DD2459: Software Reliability

Lab 4: Testing frameworks:
JUnit

Version 1.0
2014-02-24

Introduction:

Test automation frameworks are integrated systems setting up rules for automatic testing of a product. The building blocks of these frameworks simplify the automation effort that would otherwise be manual.

Testing frameworks are responsible for:

1. Establishing the format in which to express the rules
2. Creating the necessary mechanisms to hook into the application to be tested
3. Execute the tests
4. Report the results of the tests

The tests, also known as test suite, can be used in many different forms, both with and without the assistance of a development environment.

The following exercise will help you try a popular open source unit-testing framework for the Java programming language, JUnit, which belongs to the family of unit testing frameworks started by SUnit.

For the purpose of this exercise, we will also be using the popular and open source IDE, Eclipse, which is available on all computers running Ubuntu at CSC labs. The version installed at CSC labs integrates JUnit 3 and 4, but for this exercise we assume that JUnit 4 is to be used.
There is not much configuration to do, but for the sake of clarity, you are encouraged to read the concise tutorial created by Lars Vogel; in particular, the sections 3 and 4 (as of version 2.4).

The zip file (that you can find on the course webpage) has the following files:

- `AllTests.java`
  An executable helper class in case you would prefer using the command line
- `Sorter.java`
  A class implementing a version of the shell sort algorithm (callable with `Sorter.sort(int[])`)
- `SorterTestCases.java`
  The main class where you will implement the test cases for the previous sort algorithm implementation.

Since the method `Sorter.sort()` is static you will **not** have to instantiate the `Sorter` class, and you can use it whenever and wherever you feel like to (note that the implementation of the sorting algorithm will modify the array that you want to have sorted with).

Exercises:

After you have read the documentation about JUnit and get to play around Eclipse's interface for JUnit, you should create a new project (name it as you like) and configure it with JUnit 4.
Once the project has been created, copy/move the three files of the zip file you downloaded on the course's webpage into the `src` folder of the newly created project.
The provided skeleton includes a full implementation of one test in the method `testSortNElementsInRandomOrderList` in the file named `SorterTestCases.java` (and tests whether a list of integers is properly sorted by the implementation of the shell sort algorithm or not).
Do have a look at how the implementation of the test is done.

1. Run all the tests by clicking on `Run As` on the top menu of Eclipse, and then select the option `JUnit Test`.

- **a.** Why the test method `testSortNElementsInRandomOrderList` doesn't fail nor succeed and instead times out?
- **b.** Fix the problem that causes this time out (Hint: look at how the loop counters in the shell sort method behave) and re-run the tests.
  Now the test `testSortNElementsInRandomOrderList` should finish but it fails, why? Explain what happens with the test method and fix the problem that causes the test to fail.

2. Look at the remaining test methods in the class `SorterTestCases`, implement them and test them with the `Sorter.sort()` method.
Note that you do not always have to use the method `Assert.assertTrue()` in your test cases. Try different options instead of defaulting to this method (and you will learn more! :)
3. Can you think of more tests? (Maybe you would like to check your notes and solution of the previous lab for ideas…).

- **a.** Design two more tests you could do, don't forget to implement and test them.
- **b.** Would you remove any of the tests you implemented in the previous task? Why?
- **c.** JUnit allows parameterized tests. How would they help you?

4. The implementation of the `Sorter.sort()` method includes a couple of comments about invariants. Can you test these with JUnit? How?
5. Now that you have a pretty good idea of how automated testing works with JUnit Can you certify that the implementation of the shell sort algorithm is free of errors? What did/do all these tests give you as a developer?

Bibliography:

1. http://junit.sourceforge.net more info on JUnit
2. http://www.vogella.com/articles/JUnit/article.html more info on JUnit setup and usage within Eclipse
3. http://en.wikipedia.org/wiki/Shellsort more info on shell sort algorithm