CS 586: Software System Architecture

GasPump Project - Spring 2017

Name: Rohan Suresh Jain

CWID: A20378201

rjain22@hawk.iit.edu

Index

1. MDA – EFSM Model for GasPump:

This sections contains list of meta events, meta actions for MDA-EFSM along with their description, state diagram of MDA-EFSM and Pseudo code for all operations of Input processors of GasPump-1 and GasPump-2.

MDA – EFSM Events

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

MDA – EFSM Actions

PayMsg //display different methods of payment
StoreData // stores price for the gas from the temporary data store
DisplayMenu // display a menu with a list of selections
RejectMsg // displays credit card not approved/reject message
CancelMsg // display cancellation message/request
StoreCash // stores cash from the temporary data store
SetPrice(int p) // set the price for the gas identified by g identifier
ReadyMsg // displays the ready pumping message
SetInitialValues // set G =0, total = 0
Total //pump unit of gas and count number of unit disposed
Display //display amount of gas disposed
StopMsg // stop pump message
Return cash //return change/cash if any available
PrintReceipt // print a receipt

Note:
cash: contain the value of cash deposited
price: contains the price of selected gas
L: contains the number of litres already pumped
W: cash/credit flag
cash, L, price are in the data store
m: is a pointer to the MDA-EFSM object
d,d1: are pointer to the Data store object
when g=1 ; it is regular gas
when g=2 ; it is super gas
when g=3 ; it is premium gas

Pseudo Code:
Operations of Input Processor for Gas Pump 1:

Activate(Float a, Float b)
{
if((a>0) &&(b>0)) then
d->temp_a=a
d->temp_b=b
m->activate()
}

Start()
{
m->start()
}

PayCredit()
{
m->PayCredit()
}

Reject()
{
m->Reject()
}

Cancel()
{
m->Cancel()
}

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

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

StartPump()
{
m->StartPump()
}

PumpGallon()
{
m->Pump()
}

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

Operations of Input Processor for Gas Pump 2:

Activate(int a, int b, int c)
{
if((a>0) &&(b>0) && (c>0)) then
d1->temp_a=a
d1->temp_b=b
d1->temp_c=c
m->activate()
}

Start()
{
m->start()
}

PayCash(int c)
{
if(c>0)
{
d1->temp_c=c
m->PayCash()
}
}
Approved()
{
m->Approved()
}

Cancel()
{
m->Cancel()
}

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

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

StartPump()
{
m->StartPump()
}

PumpLitre()
{
if (d1->cash<(d1->G+1)*d->price)
m->StopPump()
else
m->Pump()
}

Stop ()
{
m->StopPump()
}

Receipt()
{
m->Receipt()
}

NoReceipt()
{
m->NoReceipt()
}

State Diagram of MDA-EFSM

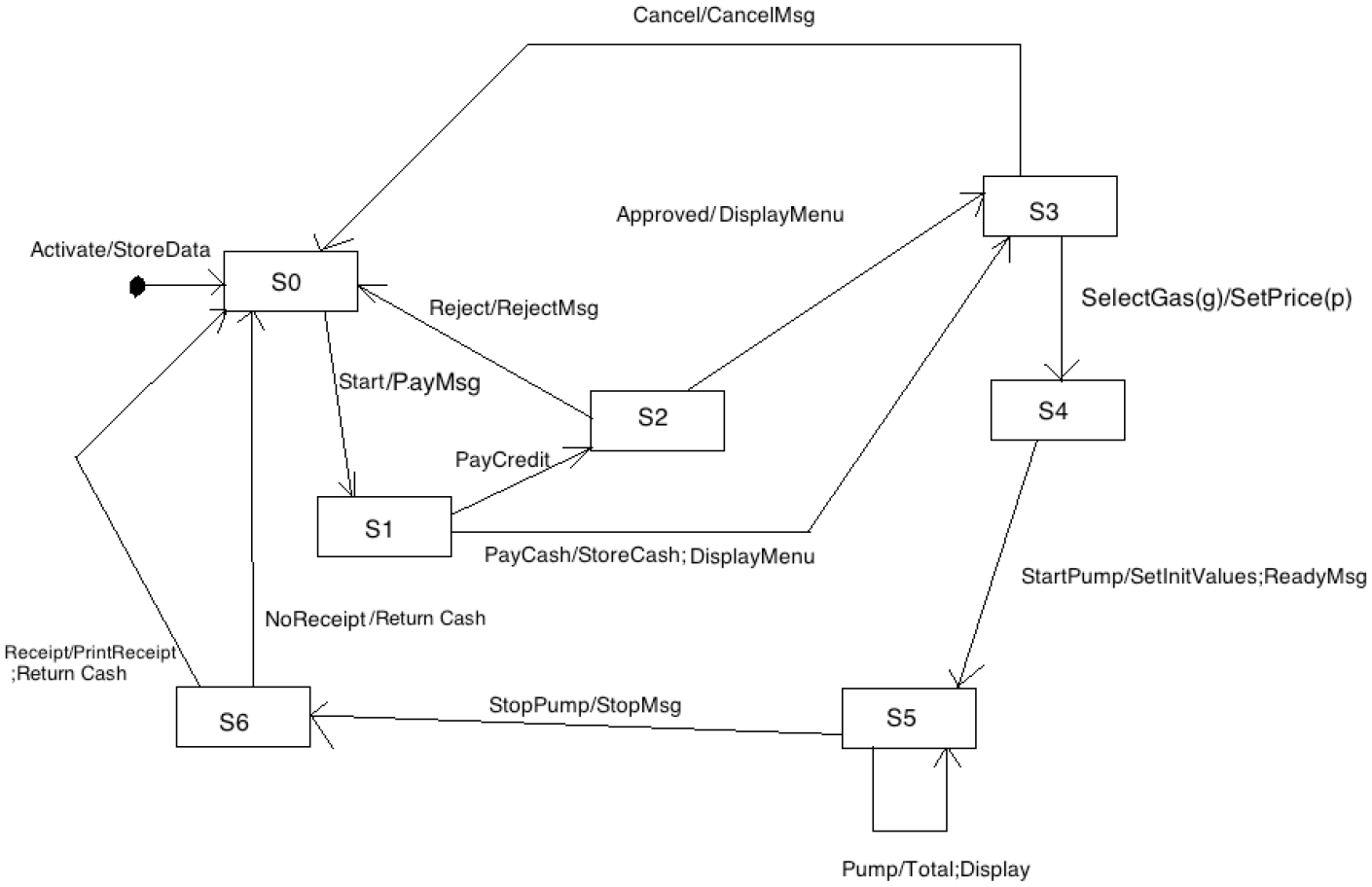

Fig 1.1 State Transition Diagram

2. Class Diagram of the MDA and GasPump components.

This section contains class diagram and responsibilities of each class and responsibilities of each operation supported by each class. Since class diagram is big in size due to its complexity, it is broken down into following parts –

1. **General Overview of Class diagram with Input Processor, Data Store and Output Processor.**
2. **GasPumps & MDA.**
3. **State Patterns.**
4. **Strategy Patterns.**
5. **Abstract Patterns.**
6. **DataStore of both GasPumps.**

2.1 General Overview of Class Diagram which includes details of Input Processor, Output Processor and Data Store.

General Overview of Class Diagram is broken down into two parts.

This is part 1 of Overview of Class Diagram which contains both GasPump-1 and GasPump-2, MDA, States & Output and links going to the strategy patterns from both Context Factory Patterns, Data Stores and Output.

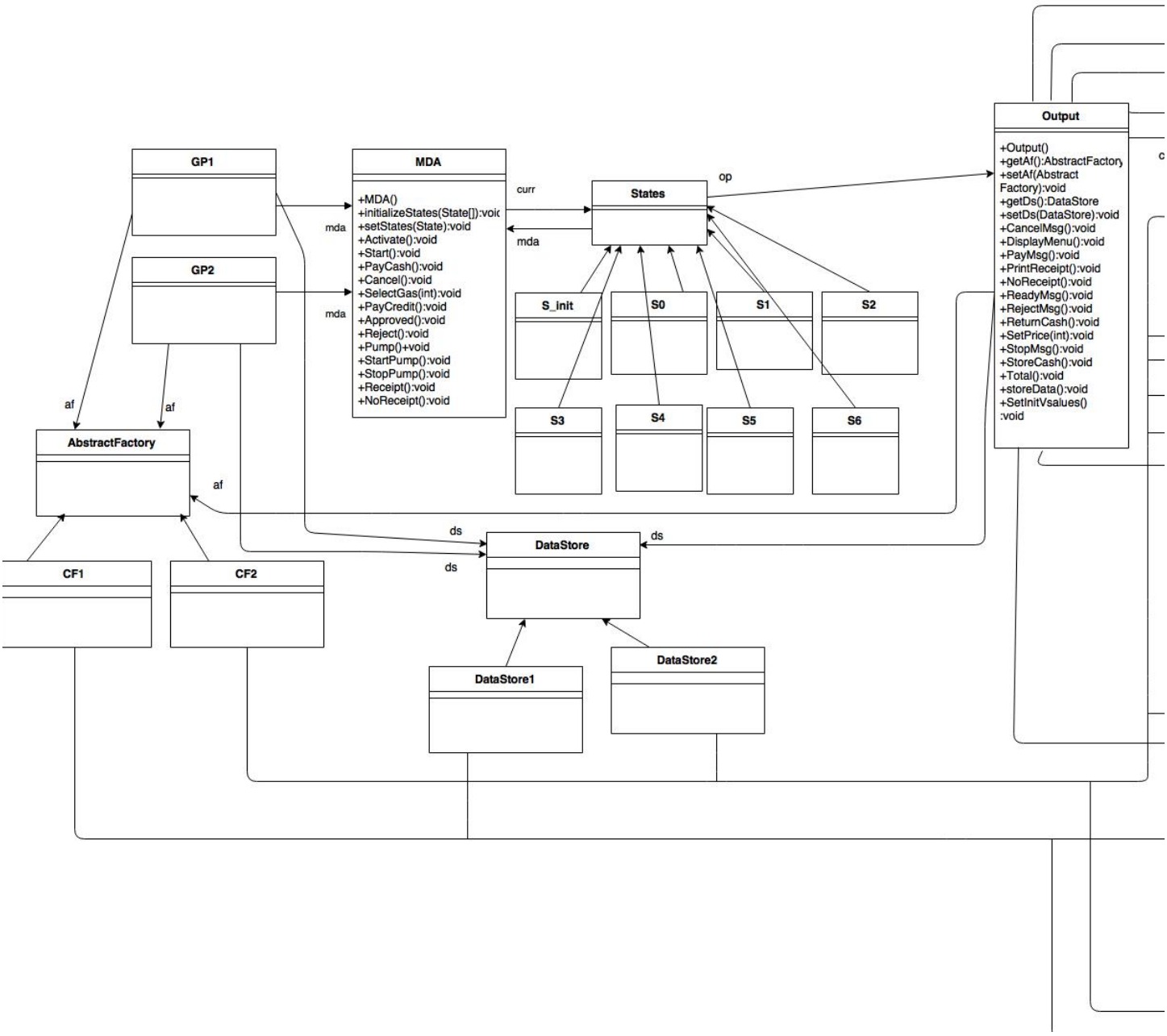

Fig 2.1.1 Part - 1 of Overview of Class Diagram

This is part 2 of Overview of Class Diagram which displays Output, Strategy Patterns and links which are coming from Context Factory 1 Pattern, Context Factory Pattern 2 and DataStore1 & DataStore2 towards the strategy patterns.

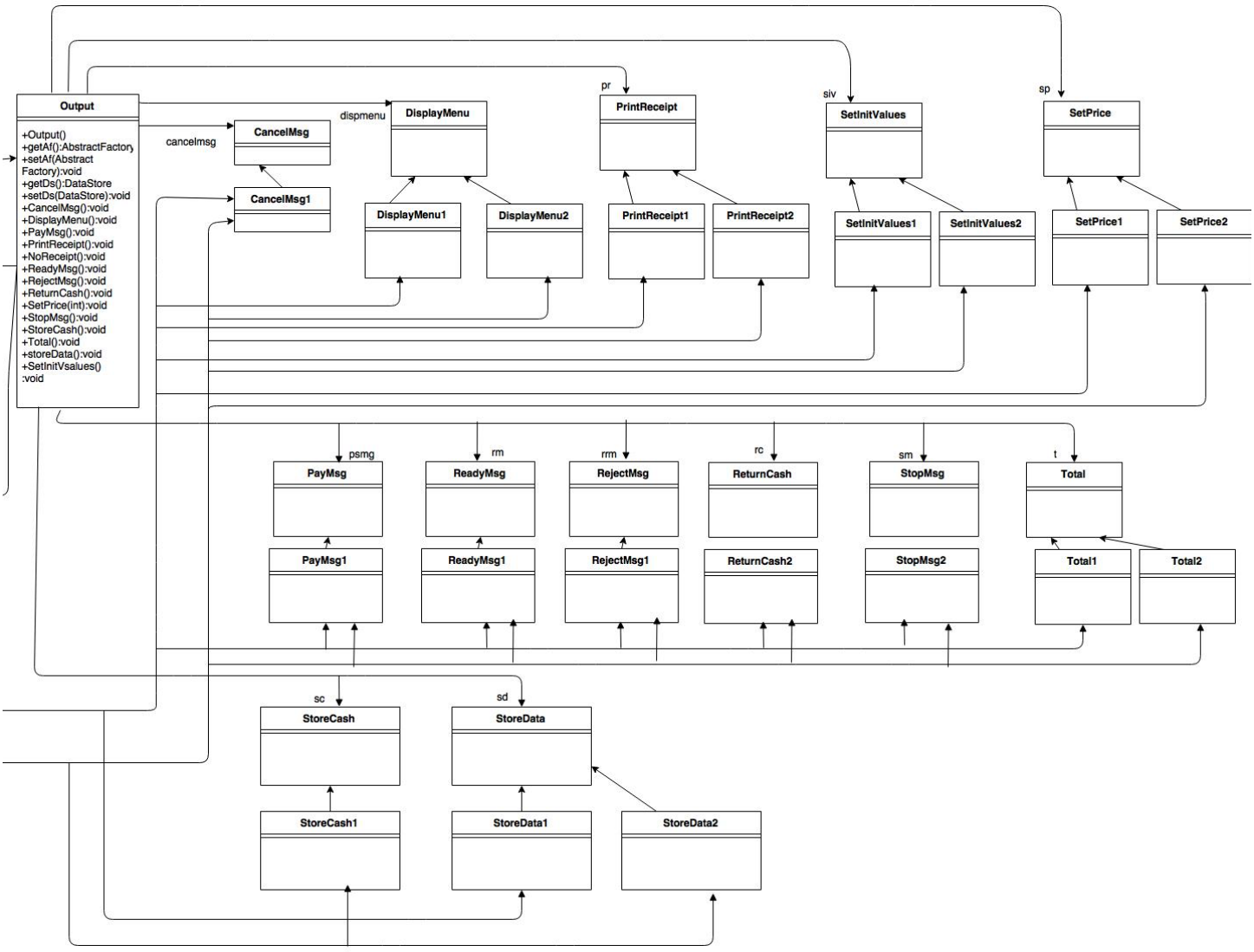

Fig 2.1.2 Part- 2 of Overview of Class Diagram

2.2 GasPumps & MDA-EFSM.

This section contains the relationship between GasPump1, GasPump2 and MDA-EFSM.

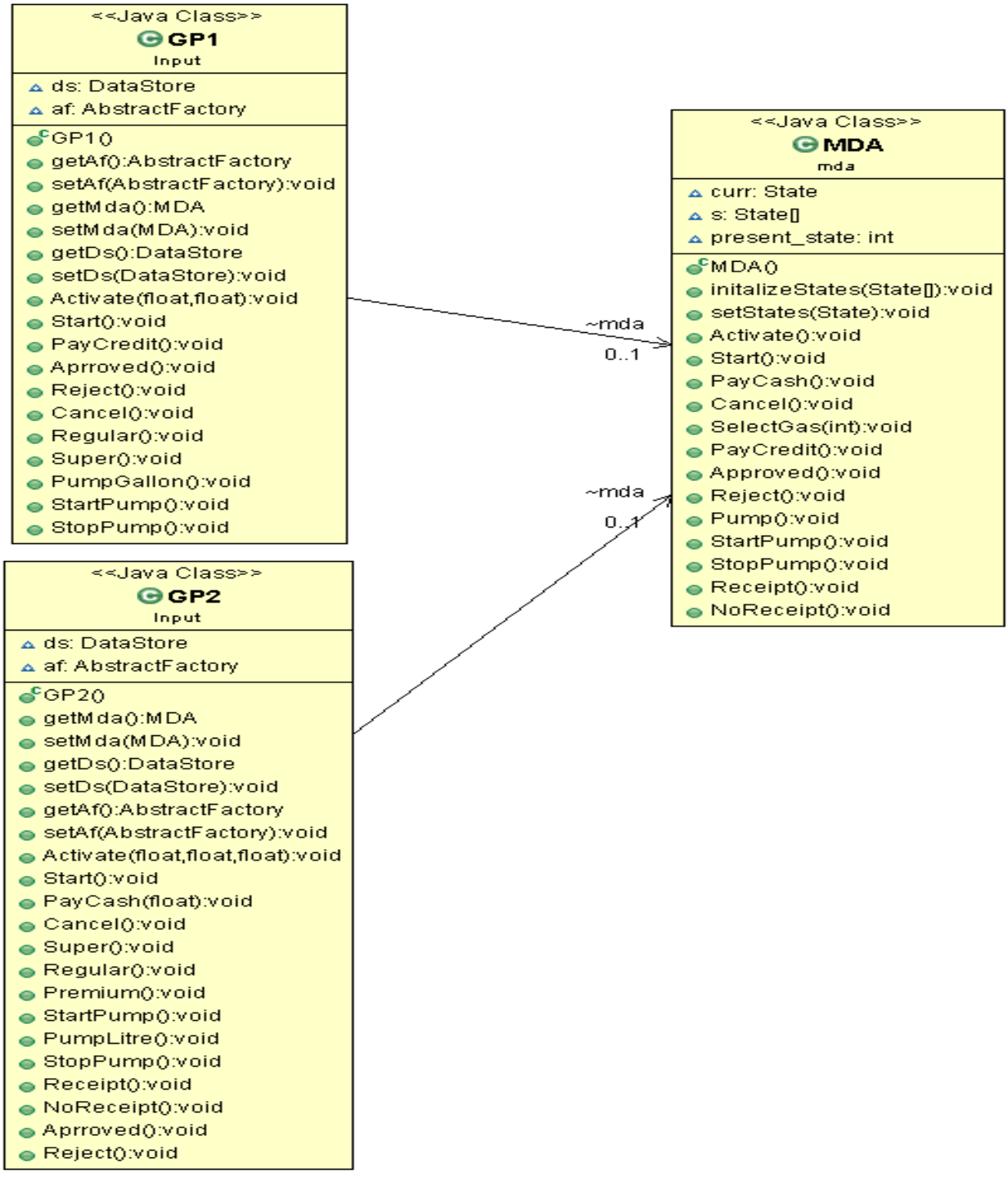

Fig 2.2 GasPump and MDA-EFSM Diagram

Responsibilities of GP1, GP2 and MDA class along with its operation details.

Class GP1:

Purpose	This is GasPump-1 main class which plays role of Input Processor in MDA architecture
Member Variables	ds: pointer to DataStore class af: pointer to AbstractFactory class.
Operations	Events in GP1-EFSM and pseudo code of each operation is explained in MDA class.

Class GP2:

Purpose	This is GasPump-2 main class which plays role of Input Processor in MDA architecture
Member Variables	ds: pointer to DataStore class af: pointer to AbstractFactory class.
Operations	Events in GP2-EFSM and pseudo code of each operation is explained in MDA class.

Class MDA:

Purpose	It contains functionalities which are common to both GasPump-1 and GasPump-2
Member Variables	curr: pointer to State class s: array of State class present_state: integer variable which store current/present state.
Operations	**MDA():** main Mda class which contains all the operations to be implemented. **initializeState():** this function is used to initialize all 8 states. **setStates():** this function is used to set the current state. **Activate():** this function activates the GasPump which calls(curr->Activate()). **Start():** this function starts the GasPump which calls(curr->Start()). **PayCash():** this function accepts cash from the user which calls(curr->PayCash()). **Cancel():** this function cancels the operation which calls(curr->Cancel()). **SelectGas(int g):** this function let user to select the gas type which calls (curr->SelectGas()). **PayCredit():** this function selects the credit card payment option which calls(curr->PayCredit()). **Approved():** this function approved the credit card payment which calls(curr->Approved()). **Reject():** this function rejects the payment method which calls(curr->Reject()). **StartPump():** this function starts the pump which calls(curr->StartPump()). **StopPump():** this function stops the pump which calls(curr->StopPump()). **Receipt():**this function prints the receipt which calls(curr->Receipt()). **NoReceipt():** this function doesn't print the receipt which calls(curr->NoReceipt()).

2.3 State Pattern: It shows relationship between MDA and state classes and provides detailed descriptions of each state class.

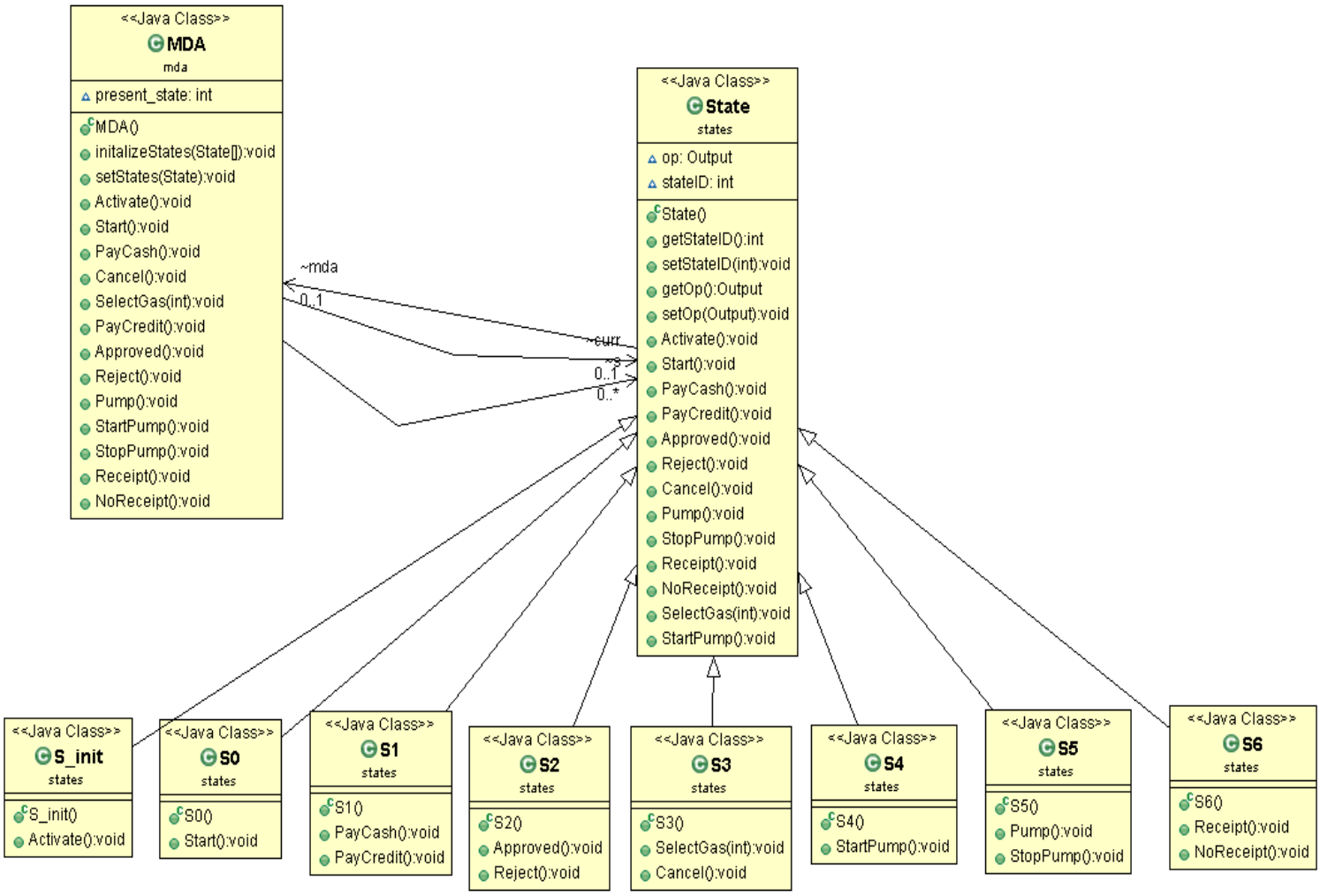

Fig 2.3 State Pattern Diagram

Responsibilities of all state classes along with its operation details.

Class State:

Purpose	This is main State class which plays role of context class, keeps track of current state in EFSM and forward calls to correct state.
Member Variables	op: pointer to Output class. stateID: integer variable to hold State ID.
Operations	**State():** main State class which contains all the operations to be implemented **getStateID():** this function is used to get current state ID. **setStateID():** this function is used to set current state. **getOp():** this function returns output of current state. **setOp():** this function is used to set output of current state. Remaining Functions are just declared in the main State class which are used in their respective classes.

Class S_init:

Purpose	It is the initial state which activates the GasPump.
Member Variables	No variables are defined here.
Operations	**Activate():** this function displays "GasPump is activated" message and stores initial data which is entered by user in this class by calling storeData() function of Output class.

Class S0:

Purpose	It is the first state which starts the GasPump.
Member Variables	No variables are defined here.
Operations	**Start():** call the PayMsg function of Output class which displays "select payment method" message.

Class S1:

Purpose	It is the second state which starts the GasPump.
Member Variables	No variables are defined here.
Operations	**PayCash():** this function store the cash accepted by user by calling storeCash() function of output class and displays menu by calling DisplayMenu() of Output class. **PayCredit():** this function is used if credit card option is selected.

Class S2:

Purpose	It is the third state which either approves or reject the payment method.
Member Variables	No variables are defined here.
Operations	**Approved():** this function approves the selected payment method and display menu by calling DisplayMenu() function of Output Class. **Reject():** this function rejects the selected payment method and display rejected message by calling Reject() function of Output Class.

Class S3:

Purpose	It is the fourth state which either let user to select gas or cancel the operation.
Member Variables	No variables are defined here.
Operations	**SelectGas(int g):** this function let user to select different gas options and set price of selected gas by calling SetPrice(g) function of Output Class. **CancelMsg():** this function cancel the current operation by calling CancelMsg() of Output class.

Class S4:

Purpose	It is the fifth state which starts the pump and set initial values and display that GasPump is ready to pump gas.
Member Variables	No variables are defined here.
Operations	**StartPump():** this function set initial values of G/L and total by calling SetInitValues() function of Output class and display "GasPump is ready to pump message" by calling ReadyMsg() function of Output class.

Class S5:

Purpose	It is the sixth state which either pumps the gas or stops the pump.
Member Variables	No variables are defined here.
Operations	**Pump():** this function pumps the gas and update the value of G/L and total by calling Total() function of Output Class. **StopPump():** this function displays "Pump is stopped" message by calling StopMsg() function of Output class.

Class S6:

Purpose	It is the seventh state which prints the receipt or it doesn't print the receipt.
Member Variables	No variables are defined here.
Operations	**PrintReceipt():** this function print the receipt which contains the amount of gas pumped and amount user have to pay for pumped gas by calling PrintReceipt() function of Output Class. **NoReceipt():** this function returns remaining amount of cash left after paying for pumped gas by calling ReturnCash() function of Output Class.

2.4 Strategy Pattern: this section represents relationship between Output class and Strategy Pattern classes.

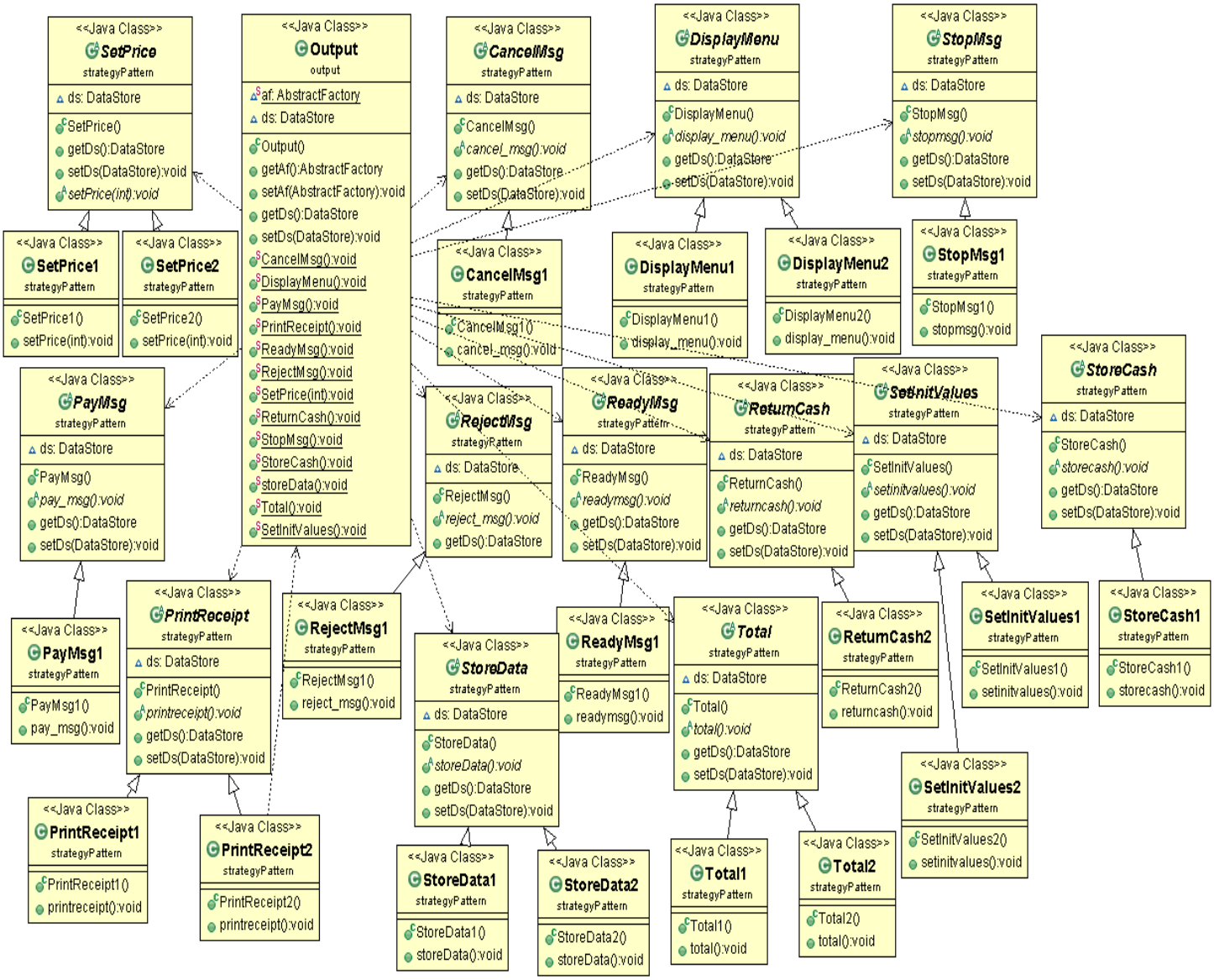

Fig 2.4 Strategy Pattern Diagram

Responsibilities of all Strategy Pattern classes & Output class along with its operation details.

Class Output:

Purpose	Plays the role of output processor in MDA architecture that issues action of strategy base class.
Member Variables	af: pointer to Abstract Factory class ds: pointer to DataStore class
Operations	**getAf():** function to return concrete factory based on GasPump1 or GasPump2. **setAf():** function to set Abstract Factory bases on GasPump1 or GasPump2 **getDs():** function to return the DataStore **setDs():** function to set the DataStore. **CancelMsg():** Call to factory to get action & set pointer(cm=af.getCancelMsg()) and call this action(cm.cancel_msg()) from strategy pattern. **DisplayMenu():** Call to factory to get action & set pointer(dm=af.getDisplayemenu()) and call this action(dm.display_menu()) from strategy pattern. **PayMsg():** Call to factory to get action & set pointer(pm=af.getPayMsg()) and call this action(pm.pay_msg()) from strategy pattern. **PrintReceipt():** Call to factory to get action & set pointer(pr=af.getPrintReceipt()) and call this action(pr.printreceipt ()) from strategy pattern. **ReadyMsg():** Call to factory to get action & set pointer(rm=af.getReadyMsg() and call this action(rm.readymsg()) from strategy pattern. **RejectMsg():** Call to factory to get action & set pointer(rrm=af.getRejectMsg() and call this action(rrm.reject_msg()) from strategy pattern. **SetPrice(int g):** Call to factory to get action & set pointer(sp=af.getSetPrice() and call this action(sp.setprice(g)) from strategy pattern. **ReturnCash():** Call to factory to get action & set pointer(rc=af.getReturnCash() and call this action(rc.returncash()) from strategy pattern. **StopMsg():** Call to factory to get action & set pointer(sm=af.getStopMsg() and call this action(sm.stopmsg()) from strategy pattern. **StoreCash():** Call to factory to get action & set pointer(sc=af.getStore() and call this action(sc.storecash()) from strategy pattern. **StoreData()**: Call to factory to get action & set pointer(sd=af.getStoreData() and call this action(sd.getStoreData()) from strategy pattern. **Total():** Call to factory to get action & set pointer(t=af.getTotal() and call this action(t.total()) from strategy pattern. **SetInitValues():** Call to factory to get action & set pointer(siv=af.getSetInitValue() and call this action(siv.setinitvalues()) from strategy pattern.

Class CancelMsg:

Purpose	Abstract class that group strategies of CancelMsg action
Member Variables	ds: pointer to DataStore.
Operations	**cancel_msg():** abstract method which will be override by subclass to display different cancel message depending on the GasPumps. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class CancelMsg1:

Purpose	Concrete strategy of CancelMsg action for GasPump1 and GasPump2.
Member Variables	No Variables are defined.
Operations	**cancel_msg():** function displays the message "operation has been cancelled"

Class DisplayMenu:

Purpose	Abstract class that group strategies of DisplayMenu action
Member Variables	ds: pointer to DataStore.
Operations	**display_msg():** abstract method which will be override by subclass to display different menu message depending on the GasPumps. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class DisplayMenu1:

Purpose	Concrete strategy of DisplayMenu action for GasPump1.
Member Variables	No Variables are defined.
Operations	**display_menu():** function displays the menu with different gas options for GasPump-1.

Class DisplayMenu2:

Purpose	Concrete strategy of DisplayMenu action for GasPump2.
Member Variables	No Variables are defined.
Operations	**display_menu():** function displays the menu with different gas options for GasPump-2.

Class PayMsg:

Purpose	Abstract class that group strategies of PayMsg action
Member Variables	ds: pointer to DataStore.
Operations	**pay_msg():** abstract method which will be override by subclass to display pay message depending on the GasPumps. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class PayMsg1:

Purpose	Concrete strategy of PayMsg action for GasPump-1 and GasPump-2.
Member Variables	No Variable are defined.
Operations	**pay_msg():** this function displays "select different payment method" message.

Class PrintReceipt:

Purpose	Abstract class that group strategies of PrintReceipt action
Member Variables	ds: pointer to DataStore.
Operations	**printreceipt():** abstract method which will be override by subclass to print different receipt depending on the GasPumps. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class PrintReceipt1:

Purpose	Concrete strategy of PrintReceipt action for GasPump1.
Member Variables	No Variables are defined.
Operations	**printreceipt():** function calculate the amount user has to pay for pumped gas.

Class PrintReceipt2:

Purpose	Concrete strategy of PrintReceipt action for GasPump2.
Member Variables	No Variables are defined.
Operations	**printreceipt():** function calculates total amount of gas pumped and total amount user has to pay at the end for pumped gas.

Class ReadyMsg:

Purpose	Abstract class that group strategies of ReadyMsg action
Member Variables	ds: pointer to DataStore.
Operations	**readymsg():** abstract method which will be override by subclass to display ready message depending on the GasPumps. **getDs():** Getter function to return the DataStore **setDs():** Setter Function set the DataStore

Class ReadyMsg1:

Purpose	Concrete strategy of ReadyMsg action for GasPump-1 and GasPump-2.
Member Variables	No Variable are defined.
Operations	**readymsg():** this function displays “Gas pump is ready to pump gas” message.

Class RejectMsg:

Purpose	Abstract class that group strategies of RejectMsg action
Member Variables	ds: pointer to DataStore.
Operations	**reject_msg():** abstract method which will be override by subclass to display reject message depending on the GasPumps. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class RejectMsg1:

Purpose	Concrete strategy of RejectMsg action for GasPump-1 and GasPump-2.
Member Variables	No Variable are defined.
Operations	**reject_msg():** this function displays "Payment method is rejected" message.

Class ReturnCash:

Purpose	Abstract class that group strategies of ReturnCash action
Member Variables	ds: pointer to DataStore.
Operations	**returncash():** abstract method which will be override by subclass to return remaining amount of cash for GasPump-2. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class ReturnCash2:

Purpose	Concrete strategy of ReturnCash action for GasPump-2.
Member Variables	No Variable are defined.
Operations	**returncash():** this function calculates the remaining amount of cash user will get once he has paid the cash for pumped gas.

Class SetInitValues:

Purpose	Abstract class that group strategies of SetInitValues action
Member Variables	ds: pointer to DataStore.
Operations	**setinitvalues():** abstract method which will be override by subclass to initialize value of G/L and total depending on the GasPumps. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class SetInitValues1:

Purpose	Concrete strategy of SetInitValues action for GasPump1.
Member Variables	No Variables are defined.
Operations	**setinitvalues():** function initializes value of G as zero and value of total as zero.

Class SetInitValues2:

Purpose	Concrete strategy of SetInitValues action for GasPump2.
Member Variables	No Variables are defined.
Operations	**setinitvalues():** function initializes value of L as zero and value of total as zero.

Class SetPrice:

Purpose	Abstract class that group strategies of SetPrice action
Member Variables	ds: pointer to DataStore.
Operations	**setPrice(int g):** abstract method which will be override by subclass to set values of different gas depending on the GasPumps. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class SetPrice1:

Purpose	Concrete strategy of SetPrice action for GasPump1.
Member Variables	No Variables are defined.
Operations	**setPrice(g):** function set prices of Regular gas or Super gas depending on what type of gas user has selected.

Class SetPrice2:

Purpose	Concrete strategy of SetPrice action for GasPump2.
Member Variables	No Variables are defined.
Operations	**setPrice(g):** function set prices of Regular gas, Super gas and Premium gas depending on what type of gas user has selected.

Class StopMsg:

Purpose	Abstract class that group strategies of StopMsg action
Member Variables	ds: pointer to DataStore.
Operations	**stopmsg():** abstract method which will be override by subclass to display stop message depending on the GasPumps. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class StopMsg1:

Purpose	Concrete strategy of StopMsg action for GasPump-1 and GasPump-2.
Member Variables	No Variable are defined.
Operations	**stopmsg():** this function displays “Pump has been stopped” message.

Class StoreCash:

Purpose	Abstract class that group strategies of StoreCash action
Member Variables	ds: pointer to DataStore.
Operations	**storecash():** abstract method which will be override by subclass to store the amount of cash inputted by the user used in GasPump2. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class StoreCash1:

Purpose	Concrete strategy of StoreCash action for GasPump-2.
Member Variables	No Variable are defined.
Operations	**storecash():** this function accepts the cash from the user.

Class StoreData:

Purpose	Abstract class that group strategies of StoreData action
Member Variables	ds: pointer to DataStore.
Operations	**storeData():** abstract method which will be override by subclass to set the price of gasses depending on the GasPump. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class StoreData1:

Purpose	Concrete strategy of StoreData action for GasPump-1.
Member Variables	No Variable are defined.
Operations	**storeData():** this function set the prices of Regular gas and Super gas.

Class StoreData2:

Purpose	Concrete strategy of StoreData action for GasPump-2.
Member Variables	No Variable are defined.
Operations	**storeData():** this function set the prices of Regular gas, Super gas and premium gas.

Class Total:

Purpose	Abstract class that group strategies of Total action
Member Variables	ds: pointer to DataStore.
Operations	**total():** abstract method which will be override by subclass to calculate the value of G/L and total depending on the GasPump. **getDs():** Getter function to return the DataStore **setDs():** Setter function set the DataStore

Class Total1:

Purpose	Concrete strategy of Total action for GasPump-1.
Member Variables	No Variable are defined.
Operations	**total():** this function calculates value of G and total and display amount of gas pumped.

Class Total2:

Purpose	Concrete strategy of Total action for GasPump-2.
Member Variables	No Variable are defined.
Operations	**total():** this function calculates value of L and total and display amount of gas pumped.

2.5 Abstract Factory Pattern: this section focuses on Abstract Factory pattern & Concrete Factory 1 Pattern and Concrete Factory 2 Pattern.

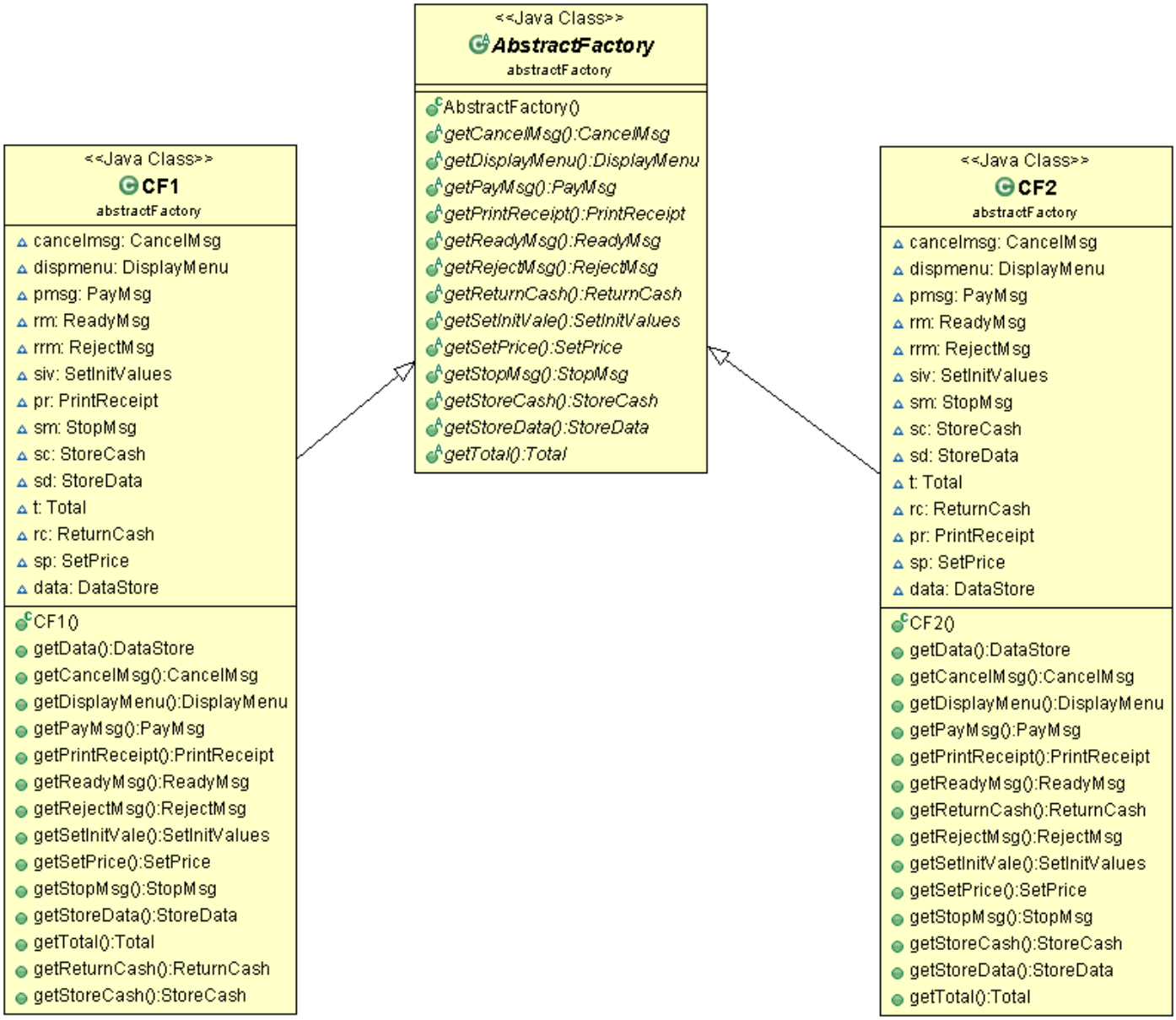

Fig 2.5 Abstract Factory Pattern Diagram

Responsibilities and operations of all Abstract Factory Pattern, Concrete Factory 1 Pattern and Concrete Factory 2 Pattern.

Class AbstractFactory:

Purpose	This is a abstract class for Factory and it groups various of concrete factories for together that are held with the implementations
Member Variables	No variables are defined here.
Operations	**getCancelMsg()-** Abstract method returning CancelMsg() Strategy. **getDisplayMenu()-** Abstract method returning DisplayMenu() Strategy. **getPayMsg()-** Abstract method returning PayMsg() Strategy. **getPrintReceipt()-** Abstract method returning PrintReceipt() Strategy. **getReadyMsg()-** Abstract method returning ReadyMsg() Strategy. **getRejectMsg()-** Abstract method returning RejectMsg() Strategy. **getReturnCash()-** Abstract method returning ReturnCash() Strategy. **getSetInitValues()-** Abstract method returning SetInitValues() Strategy. **getSetPrice()-** Abstract method returning SetPrice() Strategy. **getStopMsg()-** Abstract method returning StopMsg() Strategy. **getStoreCash()-** Abstract method returning StoreCash() Strategy. **getStoreData()-** Abstract method returning StoreData() Strategy. **getTotal()** - Abstract method returning Total() Strategy.

Class CF1:

Purpose	Concrete class for GasPump-1 which create strategies, DataStore1 instances and objects specific for GasPump1.
Member Variables	cancelmsg: object of CancelMsg() dispmenu: object of DisplayMenu() pmsg: object of PayMsg() rm: object of ReadyMsg() rrm: object of RejectMsg() siv: object of SetInitValues() pr: object of PrintReceipt() sm: object of StopMsg() sc: object of StoreCash() sd: object of StoreData() t: object of Total() rc: object of ReturnCash() sp: object of SetPrice()
Operations	**getCancelMsg()-** Setting DataStore1 instance for the object of CancelMsg() and returning it. **getDisplayMenu()-** Setting DataStore1 instance for the object of DisplayMenu() and returning it. **getPayMsg()-** Setting DataStore1 instance for the object of PayMsg() and returning it. **getPrintReceipt()** - Setting DataStore1 instance for the object of PrintReceipt() and returning it. **getReadyMsg()-** Setting DataStore1 instance for the object of ReadyMsg() and returning it. **getRejectMsg()**-Setting DataStore1 instance for the object of RejectMsg() and returning it. **getReturnCash()-** return null because it is not used in GasPump1. **getSetInitValues()-** Setting DataStore1 instance for the object of SetInitValues() and returning it. **getSetPrice()-** Setting DataStore1 instance for the object of SetPrice() and returning it. **getStopMsg()-** Setting DataStore1 instance for the object of StopMsg() and returning it. **getStoreCash()-** return null because it is not used in GasPump1. **getStoreData()-** Setting DataStore1 instance for the object of StoreData() and returning it.. **getTotal() -** Setting DataStore1 instance for the object of Total() and returning it. **getData() -** returns DataStore1.

Class CF2:

Purpose	Concrete class for GasPump-2 which create strategies, DataStore2 instances and objects specific for GasPump2.
Member Variables	cancelmsg: object of CancelMsg() dispmenu: object of DisplayMenu() pmsg: object of PayMsg() rm: object of ReadyMsg() rrm: object of RejectMsg() siv: object of SetInitValues() pr: object of PrintReceipt() sm: object of StopMsg() sc: object of StoreCash() sd: object of StoreData() t: object of Total() rc: object of ReturnCash() sp: object of SetPrice()
Operations	**getCancelMsg()-** Setting DataStore2 instance for the object of CancelMsg() and returning it. **getDisplayMenu()-**Setting DataStore2 instance for the object of DisplayMenu() and returning it. **getPayMsg()-** Setting DataStore2 instance for the object of PayMsg() and returning it. **getPrintReceipt()-**Setting DataStore2 instance for the object of PrintReceipt() and returning it. **getReadyMsg()-** Setting DataStore2 instance for the object of ReadyMsg() and returning it. **getRejectMsg**() -Setting DataStore2 instance for the object of RejectMsg() and returning it. **getReturnCash()-**Setting DataStore2 instance for the object of ReturnCash() and returning it. **getSetInitValues()-**Setting DataStore2 instance for the object of SetInitValues() and returning it. **getSetPrice()-**Setting DataStore2 instance for the object of SetPrice() and returning it. **getStopMsg()-**Setting DataStore2 instance for the object of StopMsg() and returning it. **getStoreCash()-**Setting DataStore2 instance for the object of StoreCash() and returning it. **getStoreData()-**Setting DataStore2 instance for the object of StoreData() and returning it. **getTotal() -** Setting DataStore2 instance for the object of Total() and returning it. **getData():** - returns DataStore2.

2.6 DataStore: focuses on DataStore, DataStore1 and DataStore2 details.

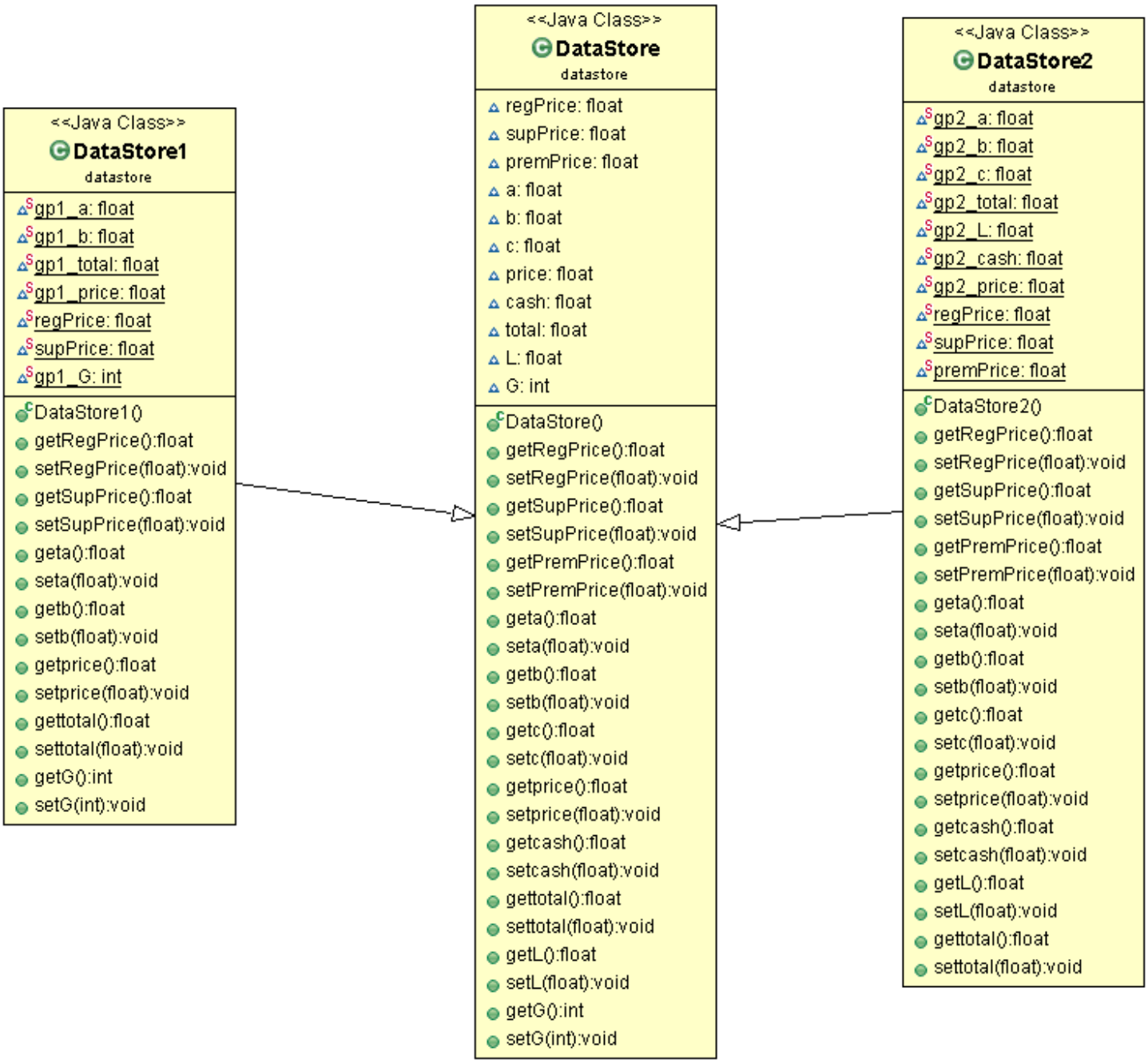

Fig 2.6 DataStore Diagram

Responsibilities and operations of all DataStore, DataStore 1 pattern and DataStore2 classes.
Class DataStore:

Purpose	This class represents the abstract class for the DataStore and is used to group the DataStore1 of GasPump1 and DataStore2 of GasPump2.
Member Variables	regPrice: float variable to store price of Regular Gas. supPrice: float variable to store price of Super Gas. premPrice: float variable to store price of Premium Gas. a: float variable to hold value of 'a' b: float variable to hold value of 'b' c: float variable to hold value of 'c' price: float variable to hold value of 'price' cash: float variable to hold value of 'cash' total: float variable to hold value of 'total' L: float variable to hold value of 'L' G: float variable to hold value of 'G'
Operations	Getter and Setter of above mentioned member variables.

Class DataStore1:

Purpose	DataStore used in implementing GasPump1's logic.
Member Variables	regPrice: float variable to store price of Regular Gas. supPrice: float variable to store price of Super Gas. gp1_a: float variable to hold value of 'a' gp1_b: float variable to hold value of 'b' gp1_price: float variable to hold value of 'price' gp1_total: float variable to hold value of 'total' gp1_G: float variable to hold value of 'G'
Operations	getRegPrice(): return price of Regular Gas. setRegPrice(float): set the price of Regular gas. getSupPrice(): return price of Super Gas. setSupPrice(float): set the price of Super gas. geta() : return value of 'a' seta(float): set the value of 'a' getb() : return value of 'b' setb(float): set the value of 'b' getprice(): return the value of 'price' setprice(float): set the value of 'price' gettotal(): return the value of 'total' settotal(float): set the value of 'total' getG(): return the value of 'G' setG(int): set the value of 'G'

Class DataStore2:

Purpose	DataStore used in implementing GasPump2's logic.
Member Variables	regPrice: float variable to store price of Regular Gas. supPrice: float variable to store price of Super Gas. premPrice: float variable to store price of Premium Gas. gp2_a: float variable to hold value of 'a' gp2_b: float variable to hold value of 'b' gp2_c: float variable to hold value of 'c' gp2_price: float variable to hold value of 'price' gp2_total: float variable to hold value of 'total' gp2_L: float variable to hold value of 'L' gp2_cash: float variable to hold value of 'cash'
Operations	getRegPrice(): return price of Regular Gas. setRegPrice(float): set the price of Regular gas. getSupPrice(): return price of Super Gas. setSupPrice(float): set the price of Super gas. getPremPrice(): return price of Premium Gas. setPremPrice(float): set the price of Premium gas. geta() : return value of 'a' seta(float): set the value of 'a' getb() : return value of 'b' setb(float): set the value of 'b' getc() : return value of 'c' setc(float): set the value of 'c' getprice(): return the value of 'price' setprice(float): set the value of 'price' gettotal(): return the value of 'total' settotal(float): set the value of 'total' getL(): return the value of 'L' setL(int): set the value of 'L' getcash(): return the value of 'cash' setcash(float): set the value of 'cash'

3. Dynamics: Sequence Diagram of two Scenarios.

3.1 Sequence Diagram of Scenario-I

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(). Since Sequence diagram for Scenario-I is big in size, it is broken down into following operations.

Activate(3.1, 4.3) -

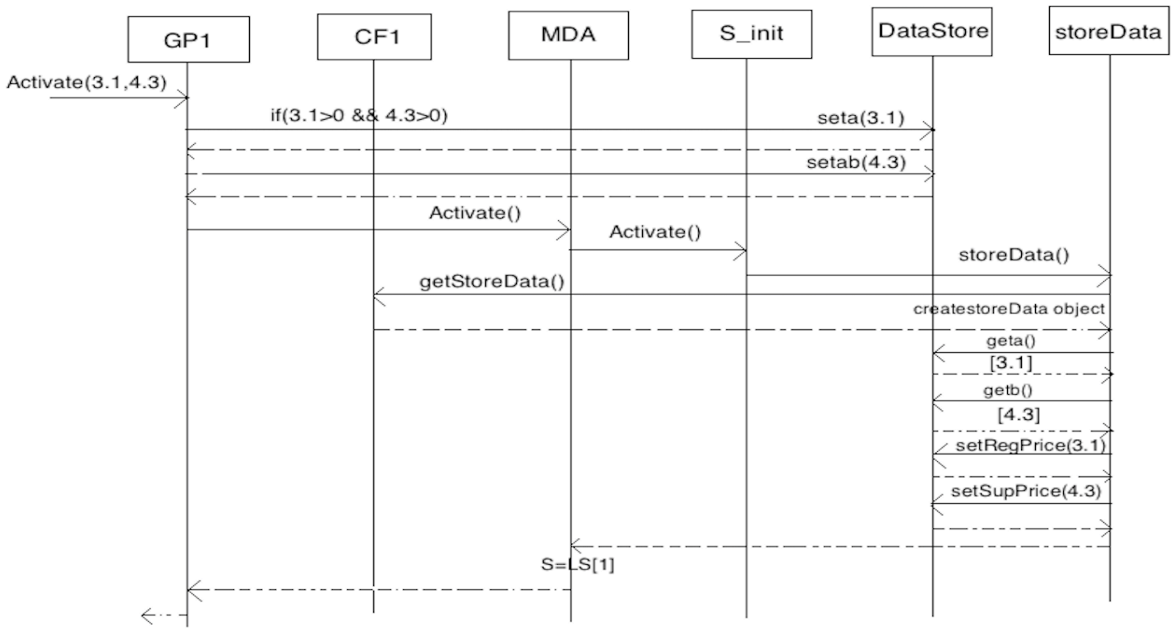

Fig 3.1.1 Scenario I – Activate(3.1, 4.3)

Start() -

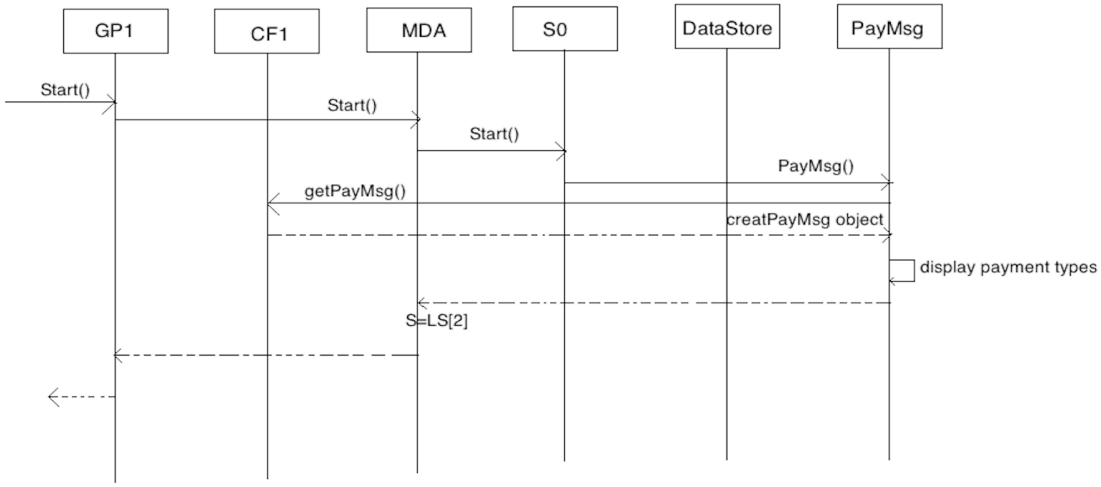

Fig 3.1.2 Scenario I – Start()

PayCredit() -

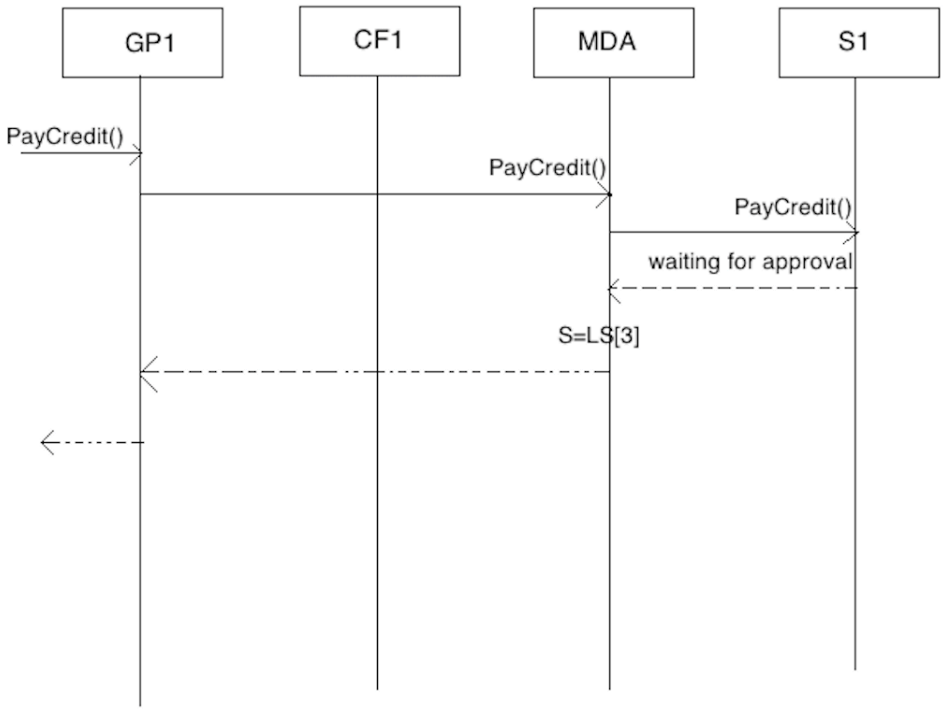

Fig 3.1.3 Scenario I – PayCredit()

Approved() -

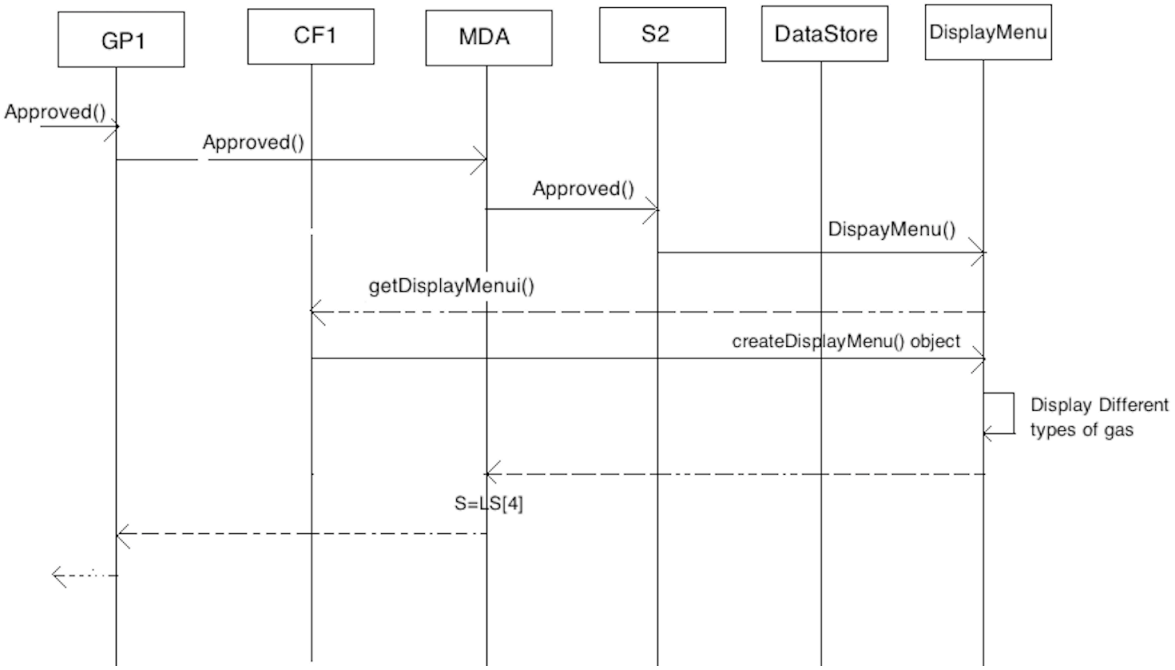

Fig 3.1.4 Scenario I – Approved()

Regular() -

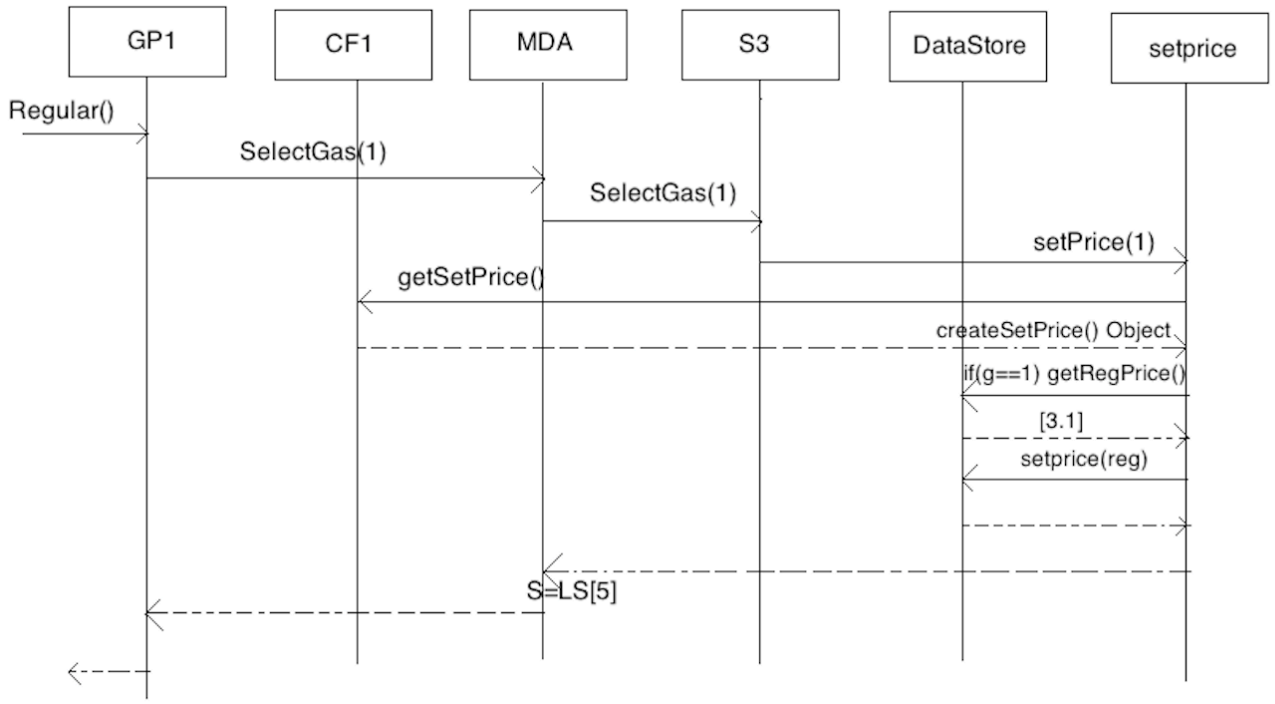

Fig 3.1.5 Scenario I – Regular()

StartPump() -

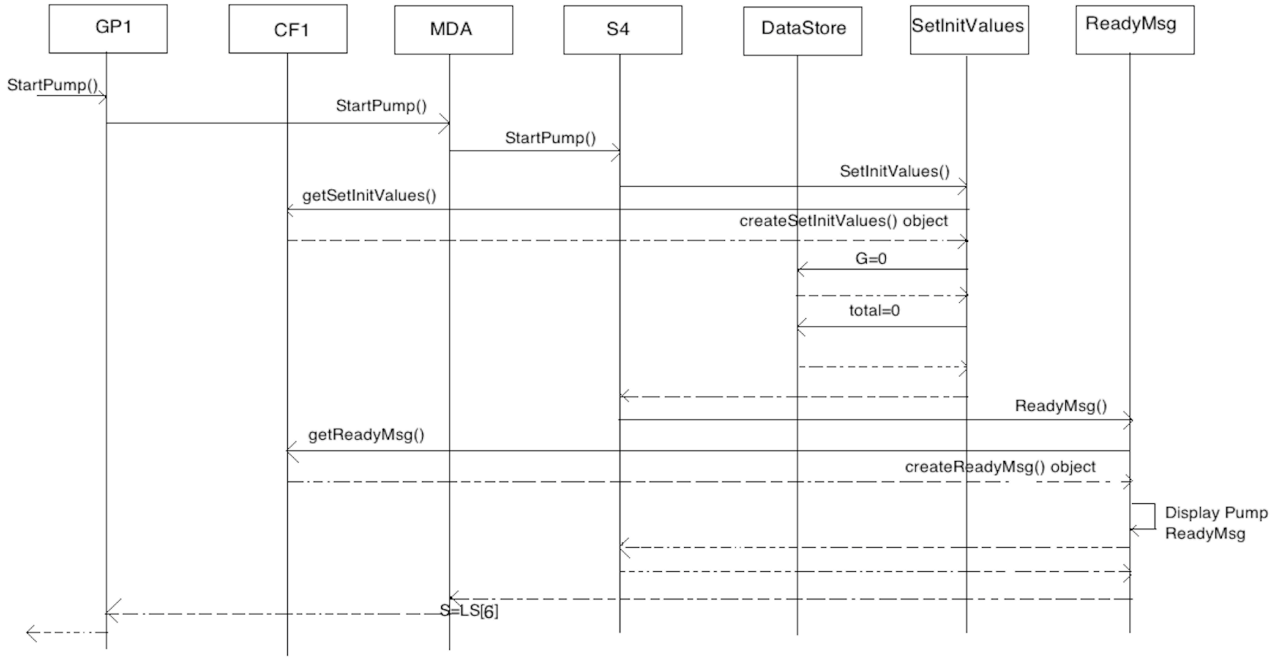

Fig 3.1.6 Scenario I – StartPump()

PumpGallon() –

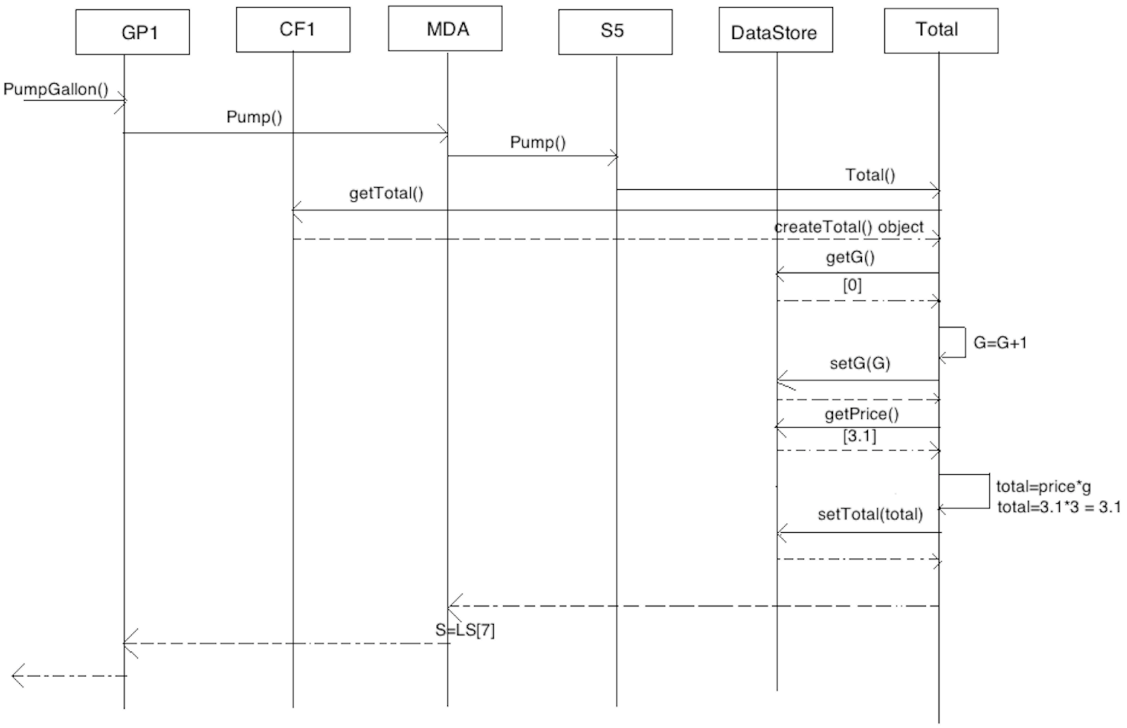

Fig 3.1.7 Scenario I – PumpGallon()

StopPump() –

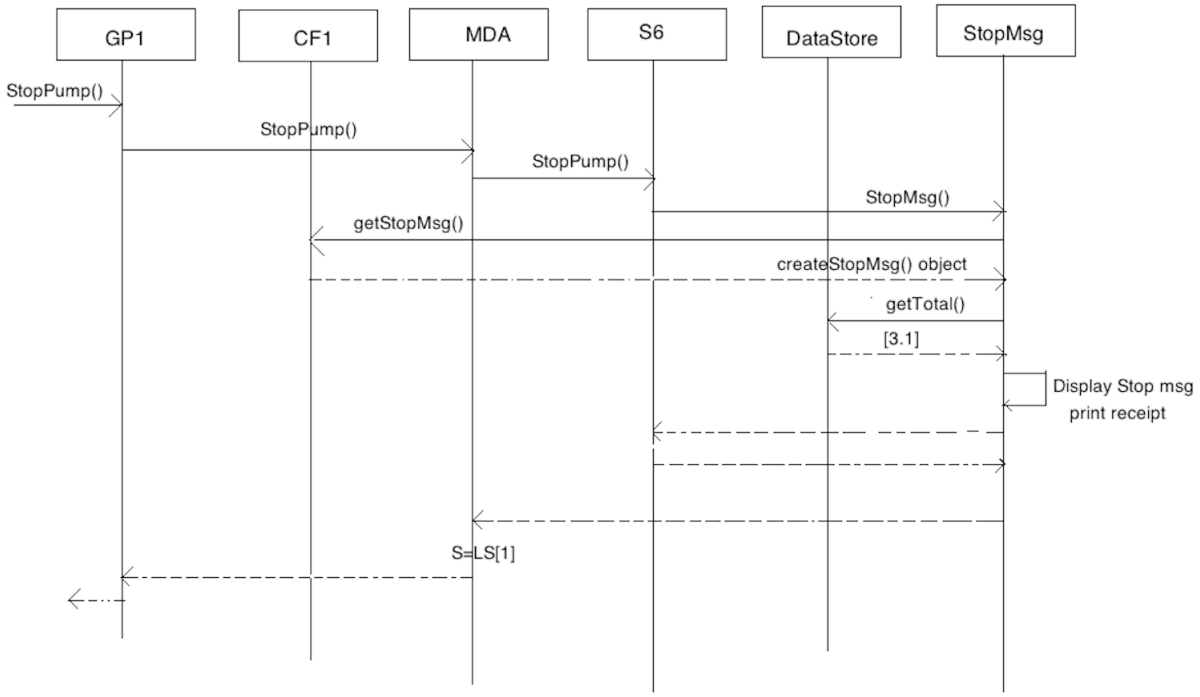

Fig 3.1.8 Scenario I – StopPump()

3.2 Sequence Diagram of Scenario-2

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(), PumpLitre(), PumpLitre(), NoReceipt() Since Sequence diagram for Scenario-II is big in size, it is broken down into following operations.

Activate(3, 4, 5) -

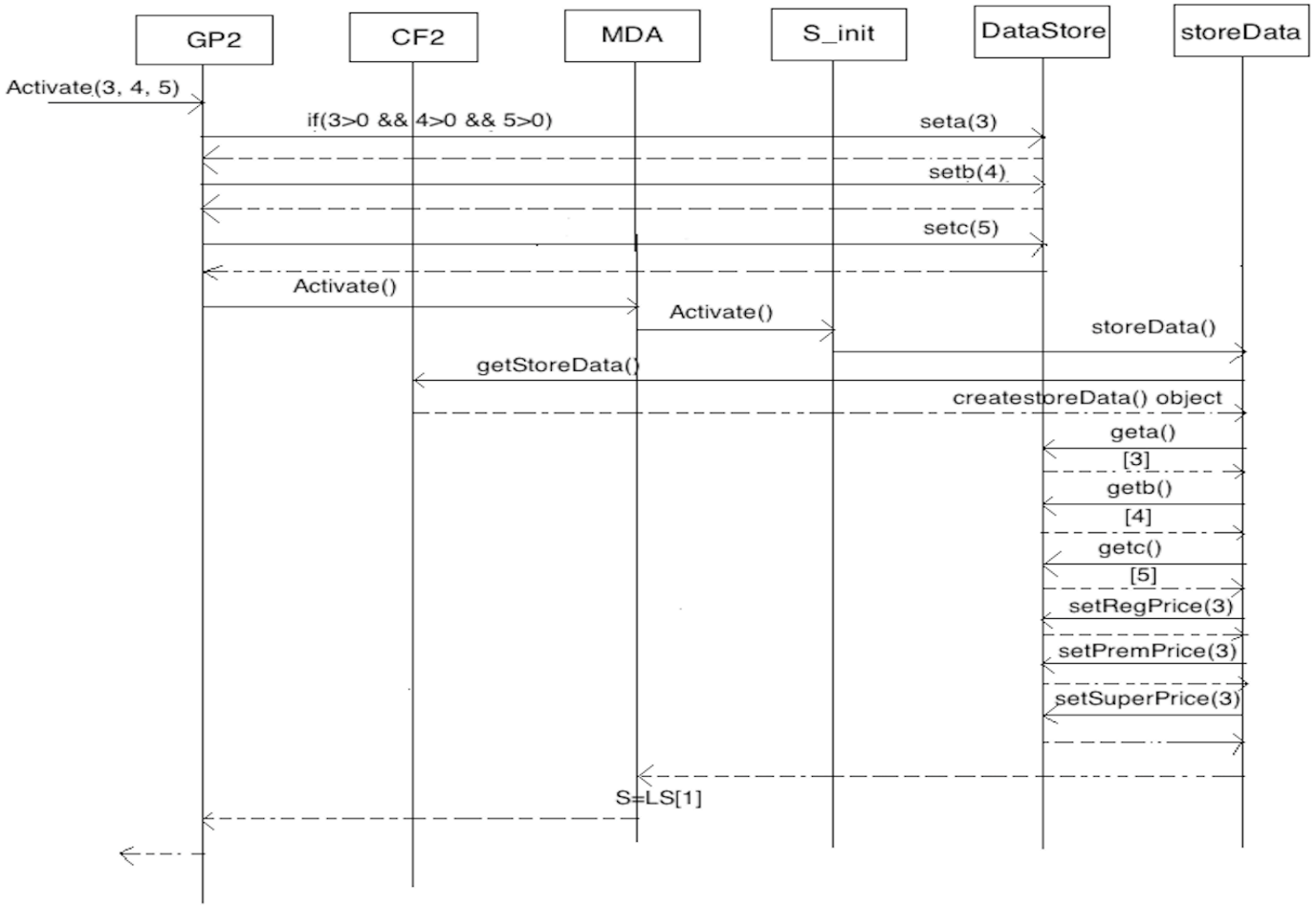

Fig 3.2.1 Scenario II – Activate(3, 4, 5)

Start() –

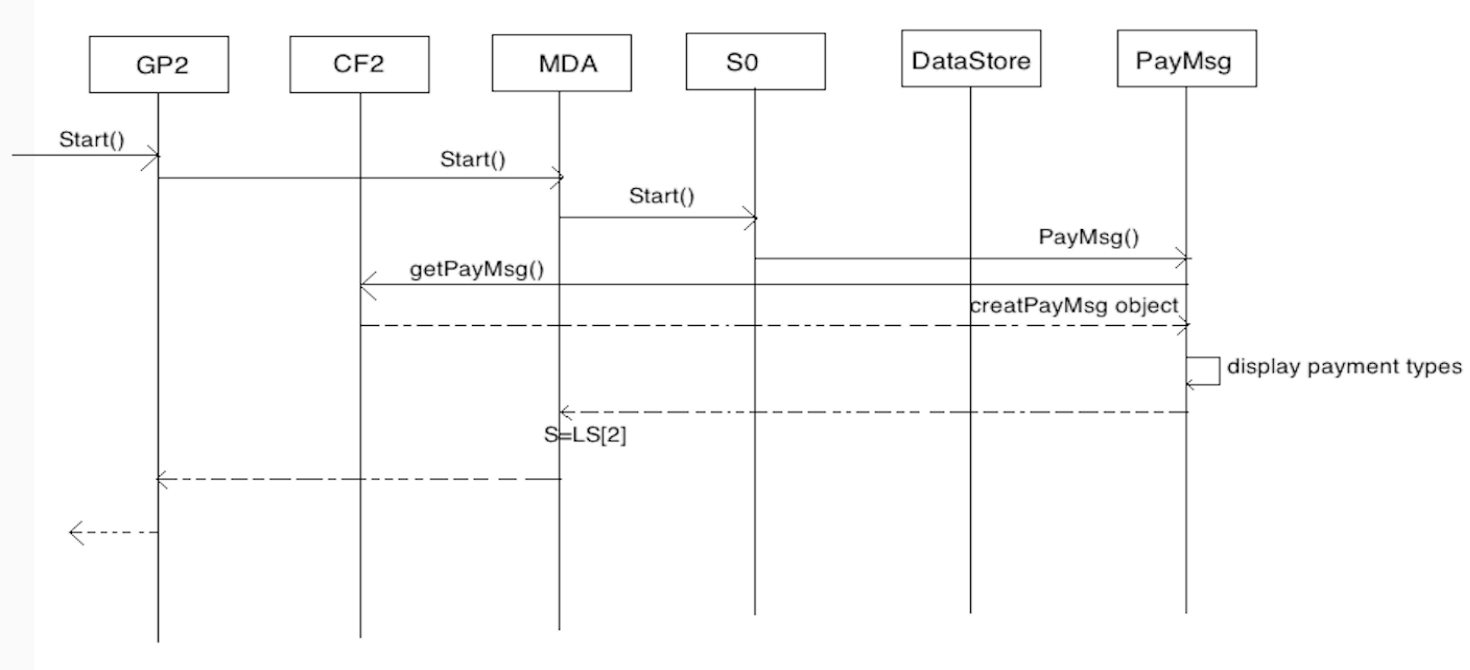

Fig 3.2.2 Scenario II – Start()

PayCash(6) –

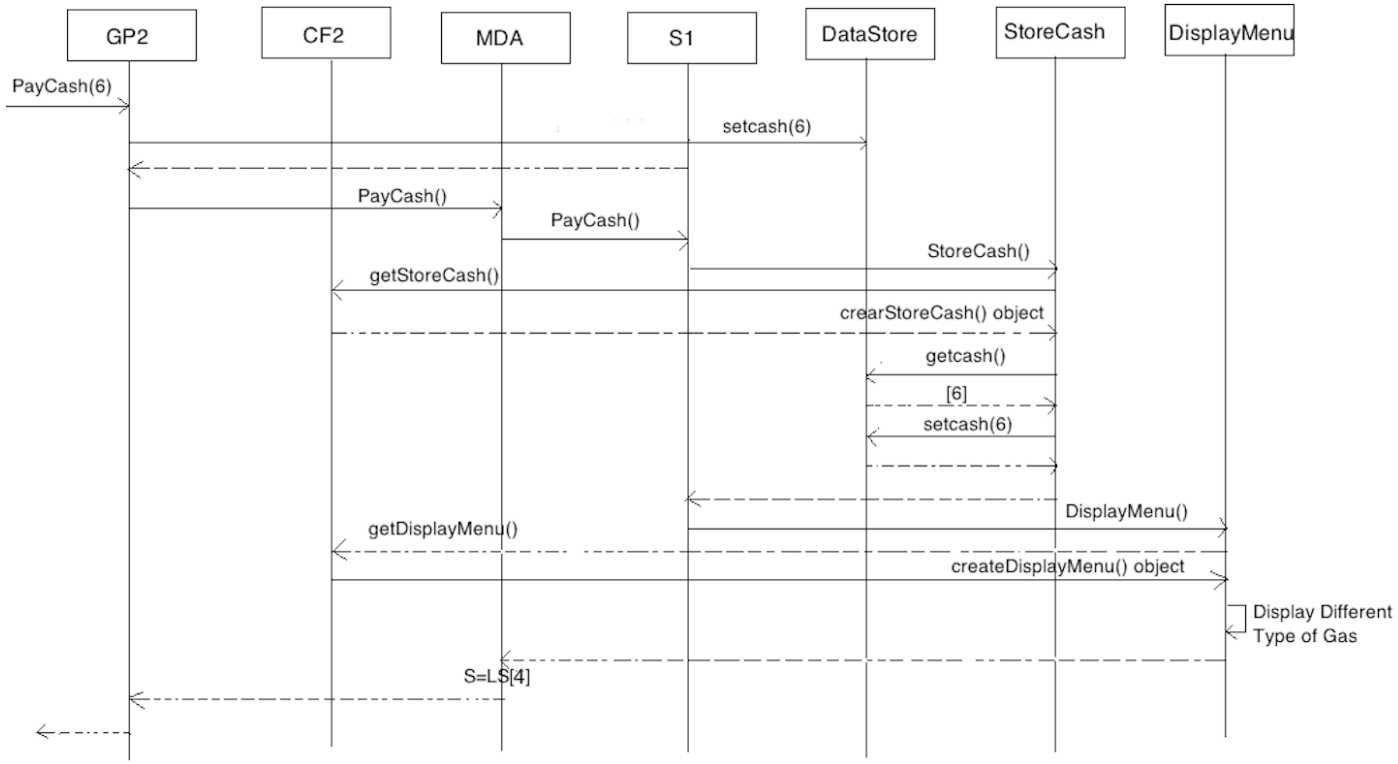

Fig 3.2.3 Scenario II – PayCash(6)

Premium() –

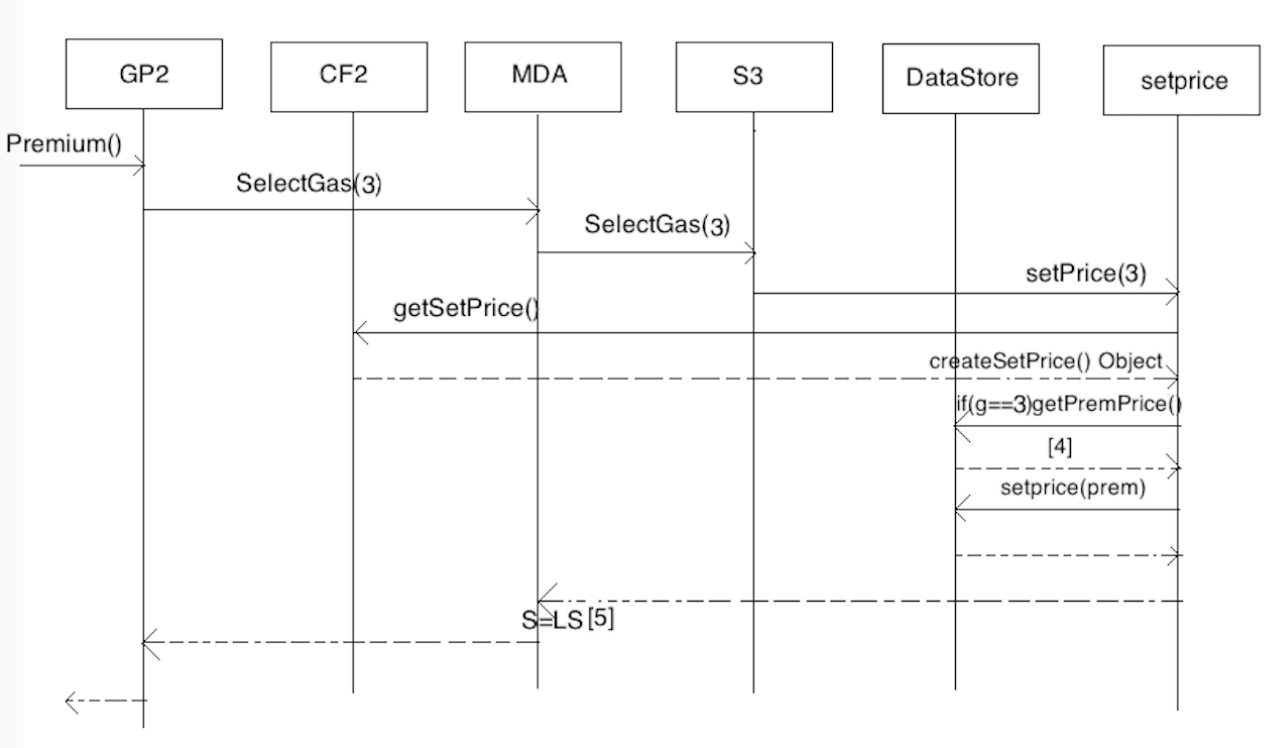

Fig 3.2.4 Scenario II – Premium()

StartPump() -

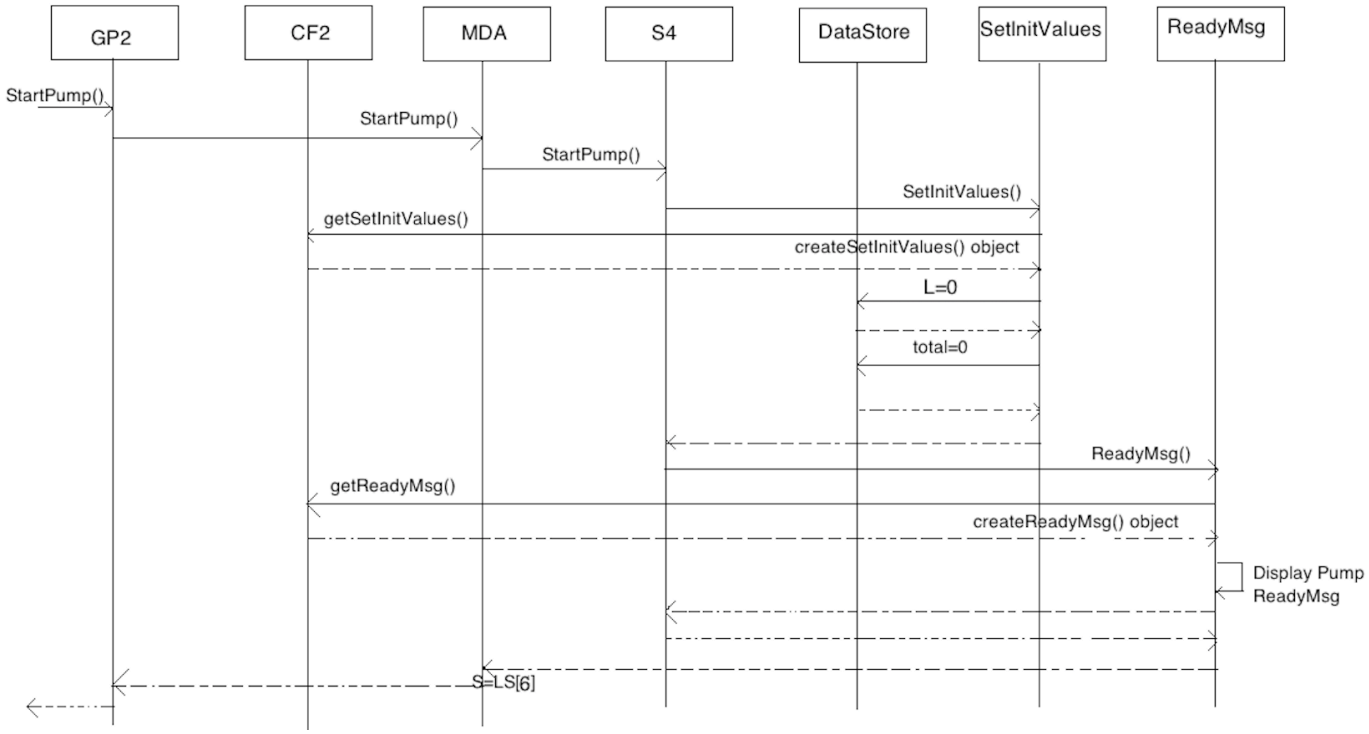

Fig 3.2.5 Scenario II – StartPump()

PumpLitre() –

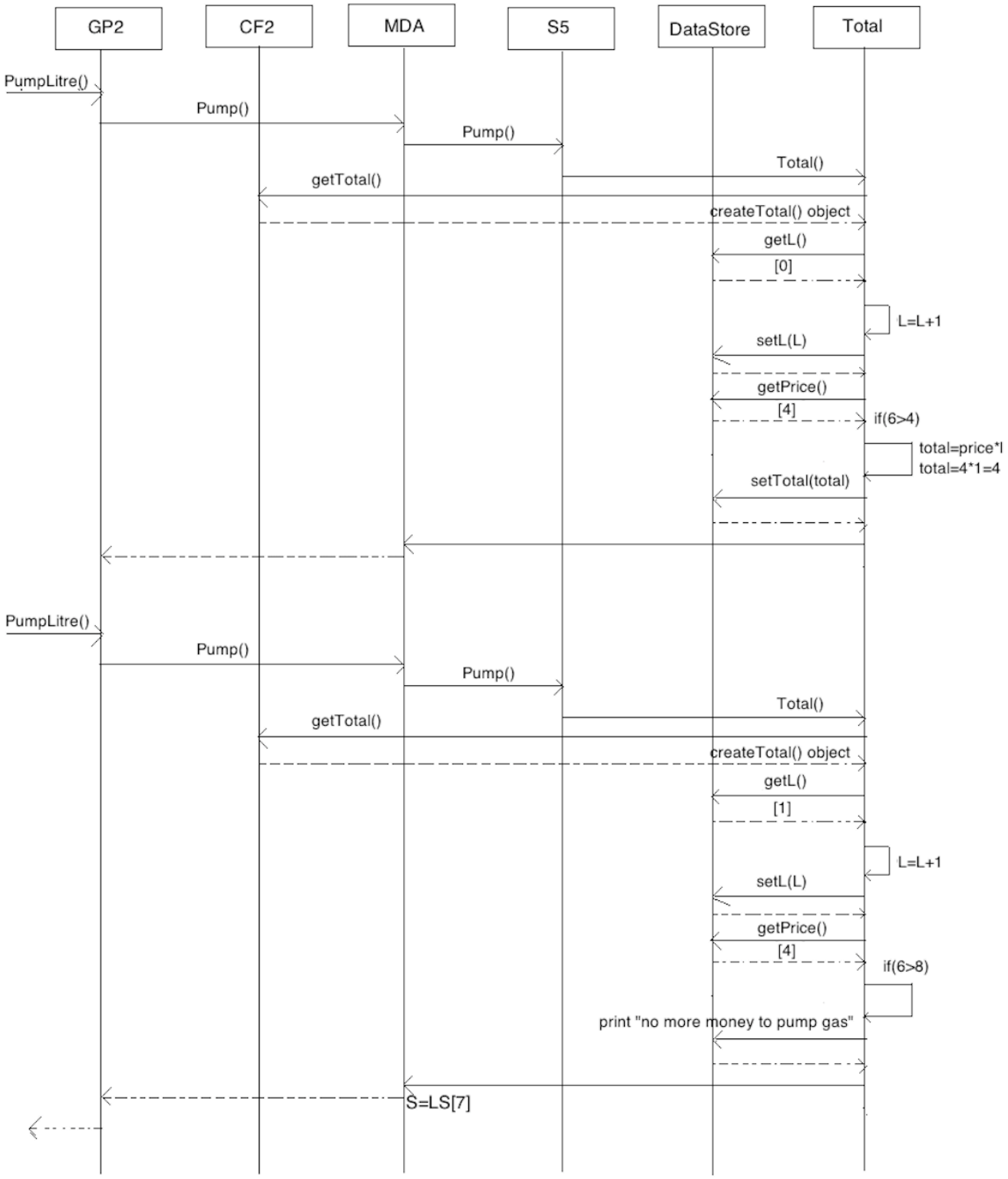

Fig 3.2.6 Scenario II – PumpLitre()

NoReceipt() -

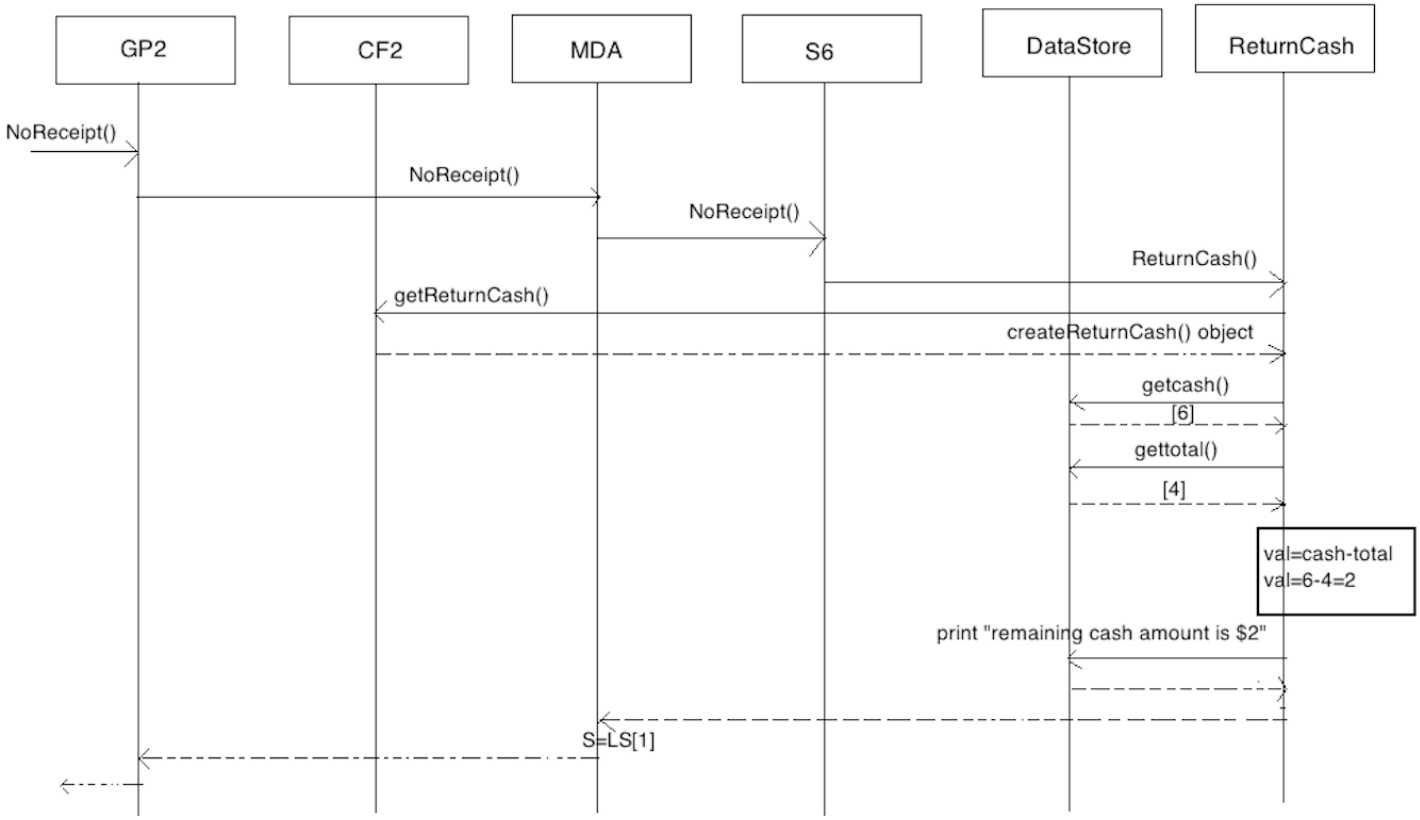

Fig 3.2.7 Scenario II – NoReceipt()

4. Source Code and Pattern:

4.1 State Patterns –

Source code of State Patterns can be found in **'/Source Code/Project_GasPump/src/states'** package. It contains code of all the states which are used for maintaining the state transition of MDA-EFSM.

There are **9** (including parent class) states in total which are as follow-

'State' class – **Parent** of all states class.
Concrete states classes – includes 'S_init', 'S0', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6' classes**.**
Client Class- MDA-EFSM

Description of all state classes are already mentioned in Section 3.

Source code of each individual state classes can be found in following way–
State class - /Source Code/Project_GasPump/src/states/State.java
S0 class - /Source Code/Project_GasPump/src/states/S0.java
S1 class - /Source Code/Project_GasPump/src/states/S1.java
S2 class - /Source Code/Project_GasPump/src/states/S2.java
S3 class - /Source Code/Project_GasPump/src/states/S3.java
S4 class - /Source Code/Project_GasPump/src/states/S4.java
S5 class - /Source Code/Project_GasPump/src/states/S5.java
S6 class - /Source Code/Project_GasPump/src/states/S6.java

4.2 Strategy Patterns –

Source code of Strategy Pattern can be found in ''**/Source Code/Project_GasPump/src/strategyPattern/'** package. It groups actions of different GasPumps.

Description of all strategy pattern classes are already mentioned in Section 3.
Different types of Strategy Pattern Classes are as follow –

Strategy Abstract Classes includes CancelMsg, DisplayMenu, PayMsg, PrintReceipt, ReadyMsg, RejectMsg, ReturnCash, SetInitValues, SetPrice, StopMsg, StoreCash, StoreData, Total.

Strategy Concrete Classes included CancelMsg1, DisplayMenu1, DisplayMenu2, PayMsg1, PrintReceipt1, PrintReceipt2 ReadyMsg1, RejectMsg1, ReturnCash2, SetInitValues1, SetInitValues2, SetPrice1, SetPrice2 StopMsg1, StoreCash2, StoreData1, StoreData2. Total1, Total 2**.**

Client Class – Output

There are **32 Strategy Pattern classes** and source code of each individual Strategy Pattern can be easily found in the above mentioned package and naming convention is done in such a way to easily identify which class belongs to which Strategy Pattern.

4.3 Abstract Factory Patterns –
Source code of Abstract Pattern and Concrete Factory Patterns can be found in -
‘**‘/Source Code/Project_GasPump/src/abstractFactory/’** package.

Description of all Abstract Factory classes are already mentioned in Section 3.
There are 3 Abstract Factory classes which are as follows –

Abstract Factory Class – AbstractFactory class
Concrete Factory Classes – includes CF1 and CF2 classes.
Client Classes – GP1, GP2, Output

Source code of each individual Abstract Factory class and Concrete Factory classes can be found in following way–
AbstractFactory class - /Source Code/Project_GasPump/src/abstractFactory/AbstractFactory.java
CF1 class - /Source Code/Project_GasPump/src/abstractFactory/CF1.java
CF2 class – /Source Code/Project_GasPump/src/abstractFactory/CF2.java

Concrete products created by concrete factories –

For GasPump1 – DataStore1, CancelMsg1, DisplayMenu1, PayMsg1, ReadyMsg1, RejectMsg1, SetInitValues1, PrintReceipt1, StopMsg1, CancelMsg1, DisplayMenu1, Total1, ReturnCash2, SetPrice1.

For GasPump2 – DataStore2, CancelMsg2, DisplayMenu2, PayMsg2, ReadyMsg2, RejectMsg2, SetInitValues2, PrintReceipt2, StopMsg2, CancelMsg2, DisplayMenu2, Total2, ReturnCash2, SetPrice2.