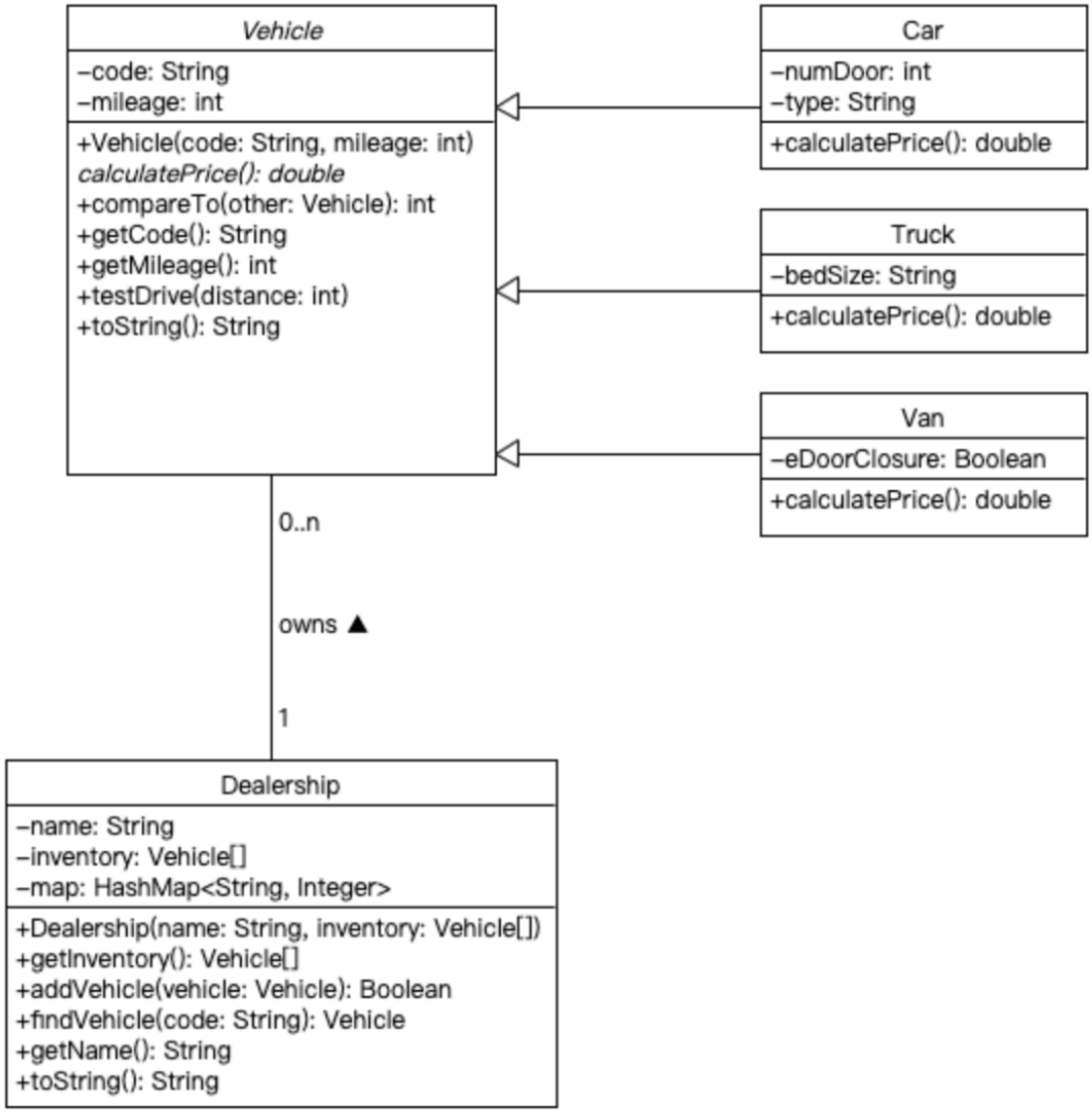
CS 1083

Assignment #4

Author: Yulong Wang

Id: 3713596

1. UML



2. Source Code

a. Dealership.java

```
import java.util.HashMap;

/**
 * @author Yulong Wang
 * @date 2021/10/10
 */
public class Dealership {
    /**
```

```java
     * The name of the dealership
     */
    private final String name;
    /**
     * The list of vehicle own by the dealer
     */
    private Vehicle[] inventory = new Vehicle[0];
    /**
     * The dictionary contain all vehicle code own by the dealer as key.
     */
    private HashMap<String, Integer> map = new HashMap<String,Integer>();

    /**
     * @param name Dealership name
     * @param inventory List of vehicle own by the dealership
     */
    public Dealership(String name, Vehicle[] inventory) {
        this.name = name;
        for(Vehicle vehicle : inventory){
            if(vehicle != null){
                addVehicle(vehicle);
            }
        }
    }

    /**
     * Add the vehicle into the inventory.
     * @param vehicle Vehicle to be added
     * @return boolean If adding succeed.
     */
    public boolean addVehicle(Vehicle vehicle){
        Vehicle[] newList = new Vehicle[inventory.length+1];
        if(!map.containsKey(vehicle.getCode())){
            map.put(vehicle.getCode(),0);
            for(int i =0;i<inventory.length;i++){
                newList[i] = inventory[i];
            }
            newList[inventory.length] = vehicle;
            this.inventory = newList;
            return true;
        }
        return false;
    }

    /**
     * Get a copy of the inventory
     * @return {@link Vehicle[]}
     */
    public Vehicle[] getInventory() {
        return inventory.clone();
    }
```

```
    /**
     * Get the name of dealership
     * @return {@link String}
     */
    public String getName() {
        return name;
    }

    /**
     * Find a vehicle in inventory
     * @param code The vehicle code to be searched
     * @return {@link String}
     */
    public String findInventory(String code) {
        for(int i=0;i<inventory.length;i++){
            if(inventory[i].getCode().equals(code)){
                return "vehicle found";
            }
        }
        return "vehicle not found";
    }

    /**
     * Convert the instance information to string
     * @return {@link String}
     */
    @Override
    public String toString() {
        String summary = name+"\n";
        for(Vehicle vehicle: inventory){
            summary+=vehicle.toString()+"\n";
        }
        return summary;
    }
}
```

b. Vehicle.java

```
import java.text.NumberFormat;
import java.util.Locale;

/**
 * This class represents a vehicle.
 * @author Yulong Wang
 * @date 2021/10/10
 */
public abstract class Vehicle implements Comparable<Vehicle>{
    /**
     * Code that identically represent a car
     */
    private String code;
    /**
```

```java
     * The mileage of the vehicle
     */
    private int mileage;

    /**
     * @param code The code that identically represent a car.
     * @param mileage The mileage of the car
     */
    public Vehicle(String code, int mileage) {
        this.code = code;
        this.mileage = mileage;
    }

    /**
     * This method calculate the price of a car.
     * @return double The price of the car.
     */
    abstract double calculatePrice()
            ;

    /**
     * This method overwrite the compare to method.
     * @param other Another vehicle to be compared
     * @return int compare result 1 as bigger, 0 as equal, -1 as smaller.
     */
    @Override
    public int compareTo(Vehicle other){
        if(this.code.charAt(0) < other.code.charAt(0)){
            return -1;
        }else if(this.code.charAt(0) > other.code.charAt(0)){
            return 1;
        }else{
            if(this.calculatePrice() < other.calculatePrice()) {
                return -1;
            }else if(this.calculatePrice() > other.calculatePrice()){
                return 1;
            }else{
                return 0;
            }
        }
    }

    /**
     * Return the code of the vehicle
     * @return {@link String} The vehicle code
     */
    public String getCode(){
        return code;
    }

    /**
     * Return the mileage of the vehicle
```

```
     * @return int The vehicle mileage
     */
    public int getMileage() {
        return mileage;
    }

    /**
     * Add mileage to the vehicle
     * @param distance The mileage to be added
     */
    public void testDrive(int distance){
        if(distance>0){
            this.mileage += distance;
        }
    }

    /**
     * Convert instance information to a string
     * @return {@link String} string contains instance information
     */
    @Override
    public String toString() {
        Locale locale = new Locale("en", "CA");
        NumberFormat formatter = NumberFormat.getCurrencyInstance(locale);
        return String.format("%-7s Mileage: %skm \n %7sCost:
"+formatter.format(calculatePrice()),this.code,this.mileage,"");
    }
}
```

c. TestDriver.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.text.NumberFormat;
import java.util.Locale;
import java.util.Scanner;

/**
 * Test driver for Vehicle and Dealership classes
 * @author Yulong Wang
 * @date 2021/10/10
 */
public class TestDriver {
    /**
     * Helper function that create a vehicle instance using string input
     * @param line The string contain vehicle information
     * @return {@link Vehicle}
     */
    public static Vehicle createVehicle(String line) throws IndexOutOfBoundsException{
        String[] splited = line.split(" ");
        char carType = splited[0].charAt(0);
        switch (carType){
```

```
            case 'T':
                return new Truck(splited[0], Integer.parseInt(splited[1]),
splited[2]);
            case 'C':
                return new Car(splited[0], Integer.parseInt(splited[1]),
Integer.parseInt(splited[2]), splited[3]);
            case 'V':
                return new Van(splited[0], Integer.parseInt(splited[1]),
Boolean.parseBoolean(splited[2]));
            default:
                return null;
        }
    }

    /**
     * Main function
     * @param args Terminal input
     * @throws FileNotFoundException Test case not found
     */
    public static void main(String[] args) throws FileNotFoundException {
        File file = new File(args[0]);
        Scanner sc = new Scanner(file);
        String name = sc.nextLine();
        int numberOfCars = sc.nextInt();
        sc.nextLine();

        Vehicle[] inventory = new Vehicle[numberOfCars];
        for(int i=0;i<numberOfCars;i++){
            try{
                Vehicle vehicle = createVehicle(sc.nextLine());
                if(vehicle != null){
                    inventory[i] = vehicle;
                }
            }catch (IndexOutOfBoundsException e){
                System.out.println("Input is not valid");
            }
        }

        Dealership dealership = new Dealership(name, inventory);
//        print dealership after init
        System.out.println(dealership.toString());
//        sort inventory
        Sorter<Vehicle> sorter = new Sorter<Vehicle>();
        Vehicle[] sorted = dealership.getInventory();
        sorter.selectionSort(sorted);
//        print output
        Locale locale = new Locale("en", "CA");
        NumberFormat formatter = NumberFormat.getCurrencyInstance(locale);
        System.out.println("Sorted Data: \n");
        System.out.println(dealership.getName());
        for(Vehicle vehicle: sorted){
            System.out.println(String.format("%-7s Price:
```

```
"+formatter.format(vehicle.calculatePrice())+"\n",vehicle.getCode()));
        }
//        search vehicles
        while(sc.hasNext()){
            System.out.println(dealership.findInventory(sc.nextLine()));
        }
    }
}
```

d. Car.java

```
/**
 * @author Yulong Wang
 * @date 2021/10/10
 */
public class Car extends Vehicle{
    /**
     * The base price of car
     */
    final static double defaultPrice = 10000;
    /**
     * The number of doors
     */
    private final int numDoor;
    /**
     * The type as hatchback (H) or trunk (T).
     */
    private final String type;

    /**
     * @param code The code id of the car
     * @param mileage The mileage of the car
     * @param numDoor The number of doors
     * @param type The type of the car.
     */
    public Car(String code, int mileage, int numDoor, String type) {
        super(code, mileage);
        this.numDoor = numDoor;
        this.type = type;
    }

    /**
     * Calculate the price of the car
     * @return double The price of the car
     */
    @Override
    double calculatePrice() {
        double price = defaultPrice;
        if(numDoor == 4){
            price *= 1.05;
        }
        if(type.equals("H")){
```

```
            price += 1000;
        }
        return price;
    }
}
```

e. Van.java

```
/**
 * @author Yulong Wang
 * @date 2021/10/10
 */
public class Van extends Vehicle{
    /**
     * The base price of a van
     */
    final static double defaultPrice = 25000.0;
    /**
     * If the van has an electrical door closure
     */
    private final Boolean eDoorClosure;

    /**
     * @param code The code id of the van
     * @param mileage The mileage of the van
     * @param eDoorClosure If the van has an electrical door closure
     */
    public Van(String code, int mileage, Boolean eDoorClosure) {
        super(code, mileage);
        this.eDoorClosure = eDoorClosure;
    }

    /**
     * Calculate the price of the van
     * @return double The price of the van
     */
    @Override
    double calculatePrice() {
        if(eDoorClosure){
            return defaultPrice*1.15;
        }else{
            return defaultPrice;
        }
    }
}
```

f. Truck.java

```
/**
 * @author Yulong Wang
 * @date 2021/10/10
```

```
 */
public class Truck extends Vehicle{
    /**
     * The base price of the truck
     */
    final static double defaultPrice = 50000.0;
    /**
     * The bed size of the truck: standard, short, or long.
     */
    private final String bedSize;

    /**
     * @param code The code id of the truck
     * @param mileage The mileage of the truck
     * @param bedSize The bed size of the truck
     */
    public Truck(String code, int mileage, String bedSize) {
        super(code, mileage);
        this.bedSize = bedSize;
    }

    /**
     * Calculate the price of the truck
     * @return double The price of the truck
     */
    @Override
    double calculatePrice() {
        switch (this.bedSize){
            case "short":
                return defaultPrice*0.9;
            case "long":
                return defaultPrice*1.1;
            default:
                return defaultPrice;
        }
    }
}
```

3.Test

a. Testcase1.dat

This test case test if the program works at normal condition

```
Cars R Us
6
T4172 0 standard
C3913 1004 2 H
V5532 12980 true
T4908 775 short
V5163 15 false
C3511 4152 4 T
```

```
V5532
C3917
T4908
```

Output:

```
(base) yulongwang@YulongdeMacBook-Pro src % java TestDriver Testcase1.dat
Cars R Us
T4172   Mileage: 0km
        Cost: $50,000.00
C3913   Mileage: 1004km
        Cost: $11,000.00
V5532   Mileage: 12980km
        Cost: $28,750.00
T4908   Mileage: 775km
        Cost: $45,000.00
V5163   Mileage: 15km
        Cost: $25,000.00
C3511   Mileage: 4152km
        Cost: $10,500.00

Sorted Data:

Cars R Us
C3511   Price: $10,500.00

C3913   Price: $11,000.00

T4908   Price: $45,000.00

T4172   Price: $50,000.00

V5163   Price: $25,000.00

V5532   Price: $28,750.00

vehicle found
vehicle not found
vehicle found
```

b. Testcase2.dat

This test case test:

1. Repeated vehicle input
2. Invalid vehicle input

```
Cars R Us
9
T4172 0 standard
T4171 0 long
C3913 1004 2 H
V5532 12980 true
```

```
T4908 775 short
V5163 15 false
C3511 4152 4 T
C3511 4152 4 T
C3510
V5532
C3917
T4908
```

Output:

```
(base) yulongwang@YulongdeMacBook-Pro src % java TestDriver Testcase2.dat
Input is not valid
Cars R Us
T4172   Mileage: 0km
        Cost: $50,000.00
T4171   Mileage: 0km
        Cost: $55,000.00
C3913   Mileage: 1004km
        Cost: $11,000.00
V5532   Mileage: 12980km
        Cost: $28,750.00
T4908   Mileage: 775km
        Cost: $45,000.00
V5163   Mileage: 15km
        Cost: $25,000.00
C3511   Mileage: 4152km
        Cost: $10,500.00

Sorted Data:

Cars R Us
C3511   Price: $10,500.00

C3913   Price: $11,000.00

T4908   Price: $45,000.00

T4172   Price: $50,000.00

T4171   Price: $55,000.00

V5163   Price: $25,000.00

V5532   Price: $28,750.00

vehicle found
vehicle not found
vehicle found
```

c. Testcase3.dat

This test case test sorting for two same condition vehicle.

```
Cars R Us
8
T4172 0 standard
T4171 0 standard
C3913 1004 2 H
V5532 12980 true
T4908 775 short
V5163 15 false
C3511 4152 4 T
C3511 4152 4 H
V5532
C3917
T4908
```

Output

```
(base) yulongwang@YulongdeMacBook-Pro src % java TestDriver Testcase3.dat
Cars R Us
T4172   Mileage: 0km
        Cost: $50,000.00
T4171   Mileage: 0km
        Cost: $50,000.00
C3913   Mileage: 1004km
        Cost: $11,000.00
V5532   Mileage: 12980km
        Cost: $28,750.00
T4908   Mileage: 775km
        Cost: $45,000.00
V5163   Mileage: 15km
        Cost: $25,000.00
C3511   Mileage: 4152km
        Cost: $10,500.00

Sorted Data:

Cars R Us
C3511   Price: $10,500.00

C3913   Price: $11,000.00

T4908   Price: $45,000.00

T4171   Price: $50,000.00

T4172   Price: $50,000.00

V5163   Price: $25,000.00

V5532   Price: $28,750.00
```

```
vehicle found
vehicle not found
vehicle found
```