COMP1035 – Coursework 2 – JUnit Testing

Starting Premise: It turns out that, since the last meeting, mediocre developer Parker has just coded away with no plan or agreement for the project with Winston. When Eliot Spencer – the Quality Assurance Manager – saw her code in Fagan Inspection, he realised the code she produced was terrible in more ways than one – almost as if it as built badly on purpose. Tara Cole has been assigned as Development Team Leader to sort it out, and Parker has been given some marking to do. Tara wants your team to begin devising tests to fix the bad code. Once you have made that plan, she then expects you to start fixing the code by first writing the tests, following proper Test-Driven Development expectations. As Software Quality manager, Eliot also hopes that you will follow some other good development principles, such as detailed commenting (including naming the developers of each function), changed history (as comments in code), paired coding, proper use of version control software – he presumes evidence of these kinds of activities will be evident in any work produced.

Note! As the original scenario was too complex and not enough money was given, Parker automatically simplified the scenario without talking to Winston. Your team does not need to follow the scenario precisely that no need to restore all the requirements and specifications, just follow Parker's ideas and help him refine the existing ones.Of course, even the existing code is written in a mess, full of irregularities and even some cannot run.

Today's Initial Task:

Take the new draft "Class Description" document ("CW2-ClassDescription.pdf") and prepare a rigorous set of tests for how the code should work. After today, the team will be using these tests to improve the code. Therefore, the team should:

- Produce a table of planned tests in markdown, using the markdown template provided ("cw2-testplan.md"), and
- Feel free to add rows, columns and more tables as needed.

The team should also consider using today's lab to set up GitLab for effective teamwork, using e.g., milestones, creating labels, dividing up work using issues, and assigning them to team members. The team is encouraged to use the issue feature to assign the task to the team members, and for every piece of work done (even small fragment), the team can update the issue to inform the team members of his/her progress. Once the issue (or task) has been completed, close the issue. Under the "Issues" in GitLab repository, there are "Boards", "Labels" and "Milestones", the team can refer to the link below for guideline, https://docs.gitlab.com/ee/user/project/issue_board.html

Main Task: The main task for this page, for which the team will be primarily assessed, is the development of good Tests in JUnit. The team has THREE classes to build tests for, which can be downloaded from the Moodle ("CW2-Code.zip").

You should try to build tests according to the team's test plan, but it is normal to realise more tests

as the team starts writing tests and debugging.

- You should document whether tests pass or fail (Result).
- If failed, the team members should document the cause, what was fixed, and when they then passed.
- Further, using proper comments, the team should note who in the team authored each test. Setup Stages:
- Stage 0: A team member creates a Java project in the src folder for everyone else to pull.
- Stage 1 (approximately Lab 07): The team should start by writing tests for the Activity, and User classes.
    - The team may want to finish this before the mid-way lab, so the team has a week to do the subsequent harder work.
- Stage 2 (approximately Lab 08): After finishing the first two classes, the team should move onto Integration Testing, by building JUnit tests for the Main class.

Secondary Task: The team may also start improving and fixing the code.

- The team should use comments in the code, to identify what changes were made, and by whom.

Tertiary Task: The team must submit one peer assessment form, so that I may know who contributed properly to the project.

Again, it is strictly important to do peer assessment for fair marking. If you did not submit any peer assessment, I will not be accepting any argument regarding to the unfair marking to any team members. Also, for the team member(s) that are not contributing, do not expect to get the same mark as your peers do.

Submission to Moodle:

Whole Team:

1. One zip of source code, containing a single Java project, with Java code and JUnit tests, that can be run by the marker. It should be clear in the JUnit and Java code (from comments) who built which code/tests. When working in pairs, both coders should be listed as authors, not just the person with the keyboard.
2. A markdown report of the team's test report, with pass/fail log, identifying tests the date that test passed. If a test fails at first, team member should document why, and what was changed in the code to pass it, especially if tests were changed, or new tests were written, because of these failed tests.

Individual Team Member:

1. Peer assessment form.

Deadline: The deadline for this work is 25th April 2024, 17:00.

Asynchronous Group Work & Timing: This is group work, and you are expected to work on this in

your own time between now and deadline and must coordinate yourselves in your own time.

- You can work “face to face” in the COMP1035 labs (unless told otherwise). These 2 hours lab sessions alone are not enough, alone, to complete this assignment well.
- If you work entirely remotely, you should still meet during the scheduled lab times for 2 hours over e.g., Ms. Teams and work as a remote team in real-time parallel.

Marking Criteria:

In this work, you are being assessed primarily on the team’s ability to write and use Unit tests.

1. The majority of marks are associated with the quality of JUnit tests, including coverage and rigour of tests, and appropriate use of a range of different JUnit features, beyond just the basic @Test with assertEquals().

2. Secondary marks are associated with the detail captured in Test Report, about when and why tests fail, and what changed to make them pass.

3. Tertiary marks are associated with Good Software Engineering Practices.

   a. If (and how) teams fix bad code and document code changes.

   b. How teams use version control.

   c. How teams use other features like raising and resolving issues, in GitLab.

Marking Rubric

Marking Rubric (30%)

Quality of Tests (Total 15%)	0-3 No tests, or very few tested at across the system.	4-7 Some tests at the easiest classes (Transaction class and Category class), and not testing many important aspects, not much variety or usefulness.	8-11 Tests performed across most of the classes/functions. Particularly good range of tests in the easier classes. Simple attempts at the harder user-input ones.	12-15 Comprehensive testing at all aspects of the systems, including harder user input ones in the Main System class. Good variety of testing approaches (using different forms of assert, catching errors, aiming tests at logic in the function).
Test Report (Total 7%)	0 No report submitted.	1-2 Report submitted, but only said that “everything’s now passes” without much/any additional contents.	3-5 Report with details of tests that pass AND fail, including e.g., dates, and what was changed to make them pass, OR Simple type of report, but with good other contents like in the next box.	6-7 Excellent clear report, clear histories for each test of when the failure, and how errors were fixed. Also include other information like team strategy, lists of decisions made, how/if/when they had meetings, if they tried splitting into subgroups etc.
Good Use of Git (Total 5%)	0 No evidence of using git effectively.	1-2 Some attempts of using aspects of git (like issues and branches etc.), but just as a formality, but not really used.	3-4 Good use of git for allocating and managing work across people, using some examples like good use of branches, issues, milestone etc.	5 Good clear team collaboration. Using branches properly for a clear reason, lots of commits, good commit messages, good use of issues, assigned to people, milestones, issue boards, labels, lots of discussion within issues etc.
Fix Changes (Only If Tests Exist, Total 3%)	0 No Java fixed.	1 Simple bits fixed in easy classes, but not well documented as changes using comments (or good commenting but on only minor changes).	2 Good code changes with e.g., authors/dates of changes, and details of changes are clearly documented in the code with comments.	3 Good code changes with clear information of who has done them, when and why, perhaps cross-referencing them to specific tests that found the change. Went as far as restructuring the code (e.g., new methods).

Notes:

Quality of Tests – a few things to look out for:

• Using a variety of types of assert (not just assert equals on everything).

• Trying things like parameterised tests.

• Tests the aim for the logic in the functions (like if statements etc.).

• BASICALLY – clear that its more than simple attempts at obvious tests.

• Also – is the style consistent (like it is a coherent submission from the team) or lots of individual work?

Test Report

• Ultimately, I want to see that the team has gone way beyond the template I gave, e.g.,

1. That they have added additional stuffs to the report in general – team strategy, decisions, and authors of different tasks etc.
2. That they have made "good tables" that have extra columns where they do things like ranking importance? Or marking what tests were supposed to do. And logging what was wrong if they did not pass, and logging what was changed to make them pass etc.

• Basically, if they do both aspects well, then they are in-line for the top marks, if they do neither well, they should be getting low marks.

Java Fix Changes

• Ultimately, I want to be able to see very clearly where Java files have changed, who made the changes and why etc.

• Do they cross reference any tests in their Java code (as part of the why?)

• Is the style of how they comment changes consistent across all the changes?

• Biggest marks would be for "refactoring" where they have improved the structure of the code and commented all about it