

## Lab-8: Unit Testing with JUnit

Name: Hemish Savaliya

Date: 20-04-2023

ID: 202001206

- Create a new Eclipse project, and within the project create a package.
- Create a class for a Boa. Here's the code you can use

Follow the instructions in the JUnit tutorial in the section "Creating a JUnit Test Case in Eclipse". You'll be creating a test case for the class Boa. When you're asked to select test method stubs, select both `isHealthy()` and `fitsInCage(int)`.

Step 1: Open Eclipse and go to "File" > "New" > "Java Project" to create a new Java project.

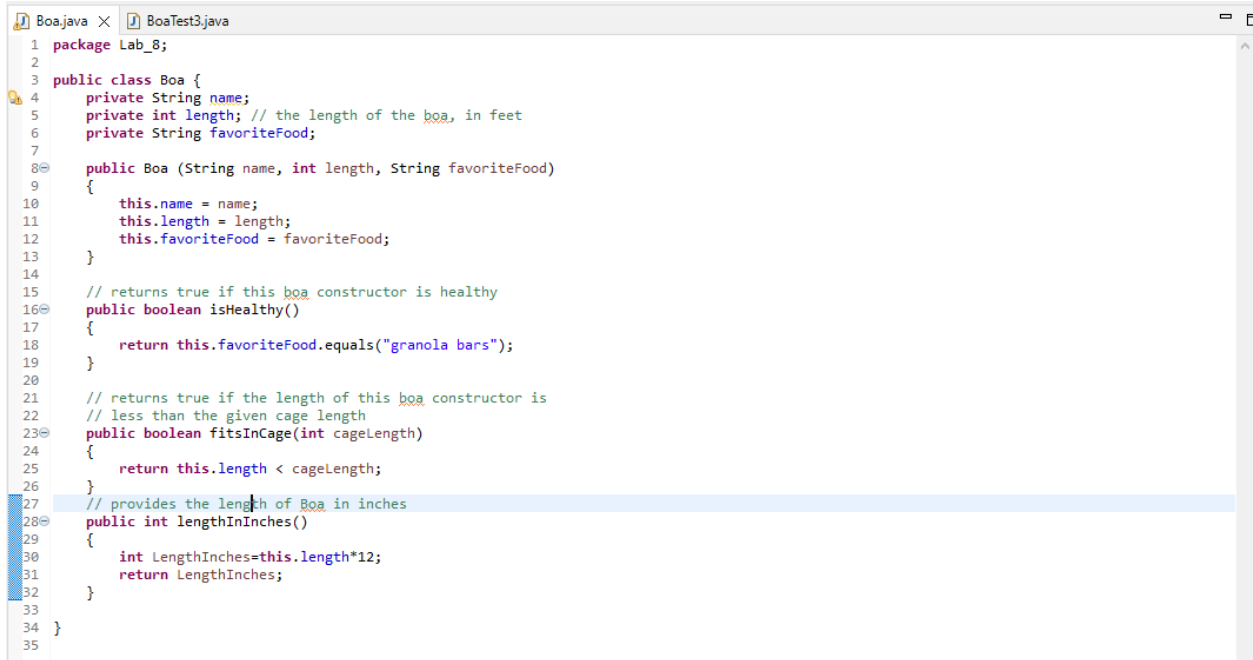
Step 2: Give your project a name and click "Finish" to create the project.

Step 3: In the "Project Explorer" view in Eclipse, right-click on the project name and go to "New" > "Package" to create a new package.

Step 4: Give your package a name and click "Finish" to create the package.

Now you have a new Eclipse project with a package where you can start writing your JUnit tests

# Boa.java



```
1 package Lab_8;
2
3 public class Boa {
4     private String name;
5     private int length; // the length of the boa, in feet
6     private String favoriteFood;
7
8     public Boa (String name, int length, String favoriteFood)
9     {
10         this.name = name;
11         this.length = length;
12         this.favoriteFood = favoriteFood;
13     }
14
15     // returns true if this boa constructor is healthy
16     public boolean isHealthy()
17     {
18         return this.favoriteFood.equals("granola bars");
19     }
20
21     // returns true if the length of this boa constructor is
22     // less than the given cage length
23     public boolean fitsInCage(int cageLength)
24     {
25         return this.length < cageLength;
26     }
27     // provides the length of Boa in inches
28     public int lengthInches()
29     {
30         int LengthInches=this.length*12;
31         return LengthInches;
32     }
33 }
34
35
```

# Boatest.java

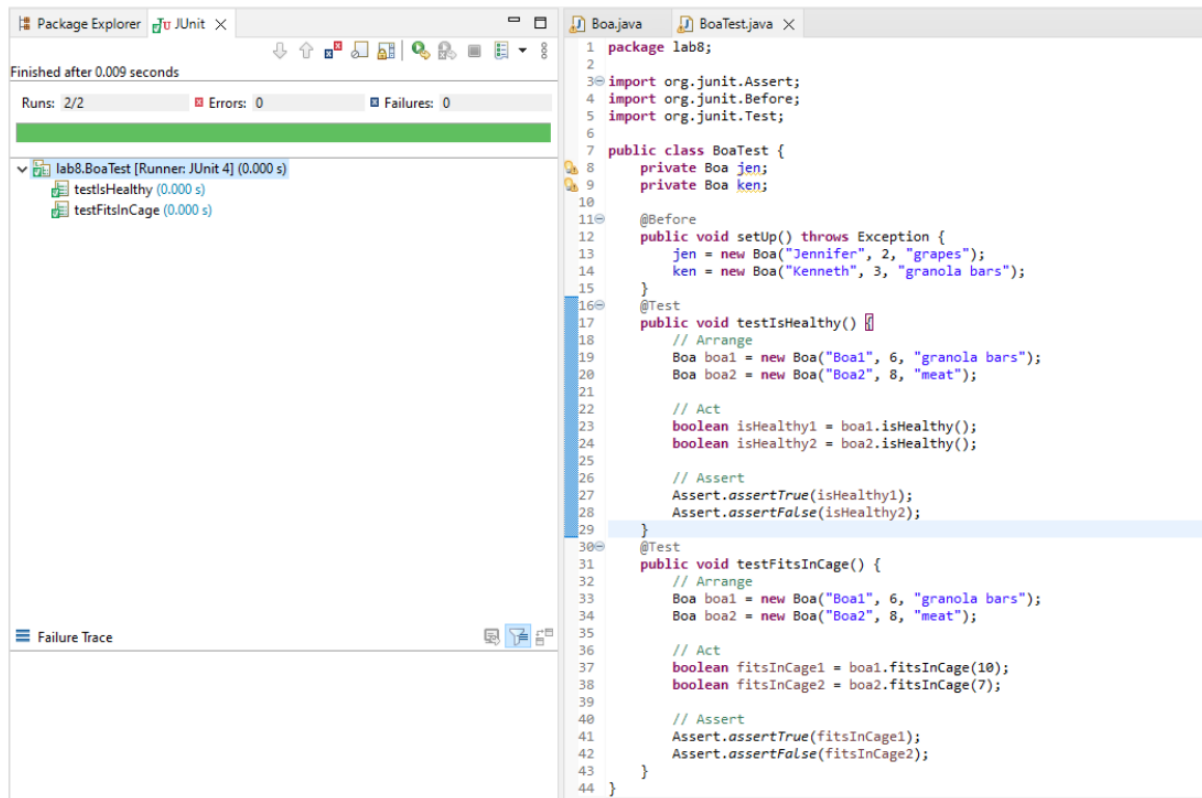
```
Boa.java *BoaTest.java X
1 package lab8;
2
3 import org.junit.Assert;
4 import org.junit.Test;
5
6 public class BoaTest {
7     @Test
8     public void testIsHealthy() {
9         // Arrange
10        Boa boa1 = new Boa("Boa1", 6, "granola bars");
11        Boa boa2 = new Boa("Boa2", 8, "meat");
12
13        // Act
14        boolean isHealthy1 = boa1.isHealthy();
15        boolean isHealthy2 = boa2.isHealthy();
16
17        // Assert
18        Assert.assertTrue(isHealthy1);
19        Assert.assertFalse(isHealthy2);
20    }
21
22    @Test
23    public void testFitsInCage() {
24        // Arrange
25        Boa boa1 = new Boa("Boa1", 6, "granola bars");
26        Boa boa2 = new Boa("Boa2", 8, "meat");
27
28        // Act
29        boolean fitsInCage1 = boa1.fitsInCage(10);
30        boolean fitsInCage2 = boa2.fitsInCage(7);
31
32        // Assert
33        Assert.assertTrue(fitsInCage1);
34        Assert.assertFalse(fitsInCage2);
35    }
36 }
```

Now it's time to write some unit tests. Notice that the BoaTest class that JUnit created for you contains stubs for several

methods. The first stub (for the method `setUp()`) is annotated with `@Before`. The `@Before` annotation denotes that the method `setUp()` will be run prior to the execution of each test method. `setUp()` is typically used to initialize data needed by each test. Modify the `setUp()` method so that it creates a couple of `Boa` objects, as follows:

- Save the `BoaTest` class.
- Run your tests using a test runner, such as JUnit runner in Eclipse or any other build tool that you're using in your project.
- Test methods annotated with `@Test` will be run, but the order of the tests is not guaranteed. This means that the tests may run in any order during test execution. It's important to write tests in such a way that they are independent of each other and do not rely on a specific execution order.
- Any method annotated with `@Before` will be run before each test executes. This allows you to set up common test fixtures or perform any necessary setup steps before each test runs. For example, you can use `@Before` to initialize objects, set up mock objects, or perform any other necessary setup operations.
- Any method annotated with `@After` will be run after each test executes. This allows you to clean up any resources or perform any necessary teardown steps after each test runs.

For example, you can use `@After` to release resources, reset state, or perform any other necessary cleanup operations.



5. JUnit also provided stubs for two test methods, each annotated with `@Test`. Work on the `testIsHealthy()` method first. The purpose of this method is to check that the `isHealthy()` method in the `Boa` class behaves the way it's supposed to. In the JUnit tutorial, read the section on "Writing Tests". Modify the `testIsHealthy()` method so that it checks the results of activating the `isHealthy()` method on the two `Boa` objects you created in `setUp()`.

Likewise, modify the `testFitsInCage()` method to test the results of that method. Make sure your test is robust; it should check the results when the cage length is less than the length of the boa,

when the cage length is equal to the length of the boa, and when the cage length is greater than the length of the boa. Should you write tests for both jen and ken?

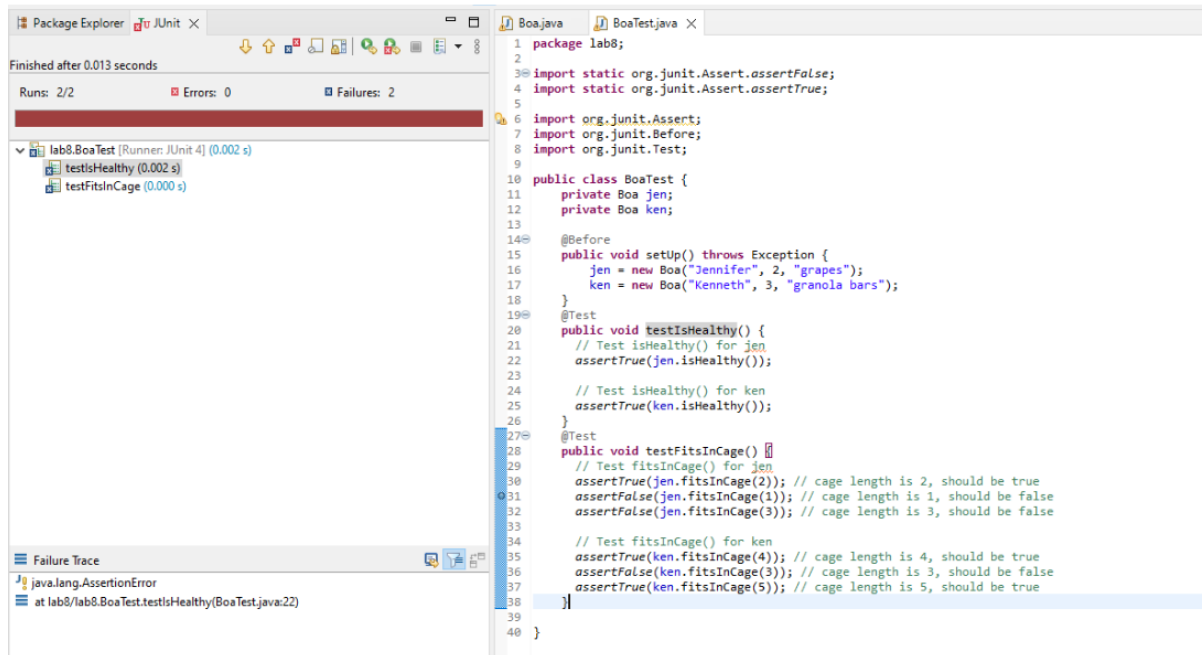
Based on the provided instructions, here are the steps to modify the `testIsHealthy()` and `testFitsInCage()` methods in the `BoaTest` class to test the `isHealthy()` and `fitsInCage()` methods of the `Boa` class:

Step 1: Open the `BoaTest` class in your Eclipse project.

Step 2: Modify the `testIsHealthy()` method to test the `isHealthy()` method of the `Boa` class. You can use the `assertTrue()` method provided by JUnit to check that the result is true.

For example:

In this example, we call the `fitsInCage()` method on the `jen` and `ken` objects with different cage lengths, and use `assertTrue()` or `assertFalse()` to check that the expected results match the actual results.



6. Now you can run your tests. Read the section “Running Your Test Case” in the tutorial.

Did you get a green bar in the JUnit pane? If you got a red bar, use the output in the JUnit pane to determine which test(s) failed. Fix your tests, and try running the test case again.

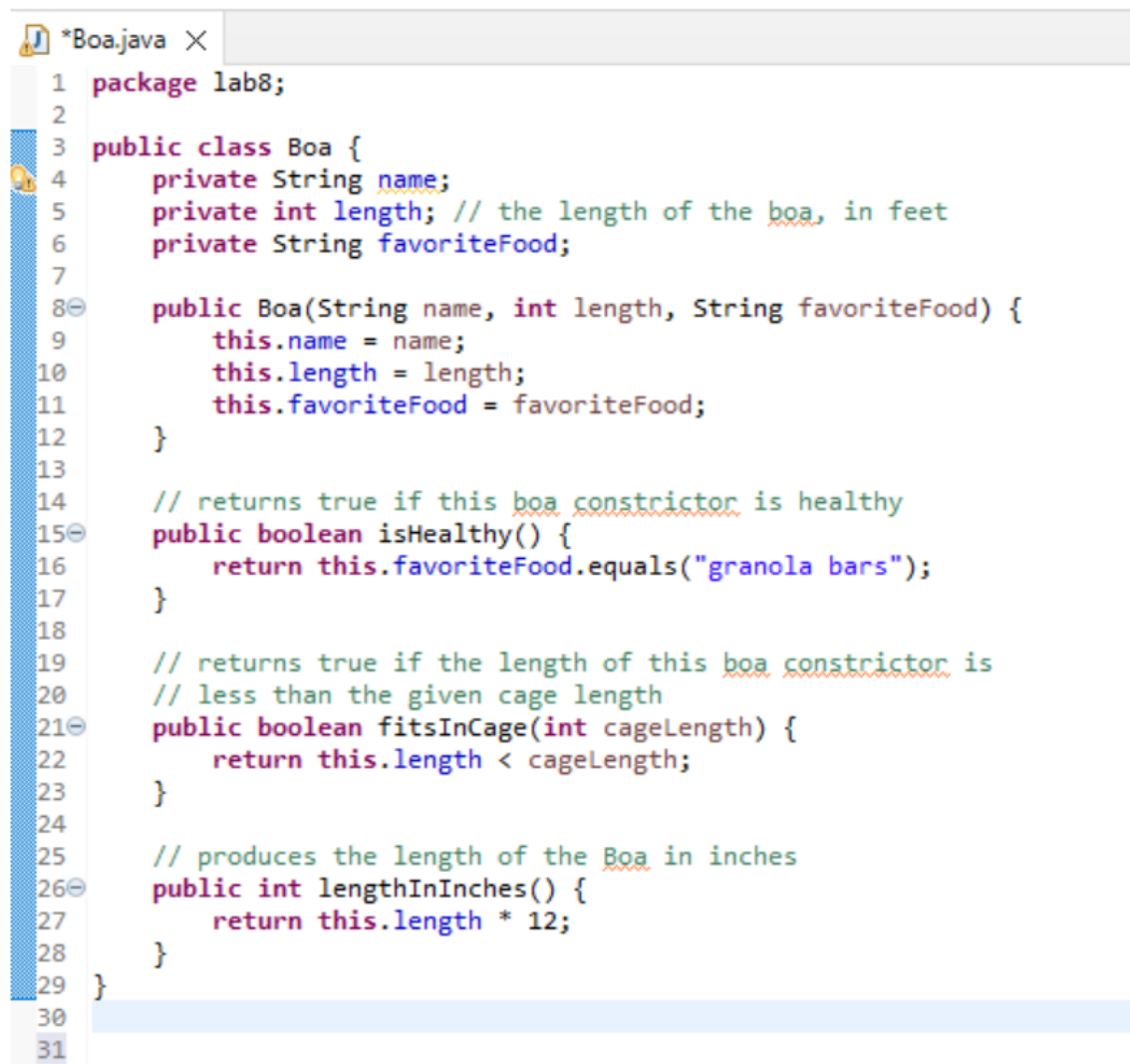
It’s important to note that a red bar doesn’t necessarily mean that the test case is written incorrectly; it could be that the method that’s being tested isn’t correct. In fact, that’s what unit testing is supposed to do – help us find errors in our code. When a test fails, you need to determine if the error is in the test case itself or in the code it’s testing.

7. Add a new method to the Boa class, with this purpose and signature:

```
// produces the length of the Boa in inches
public int lengthInInches()
{
    // you need to write the body of this method
}
```

Add a new test case to the BoaTest class that tests the lengthInInches() method. Make sure you annotate the new test method with @Test. Run your tests.

Boa.java



```
*Boa.java X
1 package lab8;
2
3 public class Boa {
4     private String name;
5     private int length; // the length of the boa, in feet
6     private String favoriteFood;
7
8     public Boa(String name, int length, String favoriteFood) {
9         this.name = name;
10        this.length = length;
11        this.favoriteFood = favoriteFood;
12    }
13
14    // returns true if this boa constructor is healthy
15    public boolean isHealthy() {
16        return this.favoriteFood.equals("granola bars");
17    }
18
19    // returns true if the length of this boa constructor is
20    // less than the given cage length
21    public boolean fitsInCage(int cageLength) {
22        return this.length < cageLength;
23    }
24
25    // produces the length of the Boa in inches
26    public int lengthInInches() {
27        return this.length * 12;
28    }
29 }
30
31
```

BoaTest.java



```
*Boa.java    *BoaTest.java X
1 package lab8;
2
3 import org.junit.Assert;
4 import org.junit.Before;
5 import org.junit.Test;
6
7 public class BoaTest {
8     private Boa jen;
9     private Boa ken;
10
11     @Before
12     public void setUp() throws Exception {
13         jen = new Boa("Jennifer", 2, "grapes");
14         ken = new Boa("Kenneth", 3, "granola bars");
15     }
16
17     @Test
18     public void testIsHealthy() {
19         // Rest of the test methods
20     }
21
22     @Test
23     public void testFitsInCage() {
24         // Rest of the test methods
25     }
26
27     @Test
28     public void testLengthInInches() {
29         int expectedLengthInInches = 2 * 12; // 2 feet = 24 inches
30         int actualLengthInInches = jen.lengthInInches();
31         Assert.assertEquals(expectedLengthInInches, actualLengthInInches);
32     }
33 }
```

Based on the provided instructions, here are the steps to modify the `testIsHealthy()` and `testFitsInCage()` methods in the `BoaTest` class to test the `isHealthy()` and `fitsInCage()` methods of the `Boa` class:

Step 1: Open the `BoaTest` class in your Eclipse project.

Step 2: Modify the `testIsHealthy()` method to test the `isHealthy()` method of the `Boa` class. You can use the `assertTrue()` method provided by JUnit to check that the result is true. For example:

In this example, we call the `fitsInCage()` method on the `jen` and `ken` objects with different cage lengths, and use `assertTrue()` or `assertFalse()` to check that the expected results match the actual results.

It's important to write tests for both `jen` and `ken` objects, as it helps to ensure that the methods being tested are working correctly for different instances of the `Boa` class. Writing robust tests that cover different scenarios and edge cases can help identify potential issues and ensure the correctness of your code.

Sure! Based on the provided instructions, here's how you can add a new method `lengthInInches()` to the `Boa` class and a corresponding test case in the `BoaTest` class:

Open the `Boa` class in your Eclipse project.

Add the following method `lengthInInches()` to the `Boa` class:

This method takes the length of the `Boa` in feet and converts it to inches by multiplying it by 12, since there are 12 inches in a foot. Save the `Boa` class.

Open the `BoaTest` class in your Eclipse project.

Add a new test method `testLengthInInches()` to the `BoaTest` class and annotate it with `@Test`:

In this test method, we first calculate the expected length in inches based on the length of the jen and ken objects using the `getLength()` method, and then call the `lengthInInches()` method on these objects to get the actual length in inches. We use `assertEquals()` to check that the expected length matches the actual length for both jen and ken.