Lab Assignment 05, Object-Oriented Programming, CSE 271, Spring 2020
Department of Computer Science and Engineering, Miami University

Testing a Java Class using JUnit

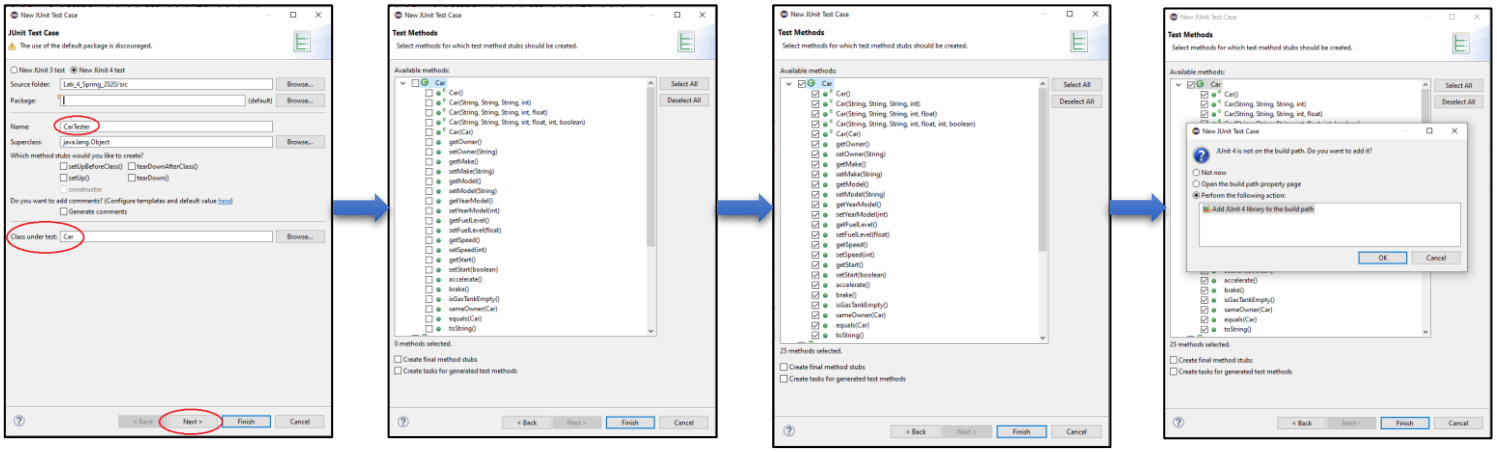
In this lab, you will practice how to test a class through testing its methods using JUnit test library. In the last assignment we tested Car class using a driver class and, in this assignment, we are going to test the same class using JUnit library. You are going to use the class ***Car*** you designed in the last lab (lab 4). You can use the class you designed or use the solution Car.java posted along with this assignment. Car class specification is given in page 4 of this document for your reference.

Preliminary reading [must]:

- JUnit test example we worked together in the lecture [posted on Canvas]
- https://courses.cs.washington.edu/courses/cse143/11wi/eclipse-tutorial/junit.shtml
- https://junit.org/junit4/javadoc/latest/org/junit/Test.html
- https://www.tutorialspoint.com/junit/junit_quick_guide.htm
- https://www.youtube.com/watch?v=ouLxs8UHxjg

Steps:

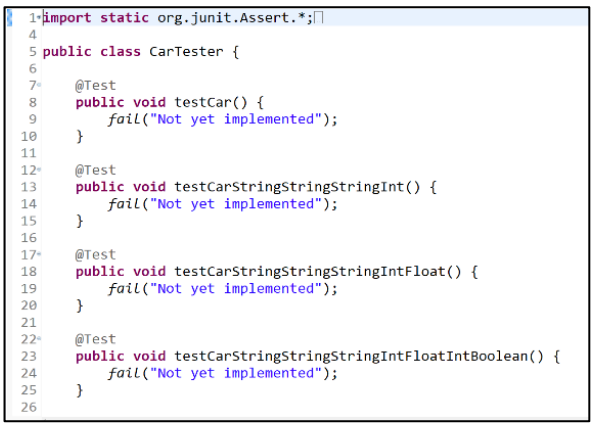
1. Create a project in Eclipse named **Lab_05.**
2. Add already defined Car class to the project. You can drag & drop or copy & paste the defined Car.java file to your project's source ("src") folder.
3. Now, right click on the src folder and select ***"New→JUnit Test Case"*** to create a new java class for JUnit testing. Select JUnit 4 library at the top of the first window. You will have a prompt where you need to provide a name for the test class. The naming convention for the name of the test class is the following… *the name of the class being tested followed by the word "Tester"*. Therefore, the name of class in this case must be **CarTester**. Also, write down the name of the class, Car you are going to test in the textbox labeled "*Class under test:".* Now click button ***"Next"*** to go to the next window where you can select a list of methods to test. After you click "***Next***" button you will see a list of methods from Car class as shown in the middle figure below. Now select all the methods and click "***Finish***". Please review the figures below. It will then ask you to add JUnit library to the project. Click "**OK**" to add it the project as shown below.



Lab Assignment 05, Object-Oriented Programming, CSE 271, Spring 2020
Department of Computer Science and Engineering, Miami University

Testing a Java Class using JUnit

4. You will see CarTester class opened in Eclipse code editor. It will have same number of methods as you selected to test. Get rid of the *fail("Not yet implemented ")* statements from all these methods as you complete those.



```
1  import static org.junit.Assert.*;
4
5  public class CarTester {
6
7      @Test
8      public void testCar() {
9          fail("Not yet implemented");
10     }
11
12     @Test
13     public void testCarStringStringStringInt() {
14         fail("Not yet implemented");
15     }
16
17     @Test
18     public void testCarStringStringStringIntFloat() {
19         fail("Not yet implemented");
20     }
21
22     @Test
23     public void testCarStringStringStringIntFloatIntBoolean() {
24         fail("Not yet implemented");
25     }
26
```

Tester Class:

1. In your CarTester class you have to test all the methods of the Car class including constructors, accessors and mutators. You can create multiple objects of the Car class for testing. When you test you need to make sure that you think about all possible scenarios for each of the methods. For example, when you test accelerate() method, besides testing to see whether a car can accelerate in a normal scenario, you need to think about edge cases: 1) there is not enough fuel, 2) the speed is already the maximum, 3) the engine is off, etc.

2. You will need to add three more test methods to CarTester for *setFuelLevel(float fuelLevel), setYearModel(int yearModel)* and *setSpeed(int speed)* to test whether they throw *IllegalArgumentException* when there is an invalid argument. The goal here is that we are going to try to set invalid values for these variables to check whether these methods throw the specific exception or not. You don't need to use any assert statements for these methods. These tests will automatically pass if methods throw *IllegalArgumentException* when we set an invalid value. Please use the following method header for these methods. There will be a total of 28 test methods including these three.

```
@Test(expected=IllegalArgumentException.class)
public void testSetYearModelException() {
        // set an invalid value in setYearModel() method
}

@Test(expected=IllegalArgumentException.class)
public void testSetFuelLevelException() {
        // set an invalid value in setFuelLevel() method
}

@Test(expected=IllegalArgumentException.class)
public void testSetSpeedException() {
        // set an invalid value in setSpeed() method
}
```

Lab Assignment 05, Object-Oriented Programming, CSE 271, Spring 2020
Department of Computer Science and Engineering, Miami University

Testing a Java Class using JUnit

3. To test constructors, setters and getters we will use a simple technique. Here is an example similar to what we discussed in the lecture. Please review this to write code for testing constructors, setters and getters.

```
// Testing default constructor
@Test
public void testStudent() {
	Student student1 = new Student();
	assertEquals(null, student1.getName());
	assertEquals(null, student1.getMajor());
	// a difference in output value for double variables
	double delta = 0.0000001;
	assertEquals(0, student1.getCgpa(), delta);
}

// Testing workhorse constructor
@Test
public void testStudentStringStringDouble() {
	Student student1 = new Student("Huan", "CS", 3.76);
	assertEquals("Huan", student1.getName());
	assertEquals("CS", student1.getMajor());
	double delta = 0.0000001;
	assertEquals(3.76, student1.getCgpa(), delta);
}

// testing a getter/accessor, getName()
@Test
public void testGetName() {
	Student student1 = new Student("Huan", "CS", 3.76);
	assertEquals("Huan", student1.getName());
}

// testing a setter/mutator, setName()
@Test
public void testSetName() {
	Student student1 = new Student(null, "CS", 3.76);
	assertEquals(null, student1.getName());
	student1.setName("Huan");
	assertEquals("Huan", student1.getName());
}
```

Important Note:
If you are done with the task within the lab then you need to show your work to one of the instructors present in the lab. Make sure the file name is correct and code is well commented. No late submission. You will get zero for the late submission.

Submission:
Submit the java file, CarTester.java to the appropriate submission folder on the Canvas by the due time.

Lab Assignment 05, Object-Oriented Programming, CSE 271, Spring 2020
Department of Computer Science and Engineering, Miami University

Testing a Java Class using JUnit

Grading Rubric:

CarTester	
Constructors (each worth 3 points)	15
Getters/Accessors (each worth 3 points)	21
Setters/Mutators (each worth 3 points)	21
Mutators with Exception (each worth 3 points)	9
accelerate() method	6
brake() method	6
isGasTankEmpty() method	5
sameOwner(Car car) method	6
equals() method	6
toString() method	5
Comments	
[Not Javadoc] Make comments for each test method explaining test strategy	-5
Total	**100**

The Car class from the last assignment for your reference

- **Class Car**:

  Create a class named **Car** which has the following *private* instance variables (fields):
  - **String owner:** This is a String object that holds the name of the owner of the car.
  - **String make:** The make field references a String object that holds the make of the car.
  - **String model:** The model field is a String object that holds the model of the car.
  - **int yearModel**: The yearModel field is an int that holds the car's year model. The year cannot be greater than 2021 and smaller than 1885.
  - **float fuelLevel**: A float that holds the current fuel level of the car. The value of fuel level ranges between 0 and 1.0. You have to make sure the value is always within this range.
  - **int speed**: The speed field is an integer that holds the car's current speed in mile per hour. The speed cannot be a negative number. The maximum speed of a car is 250 mph. You have to make sure that the speed is within the valid range when a car accelerates or brakes.
  - **boolean start**: A boolean that is true if the car engine has been turned on and false, if it is off.

  The class should also have the following methods:
  - **public Car()**
    The *empty* constructor that initializes the instance variables to the default values (speed to zero, fuel level to 1.0, start to false, year model to 2020 and strings to null).
  - **public Car(String owner, String make, String model, int yearModel)**
    A *partial* constructor that initializes the instance variables using the received parameters. Initialize speed to zero, start to false, and fuel level to 1.0.
  - **public Car(String owner, String make, String model, int yearModel, float fuelLevel)**
    Another *partial* constructor that initializes the instance variables using the received parameters. Initialize speed to zero and start to false.
  - **public Car(String owner, String make, String model, int yearModel, float fuelLevel, int speed, boolean start)**

Testing a Java Class using JUnit

The *workhorse* constructor that initializes instance variables using the received parameters.

- **public Car(Car anotherCar)**
  The *copy* constructor that initializes the instance variables using the values of anotherCar's instance variables.
- **Accessor and mutator methods for all instance variables:**
  Important: For each mutator or setter method you need to throw an IllegalArgumentException if the parameter is not within the valid range of values as described above in the instance variables section.
  - public void setOwner(String owner)
  - public void setMake(String make)
  - public void setModel(String model)
  - public void setYearModel(int yearModel)
  - public void setFuelLevel(float fuelLevel)
  - public void setSpeed(int speed)
  - public void setStart(boolean start)
  - public String getOwner()
  - public String getMake()
  - public String getModel()
  - public int getYearModel()
  - public float getFuelLevel()
  - public int getSpeed()
  - public boolean getStart()
- **public boolean accelerate()**
  The method increments the car's speed by 4 miles per hour and decreases the fuel level by 0.05. The car cannot accelerate if the engine is not on. Also, don't accelerate if the car does not have enough fuel (0.05 amount) to do that. If the current speed is the maximum, then acceleration won't increase the speed but will burn fuel. It returns true if it accelerates, increases the car speed by some amount, the car and false, otherwise.
- **public boolean brake()**
  The method decrements the car's speed by 3. You have to make sure that the car's engine is on to apply the brake and the speed cannot be negative. The speed cannot be negative as a result of a break. If you hit break while the car is running at 3 miles/hour or less then hitting the break should reduce the speed to 0 miles/hour. It returns true if it can apply brake and reduce the speed by some amount and false, otherwise.
- **public boolean isGasTankEmpty()**
  The method returns true if the fuel level of the car is less than 0.05. False, otherwise.
- **public boolean sameOwner(Car anotherCar)**
  The method returns true if the two cars have the same owner. False, otherwise. We assume that each person has a unique name.
- **public boolean equals(Car anotherCar)**
  It returns true if the two cars have the same make, model, and yearModel. False, otherwise.
- **public String toString()**
  The method returns a String representing the car object that includes owner, make, model, yearModel, fuelLevel, and speed of the car. When this method is called, it returns a String like "Owner: Samuel, Make: Toyota, Model: Camry, Year: 2020, Speed: 75, Fuel Level: 0.7".