Mohamed Azhar Shaikh
A20390780

Project Gas Pump
CS-586-01: Software Systems Arch
Spring 2017

Contact Information:
(mshaikh7@hawk.iit.edu)

Table of Contents

1. MDA-EFSM model for the GasPump components: 4

2. Class diagram(s) of the MDA and Gas Pump Components: 9

- 1. Overview 10
- 2 . GasPumps & MDA-EFSM 11
- 3. State Pattern 14
- 4. Strategy Pattern 19
- 5. Abstract Factory Pattern 30
- 6. Data Store 35

3. Dynamics 36

4. Pattern Description 44

1. MDA-EFSM model for the GasPump components:

This section includes the MDA-EFSM model for the GasPump components along with a list of the events for the MDA-EFSM, a list of actions for the MDA-EFSM, a State diagram of the MDA-EFSM, and pseudo-code for all operations of the input processors of GasPump1 and GasPump2.

For this implementation, I have chosen to implement the MDA model solution provided by myself which is as follows:

i. MDA-EFSM Events:

Activate()
Start()
Pay(int t)
Approved()
Reject()
Cancel()
SelectGas(int g)
StartPump()
Pump()
StopPump()
Receipt()
NoReceipt()

ii. MDA-EFSM Actions:

Action	Description/Responsibility
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
CancelMsg()	displays a cancellation message
SetPrice(int g)	set the price for the gas identified by g
SetInitialValues()	set initial value of Gallon(G) or Liter(L) to 0
ReadyMsg()	displays the ready for pumping message
PumpGasUnit()	disposes unit of gas, counts # of units disposed
TotalPrice()	calculates the total price
GasPumpedMsg()	displays the amount of disposed gas
StopMsg()	displays a stop pump message
PrintReceipt()	print a receipt
ReturnCash()	when payment method is cash, returns the remaining cash.

iii. Pseudo-code of all operations of the Input Processor (GasPump-1)

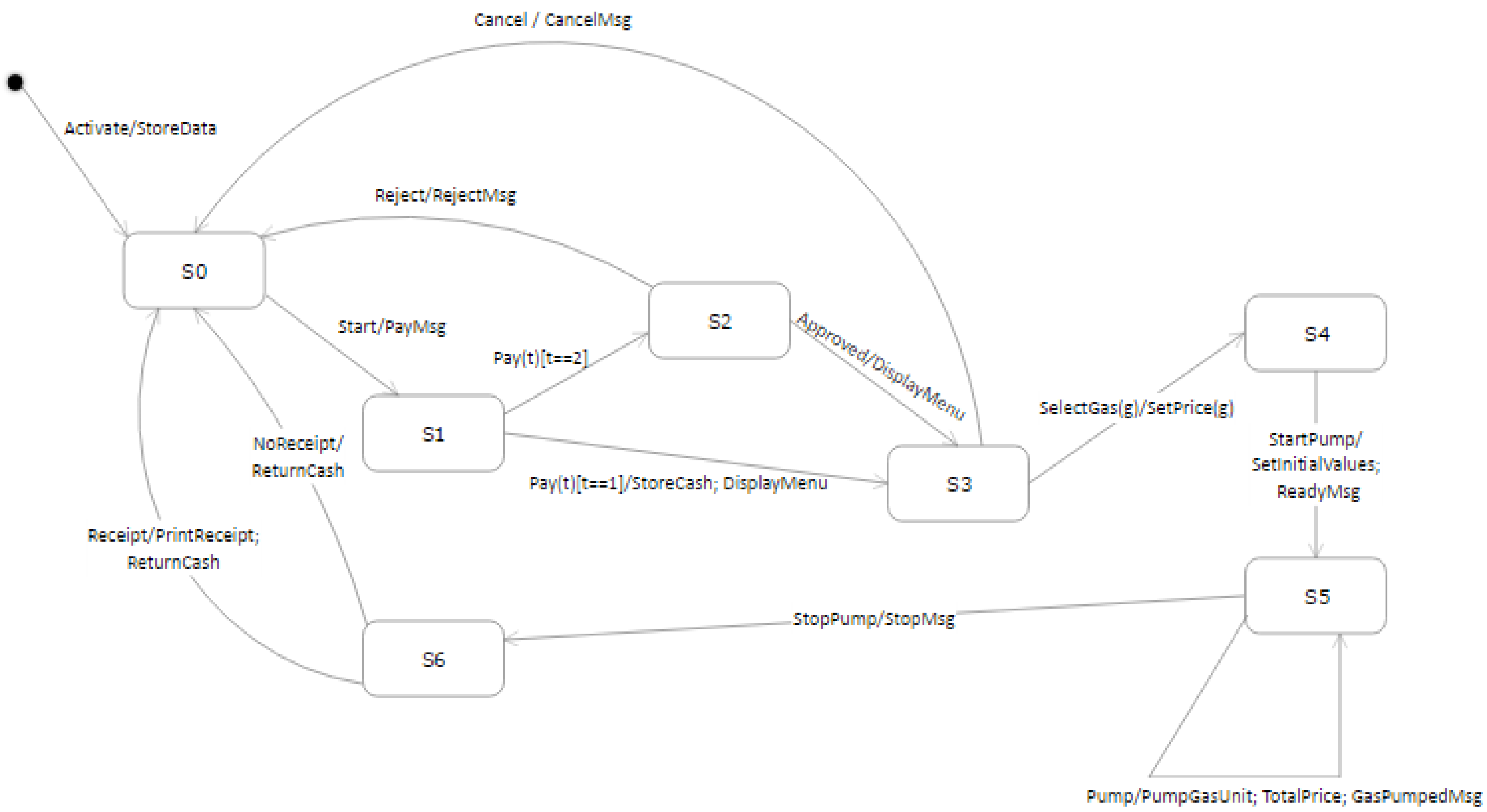
```
MDA_EFSM *m;        // pointer to the MDA-EFSM object
DataStore *ds;      // pointer to the Data Store object

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->Pay(2);  // 1-PayCash, 2-PayCredit
}

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

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

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

Super() {
        m->SelectGas(2)   // 1-Regular, 2-Super
}

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

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

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

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

Pseudo-code of all 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(int c) {
        if (c>0) {
                d->temp_c=c;
                m->Pay(1);
        }
}

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

Premium() {
        m->SelectGas(3);   // 1-Regular, 2-Super, 3-Premium
}

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

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

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

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

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

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

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

iv. State diagram of the MDA-EFSM



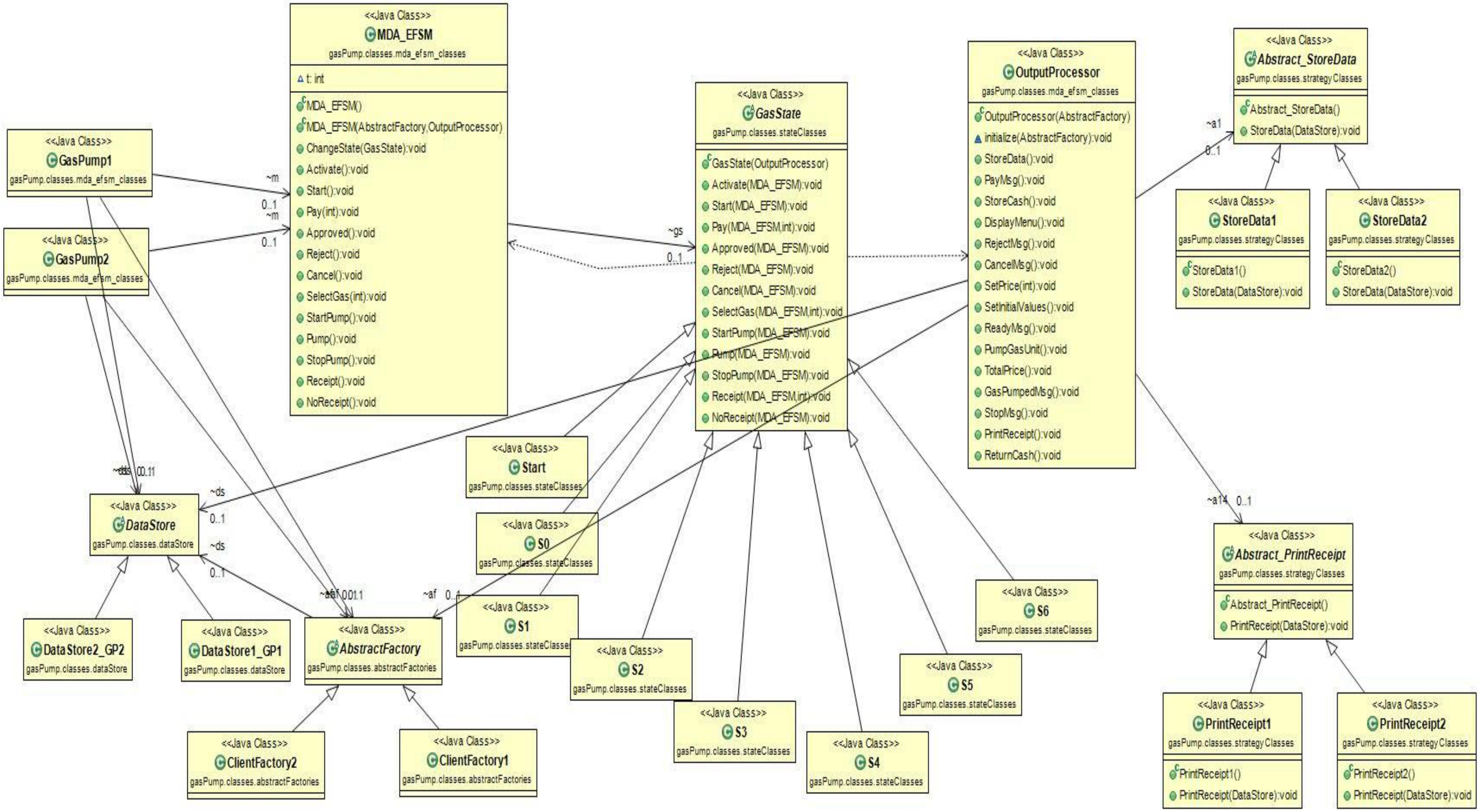
2. Class diagram(s) of the MDA and Gas Pump Components:

This section includes class diagrams along with a description of the class responsibilities and the responsibility of each operation supported by each class. Due to complexity of showing the class diagram as a whole, it is shown in parts as follows;

1. Overview – Showing all classes(compressed) with all relationships except the Concrete Factory Associations
2. GasPump & MDA-EFSM – Breakout & class details
3. State Pattern – Breakout & class details
4. Strategy Pattern – Breakout & class details
5. Abstract Factory Pattern – Breakout & class details (Included Data Store in this section)
6. Data Store – Overview of Objects

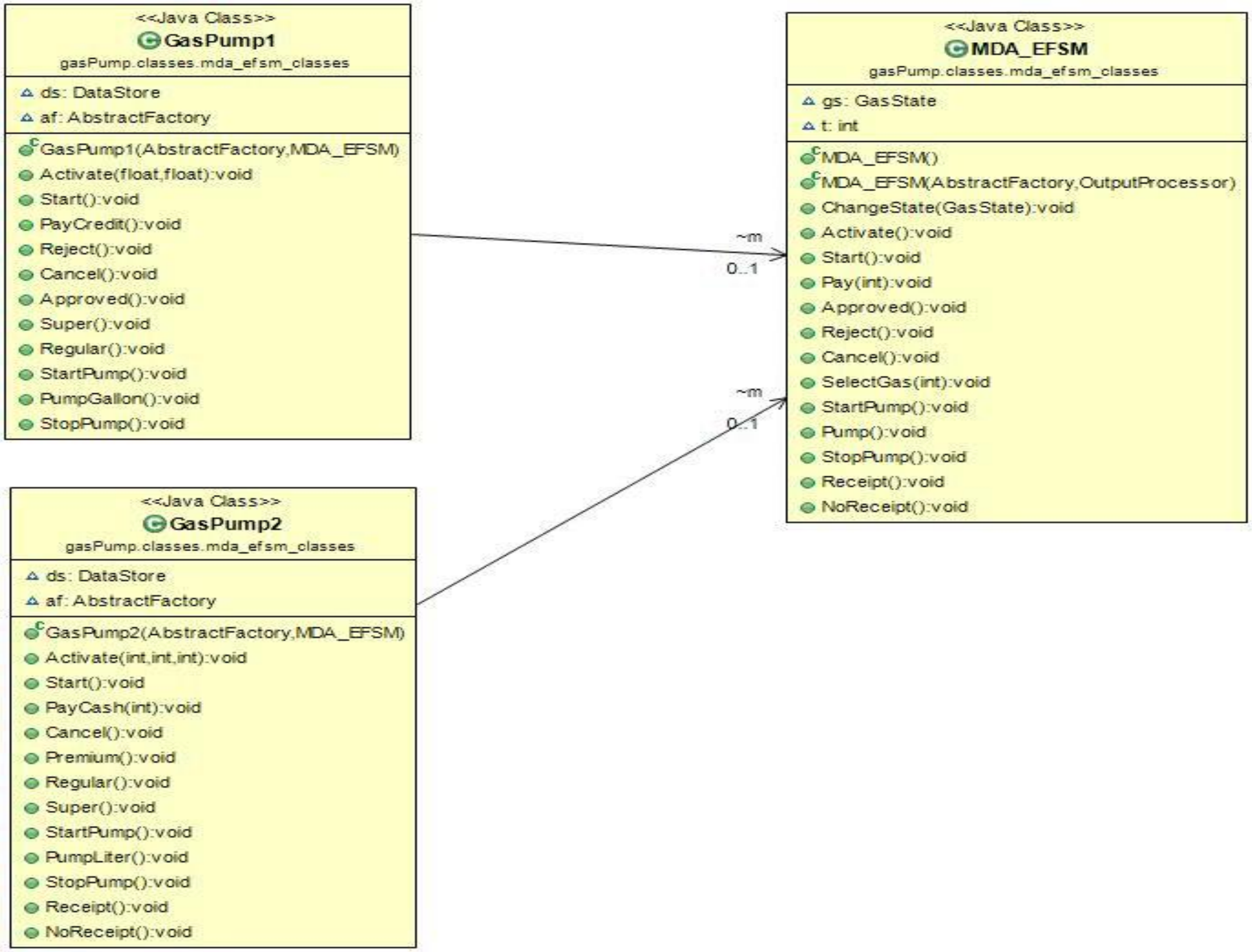
1. Overview – Compressed Classes without Concrete Factory Relationships shown as below.

(Note: Some classes attributes and operations are minimized due to large no. of classes. Also, there are many strategy classes but for the below overview only a few have been shown)



2 . GasPumps & MDA-EFSM – Focused view on only the relationship between GasPump machines & MDA-EFSM (Data Store & Abstract Factory Excluded)

(Note: Each GasPump class has a field m which is a pointer to MDA_EFSM contained in the class. Below the field is represented on the Association)



Purpose, Attributes for class operations and parameters (GasPump1 and GasmPump2)

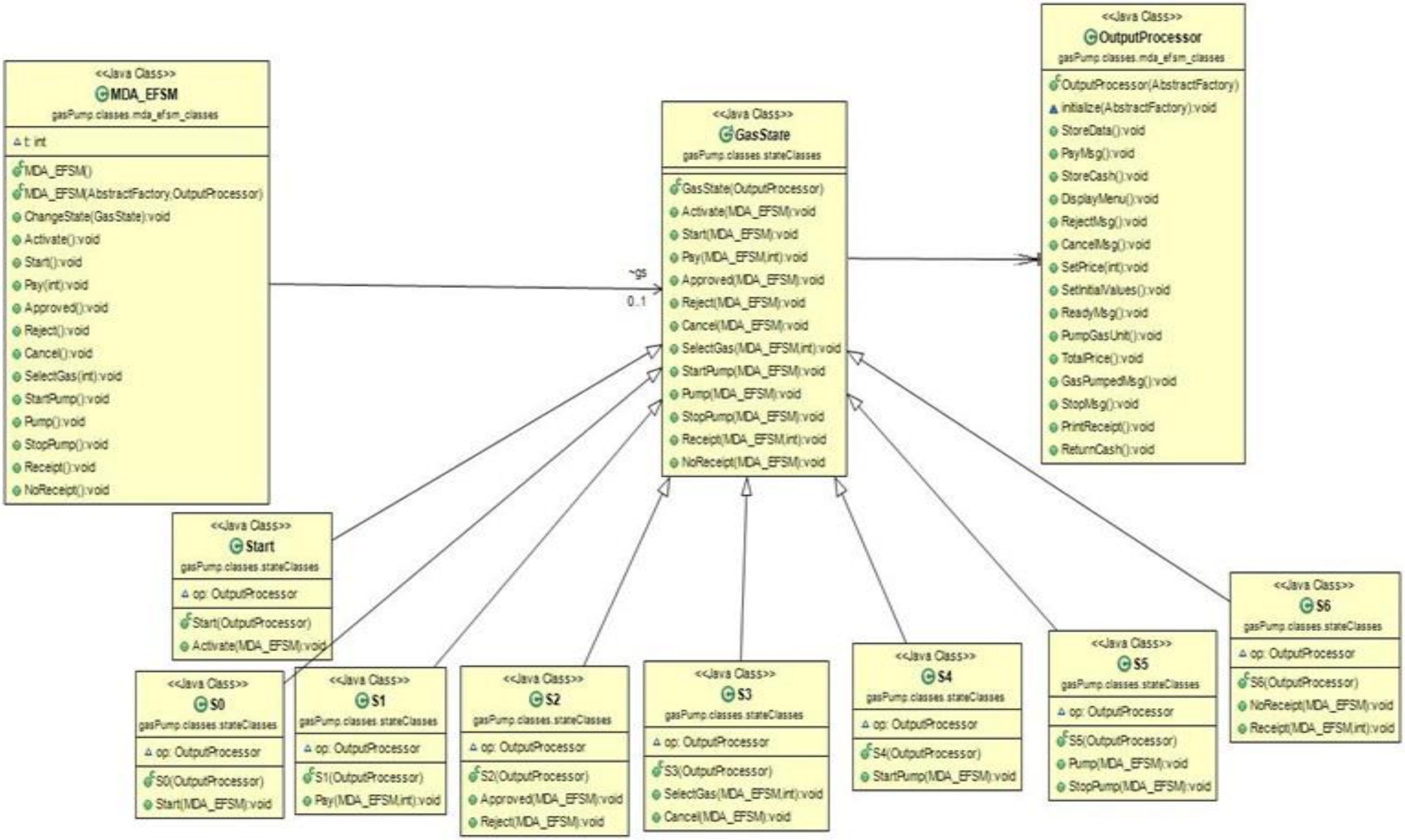
Class: GasPump1	
Purpose	This class represents a specific Gas pump implementation and contains any input specific needs of that Gas pump. As an example one specific need of this Gas pump is that it accepts payment method in the form of Credit only.
Variables & Initialization	m is a pointer to the MDA_EFSM ds is a pointer to the data store af is a pointer to the abstract factory GasPump1(AbstractFactory af1, MDA_EFSM m1) is the class constructor and is fired upon creation. It addresses any initialization and assignment needs.
GasPump1(AbstractFactory af1, MDA_EFSM m1)	Assign Concrete Factory for GasPump1 (ClientFactory1) passed by the driver and store it in af Retrieve the data store from the factory (af->getDataStore) and store in ds Assign MDA_EFSM object passed by the driver and store it in m
All other Methods	Pseudo code is shown in section 1 MDA-EFSM model for the GasPump components

Class: GasPump2	
Purpose	This class represents a specific Gas pump implementation and contains any input specific needs of that Gas pump. As an example one specific need of this Gas pump is that it accepts payment method in the form of Cash only.
Variables & Initialization	m is a pointer to the MDA_EFSM ds is a pointer to the data store af is a pointer to the abstract factory GasPump2(AbstractFactory af1, MDA_EFSM m1) is the class constructor and is fired upon creation. It addresses any initialization and assignment needs.
GasPump2(AbstractFactory af1, MDA_EFSM m1)	Assign Concrete Factory for GasPump2 (ClientFactory2) passed by the driver and store it in af Retrieve the data store from the factory (af->getDataStore) and store in ds Assign MDA_EFSM object passed by the driver and store it in m
All other Methods	Pseudo code is shown in section 1 MDA-EFSM model for the GasPump components

Purpose, Attributes for class operations and parameters (MDA_EFSM)

Class: MDA_EFSM	
Purpose	This class represents the common functionality or platform independent logic of all its clients (e.g. GasPump1 and GasPump2). It acts as a context class for State pattern. The separation of this functionality allows for reduced effort during maintenance and when new clients are implemented. As it is a separate module, it is loosely coupled and can be used with future GasPump models as well.
Variables & Initialization	GasState gs ; pointer to current gas state Int t ; to store type of payment method MDA_EFSM(AbstractFactory af, OutputProcessor op1)is the class constructor and is fired upon creation. It addresses any initialization needs. Note that the initialization receives a pointer to the abstract factory. This pointer is passed along to the Output Processor for use.
MDA_EFSM(AbstractFactory af, OutputProcessor op1)	Create a new Start State (new Start(op1)) passing along the OutputProcessor object and storing the pointer to the current state in gs.
ChangeState(GasState gs1)	Assigns the current state pointer to the new state pointed by gs1
Activate()	Call current GasState method with a pointer back to the MDA_EFSM (gs.Activate(this))
Start()	Call current GasState method with a pointer back to the MDA_EFSM (gs.Start(this))
void Pay(int t)	Call current GasState method with a pointer back to the MDA_EFSM (gs.Pay(this,t))
Approved()	Call current GasState method with a pointer back to the MDA_EFSM (gs.Approved(this))
Reject()	Call current GasState method with a pointer back to the MDA_EFSM (gs.Reject(this))
Cancel()	Call current GasState method with a pointer back to the MDA_EFSM (gs.Cancel(this))
SelectGas(int g)	Call current GasState method with a pointer back to the MDA_EFSM (gs.SelectGas(this,g))
StartPump()	Call current GasState method with a pointer back to the MDA_EFSM (gs.StartPump(this))
Pump()	Call current GasState method with a pointer back to the MDA_EFSM (gs.Pump(this))
StopPump()	Call current GasState method with a pointer back to the MDA_EFSM (gs.StopPump(this))
Receipt()	Call current GasState method with a pointer back to the MDA_EFSM (gs.Receipt(this,t))
NoReceipt()	Call current GasState method with a pointer back to the MDA_EFSM (gs.NoReceipt(this))

3. State Pattern – Focused view on only the relationship between MDA-EFSM (Context Class), the State Pattern Classes and OP



Purpose, Attributes for class operations and parameters (State Pattern Classes)

(Note: Context Class of State Pattern - MDA_EFSM described in prior section)

Class: GasState	
Purpose	This class represents the State abstract class that all of the States inherit. All methods at this level perform no actions. Any state specific actions are performed by the concrete state classes.
Variables & Initialization	GasState(OutputProcessor op)is the class constructor and is fired upon creation. It addresses any initialization if needed.
GasState(OutputProcessor op)	No Initialization needed
Activate(MDA_EFSM m)	No logic implemented, overridden by the child class if logic is needed for the state
Start(MDA_EFSM m)	No logic implemented, overridden by the child class if logic is needed for the state
Pay(MDA_EFSM m, int t)	No logic implemented, overridden by the child class if logic is needed for the state
Approved(MDA_EFSM m)	No logic implemented, overridden by the child class if logic is needed for the state
Reject(MDA_EFSM m)	No logic implemented, overridden by the child class if logic is needed for the state
Cancel(MDA_EFSM m)	No logic implemented, overridden by the child class if logic is needed for the state
SelectGas(MDA_EFSM m,int g)	No logic implemented, overridden by the child class if logic is needed for the state
StartPump(MDA_EFSM m)	No logic implemented, overridden by the child class if logic is needed for the state
Pump(MDA_EFSM m)	No logic implemented, overridden by the child class if logic is needed for the state
StopPump(MDA_EFSM m)	No logic implemented, overridden by the child class if logic is needed for the state
Receipt(MDA_EFSM m, int t)	No logic implemented, overridden by the child class if logic is needed for the state
NoReceipt(MDA_EFSM m)	No logic implemented, overridden by the child class if logic is needed for the state

Class: Start	
Purpose	This is the concrete class for the Start State, any edges leaving this state are implemented here and only executed when the MDA_EFSM is in this state.
Variables & Initialization	Start(OutputProcessor op) is the class constructor and is fired upon creation. It initializes the op object..
Start(OutputProcessor op)	Send informational note to user that this state is being initiated (print("System is currently in Start State")
Activate(MDA_EFSM m)	Call the StoreData() of the OutputProcessor class. Create new state S0 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S0(op)))

Class: S0	
Purpose	This is the concrete class for the State S0, any edges leaving this state are implemented here and only executed when the MDA_EFSM is in this state.
Variables & Initialization	S0(OutputProcessor op) is the class constructor and is fired upon creation. It initializes the op object.
S0(OutputProcessor op)	Send informational note to user that this state is being initiated (print("System is currently in Start S0")
Start(MDA_EFSM m)	Call the PayMsg() of the OutputProcessor class. Create new state S1 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S1(op)))

Class: S1	
Purpose	This is the concrete class for the State S1, any edges leaving this state are implemented here and only executed when the MDA_EFSM is in this state.
Variables & Initialization	S1(OutputProcessor op) is the class constructor and is fired upon creation. It initializes the op object.
S1(OutputProcessor op)	Send informational note to user that this state is being initiated (print("System is currently in Start S1")
Pay(MDA_EFSM m, int t)	if(t==1) Call the StoreCash() of the OutputProcessor class. Call the DisplayMenu()of the OutputProcessor class. Create new state S3 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S3(op))) Else if(t==2) Print("Waiting for Credit Approval") Create new state S2 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S2(op)))
Class: S2	
Purpose	This is the concrete class for the State S2, any edges leaving this state are implemented here and only executed when the MDA_EFSM is in this state.
Variables & Initialization	S2(OutputProcessor op) is the class constructor and is fired upon creation. It initializes the op object.
S2(OutputProcessor op)	Send informational note to user that this state is being initiated (print("System is currently in Start S2")
Approved(MDA_EFSM m)	Call the DisplayMenu()of the OutputProcessor class. Create new state S3 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S3(op)))
Reject(MDA_EFSM m)	Call the RejectMsg()of the OutputProcessor class. Create new state S0 and call MDA_EFSM ChangeState function to set the current state to the new state

	(ChangeState(new S0(op)))

Class: S3	
Purpose	This is the concrete class for the State S3, any edges leaving this state are implemented here and only executed when the MDA_EFSM is in this state.
Variables & Initialization	S3(OutputProcessor op) is the class constructor and is fired upon creation. It initializes the op object.
S3(OutputProcessor op)	Send informational note to user that this state is being initiated (print("System is currently in Start S3")
SelectGas(MDA_EFSM m, int g)	Call the SetPrice(g) of the OutputProcessor class. Create new state S4 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S4(op)))
Cancel(MDA_EFSM m)	Call the CancelMsg()of the OutputProcessor class. Create new state S0 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S0(op)))

Class: S4	
Purpose	This is the concrete class for the State S4, any edges leaving this state are implemented here and only executed when the MDA_EFSM is in this state.
Variables & Initialization	S4(OutputProcessor op) is the class constructor and is fired upon creation. It initializes the op object.
S4(OutputProcessor op)	Send informational note to user that this state is being initiated (print("System is currently in Start S4")
StartPump(MDA_EFSM m)	Call the SetInitialValues()of the OutputProcessor class. Call the ReadyMsg()of the OutputProcessor class. Create new state S5 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S5(op)))

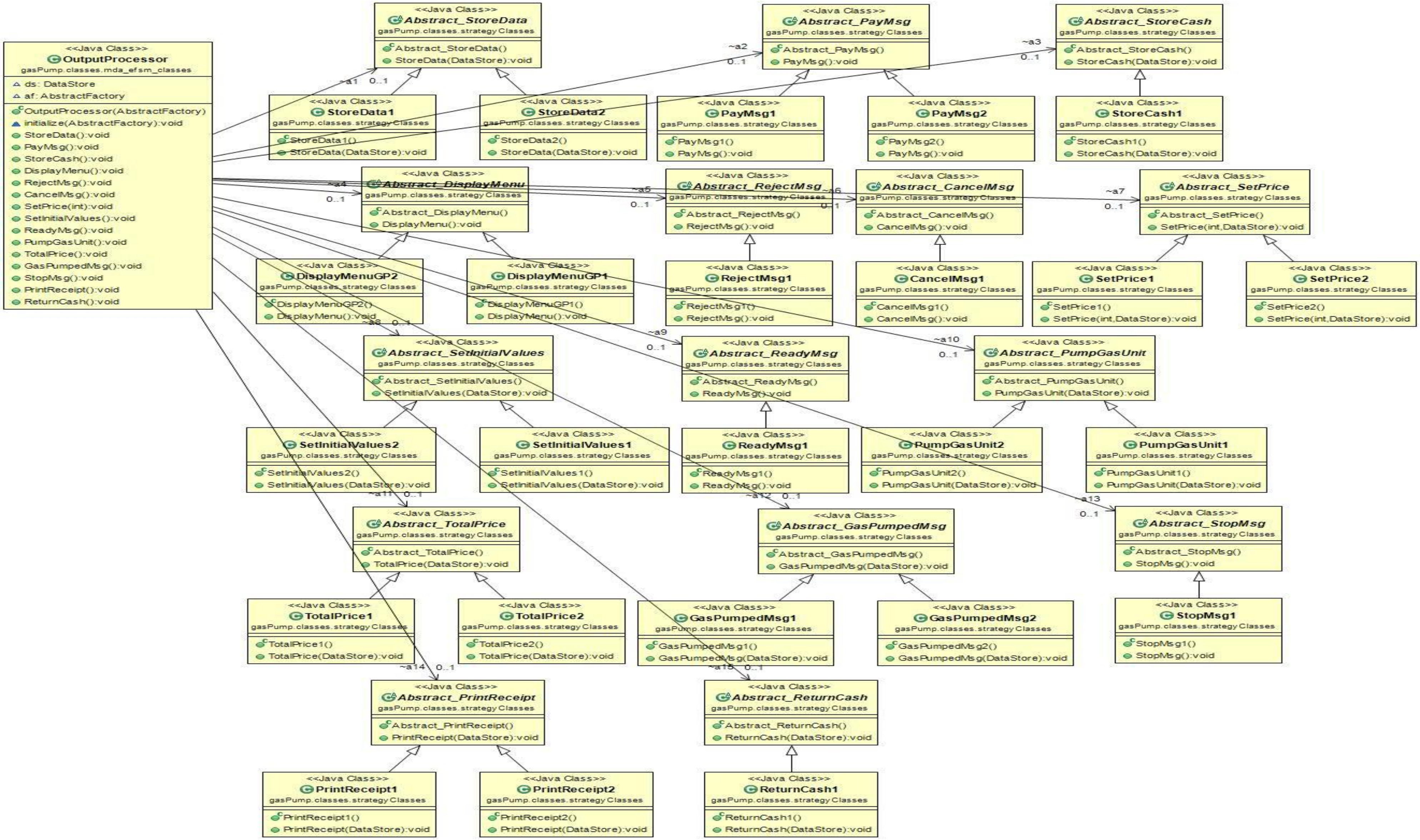
Class: S5	
Purpose	This is the concrete class for the State S5, any edges leaving this state are implemented here and only executed when the MDA_EFSM is in this state.
Variables & Initialization	S5(OutputProcessor op) is the class constructor and is fired upon creation. It initializes the op object.
S5(OutputProcessor op)	Send informational note to user that this state is being initiated (print("System is currently in Start S5")
Pump(MDA_EFSM m)	Call the PumpGasUnit ()of the OutputProcessor class. Call the GasPumpedMsg ()of the OutputProcessor class. Call the TotalPrice ()of the OutputProcessor class.
StopPump(MDA_EFSM m)	Call the StopMsg()of the OutputProcessor class.

	Create new state S6 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S6(op)))

Class: S6	
Purpose	This is the concrete class for the State S6, any edges leaving this state are implemented here and only executed when the MDA_EFSM is in this state.
Variables & Initialization	S6(OutputProcessor op) is the class constructor and is fired upon creation. It initializes the op object.
S6(OutputProcessor op)	Send informational note to user that this state is being initiated (print("System is currently in Start S6")
NoReceipt(MDA_EFSM m)	Call the ReturnCash()of the OutputProcessor class. Create new state S0 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S0(op)))

Receipt(MDA_EFSM m, int t)	Call the Printreceipt()of the OutputProcessor class.
	If(t==1)
	  Call the ReturnCash()of the OutputProcessor class.
	Create new state S0 and call MDA_EFSM ChangeState function to set the current state to the new state (ChangeState(new S0(op)))

4. Strategy Pattern – Focused view on only the relationship between OP and the Strategy Pattern Classes



Purpose, Attributes for class operations and parameters (Strategy Pattern Classes)

Class: OutputProcessor - OP	
Purpose	This class represents the Output Processor of the MDA and is the client of the various action strategies. Each action having more than one strategy has an abstract class which is associated to the OP, while actions having only one strategy are directly associated with the OP.
Variables & Initialization	af is a pointer to the Abstract Factory class and is used to generate the actions as needed ds is a pointer to the data store for this implementation a1 - a15 are pointers to their associated action classes OP(AbstractFactory af) is the class constructor and is fired upon creation. It addresses any initialization needs.
OP(AbstractFactory af)	Set the classes abstract factory pointer (af) to the factory passed (this.af=af) and calls the initialize(af) to get the store and set the classes pointer
initialize(AbstractFactory af)	Call the factory to get action 1 and set the classes pointer (a1 = af.getStoreData()) Call the factory to get action 2 and set the classes pointer (a2 = af.getPayMsg())) Call the factory to get action 3 and set the classes pointer (a3 = af.getStoreCash()) Call the factory to get action 4 and set the classes pointer (a4 = af.getDisplayMenu()) Call the factory to get action 5 and set the classes pointer (a5 = af.getRejectMsg()) Call the factory to get action 6 and set the classes pointer (a6 = af.getCancelMsg()) Call the factory to get action 7 and set the classes pointer (a7 = af.getSetPrice()) Call the factory to get action 8 and set the classes pointer (a8 = af.getSetInitialValues()) Call the factory to get action 9 and set the classes pointer (a9 = af.getReadyMsg()) Call the factory to get action 10 and set the classes pointer (a10 = af.getPumpGasUnit()) Call the factory to get action 11 and set the classes pointer (a11 = af.getTotalPrice()) Call the factory to get action 12 and set the classes pointer (a12 = af.getGasPumpedMsg()) Call the factory to get action 13 and set the classes pointer (a13 = af.getStopMsg()) Call the factory to get action 14 and set the classes pointer (a14 = af.getPrintReceipt()) Call the factory to get action 14 and set the classes pointer (a15 = af.getReturnCash())
StoreData()	Call action from strategy (a1.StoreData(ds))
PayMsg()	Call action from strategy (a2.PayMsg())
StoreCash()	Call action from strategy (a3.StoreCash(ds))
DisplayMenu()	Call action from strategy (a4.DisplayMenu())
RejectMsg()	Call action from strategy (a5.RejectMsg())
CancelMsg()	Call action from strategy (a6.CancelMsg())
SetPrice(int g)	Call action from strategy (a7.SetPrice(g,ds))
SetInitialValues()	Call action from strategy (a8.SetInitialValues(ds))
ReadyMsg()	Call action from strategy (a9.ReadyMsg())
PumpGasUnit()	Call action from strategy (a10.PumpGasUnit(ds))
TotalPrice()	Call action from strategy (a11.TotalPrice(ds))

GasPumpedMsg()	Call action from strategy (a12.GasPumpedMsg(ds))
StopMsg()	Call action from strategy (a13.StopMsg())
PrintReceipt()	Call action from strategy (a14.PrintReceipt(ds))
ReturnCash()	Call action from strategy (a15.ReturnCash(ds))

Class: Abstract_StoreData	
Purpose	This class represents the abstract class for action StoreData and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
StoreData(DataStore ds)	No logic implemented, overridden by the child class if logic is needed for the state

Class: StoreData1	
Purpose	This class represents the concrete class for action StoreData float version and is used to handle the storage of Regular,Super gas in float format.
Variables & Initialization	ds pointer passed by the OP class.
StoreData(DataStore ds)	getTemp_regularGas1 from datastore pointed by ds and set its value RegularGas1 in the data store. getTemp_superGas1 from datastore pointed by ds and set its value SuperGas1 in the data store.

Class: StoreData2	
Purpose	This class represents the concrete class for action StoreData int version and is used to handle the storage of Regular,Super and Premium gas in int format.
Variables & Initialization	ds pointer passed by the OP class.
StoreData(DataStore ds)	getTemp_regularGas2 from datastore pointed by ds and set its value RegularGas2 in the data store. getTemp_superGas2 from datastore pointed by ds and set its value SuperGas2 in the data store. getTemp_premiumGas2 from datastore pointed by ds and set its value PremiumGas2 in the data store.

Class: Abstract_PayMsg	
Purpose	This class represents the abstract class for action PayMsg and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
PayMsg()	No logic implemented, overridden by the child class if logic is needed for the state

Class: PayMsg1	
Purpose	This class represents the concrete class for action PayMsg Gasspump1 version and is used to print the payment message.
Variables & Initialization	No initialization needs.
PayMsg()	Print(" # Please Pay by : CREDIT # ")

Class: PayMsg2	
Purpose	This class represents the concrete class for action PayMsg Gasspump2 version and is used to print the payment message.
Variables & Initialization	No initialization needs.
PayMsg()	Print(" # Please Pay by : CASH # ")

Class: Abstract_StoreCash	
Purpose	This class represents the abstract class for action StoreCash and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
StoreCash()	No logic implemented, overridden by the child class if logic is needed for the state

Class: StoreCash1	
Purpose	This class represents the concrete class for action StoreCash Gasspump2 version and is used to store cash.
Variables & Initialization	ds pointer passed by the OP class.
StoreCash()	getTemp_cash from datastore pointed by ds and set its value to Cash in the data store.

Class: Abstract_DisplayMenu	
Purpose	This class represents the abstract class for action DisplayMenu and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
DisplayMenu()	No logic implemented, overridden by the child class if logic is needed for the state

Class: DisplayMenuGP1	
Purpose	This class represents the concrete class for action DisplayMenu Gasspump1 version and is used to print the display menu message.
Variables & Initialization	No initialization needs.
DisplayMenu()	Print(" Menu GP1 ")

Class: DisplayMenuGP2	
Purpose	This class represents the concrete class for action DisplayMenu Gasspump2 version and is used to print the display menu message.
Variables & Initialization	No initialization needs.
DisplayMenu()	Print(" Menu GP2 ")

Class: Abstract_RejectMsg	
Purpose	This class represents the abstract class for action RejectMsg and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
RejectMsg()	No logic implemented, overridden by the child class if logic is needed for the state

Class: RejectMsg1	
Purpose	This class represents the concrete class for action Reject Gasspump1 version and is used to print the transaction rejected message.

Variables & Initialization	No initialization needs.
RejectMsg()	Print(" Transaction Rejected !")

Class: Abstract_CancelMsg	
Purpose	This class represents the abstract class for action CancelMsg and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
CancelMsg()	No logic implemented, overridden by the child class if logic is needed for the state

Class: CancelMsg1	
Purpose	This class represents the concrete class for action CancelMsg and is used to print the cancel message.
Variables & Initialization	No initialization needs.
CancelMsg()	Print(" Current Transaction has been Cancelled ”)

Class: Abstract_SetPrice	
Purpose	This class represents the abstract class for action SetPrice and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
SetPrice(int g, DataStore ds)	No logic implemented, overridden by the child class if logic is needed for the state

Class: SetPrice1	
Purpose	This class represents the concrete class for action SetPrice GasPump1 version and is used to setting the price of Regular,Super gas in float format.
Variables & Initialization	ds pointer passed by the OP class.

SetPrice(int g, DataStore ds)	if(g==1)
	ds.setPrice(ds.getRegularGas1());
	else if(g==2)

	ds.setPrice(ds.getSuperGas1());

Class: SetPrice2	
Purpose	This class represents the concrete class for action SetPrice GasPump2 version and is used to setting the price of Regular, Super and Premium gas in int format.
Variables & Initialization	ds pointer passed by the OP class.

SetPrice(int g, DataStore ds)	if(g==1)
	ds.setPrice(ds.getRegularGas2());
	else if(g==2)
	ds.setPrice(ds.getSuperGas2());
	else if(g==3)
	ds.setPrice(ds.getPremiumGas2());

Class: Abstract_SetInitialValues	
Purpose	This class represents the abstract class for action SetInitialValues and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
SetInitialValues(DataStore ds)	No logic implemented, overridden by the child class if logic is needed for the state

Class: SetInitialValues1	
Purpose	This class represents the concrete class for action SetInitialValues GasPump1 version and is used set the initial value of Gallon and total to 0.
Variables & Initialization	ds pointer passed by the OP class.

SetInitialValues(DataStore ds)	ds.setG(0);
	ds.setTotal(0);

Class: SetInitialValues2	
Purpose	This class represents the concrete class for action SetInitialValues GasPump2 version and is used set the initial value of Liter and total to 0.
Variables & Initialization	ds pointer passed by the OP class.

SetInitialValues(DataStore ds)	ds.setL(0);
	ds.setTotal(0);

Class: Abstract_ReadyMsg	
Purpose	This class represents the abstract class for action ReadyMsg and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
ReadyMsg()	No logic implemented, overridden by the child class if logic is needed for the state

Class: ReadyMsg1	
Purpose	This class represents the concrete class for action ReadyMsg and is used to print the ready message.
Variables & Initialization	No initialization needs.
ReadyMsg()	Print(“GasPump Ready for Pumping !!!”)

Class: Abstract_PumpGasUnit	
Purpose	This class represents the abstract class for action PumpGasUnit and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
PumpGasUnit(DataStore ds)	No logic implemented, overridden by the child class if logic is needed for the state

Class: PumpGasUnit1	
Purpose	This class represents the concrete class for action PumpGasUnit GasPump1 version and is used to increment the no. of Gallons by 1
Variables & Initialization	ds pointer passed by the OP class.
PumpGasUnit(DataStore ds)	ds.setG(ds.getG()+1)

Class: PumpGasUnit2	
Purpose	This class represents the concrete class for action PumpGasUnit GasPump2 version and is used to increment the no. of Liters by 1
Variables & Initialization	ds pointer passed by the OP class.
PumpGasUnit(DataStore ds)	ds.setL(ds.getL()+1)

Class: Abstract_TotalPrice	
Purpose	This class represents the abstract class for action TotalPrice and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
TotalPrice(DataStore ds)	No logic implemented, overridden by the child class if logic is needed for the state

Class: TotalPrice1	
Purpose	This class represents the concrete class for action TotalPrice GasPump1 version and is used to set the total cost of pumping Gallons
Variables & Initialization	ds pointer passed by the OP class.
TotalPrice(DataStore ds)	ds.setTotal(ds.getPrice() * ds.getG())

Class: TotalPrice2	
Purpose	This class represents the concrete class for action TotalPrice GasPump2 version and is used to set the total cost of pumping Liters
Variables & Initialization	ds pointer passed by the OP class.
TotalPrice(DataStore ds)	ds.setTotal(ds.getPrice() * ds.getL())

Class: Abstract_GasPumpedMsg	
Purpose	This class represents the abstract class for action GasPumpedMsg and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
GasPumpedMsg(DataStore ds)	No logic implemented, overridden by the child class if logic is needed for the state

Class: GasPumpedMsg1	
Purpose	This class represents the concrete class for action GasPumpedMsg Gasspump1 version and is used to print the no of gallons pumped message.
Variables & Initialization	No initialization needs.
GasPumpedMsg(DataStore ds)	Print(" You have pumped " + ds.getG() + " Gallon(s) of gas ")

Class: GasPumpedMsg2	
Purpose	This class represents the concrete class for action GasPumpedMsg Gasspump2 version and is used to print the no of liters pumped message.
Variables & Initialization	No initialization needs.
GasPumpedMsg(DataStore ds)	Print(" You have pumped " + ds.getL() + " Liters(s) of gas ")

Class: Abstract_StopMsg	
Purpose	This class represents the abstract class for action StopMsg and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
StopMsg()	No logic implemented, overridden by the child class if logic is needed for the state

Class: StopMsg1	
Purpose	This class represents the concrete class for action StopMsg and is used to print the stopped pumping message.
Variables & Initialization	No initialization needs.
StopMsg()	Print(" Stopped Pumping !”)

Class: Abstract_PrintReceipt	
Purpose	This class represents the abstract class for action PrintReceipt and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
PrintReceipt(DataStore ds)	No logic implemented, overridden by the child class if logic is needed for the state

Class: PrintReceipt1	
Purpose	This class represents the concrete class for action PrintReceipt Gasspump1 version and is used to print receipt
Variables & Initialization	No initialization needs.

PrintReceipt(DataStore ds)	Print(" No. of Gallons Pumped = "+ds.getG())
	Print(" Total Price = $ "+ds.getTotal())

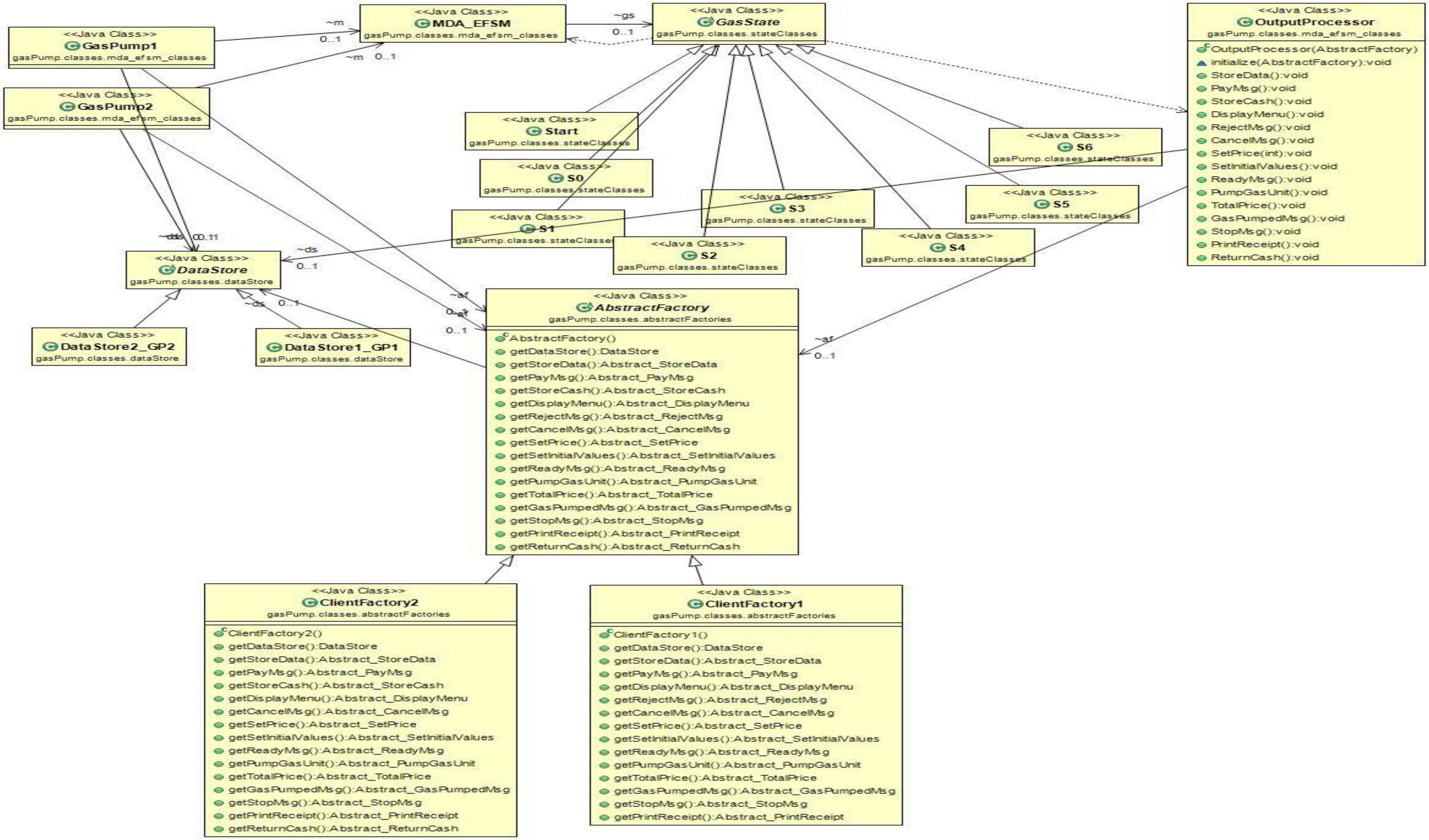
Class: PrintReceipt2	
Purpose	This class represents the concrete class for action PrintReceipt Gasspump2 version and is used to print receipt
Variables & Initialization	No initialization needs.

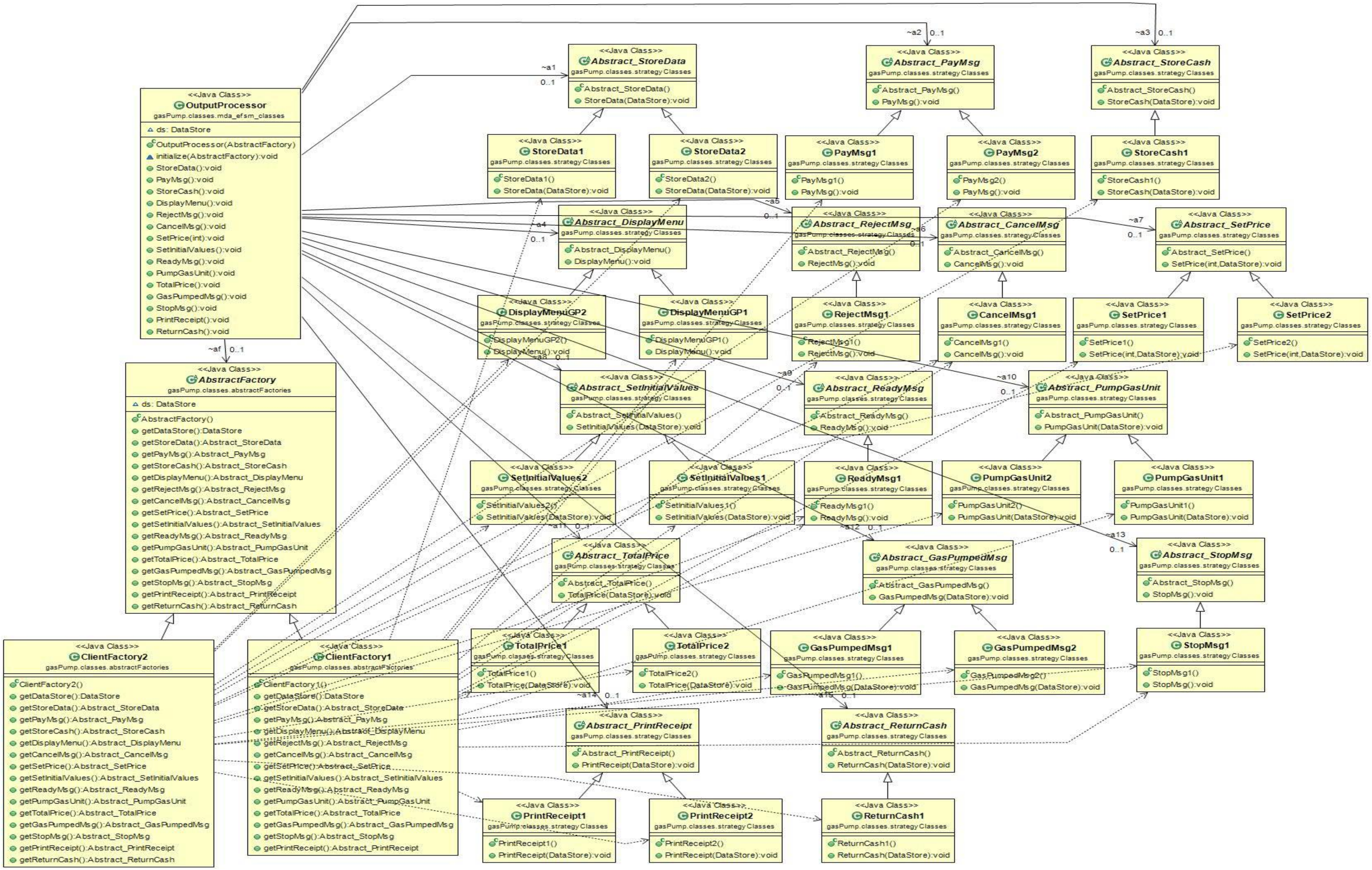
PrintReceipt(DataStore ds)	Print(" No. of Liters Pumped = "+ds.getL())
	Print(" Total Price = $ "+ds.getTotal())

Class: Abstract_ReturnCash	
Purpose	This class represents the abstract class for action ReturnCash and is used to group the various strategies used for completing the action. Action classes are instantiated by the factory and the factory provides the implementations data store pointer upon creation.
Variables & Initialization	No initialization needs.
ReturnCash(DataStore ds)	No logic implemented, overridden by the child class if logic is needed for the state

Class: ReturnCash1	
Purpose	This class represents the concrete class for action ReturnCash Gasspump2 version and is used to return the remaining cash.
Variables & Initialization	ds pointer passed by the OP class.
ReturnCash(DataStore ds)	float r_cash = ds.getCash()-ds.getTotal(); print(" Cash Returned = $ "+Math.abs(r_cash));

5. Abstract Factory Pattern – Focused view of only the Abstract & Concrete Factory elements expanded.


<<Java Class>>
GasPump1
gasPump.classes.mda_efsm_classes
<<Java Class>>
GasPump2
gasPump.classes.mda_efsm_classes
<<Java Class>>
MDA_EFSM
gasPump.classes.mda_efsm_classes
~m
0..1
~m 0..1
~gs
0..1
<<Java Class>>
GasState
gasPump.classes.stateClasses
<<Java Class>>
Start
gasPump.classes.stateClasses
<<Java Class>>
S0
gasPump.classes.stateClasses
<<Java Class>>
S1
gasPump.classes.stateClasses
<<Java Class>>
S2
gasPump.classes.stateClasses
<<Java Class>>
S3
gasPump.classes.stateClasses
<<Java Class>>
S4
gasPump.classes.stateClasses
<<Java Class>>
S5
gasPump.classes.stateClasses
<<Java Class>>
S6
gasPump.classes.stateClasses
<<Java Class>>
OutputProcessor
gasPump.classes.mda_efsm_classes
OutputProcessor(AbstractFactory)
initialize(AbstractFactory):void
StoreData():void
PayMsg():void
StoreCash():void
DisplayMenu():void
RejectMsg():void
CancelMsg():void
SetPrice(int):void
SetInitialValues():void
ReadyMsg():void
PumpGasUnit():void
TotalPrice():void
GasPumpedMsg():void
StopMsg():void
PrintReceipt():void
ReturnCash():void
~ds
0..1
~ds 0..1
~ds 0..1
<<Java Class>>
DataStore
gasPump.classes.dataStore
<<Java Class>>
DataStore2_GP2
gasPump.classes.dataStore
<<Java Class>>
DataStore1_GP1
gasPump.classes.dataStore
~af
0..1
~af
0..1
~af
0..1
<<Java Class>>
AbstractFactory
gasPump.classes.abstractFactories
AbstractFactory()
getDataStore():DataStore
getStoreData():Abstract_StoreData
getPayMsg():Abstract_PayMsg
getStoreCash():Abstract_StoreCash
getDisplayMenu():Abstract_DisplayMenu
getRejectMsg():Abstract_RejectMsg
getCancelMsg():Abstract_CancelMsg
getSetPrice():Abstract_SetPrice
getSetInitialValues():Abstract_SetInitialValues
getReadyMsg():Abstract_ReadyMsg
getPumpGasUnit():Abstract_PumpGasUnit
getTotalPrice():Abstract_TotalPrice
getGasPumpedMsg():Abstract_GasPumpedMsg
getStopMsg():Abstract_StopMsg
getPrintReceipt():Abstract_PrintReceipt
getReturnCash():Abstract_ReturnCash
<<Java Class>>
ClientFactory2
gasPump.classes.abstractFactories
ClientFactory2()
getDataStore():DataStore
getStoreData():Abstract_StoreData
getPayMsg():Abstract_PayMsg
getStoreCash():Abstract_StoreCash
getDisplayMenu():Abstract_DisplayMenu
getCancelMsg():Abstract_CancelMsg
getSetPrice():Abstract_SetPrice
getSetInitialValues():Abstract_SetInitialValues
getReadyMsg():Abstract_ReadyMsg
getPumpGasUnit():Abstract_PumpGasUnit
getTotalPrice():Abstract_TotalPrice
getGasPumpedMsg():Abstract_GasPumpedMsg
getStopMsg():Abstract_StopMsg
getPrintReceipt():Abstract_PrintReceipt
getReturnCash():Abstract_ReturnCash
<<Java Class>>
ClientFactory1
gasPump.classes.abstractFactories
ClientFactory1()
getDataStore():DataStore
getStoreData():Abstract_StoreData
getPayMsg():Abstract_PayMsg
getDisplayMenu():Abstract_DisplayMenu
getRejectMsg():Abstract_RejectMsg
getCancelMsg():Abstract_CancelMsg
getSetPrice():Abstract_SetPrice
getSetInitialValues():Abstract_SetInitialValues
getReadyMsg():Abstract_ReadyMsg
getPumpGasUnit():Abstract_PumpGasUnit
getTotalPrice():Abstract_TotalPrice
getGasPumpedMsg():Abstract_GasPumpedMsg
getStopMsg():Abstract_StopMsg
getPrintReceipt():Abstract_PrintReceipt



<<Java Class>>
OutputProcessor
gasPump.classes.mda_efsm_classes
ds: DataStore
OutputProcessor(AbstractFactory)
initialize(AbstractFactory):void
StoreData():void
PayMsg():void
StoreCash():void
DisplayMenu():void
RejectMsg():void
CancelMsg():void
SetPrice(int):void
SetInitialValues():void
ReadyMsg():void
PumpGasUnit():void
TotalPrice():void
GasPumpedMsg():void
StopMsg():void
PrintReceipt():void
ReturnCash():void
~af 0..1
<<Java Class>>
AbstractFactory
gasPump.classes.abstractFactories
ds: DataStore
AbstractFactory()
getDataStore():DataStore
getStoreData():Abstract_StoreData
getPayMsg():Abstract_PayMsg
getStoreCash():Abstract_StoreCash
getDisplayMenu():Abstract_DisplayMenu
getRejectMsg():Abstract_RejectMsg
getCancelMsg():Abstract_CancelMsg
getSetPrice():Abstract_SetPrice
getSetInitialValues():Abstract_SetInitialValues
getReadyMsg():Abstract_ReadyMsg
getPumpGasUnit():Abstract_PumpGasUnit
getTotalPrice():Abstract_TotalPrice
getGasPumpedMsg():Abstract_GasPumpedMsg
getStopMsg():Abstract_StopMsg
getPrintReceipt():Abstract_PrintReceipt
getReturnCash():Abstract_ReturnCash
<<Java Class>>
ClientFactory2
gasPump.classes.abstractFactories
ClientFactory2()
getDataStore():DataStore
getStoreData():Abstract_StoreData
getPayMsg():Abstract_PayMsg
getStoreCash():Abstract_StoreCash
getDisplayMenu():Abstract_DisplayMenu
getCancelMsg():Abstract_CancelMsg
getSetPrice():Abstract_SetPrice
getSetInitialValues():Abstract_SetInitialValues
getReadyMsg():Abstract_ReadyMsg
getPumpGasUnit():Abstract_PumpGasUnit
getTotalPrice():Abstract_TotalPrice
getGasPumpedMsg():Abstract_GasPumpedMsg
getStopMsg():Abstract_StopMsg
getPrintReceipt():Abstract_PrintReceipt
getReturnCash():Abstract_ReturnCash
<<Java Class>>
ClientFactory1
gasPump.classes.abstractFactories
ClientFactory1()
getDataStore():DataStore
getStoreData():Abstract_StoreData
getPayMsg():Abstract_PayMsg
getDisplayMenu():Abstract_DisplayMenu
getRejectMsg():Abstract_RejectMsg
getCancelMsg():Abstract_CancelMsg
getSetPrice():Abstract_SetPrice
getSetInitialValues():Abstract_SetInitialValues
getReadyMsg():Abstract_ReadyMsg
getPumpGasUnit():Abstract_PumpGasUnit
getTotalPrice():Abstract_TotalPrice
getGasPumpedMsg():Abstract_GasPumpedMsg
getStopMsg():Abstract_StopMsg
getPrintReceipt():Abstract_PrintReceipt
~a1 0..1
<<Java Class>>
Abstract_StoreData
gasPump.classes.strategyClasses
Abstract_StoreData()
StoreData(DataStore):void
<<Java Class>>
StoreData1
gasPump.classes.strategyClasses
StoreData1()
StoreData(DataStore):void
<<Java Class>>
StoreData2
gasPump.classes.strategyClasses
StoreData2()
StoreData(DataStore):void
~a2 0..1
<<Java Class>>
Abstract_PayMsg
gasPump.classes.strategyClasses
Abstract_PayMsg()
PayMsg():void
<<Java Class>>
PayMsg1
gasPump.classes.strategyClasses
PayMsg1()
PayMsg():void
<<Java Class>>
PayMsg2
gasPump.classes.strategyClasses
PayMsg2()
PayMsg():void
~a3 0..1
<<Java Class>>
Abstract_StoreCash
gasPump.classes.strategyClasses
Abstract_StoreCash()
StoreCash(DataStore):void
<<Java Class>>
StoreCash1
gasPump.classes.strategyClasses
StoreCash1()
StoreCash(DataStore):void
~a4 0..1
<<Java Class>>
Abstract_DisplayMenu
gasPump.classes.strategyClasses
Abstract_DisplayMenu()
DisplayMenu():void
<<Java Class>>
DisplayMenuGP2
gasPump.classes.strategyClasses
DisplayMenuGP2()
DisplayMenu():void
<<Java Class>>
DisplayMenuGP1
gasPump.classes.strategyClasses
DisplayMenuGP1()
DisplayMenu():void
~a5 0..1
<<Java Class>>
Abstract_RejectMsg
gasPump.classes.strategyClasses
Abstract_RejectMsg()
RejectMsg():void
<<Java Class>>
RejectMsg1
gasPump.classes.strategyClasses
RejectMsg1()
RejectMsg():void
~a6 0..1
<<Java Class>>
Abstract_CancelMsg
gasPump.classes.strategyClasses
Abstract_CancelMsg()
CancelMsg():void
<<Java Class>>
CancelMsg1
gasPump.classes.strategyClasses
CancelMsg1()
CancelMsg():void
~a7 0..1
<<Java Class>>
Abstract_SetPrice
gasPump.classes.strategyClasses
Abstract_SetPrice()
SetPrice(int,DataStore):void
<<Java Class>>
SetPrice1
gasPump.classes.strategyClasses
SetPrice1()
SetPrice(int,DataStore):void
<<Java Class>>
SetPrice2
gasPump.classes.strategyClasses
SetPrice2()
SetPrice(int,DataStore):void
~a8 0..1
<<Java Class>>
Abstract_SetInitialValues
gasPump.classes.strategyClasses
Abstract_SetInitialValues()
SetInitialValues(DataStore):void
<<Java Class>>
SetInitialValues2
gasPump.classes.strategyClasses
SetInitialValues2()
SetInitialValues(DataStore):void
<<Java Class>>
SetInitialValues1
gasPump.classes.strategyClasses
SetInitialValues1()
SetInitialValues(DataStore):void
~a9 0..1
<<Java Class>>
Abstract_ReadyMsg
gasPump.classes.strategyClasses
Abstract_ReadyMsg()
ReadyMsg():void
<<Java Class>>
ReadyMsg1
gasPump.classes.strategyClasses
ReadyMsg1()
ReadyMsg():void
~a10 0..1
<<Java Class>>
Abstract_PumpGasUnit
gasPump.classes.strategyClasses
Abstract_PumpGasUnit()
PumpGasUnit(DataStore):void
<<Java Class>>
PumpGasUnit2
gasPump.classes.strategyClasses
PumpGasUnit2()
PumpGasUnit(DataStore):void
<<Java Class>>
PumpGasUnit1
gasPump.classes.strategyClasses
PumpGasUnit1()
PumpGasUnit(DataStore):void
~a11 0..1
<<Java Class>>
Abstract_TotalPrice
gasPump.classes.strategyClasses
Abstract_TotalPrice()
TotalPrice(DataStore):void
<<Java Class>>
TotalPrice1
gasPump.classes.strategyClasses
TotalPrice1()
TotalPrice(DataStore):void
<<Java Class>>
TotalPrice2
gasPump.classes.strategyClasses
TotalPrice2()
TotalPrice(DataStore):void
~a12 0..1
<<Java Class>>
Abstract_GasPumpedMsg
gasPump.classes.strategyClasses
Abstract_GasPumpedMsg()
GasPumpedMsg(DataStore):void
<<Java Class>>
GasPumpedMsg1
gasPump.classes.strategyClasses
GasPumpedMsg1()
GasPumpedMsg(DataStore):void
<<Java Class>>
GasPumpedMsg2
gasPump.classes.strategyClasses
GasPumpedMsg2()
GasPumpedMsg(DataStore):void
~a13 0..1
<<Java Class>>
Abstract_StopMsg
gasPump.classes.strategyClasses
Abstract_StopMsg()
StopMsg():void
<<Java Class>>
StopMsg1
gasPump.classes.strategyClasses
StopMsg1()
StopMsg():void
~a14 0..1
<<Java Class>>
Abstract_PrintReceipt
gasPump.classes.strategyClasses
Abstract_PrintReceipt()
PrintReceipt(DataStore):void
<<Java Class>>
PrintReceipt1
gasPump.classes.strategyClasses
PrintReceipt1()
PrintReceipt(DataStore):void
<<Java Class>>
PrintReceipt2
gasPump.classes.strategyClasses
PrintReceipt2()
PrintReceipt(DataStore):void
~a15 0..1
<<Java Class>>
Abstract_ReturnCash
gasPump.classes.strategyClasses
Abstract_ReturnCash()
ReturnCash(DataStore):void
<<Java Class>>
ReturnCash1
gasPump.classes.strategyClasses
ReturnCash1()
ReturnCash(DataStore):void

Class: AF – Abstract Factory	
Purpose	This class represents the abstract class for the Factory and is used to group the various concrete factories together that are held within the implementation. Currently there are two factories allow for the creation of two different combinations of data stores and actions based on the clients (GasPump1 and GasPump2) needs.
Variables & Initialization	DataStore ds; // initialize Datastore object.
getDataStore()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getStoreData()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getPayMsg()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getStoreCash()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getDisplayMenu()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getRejectMsg()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getCancelMsg()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getSetPrice(int g)	No logic implemented returns null, overridden by the child class if logic is needed for the state
getSetInitialValues()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getReadyMsg()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getPumpGasUnit()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getTotalPrice()	No logic implemented returns null, overridden by the child class if logic is needed for the state
getGasPumpedMsg()	No logic implemented returns null, overridden by the child class if logic is needed for the state

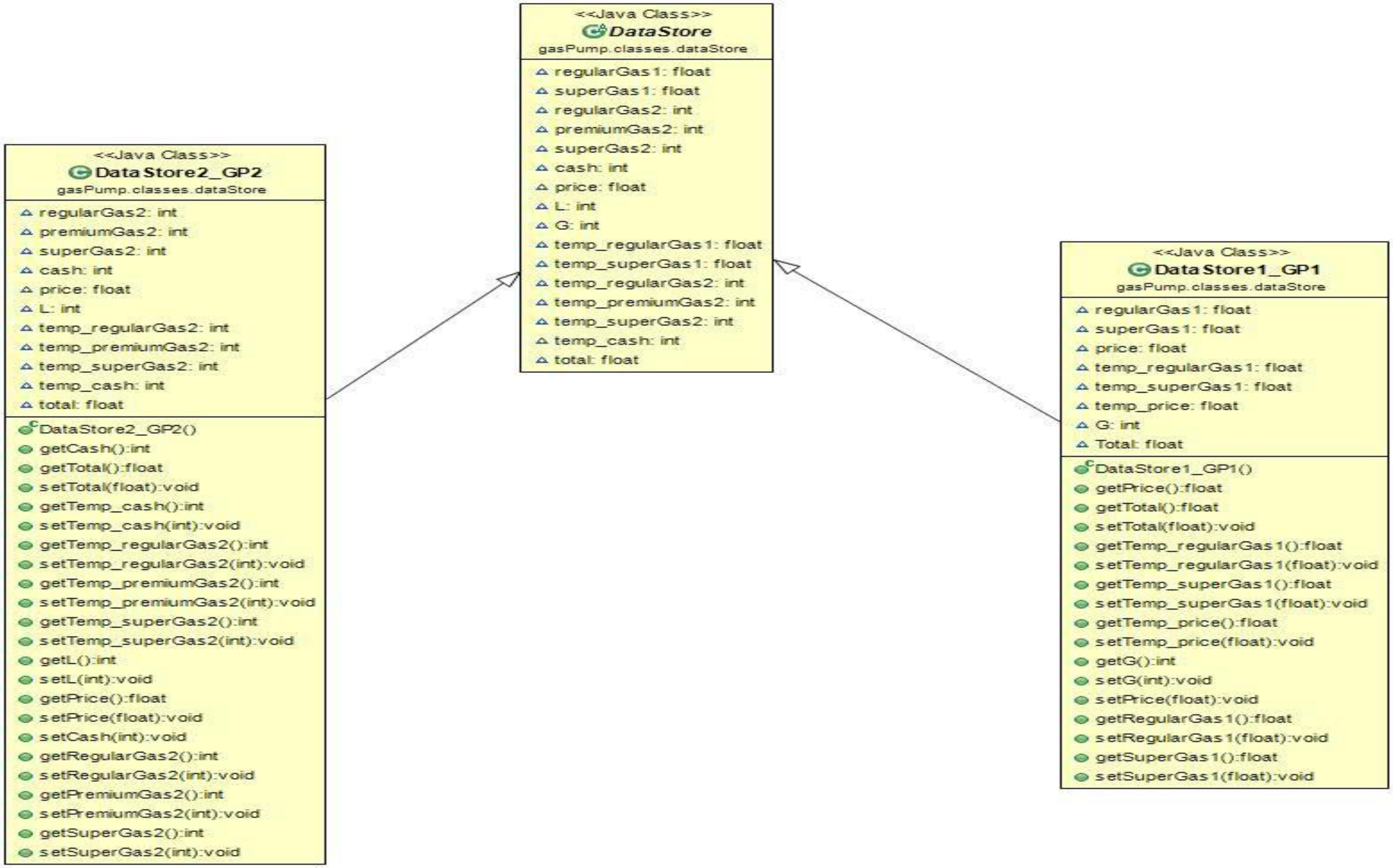
Class: ClientFactory1 (Concrete Factory for GasPump1)	
Purpose	This class represents the concrete class for GasPump1's factory and is used to handle the creation of class objects specific for GassPump1. One example is that this factory creates a specific data store associated with the factory.
Variables & Initialization	No Initialization required
ClientFactory1()	ds = new DataStore1_GP1(); // create DataStore1 object for ClientFactory1
getDataStore()	return the pointer to the data store (return ds)
getStoreData()	return new StoreData1()
getPayMsg()	return new PayMsg1()
getDisplayMenu()	return new DisplayMenuGP1()
getRejectMsg()	return new RejectMsg1()
getCancelMsg()	return new CancelMsg1()
getSetPrice(int g)	return new SetPrice1()
getSetInitialValues()	return new SetInitialValues1()
getReadyMsg()	return new ReadyMsg1()
getPumpGasUnit()	return new PumpGasUnit1()
getTotalPrice()	return new TotalPrice1()
getGasPumpedMsg()	return new GasPumpedMsg1()
getStopMsg()	return new StopMsg1()
getPrintReceipt()	return new PrintReceipt1()

Class: ClientFactory2 (Concrete Factory for GasPump2)	
Purpose	This class represents the concrete class for GasPump2's factory and is used to handle the creation of class objects specific for GassPump2. One example is that this factory creates a specific data store associated with the factory.
Variables & Initialization	No Initialization required
ClientFactory2()	ds = new DataStore2_GP2(); // create DataStore1 object for ClientFactory1
getDataStore()	return the pointer to the data store (return ds)
getStoreData()	return new StoreData2()
getPayMsg()	return new PayMsg2()
getDisplayMenu()	return new DisplayMenuGP2()
getStoreCash()	return new StoreCash1()
getCancelMsg()	return new CancelMsg1()
getSetPrice(int g)	return new SetPrice2()
getSetInitialValues()	return new SetInitialValues2()
getReadyMsg()	return new ReadyMsg1()
getPumpGasUnit()	return new PumpGasUnit2()
getTotalPrice()	return new TotalPrice2()
getGasPumpedMsg()	return new GasPumpedMsg2()

getStopMsg()	return new StopMsg1()
getPrintReceipt()	return new PrintReceipt2()
getReturnCash()	return new ReturnCash1()

6. Data Store – Only focused on data store to display expanded elements and details.

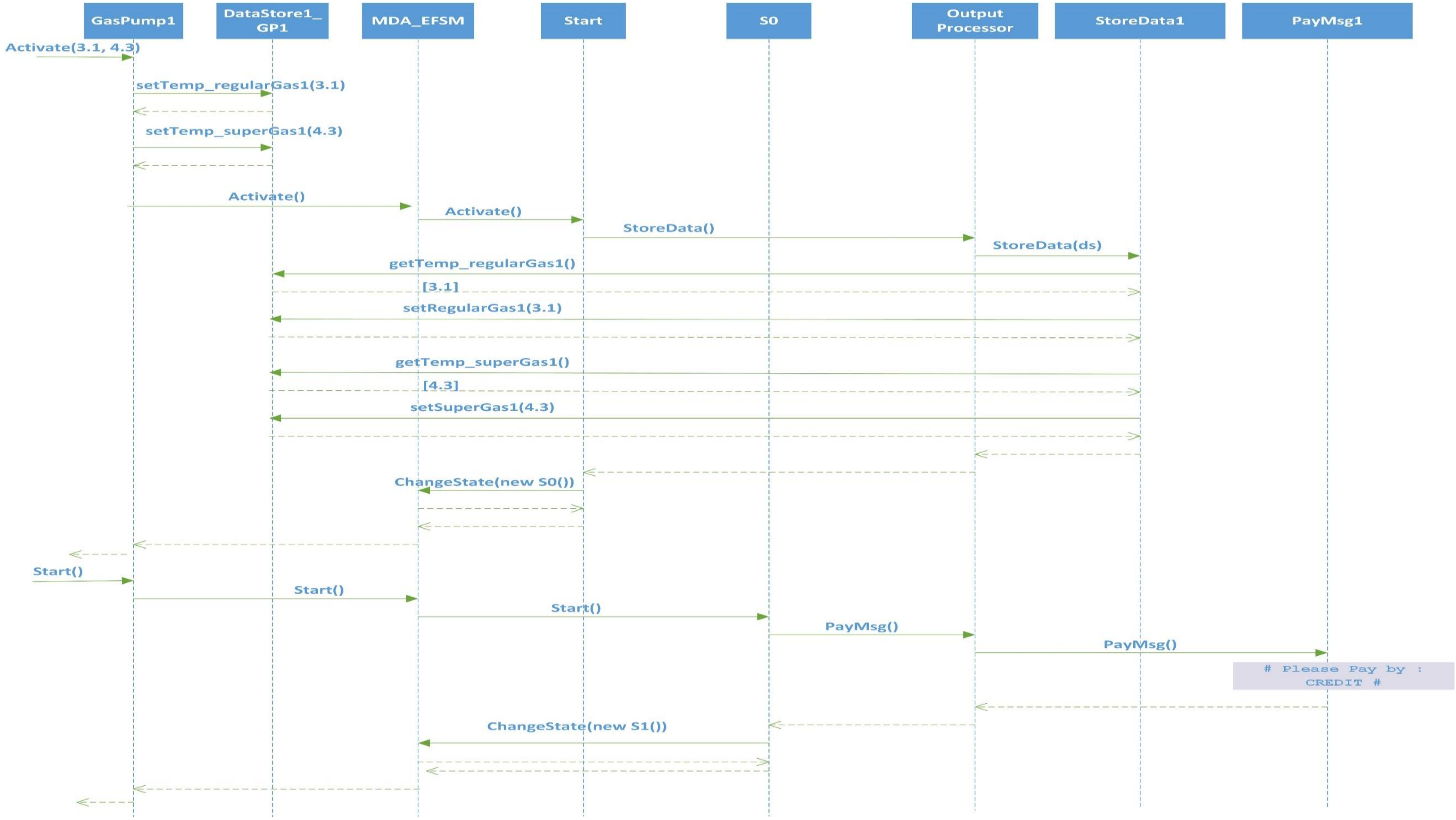
(Note: Overloaded functions in DataStore class are not shown. It is abstract Class and DataStore1_GP1 and DataStore2_GP inherits it).

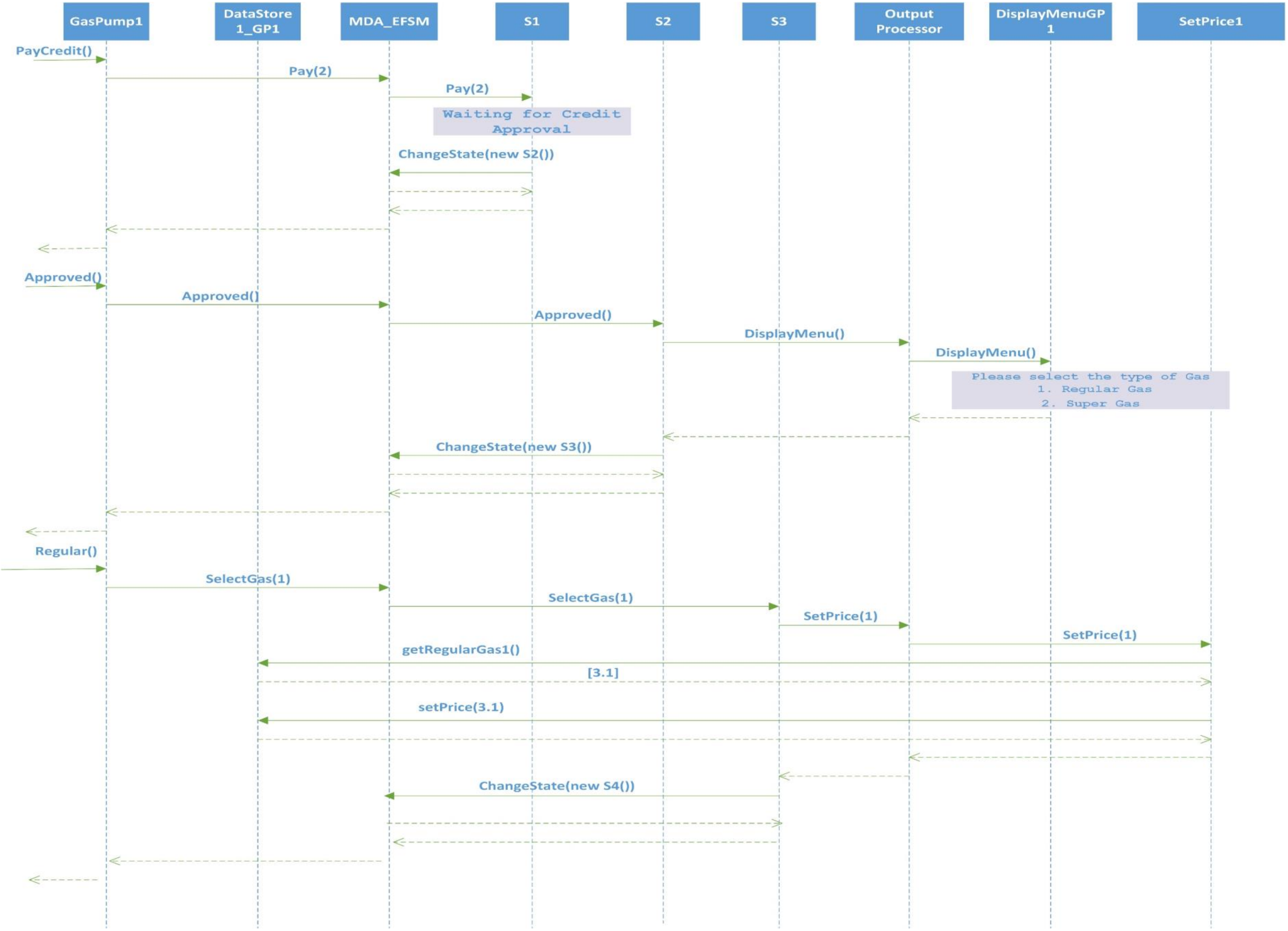


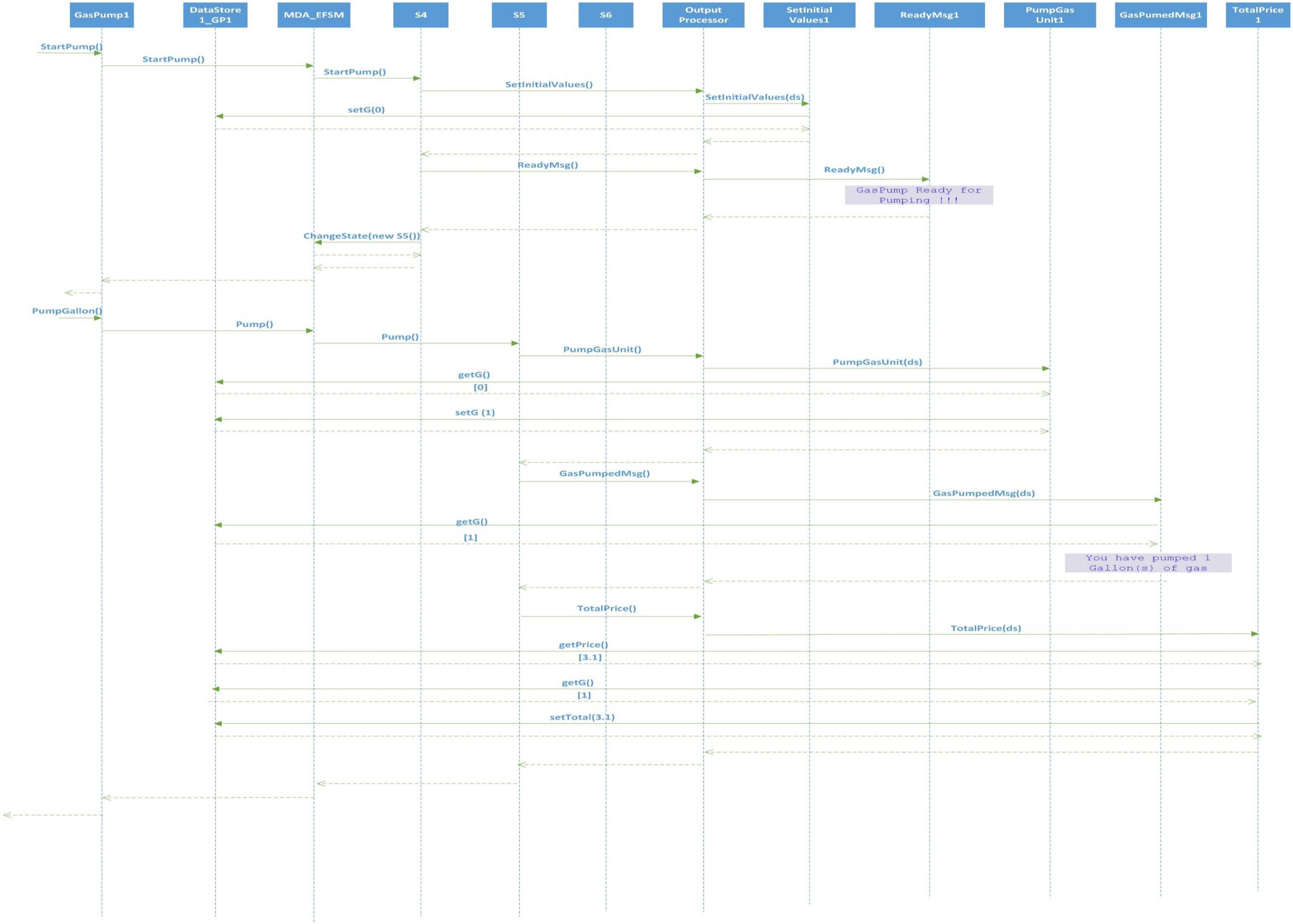
3. Dynamics

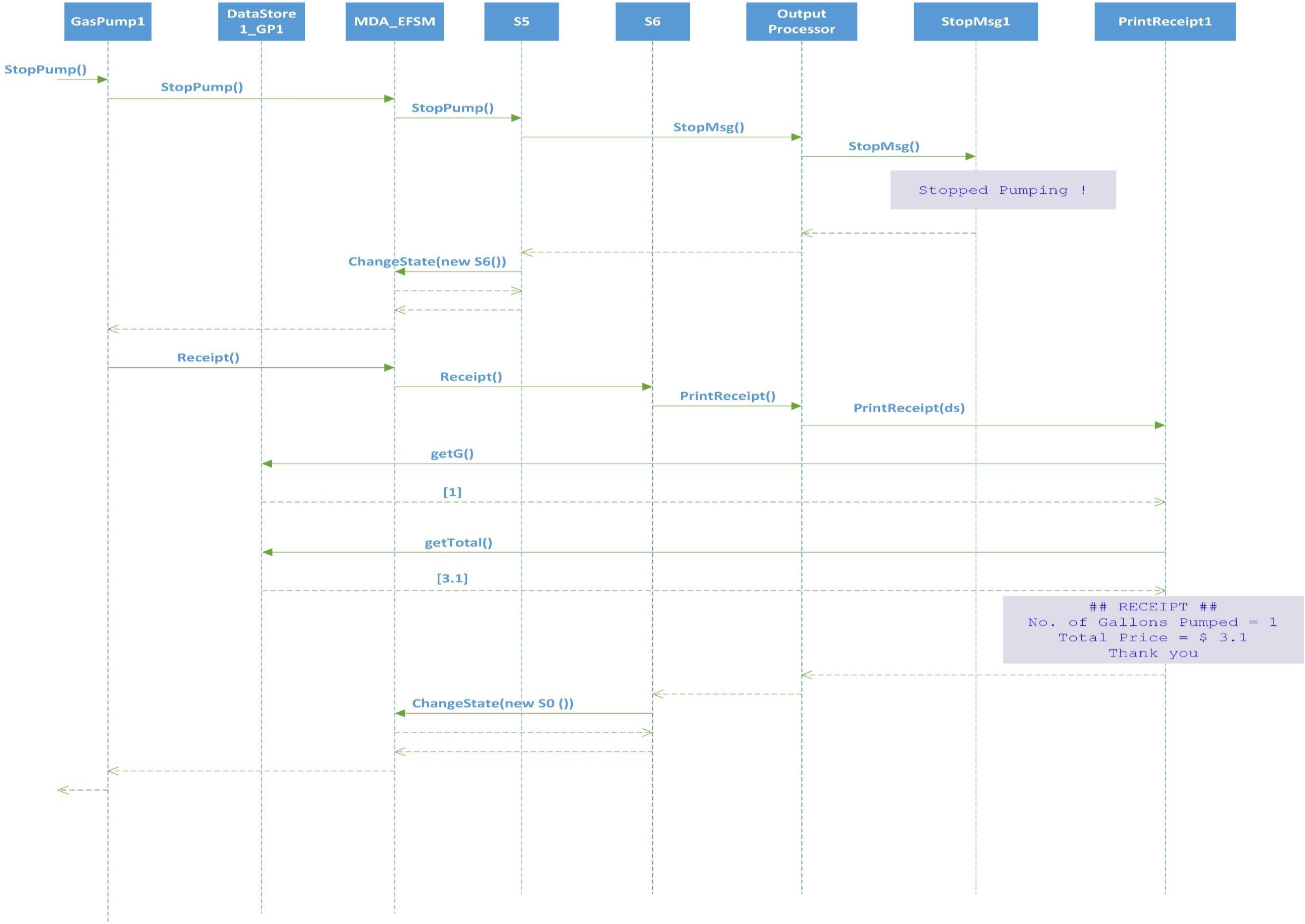
This section will outline sequence diagrams of two scenarios.

Scenario 1: GasPump-1 with the sequence of operations: Activate(3.1, 4.3), Start(), PayCredit(), Approved(), Regular(), StartPump(), PumpGallon(), StopPump()

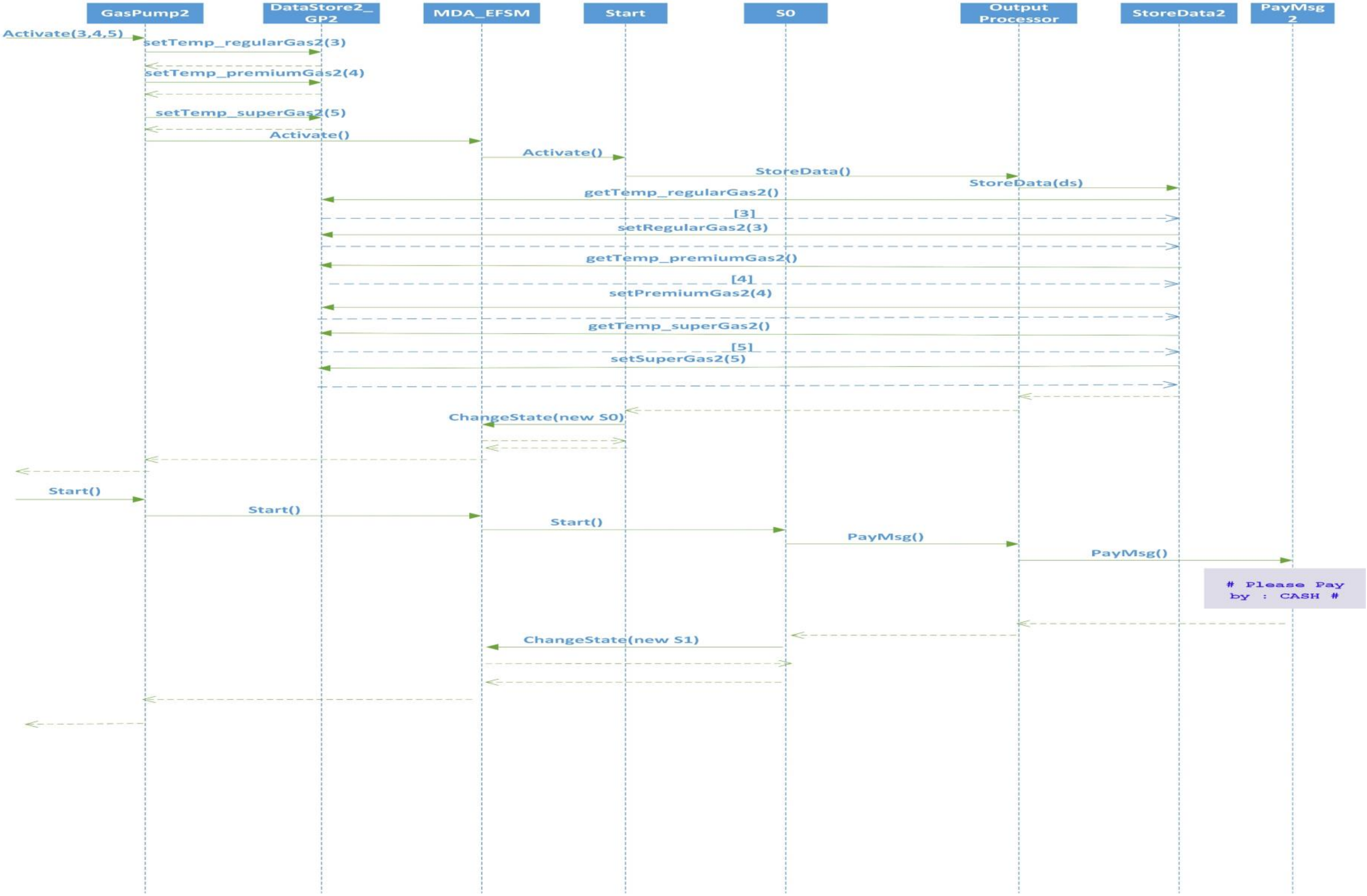


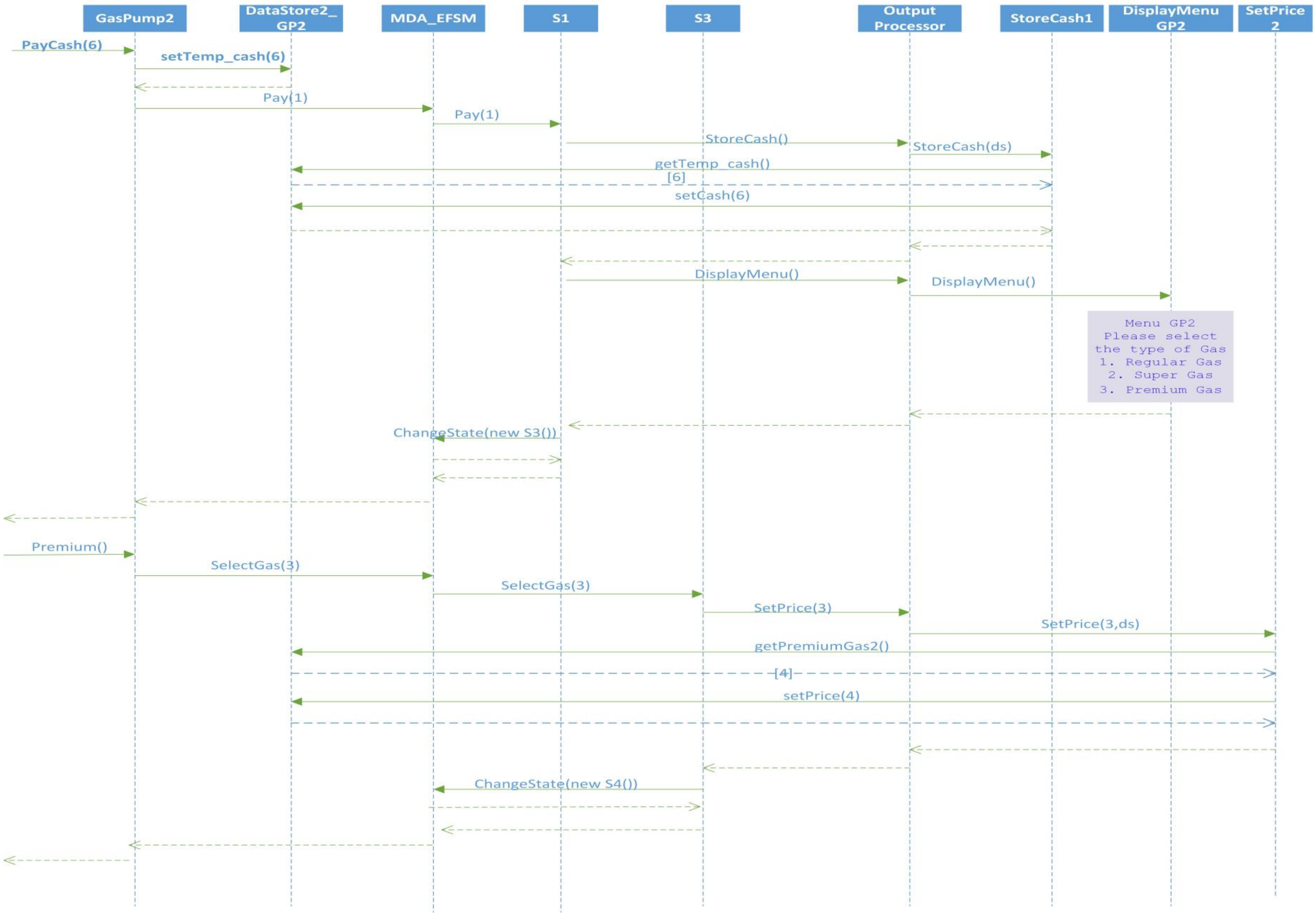

GasPump1
DataStore 1_GP1
MDA_EFSM
S1
S2
S3
Output Processor
DisplayMenuGP 1
SetPrice1
PayCredit()
Pay(2)
Pay(2)
Waiting for Credit Approval
ChangeState(new S2())
Approved()
Approved()
Approved()
DisplayMenu()
DisplayMenu()
Please select the type of Gas
1. Regular Gas
2. Super Gas
ChangeState(new S3())
Regular()
SelectGas(1)
SelectGas(1)
SetPrice(1)
SetPrice(1)
getRegularGas1()
[3.1]
setPrice(3.1)
ChangeState(new S4())

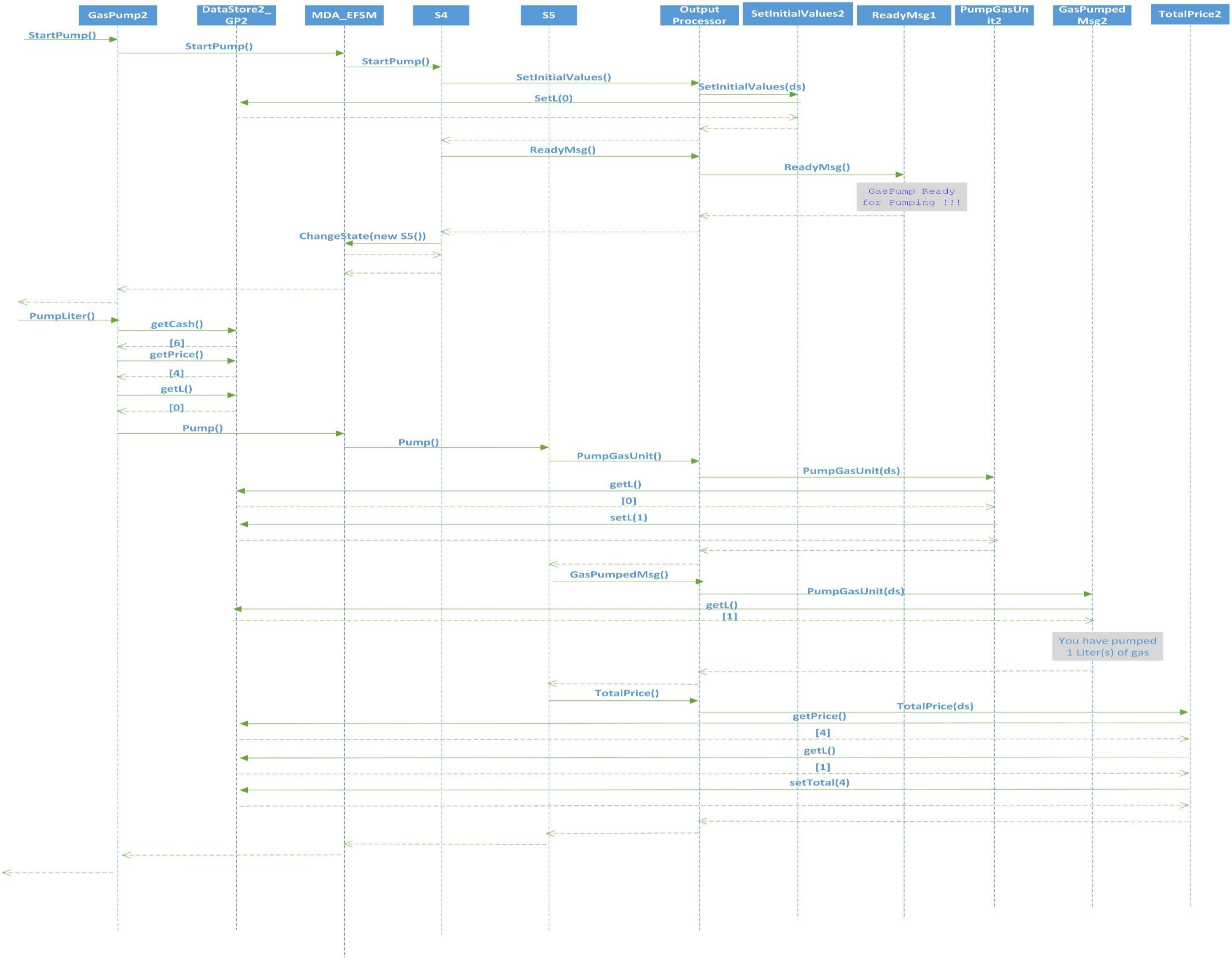

GasPump1
DataStore 1_GP1
MDA_EFSM
S4
S5
S6
Output Processor
SetInitial Values1
ReadyMsg1
PumpGas Unit1
GasPumedMsg1
TotalPrice 1
StartPump()
StartPump()
StartPump()
SetInitialValues()
SetInitialValues(ds)
setG(0)
ReadyMsg()
ReadyMsg()
GasPump Ready for Pumping !!!
ChangeState(new S5())
PumpGallon()
Pump()
Pump()
PumpGasUnit()
PumpGasUnit(ds)
getG()
[0]
setG (1)
GasPumpedMsg()
GasPumpedMsg(ds)
getG()
[1]
You have pumped 1 Gallon(s) of gas
TotalPrice()
TotalPrice(ds)
getPrice()
[3.1]
getG()
[1]
setTotal(3.1)

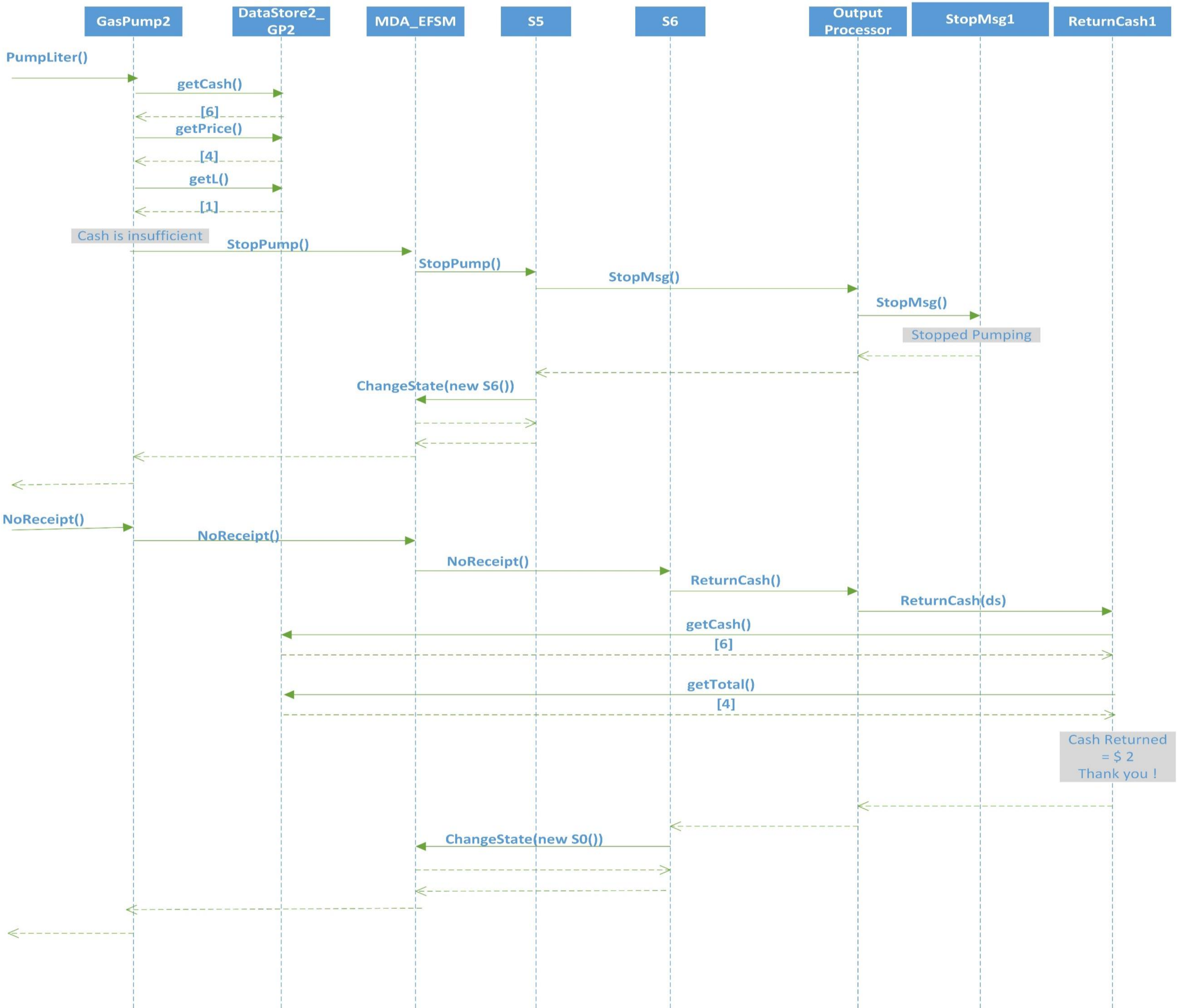

GasPump1
DataStore 1_GP1
MDA_EFSM
S5
S6
Output Processor
StopMsg1
PrintReceipt1
StopPump()
StopPump()
StopPump()
StopMsg()
StopMsg()
Stopped Pumping !
ChangeState(new S6())
Receipt()
Receipt()
PrintReceipt()
PrintReceipt(ds)
getG()
[1]
getTotal()
[3.1]
## RECEIPT ##
No. of Gallons Pumped = 1
Total Price = $ 3.1
Thank you
ChangeState(new S0 ())

Scenario 2: GasPump-1 with the sequence of operations: Activate(3.1, 4.3), Start(), PayCredit(), Approved(), Regular(), StartPump(), PumpGallon(), StopPump()




GasPump2
DataStore2_GP2
MDA_EFSM
S1
S3
Output Processor
StoreCash1
DisplayMenu GP2
SetPrice 2
PayCash(6)
setTemp_cash(6)
Pay(1)
Pay(1)
StoreCash()
StoreCash(ds)
getTemp_cash()
[6]
setCash(6)
DisplayMenu()
DisplayMenu()
Menu GP2
Please select
the type of Gas
1. Regular Gas
2. Super Gas
3. Premium Gas
ChangeState(new S3())
Premium()
SelectGas(3)
SelectGas(3)
SetPrice(3)
SetPrice(3,ds)
getPremiumGas2()
[4]
setPrice(4)
ChangeState(new S4())


GasPump2
DataStore2_GP2
MDA_EFSM
S4
S5
Output Processor
SetInitialValues2
ReadyMsg1
PumpGasUnit2
GasPumpedMsg2
TotalPrice2
StartPump()
StartPump()
StartPump()
SetInitialValues()
SetInitialValues(ds)
SetL(0)
ReadyMsg()
ReadyMsg()
GasPump Ready for Pumping !!!
ChangeState(new S5())
PumpLiter()
getCash()
[6]
getPrice()
[4]
getL()
[0]
Pump()
Pump()
PumpGasUnit()
PumpGasUnit(ds)
getL()
[0]
setL(1)
GasPumpedMsg()
PumpGasUnit(ds)
getL()
[1]
You have pumped 1 Liter(s) of gas
TotalPrice()
TotalPrice(ds)
getPrice()
[4]
getL()
[1]
setTotal(4)


GasPump2
DataStore2_ GP2
MDA_EFSM
S5
S6
Output Processor
StopMsg1
ReturnCash1
PumpLiter()
getCash()
[6]
getPrice()
[4]
getL()
[1]
Cash is insufficient
StopPump()
StopPump()
StopMsg()
StopMsg()
Stopped Pumping
ChangeState(new S6())
NoReceipt()
NoReceipt()
NoReceipt()
ReturnCash()
ReturnCash(ds)
getCash()
[6]
getTotal()
[4]
Cash Returned = $ 2 Thank you !
ChangeState(new S0())

4. Pattern Description

This section will describe which class belongs to which pattern (State Pattern, Strategy Pattern, and Abstract Factory Pattern).

State Pattern (10 Classes) – see section 3 for full description

- State Machine Context Class – MDA_EFSM
- State Abstract Class – GasState
- State Concrete Classes – Start, S0, S1, S2, S3, S4, S5, S6

Strategy Pattern (40 Classes) – see section 3 for full description

- Strategy Abstract Class – Abstract_CancelMsg, Abstract_DisplayMenu, Abstract_GasPumpedMsg, Abstract_PayMsg, Abstract_PrintReceipt, Abstract_PumpGasUnit, Abstract_ReadyMsg, Abstract_RejectMsg, Abstract_ReturnCash, Abstract_SetInitialValues, Abstract_SetPrice, Abstract_StopMsg, Abstract_StoreCash, Abstract_StoreData, Abstract_TotalPrice.
- Strategy Concrete Classes – CancelMsg1, DisplayMenuGP1, DisplayMenuGP2, GasPumpedMsg1, GasPumpedMsg2, PayMsg1, PayMsg2, PrintReceipt1, PrintReceipt2, PumpGasUnit1, PumpGasUnit2, ReadyMsg1 RejectMsg1, ReturnCash1, SetInitialValues1, SetInitialValues2, SetPrice1, SetPrice2, StopMsg1, StoreCash1, StoreData1, StoreData2, TotalPrice1, TotalPrice2
- Client Class – OutputProcessor (OP)

Abstract Factory Pattern (4 Classes excluding (2) clients and (21) objects they create) – see section 3 for full description

- Abstract Factory Class – AbstractFactory
- Concrete Factory Classes – ClientFactory1, ClientFactory2
- Client Classes – GasPump1, GasPump2, OP

Other Classes within Implementation (9 Classes excluding driver)

- Abstract Data Store Class – DataStore
- Concrete Data Store Classes – DataStore1_GP1, DataStore2_GP2
- Platform Depended Input Processor Classes – GasPump1, GasPump2
- Meta Model (PIM) Class – MDA_EFSM
- Driver - GasPumpDriver