The University of the West Indies, St. Augustine
COMP 2603 Object Oriented Programming 1
Week 3, Lab 3

Learning Objectives

- Create and use class and instance variables
- Create and use overloaded methods, constructors
- Implement an equals(..) method in a class
- Implement relationships: association, composition
- Use an ArrayList to manage multiple objects

Create a new BlueJ Project called Lab3. Add the Vehicle.java and VehicleDriver.java files provided on myElearning to your project. You may use the Vehicle class you completed in Lab 2.

Part 1: Class and instance variables

1. Modify the **Vehicle** class so that all new vehicle objects are given a unique license plate ID with the form XXXYY where X is an uppercase letter and Y is a digit from 0 to 9. Examples are shown in the sample output in step 3.

    - Add a new class variable **plateNumberCounter** (int): initialised to 1
    - Add a new instance variable **plateID** (String)
    - Add a private mutator method, **setPlateID( )**, that constructs a plateID String by appending 'TAB' to the appropriate number counter value and sets the **plateID** variable. The **plateNumberCounter** increments by 1 each time.
    - Add a public accessor method, **getPlateID( )** for the **plateID** variable.

2. Modify the **Vehicle** toString( ) method to include the plateID in the string description.

```
public String toString(){
    return "VEHICLE TANK CAPACITY: "+ getTankCapacity()
         + " FUEL TYPE: "+ getFuelType()
         + " PLATE ID: "+ getPlateID();
}
```

3. Create a new main class, **StationSimulation**, and use a loop to create 10 **Vehicle** objects and print their details. Use random int values for the **Vehicle** constructor parameters. See here for steps: https://www.baeldung.com/java-generating-random-numbers-in-range

```
//from: https://www.baeldung.com/java-generating-random-numbers-in-range
public static int getRandomNumber(int min, int max) {
    return (int) ((Math.random() * (max - min)) + min);
}
```

Observe that the **plateID** values are correctly incremented and assigned.
Some sample output (will vary due to random numbers) :

```
VEHICLE TANK CAPACITY: 144 FUEL TYPE: gasoline PLATE ID: TAB01
VEHICLE TANK CAPACITY: 72 FUEL TYPE: gasoline PLATE ID: TAB02
VEHICLE TANK CAPACITY: 96 FUEL TYPE: gasoline PLATE ID: TAB03
VEHICLE TANK CAPACITY: 108 FUEL TYPE: gasoline PLATE ID: TAB04
VEHICLE TANK CAPACITY: 20 FUEL TYPE: gasoline PLATE ID: TAB05
VEHICLE TANK CAPACITY: 48 FUEL TYPE: gasoline PLATE ID: TAB06
VEHICLE TANK CAPACITY: 16 FUEL TYPE: gasoline PLATE ID: TAB07
VEHICLE TANK CAPACITY: 44 FUEL TYPE: gasoline PLATE ID: TAB08
VEHICLE TANK CAPACITY: 180 FUEL TYPE: gasoline PLATE ID: TAB09
VEHICLE TANK CAPACITY: 153 FUEL TYPE: diesel PLATE ID: TAB10
```

Part 2: Overloaded methods and constructors

In the **Vehicle** class:

4. Add a new attribute, **vehicleClassification** (int), that stores the vehicle's classification, together with appropriate **accessor** and **mutator** methods (error checking should be done - valid values are shown in Table 1). Add the vehicle classification to the **toString( )** description using the accessor **getVehicleClassification( )**.

5. Add an overloaded accessor **getVehicleClassification( )** method that accepts a **vehicleClassification** (int) value and returns the corresponding type of vehicle according to Table 2. Tip: Use a switch block https://www.w3schools.com/java/java_switch.asp

Vehicle Classification	Type of Vehicle	Assignment
1	Motorcycle	4th parameter in overloaded Vehicle constructor
3	Light motor vehicle	Default in existing Vehicle constructor
4	Heavy motor vehicle	4th parameter in overloaded Vehicle constructor

Table 2

6. Modify the **toString( )** method so that the overloaded accessor method is also invoked

```
public String toString(){
    return "VEHICLE TANK CAPACITY: "+ getTankCapacity()
        + " FUEL TYPE: "+ getFuelType()
        + " PLATE ID: "+ getPlateID()
        + " VEHICLE CLASSIFICATION: "+ getVehicleClassification()
        + " " + getVehicleClassification(this.vehicleClassification);
}
```

Sample output at this point:

VEHICLE TANK CAPACITY: 441 FUEL TYPE: diesel PLATE ID: TAB01 VEHICLE CLASSIFICATION: 3 Light Motor Vehicle

7. Add an overloaded constructor **Vehicle(…)** that accepts a 4th parameter (int) which is used to initialise the vehicleClassification attribute**.** Observe how the original 3-argument constructor is invoked using the keyword 'this'

```
public Vehicle(int length, int breadth, int width, int vehicleClassification){
    this(length, breadth, width);
    setVehicleClassification(vehicleClassification);
}
```

8. Modify the code in the **StationSimulation** class so that the overloaded Vehicle constructor is used. Use random values for the **vehicleClassification** parameter.

Part 3: Object equality: equals(..)

9. In the **Vehicle** class, add the following **equals(Object obj)** method that checks object equality using the vehicle **plateID** and returns true if the plateIDs are equal, false otherwise.

```
public boolean equals(Object obj){
    if(obj instanceof Vehicle){
        Vehicle v = (Vehicle) obj; //casting to type Vehicle
        String otherVehiclePlateID = v.getPlateID();
        boolean result = this.plateID.equals(otherVehiclePlateID); //string equality
        return result;
    }
    return false;
}
```

Pseudocode: 1. If obj is an instance of a Vehicle object then:
- cast obj to a new Vehicle object, v
- compare v's plateID to this.plateID using String equality and return the result

2. Else return false because non-vehicle objects cannot be compared

Part 4: Implement association relationships

10. Modify the supplied **VehicleDriver** class to have 2 **Vehicle** objects, **vehicle1** and **vehicle2**, as private attributes. This sets up an association relationship between a **VehicleDriver** and a **Vehicle** where a driver can drive (up to) two specific vehicles. These should be set to null in the constructor.
11. Add the following toString( ) method to the class:

```
public String toString(){
    return getName() +
    "\n 1. " + vehicle1.toString() +
    "\n 2. " + vehicle2.toString();
}
```

12. Write a method **addVehicle(..)** that accepts a **Vehicle** object and if valid (not null), sets **vehicle1** or **vehicle2** (accordingly if vehicle1 is already initialised) to the supplied vehicle object, and returns true if successful, false otherwise (meaning both variables have been set already). Note: vehicle1 and vehicle2 should be unique.
    Tip 1: Check if an object obj is initialised as follows: if(obj == null)
    Tip 2: Use the equals(..) method to check if vehicle1 has already been set to the suppled vehicle object.

Part 5: Use an ArrayList to manage objects

13. In the StationSimulation class, create two **ArrayLists**: **drivers** and **vehicles** which hold 5 **VehicleDriver** objects and 10 **Vehicle** objects respectively using Generic Types. Some sample code is given below to help you along. Remember to include the import statement at the top of your class:

    import java.util.ArrayList;

```
ArrayList<VehicleDriver> drivers = new ArrayList<VehicleDriver>();
String[] names = {"Lou", "Sue", "Drew", "Koo", "Murphy"};
for(int i = 0; i<5; i++){
    drivers.add(new VehicleDriver(names[i]));
}
}
```

14. Traverse the **drivers** ArrayList and randomly allocate objects from the **vehicles** ArrayList to the driver objects. The code snippet below sets 1 vehicle in the driver object. Tip: use a do-while loop to set the other one.

```
for(VehicleDriver driver: drivers){
    int index = StationSimulation.getRandomNumber(0, 10);
    Vehicle v = vehicles.get(index);
    driver.addVehicle(v);
}
```

Sample output

Lou
1. VEHICLE TANK CAPACITY: 36 FUEL TYPE: gasoline PLATE ID: TAB12 VEHICLE CLASSIFICATION: 1 Motorcycle
2. VEHICLE TANK CAPACITY: 48 FUEL TYPE: gasoline PLATE ID: TAB13 VEHICLE CLASSIFICATION: 4 Heavy Motor Vehicle
Sue
1. VEHICLE TANK CAPACITY: 48 FUEL TYPE: gasoline PLATE ID: TAB16 VEHICLE CLASSIFICATION: 3 Light Motor Vehicle
2. VEHICLE TANK CAPACITY: 27 FUEL TYPE: diesel PLATE ID: TAB14 VEHICLE CLASSIFICATION: 3 Light Motor Vehicle
Drew
1. VEHICLE TANK CAPACITY: 72 FUEL TYPE: gasoline PLATE ID: TAB11 VEHICLE CLASSIFICATION: 4 Heavy Motor Vehicle
2. VEHICLE TANK CAPACITY: 36 FUEL TYPE: gasoline PLATE ID: TAB12 VEHICLE CLASSIFICATION: 1 Motorcycle
Koo
1. VEHICLE TANK CAPACITY: 36 FUEL TYPE: gasoline PLATE ID: TAB12 VEHICLE CLASSIFICATION: 1 Motorcycle
2. VEHICLE TANK CAPACITY: 128 FUEL TYPE: gasoline PLATE ID: TAB19 VEHICLE CLASSIFICATION: 3 Light Motor Vehicle
Murphy
1. VEHICLE TANK CAPACITY: 132 FUEL TYPE: gasoline PLATE ID: TAB15 VEHICLE CLASSIFICATION: 3 Light Motor Vehicle
2. VEHICLE TANK CAPACITY: 72 FUEL TYPE: gasoline PLATE ID: TAB11 VEHICLE CLASSIFICATION: 4 Heavy Motor Vehicle

Extra Exercises

(1) Compare a few **Vehicle** objects using the equals(..) method in the **StationSimulation** class and observe the effects.
(2) In the **VehicleDriver** class, write a method **canDrive(..)** that accepts a **Vehicle** object and returns true if either **vehicle1** or **vehicle2** is equal to the supplied vehicle object, false otherwise. Determine whether a Lou can drive two randomly chosen vehicles from the ArrayList.
(3) In the **Vehicle** class, assign **plateIDs** that start with different characters for the different vehicle classifications: TXX for Heavy motor vehicles, PXX for Light motor vehicles, MXX for Motorcycles.