ICS 102 Project (Term 172)

**Due:** **Each student** is required to submit, in **ICS 102** Blackboard, a zip file containing **VehicleModel.java**, **VehicleModelDriver.java**, and the **vehicles.txt** text-file by **11:59 p.m.** on **Saturday** **21st April 2018.** The submitted zip file must be in the format**: YourKFUPM ID\_ProjectGroupNumber.zip (Example: 201300000\_Group05.zip)**

Consider a **VehicleModel** class that has:

* private instance variables:
  + String modelName
  + int stockQuantity
  + int numRentedVehicles
  + double rentalPricePerDay
* An appropriate constructor
* The following public methods:
  + public String getModelName( )
  + public int getStockQuantity( )
  + public int getNumRentedVehicles()
  + public double getRentalPricePerDay( )
  + public void setStockQuantity (int stockQuantity) throws IllegalArgumentException
  + public void setRentalPricePerDay(double rentalPricePerDay) throws IllegalArgumentException
  + public double rentVehicleModel(int numVehicles, int numDays) throws IllegalArgumentException

// returns the cost of renting this many number of vehicles if there is no Exception

* + public void getVehiclesFromCustomer(int numVehicles) throws IllegalArgumentException
  + public String toString( )
  + public boolean equals(Object obj) // comparison based on name only and without case sensitivity

Implement a well-structured Java program that uses the **VehicleModel** class and **VehicleModelDriver** class to enable a car rental company to maintain the inventory of vehicles rented as well as to rent vehicles to customers. The inventory is kept in a text-file **vehicles.txt** of the form:

|  |
| --- |
| Hyundai 14 0 105.00  Yaris 12 0 147.00  Mercedes 7 0 630.00  Corolla 20 0 179.00  Ford 9 0 315.00  Honda 13 0 347.00  Chevrolet 22 0 200.00  Lexus 10 0 250.00  Kia 15 0 150.00  BMW 6 0 600.00  Nissan 20 0 170.00  Audi 11 0 550.00  GMC 16 0 683.00 |

where each line of the text-file contains a **unique** vehicle model name as a single word, the current stock quantity of the model, the current number of rented vehicles for the model, and the rental price per vehicle per day in Saudi Riyals. [Note: Assume that the car rental company has only one model for each brand name]

Your **VehicleModelDriver** program must have the following main menu:

**1. Display vehicle info for all models**

**2. Display vehicle info for a particular model**

**3. Update stock info for a particular model**

**4. Rent vehicle(s) to a customer**

**5. Get vehicle(s) from a customer**

**6. Add a new model**

**7. Delete a model**

**8. Exit**

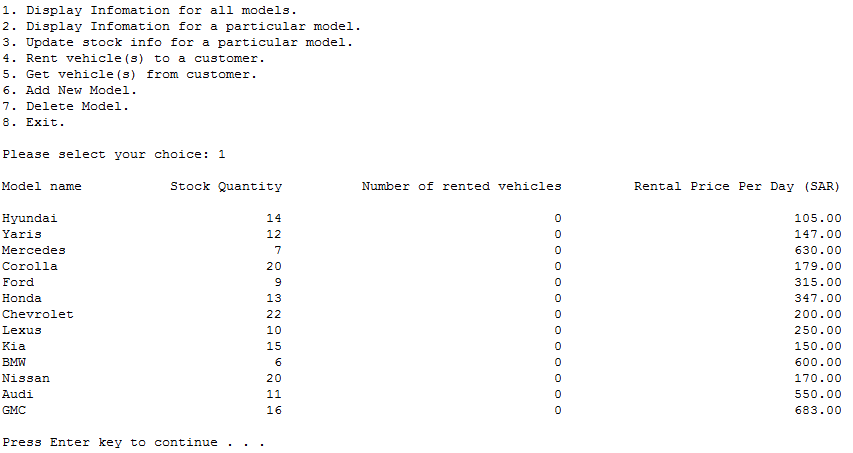
**Please select your choice:**

Your program must loop as long as option 8 has not been selected. It must display an appropriate error message if an invalid choice is entered. After executing each of the options 1 to 7, your program must pause and display the message: **“Press Enter key to continue . . .”** . Your program must display the main menu after pressing the Enter key. Each of the options 1 to 7 must be implemented in a separate static method. The sub-menu for option 3 and the code for **“Press Enter key to continue . . .”** must also be implemented is separate static methods. Your main method must recover from the following exceptions: InputMismatchException, IllegalArgumentException, and IOException.

The options must have the following behaviors:

**Option 1: Display vehicle info for all models**

It displays the vehicle information of all models. The option must be implemented by reading directly from **vehicles.txt** text-file. It then waits for the **Enter** key to be pressed before returning control to the main menu:



**Option 2: Display vehicle info for a particular model**

It prompts for and reads a model name. It then searches for this model in the **vehicles.txt** text-file **without case-sensitivity**. If the model is not found an appropriate error message is displayed, otherwise; the model information is displayed. In both cases, the option waits for the **Enter** key to be pressed before returning control to the main menu.

|  |
| --- |
|  |
|  |

**Option 3: Update stock info for a particular model**

It prompts for and reads the name of the model whose stock information is to be modified. It then searches for the model, without case-sensitivity, in the inventory text-file **vehicles.txt**. If the model is not found, an error message is displayed, and after the Enter key is pressed, control is returned to the main menu. If the model is found, the contents of the text-file are copied into **an array of objects** (count the number of lines in the text-file, and use this number as the array size). The following sub-menu is then displayed:

1. Modify stock quantity

2. Modify rental price per day

Please select your choice:

Your program must loop as long as the choice is not 1 or 2. If the choice is correct, your program must update the inventory text-file **vehicles.txt** according to the selected choice. However, if the stock quantity entered is less or equal to zero or the rental price per day is less or equal to zero, the option displays an appropriate error message:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

Whether the update operation was done or not, control is returned to the main menu after pressing the Enter key.

**Options 4. Rent vehicle(s) to a customer**

It initializes **an array of objects** with the information from the inventory file **vehicles.txt** (count the number of lines in the inventory text-file and use it as the array size).

It then rents one or more vehicles to a customer if the requested vehicle or vehicles are available and if the requested quantities are less or equal to the associated stock quantities and if the required number of days is in the range 1 to 31. Each time a vehicle is rented its bill is appended to a **bills.txt** text-file and the array of objects is updated. When the customer has rented all vehicles he wants, the inventory text file **vehicles.txt** is updated, and the customer bill is displayed on the monitor.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

Control is returned to the main menu after pressing the **Enter** key.

After the above updates (Changing stockQuantity of Honda from 14 to 16, changing rental price per day of Yaris from 147.00 to 135.00 Saudi Riyals, Renting 2 Mercedes, 1 Nissan, and 4 Honda vehicles), the contents of **vehicles.txt** will be as shown below:



**Options 5. Get vehicle(s) from a customer**

It initializes **an array of objects** with the information from the inventory file **vehicles.txt** (count the number of lines in the inventory text-file and use it as the array size).

It then returns one or more rented vehicles from a customer if the model exists and if the number of returned vehicles for that model is valid [i.e., it is not zero, negative, or more than the number of rented vehicles for the model]. The inventory text file **vehicles.txt** is updated accordingly.

|  |
| --- |
|  |
|  |
|  |
|  |

**Options 6**. **Add a new model**

To implement Option 6, search the text-file **vehicles.txt** for the name of the model to be added. If the model exists, display an error message; otherwise append the new model information to the end of the text-file by an output statement that first generates a new line; but that does not generate an extra new line at the end of the text-file. Display appropriate error messages if the stock is zero or negative or the rental price per day is zero or negative.

|  |
| --- |
|  |
|  |

**Options 7. Delete a model**

To implement option 7, search the text-file **vehicles.txt** for the name of the model to be deleted, keeping track of its line number in the text-file. If the model name does not exist or if the number of rented vehicles for this model is not zero, display an error message; otherwise read the vehicle information from the text-file into an array of **VehicleModel** objects, then overwrite the text-file with the contents of the array, but skipping the model you want to delete. Make sure you write the last line without generating an extra blank line at the end of the text-file.

**Note:**

* Your project must not use parallel arrays. IT MUST USE an array of **VehicleModel** objects.
* Your project must not use static or global variables.
* Your project must not use 2D-arrays.
* Your project must not use ArrayList or any other Java built in data structure.
* Your project must not use any Java feature not covered in ICS 102 without the consent of ICS 102 lab instructors.
* THE CODE FOR PROCESSING THE OPTIONS MUST NOT BE PLACED IN **VehicleModel.java** ; IT MUST BE IN **VehicleModelDriver.java**
* You must use meaningful variable names and proper indentation in your program.
* You may use private static methods in your program, like countLines, searchForVehicle, updateTextFile, loadToArrayOfObjects, etc.
* Use the clause **throws FileNotFoundException** or **throws IOException** for each method that performs File I/O.
* You must not share code with another project group. **Doing so will result in a zero grade and a possible F grade in the course for all groups involved.**
* Project demo slots of 10 minutes per group will be announced later. All group members are required to attend the project demo together. **A grade of zero will be given to any student not attending the Project Demo even if he submits a correct project.**
* **A grade of zero will be given to a student who will not participate with his group partners in the project.**

**Hints:**

* When you update the vehicles text-file, make sure you do not insert a blank line at the end of the file, because this will generate reading errors. Write the last file line without generating **‘\n’**
* Whenever you use **nextLine( )** that follows **next( )**, **nextInt( )** or **nextDouble( )**, make sure you use a dummy **nextLine( )** to remove **‘\n** from the input buffer.
* To return a formatted string from a method use:

**String.format(FormatString, ExpressionList)**