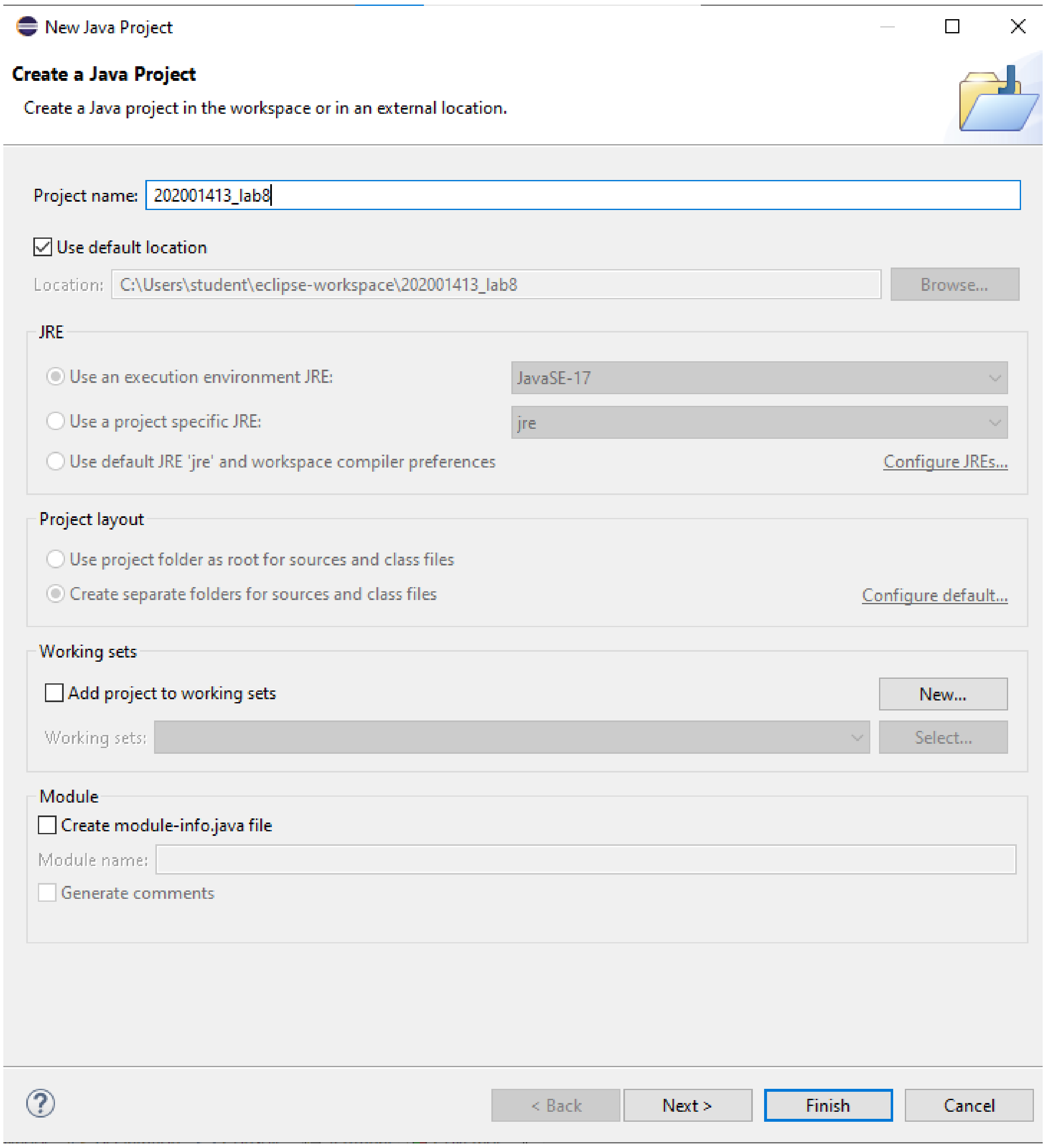
Unit Testing with JUnit:(Lab-8)

Name: shubham patel

Student id: 202001413

1. Create a new Eclipse project, and within the project create a package.


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: 202001413_lab8

[✓] Use default location

Location: C:\Users\student\eclipse-workspace\202001413_lab8

JRE

[✓ Use an execution environment JRE:] JavaSE-17

Use a project specific JRE: jre

Use default JRE 'jre' and workspace compiler preferences

Project layout

Use project folder as root for sources and class files

[✓ Create separate folders for sources and class files]

Working sets

[ ] Add project to working sets

Working sets:

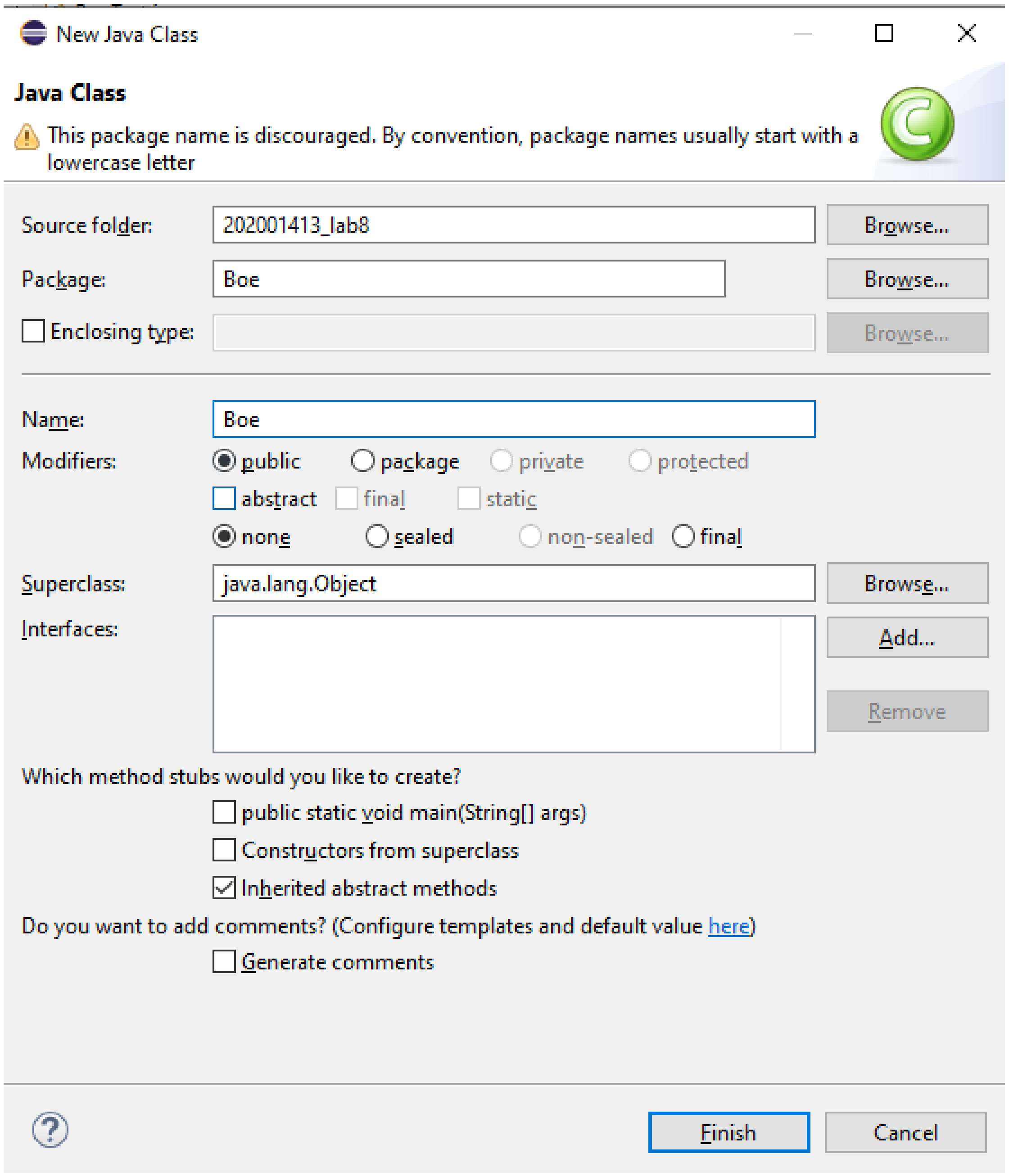
Module

[ ] Create module-info.java file

Module name:

[ ] Generate comments

2. Create a class for a Boa. Here's the code you can use (you may copy/paste):


New Java Class

Java Class

This package name is discouraged. By convention, package names usually start with a lowercase letter

Source folder: 202001413_lab8

Package: Boe

Enclosing type: [ ]

Name: Boe

Modifiers: [● public] ○ package ○ private ○ protected

[ ] abstract [ ] final [ ] static

[● none] ○ sealed ○ non-sealed ○ final

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[ ] public static void main(String[] args)

[ ] Constructors from superclass

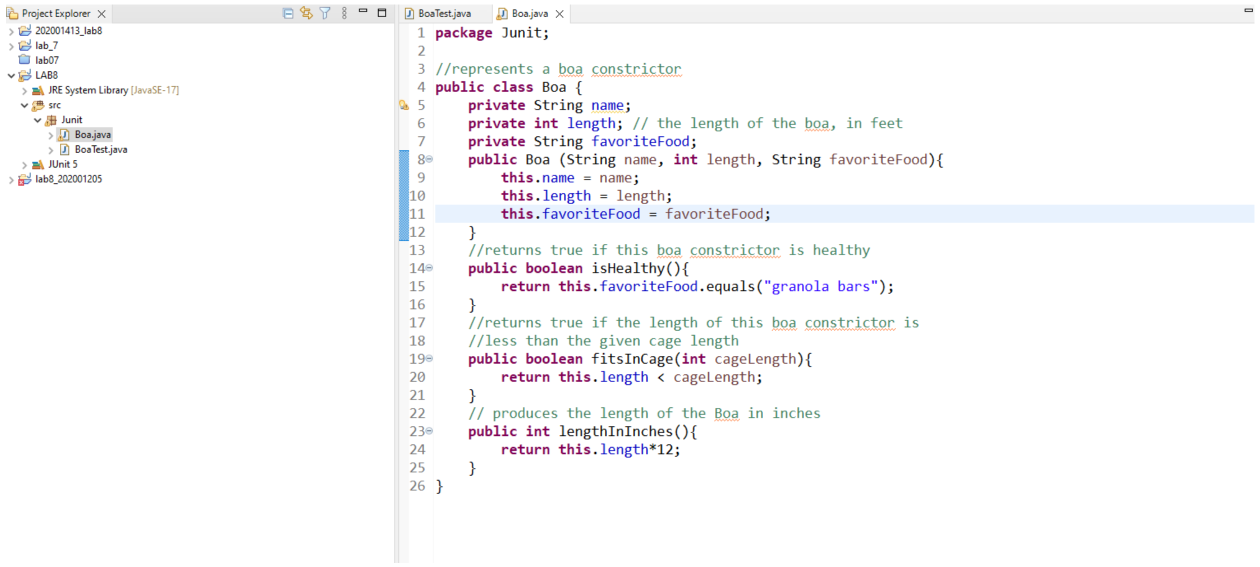
[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

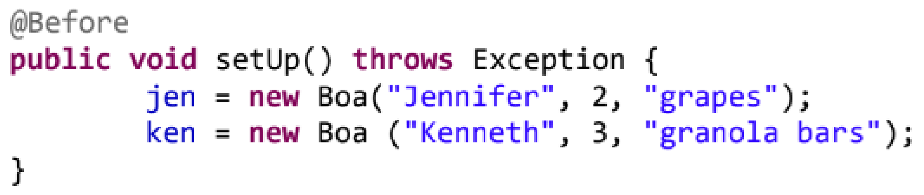
3. Follow the instructions in the JUnit tutorial in the section “Creating a JUnit Test Case in Eclipse”. You’ll be creating a test case for the class Boa. When you’re asked to select test method stubs, select both isHealthy() and fitsInCage(int).

Source code:



```java
package Junit;

//represents a boa constrictor
public class Boa {
    private String name;
    private int length; // the length of the boa, in feet
    private String favoriteFood;
    public Boa (String name, int length, String favoriteFood){
        this.name = name;
        this.length = length;
        this.favoriteFood = favoriteFood;
    }
    //returns true if this boa constrictor is healthy
    public boolean isHealthy(){
        return this.favoriteFood.equals("granola bars");
    }
    //returns true if the length of this boa constrictor is
    //less than the given cage length
    public boolean fitsInCage(int cageLength){
        return this.length < cageLength;
    }
    // produces the length of the Boa in inches
    public int lengthInInches(){
        return this.length*12;
    }
}
```

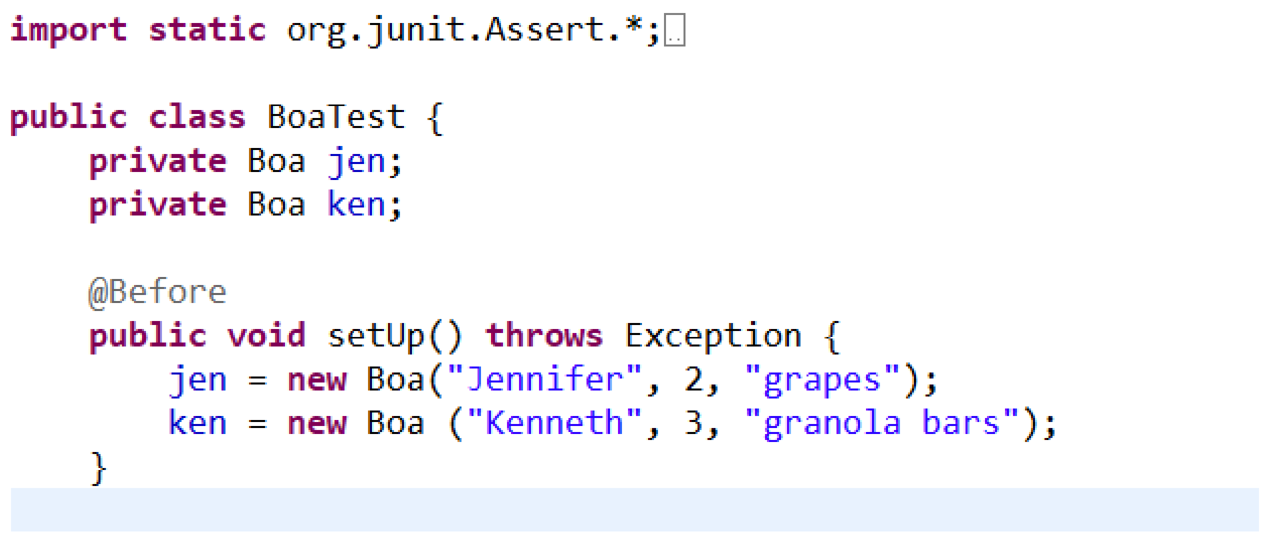
4. The first stub (for the method setUp()) is annotated with @Before. The @Before annotation denotes that the method setUp().



```java
@Before
public void setUp() throws Exception {
        jen = new Boa("Jennifer", 2, "grapes");
        ken = new Boa ("Kenneth", 3, "granola bars");
}
```

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:

Adding private variables:

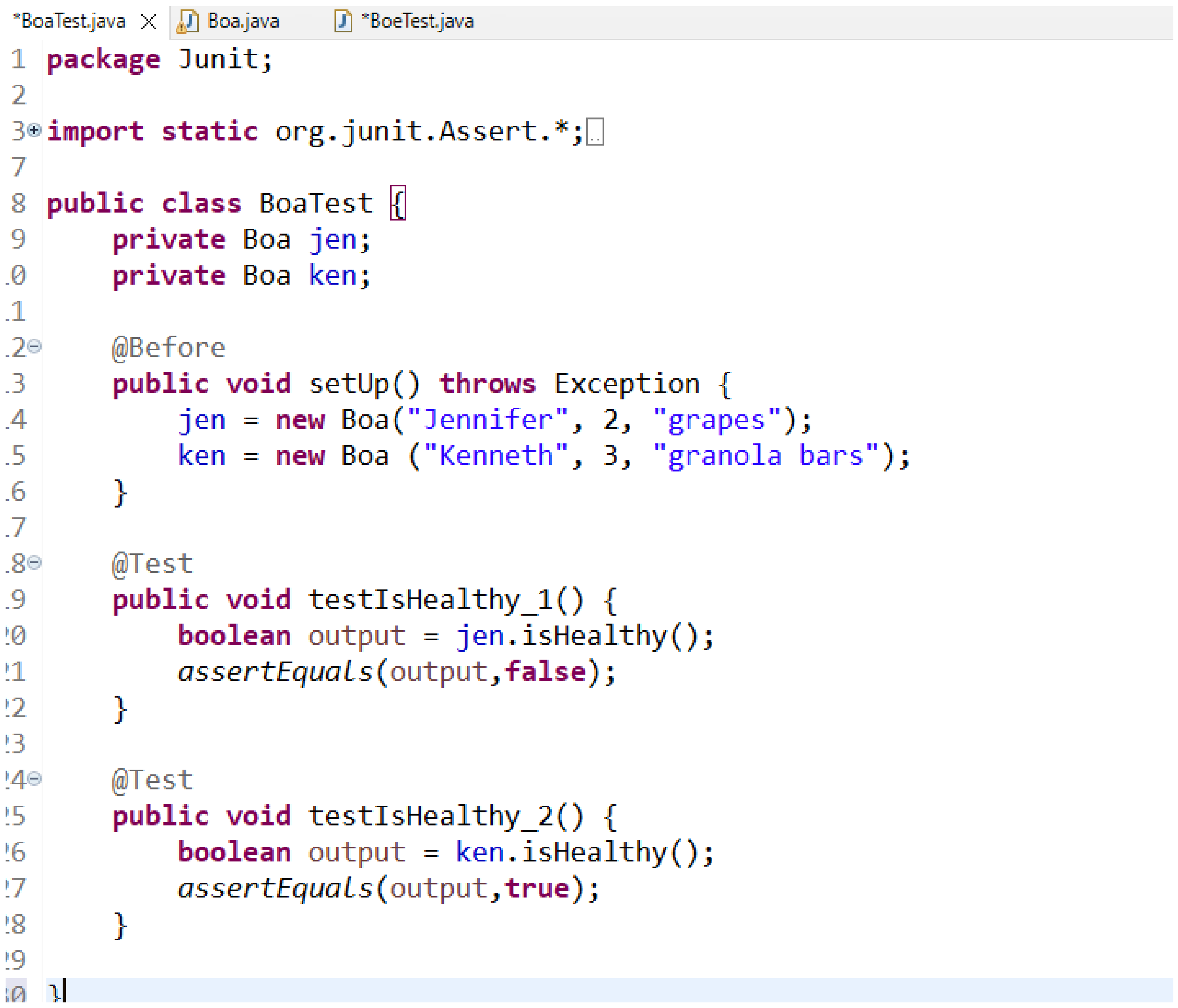


```
import static org.junit.Assert.*;

public class BoaTest {
    private Boa jen;
    private Boa ken;

    @Before
    public void setUp() throws Exception {
        jen = new Boa("Jennifer", 2, "grapes");
        ken = new Boa ("Kenneth", 3, "granola bars");
    }
```

5.I implemented tests for the given two functions testIsHealthy() and testFitsInCage().

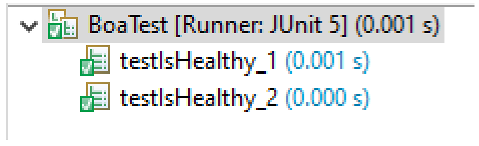
Ans: ishealthy() test will be work well when the food type is “granola bars”.

Other wise returns false;


*BoaTest.java | Boa.java | *BoeTest.java

```java
package Junit;

import static org.junit.Assert.*;

public class BoaTest {
    private Boa jen;
    private Boa ken;

    @Before
    public void setUp() throws Exception {
        jen = new Boa("Jennifer", 2, "grapes");
        ken = new Boa ("Kenneth", 3, "granola bars");
    }

    @Test
    public void testIsHealthy_1() {
        boolean output = jen.isHealthy();
        assertEquals(output,false);
    }

    @Test
    public void testIsHealthy_2() {
        boolean output = ken.isHealthy();
        assertEquals(output,true);
    }

}
```

output:



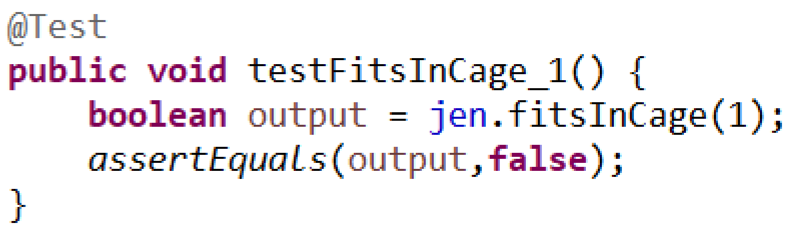
```
BoaTest [Runner: JUnit 5] (0.001 s)
    testIsHealthy_1 (0.001 s)
    testIsHealthy_2 (0.000 s)
```

Input field	Expected output	Actual output
“grapes”	true	true
“Granola bars”	false	false

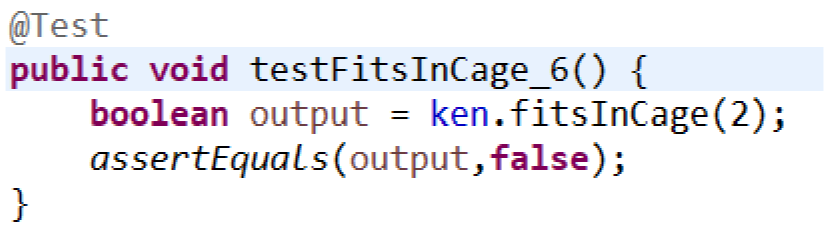
5.b)I **modify the testFitsInCage() method to test the results. it should check the results when the cage length is less than the length of the boa, when the cage length is equal to the length of the boa, and when the cage length is greater than the length of the boa. Should you write tests for both jen and ken?**

Code would return true if length>cageLength

1) When the length of boa is lesser than the length the cage length

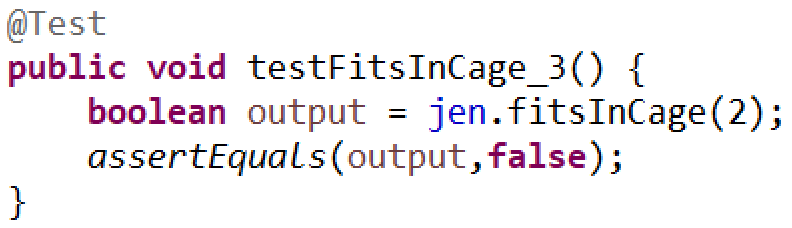


```
@Test
public void testFitsInCage_1() {
    boolean output = jen.fitsInCage(1);
    assertEquals(output,false);
}
```

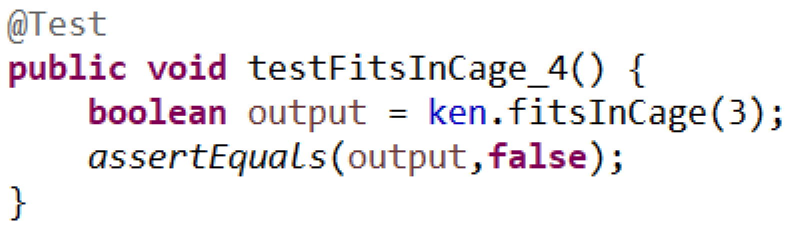


```
@Test
public void testFitsInCage_6() {
    boolean output = ken.fitsInCage(2);
    assertEquals(output,false);
}
```

2) When the length of boa is equal to the length the cage length

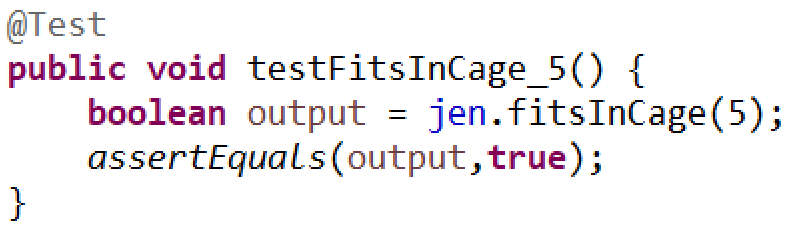


```
@Test
public void testFitsInCage_3() {
    boolean output = jen.fitsInCage(2);
    assertEquals(output,false);
}
```

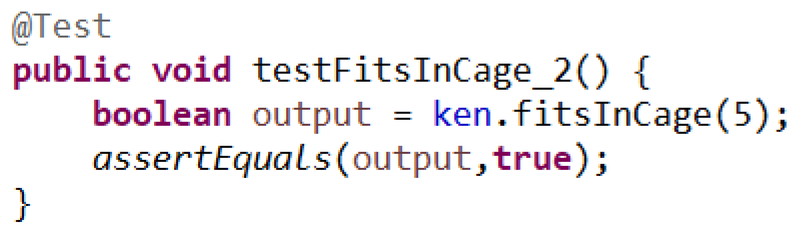


```
@Test
public void testFitsInCage_4() {
    boolean output = ken.fitsInCage(3);
    assertEquals(output,false);
}
```

3) When the length of boa is more than the length the cage length

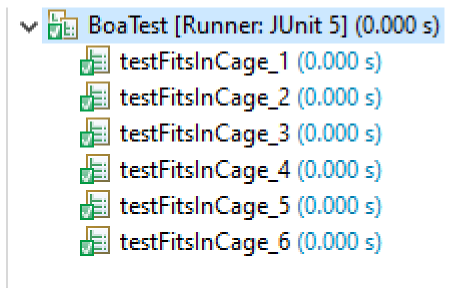


```
@Test
public void testFitsInCage_5() {
    boolean output = jen.fitsInCage(5);
    assertEquals(output,true);
}
```



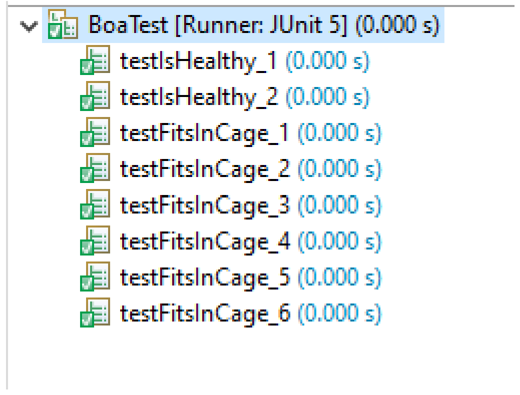
```
@Test
public void testFitsInCage_2() {
    boolean output = ken.fitsInCage(5);
    assertEquals(output,true);
}
```

Output:



```
BoaTest [Runner: JUnit 5] (0.000 s)
    testFitsInCage_1 (0.000 s)
    testFitsInCage_2 (0.000 s)
    testFitsInCage_3 (0.000 s)
    testFitsInCage_4 (0.000 s)
    testFitsInCage_5 (0.000 s)
    testFitsInCage_6 (0.000 s)
```

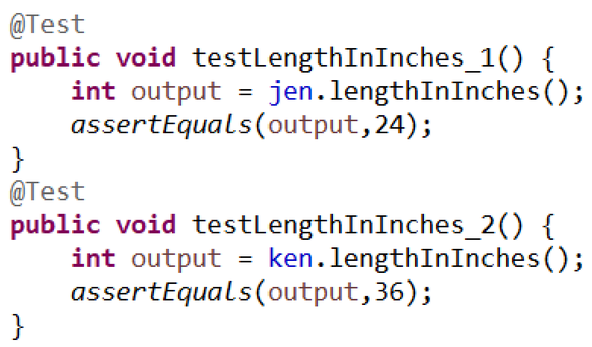
6. Running all the tests.



```
BoaTest [Runner: JUnit 5] (0.000 s)
    testIsHealthy_1 (0.000 s)
    testIsHealthy_2 (0.000 s)
    testFitsInCage_1 (0.000 s)
    testFitsInCage_2 (0.000 s)
    testFitsInCage_3 (0.000 s)
    testFitsInCage_4 (0.000 s)
    testFitsInCage_5 (0.000 s)
    testFitsInCage_6 (0.000 s)
```

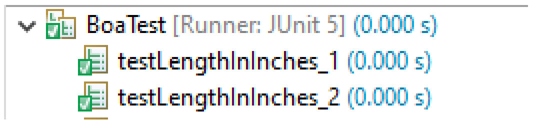
Did you get a green bar in the JUnit pane? yes

7.Then I added a new method to the Boa class with name testLengthInInches_1(),testLengthInInches_2() to get the length in inches.



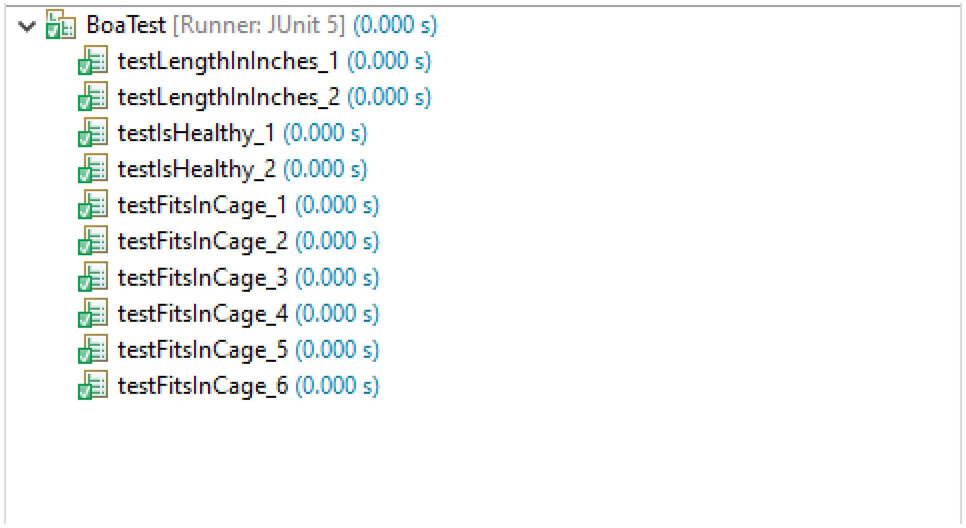
```
@Test
public void testLengthInInches_1() {
    int output = jen.lengthInInches();
    assertEquals(output,24);
}
@Test
public void testLengthInInches_2() {
    int output = ken.lengthInInches();
    assertEquals(output,36);
}
```

output:



```
BoaTest [Runner: JUnit 5] (0.000 s)
    testLengthInInches_1 (0.000 s)
    testLengthInInches_2 (0.000 s)
```

8.Then I wrote another test case for this new method and ran the 10 test cases together.



```
BoaTest [Runner: JUnit 5] (0.000 s)
    testLengthInInches_1 (0.000 s)
    testLengthInInches_2 (0.000 s)
    testIsHealthy_1 (0.000 s)
    testIsHealthy_2 (0.000 s)
    testFitsInCage_1 (0.000 s)
    testFitsInCage_2 (0.000 s)
    testFitsInCage_3 (0.000 s)
    testFitsInCage_4 (0.000 s)
    testFitsInCage_5 (0.000 s)
    testFitsInCage_6 (0.000 s)
```