|  |
| --- |
| **VehicleLastNameFirstName** |
| * vin: String * make: String * model: String * year: int * engineType: String * transmission: String * cost: double * mileage: double |
| <<constructor>> Vehicle(make: String, model: String, year: Int, engineType: String, transmission: String, cost: double, mileage: double)   * generateVIN() : String   + getVIN() : String  + getMake() : String  + getModel() : String  + getYear(): Int  + getEngineType() : String  + getTransmission() : String  + getCost() : double  + getMileage() : double  + setCost(cost: double)  + setMileage(mileage: double) |

**Lab Instructions: Implementing the Dealership System**

**Overview**

You will implement a Java program to manage a car dealership's inventory. The program will allow users to add, view, search, and remove vehicles, as well as calculate sales prices. The program will utilize object-oriented principles, including the use of a Vehicle class and an ArrayList to store vehicle objects.

**Implementation Requirements**

Your implementation should follow the outlined methods below, ensuring that they adhere to the required structure, inputs, and expected outputs.

**Class Structure**

* Implement a Dealership class with the required methods.
* The main method should handle user interaction through a menu system.
* Implement the above UML for VehicleLastNameFirstName
* **Implement generateVIN() to create a random 17-character identifier.**
* **Auto-Generating the VIN:**
* The VIN should be a 17-character random alphanumeric string.
* Allowed characters: "ABCDEFGHJKLMNPRSTUVWXYZ0123456789"
* (Avoid "I", "O" because they look like "1" and "0").
* Each vehicle gets a unique VIN when created.
* Use a loop to generate 17 random characters.
* Hint: Use Random class to generate a VIN character-by-character.
* Do not use Stringbulider, utilize an array to store randomly generated characters
* An ArrayList inventory should be created at the class level of DealershipLastNameFirstName to manage objects.
* Use printf() for string formatting

**Method Descriptions**

**1. main(String[] args)**

* This method serves as the entry point of the program.
* It should first call a method provided to you to pre-populate the inventory with sample vehicles.
* It should then display a menu in a loop, continuously prompting users for input until they choose to exit. The menu should be displayed by calling the displayMenu() method.
* User input should be handled using a Scanner object.
* Based on the user’s selection, the appropriate method should be called by utilizing a switch case.

**2. displayMenu()**

* This method should display the main menu options to the user.
* Each option should be clearly numbered.
* The menu should prompt the user to enter a choice.
* Example Output:

A black rectangle with white text

Description automatically generated

**3. addVehicle()**

* This method should prompt the user for vehicle details:
  + Make
  + Model
  + Year (Integer)
  + Engine Type (String: Gasoline, Diesel, Electric, Hybrid)
  + Transmission Type (String: Automatic, Manual)
  + Cost (Double)
  + Mileage (Double)
* A new Vehicle object should be created using the provided details.
* The new vehicle should be added to the inventory.
* A confirmation message should be displayed.

**4. viewInventory()**

* This method should display all vehicles currently stored in the inventory.
* If the inventory is empty, an appropriate message should be displayed.
* The output should be formatted into a table with the following fields:
  + **VIN**, **Make**, **Model**, **Year**, **Engine Type**, **Transmission**, **Cost**, **Mileage**
* Ensure proper spacing/alignment in output for readability.
* Use printf to format printing.

A screenshot of a computer

Description automatically generated

**5. removeVehicle()**

* This method should prompt the user to enter a **VIN** for the vehicle to be removed.
* If the **VIN** exists in the inventory:
  + Display the vehicle's details.
  + Ask the user for confirmation (yes/no).
  + If confirmed, remove the vehicle from the inventory.
  + Display a success message.
* If the **VIN** does not exist, display an error message.

**6. searchInventory()**

The searchInventory() method allows users to search for vehicles in the inventory based on different criteria: Make, Model, Year, or VIN. Once a search is performed, all matching vehicles should be displayed in a structured format.

**Method Breakdown**

1. **Display a Search Menu**
   * Prompt the user to select a search category.
   * Available options:
     + 1 → Search by Make
     + 2 → Search by Model
     + 3 → Search by Year
     + 4 → Search by VIN
   * If an invalid choice is entered, display an error message and exit the method.
2. **Retrieve and Validate Input**
   * Use scanner.nextInt() to read the menu selection.
   * Use scanner.nextLine() afterward to prevent input buffer issues.
   * Prompt the user to enter the corresponding search term.
3. **Search Logic**
   * Iterate through the list of Vehicle objects (inventory).
   * Compare the user’s input with the appropriate attribute of each Vehicle object.
   * The search should be **case-insensitive** (use .equalsIgnoreCase() for Strings).
   * When searching by **Year**, convert the integer year to a String before comparing.
   * When searching by **VIN**, return results as soon as a match is found, since VINs are unique.
4. **Store and Display Results**
   * If one or more vehicles match, add them to a temporary list.
   * If results exist, display them in a **formatted table**.
   * If no matches are found, display "No matching vehicles found."

**Implementation Steps**

1. **Prompt the user with a search menu.**
2. **Accept the user's choice and validate it.**
3. **Prompt the user for a search term based on their selection.**
4. **Iterate through the inventory and find matches.**
5. **Store matches in a temporary list.**
6. **Display results in a formatted table.**
7. **If no matches are found, display an appropriate message.**

**Implementation Notes**

* Use a **switch statement** (or if-else structure) to handle different search criteria.
* Use an **ArrayList** to store matching vehicles before displaying them.
* Use **helper methods** (e.g., printVehicleInfo(ArrayList<Vehicle>)) to format and display results.

**7. calculateSalesPrice()**

* This method should prompt the user to enter a **VIN** of the vehicle.
* If the **VIN** is found:
  + Display vehicle details.
  + Ask the user for:
    - **Tax rate** (percentage)
    - **Down payment**
    - **Annual Percentage Rate (APR)** (percentage)
    - **Loan term in months**
  + Calculate:
    - **Total price with tax**
    - **Loan amount (total price - down payment)**
    - **Monthly payment using the standard loan formula**:

A black background with white text

Description automatically generated

where:

* + M = monthly payment
  + P = loan amount
  + r = annual interest rate (decimal)
  + n = number of months
* Display the financing details in a formatted output.

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

If the **VIN** is not found, display an error message.

**8. printVehicleInfo(ArrayList<Vehicle> vehicles)**

* A helper method to format and print vehicle details in a table format.
* It should be used by viewInventory() and searchInventory().

**9. populateInventory()**

This method will be used to populate the inventory array for testing. It will be provided to you in a Java file. You will want to copy this method into your DealershipLastNameFirstName file.