CS 586 – Software System Architecture

Final Project Report

Anushka Dhar

adhar2@hawk.iit.edu

A20322969

This project is based on the coursework of CS586 Software System Architecture.
The Project includes various patterns used in software design and implements them.

Overview of Project:
It includes Gas-Pump model with design of 3 Gas Pumps each has some differences based on a bit functional and the arguments it accepts.

There are three Gas Pump components: GasPump-1, GasPump-2 and GasPump-3.

The GasPump-1 component supports the following operations:

```
Activate (int a)  // the gas pump is activated where a is the price of the gas per gallon
Start()           //start the transaction
PayCredit()       // pay for gas by a credit card
Reject()          // credit card is rejected
Cancel()          // cancel the transaction
Approved()        // credit card is approved
PayCash(int c)    // pay for gas by cash, where c represents prepaid cash
StartPump()       // start pumping gas
PumpGallon()      // one gallon of gas is disposed
StopPump()        // stop pumping gas
```

The GasPump-2 component supports the following operations:

```
Activate (float a, float b)   // the gas pump is activated where a is the price of the regular gas
                              // and b is the price of super gas per gallon
Start()                       //start the transaction
PayCredit()                   // pay for gas by a credit card
Reject()                      // credit card is rejected
Cancel()                      // cancel the transaction
Approved()                    // credit card is approved
Super()                       // Super gas is selected
Regular()                     // Regular gas is selected
StartPump()                   // start pumping gas
PumpGallon()                  // one gallon of gas is disposed
StopPump()                    // stop pumping gas
```

The GasPump-3 component supports the following operations:

Activate (float a, float b)	// the gas pump is activated where a is the price of regular gas // and b is the price of premium gas per liter
Start()	//start the transaction
PayCash(float c)	// pay for gas by cash, where c represents prepaid cash
Cancel()	// cancel the transaction
Premium()	// Premium gas is selected
Regular()	// Regular gas is selected
StartPump()	// start pumping gas
PumpLiter()	// one liter of gas is disposed
StopPump()	// stop pumping gas
Receipt()	// Receipt is requested
NoReceipt()	// No receipt

All three GasPump components are state-based components and are used to control a simple gas pumps. Users can pay by cash or a credit card. The gas pump may dispose different types of Gasoline. The price of the gasoline is provided when the gas pump is activated. The detailed behavior of GasPump components is specified using EFSM.

The several differences between GasPump components.
Aspects that vary between three 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.

This project implements three GasPumps using Model Driven Architecture. The GasPumps differ in their user interface specifications, but are similar in behavior. The main goal is to separate the Platform specific portion of the system from the Platform independent portion.
The general outline of the implementation is to use Model-Driven-Architecture as the overall design, and use an Extended-Finite-State-Machine to capture the states in which the model can be in at any point of time. The project implements the Output Processor using Strategy pattern, and uses Abstract Factory Pattern to select and initialize a list of output actions for each GasPumps.

GENERAL ARCHITECTURE:

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

Pseudo Code

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();
}
```

Operations of the Input Processor
(GasPump-2)

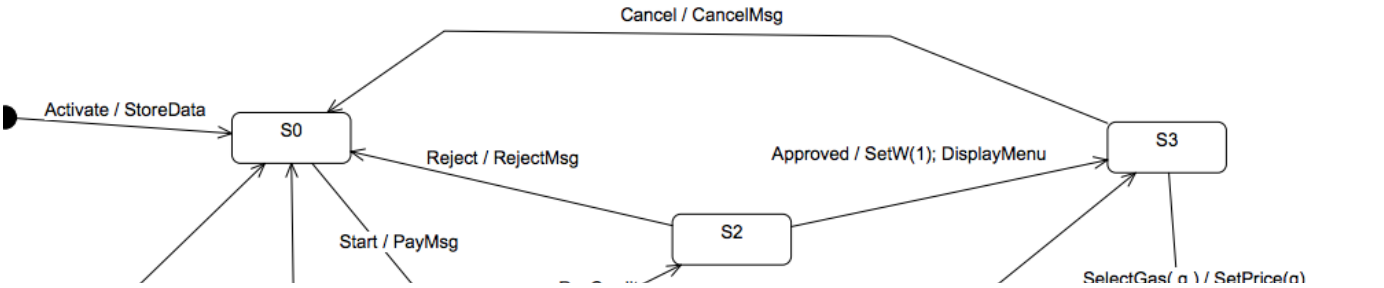
```
Activate(float a, float b) {
 if ((a>0)&&(b>0)) {
 d->temp_a=a;
 d->temp_b=b;
 m->Activate()
 }
}
Start() {
 m->Start();
}
PayCredit() {
 m->PayCredit();
}
Reject() {
 m->Reject();
}
Cancel() {
 m->Cancel();
}
Approved() {
 m->Approved();
}
Super() {
 m->SelectGas(2)
}

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

Operations of the Input Processor
(GasPump-3)

```
Activate(float a, float b) {
 if ((a>0)&&(b>0)) {
 d->temp_a=a;
```

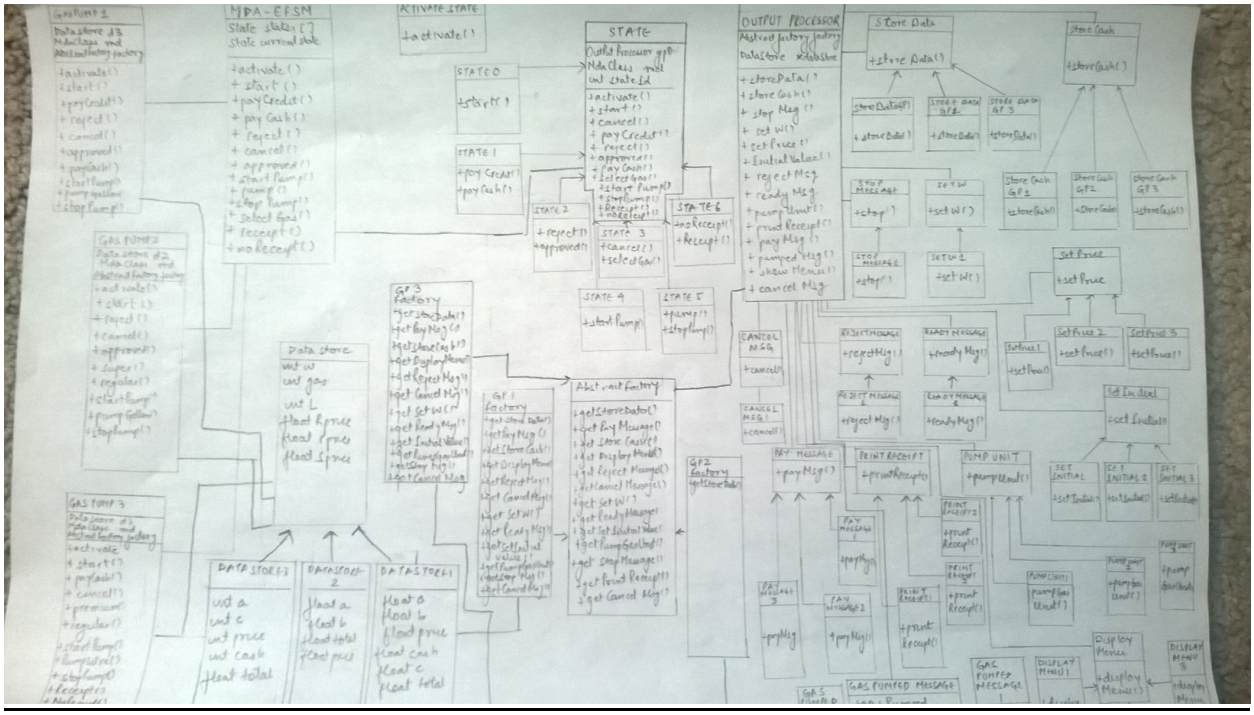
```
 d->temp_b=b;
 m->Activate()
 }
}
Start() {
 m->Start();
}
PayCash(float c) {
 if (c>0) {
 d->temp_c=c;
 m->PayCash()
 }
}
Cancel() {
 m->Cancel();
}
Premium() {
 m->SelectGas(2);
}
Regular() {
 m->SelectGas(1);
}
StartPump() {
 m->StartPump();
}

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

STATE DIAGRAM



MDA_EFSM

CLASS DIAGRAM



Code Documented

Driver Class

```
// Contains the main method and calls all the functions
import java.util.Scanner;
public class GasDriverMain {
	public static void main(String []args)
	{	GasDriverMain gasDriver = new GasDriverMain();
			gasDriver.displayOptions();
	}
	public void displayOptions()
			{
			Scanner input = new Scanner(System.in);
			System.out.println("Welcome to the Gas Station");
			System.out.println("1. Gas-Pump -1");
			System.out.println("2. Gas-Pump -2");
			System.out.println("3. Gas-Pump -3");
			System.out.println("4. Exit");

			System.out.println("\n");

			System.out.println("Select option(1-3): ");

			int   option = input.nextInt();

		switch(option)
			{
				case 1 :

			{

			DataStoreGP1 ds1 = new DataStoreGP1(); //creating object of datastore1
			GasPump1Factory factory = new GasPump1Factory(); //creating object of
factory1
```

```
                GPOutputProcessor op = new
GPOutputProcessor(factory,factory.GetDataStore()); // creating
                                        object of gas processor
                MdaClass md = new MdaClass(factory,ds1); // creating object of mda class
                GasPump1 gp1 = new GasPump1(md,ds1,factory); // creating object of gas
pump1 class

            System.out.println("Gas Pump Execution 1");

            String ch ="1";
            while(ch!= "q")
            {
                System.out.println("Select option");

                System.out.println("0- Activate ,1-Start ,2-PayCredit,3-Reject,4-Cancel, 5-
Approved,6-PayCash,7-StartPump,8-PumpGallon,9-StopPump,q-quit");

                ch = input.next();

                if(ch.equalsIgnoreCase("0"))
                {
              System.out.println("Operation Activate (int a) ");
              System.out.println("Enter the a ");
              int a = input.nextInt();

              gp1.activate(a);
                }
                else if(ch.equalsIgnoreCase("1"))
                {
              gp1.start();
                }
                else if(ch.equalsIgnoreCase("2"))
                {
                        gp1.payCredit();
                }
                else if(ch.equalsIgnoreCase("3"))
                {

              gp1.reject();
                }
                else if(ch.equalsIgnoreCase("4"))
                {

                    gp1.cancel();
                }
                else if(ch.equalsIgnoreCase("5"))
                {
```

```
                        gp1.approved();
                    }
                    else if(ch.equalsIgnoreCase("6"))
                    {
                    System.out.println("Enter the value of c ");
                    int c = input.nextInt();

                    gp1.payCash(c);
                    }
                    else if(ch.equalsIgnoreCase("7"))
                    {

                    gp1.startPump();
                    }
                    else if(ch.equalsIgnoreCase("8"))
                    {

                    gp1.pumpGallon();
                    }
                    else if(ch.equalsIgnoreCase("9"))
                    {

                    gp1.stopPump();
                    }

                    }
                break;
            }
            case 2 :

                    {
                            DataStoreGP2 ds2 = new DataStoreGP2();  // creating object of
datastore 2
                            GasPump2Factory factory = new GasPump2Factory(); //creating object
for facoty 2
                            GPOutputProcessor op = new
GPOutputProcessor(factory,factory.GetDataStore()); //creating object for output processor
                            MdaClass md = new MdaClass(factory,ds2);   // creating mda class
object
                            GasPump2 gp2 = new GasPump2(md,ds2,factory); //creating gp 2
object

                System.out.println("Gas Pump Execution 2");

                String ch ="1";
                while(ch!= "q")
                {
                        System.out.println("Select option");
```

```
                System.out.println("0- Activate,1-Start,2-PayCredit,3-Reject,4-Cancel, 5-
Approved,6-Super,7-Regular,8-StartPump,9-PumpGallon,10-StopPump,q-quit");

                ch = input.next();

                if(ch.equalsIgnoreCase("0"))
                {

            System.out.println("Enter the value of a \n");
            float a = input.nextFloat();
            System.out.println("Enter the value of b \n");

            float b = input.nextFloat();
            gp2.activate(a, b);
                }
                else if(ch.equalsIgnoreCase("1"))
                {

              gp2.start();
                }
                else if(ch.equalsIgnoreCase("2"))
                {

              gp2.payCredit();
                }
                else if(ch.equalsIgnoreCase("3"))
                {

              gp2.reject();
                }
                else if(ch.equalsIgnoreCase("5"))
                {

                        gp2.approved();
                }
                else if(ch.equalsIgnoreCase("6"))
                {

              gp2.Super();
                }
                else if(ch.equalsIgnoreCase("7"))
                {

              gp2.regular();
                }
                else if(ch.equalsIgnoreCase("8"))
                {
```

```
                    gp2.startPump();
                      }
                      else if(ch.equalsIgnoreCase("9"))
                      {

                    gp2.pumpGallon();
                      }
                      else if(ch.equalsIgnoreCase("10"))
                      {

                    gp2.stopPump();
                      }


                  }
                  break;
                }
                case 3 :
                {
                        DataStoreGP3 ds3 = new DataStoreGP3();
                        GasPump3Factory factory = new GasPump3Factory();
                        GPOutputProcessor op = new
GPOutputProcessor(factory,factory.GetDataStore());
                        MdaClass md = new MdaClass(factory,ds3);
                        GasPump3 gp3 = new GasPump3(md,ds3,factory);

                  System.out.println("Gas Pump Execution 3");

                  String ch ="1";
                  while(ch!= "q")
                  {
                          System.out.println("Select option");

                          System.out.println("0- Activate,1-Start,2-PayCash,3-Cancel,4-Premium, 5-
Regular,6-StartPump,7-PumpLitre,8-StopPump,9-Receipt,10-NoReceipt,q-quit");

                          ch = input.next();

                          if(ch.equalsIgnoreCase("0"))
                          {

                                  System.out.println("Enter the value of a \n");

                                  float a = input.nextFloat();

                                  System.out.println("Enter the value of b \n");
```

```
            float b = input.nextFloat();

            gp3.activate(a, b);
}
else if(ch.equalsIgnoreCase("1"))
{

            gp3.start();
}
else if(ch.equalsIgnoreCase("2"))
{

            System.out.println("Enter the value of c \n");

            int c = input.nextInt();

            gp3.payCash(c);
}
else if(ch.equalsIgnoreCase("3"))
{


            gp3.cancel();
}
else if(ch.equalsIgnoreCase("4"))
{
            gp3.premium();
}
else if(ch.equalsIgnoreCase("5"))
{

            gp3.regular();
}
else if(ch.equalsIgnoreCase("6"))
{

            gp3.startPump();
}
else if(ch.equalsIgnoreCase("7"))
{

            gp3.pumpLitre();
}
else if(ch.equalsIgnoreCase("8"))
{

            gp3.stopPump();
}
```

```
                    else if(ch.equalsIgnoreCase("9"))
                    {


                            gp3.Receipt();

                    }
                    else if(ch.equalsIgnoreCase("10"))
                    {

                            System.out.println("NoReceipt");

                            gp3.noReceipt();

                    }

                }

                break;
            }
            case 4 :
            {
                    System.out.println("Exiting the system Thank You");
                    break;
            }

        default :
            {
                    System.out.println("Invalid Input");
                    break;
            }
        }
}
}
```

GASPUMP1

```
// It's the input processor of Gas Pump1

public class GasPump1 {

        private DataStore ds1;
        private MdaClass md;
        private AbstractFactory factory;
        public GasPump1(MdaClass md, DataStore ds1 ,AbstractFactory factory)
        {
                this.ds1 = ds1;
```

```
        this.md = md;
   this.factory = factory;

}


public void activate(int a)
{
        if(a > 0)
        {
                System.out.println("Setting the temporary  A value  IN DATA STORE\n");
                ((DataStoreGP1) ds1).setA(a);
                md.activate(factory,ds1);
        }
        else
        {
                System.out.println("Incorrect input");
        }
}

public void start()
{
        md.start(factory,ds1);
}

public void payCredit()
{
        md.payCredit(factory,ds1);
}

public void reject()
{
        md.reject(factory,ds1);
}
public void cancel()
{
        md.cancel(factory,ds1);
}
public void approved()
{
        md.approved(factory,ds1);

}

public void payCash(int c)
{
        if(c>0)
        {
```

```
                        ((DataStoreGP1) ds1).setC(c);

                        md.payCash(factory,ds1);
                }

        }
        public void startPump()
        {

                md.selectGas(1,factory,ds1);

                md.startPump(factory,ds1);

        }
        public void pumpGallon()
        {

                if( ds1.getW()==1)
                {
                        md.pump(factory,ds1);
                }
                else if(ds1.getW()==0)
                {
                        if(((DataStoreGP1) ds1).getCash()<(ds1.getGas()+1)*((DataStoreGP1)
ds1).getPrice())
                        {

                                md.pump(factory,ds1);
                                md.receipt(factory,ds1);
                        }

                        else
                        {
                                System.out.println("Calling pump function in mda class");

                                md.stopPump(factory,ds1);
                        }
                }

        }
        public void stopPump()
        {
                md.stopPump(factory,ds1);
                md.receipt(factory,ds1);
        }
}
```

GASPUMP 2

// it's the input processor for gas pump2

```
public class GasPump2 {

	private DataStore ds2;
	private MdaClass md;
	private AbstractFactory factory;

	public GasPump2(MdaClass md, DataStore ds2 ,AbstractFactory factory)
	{
		this.ds2 = ds2;
		this.md = md;
				this.factory = factory;
	}


	public void activate(float a , float b)
	{
		 if((a > 0)&&(b > 0))
		 {

			((DataStoreGP2) ds2).setA(a);


			((DataStoreGP2) ds2).setB(b);


			md.activate(factory,ds2);

		 }
		 else
		 {
			System.out.println("Incorrect input");
		 }
	}

	public void start()
	{
		md.start(factory,ds2);
	}

	public void payCredit()
	{
```

```
            md.payCredit(factory,ds2);
}

public void reject()
{

            md.reject(factory,ds2);
}
public void cancel()
{

            md.cancel(factory,ds2);
}
public void approved()
{

            md.approved(factory,ds2);

}

public void Super()
{

            md.selectGas(2,factory,ds2);
}

public void regular()
{

            md.selectGas(1,factory,ds2);
}

public void startPump()
{

            md.startPump(factory,ds2);

}
public void pumpGallon()
{

            md.pump(factory,ds2);

}
public void stopPump()
{

            md.stopPump(factory,ds2);
```

```
                md.receipt(factory,ds2);
        }
}
```

GASPUMP3

```
public class GasPump3 {

        private DataStore ds3;
        private MdaClass md;
        private AbstractFactory factory;
        public GasPump3(MdaClass md, DataStore ds3, AbstractFactory factory)
        {
                this.ds3 = ds3;
                this.md = md;
                this.factory = factory;
        }

        public void activate(float a,float b)
        {
                 if((a > 0) && (b>0))
                 {

                         ((DataStoreGP3) ds3).setA(a);

                         ((DataStoreGP3) ds3).setB(b);


                         md.activate(factory,ds3);
                 }
                 else
                 {
                         System.out.println("Incorrect input");
                 }
        }

        public void start()
        {

                md.start(factory,ds3);
        }

        public void payCash(int c)
        {
                if(c>0)
                {
```

```
                ((DataStoreGP3) ds3).setC(c);

                md.payCash(factory,ds3);
        }

}

public void cancel()
{

        md.cancel(factory,ds3);
}

public void premium()
{

        md.selectGas(2,factory,ds3);
}

public void regular()
{

        md.selectGas(1,factory,ds3);
}
public void startPump()
{

        md.startPump(factory,ds3);
}
public void pumpLitre()
{
        if(((DataStoreGP3) ds3).getCash() < (ds3.getL()+1)*((DataStoreGP3) ds3).getPrice())
        {

                md.pump(factory,ds3);


        }
        else
        {


                md.stopPump(factory,ds3);

        }

}
```

```
	public void stopPump()
	{

		md.stopPump(factory,ds3);
	}

	public void Receipt()
	{

		md.receipt(factory,ds3);
	}

	public void noReceipt()
	{

		md.noReceipt(factory,ds3);
	}
}
```

// Over here abstract factory pattern is implemented

ABSTRACT FACTORY

// it's the abstract class for the factory

```
public abstract  class AbstractFactory {

				public abstract StoreDataAbstract getStoreData();

				public abstract PayMessageAbstract getPayMessage();

				public abstract StoreCashAbstract getStoreCash();

				public abstract DisplayMenuAbstract getDisplayMenu();

				public abstract RejectMessageAbstract getRejectMessage();

				public abstract SetWAbstract getSetW();

				public abstract SetPriceAbstract getSetPrice();

				public abstract  ReadyMessageAbstract getReadyMessage();

				public abstract SetInitialValueAbstract getSetInitialValue();

				public abstract   PumpGasUnitAbstarct getPumpGasUnit();
```

```
        public abstract GasPumpedMessageAbstract getGasPumpedMessage();

        public abstract StopMessageAbstract getStopMessage();

        public abstract PrintReceiptAbstract getPrintReceipt();

        public abstract CancelMessageAbstract getCancelMessage();

}
```

GAS pumpFACTORY 1

Concrete factory class for Gas pump 1

```
public class GasPump1Factory extends AbstractFactory{

    DataStore dataStore = new DataStoreGP1();


    public DataStore CreateDataStore()
    {
        return dataStore;
    }
    public DataStore GetDataStore()
    {
        return dataStore;
    }

    public StoreDataAbstract getStoreData()
    {
      return  new StoreDataGasPump1();
    }

    public PayMessageAbstract getPayMessage()
    {
        return new PayMessage1();
    }

    public StoreCashAbstract getStoreCash()
    {
        return new StoreCashGasPump1();
    }

    public DisplayMenuAbstract getDisplayMenu()
    {
        return new  DisplayMenuGasPump1();
    }
```

```
        public RejectMessageAbstract getRejectMessage()
        {
                return new RejectMessage1();
        }

        public SetWAbstract getSetW()
        {
                return new SetW1();
        }
        public SetPriceAbstract getSetPrice()
        {
                return new SetPrice1();
        }

        public ReadyMessageAbstract getReadyMessage()
        {
                return new ReadyMessage1();
        }

        public SetInitialValueAbstract getSetInitialValue()
        {
                return new SetInitial1();
        }
public PumpGasUnitAbstarct getPumpGasUnit()
{
          return new PumpGasUnit1();
}

public GasPumpedMessageAbstract getGasPumpedMessage()
{
         return new GasPumpedMessage1();
}

public StopMessageAbstract getStopMessage()
{
          return new StopMessage1();
}

public PrintReceiptAbstract getPrintReceipt()
{
          return new PrintReceipt1();
}
public CancelMessageAbstract getCancelMessage()
{
           return new CancelMessage1();
}
```

}

Gas Pump FACTORY 2

// concerete class for factory 2

```
public class GasPump2Factory extends AbstractFactory{

DataStore dataStore = new DataStoreGP2();


        public DataStore CreateDataStore()
        {
                return dataStore;
        }
        public DataStore GetDataStore()
        {
                return dataStore;
        }

        public  StoreDataAbstract getStoreData()
        {
           return  new StoreDataGasPump2();
        }

        public PayMessageAbstract getPayMessage()
        {
                return new PayMessage2();
        }

        public StoreCashAbstract getStoreCash()
        {
                 return null;
        }

        public DisplayMenuAbstract getDisplayMenu()
        {
                return new  DisplayMenuGasPump2();
        }

        public RejectMessageAbstract getRejectMessage()
        {
                 return new RejectMessage1();
        }

        public SetWAbstract getSetW()
        {
                return new SetW1();
        }
```

```
        public SetPriceAbstract getSetPrice()
        {
                return new SetPrice2();
        }

        public ReadyMessageAbstract getReadyMessage()
        {
                return new ReadyMessage1();
        }

        public SetInitialValueAbstract getSetInitialValue()
        {
                return new SetInitial2();
        }
 public PumpGasUnitAbstarct  getPumpGasUnit()
 {
            return new PumpGasUnit2();
 }

 public GasPumpedMessageAbstract getGasPumpedMessage()
 {
          return new GasPumpedMessage1();
 }

 public StopMessageAbstract getStopMessage()
 {
           return new StopMessage1();
 }

 public PrintReceiptAbstract getPrintReceipt()
 {
           return new PrintReceipt2();
 }
 public CancelMessageAbstract getCancelMessage()
 {
            return new CancelMessage1();
 }

}
```

FACTORY 3

// concrete class for gas pump 3 3

```
public class GasPump3Factory extends AbstractFactory {

        DataStore dataStore = newDataStoreGP3();
```

```
public DataStore CreateDataStore()
{
        return dataStore;
}
public DataStore GetDataStore()
{
        return dataStore;
}

public StoreDataAbstract getStoreData()
{
   return  new StoreDataGasPump3();
}

public PayMessageAbstract getPayMessage()
{
        return new PayMessage3();
}

public StoreCashAbstract getStoreCash()
{
         return new StoreCashGasPump3();
}

public DisplayMenuAbstract getDisplayMenu()
{
        return new  DisplayMenuGasPump3();
}

public RejectMessageAbstract getRejectMessage()
{
         return new RejectMessage1();
}

public SetWAbstract getSetW()
{
        return new SetW1();
}
public SetPriceAbstract getSetPrice()
{
        return new SetPrice3();
}

public ReadyMessageAbstract getReadyMessage()
{
        return new ReadyMessage1();
}
```

```
        public SetInitialValueAbstract getSetInitialValue()
        {
                return new SetInitial3();
        }
 public PumpGasUnitAbstarct getPumpGasUnit()
 {
            return new PumpGasUnit3();
 }

 public GasPumpedMessageAbstract getGasPumpedMessage()
 {
          return new GasPumpedMessage2();
 }

 public StopMessageAbstract getStopMessage()
 {
           return new StopMessage1();
 }

 public PrintReceiptAbstract getPrintReceipt()
 {
           return new PrintReceipt3();
 }
 public CancelMessageAbstract getCancelMessage()
 {
            return new CancelMessage1();
 }



}

// over here state pattern is implemented
```

STATES

```
public abstract class State {


        GPOutputProcessor gpOut ;

        public GPOutputProcessor getGpOut() {
                return gpOut;
        }
        public void setGpOut(GPOutputProcessor gpOut) {
                this.gpOut = gpOut;
```

```
}
public MdaClass md ;

private int stateId;

public int getStateId() {
	return stateId;
}
public void setStateId(int stateId) {
	this.stateId = stateId;
}
public void activate(AbstractFactory factory, DataStore dataStore)
{

}
public void start(AbstractFactory factory, DataStore dataStore)
{

}
public void payCredit(AbstractFactory factory, DataStore dataStore)
{

}
public void reject(AbstractFactory factory, DataStore dataStore)
{

}
public void approved(AbstractFactory factory, DataStore dataStore)
{

}

public void cancel()
{

}
public void payCash(AbstractFactory factory, DataStore dataStore)
{

}
public void selectGas(int g,AbstractFactory factory, DataStore dataStore)
{

}
public void startPump(AbstractFactory factory, DataStore dataStore)
{

}
```

```
	public void pump(AbstractFactory factory, DataStore dataStore)
	{

	}
	public void stopPump(AbstractFactory factory, DataStore dataStore)
	{

	}
	public void receipt(AbstractFactory factory, DataStore dataStore)
	{

	}
	public void noReceipt()
	{

	}
}
```

ACTIVATE STATE

```
public class ActivateState extends State {

	 public ActivateState(AbstractFactory factory, DataStore dataStore)
	   {

		setStateId(0);
		if(gpOut == null)
	    this.gpOut = new  GPOutputProcessor(factory, dataStore);

	 }

		public void activate(AbstractFactory factory, DataStore dataStore)
		{

			gpOut.storeData( dataStore);


		}

	}
```

STATE 0

```
public class State0 extends State {
```

```
        public State0(AbstractFactory factory, DataStore dataStore)
          {

                    setStateId(1);
                    if(gpOut == null)
              this.gpOut = new  GPOutputProcessor(factory, dataStore);

          }


        public void start(AbstractFactory factory, DataStore dataStore)
        {

                System.out.println("In the State 0 \n");

                System.out.println("Now calling the pay  message \n");
                 gpOut.payMsg( );


        }


}
```

STATE 1

```
public class State1 extends State{

        public State1(AbstractFactory factory, DataStore dataStore)
          {

                    setStateId(2);
                    if(gpOut == null)
              this.gpOut = new  GPOutputProcessor(factory, dataStore);

          }

       public void payCredit(AbstractFactory factory, DataStore dataStore)
       {

       }

        public void payCash(AbstractFactory factory, DataStore dataStore)
        {
                gpOut.storeData( dataStore);
                gpOut.setW(0);
```

```
                gpOut.displayMenu( );

        }


}
```

STATE 2

```
public class State2 extends State {


        public State2(AbstractFactory factory, DataStore dataStore)
          {

                setStateId(3);
                if(gpOut == null)
            this.gpOut = new  GPOutputProcessor(factory, dataStore);

        }


        public void reject(AbstractFactory factory, DataStore dataStore)
        {
                gpOut.rejectMsg();

        }

        public void approved(AbstractFactory factory, DataStore dataStore)
        {
                gpOut.setW(1);
                gpOut.displayMenu();

        }
}
```

STATE 3

```
public class State3 extends State{
        public State3(AbstractFactory factory, DataStore dataStore)
          {

                setStateId(4);
                if(gpOut == null)
            this.gpOut = new  GPOutputProcessor(factory, dataStore);
```

```
        }
        public void cancel(AbstractFactory factory, DataStore dataStore)
        {
                gpOut.cancelMsg();

        }

        public void selectGas(int g, AbstractFactory factory, DataStore  dataStore)
        {
                gpOut.setPrice(g);

        }

}
```

STATE 4

```
public class State4 extends State{


public State4(AbstractFactory factory, DataStore dataStore)
          {

                setStateId(5);
                if(gpOut == null)
            this.gpOut = new  GPOutputProcessor(factory, dataStore);

        }

                        public void startPump(AbstractFactory   factory,  DataStore dataStore)
                        {
                                gpOut.setInitialValues( dataStore);
                                gpOut.readyMsg();

                        }
}
```

STATE 5

```
public class State5 extends State {
        public State5(AbstractFactory factory, DataStore dataStore)
          {

                setStateId(6);
                if(gpOut == null)
```

```
            this.gpOut = new  GPOutputProcessor(factory, dataStore);

        }

        public void  pump(AbstractFactory factory, DataStore dataStore)
        {
          gpOut.pumpGasUnit(dataStore);
         gpOut.gasPumpedMsg(dataStore);

        }
   public void stopPump(AbstractFactory factory, DataStore dataStore)
   {
        gpOut.stopMsg(dataStore);

   }
}
```

STATE 6

```
public class State6 extends State{

        public State6(AbstractFactory factory, DataStore dataStore)
          {

                setStateId(7);
                if(gpOut == null)
            this.gpOut = new  GPOutputProcessor(factory, dataStore);

        }

 public void noReceipt(AbstractFactory factory, DataStore dataStore)
 {

 }

 public void receipt(AbstractFactory factory, DataStore dataStore)
 {
         gpOut.printReceipt( dataStore);

 }
}
```

MDA

```
public class MdaClass {
```

```
	private State[] states = new State[8] ;

	private State currentState;


	public MdaClass(AbstractFactory factory, DataStore dataStore)
	{


		states[0] = new ActivateState(factory,dataStore);
		states[1] = new State0(factory,dataStore);
		states[2] = new State1(factory,dataStore);
		states[3] = new State2(factory,dataStore);
		states[4] = new State3(factory,dataStore);
		states[5] = new State4(factory,dataStore);
		states[6] = new State5(factory,dataStore);
		states[7] = new State6(factory,dataStore);

		currentState = states[0];

	}

	public void activate(AbstractFactory factory, DataStore dataStore)
	{

		if(currentState.getStateId() == 0)
		{
			currentState.activate(factory,dataStore);
      currentState = states[1];
		}
	}

public void start(AbstractFactory factory, DataStore dataStore)
{

	if(currentState.getStateId() == 1)
	{
		currentState.start(factory,dataStore);
		currentState = states[2];
	}
}
public void payCredit(AbstractFactory factory, DataStore dataStore)
{

	if(currentState.getStateId() == 2)
	{
		currentState.payCredit(factory,dataStore);
```

```
                currentState = states[3];
        }
}

public void payCash(AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() == 2)
        {
                currentState.payCash(factory,dataStore);
                currentState = states[4];
        }
}
public void reject(AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() ==3 )
        {
                currentState.reject(factory,dataStore);
                currentState = states[1];
        }

}
public void cancel(AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() == 4)
        {
                currentState.cancel();
                currentState = states[1];
        }
}
public void approved(AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() == 3)
        {
                currentState.approved(factory,dataStore);
                currentState = states[4];
        }
}
public void startPump(AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() == 5)
        {
                currentState.startPump(factory,dataStore);
                currentState = states[6];
```

```
        }

}
public void pump(AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() == 6)
        {
                currentState.pump(factory,dataStore);
                currentState = states[6];
        }
}
public void stopPump(AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() == 6)
        {
                currentState.stopPump(factory,dataStore);
                currentState = states[7];
        }
}
public void selectGas(int g,AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() == 4)
        {
                currentState.selectGas(g,factory,dataStore);
                currentState = states[5];
        }
}
public void receipt(AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() == 7)
        {
                currentState.receipt(factory,dataStore);
                currentState = states[1];
        }
}
public void noReceipt(AbstractFactory factory, DataStore dataStore)
{

        if(currentState.getStateId() == 7)
        {
                currentState.noReceipt();
                currentState = states[1];
        }
}
```

```
}


// In this the stratergy pattern is implemented
```

OUTPUT PROCESSOR

```
public class GPOutputProcessor {

        AbstractFactory factory = null;
        DataStore dataStore = null;


         public GPOutputProcessor(AbstractFactory factory,DataStore dataStore)
        {
    this.factory = factory;
    this.dataStore = dataStore;
  }



        private StoreDataAbstract storeData;
        private StoreCashAbstract storeCash;
        private StopMessageAbstract stopMsg;
        private SetWAbstract setW;
        private SetPriceAbstract setPrice;
        private SetInitialValueAbstract initialValue;
        private RejectMessageAbstract rejectMsg;
        private ReadyMessageAbstract readyMsg;
        private PumpGasUnitAbstarct pumpUnit;
        private PrintReceiptAbstract printReceipt;
  private PayMessageAbstract payMsg;
  private GasPumpedMessageAbstract pumpedMsg;
  private DisplayMenuAbstract showMenu;
  private CancelMessageAbstract cancelMsg;



        public void storeData(DataStore dataStore)
        {
                storeData = factory.getStoreData();
                storeData.StoreData(dataStore );
        }

        public void payMsg()
        {
                payMsg = factory.getPayMessage();
```

```
            payMsg.PayMessage();
    }
    public void  storeCash()
    {   storeCash = factory.getStoreCash();
            storeCash.StoreCash(dataStore);
    }
    public void displayMenu()
    {
            showMenu = factory.getDisplayMenu();
            showMenu.DisplayMenu();
    }
    public void rejectMsg()
    {

            rejectMsg = factory.getRejectMessage();
            rejectMsg.RejectMessage();
    }
    public void setW(int w)
    {
            setW = factory.getSetW();
setW.SetW(w, dataStore);
    }

    public void setPrice(int g)
    {
            setPrice = factory.getSetPrice();
            setPrice.SetPrice(g,dataStore);
    }
    public void readyMsg()
    {
            readyMsg = factory.getReadyMessage();
            readyMsg.ReadyMessage();
    }
    public void setInitialValues(DataStore dataStore)
    {
            initialValue = factory.getSetInitialValue();
            initialValue.SetInitial(dataStore);
    }
    public void pumpGasUnit(DataStore dataStore)
    {
            pumpUnit = factory.getPumpGasUnit();
            pumpUnit.PumpGasUnit(dataStore);
    }
    public void gasPumpedMsg(DataStore dataStore)
    {
            pumpedMsg = factory.getGasPumpedMessage();
            pumpedMsg.GasPumpedMessage(dataStore);
    }
```

```
	public void stopMsg(DataStore dataStore)
	{
		stopMsg = factory.getStopMessage();
		stopMsg.StopMessage(dataStore);
	}
	public void printReceipt(DataStore dataStore)
	{
		printReceipt = factory.getPrintReceipt();
		printReceipt.PrintReceipt(dataStore);
	}
	public void cancelMsg()
	{
		cancelMsg = factory.getCancelMessage();
		cancelMsg.CancelMessage();
	}
}
```

STORE DATA ABSTRACT

```
public abstract class StoreDataAbstract {

	public abstract void StoreData(DataStore dataStore);

}
```

STORE DATA GASPUMP1

```
public class StoreDataGasPump1 extends StoreDataAbstract{


	public  void StoreData(DataStore dataStore)
	{

		((DataStoreGP1) dataStore).setPrice(((DataStoreGP1) dataStore).getA());

	}
}
```

STORE DATA GASPUMP2

```
public class StoreDataGasPump2 extends StoreDataAbstract{
```

```
	public  void StoreData(DataStore dataStore)
	{
		((DataStoreGP2) dataStore).setRPrice(((DataStoreGP2) dataStore).getA());

		((DataStoreGP2) dataStore).setSprice(((DataStoreGP2) dataStore).getB());


	}
}
```

STORE DATA GASPUMP3

```
public class StoreDataGasPump3 extends StoreDataAbstract{

	public  void StoreData(DataStore dataStore)
	{

		((DataStoreGP3) dataStore).setRPrice(((DataStoreGP3) dataStore).getA());


		((DataStoreGP3) dataStore).setPprice(((DataStoreGP3) dataStore).getB());


	}
}
```

STORE CASH ABSTRACT

```
public abstract class StoreCashAbstract {


	public abstract void StoreCash(DataStore dataStore);
}
```

STORE CASH GASPUMP1

```
public class StoreCashGasPump1 extends StoreCashAbstract{

	public void StoreCash(DataStore dataStore)
	{
		((DataStoreGP1) dataStore).setCash(((DataStoreGP1) dataStore).getCash());
		System.out.println("Cash value is set");
```

```
        }

}
```

STORE CASH GASPUMP3

```
public class StoreCashGasPump3 extends StoreCashAbstract{

                public void StoreCash(DataStore dataStore)
                {
                        ((DataStoreGP3) dataStore).setCash(((DataStoreGP3) dataStore).getCash());
                        System.out.println("Cash value is set");
                }



}
```

STOP MESSAGE ABSTRACT

```
public abstract class StopMessageAbstract {

        public abstract void StopMessage(DataStore dataStore);

}
```

STOP MESSAGE 1

```
public class StopMessage1 extends StopMessageAbstract{

         public void StopMessage(DataStore dataStore) {


                 System.out.println("GAS PUMP HAS STOPPED WORKING");



        }

}
```

SET W ABSTRACT

```
public abstract  class SetWAbstract {
```

```
	public abstract void SetW(int w, DataStore d);
}
```

SETW1

```
public class SetW1 extends SetWAbstract{

	public void SetW(int w, DataStore d) {
		d.setW(w);

	}



}
```

SET PRICE ABSTRACT

```
public  abstract class SetPriceAbstract {

	 public abstract  void SetPrice(int g,DataStore dataStore);


}
```

SET PRICE 1

```
public class SetPrice1  extends SetPriceAbstract{

	public void SetPrice(int g, DataStore dataStore) {

		System.out.println("THE PRICE OF REGULAR GAS IS SET");
	}

}
```

SET PRICE 2

```
public class SetPrice2  extends SetPriceAbstract{
```

```
	public void SetPrice(int g, DataStore dataStore) {

		System.out.println("Price is set");
		if(g == 1)
		{
			((DataStoreGP2) dataStore).setPrice(((DataStoreGP2) dataStore).getRPrice());
		}
		else if(g == 2)
		{
			((DataStoreGP2) dataStore).setPrice(((DataStoreGP2) dataStore).getSprice());
		}
	}

}
```

SETPRICE 3

```
public class SetPrice3  extends SetPriceAbstract{

	public void SetPrice(int g,DataStore dataStore) {



		if(g == 1)
		{
		   ((DataStoreGP3) dataStore).setPrice(((DataStoreGP3) dataStore).getRPrice());
		}
		else if(g==2)
		{
			((DataStoreGP3) dataStore).setPrice(((DataStoreGP3) dataStore).getPprice());
		}
	}

}
```

SET INITIAL ABSTRACT

```
public abstract  class SetInitialValueAbstract {



	 public abstract void SetInitial(DataStore dataStore);


}
```

SET INITIAL 1

```
public class SetInitial1 extends SetInitialValueAbstract{


        public  void SetInitial(DataStore dataStore) {

                dataStore.setGas(0);

                ((DataStoreGP1) dataStore).setTotal(0);
                System.out.println("your values are set");

        }
}
```

SET INITIAL 2

```
public class SetInitial2 extends SetInitialValueAbstract{


        public  void SetInitial(DataStore dataStore) {

                dataStore.setGas(0);
    //  dataStore.setL(0);
      ((DataStoreGP2) dataStore).setTotal(0);


        }
}
```

SET INITIAL 3

```
public class SetInitial3 extends SetInitialValueAbstract{


        public  void SetInitial(DataStore dataStore) {

                dataStore.setL(0);

                ((DataStoreGP3) dataStore).setTotal(0);


        }
}
```

REJECT MESSAGE ABSTRACT

```
public abstract class RejectMessageAbstract {

	public abstract void RejectMessage();

}
```

REJECT MESSAGE 1

```
public class RejectMessage1 extends RejectMessageAbstract{

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

    System.out.println("CARD IS REJECTED");
	}

}

}
```

READY MESSAGE ABSTRACT

```
public abstract class ReadyMessageAbstract {

	public abstract void ReadyMessage();

}
```

READY MESSAGE 1

```
public class ReadyMessage1 extends ReadyMessageAbstract{

	  public void ReadyMessage()
	  {
		System.out.println("System is ready to pump");
	  }

}
```

PUMP GAS UNIT ABSTRACT

```
public abstract class PumpGasUnitAbstarct  {

        public abstract void PumpGasUnit(DataStore dataStore);

}
```

PUMP GAS UNIT 1

```
public class PumpGasUnit1 extends PumpGasUnitAbstarct{

        public void PumpGasUnit(DataStore dataStore) {


                System.out.println("you asked for this much gallon");
                        dataStore.setGas(dataStore.getGas()+1)  ;
                        ((DataStoreGP1) dataStore).setTotal(((DataStoreGP1)
dataStore).getPrice()*dataStore.getGas());

}
```

PUMP GAS UNIT 2

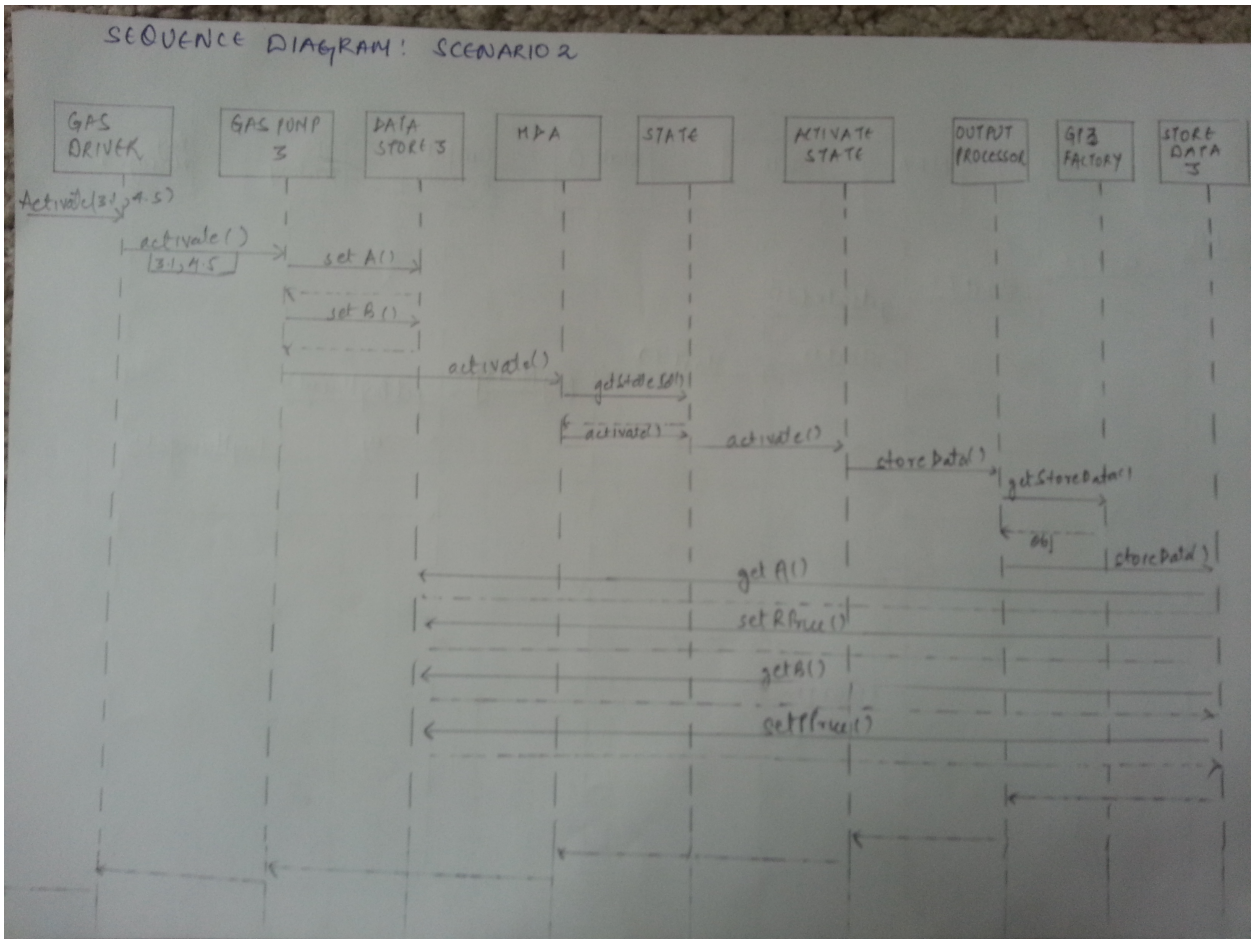
```
public class PumpGasUnit2 extends PumpGasUnitAbstarct{

        public void PumpGasUnit(DataStore dataStore) {
                // TODO Auto-generated method stub
                System.out.println("AMOUNT OF GAS YOU REQUESTED");

                dataStore.setGas(dataStore.getGas()+1);
                ((DataStoreGP2) dataStore).setTotal(((DataStoreGP2)
dataStore).getPrice()*dataStore.getGas());

        }

}
```

PUMP GAS UNIT 3

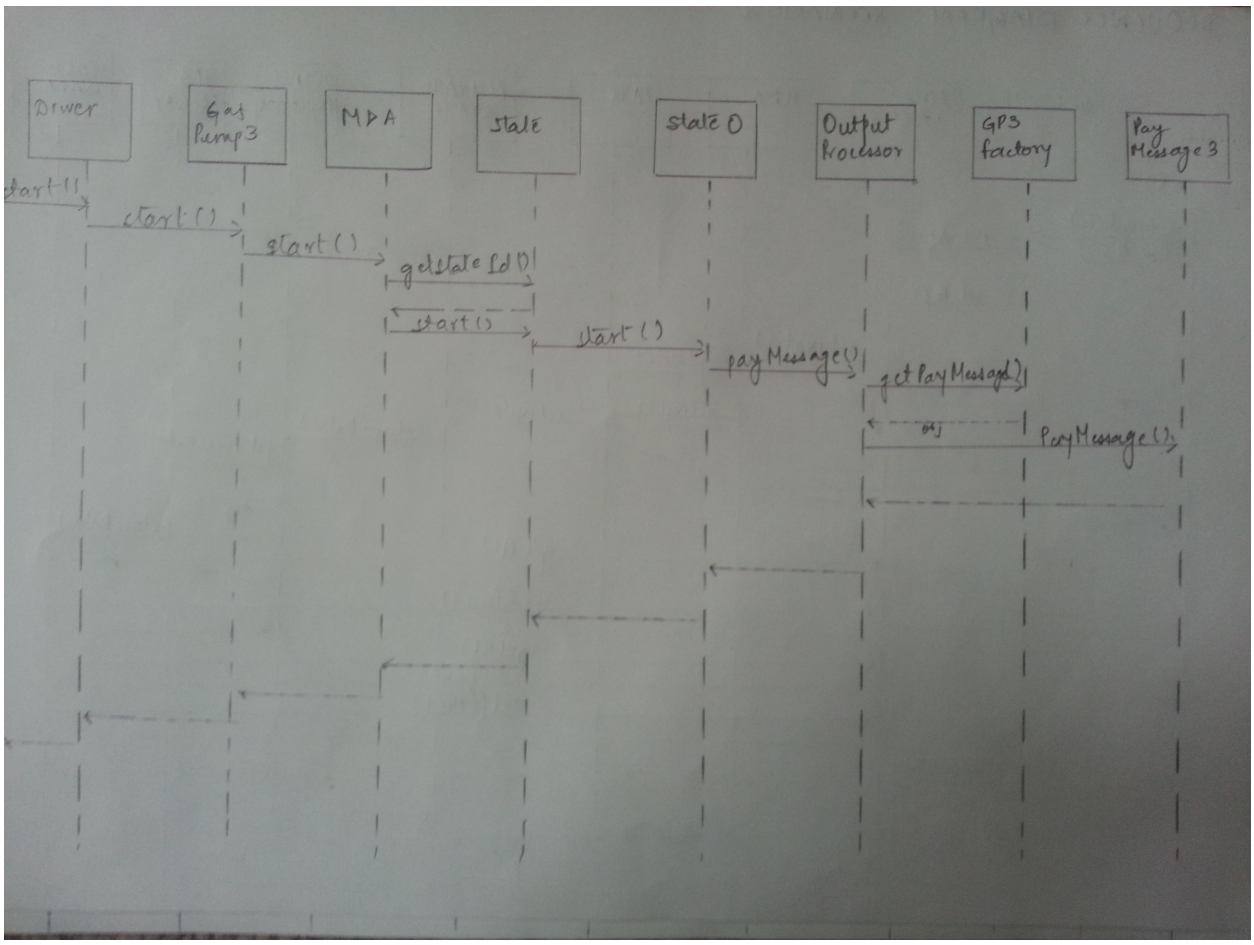
```
public class PumpGasUnit3 extends PumpGasUnitAbstarct{

        public void PumpGasUnit(DataStore  dataStore) {
```

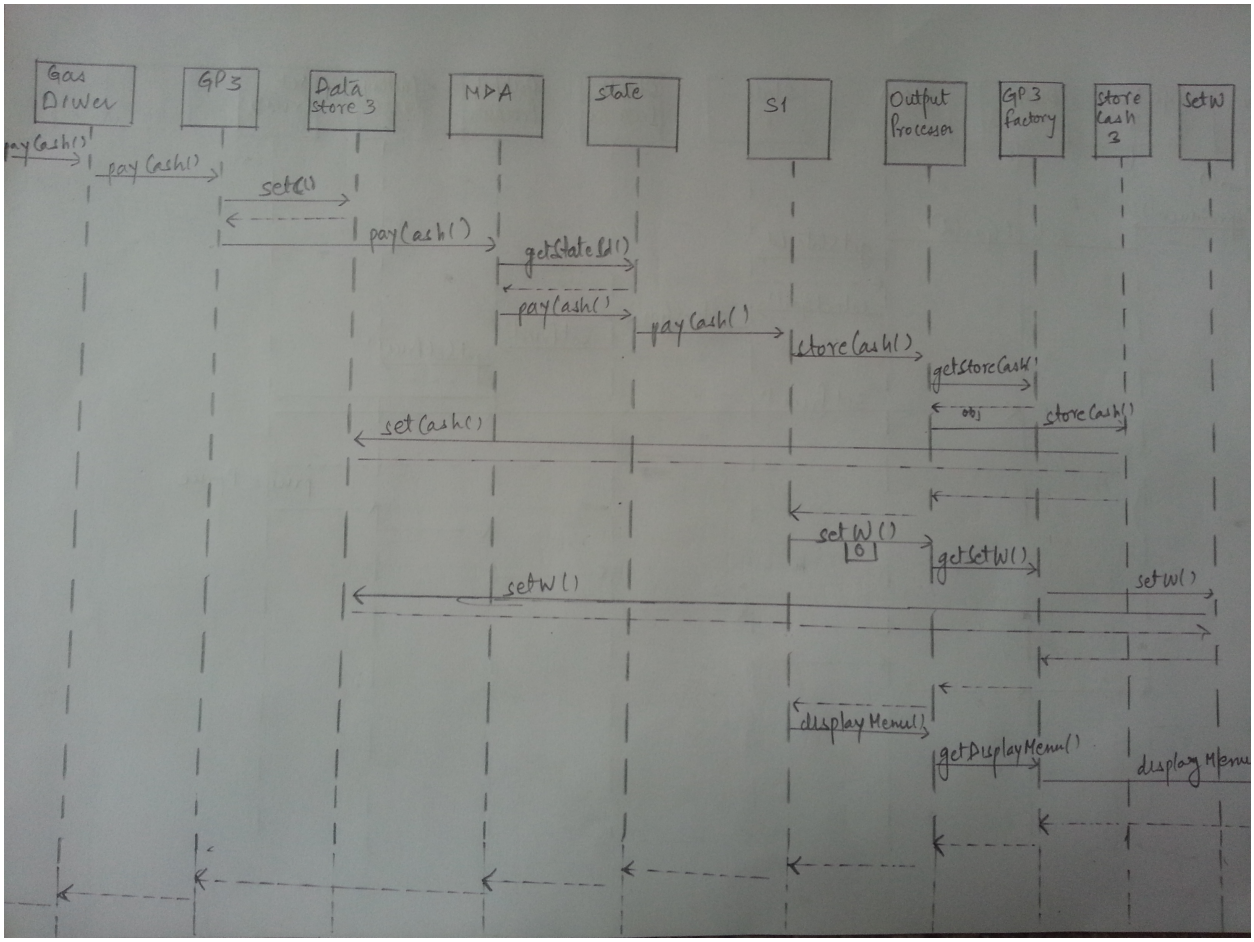
```
                    // TODO Auto-generated method stub
                      System.out.println("you asked for this much gallon");

                      dataStore.setL(dataStore.getL()+1);


                      ((DataStoreGP3) dataStore).setTotal(((DataStoreGP3)
dataStore).getPrice()*dataStore.getL());
                                }

}
```

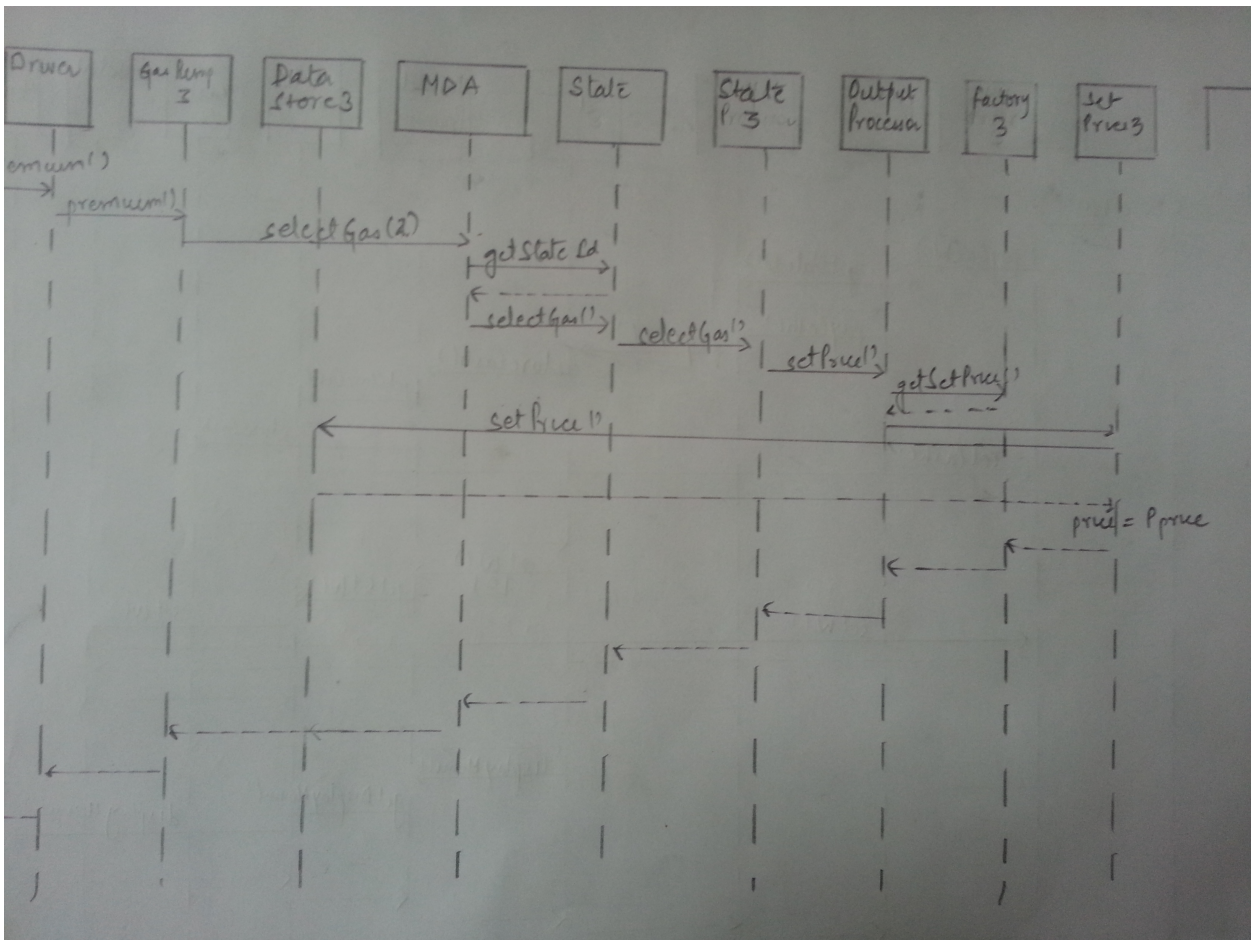
SEQUENCE DIAGR

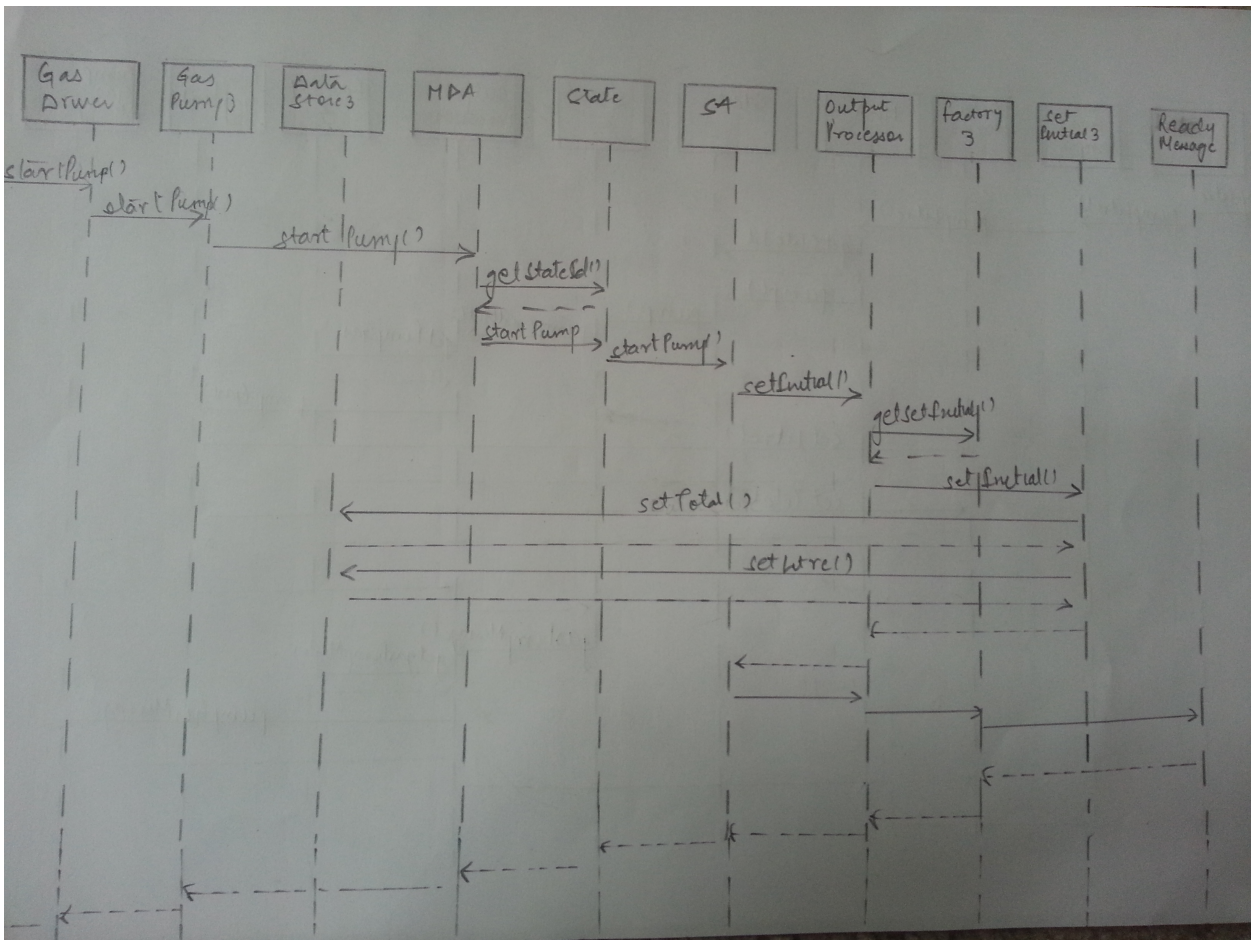


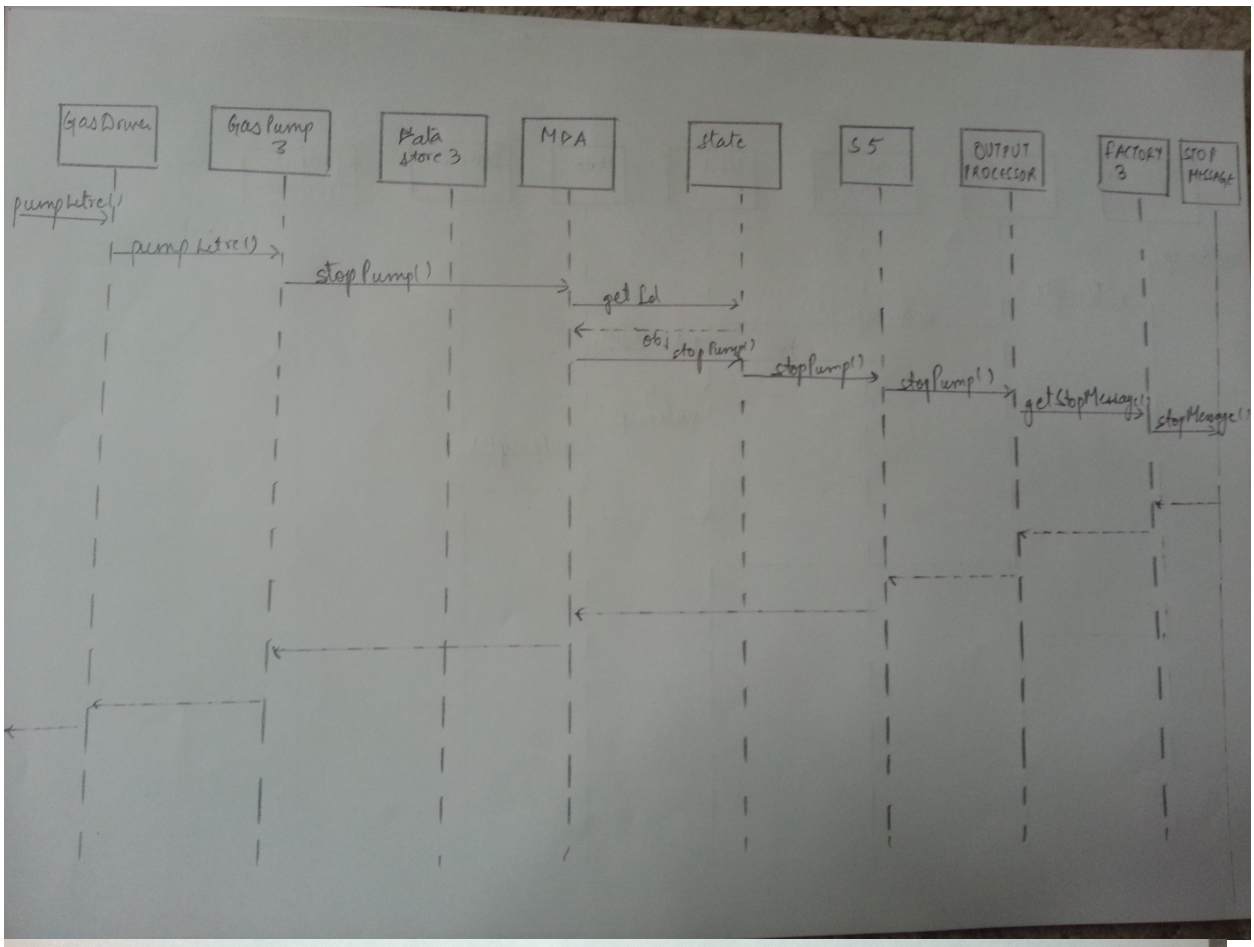
AM

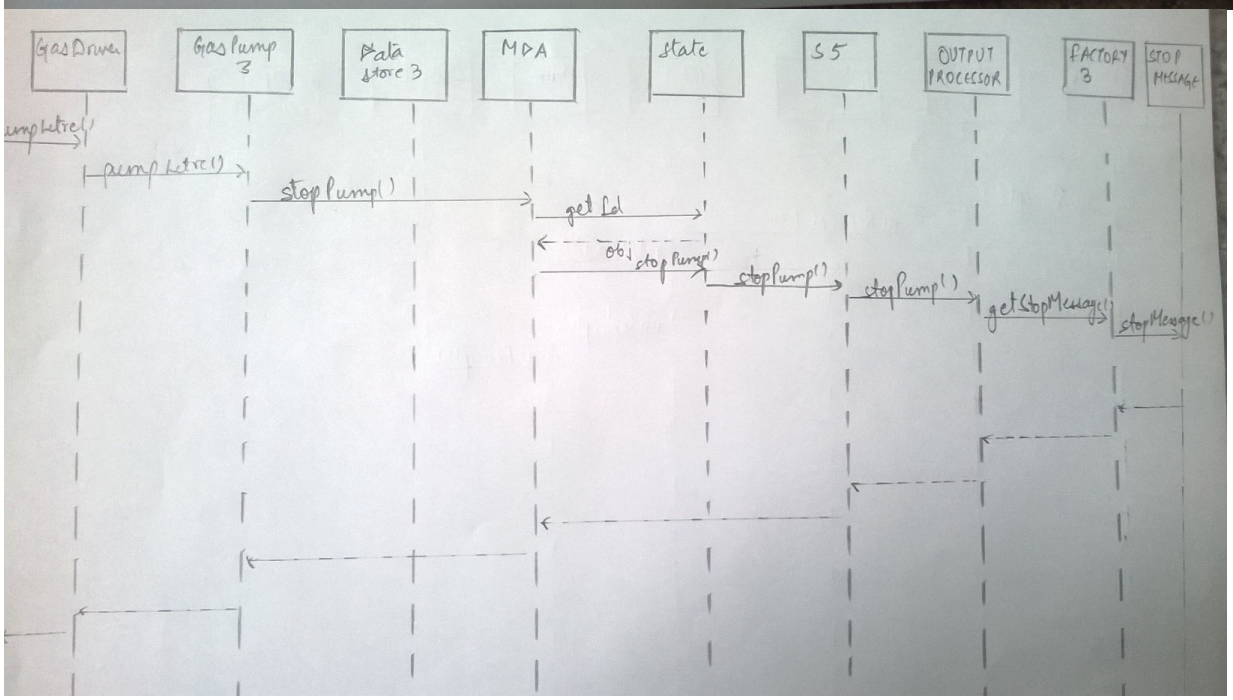

Driver
Gas Pump3
MDA
State
state 0
Output Processor
GPS factory
Pay Message 3
start()
start()
start()
getstate Id()
start()
start()
payMessage()
getPayMessage()
obj
PayMessage()

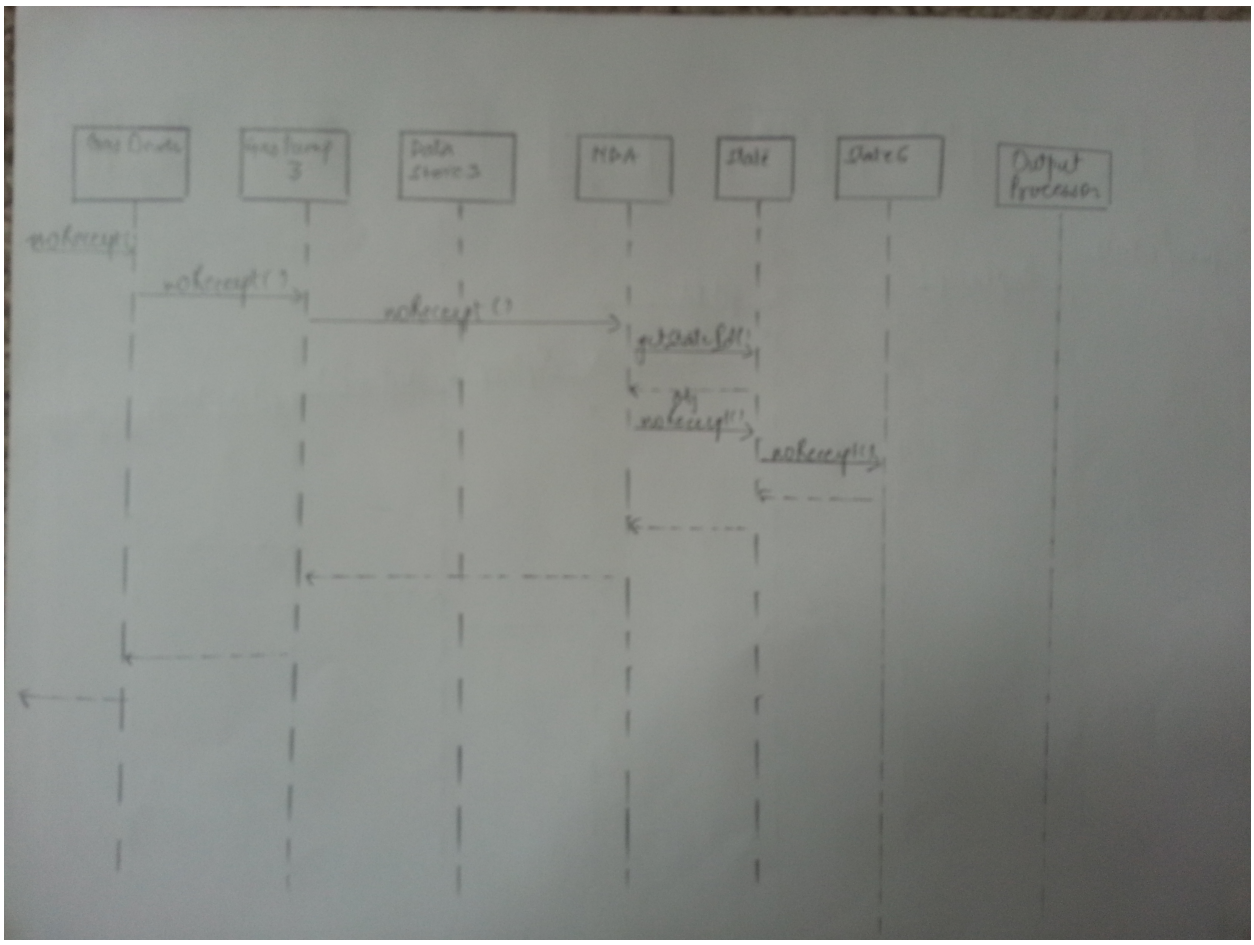

Gas Driver
GP3
Data store 3
MDA
State
S1
Output Processor
GP3 factory
Store Cash 3
SetW
pay(cash)
pay(cash)
set()
pay(cash)
getStateId()
pay(cash)
pay(cash)
storeCash()
getStoreCash()
obj
storeCash()
setCash()
setW()
0
getSetW()
setW()
setW()
displayMenu()
getDisplayMenu()
displayMenu

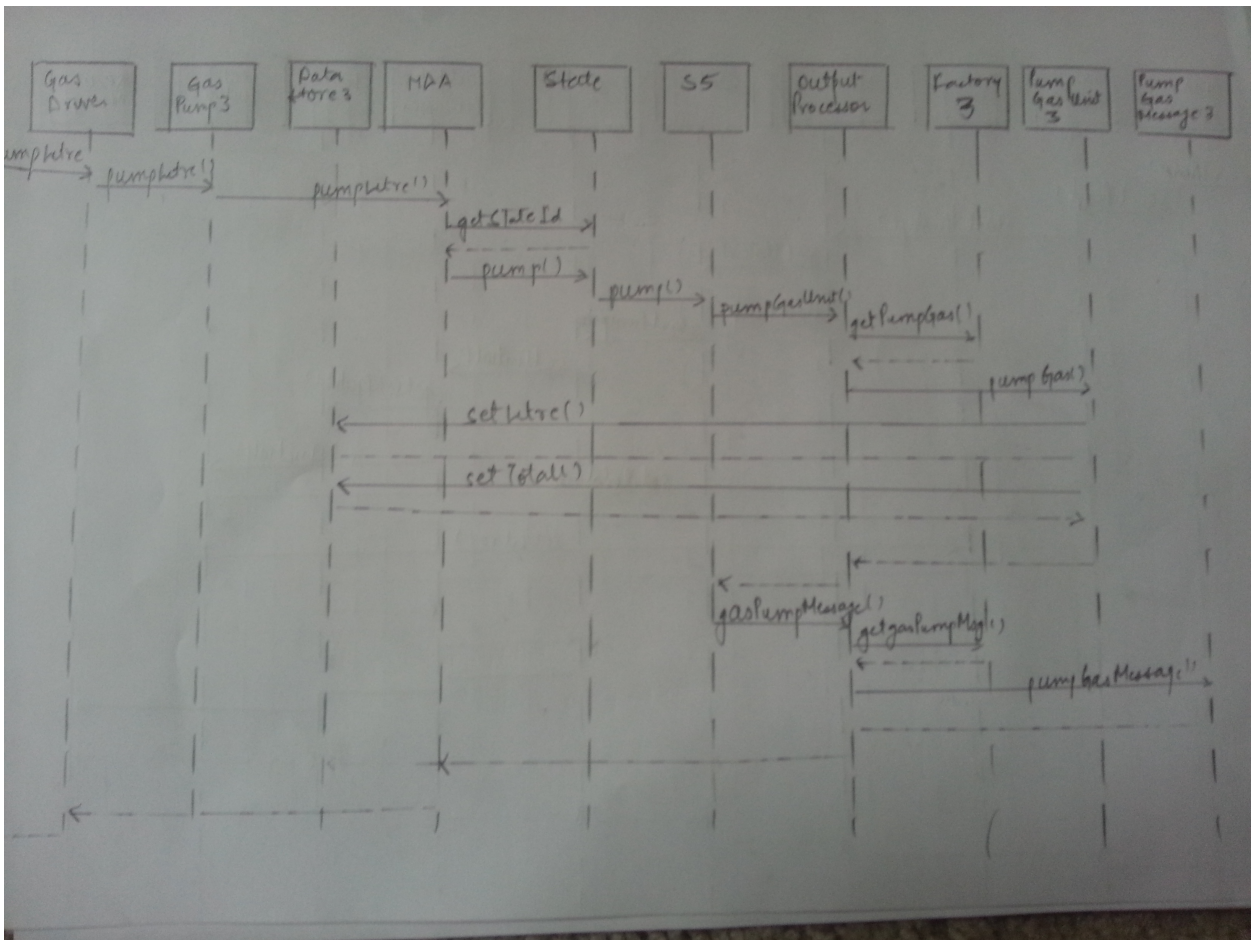

Driver
Gas Pump 3
Data Store3
MDA
State
State P. 3
Output Processor
Factory 3
Set Price3
premium()
premium()
selectGas(2)
getStateId
selectGas()
selectGas()
setPrice()
getSetPrice()
setPrice()
price = Pprice


Gas Driver
Gas Pump()
Data Store3
MDA
Gate
S4
Output Processor
factory 3
set Initial3
Ready Manage
startPump()
start Pump()
start Pump()
getStateSd()
start Pump
start Pump()
setInitial()
getSetInitial()
setInitial()
setTotal()
setLitre()

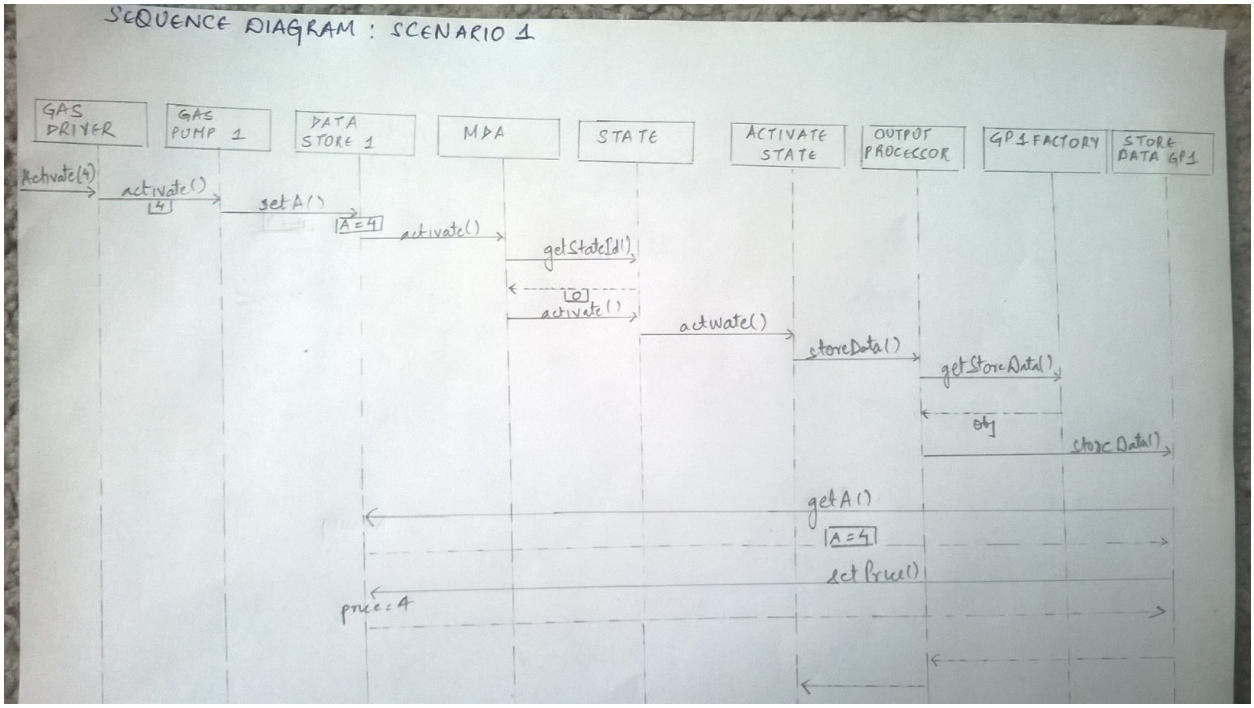

GasDriver
Gas Pump 3
Data store 3
MDA
state
S5
OUTPUT PROCESSOR
FACTORY 3
STOP MESSAGE
pumpLitre()
pumpLitre()
stopPump()
getLd
Obj
stopPump()
stopPump()
stopPump()
getStopMessage()
stopMessage()

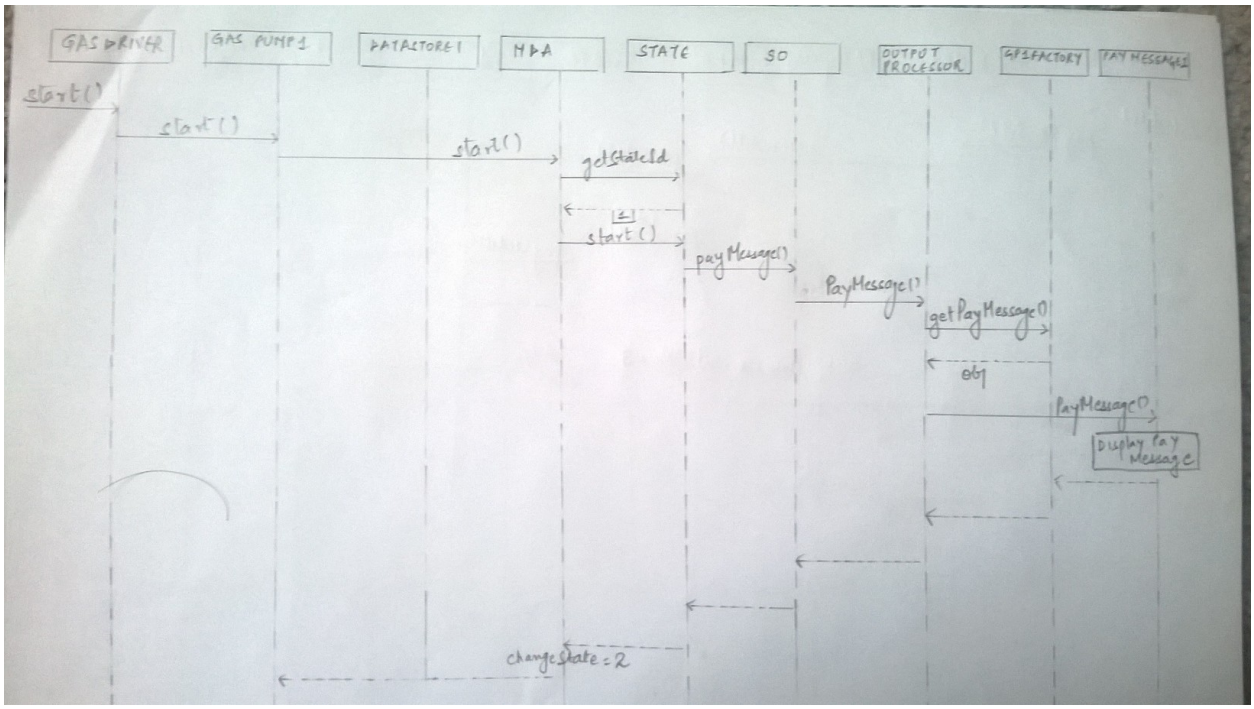

GasDriver
Gas Pump 3
Data store 3
MDA
state
S5
OUTPUT PROCESSOR
FACTORY 3
STOP MESSAGE
pumpLitre()
stopPump()
getLd
Obj
stopPump()
stopPump()
stopPump()
getStopMessage()
stopMessage()

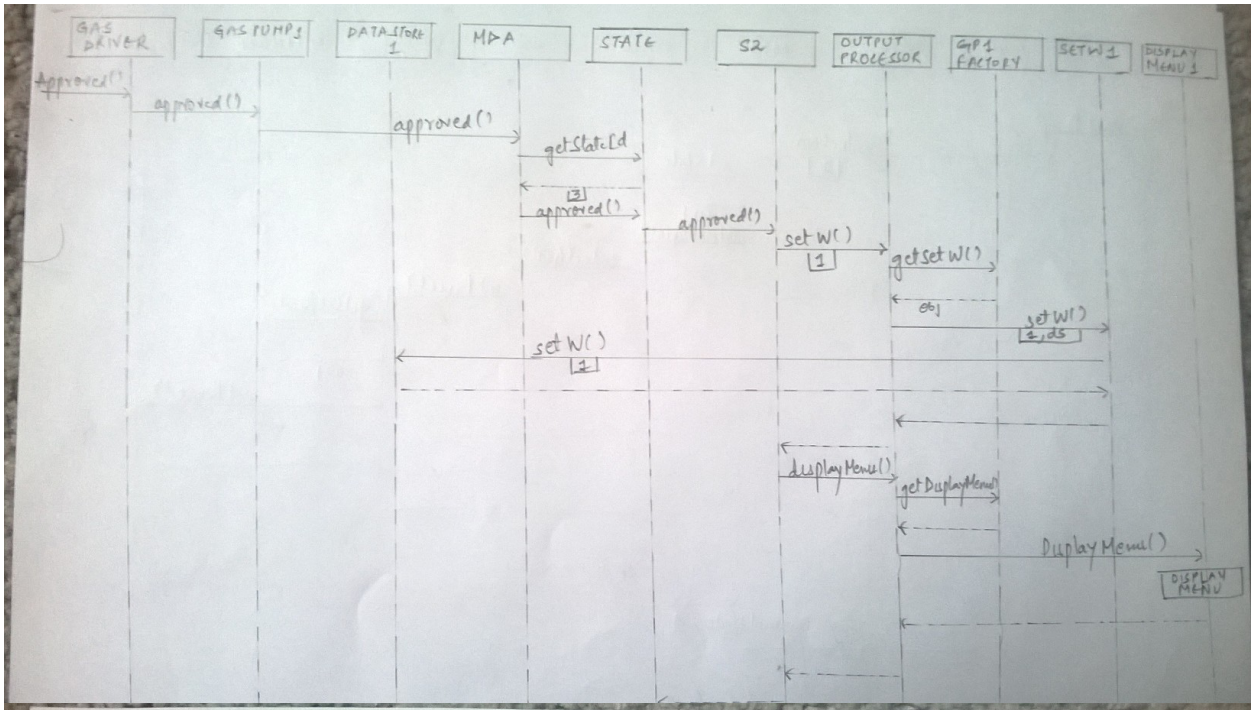

MDA
State
State G
Output Processor
noReceipt()
noReceipt()
noReceipt()
getAndSet()
noReceipt()
noReceipt()

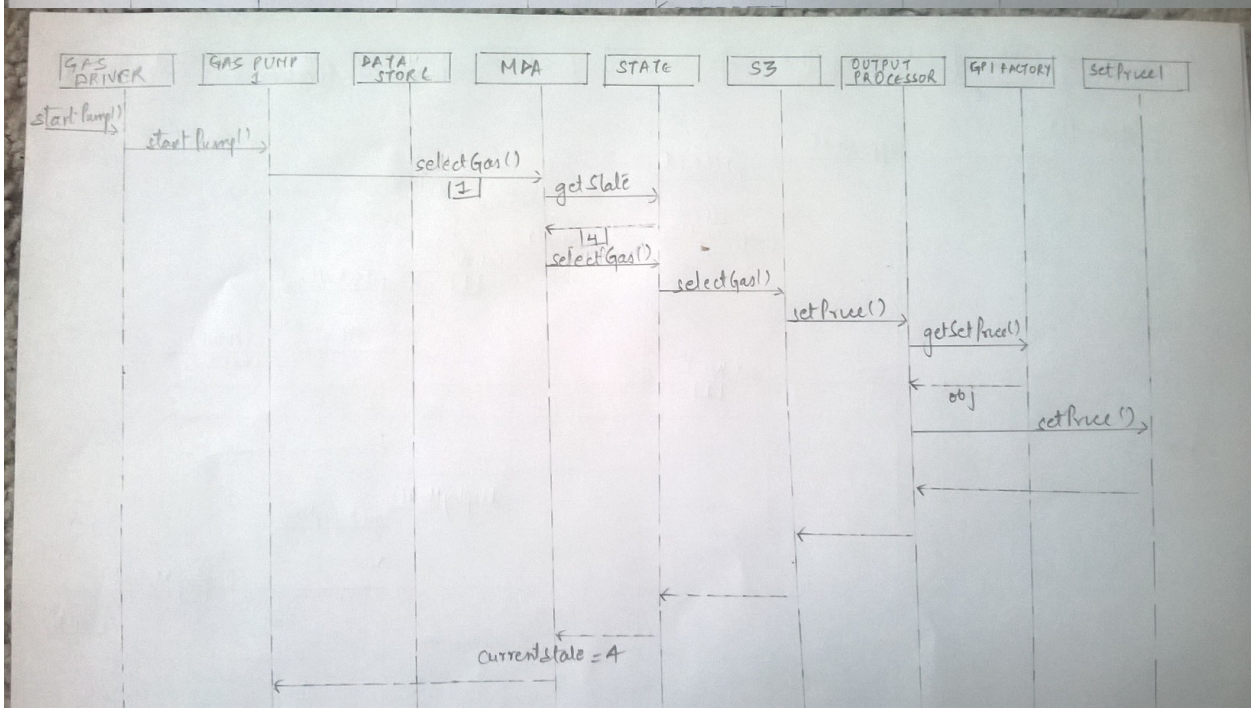

Gas Driver
Gas Pump3
Data Store3
MAA
State
S5
Output Processor
Factory 3
Pump Gas Unit 3
Pump Gas message 3
pumpLitre()
pumpLitre()
pumpLitre()
getStateId
pump()
pump()
pumpGasUnit()
getPumpGas()
pumpGas()
setLitre()
setTotal()
gasPumpMessage()
getgasPumpMsg()
pumpGasMessage()

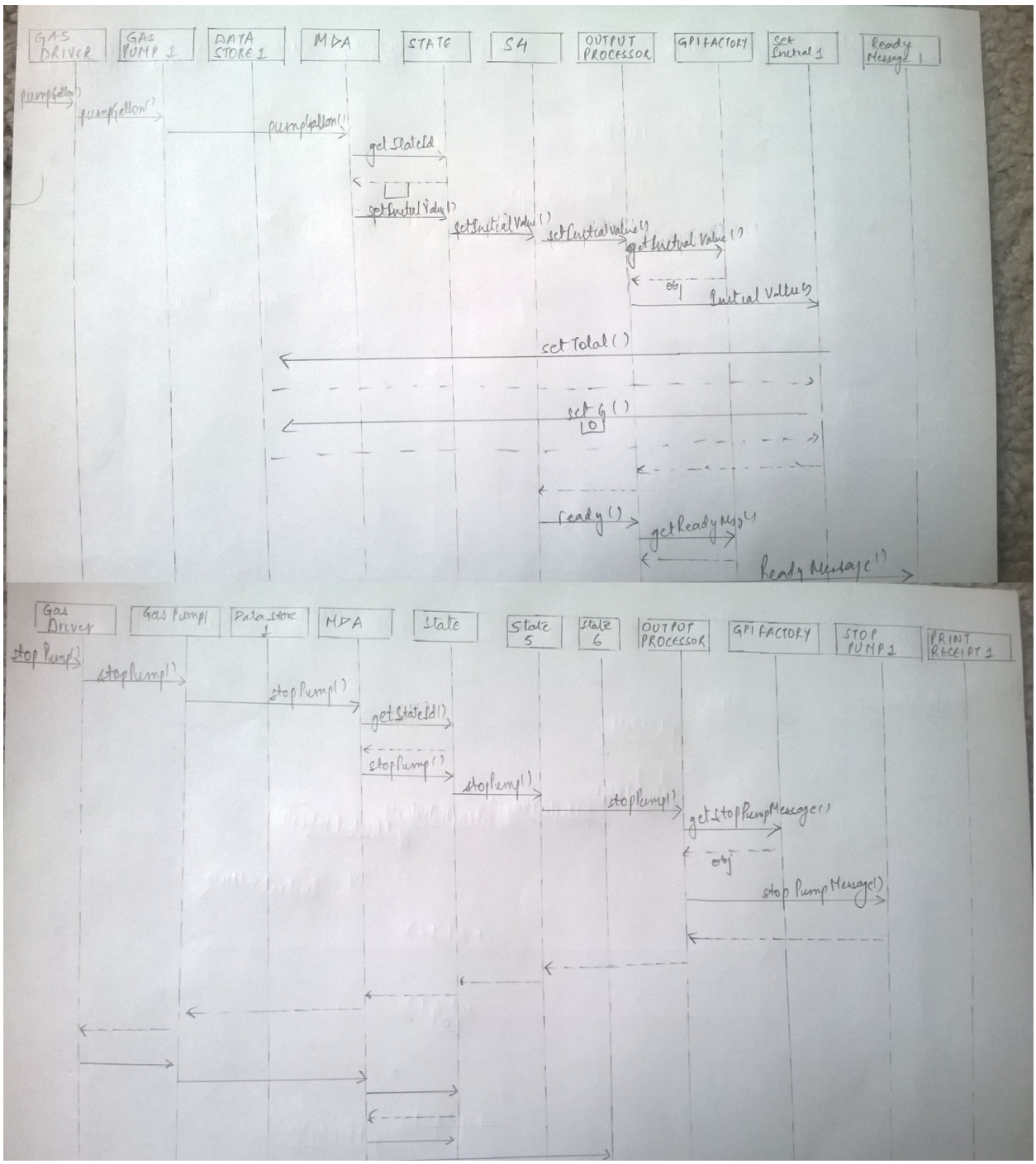
SEQUENCE 1

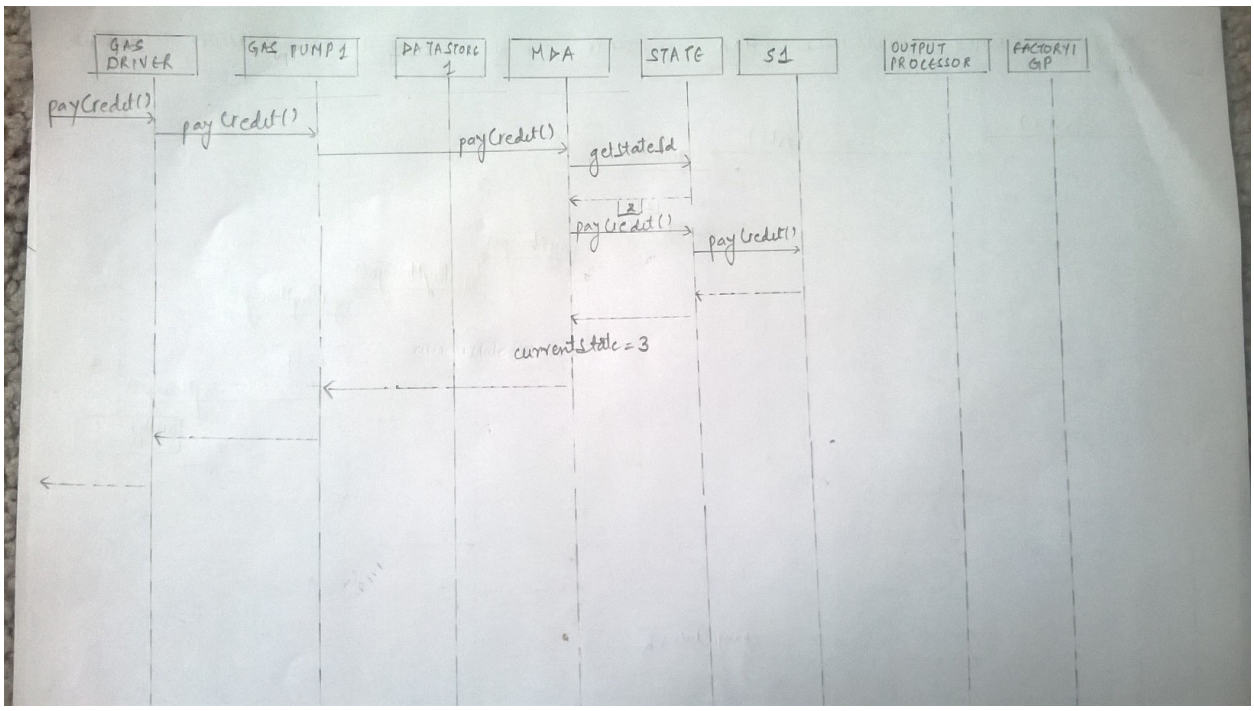

SEQUENCE DIAGRAM : SCENARIO 1
GAS DRIVER
GAS PUMP 1
DATA STORE 1
MDA
STATE
ACTIVATE STATE
OUTPUT PROCESSOR
GP1 FACTORY
STORE DATA GP1
Activate(4)
activate()
[4]
setA()
[A=4]
activate()
getStateId()
[0]
activate()
actwate()
storeData()
getStoreData()
obj
storeData()
getA()
[A=4]
setPrice()
price=4


GAS DRIVER
GAS PUMP 1
DATASTORE 1
MDA
STATE
SO
OUTPUT PROCESSOR
GP1FACTORY
PAY MESSAGE1
start()
start()
start()
getStateId
[1]
start()
payMessage()
PayMessage()
getPayMessage()
obj
PayMessage()
Display Pay Message
changeState=2


GAS DRIVER
GAS PUMP 1
DATA STORE 1
MDA
STATE
S2
OUTPUT PROCESSOR
GP1 FACTORY
SETW1
DISPLAY MENU 1
approved()
approved()
approved()
getStateId
[3]
approved()
approved()
setW()
[1]
getSetW()
obj
getW()
[1,25]
setW()
[1]
displayMenu()
getDisplayMenu()
DisplayMenu()
DISPLAY MENU


GAS DRIVER
GAS PUMP 1
DATA STORE
MDA
STATE
S3
OUTPUT PROCESSOR
GP1 FACTORY
SetPrice1
startPump()
startPump()
selectGas()
[1]
getState
[4]
selectGas()
selectGas()
setPrice()
getSetPrice()
obj
setPrice()
currentState = A


GAS DRIVER
GAS PUMP 1
DATA STORE 1
MDA
STATE
S4
OUTPUT PROCESSOR
GPI FACTORY
Set Inictial 1
Ready Message 1
pumpOn()
pumpOn()
pumpOn()
getStateId()
setInictialValue()
getInitialValue()
setInitialValue()
getInictialValue()
obj
Initial Value()
setTotal()
setG()
0
ready()
getReadyMsg()
Ready Message()
Gas Driver
Gas Pump1
Data_store
MDA
State
State 5
State 6
OUTPUT PROCESSOR
GPI FACTORY
STOP PUMP 1
PRINT RECEIPT 1
stopPump()
stopPump()
stopPump()
getStateId()
stopPump()
stopPump()
stopPump()
getStopPumpMessage()
obj
stopPumpMessage()


GAS DRIVER
GAS PUMP 1
DATASTORE 1
MDA
STATE
S1
OUTPUT PROCESSOR
FACTORY1 GP
payCredit()
payCredit()
payCredit()
getStateId
payCredit()
payCredit()
currentState = 3