Software Engineering LAB-8

Rathwa Jayraj
202001102

Unit Testing with JUnit

Provided code

```java
// 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;
    }
}
```

Lab Exercise

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

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

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).

Code for first 3 exercises

```java
@Test public void test()
{
    Boa boa = new Boa("Benny",5,"granola bars");
    assertTrue(boa.isHealthy());
}

@Test public void testIsHealthyWithFavoriteFoodNotGranolaBars()
{
    Boa boa = new Boa("Benny", 5, "mice");
    assertFalse(boa.isHealthy());
}

@Test
public void testFitsInCageWhenLengthLessThanCageLength()
{
    Boa boa = new Boa("Benny", 5, "granola bars");
    assertTrue(boa.fitsInCage(10));
}

@Test
public void testFitsInCageWhenLengthGreaterThanCageLength()
{
    Boa boa = new Boa("Benny", 20, "granola bars");
    assertFalse(boa.fitsInCage(10));
}
```

4. 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

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

(You will need to add private fields for jen and ben to the BoaTest class, otherwise the compiler will complain that there are no variables with those names.)

Code

```
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. 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?

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.

```java
@Test public void testIsHealthy()
{
    Boa jen = new Boa("Jen", 5, "granola bars");
    Boa ken = new Boa("Ken", 6, "mice");

    assertTrue(jen.isHealthy());
    assertFalse(ken.isHealthy());
}
```

```
@Test
    public void testFitsInCage()
    {
        Boa jen = new Boa("Jen", 5, "granola bars");
        Boa ken = new Boa("Ken", 6, "mice");

        assertTrue(jen.fitsInCage(6));
        assertTrue(jen.fitsInCage(5));
        assertFalse(jen.fitsInCage(4));
        assertTrue(ken.fitsInCage(7));
        assertTrue(ken.fitsInCage(6));
        assertFalse(ken.fitsInCage(5));
    }
```

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.

Added Function

```
public int lengthInInches()
{
    return this.length*12;
}
```

Test Case for this function

```java
@Test public void testLengthInInches()
{
    Boa boa = new Boa("John", 5, "grapes");

    int expectedLengthInInches = 60;
    int actualLengthInInches = boa.lengthInInches();

    assertEquals(expectedLengthInInches, actualLengthInInches);
}
```