

# Lab 3 Report

Name: 陳日揚

Student ID: 106598014

Date: 2018/4/22

## 1 Test Plan

### 1.1 Test requirements

Lab3 要求從 GeoProject 中(共 6 個 Classes, 50 個 Methods)挑選出 5 個 methods 來進行測試。首先, 必須先理解及分析待測程式的功能及目的為何? 本次 Lab 要求採用 Basis path testing 的技巧來設計 test cases, 因此我們必須先將程式轉化為控制流程圖(Control Flow Graph), 接著根據控制流程圖計算出 Cyclomatic Complexity, 最後找出 Basis test paths。我們必須根據 Basis test paths 來設計 test cases, 再將其轉換為可實際執行測試的程式碼。最後將測試的結果記錄在本測試報告中。

根據題目要求, 5 個待測的 Methods 皆必須被我們所設計出來的 Test cases 包含。同時, 整體程式的 statement coverage 要達到 60% (較 Lab2 高)。

### 1.2 Strategy

為了達成 Section1.1 所描述的需求, 將採用以下的策略:

- (1) 挑選與 Lab2 相同的 1 個 test methods, 並增加 4 個新的 test methods。
- (2) 挑選程式碼行數較多的 Method, 以增加測試覆蓋率。
- (3) 在設計測試案例前先分析與理解待測程式執行的內容, 同時也必須瞭解該領域的 domain knowledge 才能正確地定義出測試案例的輸出是否正確。
- (4) 採用 Basis Path 的技巧來設計 test cases。

### 1.3 Test activities

下列為本次測試過程中所包含的活動。

No.	Activity Name	Plan hours	Schedule Date
1	Study GeoProject	0.5Hr	2018/4/21
2	Learn Basis Path Slide	2 Hr	2018/4/21
3	Design test cases for the selected methods	8 Hr	2018/4/21-2018/4/22
4	Implement test cases	3 Hr	2018/4/22
5	Perform test	0.5Hr	2018/4/22
6	Complete Lab3 report	2Hr	2018/4/22

## 1.4 Design Approach

本次 Lab 採用 Basis path testing 的技巧來進行測試，Basis path testing 共有 X 個步驟，以下說明這次 Lab 實作這五個步驟的情況：

(1) 畫出控制流程圖：

將待測的五個 Methods 程式轉化為控制流程圖，同時紀錄圖中 Node 和 Edge 在程式中所對應到的敘述或區塊為何。

(2) 計算 Cyclomatic Complexity：

根據控制流程圖並使用以下三種方式計算 Cyclomatic Complexity：

- a. 圖中被 Edge 及 Node 所圍成的 Region 數量。
- b. Edge 數 - Node 數 + 結束點 Node 數 + 1。
- c. Prediction Node 數 + 1。

(3) 找出 Basis paths(或 Independent paths)：

上個步驟所計算出來的 Cyclomatic Complexity 為 Basis path 最多的數量，在此步驟中我們必須找出所有的 Basis paths。

(4) 設計 test cases：

設計可以達成所有 Basis paths 的 test cases。

(5) 找出 Infeasible test cases 並修正：

找出一些不可能達成的 test cases 並且透過如 tour 或 detour 的技巧來修正這些 test case，使其能夠達成覆蓋所有 Basis paths 的目的。

## 1.5 Success criteria

因為待測程式是一 Open Source 的 Library，理論上正確率應該非常高，因此本測試所設計之 Test cases 的通過率至少需達成 95%以上，並且單一 Method 的 Statement coverage 必須至少達成 85%。

## 2 Test Design

本次總共對 5 個 Method 進行測試，總共設計了 14 個 test cases。因本測試報告的版面限制，Test case 的詳細設計請參考附件 Excel 檔案([Lab3 \(Basis path test case design\).xlsx](#))。

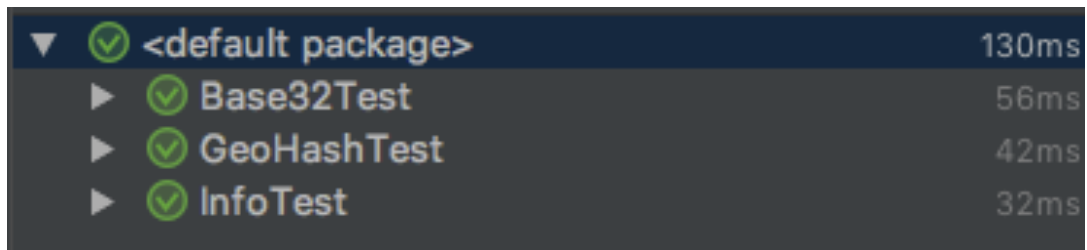
### 3 Test Implementation

本次 Lab 使用的測試工具為 Junit 4，下列挑選 Section 2 中設計三個 Test cases 實作內容，未列出的測試內容可在 GitLab 上查看。

No.	Test method	Source code
1	encodeHashToLong()	<pre>@Test public void encodeHashToLong_T1() {     long decodeHash = GeoHash.encodeHashToLong(23, 120, 0);     long assertHash = Long.parseLong("0");     assertEquals(assertHash, decodeHash); }</pre>
2	decodeHash()	<pre>@Test public void decodeHash_T3() {     LatLong latlong = new LatLong(0, 0);     LatLong decodeHash = GeoHash.decodeHash("");      assertEquals(latlong.getLat(), decodeHash.getLat(), 1);     assertEquals(latlong.getLon(), decodeHash.getLon(), 1); }</pre>
3	coverBoundingBoxMaxHashes()	<pre>@Test public void coverBoundingBoxMaxHashes_T6() {     Coverage coverage = GeoHash.coverBoundingBoxMaxHashes(80.0, 55.0, 30.0, 60.0, 5);     assertEquals("[t, v]", coverage.getHashes().toString()); }</pre>

## 4 Test Results

### 4.1 JUnit test result snapshot



▼ ✓ <default package>	130ms
▶ ✓ Base32Test	56ms
▶ ✓ GeoHashTest	42ms
▶ ✓ InfoTest	32ms

### Test Summary

72	0	0	0.237s
tests	failures	ignored	duration

**100%**  
successful

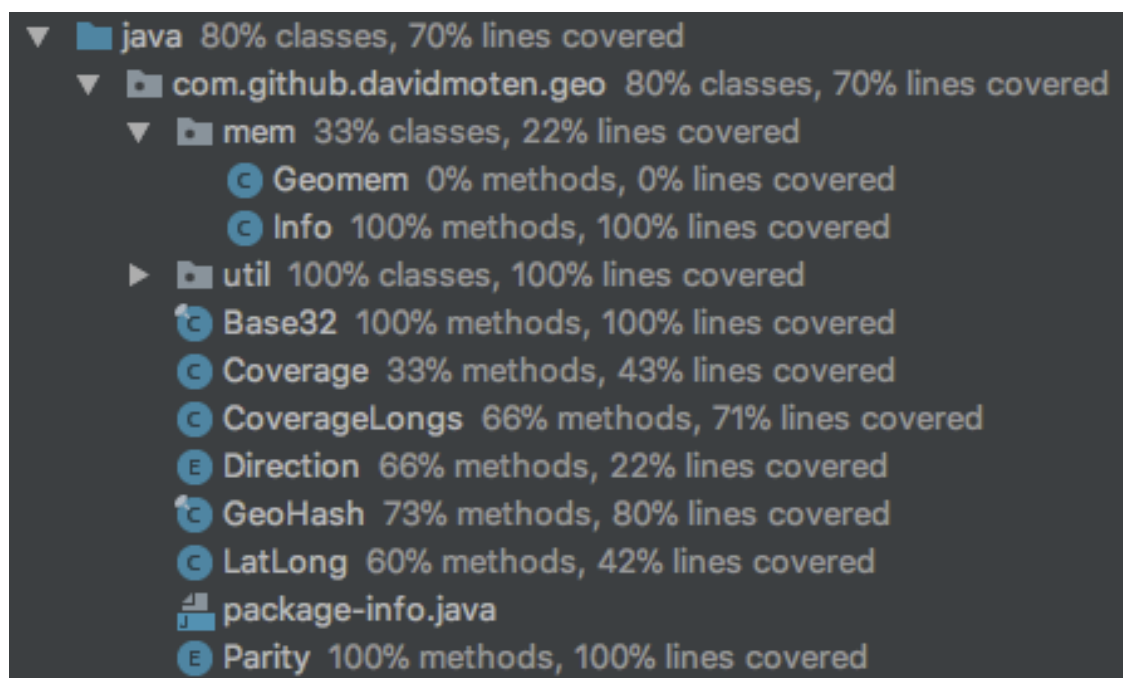
#### Packages

#### Classes

Package	Tests	Failures	Ignored	Duration	Success rate
<a href="#">com.github.davidmoten.geo</a>	57	0	0	0.166s	100%
<a href="#">com.github.davidmoten.geo.mem</a>	15	0	0	0.071s	100%

### 4.2 Code coverage snapshot

- Coverage of each selected method



▼ java 80% classes, 70% lines covered
▼ com.github.davidmoten.geo 80% classes, 70% lines covered
▼ mem 33% classes, 22% lines covered
C Geomem 0% methods, 0% lines covered
C Info 100% methods, 100% lines covered
▶ util 100% classes, 100% lines covered
C Base32 100% methods, 100% lines covered
C Coverage 33% methods, 43% lines covered
C CoverageLongs 66% methods, 71% lines covered
E Direction 66% methods, 22% lines covered
C GeoHash 73% methods, 80% lines covered
C LatLong 60% methods, 42% lines covered
package-info.java
E Parity 100% methods, 100% lines covered

- Total coverage

## geo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">com.github.davidmoten.geo</a>	<div><div></div></div>	78%	<div><div></div></div>	74%	58	159	83	354	28	78	0	10
<a href="#">com.github.davidmoten.geo.mem</a>	<div><div></div></div>	19%	<div><div></div></div>	0%	23	30	48	61	13	20	2	3
<a href="#">com.github.davidmoten.geo.util</a>	<div><div></div></div>	86%	<div><div></div></div>	100%	1	5	2	8	1	3	0	1
Total	718 of 2,379	70%	62 of 186	67%	82	194	133	423	42	101	2	14

## 4.3 CI result snapshot (5 iterations for CI)

- CI#1 ~ CI#2

README.md

pipeline

passed

coverage

50%

- CI#3

README.md

pipeline

passed

coverage

68%

- CI#4

README.md

pipeline

passed

coverage

69%

- CI#5

README.md

pipeline





















passed

coverage

70%

## ● CI Pipeline

106598014 > GeoProject > Pipelines

All 19 Pending 0 Running 0 Finished 19 Branches Tags					Run Pipeline	CI Lint
Status	Pipeline	Commit	Stages			
	#606 by  latest	✓ master → 1b7f8b3f Lab3: heightDegrees.	 			⌚ 00:01:12 📅 35 minutes ago
	#589 by 	✓ master → 9a42ce4a Lab3: fromLongToString.	 			⌚ 00:01:00 📅 about 7 hours ago
	#588 by 	✓ master → da08665f Lab3: coverBoundingBoxMaxHashes.	 			⌚ 00:01:11 📅 about 8 hours ago
	#578 by 	✓ master → 0d8e6919 Lab3: decodeHash.	 			⌚ 00:01:32 📅 about 23 hours ago
	#577 by 	✓ master → 446356e7 Lab3: encodeHashToLong.	 			⌚ 00:01:41 📅 a day ago

## 5 Summary

在此次的 Lab3 中，總共針對了 5 個 Methods 設計了 14 個 test cases，並且使用並利用 Junit 執行測試，測試的過程中共執行了五次的 CI。測試結果全數通過，且整體的 Statement coverage 為 70%，有達成 Section 1 所要求的 60% 覆蓋率，並且較上次 Lab 整體覆蓋率成長了 20%。同時，在這次 Lab 的執行過程中也學習到了 Basis path testing 的技巧，若 test case 有包含所有的 Basic path，則可以保證程式中的所有 Statements 及 Branches 皆有被執行到至少一次(包含迴圈也至少都有被執行到一次)，因此這是一種非常實用的 test case 設計技巧及方式。