CS-586 SOFTWARE SYSTEM ARCHITECTURE

PROJECT

Gas Pump

HIRAL RAMANI
4-26-2017
hramani@hawk.iit.edu
A20370004

TABLE OF CONTENTS	
A.GOAL OF PROJECT	
B.PROJECT DESCRIPTION	
1. MODEL DRIVEN ARCHITECTURE OF PROJECT COMPONENT	
2. CLASS DIAGRAMS OF MDA AND ACCOUNT COMPONENTS	
2A. STATE PATTERN	
2B.STRATEGY PATTERN	
2C. ABSTRACT FACTORY PATTERN	
3. DESCRIPTION OF ALL THE CLASS IN THE SYSTEM	
4. DYNAMICS (SEQUENCE DIAGRAM)	
4A. SCENARIO 1	
4B.SCENARIO 2	
5. PATTERN DESCIPTION	
6. SOURCE CODE	

A. **Goal of Project:**

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:

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

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

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

1. Model Driven Architecture of the ACCOUNT Components

This section includes all the MDA-EFSM model for the GasPump components along with a list of the events for the MDA-EFSM, a list of actions for the MDA-EFSM, a State diagram of the MDA-EFSM, and pseudo-code for all of the operations of the input processors of GasPump-1, GasPump -2.

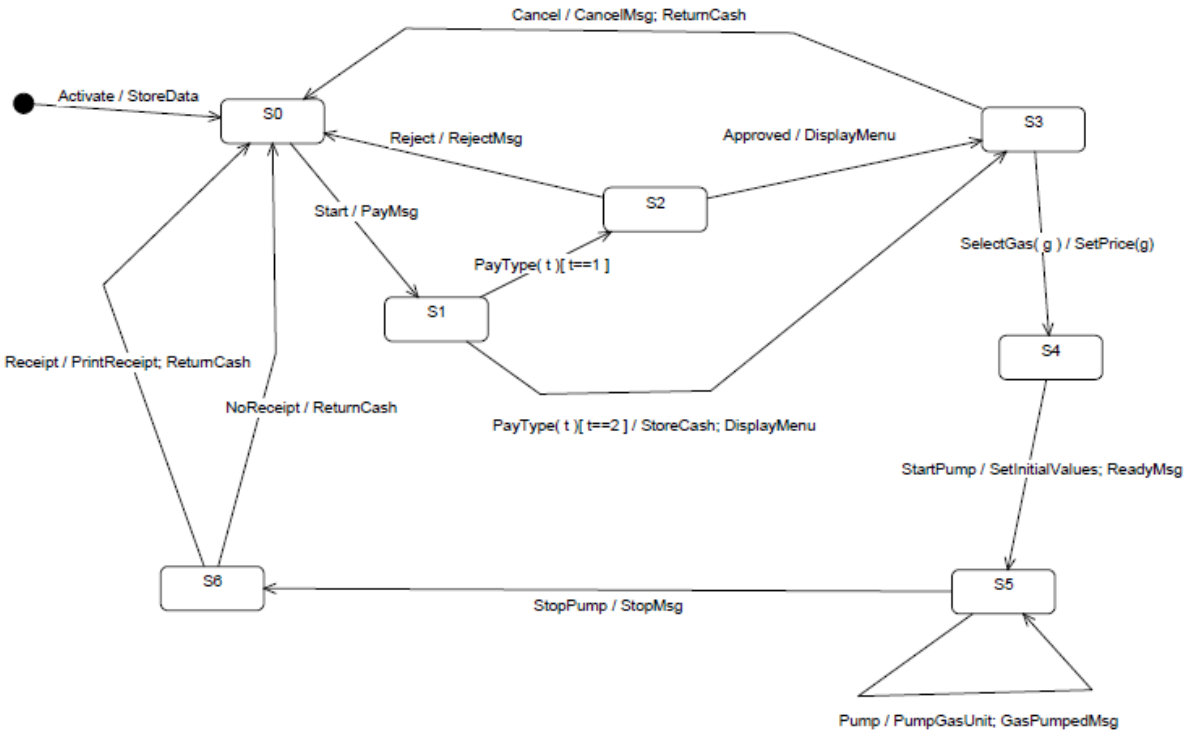
LIST OF MDA-EFSM EVENTS:

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

List of MDA-EFSM Actions:

StoreData	// stores price(s) for the gas from the temporary data store
PayMsg	// displays a type of payment method
StoreCash	// stores cash from the temporary data store
DisplayMenu	// display a menu with a list of selections
RejectMsg	// displays credit card not approved message
SetPrice(int g)	// set the price for the gas identified by g identifier
ReadyMsg	// displays the ready for pumping message
SetInitialValues	// set G (or L) and total to 0
PumpGasUnit	// disposes unit of gas and counts # of units disposed
GasPumpedMsg	// displays the amount of disposed gas
StopMsg	// stop pump message and receipt? msg (optionally)
PrintReceipt	// print a receipt
CancelMsg	// displays a cancellation message
ReturnCash	// returns the remaining cash

A state diagram of the MDA-EFSM



MDA-EFSM for Gas Pumps

GasPump 1

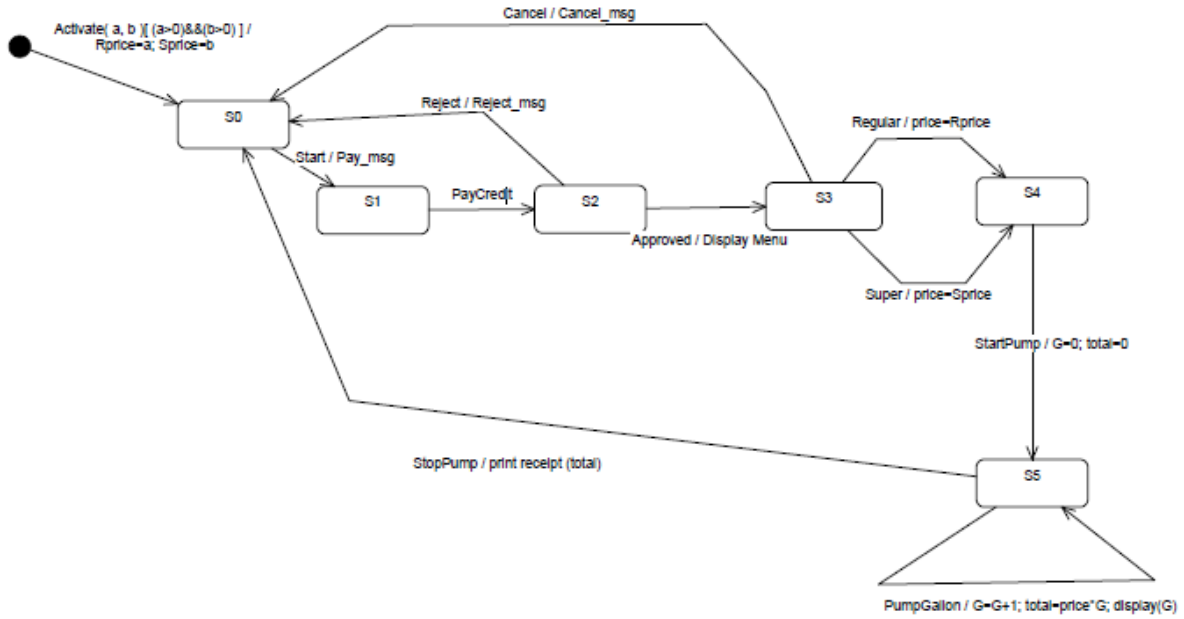


Figure 1: EFSM for GasPump-1

Operations of the Input Processor

(GasPump-1)

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

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

*-------------------------------------------------------------------------------------------------------------*
PayCredit() {
m->PayType(1);
}

*-------------------------------------------------------------------------------------------------------------*

Reject() {
m->Reject();
```

```
}

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

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

*-----------------------------------------------------------------------------------------------------------*
Super() {
m->SelectGas(2)
}

*-----------------------------------------------------------------------------------------------------------*
Regular() {
m->SelectGas(1)
}

*-----------------------------------------------------------------------------------------------------------*
StartPump() {
m->StartPump();
}
else m->IncorrectUnlock();
}

*-----------------------------------------------------------------------------------------------------------*
PumpGallon() {
m->Pump();
StopPump() {
m->StopPump();
m->Receipt();
}

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

GasPump 2

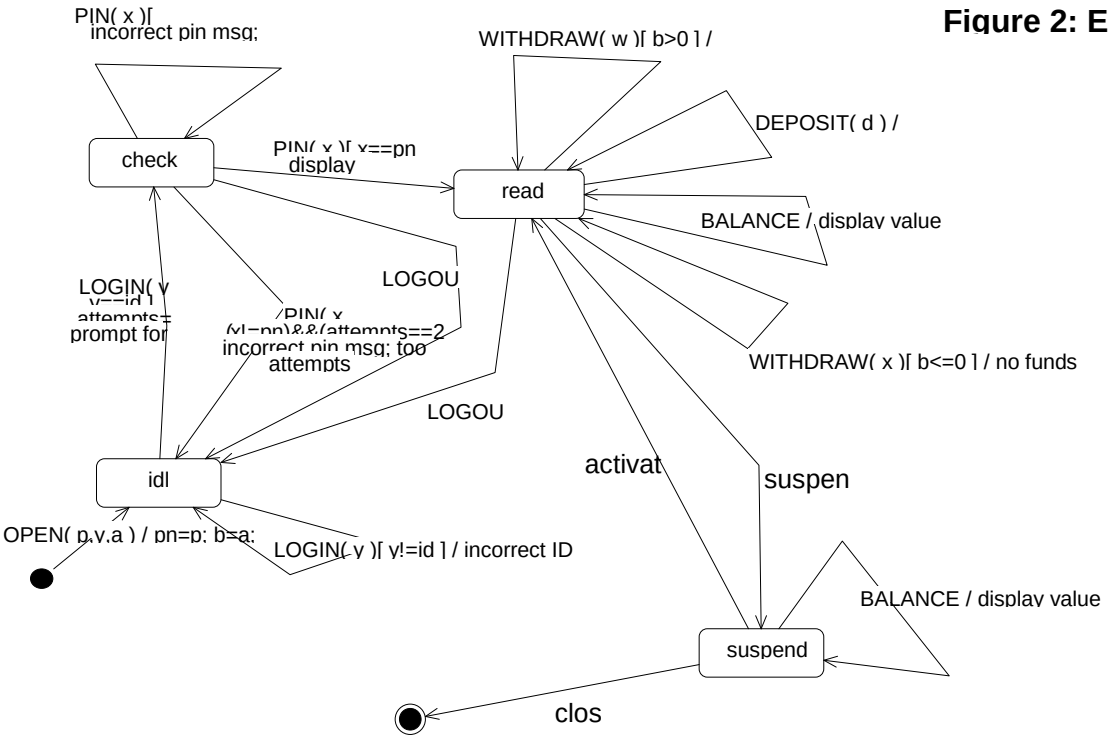


Figure 2: EFSM of

Operations of the Input Processor

(GasPump-2)

```
*-----------------------------------------------------------------------------------------------------*
Activate(int a, int b, int c) {
if ((a>0)&&(b>0)&&(c>0)) {
d->temp_a=a;
d->temp_b=b;
d->temp_c=c
m->Activate()
}
}

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

*-----------------------------------------------------------------------------------------------------*
PayCash(float c) {
if (c>0) {
d->temp_cash=c;
```

```
m->PayType(2)
}
}

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

*-----------------------------------------------------------------------------------------------------------*
Super() {
m->SelectGas(2)
}

*-----------------------------------------------------------------------------------------------------------*
Premium() {
m->SelectGas(3)
}

*-----------------------------------------------------------------------------------------------------------*
Regular() {
m->SelectGas(1)
}

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

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

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

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

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

Notice:
cash: contains the value of cash deposited
price: contains the price of the selected gas
L: contains the number of liters already pumped
cash , L, price are in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

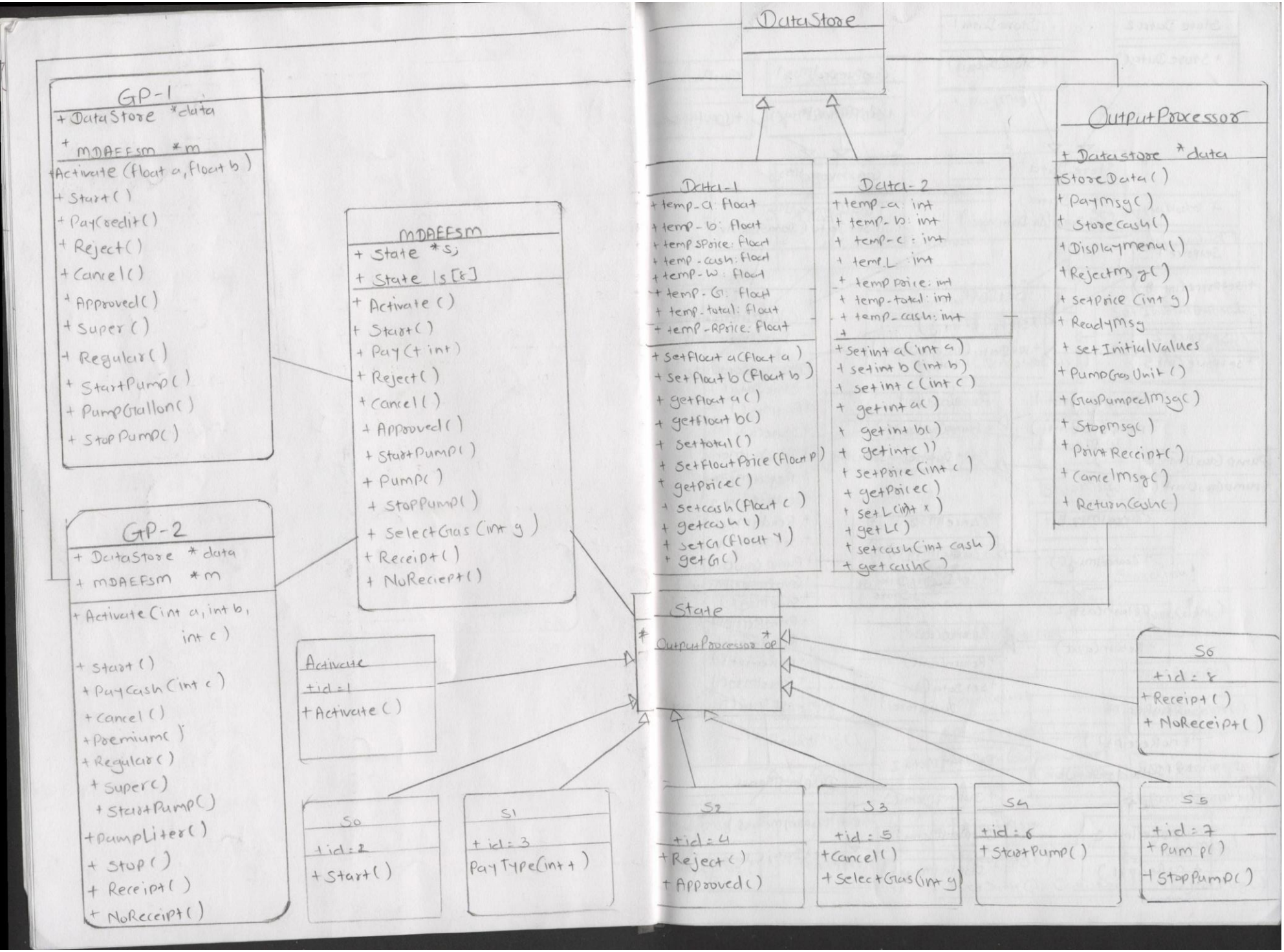
2. Class diagram(s) of the MDA and of the ACCOUNT components.

This section includes class diagrams. Due to complexity of showing the class diagram as a whole, the class diagram is show in parts as follows.

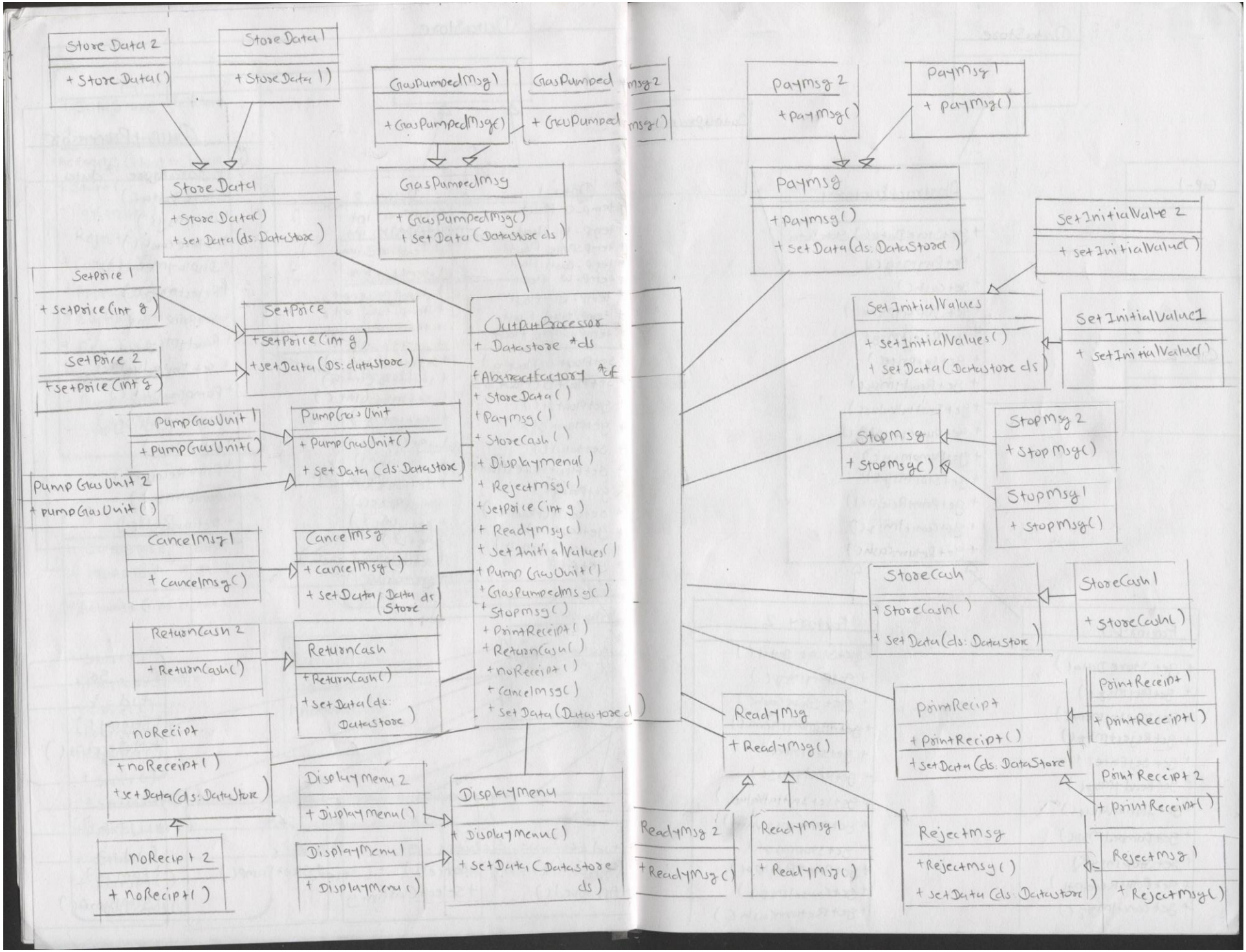
Note : Due to complexity of diagrams methods and variables of repeating classes are not written again

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

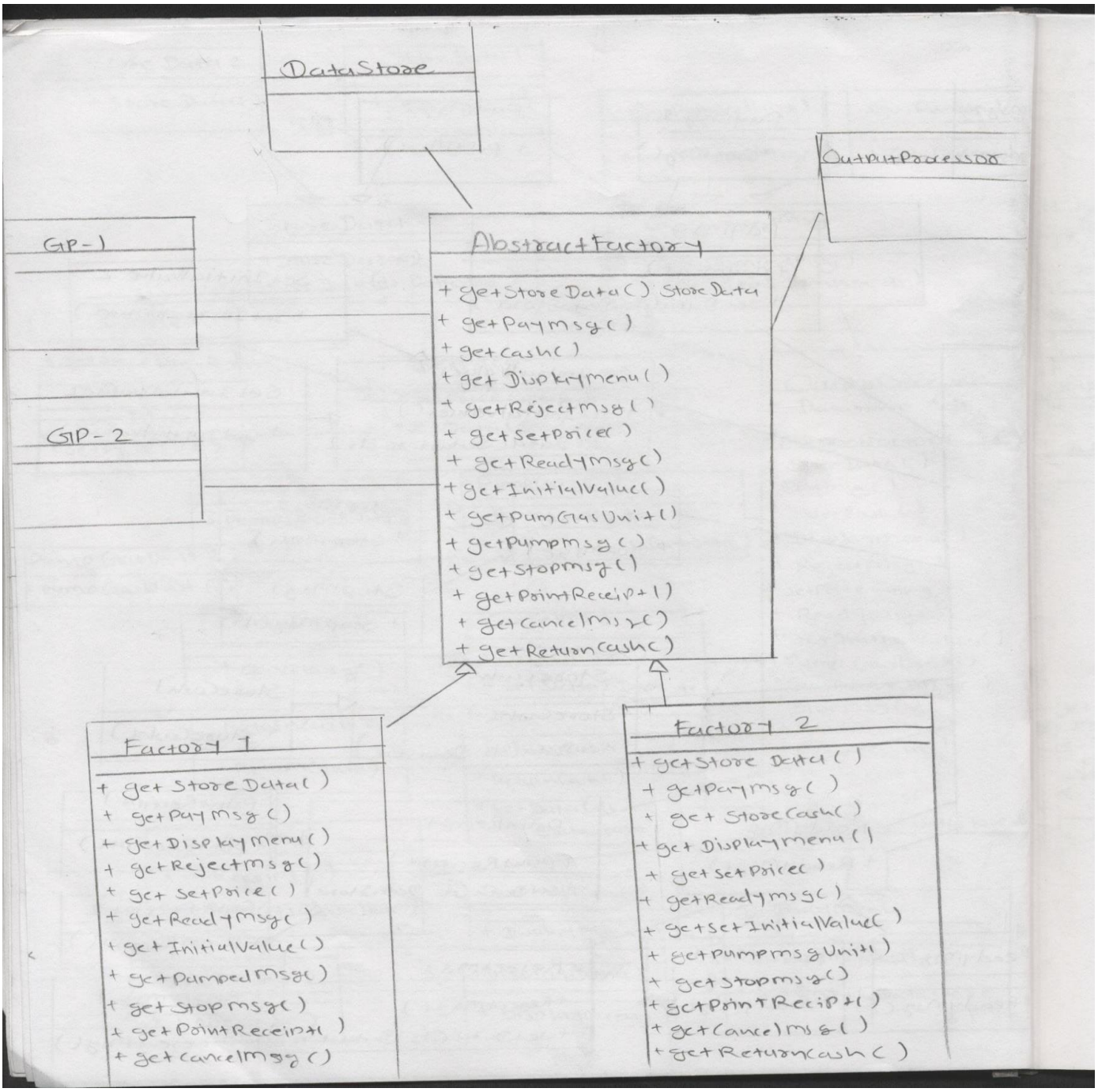
State Pattern


GP-1
+ DataStore *data
+ MDAEFSM *m
+Activate (float a, float b)
+ Start()
+ PayCredit()
+ Reject()
+ Cancel()
+ Approved()
+ Super()
+ Regular()
+ StartPump()
+ PumpGallon()
+ StopPump()
GP-2
+ DataStore *data
+ MDAEFSM *m
+ Activate(int a, int b, int c)
+ Start()
+ PayCash(int c)
+ Cancel()
+ Poemium()
+ Regular()
+ Super()
+ StartPump()
+ PumpLiter()
+ Stop()
+ Receipt()
+ NoReceipt()
MDAEFSM
+ State *s;
+ State ls[8]
+ Activate()
+ Start()
+ Pay(int)
+ Reject()
+ Cancel()
+ Approved()
+ StartPump()
+ Pump()
+ StopPump()
+ SelectGas(int g)
+ Receipt()
+ NoReciept()
Activate
+id = 1
+ Activate()
S0
+id = 2
+ Start()
S1
+ id = 3
PayType(int+)
DataStore
Data-1
+ temp_a: float
+ temp_b: float
+ tempSPrice: float
+ temp_cash: float
+ temp_W: float
+ temp_G: float
+ temp_total: float
+ temp_RPrice: float
+ SetFloat a(Float a)
+ Set float b (Float b)
+ getFloat a()
+ getfloat b()
+ Settotal()
+ SetFloatPrice(FloatP)
+ getPrice()
+ setcash(Float c)
+ getcash()
+ setG(Float Y)
+ getG()
Data-2
+ temp_a: int
+ temp_b: int
+ temp_c: int
+ temp_L: int
+ tempPrice: int
+ temp_total: int
+ temp_cash: int
+ Setint a(int a)
+ setint b(int b)
+ setint c(int c)
+ getint a()
+ getint b()
+ getint c))
+ setPrice(int c)
+ getPrice()
+ setL(int x)
+ getL()
+ setcash(int cash)
+ getcash()
OutputProcessor
+ Datastore *data
+StoreData()
+ Paymsg()
+ Storecash()
+ Displaymenu()
+ Rejectmsg()
+ setPrice(int g)
+ Readymsg
+ setInitialValues
+ PumpGasUnit()
+ GasPumpedMsg()
+ Stopmsg()
+ Print Receipt()
+ Cancelmsg()
+ ReturnCash()
State
OutputProcessor *op
S6
+id = 8
+ Receipt()
+ NoReceipt()
S2
+id = 4
+ Reject()
+ Approved()
S3
+id = 5
+ Cancel()
+ SelectGas(int g)
S4
+id = 6
+ StartPump()
S5
+id = 7
+ Pump()
+ StopPump()

Strategy Pattern


StoreData2
+StoreData()
StoreData1
+StoreData1()
GasPumpedMsg1
+GasPumpedMsg()
GasPumpedMsg2
+GasPumped()
PayMsg2
+payMsg()
PayMsg1
+payMsg()
StoreData
+StoreData()
+setData(ds: DataStore)
GasPumpedMsg
+GasPumpedMsg()
+SetData(DataStore cls)
PayMsg
+PayMsg()
+setData(ds: DataStore)
SetInitialValue2
+setInitialValue()
SetInitialValue1
+setInitialValue()
SetInitialValues
+setInitialValues()
+setData(DataStore cls)
SetPrice1
+setPrice(int g)
SetPrice
+setPrice(int g)
+setData(DS: datastore)
SetPrice2
+setPrice(int g)
OutPutProcessor
+Datastore *ds
+AbstractFactory *af
+StoreData()
+PayMsg()
+StoreCash()
+DisplayMenu()
+RejectMsg()
+setPrice(int g)
+ReadyMsg()
+setInitialValues()
+PumpGasUnit()
+GasPumpedMsg()
+StopMsg()
+PrintReceipt()
+ReturnCash()
+noReceipt()
+cancelMsg()
+setData(Datastore cls)
PumpGasUnit1
+PumpGasUnit()
PumpGasUnit
+PumpGasUnit()
+setData(cls: Datastore)
PumpGasUnit2
+pumpGasUnit()
StopMsg
+StopMsg()
StopMsg2
+StopMsg()
StopMsg1
+StopMsg()
CancelMsg1
+cancelMsg()
CancelMsg
+cancelMsg()
+setData(DataStore ds)
StoreCash
+StoreCash()
+setData(cls: DataStore)
StoreCash1
+StoreCash()
ReturnCash2
+ReturnCash()
ReturnCash
+ReturnCash()
+setData(ds: DataStore)
ReadyMsg
+ReadyMsg()
PrintRecipt
+PrintRecipt()
+setData(ds: DataStore)
PrintRecipt1
+printRecipt()
noRecipt
+noReceipt()
+setData(ds: DataStore)
DisplayMenu2
+DisplayMenu()
DisplayMenu
+DisplayMenu()
+setData(DataStore cls)
ReadyMsg2
+ReadyMsg()
ReadyMsg1
+ReadyMsg()
RejectMsg
+RejectMsg()
+setData(cls: DataStore)
PrintRecipt2
+printRecipt()
noRecipt2
+noReceipt()
DisplayMenu1
+DisplayMenu()
RejectMsg1
+RejectMsg()

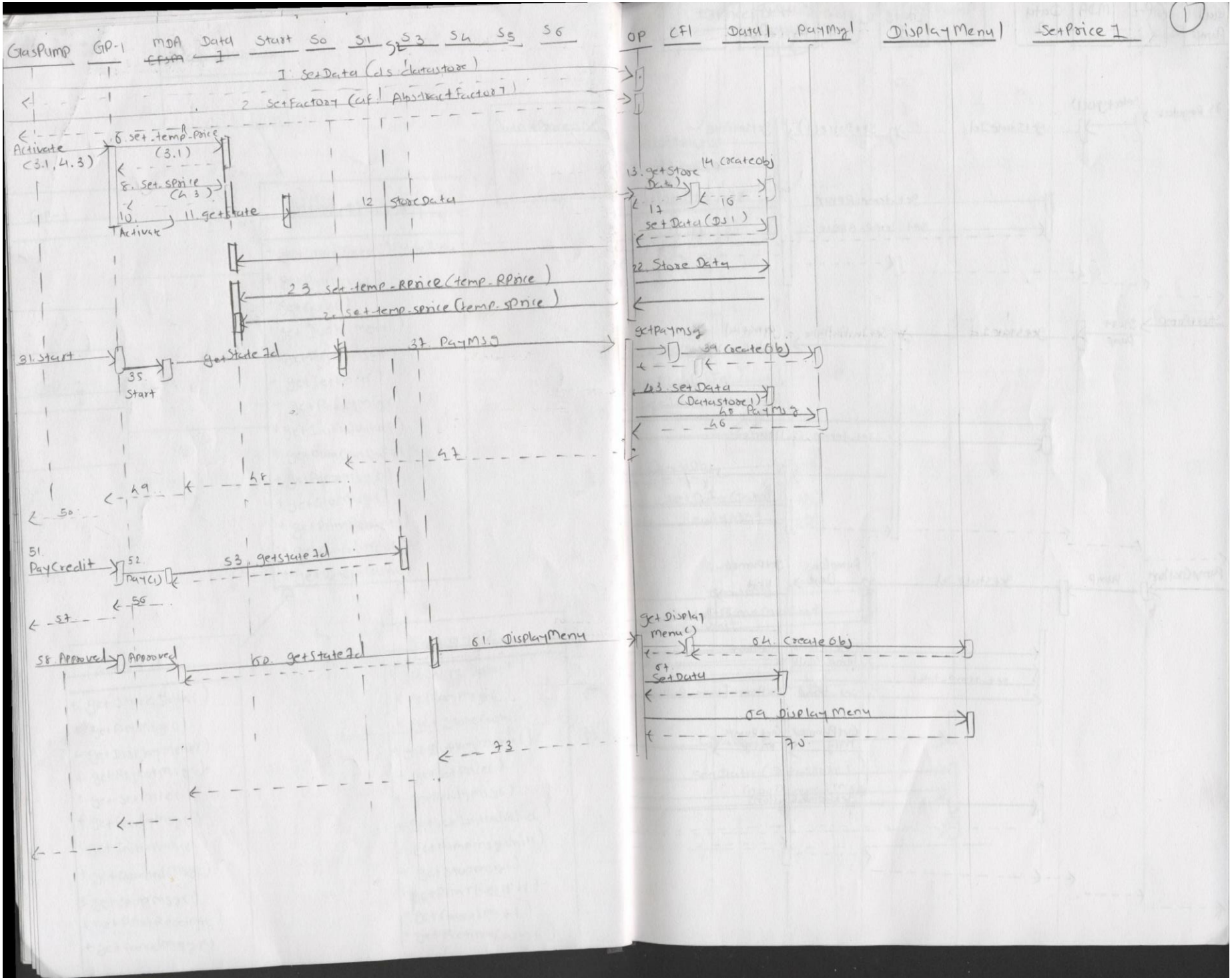
Abstarct Pattern

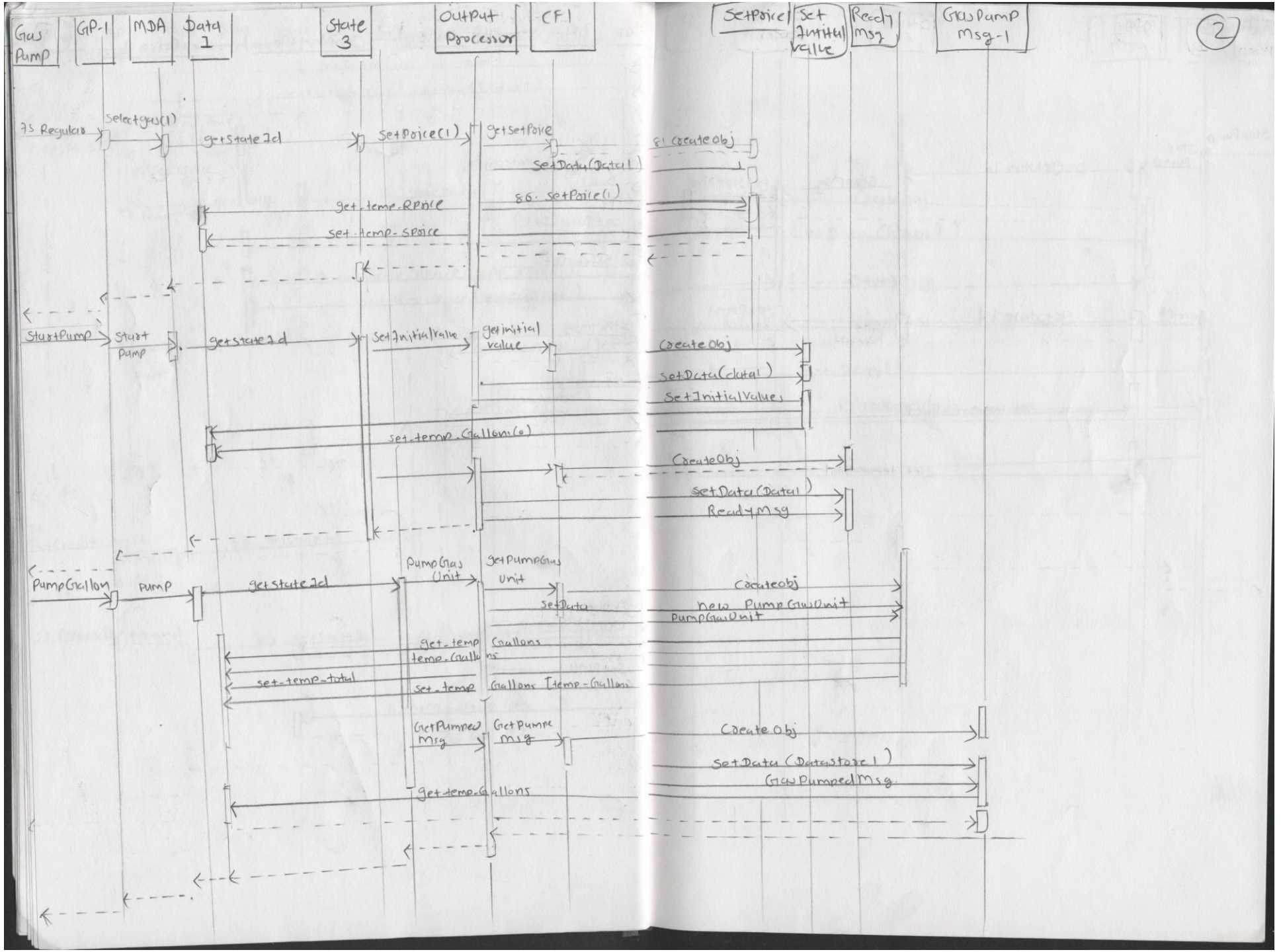


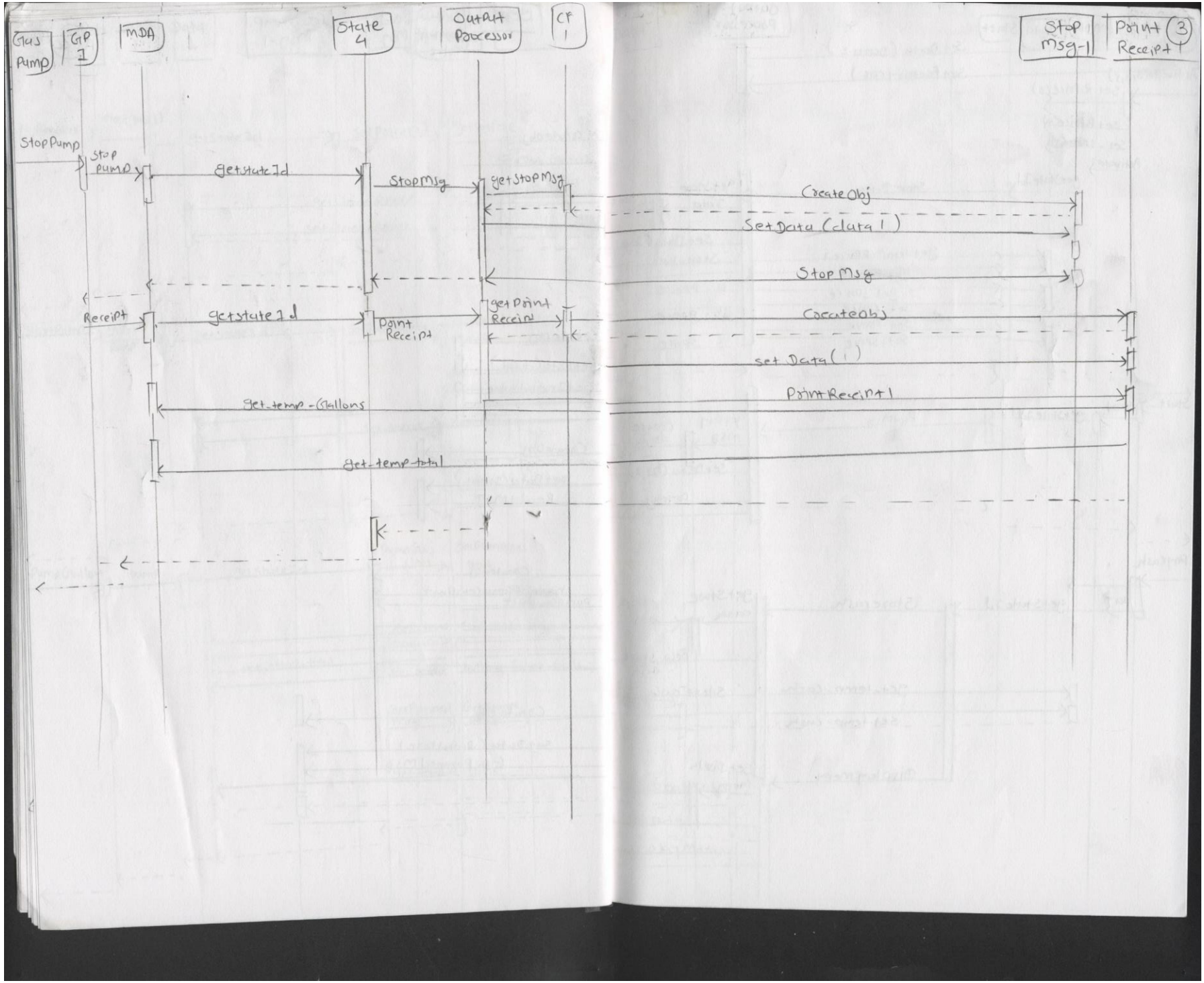
3. Dynamics

Provide two sequence diagrams for two Scenarios:

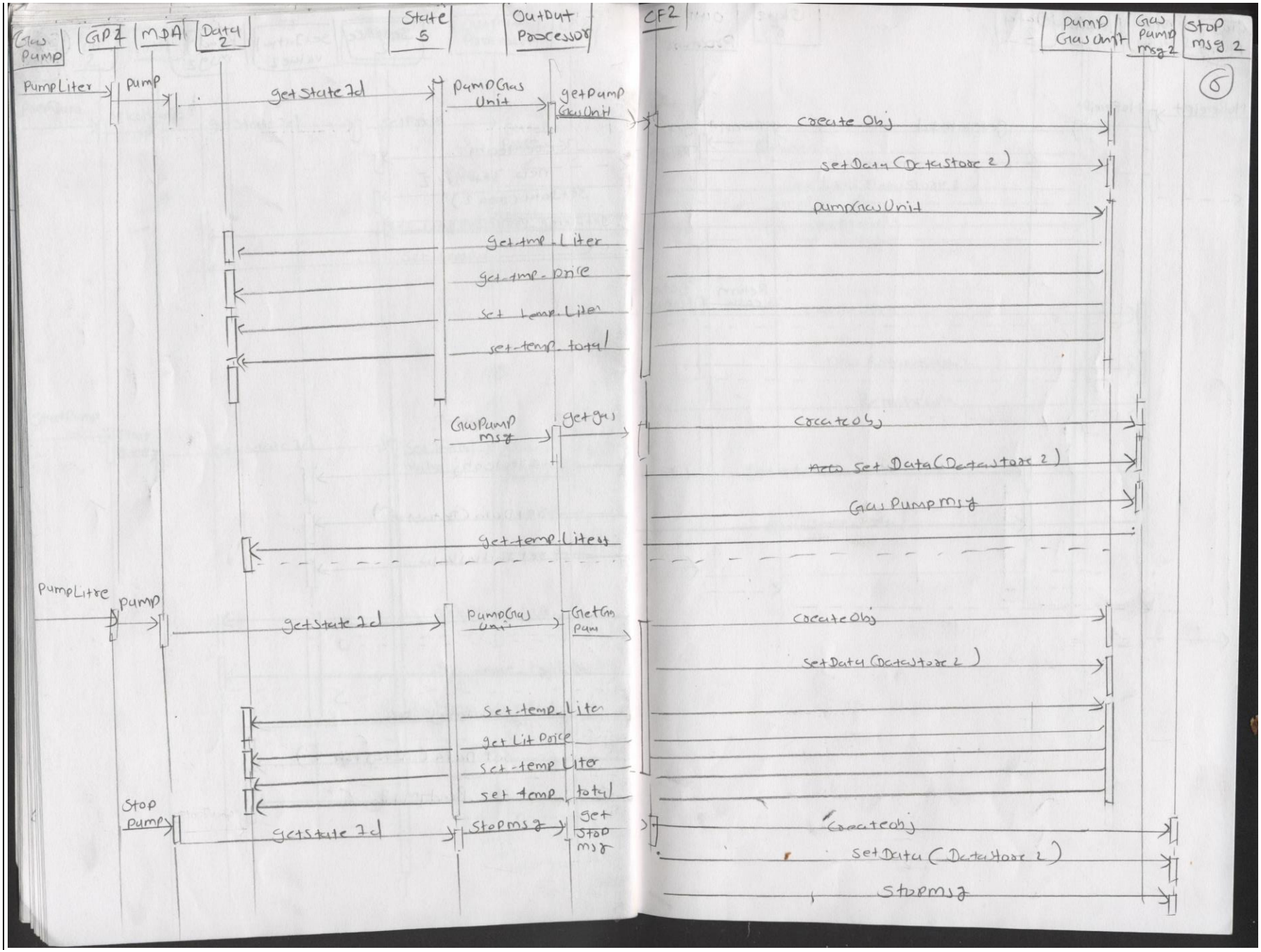
A. Scenario-I should show how one gallon of Regular gas is disposed in GasPump-1, i.e., the following sequence of operations is issued: Activate(3.1, 4.3), Start(), PayCredit(), Approved(), Regular(), StartPump(), PumpGallon(), StopPump()

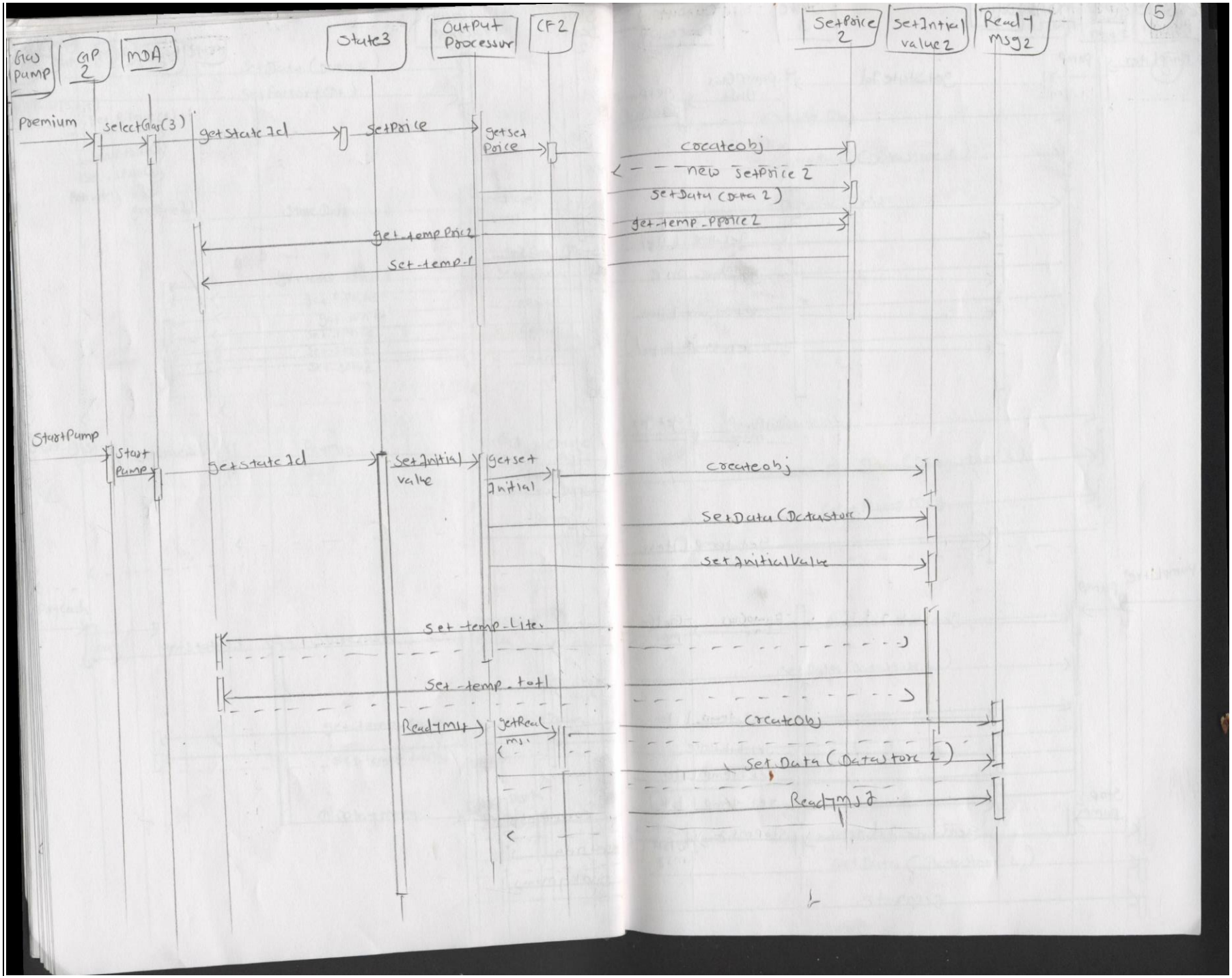

GasPump
GP-1
MDA
Data1
Start
S0
S1
S2
S3
S4
S5
S6
OP
CF1
Data1
PayMsg1
DisplayMenu1
SetPrice1
1. SetData (cls.DataStore)
2. SetFactory (CF1, AbstractFactory1)
3. set_temp_price (3.1)
4. Activate (3.1, 4.3)
8. set_temp_sprice (4.3)
10. Activate
11. getState
12. StoreData
13. getStore Data()
14. CreateObj
16
17
setData (DS1)
22. StoreData
23. set_temp_Rprice (temp_Rprice)
24. set_temp_sprice (temp_sprice)
31. start
35. Start
getState7()
37. PayMsg()
getPayMsg
39. CreateObj
43. setData (DataStore1)
45. PayMsg7
46
47
49
51
PayCredit
52. PayCl()
53. getState7()
55
57
58. Approved
Approved
60. getState7()
61. DisplayMenu
getDisplay menu()
64. CreateObj
67. setData
69. DisplayMenu
73

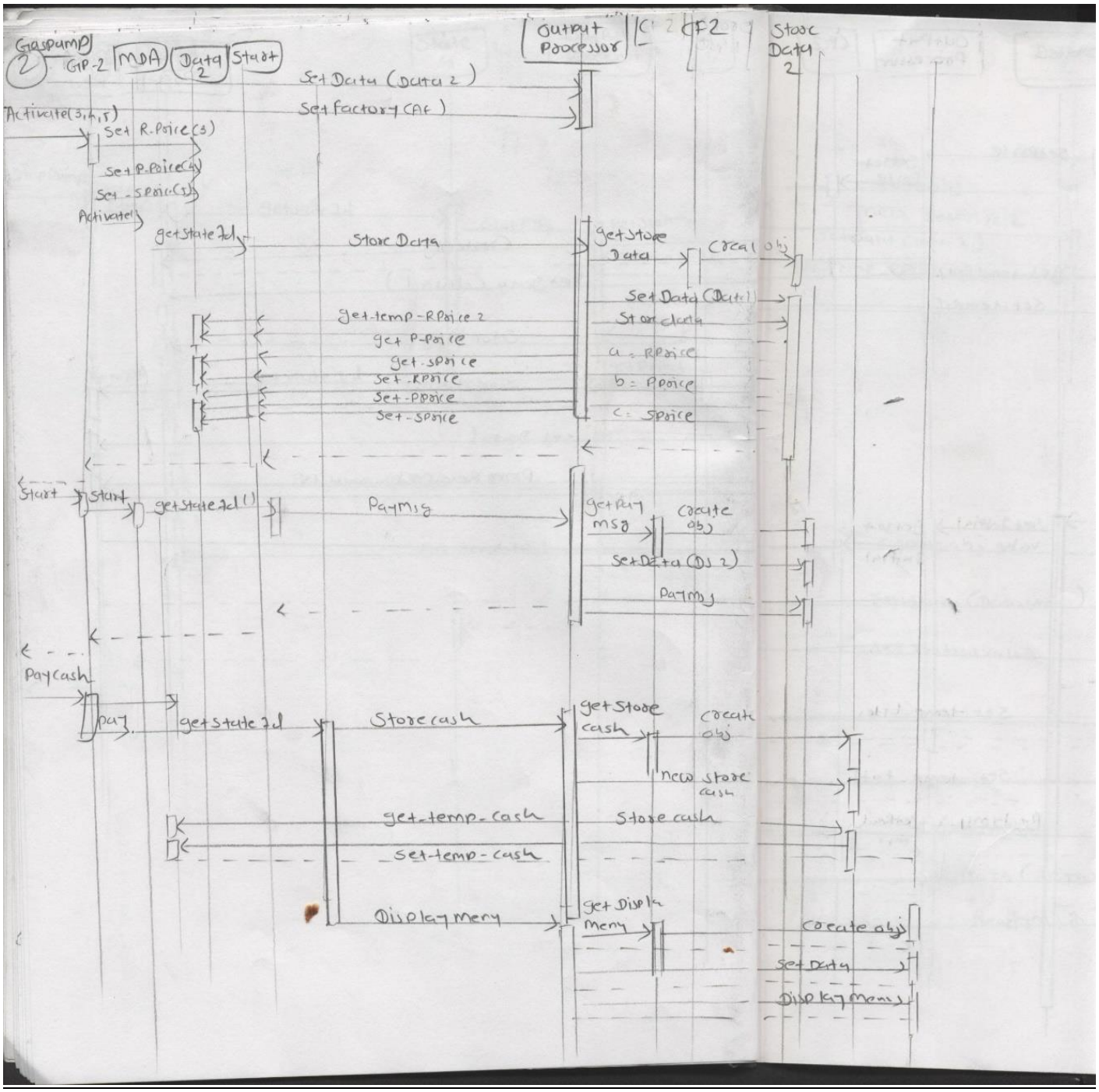

Gas Pump
GP-1
MDA
Data 1
State 3
Output Processor
CF1
SetPrice
Set Initial Value
Ready Msg
Gas Pump Msg-1
75 Regular
SelectGas(1)
getState2d
SetPrice(1)
getSetPrice
81 CreateObj
SetData(Data1)
86. SetPrice(1)
get_temp_Rprice
set_temp_Sprice
StartPump
Start Pump
getState2d
SetInitialValue
getInitial value
CreateObj
SetData(Data1)
SetInitialValue
set_temp_Gallons(0)
CreateObj
SetData(Data1)
ReadyMsg
PumpGallon
Pump
getState2d
PumpGas Unit
getPumpGas Unit
Createobj
SetData
new PumpGasUnit
PumpGasUnit
get_temp
temp_Gallons
set_temp_total
set_temp
Gallons
Gallons
Itemp_Gallons
GetPumped Msg
GetPump msg
Create Obj
SetData (Datastore1)
GasPumpedMsg
get_temp_Gallons

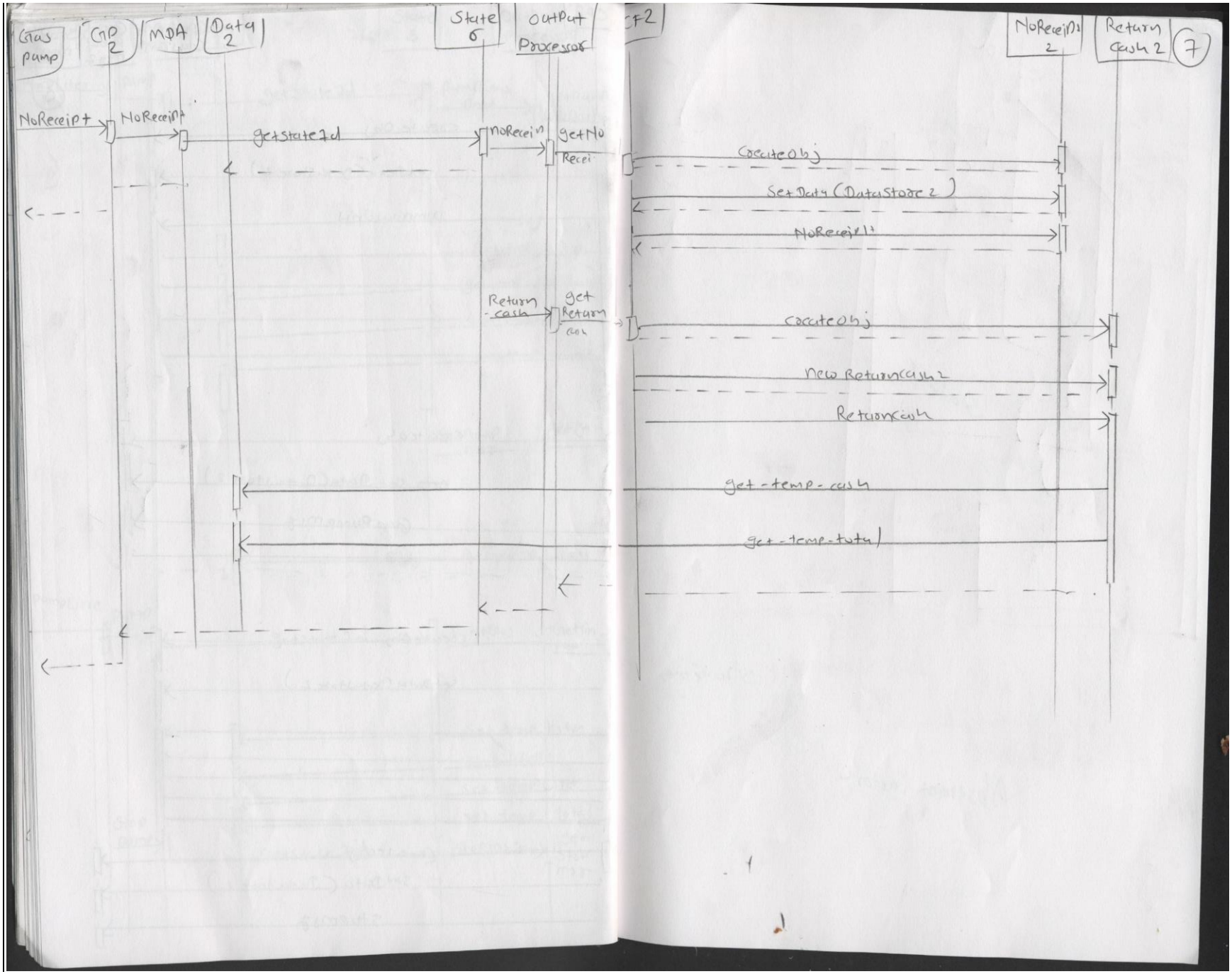

Gas Pump
GP 1
MDA
State 4
Output Processor
CF 1
Stop Msg-1
Print Receipt (3)
StopPump
stop pump
GetState Id
StopMsg
getStopMsg
CreateObj
SetData (data 1)
StopMsg
Receipt
GetState Id
Print Receipt
getPrint Receipt
CocateObj
set Data( )
PrintReceipt
Get-temp-Gallons
Get-temp-total

A. Scenario-II should show how one liter of Premium gas is disposed in GasPump-2, i.e., th following sequence of operations is issued: Activate(3, 4, 5), Start(), PayCash(6), Premium(), StartPump(), PumpLiter(), PumpLiter(), NoReceipt()


Gas Pump
GPZ
MDA
Data 2
State 5
OutDut Processor
CF2
Pump Gas Unit
Gas Pump msg 2
Stop msg 2
6
PumpLiter
pump
getStateId
PumpGas Unit
getPump GasUnit
createObj
setData(DataStore 2)
PumpGasUnit
Get-tmp-Liter
get-tmp-price
Set temp-Liter
set-temp-total
GasPump msg
getgas
createObj
Also Set Data(DataStore 2)
GasPumpMsg
Get-temp-Litess
PumpLitre
pump
getStateId
PumpGas Unit
GetGas Pum
createObj
setData(DataStore 2)
Set-temp-Liter
get Lit Price
Set-temp Liter
set temp total
Stop pump
getState Id
Stopmsg
get stop msg
createObj
setData(DataStore 2)
Stopmsg


Gas Pump
GP 2
MDA
State3
Output Processor
CF2
SetPrice 2
SetInitial value2
Ready msg2
Premium
selectGas(3)
getStateTcl
setPrice
getset Price
cocateobj
new setPrice 2
setData (Data 2)
get_temp_price2
get_temp_price2
set_temp_1
StartPump
Start Pump
getstateTcl
SetInitial value
getset Initial
cocateobj
setData (Datastore)
setInitialValue
set_temp_Litre
set_temp_total
ReadyMsg
getReady msg
createObj
setData (Datastore 2)
ReadyMsg


GasPump 2
GIP-2
MDA
Data 2
Start
Output Processor
Store Data 2
Set Data (Data 2)
Set Factory (AF)
Activate(3,4,5)
Set R.Price(3)
Set P.Price(4)
Set - Sprice(5)
Activate()
getState()
Store Data
get Store Data
Create Obj
Set Data (Data1)
get-temp-RPrice 2
get P-Price
get-Sprice
Set-RPrice
Set-PPrice
Set-Sprice
Store data
a = RPrice
b = PPrice
c = Sprice
Start
Start
getState()
PayMsg
get Pay msg
Create obj
Set Data (Obj 2)
PayMsg
PayCash
pay
getState()
Store cash
get Store cash
Create obj
new store cash
get-temp-cash
Store cash
set-temp-cash
Display Menu
get Disply Menu
Create obj
Set Data
Display Menu


Gas pump
GP 2
MDA
Data 2
State 6
Output Processor
NoReceipt
NoReceipt
getStateId
noRecein
getNo
Recei
Return cash
get Return
cash
NoReceipt 2
Return cash 2
7
CreateObj
Set Data (DataStore 2)
NoReceipt
CreateObj
new Returncash 2
Returncash
get-temp-cash
get-temp-tuta

5. Pattern Description

This part of the report should clearly indicates which parts of the source code are responsible for the implementation of the three required design patterns:
• state pattern
• strategy pattern
• abstract factory pattern.

Patterns have been outlined within various class diagrams within Section 3, and Section 5 also contains identification of which patterns each class belongs to in the source comments.

Below is an additional summary of objects and their usage;

1.State Pattern

- State Machine Context Class – me
- State Abstract Class – all_states
- State Concrete Classes – Activate, s0,s1,s2,s3,s4,s5,s6,s7
- Output (Client)Class- op

2.Strategy Pattern

Strategy Abstract Class
PayMsg
StoreCash
DisplayMenu
RejectMsg
SetPrice(int g)
ReadyMsg
SetInitialValues
PumpGasUnit
GasPumpedMsg
StopMsg
PrintReceipt
CancelMsg

3.Abstract Factory

Abstract Factory Class – AF

- Concrete Factory Classes –CF1, CF2
- Client Classes – GP1, GP2, OP

Other Classes within Implementation (7 Classes)

- Abstract Data Store Class – data
- Concrete Data Store Classes – data1,data2
- Platform Depended Input Processor Classes – GP1, GP2

- Meta Model (PIM) Class – MDAEFM
- Driver – maindriver

3. Description of all the class in the system :

a. The purpose of the class, i.e., responsibilities.
b. Responsibility of each operation supported by each class.
c. The purpose of main attributes of the class.

Purpose, Methods Class Operations and Parameters

CLASS 1

NAME:	**GP1 //Input processor of the system**
TYPE	**Concrete Class**
VARIABLES	**MDA_Object** is a pointer to the **class** which is MDAEFSM
	dataObjectis a pointer to the **Data_Base** class which is data store
	dataObjectis a pointer to the **af** class which is Abstartc FActory
METHODS	**void** setdata(Data_Base x) //sets data
	void setMDA(MDAEFSM x) // to change state according to current operation
	All other Methods // Pseudo code is called out in section 1 Model Driven Architecture .

CLASS 2

NAME:	**GP2 //Input processor of the system**
TYPE	**Concrete Class**
VARIABLES	**MDA_Object** is a pointer to the **class** which is MDAEFSM
	dataObjectis a pointer to the **Data_Base** class which is data store
	dataObjectis a pointer to the **af** class which is Abstartc FActory
METHODS	**void** setdata(Data_Base x) //sets data

void setMDA(MDAEFSM x) // to change state according to current operation

All other Methods // Pseudo code is called out in section 1 Model Driven Architecture .

CLASS 3

NAME: **MDAEFSM//** Meta Model (PIM) Class

TYPE **Concrete Class**

VARIABLES States c_state; // holds the present state
States[] ls = new States[8]; // holds the value of the state

METHODS **void** setStatesList (all_states s) // sets the current state of system

void Activate() // points towards next state when this method selected

void Start() // points towards next state when this method selected

void PayCash() // points towards next state when this method selected

void PayCredit() // points towards next state when this method selected

void Approved() // points towards next state when this method selected

void Reject() // points towards next state when this method selected

void SelectGas(int g) // points towards next state when this method this method selected

void Cancel() // points towards next state when this method selected

void StartPump() // points towards next state when this method selected

void Pump() // points towards next state when this method selected

void StopPump() // points towards next state when this method selected

void Receipt() // points towards next state when this method selected

void NoReceipt () // points towards next state when this method selected

----

CLASS 4

NAME: **ActivateStates //** GasPump is started

TYPE **Concrete Class**

VARIABLES **-**

METHODS **void** Activate ()//used for activate gaspump

CLASS 5

NAME: **s0state//** gaspump start from here

TYPE **Concrete Class**

VARIABLES **-**

METHODS **void** Start ()//System moves to idle state

CLASS 6

NAME: **s1state** // System is in ready state to perform various operations

TYPE **Concrete Class**

VARIABLES

METHODS **void** PayType (int t)//Service provided to to do payment

----

CLASS 7

NAME: **s2state //**Approve and reject the payment

TYPE **Concrete Class**

VARIABLES **-**

METHODS **void** Approved ()//Approve the payment

void Reject()//Reject the payment

CLASS 8

NAME: **s3state //** it used for cancel the payment

TYPE **Concrete Class**

VARIABLES

METHODS **void** Cancel()//cancel the payment

void SelectGas(int g)//Select gas type Regular/Premium/Super

CLASS 9

NAME: **s1 //** This denotes the sixth state of program

TYPE **Concrete Class**

VARIABLES -

METHODS **void** WithdrawBelowMinBalance()/
//Works when amount to be withdrawn in below minimum balance

CLASS 10

NAME: **s4state//**used to start the gas pump

TYPE **Concrete Class**

VARIABLES -

METHODS **void** StartPump() //It is used to start the pump

CLASS 11

NAME: **s5state//** used for stop the pump

TYPE **Concrete Class**

VARIABLES -

METHODS **void** StopPump () // stop the pump

void Pump () // used to pumpimg the gas

CLASS 12

NAME: **s6state//** used for printing receipt/noreceipt

TYPE **Concrete Class**

VARIABLES -

METHODS **void** Receipt () // print the receipt
void NoReceipt () // do not print receipt
void ReturnCash () // display remaining cash amount

CLASS 13

NAME: **States //**states class

TYPE **Concrete Class**

VARIABLES -

METHODS //ALL THE BELOW MENTIONED ARE State CLASS METHODS

```
publicvoid storevaluey(String y);   //Storethe value of y
public void Activate(){}            //Activate the gas pump
public void Start(){}               //Start the pump
public void PayCash(){}             //Pay the cash
public void PayCredit(){}           //Payment through creditcard
public void Approved(){}            //approvethe  creditcard
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(){}
public void ReturnCash(){}
```

CLASS 13

NAME: **Data_Base1**

TYPE **Concrete Class**

VARIABLES

```
static float temp_a;     //Store temporary value of a(Regular Gas)
static float temp_b;     //Store temporary value of b(Super Gas)
static int W;            //it is flag
static float price;
static float cash;
static float total;
static float G;
static float reg_price;
static float sup_price;
```

METHODS

```
public void setfloata(float a)
public void setfloatb(float b)
public float getfloata()
public float getfloatb()
public void setintc(int c)
public int getintc()
public void settotal(float c)
public float gettotal()
public void setprice(float y)
public float getprice()
public void setcash(float a)
public float getcash()
public void setG(float y)
public float getG()
public void setsup_price(float y)
public float getsup_price()
public void setreg_price(float y)
public float getreg_price()
public void setintW(int a)
public int getintW()
```

CLASS 14

NAME: **Data_Base2//** used for store data of GasPump 2

TYPE **Concrete Class**

VARIABLES

```
static float temp_a;
static float temp_b;
```

static float temp_c;
static float L;
static float reg1_price;
static float pre_price;
static float sup_price;
static float cash;
static float total;
static float price;

METHODS

static float temp_a;
static float temp_b;
static float temp_c;
static float L;
static float reg1_price;
static float pre_price;
static float sup_price;
static float cash;
static float total;
static float price;
public float getprice()
public void setprice(float y)
public void setfloata(float a)
public void setfloatb(float b)
public float getfloatb()
public float getfloata()
public void setfloatc(float c)
public float getfloatc()
public void setL(float a)
public float getL()
public void setpre_price(float a)
public float getpre_price()
public void setreg1_price(float a)
public float getreg1_price()
public void setsup_price(float a)
public float getsup_price()
public void setcash(float a)
public float getcash()
public float gettotal()
public void settotal(float a)

CLASS 15

NAME: **Data_Base//** used for stop the pump

TYPE **Abstract Class**

VARIABLES -

METHODS
static float temp_a;
static float temp_b;
static float temp_c;
static float L;
static float reg1_price;
static float pre_price;
static float sup_price;
static float cash;
static float total;
static float price;
public float getprice()
public void setprice(float y)
public void setfloata(float a)
public void setfloatb(float b)
public float getfloatb()
public float getfloata()
public void setfloatc(float c)
public float getfloatc()
public void setL(float a)
public float getL()
public void setpre_price(float a)
public float getpre_price()
public void setreg1_price(float a)
public float getreg1_price()
public void setsup_price(float a)
public float getsup_price()
public void setcash(float a)
public float getcash()
public float gettotal()
public void settotal(float a)

CLASS 16

NAME: **maindriver//** used for Handling all main functions

TYPE **Concrete Class**

VARIABLES -

METHODS **void Main() // Main methods reside inthis class**

CLASS 17

NAME: **GP1//** used for stop the pump

TYPE **Concrete Class**

VARIABLES **af afObject; //** Object of abstract factory

MDAEFSM MDA_Object; // Object of MDAEFSM class

Data_Base dataObject; // Pointer to the class Database

METHODS **PayCredit()** // Used for credit card apyment
public void setdata(Data_Base x) // set data into database class
setMDA(MDAEFSM x) // set events class MDAEFSM
setfactory(af x) // set factory

CLASS 18

NAME: **GP2//** used for stop the pump

TYPE **Concrete Class**

VARIABLES **af afObject; //** Object of abstract factory

MDAEFSM MDA_Object; // Object of MDAEFSM class

Data_Base dataObject; // Pointer to the class Database

METHODS **PayCredit()** // Used for credit card apyment
public void setdata(Data_Base x) // set data into database class
setMDA(MDAEFSM x) // set events class MDAEFSM
setfactory(af x) // set factory

CLASS 19

NAME: **s5state//** used for stop the pump

TYPE **Concrete Class**

VARIABLES -

METHODS **void** StopPump () // stop the pump
void Pump () // used to pumpimg the gas

CLASS 20

NAME: **MDAEFSM//** used for stop the pump

TYPE **Concrete Class**

VARIABLES -

METHODS NoReceipt()
Receipt(int a)
StopPump()
Pump()
StartPump()
Cancel()
SelectGas(int g)
Reject()
Approved()
PayCredit()
PayCash()
Start()
setStatesList()
setStates(States a)
Activate()

CLASS 21

NAME: **OPER//** used for Operation of the gaspump

TYPE **Concrete Class**

VARIABLES -

METHODS setPrice(int g)
storeCash()

payMsg()
setfactory(af af1)
returnCash()
setData(Data_Base d1)
storeData()
displayMenu()
cancelMsg()
rejectMsg()
printReceipt()
gasPumpedMsg()
pumpGasUnit()
readyMsg()
stopMsg()
setInitialValues()
setW(int k)
NoReceipt()

CLASS 22

NAME: **CancelMsg//** used for cancel the msg

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **cancelMsg();**
setdata(Data_Base dt)

CLASS 23

NAME: **CancelMsg1//** used for cancel msg for gaspump 1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **cancelMsg1();**

CLASS 24

NAME: **DisplayMenu//** used for display the menu

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **DisplayMenu ();**
setdata(Data_Base dt)

CLASS 25

NAME: **DisplayMenu1//** used for display the menu for gaspump 1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **displayMenu1 ();**

CLASS 26

NAME: **DisplayMenu2//** used for display the menu for gaspump 2

TYPE **Concrete Class**

VARIABLES **-**

METHODS **displayMenu2 ();**

CLASS 27

NAME: **GasPumpedMsg//** used for pump the gas

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **gasPumpedMsg ();**
setdata(Data_Base dt)

CLASS 28

NAME: **GasPumpedMsg1 //** use for pumping gaspump 1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **gasPumpedMsg1 ();**

CLASS 29

NAME: **GasPumpedMsg 2//** use for pumping gaspump 2

TYPE **Concrete Class**

VARIABLES **-**

METHODS **gasPumpedMsg ();**

CLASS 30

NAME: **NoReceipt//** used for when there is no need of receipt

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **noReceipt ();**
setdata(Data_Base dt)

CLASS 31

NAME: **NoReceipt1//** used for when there is no need of receipt for GP1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **noReceipt1();**

CLASS 32

NAME: **NoReceipt2//** used for when there is no need of receipt for GP2

TYPE **Concrete Class**

VARIABLES **-**

METHODS **noReceipt2();**

CLASS 33

NAME: **PayMsg//** used for disply pay msg the pump

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **payMsg ();**
setdata(Data_Base dt)

CLASS 34

NAME: **PayMsg 1//** used for disply pay msg the pump gaspump 1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **payMsg1();**

CLASS 35

NAME: **PayMsg2//** used for disply pay msg the pump gaspump 2

TYPE **Concrete Class**

VARIABLES **-**

METHODS **payMsg2();**

CLASS 36

NAME: **PrintReceipt//** It use for printing the receipt

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **printReceipt();**
setdata(Data_Base dt)

CLASS 36

NAME: **PrintReceipt1//** It use for printing the receipt **1**

TYPE **Concrete Class**

VARIABLES **-**

METHODS **printReceipt1 ();**

CLASS 37

NAME: **PrintReceipt2//** It use for printing the receipt **2**

TYPE **Concrete Class**

VARIABLES **-**

METHODS **printReceipt2();**

CLASS 37

NAME: **PumpGasUnit//** Used for count the pumping units the pump

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **pumpGasUnit ();**
setdata(Data_Base dt)

CLASS 38

NAME: **PumpGasUnit1** Used for count the pumping units the pump 1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **pumpGasUnit1();**

CLASS 39

NAME: **PumpGasUnit2//** Used for count the pumping units the pump 2

TYPE **Concrete Class**

VARIABLES **-**

METHODS **pumpGasUnit2();**

CLASS 39

NAME: **ReadyMsg//** used fordisplaying the ready msg

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **readyMsg ();**
setdata(Data_Base dt)

CLASS 40

NAME: **ReadyMsg1//** used for displaying the ready msg for gaspump1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **readyMsg1();**

CLASS 41

NAME: **ReadyMsg2//** used for displaying the ready msg for gaspump2

TYPE **Concrete Class**

VARIABLES **-**

METHODS **readyMsg2();**

CLASS 41

NAME: **ReturnCash//** used return the cash fro gaspump

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **returnCash ();**
setdata(Data_Base dt)

CLASS 42

NAME: **ReturnCash1// //** used return the cash fro gaspump 1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **returnCash ();**

CLASS 43

NAME: **RejectMsg//** used for reject the the payment method credit

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **rejectMsg ();**
setdata(Data_Base dt)

CLASS 44

NAME: **RejectMsg1 //** used for reject the the payment method credit

TYPE **Concrete Class**

VARIABLES **-**

METHODS **rejectMsg ();**

CLASS 45

NAME: **SetInitialValues//** used set the initials value

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **setInitialValues ();**

setdata(Data_Base dt)

CLASS 46

NAME: **SetInitialValues1**// used set the initials value for gaspump 1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **setInitialValues();**

CLASS 47

NAME: **SetInitialValues2**// used set the initials value for gaspump 2

TYPE **Concrete Class**

VARIABLES **-**

METHODS **setInitialValues();**

CLASS 48

NAME: **SetPayType**// used to set the pay type(credit/cash)

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **setPayType ();**
setdata(Data_Base dt)

CLASS 48

NAME: **SetPayType1**// used to set the pay type(credit/cash)

TYPE **Concrete Class**

VARIABLES **-**

METHODS **setPayType ();**

CLASS 49

NAME: **SetPayType2//** used for payment method cash

TYPE **Concrete Class**

VARIABLES **-**

METHODS **setPayType ();**

CLASS 50

NAME: **StoreData//** used for stor the data

TYPE **Abstract Class**

VARIABLES **-Data_Base data_Object;**

METHODS **storeData();**
setdata(Data_Base dt)

CLASS 51

NAME: **StoreData1//** used for store the data for gaspump 1

TYPE **Concrete Class**

VARIABLES **-**

METHODS **storeData();**

CLASS 52

NAME: **StoreData2//** used for store the data for gaspump 2

TYPE **Concrete Class**

VARIABLES **-**

METHODS **storeData();**