Unit testing with JUnit

Validation and Verification
University of Rennes 1

Erwan Bousse (erwan.bousse@irisa.fr)

Last update October 22, 2013

The purpose of this lab session is to manipulate a framework for automatic execution of unit test cases and understand code coverage. You will have to test a Java class using JUnit tests. You will use Maven to call the tools Javadoc and Jacoco to produce reports.

You will find the source code of the class to be tested in the course web page. The class under test is an array list whose interface is given in Figure 1. Before you begin, please answer preliminary questions below and read the instructions.

1 Preliminary questions

The following JUnit (v4) test suite returns *false* after its execution.

```
public class TestLinkedList{
        protected MyLinkedList list;
        @Before
        public void setUp() {
                list = new MyLinkedList();
        }
        @After
        public void tearDown() {
                list = null;
        }
        @Test
        public void testRemove() {
                Object o = new Object();
                list.add(o);
                list.remove(o);
                assertEquals("Testing remove method", list.size(), 0);
        }
}
```

Question 1 *This result cannot prove in any way that the method "remove" is correct. Why?*

Question 2 *Does it prove that there is an error in the method "add"?*

Question 3 *According to the name of the test case, the intent is to test the method "remove". What conditions must be met in order for this test to be effective?*

Question 4 *Does the order of test methods within the class MyLinkedList matter? Why?*

Question 5 *Suppose that we have adequately tested the method `add (Object o)` and that it appears to be correct. We note, by analyzing the code, that it calls the "addBefore" method. Can we also consider this method as sufficiently tested? Why?*

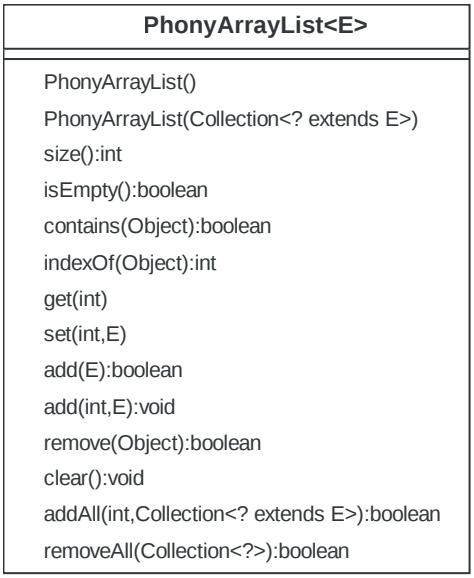

Figure 1: The class to test: PhonyArrayList

2 Unit testing

2.1 Instructions

You have to produce:

- An Eclipse Maven project in zip format (Export... → Archive). It must contain:
  - The source code of the tested/patched system
  - The source code of all your commented tests
  - The generated test report (with javadoc)
  - The generated test coverage report (with jacoco)
- A report in PDF format with:
  - The list of the bugs found (which method, which line, which test could find it)
  - The answers to the questions.

Be careful: each public method must have at least one test case!

2.2 Final questions

Question 6 *If the percentage of code covered by all of your test case is not 100%, can you nonetheless consider your test set as sufficient? Can it be impossible to cover the entire code? If yes, give an example. Do you think that code coverage is a good metric?*

Question 7 *If code coverage is a bad metric, can we do better? Do you know a technique to consider test criterions either good or bad?*