Assignment 1
White and black box testing

Model-Based Testing
DIT848/GU and TDA260/Chalmers

March, 2014

1 Introduction

The purpose of this assignment is to become familiar with JUnit, one of the most common unit testing frameworks for generating test cases for Java programs, and with coverage analysis, which is one of the most common white box testing methods. JUnit is an open source project, which means you can freely download and use it. You can have more information about it in the following links: http://www.junit.org, http://junit.sourceforge.net. We will also use EclEmma for coverage. EclEmma is an free and open source eclipse plugin available at http://www.eclemma.org/.

In this assignment you are given a specification and an implementation of a simple calculator, using JUnit you will be testing code which implements this calculator and evaluating your test coverage using EclEmma.

2 Submitting your work

You have to submit your work through the Fire system `http://xdat09.ce.chalmers.se/mbt/`. Upload your source code and .txt or .pdf file describing your answers.
The deadline for this assignment is **Wednesday, 9 April 2014**.

3 Structure

The file calculator.zip (available on the course page) contains the following files:

- src/Calculator.java : interface and implementation for a module implementing the functionality of a simple calculator. main function simply takes a string of symbols from the command line and feeds them one by one into the calculator, presenting the calculator output for each symbol.
- src/CalculatorTest.java: example showing three JUnit test cases for the calculator class.

4 Specification

The informal specification for this simple calculator is explained in what follows. The calculator has the following buttons:

- Digits: 0-9
- Operations: +-*/
- Execute: =
- Reset: C

Conceptually, the calculator has the following states holding elements:

- A buffer of digits;
- Possibly an operand (which is an integer);
- Possibly a memorized operation (which is one of add, subtract, multiply, and divide)

The functionality of the calculator is as follows:

- The buffer is continuously displayed on the screen.
- Initially, the buffer is empty, and there is neither an operand nor an operation memorized. Whenever reset is pressed, the calculator goes back to this state.
- When any digit button is pressed, the digit is added to the end of the buffer.
- If an operation button is pressed while there is no operation in memory, then the number currently in the buffer is stored in the operand, and the operand of the button is memorized. The buffer remains unchanged, but at next digit button press,the buffer is emptied before the digit is added to the buffer.
- If = is pressed while there is an operand and a memorized operation, the operand the buffer values are replaced by the sum, difference, product, or quote of the operand and the buffer (depending on which operation is memorized). The memorized operation and operand is hereby deleted.
- If an operation button is pressed while there is an operand and operation memorized, the calculator acts as if = was pressed before the operation button.
- If an operation button is pressed exactly after another operation button, the last one will be used for calculation and the first operation button will be ignored.

5 Report

The program and the test cases are intentionally somewhat inexact. So you may find that the implementation does not fulfill the above specification, or that a test case is testing something not specified. These are your tasks:

1. Make sure that the Calculator class passes all of the existing tests. Otherwise investigate the reasons for failures. Report all failures that you find.
2. In the report you do not need to fix any problem in the Calculator class, instead read the specification carefully and provide further test cases to ensure that the correct implementation of this module conforms to the specification. (As your test cases may also be used to test and evaluate an alternative implementation of this specification, write a comprehensive set of test cases.)
3. Run the test suite using EclEmma. Create more test cases to increase code coverage as much as possible. Does your test suite have full statement and branch coverage? If not, add test cases to get full coverage, or motivate that it is not possible.