Pizza Delivery Application

Your client runs a pizza delivery service in Galway and has commissioned you to write a Java application to keep track of their delivery fleet. The system will allow a user to:

- To view all vehicles in the system.
- Manage vehicles in the system, i.e. add and remove vehicles.
- Record daily delivery information for each vehicle.
- View total running costs for each vehicle and the delivery fleet.

This is an opportunity to build an application using the Java concepts, designs and best practices that we have been covering this semester.

The application has three types of ***DeliveryVehicle***: ***DeliveryBike***, ***DeliveryScooter*** & ***DeliveryCar***. Each instance of a delivery vehicle has a number of private member variables to keep track of vital vehicle information.

- *_registrationNumber* **String** unique ID to identify a vehicle
- *_vehicleType* **String** name of each vehicle type
- *_engineSize* **int** engine capacity of the vehicle
- *_daysInService* **int** total number of days vehicle is in service
- *_milesCovered* **double** total mile covered by the vehicle.
- *_deliveriesMade* **int** total number of deliveries made.
- *_fuelCostPerMile* **double** fuel cost per mile.
- *_annualMaintenenceCost* **double** annual maintenance fee

Wont be set for

The above fields are common across all vehicles. Use the concept of inheritance you have been learning in the course to group classes appropriately. Figure 1 is an example of such a design that can be used in this situation.
Note* There can be no concrete instances of a *DeliveryVehicle*.

DeliveryBike class: As an incentive to reduce pollution, the government has introduced a grant of €100 per year for each bicycle the company uses for their deliveries. Record the value of this grant in the file below.

- *_noEmissionsGrant* **double** annual government grant.

DeliveryScooter class: vehicles do not qualify for the no emissions grant but due to their small engine size of (50cc) they are exempt from annual road tax and have a fuel cost per mile of (€0.08).

DeliveryCar class: vehicles are subject to annual road tax and a fuel per mile cost. The table below shows a breakdown of the yearly tax and fuel per mile costs based on engine size of the car.

- *_yearlyTax* **double** annual road tax.

DeliveryCar	1000cc	1200cc	1300cc	1600cc	2000cc
Fuel Cost (per mile)	€0.10	€0.115	€0.135	€0.14	€0.16
Road Tax (yearly)	€150	€250	€350	€450	€500

In addition, each vehicle has an associated average annual cost of running based on the average servicing and maintenance cost of each ***DeliveryVehicle*** type per annum.

- ***DeliveryBike***: ***€ 200***
- ***DeliveryScooter:*** ***€ 1,000***
- ***DeliveryCar:*** ***€2,000***

Hint* All annual taxes and grants should be set on object creation (Constructor) along with the fuel cost per mile where applicable. To ensure high cohesion, the **DeliveryCar** class should have a ***setTax(int engineSize)*** and ***setFuelCost(int engineSize)*** methods to correctly assign the appropriate road tax and fuel per mile cost to each instance of the class.

Group Assignment: PART A – Class Hierarchy

Develop the application's class hierarchy in BlueJ. Ensure that there are *accessor/ mutator* methods for each class variable as appropriate.

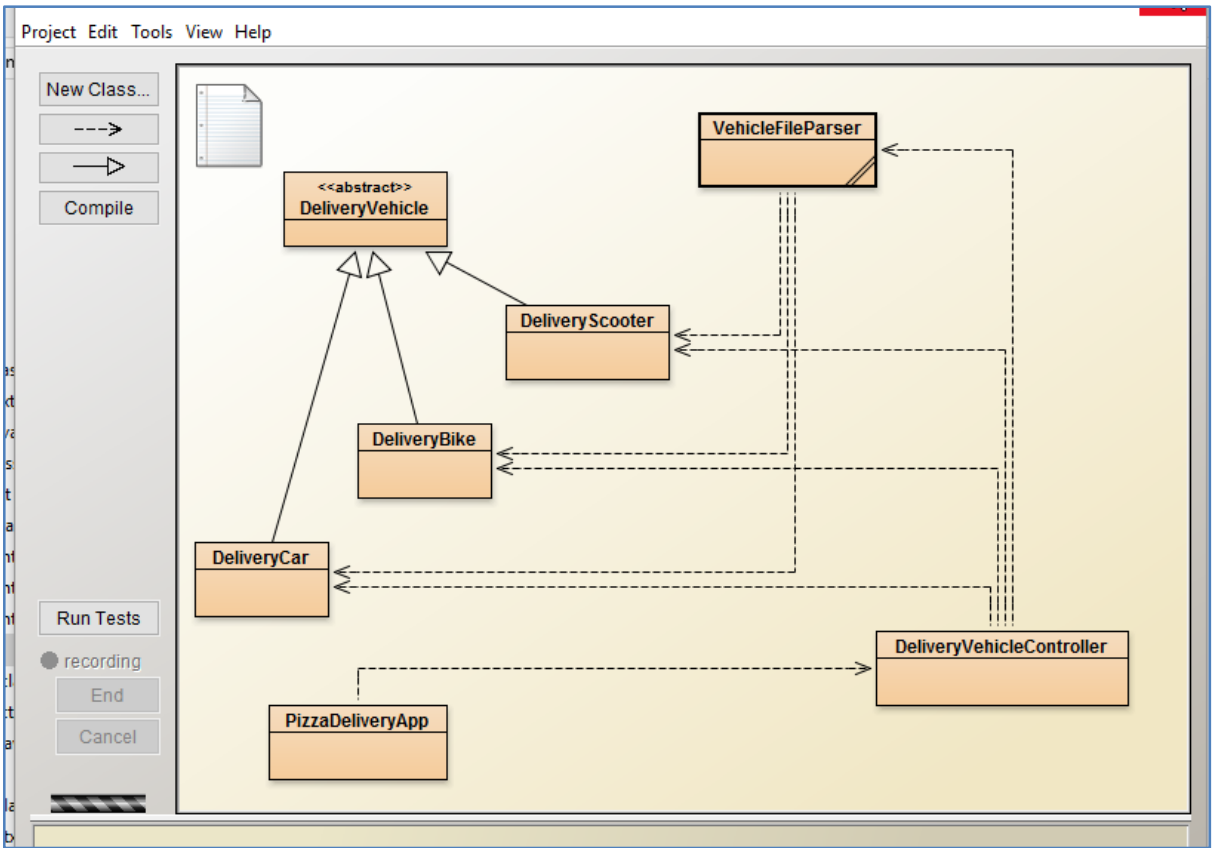


Figure 1. Class Diagram

Add appropriate methods to the ***DeliveryVehicle*** class to update the values of the private member variables when updating the daily delivery data.

- *void addMilesCoveree(double milesCovered)* to update the _milesCovered field.
- *void addDeliveriesMade(double deliveriesMade)* to update the *_deliveriesMade* field.
- *void incrementDaysInService()* to increments the *_daysInservice* field every time a vehicles daily data is updated.
- *String toString()* to override the *toString( )* method, so that the returned string is formatted appropriately to be used in a text file for storing values of each objects .
- *abstract double calculateRunningCost ()* abstract method that will be implemented in each sub class. Each sub class will have its own unique algorithm for calculating the

vehicles annual running cost, based on the information supplied in the project description.

Hint* Calculating the running costs will require to consider all costs, including taxes, fuel per mile, and grants applicable to each vehicle.

Group Assignment: PART B – DeliveryVehicleController class

To ensure we have low coupling and a good design, we will add a ***DeliveryVehicleController*** class that will be responsible for interacting with our ***DeliveryVehicle*** objects.

Our decision to use inheritance allows this class to have a single ArrayList of type *DeliveryVehicle* that is able to hold all instances of ***DeliveryBike***, ***DeliveryCar*** and ***DeliveryScooter*** objects.

Class variables are listed below and should be created in a constructor.

- *_vehicleFileParser* **VehicleFileParser** a vehicleFileParser variable that will be used to read and save vehicle object from a file. See part C.
- *_vehicleList* **ArrayList<DeliveryVehicle>** an ArrayList of type DeliveryVehicle to hold all instances of vehicles in the system.

Add appropriate methods to the ***DeliveryVehicleController*** class to interact with our ***DeliveryVehicle*** objects.

- *ArrayList<DeliveryVehicle>* ***getAllSavedVehicles()*** This method will use the **_vehicleFileParser.*getVehiclesFromFile()*** method to get all saved vehicles from a file on a disk.
- *void* ***saveAllVehicles*** *(ArrayList<DeliveryVehicle> vehicleList)* This method will use the **_vehicleFileParser.*saveAllVehiclesToFile()*** method save all vehicles in the list to a file.
- *void* ***addDeliveryVehicle****(DeliveryVehicle newVehicle)* This method will add the newly created vehicle to the *_vehicleList*.
- *DeliveryVehicle* ***getDeliveryVehicle****(String registrationNumber))* This method gets a vehicle from the *_vehicleList* based on its registration number.
- *String* ***updateDeliveryVehicle****(DeliveryVehicle vehicleToUpdate))* This method updates a specific vehicle in the list. Use a for loop to loop through the list and update the vehicle if it is found. Return a message if the vehicle is not found.

- *String* ***deleteDeliveryVehicle****(String registrationNumber))* This method removes a specific vehicle from the list. Use an iterator to loop through the list and remove the vehicle if it is found. Return a message if the vehicle is not found.

Group Assignment: PART C – VehicleFileParser class

To ensure we have low coupling and good design, we will add a **VehicleFileParser** class that will be responsible for saving and reading our vehicle objects from a file on disk. If we ever need to change the way the vehicles are saved in the future, we will just need to make changes to this one class without effecting the rest of the application.

Class variables are listed below and should be created in a constructor.

- *_filePath* **String** a string variable that will be used locate the file to read and write to. This will be passed into the constructor.

Add appropriate methods to the **VehicleFileParser** class.

- *ArrayList<DeliveryVehicle>* ***getVehiclesFromFile*** *()* This method will use the _filePath field to read the specified file on disk. Each line of the file represents a vehicle. The line is parsed to get the individual field values. Use java.nio Files and Paths to read the file.
- *void* ***saveAllVehiclesToFile*** *(ArrayList<DeliveryVehicle> vehicleList)* This method will use the java.io. FileIOutputStream and PrintStream classes to write each vehicle in the list out to the file.

Group Assignment: PART D – PizzaDeliveryApp:

The **PizzaDeliveryApp** class is used to display and retrieve data to and from the user. The class will use an instance of the **DeliveryController** class to retrieve, update and save our vehicle objects.

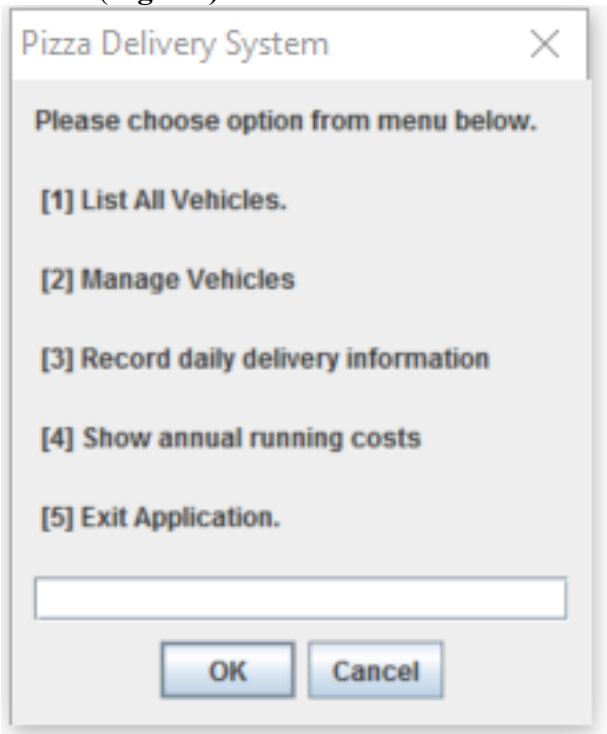
Class variable is listed below and should be newed up in the constructor (create an instance of an object).

- *_deliveryVehicleController* **DeliveryVehicleController -** this variable will be used to retrieve, save update vehicle objects.

Add appropriate methods to the **PizzaDeliveryApp**

- *void **launchApp()*** is the only public method of the **PizzaDeliveryApp** class. When it is called, a series of internal methods are used to display and retrieve data. An initial menu is displayed to the user in Figure B1.

(Fig: B1)



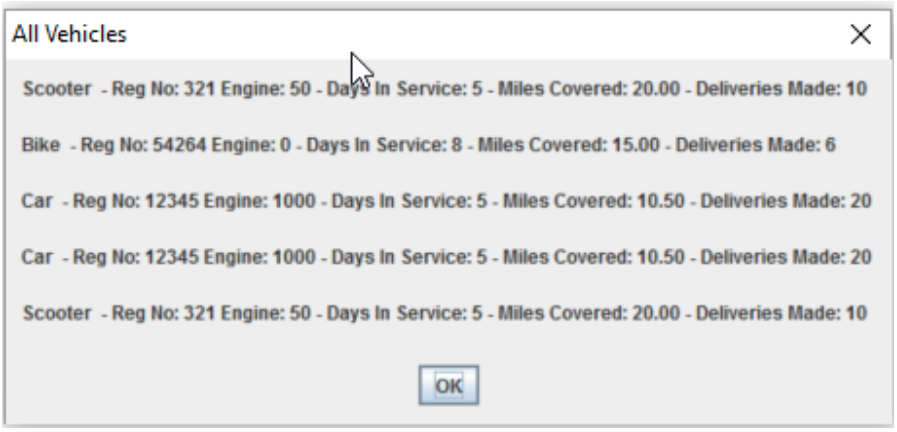
Hint* The jOptionPane class provides a number of methods to display information to users as well as prompt users to enter data.

- ***showMessageDialog***(null, message, title, icon) – displays a message to a user
- ***showInputDialog***(null, message, title, icon) - returns a String version of the user input

- ***showConfirmDialog***(null, message, title, JOptionPane.YES_NO_OPTION) – returns a 0 or 1 based on user selection.

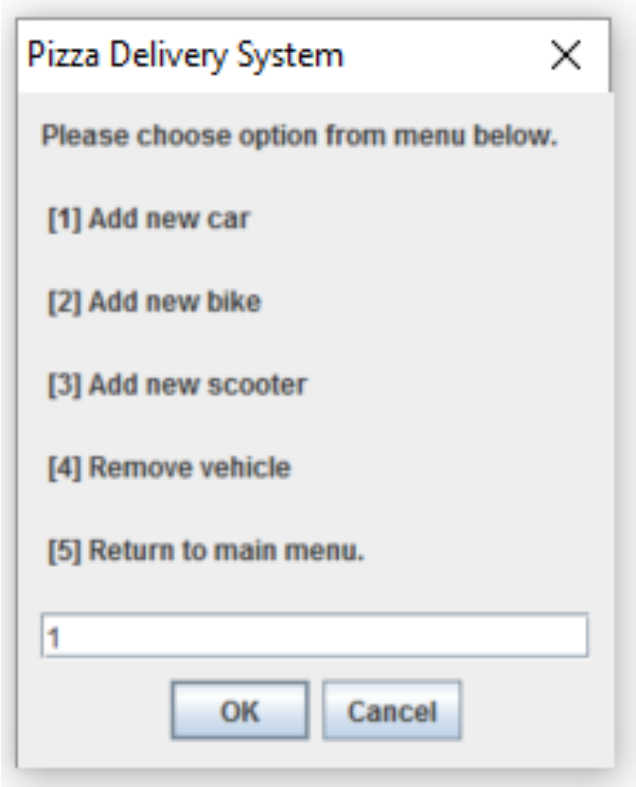
[1] List All Vehicles

If the user selects [1] List all vehicles, we use an internal method to get a list of all vehicles from file. We use a *for loop* to loop through the list and display the vehicles to the user.



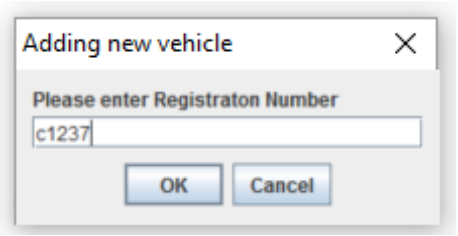
[2] Manage Vehicles

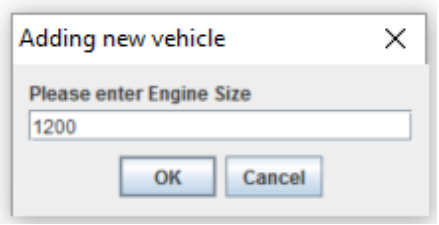
If the user selects [2] Mange Vehicles, we use an internal method to show the user a sub menu to manage vehicles.

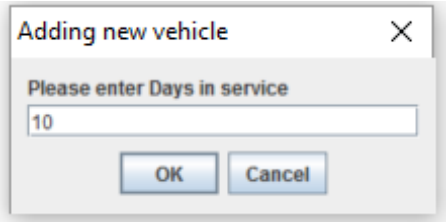


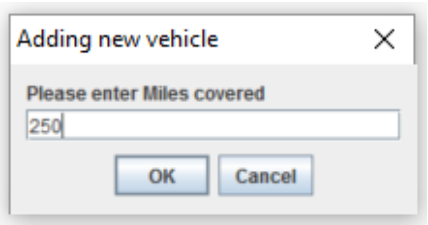
[1] Add new car

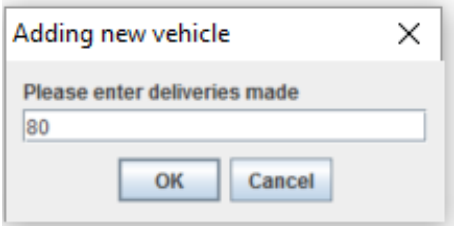
If the user selects [1] Add new car from the Manage Vehicle menu, the user is prompted to add the relevant information to add a new **DeliveryCar**. When finished, the user is presented with a success message. Sub men options [2] and [3] will add a **DeliveryBike** and **DeliveryScooter,** respectfully.

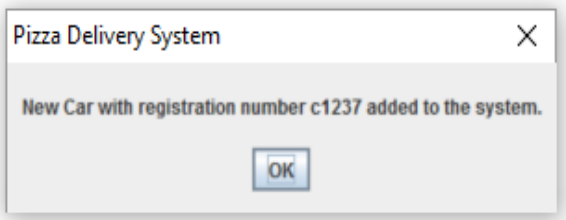






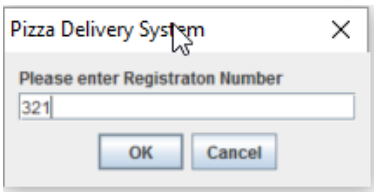


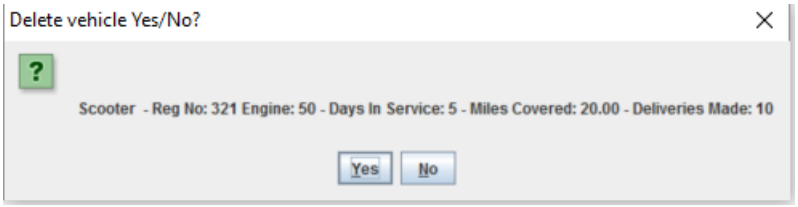


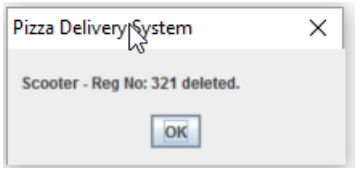


[4] Remove vehicle

If the user selects [4] Remove vehicle from the Manage Vehicle menu, the user is prompted to enter the registration number of the vehicle to delete. The user is asked to confirm before deleting. A success message is displayed when finished.

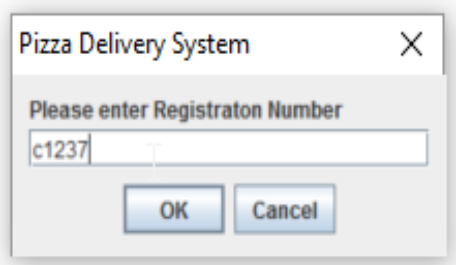


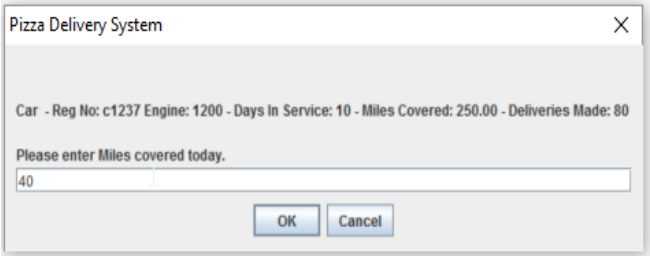


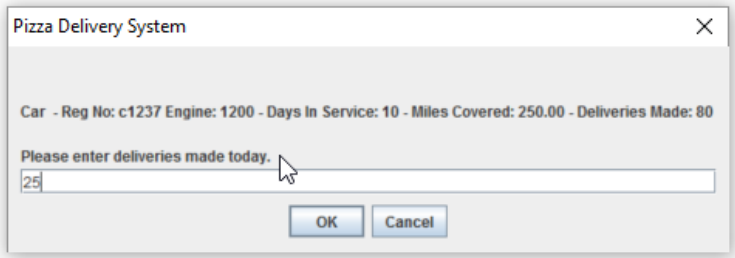


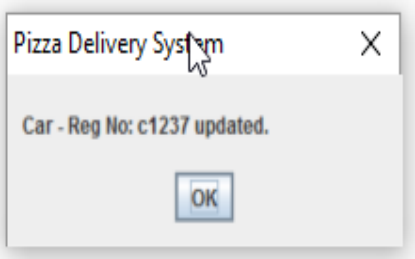
[3] Record daily delivery information

If the user selects [3] Record daily delivery information, the user is prompted to add the relevant daily delivery information. When finished, the user is shown a success message.



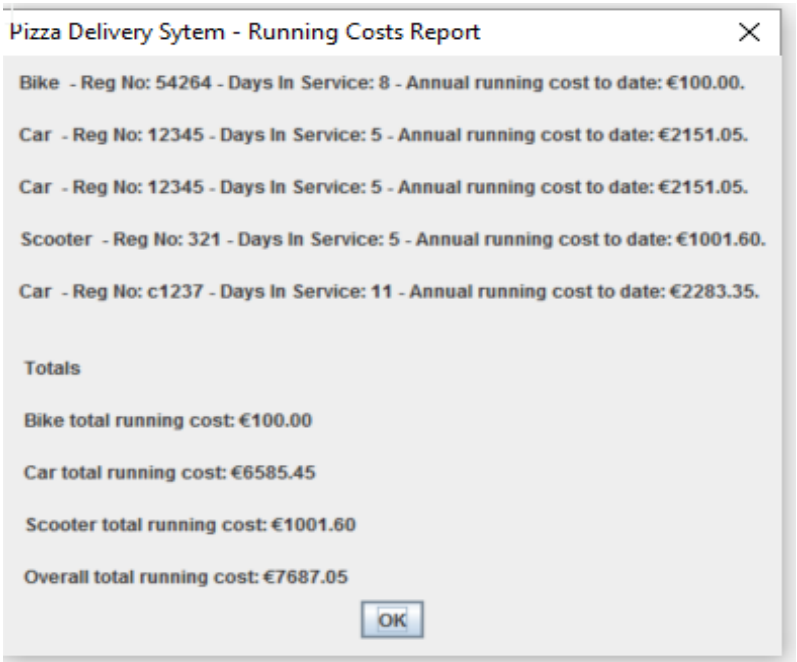






[4] Show annual running costs.

If the user selects [4] Show annual running costs, we use an internal method to get a list of all vehicles from file. We use a *for loop* to loop through the list. We call the ***calculateRunningCost()*** method of each vehicle along with keeping a running total of each vehicle type. We use these total to show a report of running costs to the user.



[5] Exit Application

Finally, if the user selects [5] Exit Application, ensure all updated data is saved before exiting the application.

Use *error handling* where appropriate.

Note: Develop any supporting / utility methods you feel you may need to meet the applications requirements. Methods listed are provided to give a structure to the program.

Please supply unit tests for the following methods:

- ***getYearlyTax()*** in **DeliveryCar** class
- ***getNoEmissionsGrant()*** in **DeliveryBike** class
- ***setNoEmissionsGrant()*** in **DeliveryBike** class

Please don't forget to follow good industry practices:

- All class variables start with an underscore
- Write clear comments
- Classes start with a capital letter
- Methods start with a small letter
- Names must be self-explanatory
- Always use crocodile brackets for if statements
- Declare class variables one per line
- Apply the rule of reusability (if appropriate)
- Apply the principle of high cohesion when applicable

Assignments submissions to follow guidelines specified on Blackboard. Please ensure that you submit code separately for each of the exercises given.

Deadline for submission is **Friday 15th of April @ 11.59pm**