**Lab 2 Report**

黃偉哲

107598019

2019/3/30

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

The Lab 2 requires to (1) select 15 methods from 6 classes of the SUT (GeoProject), (2) design Unit test cases by using **input space partitioning (ISP)** 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 ISP technique.

In particular, based on the statement coverage criterion, the **test requirements** for Lab 2 are to design test cases *with* ***ISP***for each selected method so that “*each statement of the method will be covered by at least one test case* and *the minimum statement coverage is 70% (greater than Lab 1)*”.

* 1. **Test Strategy**

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

1. select **those 10 methods that were chosen in Lab1** and **5 new methods** that are NOT selected previously. If possible, some of the methods do NOT have primitive types of input or output parameters (if possible).
2. set the objective of the minimum statement coverage to be greater than that of Lab 1 and adjust the test objective based on the time available (if necessary).
3. design the test cases for those selected methods by using the **input space partitioning (ISP)** 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 | 2019/3/26 |
| 2 | Learn **ISP** and JUnit | 1 | 2019/3/26 |
| 3 | Design test cases for the selected methods | 2 | 2019/3/27 |
| 4 | Implement test cases | 3 | 2019/3/28 |
| 5 | Perform tests | 1 | 2019/3/29 |
| 6 | Complete Lab2 report | 1 | 2019/3/30 |

* 1. **Design Approach**

The **ISP** technique will be used to design the test cases. Specifically, the possible partitions and boundary values of input parameters shall be identified first using the **Mine Map** and **domain knowledge** (if applicable). The possible **valid** combinations of the partitions (i.e., **all combination coverage**) as well as the boundary values shall be computed for the input parameters of each selected method. Each of the partition combination can be a possible test case. *Add more test cases by considering the possible values and boundary of the outputs for the methods or by using test experiences.*

* 1. **Success criteria**

All test cases designed for the selected methods must pass (or 90% of all test cases must pass) and *the statement coverage should have achieved at least 70%.*

1. **Test Design**

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

The details of the design are given below:

The Excel file of test cases…

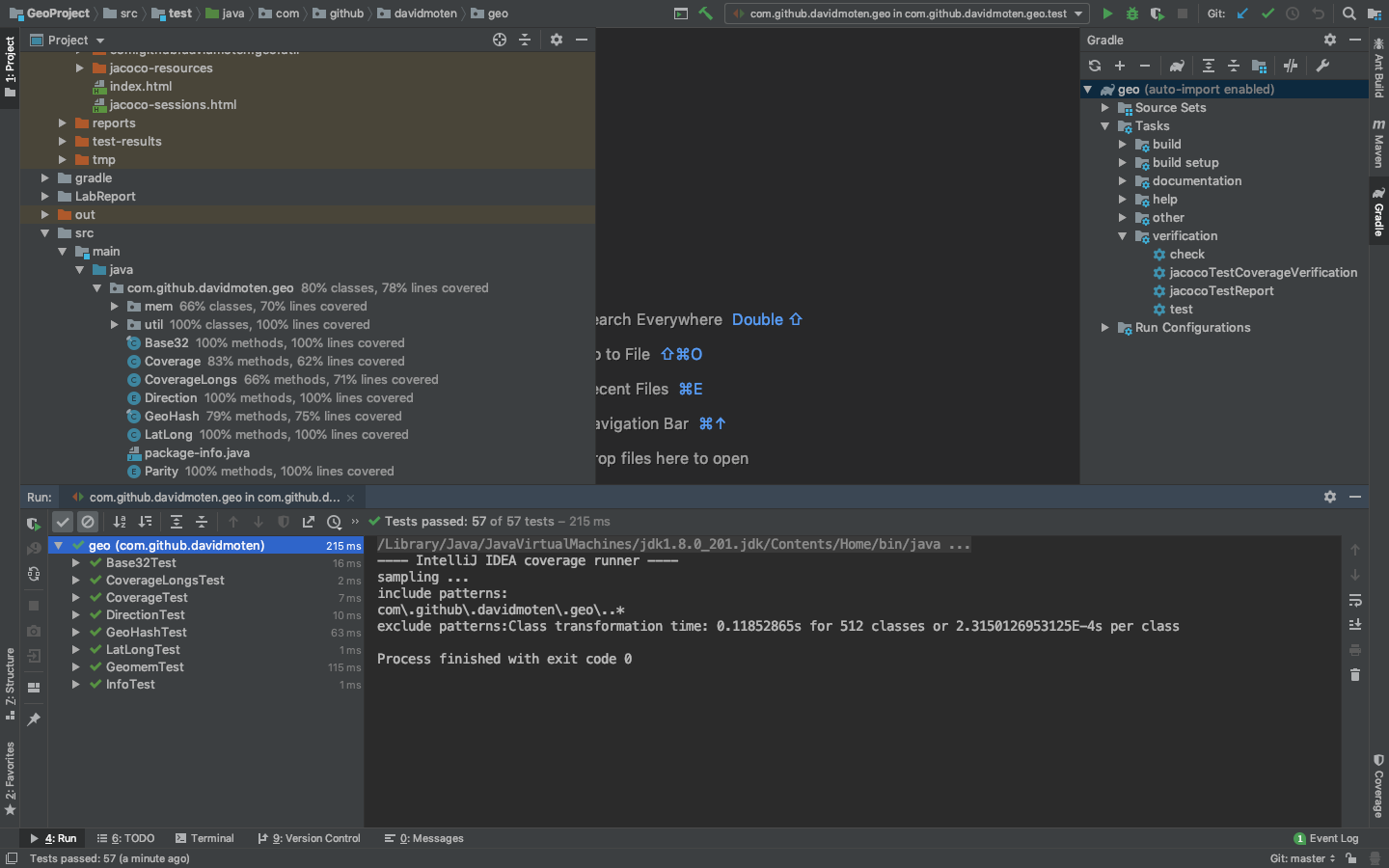
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Class** | **Method** | **Test Objective** | **Inputs** | **Expected Outputs** |
| 1 | Base32 | encodeBase32 | encodeBase32(long i, int length) | i = -75314, length = -6 | "-29jk" |
| i = -89563, length = 1 | "-2rfv" |
| i = 66666, length = -8 | "213b" |
| i = 88888, length = 7 | "0002qts" |
| 2 | Base32 | decodeBase32 | decodeBase32(String hash) | hash = 29jw | 75324 |
| hash = -29jk | -75314 |
| 3 | Base32 | getCharIndex | getCharIndex(char ch) | ch = 'b' | 10 |
| ch = 'a' | “not a base32 character: a” |
| 4 | Base32 | encodeBase32 | encodeBase32(long i) | i = -6666 | "-0000000006hb" |
| i = 75324 | "0000000029jw" |
| 5 | GeoHash | top | top(String hash) | hash = "-29xy" | "-2c8n" |
| hash = "29jw" | "29jx" |
| 6 | GeoHash | bottom | bottom (String hash) | hash = "-30xx" | "-30xr" |
| hash = "88eq" | "88em" |
| 7 | GeoHash | left | left (String hash) | hash = "-32ty" | "-32tv" |
| "66py" | "66pw" |
| 8 | GeoHash | right | right (String hash) | hash = "-00er" | "-00g2" |
| hash = "68jk" | "68js" |
| 9 | GeoHash | heightDegrees | heightDegrees(int n) | n = 2 | 5.625 |
| ch = 14 | 5.24E-09 |
| 10 | GeoHash | widthDegrees | widthDegrees(int n) | n = 13 | 5.4.190951585769653E-8 |
| n = 6 | 1.10E-02 |
| 11 | GeoHash | hashLengthToCoverBoundingBox | hashLengthToCoverBoundingBox(double topLeftLat, double topLeftLon, double bottomRightLat, double bottomRightLon) | topLeftLat = 25.0361156, topLeftLon = 121.4639264,  bottomRightLat = 25.0289061, bottomRightLon = 121.4889208 | 4 |
| topLeftLat = 25.0289061, topLeftLon = 121.4889208,  bottomRightLat = 25.0361156, bottomRightLon = 121.4639264 | 4 |
| opLeftLat = 0, topLeftLon = 0,  bottomRightLat = 0, bottomRightLon = 0 | 12 |
| 12 | GeoHash | adjacentHash | adjacentHash(String hash, Direction direction) | hash = null, directioh: Direction.LEFT | "hash must be non-null" |
| hash = "19jw", directioh: Direction.LEFT | "19jq" |
| 13 | GeoHash | decodeHash | decodeHash(String geohash) | geohash = null | "geohash cannot be null" |
| geohash = "29jw" | LatLong<LatLong [lat=-38.232421875, lon=-149.58984375]> |
| 14 | GeoHash | encodeHash | encodeHash(LatLong p, int length) | p = new LatLong(-38.232421875, -149.58984375), length = 6 | "29jws0" |
| p = new LatLong(-38.232421875, -149.58984375), length = 0 | "length must be greater than zero" |
| 15 | GeoHash | hashContains | hashContains(String hash, double lat, double lon) | hash = 29jw, lat = -38.232421875, lon = -149.58984375 | TRUE |
| hash = 29jw, lat = 0, lon = 0 | FALSE |

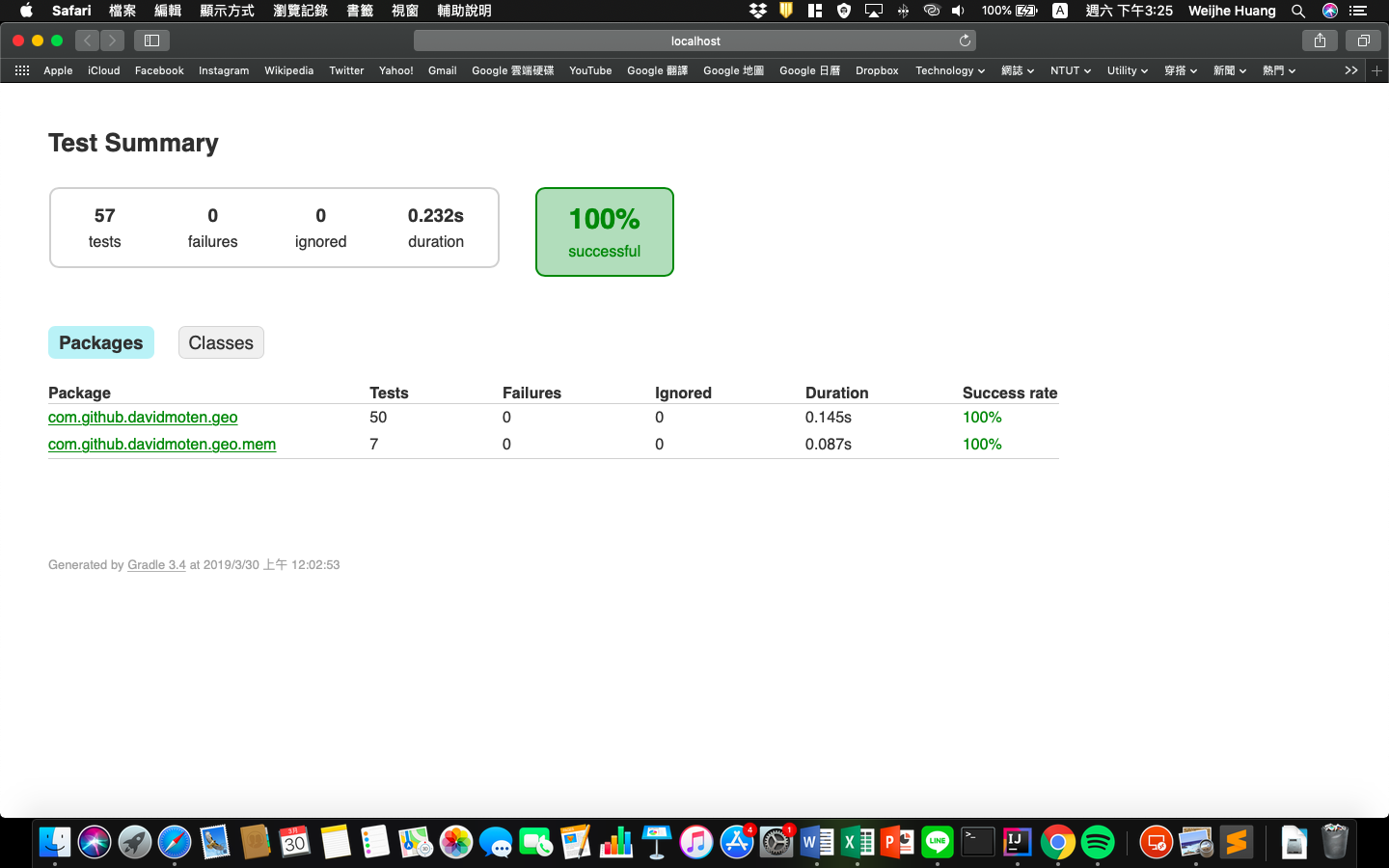
1. **Test Implementation**

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).

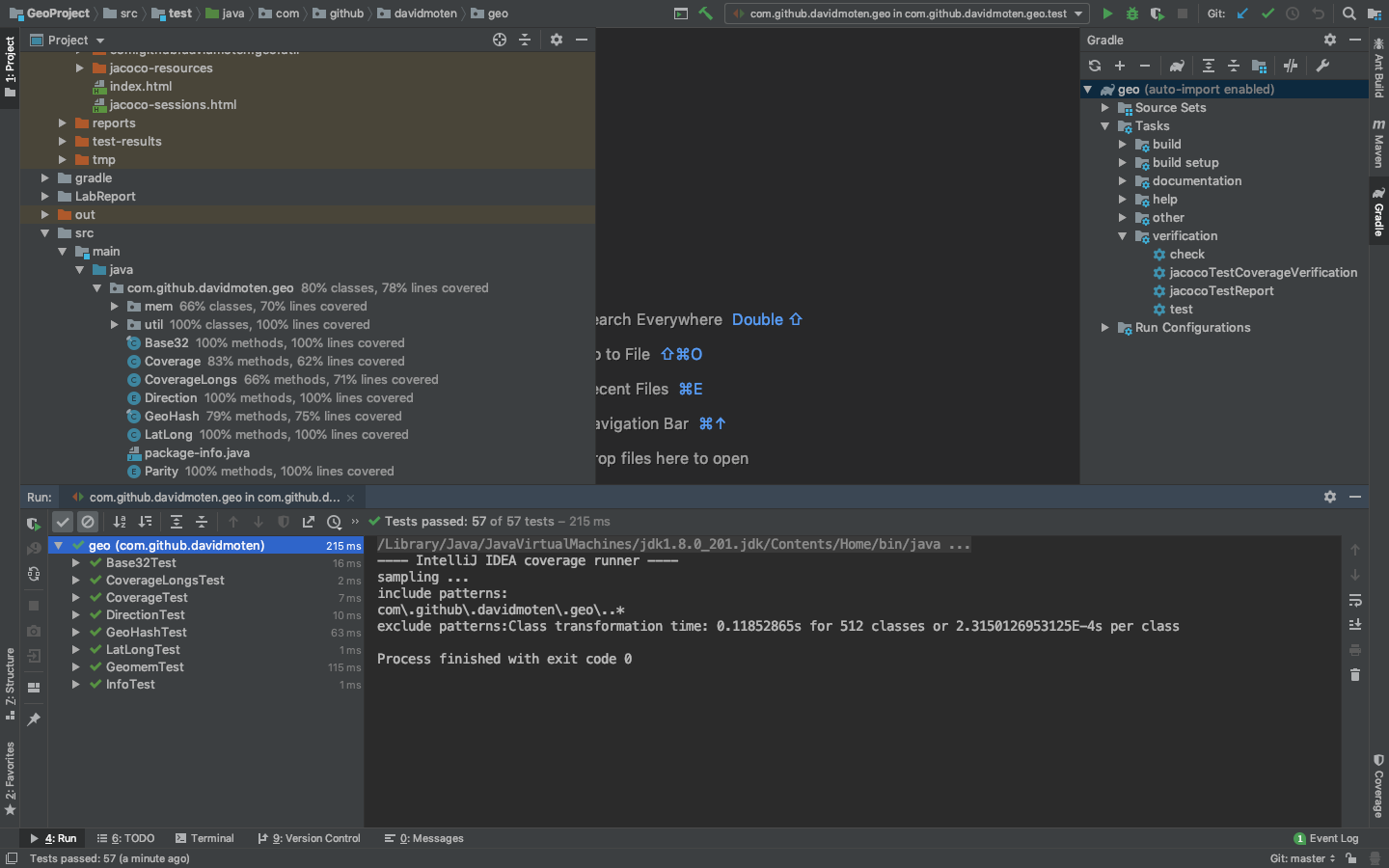
|  |  |  |
| --- | --- | --- |
| **No.** | **Test method** | **Source code** |
| 1 | GeoHash.decodeHash(String geohash) | @Test  public void decodeHash() throws Exception{  LatLong result = GeoHash.decodeHash("29jw");  LatLong ll = new LatLong(-38.232421875, -149.58984375);  assertEquals(ll.getLat(), result.getLat(),0.001);  assertEquals(ll.getLon(), result.getLon(),0.001);  } |
| 2 | GeoHash.hashContains(String hash, double lat, double lon) | @Test  public void coverBoundingBox() throws Exception{  Boolean result = GeoHash.hashContains("29jw",  -38.232421875, -149.58984375);  assertEquals(true, result);  } |
| 3 | GeoHash.widthDegrees(int n) | @Test  public void widthDegrees\_F() throws Exception{  double n = GeoHash.widthDegrees(13);  assertEquals(4.190951585769653E-8,n,0.001);  } |

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

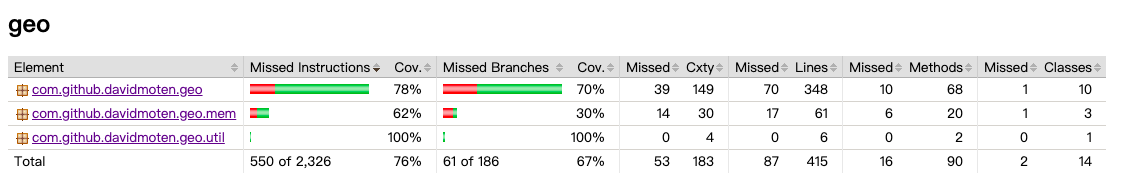
****

****

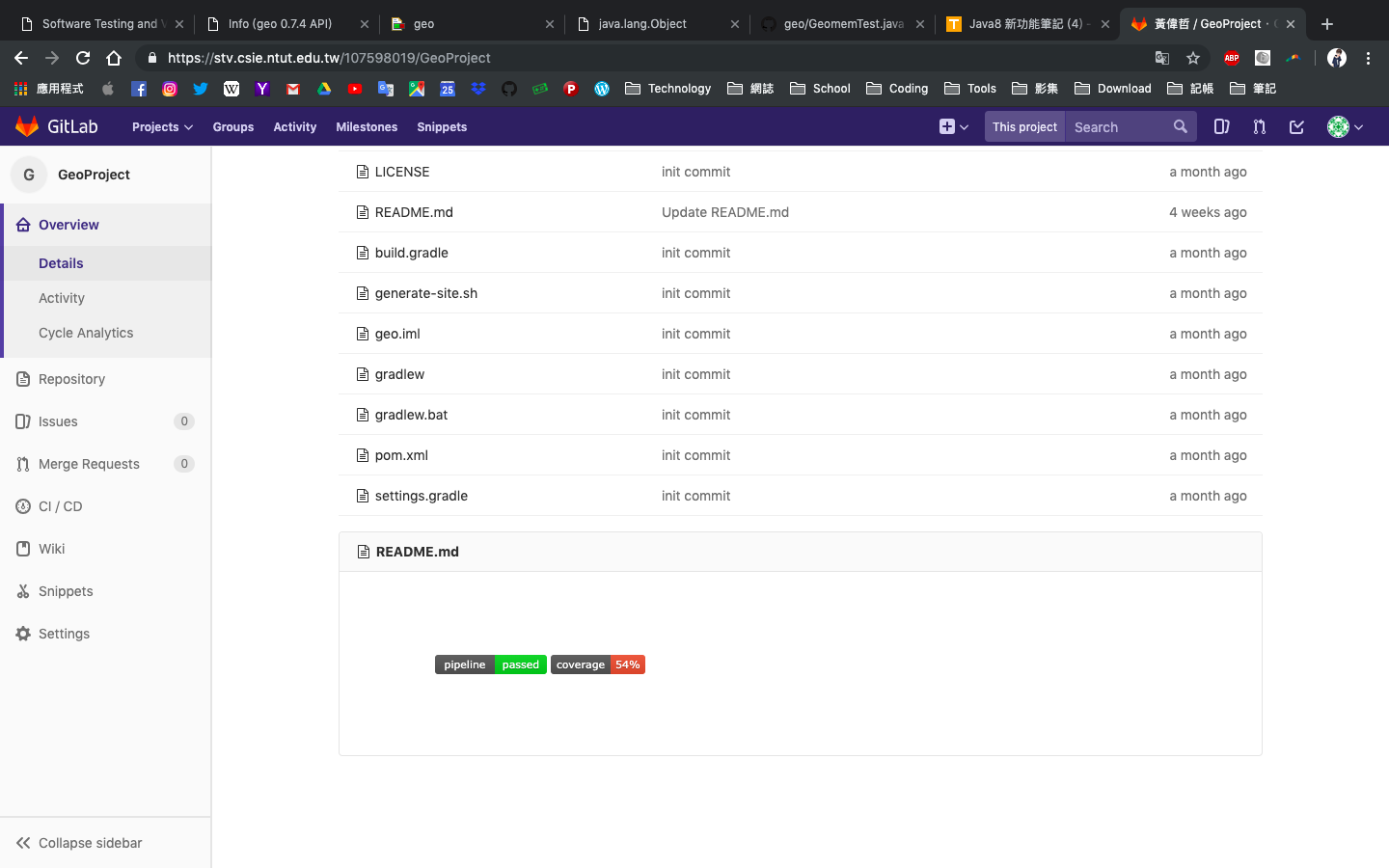
* 1. **Code coverage snapshot**
* Coverage of each selected method

****

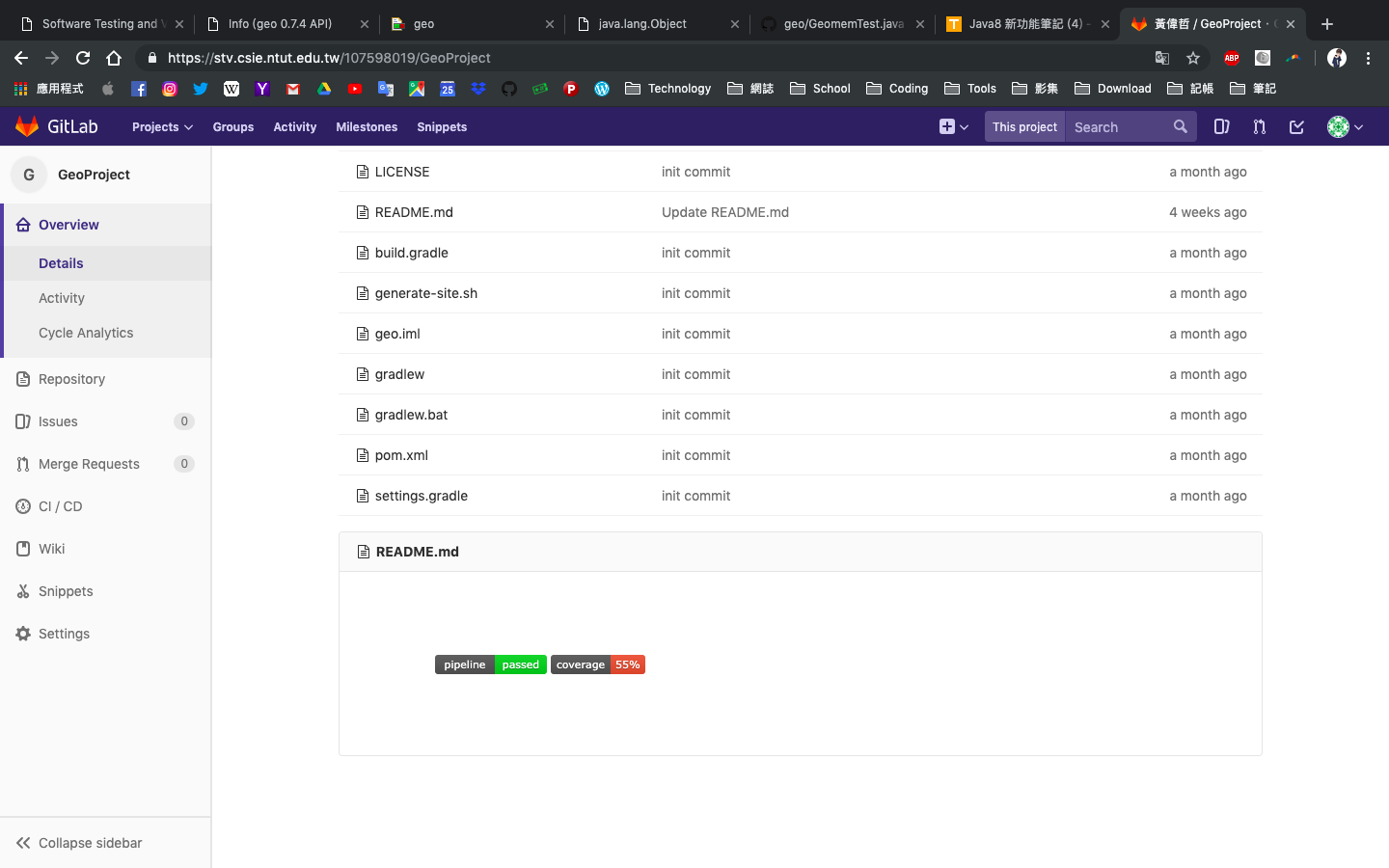
* Total coverage

****

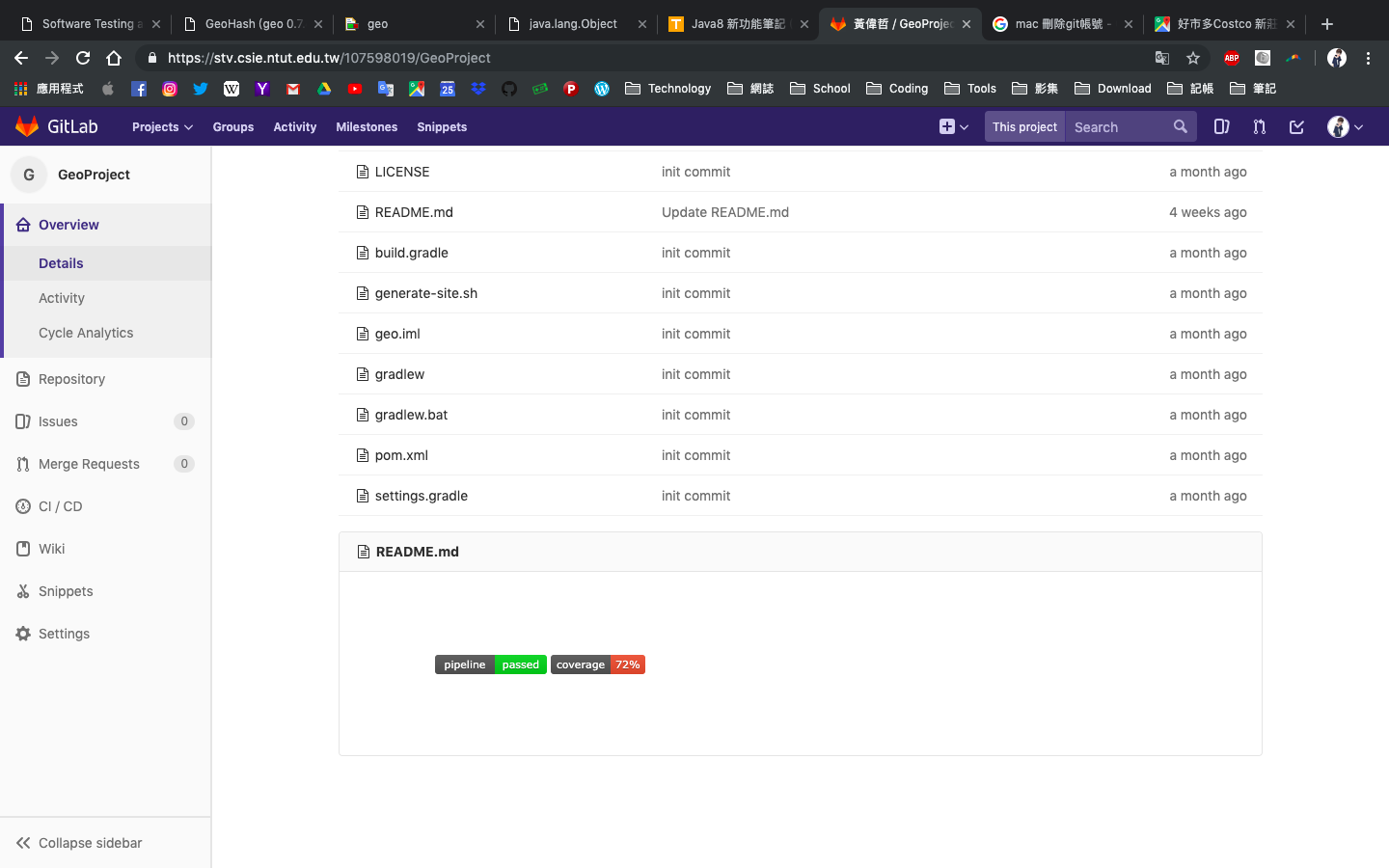
* 1. **CI result snapshot (3 iterations for CI)**
* CI#1

****

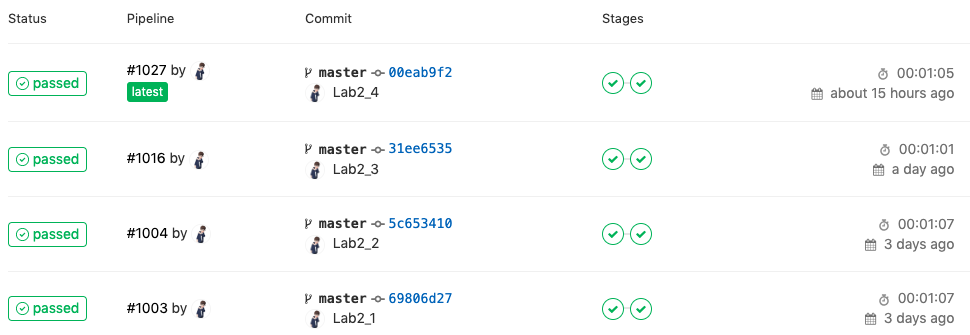
* CI#2



* CI#3

****

* CI Pipeline

****

1. **Summary**

相較Lab1，在設計test case時須考量到較多面向，不只是為了提高statement coverage而寫happy path，對同一個method會花更多間在設計，雖然不見得會提高覆蓋率，但在品質方面相對有保障。