Project Report

CS 586-Software System Architecture

AbhinavPimpalgaonkar

A20387324

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: GasPump-1 and GasPump-2.

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

1.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
2.Start() -start the transaction
3.PayCredit() - pay for gas by a credit card
4.Reject() -credit card is rejected
5.Cancel() -cancel the transaction
6.Approved() -credit card is approved
7.Super() -Super gas is selected
8.Regular() -Regular gas is selected
9.StartPump() -start pumping gas
10.PumpGallon() -one gallon of gas is disposed
11.StopPump() -stop pumping gas

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

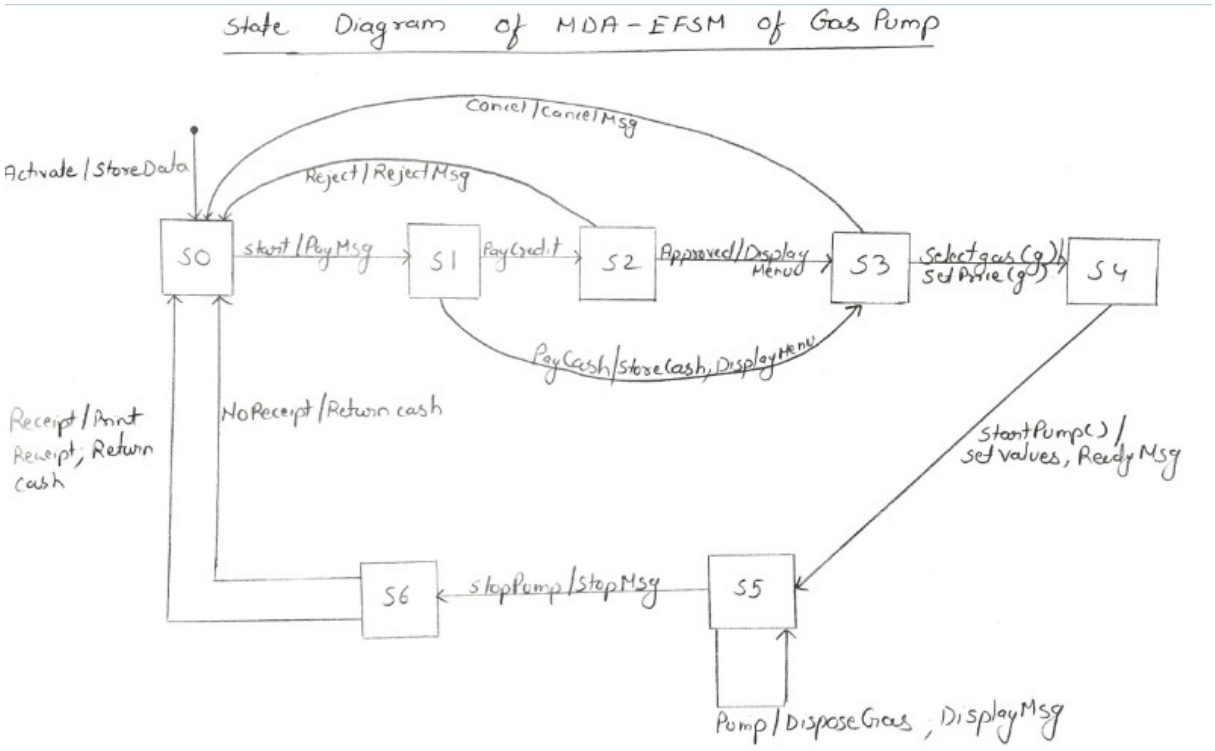
1.Activate (int a, int b, int c) -the gas pump is activated where a is the price of Regular gas, b is the price of Premium gas and c is the price of Super gas per liter
2.Start() -start the transaction
3.PayCash(int c) -pay for gas by cash, where c represents prepaid cash
4.Cancel() -cancel the transaction
5.Premium() -Premium gas is selected
6.Regular() -Regular gas is selected
7.Super() -Super gas is selected
8.StartPump() -start pumping gas
9.PumpLiter() -one liter of gas is disposed
10.Stop -stop pumping gas
11.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.
There are certain 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
h. etc.

The goal of this project is to design two GasPump components using the Model-Driven Architecture(MDA).Designing an executable metamodel, referred to as MDA-EFSM, for GasPump components. This MDA-EFSM should capture the “generic behavior” of both GasPump components and should be de-coupled from data and implementation details. Design will be representing ONLY one MDA-EFSM for both GasPump components. In addition, in the Model-Driven Architecture coupling between components should be minimized and cohesion of components should be maximized (components with highcohesion and low coupling between components)

Figure1: State Diagram for MDA-EFSM of Gas Pump Model



List of events and actions for MDA-EFSM

Name - Abhinav Pimpalgaonkar
Student Id - A20387324

CS - 586
Spring 2017

Project
MDA - EFSM

A] List of Events

Activate ()
Start ()
Pay Credit ()
Pay Cash ()
Approved ()
Select gas (int g)
StartPump ()
Pump ()
StopPump ()
Reject ()
Cancel ()
Receipt ()
NoReceipt ()

B] List of MDA - EFSM Actions

StoreData - Store price of different types of gas in temp. data store
PayMsg - Display a message for type of payment
StoreCash - store cash in temporary data store
DisplayMenu - will display a menu with a list of options
RejectMsg - Display message for credit card rejection
SetPrice(int g) - It will set the price of gas with respect to the type of gas g
ReadyMsg - Display the ready status for pumping.
SetValues - set the initial value of G=0 or L=0
DisposeGas - Count the number of units disposed & disposes unit of gas
DisplayMsg - Display the amount of gas disposed
stopMsg - Display a message for stopping pump
PrintReceipt - It will print a receipt
ReturnCash - Returns the cash
CancelMsg - Display a cancel message

Operation of Input Processor(Gas Pump 1)

Pseudcode

1) Pseudocode of input processor of Gas Pump-I

```
1) Activate (float a, float b)
   { if (a>0 && b>0)
       { d -> temp_a = a;
         d -> temp_b = b;
         m -> Activate();
   } End if

2) Start ()
   { m -> Start();
   }

3) PayCredit ()
   { m -> PayCredit();
   }

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

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

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

7) Super ()
   { m -> selectgas(2);
   }

8) Regular ()
   { m -> Selectgas(1);
   }

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

10) PumpGallon ()
    { m -> Pump();
    }

11) StopPump ()
    { m -> stopPump();
      m -> Receipt();
    }
```

KeyNote

d – Pointer to object of Data Store
m – Pointer to object of MDA EFSM
temp-a – Temporary Variable for storing a
temp-b – Temporary Variable for storing b

Operation of Input Processor(Gas Pump 2)

2) Pseudocode for Input Processor of Gas Pump 2

```
1) 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();
   } End if

2) start ()
   { m -> start();
   }

3) PayCash (int c)
   { if (c>0)
       { d -> temp_c = c;
         m -> PayCash();
   } End if

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

5) Premium ()
   { m -> selectGas(3);
   }

6) Regular ()
   { m -> selectGas(1);
   }

7) Super ()
   { m -> SelectGas(2);
   }

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

9) PumpLiter ()
   { if (d->ecash >= (d->price)*(d->L+1
       m -> Pump();
     else
       if (d->cash < (d->price)*(d->L+1)
         { m -> StopPump()
     End if
   }

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

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

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

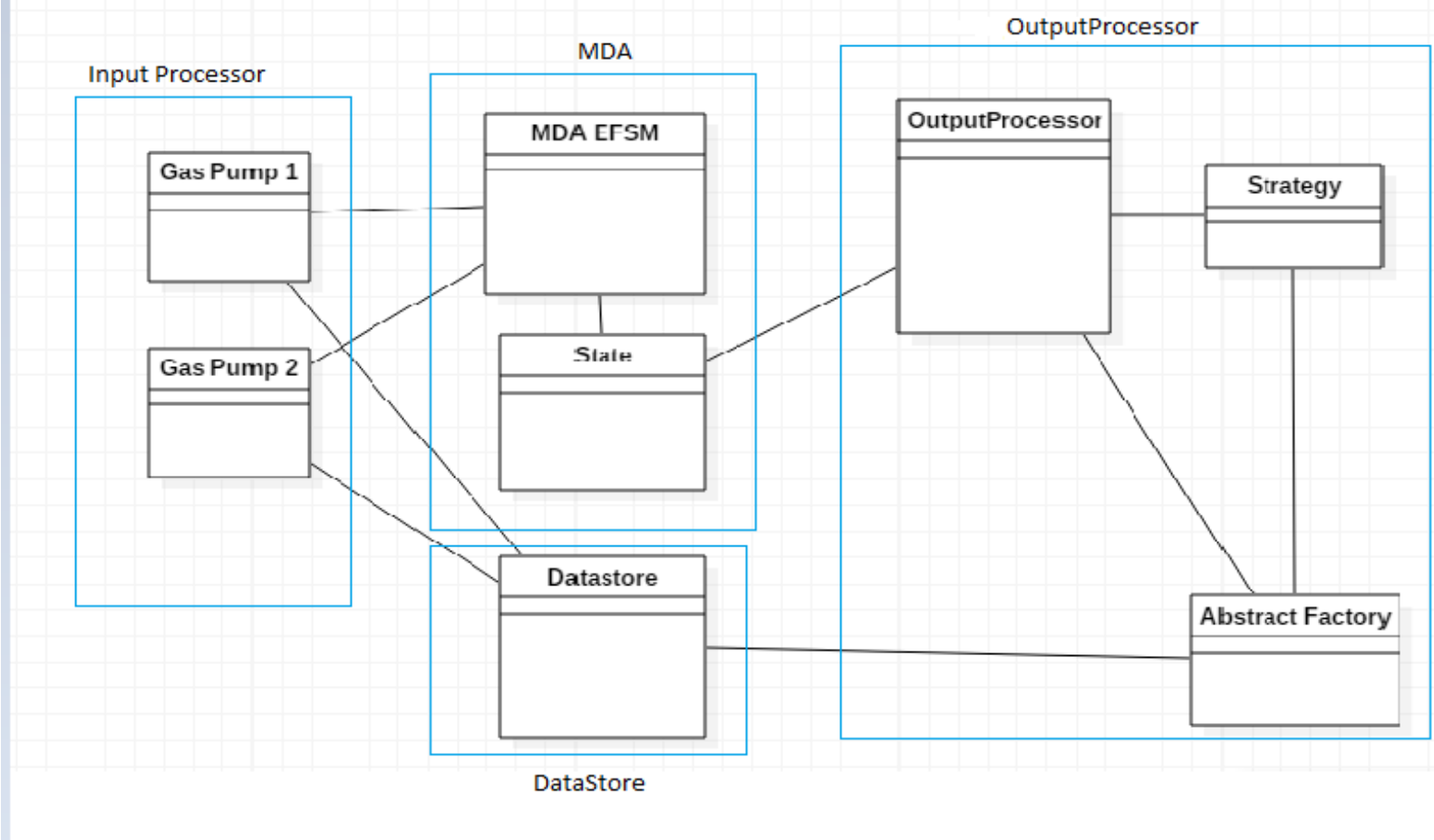
KeyNote

d – Pointer to object of DataStore
m – Pointer to object of MDA EFSM
temp-a – Temporary Variable for storing a
temp-b – Temporary variable for storing b
temp-c – Temporary variable for storing c
cash – cash value deposited
price – selected gas price
L – Number of Liters already pumped.

General Architecture of Gas Pump Component

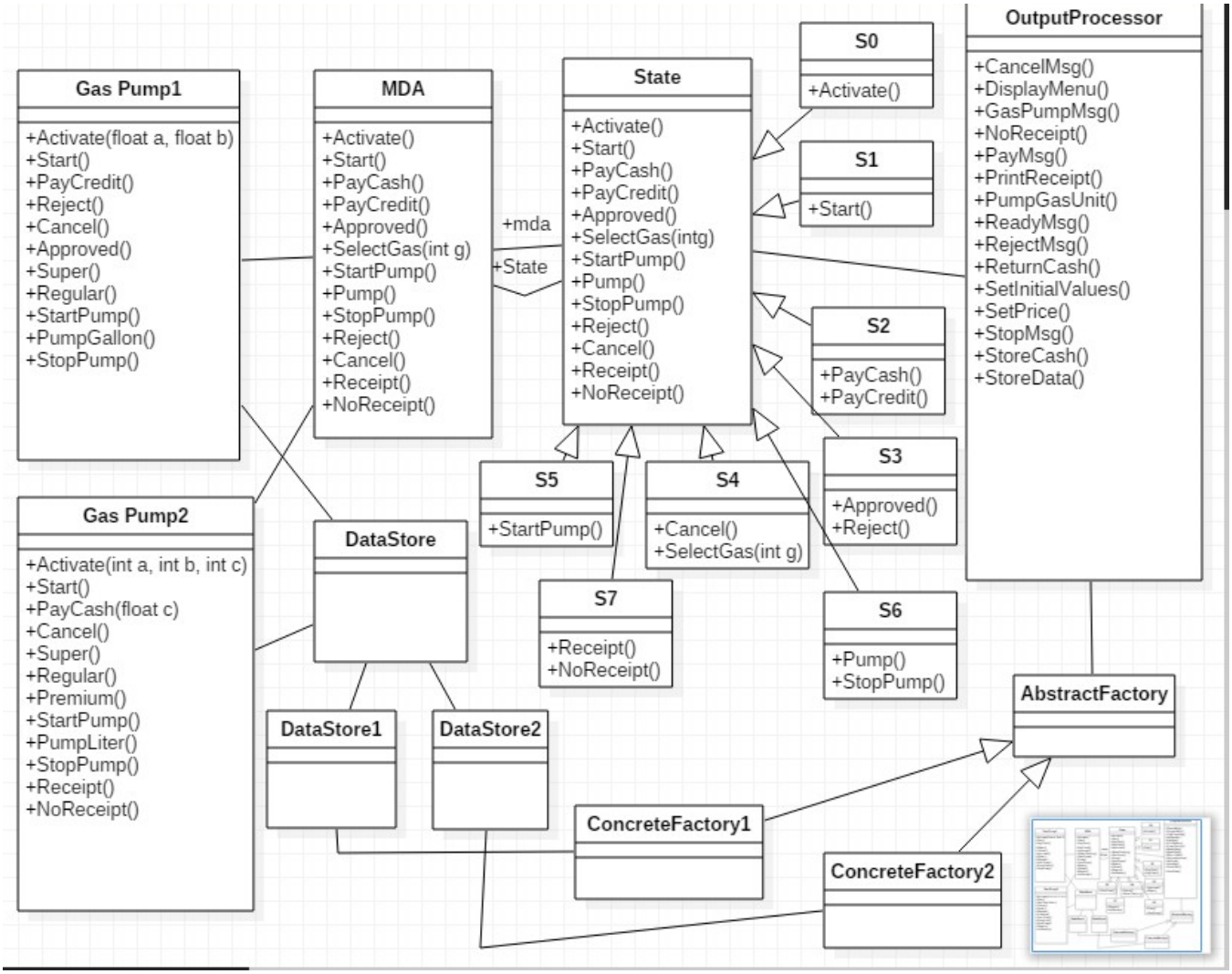
The general architecture implements two gas pump components using MDA .The project uses following design patterns

1. State Pattern
2. Strategy Pattern
3. Abstract Factory Pattern



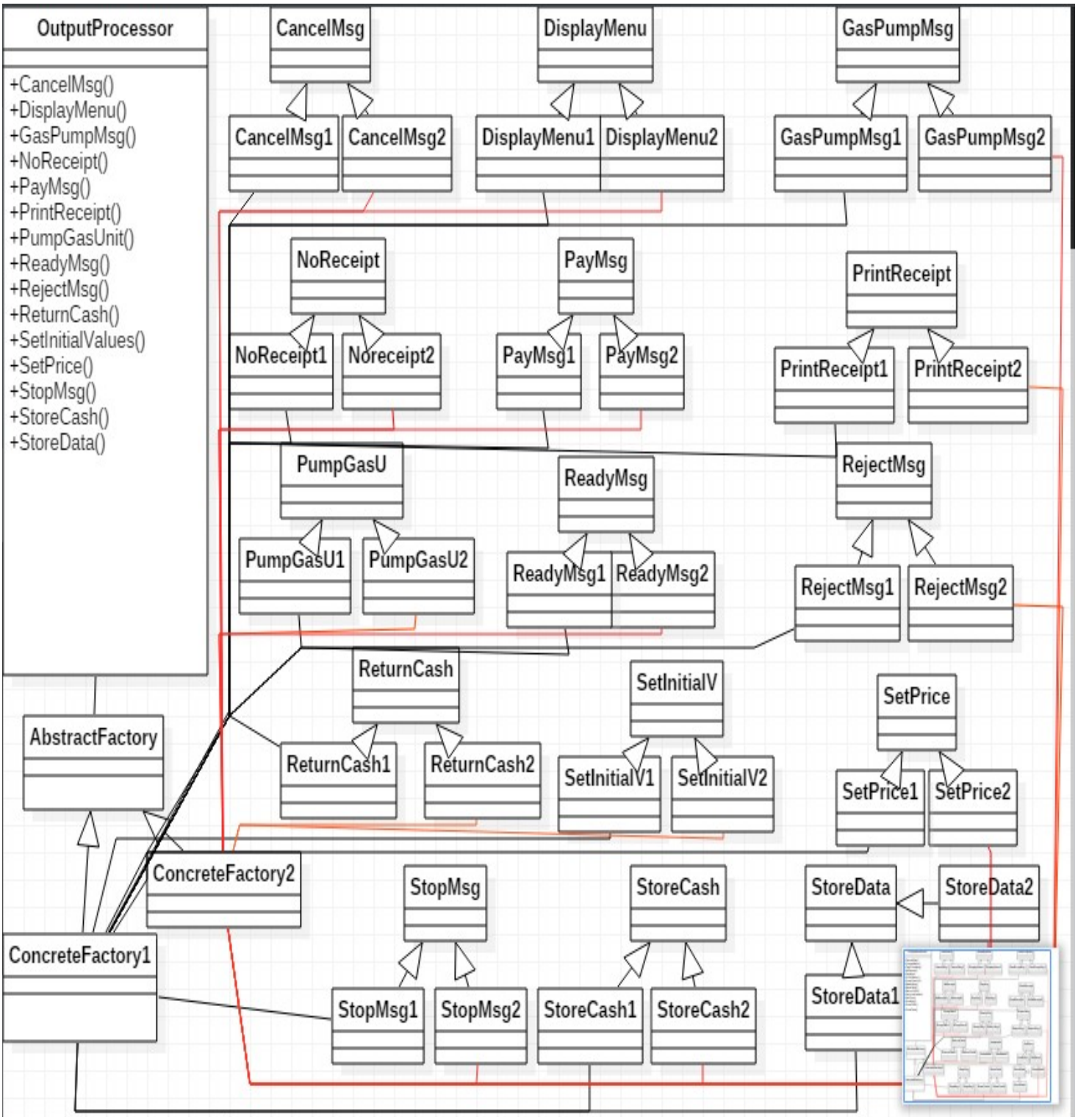
Detail Class Diagram of Gas Pump Components(State Pattern and AbstractFactory)

The detail diagram of Gas Pump will indicate objects/pointers which will be private to class. Member functions indicating the functionality of each class. Relationship between the classes.

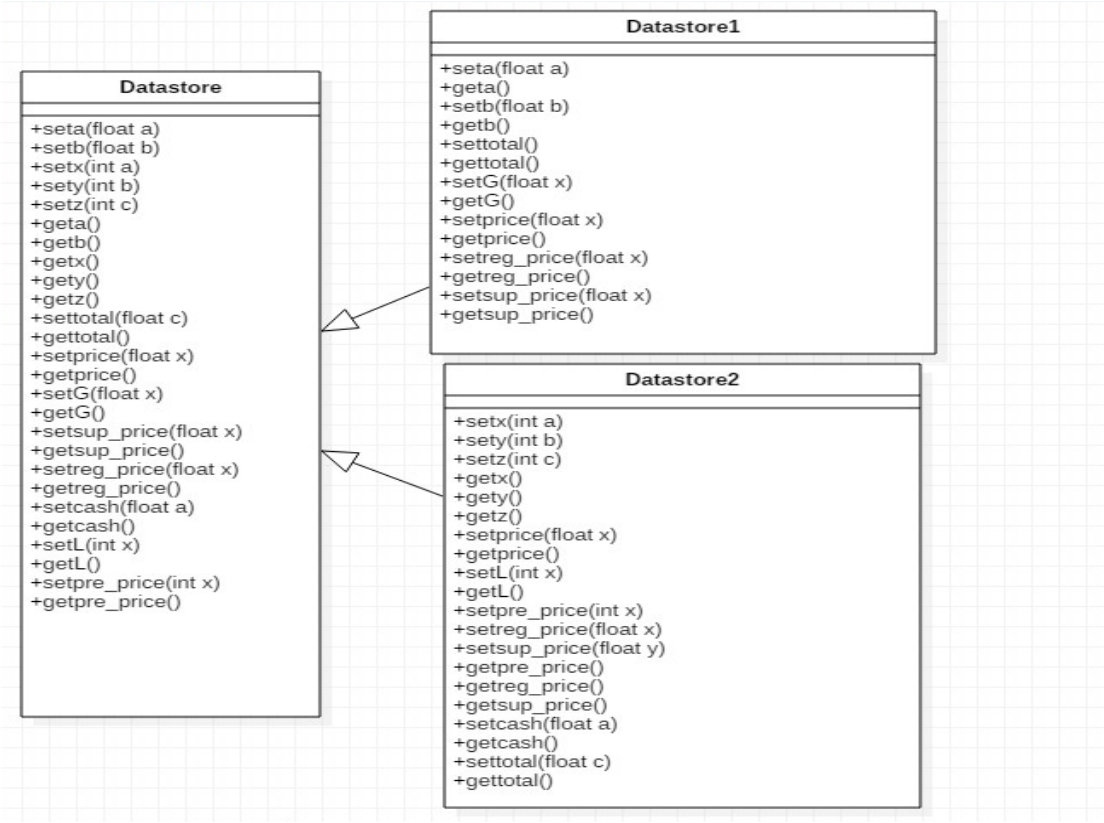


Note-Strategy pattern implementation will be showing relationship with the OutputProcessor Class and Abstract Factory Class. The implementation of strategy pattern is provided below

Detail Implementation of Strategy Pattern and Abstract Factory



DataStore Abstract Class and Subclass



Private Members/Objects declaration and responsibility in each class

1.class GP1
af abfact_object;
mda mda_object;
datastore datastore_object;

Responsibility-This class is initialize when user selects GasPump1. It consists of all the processing done by GasPump1 component.
Operations- Pseudo-code of Gas Pump 1 component is provided above which provide detail functionality of each operation

2.classGP2
af abfact_object;
mda mda_object;
datastore datastore_object;

Responsibility-This class is initialize when user selects GasPump2. It consists of all the processing done by GasPump2 component.
Operations- Pseudo-code of Gas Pump 2 component is provided above which provide detail functionality of each operation

3. Class ConcreteFactory cf1
datastore1 datastore1_object

Responsibility-This class is responsible for creating objects of the data stores and actions associated with GasPump1

Operations
1.StoreData getStoreData()-Create an object of the class StoreData1
2.PayMsg getPayMsg()-Create an object of the class PayMsg1
3.DisplayMenu getDisplayMenu()-Create an object of the class DisplayMenu1
4.RejectMsg getRejectMsg()-Create an object of the class RejectMsg1
5.CancelMsg getCancelMsg()-Create an object of the class CancelMsg1
6.SetPrice getSetPrice()-Create an object of the class SetPrice1
7.ReadyMsg getReadyMsg()-Create an object of the class ReadyMsg1
8.SetInitialValues getSetInitialValues()-Create an object of the class SetInitialValues1
9.PumpGasUnit getPumpGasUnit()-Create an object of the class PumpgasUnit1
10.GasPumpedMsg getGasPumpedMsg()-Create an object of the class GasPumpedMsg1
11.StopMsg getStopMsg()-Create an object of the class StopMsg1
12.PrintReceipt getPrintReceipt()-Create an object of the class PrintReceipt1
13.ReturnCash getReturnCash()-Create an object of the class ReturnCash1 //Dummy class
14.NoReceipt getNoReceipt()-Create an object of the class Noreceipt1
15.StoreCash getStoreCash()-Create an object of the class StoreCash1

4.Class ConcreteFactory cf2
datastore2 datastore2_object

Responsibility-This class is responsible for creating objects of the data stores and actions associated with GasPump2.
Operations-
1.StoreData getStoreData()-Create an object of the class StoreData1
2.PayMsg getPayMsg()-Create an object of the class PayMsg1
3.DisplayMenu getDisplayMenu()-Create an object of the class DisplayMenu1
4.RejectMsg getRejectMsg()-Create an object of the class RejectMsg1
5.CancelMsg getCancelMsg()-Create an object of the class CancelMsg1
6.SetPrice getSetPrice()-Create an object of the class SetPrice1
7.ReadyMsg getReadyMsg()-Create an object of the class ReadyMsg1
8.SetInitialValues getSetInitialValues()-Create an object of the class SetInitialValues1
9.PumpGasUnit getPumpGasUnit()-Create an object of the class PumpgasUnit1
10.GasPumpedMsg getGasPumpedMsg()-Create an object of the class GasPumpedMsg1
11.StopMsg getStopMsg()-Create an object of the class StopMsg1

12.PrintReceipt getPrintReceipt()-Create an object of the class PrintReceipt1
13.ReturnCash getReturnCash()-Create an object of the class ReturnCash1 //Dummy class
14.NoReceipt getNoReceipt()-Create an object of the class Noreceipt1
15.StoreCash getStoreCash()-Create an object of the class StoreCash1

5.Class datastore1
static float temp_a;
static float temp_b;
static float price;
static float total;
static float G;
static float reg_price;
static float sup_price;

Responsibility- This class is responsible for datastore1 which represents storage class for GasPump1.It contains various functions responsible for storing and retrieving temporary and permanent data for GasPump1.
Operations-
1.public void seta(float a)-Set the value of regular fuel in temporary variable
2.public void setb(float b)-set the value of super fuel in temporary variable
3.public float geta()-return value of regular fuel
4.public float getb()-return value of super fuel
5.public void settotal(float c)-Set the total amount that needs to be payed
6.public float gettotal()-return the total amount that needs to be payed
7.public void setG(float y)-Set the number of Gallons
8.public float getG()-return the number of gallons
9.public void setprice(float y)-Set the price of fuel choosed from regular or super
10.public float getprice()-return the price of fuel
11.public void setsup_price(float y)-set the price for super fuel
12.public float getsup_price()-return the priceof super fuel
13.public void setreg_price(float y)-set price of regular fuel
14.public float getreg_price()-return the price of regular fuel

6.Class datastore2
static int temp_a;
static int temp_b;
static int temp_c;
static int L;
static int reg_price;
static int pre_price;
static int sup_price;
static float cash;
static float total;

static int price;

Responsibility-This class is responsible for datastore2 which represents storage class for GasPump2.It contains various functions responsible for storing and retrieving temporary and permanent data for GasPump2.

Operations-
1.public float getprice()-Return the price of fuel selected
2.public void setprice(float y)-set the price of fuel selected
3.public void setx(int a)-set the value of regular fuel during activation in temporary variable
4.public void sety(int b)-set the value of super fuel in temporary variable
5.public int gety()-return the value stored in temporary variable for super fuel
6.public int getx()-return the value stored in temporary variable for regular fuel
7.public void setz(int a)-set the value of premium fuel during activation in temporary variable
8.public int getz()-eturn the value stored in temporary variable for premium fuel
9public void setL(int a)-set the number of liters of fuel disposed
10.public int getL()-return the number of liters of fuel disposed from pump
11.public void setpre_price(int a)-set the price of premium fuel
12.public int getpre_price()-return the price of premium fuel
13.public void setreg_price(float a)-set the price of regular fuel
14.public float getreg_price()-return the price of regular fuel
15.public void setsup_price(float a)-set the price of super fuel
16.public float getsup_price()-return the price of super fuel
17.public void setcash(float a)-set the cash value which user provided
18.public float getcash()-return the cash
19.public float gettotal()that -set the total amount that needs to be payed
20.public void settotal(float a)-return the total amount that needs to be payed

7.Class mda
s sid; //sid is used to point to the current state object
s[] list = new s[8];

Responsibility-This class is responsible for change of states. These states are responsible for performing actions. Class MDA represents the platform independent behavior and manages the State pattern behaviour.

Operation-
1.public void setStates(s a)-Responsible for setting the state id
2.public void setStatesList(s t1, s t2, s t3, s t4, s t5, s t6,s t7,s t8)-Setting the states
3.public void Activate()-call the activate method in state s0
4.public void PayCredit()-call the state method pay
5.public void Approved()-call the state method Approved
6.public void Reject()-call the state method reject
7.public void SelectGas(int g)-Call the state method SelectGas(int g)
8.public void Cancel()-Call the state method Cancel

9.public void StartPump()-call the state method StartPump
10.public void StopPump()-call the state method StopPump
11.public void Receipt()-Call the state method Receipt
12.public void NoReceipt()-Call the state method NoReceipt
13.public void Start()-Call the state method Start

8.Class OutputProcessor
static af abfact_object
datastore datastore_object

Responsibility-This class is responsible for performing meta actions.

Operations-
1.public void setfactory(af af1)-Set object for abstract factory
2.public void setData(datastore d1)-set the object for datastore
3.public static void storeData()-call the factory to get meta action and set the class pointer StoreData
4.public static void displayMenu()--call the factory to get meta action and set the class pointer DispalyMenu
5.public static void payMsg() -call the factory to get meta action and set the class pointer PayMsg
6.public static void PayCredit()-call the factory to get meta action and set the class pointer PayCredit
7.public static void rejectMsg()-call the factory to get meta action and set the class pointer rejectMsg
8.public static void printReceipt()-call the factory to get meta action and set the class pointer PrintReceipt
9.public static void cancelMsg()-call the factory to get meta action and set the class pointer CancelMsg
10.public static void storeCash()-call the factory to get meta action and set the class pointer StoreCash
11.public static void gasPumpedMsg()-call the factory to get meta action and set the class pointer gasPumpedMsg
12.public static void pumpGasUnit()-call the factory to get meta action and set the class pointer pumpGasUnit
13.public static void readyMsg()-call the factory to get meta action and set the class pointer readyMsg
14.public static void stopMsg()-call the factory to get meta action and set the class pointer StopMsg
15.public static void setPrice(int g)-call the factory to get meta action and set the class pointer SetPrice
16.public static void setInitialValues()-call the factory to get meta action and set the class pointer SetInitialValues
17.public static void NoReceipt()-call the factory to get meta action and set the class pointer NoReceipt

18.public static void ReturnCash()-call the factory to get meta action and set the class pointer ReturnCash

9.Class s
OutputProcessor outp_object;
int sid;

Responsibility-This is an abstract class responsible for implementing different state classes that will be responsible for performing actions.

Operations on classes
1.s0
Activate()-Display Activation message and call OutputProcessor StoreData function

2.s1
Start()-call OutputProcessor PayMsg function

3.s2
PayCredit()-call OutputProcessor PayCredit function
PayCash()-Call OutputProcessor StoreCash function alongwith displayMenu function

4.s3
Approved()-call OutputProcessor DisplayMenu function
Reject()-Call OutputProcessor RejectMsg function

5.s4
SelectGas(int g)-Call setprice(g) function of OutputProcessor
Cancel()-Call CancelMsg and ReturnCash function of OutputProcessor

6.s5
StartPump()-Call SetInitial Values and ReadyMsg fuction of OutputProcessor

7.s6
Pump()-Call PumpGasUnit and GasPumpedMsg function of OutputProcessor
StopPump()-Call StopPumpMsg of OutputProcessor

8.s7
Receipt()-Call Print Receipt function of OutputProcessor
NoReceipt()-Call NoReceipt and ReturnCash function of OutputProcessor

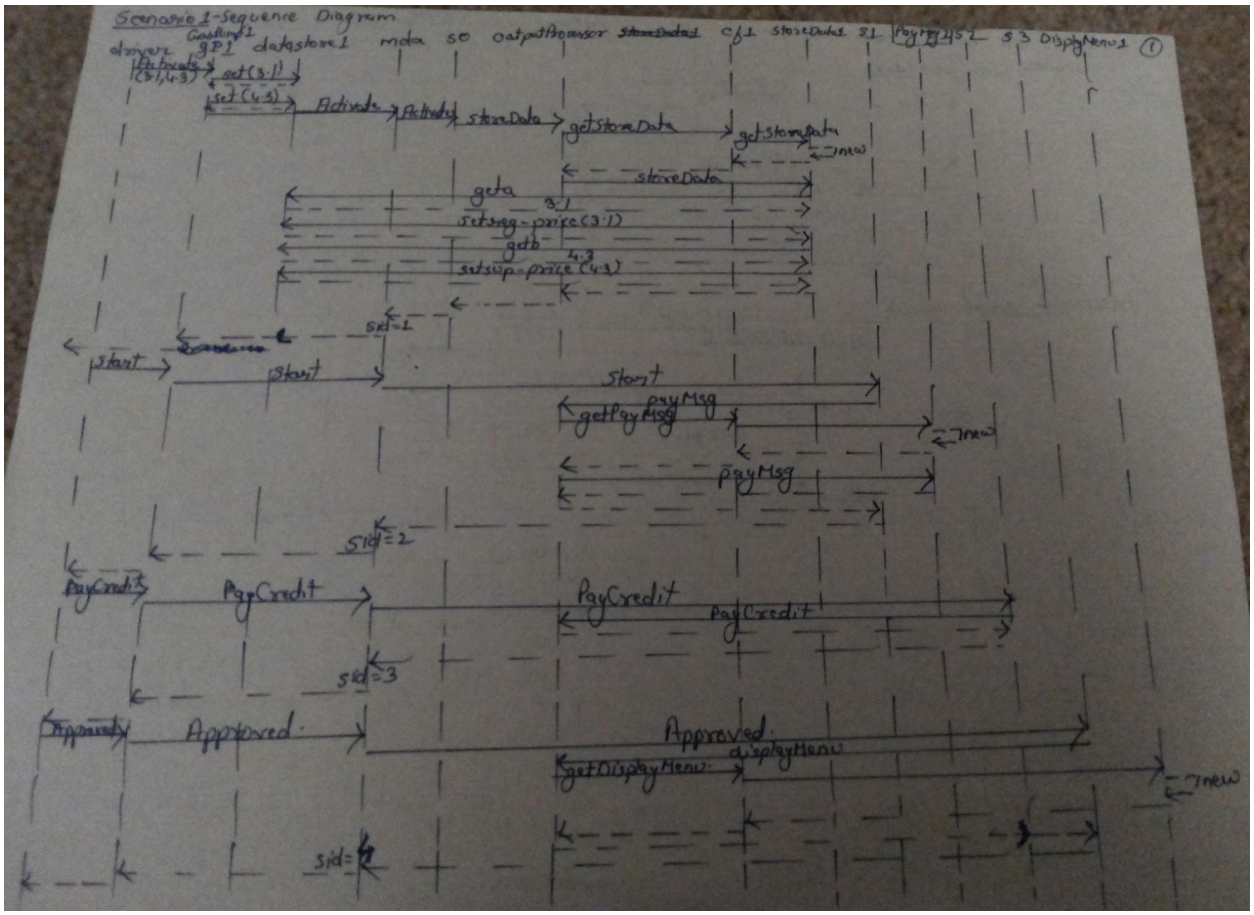
10.Strategy Pattern Classes(example getReadyMsg,PumpGasUnit,StopMsg etc)
All the abstract class that represent meta actions will contain object/pointer of OutputProcessor.

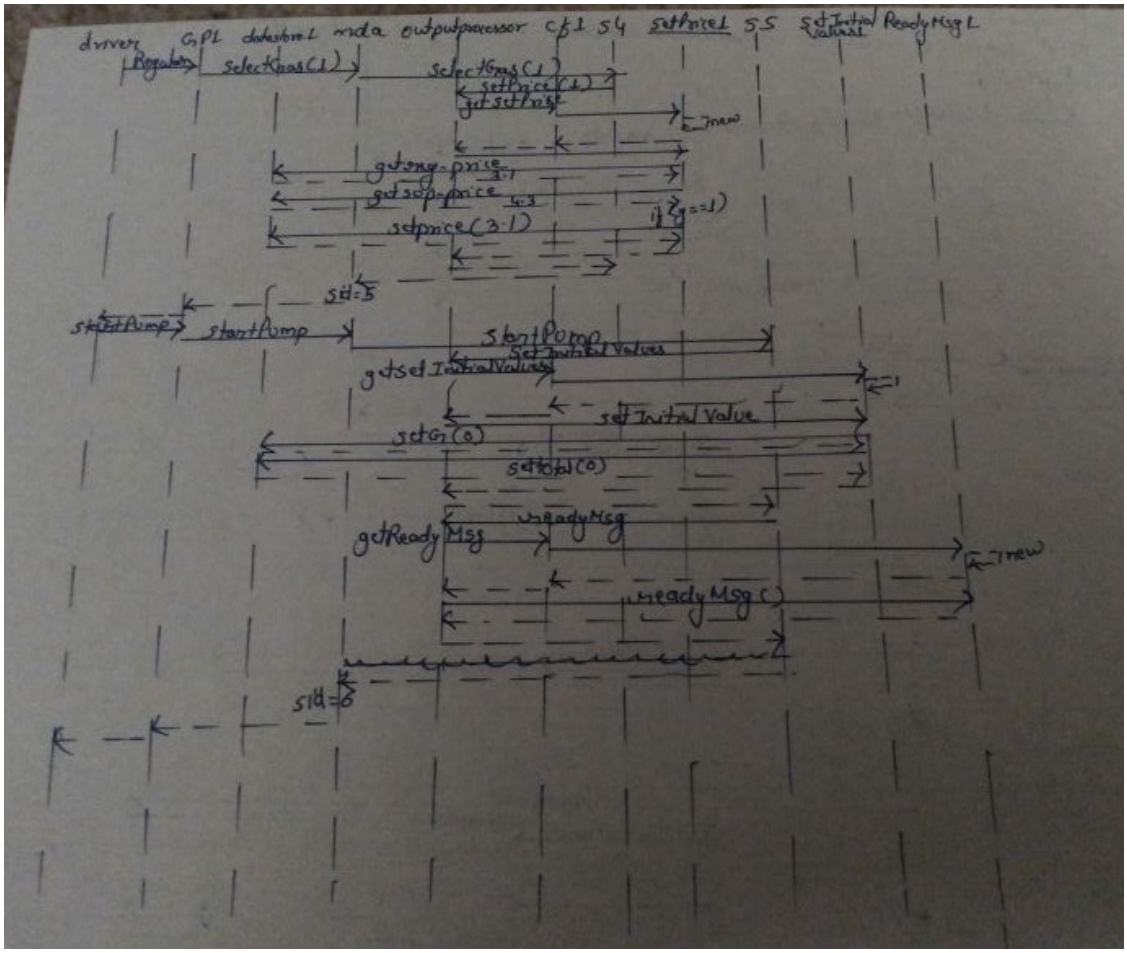
Responsibility-The classes performs various function based on the strategy choice.

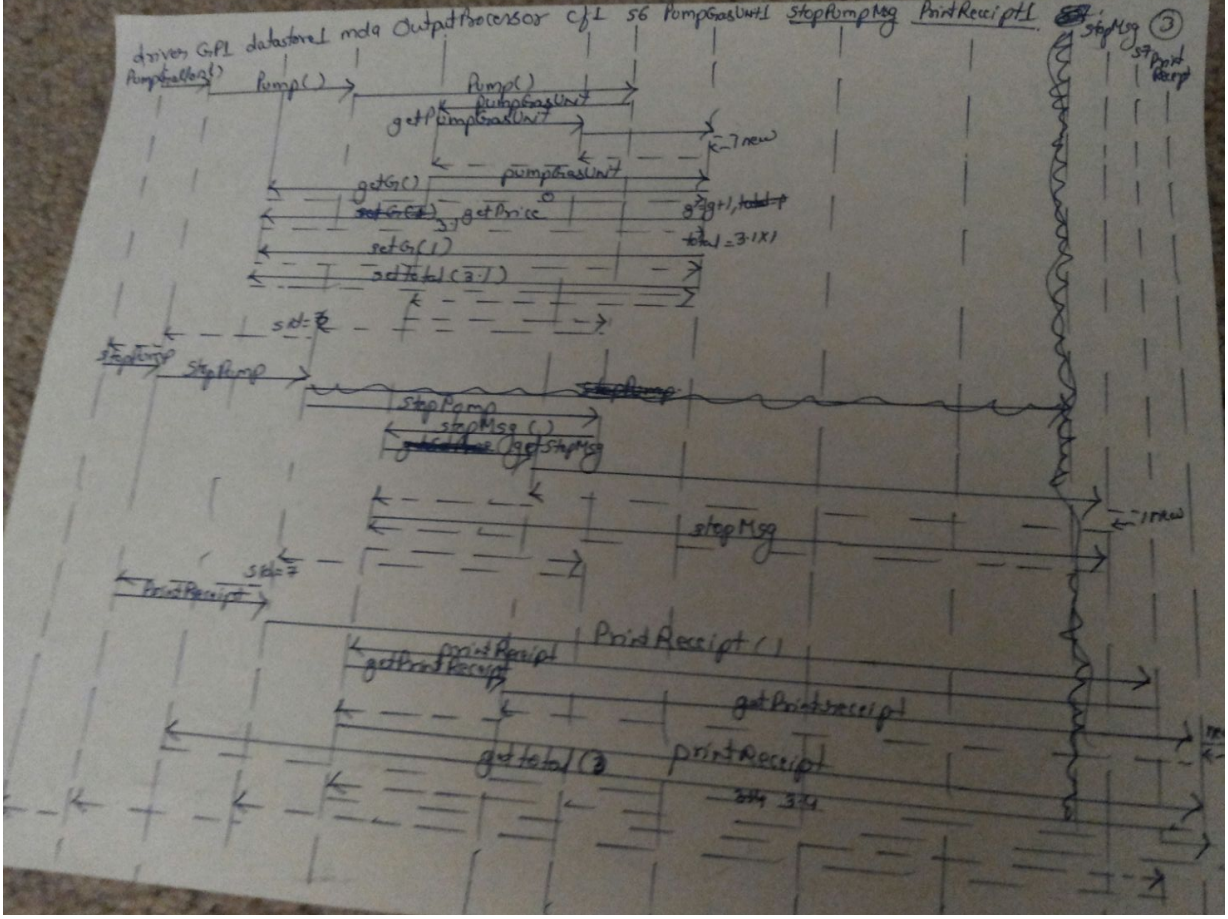
Operations-
1.StoreData- Responsible for storing value of fuels in datastore
2.PayMsg -Dispaly PayMessage
3.DisplayMenu -Diplay the list of fuels available for users
4.RejectMsg -display Reject Message
5.CancelMsg -display cancel message
6.SetPrice -Responsible for storing the price in the datastore based on the user input for fuel selected
7.ReadyMsg-Display Ready message
8.SetInitialValues-set initial value of gallon/liters and total
9.PumpGasUnit-Increment the number of gallons/liters of fuel disposed and set the total amout that needs to be payed by the user based on price of fuel selected and number of gallons disposed
10.GasPumpedMsg- Display number of gallons/liters disposed
11.StopMsg-Display stop message of the pump
12.PrintReceipt-Print the receipt
13.ReturnCash-Return the cash based on cash provided by the user and amount that needs to be payed by the user.
14.NoReceipt-Display nothing
15.StoreCash- Responsible for storing cash of the user provided during PayCash option

Sequence Diagram

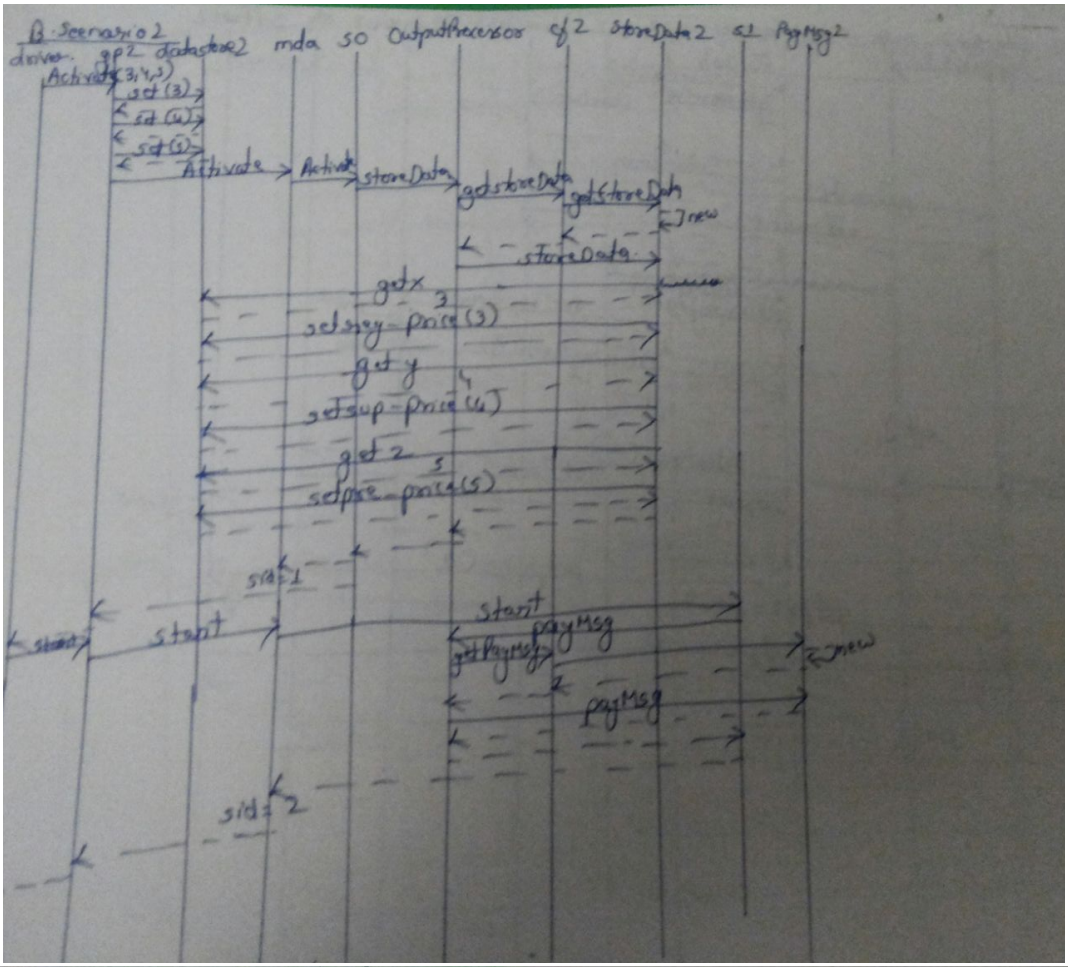
1.Scenario 1

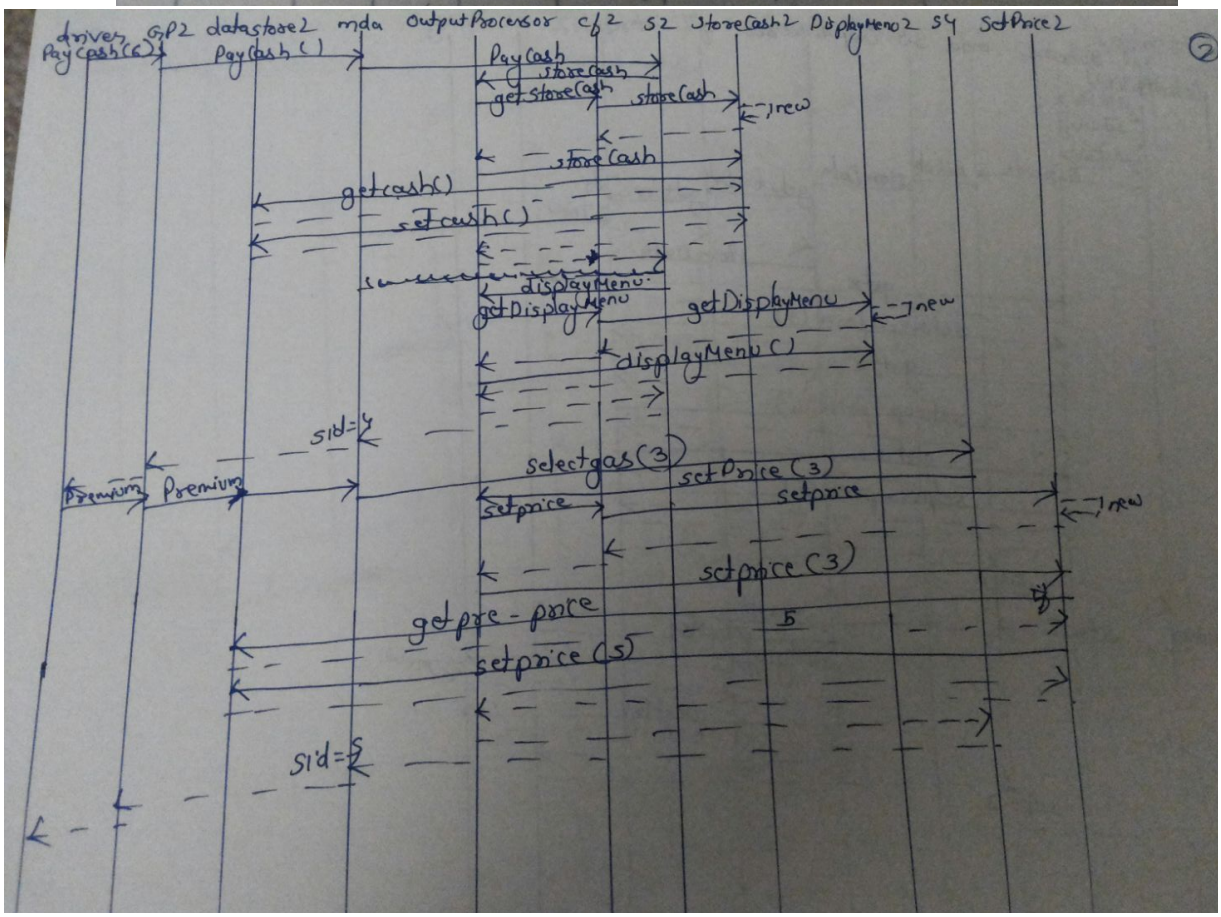


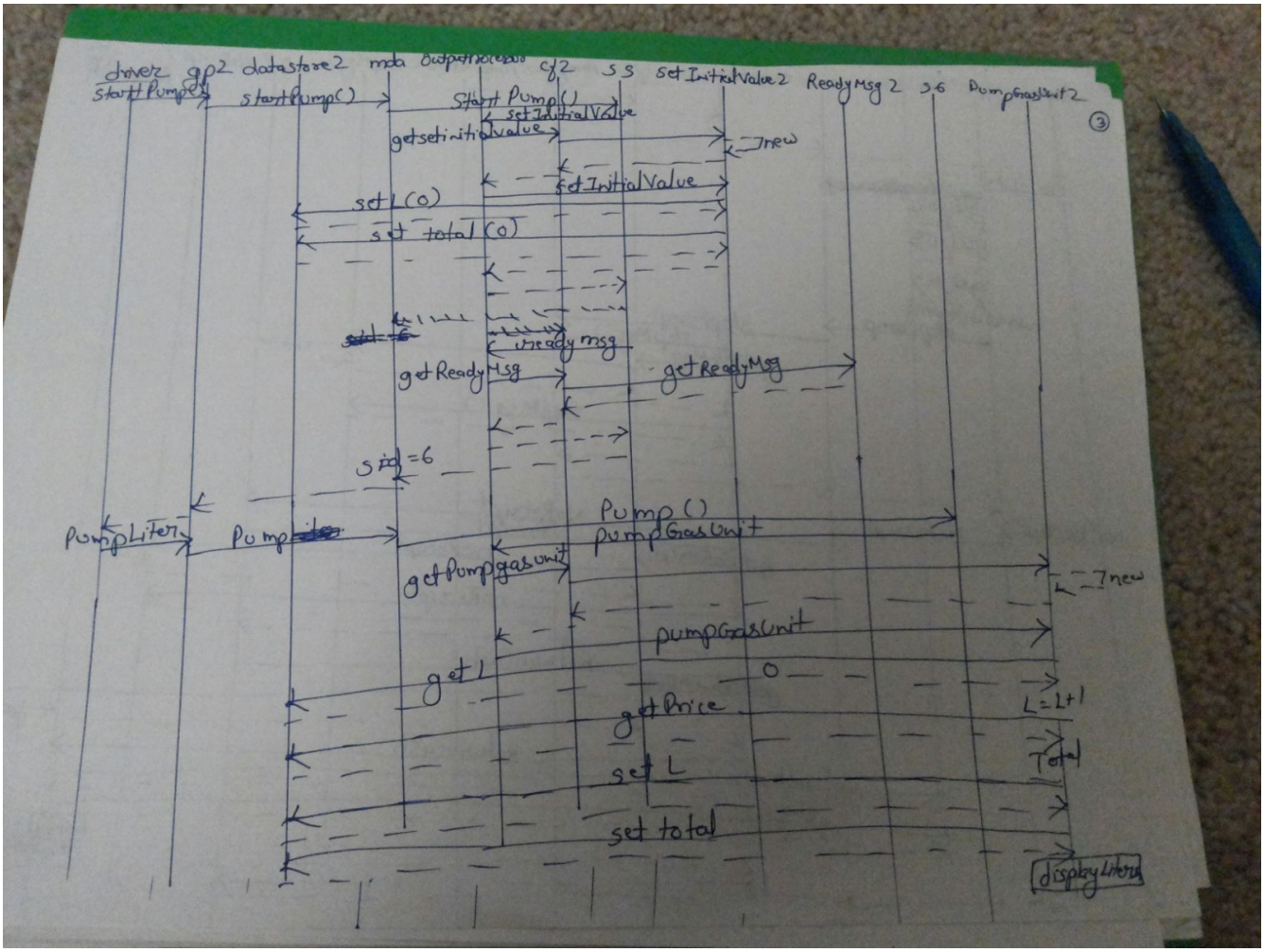

driver
GPL
datastore
mda
outputprocessor
cfl
s4
setPrice1
s5
setInitialReadyMsg
Register
selectBeas(1)
selectMsg(1)
setPrice(1)
getSetMsg
new
getLong-price
getStop-price
setprice(3:1)
i==1
s4=5
startPump
startPump
getSetInitialValues
setInitialValues
setG1(0)
setR1(0)
getReadyMsg
readyMsg
new
readyMsg()
s4=6


driver GUI datastore mda OutputProcessor cfl 56 PumpGasUnit1 StopPumpMsg PrintReceipt1
pump()
getPumpGasUnit
pumpGasUnit
getGs()
getPrice
setGs()
setTotal(3.1)
stopPump
stopMsg()
getStopMsg
stopMsg
PrintReceipt()
getPrintReceipt
getTotal()
printReceipt

Scenario 2


B. Scenario 2
driver. gP2 datastore2 mda SO OutputProcessor cf2 StoreData2 c1 PayMsg2
Activate
set(3)
set(4)
set(5)
Activate
Activate
storeData
getstoreData
getStoreData
new
storeData
getx
3
setkey-price(3)
gety
4
setsup-price(4)
getz
5
setpre-price(5)
sid=1
start
start
start
payMsg
getPayMsg
new
payMsg
sid=2


driver gP2 datastore2 mda OutputProcessor cf2 S2 StoreCash2 DisplayMenu2 S4 SetPrice2
PayCash()
PayCash()
PayCash
storeCash
getstoreCash
storeCash
new
storeCash
getcash()
setcash()
displayMenu
getDisplayMenu
getDisplayMenu
new
displayMenu()
sid=4
selectgas(3)
Premium
Premium
setPrice(3)
setprice
setprice
new
setprice(3)
getpre-price
5
setprice(5)
sid=5


driver
gp2
datastore2
cf2
SS
setInitialValue2
ReadyMsg 2
56
PumpGasUnit2
startPump()
Start Pump()
setInitialValue
getsetinitialvalue
new
setInitialValue
setL(0)
set total(0)
sid=6
ready msg
getReadyMsg
getReadyMsg
sid=6
Pump()
PumpLiter
Pump
pumpGasUnit
getPumpgasunit
new
pumpGasUnit
getL
0
getPrice
L=L+1
setL
Total
set total
displayLiters

Source Code
1.

```
package abstractfactory;

import strategy.*;


public abstract class af
{

    public abstract StoreData getStoreData();
    public abstract PayMsg getPayMsg();
    public abstract StoreCash getStoreCash();
    public abstract DisplayMenu getDisplayMenu();
    public abstract RejectMsg getRejectMsg();
    public abstract CancelMsg getCancelMsg();
    public abstract SetPrice getSetPrice();
    public abstract ReadyMsg getReadyMsg();
    public abstract SetInitialValues getSetInitialValues();
    public abstract PumpGasUnit getPumpGasUnit();
    public abstract GasPumpedMsg getGasPumpedMsg();
    public abstract StopMsg getStopMsg();
    public abstract PrintReceipt getPrintReceipt();
    public abstract ReturnCash getReturnCash();
    public abstract NoReceipt getNoReceipt();

}
```

2.

```
package abstractfactory;

import datastore.*;
import strategy.*;




public class cf1 extends af
{
     datastore1 datastore1_object= new datastore1();

          public StoreData getStoreData()
    {
                    StoreData sd_o =new StoreData1();
                    sd_o.setdata(datastore1_object);
```

```
            return new StoreData1();
}
    public SetInitialValues getSetInitialValues()
{
            SetInitialValues siv_o = new SetInitialValues1();
  siv_o.setdata(datastore1_object);
  return siv_o;
}
    public SetPrice getSetPrice()
{
            SetPrice sp_o = new SetPrice1();
        sp_o.setdata(datastore1_object);
        return sp_o;
}
    public StoreCash getStoreCash()
 {
             StoreCash sc_o = new StoreCash1();
        sc_o.setdata(datastore1_object);
        return sc_o;

 }
    public datastore getdata()
    {
        return datastore1_object;
    }

     public GasPumpedMsg getGasPumpedMsg()
{
            GasPumpedMsg gpm_o = new GasPumpedMsg1();
        gpm_o.setdata(datastore1_object);
        return gpm_o;

}
     public ReadyMsg getReadyMsg()
{
             ReadyMsg rm_o = new ReadyMsg1();
        rm_o.setdata(datastore1_object);
        return rm_o;

}
     public PayMsg getPayMsg()
{
            PayMsg pm_o = new PayMsg1();
   pm_o.setdata(datastore1_object);
   return pm_o;
```

```
}
    public CancelMsg getCancelMsg()
    {
            CancelMsg cm_o = new CancelMsg1();
  cm_o.setdata(datastore1_object);
  return cm_o;
    }
    public StopMsg getStopMsg()
    {
            StopMsg sm_o = new StopMsg1();
  sm_o.setdata(datastore1_object);
  return sm_o;
    }
    public DisplayMenu getDisplayMenu()
     {
            DisplayMenu dm_o = new DisplayMenu1();
  dm_o.setdata(datastore1_object);
  return dm_o;
     }
     public PumpGasUnit getPumpGasUnit()
     {
          PumpGasUnit pgu_o = new  PumpGasUnit1();
        pgu_o.setdata(datastore1_object);
        return pgu_o;
     }

     public RejectMsg getRejectMsg()
     {
         RejectMsg rm_o = new RejectMsg1();
        rm_o.setdata(datastore1_object);
        return rm_o;
     }


     public PrintReceipt getPrintReceipt()
     {

                PrintReceipt pr =new PrintReceipt1();
                pr.setdata(datastore1_object);
            return pr;
     }
     public NoReceipt getNoReceipt()
     {
          NoReceipt rm_o = new NoReceipt1();
```

```
            rm_o.setdata(datastore1_object);
             //return rm_o;
              return rm_o;
        }
        public ReturnCash getReturnCash()
        {
                ReturnCash rm_o = new ReturnCash1();
                rm_o.setdata(datastore1_object);
                return rm_o;
        }


}
```

3.

```
package abstractfactory;

import datastore.*;
import strategy.*;


public class cf2 extends af
{
    datastore2 datastore2_object= new datastore2();  //creates data1 object for ATM1

        public PrintReceipt getPrintReceipt()
   {

                PrintReceipt pr_o =new PrintReceipt2();
                pr_o.setdata(datastore2_object);

            return pr_o;
   }
        public StoreData getStoreData()
   {
                StoreData sd_o =new StoreData2();
                sd_o.setdata(datastore2_object);
                return sd_o;
   }
        public SetInitialValues getSetInitialValues()
   {
                SetInitialValues siv_o = new SetInitialValues2();
                siv_o.setdata(datastore2_object);
      return siv_o;
   }
```

```
    public SetPrice getSetPrice()
{
            SetPrice sp_o = new SetPrice2();
            sp_o.setdata(datastore2_object);
        return sp_o;
}
    public StoreCash getStoreCash()
 {
              StoreCash sc_o = new StoreCash2();
              sc_o.setdata(datastore2_object);
        return sc_o;
 }
    public datastore getdata()
    {
        return datastore2_object;
    }

     public GasPumpedMsg getGasPumpedMsg()
     {
            GasPumpedMsg gpm_o = new GasPumpedMsg2();
        gpm_o.setdata(datastore2_object);
        return gpm_o;

     }
     public ReadyMsg getReadyMsg()
     {
             ReadyMsg rm_o = new ReadyMsg1();
        rm_o.setdata(datastore2_object);
        return rm_o;

     }
     public PayMsg getPayMsg()
     {
            PayMsg pm_o = new PayMsg1();
    pm_o.setdata(datastore2_object);
    return pm_o;

     }
     public CancelMsg getCancelMsg()
     {
            CancelMsg cm_o = new CancelMsg1();
            cm_o.setdata(datastore2_object);
    return cm_o;
     }
     public StopMsg getStopMsg()
```

```
    {
            StopMsg sm_o = new StopMsg1();
            sm_o.setdata(datastore2_object);
  return sm_o;
    }
    public DisplayMenu getDisplayMenu()
    {
            DisplayMenu dm_o = new DisplayMenu2();
            dm_o.setdata(datastore2_object);
  return dm_o;
     }
      public PumpGasUnit getPumpGasUnit()
      {
           PumpGasUnit pgu_o = new  PumpGasUnit2();
           pgu_o.setdata(datastore2_object);
          return pgu_o;
      }

      public RejectMsg getRejectMsg()
      {
           RejectMsg rm_o = new RejectMsg1();
           rm_o.setdata(datastore2_object);
          return rm_o;
      }

      public  NoReceipt getNoReceipt()
      {
           NoReceipt nr_o = new NoReceipt2();
           nr_o.setdata(datastore2_object);
          return nr_o;
      }
      public ReturnCash getReturnCash()
      {
                  ReturnCash rm_o = new ReturnCash2();
                  rm_o.setdata(datastore2_object);
                  return rm_o;
      }

}
```

4.

```
package datastore;

public abstract class datastore {
```

```
	public void seta(float a)
	{
	}
	public void setb(float b)
	{
	}
	public float geta()
	{
		return 0;
	}

	public float getb()
	{
		return 0;
	}
	public void settotal(float c)
	{
	}
	public float gettotal()
	{
		return 0;
	}

	public void setprice(float y)
	{

	}
	public float getprice()
	{
	  return 0;
	}
	public void setG(float y)
	{

	}
public float getG()
	{
	   return 0;
	}
public void setsup_price(float y)
	{

	}
public float getsup_price()
```

```
	{
	  return 0;
	}
public void setreg_price(float y)
	{

	}
public float getreg_price()
	{
	  return 0;
	}

public void setx(int a)
	{
	}
public int getx()
	{
	   return 0;
	}
public void sety(int a)
	{
	}
public int gety()
	{
		return 0;
	}
public void setz(int a)
	{
	}
public int getz()
	{
		return 0;
	}
public float getcash()
	{
		return 0;
	}
public void setcash(float a)
	{

	}

public void setL(int a)
	{
```

```
	}
	public int getL()
	{
		return 0;
	}
	 public void setpre_price(int a)
	{

	}
	public int getpre_price()
	{
		return 0;
	}



}
```

5.

```
package datastore;

public class datastore1 extends datastore {


	static float temp_a;
	static float temp_b;
	static float price;
	static float total;
	static float G;
	static float reg_price;
	static float sup_price;

	public void seta(float a)
	{
		temp_a =a;
	}
	public void setb(float b)
	{
		temp_b=b;
	}
	public float geta()
	{
		return temp_a;
	}
```

```
	public float getb()
	{
		return temp_b;
	}

	public void settotal(float c)
	{
		 total =c;
	}
	public float gettotal()
	{
		return total;
	}

public void setG(float y)
	{
	   G=y;
	}
public float getG()
	{
	   return G;
	}
public void setprice(float y)
	{
		price=y;
	}
public float getprice()
	{
	  return price;
	}
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;
```

```
	}

}
```

6.

```
package datastore;

public class datastore2 extends datastore
{
	static int temp_a;
	static int temp_b;
	static int temp_c;
	static int L;
	static int reg_price;
	static int pre_price;
	static int sup_price;
	static float cash;
	static float total;
	static int price;

	public float getprice()
	{
		return price;
	}
	 public void setprice(float y)
	{
			price=(int)y;


	}
	 public void setx(int a)
	{
		temp_a =a;
	}
	public void sety(int b)
	{
		temp_b=b;
	}
	public int gety()
	{
		return temp_b;
	}
	public int getx()
	{
```

```
            return temp_a;
    }
    public void setz(int a)
    {
            temp_c =a;
    }
    public int getz()
    {
            return temp_c;
    }
    public void setL(int a)
    {
             L=a;
    }
    public int getL()
    {
            return L;
    }
     public void setpre_price(int a)
    {
                    pre_price=a;
    }
    public int getpre_price()
    {
            return pre_price;
    }
    public void setreg_price(float a)
    {
                    reg_price=(int)a;

    }
    public float getreg_price()
    {
            return reg_price;
    }
     public void setsup_price(float a)
    {
            sup_price=(int)a;
    }
    public float getsup_price()
    {
            return sup_price;
}
    public void setcash(float a)
    {
```

```
                        cash=a;
        }
        public float getcash()
        {
                return cash;
        }
        public float gettotal()
        {
                return total;
        }
        public void settotal(float a)
        {
                        total=a;
        }

}
```

7.

```
package driver;

import abstractfactory.*;
import datastore.*;
import mdaefsm.mda;
import java.io.*;
import java.util.*;
import state.*;
import inputpro.GP1;
import inputpro.GP2;
import outputpro.OutputProcessor;


public class driver
{
        static Scanner input=new Scanner(System.in);
        static BufferedReader buf=new BufferedReader(new InputStreamReader(System.in));
        static boolean flag=true;
        public static void main(String[] args) throws NumberFormatException, IOException
   {
                while(flag)
                {
      System.out.println("\n          \t\t\t Gas Pump Model\t \t");
                System.out.println("\n       Select Gas Pump ");
                System.out.println("\n 1. GasPump-1  ");
                System.out.println("\n 2. GasPump-2  ");
```

```
            System.out.println("\n Enter your choice ");
            int choice = -1;
            try
            {
   choice = input.nextInt();
}
            catch (InputMismatchException e)
            {
   continue;
}
if(choice==1)
{


            GP1 g1 = new GP1(); //g1 represents the object of Gas Pump1
            mda mda_object = new mda();//creating an object of the mda class
            g1.setMDA(mda_object);//Set the mda with the mda object created
            OutputProcessor OutputProcessor_o = new OutputProcessor();
            cf1 cf1_object = new cf1();
            datastore datastore_object;
            datastore_object = cf1_object.getdata();
            g1.setdata(datastore_object);
            g1.setfactory(cf1_object);

            s0 S0_object  = new s0();
            S0_object.set_OutputProcessor(OutputProcessor_o);//Setting up the connections
            S0_object.setStateId(0);//Call to the setStateId function of the state class
            s1 S1_object = new s1();
            S1_object.set_OutputProcessor(OutputProcessor_o);
            S1_object.setStateId(1);
            s2 S2_object = new s2();
            S2_object.set_OutputProcessor(OutputProcessor_o);
            S2_object.setStateId(2);
            s3 S3_object = new s3();
            S3_object.set_OutputProcessor(OutputProcessor_o);
            S3_object.setStateId(3);
            s4 S4_object = new s4();
            S4_object.set_OutputProcessor(OutputProcessor_o);
            S4_object.setStateId(4);
            s5 S5_object = new s5();
            S5_object.set_OutputProcessor(OutputProcessor_o);
            S5_object.setStateId(5);
            s6 S6_object = new s6();
            S6_object.set_OutputProcessor(OutputProcessor_o);
            S6_object.setStateId(6);
```

```
            s7 S7_object = new s7();
            S7_object.set_OutputProcessor(OutputProcessor_o);
            S7_object.setStateId(7);
            OutputProcessor_o.setData(datastore_object);
            OutputProcessor_o.setfactory(cf1_object);// Setting the object of the concrete
factory 1
            mda_object.setStates(S0_object);

mda_object.setStatesList(S0_object,S1_object,S2_object,S3_object,S4_object,S5_object,S6_ob
ject,S7_object);
            String input=null;
            int ch;
            while(true)
            {

              System.out.println("\n\n Please select your option from the list of the available
choices for GasPump-1\t\t");
                                        System.out.println("\n 0.Activate (float a,float b)");
                                        System.out.println("\n 1.Start()");
                                        System.out.println("\n 2.PayCredit()");
                                        System.out.println("\n 3.Reject()");
                                        System.out.println("\n 4.Cancel() ");
                                        System.out.println("\n 5.Approved()");
                                        System.out.println("\n 6.Super()");
                                        System.out.println("\n 7.Regular()");
                                        System.out.println("\n 8.StartPump() ");
                                        System.out.println("\n 9.PumpGallon()");
                                        System.out.println("\n 10.StopPump() ");
                                        System.out.println("\n Press 11 to exit \n\n ");
                                        input=buf.readLine();
               ch=Integer.parseInt(input);

               switch(ch)
               {
                  case 0:   System.out.println("\n Enter the value of a for activating the Gas
Pump \n");
                            float a=Float.parseFloat(buf.readLine());
                            System.out.println("\n Enter the value of b for activating the Gas
Pump \n");
                            float b=Float.parseFloat(buf.readLine());
                            g1.Activate(a,b);
                            break;
                  case 1:   g1.Start();
                      break;
                  case 2:g1.PayCredit();
```

```
                    break;
            case 3:  g1.Reject();
                    break;
            case 4:g1.Cancel();
                break;
            case 5:g1.Approved();
                break;
            case 6:g1.Super();
                   break;
            case 7:g1.Regular();
                   break;
            case 8:
                g1.StartPump();
               break;
            case 9:
               g1.PumpGallon();
               break;
            case 10:
               g1.StopPump();
               break;
            case 11:
                        return;
            default:
               System.out.println("\n \n Invalid Input Try Again");
         }
      }
}
else
if(choice==2)
{


     GP2 g2 = new GP2(); //g2 represents the object of Gas Pump2
     mda mda_object = new mda();
     g2.setMDA(mda_object);//Setting the mda with the mda object created
     OutputProcessor OutputProcessor_o = new OutputProcessor();
    cf2 cf2_object = new cf2();
    datastore datastore_object;
    datastore_object = cf2_object.getdata();
    g2.setfactory(cf2_object);// Setting the object of the concrete factory 2
    g2.setdata(datastore_object);
     s0 S0_object  = new s0();//S0_object represents the object Of state 0
     S0_object.set_OutputProcessor(OutputProcessor_o);
    S0_object.setStateId(0);
    s1 S1_object = new s1();
```

```
S1_object.set_OutputProcessor(OutputProcessor_o);
S1_object.setStateId(1);
s2 S2_object = new s2();
S2_object.set_OutputProcessor(OutputProcessor_o);
S2_object.setStateId(2);
s3 S3_object = new s3();
S3_object.set_OutputProcessor(OutputProcessor_o);
S3_object.setStateId(3);
s4 S4_object = new s4();
S4_object.set_OutputProcessor(OutputProcessor_o);
S4_object.setStateId(4);
s5 S5_object = new s5();
S5_object.set_OutputProcessor(OutputProcessor_o);
S5_object.setStateId(5);
s6 S6_object = new s6();
S6_object.set_OutputProcessor(OutputProcessor_o);
S6_object.setStateId(6);
s7 S7_object = new s7();
S7_object.set_OutputProcessor(OutputProcessor_o);
S7_object.setStateId(7);
OutputProcessor_o.setData(datastore_object);
OutputProcessor_o.setfactory(cf2_object);
mda_object.setStates(S0_object);

mda_object.setStatesList(S0_object,S1_object,S2_object,S3_object,S4_object,S5_object,S6_ob
ject,S7_object);
String input=null;
int ch;
while(true)
{
 System.out.println("\n\n Please select your option from the list of the available
choices for GasPump-2\t\t");
                                        System.out.println("\n\t\t 0.\t Activate (int a,int b,int c)
");
                                        System.out.println("\n\t\t 1.\t Start ");
                                        System.out.println("\n\t\t 2.\t PayCash(float c) ");
                                        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 Regular ");
                                        System.out.println("\n\t\t 6.\t Premium ");
                                        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 StopPump");
                                        System.out.println("\n\t\t 10.\t Receipt");
                                        System.out.println("\n\t\t 11.\t NoReceipt");
```

```
                                        System.out.println("\n\t\t Press any other 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 a to activate");
                        int a=(int)Float.parseFloat(buf.readLine());
                        System.out.println("\n\n Enter the value of b to activate");
                        int b=(int)Float.parseFloat(buf.readLine());
                        System.out.println("\n\n Enter the value of c to activate");
                        int c=(int)Float.parseFloat(buf.readLine());
                        g2.Activate(a,b,c);      //calls method activate in GasPump1
                        break;

                                                case 1: g2.Start();
                        break;

                                                case 2:
                                                        System.out.println("\n Enter the amount
you wish to pay by cash ");
                                String cash=buf.readLine();
                                float c1 =Float.parseFloat(cash);
                                g2.PayCash(c1);
                                        break;

                                                case 3: g2.Cancel();
                        break;

                                                case 4:g2.Super();
                        break;

                                                case 5:g2.Regular();
                        break;

                                                case 6:g2.Premium();
                        break;

                                                case 7: g2.StartPump();
                        break;
                                                case 8: g2.PumpLiter();
                        break;
                                                case 9: g2.StopPump();
                        break;
```

```
                                                    case 10: g2.Receipt();
                                    break;
                                                    case 11: g2.NoReceipt();
                                                            break;

                                default:
                                System.out.println("\n Invalid Input Try Again");
                                }

                                }

                        }
                        else
                                System.out.println("Invalid Input");
                                }
        }



    }
```

8.

```
//Creation of abstract factory 1,datastore data and MDA
//Gas Pump 1 list of functions is provided

package inputpro;
import abstractfactory.*;
import datastore.*;
import mdaefsm.*;

public class GP1
{
        af abfact_object;
        mda mda_object;
        datastore datastore_object;
        public void setMDA(mda m)
        {
                mda_object = m;
        }
        public void setdata(datastore d)
        {
```

```
        datastore_object = d;
    }
    public void setfactory(af f)
    {
        abfact_object = f;
    }
    public void Activate(float a,float b)
    {
        if(a>0 && b>0)
        {
            datastore_object.seta(a);
            datastore_object.setb(b);
            mda_object.Activate();
        }

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

    }
    public void PayCredit()
    {
        mda_object.PayCredit();
    }
    public void Reject()
    {
        mda_object.Reject();
    }
    public void Cancel()
    {
        mda_object.Cancel();
    }

    public void Approved()
    {
        mda_object.Approved();

    }

    public void Super()
{
       mda_object.SelectGas(2);
    }
```

```
    public void Regular()
    {
            mda_object.SelectGas(1);
        }

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

        public void PumpGallon()
        {
                mda_object.Pump();
        }
        public void StopPump()
        {
                mda_object.StopPump();
                mda_object.Receipt();

        }

}
```

9.

```
package inputpro;

import abstractfactory.af;
import datastore.*;
import mdaefsm.mda;
public class GP2
{
        af abfact_object;
        mda mda_object;
        datastore datastore_object;
        public void setMDA(mda x)
        {
                mda_object = x;
        }
        public void setdata(datastore x)
        {
                datastore_object = x;
        }
        public void setfactory(af x)
```

```
	{
		abfact_object = x;
	}
public void Activate(int a,int b,int c)
{
	if(((a>0)&&(b>0))&&(c>0))
	{
    datastore_object.setx(a);
    datastore_object.sety(b);
    datastore_object.setz(c);
    mda_object.Activate();

	}
}
public void Start()
{
   mda_object.Start();
}
public void PayCash(float c)
	{

		if(c>0)
		{
			datastore_object.setcash(c);
			mda_object.PayCash();
		}

	}

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

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

```
    }
     public void StartPump()
    {
      mda_object.StartPump();
    }
     public void PumpLiter()
     {

          float cash = datastore_object.getcash();
          float price =datastore_object.getprice();
          float L = datastore_object.getL();
          if(cash<(L+1)*price)
          {         System.out.println("Cannot Pump More");
                    mda_object.StopPump();
          }
          else
          {
                    mda_object.Pump();
          }
     }


    public void StopPump()
    {
      mda_object.StopPump();
    }
    public void Receipt()
    {
      mda_object.Receipt();
    }
    public void NoReceipt()
    {
      mda_object.NoReceipt();
    }

}
```

10

```
package mdaefsm;

import state.*;
public class mda
{
    s sid; //sid is used to point to the current state object
```

```
s[] list = new s[8];
public void setStates(s a)
{
sid = a;
}
public void setStatesList( s t1, s t2, s t3, s t4, s t5, s t6,s t7,s t8)//setting the states
{
      list[0] = t1; //start state
      list[1] = t2; //S0 state
      list[2] = t3; //S1 state
      list[3] = t4; //S2 state
      list[4] = t5; //S3  state
      list[5] = t6; //S4 state
      list[6] = t7; //S5 state
      list[7] = t8; //S6 state
}
public void Activate()
{
      int cur = sid.getStateId();
      System.out.print(cur);
      switch(cur)
      {
      case 0:
      {
        sid.Activate();
        sid = list[1];
        break;
      }
      case 1: break;
      case 2: break;
      case 3: break;
      case 4: break;
      case 5: break;
      case 6: break;
      case 7: break;
      }
}
public void Start()
{
      int cur = sid.getStateId();
      switch(cur)
      {
   case 0:break;
   case 1:
   {
```

```
      sid.Start();
       sid = list[2];
       break;
   }
   case 2:  break;
   case 3:  break;
   case 4:  break;
   case 5:  break;
   case 6:  break;
   case 7:  break;

     }
}

public void PayCash()
{
int cur = sid.getStateId();
switch(cur)
{
   case 0:  break;
   case 1:  break;
   case 2:
   {
     sid.PayCash();
      sid = list[4];
    }
   case 3:  break;
   case 4:  break;
   case 5:  break;
   case 6:  break;
   case 7: break;
     }
}
public void PayCredit()
{
int cur = sid.getStateId();
switch(cur)
{
   case 0: break;
   case 1:  break;
   case 2:
   {
      sid.PayCredit();
      sid = list[3];
   }
```

```
        case 3:  break;
        case 4:  break;
        case 5:  break;
        case 6:  break;
        case 7:  break;

    }
}


    public void Approved()
    {
    int cur = sid.getStateId();
    switch(cur)
    {
        case 0:  break;
        case 1:  break;
        case 2:  break;
        case 3:
        {
          sid.Approved();
          sid = list[4];
          break;
        }
        case 4: break;
        case 5: break;
        case 6: break;
        case 7: break;

    }
}

public void Reject()
{
    int cur = sid.getStateId();
    switch(cur)
    {
        case 0:  break;
        case 1:  break;
        case 2:  break;
        case 3:
        {
           sid.Reject();
          sid = list[1];
          break;
```

```
        }
        case 4:break;
        case 5:break;
        case 6:break;
        case 7:break;
    }
}

public void SelectGas(int g)
{
    int cur = sid.getStateId();
    switch(cur)
    {
        case 0: break;
        case 1: break;
        case 2: break;
        case 3:  break;
        case 4:
        {
           sid.SelectGas(g);
            sid = list[5];
            break;
        }
        case 5: break;
        case 6: break;
        case 7: break;
    }
}

          public void Cancel()
          {
    int cur = sid.getStateId();
    switch(cur)
    {
        case 0:  break;
        case 1:  break;
        case 2:  break;
        case 3:  break;
        case 4:
        {
           sid.Cancel();
           sid = list[1];
           break;
        }
        case 5:  break;
```

```
        case 6:  break;
        case 7:  break;
    }
}

        public void StartPump()
        {
    int cur = sid.getStateId();
    switch(cur)
    {
        case 0:  break;
        case 1:  break;
        case 2:  break;
        case 3:  break;
        case 4:  break;
        case 5:
        {
          sid.StartPump();
           sid = list[6];
           break;
        }
        case 6:  break;
        case 7:  break;
    }
}

        public void Pump()
        {
    int cur = sid.getStateId();
    switch(cur)
    {
        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");
           sid.Pump();
           sid = list[6];
           break;
       }
        case 7:  break;
```

```
    }
}

        public void StopPump()
        {
    int cur = sid.getStateId();
    switch(cur)
    {
       case 0: break;
       case 1: break;
       case 2: break;
       case 3: break;
       case 4:  break;
       case 5:  break;
       case 6:
       {
          sid.StopPump();
          sid = list[7];
          break;
       }
       case 7:  break;
         }
}
        public void Receipt()
        {
    int cur = sid.getStateId();
    switch(cur)
    {
       case 0:  break;
       case 1:  break;
       case 2: break;
       case 3:  break;
       case 4:  break;
       case 5:  break;
       case 6: break;
       case 7:
       {
          sid.Receipt();
          sid = list[1];
          break;
      }

    }
}
```

```
        public void NoReceipt()
        {
   int cur = sid.getStateId();
   switch(cur)
   {
      case 0: break;
      case 1: break;
      case 2: break;
      case 3: break;

      case 4: break;
      case 5: break;
      case 6: break;
      case 7:
        {
         sid.NoReceipt();
         sid = list[1];
         break;
        }

        }
        }

}
```

11.

```
package outputpro;
import datastore.*;
import abstractfactory.*;
import outputpro.OutputProcessor;
import strategy.*;


public class OutputProcessor
{
                static af abfact_object;
            datastore datastore_object;
            public void setfactory(af af1)
             {
                abfact_object=af1;
             }
             public void setData(datastore d1)
             {
```

```
    datastore_object=d1;
}
public static void storeData()
{

        StoreData object;
        object=abfact_object.getStoreData();
        object.storeData();
}
public static void displayMenu()
{
      DisplayMenu object;
      object=abfact_object.getDisplayMenu();
      object.displayMenu();
}
public static void payMsg()
{
      PayMsg object;
      object=abfact_object.getPayMsg();
      object.payMsg();
}
public static void PayCredit()
{
      System.out.println("Credit Card Authentication");

}
public static void rejectMsg()
{
      RejectMsg object;
      object=abfact_object.getRejectMsg();
      object.rejectMsg();
}
public static void printReceipt()
{
      PrintReceipt object;
      object=abfact_object.getPrintReceipt();
      object.printReceipt();
 }
public static void cancelMsg()
{
      CancelMsg object;
      object=abfact_object.getCancelMsg();
      object.cancelMsg();
}
public static void storeCash()
```

```
{
      StoreCash object;
      object=abfact_object.getStoreCash();
      object.storeCash();

}
public static void gasPumpedMsg()
{
      GasPumpedMsg object;
      object=abfact_object.getGasPumpedMsg();
      object.gasPumpedMsg();
}
 public static void pumpGasUnit()
{
         PumpGasUnit object;
              object=abfact_object.getPumpGasUnit();
              object.pumpGasUnit();
}

public static void readyMsg()
{
      ReadyMsg object;
      object=abfact_object.getReadyMsg();
      object.readyMsg();
}
public static void stopMsg()
{
      StopMsg object;
      object=abfact_object.getStopMsg();
      object.stopMsg();
}

public static void setPrice(int g)
{
      SetPrice object;
      object=abfact_object.getSetPrice();
      object.setPrice(g);
}
public static void setInitialValues()
{

      SetInitialValues object;
      object=abfact_object.getSetInitialValues();
      object.setInitialValues();
}
```

```
        public static void NoReceipt()
        {

            NoReceipt object;
            object=abfact_object.getNoReceipt();
            object.noReceipt();
        }
        public static void ReturnCash()
        {

            ReturnCash object;
            object=abfact_object.getReturnCash();
            object.ReturnCashs();
        }
```

12.

```
package state;

import outputpro.OutputProcessor;
public  abstract class s
{
        OutputProcessor outp_object;//object of the output processor
        int sid;
   public int getStateId()
   {
      return sid;
   }
   public void set_OutputProcessor(OutputProcessor o)
   {
      outp_object = o;
   }
   public void setStateId(int a)
   {
      sid = a;
   }
   public void Activate(){}
   public void Start(){}
   public void PayCash(){}
   public void PayCredit(){}
   public void Approved(){}
   public void Reject(){}
```

```
    public void SelectGas(int g){}
    public void Cancel(){}
    public void StartPump(){}
    public void Pump(){}
    public void StopPump(){}
    public void Receipt(){}
    public void NoReceipt(){}

}
```

13.

```
package state;

import outputpro.OutputProcessor;
public class s0 extends s
{
        @Override
    public void Activate()
    {
        System.out.println("\n The Gas pump is  Activated ");
        OutputProcessor.storeData();
    }
}
```

14.

```
package state;

import outputpro.OutputProcessor;
public class s1 extends s
{
        @Override
    public void Start()
    {
     OutputProcessor.payMsg();
    }
}
```

15.

```
package state;

import outputpro.OutputProcessor;
public class s2 extends s
{
        @Override
    public void PayCash()
```

```
    {
     OutputProcessor.storeCash();
     OutputProcessor.displayMenu();
     }
          public void PayCredit()
    {

     OutputProcessor.PayCredit();

     }
}
```

16.

```
package state;
import outputpro.OutputProcessor;
public class s3 extends s
{
          @Override
    public void Approved()
    {
      OutputProcessor.displayMenu();

    }
    public void Reject()
    {
     OutputProcessor.rejectMsg();

    }
}
```

17.

```
package state;

import outputpro.OutputProcessor;

public class s4 extends s
{
          @Override
          public void Cancel()
    {
      OutputProcessor.cancelMsg();
      OutputProcessor.ReturnCash();

    }
```

```
        public void SelectGas(int g)
    {
      OutputProcessor.setPrice(g);

    }

}
```

18.

```
package state;

import outputpro.OutputProcessor;

public class s5 extends s
{
        @Override
    public void StartPump()
    {
      OutputProcessor.setInitialValues();
      OutputProcessor.readyMsg();

    }
}
```

19

```
package state;

import outputpro.OutputProcessor;
public class s6 extends s
{
        @Override
    public void Pump()
    {
                OutputProcessor.pumpGasUnit();
                OutputProcessor.gasPumpedMsg();

    }
    public void StopPump()
    {

        OutputProcessor.stopMsg();
    }
}
```

20

```
package state;

import outputpro.OutputProcessor;
public class s7 extends s
{
        @Override
    public void Receipt()
    {
     OutputProcessor.printReceipt();


    }
        public void NoReceipt()
    {
     OutputProcessor.NoReceipt();
     OutputProcessor.ReturnCash();

    }
}
```

21

```
package strategy;

import datastore.*;
public abstract class CancelMsg
{
    datastore dobj;
    public abstract void cancelMsg();
    public void setdata(datastore dt)
    {
        dobj=dt;
    }
}
```

22

```
package strategy;

public class CancelMsg1 extends CancelMsg
{
    public void cancelMsg()
    {
            System.out.println("\n The operation has been cancelled \n");
    }
```

```
}
```

23

```
package strategy;

public class CancelMsg2 extends CancelMsg
{
   public void cancelMsg()
   {
         System.out.println("\n The operation has been cancelled \n");
   }

}
```

24

```
package strategy;

import datastore.*;
public abstract class DisplayMenu
{
  datastore dobj;
  public abstract void displayMenu();
  public void setdata(datastore dt)
    {
       dobj=dt;
    }
}
```

25

```
package strategy;

public class DisplayMenu1 extends DisplayMenu
{
   public void displayMenu()
   {
   System.out.println("\n ----------DISPLAY MENU----------");
         System.out.println("\n Select the type of gas");
         System.out.println("\n Super");
         System.out.println("\n Regular");
   }
}
```

26

```
package strategy;

public class DisplayMenu2 extends DisplayMenu
{
   public void displayMenu()
   {
   System.out.println("\n ***********DISPLAY MENU*************");
         System.out.println("\n Select option 4 for Super Fuel ");
         System.out.println("\n Select option 5 to Regular Fuel");
         System.out.println("\n Select option 6 to Premium Fuel");

   }

}
```

27

```
package strategy;

import datastore.*;
public abstract class GasPumpedMsg{
  datastore dobj;
   public abstract void gasPumpedMsg();

   public void setdata(datastore dt)
   {
        dobj=dt;
   }
}
```

28

```
package strategy;

public class GasPumpedMsg1 extends GasPumpedMsg
{
   public void gasPumpedMsg()
   {
        float g = dobj.getG();
        System.out.printf("Number of gallons of gas pumped=" +g);
   }

}
```

29

```
package strategy;

public class GasPumpedMsg2 extends GasPumpedMsg
{
   public void gasPumpedMsg()
   {
        float l = dobj.getL();
        System.out.printf("The Gas pump has sucessfully pumped L  units:"+l);

   }

}
```

30

```
package strategy;

import datastore.*;
public abstract class NoReceipt
{
        datastore dobj;
   public abstract void noReceipt();

   public void setdata(datastore dt)
   {
        dobj=dt;
   }
}
```

31

```
package strategy;

public class NoReceipt1 extends NoReceipt {
        public void noReceipt()
   {

   }
}
```

32

```
package strategy;

public class NoReceipt2 extends NoReceipt
```

```
{
   public void noReceipt()
   {

   }
}
```

33

```
package strategy;

import datastore.*;
public abstract class PayMsg
{
   datastore dobj;
   public abstract void payMsg();

   public void setdata(datastore dt)
   {
        dobj=dt;
   }
}
```

34

```
package strategy;

public class PayMsg1 extends PayMsg
{
   public void payMsg()
   {

         System.out.println("Pay By Credit:");

   }

}
```

35

```
package strategy;

public class PayMsg2 extends PayMsg
{
   public void payMsg()
   {
```

```
        System.out.println("Pay By Cash:");

    }

}
```

36

```
package strategy;

import datastore.*;
public abstract class PrintReceipt {
    datastore dobj;
    public abstract void printReceipt();

    public void setdata(datastore dt)
    {
        dobj=dt;
    }
}
```

37

```
package strategy;

public class PrintReceipt1 extends PrintReceipt
{
        @Override
    public void printReceipt()
    {
        float total = dobj.gettotal();
        float G=dobj.getG();
        System.out.printf("\n Receipt");
        System.out.printf("\nTotal Amount paid"+total);
        System.out.printf("\nGallons of gas pumped"+G);

    }

}
```

38

```
package strategy;

public class PrintReceipt2 extends PrintReceipt
{
```

```
        @Override
    public void printReceipt()
        {
        float total = dobj.gettotal();
        int L=(int)dobj.getL();
        float cash1=dobj.getcash();
        float cash_returned=cash1-total;

        System.out.printf("\n Receipt");
        System.out.printf("\nTotal Amount paid"+total);
        System.out.printf("\nGallons of gas pumped"+L);
        System.out.printf("\nCash Returned"+cash_returned);

    }

}
```

39

```
package strategy;

import datastore.*;
public abstract class PumpGasUnit
{
   datastore dobj;
    public abstract void pumpGasUnit();
    public void setdata(datastore dt)
    {
       dobj=dt;
    }
}
```

40

```
package strategy;

public class PumpGasUnit1 extends PumpGasUnit
{
        @Override
    public void pumpGasUnit()
    {

      float g=dobj.getG();
      g=g+1;
      float total;
      float price = dobj.getprice();
      total =price *g;
```

```
      //System.out.println("Abhinav"+price);
      dobj.setG(g);
      dobj.settotal(total);
     }
}
```

41

```
package strategy;

public class PumpGasUnit2 extends PumpGasUnit
{
        @Override
   public void pumpGasUnit()
   {
     int l=dobj.getL();
     l=l+1;
     float total;
     float price = dobj.getprice();
     total =price*l;
     dobj.setL(l);
     dobj.settotal(total);

    }

}
```

42

```
package strategy;

import datastore.*;
public abstract class ReadyMsg
{
   datastore dobj;
   public abstract void readyMsg();
   public void setdata(datastore dt)
   {
      dobj=dt;
   }
}
```

43

```
package strategy;
```

```
public class ReadyMsg1 extends ReadyMsg
{
   public void readyMsg()
   {

        System.out.println(" \n The Gas Pump is ready to pump");
   }

}
```

44

```
package strategy;

public class ReadyMsg2 extends ReadyMsg
{
   public void readyMsg()
   {

        System.out.println(" \n The Gas Pump is ready to pump");
   }

}
```

45

```
package strategy;

import datastore.*;
public abstract class RejectMsg
{
   datastore dobj;
   public abstract void rejectMsg();
   public void setdata(datastore dt)
   {
      dobj=dt;
   }
}
```

46

```
package strategy;

public class RejectMsg1 extends RejectMsg
{
   public void rejectMsg()
   {
         System.out.println("Rejected,Please choose option Start again");
```

```
    }

}
```

47

```
package strategy;

import datastore.*;
public abstract class ReturnCash {
    datastore dobj;

    public abstract void ReturnCashs();

    public void setdata(datastore dt)
    {
        dobj=dt;
    }
}
```

48

```
package strategy;

public class ReturnCash1 extends ReturnCash {
        public void ReturnCashs()
        {
        }

}
```

49

```
package strategy;

public class ReturnCash2 extends ReturnCash {

        @Override
    public void ReturnCashs()
    {

        float cash1=dobj.getcash();
        float total=dobj.gettotal();
        float cash_ret=cash1-total;
```

```
        System.out.printf("Cash Returned"+cash_ret);


    }

}
```

50

```
package strategy;

import datastore.*;
public abstract class SetInitialValues
{
        datastore dobj;
    public abstract void setInitialValues();
    public void setdata(datastore dt)
    {
        dobj=dt;
    }
}
```

51

```
package strategy;

public class SetInitialValues1 extends SetInitialValues
{
        public void setInitialValues()
    {

        dobj.setG(0);
        dobj.settotal(0);
    }

}
```

52

```
package strategy;

public class SetInitialValues2 extends SetInitialValues
{
        public void setInitialValues()
    {
```

```
        dobj.setG(0);
        dobj.settotal(0);
    }

}
```

53

```
package strategy;

import datastore.*;
public abstract class SetPrice
{
   datastore dobj;
    public abstract void setPrice(int g);
    public void setdata(datastore dt)
    {
      dobj=dt;
    }
}
```

54

```
package strategy;

public class SetPrice1 extends SetPrice
{
          @Override
    public void setPrice(int g)
    {
      float a=dobj.getreg_price();
      float b=dobj.getsup_price();
      if( g== 1)
      {

            dobj.setprice(a);

      }
      else if (g == 2)
      dobj.setprice(b);
    }

}
```

55

```
package strategy;

public class SetPrice2 extends SetPrice
{
        @Override
   public void setPrice(int g)
   {
                int a=(int)(dobj.getreg_price());
      int b=dobj.getpre_price();
      int c=(int)(dobj.getsup_price());
      if( g== 1)
      {

        dobj.setprice((float)a);

      }else if (g == 2)
      dobj.setprice(b);
      else if(g==3)
      dobj.setprice(c);
   }

}
```

56

```
package strategy;

import datastore.*;
public abstract class StopMsg
{
   datastore dobj;
   public abstract void stopMsg();
   public void setdata(datastore dt)
   {
      dobj=dt;
   }
}
```

57

```
package strategy;

public class StopMsg1 extends StopMsg
{
```

```
   public void stopMsg()
   {

         System.out.println("The Gas Pump has stopped Pumping");
         //System.out.println("Do you want a receipt");
         //System.out.println("\n Choose to print the Receipt ");

   }

}
```

58

```
package strategy;

public class StopMsg2 extends StopMsg
{
   public void stopMsg()
   {

         System.out.println("The Gas Pump has stopped Pumping");
        System.out.println("Do you want a receipt");
         //System.out.println("\n Choose to print the Receipt ");

   }

}
```

59

```
package strategy;

import datastore.*;
public abstract class StoreCash
{
   datastore dobj;
   public abstract void storeCash();
   public void setdata(datastore dt)
   {
      dobj=dt;
   }
}
```

60

```
package strategy;
```

```
public class StoreCash1 extends StoreCash
{
   public void storeCash()
   {
     // float c =dobj.getc();
      //float d =(float)c;
     // dobj.setcash(d);
   }

}
```

61

```
package strategy;

public class StoreCash2 extends StoreCash
{
   public void storeCash()
   {
         float c =dobj.getcash();
      float d =c;
      dobj.setcash(d);
   }

}
```

62

```
package strategy;

import datastore.*;
public abstract class StoreData
{
   datastore dobj;
   public abstract void storeData();
   public void setdata(datastore dt)
   {
      dobj=dt;
   }
}
```

63

```
package strategy;

public class StoreData1 extends StoreData
{
   @Override
   public void storeData()
   {
         float a, b;
      a=dobj.geta();
      dobj.setreg_price(a);
      b=dobj.getb();
      dobj.setsup_price(b);

   }

}
```



```
package strategy;

public class StoreData2 extends StoreData
{
         @Override
   public void storeData()
   {
     int a, b, c;
     a=dobj.getx();

     dobj.setreg_price(a);
     b=dobj.gety();
     dobj.setpre_price(b);
     c=dobj.getz();
     dobj.setsup_price(c);
   }

}
```