Massey University

159.251 Software Design and Construction

Testing Tutorial - JUnit

Prerequisites (what you are expected to know before you come to the tutorial)

1. everything that was a prerequisite in any of the previous tutorials
2. you have read the (very short) JUnit cookbook: http://junit.sourceforge.net/doc/cookbook/cookbook.htm

Objectives

1. write JUnit test cases that fail
2. write tests to verify claims and describe bugs

Description

Download the project from GitHub: https://github.com/jacobstringer/251-testing-tutorial
This project contains the following:

1. a package **`testingTutorial`** with classes **`Lecturer`** and **`Student`**

Your task is to write JUnit5 test cases to verify whether the following claims are true. Remember that since this is a Maven project, you need to place the test files in the `src/test/java/<package>` folder.

This tutorial is easier if you have some basic understanding of propositional logic, covered in 159.271. If you have not covered this or want a refresher, I would recommend reading https://en.wikipedia.org/wiki/Material_implication_(rule_of_inference) or https://www.tutorialspoint.com/discrete_mathematics/discrete_mathematics_propositional_logic.htm, which covers how to change implications into the **and**, **not** and **or** operators we have available in Java.

1. In **`Lecturer`**, the contract between `equals` and `hashCode` is broken - remember that if two objects are equal, it **implies** that they will also have the same hashCode. Find an example of two different objects which break this implication. Write a test case which

assumes that the contract holds, and therefore exposes this flaw (i.e. it will fail, showing that the supplied code is not correct). Write a similar test case for Student that tests the contract (and thus will pass).

Note that in an implication A → B (equal objects (A) imply that they have the same hashcode (B)), it reduces to: ¬A ∨ B.
If we want to show a failing test case, we need the inverse of this: A ∧ ¬B. What is a sensible way of dealing with ¬A?

2. Java contains the class `java.util.IdentityHashMap`. The documentation of this class contains the following warning: *"This class is not a general-purpose Map implementation! While this class implements the Map interface, it intentionally violates Map's general contract, which mandates the use of the equals method when comparing objects. This class is designed for use only in the rare cases wherein reference-equality semantics are required".* This means that when storing a value with a key k1 in a map, it should be possible to lookup the value with another key k2 that is **equal**, but not necessarily identical to k1. However, for `IdentityHashMap`, identity is required. Write a test case(s) that shows how the behaviour of `IdentityHashMap` differs from other `Map` implementations such as `HashMap`. The test for `IdentityHashMap` should **fail**. Write a similar test that uses `HashMap` instead of `IdentityHashMap`. This test case should **succeed**.

3. `java.util.Collections` contains a static method `unmodifiableList` that takes a list and returns an unmodifiable list view. Unmodifiable means that all methods that would change the list throw an `UnsupportedOperationException`. Write a test case that verifies this behaviour for the `add()` method, and a test case to check whether it is still possible to iterate over the unmodifiable list (hint: you can iterate over it if it is an instance of Iterable).

What to Submit

The zipped project folder.

Hints

Contracts often have the form of a simple implication (rule) "if prerequisite then conclusion". Programming languages like Java have no built-in support for this, however, this implication is equivalent to "not prerequisite or conclusion", and this can be easily expressed with standard language constructs (! and | respectively). Once you have constructed such an expression, it is easy to write an assertion using assertTrue.

However, there is an alternative: JUnit has an explicit feature for preconditions (prerequisites) -- assumptions. This is also discussed in the lecture notes. Basically, you can write a contract test as follows: if (assumption) then (assertion).