CSE 271 Lab 5 – JUnit Testing
Spring 2022
Assigned: 2/24/2022
Due: 2/27/2022

Introduction:

In this lab, we will practice writing JUnit tests for a custom class. Specifically, we will utilize the Car.java class from Lab 4 and create JUnit tests to effectively test it rather than the ad-hoc method we used previously. You may use your Car.java solution from Lab 4 or you may use the posted solution (downloadable in the Lab 5 Canvas folder) for this Lab. Also, the Car.java class specification is listed at the bottom of this document for you to reference when needed.

Lab Instructions:

1. Create a project in Eclipse named **Lab5**.
2. Add the Car.java class to your project workspace in the **src** folder. You can copy and paste or drag and drop it into the folder.
3. Right click on the **src** folder and select **New → JUnit Test Case** to create a new Java JUnit test class. Make sure you have the "**New JUnit 4 test**" radio button selected at the top of the window that appears. Next, in the name text box, follow the naming convention discussed in the lecture and name this class **CarTester**. That is, the name of the class we are testing, followed by the word *Tester*. Finally, at the bottom of the window, there is a text box labeled "**Class under test:**". Here, type the name of the class, Car, that we are testing. While this is not necessary for the tests to execute, it is good practice to include this. Now, click "**Next**" to see a list of methods provided in your Car.java class. You can select which methods you want to create a test for by selecting the checkboxes in the window (see which methods you need to include below). This provides a set of test method stubs for you to fill in. Or, you can skip this, click finish, and make the methods yourself. Once you click "**Finish**", it will ask you to add a JUnit library to your project. Click "**OK**" to add it to the project.
4. Your **CarTester** class should now be opened in your Eclipse IDE. If you selected checkboxes for methods to test, you will have a list of method stubs. If not, you will need to make the method stubs yourself. Below is a list of constructors and methods that are **required** for you to test. Please follow the corresponding lecture code for examples on how to write the JUnit tests.
    a. public Car()
    b. public Car(String owner, String make, String model, int yearModel)
    c. public Car(String owner, String make, String model, int yearModel, double fuelLevel)
    d. public Car(String owner, String make, String model, int yearModel, double fuelLevel, int speed, boolean start)
    e. public Car(Car anotherCar)
    f. public boolean accelerate()
    g. public boolean brake()
    h. public boolean isGasTankEmpty()
    i. public boolean sameOwner(Car anotherCar)
    j. public boolean equals(Object o)
    k. public String toString()
    l. public void setYearModel(int yearModel)
    m. public void setFuelLevel(double fuelLevel)
    n. public void setSpeed(int speed)

5. When writing your test cases, make sure you not only test the acceptable cases, but also the unacceptable cases as well. For example, when you test the accelerate() method, you need to think about the normal scenario as well as the edge cases (not enough fuel, speed is already at the maximum, engine is off, etc.).
6. When testing your setters that have validation checks, be sure to include the correct expected exception (IllegalArgumentException).

Submission Instructions:

After you have completed the lab assignment, locate your source code (**CarTester.java**) in your workspace and submit it to the corresponding Lab 5 Canvas folder, not the CODE plugin.

Rubric:

Task	Grade
CarTester	
Test Constructors (3 points each)	15
Test setters with validation checking (5 points each)	15
Test accelerate() method	10
Test brake() method	10
Test isGasTankEmpty() method	10
Test sameOwner() method	10
Test equals(Object o) method	10
Test toString() method	10
Adequate JavaDoc included in the CarTester.java class and followed the Miami University coding guidelines	10
Total	**100**

Below is the Car.java class description from the previous Lab 4 assignment to use as reference.

Class Car:

Create a class named **Car** that has the following private Instance Properties:

- **int yearModel –** The yearModel field is an integer of the car's year. The year cannot be greater than 2022 and smaller than 1885.
- **String make –** The make field is a String of the car's make.
- **String model –** The model field is a String that holds the model of the car.
- **String owner –** The owned field is a String that holds the name of the owner of the car.
- **int speed –** The speed Instance Property is an integer that holds contains the car's current speed in miles per hour (mph). The speed cannot be a negative number. The maximum speed of a car is 250 mph. You have to ensure that the speed is within the valid range when a car accelerates or brakes.
- **double fuelLevel –** A double that holds the current fuel level of the car. This value ranges from 0 to 1.0. You have to make sure the value is always within this.

- **boolean start –** A boolean that is true if the car engine has been turned on and false if it is off.

The **Car** class should contain the following methods:

- **public Car() –** The default (or empty) constructor that initializes the instance properties to the default values (speed to 0, fuel level to 1.0, start to false, year model to 2022, and all strings to "").

- **public Car(String owner, String make, String model, int yearModel) –** A partial constructor that initializes the instance properties using the input parameters. Initialize the speed to 0, start to false, and fuel level to 1.0.

- **public Car(String owner, String make, String model, int yearModel, double fuelLevel) –** A partial constructor that initializes the instance properties using the input parameters. Initialize the speed to 0 and start to false.

- **public Car(String owner, String make, String model, int yearModel, double fuelLevel, int speed, boolean start) –** The workhorse constructor that initializes all instance properties with the input parameters.

- **public Car(Car anotherCar) –** The copy constructor that initializes the instance properties using the values of another Car's instance properties.

- **public boolean accelerate() –** This method increments the car's speed by 4 mph and decreases the car's fuelLevel by 0.05. The car cannot accelerate if the engine is not on. Also, do not accelerate if the car does not have enough fuel (at least 0.05). If the current speed is the maximum (250 mph) then the acceleration will not increase the speed but will decrement the fuel. The method returns true if the car accelerates (i.e., increases the car speed by some amount) and false, otherwise.

- **public boolean brake() –** This method decrements the car's speed by 3 mph. The car must have its engine on in order to apply the brake. The speed cannot be negative as a result of a brake. If you brake when the car's speed is 3 mph or less, then the brake call will reduce the speed to 0 mph. The method returns true if it can apply the brake and reduce the speed by some amount and false, otherwise.

- **public boolean isGasTankEmpty() –** This method returns true if the fuel level of the car is less than 0.05 and false, otherwise.

- **public boolean sameOwner(Car anotherCar) –** This method returns true if the two cars have the same owner and false, otherwise. We assume that each car owner as a unique name.

- **public boolean equals(Object o) –** Override the default Object class equals() method. This method returns true if the two cars have the same make, model, and yearModel and false, otherwise.

- **public String toString() –** Override the default Object class toString(). This method returns a string representation of the car that includes the owner, make, model, yearModel, fuelLevel, and speed of the car. An example return of this method call is as follows. "Owner: Samuel, Make: Toyota, Model: Camry, Year: 2020, Speed: 75, Fuel Level: 0.70".

- **A set of getters and setters for all Instance Properties.** Look at the Week 3 and 4 Lecture Code for examples.