**Functions Description in each Class**

1. Class Driver (main method)

main()

the driver for the whole application, There are two internal variables: pump_type and input. The first one is to acquire the input from the user to select gasPump; The second one is to acquire the input from the user to select operations for each state.

2. Class GasPump1

*ip is pointer to IPGasPump1

GasPump1(AbstractFactory af)

the constructor , acquire the Input Processor object for gasPump1 through AbstractFactory object and set the Input Processor object to *ip;

Activate(Float a, Float b)

call the Activate function from the IPGasPump1 pointer.

Start()

call the Start function from the IPGasPump1 pointer.

PayCredit(), Approve(), Reject(), Regular(), Super(), Cancel(), StartPump(), PumpGallon(), StopPump()

these functions are similar.

3. Class GasPump2

*ip is pointer to IPGasPump2

GasPump1(AbstractFactory af)

the constructor , acquire the Input Processor object for gasPump2 through AbstractFactory object and set the Input Processor object to *ip;

Activate(int a , int b , int c )

call the Activate function from the IPGasPump2 pointer.

Start()

call the Start function from the IPGasPump2 pointer.

PayCash(Float cash), Cancel(), Regular(), Super(), Premium(), StartPump(), PumpLiter(), Stop(), Receipt(), NoReceipt()

these functions are similar.

4. Class AbstractFactory (Abstract Class)

getDataObj(), getInputProcessor, getStateMachine, getOutputProcessor()

all are virtual methods, to be implemented by the concrete gasPump factory classes

5. Class GasPump1_Factory

*data is pointer to DataForGasPump1;

*ip is pointer to new IPGasPump1;

*model is pointer to new StateMachine;

*op is pointer to new OutputProcessor;

GasPump1_Factory()

the constructor function to set the concrete actions, data and EFSM model for gasPump-1 through the input and output processor.

6. Class GasPump2_Factory

*data is pointer to DataForGasPump2;

*ip is pointer to new IPGasPump2;

*model is pointer to new StateMachine;

*op is pointer to new OutputProcessor;

GasPump2_Factory()

the constructor function to set the concrete actions, data and EFSM model for gasPump-2 through the input and output processor.

7. Class Data(Abstract)

no operation in this abstract class

8. Class DataForGasPump1

parameters:

| | |
|---|---|
| gasType | //The name of selected gas type |
| rPrice | //The price of regular gas |
| sPrice | //The price of super gas |
| price | //The price of selected gas |
| G | //The total number of gas that is pumped |
| total | //The total price of the pumped gas |
| temp_a, temp_b | // temporary variables |

getGasType(), setGasType(String gasType)

getters and setters methods for the parameters above;

9. Class DataForGasPump2

parameters:

| | |
|---|---|
| gasType | //The name of selected gas type |
| rPrice | //The price of regular gas |
| sPrice | //The price of super gas |
| pPrice | //The price of premium gas |
| price | //The price of selected gas |
| L | //The total number of gas that is pumped |
| total | //The total price of the pumped gas |
| cash | //The amount of cash from the user |
| temp_a, temp_b, temp_c | // temporary variables |

getGasType(), setGasType(String gasType)
getters and setters methods for the parameters above;

10. Class InputProcessor (Abstract )

*data is pointer to Data;
*model is pointer to StateMechine;

getData(), setData(Data data), getModel(), setModel(StateMachine model)
getters and setters methods for the parameters above;

11. Class IPGasPump1

Activate(float a, float b), Start(), PayCredit(), Approve(), Reject(), Regular(), Super(), Cancel(), StartPump(), PumpGallon(), StopPump()
All these operations supported by gaspump-1 and pseudo code given in MDAEFSM

12. Class IPGasPump2

Activate(int a , int b , int c ), Start(), PayCash(Float cash), Cancel(), Regular(), Super(), Premium(), StartPump(), PumpLiter(), Stop(), Receipt(), NoReceipt()
All these operations supported by gaspump-2 and pseudo code given in MDAEFSM

13. Class OutputProcessor

*data is pointer to Data;
*cancelMsg is pointer to AbstractCancelMsg;
*displayMenu is pointer to AbstractDisplayMenu;
*gasPumpedMsg is pointer to AbstractGasPumpedMsg;
*printReceipt is pointer to AbstractPrintReceipt;
*pumpGasUnit is pointer to AbstractPumpGasUnit;
*readyMsg is pointer to AbstractReadyMsg;

*rejectMsg is pointer to AbstractRejectMsg;
*returnCash is pointer to AbstractReturnCash;
*setInitialValues is pointer to AbstractSetInitialValues;
*setPrice is pointer to AbstractSetPrice;
*stopMsg is pointer to AbstractStopMsg;
*storeCash is pointer to AbstractStoreCash;
*storeData is pointer to AbstractStoreData;

CancelMsg()
Call the CancelMsg() methods from the CancelMsg object through the pointer

DisplayMenu(), GasPumpedMsg(), PayMsg(), PrintReceipt(), PumpGasUnit(), ReadyMsg(), RejectMsg(), ReturnCash(), SetInitialValues(), SetPrice(int g), StopMsg(), StoreCash(), StoreData()
these functions are similar.

getData(), setCancelMsg(AbstractCancelMsg cancelMsg), setData(Data data), setDisplayMenu(AbstractDisplayMenu displayMenu), setGasPumpedMsg(AbstractGasPumpedMsg gasPumpedMsg)
getters and setters methods for the parameters above;

14. Class StateMachine
*s is pointer to State object;
*LS is a list of State;
*op is pointer to OutputProcessor object;

void activate(), start(), payType(int t), approved(), reject(), selectGas(int g), cancel(), startPump(), pump(), stopPump(), receipt(), noReceipt()
All methods are call state machine methods as per operation selected on system.

15. Class State (Abstract)
*model is pointer to StateMechine;

activate(), start(), payType(int t), approved(), reject(), selectGas(int g), cancel(), startPump(), pump(), stopPump(), receipt(), noReceipt()
print error message as the initial operation for each events, so if the functions are not override in a state class, it means it is not supported in that state. If the concrete state class calls the event function that not override, the error message will display.

16. Class InitState

activate()
Activate gas-pump and store data of that

17. Class S0

    start()
    give payment methods available for gasPump

18. Class S1

    payType(int t)
    for different payment methods, do different actions.

19. Class S2

    approved()
    credit card is approved, display the options for the gasolines.

    reject()
    credit card is rejected, display the reject message.

20. Class S3

    selectGas(int g)
    according to the option(input g) from the user, set the unit price and gasType

    cancel()
    cancel transaction and give message; Return the remaining cash

21. Class S4

    startPump()
    start pump with initial values and give ready message

22. Class S5

    pump()
    calculate pump unit and give message of pumped gas

    stopPump()
    stop pump and give message of stop

23. Class S6

receipt()
Print receipt with total amount; return the remaining cash

noReceipt()
Return the remaining cash

24. Class AbstractCancelMsg (Abstract)

cancelMsg()
virtual methods, to be implemented by the concrete CancelMsg classeses. Use strategy design patterns, so that the specified algorithm can be independently from clients that use it.

25. Class CancelMsg1

cancelMsg()
the concrete implementations of cancelMsg function, called by gasPump-1, print cancel message.

26. Class CancelMsg2

cancelMsg()
the concrete implementations of cancelMsg function, called by gasPump-2, print cancel message.

all other abstract and concrete action classes are similar to class #25, #26 and #27.
……