Lab 1 Report

Name 林柔璟 Student ID 108598003 Date

1 Test Plan

1.1 Test requirements

The Lab 1 requires to (1) select **15 methods** from **6 classes** of the SUT (GeoProject), (2) design Unit test cases based on the experience or intuition for the selected methods, (3) develop test scripts to implement the test cases, (4) execute the test script on the selected methods, and (5) report the test results.

In particular, based on the statement coverage criterion, the **test requirements** for Lab 1 are to design test cases for each selected method so that "each statement of the method will be covered by <u>at least one test case</u> and the <u>minimum</u> statement coverage is 40%".

1.2 Strategy

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

- (1) select those <u>public</u> methods that are easy to understand and have <u>primitive</u> <u>types</u> of input and output parameters (if possible).
- (2) set the objective of the minimum statement coverage to be 50% initially and (if necessary) adjust the objective based on the time available.
- (3) learn the necessary skills and tools as soon as possible.
- (4) design the test cases for those selected methods by considering
 - i. the possible valid values and combinations of the input parameters.
 - ii. the **boundary values** of the <u>input parameters</u>.

1.3 Test activities

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

No.	Activity Name	Plan hours	Schedule Date
1	Environment Setting	1	17 th , March
2	Study GeoProject	3	17 th , March
3	Learn JUnit	1	17 th , March
4	Design test cases for the selected methods	2	18 th , March
5	Implement <u>Base32</u> test cases	1	18 th , March
6	Perform Base32 test	1	18 th , March

7	Implement <u>Coverage</u> test cases	1	19 th , March
8	Perform Coverage test	1	19 th , March
9	Implement CoverageLong test cases	1	21 th , March
10	Perform CoverageLong test	1	21 th , March
11	Implement <u>Direction</u> test cases	0.5	22 th , March
12	Perform <u>Direction</u> test	0.5	22 th , March
13	Implement GeoHash test cases	1	23 th , March
14	Perform GeoHash test	1	23 th , March
15	Implement <u>Info</u> test cases		24 th , March
16	Perform <u>Info</u> test	1	24 th , March
17	Complete Lab1 report	5	24 th -25 th , March

1.4 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 50%*.

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	Test Objective	Inputs	Expected Outputs
		A. encodeBase32(long i, int length) B. encodeBase32(long i)	Returns the base 32 encoding of the given length from a geohash	A. Base32.encodeBase32(75324, 4) //positive Base32.encodeBase32(-122,4) //negative B. Base32.encodeBase32((long) 32.0);	A. 29jw -003u B. 000000000 010
1	Base32	C. decodeBase32(String hash)	Returns the conversion of a base32 geohash to a long	<pre>C. Base32.decodeBase32("w") //positive Base32.decodeBase32("-j") //negative</pre>	C. 28 -17
		<pre>D. getCharIndex(char ch)</pre>	Throws an IllegalArgument Exception if the character is not found in the array.	<pre>D. Base32.getCharIndex('-')</pre>	D. Throw message: "not a base32 character : -"

2	Coverage	<pre>A. Coverage(Set< String> hashes, double ratio)</pre>	How well the coverage is covered by the hashes. Will be >=1. Closer to 1,	<pre>getHashSets.add("22"); getHashSets.add("33"); getHashSets.add("44"); A. Coverage(getHashSets</pre>	A. Hashes = getHashSe ts Ratio = 2.5
		B. Coverage(Cove rageLongs coverage)	the close the coverage is to the region in question.	<pre>coverageLongs = new CoverageLongs(hashes ,3,2.5); Coverage coverage_long = new Coverage(coverageLon gs); //CoverageLongs Test</pre>	B. getHashLe ngth = 1 Ratio = 2.5
		C. getHashes()	Returns the hashes which are expected to be all of the same length.	<pre>getHashSets.add("aaa") getHashSets.add("bbb") Coverage getHashCoverage = new Coverage(getHashSets,5.5) C. getHashCoverage.getH ashes()</pre>	C. getHashSe ts
		D. getRatio()	Returns the measure of how well the hashes cover a region.	Coverage(getHashSets,0.005) D. getRatioCoverage.get Ratio()	D. Ratio = 0.005
		E. getHashLength ()	Returns the length in characters of the first hash returned by an iterator on the hash set.	Coverage getHashLengthCoverage = new Coverage(getHashSets,2.4) E. getHashLengthCoverag e.getHashLength() //hashes.size() == 0 getHashSets.add("something") getHashLengthCoverag e.getHashLength() //hashes.size() != 0	E. HashLengt h(): > When hash size = 0: 0 > When hash size ≠ 0: 9
		F. toString()	Show Coverage [hashes, ratio]	Coverage testToStringCoverage = new Coverage(getHashSets,6.66)	F. "Coverage [hashes=[

		A. getHashes()	Returns the hashes which are expected to	<pre>F. testToStringCoverage .toString() hash = new long[]{33,22}; CoverageLongs coverageLongsGetHashes = new</pre>	<pre>], ratio=6.6 6]" A. hash = long[]{33</pre>
		- The get matrices ()	be all of the same length	CoverageLongs(hash, 2, 4.5); A. coverageLongsGetHash es.getHashes()	,22}
3		B. getRatio()	Returns the measure of how well the hashes cover a region	<pre>B. coverageLongsGetRati o.getRatio()</pre>	B. 3.0
	CoverageLongs C. getHashLeng () D. getCount()	C. getHashLength ()	Returns the length in characters of the first hash returned by an iterator on the hash set //compare[0] with 0000 1111(0x0f)	<pre>long[] hashZero = new long[]{}; CoverageLongs coverageLongsGetZeroLength = new CoverageLongs(hashZero,0,3.3); C. coverageLongsGetZero Length.getHashLength () //length = 0</pre>	<pre>C. Hash Length: > When length = 0: 0 > When length ≠ 0: 1</pre>
		D. getCount()	Count hash element	<pre>hash = new long[]{33,22}; CoverageLongs coverageLongsGetCount = new CoverageLongs(hash, count,3.00); D. coverageLongsGetCoun t.getCount()</pre>	D. 2

A GeoHash B. adjacentHash(4 GeoHash	direction, int steps) Method: adjacentHash() B. adjacentHash, String hash, Direction direction) Method: testAdjacentH	adjacent hash N steps in the given Direction. A negative N will use the opposite Direction Returns the adjacent hash in given	> adjacentHash = geoHash.adjacentHa sh("zzz", Direction.TOP); //out of border(top) > adjacentHash = geoHash.adjacentHa sh("145",Direction .BOTTOM); //out of border(bottom) > adjacentHash = geoHash.adjacentHa sh("000",Direction .LEFT); //out of border(left) > adjacentHash = geoHash.adjacentHa sh("ppp",Direction .RIGHT); //out of	"14y" B. normal: "11y" out of border:
-----------------------------	-----------	--	---	---	---------------------------------------

		D. left (String hash)	Returns the adjacent hash to the left (west)	<pre>hash = "11w"; D. geoHash.left(hash)</pre>	D. "11t"
		E. top (String hash)	Returns the adjacent hash to the top (north)	<pre>hash = "11w"; E. geoHash.top(hash))</pre>	E. "11y"
	F. bottom (String hash) Returns the adjacent hash to the bottom (south)		<pre>hash = "11w"; F. geoHash.bottom(hash)</pre>	F. "11q"	
		<pre>G. neighbours() Method: neighbours(St ring hash)</pre>	Returns a list of the 8 surrounding hashes for a given hash in order: left,right,top, bottom,left- top,left- bottom,right- top,right- bottom.	List <string> neighbours = new ArrayList<string>(); neighbours = geoHash.neighbours(hash); List<string> compare = new ArrayList<string>(); compare.add("11t"); //left compare.add("11y"); //right compare.add("11y"); //top compare.add("11v"); //left-top compare.add("11v"); //left-bottom compare.add("11z"); //right-top compare.add("11r"); //right-bottom G. neighbours</string></string></string></string>	G. compare
5	Direction	A. opposite()	Returns the opposite direction	<pre>Direction bottom = Direction.BOTTOM; Direction top = Direction.TOP; Direction left = Direction.LEFT; Direction right = Direction.RIGHT; A. bottom.opposite() top.opposite() left.opposite() right.opposite()</pre>	A. Opposite of >BOTTOM: TOP >TOP: BOTTOM >LEFT: RIGHT >RIGHT: LEFT
6	Info	A. id()	id of Info	<pre>String a = "a"; Optional<string> id = Optional.of(a); Info info = new Info(25.5, 30.0, 10000, 555, id); A. info.id()</string></pre>	A. Optional. of("a")

		B. lat()	Latitude of Info	<pre>String a = "a"; Optional<string> id = Optional.of(a); Info info = new Info(25.5, 30.0, 10000, 555, id); B. info.lat()</string></pre>	B. 25.5
		C. lon()	Longitude of Info	<pre>String a = "a"; Optional<string> id = Optional.of(a); Info info = new Info(25.5, 30.0, 10000, 555, id); C. info.lon()</string></pre>	C. 30.0
	D. time()	Time of Info	<pre>String a = "a"; Optional<string> id = Optional.of(a); Info info = new Info(25.5, 30.0, 10000, 555, id); D. info.time()</string></pre>	D. 10000	
	E. value()	Value of Info	<pre>String a = "a"; Optional<string> id = Optional.of(a); Info info = new Info(25.5, 30.0, 10000, 555, id); E. info.value()</string></pre>	E. 555	
	F. toString()	Show all information of Info	<pre>String a = "a"; Optional<string> id = Optional.of(a); Info info = new Info(25.5, 30.0, 10000, 555, id); F. info.toString()</string></pre>	F. "Info [lat=25.5, lon=30.0, time=10000, value=555, id=Optional .of(a)]"	

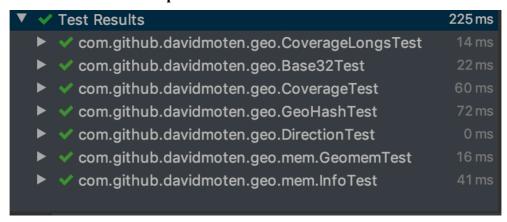
3 Test Implementation

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 test script implementations can be found in the <u>link</u> (or JUnit files).

No.	Test method	Source code
1	encodeBase32()	https://stv.csie.ntut.edu.tw/rojeanlin/GeoProj
2	decodeBase32()	ect/blob/master/src/test/java/com/github/da
3	getCharIndexException()	vidmoten/geo/Base32Test.java
4	testCoverage()	https://stv.csie.ntut.edu.tw/rojeanlin/GeoPr
5	getHashes()	oject/blob/master/src/test/java/com/github
6	getRatio()	/davidmoten/geo/CoverageTest.java
7	getHashLength()	
8	testToString()	
9	getHashes()	https://stv.csie.ntut.edu.tw/rojeanlin/GeoPr
10	getRatio()	oject/blob/master/src/test/java/com/github
11	getHashLength()	/davidmoten/geo/CoverageLongsTest.java
12	testToString()	
13	getCount()	
	opposite()	https://stv.csie.ntut.edu.tw/rojeanlin/GeoPr
14		oject/blob/master/src/test/java/com/github
		/davidmoten/geo/DirectionTest.java
15	adjacentHash()	https://stv.csie.ntut.edu.tw/rojeanlin/GeoPr
16	right()	oject/blob/master/src/test/java/com/github
17	left()	/davidmoten/geo/GeoHashTest.java
18	top()	
19	bottom()	
20	testAdjacentHash()	
21	neighbours()	
22	id()	https://stv.csie.ntut.edu.tw/rojeanlin/GeoPr
23	lat()	oject/blob/master/src/test/java/com/github
24	lon()	/davidmoten/geo/mem/InfoTest.java
25	time()	
26	value()	
27	testToString()	

4 Test Results

4.1 JUnit test result snapshot



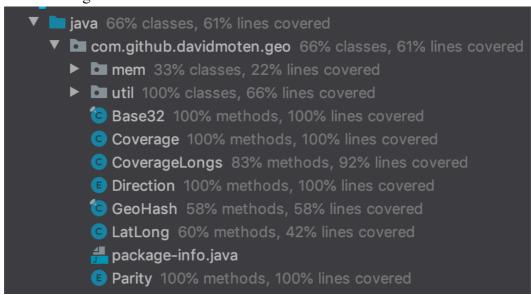
Test Summary



geo/build/reports/tests/test/index.html

4.2 Code coverage snapshot

• Coverage of each selected method



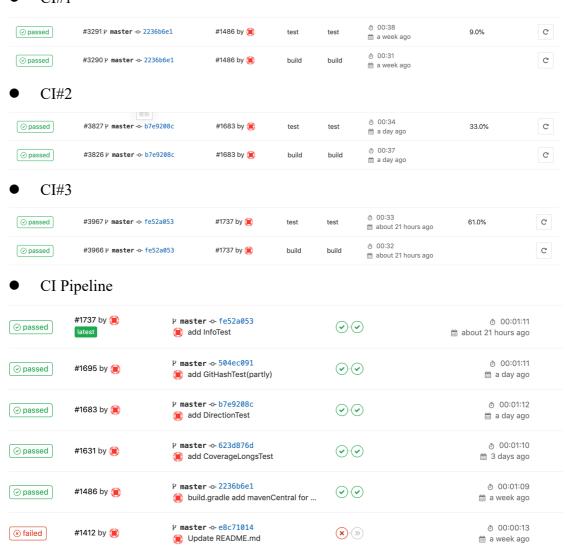
Total coverage

geo

Element	Missed Instructions +	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.github.davidmoten.geo		68%		55%	62	149	114	348	21	68	2	10
# com.github.davidmoten.geo.mem	=	19%	=	0%	23	30	48	61	13	20	2	3
com.github.davidmoten.geo.util		36%	1	50%	2	4	2	6	0	2	0	1
Total	903 of 2.326	61%	94 of 186	49%	87	183	164	415	34	90	4	14

4.3 CI result snapshot (3 iterations for CI)

• CI#1



5 Summary

In Lab 1, 27 test cases have been designed and implemented using JUnit. The test is conducted in 4 CI and the execution results of the 15 test methods are all passed. The total statement coverage of the test is 61%. Thus, the test requirements described in Section 1 are satisfied. Some lessons learned in this Lab are ...I learnt Junit and GeoHash algorithm in this lab.