CS 586 Project

Submitted By:
Rumin Shah
A20369998

1. MDA-EFSM model for GasPump components

1.1 MDA-EFSM Events:

Activate()
Start()
PayCredit()
PayCash()
Reject()
Cancel()
Approved()
StartPump()
Pump()
SelectGas(int g)
StopGasPump()
Receipt()
NoReceipt()

1.2 MDA-EFSM Actions:

StoreData	// stores price(s) for the gas from the temporary data store
PayMsg	// displays a type of payment method
StoreCash	// stores cash from the temporary data store
DisplayMenu	// display a menu with a list of selections
RejectMsg	// displays credit card not approved message
SetPrice(int g)	// set the price for the gas identified by g identifier
ReadyMsg	// displays the ready for pumping message
InitializeValues	// 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
ReturnCash	// returns deposited cash which remains after deducted from the total price of the gas pumped
CancelMsg	// displays a cancellation message

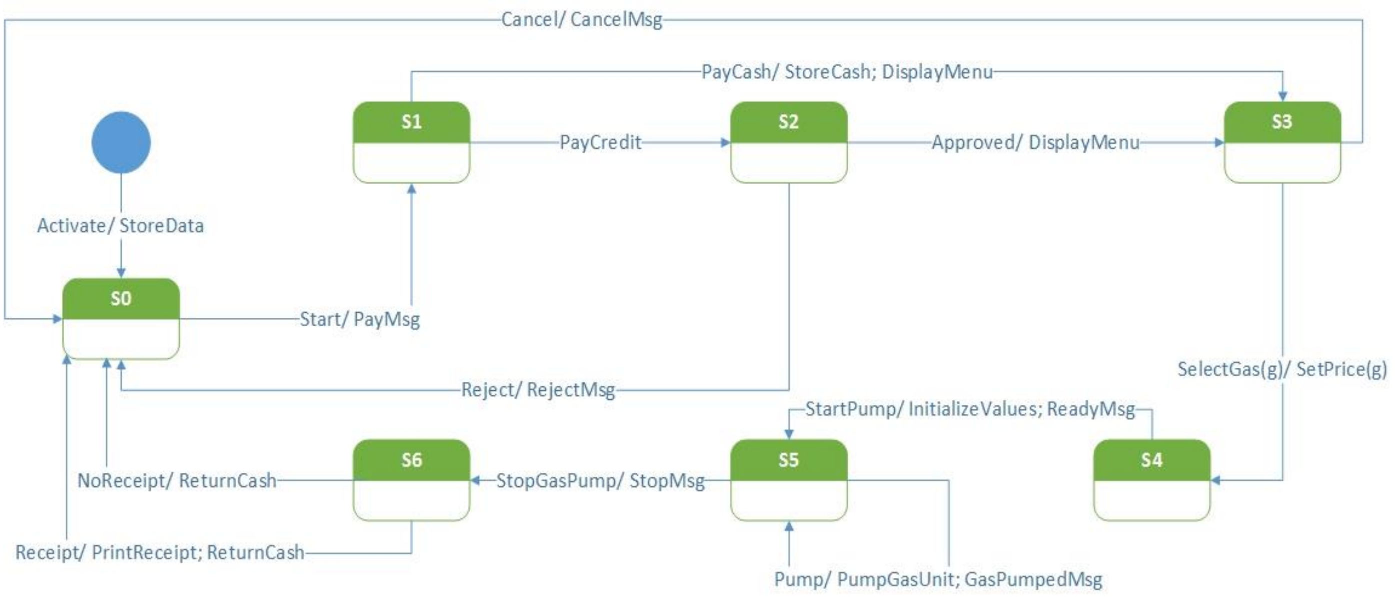
1.3.1 Operations of Input Processor (GasPump1)

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

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

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

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

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

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

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

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

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

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


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

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

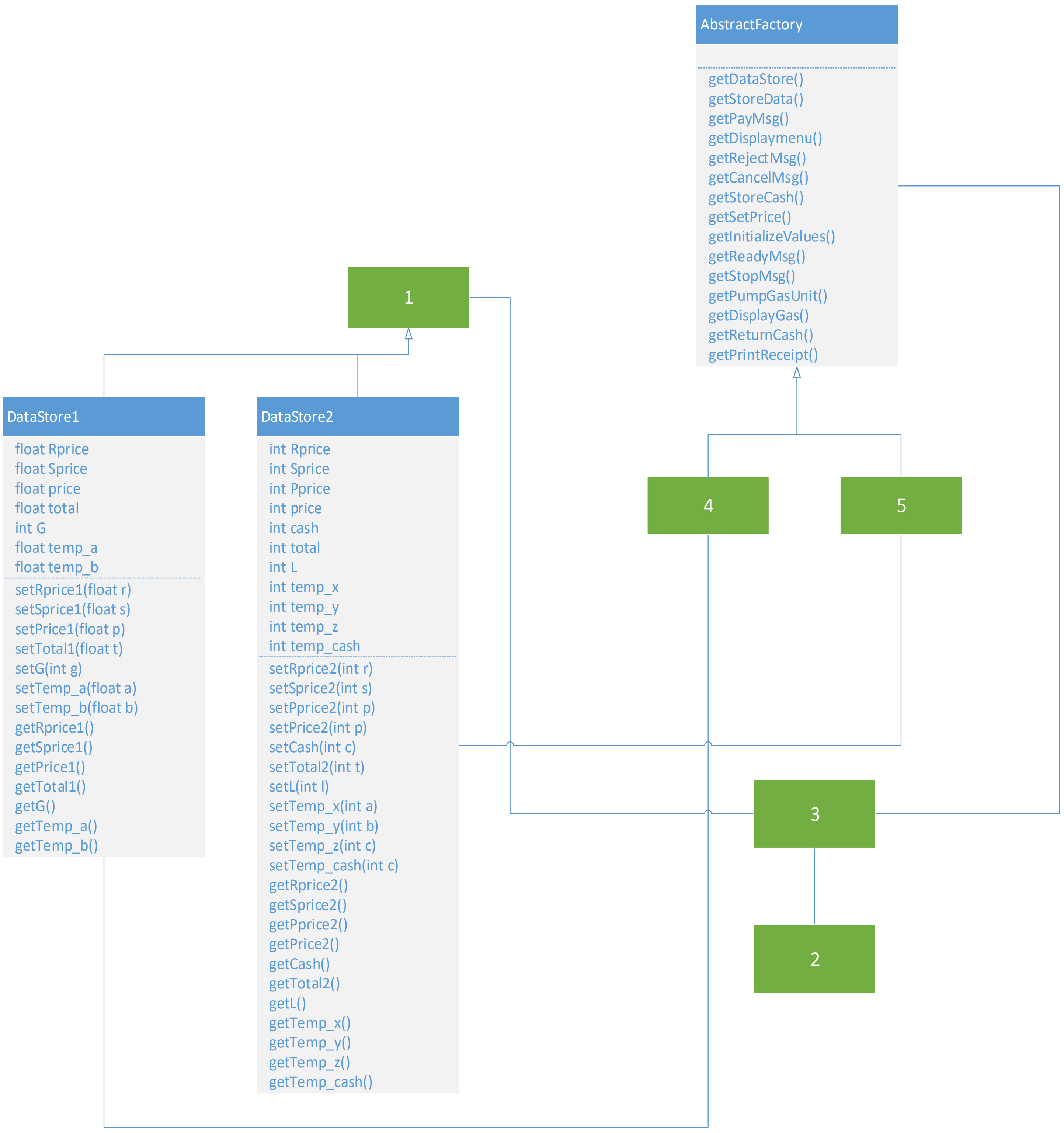
1.3.2 Operations of Input Processor (GasPump2)

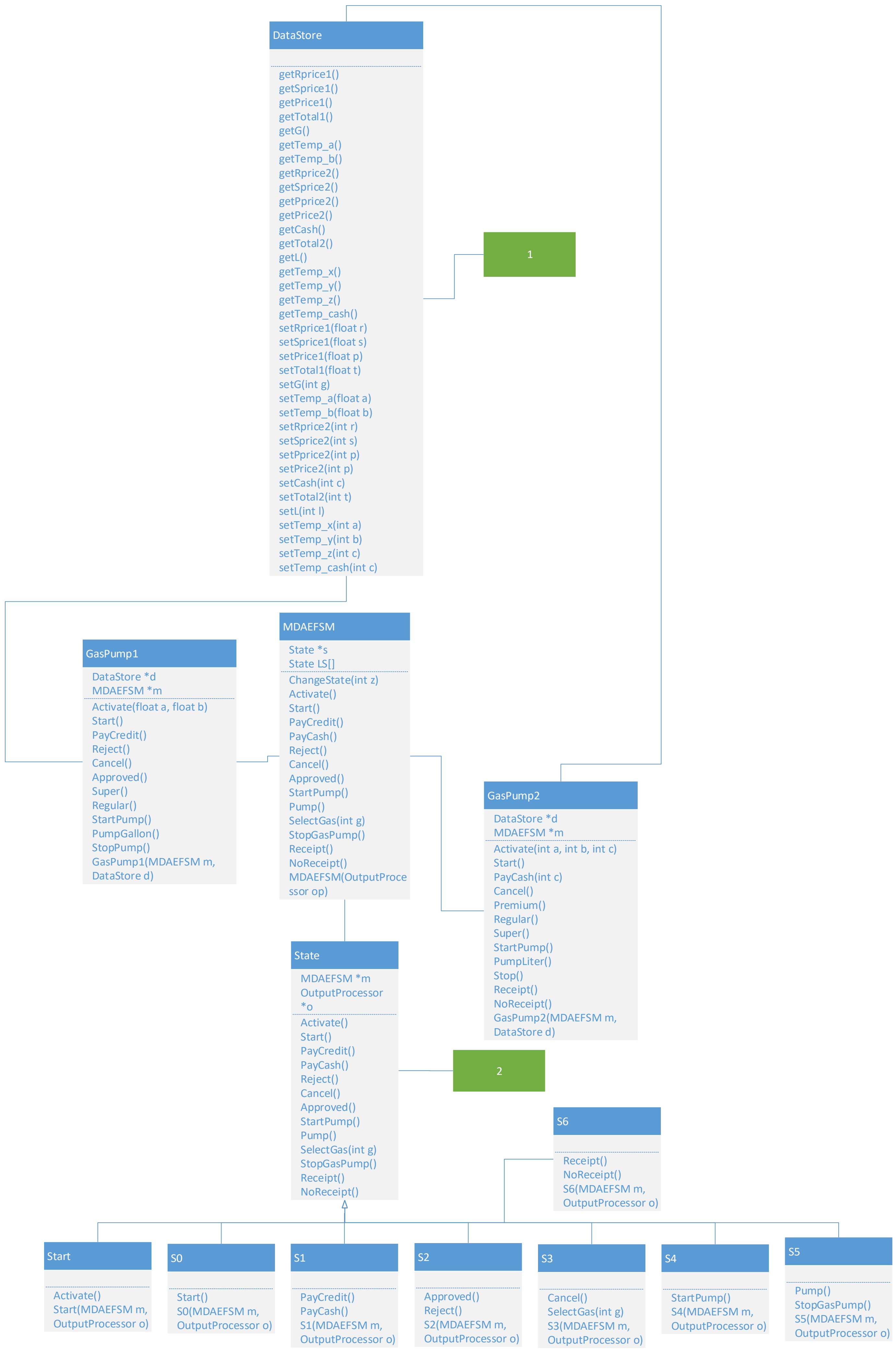
```
Activate(int a, int b, int c) {
        if ((a>0)&&(b>0)&&(c>0) {
                d->temp_x=a;
                d->temp_y=b;
                d->temp_z=c;
                m->Activate()
        }
}

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

PayCash(int c) {
        if (c>0) {
                d->temp_cash=c;
                m->PayCash()
        }
}

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

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

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

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

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

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

Stop() {
        m->StopGasPump();
}

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

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

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

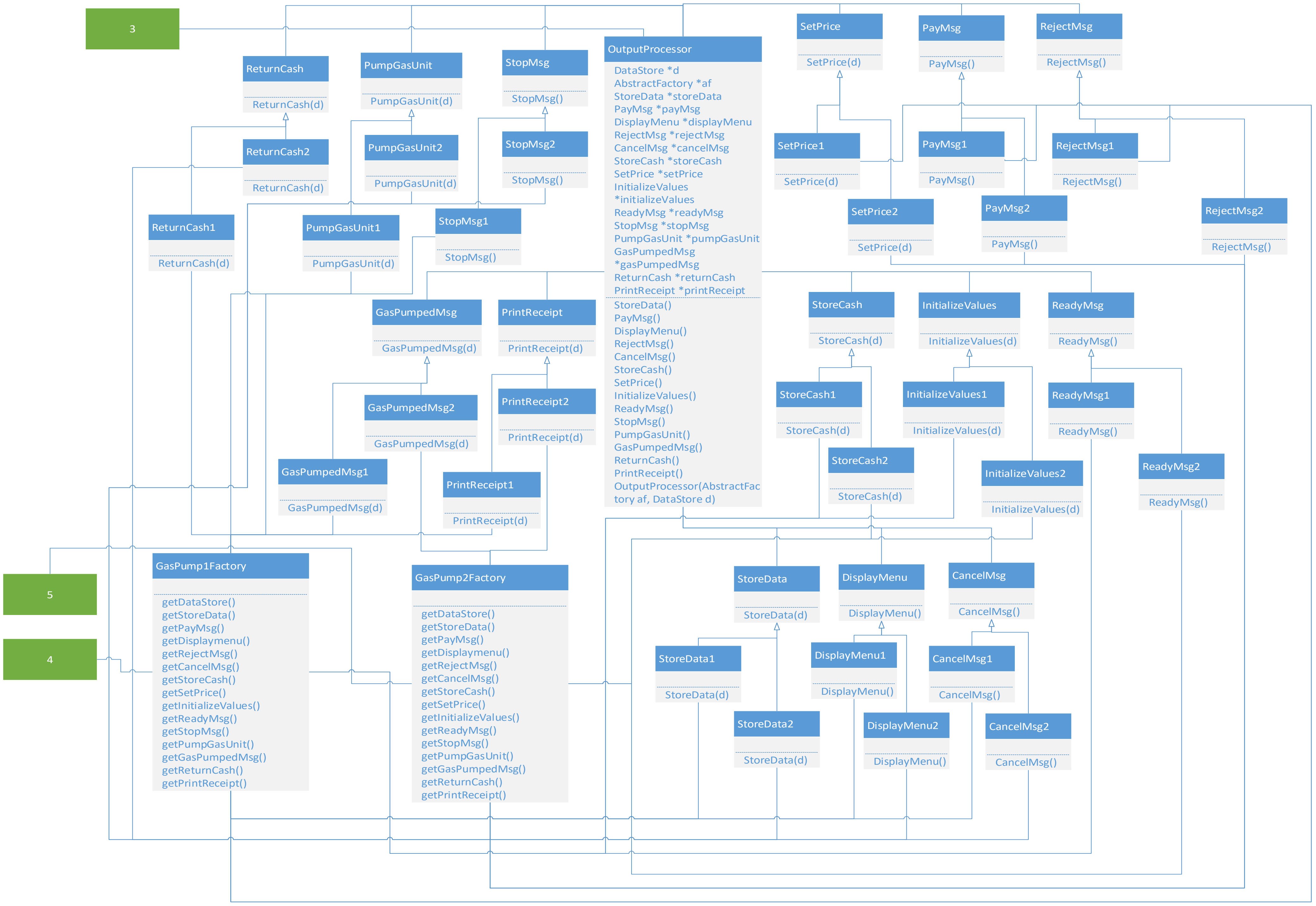
1.4 State Diagram of the MDA-EFSM


Cancel/ CancelMsg
PayCash/ StoreCash; DisplayMenu
S1
PayCredit
S2
Approved/ DisplayMenu
S3
Activate/ StoreData
S0
Start/ PayMsg
SelectGas(g)/ SetPrice(g)
Reject/ RejectMsg
StartPump/ InitializeValues; ReadyMsg
S6
StopGasPump/ StopMsg
S5
S4
NoReceipt/ ReturnCash
Receipt/ PrintReceipt; ReturnCash
Pump/ PumpGasUnit; GasPumpedMsg

2. Class Diagram




DataStore
getRprice1()
getSprice1()
getPrice1()
getTotal1()
getG()
getTemp_a()
getTemp_b()
getRprice2()
getSprice2()
getPprice2()
getPrice2()
getCash()
getTotal2()
getL()
getTemp_x()
getTemp_y()
getTemp_z()
getTemp_cash()
setRprice1(float r)
setSprice1(float s)
setPrice1(float p)
setTotal1(float t)
setG(int g)
setTemp_a(float a)
setTemp_b(float b)
setRprice2(int r)
setSprice2(int s)
setPprice2(int p)
setPrice2(int p)
setCash(int c)
setTotal2(int t)
setL(int l)
setTemp_x(int a)
setTemp_y(int b)
setTemp_z(int c)
setTemp_cash(int c)
1
MDAEFSM
State *s
State LS[]
ChangeState(int z)
Activate()
Start()
PayCredit()
PayCash()
Reject()
Cancel()
Approved()
StartPump()
Pump()
SelectGas(int g)
StopGasPump()
Receipt()
NoReceipt()
MDAEFSM(OutputProcessor op)
GasPump1
DataStore *d
MDAEFSM *m
Activate(float a, float b)
Start()
PayCredit()
Reject()
Cancel()
Approved()
Super()
Regular()
StartPump()
PumpGallon()
StopPump()
GasPump1(MDAEFSM m, DataStore d)
GasPump2
DataStore *d
MDAEFSM *m
Activate(int a, int b, int c)
Start()
PayCash(int c)
Cancel()
Premium()
Regular()
Super()
StartPump()
PumpLiter()
Stop()
Receipt()
NoReceipt()
GasPump2(MDAEFSM m, DataStore d)
State
MDAEFSM *m
OutputProcessor *o
Activate()
Start()
PayCredit()
PayCash()
Reject()
Cancel()
Approved()
StartPump()
Pump()
SelectGas(int g)
StopGasPump()
Receipt()
NoReceipt()
2
S6
Receipt()
NoReceipt()
S6(MDAEFSM m, OutputProcessor o)
Start
Activate()
Start(MDAEFSM m, OutputProcessor o)
S0
Start()
S0(MDAEFSM m, OutputProcessor o)
S1
PayCredit()
PayCash()
S1(MDAEFSM m, OutputProcessor o)
S2
Approved()
Reject()
S2(MDAEFSM m, OutputProcessor o)
S3
Cancel()
SelectGas(int g)
S3(MDAEFSM m, OutputProcessor o)
S4
StartPump()
S4(MDAEFSM m, OutputProcessor o)
S5
Pump()
StopGasPump()
S5(MDAEFSM m, OutputProcessor o)


3
ReturnCash
ReturnCash(d)
PumpGasUnit
PumpGasUnit(d)
StopMsg
StopMsg()
OutputProcessor
DataStore *d
AbstractFactory *af
StoreData *storeData
PayMsg *payMsg
DisplayMenu *displayMenu
RejectMsg *rejectMsg
CancelMsg *cancelMsg
StoreCash *storeCash
SetPrice *setPrice
InitializeValues
*initializeValues
ReadyMsg *readyMsg
StopMsg *stopMsg
PumpGasUnit *pumpGasUnit
GasPumpedMsg
*gasPumpedMsg
ReturnCash *returnCash
PrintReceipt *printReceipt
StoreData()
PayMsg()
DisplayMenu()
RejectMsg()
CancelMsg()
StoreCash()
SetPrice()
InitializeValues()
ReadyMsg()
StopMsg()
PumpGasUnit()
GasPumpedMsg()
ReturnCash()
PrintReceipt()
OutputProcessor(AbstractFactory af, DataStore d)
SetPrice
SetPrice(d)
PayMsg
PayMsg()
RejectMsg
RejectMsg()
ReturnCash2
ReturnCash(d)
PumpGasUnit2
PumpGasUnit(d)
StopMsg2
StopMsg()
SetPrice1
SetPrice(d)
PayMsg1
PayMsg()
RejectMsg1
RejectMsg()
SetPrice2
SetPrice(d)
PayMsg2
PayMsg()
RejectMsg2
RejectMsg()
ReturnCash1
ReturnCash(d)
PumpGasUnit1
PumpGasUnit(d)
StopMsg1
StopMsg()
GasPumpedMsg
GasPumpedMsg(d)
PrintReceipt
PrintReceipt(d)
StoreCash
StoreCash(d)
InitializeValues
InitializeValues(d)
ReadyMsg
ReadyMsg()
GasPumpedMsg2
GasPumpedMsg(d)
PrintReceipt2
PrintReceipt(d)
StoreCash1
StoreCash(d)
InitializeValues1
InitializeValues(d)
ReadyMsg1
ReadyMsg()
StoreCash2
StoreCash(d)
InitializeValues2
InitializeValues(d)
ReadyMsg2
ReadyMsg()
GasPumpedMsg1
GasPumpedMsg(d)
PrintReceipt1
PrintReceipt(d)
5
GasPump1Factory
getDataStore()
getStoreData()
getPayMsg()
getDisplaymenu()
getRejectMsg()
getCancelMsg()
getStoreCash()
getSetPrice()
getInitializeValues()
getReadyMsg()
getStopMsg()
getPumpGasUnit()
getGasPumpedMsg()
getReturnCash()
getPrintReceipt()
4
GasPump2Factory
getDataStore()
getStoreData()
getPayMsg()
getDisplaymenu()
getRejectMsg()
getCancelMsg()
getStoreCash()
getSetPrice()
getInitializeValues()
getReadyMsg()
getStopMsg()
getPumpGasUnit()
getGasPumpedMsg()
getReturnCash()
getPrintReceipt()
StoreData
StoreData(d)
DisplayMenu
DisplayMenu()
CancelMsg
CancelMsg()
StoreData1
StoreData(d)
DisplayMenu1
DisplayMenu()
CancelMsg1
CancelMsg()
StoreData2
StoreData(d)
DisplayMenu2
DisplayMenu()
CancelMsg2
CancelMsg()

3. Classes and its Operations

GasPump1

This class acts as an Input Processor for GasPump-1

Operations:

```
-- public void Activate(float a, float b)

Activate Gas Pump-1 and takes price of regular gas and super gas from the user

Set price of regular gas to temporary variable

Set price of super gas to temporary variable

Calls Activate() in MDAEFSM class

-- public void Start()

Operates when user selects Start

Calls Start() in MDAEFSM class

-- public void PayCredit()

Operates when user selects PayCredit

Calls PayCredit() in MDAEFSM class

-- public void Reject()

Operates when user selects Reject

Calls Reject() in MDAEFSM class

-- public void Cancel()

Operates when user selects Cancel

Calls Cancel() in MDAEFSM class

-- public void Approved()

User selects Approved

Calls Approved() in MDAEFSM class

-- public void Super()

Operates when user selects Super gas

Calls SelectGas() in MDAEFSM class and tell it that the user has selected Super gas

-- public void Regular()

Operates when user selects Regular gas

Calls SelectGas() in MDAEFSM class and tell it that the user has selected Regular gas

-- public void StartPump()

Operates when user selects Start Pump

Calls StartPump() in MDAEFSM class
```

-- **public void** PumpGallon()

Operates when user selects Pump Gallon

Calls Pump() in MDAEFSM class

-- **public void** StopPump()

Operates when user selects Stop Pump

Calls StopGasPump() in MDAEFSM class

Calls Receipt() in MDAEFSM class

GasPump2

This class acts as an Input Processor for GasPump-2

Operations:

-- **public void** Activate(**float** a, **float** b)

Activate Gas Pump-1 and takes price of regular gas and super gas from the user

Set price of regular gas to temporary variable

Set price of super gas to temporary variable

Set price of premium gas to temporary variable

Calls Activate() in MDAEFSM class

-- **public void** Start()

Operates when user selects Start

Calls Start() in MDAEFSM class

-- **public void** PayCash()

Operates when user selects PayCash

Check if user enters value of cash as positive integer

Set cash deposited to temporary variable in Data Store 2 class

Calls PayCash() in MDAEFSM class

-- **public void** Cancel()

Operates when user selects Cancel

Calls Cancel() in MDAEFSM class

-- **public void** Premium()

Operates when user selects Premium gas

Calls SelectGas() in MDAEFSM class and tell it that the user has selected Premium gas

```
-- public void Super()

Operates when user selects Super gas

Calls SelectGas() in MDAEFSM class and tell it that the user has selected Super gas

-- public void Regular()

Operates when user selects Regular gas

Calls SelectGas() in MDAEFSM class and tell it that the user has selected Regular gas

-- public void StartPump()

Operates when user selects Start Pump

Calls StartPump() in MDAEFSM class

-- public void PumpLiter()

Operates when user selects Pump Liter

Fetches cash stored from Data Store 2 class

Fetches Liters from Data Store 2 class

Fetches price of the selected gas from Data Store class

Checks if user has enough cash to pump liters and then calls Pump() and StopGasPump() in
MDAEFSM class accordingly

-- public void Stop()

Operates when user selects Stop Pump

Calls StopGasPump() in MDAEFSM class

-- public void Receipt()

Operates when user selects Receipt

Calls Receipt() in MDAEFSM class

-- public void NoReceipt()

Operates when user selects NoReceipt

Calls NoReceipt() in MDAEFSM class
```

DataStore

This class acts as an abstract class containing abstract methods and is responsible for storing data of both the Gas Pumps

Operations:

```
All the operations in this class are abstract methods which has implementations in DataStore1
and DataStore2

public abstract float getRprice1();

public abstract void setRprice1(float Rprice1);

public abstract float getSprice1();

public abstract void setSprice1(float Sprice1);

public abstract float getPrice1();

public abstract void setPrice1(float price1);

public abstract float getTotal1();

public abstract void setTotal1(float total1);

public abstract int getG();

public abstract void setG(int G);

public abstract float getTemp_a();

public abstract void setTemp_a(float a);

public abstract float getTemp_b();

public abstract void setTemp_b(float b);

public abstract int getRprice2();

public abstract void setRprice2(int Rprice2);

public abstract int getSprice2();

public abstract void setSprice2(int Sprice2);

public abstract int getPprice2();

public abstract void setPprice2(int Pprice2);

public abstract int getPrice2();

public abstract void setPrice2(int price2);

public abstract int getCash();

public abstract void setCash(int c);

public abstract int getTotal2();

public abstract void setTotal2(int total2);

public abstract int getL();

public abstract void setL(int L);
public abstract int getTemp_x();

public abstract void setTemp_x(int a);

public abstract int getTemp_y();

public abstract void setTemp_y(int b);
public abstract int getTemp_z();

public abstract void setTemp_z(int c);

public abstract int getTemp_cash();

public abstract void setTemp_cash(int c);
```

DataStore1

This class acts as a concrete class for storing data of GasPump-1

Operations:

```
-- public float getRprice1()

Fetches Regular gas price

-- public void setRprice1(float Rprice1)

Sets Regular gas price

-- public float getSprice1()

Fetches Super gas price

-- public void setSprice1(float Sprice1)

Sets Super gas price

-- public float getPrice1()

Fetches price of the selected gas by the user

-- public void setPrice1(float price1)

Sets price of the selected gas by the user

-- public int getG()

Fetch gallons pumped by the user

-- public void setG(int G)

Sets gallons pumped by the user

-- public float getTemp_a()

Fetches price of regular gas stored in temporary variable

-- public void setTemp_a(float a)

Sets price of regular gas in temporary variable
```

-- **public float** getTemp_b()

Fetches price of super gas stored in temporary variable

-- **public void** setTemp_b(**float** b)

Sets price of super gas in temporary variable

-- **public float** getTotal1()

Fetches total price of the pumped gas

-- **public void** setTotal1(**float** total1)

Sets total price of the pumped gas

DataStore2

This class acts as a concrete class for storing data of GasPump-2

Operations:

--**public int** getRprice2()

Fetches Regular gas price

-- **public void** setRprice2(**int** Rprice2)

Sets Regular gas price

-- **public int** getSprice2()

Fetches Super gas price

-- **public void** setSprice2(**int** Sprice2)

Sets Super gas price

-- **public int** getPprice2()

Fetches Premium gas price

-- **public void** setPprice2(**int** Pprice2)

Sets Premium gas price

-- **public int** getPrice2()

Fetches price of the selected gas by the user

-- **public void** setPrice2(**int** price2)

Sets price of the selected gas by the user

-- **public int** getCash()

Fetches cash deposited by the user

-- **public void** setCash(**int** c)

Sets cash deposited by the user

```
-- public int getTotal2()

Fetches total price of the pumped gas

-- public void setTotal2(int total2)

Sets total price of the pumped gas

-- public int getL()

Fetches liters pumped by the user

-- public void setL(int L)

Sets liters pumped by the user

-- public int getTemp_x()

Fetches price of regular gas stored in temporary variable

-- public void setTemp_x(int a)

Sets price of regular gas in temporary variable

-- public int getTemp_y()

Fetches price of super gas stored in temporary variable

-- public void setTemp_y(int b)

Sets price of super gas in temporary variable

-- public int getTemp_z()

Fetches price of premium gas stored in temporary variable

-- public void setTemp_z(int c)

Sets price of premium gas in temporary variable

-- public int getTemp_cash()

Fetches cash deposited by the user stored in temporary variable

-- public void setTemp_cash(int c)

Sets cash deposited by the user in temporary variable
```

MDAEFSM

This class acts as MDA-EFSM for both Gas Pump components GasPump-1 and GasPump-2 and is a part of State Pattern

Operations:

```
-- public void ChangeState(int z)

Changes state from several state classes

/* ************Methods responsible for State Design Pattern************ */

-- public void Activate()

Calls Activate() of appropriate State class

-- public void Start()

Calls Start() of appropriate State class

-- public void PayCredit()

Calls PayCredit() of appropriate State class

-- public void PayCash()

Calls PayCash() of appropriate State class

-- public void Reject()

Calls Reject() of appropriate State class

-- public void Approved()

Calls Approved() of appropriate of State class

-- public void Cancel()

Calls Cancel() of appropriate State class

-- public void SelectGas(int g)

Calls SelectGas() of appropriate State class

-- public void StartPump()

Calls StartPump() of appropriate State class

--public void Pump()

Calls Pump() of appropriate State class

-- public void StopGasPump()

Calls StopGasPump() of appropriate State class

-- public void Receipt()

Calls Receipt() of appropriate State class

--public void NoReceipt()

Calls NoReceipt() of appropriate State class
```

State

This class acts as an abstract class for several other state classes which extends this class and it a part of the State Pattern

Operations:

Methods of this class have their implementation in its concrete child classes

```
-- public void Activate()

-- public void Start()

-- public void PayCredit()

-- public void PayCash()

-- public void Reject()

-- public void Approved()

-- public void Cancel()

-- public void SelectGas(int g)

-- public void StartPump()

-- public void Pump()

-- public void StopGasPump()

-- public void Receipt()

-- public void NoReceipt()
```

Start

This class extends State Class and acts as its concrete child and it is a part of the State Pattern

Operations:

```
-- public void Activate()

Stores entered price of the different types of gas based on the selected gas pumps from Output
Processor.

Changes state to S0
```

S0

This class extends State Class and acts as its concrete child and it is a part of the State Pattern

Operations:

```
-- public void Start()

Displays PayMsg from Output Processor

Changes state to S1
```

S1

This class extends State Class and acts as its concrete child and it is a part of the State Pattern

Operations:

```
-- public void PayCredit()

Changes state to S2

-- public void PayCash()

Displays Menu from Output Processor

Stores deposited Cash from Output Processor

Changes state to S3
```

S2

This class extends State Class and acts as its concrete child and it is a part of the State Pattern

Operations:

```
-- public void Approved()

Displays Menu from Output Processor

Changes state to S3

-- public void Reject()

Display RejectMsg from Output Processor

Changes state to S0
```

S3

This class extends State Class and acts as its concrete child and it is a part of the State Pattern

Operations:

```
-- public void Cancel()

Displays CancelMsg from Output Processor

Changes state to S0

-- public void SelectGas(int g)

Sets price of the selected gas from Output Processor

Changes state to S4
```

S4

This class extends State Class and acts as its concrete child and it is a part of the State Pattern

Operations:

```
-- public void StartPump()

Initializes values of gas and total from Output Processor

Displays ReadyMsg from Output Processor

Changes state to S5
```

S5

This class extends State Class and acts as its concrete child and it is a part of the State Pattern

Operations:

```
-- public void Pump()

Pumps specific unit of gas based on the selected Gas Pump from Output Processor

Displays total units of gas pumped and total price from the Output Processor

-- public void StopGasPump()

Displays StopMsg from Output Processor

Changes state to S6
```

S6

This class extends State Class and acts as its concrete child and it is a part of the State Pattern

Operations:

```
public void Receipt()

Displays total gas pumped and total price of the gas pumped from Output Processor

Returns remaining cash from Output Processor

Changes state to S0

-- public void NoReceipt()

Returns remaining cash from Output Processor

Changes state to S0
```

OutputProcessor

This class acts as an Output Processor for both Gas Pump components GasPump-1 and GasPump-2 and also it acts as a context class for Strategy Pattern and is also a part of Abstract Factory Pattern

Operations:

`-- public void StoreData()`

Stores prices of different gases based on selected gas pump

`-- public void PayMsg()`

Displays PayMsg of selected gas pump

`-- public void DisplayMenu()`

Displays Menu of selected gas pump

`-- public void RejectMsg()`

Displays RejectMsg of selected gas pump

`-- public void CancelMsg()`

Displays CancelMsg of selected gas pump

`--public void StoreCash()`

Stores deposited cash of selected gas pump

`-- public void SetPrice(int g)`

Sets price of selected type of the gas of selected gas pump

`-- public void InitializeValues()`

Initializes values of gas and total of selected gas pump

`-- public void ReadyMsg()`

Displays ReadyMsg of selected gas pump

`-- public void StopMsg()`

Displays StopMsg of selected gas pump

`-- public void PumpGasUnit()`

Displays appropriate units of gas pumped selected gas pump

`-- public void GasPumpedMsg()`

Displays total units of gas pumped of selected gas pump

`-- public void ReturnCash()`

Returns remaining cash after using selected gas pump

`public void PrintReceipt()`

Prints Receipt after using selected gas pump

AbstractFactory

This is an interface for Abstract Factory Design Pattern

Operations:

Since it is an interface, there are only signatures of methods.

```
DataStore getDataStore();

StoreData getStoreData();

PayMsg getPayMsg();

DisplayMenu getDisplayMenu();

RejectMsg getRejectMsg();

CancelMsg getCancelMsg();

StoreCash getStoreCash();

SetPrice getSetPrice();

InitializeValues getInitializeValues();

ReadyMsg getReadyMsg();

StopMsg getStopMsg();

PumpGasUnit getPumpGasUnit();

GasPumpedMsg getGasPumpedMsg();

ReturnCash getReturnCash();

PrintReceipt getPrintReceipt();
```

GasPump1Factory

This class acts as a concrete Factory for GasPump1 in Abstract Factory Design Pattern

Operations:

```
-- public DataStore getDataStore()

Returns DataStore1 object

-- public StoreData getStoreData()

Returns StoreData1 object

-- public PayMsg getPayMsg()

Returns PayMsg1 object

-- public DisplayMenu getDisplayMenu()

Returns DisplayMenu1 object
```

-- **public** RejectMsg getRejectMsg()

Returns RejectMsg1 object

-- **public** CancelMsg getCancelMsg()

Returns CancelMsg1 object

-- **public** StoreCash getStoreCash()

Returns StoreCash1 object

-- **public** SetPrice getSetPrice()

Returns SetPrice1 object

-- **public** InitializeValues getInitializeValues()

Returns InitializeValues1 object

-- **public** ReadyMsg getReadyMsg()

Returns ReadyMsg1 object

-- **public** StopMsg getStopMsg()

Returns StopMsg1 object

-- **public** PumpGasUnit getPumpGasUnit()

Returns PumpGasUnit1 object

-- **public** GasPumpedMsg getGasPumpedMsg()

Returns GasPumpedMsg1 object

-- **public** ReturnCash getReturnCash()

Returns ReturnCash1 object

-- **public** PrintReceipt getPrintReceipt()

Returns PrintReceipt1 object

GasPump2Factory

This class acts as a concrete Factory for GasPump1 in Abstract Factory Design Pattern

Operations:

-- **public** DataStore getDataStore()

Returns DataStore2 object

-- **public** StoreData getStoreData()

Returns StoreData2 object

-- **public** PayMsg getPayMsg()

Returns PayMsg2 object

```
-- public DisplayMenu getDisplayMenu()

Returns DisplayMenu2 object

-- public RejectMsg getRejectMsg()

Returns RejectMsg2 object

-- public CancelMsg getCancelMsg()

Returns CancelMsg2 object

-- public StoreCash getStoreCash()

Returns StoreCash2 object

-- public SetPrice getSetPrice()

Returns SetPrice2 object

-- public InitializeValues getInitializeValues()

Returns InitializeValues2 object

-- public ReadyMsg getReadyMsg()

Returns ReadyMsg2 object

-- public StopMsg getStopMsg()

Returns StopMsg2 object

-- public PumpGasUnit getPumpGasUnit()

Returns PumpGasUnit2 object

-- public GasPumpedMsg getGasPumpedMsg()

Returns GasPumpedMsg2 object

-- public ReturnCash getReturnCash()

Returns ReturnCash2 object

-- public PrintReceipt getPrintReceipt()

Returns PrintReceipt2 object
```

CancelMsg

This is an interface for CancelMsg

Operations:

```
public interface CancelMsg {
	public void CancelMsg(); //CancelMsg
}
```

CancelMsg1

This class implements CancelMsg Interface and GasPump1Factory uses this class

Operations:

```
-- public void CancelMsg()

Displays CancelMsg for GasPump1
```

CancelMsg2

This class implements CancelMsg Interface and GasPump2Factory uses this class

Operations:

```
-- public void CancelMsg()

Displays CancelMsg for GasPump2
```

DisplayMenu

This is an interface for DisplayMenu

Operations:

```
public interface DisplayMenu {
      public void DisplayMenu(); //Display Menu
}
```

DisplayMenu1

This class implements DisplayMenu Interface and GasPump1Factory uses this class

Operations:

```
public void DisplayMenu()

Displays menu asking user to select type of gas for GasPump1
```

DisplayMenu2

This class implements DisplayMenu Interface and GasPump2Factory uses this class

Operations:

```
public void DisplayMenu()

Displays menu asking user to select type of gas for GasPump2
```

GasPumpedMsg

This is an interface for GasPumpedMsg

Operations:

```
public interface GasPumpedMsg {
	public void GasPumpedMsg(DataStore d); //GasPumpedMsg
}
```

GasPumpedMsg1

This class implements GasPumpedMsg Interface and GasPump1Factory uses this class

Operations:

```
public void GasPumpedMsg(DataStore d)

Displays total units of gas pumped by user for GasPump1
```

GasPumpedMsg2

This class implements GasPumped Interface and GasPump2Factory uses this class

Operations:

```
Displays total units of gas pumped by user for GasPump2
```

InitializeValues

This is an interface for InitializeValues

Operations:

```
public interface InitializeValues {
	public void InitializeValues(DataStore d);
}
```

InitializeValues1

This class implements InitializeValues Interface and GasPump1Factory uses this class

Operations:

```
public void InitializeValues(DataStore d)

Sets G=0
Sets total=0 for GasPump1
```

InitializeValues2

This class implements InitializeValues Interface and GasPump2Factory uses this class

Operations:

```
public void InitializeValues(DataStore d)

Sets L=0
Sets total=0 for GasPump2
```

PayMsg

This is an interface for PayMsg

Operations:

```
public interface PayMsg {
	public void PayMsg(); //PayMsg
}
```

PayMsg1

This class implements PayMsg Interface and GasPump1Factory uses this class

Operations:

```
public void PayMsg()

Displays payment method for the user for GasPump1
```

PayMsg2

This class implements PayMsg Interface and GasPump2Factory uses this class

Operations:

```
public void PayMsg()

Displays payment method for the user for GasPump2
```

PrintReceipt

This is an interface for PrintReceipt

Operations:

```
public interface PrintReceipt {
	public void PrintReceipt(DataStore d); //Print receipt

}
```

PrintReceipt1

This class implements PrintReceipt Interface and GasPump1Factory uses this class

Operations:

```
public void PrintReceipt(DataStore d)

Displays total Gas pumped

Displays total price of the pumped gas for GasPump1
```

PrintReceipt2

This class implements PrintReceipt Interface and GasPump2Factory uses this class

Operations:

```
public void PrintReceipt(DataStore d)

Displays total Gas pumped

Displays total price of the pumped gas for GasPump2
```

PumpGasUnit

This is an interface for PumpGasUnit

Operations:

```
public interface PumpGasUnit {
	public void PumpGasUnit(DataStore d); //Pump Gas Unit

}
```

PumpGasUnit1

This class implements PumpGasUnit Interface and GasPump1Factory uses this class

Operations:

```
public void PumpGasUnit(DataStore d)

Calculates total gas pumped

Calculates total price of the pumped gas for GasPump1
```

PumpGasUnit2

This class implements PumpGasUnit Interface and GasPump2Factory uses this class

Operations:

```
public void PumpGasUnit(DataStore d)

Calculates total gas pumped

Calculates total price of the pumped gas for GasPump2
```

ReadyMsg

This is an interface for ReadyMsg

Operations:

```
public interface ReadyMsg {
    public void ReadyMsg(); //ReadyMsg

}
```

ReadyMsg1

This class implements ReadyMsg Interface and GasPump1Factory uses this class

Operations:

```
public void ReadyMsg()
```

```
Displays Msg that the Gas pump is ready to pump for GasPump1
```

ReadyMsg2

This class implements ReadyMsg Interface and GasPump2Factory uses this class

Operations:

```
public void ReadyMsg()
```

```
Displays Msg that the Gas pump is ready to pump for GasPump2
```

RejectMsg

This is an interface for RejectMsg

Operations:

```
public interface RejectMsg {
    public void RejectMsg(); //RejectMsg

}
```

RejectMsg1

This class implements RejectMsg Interface and GasPump1actory uses this class

Operations:

```
public void RejectMsg()
```

```
Displays Msg that the credit card is rejected for GasPump1
```

RejectMsg2

This class implements RejectMsg Interface and GasPump2Factory uses this class

Operations:

```
public void RejectMsg()
```

```
Displays Msg that the credit card is rejected for GasPump2
```

ReturnCash

This is an interface for ReturnCash

Operations:

```
public interface ReturnCash {
    public void ReturnCash(DataStore d);//Return Cash

}
```

ReturnCash1

This class implements ReturnCash Interface and GasPump1Factory uses this class

Operations:

```
public void ReturnCash(DataStore d)
```

```
No implementation
```

ReturnCash2

This class implements ReturnCash Interface and GasPump2Factory uses this class

Operations:

```
public void ReturnCash(DataStore d)
```

```
Calculates and displays remaining cash for GasPump2
```

SetPrice

This is an interface for SetPrice

Operations:

```
public interface SetPrice {
      public void SetPrice(int g, DataStore d); //SetPrice
}
```

SetPrice1

This class implements SetPrice Interface and GasPump1Factory uses this class

Operations:

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

Checks if selected gas is regular or super

Fetches regular gas price or super gas price based on the gas user has selected for GasPump1
```

SetPrice2

This class implements SetPrice Interface and GasPump2Factory uses this class

Operations:

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

Checks if selected gas is regular or super

Fetches regular gas price or super gas price based on the gas user has selected for GasPump1
```

StopMsg

This is an interface for StopMsg

Operations:

```
public interface StopMsg {
    public void StopMsg(); //StopMsg

}
```

StopMsg1

This class implements StopMsg Interface and GasPump1Factory uses this class

Operations:

```
public void StopMsg()

Displays Msg that the gas pump has stopped for GasPump1
```

StopMsg2

This class implements StopMsg Interface and GasPump2Factory uses this class

Operations:

```
public void StopMsg()

Displays Msg that the gas pump has stopped and asks user for the receipt for GasPump2
```

StoreCash

This is an interface for StoreCash

Operations:

```
public interface StoreCash {
    public void StoreCash(DataStore d);//Store Cash

}
```

StoreCash1

This class implements StoreCash Interface and GasPump1Factory uses this class

Operations:

```
public void StoreCash(DataStore d)

No implementation
```

StoreCash2

This class implements StoreCash Interface and GasPump2Factory uses this class

Operations:

```
public void StoreCash(DataStore d)

Store value of deposited cash in DataStore1 for GasPump2
```

StoreData

This is an interface for StoreData

Operations:

```
public interface StoreData {
    public void StoreData(DataStore d);//Store Data

}
```

StoreData1

This class implements StoreData Interface and GasPump1Factory uses this class

Operations:

```
public void StoreData(DataStore d) {

Stores the prices of regular gas and super gas in DataStore1 for GasPump1
```

StoreData2

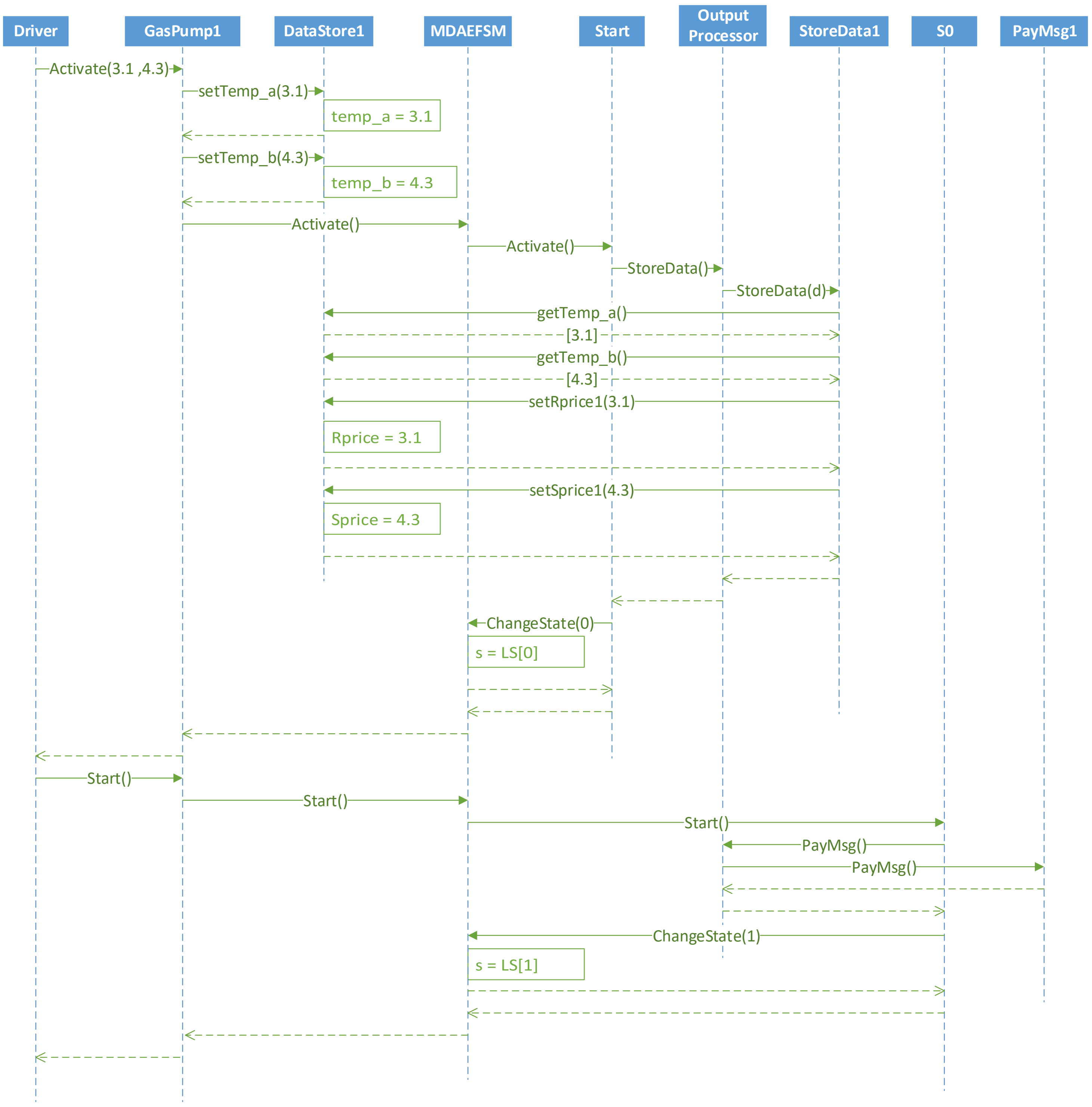
This class implements StoreData Interface and GasPump2Factory uses this class

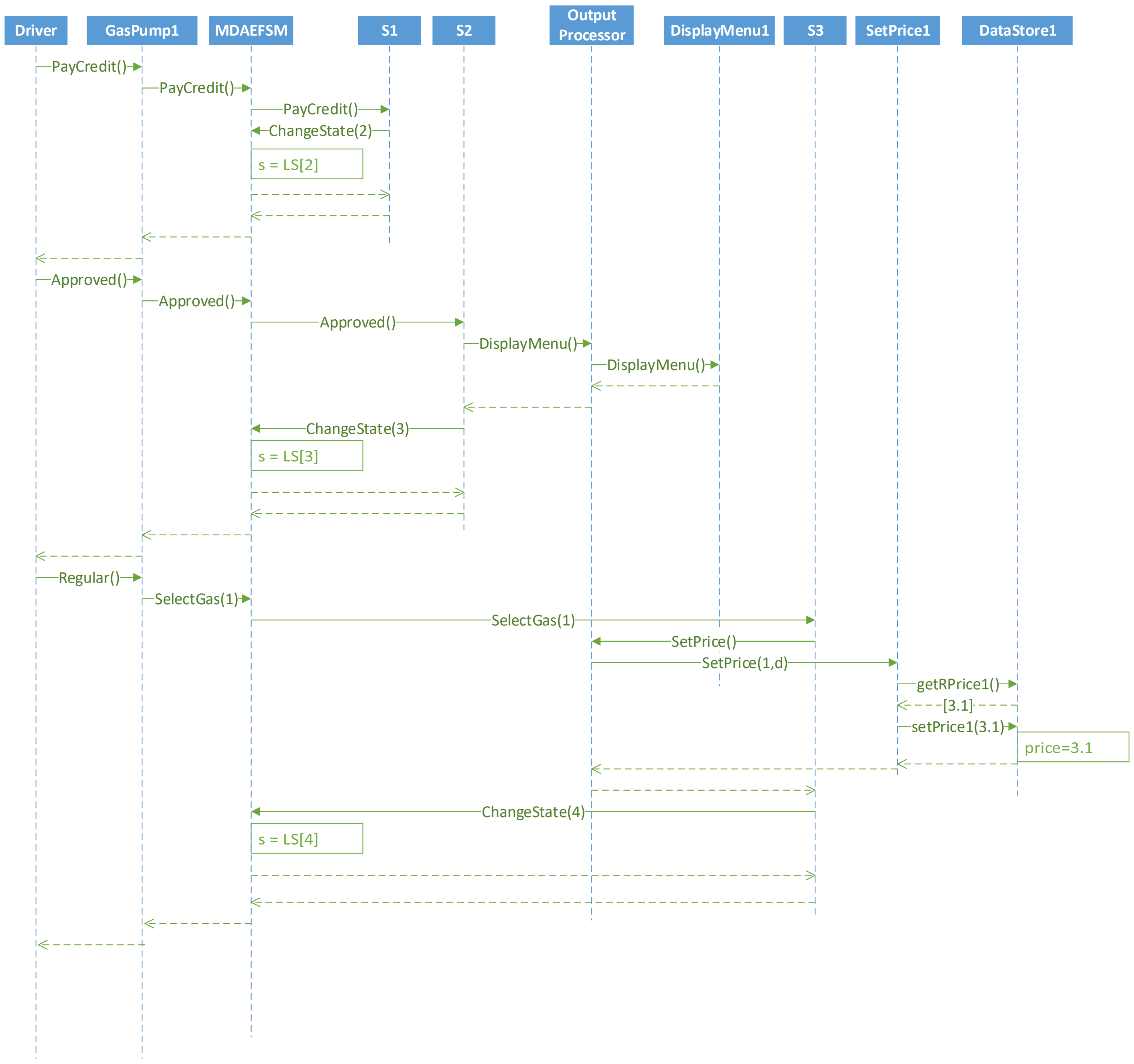
Operations:

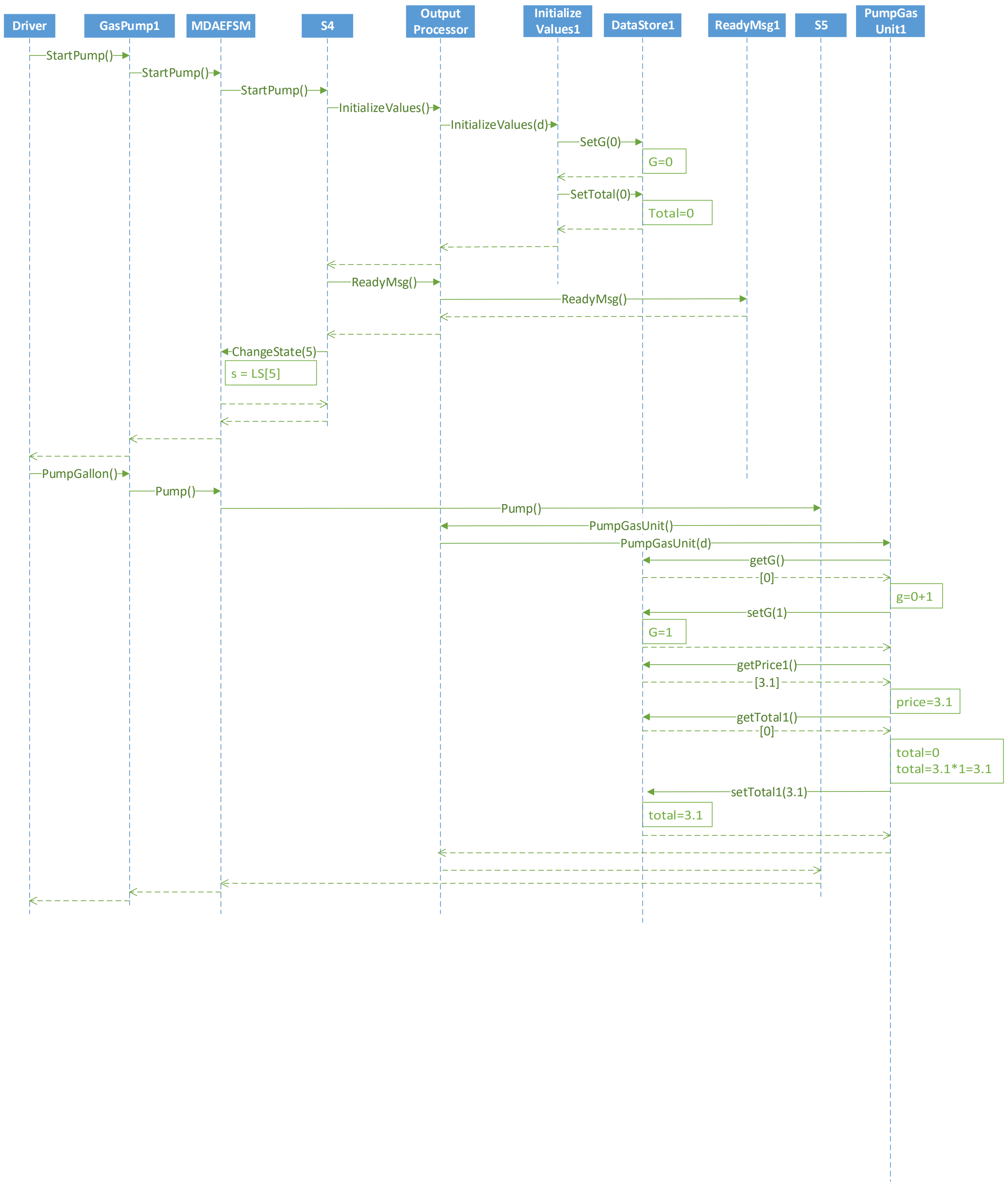
```
public void StoreData(DataStore d) {

Stores the prices of regular gas, super gas and premium gas in DataStore2 for GasPump2
```

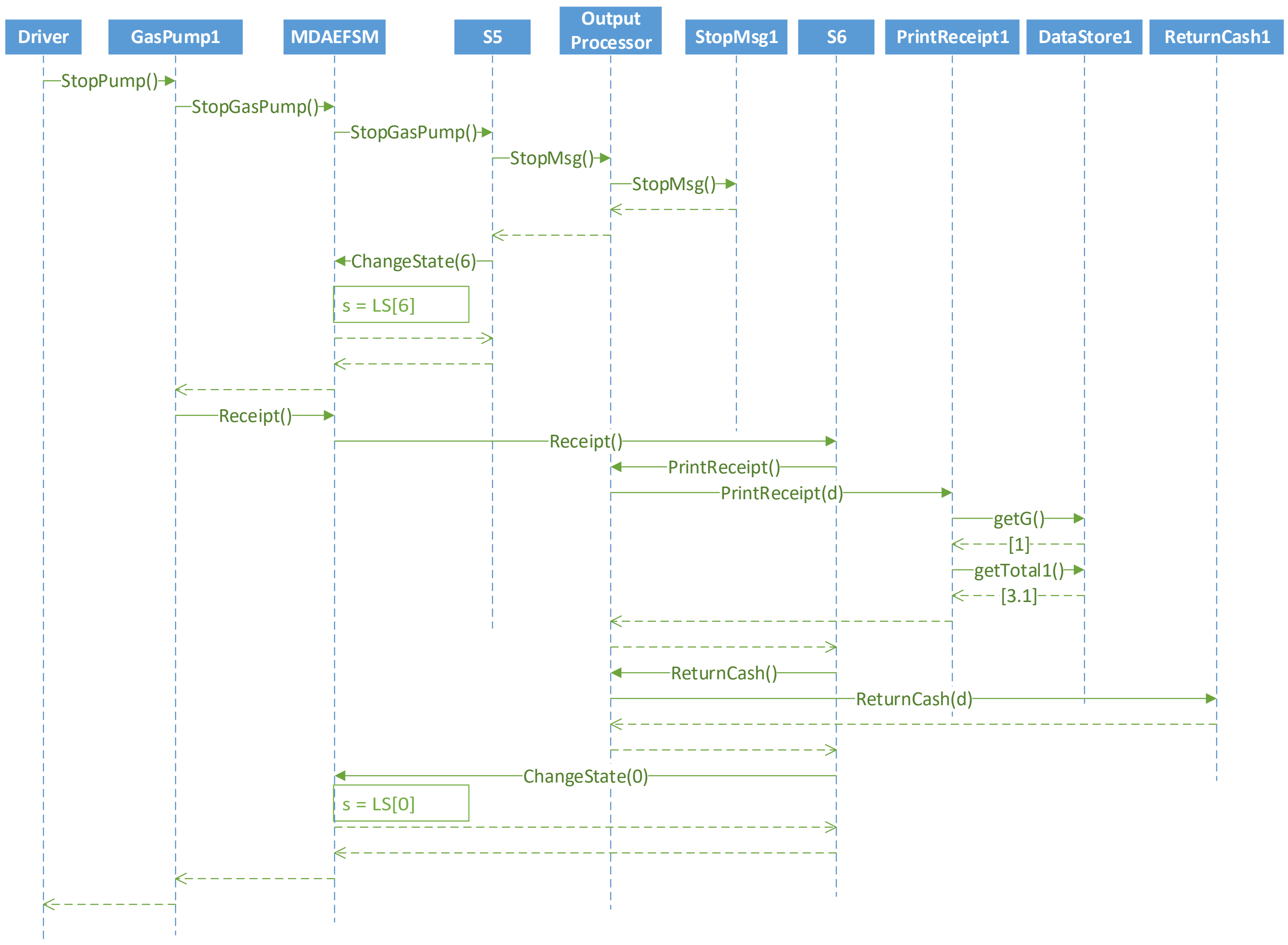
4. Sequence Diagrams:

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

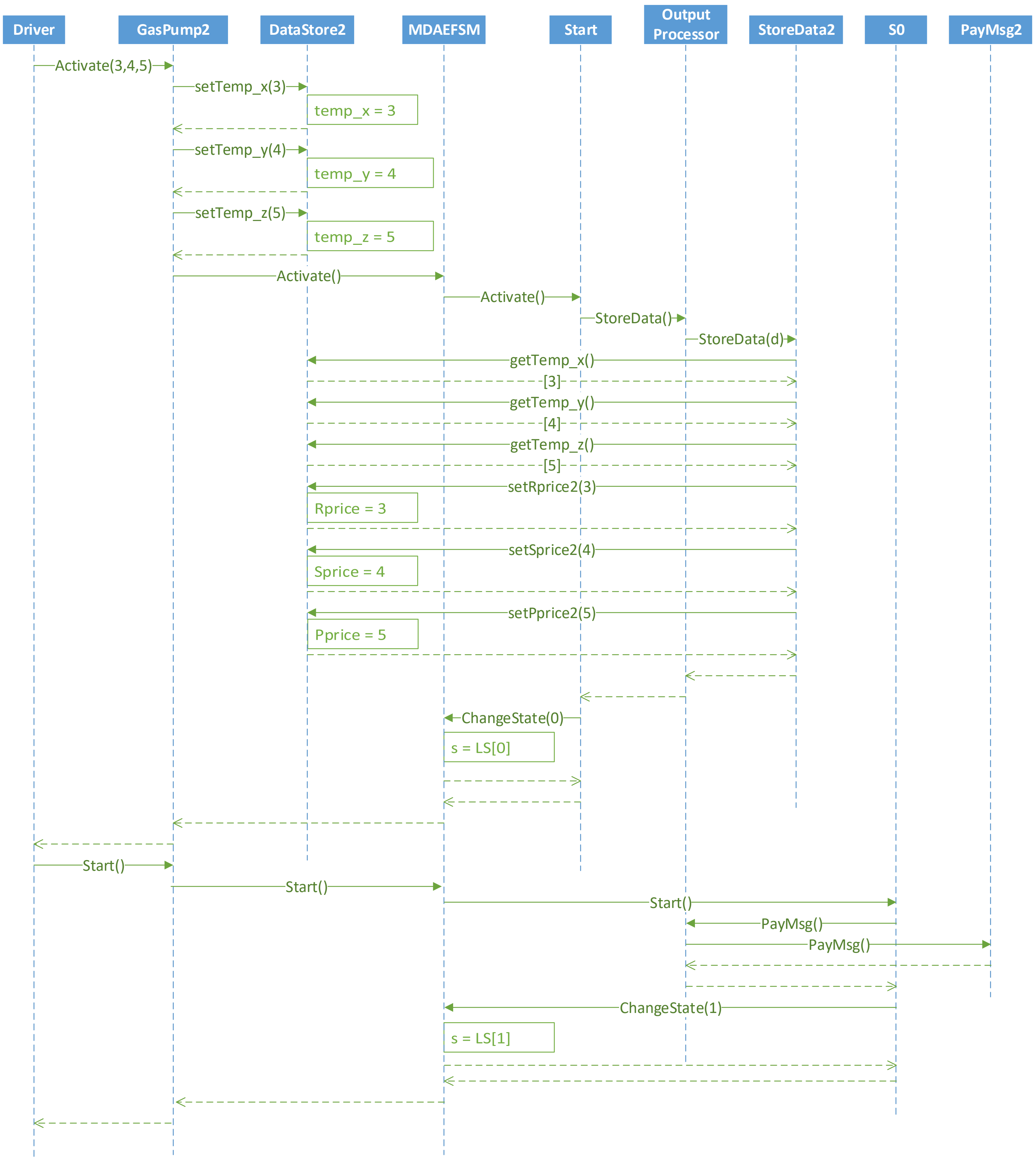


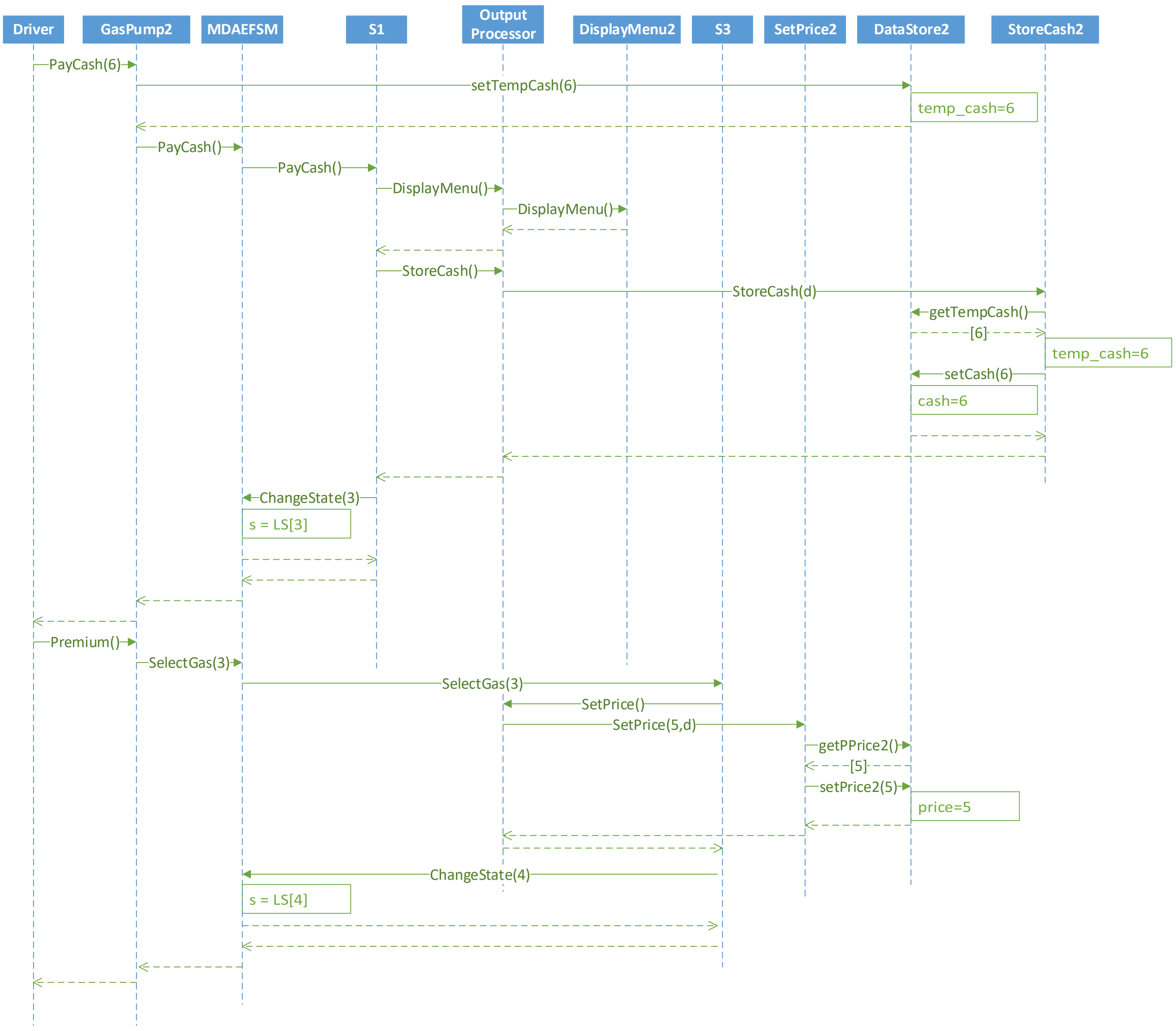

Driver
GasPump1
MDAEFSM
S1
S2
Output Processor
DisplayMenu1
S3
SetPrice1
DataStore1
PayCredit()
PayCredit()
PayCredit()
ChangeState(2)
s = LS[2]
Approved()
Approved()
Approved()
DisplayMenu()
DisplayMenu()
ChangeState(3)
s = LS[3]
Regular()
SelectGas(1)
SelectGas(1)
SetPrice()
SetPrice(1,d)
getRPrice1()
[3.1]
setPrice1(3.1)
price=3.1
ChangeState(4)
s = LS[4]

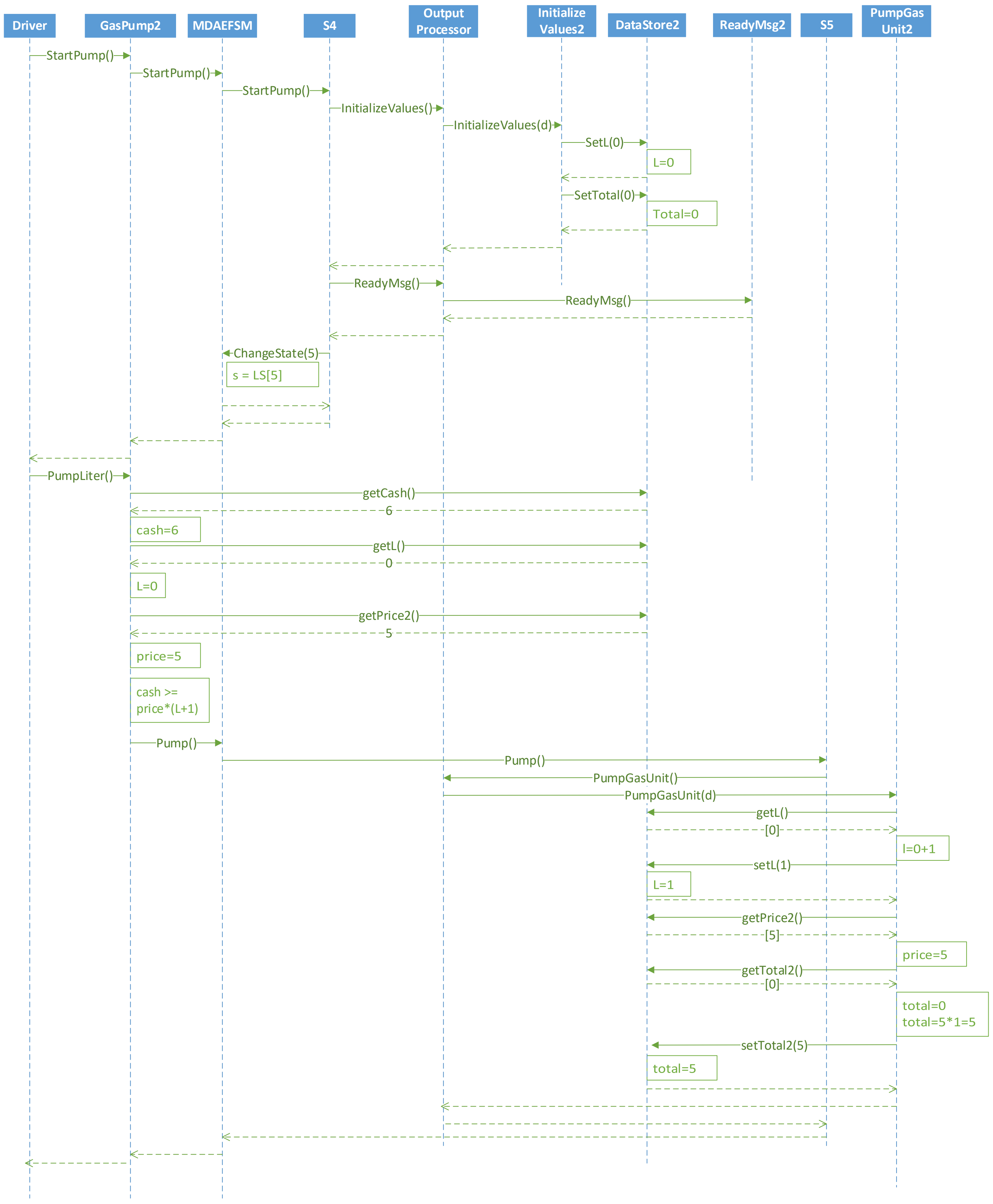

Driver
GasPump1
MDAEFSM
S4
Output Processor
Initialize Values1
DataStore1
ReadyMsg1
S5
PumpGas Unit1
StartPump()
StartPump()
StartPump()
InitializeValues()
InitializeValues(d)
SetG(0)
G=0
SetTotal(0)
Total=0
ReadyMsg()
ReadyMsg()
ChangeState(5)
s = LS[5]
PumpGallon()
Pump()
Pump()
PumpGasUnit()
PumpGasUnit(d)
getG()
[0]
g=0+1
setG(1)
G=1
getPrice1()
[3.1]
price=3.1
getTotal1()
[0]
total=0
total=3.1*1=3.1
setTotal1(3.1)
total=3.1

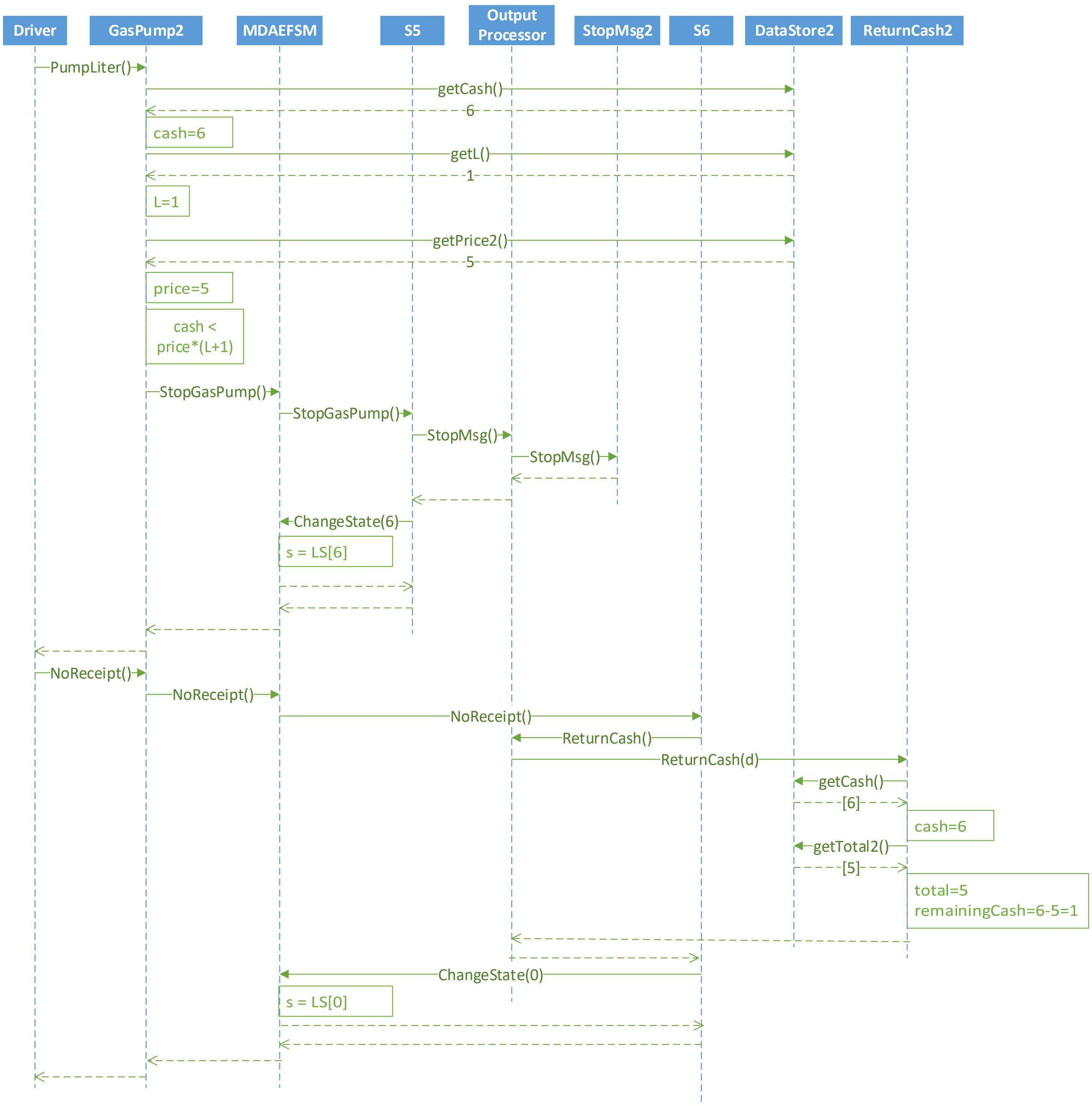

Driver
GasPump1
MDAEFSM
S5
Output Processor
StopMsg1
S6
PrintReceipt1
DataStore1
ReturnCash1
StopPump()
StopGasPump()
StopGasPump()
StopMsg()
StopMsg()
ChangeState(6)
s = LS[6]
Receipt()
Receipt()
PrintReceipt()
PrintReceipt(d)
getG()
[1]
getTotal1()
[3.1]
ReturnCash()
ReturnCash(d)
ChangeState(0)
s = LS[0]

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




Driver
GasPump2
MDAEFSM
S1
Output Processor
DisplayMenu2
S3
SetPrice2
DataStore2
StoreCash2
PayCash(6)
setTempCash(6)
temp_cash=6
PayCash()
PayCash()
DisplayMenu()
DisplayMenu()
StoreCash()
StoreCash(d)
getTempCash()
[6]
temp_cash=6
setCash(6)
cash=6
ChangeState(3)
s = LS[3]
Premium()
SelectGas(3)
SelectGas(3)
SetPrice()
SetPrice(5,d)
getPPrice2()
[5]
setPrice2(5)
price=5
ChangeState(4)
s = LS[4]


Driver
GasPump2
MDAEFSM
S4
Output Processor
Initialize Values2
DataStore2
ReadyMsg2
S5
PumpGas Unit2
StartPump()
StartPump()
StartPump()
InitializeValues()
InitializeValues(d)
SetL(0)
L=0
SetTotal(0)
Total=0
ReadyMsg()
ReadyMsg()
ChangeState(5)
s = LS[5]
PumpLiter()
getCash()
6
cash=6
getL()
0
L=0
getPrice2()
5
price=5
cash >= price*(L+1)
Pump()
Pump()
PumpGasUnit()
PumpGasUnit(d)
getL()
[0]
l=0+1
setL(1)
L=1
getPrice2()
[5]
price=5
getTotal2()
[0]
total=0
total=5*1=5
setTotal2(5)
total=5


Driver
GasPump2
MDAEFSM
S5
Output Processor
StopMsg2
S6
DataStore2
ReturnCash2
PumpLiter()
getCash()
6
cash=6
getL()
1
L=1
getPrice2()
5
price=5
cash < price*(L+1)
StopGasPump()
StopGasPump()
StopMsg()
StopMsg()
ChangeState(6)
s = LS[6]
NoReceipt()
NoReceipt()
NoReceipt()
ReturnCash()
ReturnCash(d)
getCash()
[6]
cash=6
getTotal2()
[5]
total=5
remainingCash=6-5=1
ChangeState(0)
s = LS[0]