CST8132 Lab 6

Street Simulation

Due Date and Demonstration

See Blackboard for Due date. Demonstration required.

Lab Objectives

The purpose of this Lab is to put together a solution to a problem from start to finish, with proper documentation and Junit test classes. When this lab is completed, you will have more experience with the following aspects of programming:

- Documentation
  - UML
  - Javadoc
- Solving and coding a Polymorphism programming problem
- JUnit and test classes.

Description

This Lab consists of the following steps:

1. Read the description of the problem
2. Create a UML static class diagram to represent the classes that will solve the problem
3. Use Eclipse to implement your solution.
4. Use Eclipse and JUnit to generate a test class for Car class and the Bicycle class.

Problem Statement

Using suitable tool, create the UML static class diagram for the objects in the programming exercise described below. Be sure your diagram shows inheritance relationships, constructors, attributes, and behaviors, for *all* of the objects, including the Street.

Using Eclipse, write an Object Oriented program incorporating Abstraction, Encapsulation, Polymorphism and Inheritance that uses an array which will be an attribute of a Street object where the array contains two of each of the following Vehicles:

Car

- Name: a String “Car”+index number. I.e (Car0, Car2..)
- Number of wheels: 4
- Current speed: starts at 0 k/h
- Make a noise: “purr” if the speed is 0; or “vroom” if the speed is greater than 0
- Pushing the pedal: causes speed to increase by 10 k/h

Bicycle

- Name: a String “Bicycle”+index number: I.e (Bicycle1, Bicycle3…)
- Number of wheels: 2

- Current speed: starts at 0 k/h
- Make a noise: “sigh” if the speed is 0; or “grunt” if the speed is greater than 0
- pushing the pedal: it depends on the speed: if speed is at least 40 k/h, there is no change; otherwise, it causes the speed to increase by 4 k/h

Write down Java classes, including an *abstract* Vehicle class with *abstract* methods, to represent these vehicles. Implement the speed and number of wheels as *attributes*, and consider the other two (making noise, and pushing the pedal) to be *behaviors*. Implement the behaviors as described above, where a noise is made by *returning* a suitable string to simulate making a noise. The vehicles will not print anything: the Street object will do all the printing.

To simulate the street, a template for the class is provided for you below. After initializing your array with the four vehicles, then simulate time passing with a suitably named method that includes a loop such that in each loop iteration, all of the following things happen:

- every vehicle’s name and speed is printed
- every vehicle makes its noise
- a single random vehicle has its pedal pushed

A template for the `Street` class is provided for you to fill in below.

Your vehicle array is to consist of 4 vehicles, 2 cars and 2 bicycles – each vehicle should have unique index, regardless of whether it is a car or bicycle.

For your random number generator, to get a random number between 0 and N - 1, you may use the nextInt(N) method of the `randomNumbers` object in the template for the Street class below.

```
public class Street {
    private static final Random randomNumbers = new Random();
    private Vehicle[] vehicles;

    public Street(){
        /* your constructor code to create and
           initialize your vehicles array */
    }

    public void simulate(){

        for (int i = 0 ; i < 6 ; i++) {
            /* this is your loop for your simulation */

        }
    }

    public static void main (String[] args) {
        Street thestreet = new Street();
        thestreet.simulate();
    }
}
```

Sample output of program:

```
Update on the street:
Car0, speed: 0
Bicycle1, speed: 0
Car2, speed: 0
Bicycle3, speed: 0
Car0, noise: purr
Bicycle1, noise: sigh
Car2, noise: purr
Bicycle3, noise: sigh
Pedal of Car2 was pushed

Update on the street:
Car0, speed: 0
Bicycle1, speed: 0
Car2, speed: 10
Bicycle3, speed: 0
Car0, noise: purr
Bicycle1, noise: sigh
Car2, noise: vroom
Bicycle3, noise: sigh
Pedal of Car2 was pushed

Update on the street:
Car0, speed: 0
Bicycle1, speed: 0
Car2, speed: 20
Bicycle3, speed: 0
Car0, noise: purr
Bicycle1, noise: sigh
Car2, noise: vroom
Bicycle3, noise: sigh
Pedal of Car2 was pushed

Update on the street:
Car0, speed: 0
Bicycle1, speed: 0
Car2, speed: 30
Bicycle3, speed: 0
Car0, noise: purr
Bicycle1, noise: sigh
Car2, noise: vroom
Bicycle3, noise: sigh
Pedal of Car0 was pushed

Update on the street:
Car0, speed: 10
Bicycle1, speed: 0
Car2, speed: 30
Bicycle3, speed: 0
Car0, noise: vroom
Bicycle1, noise: sigh
Car2, noise: vroom
Bicycle3, noise: sigh
Pedal of Car0 was pushed

Update on the street:
```

```
Car0, speed: 20
Bicycle1, speed: 0
Car2, speed: 30
Bicycle3, speed: 0
Car0, noise: vroom
Bicycle1, noise: sigh
Car2, noise: vroom
Bicycle3, noise: sigh
Pedal of Car0 was pushed
```

Testing with Junit (Junit4)

Now create test classes for your program, using JUnit and Eclipse.

Create a Junit4 test case called "CarTest".

- Create a new instance of a Car.
- Verify its speed is 0.
- Verify its sound is "purr"
- Push pedal.
- Verify its speed is 10.
- Verify its sound is "vroom".

Create a Junit4 test case called "BicycleTest".

- Create a new instance of a Bicycle.
- Verify its speed is 0.
- Verify its sound is "sigh"
- Push pedal.
- Verify its speed is 4.
- Verify its sound is "grunt".

Submission

The submission process for this assignment is the same as before, but be sure your Javadoc output directory, and UML class diagram file (format: jpg, png, or PowerPoint), is included:

1. demonstrate your simulation to your lab instructor.
2. Demonstrate your Junit tests to your lab instructor.
3. submit a zip archive of your deliverables folder: Lastname_Firstname_CST8132_Lab6.zip

Grading Scheme

Demonstration required: 4 marks (without demonstration your assignment receives an overall grade of zero)

Properly written Java code for the simulation described above: 2 marks

Properly written Javadoc comments and Javadoc output: 2 marks

UML class diagram: 2 marks