Lab 3 Report

張嗣岱
107598069
4/24

1 Test Plan

1.1 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 and 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

(6) specify your experiences of designing test cases systematically using the ISP technique.

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

1.2 Test Strategy

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

(1) select **those 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 **loop** structures (as many as possible).

(2) set the objective of the minimum statement coverage to be greater than that of Lab 2 and adjust the test objective based on the time available (if necessary).

(3) design the test cases for those selected methods by using the **basis path and graph coverage** testing technique.

1.3 Test activities

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

No.	Activity Name	Plan hours	Schedule Date
1	Study GeoProject	2	4/19
2	Learn **basis path and graph coverage**	5	4/20
3	Design test cases for the selected methods	5	4/21
4	Implement test cases	3	4/22
5	Perform tests	3	4/22
6	Complete Lab3 report	2	4/24

1.4 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) and branch (edge) coverage**) shall be derived from the CFG. The possible **inputs** and **expected outputs** for the derived test paths shall be computed for each selected method. *Add more test cases by considering to satisfy other coverage criteria, such as edge-pair, all-use, or prime-path coverage criteria.*

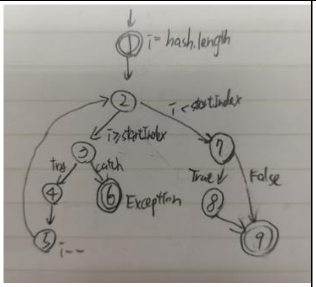
1.5 Success criteria

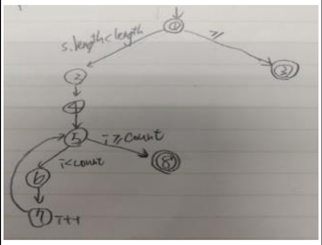
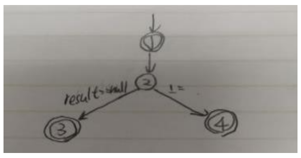
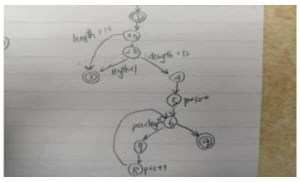
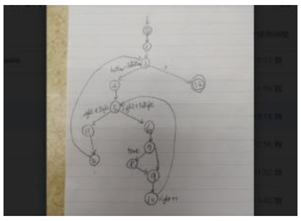
All test cases designed for the selected methods must pass (or 90% of all test cases must pass) and *both statement and branch coverage should have achieved at least 90%, respectively*.

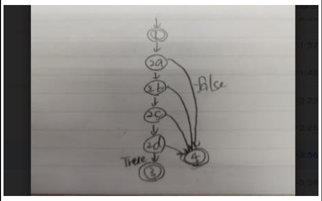
2 Test Design

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	decodeBase32ength		P1:{1,2,3,4,5,2,7,9}
				P2:{1, 2, 7, 8, 9}
				P3: {1, 2, 3, 6}
					"1b"	42

2	Base32	padLeftWithZeroToLength		P1: {1, 2, 4, 5, 6, 7, 5, 8}, P2: {1, 3}	"abc",4	0abc
3	Base32	getCharIndex		P1: {1, 2, 3}, P2: {1, 2, 4}	'1'	1
4	GeoHash	fromLongToString		P1: {1, 2a, 2b, 4, 5, 6, 7, 8, 9}, P2: {1, 2a, 3}, P3: {1, 2b, 3}	1	“0”
5	GeoHash	gridAsString		P1: {1, 2, 3, 4, 5, 6, 7, 9, 10, 5, 6, 7, 8, 9, 10, 5, 11, 12, 3, 13}	"d",-1,-1,1,1,[f]	"c F g \n" + "9 d e \n"+ "3 6 7 \n"

6	GeoMem	createRegionFilter		P1: {1, 2a, 4}, P2: {1, 2a, 2b, 4}, P3: {1, 2a, 2b, 2c, 4}, P4: {1, 2a, 2b, 2c, 2d, 4}, P5: {1, 2a, 2b, 2c, 2d, 3}	0,0,-100,100	true

The details of the design are given below:
The Excel file of test cases…

3 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 (or JUnit files).

No.	Test method	Source test code

1	Base32.decodeBase32ength(hash)	`@Test`
		`public void decodeBase32() {`
		`    assertEquals(42,Base32.decodeBase32("1b"));`
		`    assertEquals(-42,Base32.decodeBase32("-1b"));`
		`    try{`
		`        assertEquals(-1,Base32.decodeBase32("a"));`
		`    }catch (IllegalArgumentException e){`
		`        e.printStackTrace();`
		`    }`
		`}`

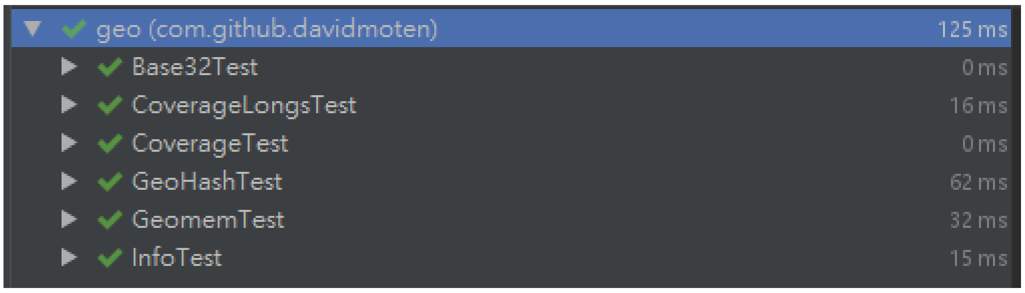
2	Base32. padLeftWithZserosToLength (s,length)	`@Test`
		`public void padLeftWithZerosToLength() {`
		` assertEquals("0abc",Base32.padLeftWithZerosToLength("abc",4));`
		` assertEquals("abc",Base32.padLeftWithZerosToLength("abc",-4));`
		`}`

3	GeoHash.fromLongToString(hash)	`@Test`
		`public void fromLongToString() {`
		`    //e//`

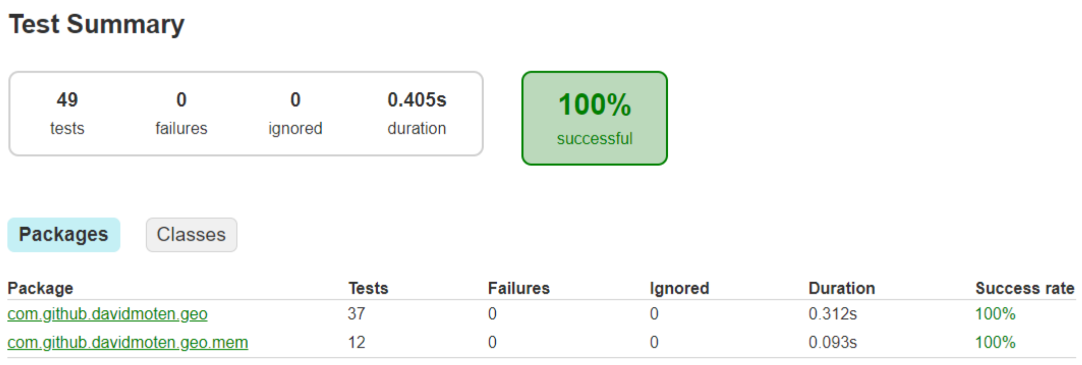
```
    long hash =1;
    assertEquals("0",GeoHash.fromLongToString(hash));
    try {
        hash = 13;
        assertEquals("0",GeoHash.fromLongToString(hash));
    }catch (IllegalArgumentException e){
        System.out.println(e);
    }
    try {
        hash = 0;
        assertEquals("0",GeoHash.fromLongToString(hash));
    }catch (IllegalArgumentException e){
        System.out.println(e);
    }
}
```

4 Test Results

4.1 JUnit test result snapshot



```
▼ ✓ geo (com.github.davidmoten)      125 ms
    ▶ ✓ Base32Test                     0 ms
    ▶ ✓ CoverageLongsTest             16 ms
    ▶ ✓ CoverageTest                   0 ms
    ▶ ✓ GeoHashTest                   62 ms
    ▶ ✓ GeomemTest                    32 ms
    ▶ ✓ InfoTest                      15 ms
```



Test Summary

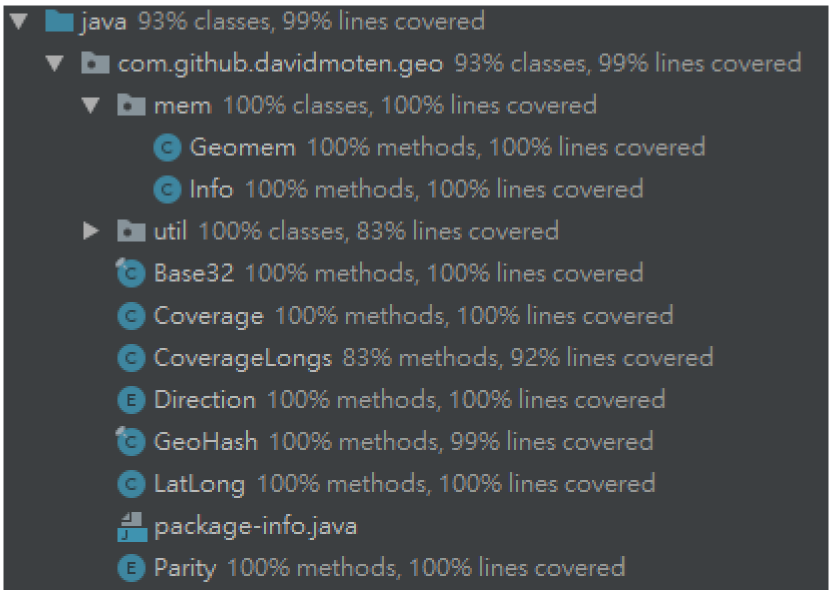
49	0	0	0.405s	100%
tests	failures	ignored	duration	successful

Packages Classes

Package	Tests	Failures	Ignored	Duration	Success rate
com.github.davidmoten.geo	37	0	0	0.312s	100%
com.github.davidmoten.geo.mem	12	0	0	0.093s	100%

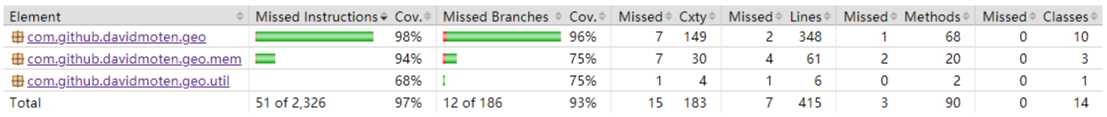
4.2 Code coverage snapshot

● Coverage of each selected method



```
▼ java 93% classes, 99% lines covered
  ▼ com.github.davidmoten.geo 93% classes, 99% lines covered
    ▼ mem 100% classes, 100% lines covered
        Geomem 100% methods, 100% lines covered
        Info 100% methods, 100% lines covered
    ▶ util 100% classes, 83% lines covered
      Base32 100% methods, 100% lines covered
      Coverage 100% methods, 100% lines covered
      CoverageLongs 83% methods, 92% lines covered
      Direction 100% methods, 100% lines covered
      GeoHash 100% methods, 99% lines covered
      LatLong 100% methods, 100% lines covered
      package-info.java
      Parity 100% methods, 100% lines covered
```

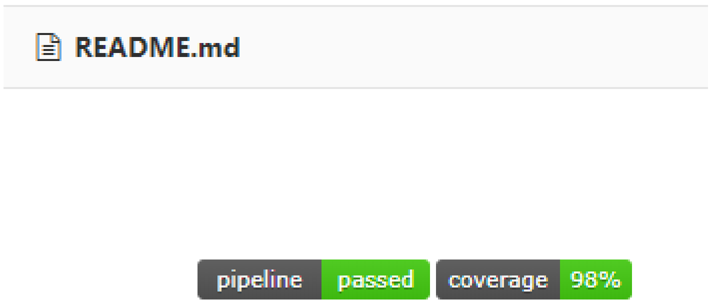
- Total coverage



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.github.davidmoten.geo		98%		96%	7	149	2	348	1	68	0	10
com.github.davidmoten.geo.mem		94%		75%	7	30	4	61	2	20	0	3
com.github.davidmoten.geo.util		68%		75%	1	4	1	6	0	2	0	1
Total	51 of 2,326	97%	12 of 186	93%	15	183	7	415	3	90	0	14

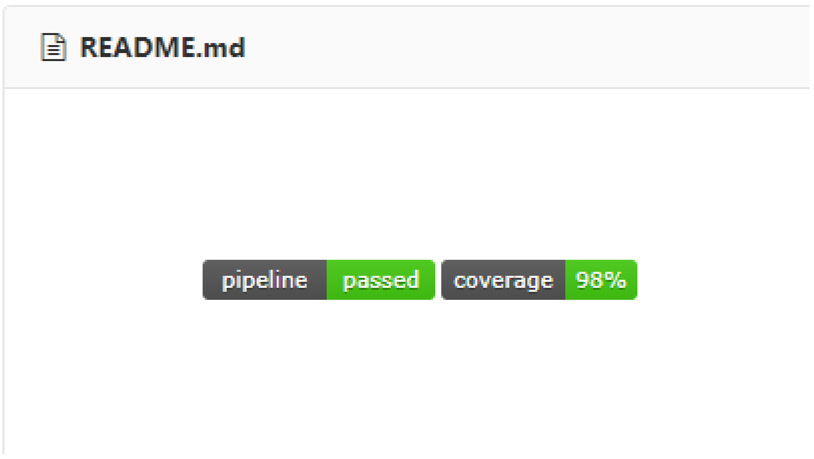
4.3 CI result snapshot (3 iterations for CI)

- CI#1

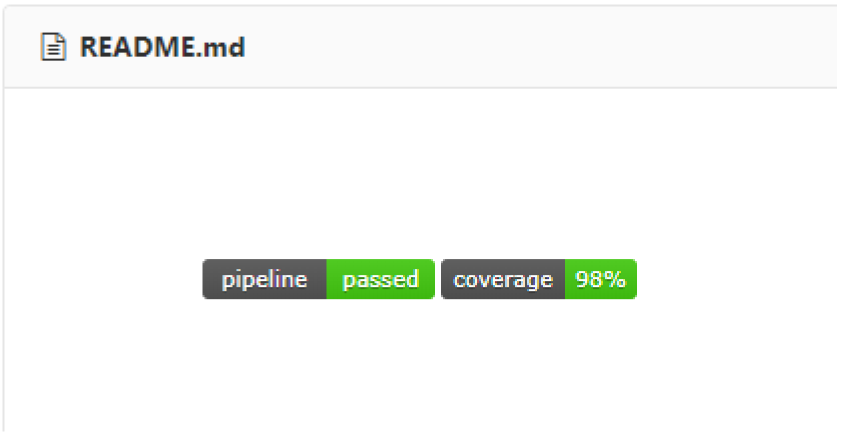


README.md

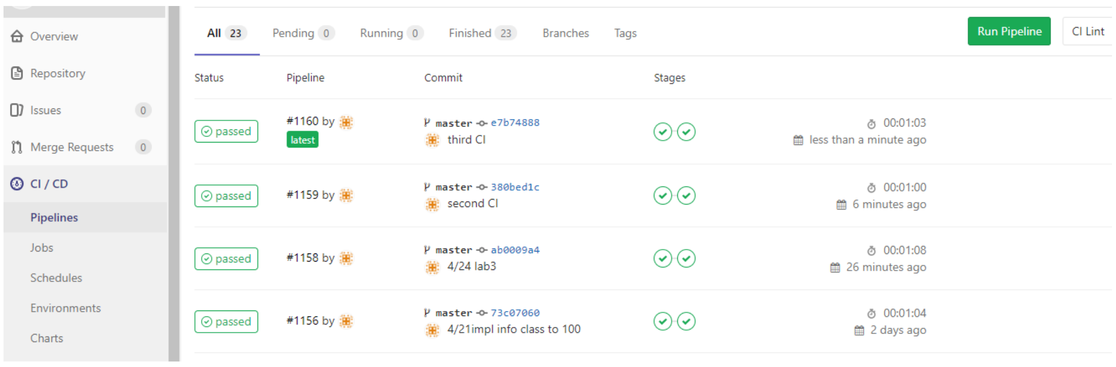
- CI#2



- CI#3



- CI Pipeline



Status	Pipeline	Commit	Stages	
passed	#1160 by latest	master e7b74888 third CI	✓ ✓	00:01:03 less than a minute ago
passed	#1159 by	master 380bed1c second CI	✓ ✓	00:01:00 6 minutes ago
passed	#1158 by	master ab0009a4 4/24 lab3	✓ ✓	00:01:08 26 minutes ago
passed	#1156 by	master 73c07060 4/21impl info class to 100	✓ ✓	00:01:04 2 days ago

5 Summary

In Lab 3,6 **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 5 test methods are all passed**. **The total statement and branch coverage of the test are 97% and 93%, respectively.** Thus, the test requirements described in Section 1 are satisfied. I think that design by basis path is much fun than ISP cause I can focus on the source code which I didn't reach.