PROGRAM 4 – Vehicle Rental Agency

250 pts.
(Due dates below)

Problem Description

You are to develop a Java program, using an object-oriented design, capable of maintaining vehicle rental reservations for a vehicle rental agency that rents cars, SUVs and minivans.

Problem Scenario

The agency offers various model cars, SUVs and minivans for rent. Customers may rent a vehicle for a period of days, weeks, or months. Monthly rentals are cheaper (on a daily basis) than weekly rentals, and weekly rentals are cheaper than daily rentals. There is also a mileage charge and optional daily insurance. Rates differ for cars, SUVs and minivans.

Users of the System

There are **two types of users** of the system, Employees and Managers. The **use cases** for each type of user are given below.

Employees of the rental agency may do the following:

(1) view current rates for a given vehicle type, including insurance rates and per mile charge
(2) view available vehicles (selected from cars, SUVs or minivans).
(3) display estimated cost for a given vehicle type, rental period, estimated miles, and optional insurance
(4) reserve a vehicle for a specific vehicle and rental period
(5) display a reservation (for a given customer name)
(6) cancel a reservation
(7) process a returned vehicle

Managers of the vehicle rental agency may:

(1) view/update current rates (including per mile charge and daily insurance rates)
(2) view all vehicles (each indicated as reserved or unreserved)
(3) view all reservations
(4) view all transactions (i.e., history of returned and paid rentals).

Vehicle Information

Vehicles are identified by a unique **VIN** (**Vehicle Identification Number**). The set of features maintained for cars, SUVs and minivans are different. Cars are specified as a specific make and model, a miles per gallon rating, and seating capacity. SUVs and minivans are specified as a particular make and model, a miles per gallon rating, seating capacity, and cargo capacity.

Given in the last pages is the list of vehicles, including the given VINs that **you must use** for this assignment.

Program Requirements

1. The program must be able to maintain a collection of specific model vehicles of type Car, SUV, and Minivan, and be able to display the vehicles in the following format:

   Toyota Prius (Car) MPG: 57 Seating: 4 VIN: ABD456
   Honda Pilot Hybrid (SUV) MPG: 36 Seating: 7 Cargo Storage: 6 cu. ft. VIN: KBS698
   Chrysler Pacifica Hybrid (Minivan) MPG: 36 Cargo Storage: 9 cu. ft. VIN: BFJ386

2. The program must be able to maintain a set of current rates for Cars, SUVs and Minivans, allow rates to be updated, and display the rates in the following format,

   Car Rates Daily: $24.95 Weekly: $169.95 Monthly: $514.95 Per Mile: $0.16 Daily Insur: $14.95

3. The program must be able to provide an estimated cost of a rental based on the vehicle type, expected rental period, expected number of miles to be driven, and optional insurance.
4. The program must allow vehicles to be reserved for a given time period, and optional daily insurance by use of a credit card.
5. The program must allow reservations to be canceled (by credit card number reserved under and vin).
6. The program must process a returned vehicle to (a) display the amount due (for the number of days had and mileage driven), (b) store the transaction, and (c) remove the reservation for the vehicle. To calculate the cost the daily, weekly or monthly rate will be applied, with any left-over days pro-rated at the weekly or monthly rate (40 days would be 31 days at the monthly rate, and the remaining 9 days each at a rate of (monthly rate / 31).
7. The program must be able to display all reservations.
8. The program must store and display all transactions. The should include the name of the customer, their credit card number, the type and model of vehicle rented, the number of days had, the number of miles driven, whether the optional insurance was accepted, and the total cost displayed as follows:

   Customer Name (card #3212546453245879), Car: Toyota Prius, 3 days, 540 mi (insur declined) $120.54

Rates to Use

Following are the rental rates, per mile charges, and insurance rates that you must use for each vehicle type:

	Daily Rate	Weekly Rate	Monthly Rate	Mileage Charge	Daily Insurance
Car	$24.95	$169.95	$514.95	0.16 / mile	$14.95 / day
SUV	$29.95	$189.95	$679.95	0.16 / mile	$14.95 / day
Minivan	$36.95	$224.95	$687.95	0.21 / mile	$19.95 / day

How Rental the Rates are Handled

The VehicleRates class stores the rental rates for a given vehicle type. The CurrentRates class is an aggregator class that is constructed with three VehicleRates objects: for Cars, SUVs and Minivans. Methods setCarRates, setSuvRates and setMinivanRates are called to update the rates (passed a new VehicleRates object). In addition, the VehicleRates class contains method for calculating charges (calcEstimatedCost and calcActualCost).

Ensuring That a Customer is Charged the Rates Quoted

Since rates may change *after* a customer has reserved a vehicle, we need to ensure that they are not charged a different rate when the vehicle is returned. Thus, a copy of the appropriate current rates object is made and attached to the vehicle reserved (by the setQuotedRates method of the Vehicle class). These are the rates (not the current rates) that are ultimately used to calculate the rental charges when the vehicle is returned.

User Interfaces

Note the difference in the sense of "interface" here. The **UserInterface** below is a Java interface type. That is, it is a "pure abstract class" (i.e., an abstract class with all abstract methods). **EmployeeUI** and **ManagerUI** are *user* interfaces that provide a means of interacting with the system.

```
import java.util.Scanner;
public interface UserInterface {

    // The start method begins the command loop of a text-based user interface (UI)
    public void start(Scanner input);
}

public class EmployeeUI_Interface implements UserInterface {
    // No constructor needed, calls static methods of the SystemInterface.
    // Method start begins a command loop that repeatedly: (a) displays a menu of options,
    // (b) gets the selected option from the user, and (c) executes the corresponding command.
    private boolean quit = false;
    public void start(Scanner input) {

        int selection;

        // command loop
        while(!quit) {
            displayMenu();
            selection = getSelection(input):
            execute(selection, input);
        }
    }

    // ------- private methods

    private void execute(int selection, Scanner input)

        int veh_type;
        String vin, creditcard_num;
        String[] display_lines;
        RentalDetails rental_details;
        ReservationDetails reserv_details;

        switch(selection) {

            // display rental rates
            case 1: veh_type = getVehicleType(input);
                    switch(veh_type) {
                       case 1: display_lines = SystemInterface.getCarRates(); break;
                       case 2: display_lines = SystemInterface.getSUVRates(); break;
                       case 3: display_lines = SystemInterface.getMinivanRates(); break;
                    }
                    displayResults(display_lines);
                    break;

            // display available vehicles
            case 2: veh_type = getVehicleType(input);
                    switch(veh_type) {
                       case 1: display_lines = SystemInterface.getAvailCars(); break;
                       case 2: display_lines = SystemInterface.getAvailSUVs(); break;
                       case 3: display_lines = SystemInterface.getAvailMinivans(); break;
                    }
                    displayResults(display_lines);
                    break;
```

```
            // display estimated rental cost
            case 3: rental_details = getRentalDetails(input);
                    display_lines = SystemInterface.calcRentalCost(rental_details);
                    displayResults(display_lines);
                    break;

            // make a reservation
            case 4: reserv_details = getReservationDetails(input);
                    display_lines = SystemInterface.makeReservation(reserv_details);
                    displayResults(display_lines);
                    break;

            // cancel a reservation
            case 5: vin = getVIN(user_input);
                    display_lines = SystemInterface.cancelReservation(creditcard_num, vin);
                    displayResults(display_lines);
                    break;

            // process returned vehicle
            case 6: creditcard_num = getCreditCardNum(user_input);
                    vin = getVIN(input);
                    display_lines = SystemInterface.processReturnedVehicle(vin,
                                                num_day_used,num_miles_driven);
                    displayResults(display_lines);
                    break;

            // quit program
            case 7: quit = true;
        }
        private void displayMenu() {   }
        // displays the user options

        private int getSelection(Scanner input) {   }
        // prompts user for selection from menu (continues to prompt if selection < 1 or selection > 8)

        private String getVIN(Scanner input)
        // prompts user to enter VIN for a given vehicle (does not do any error checking on the input) {    }

        private int getVehicleType(Scanner input)
        // prompts user to enter 1, 2, or 3, and returns (continues to prompt user if invalid input given) {    }

        private RentalDetails getRentalDetails(Scanner input)
        // prompts user to enter required information for an estimated rental cost (vehicle type, estimated
        // number of miles expected to be driven, expected rental period and optional insuranc, returning the
        // result packaged as a RentalDetails object (to pass in method calls to the SystemInterface) {   }

        private ReservationDetails getReservationDetails(Scanner input)
        // prompts user to enter required information for making a reservation (VIN of vehicle to reserve,
        // credit card num, rental period, and optional insurance), returning the result packaged as a
        // ReservationDetails object (to pass in method calls to the SystemInterface)  {    }

        private void displayResults(String[] lines)
        // displays the array of strings passed, one string per screen line {    }
}
```

NOTE: The **ManagerUI_Interface** class is designed similarly.

System Interface

The system interface is the intermediary between the user interface and "back end" system objects. It contains all static methods, one for each use case (and static variables of type CurrentRates, Vehicles and Transactions), providing all of the needed functionality. These are the only methods that the user interfaces directly call, each returning an array of strings which the user interfaces simply display. Given that the class contains all static methods (and static variables) an object instance is not constructed for the class.

```
public class SystemInterface {

    private static CurrentRates agency_rates;
    private static Vehicles agency_vehicles;
    private static Transactions transactions_history;

    // used to init static variables (in place of a constructor)
    public static void initSystem(CurrentRates r, Vehicles v, Transactions t) {
        agency_rates = r;
        agency_vehicles = v;
        transactions_history = t;
    }

    // used to check if SystemInterface initialized
    public boolean initialized() {
        return agency_rates != null;
    }

    // Methods makeReservation, cancelReservation and updateXXXRates return an acknowledgement
    // of successful completion of the requested action (e.g. "Vehicle ABC123 successfully reserved"). Method
    // processReturnedVehicle returns the cost for the returned vehicle (e.g., "Total charge for VIN ABC123 for
    // 3 days, 233 miles @  0.15/mile and daily insurance @ 14.95/day = $xxx.xx.)

    // Current Rates Related Methods
    public static String[ ] getCarRates() { ... }
    public static String[ ] getSUVRates() { ... }
    public static String[ ] getMinivanRates() { ... }
    public static String[ ] updateCarRates(VehicleRates rates) { ... }

    public static String[ ] updateSUVRates(VehicleRates rates) { ... }

    public static String[ ] updateMinivanRates(VehicleRates rates) { ... }

    public static String[ ] calcRentalCost(RentalDetails rental_details) { ... }

    public static String[ ] processReturnedVehicle(String vin, int num_days_used, int num_miles_driven) { ... }

    // Note that the rates to be used are retrieved from the VehicleRates object stored with the specific vehicle
    // object, and the daily insurance option is retrieved from the ReservationDetails object of the rented vehicle

    // Vehicle Related Methods
    public static String[ ] getAvailCars() { ... }
    public static String[ ] getAvailSUVs() { ... }
    public static String[ ] getAvailMinivans() { ... }
    public static String[ ] getAllVehicles() { ... }
    public static String[ ] makeReservation(ReservationDetails resv) { ... }
    public static String[ ] cancelReservation(String vin) { ... }
    public static String[ ] getReservation(String vin) { ... }
    public static String[ ] getAllReservations() { ... }

    // Transactions Related Methods
    public static String[ ] addTransaction(String credit_card, String customer_name, String vehicle_type,
                                           String rental_period, String miles_driven, String rental_cost) { ... }
    public static String[ ] getAllTransactions() { ... }
```

Example Implementation of the SystemInterface (method getAvailCars)

```
public static String[] getAvailCars() {
    // count the number of available cars
    int num_cars = 0;
    Vehicle v;
    vehs.reset();

    while(vehs.hasNext()) {
        v = vehs.getNext();
        if(v instanceof Car && !v.isReserved())
            num_cars = num_cars + 1;
    }

    // create appropriate size array
    String[ ]  avail_cars = new String[num_cars];

    // populate the array with available cars
    int i = 0;
    vehs.reset();

    while(vehs.hasNext()) {
        v = vehs.getNext();
        if(v instanceof Car && !v.isReserved())
            avail_cars[i++]  = v.toString();
    }
    return avail_cars;
}
```

Getting Things Started

Java Main Method (The code for this is given below)

The main method of the program is responsible for:

- Constructing (and populating) the CurrentRates, Vehicles, and Transactions objects
- Passing these objects to the initSystem method of the SystemInterface class
- Constructing and starting the execution of the appropriate user interface

System Initialization

A separate static method, called **initSystem**, of the SystemInterface is called by the main method and passed constructed and populated CurrentRates and Vehicles objects. It will also pass (an initially empty) Transaction object. The initSystem method assigns the three static variables of the System Interface class to the four aggregation objects passed. The initSystem method takes the places of what a constructor would do. But since this is a class with all static methods, an object instance of the SystemInterface is never constructed.

Request of User Interface Type

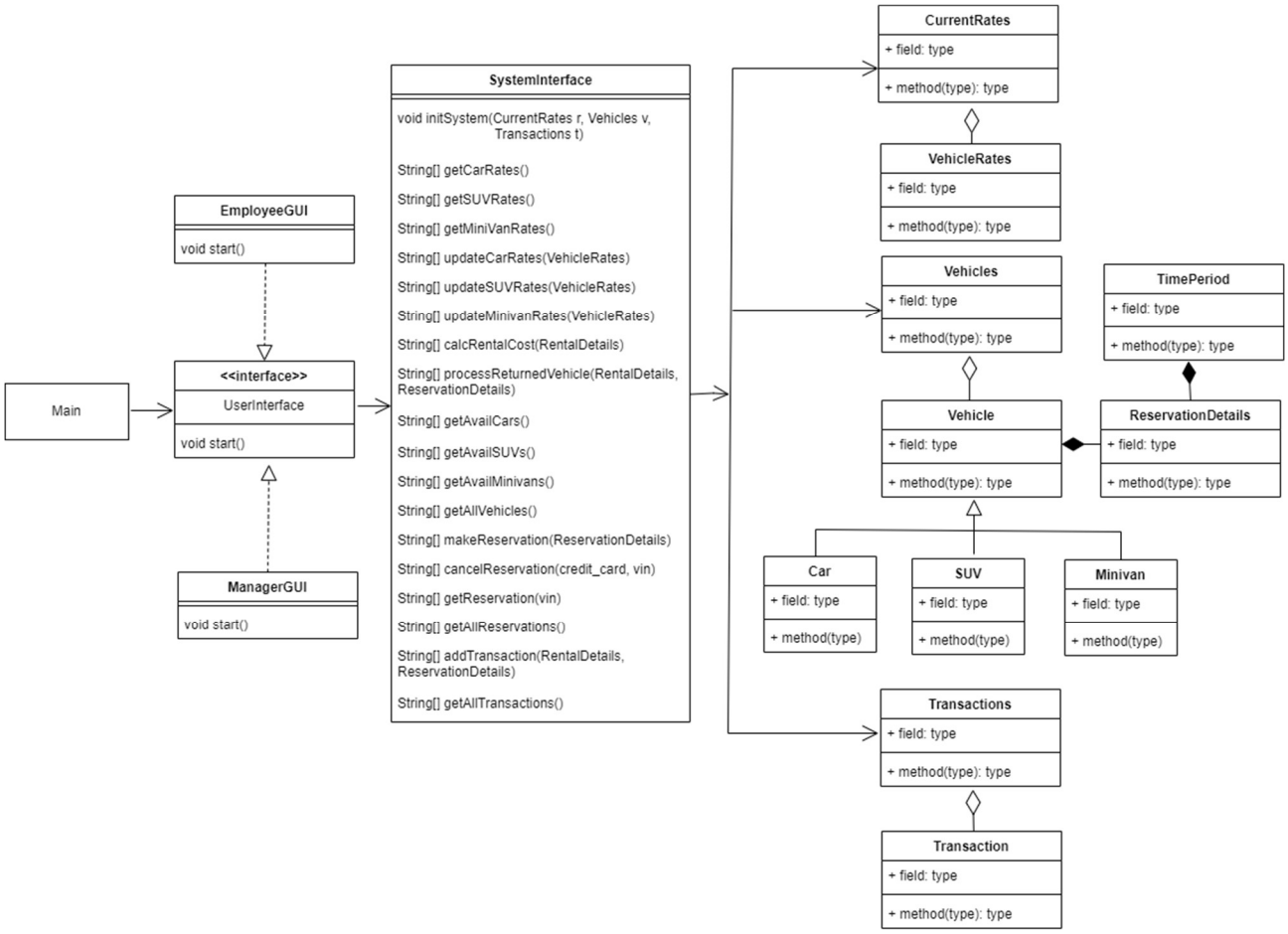
- The user will be prompted as to whether they are an employee or manager. (We will just assume that user validation, i.e., password entry, has been done.)
- The appropriate user interface class will be instantiated.
- The start method of the particular user interface object will be called and begin the command loop, allowing the user to begin entering commands.

Main Program

```
public class AgencyRentalProgram {

    public static void main(String[ ] args) {

        // Provide Hard-coded Current Agency Rates
        CurrentRates agency_rates = new CurrentRates(new VehicleRates(xx,xx,xx,xx,xx),  // cars
                                                     new VehicleRates(xx,xx,xx,xx,xx),  // SUVs
                                                     new VehicleRates(xx,xx,xx,xx,xx)); //minivans

        // Create an Initially Empty Vehicles Collection, and Populate
        Vehicles agency_vehicles = new Vehicles();
        populate(agency_vehicles);    // supporting private static method (to be added)

        // Create Initially Empty Transactions Object
        Transactions transactions = new Transactions();

        // Establish User Interface
        Scanner input = new Scanner(System.in);
        UserInterface ui;
        boolean quit = false;

        // Create Requested UI and Begin Execution
        while(!quit) {  // (allows switching between Employee and Manager user interfaces while running)

            ui = getUI(input);

            if(ui == null)
                quit = true;
            else {
                // Init System Interface with Agency Data (if not already initialized)
                if(!SystemInterface.initialized())
                    SystemInterface.initSystem(agency_rates, agency_vehicles, transactions);

                // Start User Interface
                ui.start(input);
            }
        }
    }

    public static UserInterface getUI(Scanner input) {
        boolean valid_selection = false;

        while(!valid_selection) {
            System.out.print("1 – Employee, 2 – Manager, 3 – quit: ");

            selection = input.nextInt();
            if(selection == 1) {
                return new EmployeeInterface();
                valid_selection = true;
            }
            else
            if(selection == 2) {
                return new ManagerInterface();
                valid_selection = true;
            }
            else
            if(selection == 3) {
                return null;
                valid_selection = true;
            }
            else
                System.out.println("Invalid selection – please reenter");
        }
        return ui;
    }
}
```

UML Class Diagram

Following is a UML class diagram for this project. This diagram indicates the classes that you are to use, and the relationships between them. The line connecting the Vehicle and ReservationDetails classes (with the filled diamond head at the Vehicle class) indicates that a ReservationDetails object is a member of a Vehicle object (i.e., composition). The notation 0..1 indicates that there may be 0 or 1 reservations for each vehicle. The unfilled, closed arrowhead indicates that the Car, SUV and Minivan are subclasses of the (abstract) Vehicle class. The open diamond head indicates that the Vehicles class is a collection (aggregation) of Vehicle objects.



Java Interfaces Needed

UserInterface

```
public interface UserInterface {
    public void start();
}
```

Abstract Classes Needed

Abstract Vehicle Class (and Subclasses Car, SUV and Minivan)

instance variables

```
private String description          // make-model for cars/suvs, length for minivans
private int mpg                     // miles per gallon
private String vin                  // unique vehicle identification number
private ReservationDetails resv     // If null, then vehicle not reserved
private VehicleRates rates;         // only assigned when vehicle reserved
```

constructor

```
public Vehicle(String description, int mpg, String vin) {
        this.description = description;
        this.mpg = mpg;
        this.vin = vin;
        resv = null;  // init as "no reservation"
        cost = null;  // used to hold rates at the time of the reservation }
```

methods

```
public String getDescription() { ... }
public int getMpg() { ... }
public String getVIN() { ... }
public ReservationDetails getReservation() { ... }
public VehicleRates getQuotedRates() { ... }
public boolean isReserved()  { ... }
public void setReservation(ReservationDetails) { ... } - throws ReservedVehicleException (if veh reserved)
public void setQuotedRates(VehicleRates cost) { ... }
public cancelReservation() { ... } - throws UnreservedVehicleException if reservation doesn't exist
public abstract String toString();  // ABSTRACT METHOD – implemented in each subclass
```

Information-Store Classes Needed (all immutable)

Car, SUV and Minivan Classes

Contain a constructor, possible instance variables, and appropriate implementation of the toString method.

VehicleRates Class

instance variables (for storing daily, weekly, and monthly rates, mileage charge, and daily insurance rate.

```
private double daily_rate;
private double weekly_rate;
private double monthly_rate;
private double per_mile_charge;
private double daily_insurance;
```

constructor

```
public VehicleRates(double daily_rate, double weekly_rate, double monthly_rate, double per_mile_charge,
                    double daily_insur_rate)
```

copy constructor
public VehicleRates(VehicleRates other);

methods
public double getDailyRate(); // cost per day
public double getWeeklyRate(); // cost per week
public double getMonthlyRate(); // cost per month
public double getMileageChrg(); // cost per mile, based on vehicle type
public double getDailyInsurRate(); // insurance cost (per day)
public String toString();

Only need to contain a constructor, and appropriate implementation of the abstract methods of the VehicleRates class (each returning a hard-coded set of rates).

TimePeriod Class

instance variables
private char unit; // 'd'-days. 'w'-weeks, 'm'-months
private int quantity; // how many days, weeks or months

methods
appropriate constructor, getter and toString methods.

ReservationDetails Class

instance variables
private String customer_name
private String credit_card_num;
private TimePeriod rental_period;
private boolean insurance_selected; // set to `true` if optional daily insurance wanted
private String VIN;

methods
appropriate constructor, getter and toString methods.

RentalDetails Class

Note that RentalDetails objects are not stored, and therefore do not appear in the class diagram. It is only used for passing the information related to a rental as a single parameter (object).

instance variables
private String vehicle_type
private TimePeriod rental_period;
private int num_miles_driven;
private boolean insurance_selected; // set to `true` if optional daily insurance wanted

methods
appropriate constructor, getter and toString methods.

Transaction Class

instance variables
private String creditcard_num;
private String customer_name;
private String vehicle_type; // car, SUV or Minivan
private String rental_period; // days, week, months
private String miles_driven;
private String rental_cost;

methods
(appropriate constructor)
public String toString() {... }

Aggregator Classes Needed

CurrentRates Class (aggregation of VehicleRates objects)

instance variable

private VehicleRates[ ] rates = new VehicleRates[3]; // array of size 3 to store rates for three types of vehicles

methods

constructor – passed CarRates, SUVRates and MinivanRates objects to assign in rates array.

public VehicleRates getCarRates() { ... } public void setCarRates(VehicleRates r){ … }
public VehicleRates getSUVRates() { ... } public void setSUVRates(VehicleRates r){ … }
public VehicleRates getMinivanRates() { ... } public void setMinivanRates(VehicleRates r){ … }

public double calcEstimatedCost(int vehicleType,TimePeriod estimatedRentalPeriod,
int estimatedNumMiles,
boolean dailyInsur) { ... }

- called to calculate a cost quote for a customer given an estimated number of days will use and estimated number of miles will drive, and whether the daily insurance wanted.

- uses the CURRENT standard rates for the vehicle type (car, SUV or minivan)

public double calcActualCost(VehicleRates rates, int num_days_used, int NumMilesDriven,
boolean dailyInsur) { ... }

- called to calculate charge of a returned vehicle (for number of days used, num miles driven, and whether daily insurance was selected).

- uses the QUOTED RATE, the rates as they were at the time of the reservation.

Vehicles Class (aggregation of Vehicle objects) * **MUST USE A LINKED LIST** *

instance variables

private VehicleNode head; // linked list of Vehicle objects
private VehicleNode current; // index of current vehicle used by iterator methods

methods

(appropriate constructor)

public void add(Vehicle v)
public Vehicle getVehicle(VIN) // throws VINNotFoundException if no vehicle found for provided VIN

iterator methods

public void reset() // resets to first vehicle in list
public boolean hasNext() // returns true if more vehicles in list to retrieve
pubic Vehicle getNext() // returns next vehicle in list

Transactions Class (aggregation of Transaction objects)

instance variable

private Transaction[ ] transactions; // array of Transaction objects (Java ArrayList type may also be used)

methods

(appropriate constructor)

public void add(Transaction tran) { ... }

iterator methods

(similar to Vehicles class)

Vehicle Data to Use

You are to use the following vehicle information WITH THE VINs GIVEN.

Cars:

- Toyota Prius: 57 mpg combined, VIN ABD456
- Honda Insight: 55 mpg combined, VIN DFE123
- Hyundai Elantra Hybrid: 53 mpg combined, VIN JKH857

SUVs:

- Toyota RAV4 Hybrid: 39 mpg combined, VIN DPF450
- Ford Explorer Hybrid: 31 mpg combined, VIN WHC302
- Honda Pilot Hybrid: 36 mpg combined, VIN KBS698
- Lexus NX 450h+: 37 mpg combined, VIN GEK334

Minivans:

- Toyota Sienna: 36 mpg combined, VIN AGH890
- Chrysler Pacifica Hybrid: 82 MPGe, VIN BFJ386
- Honda Odyssey: 22 mpg combined, VIN KCM341
- Kia Carnival: 22 mpg combined, VIN THS580

Due Dates of Vehicle Rental Agency Program

Stage 1 – Due Monday, April 14th by 11:59pm (50 pts.)

(a) Vehicle-Related Classes

Implement classes **Vehicle**, **Car**, **SUV**, **Minivan** and **Vehicles**.

Develop a test driver for your Vehicles class and fully test it

(b) Transaction-Related Classes

Implement classes **Transaction** and **Transactions**.

Develop a test driver for your Transactions class and fully test it.

(c) Rates-Related Class

Implement classes **VehicleRates** and **CurrentRates.**

Develop a test driver for your CurrentRates class and fully test it.

Stage 2 - Due Monday, April 28th by 11:59pm (100 pts.)

Implement and **develop a test driver** for the System Interface methods (implementing each use case).

Stage 3 - Due Monday, May 12th by 11:59pm (100 pts.)

Implement the Employee and Manager user interfaces (each implementing UserInterface) and test the complete system.