-efsm
}
void StartPump(){
    m.StartPump();//Calling StartPump function of mda-efsm
}
void PumpGallon(){
    m.Pump();//Calling Pump function of mda-efsm
}
void StopPump(){
    m.StopPump();//Calling StopPump function of mda-efsm
    m.Receipt();//Calling Receipt function of mda-efsm
}
}
```

### GasPump3.java

```
public class GasPump3 {
    private MDA_EFSM m;
    private Data d;
    private AbstractFactory af;

    public GasPump3(MDA_EFSM mobj) {
        m = mobj;
        af = new ConcreteFactory3();
        d=af.getDataObj();
        m.setAbstractFactory(af);
    }
}
```

```
public void Activate(float a, float b){
    if((a>0)&&(b>0)){
        d.setTemp_A(a);//Storing value of a in TempA
        d.setTemp_B(b);//Storing value of b in TempB
        m.Activate();//Calling Activate function of mda-efsm
    }
    else
        System.out.println("\tPrice should be a positive number");
}
public void Start(){
    m.Start();//Calling Start function of mda-efsm
}
void PayCash(float c){
    if(c>0){
        d.setTemp_C(c);//Storing value of c in TempC
        m.PayCash();//Calling PayCash function of mda-efsm
    }
    else System.out.print("Cash amount should be a positive number");
}
void Cancel(){
    m.Cancel();//Calling Cancel function of mda-efsm
}
void Premium(){
    m.SelectGas(2);//Calling SelectGas function of mda-efsm
}
void Regular(){
    m.SelectGas(1);//Calling SelectGas function of mda-efsm
}
void StartPump(){
    m.StartPump();//Calling StartPump function of mda-efsm
}
void PumpLiter(){
    float cash = d.get_Cash();//Getting Cash value from DataStore
    int L = d.getL();//Getting # of Liters from DataStore
    float price = d.get_Price();//Getting price per Gallon from DataStore
    if(cash < (L + 1)*price)
        m.StopPump();//Calling StopPump function of mda-efsm
    else
        m.Pump();//Calling Pump function of mda-efsm
}
void StopPump(){
    m.StopPump();//Calling StopPump function of mda-efsm
}
void Receipt(){
    m.Receipt();//Calling Receipt function of mda-efsm
}
void NoReceipt(){
```

```
        m.NoReceipt();//Calling NoReceipt function of mda-efsm
    }
}
```

### **AbstractFactory.java**

```
public abstract class AbstractFactory {
    protected Data ds; //Data_Store object that store all data specific to various GasPump
    protected StoreData sd; //StoreData object which provides StoreData function.
    protected DisplayMsgs dm; //DisplayMsgs obj provides all putput related display msgs.
    protected StoreCash sc; //StoreCash obj provides method to store cash
    protected PumpGasUnit pgu; //PumpGasUnit obj provides method to calculate #of gallons
    pumped.

    abstract Data getDataObj();
    abstract StoreData getStoreDataObj();
    abstract DisplayMsgs getDisplayMsgObj();
    abstract StoreCash getStoreCashObj();
    abstract PumpGasUnit getPumpGasUnitObj();
}
```

### **ConcreteFactory1.java**

```
public class ConcreteFactory1 extends AbstractFactory {
    public ConcreteFactory1() {
        ds = new Data1();
    }
    @Override
    Data getDataObj() {
        return ds;
    }
    @Override
    StoreData getStoreDataObj() {
        sd = new StoreData1(ds);
        return this.sd;
    }
    @Override
    DisplayMsgs getDisplayMsgObj() {
        dm = new DisplayMsgs1(ds);
        return dm;
    }
    @Override
    StoreCash getStoreCashObj() {
        sc = new StoreCash1(ds);
        return sc;
    }
}
```

```
@Override
PumpGasUnit getPumpGasUnitObj() {
    pgu =new PumpGasUnit1(ds);
    return pgu;
}
}
```

### ConcreteFactory2.java

```
public class ConcreteFactory2 extends AbstractFactory {
    public ConcreteFactory2() {
        ds = new Data2();
    }
    @Override
    Data getDataObj() {
        return ds;
    }
    @Override
    StoreData getStoreDataObj() {
        sd = new StoreData2(ds);
        return this.sd;
    }
    @Override
    DisplayMsgs getDisplayMsgObj() {
        dm = new DisplayMsgs2(ds);
        return dm;
    }
    @Override
    StoreCash getStoreCashObj() {
        sc = new StoreCash2(ds);
        return sc;
    }
    @Override
    PumpGasUnit getPumpGasUnitObj() {
        pgu =new PumpGasUnit2(ds);
        return pgu;
    }
}
```

### ConcreteFactory3.java

```
public class ConcreteFactory3 extends AbstractFactory {
    public ConcreteFactory3() {
        ds = new Data3();
    }
    @Override
```

```

Data getDataObj() {
    return ds;
}
@Override
StoreData getStoreDataObj() {
    sd = new StoreData2(ds);
    return this.sd;
}
@Override
DisplayMsgs getDisplayMsgObj() {
    dm = new DisplayMsgs3(ds);
    return dm;
}
@Override
StoreCash getStoreCashObj() {
    sc = new StoreCash3(ds);
    return sc;
}
@Override
PumpGasUnit getPumpGasUnitObj() {
    pgu = new PumpGasUnit3(ds);
    return pgu;
}
}

```

## Data.java

```

public abstract class Data {
    protected int W, G, L;
    protected float temp_A, temp_B, temp_C, cash, price;

    void setTempA(int a){ }
    void setTemp_A(float a){ }
    void setTemp_B(float b){ }
    void setPrice(int gastype){ }
    void setRegular(float a){ }
    void setSuperOrPremium(float b){ }
    int getTempA(){return -1;}
    float getTemp_A(){return -1;}
    float getTemp_B(){return -1;}
    int getPrice(){return -1; }
    float get_Price() {return -1;}
    void setTempC(int c){ }
    int getTempC(){return -1;}
    void setTemp_C(float c){ }
    float getTemp_C(){return -1;}
}

```

```
void setCash(int c){}  
int getCash(){return -1;}  
void set_Cash(float c){}  
float get_Cash(){return-1;}  
  
abstract void setG(int g);  
abstract int getG();  
abstract void setL(int L);  
abstract int getL();  
}
```

### **Data1.java**

```
public class Data1 extends Data{  
    @Override  
    void setTempA(int a) { temp_A = a; }  
    @Override  
    int getTempA() { return (int)temp_A; }  
    @Override  
    void setPrice(int gastype) {  
        if(gastype == 1)  
            price = (int)temp_A;  
    }  
    @Override  
    int getPrice() {return (int)price; }  
    @Override  
    void setTempC(int c) { temp_C = c; }  
    @Override  
    int getTempC() { return (int) temp_C; }  
    @Override  
    void setCash(int c) { cash = c; }  
    @Override  
    int getCash() { return (int) cash; }  
    @Override  
    void setG(int g) { G = g; }  
    @Override  
    int getG() { return G; }  
    @Override  
    void setL(int L) {}  
    @Override  
    int getL() {return -1;}  
}
```

### **Data2.java**

```
public class Data2 extends Data{  
    @Override
```



```
void setTemp_A(float a) {temp_A = a;}
@Override
void setTemp_B(float b) {temp_B = b;}
@Override
void setPrice(int gastype) {
    if(gastype == 2){ //if gastype is super
        price = temp_B;
        System.out.println("\tSUPER GAS IS SELECTED:\n");
    }
    else{ //if gastype is regular
        price = temp_A;
        System.out.println("\tREGULAR GAS IS SELECTED:\n");
    }
}
@Override
float get_Price() {return price;}
@Override
void setG(int g) {G = g;}
@Override
int getG() { return G; }
@Override
void setL(int L) {}
@Override
int getL() {return -1;}
}
```

### Data3.java

```
public class Data3 extends Data{
    @Override
    void setTemp_A(float a) { temp_A = a;}
    @Override
    void setTemp_B(float b) {temp_B = b;}
    @Override
    void setTemp_C(float c) {temp_C = c;}
    @Override
    float getTemp_C() { return temp_C; }
    @Override
    void set_Cash(float c) {cash = c;}
    @Override
    float get_Cash() { return cash;}
    @Override
    void setPrice(int gastype) {
        if(gastype == 2){ //if gastype is premium
            System.out.println("\tPREMIUM GAS IS SELECTED:\n");
            price = temp_B;
        }
    }
}
```

```

    }
    else{ //if gastype is regular
        System.out.println("\tREGULAR GAS IS SELECTED:\n");
        price = temp_A;
    }
}
@Override
float get_Price() { return price;}
@Override
void setG(int g) {}
@Override
int getG() { return -1; }
@Override
void setL(int l) { L =l;}
@Override
int getL() {return L;}
}

```

### MDA\_EFSM.java

```

public class MDA_EFSM {
    private State[] s = new State[8];
    private State sobj;
    private OutputProcessor op;
    public MDA_EFSM() {
        op = new OutputProcessor();
        s[0] = new Start(this, op);
        s[1] = new S0(this, op);
        s[2] = new S1(this, op);
        s[3] = new S2(this, op);
        s[4] = new S3(this, op);
        s[5] = new S4(this, op);
        s[6] = new S5(this, op);
        s[7] = new S6(this, op);
        sobj=s[0];
    }

    //MDA_EFSM Events
    void Activate(){ sobj.Activate(); }
    void Start(){ sobj.Start(); }
    void PayCredit(){ sobj.PayCredit(); }
    void PayCash(){ sobj.PayCash(); }
    void Reject(){ sobj.Reject(); }
    void Cancel(){ sobj.Cancel(); }
    void Approved(){ sobj.Approved(); }
    void StartPump(){ sobj.StartPump(); }
}

```

```
void Pump(){ subj.Pump(); }
void StopPump(){ subj.StopPump(); }
void SelectGas(int g){ subj.SelectGas(g); }
void Receipt(){ subj.Receipt(); }
void NoReceipt(){ subj.NoReceipt(); }
void ChangeState(int i){
    subj=s[i]; // changes state as per value of i
}
void setAbstractFactory(AbstractFactory af){
    op.setAbstractFactory(af);
}
}
```

### State.java

```
public abstract class State {
    protected OutputProcessor op;
    protected MDA_EFSM mda;

    public State(MDA_EFSM m, OutputProcessor p) {
        mda = m;
        op = p;
    }
    public void Activate(){}
    public void Start(){}
    public void PayCredit(){}
    public void PayCash(){}
    public void Reject(){}
    public void Cancel(){}
    public void Approved(){}
    public void StartPump(){}
    public void Pump(){}
    public void StopPump(){}
    public void SelectGas(int g){}
    public void Receipt(){}
    public void NoReceipt(){}
}
```

### Start.java

```
public class Start extends State {

    public Start(MDA_EFSM mda, OutputProcessor op) {
        super(mda,op);
    }

    @Override
```

```
public void Activate() {
    op.StoreData(); //Calling OutputProcessor's StoreData method.
    mda.ChangeState(1); //Changing the state to point to S0
}
}
```

### **S0.java**

```
public class S0 extends State{
    public S0(MDA_EFSM mda, OutputProcessor op) {
        super(mda,op);
    }
    @Override
    public void Start() {
        op.PayMsg();//Calling OutputProcessor's StoreData method.
        mda.ChangeState(2);//Changing the state to point to S1
    }
}
```

### **S1.java**

```
public class S1 extends State{
    public S1(MDA_EFSM mda, OutputProcessor op) {
        super(mda,op);
    }
    @Override
    public void PayCash() {
        op.SetW(0);//Calling OutputProcessor's SetW method.
        op.StoreCash(); //Calling OutputProcessor's Storecash method.
        op.DisplayMenu();//Calling OutputProcessor's DisplayMenu method.
        mda.ChangeState(4);//Changing the state to point to S3
    }
    @Override
    public void PayCredit() {
        System.out.println("\n\tWaiting for approval:\n");
        mda.ChangeState(3);//Changing the state to point to S2
    }
}
```

### **S2.java**

```
public class S2 extends State{
    public S2(MDA_EFSM m, OutputProcessor p) {
        super(m, p);
    }
    @Override
```

```
public void Approved() {
    op.SetW(1);//Calling OutputProcessor's SetW method.
    op.DisplayMenu();//Calling OutputProcessor's DisplayMenu method.
    mda.ChangeState(4);//Changing the state to point to S3
}
@Override
public void Reject() {
    op.RejectMsg();//Calling OutputProcessor's SetW method.
    mda.ChangeState(1);//Changing the state to point to S0
}
}
```

### **S3.java**

```
public class S3 extends State{
    public S3(MDA_EFSM mda, OutputProcessor op) {
        super(mda,op);
    }
    @Override
    public void SelectGas(int g) {
        op.SetPrice(g);//Calling OutputProcessor's SetPrice method.
        mda.ChangeState(5);//Changing the state to point to S4
    }
    @Override
    public void Cancel() {
        op.CancelMsg();//Calling OutputProcessor's CancelMsg method.
        mda.ChangeState(1);//Changing the state to point to S0
    }
}
```

### **S4.java**

```
public class S4 extends State{
    public S4(MDA_EFSM mda, OutputProcessor op) {
        super(mda,op);
    }
    @Override
    public void StartPump() {
        op.SetInitialValues();//Calling OutputProcessor's SetInitialValues method.
        op.ReadyMsg();//Calling OutputProcessor's ReadyMsg method.
        mda.ChangeState(6);//Changing the state to point to S5
    }
}
```

### **S5.java**

```
public class S5 extends State{
    public S5(MDA_EFSM mda, OutputProcessor op) {
        super(mda,op);
    }
    @Override
    public void Pump() {
        op.PumpGasUnit();//Calling OutputProcessor's PumpGasUnit method.
        op.GasPumpedMsg();//Calling OutputProcessor's GasPumpedMsg method.
    }
    @Override
    public void StopPump() {
        op.StopMsg();//Calling OutputProcessor's StopMsg method.
        mda.ChangeState(7);//Changing the state to point to S6
    }
}
```

### **S6.java**

```
public class S6 extends State{
    public S6(MDA_EFSM mda, OutputProcessor op) {
        super(mda,op);
    }
    @Override
    public void NoReceipt() {
        System.out.println("\n\tNO RECEIPT:\n");
        mda.ChangeState(1);//Changing the state to point to S0
    }
    @Override
    public void Receipt() {
        op.PrintReceipt();//Calling OutputProcessor's PrintReceipt method.
        mda.ChangeState(1);//Changing the state to point to S0
    }
}
```

### **OutputProcessor.java**

```
public class OutputProcessor {
    private AbstractFactory af;
    Data ds;
    //MDA_EFSM Actions
    void StoreData(){
        StoreData sd = af.getStoreDataObj();
        sd.StoreData();
        System.out.println("\n\tGAS PUMP IS ACTIVATED\n");
    }
    void PayMsg(){ //add menu for selecting cash or credit
```

```
    DisplayMsgs dm = af.getDisplayMsgObj();
    dm.PayMsg();
}
void StoreCash(){
    StoreCash sc = af.getStoreCashObj();
    sc.StoreCash();
}
void RejectMsg(){
    DisplayMsgs dm = af.getDisplayMsgObj();
    dm.RejectMsg();
}
void DisplayMenu(){
    DisplayMsgs dm = af.getDisplayMsgObj();
    dm.DisplayMsg();
}
void CancelMsg(){
    System.out.println("\n\tTRANSACTION CANCELLED.\n");
}
void SetW(int w){
    Data ds = af.getDataObj();
    ds.W =w;
}
void SetPrice(int g){
    Data ds = af.getDataObj();
    ds.setPrice(g);
}
void ReadyMsg(){
    System.out.println("\n\tSTART PUMPING:\n");
}
void SetInitialValues(){
    Data ds = af.getDataObj();
    ds.setG(0);
    ds.setL(0);
}
void PumpGasUnit(){
    PumpGasUnit pgu = af.getPumpGasUnitObj();
    pgu.PumpGasUnit();
}
void GasPumpedMsg(){
    DisplayMsgs dm = af.getDisplayMsgObj();
    dm.GasPumpedMsg();
}
void StopMsg(){
    DisplayMsgs dm = af.getDisplayMsgObj();
    dm.StopMsg();
}
```

```
void PrintReceipt(){
    DisplayMsgs dm = af.getDisplayMsgObj();
    dm.PrintReceipt();
}
void setAbstractFactory(AbstractFactory afactory){
    af = afactory;
}
}
```

### **StoreData.java**

```
public abstract class StoreData {
    protected Data ds;
    public StoreData(Data d) {
        ds = d;
    }
    abstract void StoreData();
}
```

### **StoreData1.java**

```
public class StoreData1 extends StoreData{
    public StoreData1(Data d) {
        super(d);
    }
    @Override
    void StoreData() {
        int price = ds.getTempA();
        ds.setPrice(price);
    }
}
```

### **StoreData2.java**

```
public class StoreData2 extends StoreData{
    public StoreData2(Data d) {
        super(d);
    }
    @Override
    void StoreData() {
        float regular = ds.getTemp_A();
        ds.setRegular(regular);
        float superorpremium = ds.getTemp_B();
        ds.setSuperOrPremium(superorpremium);
    }
}
```



**StoreCash.java**

```
public abstract class StoreCash {  
    protected Data ds;  
    public StoreCash(Data d) {  
        ds = d;  
    }  
    abstract void StoreCash();  
}
```

**StoreCash1.java**

```
public class StoreCash1 extends StoreCash {  
    public StoreCash1(Data d) {  
        super(d);  
    }  
    @Override  
    void StoreCash() {  
        int cash = ds.getTempC();  
        ds.setCash(cash);  
    }  
}
```

**StoreCash2.java**

```
public class StoreCash2 extends StoreCash {  
    public StoreCash2(Data d) {  
        super(d);  
    }  
    @Override  
    void StoreCash() {}  
}
```

**StoreCash3.java**

```
public class StoreCash3 extends StoreCash {  
    public StoreCash3(Data d) {  
        super(d);  
    }  
    @Override  
    void StoreCash() {  
        float cash = ds.getTemp_C();  
        ds.set_Cash(cash);  
    }  
}
```

## DisplayMsgs.java

```
public abstract class DisplayMsgs {  
    protected Data ds;  
    public DisplayMsgs(Data ds) {  
        this.ds = ds;  
    }  
    abstract void PayMsg();  
    abstract void DisplayMsg();  
    abstract void RejectMsg();  
    abstract void GasPumpedMsg();  
    abstract void StopMsg();  
    abstract void PrintReceipt();  
}
```

## DisplayMsgs1.java

```
public class DisplayMsgs1 extends DisplayMsgs {  
    public DisplayMsgs1(Data ds) {  
        super(ds);  
    }  
    void PayMsg() {  
        System.out.println("\n\tSelect Payment:");  
        System.out.println("\t2. Credit");  
        System.out.println("\t3. Cash\n");  
    }  
    @Override  
    void DisplayMsg() {  
        if(ds.W == 1)  
            System.out.println("\n\t The Credit Card is approved.\n");  
    }  
    @Override  
    void RejectMsg() {  
        System.out.println("\n\t The Credit Card is not approved.\n");  
    }  
    @Override  
    void GasPumpedMsg() {  
        int G = ds.getG();  
        System.out.println("\n\t"+G+" GALLONS PUMPED\n");  
    }  
    @Override  
    void StopMsg() {  
        if(ds.W == 0){  
            int cash = ds.getCash();  
            int G = ds.getG();  
        }  
    }  
}
```

```
        int price = ds.getPrice();
        if(cash < (G + 1)*price){
            System.out.println("\n\tPUMP STOPPED - NOT SUFFICIENT AMOUNT OF
CASH\n");
            return;
        }
    }
    System.out.println("\n\tPUMP IS STOPPED\n");
}
@Override
void PrintReceipt() {
    int total = ds.getPrice() * ds.getG();
    System.out.println("\n\tRECEIPT");
    System.out.println("\t# OF GALLONS: "+ds.getG());
    System.out.println("\tTOTAL: $" + total + "\n");
}
}
```

### DisplayMsgs2.java

```
public class DisplayMsgs2 extends DisplayMsgs {
    public DisplayMsgs2(Data ds) {
        super(ds);
    }
    void PayMsg() {
        System.out.println("\n\tPAY BY CREDIT:\n");
    }
    @Override
    void DisplayMsg() {
        System.out.println("\n\tSELECT TYPE OF GAS");
        System.out.println("\t6. REGULAR");
        System.out.println("\t7. SUPER\n");
    }
    @Override
    void RejectMsg() {
        System.out.println("\n\t The Credit Card is not approved.\n");
    }
    @Override
    void GasPumpedMsg() {
        int G = ds.getG();
        System.out.println("\n\t"+G+" GALLONS PUMPED\n");
    }
    @Override
    void StopMsg() {
        System.out.println("\n\tPUMP IS STOPPED\n");
    }
}
```

```
@Override
void PrintReceipt() {
    float total = ds.get_Price() * ds.getG();
    System.out.println("\n\tRECEIPT");
    System.out.println("\t# OF GALLONS: "+ds.getG());
    System.out.println("\tTOTAL: $" +total+"\n");
}
}
```

### DisplayMsgs3.java

```
public class DisplayMsgs3 extends DisplayMsgs {
    public DisplayMsgs3(Data ds) {
        super(ds);
    }
    void PayMsg() {
        System.out.println("\nPAY BY CASH:");
    }
    @Override
    void DisplayMsg() {
        System.out.println("\n\tSELECT TYPE OF GAS");
        System.out.println("\tREGULAR");
        System.out.println("\tPREMIUM\n");
    }
    @Override
    void RejectMsg() {}
    @Override
    void GasPumpedMsg() {
        int L = ds.getL();
        System.out.println("\n\t"+L+" LITERS PUMPED\n");
    }
    @Override
    void StopMsg() {
        if(ds.W == 0){
            float cash = ds.get_Cash();
            int L = ds.getL();
            float price = ds.get_Price();
            if(cash < (L +1)*price){
                System.out.println("\n\tPUMP STOPPED - NOT SUFFICIENT AMOUNT OF
CASH\n");
                System.out.println("\n\tDO YOU WANT A RECEIPT?\n");
                return;
            }
        }
        System.out.println("\n\tPUMP IS STOPPED");
        System.out.println("\tDO YOU WANT A RECEIPT?\n");
    }
}
```

```
    }  
    @Override  
    void PrintReceipt() {  
        float total = ds.get_Price() * ds.getL();  
        System.out.println("\n\tRECEIPT");  
        System.out.println("\t# OF LITERS: "+ds.getL());  
        System.out.println("\tTOTAL: $" +total+"\n");  
    }  
}
```

### **PumpGasUnit.java**

```
public abstract class PumpGasUnit {  
    protected Data ds;  
    public PumpGasUnit(Data d) {  
        ds = d;  
    }  
    abstract void PumpGasUnit();  
}
```

### **PumpGasUnit1.java**

```
public class PumpGasUnit1 extends PumpGasUnit{  
    public PumpGasUnit1(Data d) {  
        super(d);  
    }  
    @Override  
    void PumpGasUnit() {  
        ds.setG(ds.getG()+1);  
    }  
}
```

### **PumpGasUnit2.java**

```
public class PumpGasUnit2 extends PumpGasUnit{  
    public PumpGasUnit2(Data d) {  
        super(d);  
    }  
    @Override  
    void PumpGasUnit() {  
        ds.setG(ds.getG()+1);  
    }  
}
```

### **PumpGasUnit3.java**

```
public class PumpGasUnit3 extends PumpGasUnit{
    public PumpGasUnit3(Data d) {
        super(d);
    }
    @Override
    void PumpGasUnit() {
        ds.setL(ds.getL()+1);
    }
}
```

### mainClass.java

```
import java.util.Scanner;
public class mainClass {
    public static void main(String[] args){
        MDA_EFSM mda = new MDA_EFSM();
        Scanner in = new Scanner(System.in);

        //Menu options
        System.out.println("\tSELECT GP-1, GP-2 OR GP-3:\n");
        System.out.println("\t1. GP-1\n\t2. GP-2\n\t3. GP-3\n");
        System.out.print("\tSelect GP: ");
        int gptype;
        try{
            gptype= Integer.parseInt(in.nextLine());
        }catch(NumberFormatException ne){
            gptype = 1;
        }
        switch(gptype){
            case 1: showGP1Options(mda);
                    break;
            case 2: showGP2Options(mda);
                    break;
            case 3: showGP3Options(mda);
                    break;
            default:showGP1Options(mda);
                    break;
        }
    }

    private static void showGP1Options(MDA_EFSM m) {
        GasPump1 gp1 = new GasPump1(m);
        Scanner sc =new Scanner(System.in);

        System.out.println("\t\tGP-1 Execution\n");
        for(;;){
```

```
System.out.println("\tSelect Operations:");
System.out.println("0-Activation, 1-Start, 2-PayCredit, 3-PayCash, 4-Reject, 5-
Approved, "
    + "\n6-Cancel, 7-StartPump, 8-PumpGallon, 9-StopPump, q-Quit");
System.out.print("Enter your choice:");
String choice = sc.nextLine();

switch(choice){
    case "0":System.out.println("\tOperation Activation(int a)");
        System.out.print("\tEnter value of the parameter a: ");
        int a;
        try{
            a=Integer.parseInt(sc.nextLine());
        }catch(NumberFormatException ne){
            a=-1;
        }
        gp1.Activate(a);
        break;
    case "1":System.out.println("\tOperation Start()");
        gp1.Start();
        break;
    case "2":System.out.println("\tOperation PayCredit()");
        gp1.PayCredit();
        break;
    case "3":System.out.println("\tOperation PayCash(int c)");
        System.out.print("\tEnter value of the parameter c: ");
        int c;
        try{
            c=Integer.parseInt(sc.nextLine());
        }catch(NumberFormatException ne){
            c=-1;
        }
        gp1.PayCash(c);
        break;
    case "4":System.out.println("\tOperation Reject()");
        gp1.Reject();
        break;
    case "5":System.out.println("\tOperation Approved()");
        gp1.Approved();
        break;
    case "6":System.out.println("\tOperation Cancel()");
        gp1.Cancel();
        break;
    case "7":System.out.println("\tOperation StartPump()");
        gp1.StartPump();
        break;
```

```
        case "8":System.out.println("\tOperation PumpGallon()");
            gp1.PumpGallon();
            break;
        case "9":System.out.println("\tOperation StopPump()");
            gp1.StopPump();
            break;
        case "q":System.exit(0);
        default: System.out.println("\tInvalid choice\n");
    }
} //end of for
} //end of function showGP1Options

private static void showGP2Options(MDA_EFSM m) {
    GasPump2 gp2 = new GasPump2(m);
    Scanner sc =new Scanner(System.in);

    System.out.println("\t\tGP-2 Execution\n");
    for(;;){
        System.out.println("\tSelect Operations:");
        System.out.println("0-Activation, 1-Start, 2-PayCredit, 3-Reject, 4-Approved, 5-Cancel,"
            + "\n6-Regular, 7-Super, 8-StartPump, 9-PumpGallon, a-StopPump, q-Quit");
        System.out.print("Enter your choice:");
        String choice = sc.nextLine();

        switch(choice){
            case "0":System.out.println("\tOperation Activation(int a)");
                System.out.print("\tEnter value of the parameter a: ");
                float a;
                try{
                    a=Float.parseFloat(sc.nextLine());
                }catch(NumberFormatException ne){
                    a=-1;
                }
                System.out.print("\tEnter value of the parameter b: ");
                float b;
                try{
                    b=Float.parseFloat(sc.nextLine());
                }catch(NumberFormatException ne){
                    b=-1;
                }
                gp2.Activate(a,b);
                break;
            case "1":System.out.println("\tOperation Start()");
                gp2.Start();
                break;
            case "2":System.out.println("\tOperation PayCredit()");
```



```
        gp2.PayCredit();
        break;
    case "3":System.out.println("\tOperation Reject()");
        gp2.Reject();
        break;
    case "4":System.out.println("\tOperation Approved()");
        gp2.Approved();
        break;
    case "5":System.out.println("\tOperation Cancel()");
        gp2.Cancel();
        break;
    case "6":System.out.println("\tOperation Regular()");
        gp2.Regular();
        break;
    case "7":System.out.println("\tOperation Super()");
        gp2.Super();
        break;
    case "8":System.out.println("\tOperation StartPump()");
        gp2.StartPump();
        break;
    case "9":System.out.println("\tOperation PumpGallon()");
        gp2.PumpGallon();
        break;
    case "a":System.out.println("\tOperation StopPump()");
        gp2.StopPump();
        break;
    case "q":System.exit(0);
    default: System.out.println("\tInvalid choice\n");
    }
} //end of for
} //end of function showGP2Options

private static void showGP3Options(MDA_EFSM m) {
    GasPump3 gp3 = new GasPump3(m);
    Scanner sc =new Scanner(System.in);

    System.out.println("\t\tGP-3 Execution\n");
    for(;;){
        System.out.println("\tSelect Operations:");
        System.out.println("0-Activation, 1-Start, 2-PayCash, 3-Cancel, 4-Regular, 5-Premium,"
            + "\n6-StartPump, 7-PumpLiter, 8-StopPump, 9-Receipt, a-NoReceipt, q-Quit");
        System.out.print("Enter your choice:");
        String choice = sc.nextLine();

        switch(choice){
            case "0":System.out.println("\tOperation Activation(int a)");
```

```
        System.out.print("\nEnter value of the parameter a: ");
        float a;
        try{
            a=Float.parseFloat(sc.nextLine());
        }catch(NumberFormatException ne){
            a=-1;
        }
        System.out.print("\nEnter value of the parameter b: ");
        float b;
        try{
            b=Float.parseFloat(sc.nextLine());
        }catch(NumberFormatException ne){
            b=-1;
        }
        gp3.Activate(a,b);
        break;
    case "1":System.out.println("\tOperation Start()");
        gp3.Start();
        break;
    case "2":System.out.println("\tOperation PayCash(int c)");
        System.out.print("\nEnter value of the parameter c: ");
        int c;
        try{
            c=Integer.parseInt(sc.nextLine());
        }catch(NumberFormatException ne){
            c=-1;
        }
        gp3.PayCash(c);
        break;
    case "3":System.out.println("\tOperation Cancel()");
        gp3.Cancel();
        break;
    case "4":System.out.println("\tOperation Regular()");
        gp3.Regular();
        break;
    case "5":System.out.println("\tOperation Premium()");
        gp3.Premium();
        break;
    case "6":System.out.println("\tOperation StartPump()");
        gp3.StartPump();
        break;
    case "7":System.out.println("\tOperation PumpGallon()");
        gp3.PumpLiter();
        break;
    case "8":System.out.println("\tOperation StopPump()");
        gp3.StopPump();
```

```
        break;
    case "9":System.out.println("\tOperation Receipt()");
        gp3.Receipt();
        break;
    case "a":System.out.println("\tOperation NoReceipt()");
        gp3.NoReceipt();
        break;
    case "q":System.exit(0);
    default: System.out.println("\tInvalid choice\n");
    }
} //end of for
} //end of function showGP3Options
}
```