xUnit testing – basics using VSCode and Java Junit 4 as example

The links to setup VSCode and Junit4

- Setup VSCode and Junit 4 https://www.youtube.com/watch?v=LRkqvZs857c&list=PL25sMKw559Gg9xlmLVLkmpS5XZJ-pvCd-
- How to start a unit test in VSCode https://www.youtube.com/watch?v=60yrTfVdFwo

- Note that, the examples in these videos are Junit 4 but we suggest using Junit 5

- JUnit4
  - https://junit.org/junit4/
- JUnit5
  - https://junit.org/junit5/docs/current/user-guide/#writing-tests
- 如果你的 import 裡面有 jupiter 這樣的字眼，你所用的是 JUnit5
- 當你照著前面的 youtube 影片安裝之後，JUnit5 也是可以用的。

Let's continue the example in the youtube but write the test cases into a separate class

src > App.java > ...

```java
public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        System.out.println("Hello, World2!");
    }
   public static int higher(int x, int y) {
    if (x>y) {
        return x ;
    } else {
        return y ;
    }
   }

}
```

Click to run all test cases

Run a single test case

src > testApp.java > ...

```java
import static org.junit.Assert.assertEquals;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class testApp {

    @Test
    public void testHigher() {
        assertEquals(25, App.higher(25,13));
        System.out.println("test higher 1");
    }
    @Test
    public void testHigher2() {
        assertEquals(25, App.higher(0, 25)) ;
        System.out.println("test higher 2");
    }
}
```

Switch to debug console to see print out

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

```
test higher 2
test higher 1
```

Annotations in JUnit

Annotations in JUnit5

- https://www.geeksforgeeks.org/annotations-in-java/

BeforeEach 指的是每個 test case 被執行起來之前都會先被執行
method 不能加 static

AfterALL 指的是所有的 test cases 都被執行結束之後
method 一定static

Annotations 所造成的執行順序

```java
import static org.junit.Assert.assertEquals;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class testApp {
    @BeforeEach
    public void setUp() {
        System.out.println("this is a start");
    }
    @AfterEach
    public void tearOneDown() {
        System.out.println("this is the one end ");
    }
    @AfterAll
    public static void tearDown() {
        System.out.println("this is the end ");
    }
    @Test
    public void testHigher() {
        assertEquals(25, App.higher(25,13));
        System.out.println("test higher 1");
    }
    @Test
    public void testHigher2() {
        assertEquals(25, App.higher(0, 25)) ;
        System.out.println("test higher 2");
    }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
this is a start
test higher 2
this is the one end
this is a start
test higher 1
this is the one end
this is the end
```

Test Oracles
Theory and Principles

Automatic Test Case Generation and Test oracles

- If test cases can be derived automatically and then these test cases can be executed automatically and then report the error – **What a perfect world !** We can fire all the testers.
- In practice, this is only a dream
- In many applications, effective test cases can only be derived by humans (自動化的幫你執行程式，例如幫你點擊 GUI 元件，這是可行的，但是你為何沒有用過? 那些人，那些神祕的地方用過這個自動化測試技術? 有聽過 monkey tests?)
- In many applications, executions requires human to monitor and validate if program behaviors are conformed to specs. (幫你自動化執行程式其實不是主要的困難，而是自動化執行你的待測系統時，要能夠知道你的程式出錯了，或是有 bug)

Test Oracles (別跟 Oracle DB 混為一談)

- Oracle – In ancient Greece, an oracle was a priest or priestess who made statements about future events or about the truths

- A **test oracle** is a program that can determine if the program behaviors or program input/output are conformed to specs.

Testing automation

- For testing automation, test oracle must be presented to determine the conformity between specs and programs.
- In some applications, test oracles are easy to derive
- In most applications, test oracles are difficult to implement or in theory, impossible to derive automatically

- Yes, Building test automation is not easy

One of the Holy Grail problems in Software Engineering

Indiana Jones inspiration

The "Holy Grail" chalice used as a prop in the movie "Indian Jones and the Last Crusade."

Assert (Test oracle to determine the success or failure of your test)

- assertEquals(expected, actual)
- assertEquals(message, expected, actual)
- assertEquals(expected, actual, delta)
- assertEquals(message, expected, actual, delta)
- assertFalse(condition)
- assertFalse(message, condition)
- Assert(Not)Null(object)
- Assert(Not)Null(message, object)
- Assert(Not)Same(expected, actual)
- Assert(Not)Same(message, expected, actual)
- assertTrue(condition)
- assertTrue(message, condition)

About Test Oracle

- In xUnit tests, 理論上你把你寫的 class (Class Under Test, CUT) 視為一個黑箱
- 你的測試行為
  - 用你的 CUT 建立物件
  - 餵資料給你的物件
  - 呼叫你的 CUT 的 method
  - 然後撰寫 test oracle (assert) 來驗證程式執行結果是否與你準備好的答案一樣
- 理論上，你應該只可以跟你的 CUT 提供的 public accessing member/methods/... 來進行互動

Black Box Testing

Test Cases

你能夠對一個時鐘做什麼動作呢?

- 轉時針/分針
- 裝電池/拆電池
- 觀察面板然後讀出時間

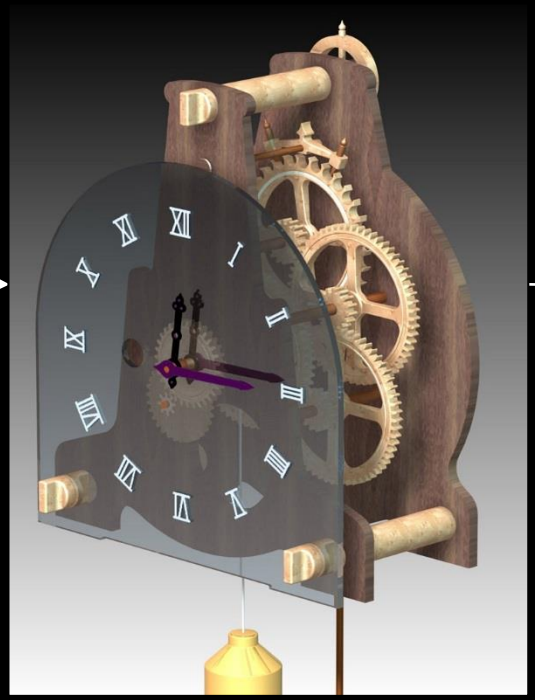
Outputs

Unit test = Black box Testing+ White Box Testing

- 實務上，當你在做單元測試的時候你其實是有 source code 在手
- 有時候為了測試需要，你也可以額外增加一些程式碼 (code for test purpose) 到你的 CUT 裡面
  - 讓你能夠讀取 internal 變數值供 test code 進行判斷
  - Constructor injection (當你需要使用 fake object 像是 test stub, mocks..)

White box testing
(一邊看著你的 source code 一邊推導你的 test cases)

Test cases



output

You test an equipment by observing the internal behaviors of the equipment

also known as structural testing (text book)

Let’s test a studentCollection implemented by ArrayList

```java
import java.util.ArrayList;
import java.util.List;
public class studentCollection {
    private List<String> students = new ArrayList<String>();
    public void remove(String name) {
        students.remove(name);
    }
    public void add(String name) {
        students.add(name);
    }
    public void removeAll(){
        students.clear();
    }
    public int sizeOfStudent() {
        return students.size();
    }

}
```

src > testStudentCollection.java > testStudentCollection > testRemove()

```java
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class testStudentCollection {
    studentCollection obj = new studentCollection();
    @Test
    public void testAdd() {
        System.out.println("The size of obj = "+obj.sizeOfStudent());
        obj.add("Emma");
        obj.add("Ronan");
        obj.add("Antonio");
        obj.add("Paul");
        assertEquals("Adding 4 student to list", 4, obj.sizeOfStudent());
    }
    @Test
    public void testSize() {
        System.out.println("The size of obj = "+obj.sizeOfStudent());
        obj.add("Emma");
        obj.add("Ronan");
        obj.add("Antonio");
        assertEquals("Checking size of List", 3, obj.sizeOfStudent());
    }
    @Test
    public void testRemove() {
        System.out.println("The size of obj = "+obj.sizeOfStudent());
        obj.add("Antonio");
        obj.add("Paul");
        obj.remove("Paul");
        assertEquals("Removing 1 student from list", 1, obj.sizeOfStudent());
    }
    @Test
    public void removeAll() {
        System.out.println("The size of obj = "+obj.sizeOfStudent());
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Filter (e.g. text, !exclude)

```
The size of obj = 0
```

Notes

- 每一次 test method 被執行之前 obj 都會被重新建立，意思是 testStudentCollection class 每一次會被 new 一次，
- 如果你只有一個 sizeOfStudent() 可以 probe CUT studentCollection
-

Let’s add a bubble sort method and write a straightforward test oracle about it

```java
1   public class App {
        Run | Debug
2       public static void main(String[] args) throws Exception {…
11      public static int higher(int x, int y) {…
18      public static int[] asscendingBubbleSort(int[] inputarray) {
19          Boolean change ;
20          int temp ;
21          change = false ;
22          do {
23              change = false ;
24              for (int i=0; i< inputarray.length-1; i++) {
25                  if (inputarray[i] > inputarray[i+1]) {
26                      temp = inputarray[i] ;
27                      inputarray[i] = inputarray[i+1] ;
28                      inputarray[i+1] = temp ;
29                      change = true ;
30                  }
31              }
32          } while (change);
33          return inputarray ;
34      }
35
36  }
```

43-42 的 test oracle
有個很有趣的特質

```java
10  public class testApp {
11      @BeforeEach
12      public void setUp() {…
15      @AfterEach
16      public void tearOneDown() {…
19      @AfterAll
20      public static void tearDown() {…
23      @Test
24      public void testHigher() {…
28      @Test
29      public void testHigher2() {…
33
34      @Test
35      public void testAscendingBubbleSort() {
36          int data[] = new int[] { 5,4,3,2,1} ;
37          // int returndata[] ;
38          App.asscendingBubbleSort(data) ;
39          for (int i=0;i<data.length; i++) {
40              System.out.println(data[i]);
41
42          }
43          for (int i=0;i<data.length-1;i++) {
44              assertTrue(data[i] < data[i+1]);
45          }
46      }
47  }
48
```

```java
public void testHigher() {
    assertEquals(25, App.higher(25,13));
    System.out.println("test higher 1");
}
@Test
public void testHigher2() {
    assertEquals(25, App.higher(0, 25)) ;
    System.out.println("test higher 2");
}
```

```java
public void testAdd() {
    System.out.println("The size of obj = "+obj.sizeOfStudent());
    obj.add("Emma");
    obj.add("Ronan");
    obj.add("Antonio");
    obj.add("Paul");
    assertEquals("Adding 4 student to list", 4, obj.sizeOfStudent());
}
@Test
public void testSize() {
    System.out.println("The size of obj = "+obj.sizeOfStudent());
    obj.add("Emma");
    obj.add("Ronan");
    obj.add("Antonio");
    assertEquals("Checking size of List", 3, obj.sizeOfStudent());
}
```

- 觀察 1: 你的 assertXXXXX 是隨著 test case 而變動的
- 觀察 2: 舉 testAdd() 為例，4 是隨著測試 scenarios 而變動。如果你多加一個 obj.add，你就得改成 5

Input independent test oracle

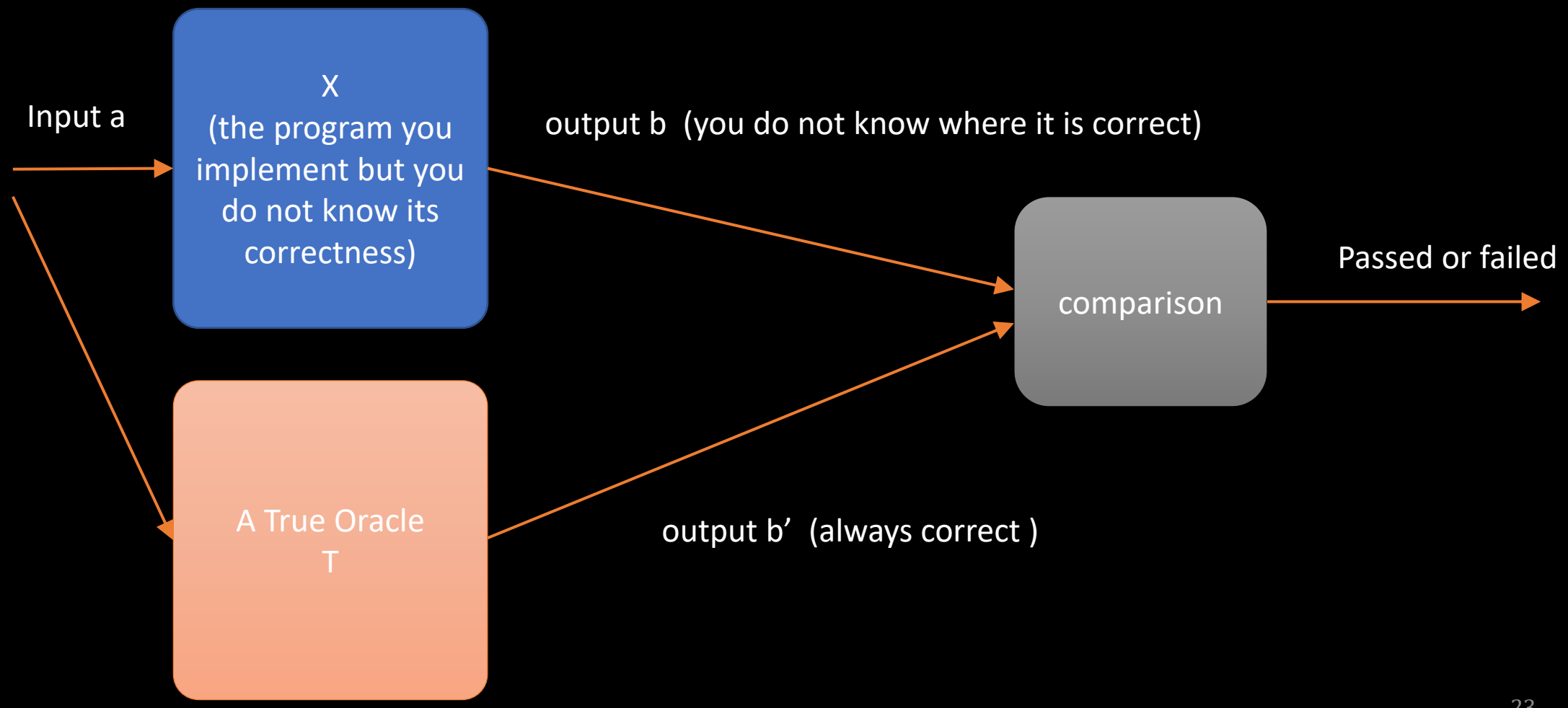
```
@Test
public void testAscendingBubbleSort() {
    int data[] = new int[] { 5,4,3,2,1} ;
    // int returndata[] ;
    App.asscendingBubbleSort(data) ;

    for (int i=0;i<data.length-1;i++) {
        assertTrue(data[i] < data[i+1]);
    }
}
}
```

- 觀察: 你會發現你可以新增任一個 bubblesort test cases，但是都共用一套 test oracle 就可以了
- 這樣不就是夢想中全能的測試自動化，test data 甚至可以 randomly generated?
- 我們是不是就都來找這樣子的 test oracle 就好了?

Test Oracle ???

- A test oracle is a program that can determine if the program behaviors or program input/output are conformed to specs; that is, passed or failed.
- Here we introduce three kinds of test oracle to understand the theory behind
  - True Oracle
  - Partial Oracle
  - Test-Oriented oracle (xUnit test)

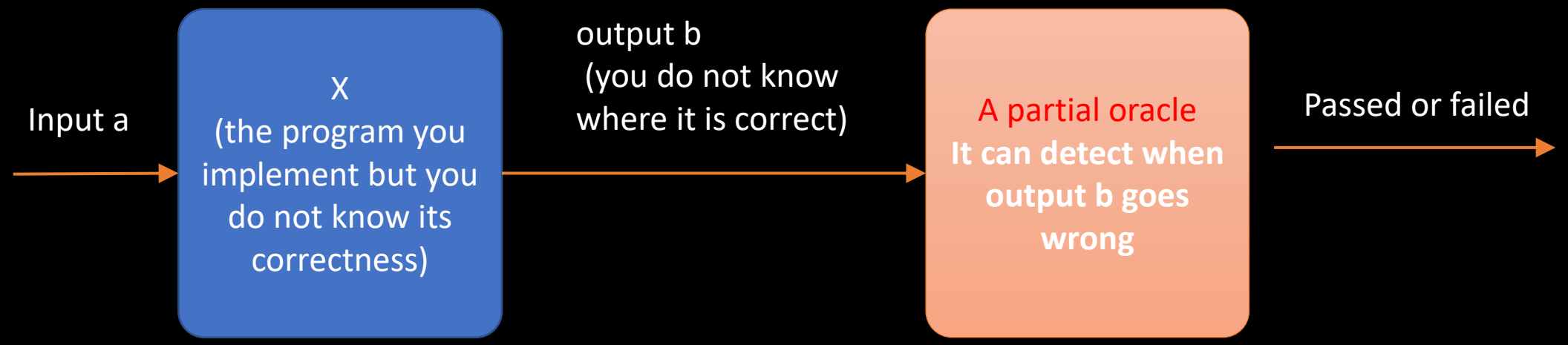
True Oracle


Input a
X
(the program you implement but you do not know its correctness)
output b (you do not know where it is correct)
comparison
Passed or failed
A True Oracle
T
output b' (always correct)

True Oracle

- If true oracle T already exist, why do you need to implement system X?
- So, in computation theory, once you have a true oracle T, you already have a system X
- True oracles simply do not exist in general

Partial Oracle



Example

```
pre_array = array;  //  make a copy of array before sorting
mySort(n,array); // your sorting algorithm of an ascending order
// a true oracle  (nonexisted !!)
correct_array = TrueOracle(prearray) ;
AssertEqual(array, correct_array)

// a partial oracle
for (i=0;i<n ;i++) {
    assertTrue(array[i] <= array[i+1]);
}
```

True/Partial Oracles

- True Oracle does not always exist
- Partial Oracle may exist for some problems but it is difficult to find in most cases
  - Try to find the partial oracle for an AVL tree.
- True Oracle/Partial Oracle if exists, it can deal with all inputs.
- You can simple generate many test data as many as possible
- That would be a dream comes true !!! Real test automation (test case generation) to find your bugs.

- Don't be silly !! Not workable for too many practical problems. (e.g., design contract)
- Abandoned by programmers in practice.

Test Oriented Oracle

- xUnit test is a test oriented oracle
- Mostly your oracle only work for a test
- It does not work for all inputs.

- in xUnit tests, you often need to write adequate/critical tests to be useful

ParameterizedTest vs Partial Test Oracles

- JUNIT 允許你把 test oracle 的變數參數化(如右上圖)，也能夠把參數存到 CSV
- https://www.baeldung.com/parameterized-tests-junit-5
- 不過這與 Partial test oracles 還是不太一樣的
- ParameterizedTest 能參數化 test oracles 的變數，但是行為就不見得。
- 以 testAdd() 為例，obj.add 的行數稱之為行為，這個行為也要能被參數化

```java
@ParameterizedTest
@ValueSource(ints = {1, 3, 5, -3, 15, Integer.MAX_VALUE}) // six numbers
void isOdd_ShouldReturnTrueForOddNumbers(int number) {
    assertTrue(Numbers.isOdd(number));
}
```

```java
public void testAdd() {
    System.out.println("The size of obj = "+obj.sizeOfStudent());
    obj.add("Emma");
    obj.add("Ronan");
    obj.add("Antonio");
    obj.add("Paul");
    assertEquals("Adding 4 student to list", 4, obj.sizeOfStudent());
}
@Test
public void testSize() {
    System.out.println("The size of obj = "+obj.sizeOfStudent());
    obj.add("Emma");
    obj.add("Ronan");
    obj.add("Antonio");
    assertEquals("Checking size of List", 3, obj.sizeOfStudent());
}
```

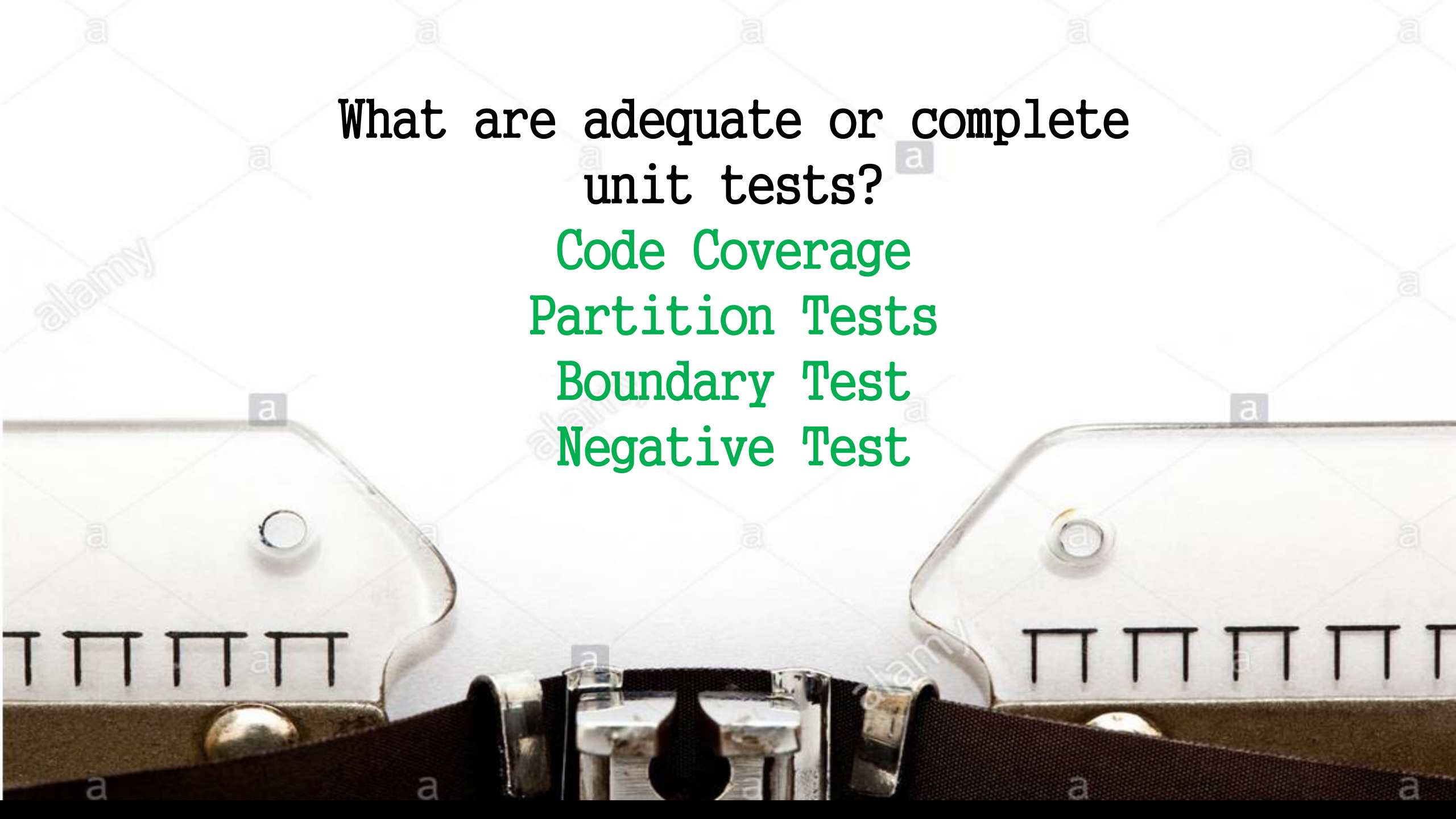
What are adequate or complete unit tests?

Code Coverage

Partition Tests

Boundary Test

Negative Test



大哉問:

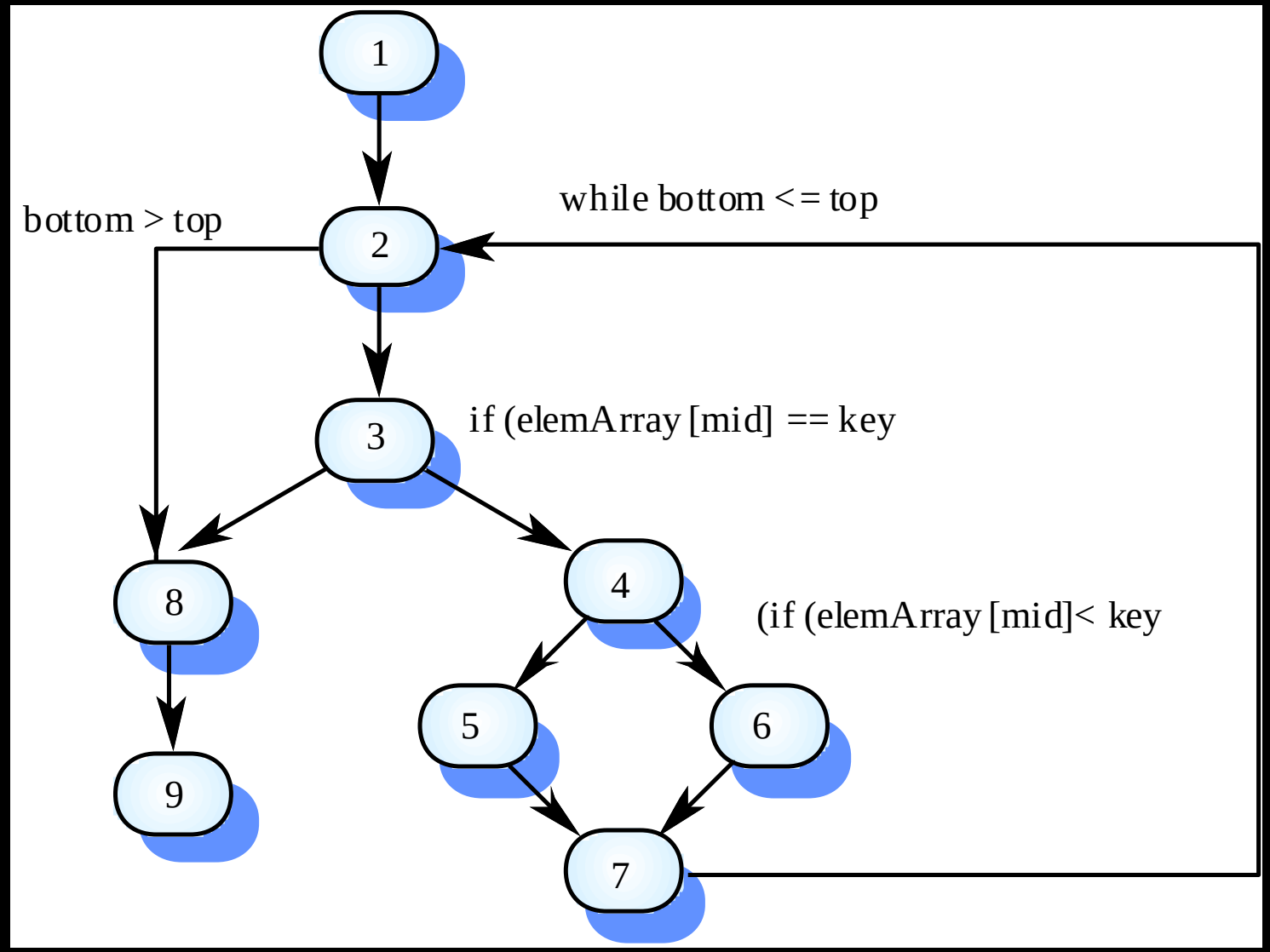
你怎麼確認你的程式沒有問題 (你怎麼知道你測試夠了?)

這並不是一個可以簡單回答的的問題

但是通常這是一個立即有效的實力偵測指標

Code Coverage

Code Coverage Theory

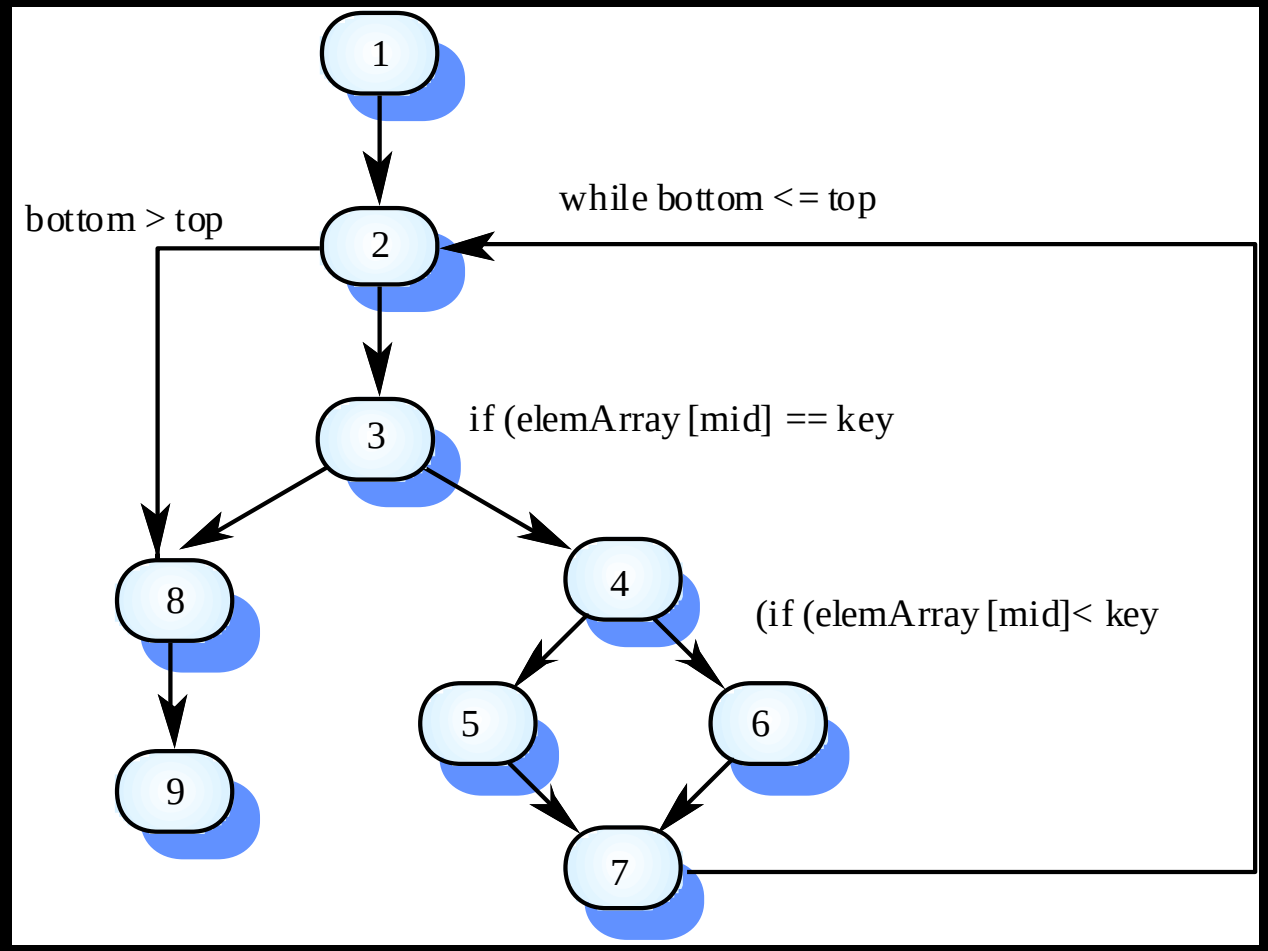


The weakest criteria

- Statement coverage
  1 test case to run 1 2 3 8 9

  1 test case to run 1 2 4 6 7 2

  1 test case to run 1 2 4 5 7 2 8
- You only need 3 test cases to have each statement at least executed once.



Comments about statement coverage

- Testing 的最低標
- 以 higher() 為例，你只要兩個 test cases 就能達標 (100%)
- 以 asscendingBubbleSort() 為例，你只要一個 5 4 3 2 1 測試案例就能達標 (100%)
- 所以 statement coverage 達標是個非常低的測試標準。

```java
public static int higher(int x, int y) {
 if (x>y) {
     return x ;
 } else {
     return y ;
 }
}
```

```java
public static void asscendingBubbleSort(int[] inputarray) {
  Boolean change ;
  int temp ;
  change = false ;
  do Boolean change - App.asscendingBubbleSort(int[])
    change = false ;
    for (int i=0; i< inputarray.length-1; i++) {
        if (inputarray[i] > inputarray[i+1]) {
            temp = inputarray[i] ;
            inputarray[i] = inputarray[i+1] ;
            inputarray[i+1] = temp ;
            change = true ;
        }
    }
  } while (change);
```

Branch coverage

- each branch can produce two choices, every choice combinations should all be exercised
- there are 3 branches, so at most 2 * 2 * 2 branching choices should be exercised if loop is not considered.
- So at least you need to find test cases to meet this criteria

  1 2 8 9

  1 2 3 8 9

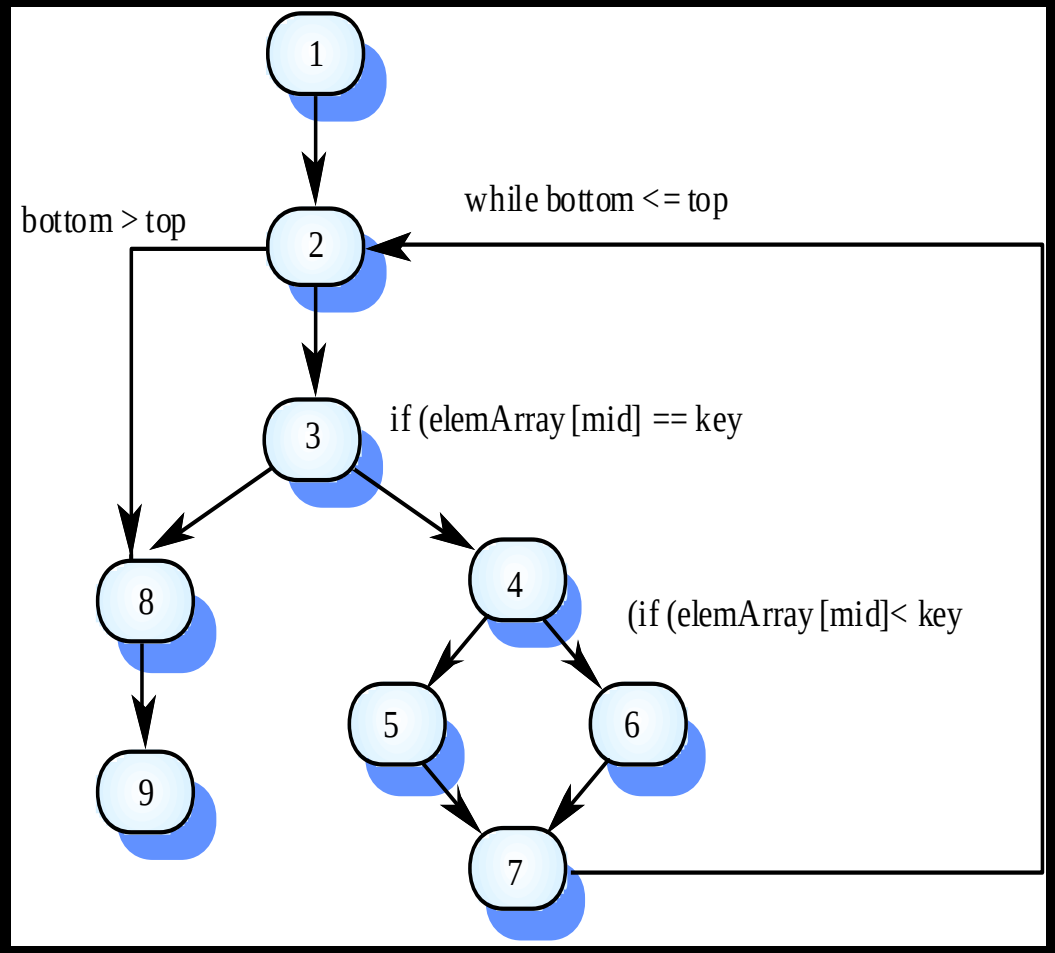
  1 2 3 4 5 7 2 8 9

  1 2 3 4 5 7 2 3 8 9

  1 2 3 4 6 7 2 8 9

  1 2 3 4 6 7 2 3 8 9

- In the example, there are less than 8 branching choices because it is not a complete tree



Path Coverage

- The highest coverage
- Try to find test cases which can cover all the paths
- equal to exhaustive testing if you want to cover them all, which is impossible
- You need to find infinite test cases which have finite (if the program must stop) or infinite length (if the program may run forever)

  1 2 8 9 (finite length)

  1 2 3 8 9 (finite length)

  1 2 3 4 5 7 2 8 9

  1 2 3 4 5 7 2 3 4 5 7 2 8 9

  1 2 3 4 5 7 2 3 4 6 7 2 3 8 9

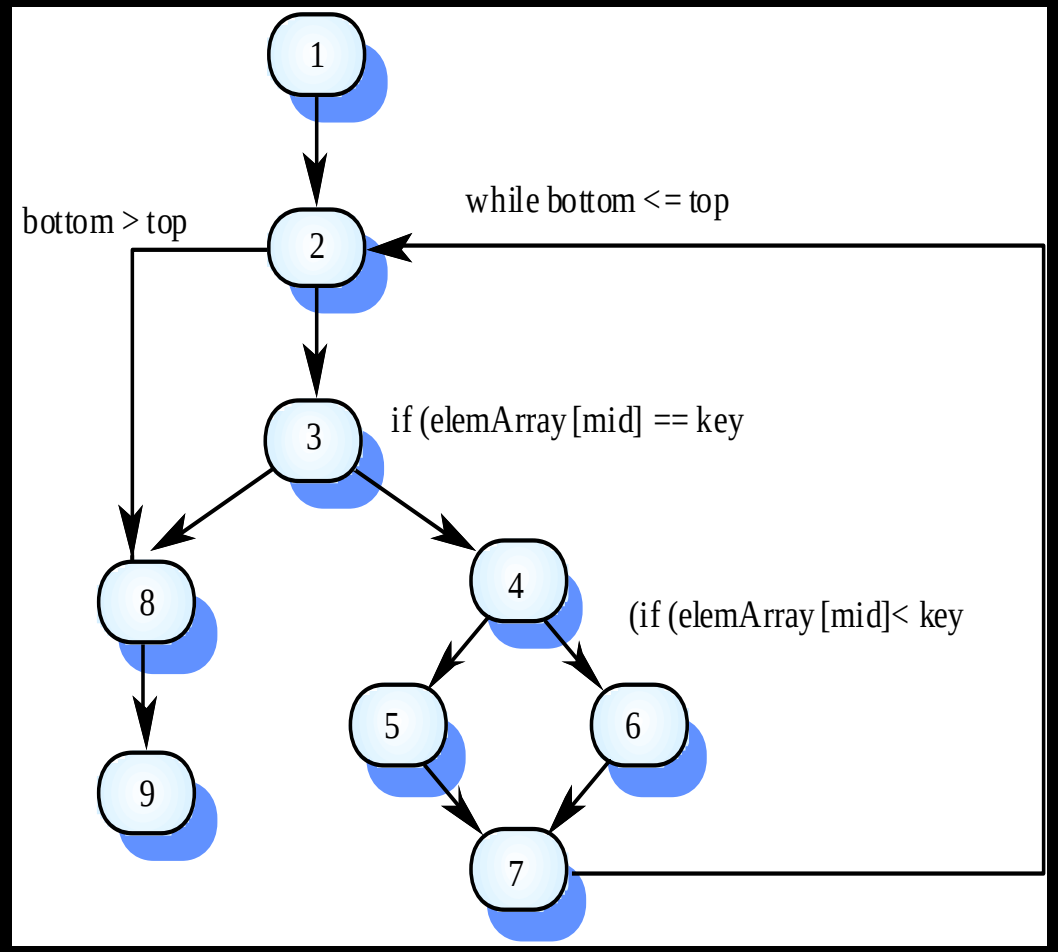
  1 2 (3 4 5 7 2) * 8 9 (finite length but very long sequence)

  1 2 (3 4 5 7 2 | 3 4 6 7 2) * 8 9 (finite length with permutations)

  1 2 (3 4 5 7 2) ω 8 9

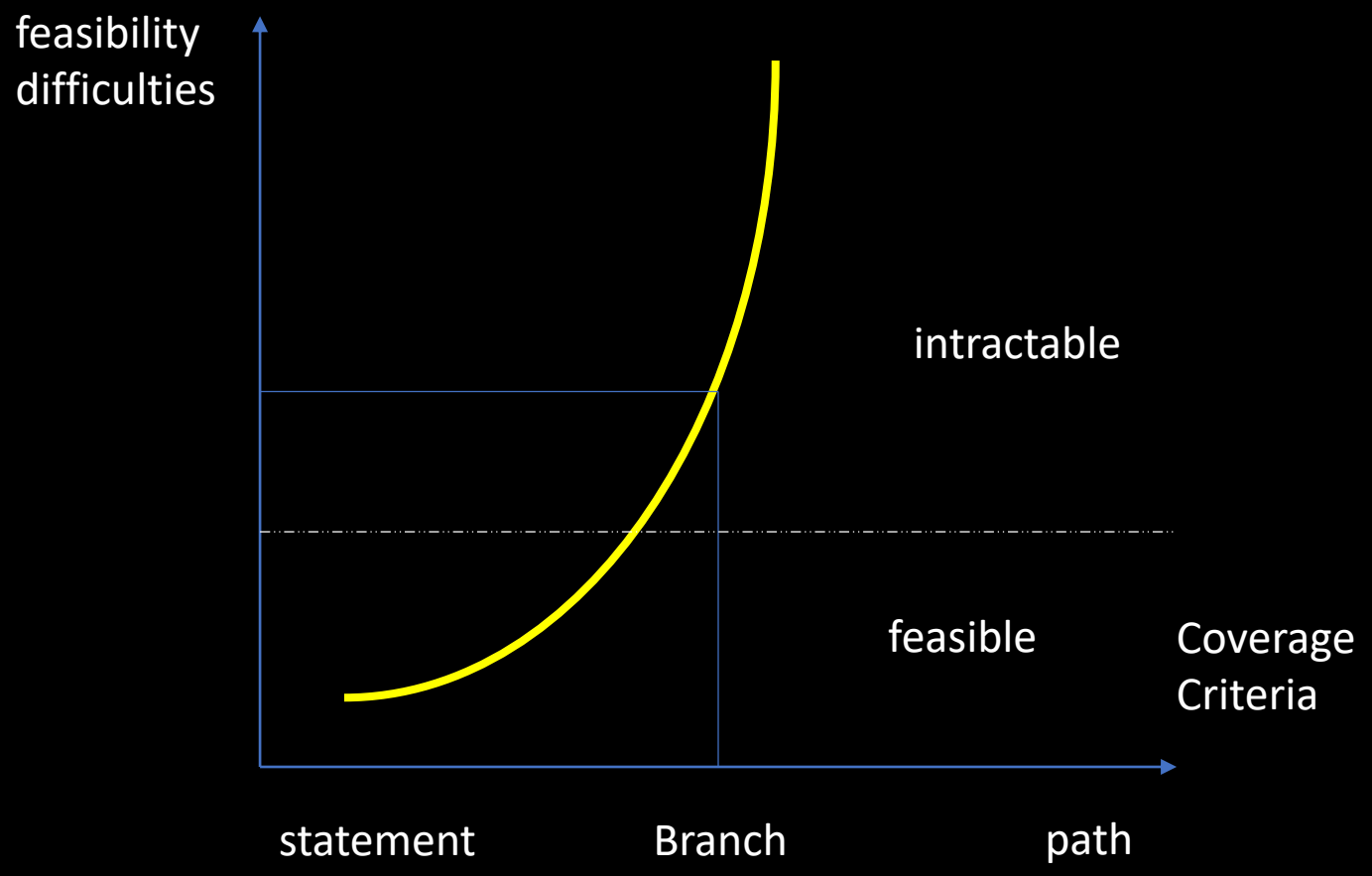
  .....

  .....
- where * is finite iteration in automata theory
- where ω is infinite iteration in automata theory



Summary about branch Coverage

- 理論上非常有趣，你如果想達到的標準是 branch coverage，實務上他已經很困難了
- 聽說過 branch coverage 是金管會證交系統的驗收標準



ScreenShots of Code Coverage Tools

Code Coverage

TablePlanner_3_1_1.exe:Fri Jan 11 16:20:08 2008 (Ready) - Coverage Validator -

File Edit Configure Managers Query Tools Help

Summary | Coverage | Functions | Files and Lines

File
Totals
c:\perfecttableplan\product\trunk\src\tpgradientdlgimpl.cpp
c:\perfecttableplan\product\trunk\src\tpgradientpreviewlabel.c
c:\perfecttableplan\product\trunk\src\tpgradientwidgetimpl.cp
c:\perfecttableplan\product\trunk\src\tpgraphicalobject.cpp
c:\perfecttableplan\product\trunk\src\tpgraphicsitem.cpp
c:\perfecttableplan\product\trunk\src\tpgraphicsscene.cpp
c:\perfecttableplan\product\trunk\src\tpgraphicstip.cpp
c:\perfecttableplan\product\trunk\src\tpgraphicsview.cpp
c:\perfecttableplan\product\trunk\src\tpgroup.cpp
c:\perfecttableplan\product\trunk\src\tpgrouplistviewitem.cpp
c:\perfecttableplan\product\trunk\src\tpgroupwidgetimpl.cpp
c:\perfecttableplan\product\trunk\src\tpgroupslistview.cpp
c:\perfecttableplan\product\trunk\src\tpguest.cpp
c:\perfecttableplan\product\trunk\src\tpguestchartview.cpp
c:\perfecttableplan\product\trunk\src\tpguestchartviewfilterim
c:\perfecttableplan\product\trunk\src\tpguestdlgimpl.cpp
c:\perfecttableplan\product\trunk\src\tpguestfamilydlgimpl.cp
c:\perfecttableplan\product\trunk\src\tpguestlistviewitem.cpp
c:\perfecttableplan\product\trunk\src\tpguestnameconfabulatc
c:\perfecttableplan\product\trunk\src\tpguestviewfilterimpl.cpp
c:\perfecttableplan\product\trunk\src\tpguestwidgetimpl.cpp
c:\perfecttableplan\product\trunk\src\tpguestslistview.cpp
c:\perfecttableplan\product\trunk\src\tpguestsscrollview.cpp
c:\perfecttableplan\product\trunk\src\tphtmldocument.cpp

Refresh | Graphical... | Sort: None

c:\perfecttableplan\product\trunk\src\tpguest.cpp
C:\perfecttableplan\product\tags\3.1.1c\binaries\windows\program\TablePlanner_3_1_1.exe
Num Lines 167, Visited 114, Percent Visited 98.28%, Total Number of Visits 44320646, Not hooked 51

Unvisited | Line | Group | Visited

319			
320			QString
321			TPGuest::ageToString(TPGuest::Age age)
322	13,015	✓	{
323	13,015	✓	QString s;
324	466	✓	switch (age)
325			{
326			case Adult:
327	12,549	✓	s = "adult";
328			break;
329			
330			case Child:
331	466	✓	s = "child";
332	0	✗	break;
333			
334			default:
335			assert(false);
336			break;
337			}
338	13,015	✓	return s;
339	13,015	✓	}
340			
341			TPGuest::Age
342			TPGuest::stringToAge(const QString& age)
343	8,634	✓	{
344			TPGuest::Age a;
345	8,634	✓	if (age.lower() == "adult")
346			{
347	0	✗	a = Adult;
348			}
349	1,049	✓	else if (age.lower() == "child")
350			{
351	1,049	✓	a = Child;
352			}
353			else
354			{
355	0		qWarning("undefined age");
356			a = Adult;
357			}
358			return a;
359	8,161	✓	}

Ready | 70971 | 50270 | 23396 | Collect:On | TablePlanner_3_1_1.exe:Fri Jan 11 16:20:08 2008 (Ready)

Code Coverage Reports

AQtime - C:\AQA\AQtime7ScreencastScripts\Using AQtime Coverage Analysis in Automated Tests\Using AQtime Coverage Analysis in Automated Tests\Coverage_exe.aqt

File Edit View Project Run Options Help

Light Coverage Profiler

Start Page | Setup | Results

Explorer

Name
- Last Results
  - 12/10/2010 11:07:0...
    - Routines Data
    - Modules Data
    - Source Files Data
  - 12/7/2010 6:22:00 PM
  - 12/7/2010 10:43:27 ...
- Saved Results
- Merged Results

Report | Summary

Routine Name	Total Lines	Lines Uncovered	% Covered	Mark	Method ...
MainForm::.ctor	5	0	100.00 %		
MainForm::Dispose	10	10	0.00 %		
MainForm::InitializeComponent	113	0	100.00 %		
MainForm::Main	3	1	66.67 %		
MainForm::DoActionA	3	0	100.00 %		
MainForm::DoActionB	3	3	0.00 %		
MainForm::DoActionC	3	3	0.00 %		
MainForm::CoverageTest	9	4	55.56 %		
MainForm::CloseButton_Click	3	3	0.00 %		
MainForm::ExecuteButton_Click	9	1	88.89 %		
MainForm::LinkLabel1_LinkClicked	10	10	0.00 %		
<Unknown PInvoke>			0.00 %		

Editor

File name: Cannot find the source file.

Search Directories...
Project Search Directories...

Event View | Monitor | Disassembler | Editor | Details | Call Graph | Call Tree | PE Reader

Assistant

Analyze Results

View Results in AQtime Panels

The main profiling results are displayed in the Report panel. The Summary panel gives you brief statistics on the entire profiler run. AQtime uses some other panels to provide more information in addition to results displayed in the Report and Summary panels:

- Summary
- Report
- Explorer
- Details
- Call Graph
- Call Tree
- Editor
- Disassembler
- PE Reader
- More about AQtime panels
- More about analysis of results

Manage Profiling Results

Choose a category of results:

Routines Data

- Search Results...
- Filter Results...

Apply a view:

<Current View>

- Show Additional Fields...
- Format Columns...
- Comparing and Merging Results

Export or Print Results

- Save to a File...
- Load from a File...
- Export to HTML, XML, Excel or Text File...
- Print Results...

Code coverage JaCoCo + Maven in VSCode

- 給認真的同學
- https://medium.com/@karlrombauts/setting-up-unit-testing-for-java-in-vs-code-with-maven-3dc75579122f

```java
public class BankAccount {
    private int balance = 0;

    public BankAccount(int startingBalance) {
        this.balance = startingBalance;
    }

    public boolean withdraw(int amount) {
        if(balance ≥ amount) {
            balance -= amount;
            return true;
        }
        return false;
    }

    public int deposit(int amount) {
        balance += amount;
        return balance;
    }
```

VsCodeJavaUnitTests > com.karlrombauts.VsCodeJavaUnitTests > BankAccount Session

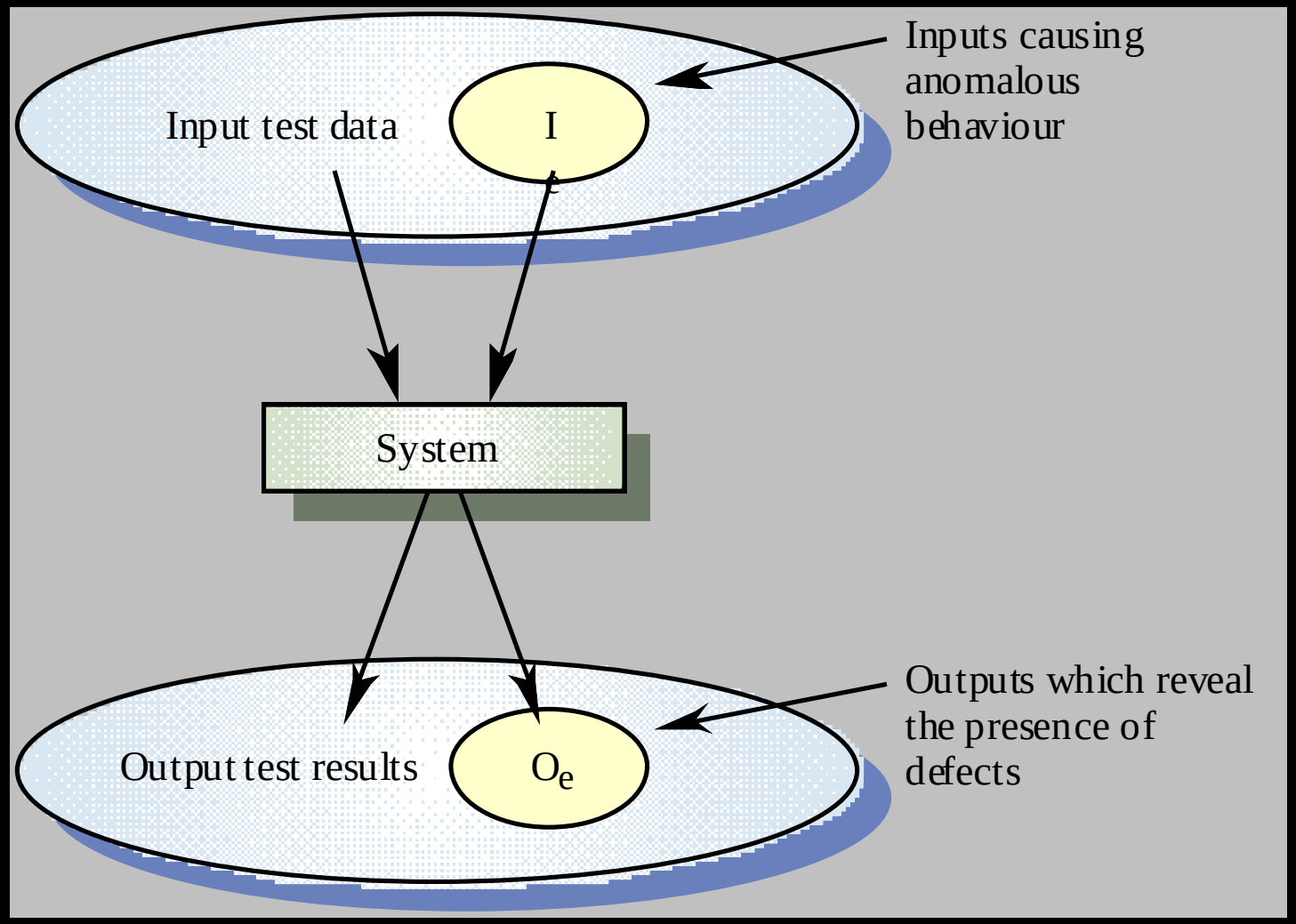
BankAccount

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Miss
deposit(int)		0%		n/a	1	1	2	2	
getBalance()		0%		n/a	1	1	1	1	
withdraw(int)		85%		50%	1	2	1	4	
BankAccount(int)		100%		n/a	0	1	0	4	0
Total	14 of 35	60%	1 of 2	50%	3	5	4	11	2

Created with JaCoCo 0.8.4.201905082037

Partition Tests

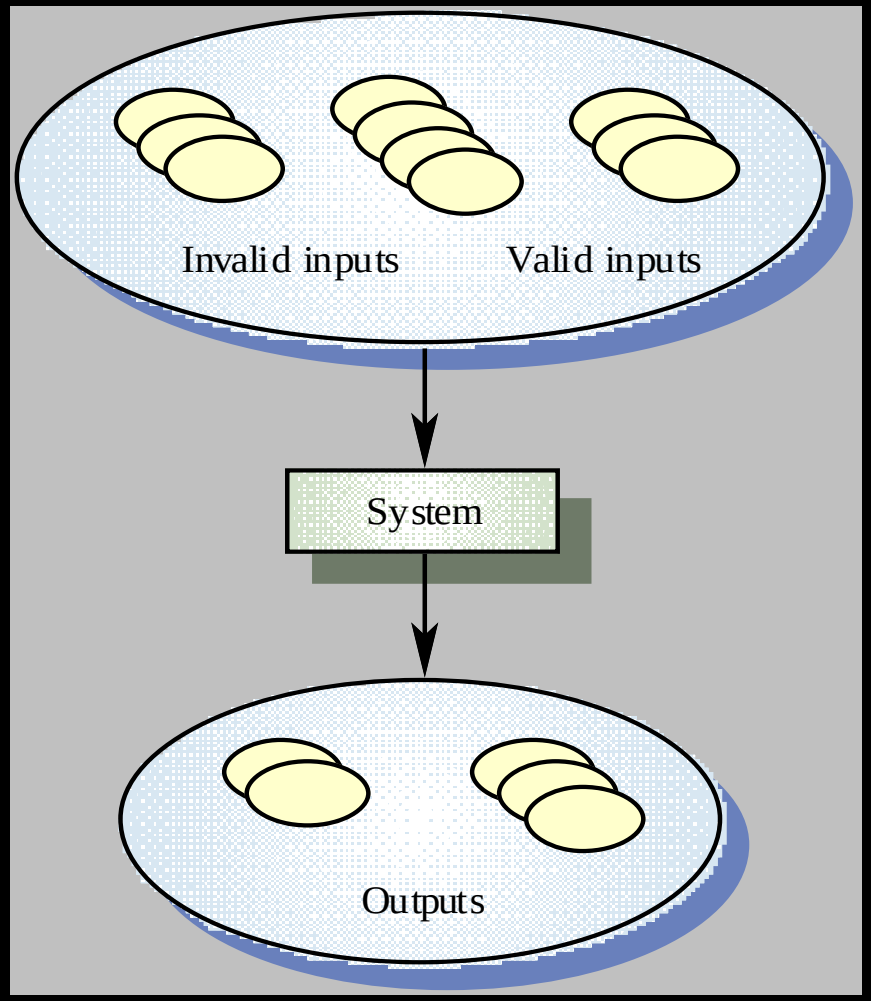
人類最古老的測試理論 Partition Tests



Equivalence partitioning (partition testing)

- Input data and output results often fall into different classes where all members of a class are related
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each partition

Equivalence partitioning



Partition Tests in the two examples

- In higher(), 兩個 partitions。1. 數字前者小於後者， 2. 數字後者小於前者
- In ascendingBubbleSort()，程式碼雖然短，但是因為有陣列(變數)。陣列裡面的 partitions 非常多。舉例來說 1 2 3 4 5 與 10 25 33 78 99 都是遞增，他們執行的程式碼行數(path coverage) 都是 17 18 19 18 19 18 19 18 19 25 26 屬於同一個 partition。執行起來他們的 path trace 會一模一樣
- 理論上你必須要測的 partitions 有
  - 5 4 3 2 1 (遞減)
  - 1 2 3 4 5 (遞增)
  - 1 1 3 4 5 (有相同值)
  - 3 3 3 4 5
  - 1 5 2 4 3 (上下震盪)
  - .....
- 達到 statement coverage 很簡單，但是從 partition tests 來看這卻是個相對複雜的問題

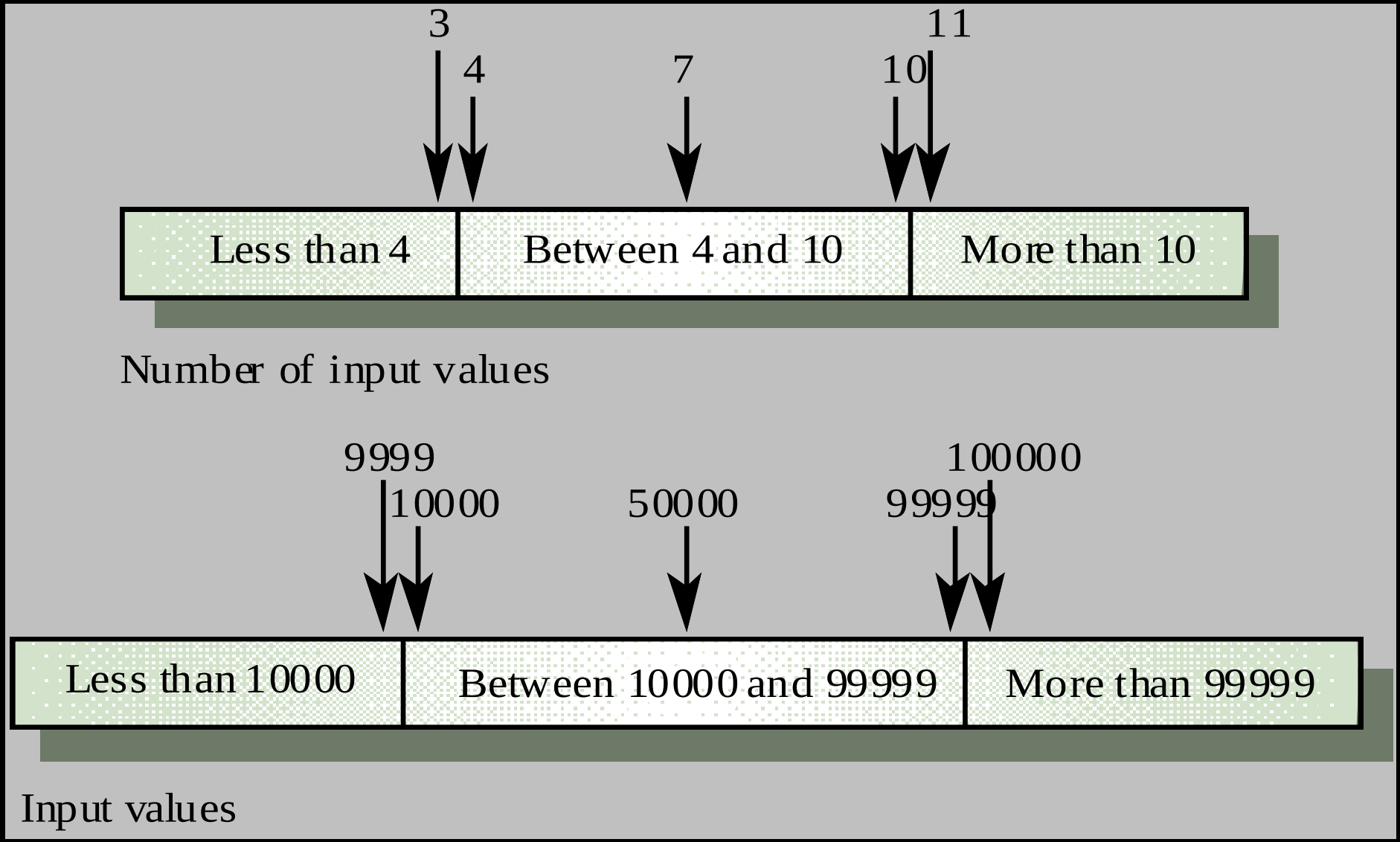
```java
public static int higher(int x, int y) {
 if (x>y) {
     return x ;
 } else {
     return y ;
 }
}
```

```java
public static void asscendingBubbleSort(int[] inputarray) {
    Boolean change ;
    int temp ;
    change = false ;
    do {
      change = false ;
      for (int i=0; i< inputarray.length-1; i++) {
          if (inputarray[i] > inputarray[i+1]) {
              temp = inputarray[i] ;
              inputarray[i] = inputarray[i+1] ;
              inputarray[i+1] = temp ;
              change = true ;
          }
      }
    } while (change);
    // return inputarray ;
```

How to compute partitions in xUnit tests?

- White box testing
- 基本上你可以從 source code 的 if-then-else 做個 macro partition，通常這麼做你也完成了大架構的 statement code coverage
- 如果有 loop 那就得從主控制 loop 的變數與記憶體著手 (e.g., sorting problems)，從那一邊你大概能得到 partitions 的基本訊息。基本上同一個 partition 的 test data，都會讓你的程式碼走過一樣的 path (這其實是種本能，如果你過去coding 經驗已經昇華到了某個境界)
- 每一個 partition, 挑一個測試案例

Equivalence partitions and boundary data selection



Boundary Tests

Boundary Testing

- 如果你的腦袋瓜先有 partition (equivalence) testing 的概念，那 boundary tests 已經在前一頁解釋了
- 軟體錯誤通常很容易在 boundary values 被暴露出來。 The density of defects at boundaries is more.
- 厲害的 QA 通常就是厲害在 Boundary value analysis
- 不要被 partition 的 boundary value 局限了你的想像。

Boundary Tests in Higher() - 你應該測什麼?

```
public static int higher(int x, int y) {
 if (x>y) {
     return x ;
 } else {
     return y ;
 }
}
```

- x=y
- x=0, x=最大值，最小值
- Y=0, y=最大值, 最小值

Boundary Tests in BubbleSort() - 你應該測什麼?

```java
12    public static void asscendingBubbleSort(int[] inputarray) {
13        Boolean change ;
14        int temp ;
15        change = false ;
16        do {
17            change = false ;
18            for (int i=0; i< inputarray.length-1; i++) {
19                if (inputarray[i] > inputarray[i+1]) {
20                    temp = inputarray[i] ;
21                    inputarray[i] = inputarray[i+1] ;
22                    inputarray[i+1] = temp ;
23                    change = true ;
24                }
25            }
26        } while (change);
27        // return inputarray ;
```

- 5 4 3 2 1 (絕對遞減)
- 1 2 3 4 5 (絕對遞增)
- 1 1 1 1 1 (同值)
- 9 1 1 1 1 (觀察9 能否從前頭移到尾) 這叫做行為上的 boundary case
- 9 9 9 9 1 (觀察1 能否從尾移到前頭)這叫做行為上的 boundary case
- 還有嗎?

Boundary Tests in studentCollection - 你應該測什麼

```java
3  public class testStudentCollection {
4      studentCollection obj = new studentCollection();
5      @Test
6      public void testAdd() {
7          System.out.println("The size of obj = "+obj.sizeOfStudent());
8          obj.add("Emma");
9          obj.add("Ronan");
10         obj.add("Antonio");
11         obj.add("Paul");
12         assertEquals("Adding 4 student to list", 4, obj.sizeOfStudent()).
13     }
14     @Test
15     public void testSize() {
16         System.out.println("The size of obj = "+obj.sizeOfStudent());
17         obj.add("Emma");
18         obj.add("Ronan");
19         obj.add("Antonio");
20         assertEquals("Checking size of List", 3, obj.sizeOfStudent());
21     }
```

- N 個 add + N 個 remove (測試 obj 為空的狀況)
- add(“Emma”)+add(“Emma”) 同名看看會不會掛掉
- 999999*add 測試 studentCollection 的容量上限
- obj 為空的狀況+ N 個 remove()
- Add(“xxxx.....”) 字串的長度超出

如果你不太有想像力，最簡單的方法就是把 source code 挖出來，看看程式碼的所有 magic number 然後把 unit tests 給他寫上去

Negative Tests (給不合法，超出範圍的參數值，看看你的程式碼會不會掛掉)

Negative Tests in studentCollection - 你應該測什麼

```java
3  public class testStudentCollection {
4      studentCollection obj = new studentCollection();
5      @Test
6      public void testAdd() {
7          System.out.println("The size of obj = "+obj.sizeOfStudent());
8          obj.add("Emma");
9          obj.add("Ronan");
10         obj.add("Antonio");
11         obj.add("Paul");
12         assertEquals("Adding 4 student to list", 4, obj.sizeOfStudent());
13     }
14     @Test
15     public void testSize() {
16         System.out.println("The size of obj = "+obj.sizeOfStudent());
17         obj.add("Emma");
18         obj.add("Ronan");
19         obj.add("Antonio");
20         assertEquals("Checking size of List", 3, obj.sizeOfStudent());
21     }
```

- Add(x):
  - X 的字串超出定義的長度
  - X 的字串有不符合規定的跳脫字元
  - X 的字串有非 ASCII 的亂碼
- 不斷地 remove() 看看 obj 能否正常運作

What are advanced unit tests

Exception handling tests

Object-Oriented tests

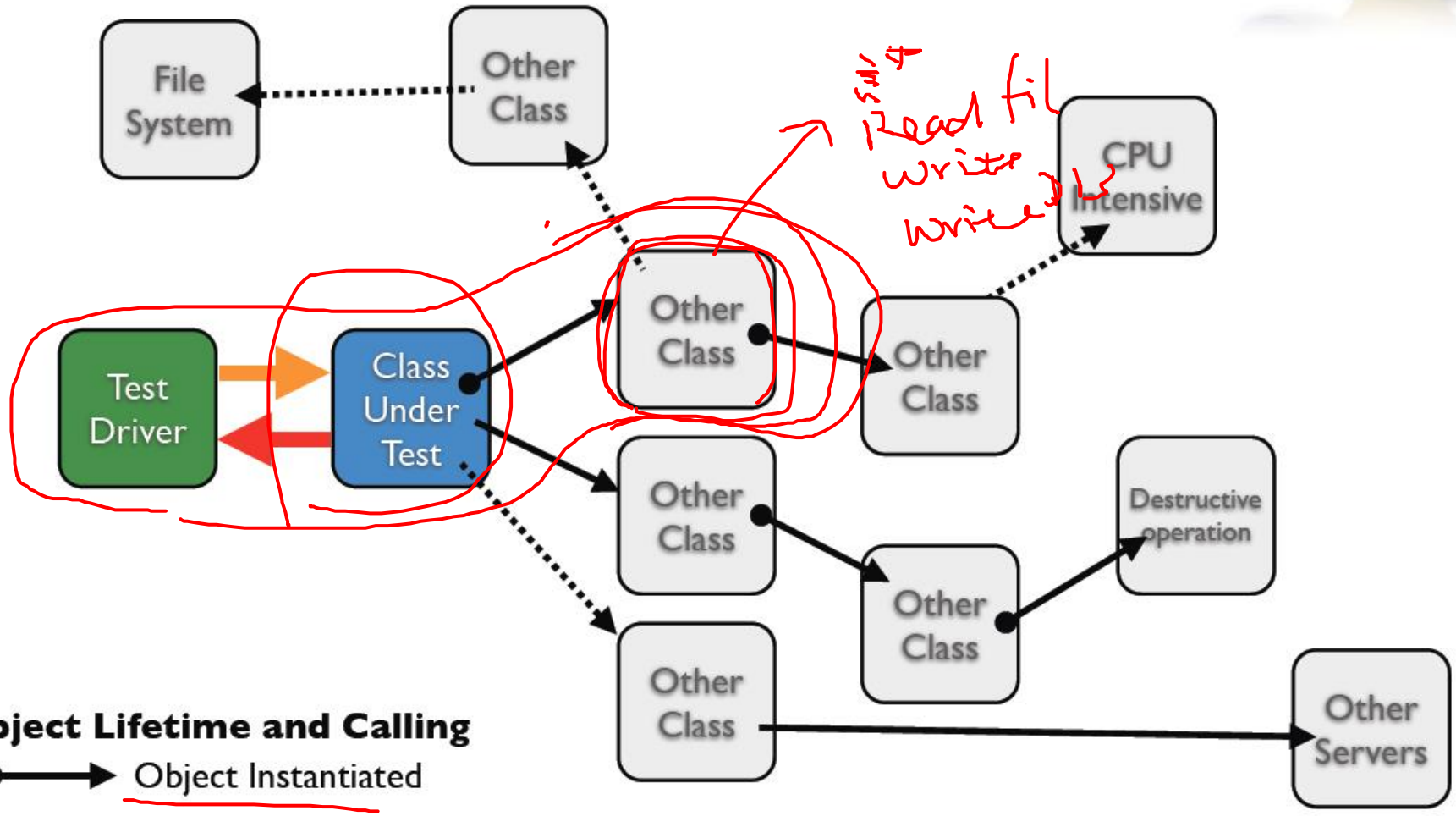
Coming SOON

不要寫 hard-to-test code
讓你的 test code 寫不出來

Coming SOON

Unit tests in isolation

Unit Testing a Class

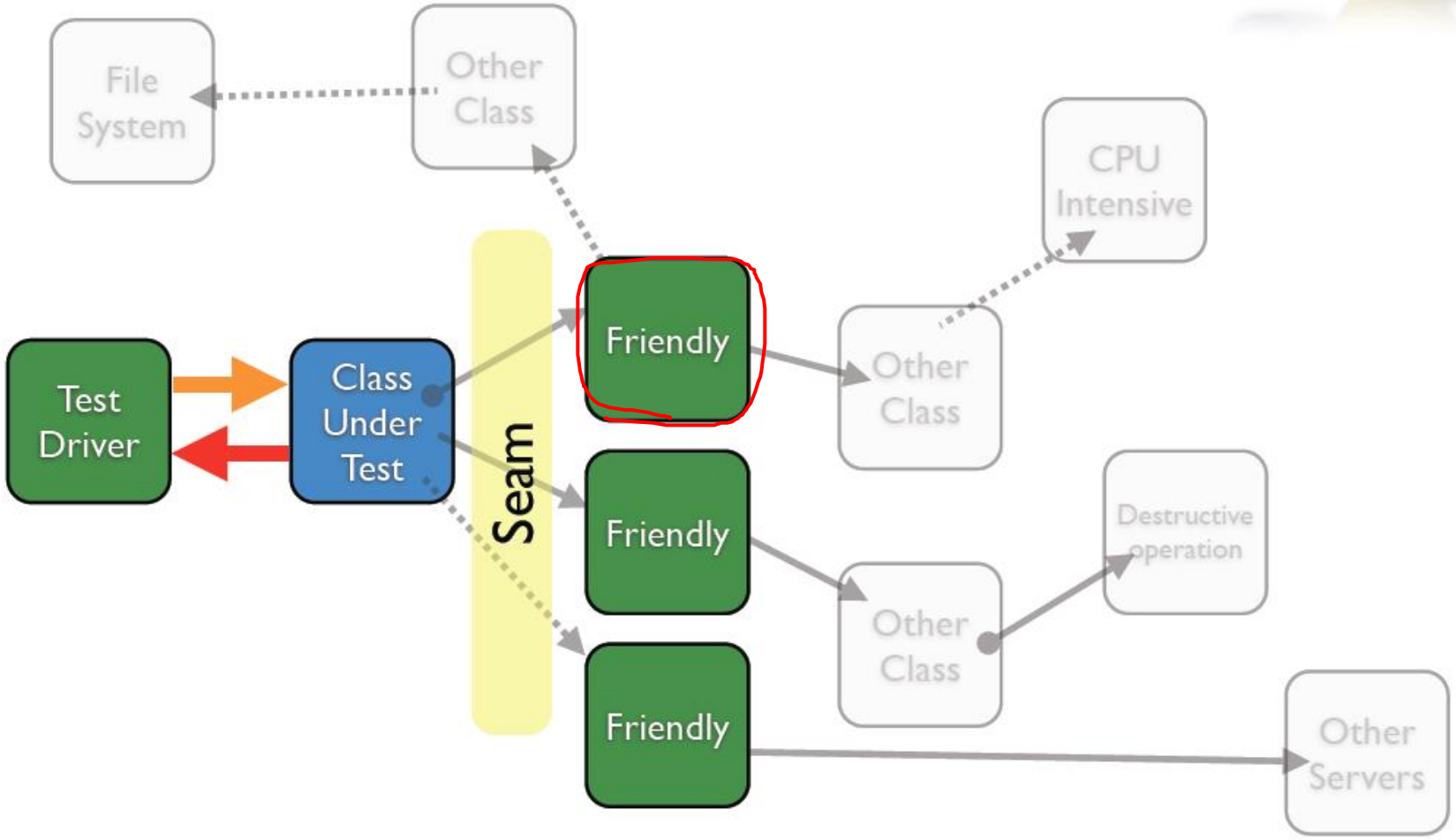


Object Lifetime and Calling

- Object Instantiated
- Object Passed In
- Global Object

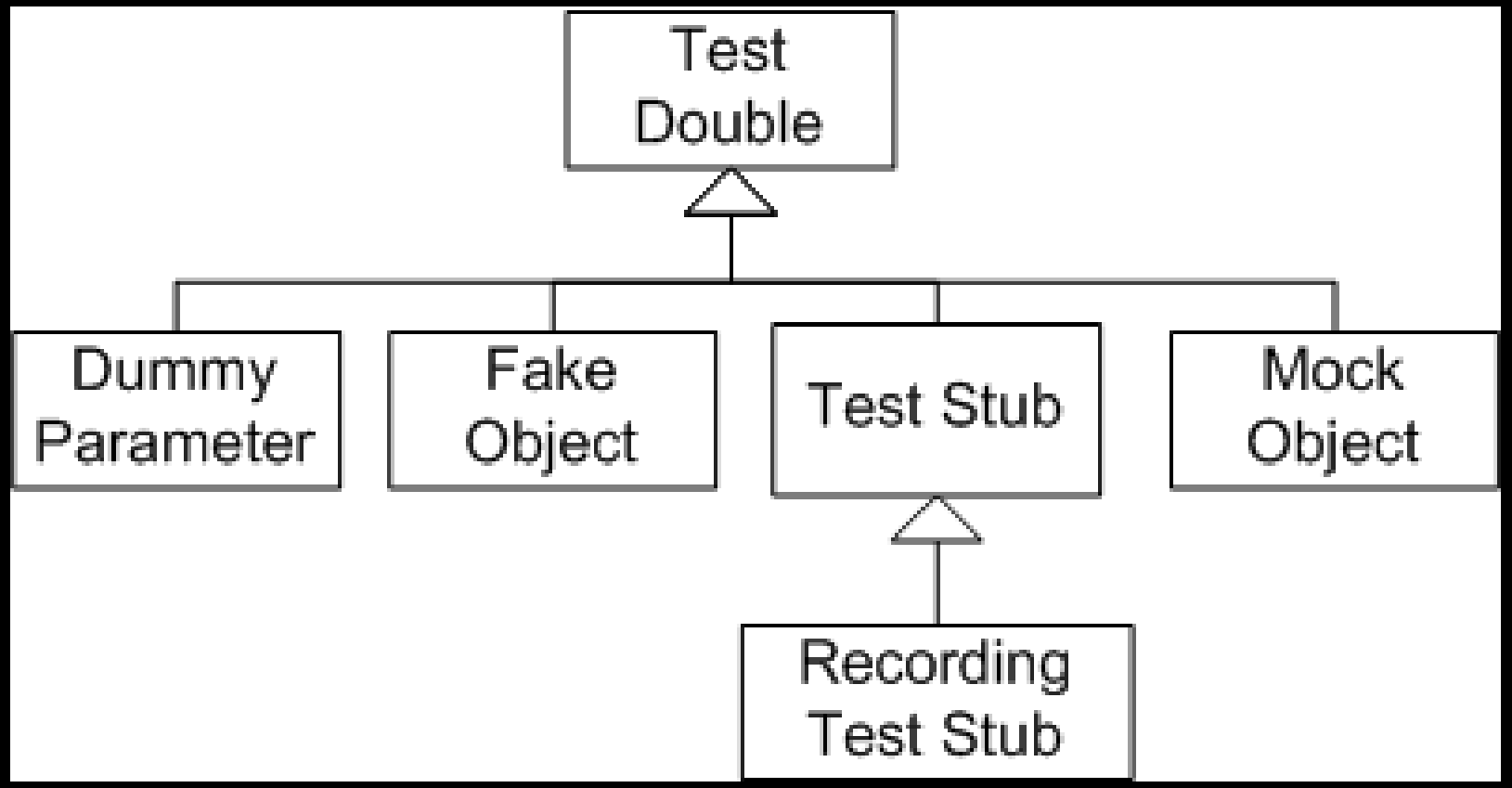
Google

Unit Testing a Class

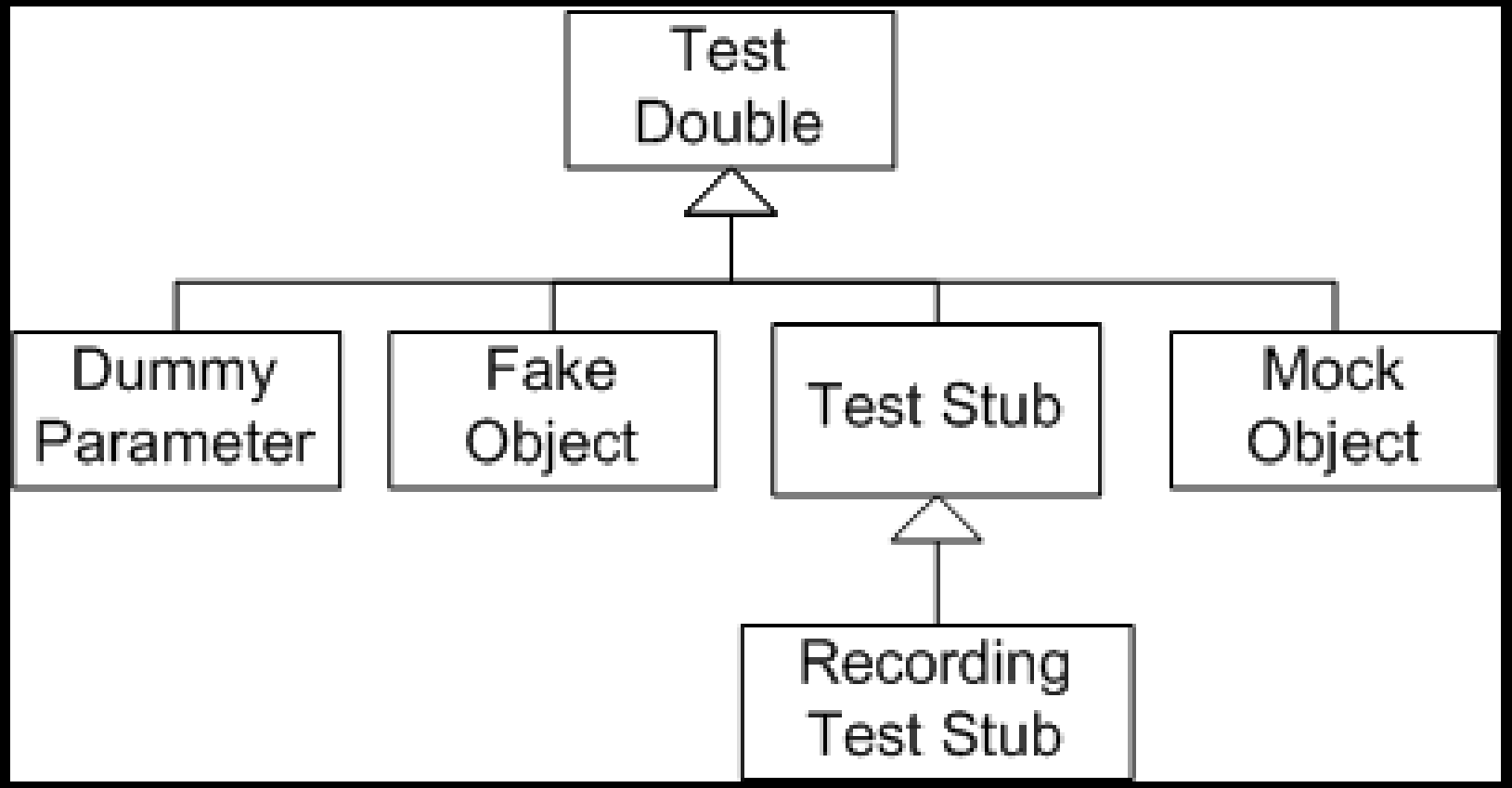


Google

Mocks and Other Test Doubles



Mocks and Other Test Doubles



Dummy Object (Parameters)

- Dummy is simple of all. It's a placeholder required to pass the unit test. Unit in the context (SUT) doesn't exercise this placeholder. Dummy can be something as simple as passing 'null' or a void implementation with exceptions to ensure it's never leveraged.

```
[TestMethod]
public void PlayerRollDieWithMaxFaceValue()
{
var dummyBoard = new Mock<IBoard>();
var player = new Player(dummyBoard.Object, new Die() ); //null too would have been just fine
player.RollDie();
Assert.AreEqual(6, player.UnitsToMove);
}
```

Fake Objects

- **Fake** is used to simplify a dependency so that unit test can pass easily. There is very thin line between Fake and Stub which is best described here as – "a Test Stub acts as a control point to inject indirect inputs into the SUT the Fake Object does not. It merely provides a way for the interactions to occur in a self-consistent manner.

```
public interface IProductRepository
{
void AddProduct(IProduct product);
IProduct GetProduct(int productId);
}

public class FakeProductRepository : IProductRepository
{
List<IProduct>
_products = new List<IProduct>();
public void AddProduct(IProduct product)
{
//...
}
```

```
public IProduct GetProduct(int productId)
{
//...
}
}

[TestMethod]
public void BillingManagerCalcuateTax()
{
var fakeProductRepository = new FakeProductRepository();
BillingManager billingManager = new BillingManager(fakeProductRepository);
//...
}
```

Test Stub

- **Stub** is used to provide indirect inputs to the SUT coming from its collaborators / dependencies. These inputs could be in form of objects, exceptions or primitive values. Unlike Fake, stubs are exercised by SUT.

Listing 3.4 Injecting our stub using constructor injection

```
public class LogAnalyzer
    {
        private IExtensionManager manager;

        public LogAnalyzer ()
        {
            manager = new FileExtensionManager();
        }
        public LogAnalyzer(IExtensionManager mgr)
        {
            manager = mgr;
        }

MUT     public bool IsValidLogFileName(string fileName)
        {
            return manager.IsValid(fileName);
        }
    }
```

Defines production code

Creates object in production code

Defines constructor that can be called by tests

constructor Inject

Mock

- **Mock** – Like Indirect Inputs that flow back to SUT from its collaborators, there are also Indirect Outputs. Indirect outputs are tricky to test as they don't return to SUT and are encapsulated by collaborator. Hence it becomes quite difficult to assert on them from a SUT standpoint. This is where behavior verification kicks in. Using behavior verification we can set expectations for SUT to exhibit the right behavior during its interactions with collaborators.
- Classic example of this is logging. When a SUT invokes logger it might quite difficult for us to assert on the actual log store (file, database, etc.). But what we can do is assert that logger is invoked by SUT.
- The assertion cannot be done by asserting return values of a method.

```
[TestMethod]
public void ModuleThrowExceptionInvokesLogger()
{
var mock = new Mock<ILogger>();
Module module = new Module();
ILogger logger = mock.Object;
module.SetLogger(logger);
module.ThrowException("Catch me if you can");
mock.Verify( m => m.Log( "Catch me if you can" ) );
}
```

You want an SUT called Module successfully send an exception string to a logger

Skip

Creation of a SUT which will send exception string to a logger

Create a fake logger for testing only

Set the fake logger for SUT

Invoke the method under test

Assert whether the logger receives the same message

Test Doubles

- A Test Double is any object or component that we install in place of the real component specifically so that we can run a test. Depending on the reason for why we are using it, it can behave in one of four basic ways:

- A Dummy Object is a placeholder object that is passed to the SUT as an argument but is never actually used.

- A Test Stub is an object that is used by a test to replace a real component on which the SUT depends so that the test can control the indirect inputs of the SUT. This allows the test to force the SUT down paths it might not otherwise exercise. A more capable version of a Test Stub, the Recording Test Stub can be used to verify the indirect outputs of the SUT by giving the test a way to inspect them after exercising the SUT.

- A Mock Object is an object that is used by a test to replace a real component on which the SUT depends so that the test can verify its indirect outputs.

- A Fake Object is an object that replaces the functionality of the real depended-on component with an alternate implementation of the same functionality.

MOCKS!!!! Some Terms

- Mock Object - an object that pretend to be another object, and allows to set expectations on its interactions with another object.
- Interaction Based Testing - you specify certain sequence of interactions between objects, initiate an action, and then verify that the sequence of interactions happened as you specified it.
- State Based Testing - you initiate an action, and then check for the expected results (return value, property, created object, etc).
- Expectation - general name for validation that a particular method call is the expected one.
- Record & Replay model - a model that allows for recording actions on a mock object, and then replaying and verifying them. All mocking frameworks uses this model. Some (NMock, TypeMock.Net, NMock2) uses it implicitly and some (EasyMock.Net, Rhino Mocks) uses it explicitly.