Advanced Java Programming Assignment on

Gas Station Simulation

Submitted in partial fulfillment of the requirements
for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

Amity University
Noida, Uttar Pradesh

Under the guidance of:

R Dinita

Submitted By

Tanya Jain
A2372016006
6-CSE-3C

Acknowledgements

This project was a result of a great deal of support both moral and intellectual, I received from people around me. First of all I would like to thank the Almighty for blessing me with the determination required to successfully complete the term paper. I'm grateful to Mr Raz Dinita for teaching me the subject with utmost diligence and helping me throughout the course. I am also thankful to my programme leader Mrs Shipra Sarawat for her faith in my capabilities. I also place on record, my sincere gratitude to one and all, who directly or indirectly, have lent their helping hand in this venture.

Tanya JAIN

Abstract

Automated Gas Station Simulation is a console based application completed as an assignment of The Advanced Java Programming course as a partial fulfilment of the undergraduate degree. The application refers to a forecourt with 9 fuel pumps arranged equally over 3 lanes.

Every random number of seconds a vehicle is created and awaits to be fuelled for a random amount of time, that is regarded in the application as drivers' agitation. The fuelling starts when a vehicle waiting in the queue is automatically assigned to an available pump. On finishing, the vehicle leaves the forecourt and the amount of fuel dispense is recorded.

If the drivers' agitation time surpasses before the vehicle is sent to pump, the vehicle leaves the forecourt without fuelling and heads to another gas station about five miles down the line.

A pump can service only one vehicle at a time and the following global counters are calculated throughout the application.

- Total amount of fuel dispensed in litres in the app's lifetime
  - TOTAL_UNLEADED_LITRES_DISPENSED: int
  - TOTAL_DIESEL_LITRES_DISPENSED: int
  - TOTAL_LPG_LITRES_DISPENSED: int
- The amount of money these litres dispensed equate to
  AMOUNT_EQUIVALENCE_TO_LITRES_DISPENSED: double
- The 1% commission of litres dispensed earned by the employee
  COMMISSION_EARNED_BY_EMPLOYEE: float
- Number of vehicles fuelled
  NUMBER_OF_VEHICLES_FUELLED
- Number of vehicles left without fuelling
  NUMBER_OF_VEHICLES_LEFT_WITHOUT_FUELLING
- Detailed list of each fuelling transaction
  ArrayList<Transaction> transactions

Contents

Chapter 1

The Gas Station Simulation Guide

1.1 main() is invoked

The Gas Station simulation program starts with the creation of an object of the main class ‘TanyaPetrolPump’signifying the creation of the gasStation with 3 lanes containing 3 pumps each.

Okay

1.1.1 The *TanyaPetrolPump()* constructor is invoked.

The Class member ‘gasStationMap’is initialized to a 2d array of the ‘Pump’class.

1.1.2 The *initGasStationMap()* method is invoked.

Elements of the class member ‘gasStationMap’are further initialized as an object of class âĂŸPumpâĂŹ.

The *Pump()* constructor is invoked

The default value for the class member ‘*pumpUsage*’is set to false.

1.2 The startQueue() method is invoked.

- A new Timer variable ‘*vehicleAdditionTimer*’is initialized and started with a delay fetched from the value of the variable ‘*VEHICLE_GENERATION_TIMER_DELAY*’from the class ‘Config’.
- The vehicle is randomly generated from 1500 to 2200 miliseconds.
- The *generateVehicle()* method is invoked from the class âĂŸDataâĂŹ.

1.2.1 The *generateVehicle()* method is invoked

- Plate Number and Vehicle Type are generated on random depending on which the Fuel Type for the vehicle is randomly generated.
- Vehicle can be of the type: Car, Van or HGV.
- Fuel can be of the type: Unleaded, Diesel or LPG.
- A Car can function on any type of fuel.
- A Van can function on either LPG or Diesel.
- An HGV can only function on Diesel.
- Depending on the random generated data, a vehicle object is randomly generated from either of the derived classes of 'Car', 'Van'or 'HGV'of the base class 'Vehicle'.
- The *Vehicle(String, int)* constructor is invoked and the following variables are given a value.
- The vehicle is added to the *ArrayList<Vehicle> queue* in class Data.

The *Vehicle(String, int)* constructor is invoked

- *fuelTankCapacity: int* - The maximum capacity of the vehicle is set.
  - Car can contain a maximum of 40L of fuel.
  - Van can contain a maximum of 80L of fuel.
  - HGV can contain a maximum of 150L of fuel.
- *fuelLevel: int* - Random fuel level is set which can't be greater than the quarter of 'fuelTankCapacity'.
- *plateNumber: String* - The plate number of the vehicle is assigned.
- *vehicleWaitTime: int* - A random delay from 1000 to 2000 milliseconds is set for the *makeVehicleWait* timer via the variable '*VEHICLE_WAIT_TIME*'from class 'Config'.
- *MakeVehicleTimer: Timer* - the timer on completion removes the vehicle from queue if still in queue without fuelling and increments by 1 the counter '*NUMBER_OF_VEHICLES_LEFT_WITHOUT_FUELLING*'from class 'Config'. It is not supposed to repeat and is started as soon as the vehicle is created.

1.3 The emphstartCheckQueue() method is invoked.

- A new Timer variable ‘*checkQueueTimer*’is initialized and started with a delay fetched from the value of the variable ‘*CHECK_QUEUE_TIMER_DELAY*’from the class ‘Config’.
- The *checkQueue(gasStationMap)* method is invoked from the class ‘Data’.

1.3.1 The *checkQueue(gasStationMap)* method is invoked

- The statements further in the method aren’t executed if:
  - if all the pumps are busy (maximum possible vehicles are present at the pump)
  - if no vehicles are waiting to be fuelled in the queue.
- Otherwise, while there is a vehicle in queue:
  - Check which pump is empty.
  - Remove the vehicle from the queue.
  - Invoke the method *FuelVehicle(Vehicle)* from class ‘Pump’.

Invoke the method *FuelVehicle(Vehicle)*

- Send the vehicle to the pump, that is, add the vehilce details to *ArrayList<Vehicle> atPumpVehicles*.
- Set the pump usage to true.
- Calculate the litresDispensed
- Calculate the fuellingDuration, which is, litresDispensed / Config.DISPENSING_CAPABILITY
- A new Timer variable ‘fuellingTimer’is initialized with the delay as fuellingDuration.
- When the timer completes:
  - Vehicle is added to the *ArrayList<Vehicle> fuelledVehicles* and is removed from the *ArrayList<Vehicle> atPumpVehicles*.
  - *pumpUsage* is set to false.
  - *Config.NUMBER_OF_VEHICLES_FUELLED* is incremented by 1.
  - The method *generateTransaction(Vehicle , int)* is invoked.

        - ∗ Variable ‘cost’is calculated based on the litre of fuel dispensed to fuel the vehicle.
        - ∗ This cost is added to the global counter ‘ AMOUNT_EQUIVALENCE_TO_LITRES_DISPENSED’.
        - ∗ The global counter ‘COMMISSION_EARNED_BY_EMPLOYEE’is incremented as 1% of the litres dispensed.
        - ∗ The transaction is added to the *ArrayList<Transaction> transactions*.

1.3.2 The *startDrawUI()* method is invoked.

- A new Timer variable ‘drawUITimer’is initialized and started with a delay fetched from the value of the variable ‘DRAWUI_TIMER_DELAY’from the class ‘Config’.

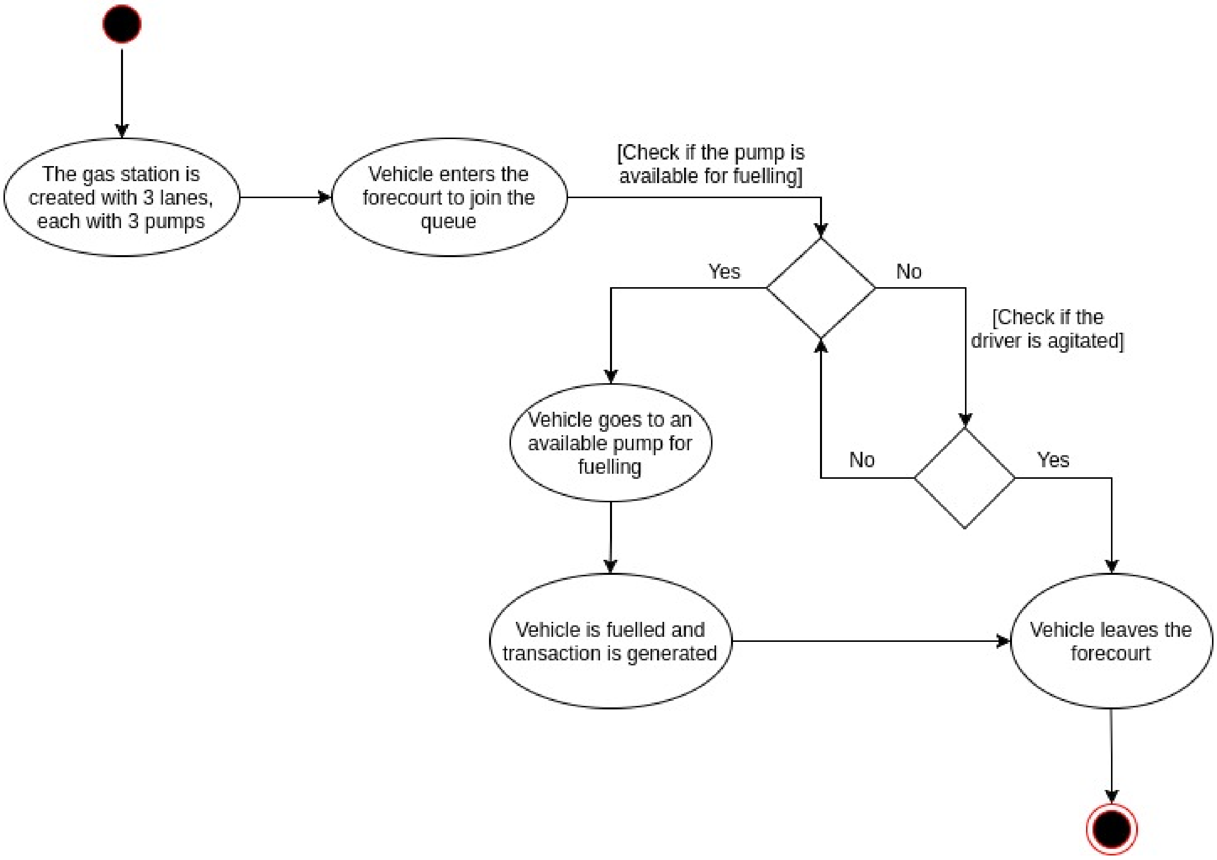
- On completion of the timer, the method drawUI() is invoked from the package tanyapetrolpumps.models.GUI and class ‘MainWindow’.

    - The method *drawQueue()* is invoked to display the list queue.
    - The method *drawStation()* is invoked to display the available pumps.
    - The method *drawInServiceVehicles()* is invoked to display the list atPumpVehicles.
    - The method *drawTransactions()* is invoked to display the list transactions.

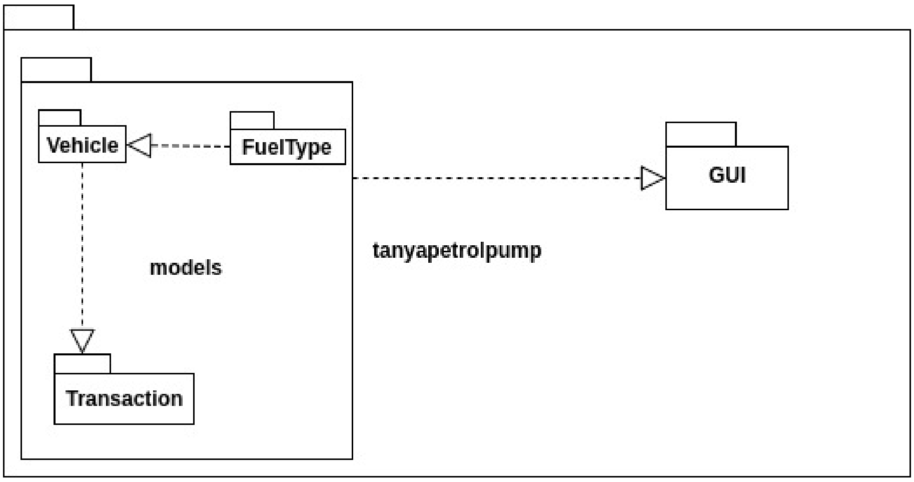
Chapter 2

Unified Modelling Language

2.1 Activity Diagram

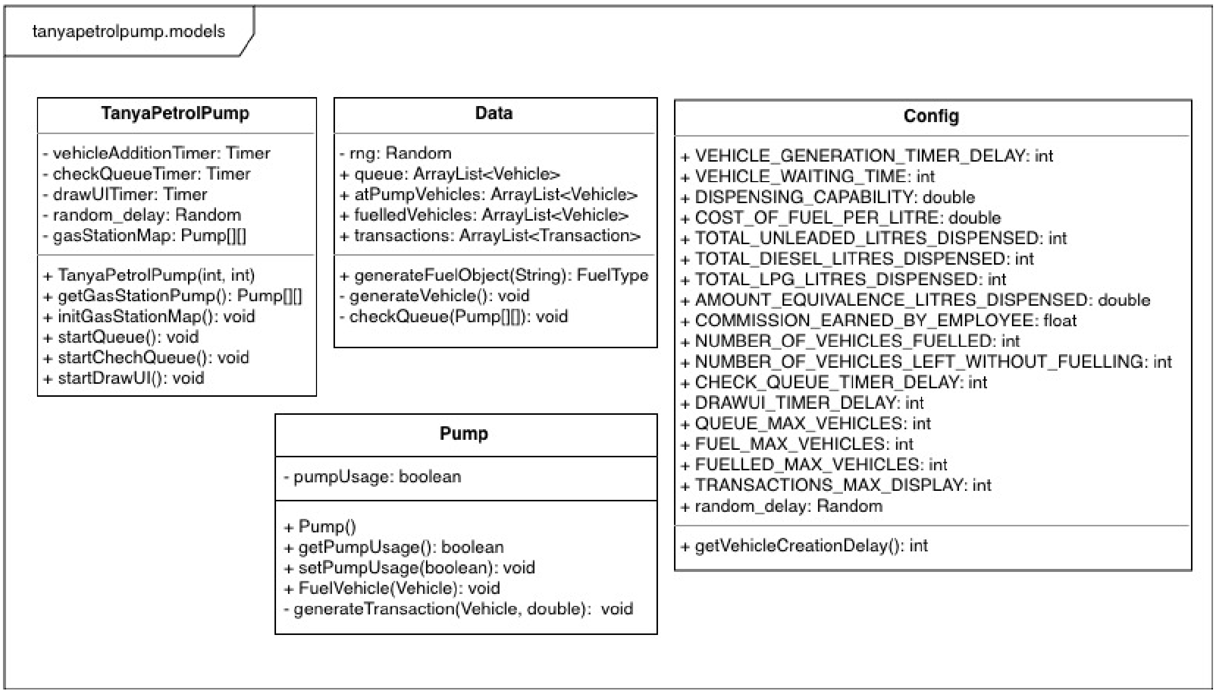

The gas station is created with 3 lanes, each with 3 pumps
Vehicle enters the forecourt to join the queue
[Check if the pump is available for fuelling]
Yes
No
[Check if the driver is agitated]
Vehicle goes to an available pump for fuelling
No
Yes
Vehicle is fuelled and transaction is generated
Vehicle leaves the forecourt

2.2 Package Diagram

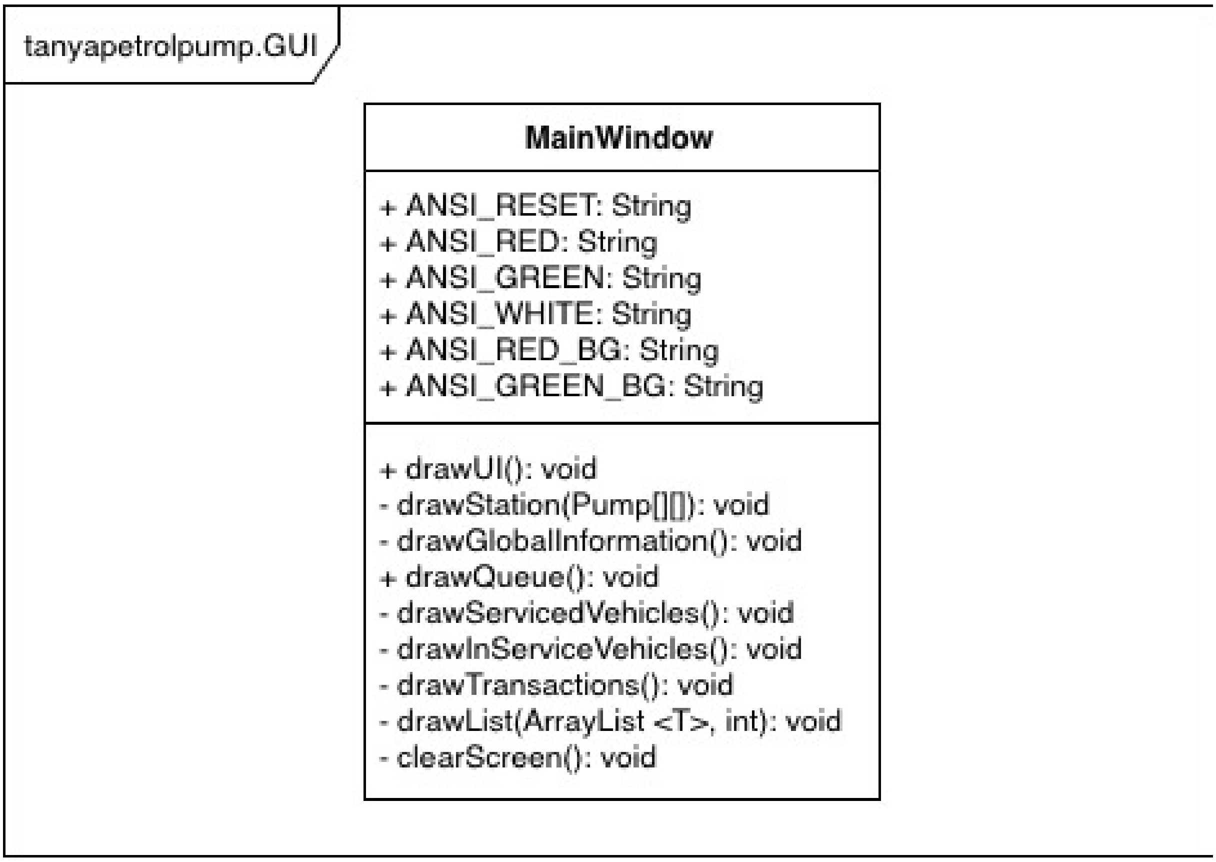


2.3 Class Diagrams

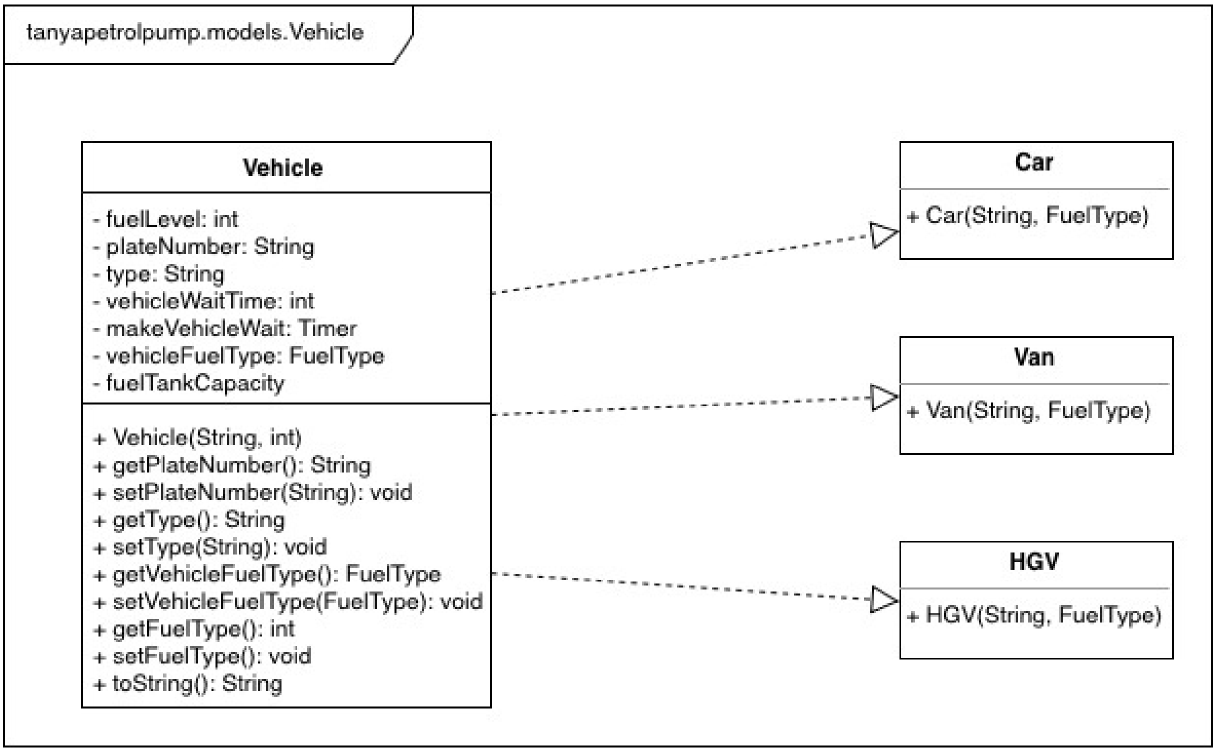
2.3.1 package: tanyapetrolpump.models


tanyapetrolpump.models
TanyaPetrolPump
- vehicleAdditionTimer: Timer
- checkQueueTimer: Timer
- drawUITimer: Timer
- random_delay: Random
- gasStationMap: Pump[][]
+ TanyaPetrolPump(int, int)
+ getGasStationPump(): Pump[][]
+ initGasStationMap(): void
+ startQueue(): void
+ startChechQueue(): void
+ startDrawUI(): void
Data
- rng: Random
+ queue: ArrayList<Vehicle>
+ atPumpVehicles: ArrayList<Vehicle>
+ fuelledVehicles: ArrayList<Vehicle>
+ transactions: ArrayList<Transaction>
+ generateFuelObject(String): FuelType
- generateVehicle(): void
- checkQueue(Pump[][]): void
Config
+ VEHICLE_GENERATION_TIMER_DELAY: int
+ VEHICLE_WAITING_TIME: int
+ DISPENSING_CAPABILITY: double
+ COST_OF_FUEL_PER_LITRE: double
+ TOTAL_UNLEADED_LITRES_DISPENSED: int
+ TOTAL_DIESEL_LITRES_DISPENSED: int
+ TOTAL_LPG_LITRES_DISPENSED: int
+ AMOUNT_EQUIVALENCE_LITRES_DISPENSED: double
+ COMMISSION_EARNED_BY_EMPLOYEE: float
+ NUMBER_OF_VEHICLES_FUELLED: int
+ NUMBER_OF_VEHICLES_LEFT_WITHOUT_FUELLING: int
+ CHECK_QUEUE_TIMER_DELAY: int
+ DRAWUI_TIMER_DELAY: int
+ QUEUE_MAX_VEHICLES: int
+ FUEL_MAX_VEHICLES: int
+ FUELLED_MAX_VEHICLES: int
+ TRANSACTIONS_MAX_DISPLAY: int
+ random_delay: Random
+ getVehicleCreationDelay(): int
Pump
- pumpUsage: boolean
+ Pump()
+ getPumpUsage(): boolean
+ setPumpUsage(boolean): void
+ FuelVehicle(Vehicle): void
- generateTransaction(Vehicle, double): void

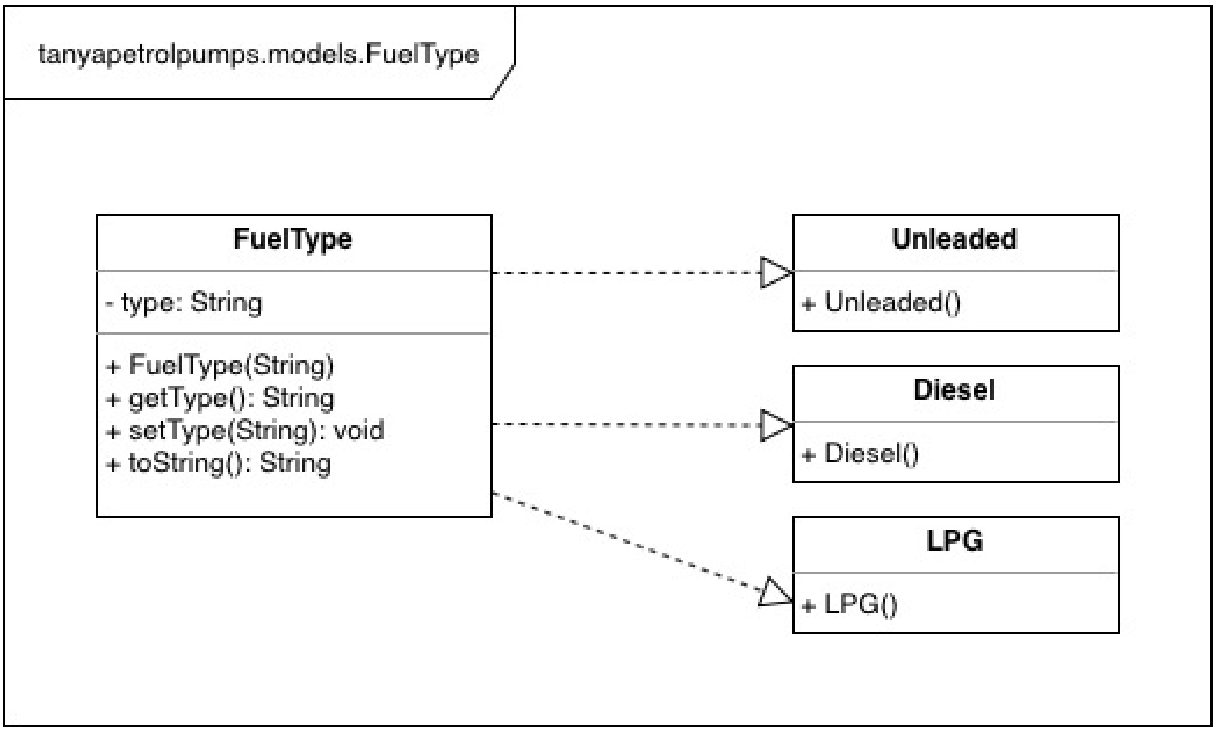
2.3.2 package: tanyapetrolpump.models.GUI



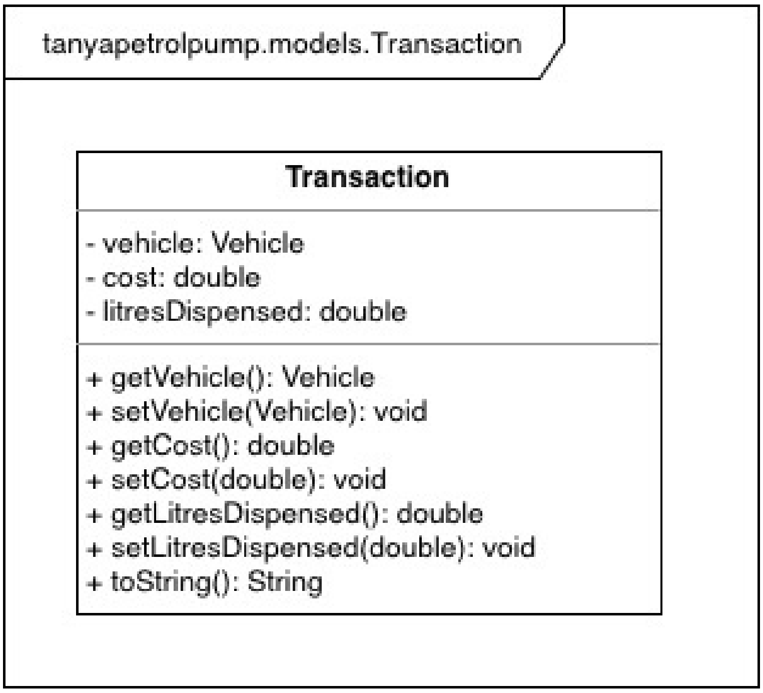
2.3.3 package: tanyapetrolpump.models.Vehicle



2.3.4 package: tanyapetrolpump.models.FuelType



2.3.5 package: tanyapetrolpump.models.Transaction


tanyapetrolpump.models.Transaction
Transaction
- vehicle: Vehicle
- cost: double
- litresDispensed: double
+ getVehicle(): Vehicle
+ setVehicle(Vehicle): void
+ getCost(): double
+ setCost(double): void
+ getLitresDispensed(): double
+ setLitresDispensed(double): void
+ toString(): String

Chapter 3

Testing

3.1 Test Plan

#	Test Purpose	Expected Outcome
1	Creation of Vehicles	Vehicles arrive randomly between 1000 milliseconds and 2000 milliseconds. Vehicles are randomly chosen between Car, Van, or HGV. Cars has 40 liter capacity with either Diesel, LPG, or unleaded as fuel type. Vans has 80 liter capacity with either Diesel or LPG fuel type. HGVs have 150 liter capacity with only diesel fuel type. Tank level are randomly filled below a quarter of the tank capacity. Vehicles are added to the waiting queue if no pumps are available and if there are less than 5 cars in the waiting queue.

2	Servicing of Vehicles	Busy pump adds fuel to the vehicle at every time interval. Total dispensed fuel is updated after every fuelling. Pump becomes available once the vehicle is fuelled. When a lane is available, it will get the next vehicle from the waiting queue. When a vehicle is agitated from the waiting queue, they leave the forecourt (queue).
3	Calculation of Counters	Number of serviced vehicles are updated for every vehicle that completes. Number of agitated vehicles are updated for every vehicle that failed to be serviced. Total dispensed fuel is correctly calculated. Total profit is calculated based on the dispensed fuel. Commission rate is calculated List of all fuel transactions are displayed.

3.2 Test Log

There are no inputs as it is an automated simulation with randomly generated elements.

#	**Actual Outcome**	**Comments**
1	Same as Expected	All OK
2	Same as Expected	All OK
3	Same as Expected	All OK