Unit Testing with JUnit

Goal:

The goal of this lab is to learn how to use JUnit to write unit tests for Java programs.

Preliminary – Learn about JUnit in Eclipse:

Unit testing's fundamental objective is to take the smallest tested component of an application, separate it from the rest of the code, and ascertain whether or not it operates as expected. Before being included into the rest of the programme, each item is tested independently. To put it another way, classes should be tested apart from other classes (test the class's methods before using them elsewhere). Due to the fact that a significant portion of problems are found during unit testing, unit testing has demonstrated its utility.

Lab Exercises:

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):

```
// 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;
   }
}
```

3. Follow the guidelines in the JUnit tutorial's section on "Creating a JUnit Test Case in Eclipse". You will write a test case for the Boa class. When prompted to choose test method stubs, choose both fitsInCage(int) and isHealthy().

4. Start composing some unit tests now. Observe that JUnit included stubs for a number of methods in the BoaTest class it generated for you. The function setUp()'s initial stub is marked with the annotation @Before. The @Before annotation indicates that the method setUp() will be called before each test method is executed. The usual method for initializing the data required by each test is setUp(). Make the following changes to the setUp() function so that it generates a few Boa objects:

```
@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.)

5. Two test methods with the annotation @Test were also given as stubs by JUnit. First, focus on the testIsHealthy() function. This function's objective is to verify that the Boa class's isHealthy() method operates as intended. Read the "Writing Tests" part of the JUnit tutorial. Change the testIsHealthy() function to activate the isHealthy() method on the two Boa objects you generated in setup(), then examine the results.

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.

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.

8. Here are some other things you should know about unit testing and JUnit:
-Each method annotated with @Test will be run, but the order of the tests is not guaranteed.
-Any method annotated with @Before will be run before each test executes.
-Any method annotated with @After will be run after each test executes.