**Lab 3 Report**

Name：劉宏德

Student ID：108598004

Date：2020/05/16

1. **Test Plan**
   1. **Test requirements**

The Lab 3 requires to (1) select 6 methods from 6 classes of the SUT (GeoProject), (2) design Unit test cases by using **basis path or graph coverage** technique for the selected methods, (3) develop test scripts to implement the test cases, (4) execute the test scripts on the selected methods, (5) report the test results, and (6) specify your experiences of designing test cases systematically using the graph coverage technique.

In particular, based on the target coverage criteria (i.e., statement, branch, or others), the **test requirements** for Lab 3 are to design test cases *with* **graph coverage technique**for each selected method so that “*each statement and branch (or path) of the method under test will be covered by at least one test case* and *the both minimum* ***statement*** *(node) and* ***branch*** *(edge) coverage are greater than those of Lab 2 and 90%, respectively*.”

* 1. **Test Strategy**

To satisfy the test requirements listed in Section 1, a proposed strategy is to

1. select **3 methods that were chosen in Lab1 or Lab2** and **3 new methods** that are NOT selected previously. The selected methods MUST contain **predicate** and/or **loop** structures (as many as possible).
2. set the objective of the minimum statement or branch (or path) coverage to be greater than that of Lab 2 and adjust the test objective (e.g., 90%, 95% or 100%) based on the time available (if necessary).
3. design the test cases for those selected methods by using the **basis path or graph coverage** testing technique.
   1. **Test activities**

To implement the proposed strategy, the following activities are planned to perform.

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Activity Name** | **Plan hours** | **Schedule Date** |
| 1 | Study GeoProject | 1 | 2020/05/14 |
| 2 | Learn **basis path and graph coverage** | 3 | 2020/05/14 |
| 3 | Draw the graph | 5 | 2020/05/14~2020/05/17 |
| 4 | Design test cases for the selected methods | 5 | 2020/05/14~2020/05/17 |
| 5 | Implement test cases | 5 | 2020/05/14~2020/05/17 |
| 6 | Perform tests and check code coverage. If not satisfy, design more test cases… | 2 | 2020/05/14~2020/05/17 |
| *7* | Complete Lab3 report |  |  |

* 1. **Design Approach**

The **basis path and graph coverage** technique will be used to design the test cases. Specifically, the control flow graph (CFG) of each selected method shall be drawn first, and the possible test paths that satisfy the test requirements (i.e., **statement (node), branch (edge), or path coverage**) shall be derived from the CFG. The possible **inputs** and **expected outputs** for the derived test paths shall be computed from the specification of SUT for each method under test. *Add more test cases by considering to satisfy other coverage criteria, such as edge-pair, all-use, or prime-path coverage criteria.*

* 1. **Success criteria**

All test cases designed for the selected methods must pass and *both statement and branch (or path) coverage should have achieved at least 90%, respectively.*

1. **Test Design**

To fulfill the test requirements listed in section 1.1, the following methods are selected and corresponding test cases are designed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Class** | **Method** | **CFG** | **Basis Path** | **Inputs** | **Expected Outputs** |
| 1 | Base32 | encodeBase32(long I, int length) | 見excel | P1：{1, 2, 3, 4, 5, 4, 6, 7, 8}  P2：{1, 2, 3, 4, 6, 7, 9}  P3：{1, 2, 4, 6, 7, 8} | 1. i=33, length = 0 2. i = 10, length = 0 3. i = -10, length = 0 | 1. “11” 2. “b” 3. “-b” |
| 2 | Base32 | decodeBase32(String hash) | 見excel | P1：{1, 2, 3, 4, 6, 7, 3, 8, 9, 10}  P2：{1, 2, 3, 4, 5}  P3：{1, 2, 3, 8, 10} | 1. hash = “-1b” 2. hash = “a” 3. hash = “” | 1. -42 2. Illegal Argument Exception 3. 0 |
| 3 | GeoHash | adjacentHash(String hash, Direction direction) | 見excel | P1：{1, 3, 5, 6, 8, 9, 10, 11}  P2：{1, 3, 5, 6, 8, 9, 11}  P3：{1, 3, 5, 6, 7}  P4：{1, 3, 4}  P5：{1, 2} | 1. hash = “2b”, direction = RIGHT 2. hash = “11”, direction = RIGHT 3. hash = “x”, direction = RIGHT 4. hash = “”, direction = RIGHT 5. hash = null, direction = RIGHT | 1. “30” 2. “13” 3. “8” 4. Illegal argument exception 5. Illegal argument exception |
| 4 | GeoHash | adjacentHash(String hash, Direction direction, int steps) | 見excel | P1：{1, 2, 1, 3, 4, 5, 8, 9, 5, 10}  P2：{1, 3, 4, 5, 6}  P3：{1, 3, 4, 5, 7}  P4：{1, 3, 4, 5, 10} | 1. hash = “2b”, direction = LEFT, steps = -1 2. hash = “”, direction = LEFT, steps = 1 3. hash = null, direction = LEFT, steps = 1 4. hash = “2b”, direction = LEFT, steps = 0 | 1. “30” 2. Illegal argument exception 3. Illegal argument exception 4. “2b” |
| 5 | GeoHash | encodeHash(double latitude, double longitude, int length) | 見excel | P1：{1, 3, 5, 7}  P2：{1, 3, 5, 6}  P3：{1, 3, 4}  P4：{1, 2} | 1. latitude = 90, longitude = 90, length = 3 2. latitude = 90, longitude = 90, length = 13 3. latitude = 91, longitude = 90, length = 3 4. latitude = 90, longitude = 90, length = 3 | 1. “ypb” 2. Illegal argument exception 3. Illegal argument exception 4. Illegal argument exception |
| 6 | CoverageLongs | getHashLength() | 見excel | P1：{1, 2}  P2：{1, 3} | 1. CoverageLongs(hashes = [5, 9, 1], count = 0, ratio = 1.1) 2. CoverageLongs(hashes = [5, 9, 1], count = 1, ratio = 1.1) | 1. 0 2. 5 |
| 7 | Base32 | getCharIndex(char ch) | 見excel | P1：{1, 2, 3}  P2：{1, 2, 4} | 1. ch = ‘a’ 2. ch = ‘b’ | 1. illegal argument exception 2. 10 |
| 8 | Base32 | padLeftWithZerosToLength(String s, int length) | 見excel | P1：{1, 2, 3, 4, 5, 6, 4, 7}  P2：{1, 8} | 1. s = “1”, length = 2 2. s = “11”, length = 1 | 1. “01” 2. “11” |

The details of the design are given below:

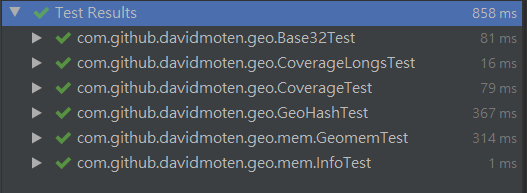
The Excel file of test cases…

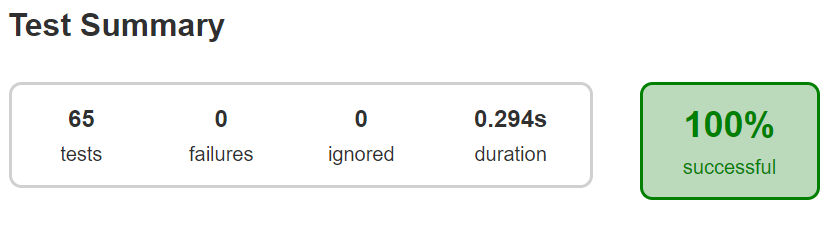
1. **Test Implementation**

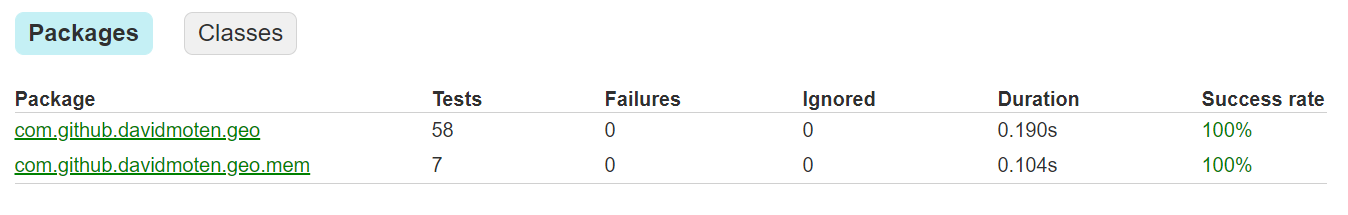
The design of test cases specified in Section 2 was implemented using JUnit 4. The test scripts of 3 selected test cases are given below. The rest of the test script implementations can be found in the [link](https://github.com) (or JUnit files).

|  |  |  |
| --- | --- | --- |
| **No.** | **Test method** | **Source test code** |
| 1 | enodeBase32() |  |
| 2 | decodeBase32() |  |
| 3 | adjacentHash() |  |

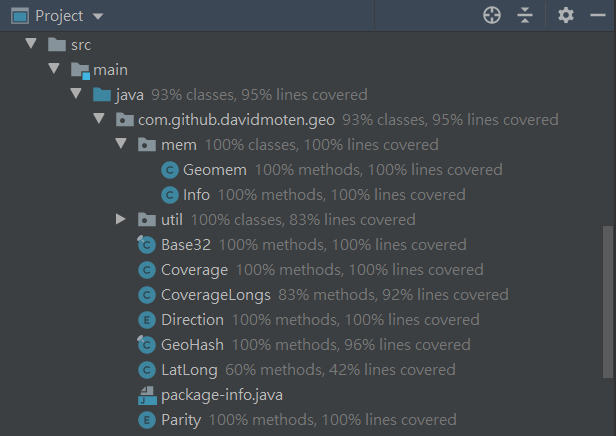
1. **Test Results**
   1. **JUnit test result snapshot**



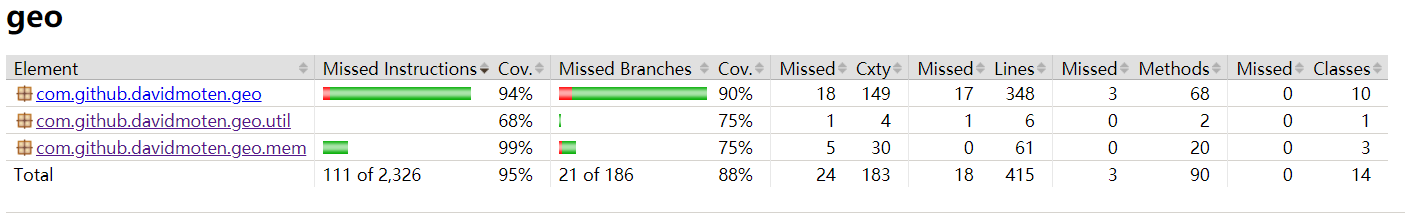




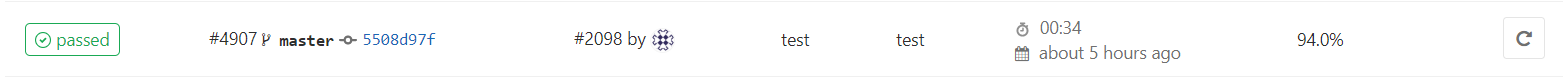
* 1. **Code coverage snapshot**
* Coverage of each selected method under test



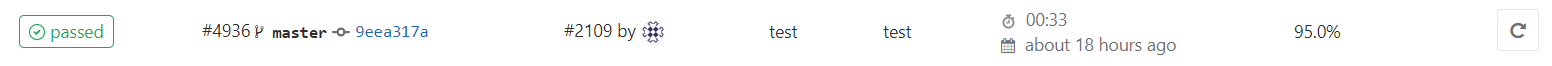
* Total coverage



* 1. **CI result snapshot (3 iterations for CI)**
* CI#1



* CI#2



* CI#3
* CI Pipeline



1. **The Coverage Comparison**

The code coverage of Lab1 (and/or Lab2) and Lab3 are listed in the below Table. The results show that the statement and branch coverage are increased from X% to Y% in Lab3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Test method** | **Lab1 (or Lab2)** | | **Lab3** | |
| **statement coverage** | **branch coverage** | **statement coverage** | **branch coverage** |
| 1 | encodeBase32(long i, int length) | 100% | | 100% | |
| 2 | decodeBase32(String hash) | 100% | | 100% | |
| 3 | adjacentHash(String hash, Direction direction) | 100% | | 100% | |
| 4 | adjacentHash(String hash, Direction direction, int steps) | 100% | | 100% | |
| 5 | encodeHash(double latitude, double longitude, int length) | 0% | | 100% | |
| 6 | getHashLength() | 100% | | 100% | |
| 7 | getCharIndex() | 100% | | 100% | |
| 8 | padLeftWithZerosToLength() | 89% | | 100% | |

1. **Summary**

In Lab 3, **8 test cases have been designed and implemented using JUnit and the basis path/graph coverage technique**. The test is conducted in 3 CI and **the execution results of the 8 test methods are all passed**. **The total statement and branch coverage of the test are 100%, respectively.** Thus, the test requirements described in Section 1 are satisfied.