MDA-EFSM PROJECT REPORT BY

RISHABH SHAH (A20381143)

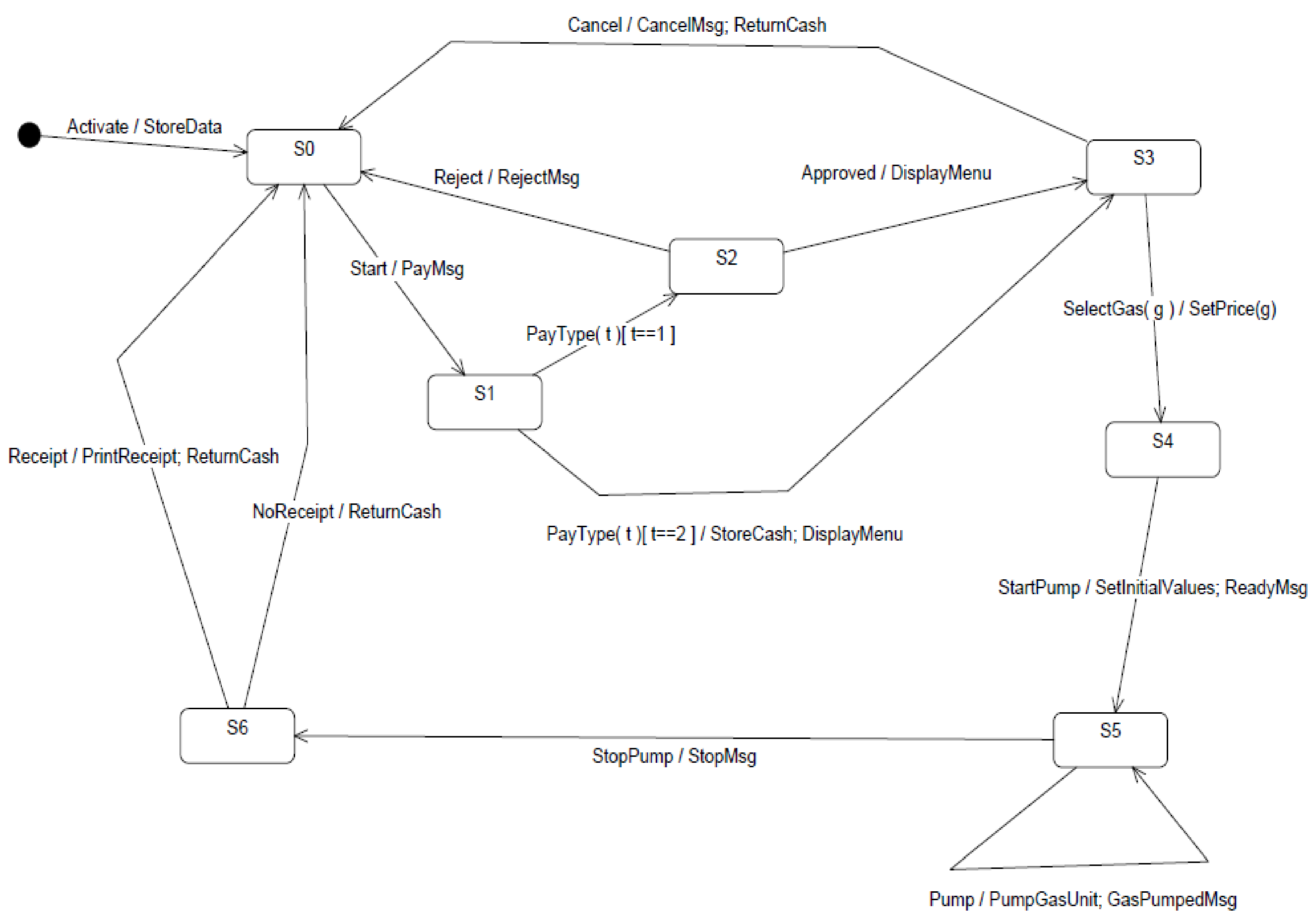
ILLINOIS INSTITUTE OF TECHNOLOGY

1. MDA-EFSM model for the GasPump components
   a. A list of meta events for the MDA-EFSM
      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()

   b. A list of meta actions for the MDA-EFSM with their descriptions.

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

c. A state diagram of the MDA-EFSM


Cancel / CancelMsg; ReturnCash
Activate / StoreData
S0
Reject / RejectMsg
Approved / DisplayMenu
S3
S2
Start / PayMsg
PayType(t)[t==1]
SelectGas(g) / SetPrice(g)
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

d. Pseudo-code of all operations of Input Processors of GasPump-1 and GasPump-2.
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();

}
```

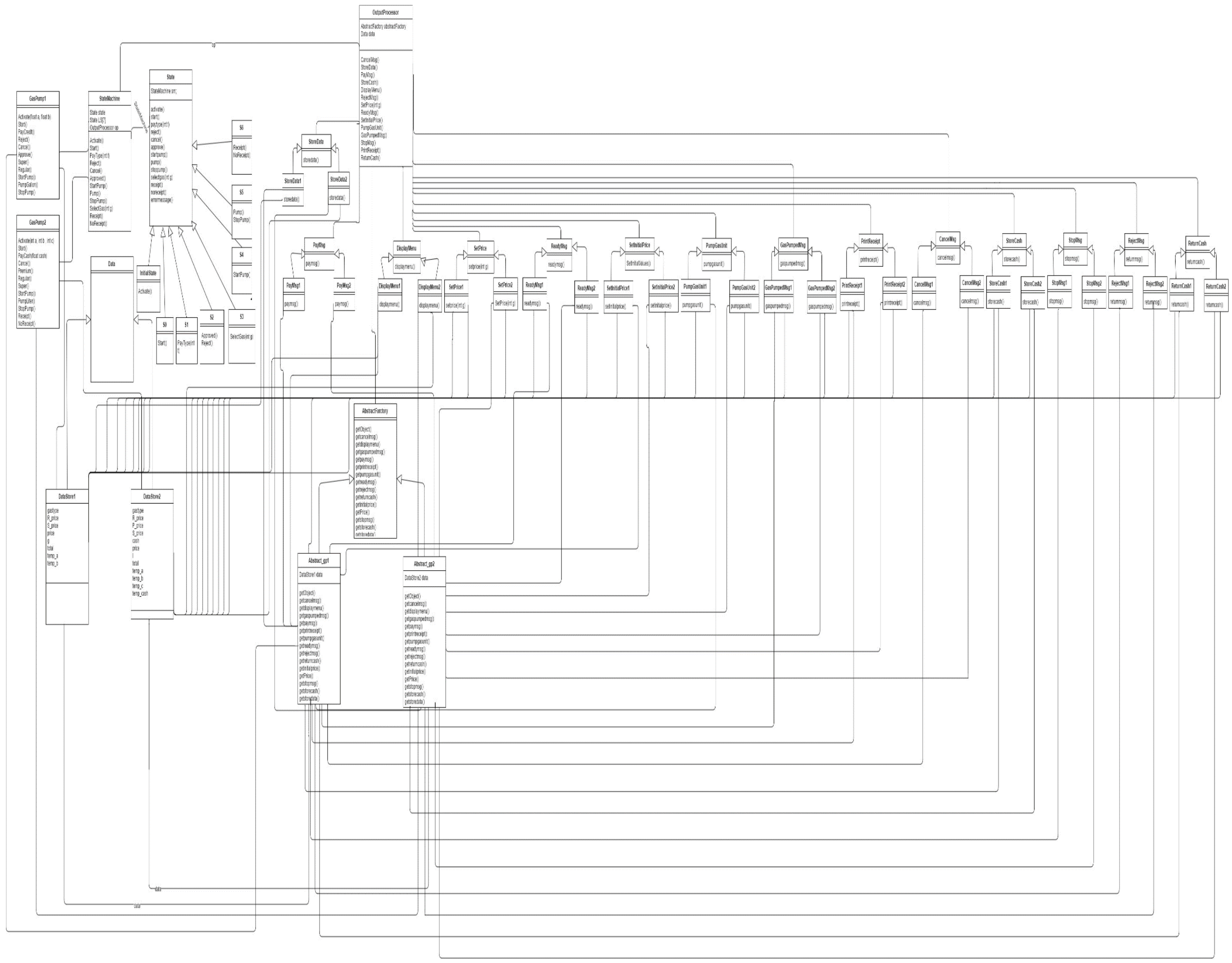
```
PayCredit() { m->PayType(1);

}

Reject() { m->Reject();

}

Cancel() { m->Cancel();

}

Approved() { m->Approved();

}

Super() { m->SelectGas(2)

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

}

StartPump() { m->StartPump();

}

PumpGallon() {
m->Pump();

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

}
```

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) {
```

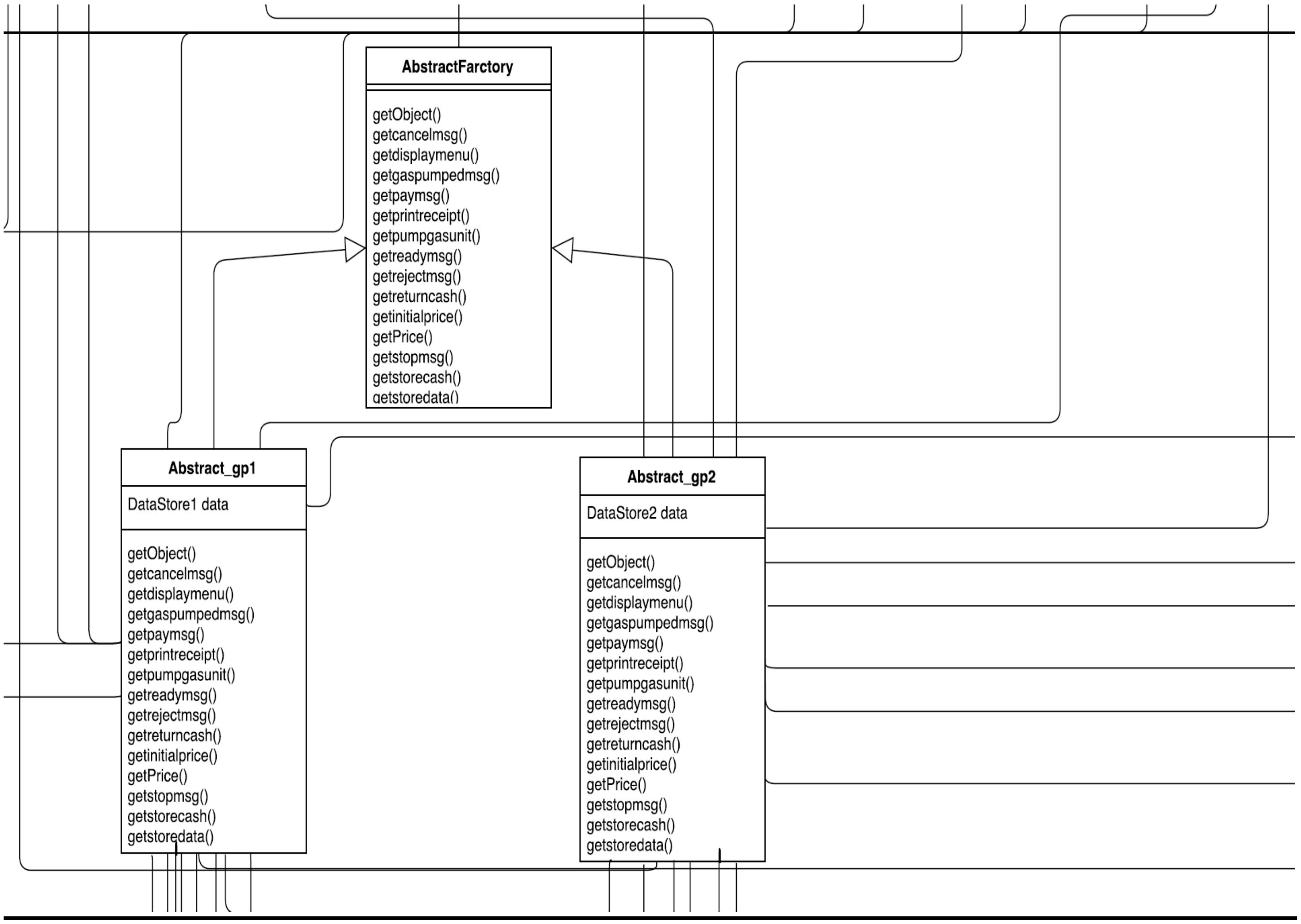
```
d->temp_cash=c; m->PayType(2)
}

}

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();

}
```

CLASS DIAGRAM: -

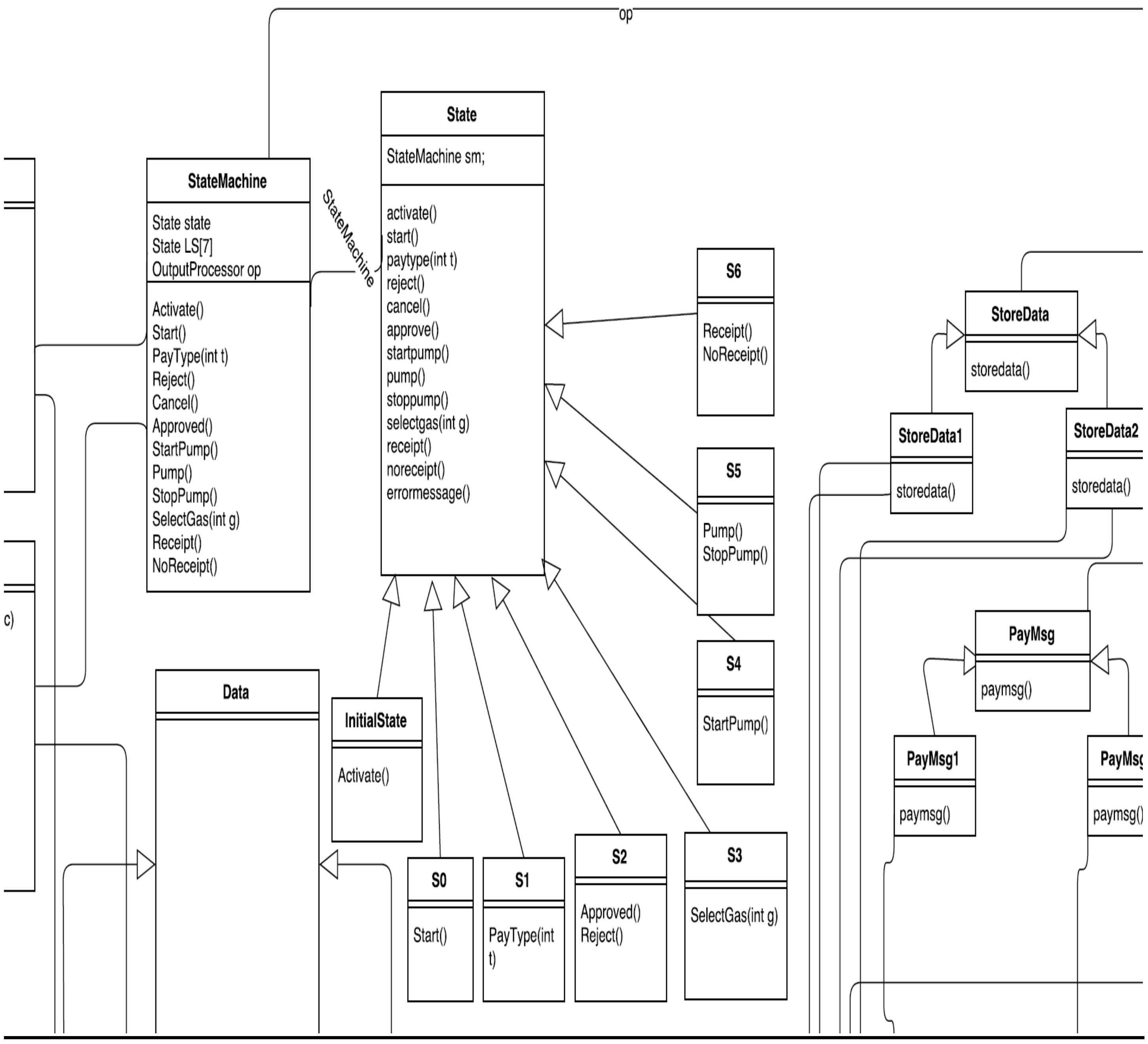


GasPump1
Activate(float a, float b)
Start()
PayCredit()
Reject()
Cancel()
Approve()
Super()
Regular()
StartPump()
PumpGallon()
StopPump()
GasPump2
Activate(int a, int b , int c)
Start()
PayCash(float cash)
Cancel()
Premium()
Regular()
Super()
StartPump()
PumpLiter()
StopPump()
Receipt()
NoReceipt()
StateMachine
State state
State LS[7]
OutputProcessor op
Activate()
Start()
PayType(int t)
Reject()
Cancel()
Approved()
StartPump()
Pump()
StopPump()
SelectGas(int g)
Receipt()
NoReceipt()
StateMachine
op
State
StateMachine sm;
activate()
start()
paytype(int t)
reject()
cancel()
approve()
startpump()
pump()
stoppump()
selectgas(int g)
receipt()
noreceipt()
errormessage()
S6
Receipt()
NoReceipt()
S5
Pump()
StopPump()
S4
StartPump()
Data
InitialState
Activate()
S0
Start()
S1
PayType(int t)
S2
Approved()
Reject()
S3
SelectGas(int g)
OutputProcessor
AbstractFactory abstractFactory
Data data
CancelMsg()
StoreData()
PayMsg()
StoreCash()
DisplayMenu()
RejectMsg()
SetPrice(int g)
ReadyMsg()
SetInitialPrice()
PumpGasUnit()
GasPumpedMsg()
StopMsg()
PrintReceipt()
ReturnCash()
StoreData
storedata()
StoreData1
storedata()
StoreData2
storedata()
PayMsg
paymsg()
PayMsg1
paymsg()
PayMsg2
paymsg()
DisplayMenu
displaymenu()
DisplayMenu1
displaymenu()
DisplayMenu2
displaymenu()
SetPrice
setprice(int g)
SetPrice1
setprice(int g)
SetPrice2
SetPrice(int g)
ReadyMsg
readymsg()
ReadyMsg1
readymsg()
ReadyMsg2
readymsg()
SetInitialPrice
SetInitialValues()
SetInitialPrice1
setinitialprice()
SetInitialPrice2
setinitialprice()
PumpGasUnit
pumpgasunit()
PumpGasUnit1
pumpgasunit()
PumpGasUnit2
pumpgasunit()
GasPumpedMsg
gaspumpedmsg()
GasPumpedMsg1
gaspumpedmsg()
GasPumpedMsg2
gaspumpedmsg()
PrintReceipt
printreceipt()
PrintReceipt1
printreceipt()
PrintReceipt2
printreceipt()
CancelMsg
cancelmsg()
CancelMsg1
cancelmsg()
CancelMsg2
cancelmsg()
StoreCash
storecash()
StoreCash1
storecash()
StoreCash2
storecash()
StopMsg
stopmsg()
StopMsg1
stopmsg()
StopMsg2
stopmsg()
RejectMsg
returnmsg()
RejectMsg1
returnmsg()
RejectMsg2
returnmsg()
ReturnCash
returncash()
ReturnCash1
returncash()
ReturnCash2
returncash()
AbstractFarctory
getObject()
getcancelmsg()
getdisplaymenu()
getgaspumpedmsg()
getpaymsg()
getprintreceipt()
getpumpgasunit()
getreadymsg()
getrejectmsg()
getreturncash()
getinitialprice()
getPrice()
getstopmsg()
getstorecash()
getstoredata()
DataStore1
gastype
R_price
S_price
price
g
total
temp_a
temp_b
DataStore2
gastype
R_price
P_price
S_price
cash
price
l
total
temp_a
temp_b
temp_c
temp_cash
Abstract_gp1
DataStore1 data
getObject()
getcancelmsg()
getdisplaymenu()
getgaspumpedmsg()
getpaymsg()
getprintreceipt()
getpumpgasunit()
getreadymsg()
getrejectmsg()
getreturncash()
getinitialprice()
getPrice()
getstopmsg()
getstorecash()
getstoredata()
Abstract_gp2
DataStore2 data
getObject()
getcancelmsg()
getdisplaymenu()
getgaspumpedmsg()
getpaymsg()
getprintreceipt()
getpumpgasunit()
getreadymsg()
getrejectmsg()
getreturncash()
getinitialprice()
getPrice()
getstopmsg()
getstorecash()
getstoredata()
data
data

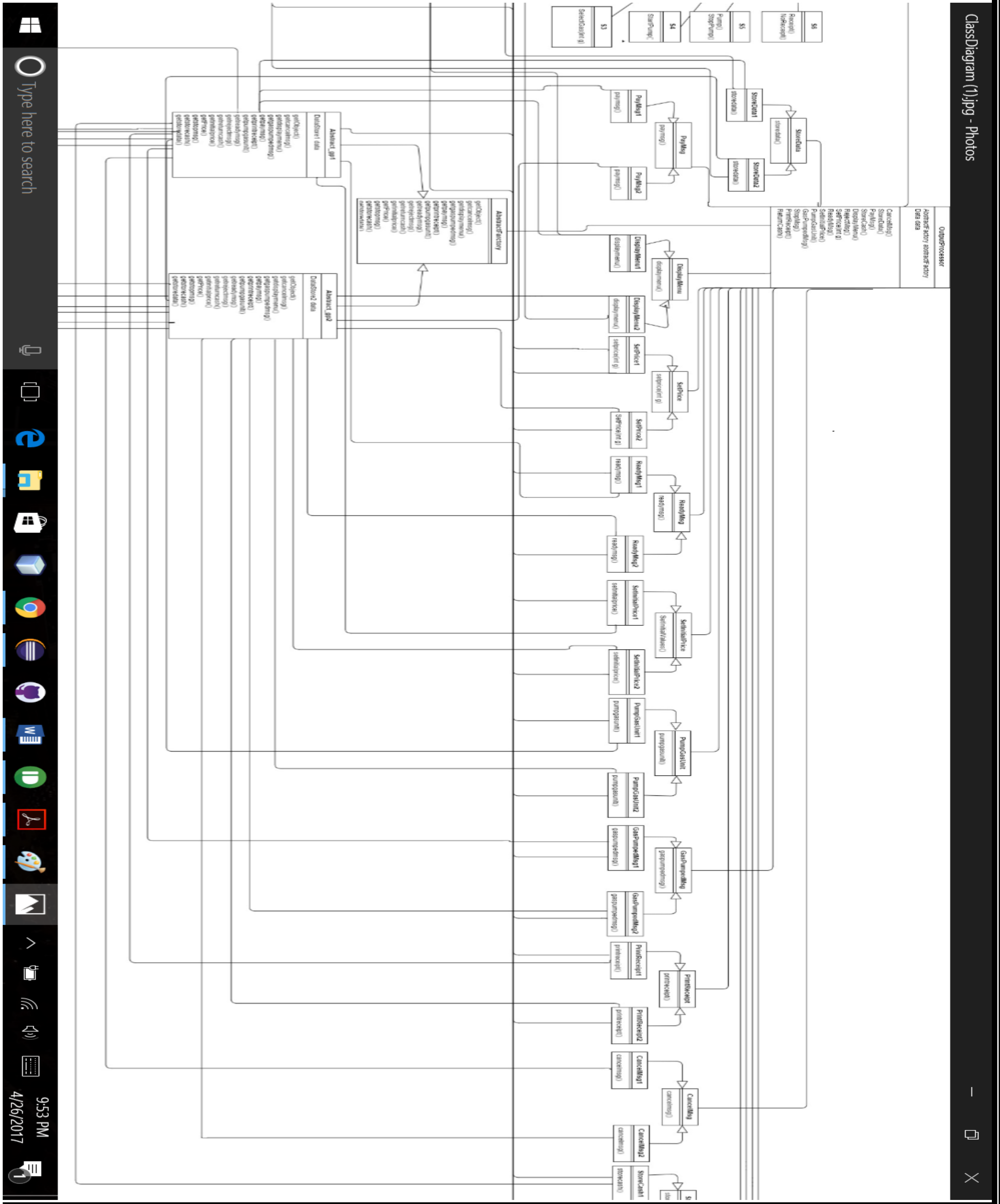
Abstract Factory: -



State Pattern: -



Strategy Pattern: -



<u>List of classes:</u>

1. GasPump_1 //providing all gas pump operations for gas pump 1.

void Activate(float a, float b)

//store gas price and activate pump, initiate same method in MDA-EFSM class.

void Start()

//show start menu, initiate same method in MDA-EFSM.

void PayCredit()

//pay by credit, needs approval, initiate same method in MDA-EFSM.

void Approve()

//approve credit card, initiate same method in MDA-EFSM.

void Cancel()

//cancel ongoing process, initiate same method in MDA-EFSM.

void regular()

//select regular gas, initiate same method in MDA-EFSM.

void Super()

//select Super gas, initiate same method in MDA-EFSM.

void PumpGallon()

//pump 1 gallon each time, initiate pump() method in MDA-EFSM,

void Reject()

//credit card has been rejected.

void StartPump()

//show read message to start pump, initiate same method in
MDA-EFSM.

void StopPump()

//show stop message to stop pump, initiate same method in
MDA-EFSM.

2. GasPump_2 //providing all gas pump operations for gas pump 2.

void activate(int a, int b, int c)

//activate with regular and super prices stored in data store.

void Start()

//show start menu, initiate same method in MDA-EFSM.

void payCash(float cash)

//pay amount of c by cash.

void cancel()

//cancel ongoing process, initiate same method in MDA-EFSM.

void regular()

//select regular gas, initiate same method in MDA-EFSM.

void Super()

//select Super gas, initiate same method in MDA-EFSM.

void premium()

//select premium gas, initiate same method in MDA-EFSM.

void startPump()

//show read message to start pump, initiate same method in MDA-EFSM.

void pumpLiter()

//pump 1 liter each time, initiate pump() method in MDA-EFSM, if pay by cash, initiate stopPump() in MDA-EFSM when there's not enough balance, initiate printReceipt() in MDA-EFSM after pump stopped.

void StopPump()

//show stop message to stop pump, initiate same method in MDA-EFSM.

void Receipt()

//finished pump with receipt printed, initiate same method in

MDA-EFSM.

void NoReceipt()

//finished pump without print receipt, initiate same method in

MDA-EFSM.

3. **DataStore //abstract class providing access to subclasses**.

4. **DataStore1 //store data for gas pump 1.**

Note: In DataStore1 for the sake of shorter code and simplicity, fields are accessed directly, instead of through getters and setters.

public String gasType; //this is for storing the type of gas 1 for regular , 2 for super and 3 for premium.

public float R_price; //this is for storing the price of regular gas.

public float S_price; //this is for storing the price of super gas.

public float price; //this is for storing price.

public int g;

public float total; // this variable is for storing total price.

// temporary variables

public float a;

public float b;

5. DataStore_2 //store data for gas pump 2.

Note: In DataStore2 for the sake of shorter code and simplicity, fields are accessed directly, instead of through getters and setters.

```
public String   gasType; // this is used for storing the gastype.
public int      R_price; // this is use to store the price of regular gas.
public int      S_price; // this is use to store the price of super gas.
public int      P_price; // this is use to store the price of premium gas.
public float    cash; // this is use to store cash.
public int      price;
public int      l;
public int      total; // this is use to store total.


// temporary variables
public int a; //regular
public int b; // super
public int c; // premium
public float temp_cash;
```

6. Abstract_Factory //abstract classes grouping factory classes and provide access to them.

7. Abstractgp_1 //factory class for gas pump 1.

```
public getcancelmsg() {
            // TODO Auto-generated method stub
            // returns the cancelmsg class which describes the cancel message for gaspump 1.
                  }
public getdisplaymenu() {
            // TODO Auto-generated method stub
            // returns displaymenu class which displays menu for gaspump 1.
                  }
```

```
public getgaspumpedmsg() {
                    /*
                     * Returns the GasPumpedMsg class that performs action for displaying the message that informs
                     * the user that a unit of gas has been pumped using GasPump1
                     *  */
          }
public getpaymsg() {
                    // TODO Auto-generated method stub
                    // returns payment message which is appropriate for gaspump 1
          }
public getprintreceipt() {
                    // TODO Auto-generated method stub
                    //  returns printreceipt class which is responsible for printing message for gaspump 1.
                              }

public getpumpgasunit() {
                    // TODO Auto-generated method stub
                    // returns a PumpUnitGas class which pumps 1 gallon of gas at a time.
                              }
public getreadymsg() {
                    // TODO Auto-generated method stub
                    // returns ReadyMsg which is used to notify user that they can start pumping gas.
                              }

Public getrejectmsg() {
                    // TODO Auto-generated method stub
                    // returns RejectMsg which notifies user that due to some error they cannot pursue further.
                              }
public ReturnCash getreturncash() {
                    // TODO Auto-generated method stub
                    // gaspump 1 doesnot support any cash payment so this method will return nothing.
```

```
                                }
public getinitialprice() {
                    // TODO Auto-generated method stub
                    // returns SetInitialPrice class which is used to set the initial values before the starting of puming
of gas.
          }
public SetPrice getPrice() {
                    // TODO Auto-generated method stub
                    // returns SetPrice class which is used to set the value of gas according to the requirements of the
gaspump 1.
          }
public getstopmsg() {
                    // TODO Auto-generated method stub
                    // returns StopMsg class which notifies users that pumping of gas has been stopped.
                    }
public getstorecash() {
                    // TODO Auto-generated method stub
                    // returns StoreCash object appropriate for GasPump 1
          }
public getstoredata() {
                    // TODO Auto-generated method stub
                    // returns the StoreData action strategy class appropriate for storing needed input data
                    }
```

Abstractgp_2 //factory class for gas pump 1.

```
public getcancelmsg() {
                    // TODO Auto-generated method stub
                    // returns the cancelmsg class which describes the cancel message for gaspump 2.
                                }
```

```
public getdisplaymenu() {
                    // TODO Auto-generated method stub
                    // returns displaymenu class which displays menu for gaspump 2.
                              }

public getgaspumpedmsg() {
                    /*
                     * Returns the GasPumpedMsg class that performs action for displaying the message that informs
                     * the user that a unit of gas has been pumped using GasPump2
                     *  */
          }
public getpaymsg() {
                    // TODO Auto-generated method stub
                    // returns payment message which is appropriate for gaspump 2
          }
public getprintreceipt() {
                    // TODO Auto-generated method stub
                    //  returns printreceipt class which is responsible for printing message for gaspump 2.
                              }

public getpumpgasunit() {
                    // TODO Auto-generated method stub
                    // returns a PumpUnitGas class which pumps 1 liter of gas at a time.
                              }
public getreadymsg() {
                    // TODO Auto-generated method stub
                    // returns ReadyMsg which is used to notify user that they can start pumping gas.
                              }

Public getrejectmsg() {
                    // TODO Auto-generated method stub
                    // returns RejectMsg which notifies user that due to some error they cannot pursue further.
                              }
public ReturnCash getreturncash() {
                    // TODO Auto-generated method stub
                    // gaspump 2 will return the cash amount which is left to disburse.                      }
public getinitialprice() {
                    // TODO Auto-generated method stub
                    // returns SetInitialPrice class which is used to set the initial values before the starting of puming
of gas.
          }
public SetPrice getPrice() {
                    // TODO Auto-generated method stub
                    // returns SetPrice class which is used to set the value of gas according to the requirements of the
gaspump 1.
          }
public getstopmsg() {
                    // TODO Auto-generated method stub
                    // returns StopMsg class which notifies users that pumping of gas has been stopped.
                    }
public getstorecash() {
                    // TODO Auto-generated method stub
                    // returns StoreCash object appropriate for GasPump 2
          }
public getstoredata() {
                    // TODO Auto-generated method stub
                    // returns the StoreData action strategy class appropriate for storing needed input data
                    }
```

<u>MDA.EFSM PACKAGE CLASSES: -</u>

a. Initial State: -
   // this is the initial state in MDA.EFSM
   // it has activate() meta event.

b. S0
   //this is second state in MDA.EFSM
   // it has start() meta event

c. S1
   // this is third state in MDA.EFSM

d. S2
   // this is fourth state in MDA.EFSM

e. S3
   /// this is fifth state in MDA.EFSM

f. S4
   //// this is sixth state in MDA.EFSM

g. S5
   /// this is seventh state in MDA.EFSM

h. S6
   /// this is eighth state in MDA.EFSM

i. State // this is a state class in MDA.EFSM

➔ This class is the abstract State superclass in the De-centralized State Design Pattern.
➔ * In this State methods are initially defined to print a "errormessage" message.
➔ * Each state subclass inherits these methods and overrides the appropriate ones.
➔ * This means that methods that do not get overridden will print a "errormessage" message
➔ * if they are called from a state that does not allow them to be called

j. StateMachine // this is an // it serves as a VM class in <u>De</u>-centralized state design pattern.
   //state classes are use for performing actions and state transitions. It also consist of getters and setters methods.

Strategy Patterns: -

CancelMsg() //abstract class grouping subclasses and providing access.

- ➔ CancelMsg1()

Getcancelmsg()

This class is for gaspump 1.

//display cancel message.

- ➔ CancelMsg2()

Getcancelmsg()

This class is for gaspump 2.

//display cancel message

DisplayMenu() //abstract class grouping subclasses and providing access.

- ➔ DisplayMenu1()

```
This class is use to print the menu.
/ *  It is also use to print the credit card approval message.
 *  displaymenu() method is use to show the menu of available gases for gaspump1.
 *  */
```

- ➔ DisplayMenu2()

```
/*
 * This class is use to print the menu.
 *  displaymenu() method is use to show the menu of available gases for gaspump2.
 *  */
```

GasPumpedMsg() //abstract class grouping subclasses and providing access.

- ➔ GasPumpedMsg1()
  //GasPump1 action responsible for printing a message that gas has been pumped.
  Gaspumpedmsg()
  // this method is use to show that 1 gallon of gas has been pumped.

- ➔ GasPumpedMsg2()
  //GasPump1 action responsible for printing a message that gas has been pumped.
  Gaspumpedmsg()
  // this method is use to show that 1 liter of gas has been pumped.

PayMsg() //abstract class grouping subclasses and providing access.

- ➔ payMsg1()
  // GasPump1 method used to prompt message to select payment type.
  Void paymsg() // method use to prompt message.

- ➔ PayMsg2()
  // GasPump2 method used to prompt message to select payment type.
  Void paymsg() // method use to prompt message.

PrintReceipt() //abstract class grouping subclasses and providing access.

- ➔ PrintReceipt1()
  // GasPump1 method use for printing a receipt.

  Void printreceipt()
  // print receipt by reading appropriate values.

- ➔ PrintReceipt2()
  // GasPump2 method use for printing a receipt.

  Void printreceipt()
  // print receipt by reading appropriate values.

PumpGasUnit() //abstract class grouping subclasses and providing access.

➔ PumpGasUnit1()

//method responsible for pumping a gallon of gas in gaspump1.

Void pumpgasunit()
// pumping 1 gallon gas and updating values.

➔ PumpGasUnit2()
//method responsible for pumping a gallon of gas in gaspump1.

Void pumpgasunit()
// pumping 1 gallon gas and updating values.

ReadyMsg() //abstract class grouping subclasses and providing access.

➔ ReadyMsg1()

// this method is use to print ready message for gaspump1.

Void readymsg()
//print a message that gaspump1 is ready to dispense 1 gallon of gas.

➔ ReadyMsg2()
// this method is use to print ready message for gaspump1.

Void readymsg()
//print a message that gaspump1 is ready to dispense 1 gallon of gas.

RejectMsg() //abstract class grouping subclasses and providing access.

➔ RejectMsg1()
// this class is use to print credit card rejection message for gaspump1.

Void rejectmsg()
//printing credit card declined message.

➔ RejectMsg2()
// gaspump2 doesnot support any credit card payment so no error message.

ReturnCash() //abstract class grouping subclasses and providing access.

➔ ReturnCash1()
// this method does nothing under current design.

➔ ReturnCash2()
// GasPump2 returncash is responsible for retruning the remaining amount of cash.

Void returncash()
// this method will first calculate the total bill amount generated and then it will calculate the change necessary. if there is any change left then it will return back.

SetInitialPrices() //abstract class grouping subclasses and providing access.

➔ SetInitialPrice1()
//initializing the necessary attributes to begin a transaction calculation for GasPump1.

Void Setinitialprice()
//Set the number of gallons pumped and payment balance initially to zero for this transaction.

➔ SetInitialPrice2()
//initializing the necessary attributes to begin a transaction calculation for GasPump1.

Void Setinitialprice()
//Set the number of gallons pumped and payment balance initially to zero for this transaction.

SetPrice() //abstract class grouping subclasses and providing access.

➔ SetPrice1()
// SetPrice is use to update the price based on selected Gas type.

Void setprice(int g)

// set the price per gallon of whichever gas is selected.
// g = 1 i.e Regular Gas.
// g = 2 i.e Super Gas.

- ➔ SetPrice2()

  // SetPrice is use to update the price based on selected Gas type.

  Void setprice(int g)

  // set the price per gallon of whichever gas is selected.
                        // g = 1 i.e Regular Gas.
                        // g = 2 i.e Super Gas.
                        // g=3 i.e premium gas.

StopMsg() //abstract class grouping subclasses and providing access.

- ➔ StopMsg1()

  // this method is use to inidcate that pumping is stopped for GasPump1.
  Void stopmsg()

- ➔ StopMsg2()

  // this method is use to inidcate that pumping is stopped for GasPump2.
  Void stopmsg()

StoreCash() //abstract class grouping subclasses and providing access.

- ➔ StoreCash1()

  // This method is for GasPump 1 however this method will never get called as there is no PayCash method in GasPump 1.

  Void storecash()

- ➔ StoreCash2()
  // This method is for GasPump 2 method in GasPump .

  Void storecash()

StoreData() //abstract class grouping subclasses and providing access.

- ➔ StoreData1()

```
/*
 * GasPump1 StoreData action responsible for storing the "a" and "b" price parameters specified by
method "Activate" of the InputProcessor for GasPump1
*/
d.R_price = d.a;
                  d.S_price = d.b;
```

- ➔ StoreData2()

```
/*
 * GasPump2 StoreData action responsible for storing the "a" "b" and "c" price parameters specified by
method "Activate" of the InputProcessor for GasPump1
*/


                  d.R_price = d.a;
                  d.S_price = d.b;
                  d.P_price = d.c;
```

<u>OutputProcessor :-</u>

//This class is the general output processor for the gas pump system.

// Each meta-action in this class calls the platform specific implementation of the action.

//This class acts as the "Client" class in the strategy design pattern.

void   cancelMsg()

// call according actions in abstract factory.

void   displayMenu()

// call according actions in abstract factory.

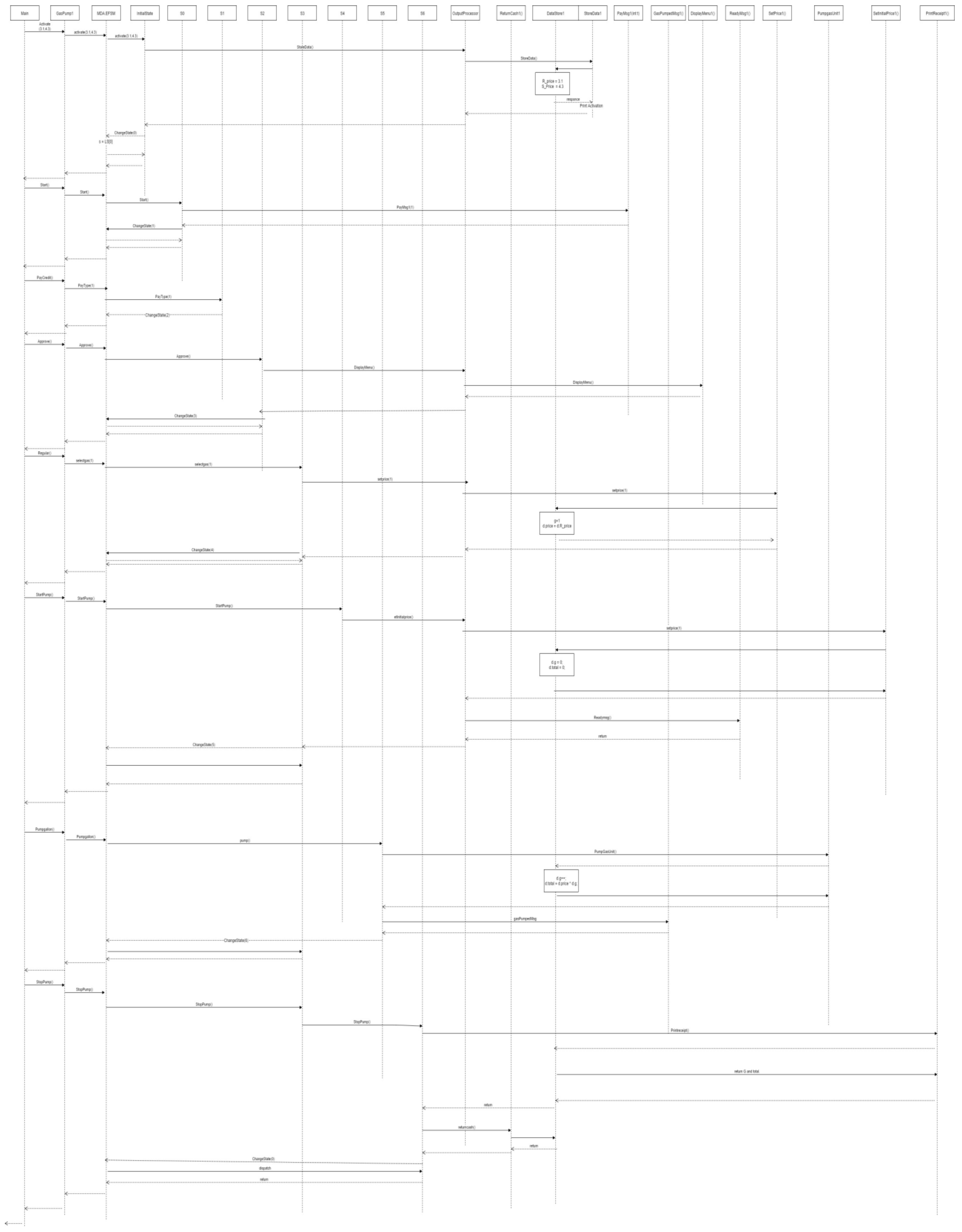
void   gasPumpedMsg()

// call according actions in abstract factory.

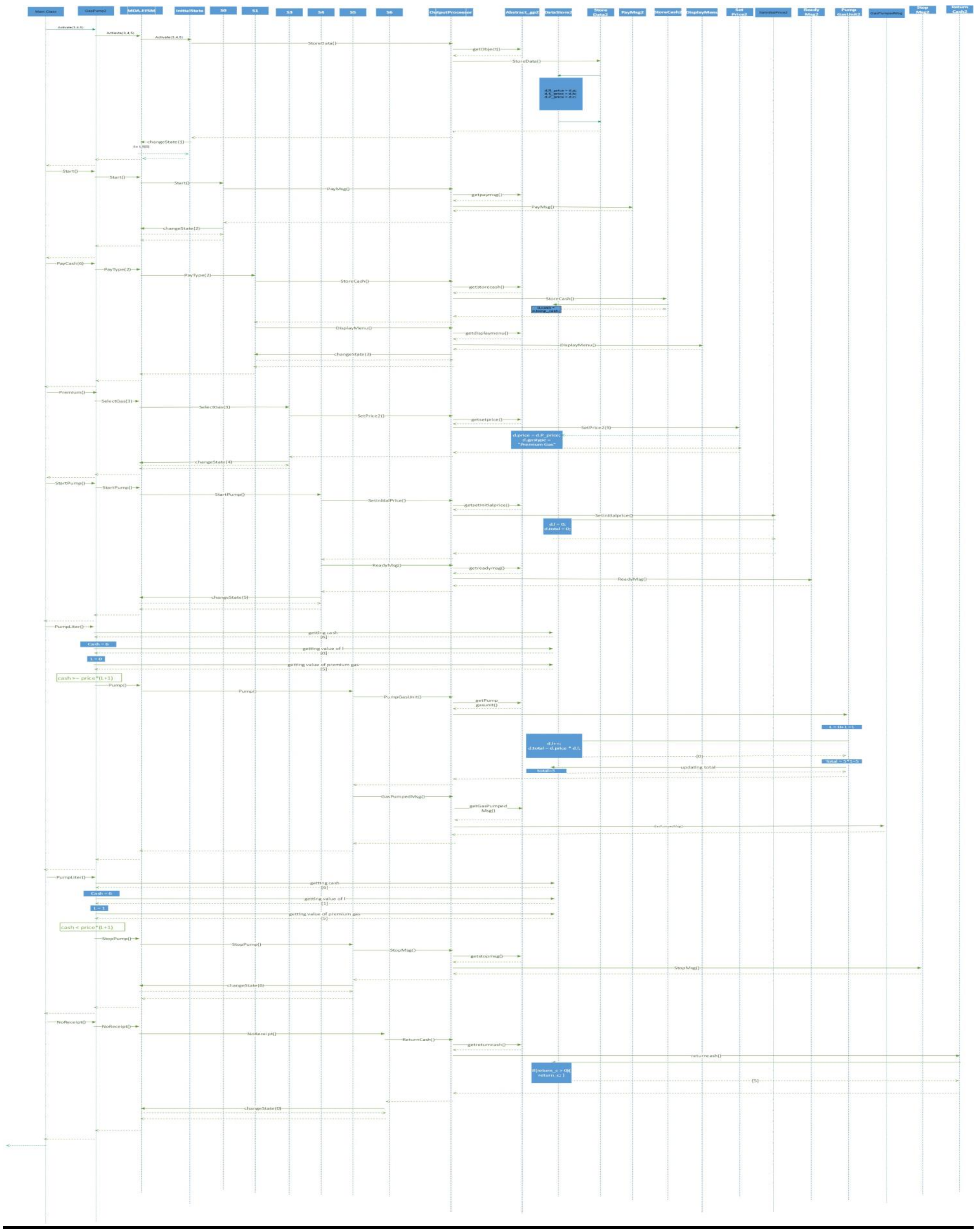
void   payMsg()

```
// call according actions in abstract factory.


void    printReceipt()
// call according actions in abstract factory.


void    pumpGasUnit()
// call according actions in abstract factory.


void    readyMsg()
// call according actions in abstract factory.


void    rejectMsg()
// call according actions in abstract factory.


void    setInitialValues()
// call according actions in abstract factory.


void    setPrice(int g)


// call according actions in abstract factory.


void    stopMsg()
// call according actions in abstract factory.


void    storeCash()
// call according actions in abstract factory.


void    storeData()
//      call according actions in abstract factory.
```

Sequence Diagram 1:-


Main
GasPump1
MDA.EFSM
InitialState
S0
S1
S2
S3
S4
S5
S6
OutputProcessor
ReturnCash1()
DataStore1
StoreData1
PayMsg1(int t)
GasPumpedMsg1()
DisplayMenu1()
ReadyMsg1()
SetPrice1()
PumpgasUnit1
SetinitialPrice1()
PrintReceipt1()
Activate (3.1,4.3)
activate(3.1,4.3)
activate(3.1,4.3)
StaleData()
StoreData()
R_price = 3.1
S_Price = 4.3
responce
Print Activation
ChangeState(0)
s = LS[0]
Start()
Start()
Start()
PayMsg1(1)
ChangeState(1)
PayCredit()
PayType(1)
PayType(1)
ChangeState(2)
Approve()
Approve()
Approve()
DisplayMenu()
DisplayMenu()
ChangeState(3)
Regular()
selectgas(1)
selectgas(1)
setprice(1)
setprice(1)
g=1
d.price = d.R_price
ChangeState(4)
StartPump()
StartPump()
StartPump()
setinitialprice()
setprice(1)
d.g = 0;
d.total = 0;
Readymsg()
return
ChangeState(5)
Pumpgallon()
Pumpgallon()
pump()
PumpGasUnit()
d.g++;
d.total = d.price * d.g;
gasPumpedMsg
ChangeState(6)
StopPump()
StopPump()
StopPump()
StopPump()
Printreceipt()
return G and total
return
returncash()
return
ChangeState(0)
dispatch
return

Sequence Diagram 2:-


Main Class
GasPump2
MDA-EFSM
InitialState
S0
S1
S3
S4
S5
S6
OutputProcessor
Abstract_gp2
DataStore2
Store Data2
PayMsg2
StoreCash2
DisplayMenu
Set Price2
SetInitialPrice2
Ready Msg2
Pump GasUnit2
GasPumpedMsg
Stop Msg2
Return Cash2
Activate(3,4,5)
StoreData()
getObject()
d.R_price = d.a; d.S_price = d.b; d.P_price = d.c;
changeState(1)
Start()
PayMsg()
getpaymsg()
changeState(2)
PayCash(6)
PayType(2)
StoreCash()
getstorecash()
d.cash = d.temp_cash;
DisplayMenu()
getdisplaymenu()
changeState(3)
Premium()
SelectGas(3)
SetPrice2()
getsetprice()
SetPrice2(5)
d.price = d.P_price; d.gastype = "Premium Gas"
changeState(4)
StartPump()
SetInitialPrice()
getsetinitialprice()
SetInitialprice()
d.l = 0; d.total = 0;
ReadyMsg()
getreadymsg()
changeState(5)
PumpLiter()
getting cash
[6]
Cash = 6
getting value of l
[0]
L = 0
getting value of premium gas
[5]
cash >= price*(L+1)
Pump()
PumpGasUnit()
getPump gasunit()
L = 0+1 =1
d.l++; d.total = d.price * d.l;
[0]
Total = 5*1=5
updating total
total=5
GasPumpedMsg()
getGasPumped Msg()
GasPumpedMsg()
[1]
L = 1
cash < price*(L+1)
StopPump()
StopMsg()
getstopmsg()
changeState(6)
NoReceipt()
ReturnCash()
getreturncash()
returncash()
if(return_c > 0){ return_c; }
changeState(0)