**Lab 2 Report**

謝狄烽

110598087

2022/04/12

**Contents**

[**1** **Test Plan** 2](#_Toc101130240)

[**1.1** **Test requirements** 2](#_Toc101130241)

[**1.2** **Test Strategy** 2](#_Toc101130242)

[**1.3** **Test activities** 2](#_Toc101130243)

[**1.4** **Design Approach** 2](#_Toc101130244)

[**1.5** **Success criteria** 3](#_Toc101130245)

[**2** **Test Design** 3](#_Toc101130246)

[**3** **Test Implementation** 5](#_Toc101130247)

[**4** **Test Results** 6](#_Toc101130248)

[**4.1** **JUnit test result snapshot** 6](#_Toc101130249)

[**4.2** **Code coverage snapshot** 7](#_Toc101130250)

[**4.3** **CI result snapshot (3 iterations for CI)** 7](#_Toc101130251)

[**5** **Summary** 8](#_Toc101130252)

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 | 2022/04/09 |
| 2 | Learn **ISP** and JUnit | 3 | 2022/04/09 |
| 3 | Design test cases for the selected methods | 5 | 2022/04/10 |
| 4 | Implement test cases | 5 | 2022/04/11 |
| 5 | Perform tests | 3 | 2022/04/12 |
| 6 | Complete Lab2 report | 2 | 2022/04/12 |

* 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Class** | **Method** | **Test Objective** | **Inputs** | **Expected Outputs** |
| **1** | **Base32** | *encodeBase32 (long, int)* | Encode with fixed length | 1252921504606846975L, -2 | -12su5g1fsmzzz |
| **2** | *decodeBase32 (Str)* | If I understand how it works | “” | 0 |
| **3** | *getCharIndex (char)* | Throwing exception correctly | ‘o’ | Throw IllegalArgument Exception |
| **4** | **CoverageLongs** | *CoverageLongs (long[], int, double)* | Throwing exception correctly | {30893887395, 30893887396L},3, 100 | ArrayIndexOutOf BoundsException |
| **5** | **Coverage** | *Coverage (Set<Str>, double)* | Construct by geohashes | {wsqqmx4, wsqqmx3}, 0.5 | “Coverage [hashes=[ wsqqmx4, wsqqmx3], ratio=0.5]” |
| **6** | **GeoHash** | *adjacentHash (Str, direction, int)* | Move position with step | "wsqqmx4", Direction.TOP, 2 | “wsqqmxd” |
| **7** | *neighbours (Str)* | Get position nearby | "wsqqmx4" | {wsqqmx1, wsqqmx5, wsqqmx6, wsqqmwf, wsqqmx3, wsqqmwc, wsqqmx7, wsqqmwg} |
| **8** | *encodeHash (double, double)* | Generate correct geohash | 25.04386042, 121.53397182 | “wsqqmx474ccw” |
| **9** | *encodeHash (LatLong)* | Handel object correctly | LatLong latlon | wsqqqqqqqebj |
| **10** | *fromLongToString (long)* | Throwing exception correctly | 13089388739600L | Throw IllegalArgument Exception |
| **11** | *hashContains (Str, double, double)* | Check geshash area includes giving lat lon | “wsqqmx4”, 25.04386042, 121.53397182 | *True* |
| **12** | *gridAsString (Str, int, Set<String>)* | Grid nearby geohash correctly | “wsqq”, 1, {wsqq} | wsqp wsqr wsqx  wsqn WSQQ wsqw  wsqj wsqm wsqt |
| **13** | **Geomem** | *add (Info)* | Adding Info object correctly | Info info<String, String> | [Info [lat=25.04, lon=121.53, time=1647265758, value=SID, id=Optional.of(8087)]] |
| **14** | *find(double,double,double,double,long,long)* | Find objects correctly | 26,121,25,122, 1647265750, 1647265760 | Info info<String, String> |
| **15** | **Info** | *Info (double, double, long, T, Optional<R>)* | Construct correctly | 25.0438604, 21.5339717, 1646110492, "value", Optional.of ("1105980") | "Info [lat=25.0438604, lon=21.5339717, time=1646110492, value=value, id=Optional.of (1105980)]" |

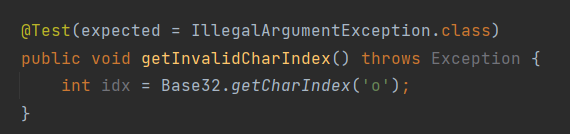
The details of the design are given below:

LAB2 ISP test case design.xlsx

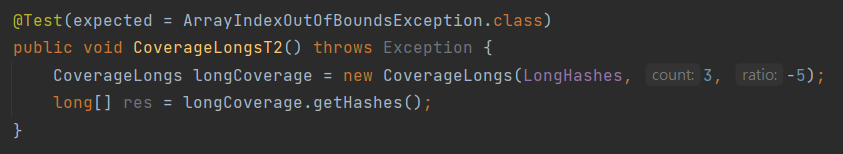
1. **Test Implementation**

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 test script implementations can be found in JUnit files.

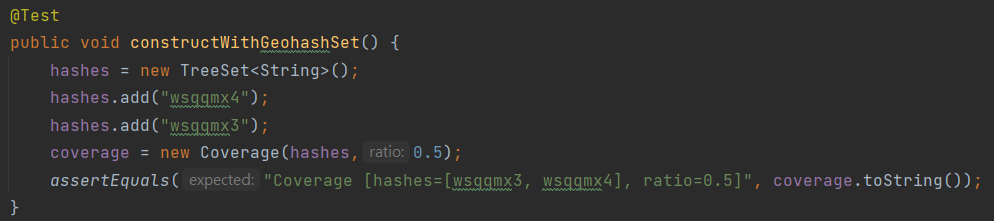
**Test method**: *Base32.getCharIndex ()*



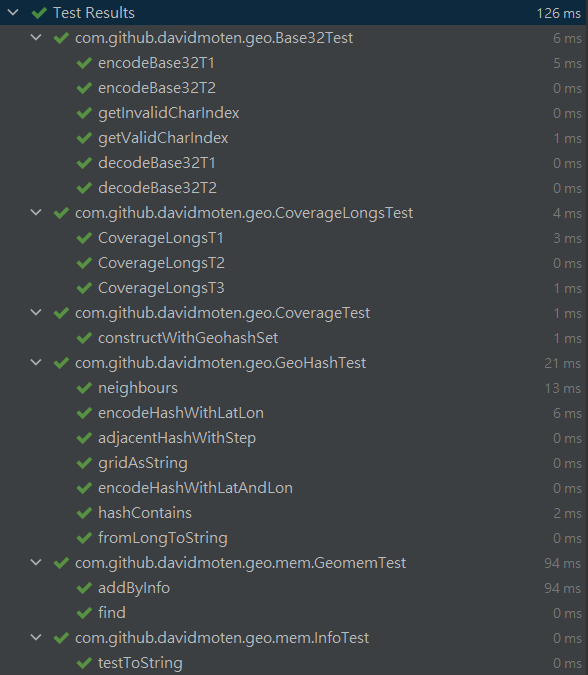
**Test method**: *CoverageLongs.CoverageLongs()*

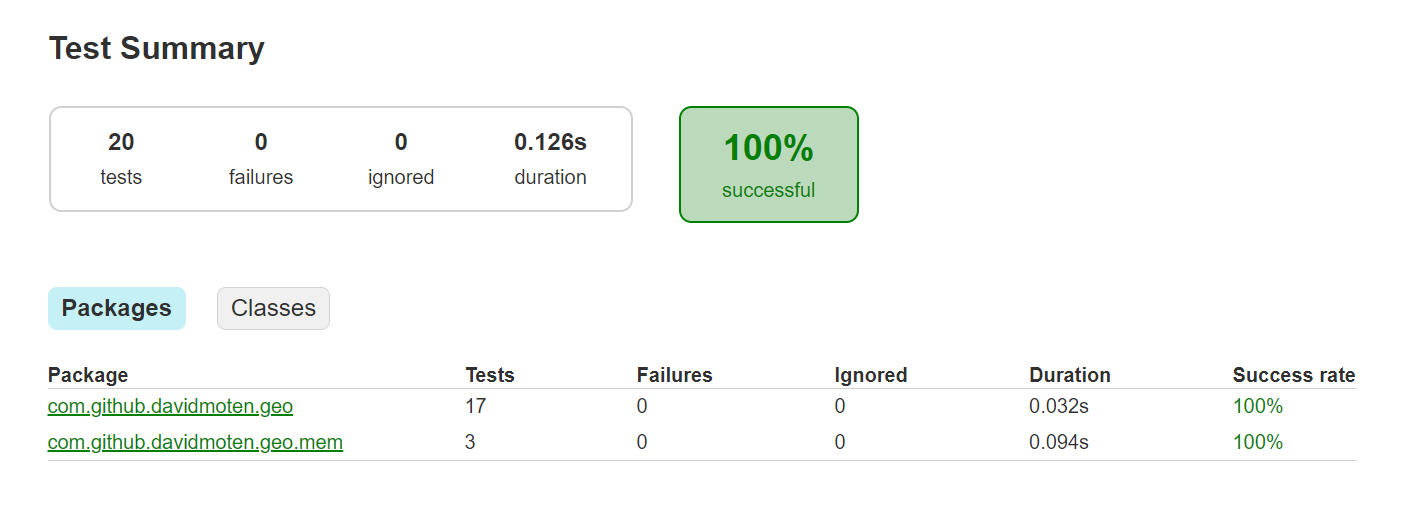


**Test method**: *Coverage.Coverage ()*

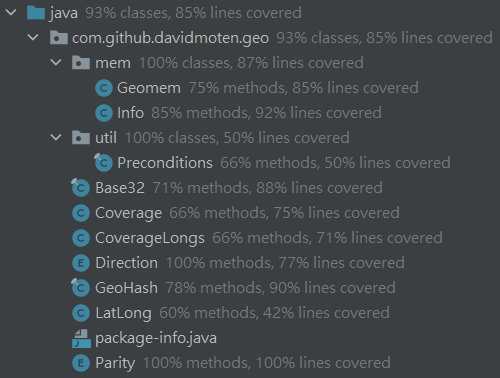


1. **Test Results**
   1. **JUnit test result snapshot**

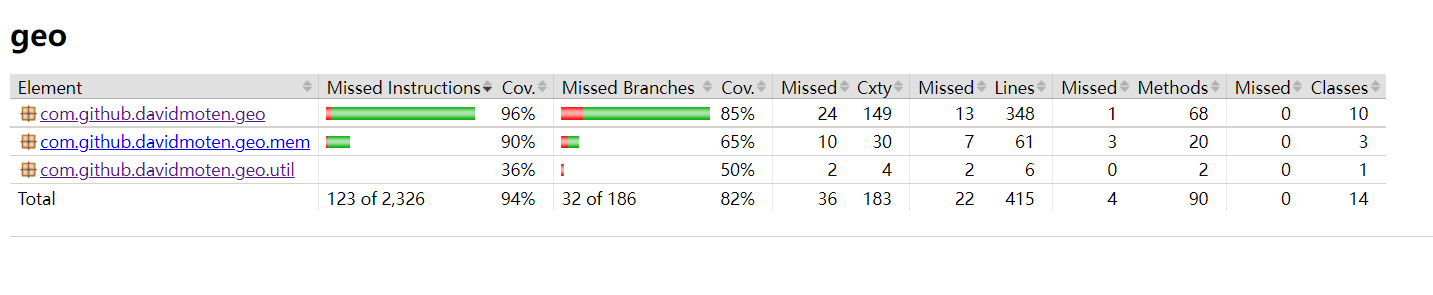




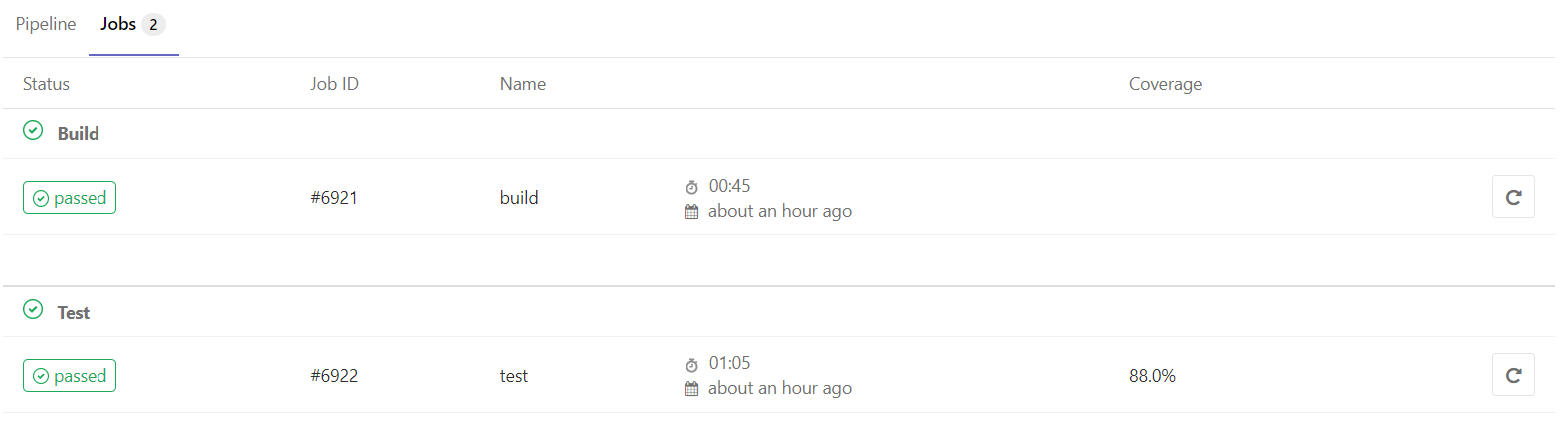
* 1. **Code coverage snapshot**
* Coverage of each selected method



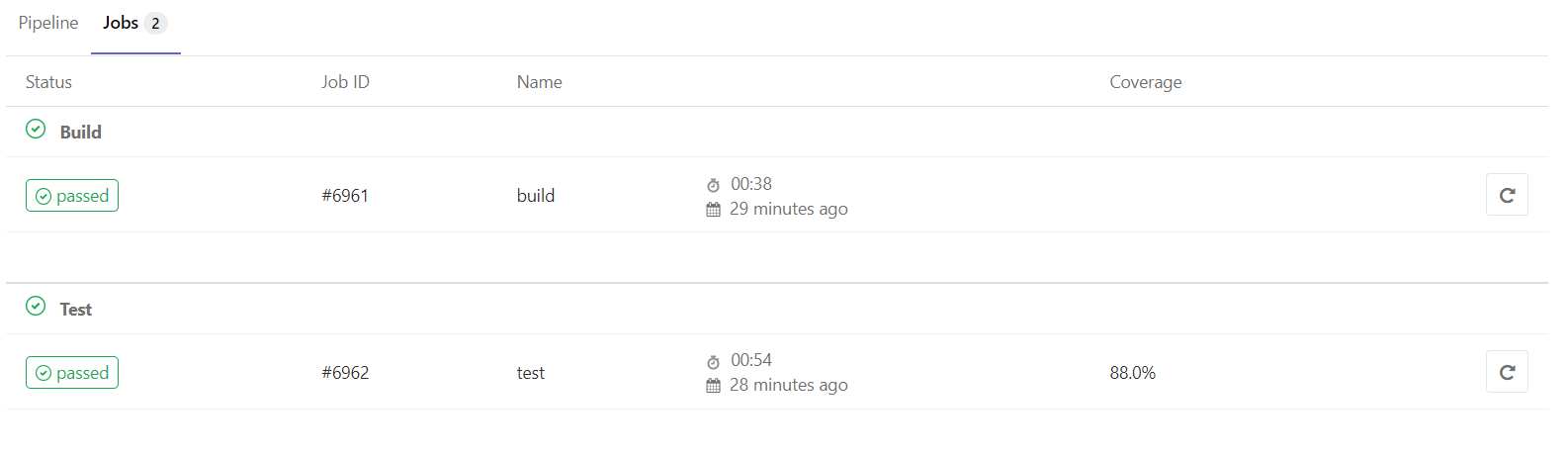
* Total coverage



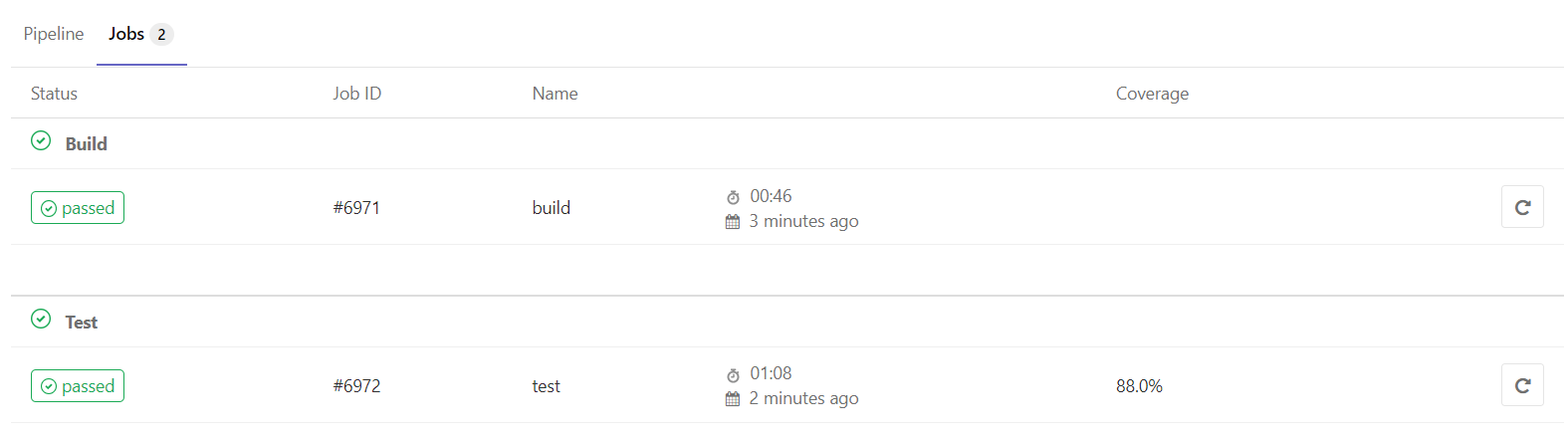
* 1. **CI result snapshot (3 iterations for CI)**
* CI#1



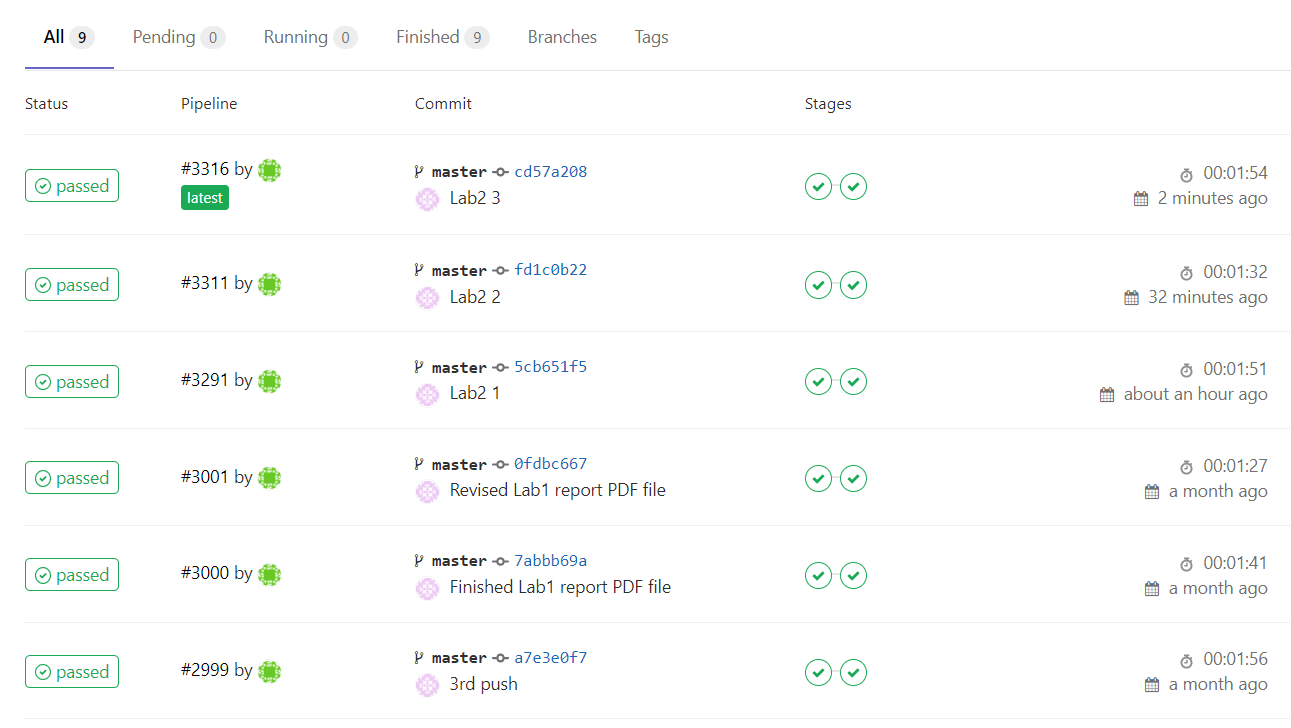
* CI#2



* CI#3



* CI Pipeline



1. **Summary**

In Lab 2, **15 test cases have been designed and implemented using JUnit and the ISP technique**. The test is conducted in 3 CI and **the execution results of the 15 test methods are all passed**. **The total statement coverage of the test is 70%.** Thus, the test requirements described in Section 1 are satisfied. Some lessons learned in this Lab are the technique of using Input Space Partition, it is harder than I thought, it needs to consider all of the condition that may happens on inputs, and then we need to exclude all the input combination that are infeasible, so we can get the feasible input for the test.