Abinaya Janakan
A20376287

CS586 PROJECT

GasPump-1MDA-EFSM Events:	**MDA-EFSM Actions:** // responsibilities

Activate(float a, float b)
Start()
PayCredit()
Approved()
Reject()
Cancel()
Regular()
Super()
StartPump()
PumpGallon()
StopPump()
	PayMsg // displays payment type
	RejectMsg // displays reject message
	CanceMsg // displays the cancellation message
	DisplayMenu // displays menu with the type and price of the gas
	Display(S) // displays the maount of gasoline disposed
	Print Receipt (total) // prints receipt

GasPump-2 **MDA-EFSM Events:**	**MDA-EFSM Actions:**

Activate(int a, int b, int c)
Start()
PayCash(int c)
Cancel()
Regular()
Super()
Premium()
StartPump()
PumpLiter()
Stop()
NoReceipt()
Receipt()
	PayMsg // displays payment type
	CancelMsg // displays cancellation message
	DisplayMenu // displays menu with the type of gasoline
	ReturnCash(cash-total) // returns the balance cash
	PrintReceipt (L, total) // prints receipt
	Display(L) // displays the litre of gas disposed

Abinaya Janakan
A20376287

MDA-EFSM EVENTS FOR GAS PUMPS:	**MDA-EFSM ACTIONS FOR GAS PUMPS:**

Activate()
Start()
PayCredit()
Approved()
StartPump()
Pump()
StopPump()
Reject()
Cancel()
Selectgas(int x)
Receipt()
NoReceipt()
PayCash()
	StoreData //Stores the price of the gas.
	StoreCash() // Stores the cash value
	PayMsg // Display payment method
	DisplayMenu // display menu with the types of gas and their price
	SetS(int a) // set the value credit balance
	ReadyMsg() // displays the ready message
	SetinitialValues // set g to 0
	SetPrice(int p) // set the price of the gas selected
	PumpGas // pumps gas unit and counts the number of units disposed
	Message // Displays the message stating the amount of gas pumped
	CancleMsg/ dispalys cancel message
	StopMsg // dispalys the stop pump message
	PrintReceipt // Prints the receipt

Pseudo code for the Gas pump1:	**Pseudo code for the Gas pump2:**

ds: data store
m: pointer to MDA-EFSM object
 	
Activate(float a, float b) {
if ((a>0)&&(b>0))
{
ds->tempa = a;
ds->tempb = b;
m->Activate()
}
}
 	
Start() {
m->Start();
}
Reject() {
m->Reject();
}
Super() {
	ds: data store
	m: pointer to MDA-EFSM object
	L: number of liters of gasoline pumped.
	Cash: amount of cash deposited
	Price:price of the gasoline selected
	Data store stores the value of cash,L,price
	
	Activate(int a, int b, int c) {
	if ((a>0) && (b>0) && (c>0)) {
	ds->temp_a=a;
	ds->temp_b=b;
	ds->temp_c=c;
	m->Activate()
	}
	}
	Start() {
	m->Start();
	}
	
	StartPump() {
	m->StartPump();

```
m->SelectGas(2)
}


PayCredit() {
m->PayCredit();
}
Cancel() {
m->Cancel();
}
StartPump() {
m->StartPump();
}
Approved() {
m->Approved();
}


PumpGallon() {
m->Pump();

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

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

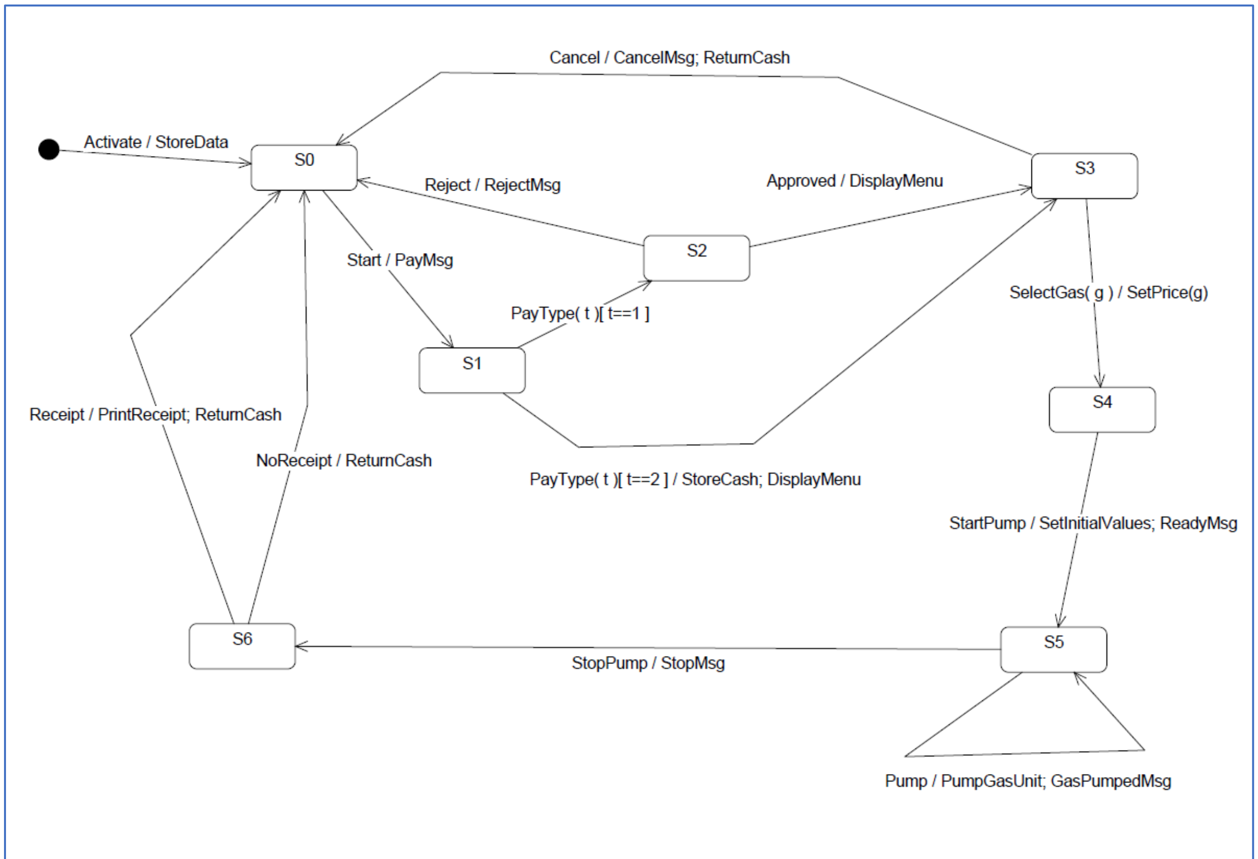
```
}
Cancel() {
m->Cancel();
}
Regular() {
m->SelectGas(1);
}
PayCash(int c)
{
if (c>0) {
ds->temp_c=c;
m->PayCash()
}
}
Receipt() {
m->Receipt();
}
PumpLiter() {
if (ds->cash<(ds->L+1)*ds->price)
        m->Stop();
else m->Pump()
}
Super(){
m->SelectGas(3);
}

Premium() {
m->SelectGas(2);
}
Stop() {
m->StopPump();
}

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

Abinaya Janakan
A20376287

MDA EFSM FOR THE GAS PUMPS:


Cancel / CancelMsg; ReturnCash
Activate / StoreData
S0
Reject / RejectMsg
Approved / DisplayMenu
S3
S2
Start / PayMsg
SelectGas(g) / SetPrice(g)
PayType(t)[t==1]
S1
S4
Receipt / PrintReceipt; ReturnCash
NoReceipt / ReturnCash
PayType(t)[t==2] / StoreCash; DisplayMenu
StartPump / SetInitialValues; ReadyMsg
S6
S5
StopPump / StopMsg
Pump / PumpGasUnit; GasPumpedMsg

MDA-EFSM for Gas Pumps

pump/pumpgas/message

Abinaya Janakan
A20376287

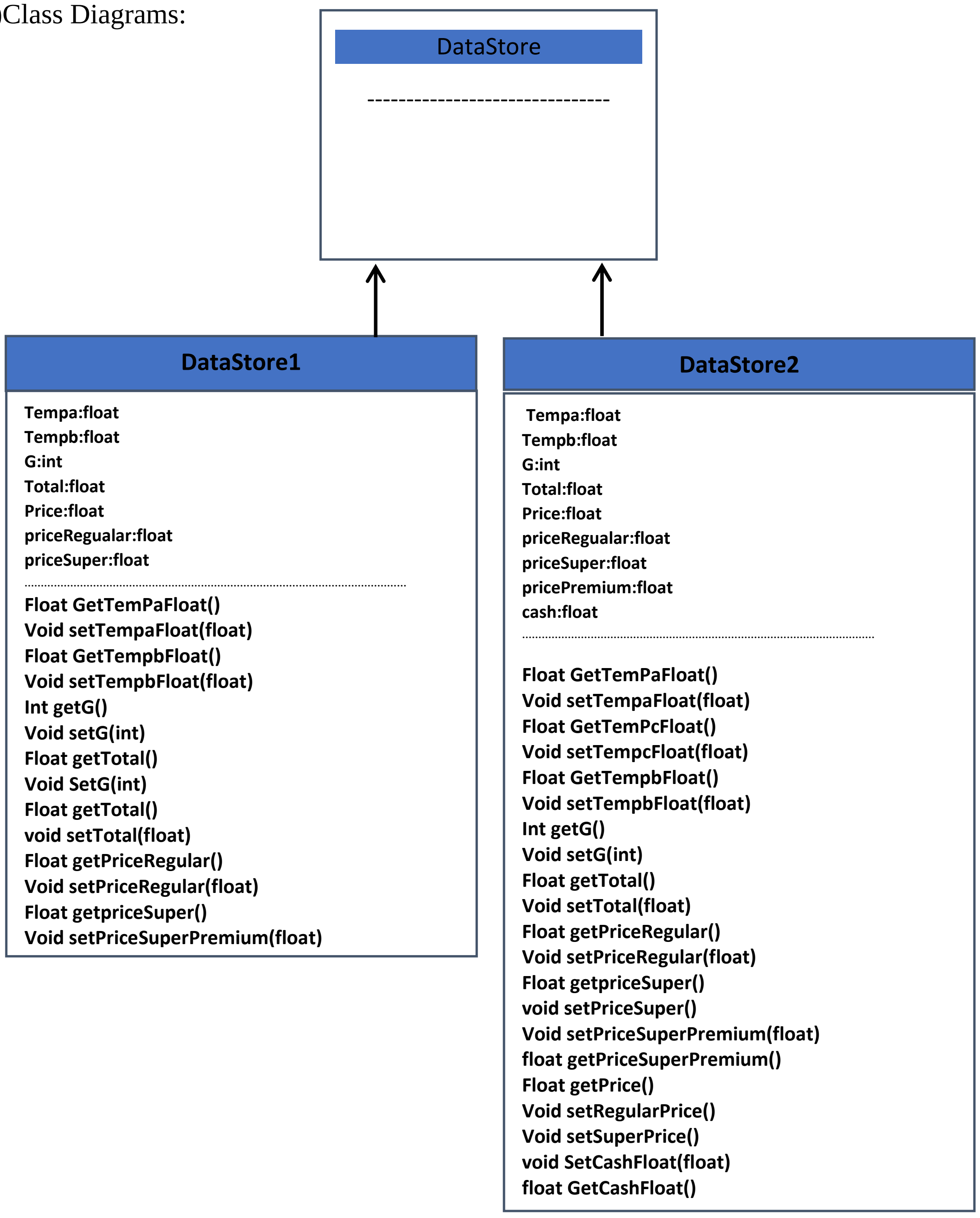
Introduction:
This project is based on the course work of CS586 (Software System Architecture). This project is implemented using three design patterns namely
a. Strategy Pattern
b. Abstract Factory Pattern
c. State Pattern

Strategy Pattern:
The **strategy pattern** is a software design pattern that enables an algorithm's behavior to be selected at runtime. The strategy pattern
- Defines a family of algorithms,
- Encapsulates each algorithm, and
- Makes the algorithms interchangeable within that family.

Abstract Factory Pattern:
Abstract Factory pattern is an interface is responsible for creating a factory of related objects without explicitly specifying their classes.
State Pattern:
State Pattern is used to encapsulate varying behavior for the same object based on its internal state. This can be a cleaner way for an object to change its behavior at runtime.

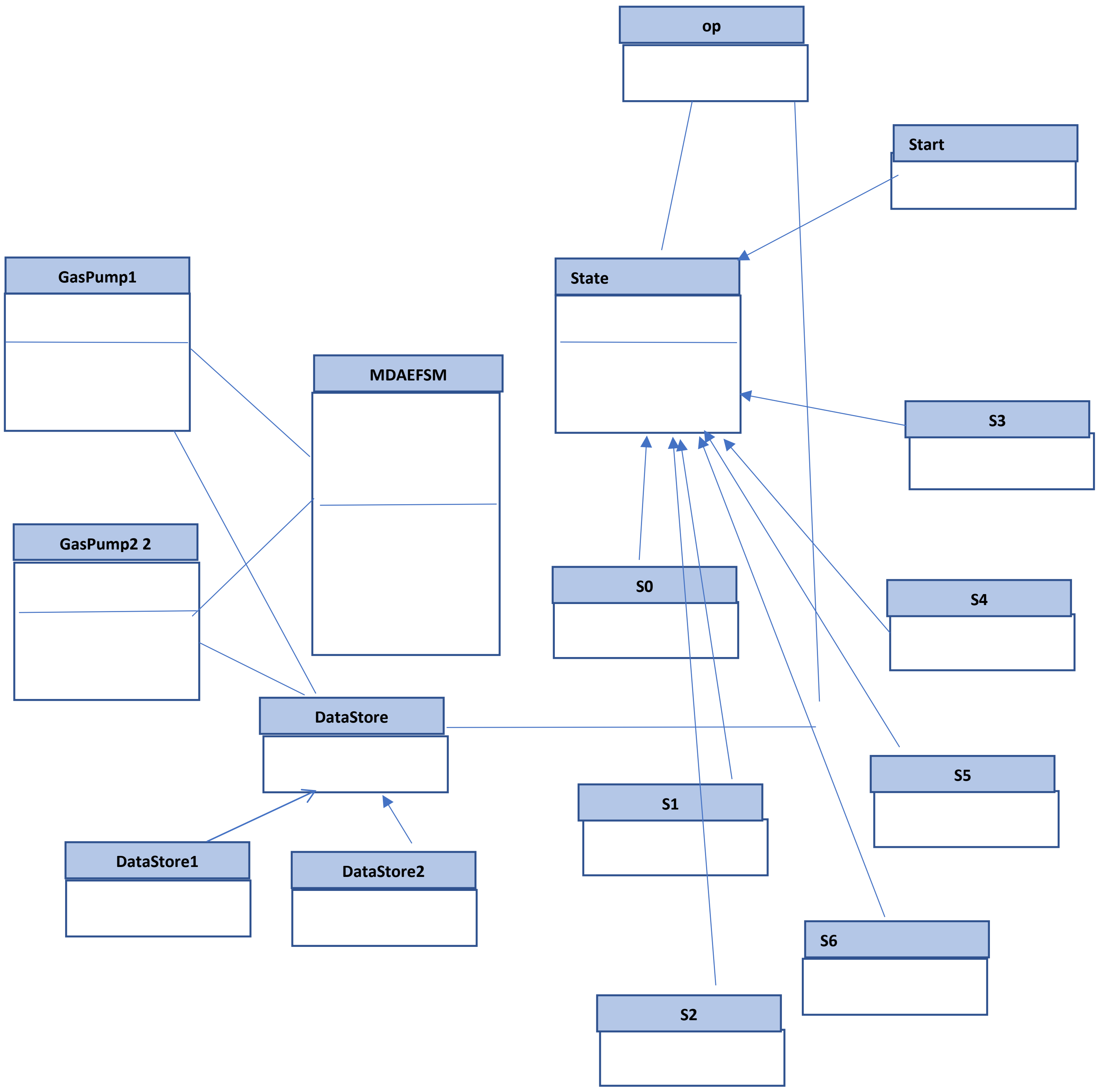
Abinaya Janakan
A20376287

2)Class Diagrams:

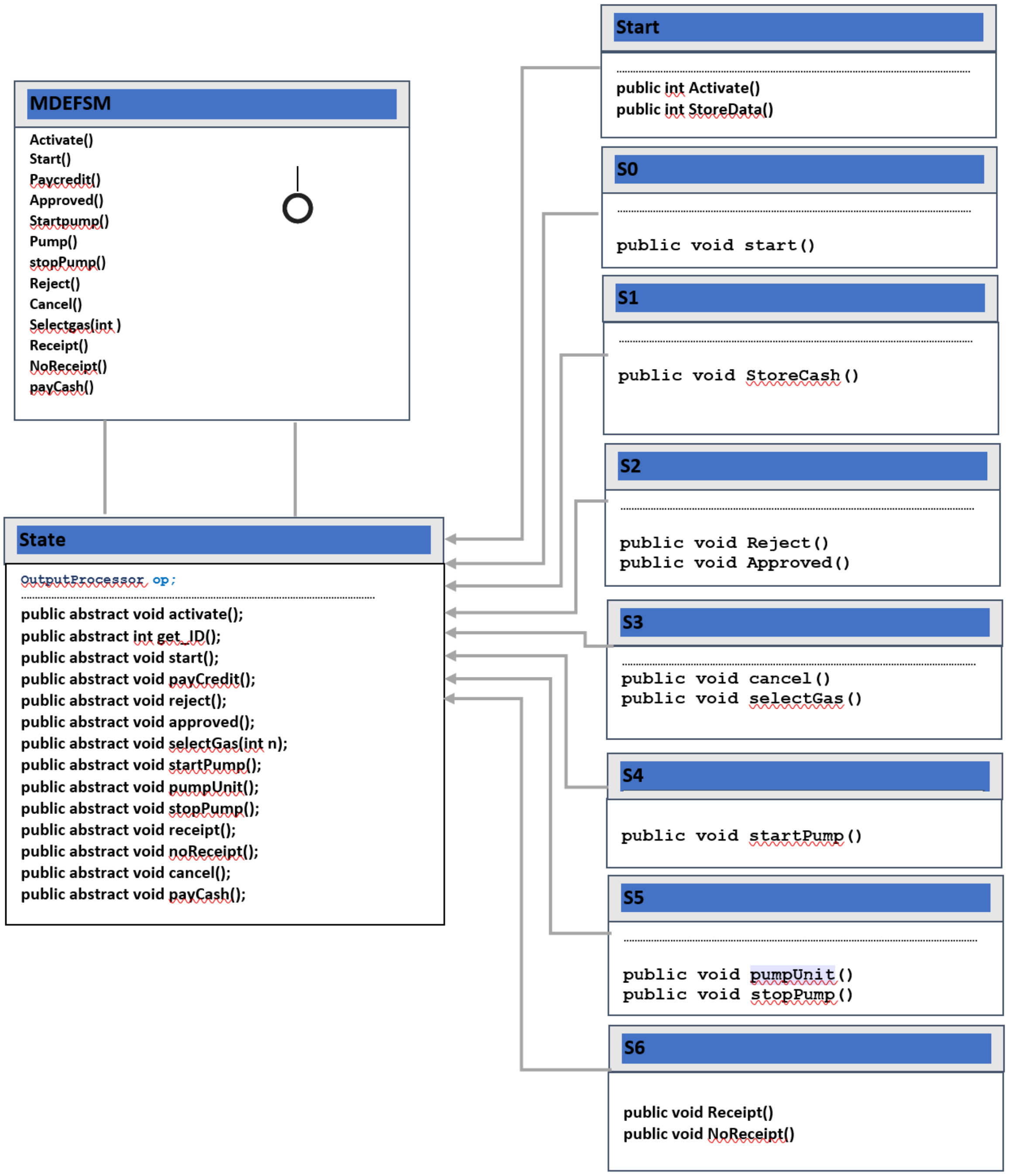


Abinaya Janakan
A20376287

State Pattern:

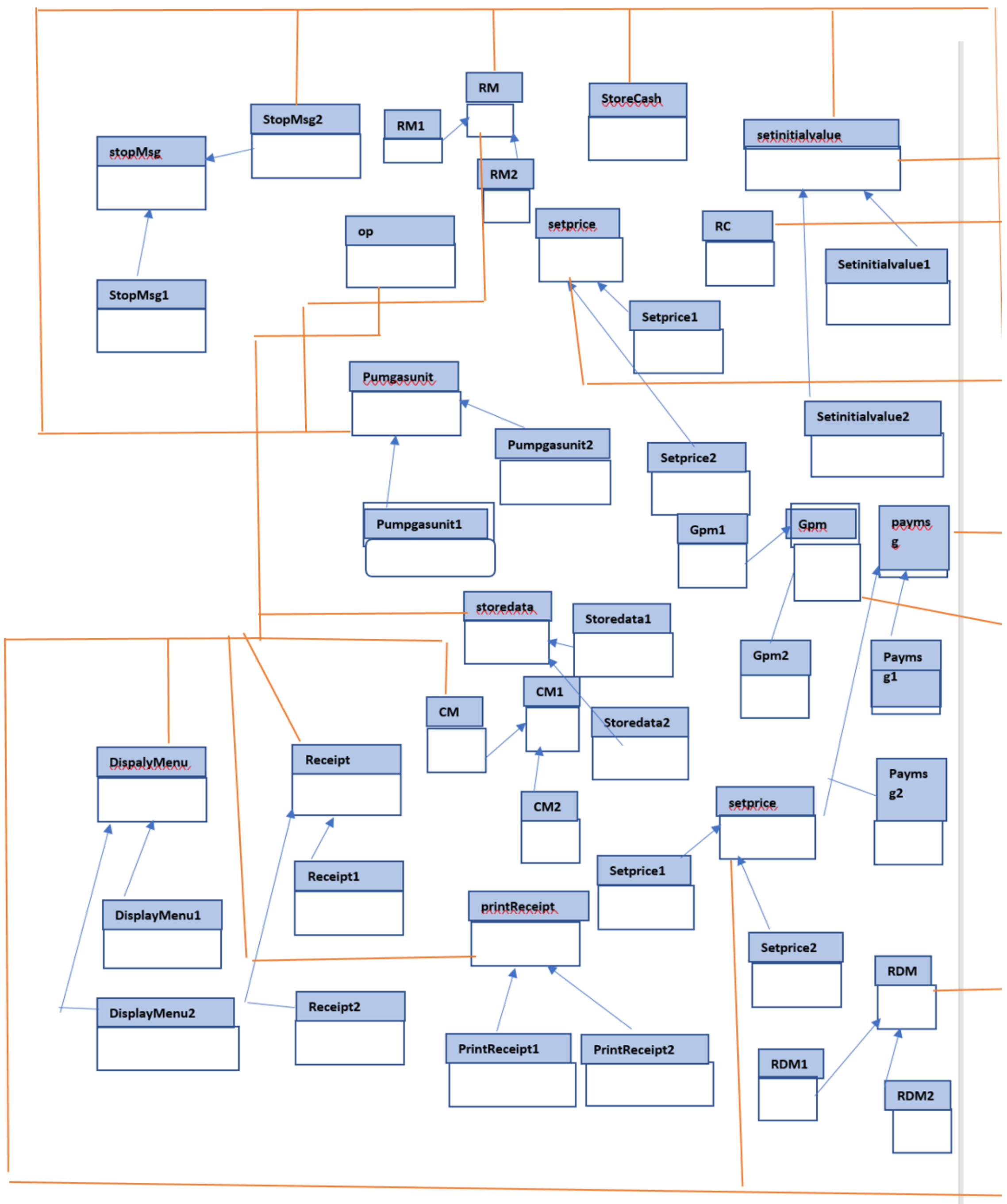


Abinaya Janakan
A20376287

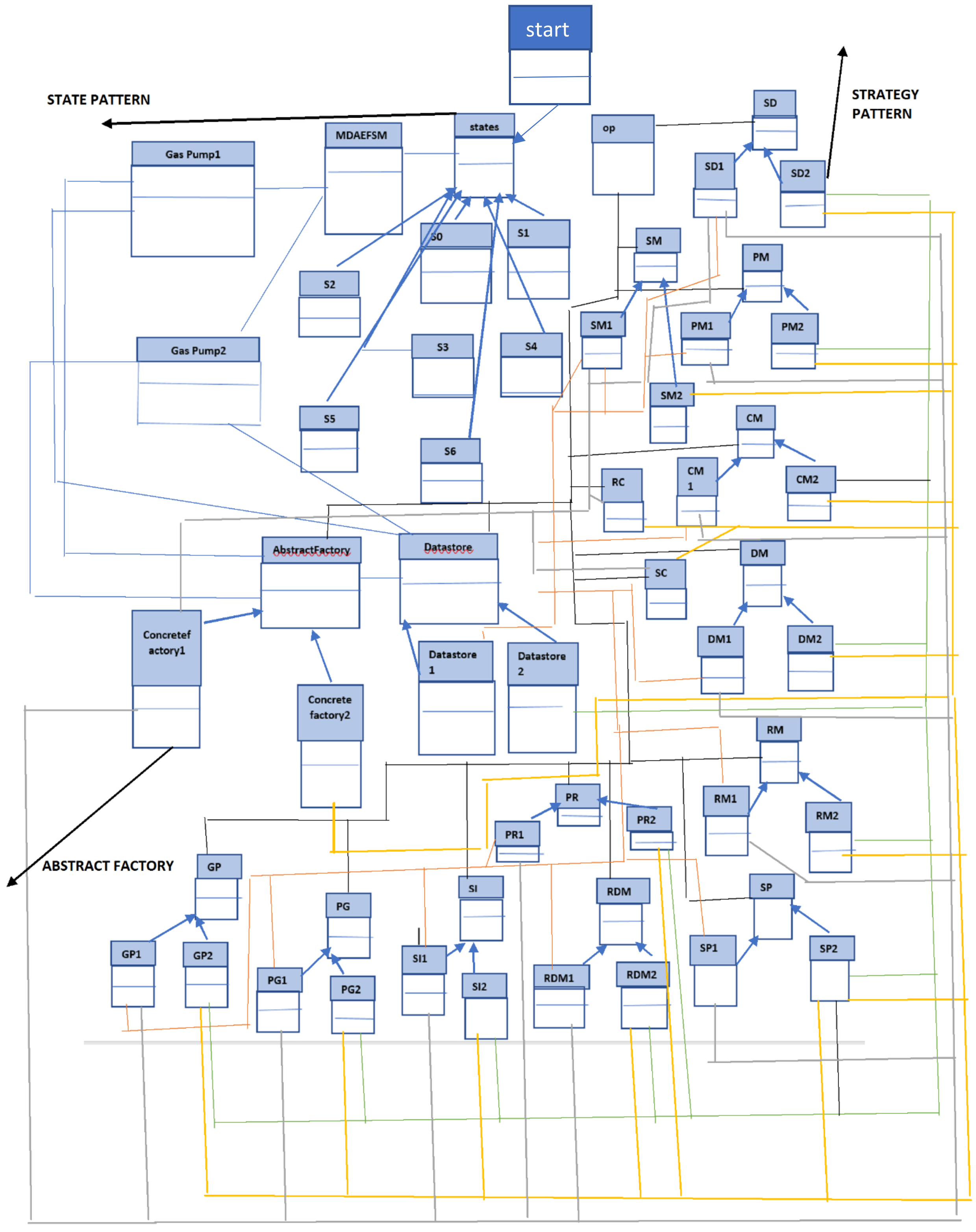


Abinaya Janakan
A20376287

Strategy Pattern:



Abinaya Janakan
A20376287



Abinaya Janakan
A20376287

SD:StoreData
- SD1:StoreData1
- SD2:StoreData2

PM:PayMsg
- PM1:PayMsg1
- PM2:PayMsg2

CM: CancelMsg1
- CM1:CancelMsg1
- CM2: CancelMsg2

DM:DisplayMenu
- DM1: DisplayMenu1
- DM2: DisplayMenu2

RM:RejectMsg
- RM1: RejectMsg1
- RM2: RejectMsg2

SP:SetPrice
- SP1: SetPrice1
- SP2: SetPrice2

RDM: ReadyMsg1
- RDM1:ReadyMsg1
- RDM2: ReadyMsg2

SI:SetInitialVal
- SI1: SetInitialVal1
- SI2: SetInitialVal2

PG:PumpGas
- PG1: PumpGas1

PG2: PumpGas2

GP:GasPumpedMsg

GP1: GasPumpedMsg1

GP2: GasPumpedMsg2

SM:StopMsg

SM1: StopMsg1

SM2: StopMsg2

RC:ReturnCash

SC:StoreCash

PR:PrintReceipt

PR1:PrintReceipt1

PR2: PrintReceipt

3)Purpose functions and attributes for the classes:

Gaspumps:

Class GasPump1: **Purpose and Attributes:** **gasPump1(MDAEFSM m,Datastore ds):**	This class represents GasPump1 implementation m: is a pointer to the MDA-EFSM object ds: is a pointer to the Data Store object Stores object of MDAEFSM in m and Object of Datastore in ds

Activate(float a, float b)	**// stores values of a and b in temp_a and temp_b**
	{
	if ((a>0)&&(b>0)) {
	ds->setTempaFloat=a;
	ds->setTempbFloat=b;
	m->Activate()
	}

Start()
PayCredit()
Approved()
	m->Start(); //invokes start
	m->PayType(1); // calls PayType1()
	m->Approved(); // calls Approved()
	m->Reject() //calls reject()

Reject()
Cancel()
Approved()
 	
Regular()
Super()
 	
StartPump()
PumpGallon()
StopPump()
	m->Cancel(); //calls cancel()
	m->Approved(); // calls approved()
	m->SelectGas(1) // selects gas type 1
	m ->SelectGas(2) // selects gas type 2
	m->StartPump(); // invokes startpump
	m->Pump() // invokes pump
	m->StopPump(); //stops pump and prints receipt
	m->Receipt();

Class GasPump2:
Purpose and Attributes:
 	
gasPump2(MDAEFSM m,Datastore ds):
 	
Activate(int a, int b, int c)
 	
Start()
PayCash(float c)
 	
Cancel()
Super()
Premium()
Regular()
StartPump()
PumpLiter()
	
	This class represents GasPump2 implementation
	m: is a pointer to the MDA-EFSM object
	ds: is a pointer to the Data Store object
	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
	if ((a>0)&&(b>0)&&(c>0)) {
	ds->setTempaInt=a; //stores values of a,b,c
	ds->setTempbInt=b; in temp_a;temp_b
	ds->setTempcInt=c and temp_c
	m->Activate()
	
	m->Start();
	if (c>0) {
	ds->temp_cash=c;
	m->PayType(2)
	
	m->Cancel();
	m->SelectGas(2);
	m->SelectGas(3);
	m->SelectGas(1);
	m->StartPump();
	if (d->cash<(d->L+1)*d->price)

Stop()
Receipt()
NoReceipt()
	m->StopPump();
	else m->Pump()
	m->StopPump();
	m->Receipt();
	m->NoReceipt();

State pattern:

Class MDAEFSM **Attributes and purpose:**	This class implements the common functionalities among the two Gas Pumps Af is an object of Abstract Factory class Op is an object of OP class.
MDAEFSM	It is a constructor it performs state changes.It consists of a list of states and a pointer to the current state.
Activate()	Call state function activate()
Start()	Call state function start()
PayType(int t) //credit: t=1; cash: t=2	Call state function PayType()
Reject()	Call state function Reject()
Cancel()	Call state function cancel()
Approved()	Call state function Approved()
StartPump()	Call state function StartPump()
Pump()	Call state function Pump()
StopPump()	Call state function StopPump()
SelectGas(int g)	Call state function SelectGas()
Receipt()	Call state function Receipt()
NoReceipt()	Call state function NoReceipt()
SetStates()	Sets MDAEFSM state to current state
GetState()	Returns object of current state

Class start

Attributes and purpose:	Start() is a constructor.This is the concrete class for the start class
Activate()	Gets the object of S0 from MDAEFSM sets the current state to s0
StoreData()	Stores the values of a, b and c in the output processor

Class s0

Attributes and purpose:	Start() is a constructor.This is the concrete class for the start class

Start()	Displays the pay message.Gets the s1 object from MDAEFSM and sets the state to s1

Class s1

Attributes and purpose:
 	
PayType(t)
 	
Storecash()
	Start() is a constructor.This is the concrete class for the start class
	If t==1
	Call Paycredit()
	Get S2 state object from the MDAEFSM and set the state to S2
	Elseif t==1
	Call Paycash()
	DisplayMenu
	Get S3 state object from the MDAEFSM and set the state to S3
	store cash value

Class s2

Attributes and purpose:
 	
Approved()
Reject()
	Start() is a constructor.This is the concrete class for the start class
	Dispalymenu
	Get S3 state object from the MDAEFSM and set the state to S3.
	Display Reject message
	Get S0 state object from the MDAEFSM and set the state to S0

Class s3

Attributes and purpose: **Cancel()**	Start() is a constructor.This is the concrete class for the start class Display cancel message

Selectgas(int g)	Set the price as the value equal to g Get S4 state object from the MDAEFSM and set the state to S4.

Class s4

Attributes and purpose: **startPump()**	Start() is a constructor.This is the concrete class for the start class Sets initial values and displays ready message. Get S5 state object from the MDAEFSM and set the state to S5.

Class s5

Attributes and purpose:
 	
Pump()
 	
StopPump()
	Start() is a constructor.This is the concrete class for the start class
	Pumps gas unit
	Display gas pumped message
	Display stop message.
	
	Get S6 state object from the MDAEFSM and set the state to S06

Class s6

Attributes and purpose:
 	
Receipt()
NoReceipt()
	Start() is a constructor.This is the concrete class for the start class
	Prints receipt.
	Get S0 state object from the MDAEFSM and set the state to S0.

Strategy Pattern:

Class StoreData:

Attributes and purpose	This class represents the abstract Factory class of action StoreData. ds is an object to the DataStore.

StoreData()	Abstract method

Class StoreData1:

Attributes and purpose	This class represents the concrete class for action StoreData for the gaspump1 ds is an object to the DataStore.
StoreData()	stores price(s) for the gas from the temporary data store

Class StoreData2:

Attributes and purpose	This class represents the concrete class for action StoreData for the gaspump2 ds is an object to the DataStore.
StoreData()	stores price(s) for the gas from the temporary data store

Class PayMsg:

Attributes and purpose	This class represents the abstract Factory class of action PayMsg. ds is an object to the DataStore.
payMsg()	Abstract method

Class PayMsg1:

Attributes and purpose	This class represents the concrete class for action payMsg1()for the gaspump1 ds is an object to the DataStore.
payMsg()	displays a type of payment method

Class PayMsg2:

Attributes and purpose	This class represents the concrete class for action payMsg2()for the gaspump2 ds is an object to the DataStore.
payMsg()	displays a type of payment method

Class CancelMsg:

Attributes and purpose	This class represents the abstract Factory class of action StoreData. ds is an object to the DataStore.
CancelMsg()	Abstract method

Class CancelMsg1:

Attributes and purpose	This class represents the concrete class for action payMsg1()for the gaspump1 ds is an object to the DataStore.
CancelMsg()	stores price(s) for the gas from the temporary data store

Class CancelMsg2:

Attributes and purpose	This class represents the concrete class for action payMsg1()for the gaspump1 ds is an object to the DataStore.
CancelMsg()	stores price(s) for the gas from the temporary data store

Class DisplayMenu:

Attributes and purpose	This class represents the abstract Factory class of action StoreData. ds is an object to the DataStore.
DisplayMenu1()	Abstract method

Class DisplayMenu1:

Attributes and purpose **DisplayMenu1()**	This class represents the concrete class for action payMsg1()for the gaspump1 ds is an object to the DataStore. stores price(s) for the gas from the temporary data store

Class DisplayMenu2:

Attributes and purpose **DisplayMenu1()**	This class represents the concrete class for action payMsg1()for the gaspump1 ds is an object to the DataStore. stores price(s) for the gas from the temporary data store

Class RejectMsg:

Attributes and purpose **RejectMsg()**	This class represents the abstract Factory class of action StoreData. ds is an object to the DataStore. Abstract method

Class RejectMsg1:

Attributes and purpose **RejectMsg()**	This class represents the concrete class for action payMsg1()for the gaspump1 ds is an object to the DataStore. stores price(s) for the gas from the temporary data store

Class RejectMsg2:

Attributes and purpose	This class represents the concrete class for action payMsg1()for the gaspump1 ds is an object to the DataStore.
RejectMsg()	stores price(s) for the gas from the temporary data store

Class SetPrice():

Attributes and purpose	This class represents the abstract Factory class of action SetPrice. ds is an object to the DataStore.
SetPrice()	Abstract method

Class SetPrice1():

Attributes and purpose	This class represents the concrete class for action payMsg1()for the gaspump1 ds is an object to the DataStore.
SetPrice()	stores price(s) for the gas from the temporary data store

Class SetPrice2():

Attributes and purpose	This class represents the concrete class for action payMsg1()for the gaspump1 ds is an object to the DataStore.
SetPrice()	stores price(s) for the gas from the temporary data store

Class ReadyMsg():

Attributes and purpose	This class represents the abstract Factory class of action ReadyMsgs. ds is an object to the DataStore.

ReadyMsg()	Abstract method

Class ReadyMsg1():

Attributes and purpose **ReadyMsg()**	This class represents the concrete class for action ReadyMsg1()for the gaspump1 ds is an object to the DataStore. displays the ready for pumping message

Class ReadyMsg2():

Attributes and purpose **ReadyMsg()**	This class represents the concrete class for action ReadyMsg2()for the gaspump2 ds is an object to the DataStore. displays the ready for pumping message

Class SetInitialVal():

Attributes and purpose **SetInitialVal()**	This class represents the abstract Factory class of action SetInitialVal. ds is an object to the DataStore. Abstract method

Class SetInitialVal1():

Attributes and purpose **SetInitialVal()**	This class represents the concrete class for action SetInitialVal1()for the gaspump1 ds is an object to the DataStore. set G (or L) and total to 0

Class SetInitialVal2():

Attributes and purpose	This class represents the concrete class for action SetInitialVal2()for the gaspump2

SetInitialVal()	ds is an object to the DataStore. set G (or L) and total to 0

Class PumpGas():

Attributes and purpose **pumpgas()**	This class represents the abstract Factory class of action Pumpgas. ds is an object to the DataStore. Abstract method

Class PumpGas1():

Attributes and purpose **pumpgas()**	This class represents the concrete class for action PumpGas()for the gaspump1 ds is an object to the DataStore. disposes unit of gas and counts # of units disposed

Class PumpGas2():

Attributes and purpose **pumpgas()**	This class represents the concrete class for action pumpGas()for the gaspump2 ds is an object to the DataStore. disposes unit of gas and counts # of units disposed

Class GasPumpedMsg():

Attributes and purpose **GasPumpedMsg()**	This class represents the abstract Factory class of action GasPumpedMsg. ds is an object to the DataStore. Abstract method

Class GasPumpedMsg1():

Attributes and purpose **GasPumpedMsg()**	This class represents the concrete class for action GasPumpedMsg()for the gaspump1 ds is an object to the DataStore. displays the amount of disposed gas

Class GasPumpedMsg2():

Attributes and purpose **GasPumpedMsg()**	This class represents the concrete class for action gasPumpedMsg()for the gaspump1 ds is an object to the DataStore. displays the amount of disposed gas

Class stopMsg():

Attributes and purpose **stopMsg()**	This class represents the abstract Factory class of action StopMsg(). ds is an object to the DataStore. Abstract method

Class stopMsg1():

Attributes and purpose **StopMsg()**	This class represents the concrete class for action StopMsg()for the gaspump1 ds is an object to the DataStore. stop pump message and printss receipt

Class stopMsg2():

Attributes and purpose **StopMsg()**	This class represents the concrete class for action stopMsg()for the gaspump2 ds is an object to the DataStore. stop pump message and receipt? msg (optionally)

Class ReturnCash():

Attributes and purpose	This class represents the concrete class for action Returncash()

Returncash()	ds is an object to the DataStore. stores price(s) for the gas from the temporary data store

Class StoreCash():

Attributes and purpose	This class represents the concrete class for action storecash() ds is an object to the DataStore.
StoreCash()	stores cash from the temporary data store

Class PrintReceipt():

Attributes and purpose	This class represents the abstract Factory class of action PrintReceipt. ds is an object to the DataStore.
printReceipt()	Abstract method

Class PrintReceipt1():

Attributes and purpose	This class represents the concrete class for action PrintReceipt()for the gaspump1 ds is an object to the DataStore.
printReceipt()	print a receipt

Class PrintReceipt2():

Attributes and purpose	This class represents the concrete class for action PrintReceipt()for the gaspump2
printReceipt()	print a receipt

Class OutputProcessor

Purpose and Attributes	This class represents the Output processor of the MDA. af is an object the Abstract Factory class ds is an object of the data store.
Output()	Constructor
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

Abstract Factory Pattern

Class AbstractFactory

Purpose and Attributes	This class represents the Abstract class for factory that groups together different classes of the Accounts
AbstractFactory	Create objects for all the MDAEFSM actions

Class ConcreteFactory1

Purpose and Attributes	This class represents the concrete class for the GasPump1's factory and is used to handle the creation of class objects specific for GasPump1
Actions	

	Create objects to every Strategy Classes for Gaspump1 Return the objects of every Strategy Class for GasPump1

Class ConcreteFactory2

Purpose and Attributes **Actions**	This class represents the concrete class for Gaspump2's factory and is used to handle the creation of class objects specific for gaspump2. Create objects to every Strategy Classes for gaspump2. Return the objects of every Strategy Class for gaspump2

DataStore

Class datastore:

Purpose and attributes	This is abstract class for the data store It is extended by DataStore1 and DataStore2

Class Datastore1: (gaspump1)

Purpose and attributes **Variables** **Functions** **Float GetTemloaaFloat()** **Void setTempaFloat(float)** **Float GetTempbFloat()** **Void setTempbFloat(float)** **Int getG()** **Void setG(int)**	This class represents the concrete class for the Data Store used for GasPump1. **Temporary variables** Tempa // stores float a variable Tempb // stores float b variable **Permanent variables** G // static int variable (gas type variable) Total // static float Price // static float priceRegualar // static float priceSuper // static float

Float getTotal() **Void setTotal(float)** **Float getPriceRegular()** **Void setPriceRegular(float)** **Float getPriceSuper()** **Void setPriceSuperPremium(float)** **Float getPrice()** **Void setRegularPrice()** **Void setSuperPrice()**	

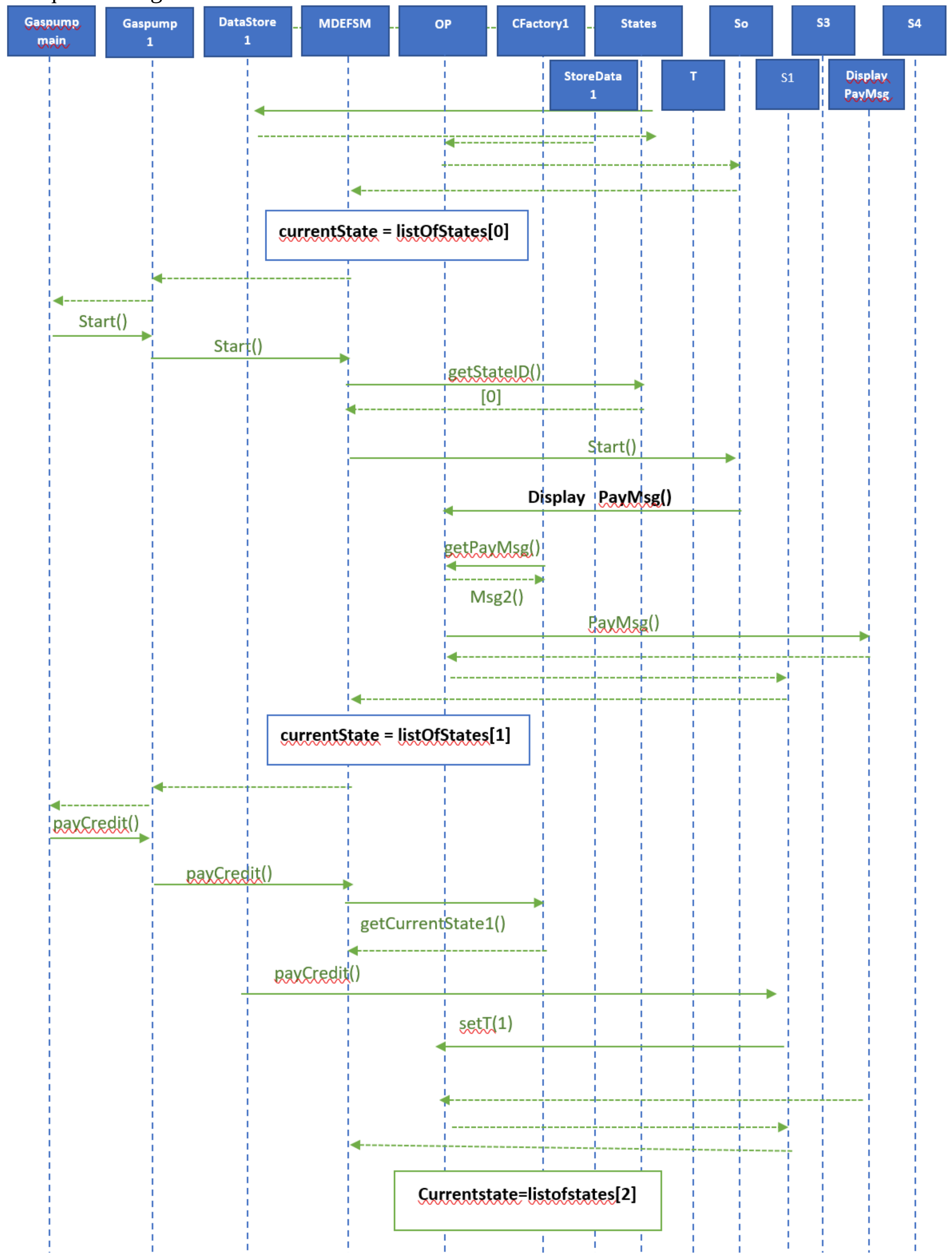
Class Datastore2: (gaspump2)

Purpose and attributes **Variables**	This class represents the concrete class for the Data Store used for GasPump2. **Temporary variables** Tempa // stores float a variable Tempb // stores float b variable Tempc // stores float a variable **Permanent variables** G // static int variable (gas type variable) Total // static float Cash // static float variable Price // static float priceRegular // static float priceSuper // static float pricePremium // static float
Functions **Float GetTempaFloat()** **Void setTempafloat(float)** **Float GetTempbFloat()** **Void setTempbFloat(float)** **Float GetTempcFloat(float)** **Void setTempcFloat(Float)** **Int getG()** **Void setG(int)** **Float getTotal()** **Void setTotal(float)** **Float getPriceRegular()** **Void setPriceRegular()** **Float getPriceSuper()** **Void setPriceSuper()**	

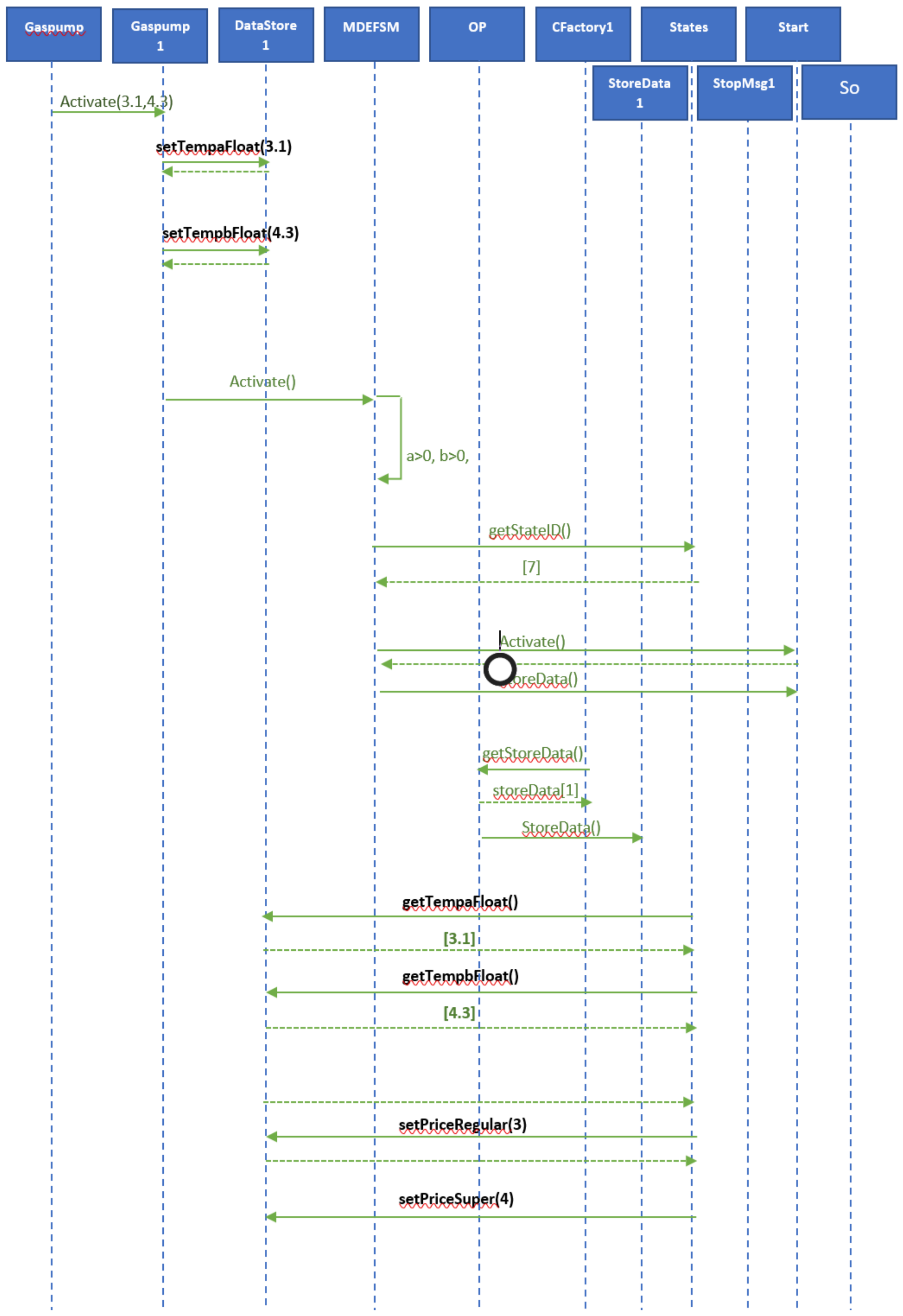
Float getPriceSuperPremium() **Float getPriceSuperPremium()** **Float getPrice()** **Float getCashFloat()** **Float setCashFloat(float)** **Void setRegularPrice()** **Void setSuperPrice()**	

Abinaya Janakan
A20376287

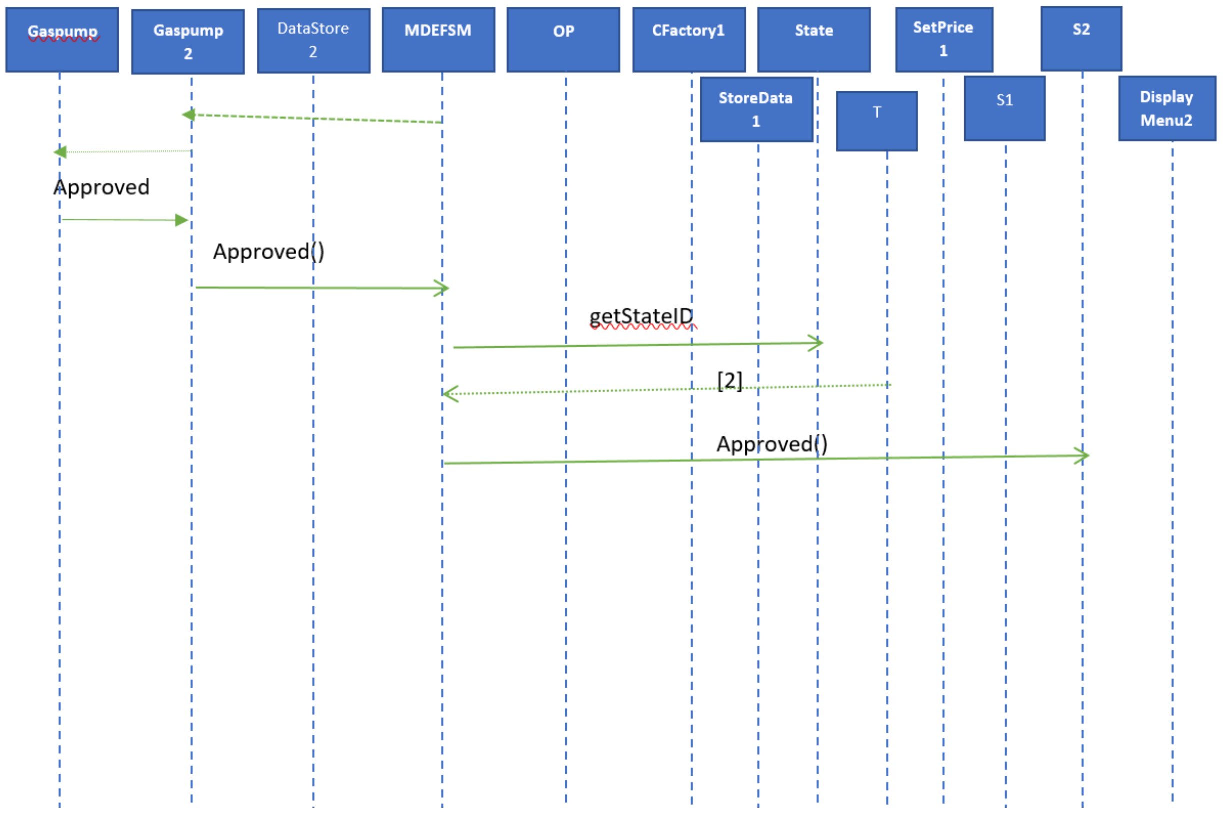
4.Sequence diagram



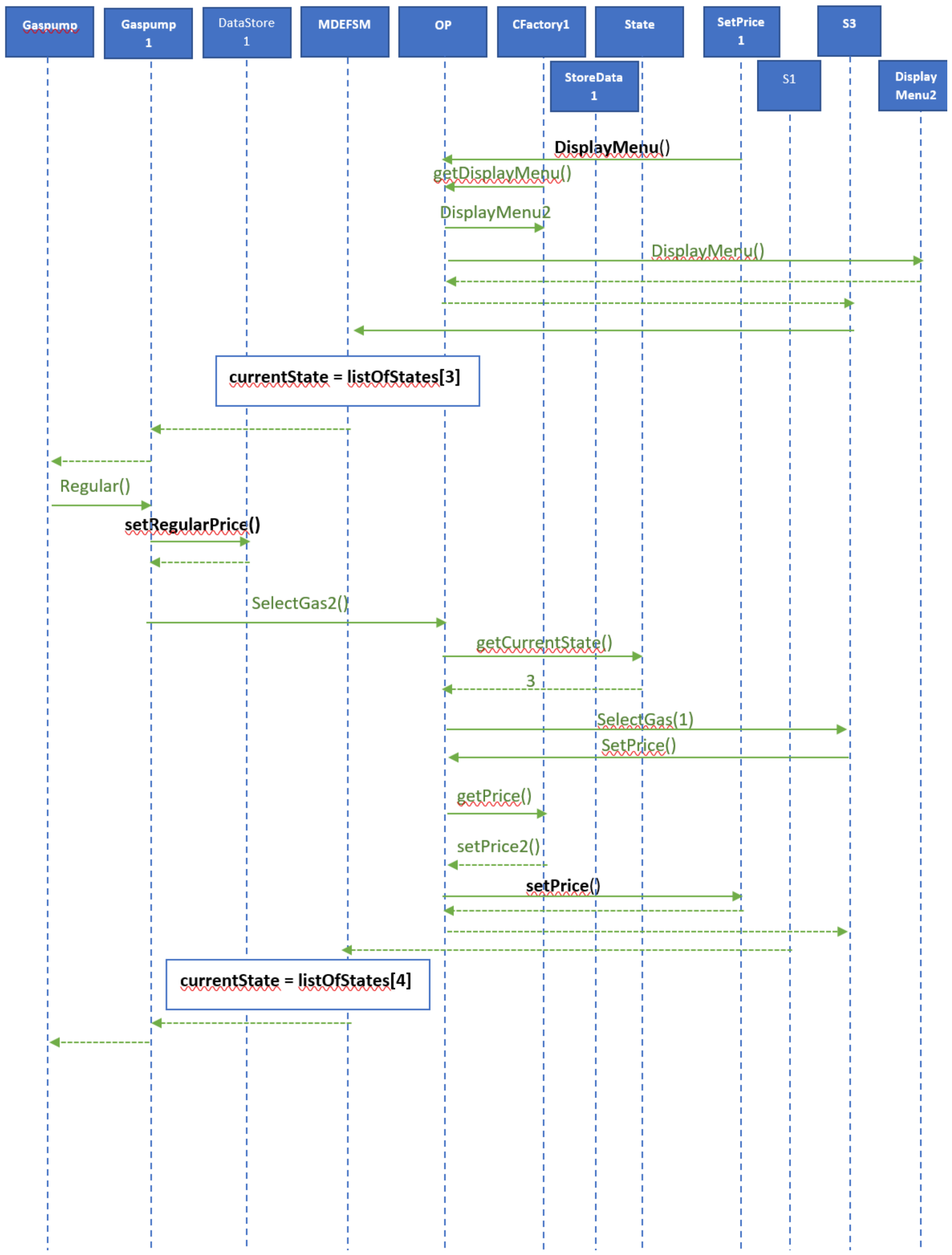
Abinaya Janakan
A20376287



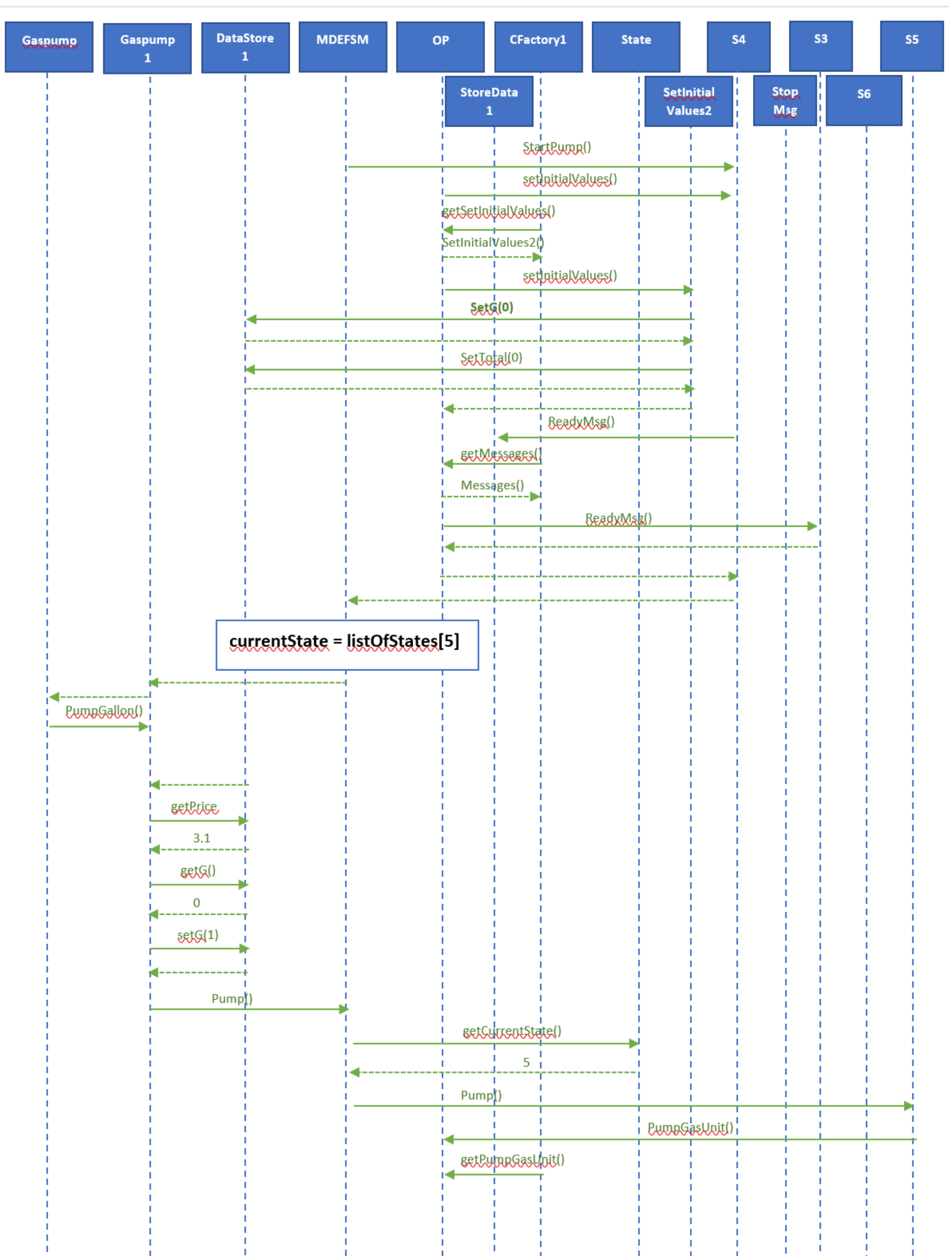
Abinaya Janakan
A20376287



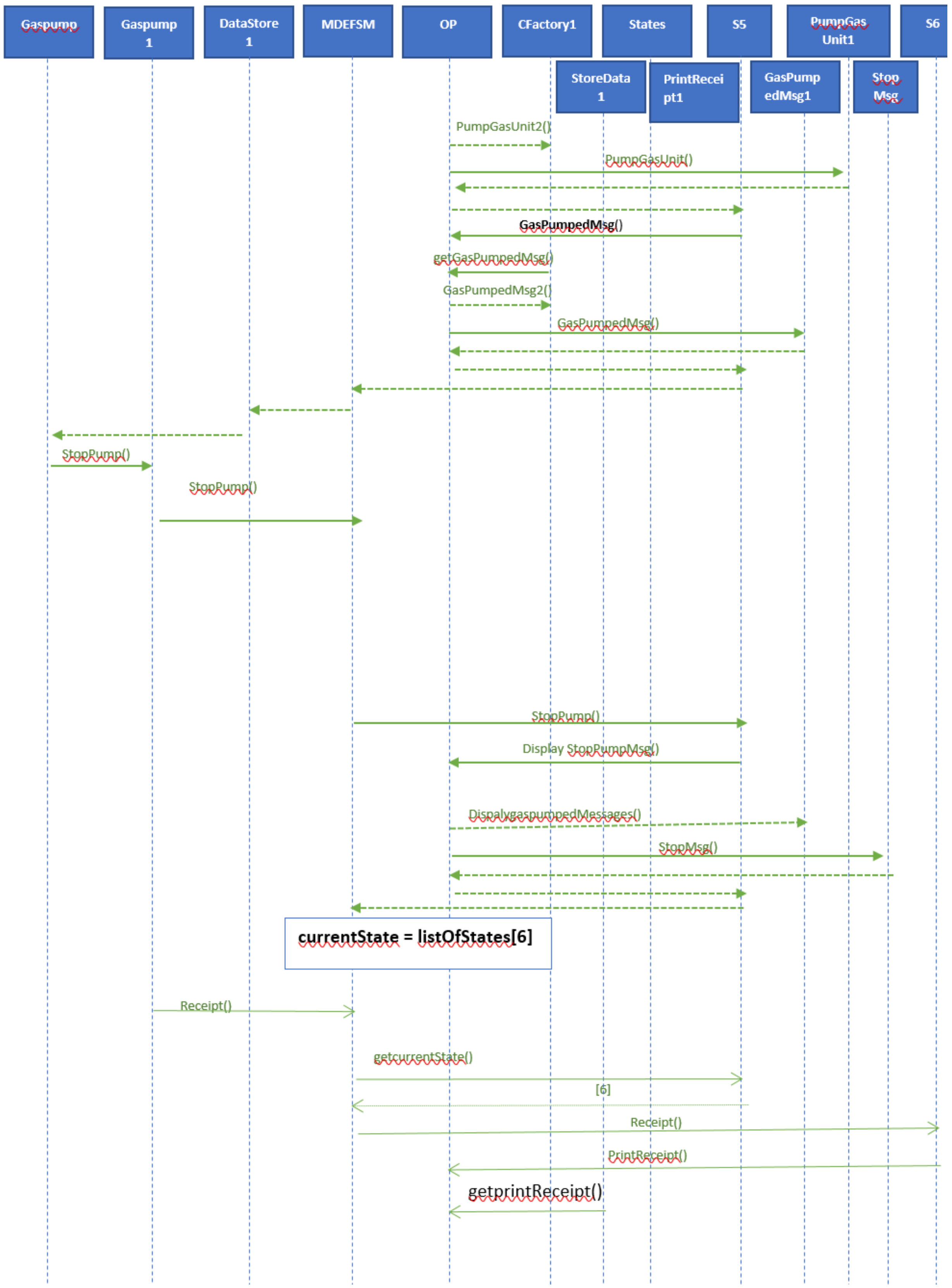
Abinaya Janakan
A20376287



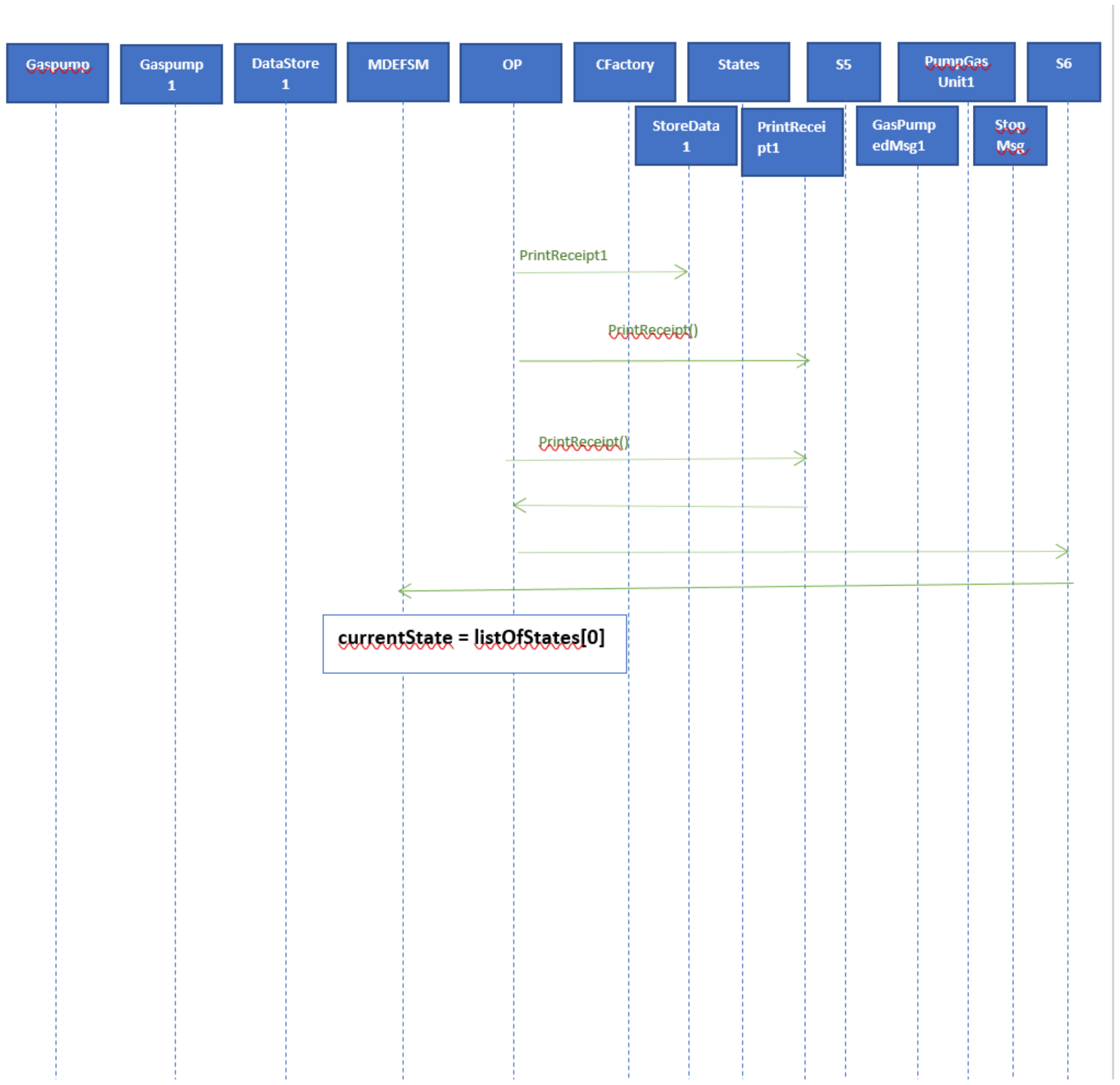
Abinaya Janakan
A20376287



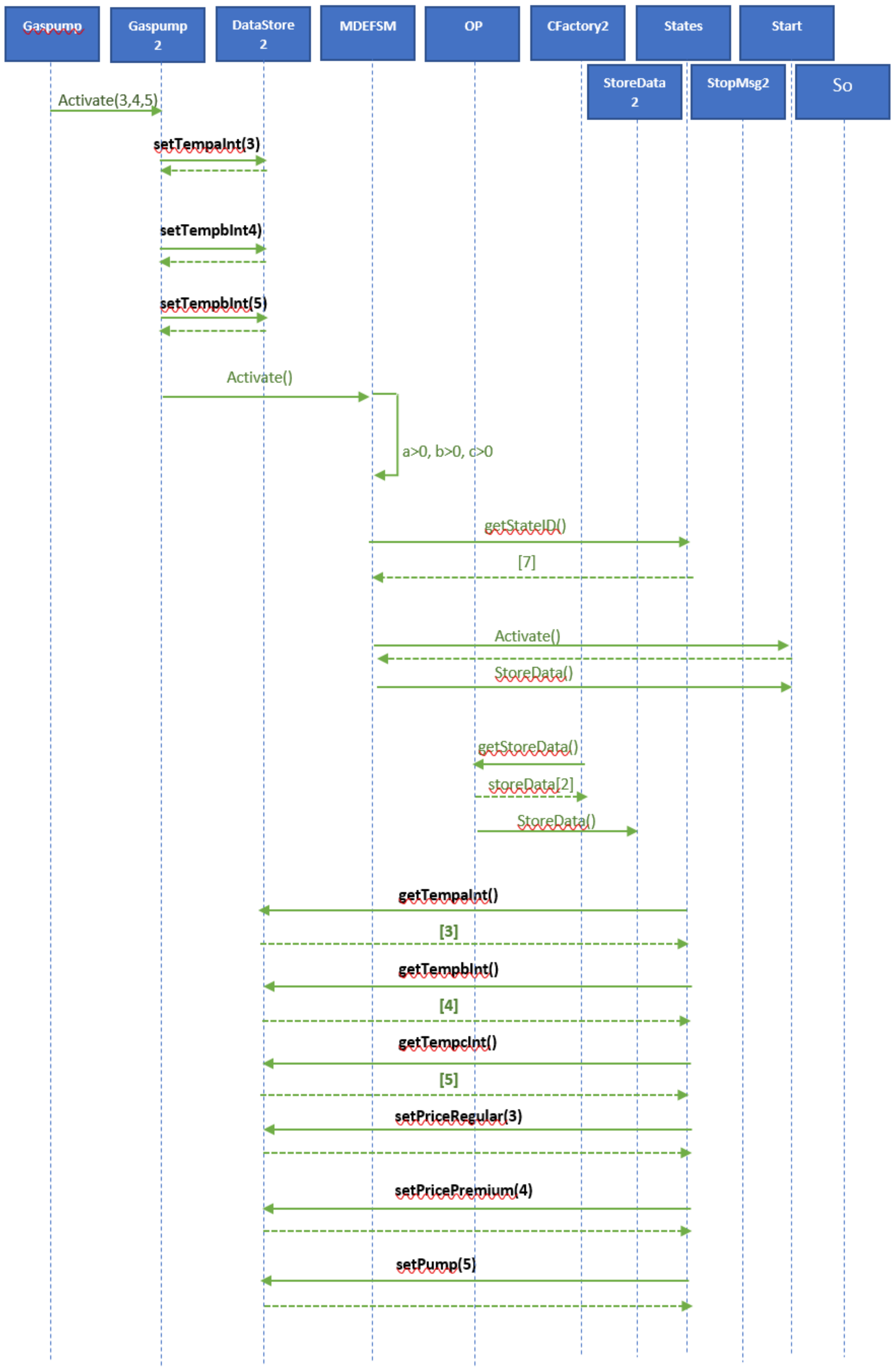
Abinaya Janakan
A20376287



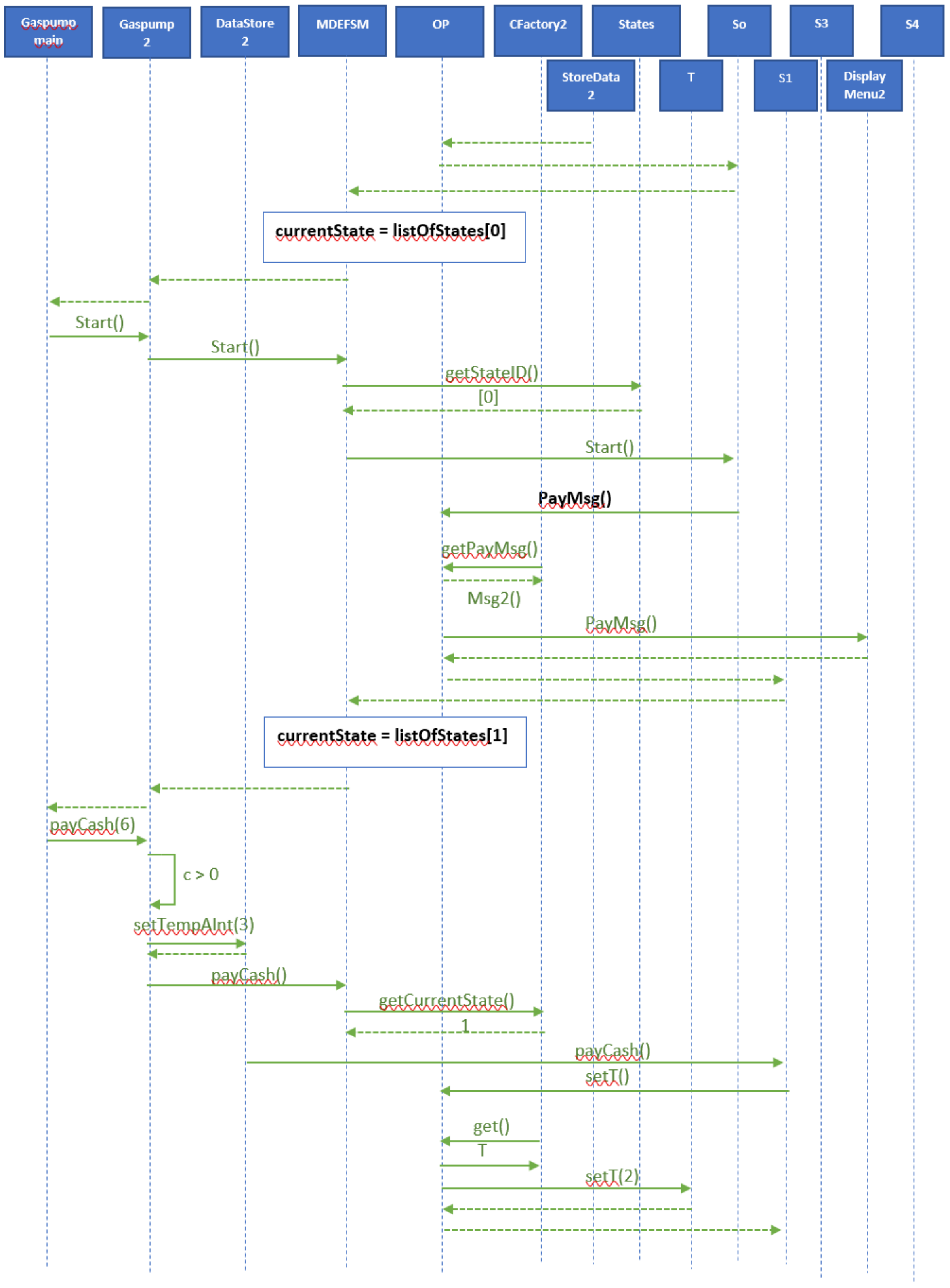
Abinaya Janakan
A20376287



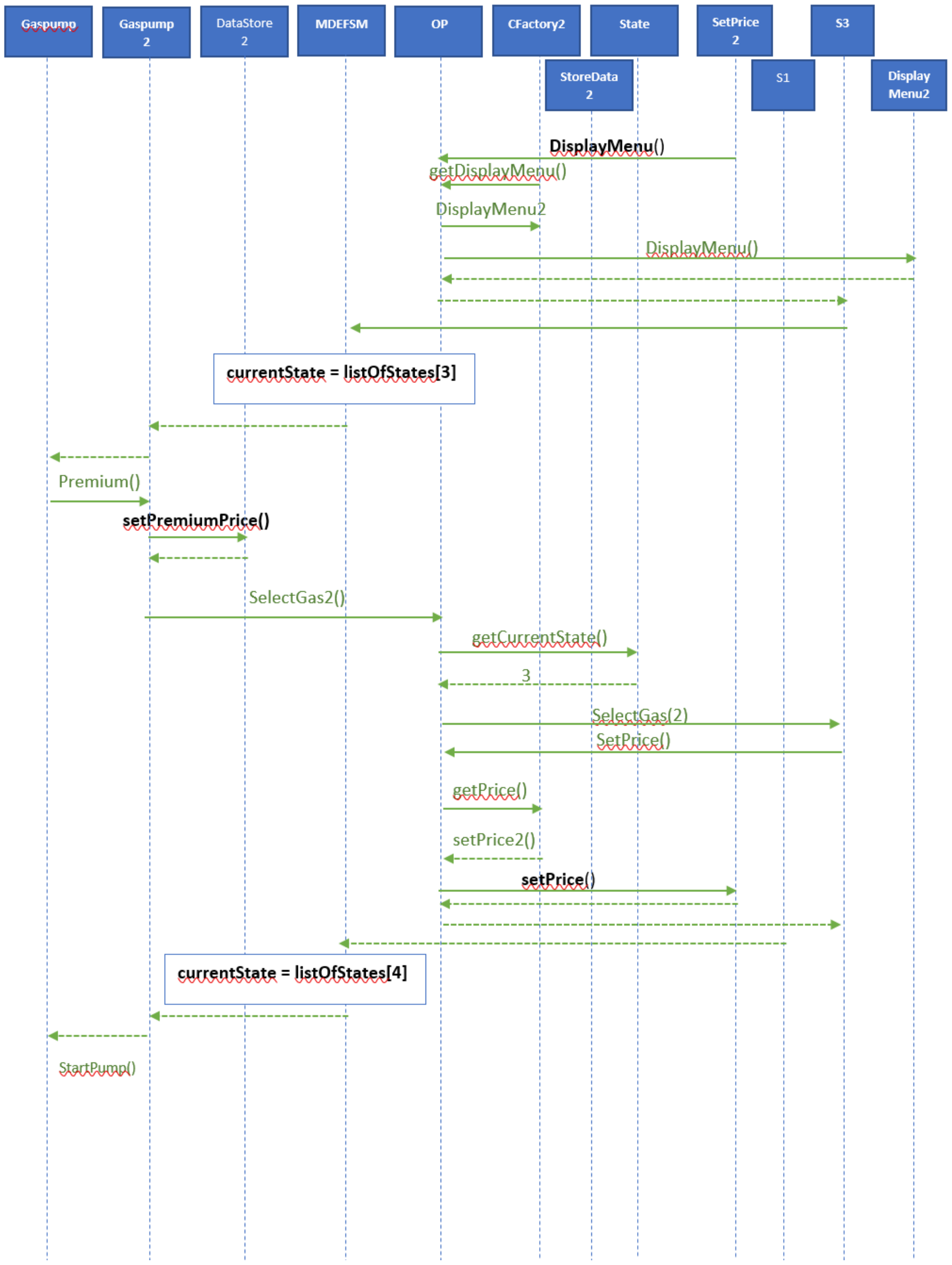
Abinaya Janakan
A20376287



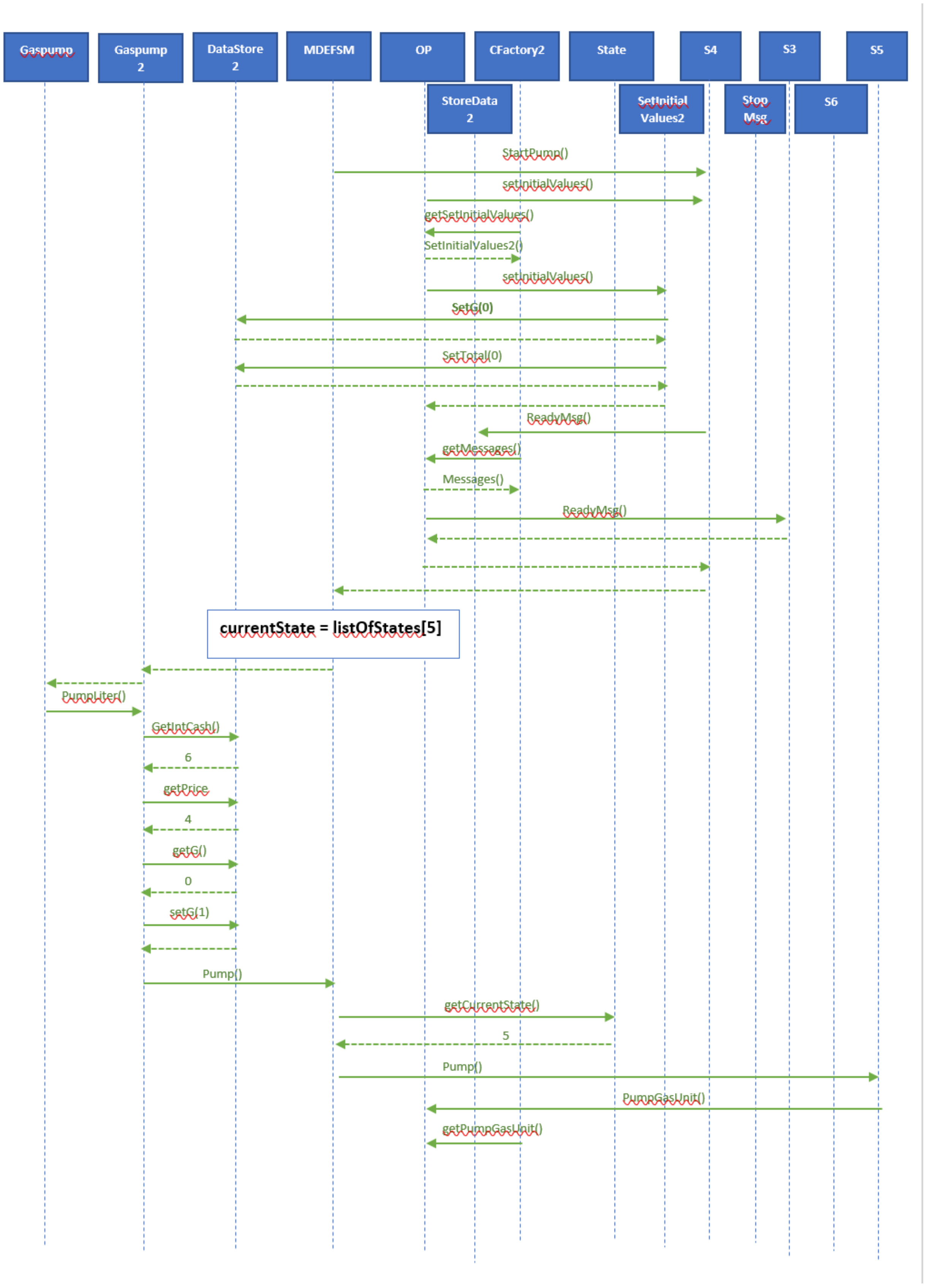
Abinaya Janakan
A20376287



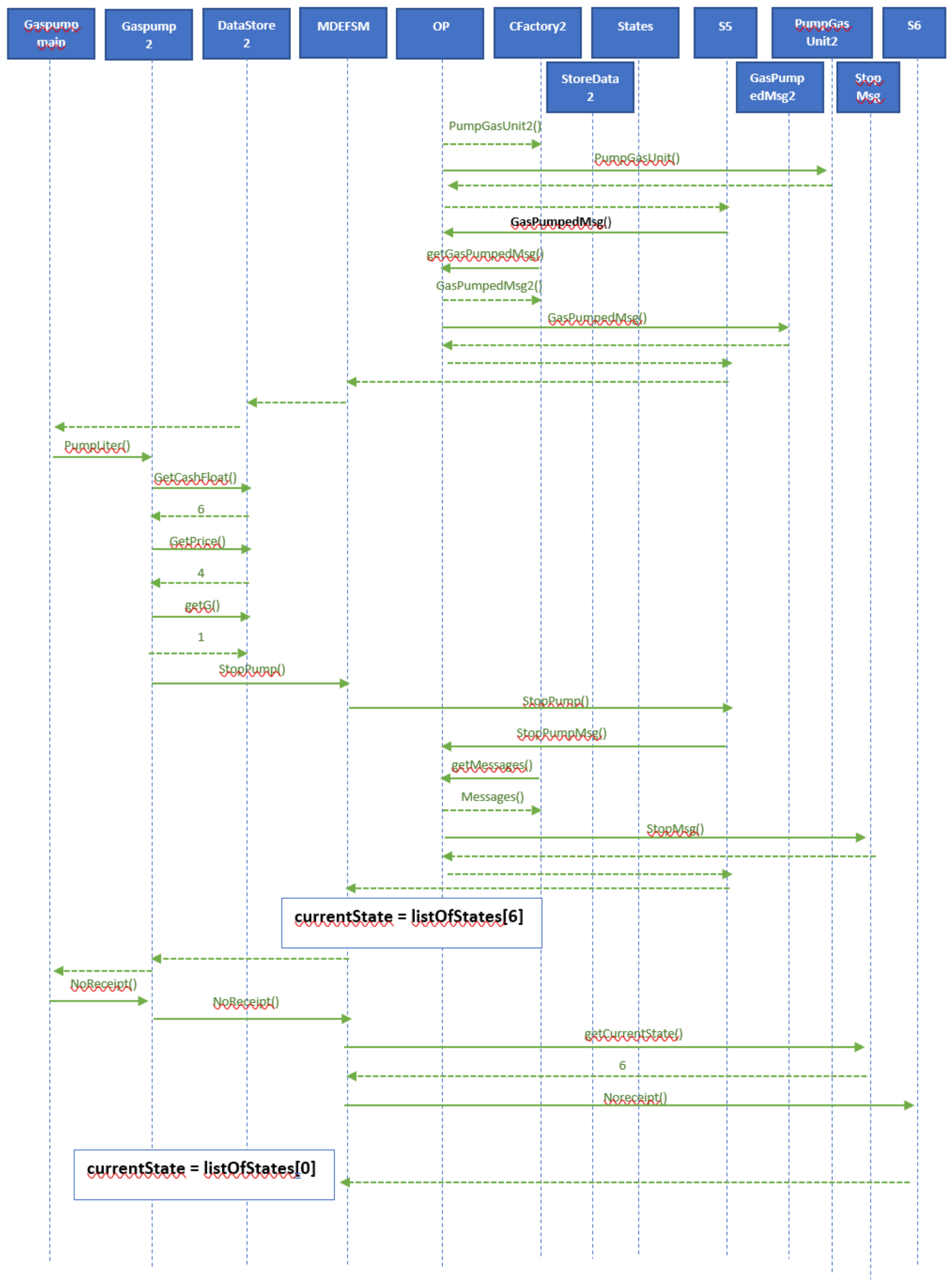
Abinaya Janakan
A20376287



Abinaya Janakan
A20376287



Abinaya Janakan
A20376287



5.Patterns and Source Code:

State Pattern:

- State.
- S0.
- S1.
- S2.
- S3.
- S4.
- S5.
- S6.

Strategy pattern:aj

- StoreData

• DisplayMenu.java
• PumpGasUnit .java
• PrintReceipt.java
• StoreCash.java
• DisplayPumpedMsg .java
• SetInitialValues.java
• SetPrice .java
• DisplayPayMsg.java

Abstract Factory:

public abstract StoreData getStoreData();

- public abstract DisplayMenu getDisplayMenu();
- public abstract PumpGasUnit getPumpGasUnit();
- public abstract PrintReceipt getPrintReceipt();
- public abstract StoreCash getStoreCash();
- public abstract DisplayGasPumpedMsg getDisplayGasPumpedMsg();
- public abstract SetInitialValues getSetInitialValues();
- public abstract SetPrice getSetPrice();
- public abstract Messages getMessages();

- public abstract DisplayPayMsg getDisplayPayMsg();

Abinaya Janakan
A20376287

Source Code:

```
package SSAproject;



        /**
        *
        *
        * This is the main class for the gaspump project
        * It lists the gaspump types and propmts for user input to make a selection
        */
        import java.io.BufferedReader;

        import java.io.InputStreamReader;




        public class GasPump {
        public static void main(String[] args) {
        try{
                int ch=0;

        BufferedReader buf=new BufferedReader(new InputStreamReader(System.in));
        // User is prompted to select a gaspump type
        System.out.println("Select the type of Gas Pump :");
        System.out.println("1. GasPump-1 :");
        System.out.println("2. GasPump-2 :");
        ch=(int)Float.parseFloat(buf.readLine()); // reads the user input
        System.out.println("You have selected the gas pump: "+ch);
        switch(ch)
        {

                case 1:
                {
                GasPump1 gasPump1 = new GasPump1(); //create the gaspump 1 object
                S0 s0 = new S0(); //Instantiate state s0
                S1 s1 = new S1();//Instantiate state s1
                S2 s2 = new S2();//Instantiate state s2
                S3 s3 = new S3();//Instantiate state s3
                S4 s4 = new S4();//Instantiate state s4
                S5 s5 = new S5();//Instantiate state s5
                S6 s6 = new S6();//Instantiate state s6
                Start s7 = new Start();
```

```
		MdaEfsm m= new MdaEfsm(); //instantiate MDAEFSM
		OutputProcessor op = new OutputProcessor(); //instantiate OP
		ConcreteFactory1 cf1 = new ConcreteFactory1(); //instantiate  Concrete factory
class
		DataStore ds;
		ds = cf1.getData(); //get data of concretefactory2
		gasPump1.setMdaEfsm(m); //set Gaspump1 object to access MDA
		gasPump1.setFactory(cf1); //set Gaspump2 object to use CF2
		gasPump1.setData(ds);
		s0.setOutputProcessor(op); //connnect state S0 to the output processor
		s0.setStateId(0);      //set Id of So to 0
		s1.setOutputProcessor(op);//connnect state S1 to the output processor
		s1.setStateId(1);      //set Id of S1 to 1
		s2.setOutputProcessor(op);//connnect state S2 to the output processor
		s2.setStateId(2);       //set Id of S2 to 2
		s3.setOutputProcessor(op);//connnect state S3 to the output processor
		s3.setStateId(3);    //set Id of S3 to 3
		s4.setOutputProcessor(op);//connnect state S4 to the output processor
		s4.setStateId(4);      //set Id of S4 to 4
		s5.setOutputProcessor(op);//connnect state S5 to the output processor
		s5.setStateId(5);           //set Id of S5to 5
		s6.setOutputProcessor(op);//connnect state S6 to the output processor
		s6.setStateId(6);          //set Id of S6 to 6
		s7.setOutputProcessor(op);//connnect state S7 to the output processor
		s7.setStateId(7);      //set Id of S7 to 7
		m.setListOfStates(s0,s1,s2,s3,s4,s5,s6,s7); //set all states to MDAEFSM
		op.setData(ds);
		op.setAbstractFactory(cf1); //set concerete factory 1 object
		String input=null;
		int i;
		while(true)
		{
		System.out.println("Enter operation: \n 1:activate 2.start 3.paycredit 4.reject
5.cancel 6.approved 7.super 8.regular 9.startpump 10.pumpgallon 11.stoppump
12.getCurrentState(testing purposes)");
		input=buf.readLine();
		i=Integer.parseInt(input);
		switch(i)
		{
		case 1:  // gets the price of regular and super gas
		System.out.println("Enter price of regular gas");
		float a = Float.parseFloat(buf.readLine());
		System.out.println("Enter price of super gas");
float b = Float.parseFloat(buf.readLine());
gasPump1.Activate(a,b);
break;
case 2:
gasPump1.Start(); //invokes start operation of the gas pump1
break;
case 3:
gasPump1.PayCredit(); //invokes paycredit operation of the gas pump1
break;
case 4:
gasPump1.Reject();//invokes reject operation of the gas pump1
break;
case 5:
gasPump1.Cancel(); //invokes cancel operation of the gas pump1
break;
case 6: //Card approved
gasPump1.Approved();//invokes approved operation of the gas pump1
break;
case 7: //super gas
gasPump1.Super();//invokes super operation of the gas pump1
break;
case 8: //regular gas
gasPump1.Regular();//invokes start operation of the gas pump1
break;
case 9: //startpump
gasPump1.StartPump();//invokes startpump operation of the gas pump1
break;
case 10: //pumpGallon
gasPump1.PumpGallon();//invokes pumpgallon operation of the gas pump1
break;
case 11:gasPump1.StopPump(); //invokes stoppump operation of the gas pump1
break;
case 12:
System.out.println(gasPump1.getCurrentStateId() ); // checks the current state
break;
default:
System.out.println("invalid input,enter a valid operation");
}
}
}
case 2:
{
GasPump2 gasPump2 = new GasPump2(); //create the gaspump 1 object
```

```
			S0 s0 = new S0(); //instantiate state so
			S1 s1 = new S1(); //instantiate state s1
			S2 s2 = new S2();  //instantiate state s2
			S3 s3 = new S3(); //instantiate state s3
			S4 s4 = new S4();   //instantiate state s4
			S5 s5 = new S5();    //instantiate state s5
			S6 s6 = new S6();  //instantiate state s6
			Start s7 = new Start();
			MdaEfsm m= new MdaEfsm(); //instantiate MDAEFSM
			OutputProcessor op = new OutputProcessor(); //instantite OP
			ConcreteFactory2 cf2= new ConcreteFactory2(); //instantiate CF3
			DataStore ds;
			ds = cf2.getData(); //get Data from cf3
			gasPump2.setMdaEfsm(m); //set gasPump MDA pointer
			gasPump2.setFactory(cf2); //set gaspump CF pointer
			gasPump2.setData(ds); //set Datastore2 object to Gaspump2 object
			s0.setOutputProcessor(op);       //connnect state S0 to the output processor
			s0.setStateId(0); //set Id of So to 0
			s1.setOutputProcessor(op);//connnect state S1 to the output processor
			s1.setStateId(1);//set Id of S1 to 1
			s2.setOutputProcessor(op);//connnect state S2 to the output processor
			s2.setStateId(2);//set Id of S2 to 2
			s3.setOutputProcessor(op);//connnect state S3 to the output processor
			s3.setStateId(3);//set Id of S3 to 3
			s4.setOutputProcessor(op);//connnect state S4 to the output processor
			s4.setStateId(4);//set Id of S4 to 4
			s5.setOutputProcessor(op);//connnect state S5 to the output processor
			s5.setStateId(5);//set Id of S5 to 5
			s6.setOutputProcessor(op);//connnect state S6 to the output processor
			s6.setStateId(6);//set Id of S6 to 6
			s7.setOutputProcessor(op);//connnect state S7 to the output processor
			s7.setStateId(7);//set Id of S7 to 7
			m.setListOfStates(s0,s1,s2,s3,s4,s5,s6,s7); //set all State pattern states to 
MDAefsm
			op.setData(ds); //set OP data as Data 2
			op.setAbstractFactory(cf2); //set OP to use CF2
			String input=null;
			int i;
			while(true)
			{
			System.out.println("Enter operation: \n 1:activate 2.start 3.paycash 4.cancel 
5.premium 6.regular 7.Super 8.startpump 9.pumpliter 10.stoppump 11.receipt 12.noreceipt 
13.getCurrentState2");
```

```
input=buf.readLine();
i=Integer.parseInt(input);
switch(i)
{
case 1: //take input of regular and premium prices and activate
System.out.println("Enter price of regular gas");
int a = Integer.parseInt(buf.readLine());
System.out.println("Enter price of premium gas");
int b = Integer.parseInt(buf.readLine());
System.out.println("Enter price of super gas");
int c1 =Integer.parseInt(buf.readLine());
gasPump2.Activate(a,b,c1);
break;
case 2:
gasPump2.Start(); //invokes start operation of the gas pump2
break;
case 3:
System.out.println("enter cash \t");         //invokes start operation of the gas
pump2
int cash = (int)Float.parseFloat(buf.readLine());
gasPump2.PayCash(cash);
break;
case 4:
gasPump2.Cancel(); //invokes start operation of the gas pump2
break;
case 5: //set premium gas
gasPump2.Premium(); //invokes premium operation of the gas pump2
break;
case 6: //set regular Gas
gasPump2.Regular(); //invokes regular operation of the gas pump2
break;
case 7: //set super Gas
        gasPump2.Super(); //invokes regular operation of the gas pump2
        break;
case 8: //startPump
gasPump2.StartPump(); //invokes startpump operation of the gas pump2
break;
case 9: //pump gas in LITERS
gasPump2.PumpLiter(); //invokes pumpliter operation of the gas pump2
break;
case 10: //stopPump
gasPump2.StopPump();//invokes stoppump operation of the gas pump2
break;
```

```
                case 11: //Receipt
                gasPump2.Receipt(); //invokes receipt operation of the gas pump2
                break;
                case 12: //No receipt to be printed
                gasPump2.NoReceipt(); //invokes Noreceipt operation of the gas pump2
                break;
                case 13:
                System.out.println(gasPump2.getCurrentStateId() ); //prints the current state
                break;
                default:
                System.out.println("invalid , please enter a valid operation");
                }
                }
                }
                }
                }
                catch(Exception ex) //catch exceptions if any
                {
                System.out.println(ex); //print exception
                }
                }
        }
```

2.Class GasPump1

```
package SSAproject;

public class GasPump1 {

        /**
        *
        *
        * GasPump1 class consists of functions for the gas pump1
        * it invokes methods in MDAEFSM
        *
        */

        MdaEfsm m;
        AbstractFactory af;
        DataStore ds;
        //sets reference to mdaefsm object
        public void setMdaEfsm(MdaEfsm m) {
        this.m = m;
        }
```

```
//Sets referance to concrete factory 1 object
public void setFactory(ConcreteFactory1 cf1) {
this.af = cf1;
}
//Seta reference to the Datastore object
public void setData(DataStore ds) {
this.ds = ds;
}
//checks if a and b is >0 and sets the value of a nd b
public void Activate(float a,float b) {
if(a>0 && b>0){
ds.setTempaFloat(a);
ds.setTempbfloat(b);
m.Activate();
}
}
//invokes start of the mdaefsm object
public void Start() {
m.Start();
}
//invokes PayCredit using the  MDAEFSM object
public void PayCredit() {
m.PayCredit();
}
//invokes Reject using the  MDAEFSM object
public void Reject() {
m.Reject();
}
//invokes Cancel using the  MDAEFSM object
public void Cancel() {
m.Cancel();
}
//invokes Approved using the  MDAEFSM object
public void Approved() {
m.Approved();
}
//sets price for Supergas and invokes Super() using the  MDAEFSM object
public void Super() {
((DataStore1)ds).setSuperPrice();
m.SelectGas(2);
}
//set price for  regular price and invokes PayCredit using the  MDAEFSM object
public void Regular(){
((DataStore1)ds).setRegularPrice();
m.SelectGas(1);
}
```

```
//invokes startpump using the  MDAEFSM object
public void StartPump() {
m.StartPump();
}
//increments the value of G and forwards to MDAEFSM
public void PumpGallon() {
ds.setG((((DataStore1)ds).getG() + 1));
m.Pump();
}
//invokes stoppump and Receipt using the  MDAEFSM object
public void StopPump() {
m.StopPump();
m.Receipt();
}
//invokes getCurrentStateId using the  MDAEFSM object
public int getCurrentStateId(){
return m.getCurrentStateId();
}
}
```

3.Class GasPump2:

```
package SSAproject;
/**
*
*
* GasPump2 class consists of functions for the gas pump1
* it invokes methods in MDAEFSM
*
*/

public class GasPump2 {
	MdaEfsm m;
	AbstractFactory af;
	DataStore ds;
	//set reference to MdaEFSM object
	public void setMdaEfsm(MdaEfsm m) {
	this.m = m;
	}
	//Set reference to concrete factory1 object
	public void setFactory(ConcreteFactory2 cf2) {
	this.af = cf2;
	}
	//Set reference to Data object
	public void setData(DataStore ds) {
	this.ds = ds;
}
//checks if the values are of a,b,c are >0 and sets the values.
public void Activate(int a,int b,int c1) {
if(a>0 && b>0 && c1>0){
((DataStore2)ds).setTempaFloat(a);
((DataStore2)ds).setTempbfloat(b);
((DataStore2)ds).setTempbfloat(c1);

m.Activate();
}
}
//invokes PayCredit using the  MDAEFSM object
public void Start() {
m.Start();
}
//invokes PayCash using the  MDAEFSM object
public void PayCash(int c) {
if(c>0){
((DataStore2)ds).setFloatTempC(c);
m.PayCash();
}
}
//invokes Cancel using the  MDAEFSM object
public void Cancel() {
m.Cancel();
}
//invokes Premium using the  MDAEFSM object and sets Data
public void Premium() {
((DataStore2)ds).setPremiumPrice();
m.SelectGas(2);
}
//invokes Regular using the  MDAEFSM object
public void Regular(){
((DataStore2)ds).setRegularPrice();
m.SelectGas(1);
}
//invokes Super using the  MDAEFSM object
        public void Super(){
        ((DataStore2)ds).setPriceSuper(0);
        m.SelectGas(3);
        }
////invokes StartPump using the  MDAEFSM object
public void StartPump() {
m.StartPump();
}
//////invokes PumpLitre using the  MDAEFSM object
```

```
	public void PumpLiter() {
	if(((DataStore2)ds).getFloatCash() < ( ((DataStore2)ds).getG() + 1) *
((DataStore2)ds).getPrice()){
	m.StopPump();
	}
	else{
	((DataStore2)ds).setG( ( ((DataStore2)ds).getG() + 1) );
	m.Pump();
	}
	}
	////invokes StopPump using the  MDAEFSM object
	public void StopPump() {
	m.StopPump();
	}
	////invokes Receipt using the  MDAEFSM object
	public void Receipt(){
	m.Receipt();
	}
	////invokes NoReceipt using the  MDAEFSM object
	public void NoReceipt(){
	m.NoReceipt();
	}
	////invokes getCurrentStateID using the  MDAEFSM object
	public int getCurrentStateId() {
	return m.getCurrentStateId();
	}

		}
```

4.Mdaefsm

```
package SSAproject;

public class MdaEfsm {
	State currentState;
	State[] listOfStates = new State[8];
	public void setState(State states){
	currentState = states;
	}
	//this function contains a list of states
	public void setListOfStates(State a,State b,State c,State d,State e,State f,State g,State h){
	listOfStates[0] = a;
	listOfStates[1] = b;
	listOfStates[2] = c;
	listOfStates[3] = d;
	listOfStates[4] = e;
	listOfStates[5] = f;
```

```
listOfStates[6] = g;
listOfStates[7] = h;
this.currentState = listOfStates[7];
}

public void Activate(){
int currState = currentState.getStateId();
switch(currState)
{
case 0: break;
case 1: break;
case 2: break;
case 3: break;
case 4: break;
case 5: break;
case 6: break;
case 7:
currentState.Activate();
currentState.StoreData();
currentState = listOfStates[0];
break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void Start(){
int currState = currentState.getStateId();
switch(currState){
case 0:
currentState.Start();
currentState = listOfStates[1];
break;
case 1: break;
case 2: break;
case 3: break;
case 4: break;
case 5: break;
case 6: break;
case 7: break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void PayCredit(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1:
```

```
currentState.PayCredit();
currentState = listOfStates[2];
break;
case 2: break;
case 3: break;
case 4: break;
case 5: break;
case 6: break;
case 7: break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void PayCash(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1:
currentState.PayCash();
currentState.StoreCash();

currentState = listOfStates[3];
break;
case 2: break;
case 3: break;
case 4: break;
case 5: break;
case 6: break;
case 7: break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void Reject(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1: break;
case 2:
currentState.Reject();
currentState = listOfStates[0];
break;
case 3: break;
case 4: break;
case 5: break;
case 6: break;
case 7: break;
};
```

```
}
//forwards to  current state if currState matches the id of CurrentState
public void Cancel(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1: break;
case 2: break;
case 3:
currentState.Cancel();
currentState = listOfStates[0];
break;
case 4: break;
case 5: break;
case 6: break;
case 7: break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void Approved(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1: break;
case 2:
currentState.Approved();

currentState = listOfStates[3];
break;
case 3: break;
case 4: break;
case 5: break;
case 6: break;
case 7: break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void StartPump(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1: break;
case 2: break;
case 3: break;
case 4:
//to5
currentState.StartPump();
currentState = listOfStates[5];
break;
case 5: break;
case 6: break;
case 7: break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void Pump(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1: break;
case 2: break;
case 3: break;
case 4: break;
case 5:
//to5
currentState.Pump();
break;
case 6: break;
case 7: break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void StopPump(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1: break;
case 2: break;
case 3: break;
case 4: break;
case 5:
//to6
currentState.StopPump();
currentState = listOfStates[6];
break;
case 6: break;
case 7: break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void SelectGas(int g){
int currState = currentState.getStateId();
```

```
switch(currState){
case 0: break;
case 1: break;
case 2: break;
case 3:
currentState.SelectGas(g);
//currentState.SetPrice(g);
currentState = listOfStates[4];
break;
case 4: break;
case 5: break;
case 6: break;
case 7: break;
};
}
//forwards to  current state if currState matches the id of CurrentState
public void Receipt(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1: break;
case 2: break;
case 3: break;
case 4: break;
case 5: break;
case 6:
//to0
currentState.Receipt();
currentState = listOfStates[0];
break;
case 7: break;
};
}
//forwards to same function name of current state if the current state is correct
public void NoReceipt(){
int currState = currentState.getStateId();
switch(currState){
case 0: break;
case 1: break;
case 2: break;
case 3: break;
case 4: break;
case 5: break;
case 6:
//to0
currentState.NoReceipt();
```

```
        currentState = listOfStates[0];
        break;
        case 7: break;
        };
        }

        public int getCurrentStateId() {
        return currentState.getStateId();
        }

}
```

5.OutPut processor

```
package SSAproject;

public class OutputProcessor {
        AbstractFactory af;
        DataStore ds;
        Messages messages;
        //sets the abstract factory object to the concrete factory object
        public void setAbstractFactory(AbstractFactory af) {
        this.af = af;
        }
        //set the Datastore object
        public void setData(DataStore ds) {
        this.ds = ds;
        }
        //returns the concrete object of the function StoreData
        public void StoreData(){
        StoreData storeData;
        storeData = af.getStoreData();
        storeData.setData(ds);
        storeData.storeData();
        }
        //returns the concrete object of the function StoreCash
        public void StoreCash(){
        StoreCash storeCash = af.getStoreCash();
        storeCash.setCash(ds);
        storeCash.storeCash();
        }
        //returns the concrete object of the function DisplayMenu
        public void DisplayMenu(){
        DisplayMenu displayMenu;
        displayMenu = af.getDisplayMenu();
        displayMenu.displayMenu();
        }
        //returns the concrete object of the function DisplayPayMsg
```

```
public void DisplayPayMsg(){
DisplayPayMsg displayPayMsg;
displayPayMsg = af.getDisplayPayMsg();
displayPayMsg.displayPayMsg();
}
//returns the concrete object of the function PumpGasUnit
public void PumpGasUnit(){
PumpGasUnit pumpGasUnit;
pumpGasUnit = af.getPumpGasUnit();
pumpGasUnit.pumpGasUnit();
}
//returns the concrete object of the function PrintReceipt
public void PrintReceipt(){
PrintReceipt printReceipt;
printReceipt = af.getPrintReceipt();
printReceipt.printReceipt(ds);
}
//returns the concrete object of the function DisplayRejectMsg
public void DisplayRejectMsg(){
messages = af.getMessages();
messages.displayRejectMsg();
}

//returns the concrete object of the function SetPrice
public void SetPrice(int g){
SetPrice setPrice = af.getSetPrice();
setPrice.setPrice(ds);
}
//returns the concrete object of the function DisplayReadyMsg
public void DisplayReadyMsg(){
messages = af.getMessages();
messages.displayReadyMsg();
}
//returns the concrete object of the function SetInitialValues
public void SetInitialValues(){
this.ds.setG(0);
this.ds.setTotal(0);
}
//returns the concrete object of the function DisplayGasPumpedMsg
public void DisplayGasPumpedMsg(){
DisplayGasPumpedMsg displayGasPumpedMsg = af.getDisplayGasPumpedMsg();
displayGasPumpedMsg.displayGasPumpedMsg(ds);
}
//returns the concrete object of the function DisplaycancelMsg
public void DisplayCancelMsg(){
messages = af.getMessages();
```

```
        messages.displayCancelMsg();
        }
        //returns the concrete object of the function DisplayStopMSg
        public void DisplayStopMsg() {
        messages = af.getMessages();
        messages.displayStopMsg();
        }

}
```

6.DataStore

```
package SSAproject;

public abstract class DataStore {

                public int getIntTempA() {
                        return 0;
                        }
                        public void setIntTempA(int tempA) {}
                        public int getIntTempC() {
                        return 0;
                        }
                        public void setIntTempC(int tempC) {}

                        public void setTempc1(float tempc1){}

                        public int getG() {
                        return 0;
                        }
                        public void setG(int g) {}
                        public int getIntPrice() {
                        return 0;
                        }
                        public void setIntPrice(int price) {}
                        public int getIntCash() {
                        return 0;
                        }
                        public void setIntCash(int cash) {}
                        public float getFloatTempA() {
                        return 0;
                        }
                        public void setTempaFloat(float tempA) {}
                        public float getFloatTempB() {
                        return 0;
                        }
                        public void setTempbfloat(float tempB) {
                        }
                        public float getPriceRegular() {
                        return 0;
                        }
                        public void setPriceRegular(float priceRegular) {
                        }
                        public float getPriceSuperPremium() {
                        return 0;
                        }
                        public void setPriceSuperPremium(float priceSuperPremium) {
                        }
                        public float getPriceSuper() {
                                return 0;
                                }
                                public void setPriceSuper(float priceSuper) {
                                }
                        public float getFloatTempC() {
                        return 0;
                        }
                        public float getTempc1(){
                                return 0;
                        }
                        public void setFloatTempC(float tempC) {
                        }
                        public void setTotal(int total){}
                        public float getFloatCash() {
                        return 0;
                        }
                        public void setFloatCash(float cash) {
                        }
                        public float getTotal(){ return 0;}
}
```

7.DataStore1

```
package SSAproject;

public class DataStore1 extends DataStore{
        static float tempA; //stores user input a
        static float tempB; //stores user input b

        static int g; //gas pumped unit
        static float total; //total
        static float price; //price of gas selected for pumping
        static float priceRegular; //stores price of regular gas
        static float priceSuper; //stores price of super gas
        public float getFloatTempA() {
        return tempA;
        }
        public void setTempaFloat(float tempA) {
        DataStore1.tempA = tempA;
        }
        public float getFloatTempB() {
        return tempB;
        }
        public void setTempbfloat(float tempB) {
        DataStore1.tempB = tempB;
        }

        public int getG() {
        return g;
        }
        public void setG(int g) {
        DataStore1.g = g;
        }
        public float getTotal(){
        DataStore1.total = (DataStore1.g * DataStore1.price); //calculates total from current g
and price and returns
        return DataStore1.total;
        }
        public float getPriceRegular() {
        return priceRegular;
        }
        public void setPriceRegular(float priceRegular) {
        DataStore1.priceRegular = priceRegular;
        }
        public float getPriceSuperPremium() {
        return priceSuper;
        }
        public void setPriceSuperPremium(float priceSuperPremium) {
        DataStore1.priceSuper = priceSuperPremium;
        }
        public void setTotal(float total) {
        DataStore1.total = total;
        }
        public float getPrice() {
        return price;
        }
        public void setRegularPrice() {
        DataStore1.price = priceRegular;
        }
```

```
	public void setSuperPrice() {
	DataStore1.price = priceSuper;
	}

}
```

8.DataStore2

```
package SSAproject;

public class DataStore2 extends DataStore{
	static float tempA; //stores user input a
	static float tempB; //stores user input b
	static float tempC; //stores user input c
	static int g; //units of gas pumped
	static float tempc1;
	static float total; //total for printing receipt
	static float cash; //stores cash paid by user
	static float price; //stores price of gas which is selected for pumping
	static float priceRegular; //stores price of regular gas
	static float pricePremium; //stores price of premium gas
	static float priceSuper;
	public float getFloatTempA() {
	return tempA;
	}
	public void setTempaFloat(float tempA) {
	DataStore2.tempA = tempA;
	}
	public void setTempc1Float(float tempc1) {
		DataStore2.tempc1 = tempc1;
		}
	public float getTempc1Float() {
		return tempc1;
		}
	public float getFloatTempB() {
	return tempB;
	}
	public void setTempbfloat(float tempB) {
	DataStore2.tempB = tempB;
	}
	public float getFloatTempC() {
	return tempC;
	}
	public void setFloatTempC(float tempC) {
	DataStore2.tempC = tempC;
	}
	public int getG() {
	return g;
```

```
        }
        public void setG(int l) {
        DataStore2.g = l;
        }


        public float getTotal(){
        DataStore2.total = (DataStore2.g * DataStore2.price); //calculates total from current g
and price and returns
        return DataStore2.total;
        }
        public void setTotal(float total){
        DataStore2.total= total;
        }
        public float getPriceRegular() {
        return priceRegular;
        }
        public void setPriceRegular(float priceRegular) {
        DataStore2.priceRegular = priceRegular;
        }
        public float getFloatCash(){
        return cash;
        }
        public float getPriceSuperPremium() {
        return pricePremium;
        }
        public void setPriceSuperPremium(float priceSuperPremium) {
        DataStore2.pricePremium = priceSuperPremium;
        }
        public float getPriceSuper() {
                return priceSuper;
                }
                public void setPriceSuper(float priceSuper) {
                DataStore2.priceSuper = priceSuper;
                }
        public void setFloatCash(float cash) {
        DataStore2.cash = cash;
        }
        public float getPrice() {
        return price;
        }
        public void setPremiumPrice() {
        DataStore2.price = pricePremium;
        }
        public void setRegularPrice() {
        DataStore2.price = priceRegular;
        }
        public void setSuperPrice() {
                DataStore2.price = priceSuper;
                }
        public void setTempc1float(float tempc1) {
                DataStore2.tempc1 = tempc1;
                }
                public float getFloatTempC1() {
                return tempc1;
                }




}


//State Pattern Starts here:
```

9.State

```
package SSAproject;
/**
*
*
* This class is used by all state objects and includes methods that are going be called by the state objects
*
*/

public class State {
        int stateId;
        OutputProcessor op; //pointer to OP
        public int getStateId(){
        return stateId;
        }
        public void setStateId(int stateId) {
        this.stateId = stateId;
        }
        public void setOutputProcessor(OutputProcessor outputProcessor) {
        this.op = outputProcessor;
        }
        public void Activate(){}; //Start state
        public void Start(){}; //S0 state
        public void PayCredit(){}; //S1 state
        public void PayCash(){}; //S1 state
        public void Reject(){}; //S2 state
        public void Cancel(){}; //S3 state
        public void Approved(){}; //S2 state
        public void StartPump(){}; //S4 state
        public void Pump(){}; //S5 state
        public void StopPump(){}; //S5 state
        public void SelectGas(int G){}; //S3 state
        public void Receipt(){}; //S6 state
        public void NoReceipt(){} //S6 state
        public void StoreData(){} //S1 state
        public void StoreCash(){} //S1 state
        public void SetPrice(int g) {}; //S3 state

}
```

10.s0

```
public class S0  extends State{
        /**
        *
        *
        * This class extends State and implements methods for S0 state
        *
        */

        //forward to outputprocessor
        public void Start(){
        op.DisplayPayMsg();
        }
        }
```

11.s1

```
package SSAproject;
/**
*
*
* This class extends State and implements methods for S1 state
*
*/

public class S1 extends State{
        public void PayCredit(){
        }
        //forward to outputprocessor
        public void PayCash(){

        op.DisplayMenu();
        }
        //forward to outputprocessor
        public void StoreCash(){
        op.StoreCash();
        }


}
```

12.s2

```
package SSAproject;
/**
*
*
* This class extends State and implements methods for S2 state
*
*/
public class S2 extends State {
          //forward to outputprocessor
          public void Approved(){

                    op.DisplayMenu();
          }
          //forward to outputprocessor
          public void Reject(){
          op.DisplayRejectMsg();
          }
          }
```

13.s3

```
package SSAproject;
/**
*
*
* This class extends State and implements methods for S3 state
*
*/
public class S3 extends State{


          //forward to outputprocessor
          public void Cancel(){
          op.DisplayCancelMsg();
          }
          //forward to outputprocessor
          public void SelectGas(int g){
          op.SetPrice(g);
          }
}
```

14.s4

```
package SSAproject;
/**
*
*
* This class extends State and implements methods for S3 state
*
*/

public class S4 extends State {
        public void StartPump(){
                op.SetInitialValues();
                op.DisplayReadyMsg();
                }

}
```

15.s5

```
package SSAproject;
/**
*
*
* This class extends State and implements methods for S3 state
*
*/

public class S5 extends State {

                //forward to outputprocessor
                public void Pump(){
                op.PumpGasUnit();
                op.DisplayGasPumpedMsg();
                }
                //forward to outputprocessor
                public void StopPump(){
                op.DisplayStopMsg();
                }
                }
```

16.s6

```
package SSAproject;
/**
*
*
* This class extends State and implements methods for S3 state
```

```
*
*/

public class S6 extends State{
	//forward to outputprocessor
	public void Receipt(){
	op.PrintReceipt();
	}
	public void NoReceipt(){
	//System.out.println("Transaction over, No receipt");
	}

}

// Abstract Factory Pattern starts here:
```

17.AbstractFactory

```
package SSAproject;
/*
* This class defines  methods that will be implemented by the concretefactory classes 1 and 2
inheriting it.
*
*/

public abstract class AbstractFactory {
	public abstract StoreData getStoreData();
	public abstract DisplayMenu getDisplayMenu();
	public abstract PumpGasUnit getPumpGasUnit();
	public abstract PrintReceipt getPrintReceipt();
	public abstract StoreCash getStoreCash();
	public abstract DisplayGasPumpedMsg getDisplayGasPumpedMsg();
	public abstract SetInitialValues getSetInitialValues();
	public abstract SetPrice getSetPrice();
	public abstract Messages getMessages();
	public abstract DisplayPayMsg getDisplayPayMsg();


}
```

18.concrete Factory1

```
package SSAproject;
/**
*
*
* Implements all the methods from abstractfactory
* which will be used by GasPump1 and its methods to get specific concrete objects
*
*/
```

Abinaya Janakan
A20376287

```
public class ConcreteFactory1 extends AbstractFactory {
	//get object of type Data2
	public DataStore1 getData() {
	DataStore1 ds1 = new DataStore1();
	return ds1;
	}
	//get object of type StoreData2
	@Override
	public StoreData getStoreData() {
	StoreData1 storedata1 = new StoreData1();
	return storedata1;
	}
	//get object of type DisplayMenu2
	@Override
	public DisplayMenu getDisplayMenu() {
	DisplayMenu1 displayMenu1 = new DisplayMenu1();
	return displayMenu1;
	}
	//get object of type PumpGasUnit2
	@Override
	public PumpGasUnit getPumpGasUnit() {
	PumpGasUnit1 pumpGasUnit1 = new PumpGasUnit1();
	return pumpGasUnit1;
	}
	//get object of type PrintReceipt2
	@Override
	public PrintReceipt getPrintReceipt() {
	PrintReceipt1 printReceipt1 = new PrintReceipt1();
	return printReceipt1;
	}
	//will ret null because cash is not used in GP2
	@Override
	public StoreCash getStoreCash() {
	return null;
	}
	//get object of type DisplayGasPumpedMsg2
	@Override
	public DisplayGasPumpedMsg getDisplayGasPumpedMsg() {
	DisplayGasPumpedMsg1 displayGasPumpedMsg1 = new DisplayGasPumpedMsg1();
	return displayGasPumpedMsg1;
	}
	//get object of type SetInitialValues2
	@Override
	public SetInitialValues getSetInitialValues() {
	SetInitialValues1 setInitialValues1 = new SetInitialValues1();
	return setInitialValues1;
	}
	//get object of type SetPrice2
	@Override
	public SetPrice getSetPrice() {
	SetPrice1 setPrice1 = new SetPrice1();
	return setPrice1;
	}
	//get object of type Messages
	@Override
	public Messages getMessages() {
	// TODO Auto-generated method stub
	Messages messages= new Messages();
	return messages;
	}

	//get object of type DisplayPayMsg2
	@Override
	public DisplayPayMsg getDisplayPayMsg() {
	// TODO Auto-generated method stub
	DisplayPayMsg1 displayPayMsg1 = new DisplayPayMsg1();
	return displayPayMsg1;
	}
}
```

19.Concrete Factory2

```
package SSAproject;
/**
*
*
* Implements all the methods from abstractfactory
* which will be used by GasPump2 and its methods to get specific concrete objects
*
*/
public class ConcreteFactory2 extends AbstractFactory{
	//get object of type Data3
	public DataStore2 getData() {
	DataStore2 ds2 = new DataStore2();
	return ds2;
	}
	//get object of type StoreData3

	public StoreData getStoreData() {
	StoreData2 storedata2 = new StoreData2();
	return storedata2;
	}
	//get object of type DisplayMenu3
```

```
public DisplayMenu getDisplayMenu() {
DisplayMenu2 displayMenu2= new DisplayMenu2();
return displayMenu2;
}
//get object of type PumpGasUnit3

public PumpGasUnit getPumpGasUnit() {
PumpGasUnit2 pumpGasUnit2 = new PumpGasUnit2();
return pumpGasUnit2;
}
//get object of type PrintReceipt3

public PrintReceipt getPrintReceipt() {
PrintReceipt2 printReceipt2 = new PrintReceipt2();
return printReceipt2;
}
//get object of type StoreCash3

public StoreCash getStoreCash() {
StoreCash2 storeCash2 = new StoreCash2();
return storeCash2;
}
//get object of type DisplayGasPumpedMsg3

public DisplayGasPumpedMsg getDisplayGasPumpedMsg() {
DisplayGasPumpedMsg2 displayGasPumpedMsg2 = new DisplayGasPumpedMsg2();
return displayGasPumpedMsg2;
}
//get object of type SetInitialValues3

public SetInitialValues getSetInitialValues() {
SetInitialValues2 setInitialValues2= new SetInitialValues2();
return setInitialValues2;
}
//get object of type SetPrice3

public SetPrice getSetPrice() {
SetPrice2 setPrice2 = new SetPrice2();
return setPrice2;
}
//get object of type Messages

public Messages getMessages() {
// TODO Auto-generated method stub
Messages messages= new Messages();
```

```
        return messages;
        }

        public DisplayPayMsg getDisplayPayMsg() {
        // TODO Auto-generated method stub
        DisplayPayMsg2 displayPayMsg2 = new DisplayPayMsg2();
        return displayPayMsg2;
        }

}

//Strategy patten starts here
```

20.Messages

```
package SSAproject;
/**
*
*
* includes methods that will print  messages for the gas pumps 1 and 2
*
*/

public class Messages {
        public void displayRejectMsg() {
                System.out.println("\n the Card  has been rejected ");
                }
                //ready msg
                public void displayReadyMsg() {
                System.out.println("\n The pump is ready ");
                }
                //stop msg
                public void displayStopMsg() {
                System.out.println("The gas has been pumped");
                }
                //cancel msg
                public void displayCancelMsg() {
                System.out.println("\n The transaction has been cancelled");
                }

}
```

21.StoreData

```
package SSAproject;
/**
*
*
* this is an Abstract class whose methods will be called by concrete classes for implementation
*
```

```
*/

public abstract class StoreData {
	DataStore ds;
	public abstract void storeData();
	public void setData(DataStore ds) {
	this.ds = ds;
	}

}
```

21.StoreData1

```
package SSAproject;
/**
*
*
* extends abstract class StoreData and implements it for GasPump1
*
*/

public class StoreData1 extends StoreData {
	public void storeData() {
		float a = ds.getFloatTempA();
		float b = ds.getFloatTempB();
		ds.setPriceRegular(a);
		ds.setPriceSuperPremium(b);

		}
}
```

22.StoreData2

```
package SSAproject;
/**
*
*
* extends abstract class StoreData and implements it for GasPump2
*
*/

public class StoreData2 extends StoreData {
	//stores user input values for premium and regular gas
	public void storeData() {
		float a = ds.getFloatTempA();
		float b = ds.getFloatTempB();
		ds.setPriceRegular(a);
		ds.setPriceSuperPremium(b);
		}
}
```

23.DisplayMenu

```
package SSAproject;

public abstract class DisplayMenu {
	/**
	*
	*
	* Abstract class whose methods will be called by concrete classes for implementation
	*
	*/

	public abstract void displayMenu();
	}
```

24.DisplayMenu1

```
package SSAproject;
/**
*
*
* extends display menu and implements DisplayMenu for GasPump1
*
*/
public class DisplayMenu1 extends DisplayMenu {
	//display choice of gas
	@Override
	public void displayMenu() {
	System.out.println(" Select Gas: 1. Regular OR 2. Super " );
	}

}
```

25.DisplayMenu2

```
package SSAproject;
/**
*
*
* extends display menu and implements DisplayMenu for GasPump2
*
*/
public class DisplayMenu2 extends DisplayMenu{

		//display choice of gas
		@Override
		public void displayMenu() {
```

```
                    System.out.println(" Select Gas: 1. Regular OR 2. Premium " );
                    }
}
```

26.PumpGasUnit

```
package SSAproject;
/**
*
*
* Abstract class whose methods will be called by concrete classes for implementation
*
*/

public abstract class PumpGasUnit {
          public abstract void pumpGasUnit();

}
```

27.PumpGasUnit1

```
package SSAproject;
/**
*
*
* extends PumpGasUnit and implements PumpGasUnit for GasPump1
*
*/
public class PumpGasUnit1 extends PumpGasUnit {
          //mention that a unit of gas has been pumped
          @Override
          public void pumpGasUnit() {
          System.out.println("One gallon of gas has been pumped from gas pump1 " );
          }

}
```

28.PumpGasUnit2

```
package SSAproject;
/*
*
* extends PumpGasUnit and implements PumpGasUnit for GasPump2
*
*/

public class PumpGasUnit2 extends PumpGasUnit {
          //mention that a unit of gas has been pumped
          @Override
          public void pumpGasUnit() {
          System.out.println("One litre of gas has been pumped from gas pump1 " );
```

```
        }

}
```

29.DisplayPayMsg

```
package SSAproject;
/**
*
*
* Abstract class whose methods will be called by concrete classes for implementation
*
*/

public abstract class DisplayPayMsg {
        public abstract void displayPayMsg();


}
```

30.DisplayPayMsg1

```
package SSAproject;
/**
*
*
* extends abstract class and implements DisplayPayMsg for GasPump1
*
*/

public class DisplayPayMsg1 extends DisplayPayMsg {
        public void displayPayMsg() {
                // TODO Auto-generated method stub
                System.out.println("Insert your Credit Card");
                }

}
```

31.DisplayPayMsg2

```
package SSAproject;
/**
*
*
* extends abstract class and implements DisplayPayMsg for GasPump2
*
*/

public class DisplayPayMsg2 extends DisplayPayMsg {
        public void displayPayMsg() {

        // TODO Auto-generated method stub
```

```
		System.out.println("Pay cash");
		}


}
```

32.SetPrice

```
package SSAproject;
/**
*
*
* Abstract class whose methods will be called by concrete classes for implementation
*
*/

public abstract class SetPrice {
		public abstract void setPrice(DataStore ds);

}
```

33.SetPrice1

```
package SSAproject;
/**
*
*
* extends SetPrice and implements SetPrice for GasPump1
*
*/

public class SetPrice1 extends SetPrice{
		@Override
		public void setPrice(DataStore ds) {
		// TODO Auto-generated method stub
		System.out.println("Price set at "+ ((DataStore1)ds).getPrice());
		}


}
```

34.SetPrice2

```
package SSAproject;
/**
*
*
* extends SetPrice and implements SetPrice for GasPump2
*
*/

public class SetPrice2 extends SetPrice{
	@Override
	public void setPrice(DataStore ds) {
	// TODO Auto-generated method stub
	System.out.println("Price set at "+ ((DataStore2)ds).getPrice());
	}
}
```

35.SetInitialValues

```
package SSAproject;
/*
*
* Abstract class whose methods will be called by concrete classes for implementation
*
*/

public abstract class SetInitialValues {
	public abstract void setInitialValues(DataStore ds);

}
```

36.SetInitialValues1

```
package SSAproject;
/**
*
*
* extends abstract class and implements SetInitialValues for GasPump1
*
*/
public class SetInitialValues1 extends SetInitialValues {
	//set units of gas pumped and total cost to zero
	@Override
	public void setInitialValues(DataStore ds) {
	ds.setG(0);
	ds.setTotal(0);
	}


}
```

37.SetInitilValues2

```
package SSAproject;
/**
*
*
* extends abstract class and implements SetInitialValues for GasPump2
*
*/
public class SetInitialValues2 extends SetInitialValues {
	//set units of gas pumped and total cost to zero
	@Override
	public void setInitialValues(DataStore ds) {
	ds.setG(0);
	ds.setTotal(0);
	}
	}
```

38.DisplayGasPumpedMsg

```
package SSAproject;
/**
*
*
* Abstract class whose methods will be called by concrete classes for implementation
*
*/

public abstract class DisplayGasPumpedMsg {
	public abstract void displayGasPumpedMsg(DataStore ds);
	}
```

39.DisplayGasPumpedMsg1

```
package SSAproject;
/**
*
*
* Abstract class whose methods will be called by concrete classes for implementation
*
*/

public abstract class DisplayGasPumpedMsg {
	public abstract void displayGasPumpedMsg(DataStore ds);
	}
```

40.DisplayGasPumpedMsg2

```
package SSAproject;
/**
*
*
* Extends Abstract class DisplayGasPumpedMsg and implements methods in
DisplayGasPumpedMsg for GasPump2
*
*/

public class DisplayGasPumpedMsg2 extends DisplayGasPumpedMsg {
	//prnt the number of liters pumped
	@Override
	public void displayGasPumpedMsg(DataStore ds) {
	System.out.println("Gas Pumped : "+ ds.getG() +" liters");
	}

}
```

41.StoreCash

```
package SSAproject;
/**
*
*
* Abstract class whose methods will be called by concrete classes for implementation
*
*/

public abstract class StoreCash {
	DataStore ds;
	public abstract void storeCash();
	public void setCash(DataStore ds) {
	this.ds = ds;
	}
}
```

41.StoreCash2

```
package SSAproject;
/**
*
*
* Extends Abstract class and implements StoreCash for GasPump3
*
*/
public class StoreCash2 extends StoreCash {
	//store user given cash in data3
	@Override
	public void storeCash() {
	float c = ds.getFloatTempC(); //System.out.println("float temp c is"+ c);
	ds.setFloatCash(c);
	}

}
```

42.PrintReceipt

```
package SSAproject;
/**
*
*
```

```
* Abstract class whose methods will be called by concrete classes for implementation of
PrintReceipt
*
*/

public abstract class PrintReceipt {
	public abstract void printReceipt(DataStore ds);

}
```

43.PrintReceipt1

```
package SSAproject;
/*
*
* Extends Abstract class and implementss PrintReceipt for GasPump1
*
*/

public class PrintReceipt1 extends PrintReceipt {
	//print receipt by showing number of gallons pumped and total cost
	@Override
	public void printReceipt(DataStore ds) {
	System.out.println("GasPump1 receipt: "+ ds.getG() +"gallons for $"+ ds.getTotal());
	}

}
```

44.PrintReceipt2

```
package SSAproject;
/**
*
*
* Extends Abstract class and implementats PrintReceipt for GasPump3
*
*/
public class PrintReceipt2 extends PrintReceipt {
	//print receipt by showing number of liters pumped and total cost
	@Override
	public void printReceipt(DataStore ds) {
	System.out.println("GasPump2 receipt: "+ ds.getG() +"liters for $"+ ds.getTotal());
	float return_cash=ds.getTotal()-ds.getFloatCash();
	System.out.println("GasPump2 return cash:  $"+ return_cash);

	}

}
```

Abinaya Janakan
A20376287