Jafar Alzoubi A20501723

REPORT

I used The MDA EFSM that you posted on Blackboard.

1. MDA-EFSM model for the GP components

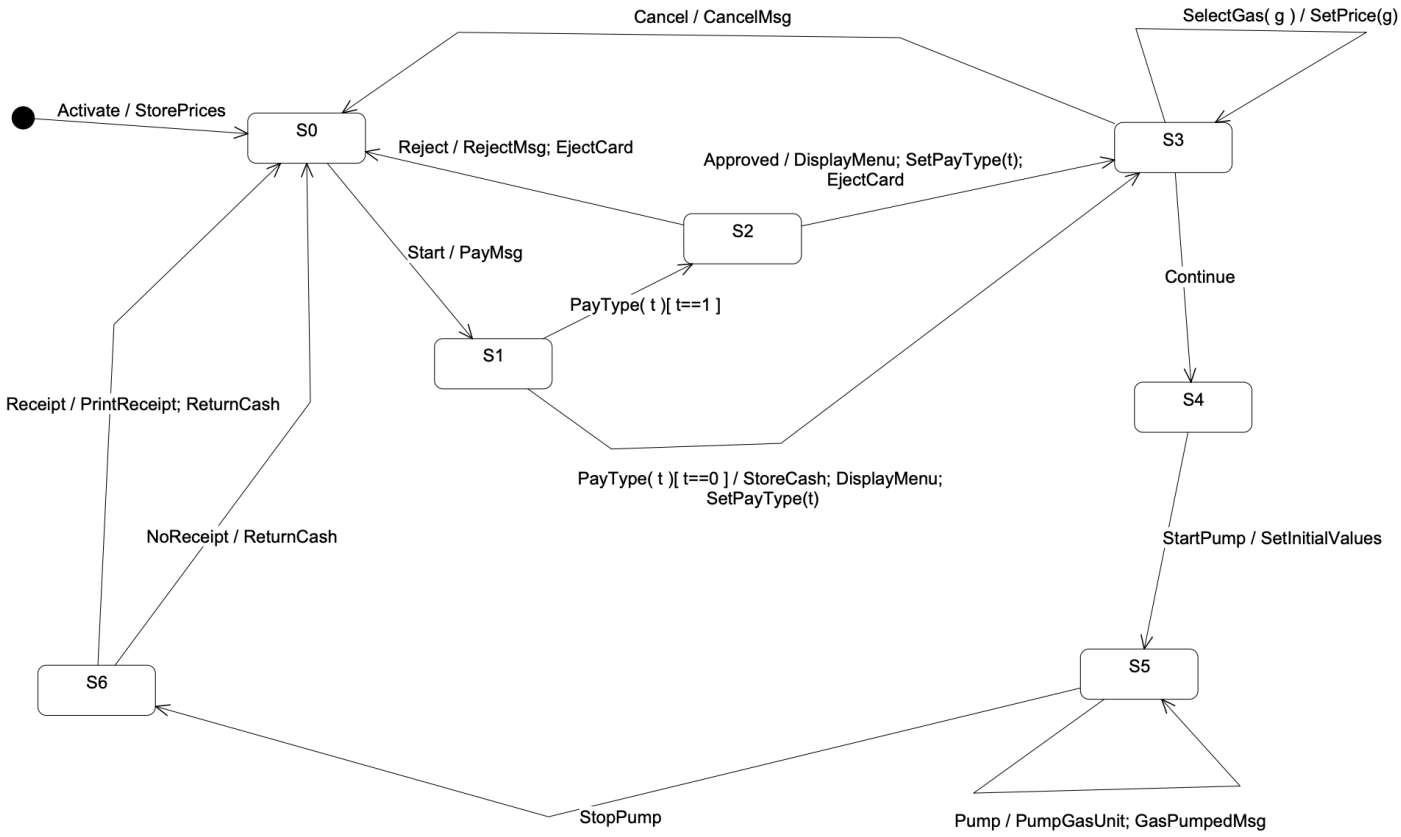
a. A list of meta events for the MDA-EFSM

```
Activate()
Start()
PayType(int t)          //credit: t=1; cash: t=0;
Reject()
Cancel()
Approved()
StartPump()
Pump()
StopPump()
SelectGas(int g)        // Regular: g=1; Diesel: g=2; Premium: g=3
Receipt()
NoReceipt()
Continue()
```

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

```
StorePrices()           // stores price(s) for the gas from the temporary data store
PayMsg()                // displays a type of payment method
StoreCash()             // stores cash from the temporary data store
DisplayMenu()           // display a menu with a list of selections
RejectMsg()             // displays credit card not approved message
SetPrice(int g)         // set the price for the gas identified by g identifier as in SelectGas(int g);
SetInitialValues()      // set G (or L) and total to 0;
PumpGasUnit()           // disposes unit of gas and counts # of units disposed and computes Total
GasPumpedMsg()          // displays the amount of disposed gas
PrintReceipt()          // print a receipt
CancelMsg()             // displays a cancellation message
ReturnCash()            // returns the remaining cash
SetPayType(t)           // Stores pay type t to variable w in the data store
EjectCard()             // Card is ejected
```

c. A state diagram of the MDA-EFSM



d. Pseudo-code of all operations of Input Processors of Gas Pump: GP-1 and GP

Operations of the Input Processor (GasPump-1)

```
Activate(int a) {
        if (a>0) {
            d->temp_a=a;
            m->Activate()
        }
}

Start() {
        m->Start();
}
PayCash(int c) {
        if (c>0) {
            d->temp_c=c;
            m->PayType(0)
        }
}

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

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

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

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

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

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

Notice:
cash: contains the value of cash deposited
price: contains the price of the gas
L: contains the number of liters already pumped
w: pay type flag (cash: w=0; credit: w=1)
cash, *L*, *price, w:* are in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

Operations of the Input Processor (GasPump-2)

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

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


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

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

Diesel() {
        m->SelectGas(2);
        m->Continue();
}
```

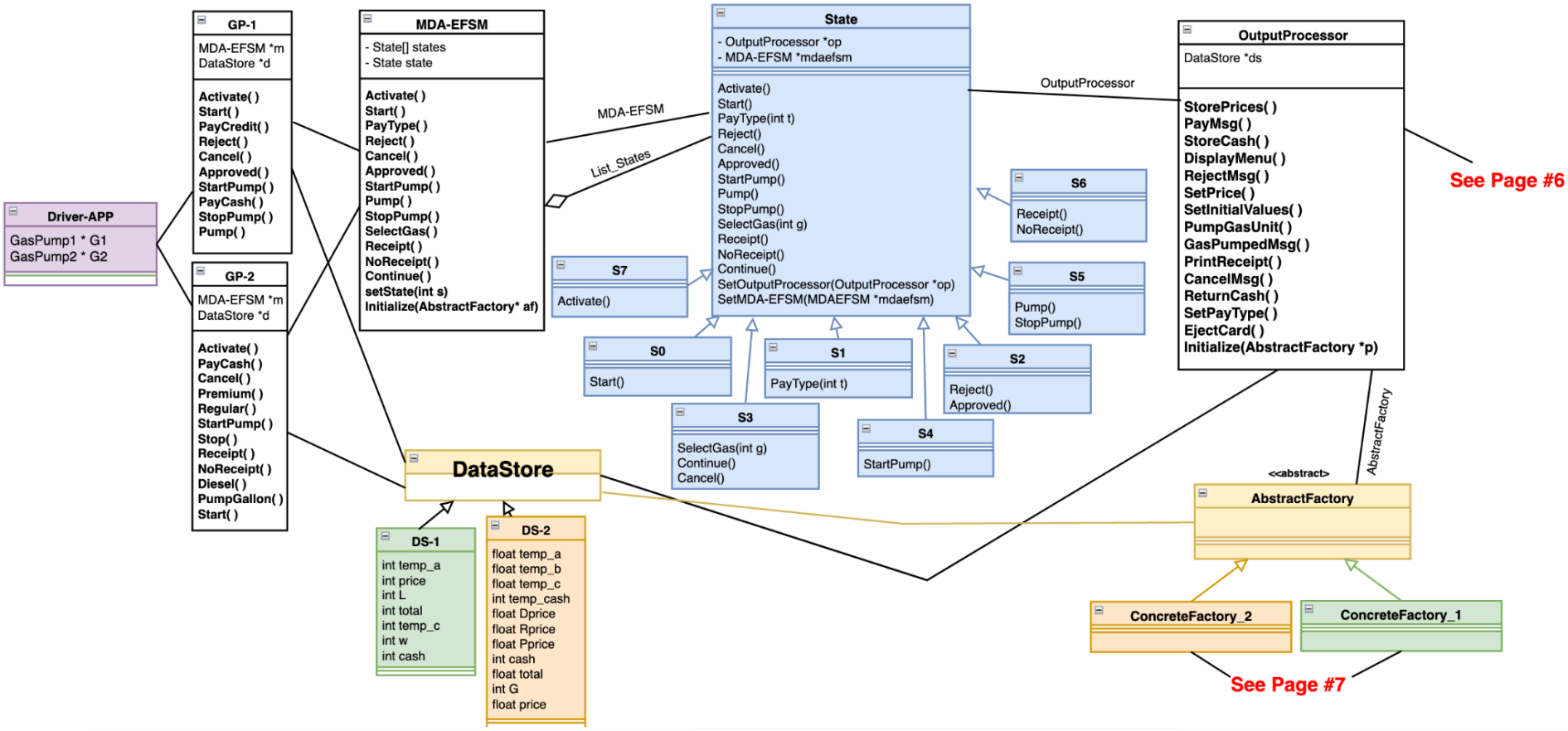
```
Premium() {
        m->SelectGas(3);
        m->Continue();
}

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

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

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

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

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

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

Notice:
cash: contains the value of cash deposited
price: contains the price of the selected gas
G: contains the number of Gallons already pumped

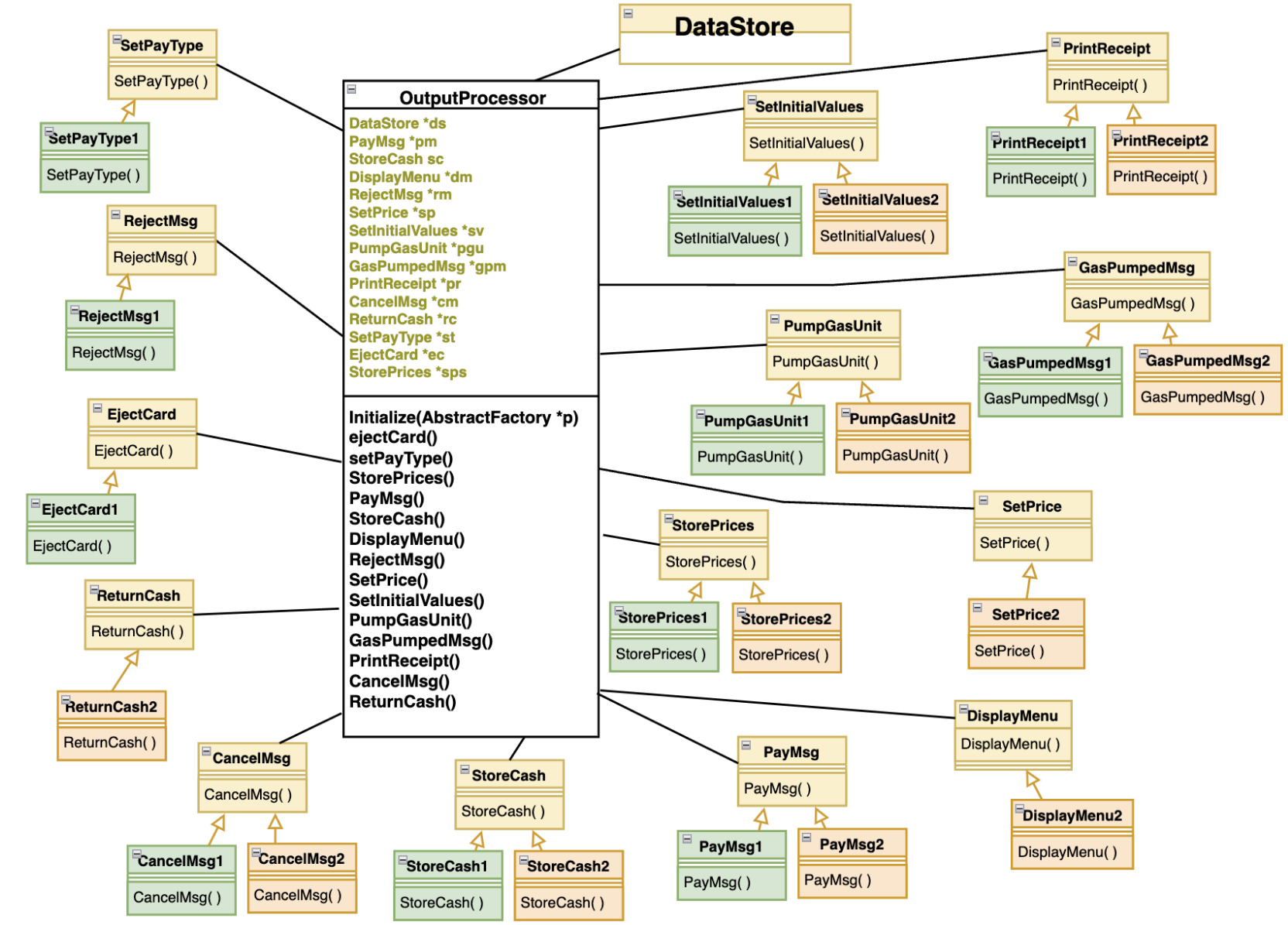
cash, *G*, *price* are in the data store
m: is a pointer to the MDA-EFSM object
d: is a pointer to the Data Store object

2. Class diagram(s) of the MDA of the GP components. In your design, you MUST use the following

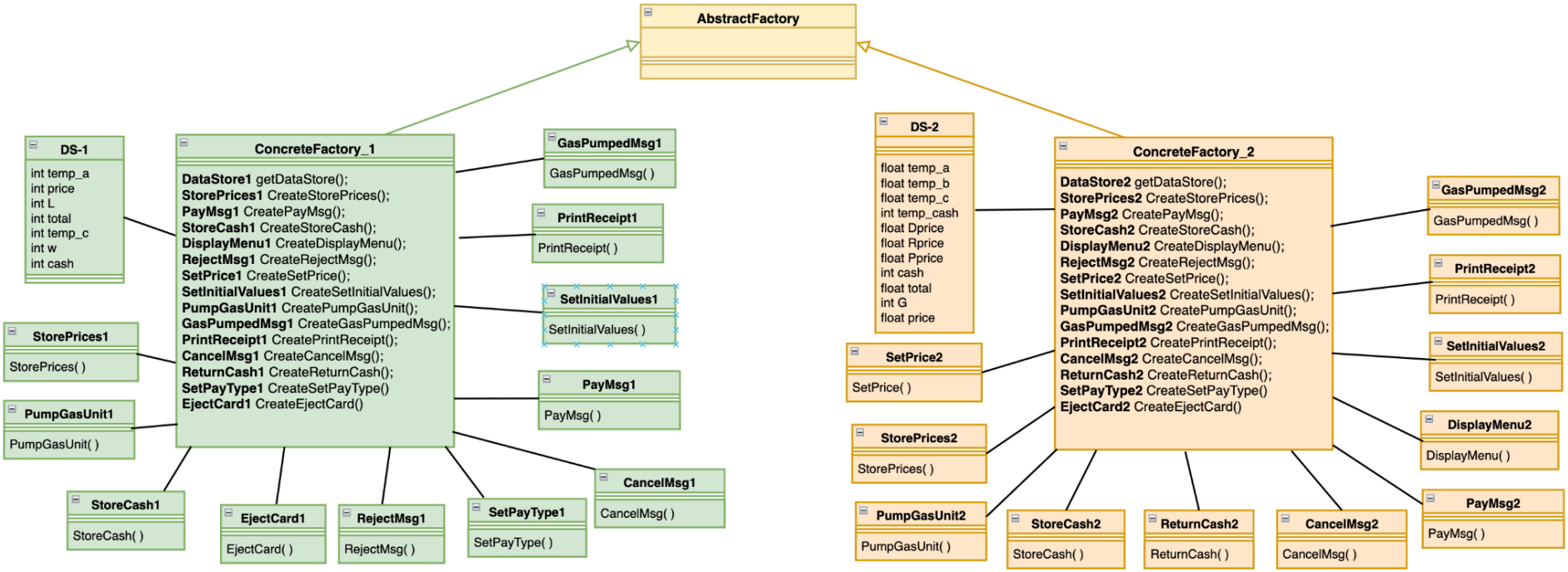
I have Split the Class into a couple multiple pages.



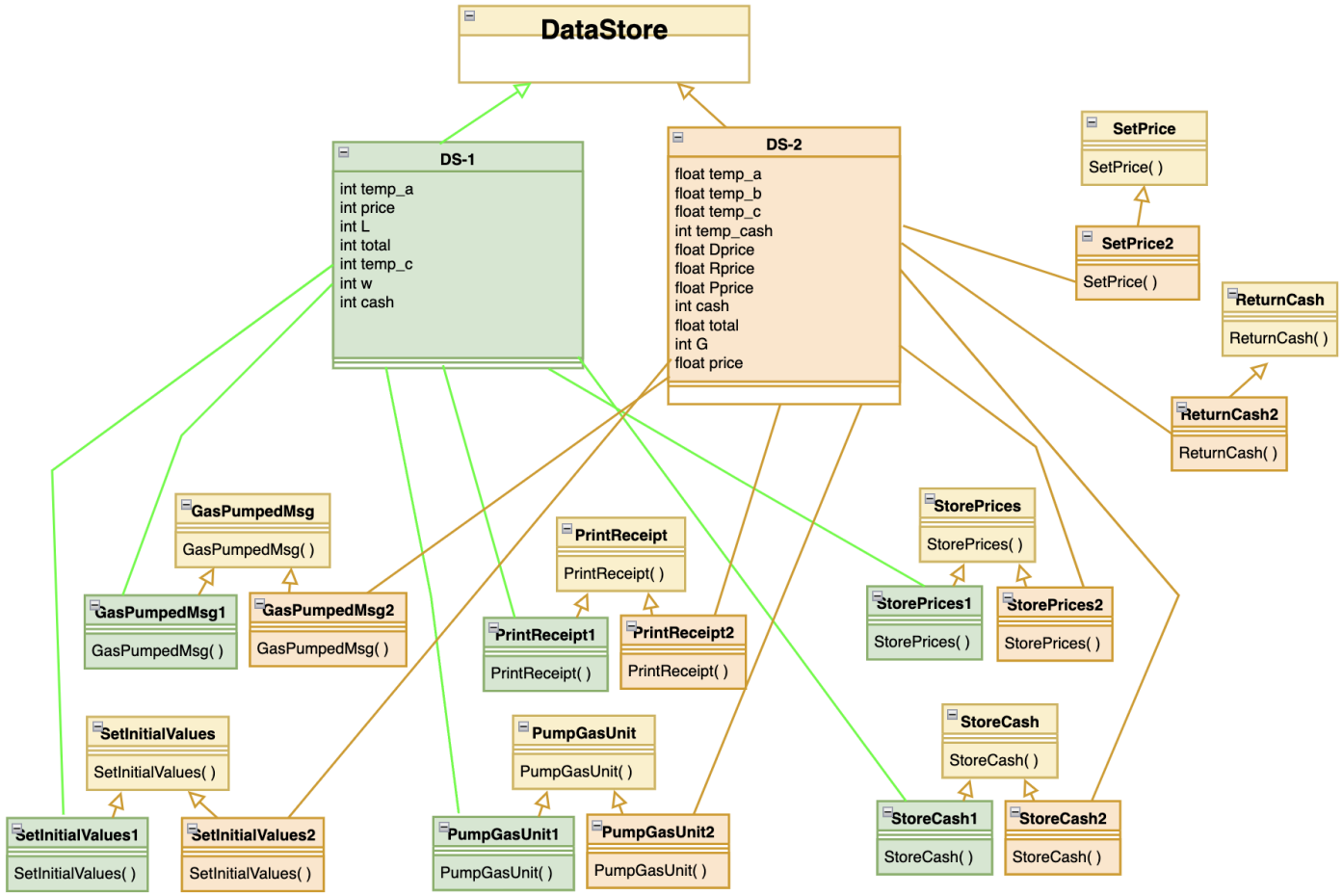
2. Class diagram(s) Contend.



2. Class diagram(s) Contend.



2. Class diagram(s) Contend.



3. For each class in the class diagram(s), you should:

a&b. Describe the purpose of the class, and responsibility of each operation supported by each class.

Class GasPump1

Purpose of the class: represents the domain specific for gas pump1. It stores data temporarily and invokes events on the MDA EFSM.

Responsibility of each operation:

Activate(int a): Activates GasPump1 with a specified price value, stores (a) value temporarily and invokes mda_efsm.

Start(): Starts the GasPump1 operation.

PayCash(int c): Allows payment with cash, setting the temporary cash value and payment type.

PayCredit(): Allows payment with a credit card, setting the payment type.

Reject(): Handles rejected payment and calls the reject method on MDA-EFSM.

Cancel(): Cancels the transaction, calls the cancel method on MDA-EFSM.

Approved(): Handles approved payment, calls the Approved method on MDA-EFSM.

StartPump(): Starts pumping fuel, calls the continue then Pump methods on MDA-EFSM.

Pump(): Continues pumping fuel, handling different scenarios based on payment type and available cash.It might call stop pump and Receipt on MDA-EFSM in case of cash finish.

StopPump(): Stops pumping fuel call stop pump and Receipt on MDA-EFSM..

Initialize(AbstractFactory af): initialize the object types like DataStore and MDA-EFSM and pass the correct type of the AbstractFactory to the MDA-EFSM.

Class GasPump2

Purpose of the class: GasPump2 encapsulates the functionality specific to GasPump2, managing temporary data storage and triggering events on the MDA-EFSM.

Responsibility of each operation:

Initialize(AbstractFactory af): Initializes GasPump2 by creating instances of DataStore and MDA-EFSM using the provided abstract factory.

Activate(float a, float b, float c): Activates GasPump2 with specific fuel prices, storing temporary values for fuel types A, B, and C, then triggering the activation event on the MDA-EFSM.

Start(): Initiates the operation of GasPump2 by starting the MDA-EFSM.

PayCash(int c): Enables payment with cash, setting the temporary cash value and indicating the payment type to the MDA-EFSM.

Cancel(): Cancels the ongoing transaction by invoking the cancel method on the MDA-EFSM.

Diesel(), Premium(), Regular(): Allows the selection of different fuel types (diesel, premium, regular) by triggering the respective events on the MDA-EFSM.

StartPump(): Starts pumping fuel by triggering the corresponding event on the MDA-EFSM.

PumpGallon(): Continues pumping fuel, ensuring sufficient cash for each gallon pumped, and stopping the pump if cash is insufficient. It interacts with the MDA-EFSM to continue or stop pumping accordingly.

Stop(): Stops pumping fuel by invoking the stop pump method on the MDA-EFSM.

Receipt(): Prints the receipt by calling the receipt method on the MDA-EFSM.

NoReceipt(): Specifies not to print the receipt by calling the corresponding method on the MDA-EFSM.

Class MDA-EFSM

Purpose of the class: MDA-EFSM Domain Dependant represents the Marstons-Drive-At-Express Fuel Service Machine, orchestrating the state transitions and actions *based on external events*.

Responsibility of each operation:

Initialize(AbstractFactory af): Initializes the FSM by creating instances of states and OutputProcessor using the provided abstract factory, sets initial state to S7, and assigns MDA-EFSM and OutputProcessor references for each state.

Activate(): Triggers activation action in the current state.

Start(): Initiates the operation by triggering start action in the current state.

PayType(int t): Sets payment type and triggers corresponding action in the current state.

Reject(): Handles rejected payment by invoking reject action in the current state.

Cancel(): Cancels the ongoing transaction by invoking cancel action in the current state.

Approved(): Handles approved payment by invoking approved action in the current state.

StartPump(): Initiates pumping fuel by invoking start pump action in the current state.

Pump(): Continues pumping fuel by invoking pump action in the current state.

StopPump(): Stops pumping fuel by invoking stop pump action in the current state.

SelectGas(int g): Selects the type of gas and triggers corresponding action in the current state.

Receipt(): Prints the receipt by invoking receipt action in the current state.

NoReceipt(): Specifies not to print the receipt by invoking no receipt action in the current state.

Continue(): Allows the FSM to continue its operation by invoking continued action in the current state.

setState(int s): Sets the current state of the FSM depending on s value.

(State Pattern.)

Class State(*I chosen to use Decentralized version*)

Purpose of the class: an abstract base class defining the interface for various states within the MDA-EFSM , facilitating state-specific behavior.

Responsibility of each operation:

Activate(): Abstract method to activate the state.

Start(): Abstract method to start the state.

PayType(int var1): Abstract method to set payment type in the state.

Reject(): Abstract method to handle rejected payment in the state.

Cancel(): Abstract method to cancel the ongoing transaction in the state.

Approved(): Abstract method to handle approved payment in the state.

StartPump(): Abstract method to start pumping fuel in the state.

Pump(): Abstract method to pump fuel in the state.

StopPump(): Abstract method to stop pumping fuel in the state.

SelectGas(int var1): Abstract method to select gas type in the state.

Receipt(): Abstract method to print receipt in the state.

NoReceipt(): Abstract method to specify not to print receipt in the state.

Continue(): Abstract method to continue operation in the state.

setOutputProcessor(OutputProcessor o): Sets the OutputProcessor for the state.

setMDAEFSM(MDA-EFSM mda): Sets the MDA-EFSM reference for the state to allow changing states.

Class S0

Purpose of the class: S0 represents the second state of the GasPump, it comes after S7 activate action to perform Start action.
Responsibility of each operation:

Activate(): No action is required in this state.
Approved(): No action is required in this state.
Cancel(): No action is required in this state.
Continue(): No action is required in this state.
NoReceipt(): No action is required in this state.
PayType(int var1): No action is required in this state.
Pump(): No action is required in this state.
Receipt(): No action is required in this state.
Reject(): No action is required in this state.
SelectGas(int var1): No action is required in this state.
***Start(): Displays the payment message and transitions to the next state (S1).**
StartPump(): No action is required in this state.
StopPump(): No action is required in this state.

Class S1

Purpose of the class: S1 represents the state after the transaction has started in the GasPump. It handles the payment type selection and transitions to the appropriate states accordingly.

Responsibility of each operation:

Activate(): No action is required in this state.

Approved(): No action is required in this state.

Cancel(): No action is required in this state.

Continue(): No action is required in this state.

NoReceipt(): No action is required in this state.

***PayType(int t): Handles the selection of payment type. If payment is by credit card, transitions to state S2, otherwise, stores cash, displays menu, sets payment type, and transitions to state S3 for cash payment.**

Pump(): No action is required in this state.

Receipt(): No action is required in this state.

Reject(): No action is required in this state.

SelectGas(int var1): No action is required in this state.

Start(): No action is required in this state.

StartPump(): No action is required in this state.

StopPump(): No action is required in this state.

Class S2

Purpose of the class: S2 represents the state after the credit card is approved in the GasPump. It handles actions after credit card approval and transitions to the next state accordingly.

Responsibility of each operation:

Activate(): No action is required in this state.

***Approved(): Prints approval message, call displays menu, sets payment type to credit, ejects card, and transitions to state S3.**

Cancel(): No action is required in this state.

Continue(): No action is required in this state.

NoReceipt(): No action is required in this state.

PayType(int var1): No action is required in this state.

Pump(): No action is required in this state.

Receipt(): No action is required in this state.

***Reject(): Displays rejection message, ejects card, and transitions to state S0.**

SelectGas(int var1): No action is required in this state.

Start(): No action is required in this state.

StartPump(): No action is required in this state.

StopPump(): No action is required in this state.

Class S3

Purpose of the class: S3 represents the state after the payment type is selected in the GasPump. It handles actions related to transaction cancellation, continuation and selectGas.

Responsibility of each operation:

Activate(): No action is required in this state.
Approved(): No action is required in this state.
***Cancel(): Displays cancel message, returns cash, and transitions to state S0.**
***Continue(): Transitions to state S4.**

NoReceipt(): No action is required in this state.
PayType(int var1): No action is required in this state.
Pump(): No action is required in this state.
Receipt(): No action is required in this state.
Reject(): No action is required in this state.
***SelectGas(int g): Sets the price for the selected gas type.**
Start(): No action is required in this state.
StartPump(): No action is required in this state.
StopPump(): No action is required in this state.

Class S4

Purpose of the class: S4 represents the state after the user selects to continue the transaction in the GasPump. It handles the action when the user starts the pump.

Responsibility of each operation:

Activate(): No action is required in this state.
Approved(): No action is required in this state.
Cancel(): No action is required in this state.
Continue(): No action is required in this state.
NoReceipt(): No action is required in this state.
PayType(int var1): No action is required in this state.
Pump(): No action is required in this state.
Receipt(): No action is required in this state.
Reject(): No action is required in this state.
SelectGas(int var1): No action is required in this state.
Start(): No action is required in this state.
***StartPump(): Prints a message, sets initial values, and transitions to state S5.**
StopPump(): No action is required in this state.

Class S5

Purpose of the class: S5 represents the state when the user is pumping gas in the GasPump. It handles actions related to pumping gas and transitioning to the next state when the pump is stopped.

Responsibility of each operation:

Activate(): No action is required in this state.
Approved(): No action is required in this state.
Cancel(): No action is required in this state.

Continue(): No action is required in this state.
NoReceipt(): No action is required in this state.
PayType(int var1): No action is required in this state.
***Pump(): Calls methods to pump gas unit and display gas pumped message.**
Receipt(): No action is required in this state.
Reject(): No action is required in this state.
SelectGas(int var1): No action is required in this state.
Start(): No action is required in this state.
StartPump(): No action is required in this state.
***StopPump(): Prints a message, stops the pump, and transitions to state S6.**

Class S6

Purpose of the class: S6 represents the state when the transaction is completed It handles actions related to returning cash, print and not printing receipt to the user and transitioning to state S0.

Responsibility of each operation:

Activate(): No action is required in this state.
Approved(): No action is required in this state.
Cancel(): No action is required in this state.
Continue(): No action is required in this state.
***NoReceipt(): Returns cash to the user and transitions to state S0.**
PayType(int var1): No action is required in this state.
Pump(): No action is required in this state.
Receipt(): Prints the receipt, returns cash to the user, and transitions to state S0.
Reject(): No action is required in this state.
SelectGas(int var1): No action is required in this state.
Start(): No action is required in this state.
StartPump(): No action is required in this state.
StopPump(): No action is required in this state.

Class S7

Purpose of the class: S7 represents the initial state of the Gas Pump, activated when the GasPump is activated. It stores prices and transitions to state S0.

Responsibility of each operation:

***Activate(): Stores prices and transitions to state S0.**
Approved(): No action is required in this state.

Cancel(): No action is required in this state.
Continue(): No action is required in this state.
NoReceipt(): No action is required in this state.
PayType(int var1): No action is required in this state.
Pump(): No action is required in this state.
Receipt(): No action is required in this state.
Reject(): No action is required in this state.
SelectGas(int var1): No action is required in this state.
Start(): No action is required in this state.
StartPump(): No action is required in this state.
StopPump(): No action is required in this state.

Class OutputProcessor

Purpose of the class: OutputProcessor is responsible for processing outputs based on the current state of the Gas Pump. It initializes various components and executes corresponding strategies to handle different actions.

Responsibility of each operation:

Constructor: Initializes the OutputProcessor.

Initialize(AbstractFactory var1): Initializes the OutputProcessor with an abstract factory, setting up each component using the factory's methods.

ejectCard(): Executes the strategy to eject the card.

setPayType(int t): Executes the strategy to set the payment type.

StorePrices(): Executes the strategy to store gas prices.

PayMsg(): Executes the strategy to display the payment message.

StoreCash(): Executes the strategy to store cash.

DisplayMenu(): Executes the strategy to display the menu.

RejectMsg(): Executes the strategy to display the rejection message.

SetPrice(int g): Executes the strategy to set the gas price.

SetInitialValues(): Executes the strategy to set initial values.

PumpGasUnit(): Executes the strategy to pump gas.

GasPumpedMsg(): Executes the strategy to display the gas pumped message.

PrintReceipt(): Executes the strategy to print the receipt.

CancelMsg(): Executes the strategy to display the cancellation message.

ReturnCash(): Executes the strategy to return cash.

(AbstractFactory Pattern)
Class AbstractFactory

Purpose of the class: AbstractFactory is responsible for creating components related to gas pump operations depending on strategies. It defines abstract factory methods for creating various components used in gas pump operations.

Responsibility of each operation:

getDataStore(): Abstract method to retrieve the data store.
CreateStorePrices(): Abstract method for creating a component for storing prices.
CreatePayMsg(): Abstract method for creating a component for displaying the pay message.
CreateStoreCash(): Abstract method for creating a component for storing cash.
CreateDisplayMenu(): Abstract method for creating a component for displaying the menu for diesel, regular, and premium fuel types.
CreateRejectMsg(): Abstract method for creating a component for displaying the rejection message.
CreateSetPrice(): Abstract method for creating a component for setting prices.
CreateSetInitialValues(): Abstract method for creating a component for setting initial values.
CreatePumpGasUnit(): Abstract method for creating a component for pumping gas units.
CreateGasPumpedMsg(): Abstract method for creating a component for displaying the gas pumped message.
CreatePrintReceipt(): Abstract method for creating a component for printing the receipt.
CreateCancelMsg(): Abstract method for creating a component for displaying the cancel message.
CreateReturnCash(): Abstract method for creating a component for returning cash.
CreateSetPayType(): Abstract method for creating a component for setting the payment type.
CreateEjectCard(): Abstract method for creating a component for ejecting the card.

Class ConcreteFactory1

Purpose of the class: ConcreteFactory1 implements the AbstractFactory interface to create components specifically for GasPump1. It provides methods for creating each component required for GasPump1 operations.

Responsibility of each operation:

getDataStore(): Method to create or retrieve the data store1 specific to GasPump1.

CreateStorePrices(): Method to retrieve the StorePrices1 component for GasPump1.

CreatePayMsg(): Method to retrieve the PayMsg1 component for GasPump1.

CreateStoreCash(): Method to retrieve the StoreCash1 component for GasPump1.

CreateRejectMsg(): Method to retrieve the RejectMsg1 component for GasPump1.

CreateSetPrice(): Method to retrieve the SetPrice1 component for GasPump1.

CreateSetInitialValues(): Method to retrieve the SetInitialValues1 component for GasPump1.

CreatePumpGasUnit(): Method to retrieve the PumpGasUnit1 component for GasPump1.

CreateGasPumpedMsg(): Method to retrieve the GasPumpedMsg1 component for GasPump1.

CreatePrintReceipt(): Method to retrieve the PrintReceipt1 component for GasPump1.

CreateCancelMsg(): Method to retrieve the CancelMsg1 component for GasPump1.

CreateSetPayType(): Method to retrieve the SetPayType1 component for GasPump1.

CreateEjectCard(): Method to retrieve the EjectCard1 component for GasPump1.

Class ConcreteFactory2

Purpose of the class: ConcreteFactory2 implements the AbstractFactory interface to create components specifically for GasPump2. It provides methods for creating each component required for GasPump2 operations.

Responsibility of each operation:

getDataStore(): Method to create or retrieve the data store2 specific to GasPump2.

CreateStorePrices(): Method to retrieve the StorePrices2 component for GasPump2.

CreatePayMsg(): Method to retrieve the PayMsg2 component for GasPump2.

CreateStoreCash(): Method to retrieve the StoreCash2 component for GasPump2.

CreateDisplayMenu(): Method to retrieve the DisplayMenu2 component for GasPump2.

CreateSetPrice(): Method to retrieve the SetPrice2 component for GasPump2.

CreateSetInitialValues(): Method to retrieve the SetInitialValues2 component for GasPump2.

CreatePumpGasUnit(): Method to retrieve the PumpGasUnit2 component for GasPump2.

CreateGasPumpedMsg(): Method to retrieve the GasPumpedMsg2 component for GasPump2.

CreatePrintReceipt(): Method to retrieve the PrintReceipt2 component for GasPump2.

CreateCancelMsg(): Method to retrieve the CancelMsg2 component for GasPump2.

CreateReturnCash(): Method to retrieve the ReturnCash2 component for GasPump2.

(Strategy Pattern)

Class: CancelMsg

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for canceling the transaction and displaying a message to the user.

Responsibility of operation:

abstract displayCancelMsg(): Displays a message indicating the cancellation of the transaction. (To be implemented in the derived class)

Class: DisplayMenu

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for displaying the menu options to the user.

Responsibility of operation:

abstract displayMenu(): Displays the available menu options to the user. (To be implemented in the derived class)

Class: EjectCard

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for handling the ejection of the card from the system.

Responsibility of operation:

abstract ejectCard(): Ejects the card from the system. (To be implemented in the derived class)

Class: GasPumpedMsg

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for displaying a message indicating the amount of gas pumped.

Responsibility of operation:

abstract displayGasPumpedMsg(): Displays a message indicating the amount of gas pumped. (To be implemented in the derived class)

Class: PayMsg

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for displaying a message prompting the user to pay.

Responsibility of operation:

abstract displayPayMsg(): Displays a message prompting the user to pay. (To be implemented in the derived class)

Class: PrintReceipt

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for handling the printing of the transaction receipt.

Responsibility of operation:

abstract printReceipt(): Prints the transaction receipt. (To be implemented in the derived class)

Class: PumpGasUnit

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for managing the pumping of gas units during a transaction.

Responsibility of operation:

abstract pumpGasUnit(): Controls the pumping of gas units during the transaction. (To be implemented in the derived class)

Class: RejectMsg

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for displaying a message to the user when their card is rejected.

Responsibility of operation:

abstract displayRejectMsg(): Displays a message to the user indicating card rejection. (To be implemented in the derived class)

Class: ReturnCash

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for handling the return of excess cash to the user.

Responsibility of operation:

abstract returnCash(): Returns excess cash to the user. (To be implemented in the derived class)

Class: SetInitialValues

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for setting initial values for the transaction.

Responsibility of operation:

abstract setInitialValues(): Sets initial values for the transaction. (To be implemented in the derived class)

Class: SetPayType

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for setting the payment type for the transaction.

Responsibility of operation:

abstract setPayType(int x): Sets the payment type for the transaction. (To be implemented in the derived class)

Class: SetPrice

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for setting the price of gas.

Responsibility of operation:

abstract setPrice(int t): Sets the price of gas. (To be implemented in the derived class)

Class: StoreCash

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for storing the cash amount received from the user.

Responsibility of operation:

abstract storeCash(): Stores the cash amount received from the user. (To be implemented in the derived class)

Class: StorePrices

Purpose: This is an abstract class that needs to be implemented by the derived class to provide functionality for storing the prices of gas.

Responsibility of operation:

abstract storePrices(): Stores the prices of gas. (To be implemented in the derived class)

(Strategies 1 for GasPump1)

Class: CancelMsg1

Purpose: This class extends the abstract CancelMsg class and provides a specific implementation for displaying a cancellation message.

Responsibility of operation:

void CancelMsg(): Displays a message indicating the cancellation of the transaction.

Class: EjectCard1

Purpose: This class extends the abstract EjectCard class and provides a specific implementation for ejecting the credit card from the system.

Responsibility of operation:

void EjectCard(): Ejects the credit card from the system.

Class: GasPumpedMsg1

Purpose: This class extends the abstract GasPumpedMsg class and provides a specific implementation for displaying a message indicating the amount of gas pumped.

Responsibility of operation:

void GasPumpedMsg(DataStore ds): Displays a message indicating the amount of gas pumped.

Class: PayMsg1

Purpose: This class extends the abstract PayMsg class and provides a specific implementation for displaying a message prompting the user to select a payment method.

Responsibility of operation:

void PayMsg(): Displays a message prompting the user to select a payment method.

Class: PrintReceipt1

Purpose: This class extends the abstract PrintReceipt class and provides a specific implementation for printing the transaction receipt.

Responsibility of operation:

void PrintReceipt(DataStore ds): Prints the transaction receipt.

Class: PumpGasUnit1

Purpose: This class extends the abstract PumpGasUnit class and provides a specific implementation for pumping gas units during a transaction.

Responsibility of operation:

void PumpGasUnit(DataStore ds): Controls the pumping of gas units during the transaction.

Class: RejectMsg1

Purpose: This class extends the abstract RejectMsg class and provides a specific implementation for displaying a message when the credit card is rejected.

Responsibility of operation:

void RejectMsg(): Displays a message indicating that the credit card is not approved.

Class: SetInitialValues1

Purpose: This class extends the abstract SetInitialValues class and provides a specific implementation for setting initial values for the transaction.

Responsibility of operation:

void SetInitialValues(DataStore ds): Sets initial values for the transaction.

Class: SetPayType1

Purpose: This class extends the abstract SetPayType class and provides a specific implementation for setting the payment type for the transaction.

Responsibility of operation:

void SetPayType(DataStore ds, int x): Sets the payment type for the transaction.

Class: StoreCash1

Purpose: This class extends the abstract StoreCash class and provides a specific implementation for storing the cash amount received from the user.

Responsibility of operation:

void StoreCash(DataStore ds): Stores the cash amount received from the user.

Class: StorePrices1

Purpose: This class extends the abstract StorePrices class and provides a specific implementation for storing the price of gas.

Responsibility of operation:

void StorePrices(DataStore ds): Stores the price of gas.

(Strategies 2 for GasPump2)

Class: CancelMsg2

Purpose: This class extends the abstract CancelMsg class and provides a specific implementation for displaying a cancellation message.

Responsibility of operation:

void CancelMsg(): Displays a message indicating the cancellation of the transaction.

Class: DisplayMenu2

Purpose: This class extends the abstract DisplayMenu class and provides a specific implementation for displaying menu options related to different types of fuel.

Responsibility of operation:

void DisplayMenu(): Displays the menu options for selecting fuel types.

Class: GasPumpedMsg2

Purpose: This class extends the abstract GasPumpedMsg class and provides a specific implementation for displaying a message indicating the amount of gas pumped in gallons.

Responsibility of operation:

void GasPumpedMsg(DataStore ds): Displays a message indicating the number of gallons of gas pumped.

Class: PayMsg2

Purpose: This class extends the abstract PayMsg class and provides a specific implementation for displaying payment options related to cash.

Responsibility of operation:

void PayMsg(): Displays payment options for only cash.

Class: PrintReceipt2

Purpose: This class extends the abstract PrintReceipt class and provides a specific implementation for printing the transaction receipt with details of the gallons pumped and the total amount.

Responsibility of operation:

void PrintReceipt(DataStore ds): Prints the transaction receipt with details of the gallons pumped and the total amount.

Class: PumpGasUnit2

Purpose: This class extends the abstract PumpGasUnit class and provides a specific implementation for pumping gas units in gallons during a transaction.

Responsibility of operation:

void PumpGasUnit(DataStore ds): Controls the pumping of gas units in gallons during the transaction.

Class: ReturnCash2

Purpose: This class extends the abstract ReturnCash class and provides a specific implementation for calculating and returning the excess cash to the user.

Responsibility of operation:

void ReturnCash(DataStore ds): Calculates and returns the excess cash to the user.

Class: SetInitialValues2

Purpose: This class extends the abstract SetInitialValues class and provides a specific implementation for setting initial values for the transaction related to gallons pumped and total amount.

Responsibility of operation:

void SetInitialValues(DataStore ds): Sets initial values for the transaction related to gallons pumped and total amount.

Class: SetPrice2

Purpose: This class extends the abstract SetPrice class and provides a specific implementation for setting the price of gas based on the selected fuel type.

Responsibility of operation:

void SetPrice(DataStore ds, int t): Sets the price of gas based on the selected fuel type.

Class: StoreCash2

Purpose: This class extends the abstract StoreCash class and provides a specific implementation for storing the cash amount received from the user.

Responsibility of operation:

void StoreCash(DataStore ds): Stores the cash amount received from the user.

Class: StorePrices2

Purpose: This class extends the abstract StorePrices class and provides a specific implementation for storing the prices of different fuel types permanently.

Responsibility of operation:

void StorePrices(DataStore ds): Stores the prices of different fuel types permanently.

(Data Stores)

Class DataStore

Purpose of the class: it used to be an umbrella for the two types of data store1 and data store2.

Responsibility of each operation:

There is no method in this class. I used public data types for the variables to be accessed from the outside.

Class DataStore1

Purpose of the class: It contains the data and the variables that need to operate GasPump1.

It has:

```
public int temp_c;      // Temporary variable c for the cash
public int temp_a;      // Temporary variable a
public int w;           // Payment type indicator (0 for cash, 1 for credit)
public int cash;        // Available cash
public float price;     // Price per Litter
public int L;           // Number of Litters pumped
public float total;     // Total amount of fuel pumped
```

Class DataStore2

Purpose of the class: It contains the data and the variables that need to operate GasPump2.

It has:

```
public float temp_Ta;   // Temporary variable to hold the value of a when activate the GP2
public float temp_Tb;   // Temporary variable to hold the value of b when activate the GP2
public float temp_Tc;   // Temporary variable to hold the value of c when activate the GP2
public int temp_cash;   // Temporary cash variable
public float Dprice;    // Diesel price
public float Rprice;    // Regular gasoline price
public float Pprice;    // Premium gasoline price
public int G;           // Number of Gallons pumped
public float total;     // Total amount of fuel pumped
```

(Driver)

Class App:

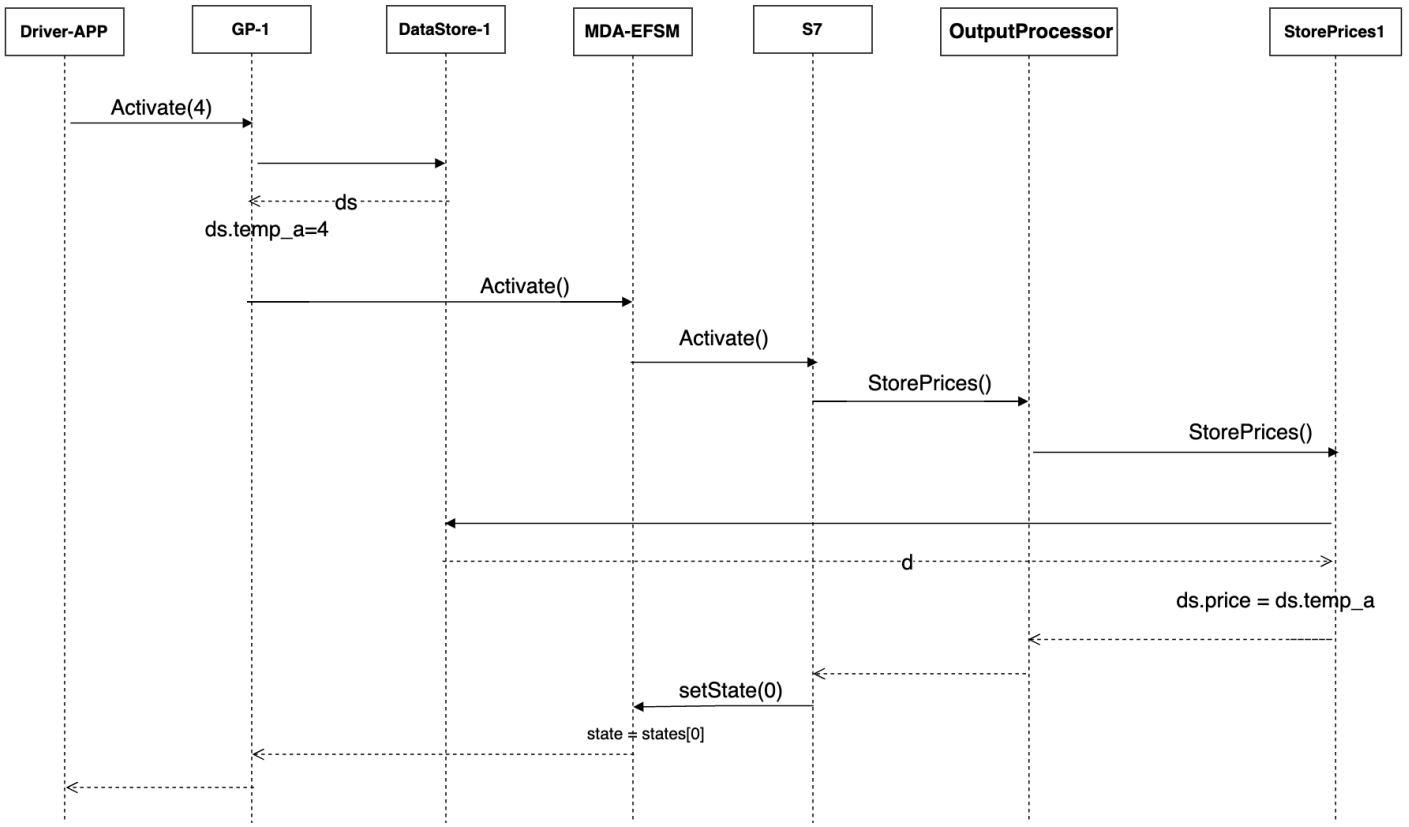
The App class serves as the entry point for managing gas pumps, offering a menu-driven interface for interacting with GasPump1 and GasPump2 objects. It presents a set of operations for each gas pump type and allows users to activate pumps, initiate transactions, select payment methods, and perform other pump-related actions through a command-line interface.

4. Dynamics. Provide two sequence diagrams for two Scenarios:

Scenario-I From (Page # 31 - page# 35)

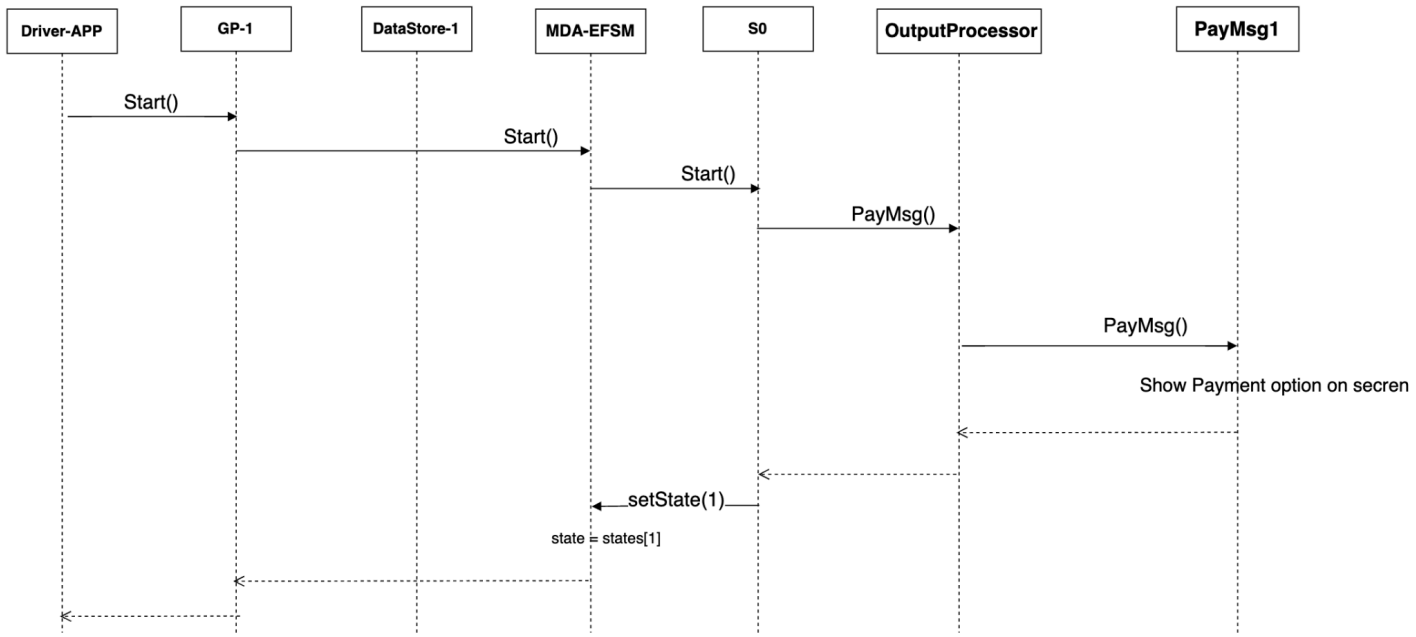
4.1) Scenario-I // *I split it to improve the readability.*

GasPump1 operation Activate(4)



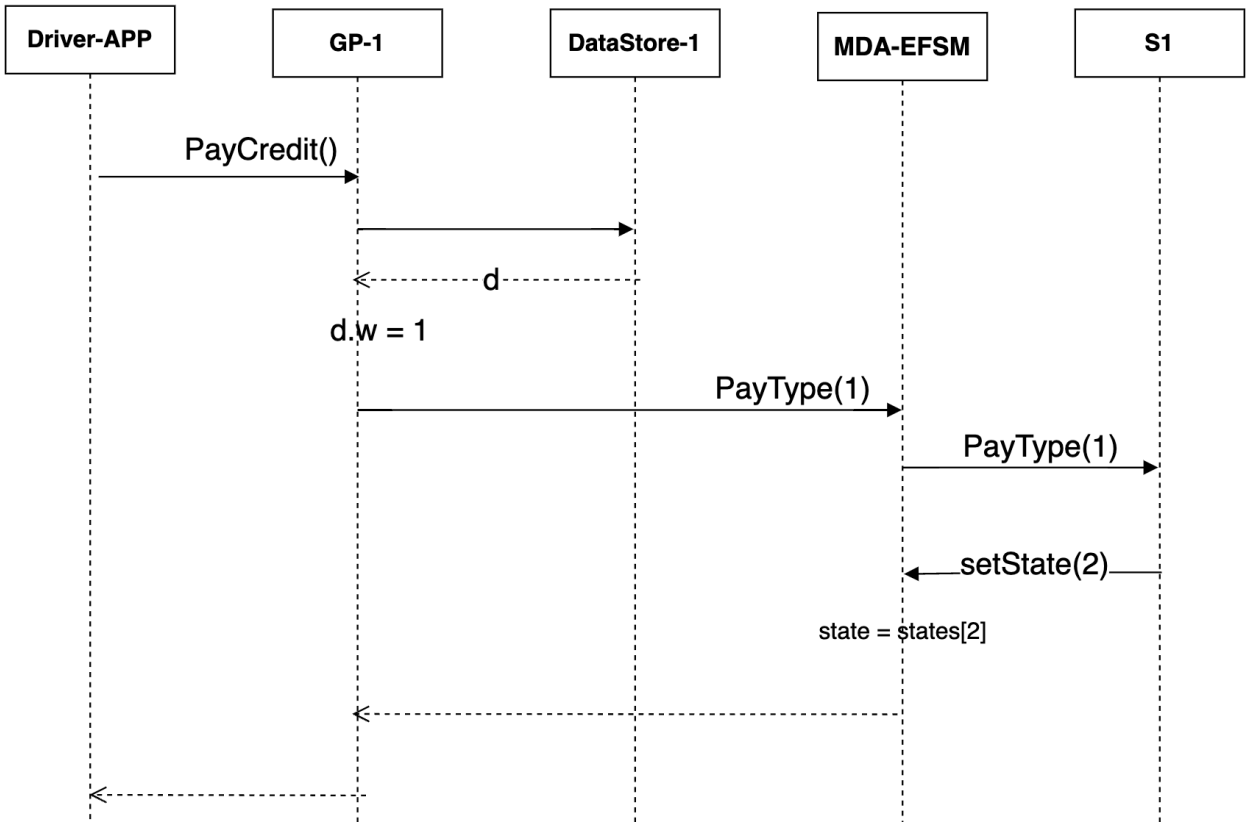
4.1) Scenario-I

GasPump1 operation Start()



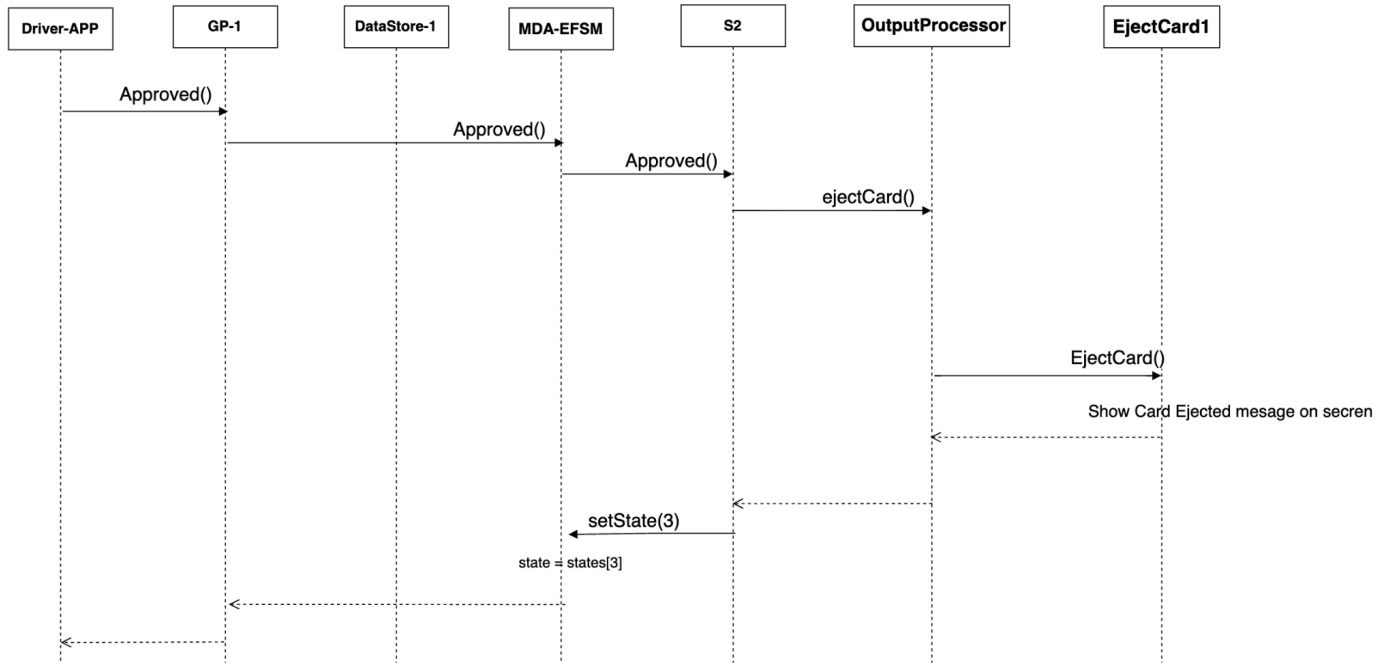
4.1) Scenario-I

GasPump1 operation operation PayCredit()



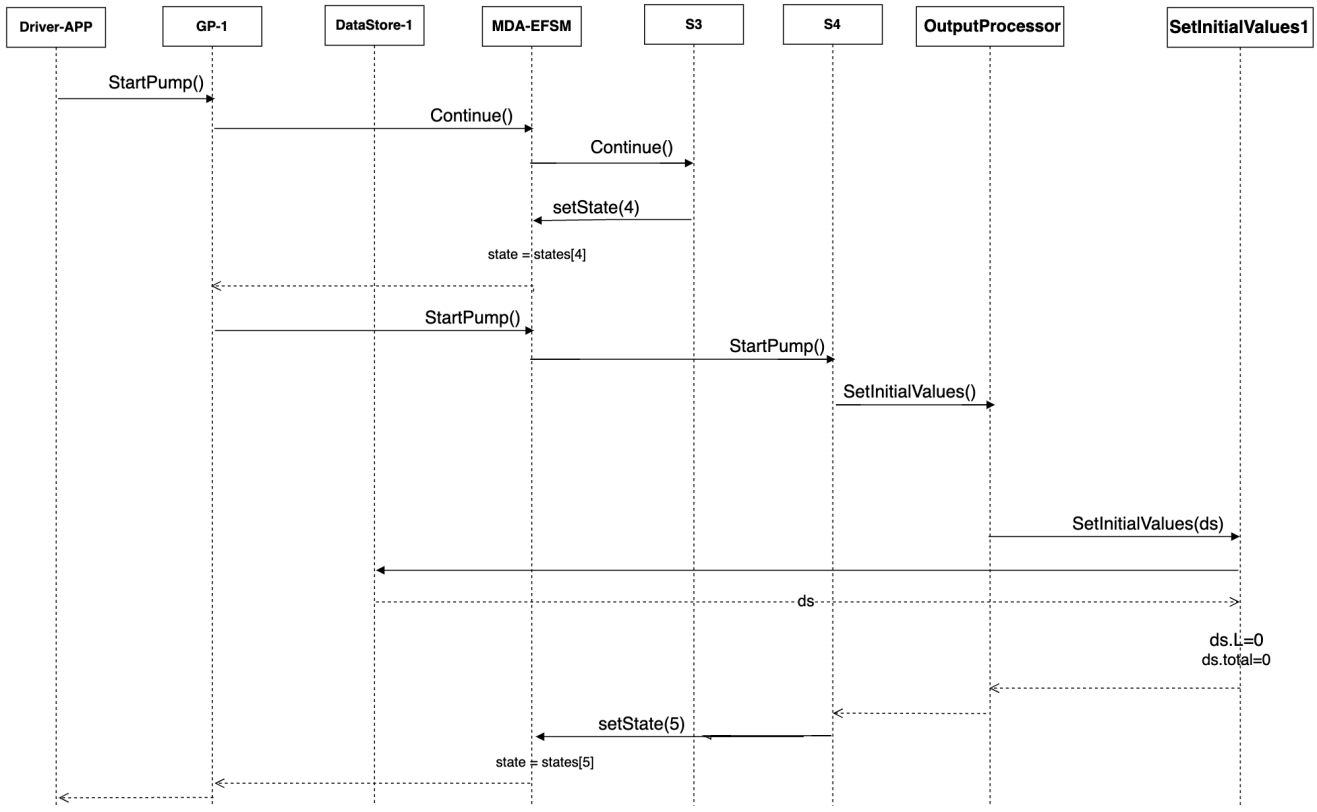
4.1) Scenario-I

GasPump1 operation operation Approved()



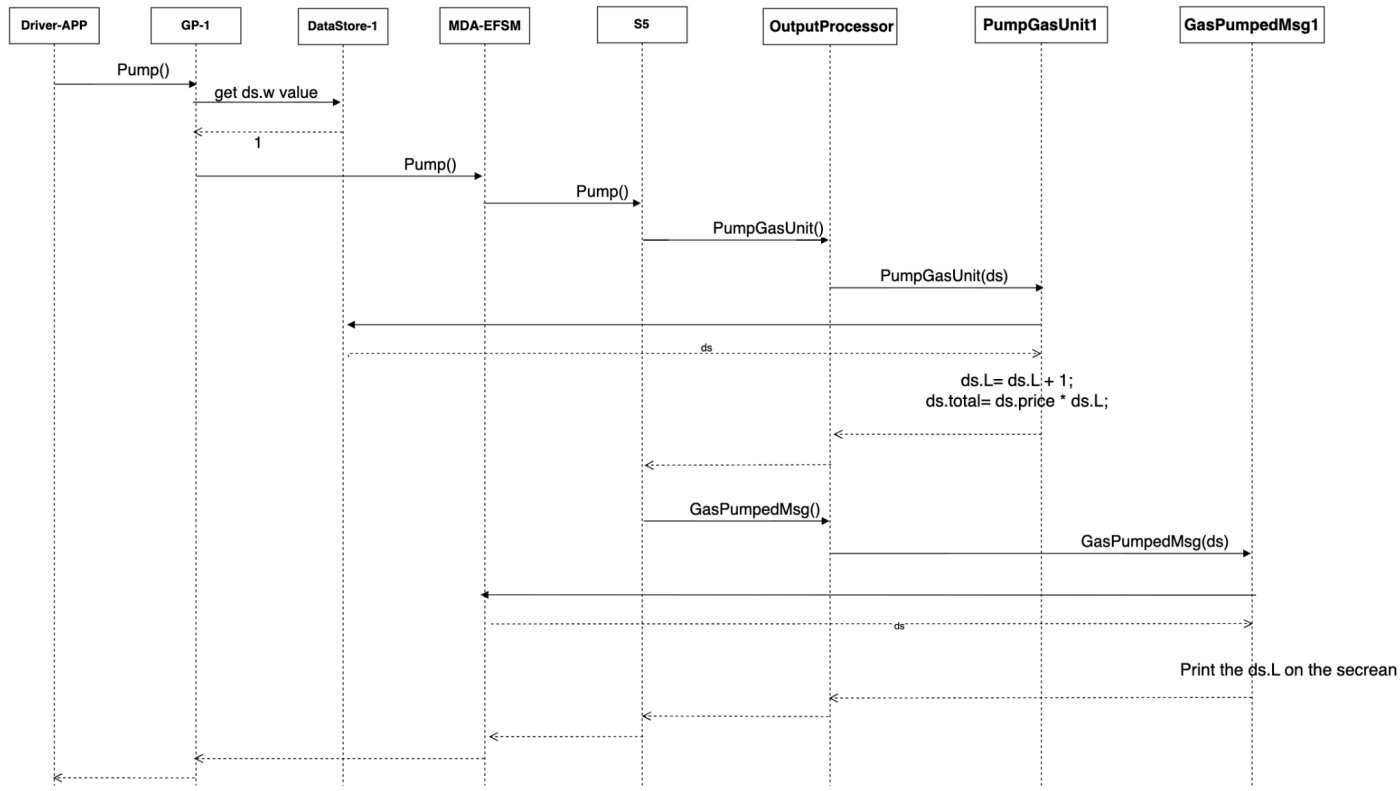
4.1) Scenario-I

GasPump1 operation operation StartPump()



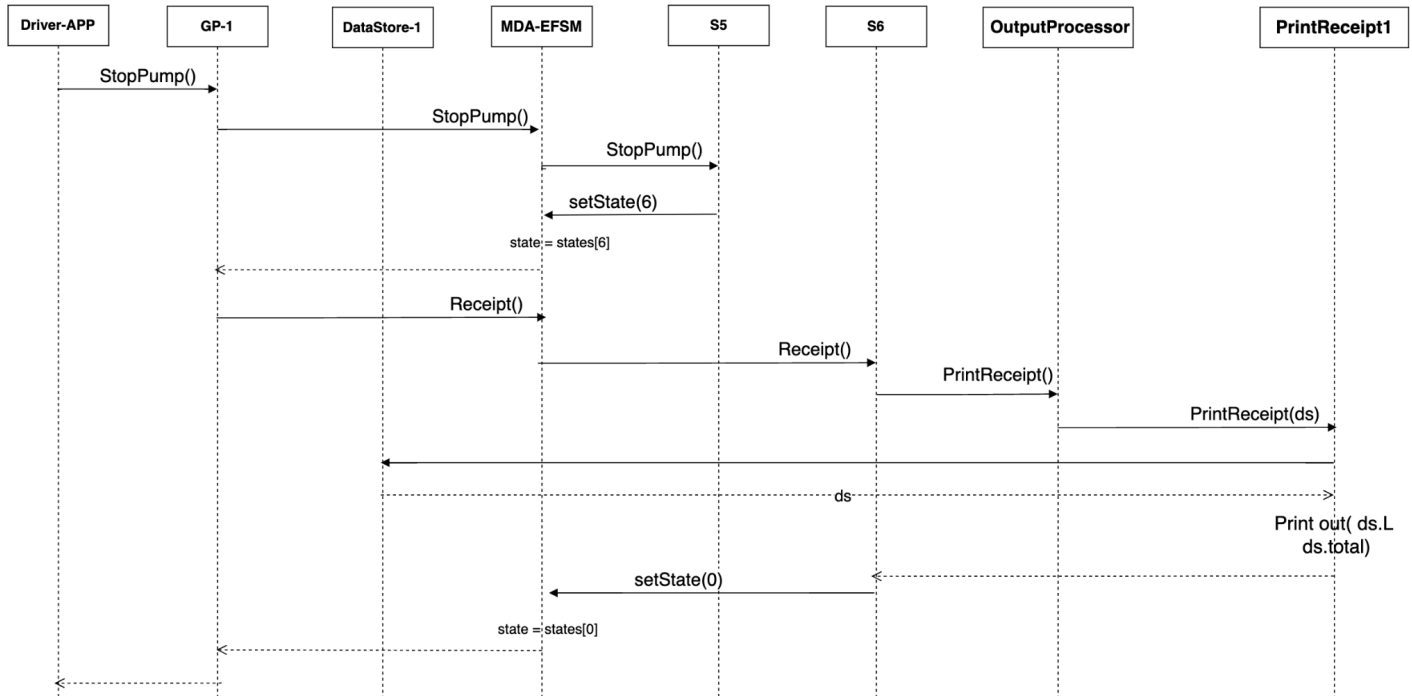
4.1) Scenario-I

GasPump1 operation operation Pump()



4.1) Scenario-I

GasPump1 operation operation StopPump()

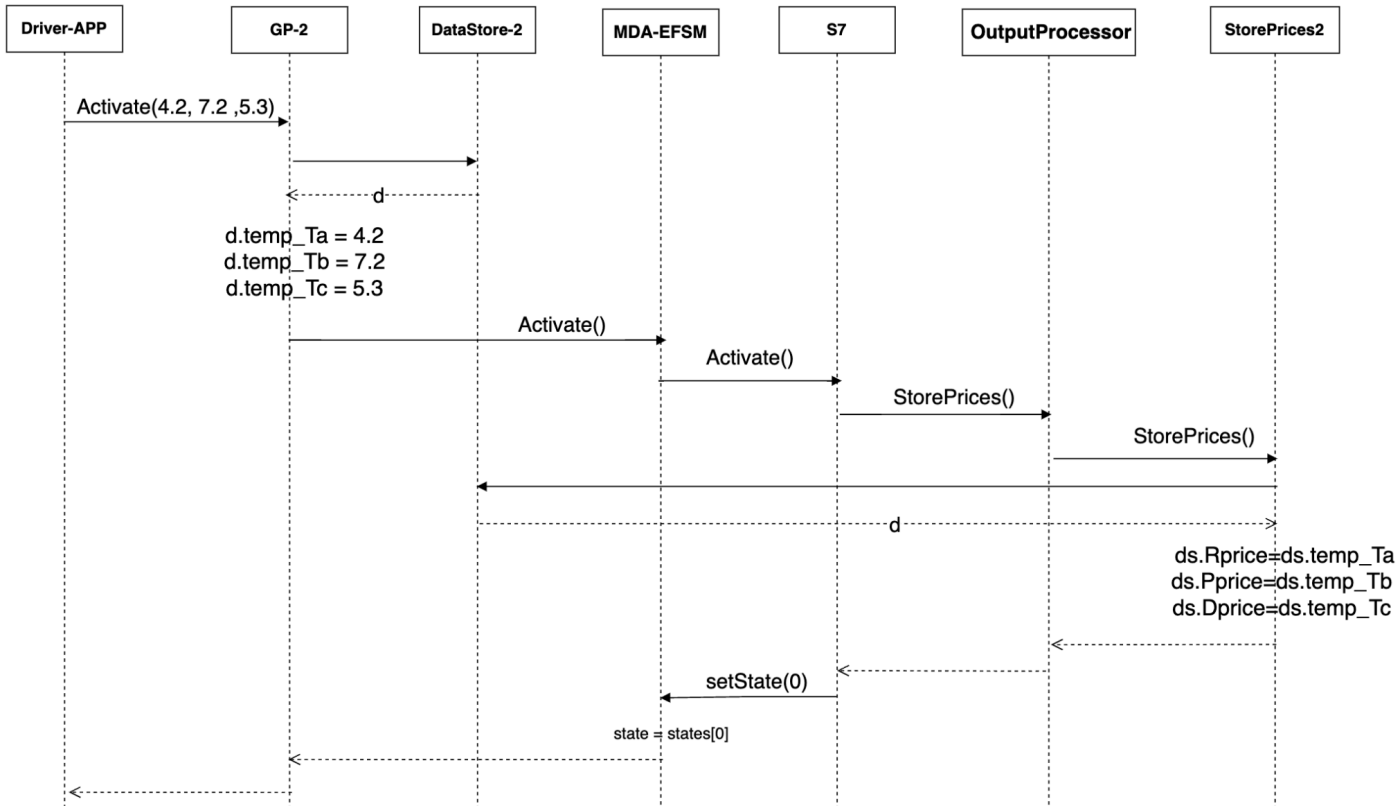


4. Dynamics. Provide two sequence diagrams for two Scenarios:

Scenario-II From (Page # 36 - page #40)

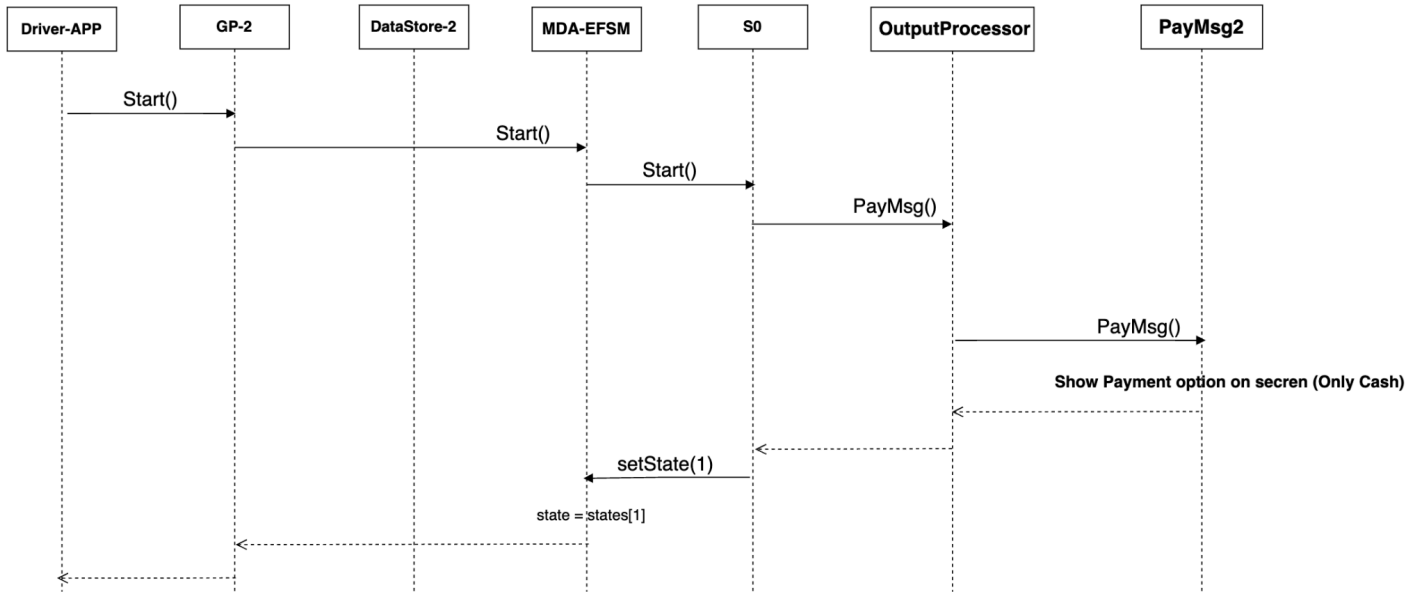
4.2) Scenario-II // *I split it to improve the readability.*

GasPump2 operation Activate(4.2, 7.2, 5.3)



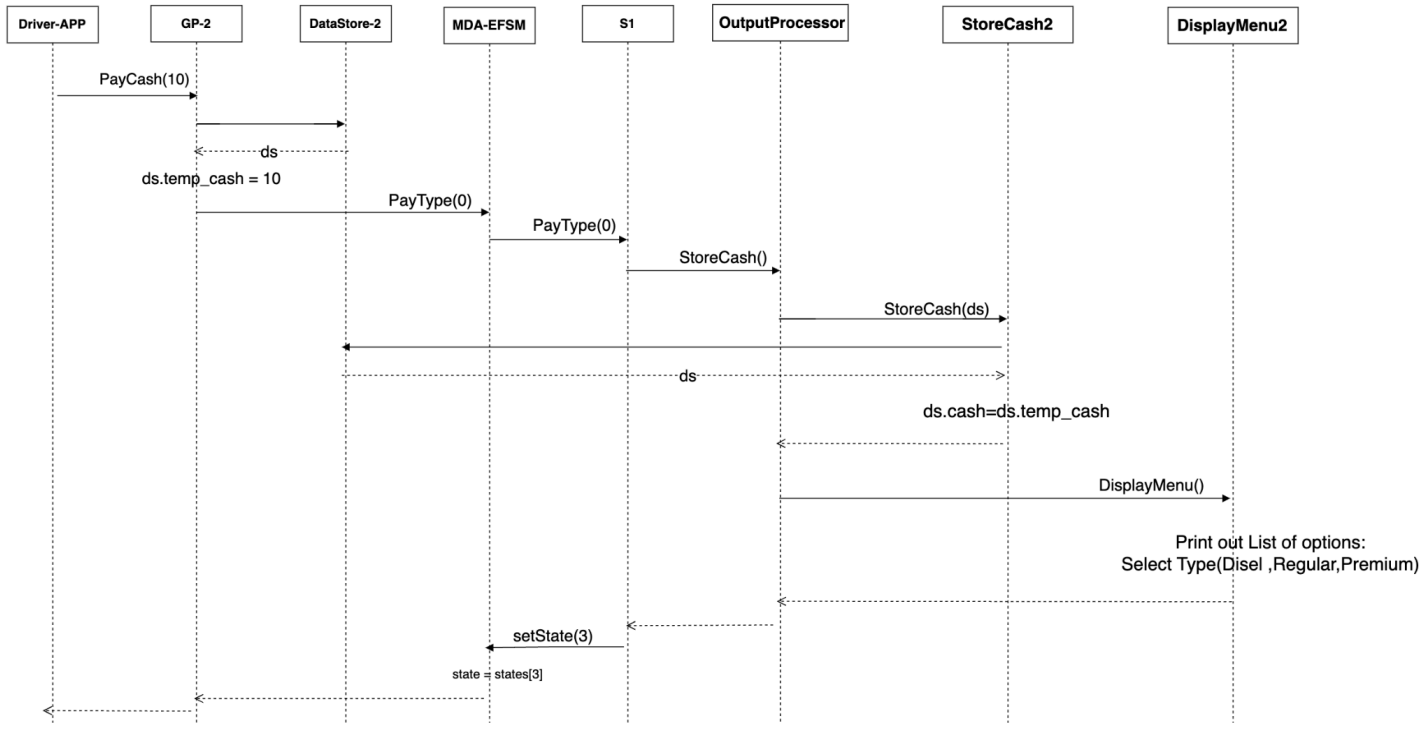
4.2) Scenario-II

GasPump2 operation Start()



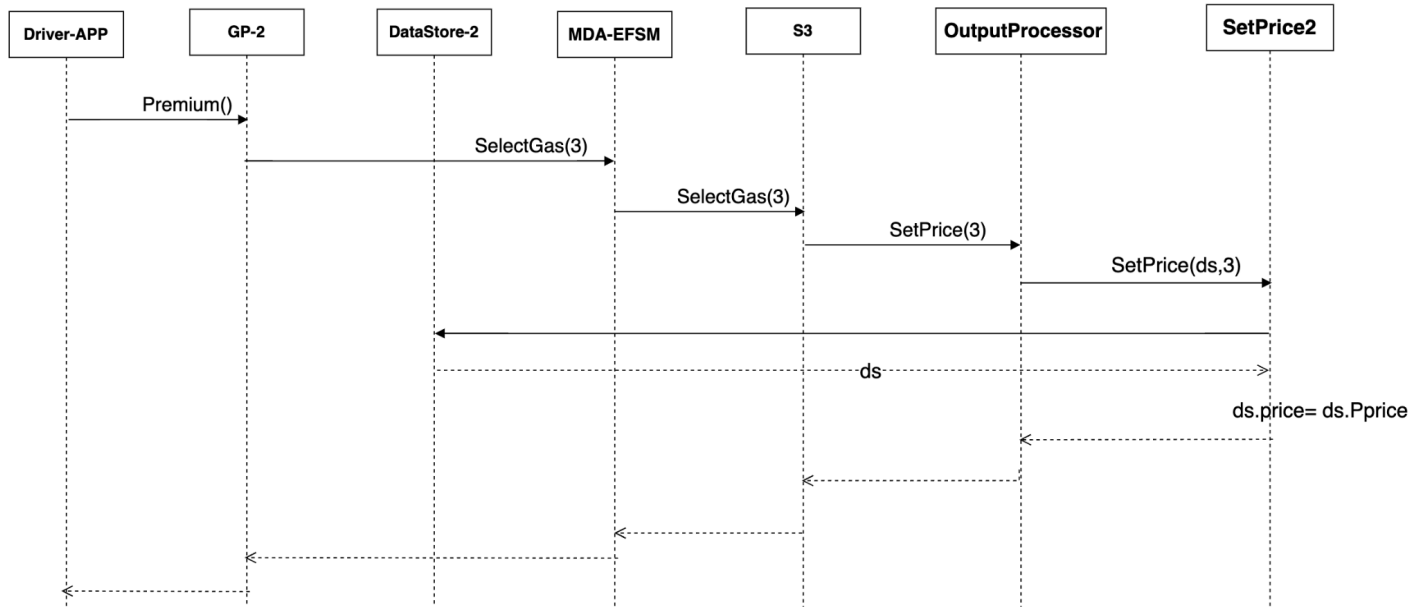
4.2) Scenario-II

GasPump2 operation PayCash(10)



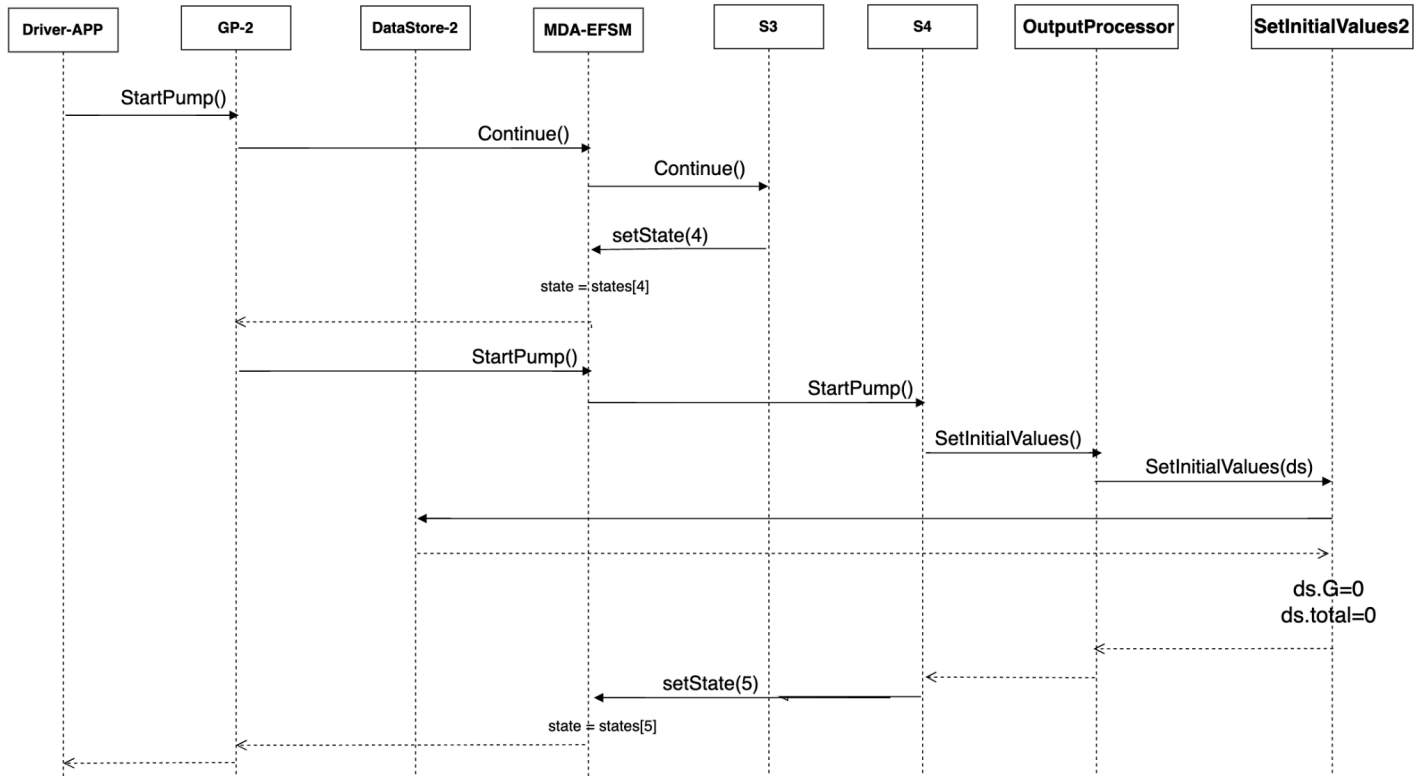
4.2) Scenario-II

GasPump2 operation Premium()



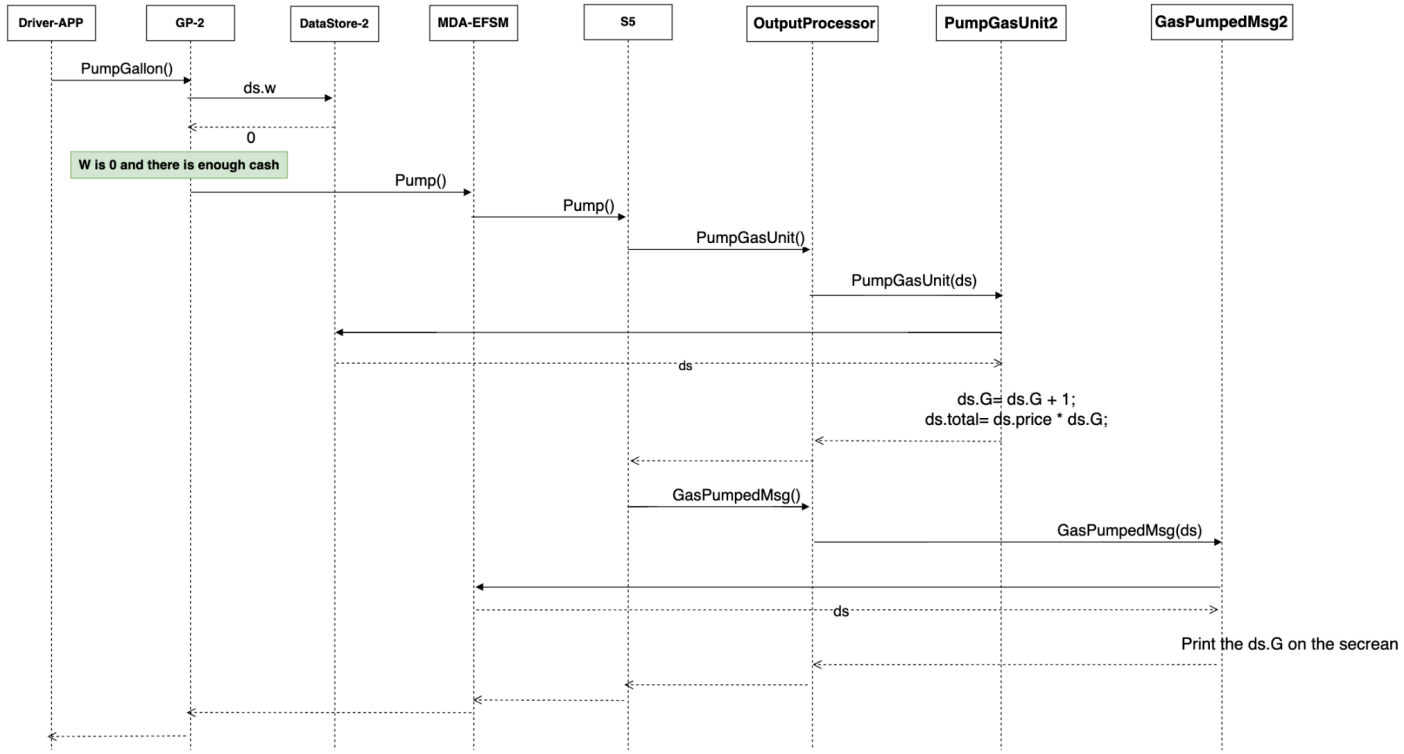
4.2) Scenario-II

GasPump2 operation StartPump()



4.2) Scenario-II

GasPump2 operation First PumpGallon()



4.2) Scenario-II

GasPump2 operations Second PumpGallon() & Receipt()