JUnit
Unit Testing with JUnit

Why Unit Testing?
Unit testing is a great way to make sure that your code produces the behavior you intended when you first started writing it. In addition, a JUnit test case in Eclipse can help you to write your class. Finally, unit tests can enable you to verify the continued validity of your code after modifications.

Creating a JUnit Test (and Using It to Create a Java Class)
We will be creating an Employee class that stores an employee by first name, last name, and salary (all of which are passed in to the constructor) and that has three methods: getName() (which returns the employee’s full name), getSalary() (which returns the employee’s salary), and changeSalary(double percentage) (which changes the employee’s salary by a given percentage).

Open Eclipse and start a new Java project, entitled “Employee,” as you learned to do in Tutorial #1 (remember to create separate source and output folders).

Now select File -> New -> JUnit Test Case. You will see the following screen:

Choose Yes.

Enter the Name of your test: “EmployeeTest”
Set your Package as: “cs201.uvaalias.lab3” (replace uvaalias with your UVA alias, e.g., jac4dt)
Check the box for setUp(), which will cause a method to be automatically created for you.

Select Finish.
Now create an Employee for your test case to use by adding the following instance variable:

```java
Employee employee;
```

Instantiate it in your setUp() method, which is called prior to every test, using the following line:

```java
employee = new Employee("Austin", "Powers", 70000.00);
```

Your code should now look like this:

```java
public class EmployeeTest extends TestCase {
    Employee employee;

    /* See TestCase#setUp() */
    protected void setUp() throws Exception {
        employee = new Employee("Austin", "Powers", 70000.00);
        super.setUp();
    }
}
```

We will need three tests, one for each of our methods. Thus, add empty testGetName(), testGetSalary(), and testChangeSalary() methods to your JUnit test. These should be public methods, and they will not return anything:

```java
protected void setUp() throws Exception {
    employee = new Employee("Austin", "Powers", 70000.00);
    super.setUp();
}

public void testGetName() {
    
}

public void testGetSalary() {
    
}

public void testChangeSalary() {
    
}
```

First, let's fill in the testGetName() method. We want this test to retrieve the results of the getName() method and ensure that they are equal to the expected results. Thus, add the following code:

```java
String expected = "Austin Powers";
String actual = employee.getName();

assertEquals(expected, actual);
```

The assertEquals method has several forms, one of which checks for equality between Strings. Another form of the assertEquals method checks for equality between numbers of type double. Because rounding errors can occur with doubles, this form of assertEquals requires a variable representing the allowable margin of error. The method checks that the actual value is contained in the range created by the expected value plus or minus the margin of error. Employee salaries are stored as type double, so we will use this form of assertEquals for our testGetSalary() method:

```java
double expected = 70000.00;
double actual = employee.getSalary();
double marginOfError = 0.0001;

assertEquals(expected, actual, marginOfError);
```
Finally, we need to check our testChangeSalary() method. Let’s increase Austin Power’s salary by 5% and check the result:

```java
double percentIncrease = 5.00;
employee.changeSalary(percentIncrease);

double expected = 73500.00;
double actual = employee.getSalary();
double marginOfError = 0.0001;

assertEquals(expected, actual, marginOfError);
```

Your JUnit test is now complete, but Eclipse detects a number of problems. Part of the problem is that Eclipse detects that you have no Employee class. Perhaps Eclipse can help us create this class. Click on the light bulb to the left of our declaration of employee:

Several options will appear. Select **Create class ‘Employee’**

A window will pop up. Unselect **public static void main(String[] args)**

Select **Finish**.

Now you have an empty Employee class, but Eclipse still detects problems. Go back to EmployeeTest and select the light bulb to the left of your instantiation of your Employee class (i.e., where you call the constructor). Select **Create constructor ‘Employee(String, String, double)’**
Similarly, click on the light bulbs to the left of your getName(), getSalary(), and changeSalary() methods in your EmployeeTest and have Eclipse create these methods for you in Employee. Save your Employee and EmployeeTest classes.

Now go to your EmployeeTest class and select Run -> Run As… -> JUnit Test. A red bar will pop up indicating that some tests (actually, all tests) failed. This is likely because you have not yet added instance variables to your Employee class or filled in that class’s methods. Add firstName, lastName, and salary instance variables to your Employee class. Also add code to your constructor to set these variables and save your class:

```java
public class Employee {
    private String firstName;
    private String lastName;
    private double salary;

    /*
      * @param string
      * @param string2
      * @param d
      */
    public Employee(String string, String string2, double d) {
        firstName = string;
        lastName = string2;
        salary = d;
    }
}
```

Now try running your JUnit tests again. The results should be no better than before, because our methods are still empty. Let’s first fill in the getName() method:

```java
public String getName() {
    return (firstName + " " + lastName);
}
```

Try running the JUnit test case again. This time, you have only two errors, so you have made some progress. Now let’s try to cheat and have our getSalary() method always return 70000:

```java
public double getSalary() {
    return 70000;
}
```

Run the JUnit test case again. Only one failure now remains, and that failure is in testChangeSalary(). Let’s fill in the logic for the corresponding method in Employee:

```java
public void changeSalary(double percentIncrease) {
    salary *= (1 + percentIncrease/100);
}
```

Note that our variable naming is poor here, since percentIncrease could be negative, meaning a decrease. In general, you would want to rename this variable, give your constructor parameters more meaningful names, and add commenting. For the purposes of this tutorial (and this tutorial alone!), let’s ignore these sloppy practices and re-run our EmployeeTest. One problem still seems to exist: we expect 73500 somewhere, but we get 70000.
Double-click on the line below `junit.framework.AssertionFailedError: ...`, and this will take you directly to the assertion where the problem emerges:

The value that `getSalary()` returns seems to be 70000, even though we expect 73500. Thus, our Employee code must contain one of the following two problems:

1. `changeSalary()` must incorrectly change the salary, or
2. `getSalary()` must return the wrong number.

Let’s check our `getSalary()` method first. Recall that our code always returns 70000 here. Let’s replace 70000 with the variable `salary` and run `EmployeeTest` again. This time, we should have no errors. Good job!

Using a Java Class to Create a JUnit Test

Although writing a complete JUnit test before starting the corresponding class is often helpful, the following four-step process is also popular:

1. Write a Java class with empty method stubs, i.e.:
   ```java
   public class Employee {
     public Employee(String string, String string2, double d) {
     }

     public String getName() {
     }

     public double getSalary() {
     }

     public void changeSalary(double percentIncrease) {
     }
   }
   ```
2. Use the skeleton class to generate a basic JUnit test
3. Complete the JUnit test
4. Fill in your class

Let’s see how we would generate a JUnit test from a preexisting class. We will use our Employee class, which we have already completed. Remember that we would normally generate the test for a class skeleton, as specified in step 1, however.

Select `EmployeeTest.java` in your Package Explorer:
Hit delete. Eclipse will ask whether you are sure that you want to delete the file. Choose Yes.

Now right-click on Employee.java in the Package Explorer and select New -> JUnit Test Case

A New JUnit Test Case window will pop up. Select setUp() in the window, since we will need a setUp() method:

Select Next.

In the following window, select getName(), getSalary(), and changeSalary():

Select Finish.
Now you have a skeleton JUnit test. Let’s fill it in. Add our Employee declaration and instantiation as before:

```java
import Employee;

public class EmployeeTest extends TestCase {
    Employee employee;

    @BeforeEach
    protected void setUp() throws Exception {
        employee = new Employee("Austin", "Powers", 70000);
        super.setUp();
    }

    // test methods here...
}
```

Also, fill in your testGetName(), testGetSalary(), and testChangeSalary() methods using the same code as before.

Now run your JUnit test. If the JUnit window does not show up, click on the JUnit tab above the Package Explorer:

![JUnit class structure](image)

All test cases should pass.

Congratulations! You have completed this JUnit tutorial.