Massey University

159.251 Software Design and Construction

Tutorial 5 - 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. be able to add JUnit to the build path of an Eclipse project
3. 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 **tutorial-template** for this tutorial from stream and import it into Eclipse. This project contains the following:

1. a package **`nz.ac.cs.sdc.tutorial5`** with classes **`Lecturer`** and **`Student`**

Create a package **`test.nz.ac.cs.sdc.tutorial85`**. In this package, write JUnit 4 test cases to verify whether the following claims are true.

1. In **`Lecturer`**, the contract between `equals` and `hashCode` is broken. I.e., write a test that expresses this contract (by comparing the hash code of two equal instances of **`Lecturer`**) that **fails**. Write a similar test for Student. In Student, hashCode and equals are correctly implemented, so this test case should **succeed**.

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**.

Challenge for good students

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.

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).