SED (CITS2220) Assignment 2, 2011

Verification and Validation

**Aim:**
This aim of this assignment is to develop students’ competence in designing, executing and reporting test cases using a variety of test methods. You will gain experience working as a software tester.

**Task:**
The assignment consists of 2 Parts. For the first part you must implement a java programme to solve at one of two given problems, and give an ad-hoc analysis of its correctness. You may deliberately insert faults into your code, and these should be described in your report. For the second part you should swap your code for part 1 with another student in this unit, and perform three different types of test on the code. For each test you should submit a written report, as described below.

**Working Alone:**
Aside from swapping the source code for Part 1, you will be required to work alone in this project. The work you submit for this assignment must be “the sole effort of an individual student who hands in a piece of work”. The source code submitted for Part 1 must also be an individual effort. For the full policy see http://www.csse.uwa.edu.au/departmental/publications/policy.on.plagiarism.html

**Project Planning:**
This assignment is worth 20% of the total marks for SED. Part 1 is worth only 10% and will not be assessed on correctness, only on the analysis. Each of the three tests in part 2 are worth 30% each. These are not assessed on how many faults they detect, but rather how you design and execute the test, and your analysis of the tests effectiveness. You are expected to spend 20 to 25 hours on this assignment, including background reading and scheduled laboratory times.

**Important Dates:**

- **Submission Deadline: 5pm Tuesday 31st May 2010 (week 13).** Completed assignments (Print outs of source code, and reports on the 4 tasks), together with a completed cover sheet (see attached), should be handed in to the Computer Science Office 1.31A before the deadline. Soft copies of the report, code and test files should also be submitted through: https://secure.csse.uwa.edu.au/run/cssubmit

Please note that the School's late policy of 20% reduction in the mark for each day or part thereof the assignment is late will be applied. See http://www.cs.uwa.edu.au/departmental/publications/latesubmission.html and make sure you get your assignment in on time (even if not finished).
Part I:

Deliverables:
Provide Java code to solve one of the problems below. The method should have the exact method signature described below. Note the source code will not be marked. It is simply required to be an honest attempt to meet the specification given. You should aim to complete this in the first week. You are then required to write a short report (1 page at most) giving the following details

- How does your code work (describe the process and the logic used).
- How would you describe the quality of the code (commenting, formatting, efficiency)
- How confident are you the code is correct (are there any cases that you know it fails).
- How could your code be improved.

You may include deliberate errors in this code, and these should be described in the report. Once you have completed this you should swap your code with another person in the unit, and commence Part II. However, you should not share your report with the other person. If you have difficulty with Java, please see the unit coordinator for help with syntax and the test harness.

Problems

A. Before the invention of July (by Julius Caeser) and August (by Augustus Caeser) there were only 10 months in the year. We will suppose that the odd months (Jan, Mar, May, Sep, Nov) all had 37 days, and the even months (Feb, Apr, Jun, Oct, Dec) all had 36 days, and ignore leap years. Your program is required to take two integers, \(d, m\) representing the day and month in the modern calendar (with 12 months) and produce a string "<month>, <day>" representing the corresponding date in the old calendar, or the string "Invalid date" if the input does not represent a valid date.

Method signature: `public String oldDate(int day, int month)`
Sample Input: 16 4
Sample output: March, 33

B. You are required to write a programme that classifies a triangle, given the lengths of the triangle's three sides, \(a, b,\) and \(c\). To be a triangle, the length of any side must be strictly less than the sum of the other two sides. A triangle is Equilateral if all three side lengths are the same, Isosceles if exactly two side lengths are the same and Scalene otherwise. If a triangle is not equilateral, it must also be classified as acute, right-angled or obtuse. A triangle is Obtuse if the square of one side is greater than the sum of the squares of the other two side, a triangle is Rightangle if the square of one side equals the sum of the squares of the other two sides, and otherwise it is Acute. Given the three side lengths as positive integers your programme must classify the triangle as Not a Triangle, Equilateral, <Obtuse, Rightangle,Acute> Isosceles, or <Obtuse, Rightangle, Acute> Scalene.

Method signature: `public String whichTriangle(int a, int b, int c)`
Sample Input: 4 5 6
Sample output: Acute Scalene
PART II DELIVERABLES:

These tests should be carried on the other students attempted solution to the problems above. The purpose of these tasks is to explore the effectiveness of different testing methods. You should not expect each test method to uncover every possible error.

TASK 1: CODE INSPECTION [30%]
Perform a code inspection on the given class using the Java Code Inspection Checklist (available from SED materials page). Your checklist answers should make reference to the code – both when you are claiming the checklist item is satisfied, and when that item identifies a problem in the code. Mark up the code (with coloured pens) clearly showing errors to be corrected, and making a note of the correction required.
Hand in:
1. A completed checklist clearly marked with the result of each check point and justification with reference to the code as necessary
2. Marked up Java code which has been inspected.
3. Brief report on whether your inspection shows that the program is logically correct or not and whether your inspection show that the program meets good coding practice or not? Justify your answers.

TASK 2: UNIT TESTING USING EQUIVALENCE CLASSES [30%]
Your task is to design black box test cases based on equivalence classes you have identified for the given class. You should consider both expected and invalid inputs (but only inputs meeting the method signatures above should be considered). In selecting the equivalence classes, you should only use the problem description above, and not the actual source code. You should build a set of input cases that cover the equivalence classes you have identified.
Hand in:
1. A description of how you selected test cases,
2. Your test case input file,
3. Brief test report on the results of executing these test cases, and the effectiveness of equivalence class testing as compared to the other methods.

TASK 3: UNIT TESTING USING PATH COVERAGE [30%]
Your task is to design white box test cases to cover the given class. Show how you derived the tests from the code’s structure. Execute your tests and report on the results.
Hand in:
1. A structure diagram for the code
2. A description of how you selected test cases, and what degree of coverage you sought.
3. Your test case input file, and
4. A brief test report on the results of executing these test cases on the code, and a comparison of the error detection properties of path coverage tests, compared with the other test methods considered here.
### CITS2220 Assignment 2 Cover Sheet 2011

(Please fill in your name, student number, and signature and attach this sheet to the front of your Assignment 2 submission)

<table>
<thead>
<tr>
<th>Family name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Given names</td>
<td></td>
</tr>
<tr>
<td>Student Number</td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td></td>
</tr>
<tr>
<td>SUT Author</td>
<td></td>
</tr>
</tbody>
</table>

**Part I: Ad hoc analysis [10%]**
- Genuine attempt to solve the given problem.
- Clear description of the logic behind the method.
- Code quality and correctness analysed and all conclusions justified.

**Part II, Task 1: Code Inspection [30%]**
- Checklist completed and correctness claims justified with reference to the code.
- Test results for each checklist point are clearly presented.
- Code marked up clearly – problems identified, corrections suggested.
- Comment on the type of errors (logical, layout) which were detected.

**Part II, Task 2: Equivalence Class Tests [30%]**
- Equivalence classes well chosen: cover input space, not too many classes.
- Justified choice of equivalence classes and any assumptions made.
- Test results for each executed test case clearly presented, one test cases per class.
- Test cases able to detect faults in the given class, if any.
- Comparative analysis of the test's effectiveness.

**Part II, Task 3: Path Coverage Tests [30%]**
- Flow diagram is a faithful representation of the given method.
- Set of paths giving appropriate path coverage is correctly identified.
- Suitable test data selected to drive the code through all identified paths.
- Test results for each executed test case are clearly presented.
- Comparative analysis of the test's effectiveness.

**MAJOR STRENGTHS**

**MAJOR WEAKNESSES**

**MARKER:**  | **TOTAL MARK**