CSC232: Object-Oriented Software Development
Homework 02 – Creating a Vehicle Class
Due: Day 07 – Friday, September 10th @ beginning of class

In this homework, you will practice creating a single Java class and selecting the appropriate type for its objects' member variables and methods. Follow the directions below to complete this homework – be sure to read what I am asking for carefully as each component of your solution will be worth points and graded accordingly. If you wish, you may work with a single partner, however, you must email me your partner's name (CC them on the email) and only one member should submit a solution to Moodle.

Startup Steps

- On your computer, create a new folder (i.e., project) named **CSC232_HW2** – this will be the folder that you will save your Java classes in for this homework assignment
- Open Visual Studio Code, and select **File >> Open …** option, then navigate to and select the CSC232_HW2 folder in the dialog window that appears
- At this point, within Visual Studio Code, you should now see a collapsible/expandable section with the label **CSC232_HW2** in the Explorer pane on the left side. If the Explorer pane is not visible, select **View >> Explorer** or click on the icon that looks like "two sheets of paper" in the top-left corner.
- As we discussed in class, every Java program needs a class that contains the main( ) method. Add a **Driver.java** file to your project by hovering your mouse over the collapsible/expandable section with the label CSC232_HW2. You should see an icon of "a sheet of paper with a + sign" that reads **New File** when you hover on top of it. Click on this icon and a new file will appear waiting for you to provide its name – name this new file **Driver.java**
    - **Important**: You must put the **.java** extension on the end of the name, otherwise your computer will not know that Driver.java contains Java code
- Next, add the necessary code to define your Driver class – then, add the necessary code to define the main( ) method.
    - **Important**: In Java, the name of the class must always match the name of the file that it is stored in (excluding the .java extension). Therefore, your class's name must be Driver (lowercase vs uppercase matters)
- When you are finished with the steps above, a **Run** link should appear over the main( ) method. (If a Run link does not appear, be sure to rewatch the videos on Moodle in the Course Resources section for how to download and install the JDK, Visual Studio Code, and the Java Extensions Pack). Once the Run link is visible, click on it to run/execute your program. **Recall**: All Java programs begin executing at the main( ) method that you selected, therefore you can always trace through exactly what your code will do.
    - A window should appear at the bottom with the **Terminal** tab selected – this is the tab where your Java program will print its output. You may wish to add a System.out.println("Hello World") statement to your main( ) method to ensure everything is working correctly before continuing to the next steps
    - **Tip**: For this homework, I recommend using your main( ) method to create objects that test the creation of objects and calling their methods to ensure they are working correctly

Homework Steps

- Add a **Vehicle.java** file to your project's folder within Visual Studio Code and define a Vehicle class
    - **Reminder**: A class is a blueprint where we define the variables and methods that all objects of that type will have
- Define the **member variables** listed below in your Vehicle class that all Vehicle objects that you construct will contain. **Important**: You will be graded on selecting the appropriate **type** for each member variable given the range of values that traditionally would be assigned to it:
    - A member variable named **name** that can refer to the vehicle's full name (e.g., Honda Civic, Ford Mustang, Toyota Camry, etc.)
    - A member variable named **year** that stores the 4-digit year that the vehicle was manufactured (e.g., 1998, 2009, 2012, etc.)
    - A member variable named **odometer** that stores the number of miles that the vehicle has traveled since it was manufactured – you must use the appropriate integer-based type – do not use a floating-point type (see What is an Odometer if you want more information).
    - A member variable named **tripmeter** that stores the number of miles that the vehicle has traveled since the trip meter was last reset (i.e., the driver can reset the trip meter but cannot reset the odometer) – you must use the appropriate integer-based type – do not use a floating-point type
    - A member variable named **gallonsRemaining** that stores the number of gallons that remain in the vehicle's tank – you must use the appropriate integer-based type – do not use a floating-point type
    - A member variable named **gallonsCapacity** that stores the number of gallons that the vehicle's tank can maximally hold – you must use the appropriate integer-based type – do not use a floating-point type
    - **Important**: You should not add any additional member variables besides those that were requested above
- Finally, write the following **methods** listed below that a Vehicle object should be able to perform/calculate if we were to call them:
    - A method named **calculateAverageYearlyMiles** that takes the current year (e.g., 2020, 2021, 2022, etc.) as a formal parameter and returns "*the average number of miles the vehicle has traveled per year since it was manufactured*" as a floating-point type. Be sure to test this method to ensure it is returning floating-point values correctly.
        - **Important**: This method should not print any values using System.out.println( ) – it should return a single value (i.e., the average yearly miles traveled)
    - A method named **calculateMilesPerGallon** that does not require any formal parameters and that returns "*the average number of miles traveled per gallon since the driver reset the trip meter*" as a floating-point type
        - **Important**: This method should not print any values using System.out.println( ) – it should return a single value (i.e., the average miles per gallon) as a floating-point type
    - A method named **calculateMilesRemaining** that does not require any formal parameters and returns "*the estimated number of miles remaining until there is no more gas in the vehicle*". Your calculation will require the miles per gallon value that you calculated in the previous method, however, it is a bad programming practice to copy-and-paste calculations from one method to another. Therefore, you should review how to call the previous method to obtain its result in this method's code.
        - **Important**: This method should not print any values using System.out.println( ) – it should return a single value (i.e., the estimated number of miles remaining until there is no more gas in the vehicle)

- A method named **printMilesRemainingSummary** that does not require any formal parameters and does not return any value. Instead, this method should print how many miles remain after each gallon (in descending order). As an **example**, if a vehicle had 6 gallons remaining and it was averaging 30 miles per gallon, then this method should print the summary to the console in exactly the format provided below:

  6 gallons: 180 miles remaining
  5 gallons: 150 miles remaining
  4 gallons: 120 miles remaining
  3 gallons: 90 miles remaining
  2 gallons: 60 miles remaining
  1 gallons: 30 miles remaining
  0 gallons : 0 miles remaining

- A method named **checkStatus** that does not require any formal parameters and behaves as follows:
  - When the odometer has exceeded 100,000 miles, a status message "High Mileage" should be printed to the console
  - When only 10% of the fuel capacity is remaining in the tank, a status message "Low Fuel" should be printed to the console
  - Finally, when the average miles per gallon is less than 10.0, a status message "Poor Fuel Efficiency" should be printed to the console. Otherwise, when the average miles per gallon is between 30.0 and 40.0, a status message "High Fuel Efficiency" should be printed to the console
  - **Important**: It is possible for multiple status messages to be printed to the console if their conditions are met

Testing Steps

- Before submitting your solution, you should use your Driver's main( ) method to create Vehicle objects with different values for their member variables
- Then, you should call each Vehicle object's methods to check whether they are working correctly
- **Important**: Be sure to think about different scenarios that your method's code might need to handle (i.e., are there any scenarios that might "break" your code and cause the program to behave incorrectly)

Submission Steps

- Open up the Moodle in a web browser and navigate to our CSC232 course's page
- Upload the **.java** files OR a ZIP file containing your folder to the assignment's submission item on Moodle
  - **Important**: If you worked with a partner, only **one** member should submit a solution – be sure that you have emailed me who you were working with