Bug 追蹤系統及程式除錯技術
Bug Tracking System and Debugging Techniques

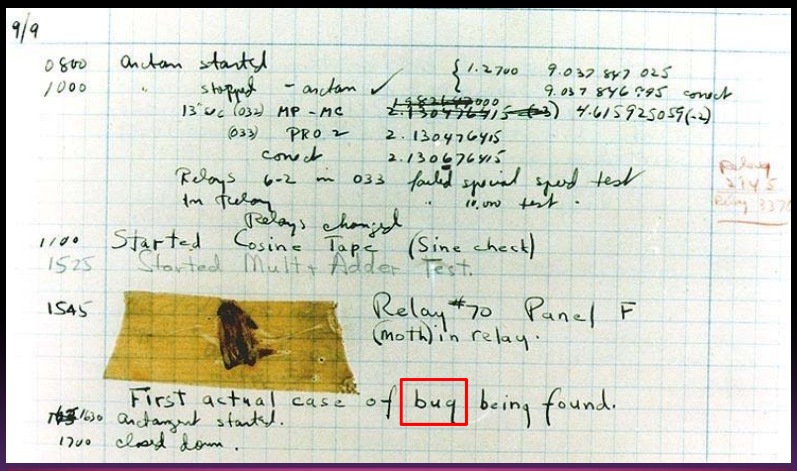
編輯委員
國立中央大學 資訊工程系 鄭永斌 教授

PART I - Knowing Bugs

什麼是BUG?

A **software bug**（臭蟲）is the common term used to describe an error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways.

Why is it called bug?



1946 Grace Murray Hopper（電腦科學的女性先驅，發明 COBOL 語言）於Mark III 的真空管電腦發現一隻飛蛾導致軟體錯誤所寫下的手稿珍藏於National Museum of American History

Testing vs. Debugging

- 兩種技術常常被混為一談
- 軟體測試(software testing) 是
  - 透過各種測試的技術與工具發現BUG
  - 通常由QA (Tester)人員負責
  - 使用測試輔助工具
- 軟體除錯(debugging)
  - BUG 發現之後，找出問題的源頭，修正程式的行為，使錯誤能在未來的測試的過程中不再出現
  - 由程式開發人員負責
  - 使用除錯工具

Bug 無所不在!

老闆：我們的目標是創造沒有bug的軟體。從今以後，任何員工只要找到而且修好一個bug，我就發給他10美元獎金。

Bug 無所不在!

員工：耶！我們發了，讚！讚！讚！

7

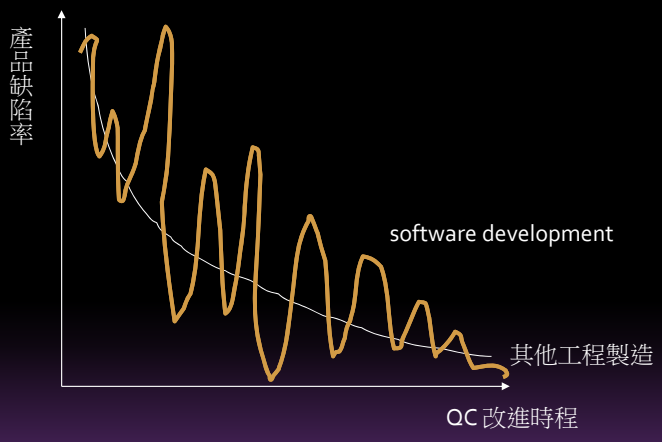
Bug 無所不在!

老闆：我希望獎金制度能推動員工們正確的行為。

員工：今天下午，我就要為我自己寫出一台全新的休旅車。

8

The QC story



9

Software Quality

- No humans (programmers) are perfect
- 由於軟體產業的生產工具是人，人注定會犯錯。如果還是很多人一起合作，還會犯更多的錯!
- 如果你想要建造真的能給人使用的軟體 – 軟體測試，在實務界非常重要

10

How programmers create bugs easily

Linux Device Driver bugs

[Dingo: Taming device drivers, 2009]

Driver	#loc	#bugs
USB		
RTL8150 USB-to-Ethernet adapter	827	16
EL1210a USB-to-Ethernet adapter	710	2
KL5kusb101 USB-to-Ethernet apapter	925	15
Generic USB network driver	1028	45
USB hub	2234	67
USB-to-serial converter	989	50
USB mass storage	803	23
Firewire		
IEEE1394 Ethernet controller	1413	22
SBP-2 transport protocol	1713	46
PCI		
Mellanox InfiniHost InfiniBand adapter	11718	123
BNX2 Ethernet adapter	5412	51
i810 frame buffer	2920	16
CMI8338 audio	2660	22
		498

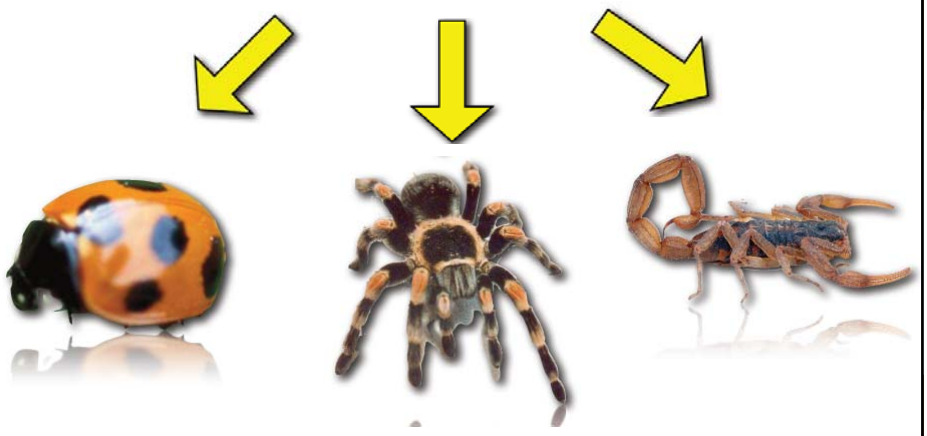
11

BUG是如何被製造出來的?

- 程式碼中的人為錯誤（例如：打字的錯誤）
- 設計與邏輯的錯誤
- 記憶體管理錯誤
- 規格錯誤
- 第三方軟體（程式庫)的錯誤
- 編譯器造成的錯誤（遠遠較為稀少）

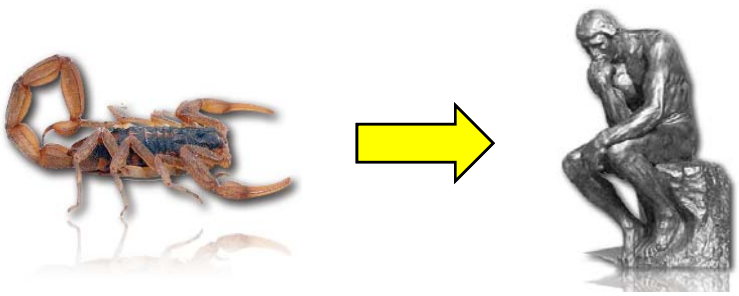
12

BUG的分類



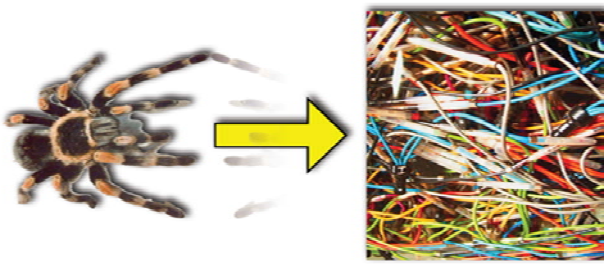
Logical Bug

- Logical Bug(邏輯錯誤導致的臭蟲)
  - 由於系統設計不週詳，資料結構設計不良，邏輯思考不夠周延所導致的臭蟲。
  - 通常修復的代價很高可能需要翻修許多程式或資料結構



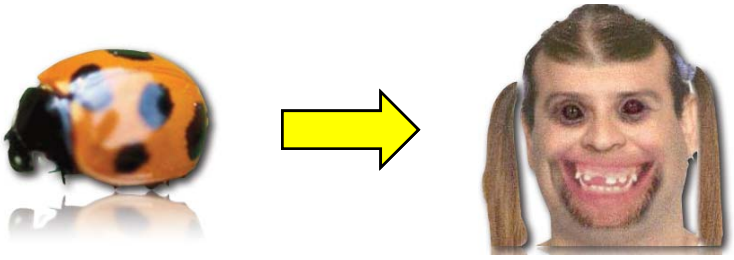
Wiring Bug

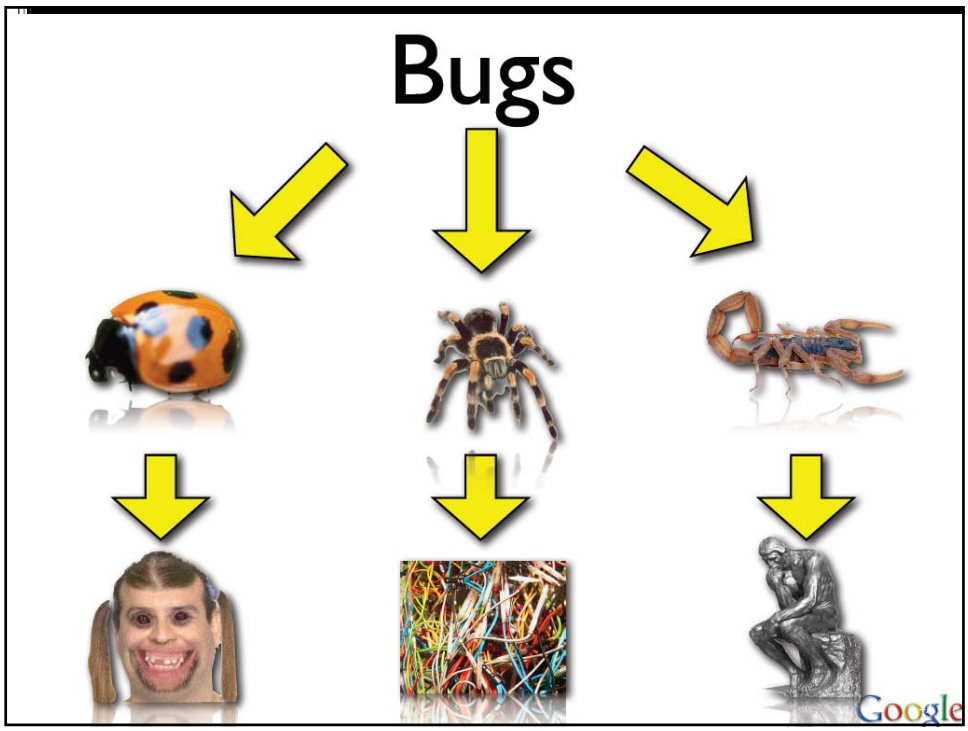
- Wiring Bug(線路錯亂導致的臭蟲)
  - 程式碼的**可能執行路徑太多**，未經過深思，以比較好的程式技巧或演算法來解決問題。
  - 修正此一類的錯誤需要費一點力氣，但是不需要大幅整修既有的程式碼，但是常常會越修越雜亂



Rendering Bug

- Rendering Bug(寫code失誤導致的臭蟲)
  - 一旦發現錯誤, 錯誤的原因很明顯，**白目或無嘸頭**，例如少了一個分號，或== 寫成 =





Cost of Fixing a bug

	Probability of Finding a bug	Difficulty of identifying the underlying reason of the bug	Difficulty of fixing the bug
Logical bug	High	Hard	Hard
Wiring bug	Medium	Medium	Medium
Rendering bug	Medium	Easy	Easy

18

Comments

- Probability of finding a bug
  - Low:可能執行了許多測試案例才會出現，但是不見得特別去設計測試案例來逼迫你的程式碼執行到某部份的程式碼
  - High:稍微執行一些測試案例就會發現
- difficulty to find the bugs –
  - Low: 不用太困難刁專的測試案例就可以發現
  - High:需要特別設計的測試案例才能讓你的程式行為能夠到達錯誤觸發的滿足條件


19

不尋常的Bug（Unusual Bug）

- 現代的軟硬體不斷地精進，引入了許多新的軟體技術。或因為時代的演進讓某些較複雜的技術普及到一般的程式應用
  - 多執行緒 Multithreading
  - 網路 Networking
  - GUI 與物件導向系統 –
    - 將系統部分執行緒的行為封裝與遮蔽起來
  - Signal/CallBack（Interrupt-Driven Like System）
- 上述的技術引入了許多不尋常的 bug


20

不尋常的Bug（Unusual Bug）

- Heisengbug
- Bohrbug
- Schrödinbug
- Mandelbug
- Statistical bug

不尋常的bug（Unusual bug）

- Heisengbug（海森堡臭蟲）
  - A **heisenbug** is a software bug that seems to disappear or alter its behavior when one attempts to study it
  - 源自海森堡測不準原理（Heisenberg Uncertainty Principle）
  - 很難重現 bug
  - 雖然可以重現，但是一旦觀察之後就消失
  - 程式行為在 debug mode, optimization mode, release mode 不一樣
- Examples
  - race condition
  - deadlock
  - memory not properly cleaned and initiated.

不尋常的bug（Unusual bug）

- Bohrbug
  - 特定條件成立才會發生的 bug，但是通常要讓條件成立不容易，一般測試很難涵蓋到
  - Bohrbug的行為與Heisenbug相反，也就是說，當處理這種bug時，這種bug並不會消失或改變特性
- Examples
  - An overflow bug


23

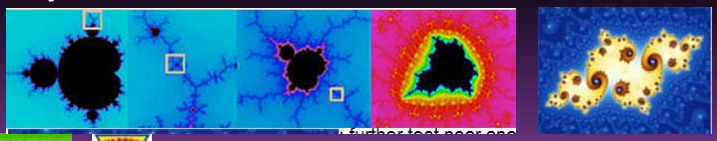
不尋常的bug（Unusual bug）

- Schrödinbug（薛丁格臭蟲）
  - 當讀過某段程式碼後，就立刻發現這段code存在著很容易被觸發的bug。而這種bug沒有被觸動，僅僅是運氣好而已。
  - 通常是程式人員以非正規，不正常的方式撰寫程式碼。這段程式碼只有在極端的情況下才會被執行，但是一旦執行起來則立即見光死
    - 註：這位程式設計人員根本不應該被聘僱
    - 註：你需要 code review 來防止
- Example
  - 副程式直接回傳一個預先計算好的數值，而沒有實質實作該做的運算


24

不尋常的bug（Unusual bug）

- Mandelbug
  - A bug whose underlying causes are so complex and obscure as to make its behavior appear chaotic or even non-deterministic.
  - An example of this type of bug would be a fundamental design flaw in an operating system.



25

不尋常的bug（Unusual bug）

- Statistical bug
  - 無法在單一一次執行重現的臭蟲
  - 必須要在不斷累積的執行當中才能展現的臭蟲
- Example
  - 程式牽涉到機率與亂數的運算

26

A Joke

有天晚上我參加一個老同學的喜宴，
同桌的人有律師、會計師，
只有我一個是電腦業界的人。
為了引起話題，我說我羨慕其他行業的人，像是律師、會計師等，在學校學的那一套終生都受用，每年就算有新法條也不會改太多。
不像我們電腦界的人，每天都在K新的技術手冊，新產品新技術隨時大翻新，改朝換代的速度讓人措手不及，所以學電腦的人看起來都比較蒼老，因為太辛苦了。

我話剛說完，一位會計師馬上回我一句：「你錯了，其實我們最羨慕你們電腦界的人，因為沒有任何行業的消費者像你們電腦界的消費者一樣好欺負。」

頓時我成了眾矢之的，
這些電腦使用者的抱怨全部集中到我身上來。
其中一位仁兄還舉了個有趣的例子，

他說：
「如果傢俱業和電腦業一樣，世界會變成什麼樣子？」以下是他講的故事：

如果傢俱業跟電腦業一樣，比如說我到傢俱店買了一張桌子，搬回家往地板上一放，啪啦一聲桌子就塌了。這時候我不會生氣、不會罵人，我會先自己檢查一下出了什麼錯。

我會先檢討自己，
是不是我做錯了什麼，
或是我對桌子的使用不夠熟悉。

於是我會去買書來看
(書名可能是《快快樂樂學修桌子》、
《21天學會修桌子》、《修桌子技巧與實例》、
或是《修桌子的聖經》)。
要是書看得不太懂，
我會再花錢去報名上修桌子的課程。
學完之後還是修不好，
我會請其他比較懂得修桌子的朋友來幫忙。
最後沒有辦法，
終於我打電話給原先的傢俱行，
(可能還要購買《技術支援方案》)，
結果他們跟我說：
「唉呀！你買到的是搶鮮版啦！
本來就應該有問題的」。

於是我恍然大悟原來是自己的錯，
我就再去買一張「正式版」的桌子。

回家一擺還是啪啦又塌了！

修了半天還是有問題，
再請傢俱行的技術人員來做仔細檢查，
最後終於發現問題的所在——
「我家的地板和桌子不相容」，
又是我自己的錯，
於是我得趕快幫家裡的地板升級.....。

等一切都忙完了，
桌子可以使用了，
我趴在桌上寫字，
心裡充滿了成就感，
我很得意地跟網友分享我修理桌子的經驗，
並暗自慶幸自己在科技的潮流上沒有落伍.....。

PART II - DEBUGGER

An Overview of Debugging

- 除錯(Debug)，是程式開發的必然過程。據研究，程式人員花在除錯上的時間約佔1/3的工作時間。
- 正規課程不太教授的內容
- 教師認為學生程式寫多了就應該會自行學會的技術(?)
- 除錯的困難度與系統的複雜度成正比
- 除錯的困難度會與使用的 programming language 正相關 - e.g., 高階語言除錯比較容易

軟體工程諺語

問題：你同意除錯很困難嗎？甚至比寫code還難？

- Q32: 除錯的困難度幾乎是寫作該程式碼的兩倍。因此，如果你竭盡才智寫作一段程式，那麼依據定義，你不夠聰明到可以解決這段程式的錯誤。

  B.W. Kernighan

- Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

  B.W. Kernighan.


How do you fix a bug?

- 選擇題：
  1. Read the source code and identify the bug
  2. Step by step check if each step has any problems (using print statements or a debugger)
  3. Divide and Conquer (divide the program into smaller ones and determine the one with problem)
  4. Request a friend to fix the bug for you


How do you fix a bug?

- Read the source code and identify the bug
  - Inefficient
  - Sometimes works because the bug is so 白目
- Step by step check if each step has any problems
  - Using steps in a debugger and check variables in each break point
  - Doable but time consuming
- Divide and Conquer
  - Check middle point results (set break points)
  - If the results are correct, the chance that the bug is hidden in the second half of the program is much higher
  - Keep narrowing the search by setting break points


33

Typical Debugging Process

1. **重現** bug (can be a non-trivial task in many applications)
2. 視狀況**簡化**測試案例與問題
   - 例子1: 程式讀取很大的資料檔，在過程中發生錯誤。想辦法找到簡化的資料檔，還是可以造成一樣的錯誤，除錯過程就可以比較輕鬆與有效率。
   - 例子2: 程式進行了許多步驟後造成錯誤，藉由省略某些步驟，試看看是否仍然有bug (Divide and Conquer)，如果一樣，就可以省略這些步驟。
   - 這個過程通常是由經驗累積的除錯智慧。

more…


34

Typical Debugging Process

3. **加速重現**bug的流程，以提升效率
   - 使用單元測試，直接呼叫有問題的副程式
   - 改程式，省略某些步驟
4. 使用**除錯器**(或列印變數)一步一步找出問題的源頭
   - 通常需要反覆執行很多次
   - 注意：Debugger也可以在單元測試的測試案例裡執行，協助除錯。

重現程式的臭蟲 (Reproduce the Bugs)

- 這件事情在許多情境下，可能不是一件直覺與簡單的事情, 尤其
  - Bug屬於 unusual bugs
  - Bug會隨機出現，不明原因
  - 難以重現或重建臭蟲發生的執行環境
  - 臭蟲發生於累積的資料，但是資料的內容無法取得（使用者須保密資料）

Tracing Techniques - PRINT

- PRINT variables
  - 會寫程式的人通常會使用的最基本的技巧
  - 在程式碼中插入 printf/cout 來列印關鍵位置的重要變數內容
  - 一般列印到螢幕或檔案
  - 除非插入等待的指令, 否則列印完畢不會停止
  - 讓程式碼在關鍵的地方產生log 也是屬於這類的技巧

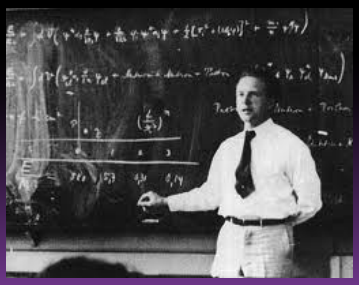

Tracing Techniques - PRINT

PRINT variable 在實務上有下列問題

- 若要更改觀察的內容(譬如要多印幾個變數)，需要修改程式碼。但是大系統的重新編譯過程可能非常久，為了DEBUG而重新編譯整個系統，有時不僅愚蠢而且沒有效率。
- 一旦列印的程式碼變得很複雜， PRINT本身可能成為需要被除錯的對象
- 可能使原始程式變得不易閱讀，DEBUG結束後，必須移除PRINT。若不移除，變成垃圾程式。
- 有時無法適用(例如GUI的應用可能無console可供輸出)
- 造成Probe Effect (next page)
  - 需要高效能的程式若加入PRINT 指令(尤其是輸出到螢幕或檔案), 會讓程式執行變得非常緩慢，或製造更多的問題


Probe effect（探測效應）

- 你為了瞭解程式的內部狀態，所以做了探測（例如加入PRINT指令，或使用Debugger）
- 探測本身其實也會改變系統的行為，此效應稱之為 probe effect
- 海森堡測不準原理



39

Probe Effect Example

問題：請猜測下面2個程式的執行時間？(使用i7 CPU/Windows 8.1 x64/Visual Studio 2013)

(a)

```
double value = 0;
for (int i = 0; i < 1000000; i++) {
   value += (double) i * i;
}
cout << "value = " << value << endl;
```

0 ms

(b)

```
double value = 0;
for (int i = 0; i < 1000000; i++) {
   value += (double) i * i;
   cout << i << " " << value << endl;
}
cout << "value = " << value << endl;
```

16 分

40

Probe Effect in Practice

- 若你待除錯的程式(debuggee)是只有一個執行緒的sequential program
  - 大部分只會影響效能，而不影響程式的對錯(學校課程作業屬於此大宗)，程式只是比較慢走到你想要的地方
  - 錯誤往往還是能不斷地重現


41

Probe effect

- 若程式是具有多執行緒，分散式網路應用，平行運算…
  - 真實世界的軟體通常屬於此類
  - 除錯過程所帶來的 probe effect (利用 print 或 debugger) 不只影響效能，還可能改變執行緒交互與同步的順序(interleaving)
  - 程式不見得走到你想要的地方
  - 錯誤不見得能 reproduce


42

Tracing Techniques - Debugger

- Debugger (System Program)
  - 讓你設中斷點，不用重新編譯整個系統
  - 在中斷點提供你檢查變數的功能
  - 提供 call stack 的資訊
  - 執行過程與 source code 對照
  - 可以一行一行的執行程式，並立即觀察結果
  - 增進除錯的效率
  - 如果你認為不需要用，那你寫的程式都不夠大與複雜


An Introduction to Debugger

- 一個重要的系統程式 (system program)
  - 任何開發平台如果要吸引程式開發人員為其寫程式，Debugger必須要提供的系統程式之一
  - 一般的平台所提供的系統程式包括 compiler, linker, debugger, assembler …
- debugger 本身是一個複雜的程式
- 其最原始的使用模式為 console-mode (command line)


Turn on Debugging Information

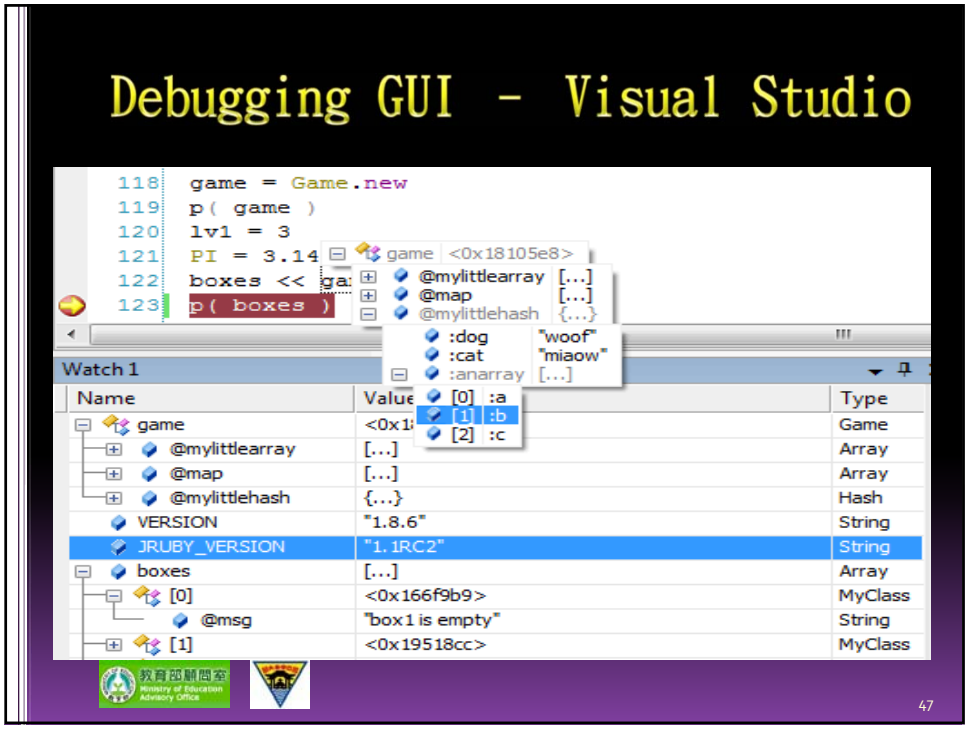
- Debugging information 會增加大約10-20%的 object file 大小
- 如果你不在除錯模式底下執行，一般而言不會減緩程式的執行速度
- 不過，Compiler 有個 optimization 的選項，一般在這個選項底下， debugging information 會失去作用

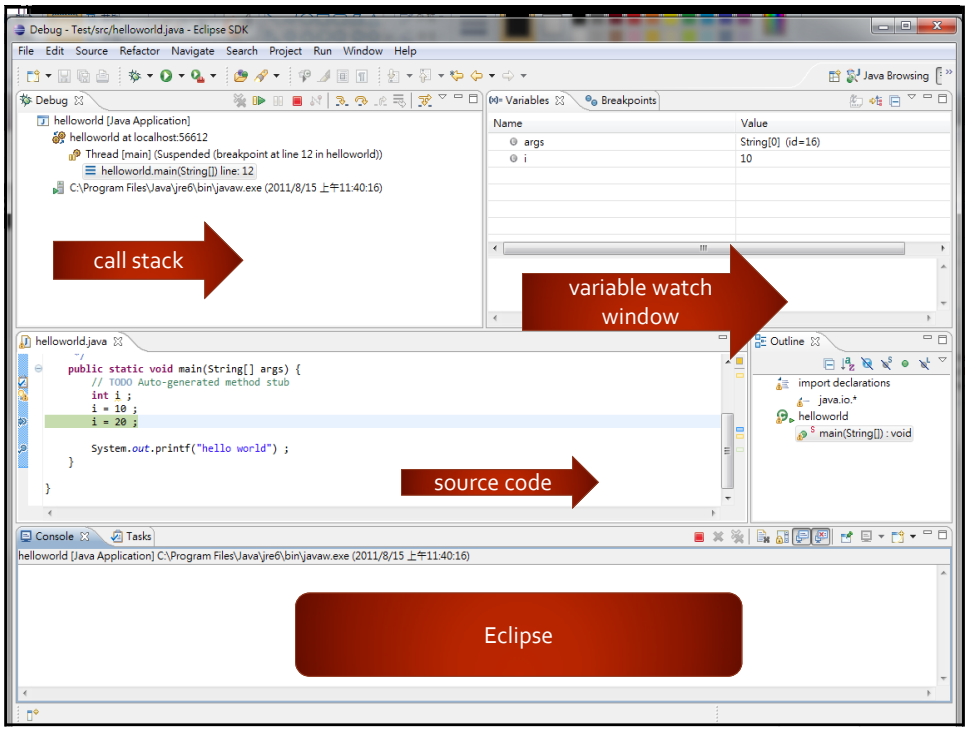

45

Debugger Wrapped by GUI

- 為了解決 console mode debugger 不容易使用的問題，目前整合開發環境都會將 debugger 與 compiler, editor 做整合，使得除錯過程能更友善
- 有時候稱之為 Visual Debugger


46


Debugging GUI - Visual Studio
118 game = Game.new
119 p(game)
120 lv1 = 3
121 PI = 3.14
122 boxes <<
123 p(boxes)
game <0x18105e8>
@mylittlearray [...]
@map [...]
@mylittlehash {...}
:dog "woof"
:cat "miaow"
:anarray [...]
[0] :a
[1] :b
[2] :c
Watch 1
Name
Value
Type
game
@mylittlearray [...] Array
@map [...] Array
@mylittlehash {...} Hash
VERSION "1.8.6" String
JRUBY_VERSION "1.1RC2" String
boxes [...] Array
[0] <0x166f9b9> MyClass
@msg "box1 is empty" String
[1] <0x19518cc> MyClass
47

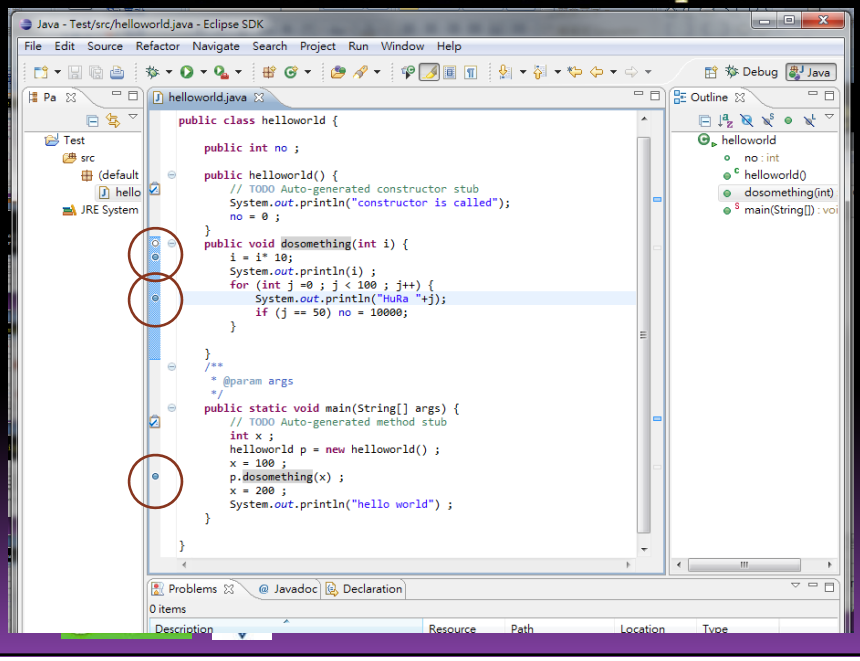

Debug - Test/src/helloworld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
helloworld [Java Application]
helloworld at localhost:56612
Thread [main] (Suspended (breakpoint at line 12 in helloworld))
helloworld.main(String[]) line: 12
C:\Program Files\Java\jre6\bin\javaw.exe (2011/8/15 上午11:40:16)
call stack
Variables
Breakpoints
Name
Value
args
String[0] (id=16)
i
10
variable watch window
helloworld.java
public static void main(String[] args) {
// TODO Auto-generated method stub
int i ;
i = 10 ;
i = 20 ;
System.out.printf("hello world") ;
}
}
source code
Outline
import declarations
java.io.*
helloworld
main(String[]) : void
Console
Tasks
helloworld [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (2011/8/15 上午11:40:16)
Eclipse

Lab 1 – Let's play with Java debugger under Eclipse

- 請開啟 Eclipse Classic 3.7
- New a project called test
- Enter a helloworld.java as follows (可以在trac/bugreport 的首頁找到這個程式碼)

49

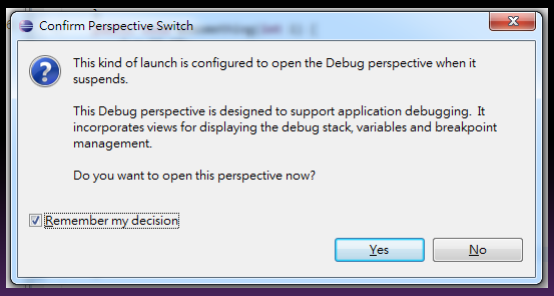
Enter a hello world example



50

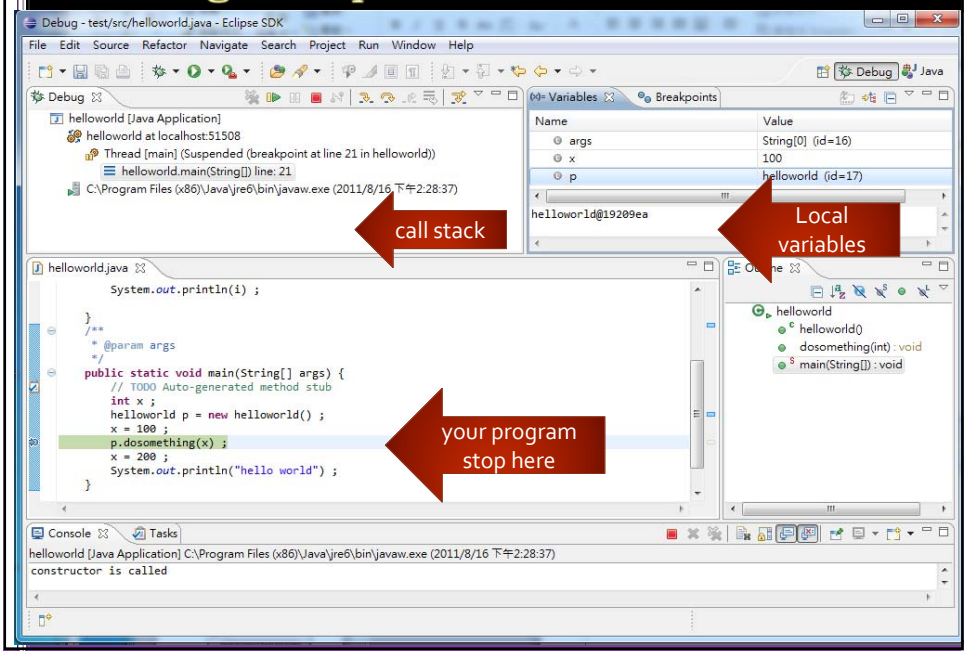
Let's GO

- Set break points
- Debug the program (F11)



51

A Debug Perspective



Key Bindings

Table 1. Debugging Key bindings

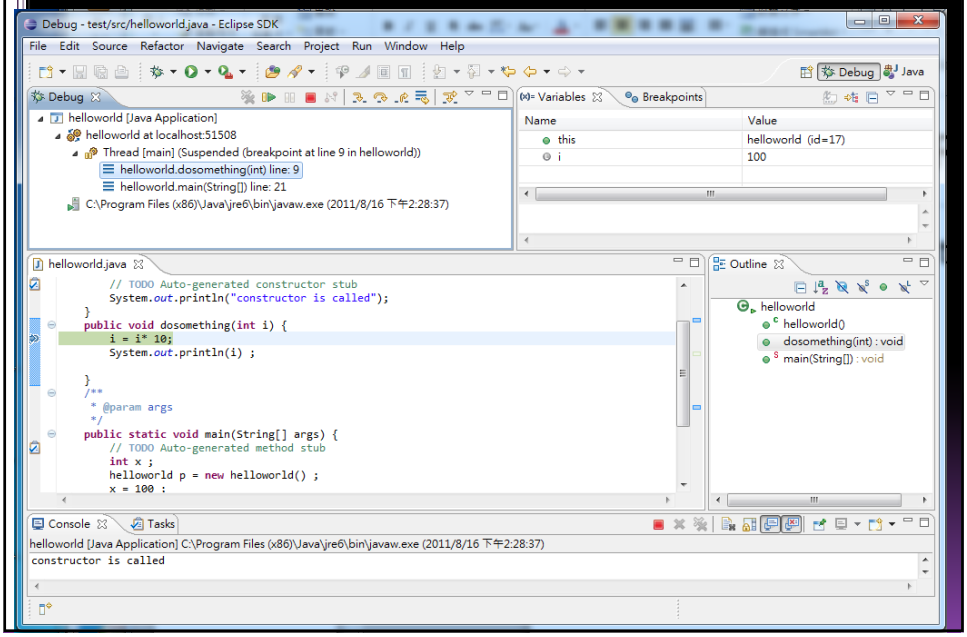
Command	Description
F5	Goes to the next step in your program. If the next step is a method / function this command will jump into the associated code.
F6	F6 will step over the call, e.g. it will call a method / function without entering the associated code.
F7	F7 will go to the caller of the method/ function. So this will leave the current code and go to the calling code.
F8	Use F8 to go to the next breakpoint. If no further breakpoint is encountered then the program will normally run.

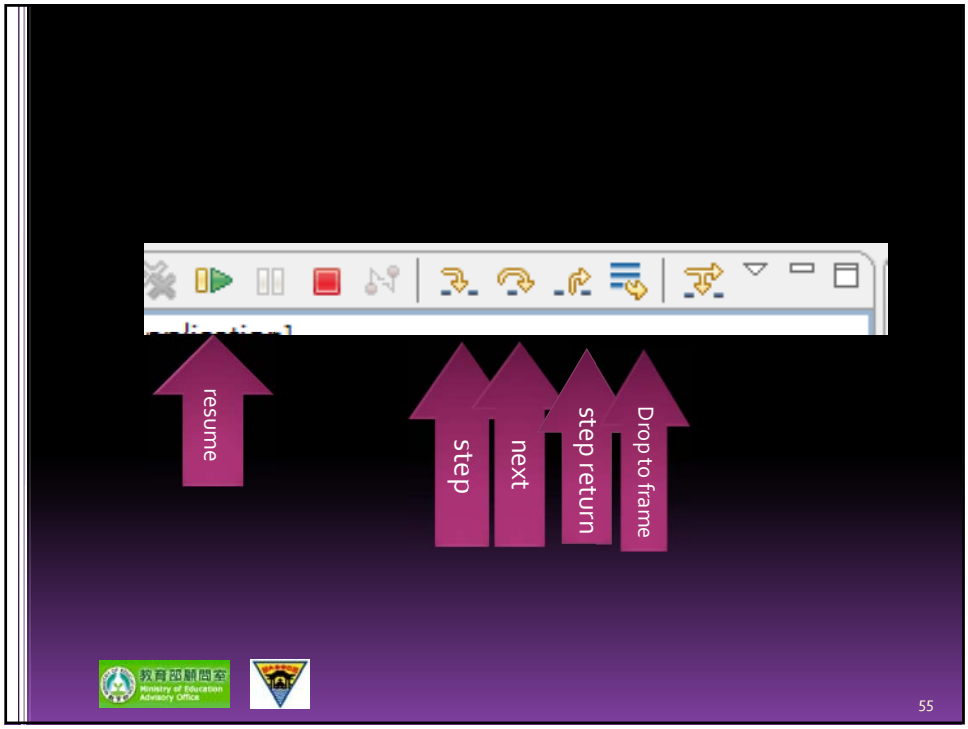
You can of course use the ui to debug. The following displays the keybindings for the debug buttons.

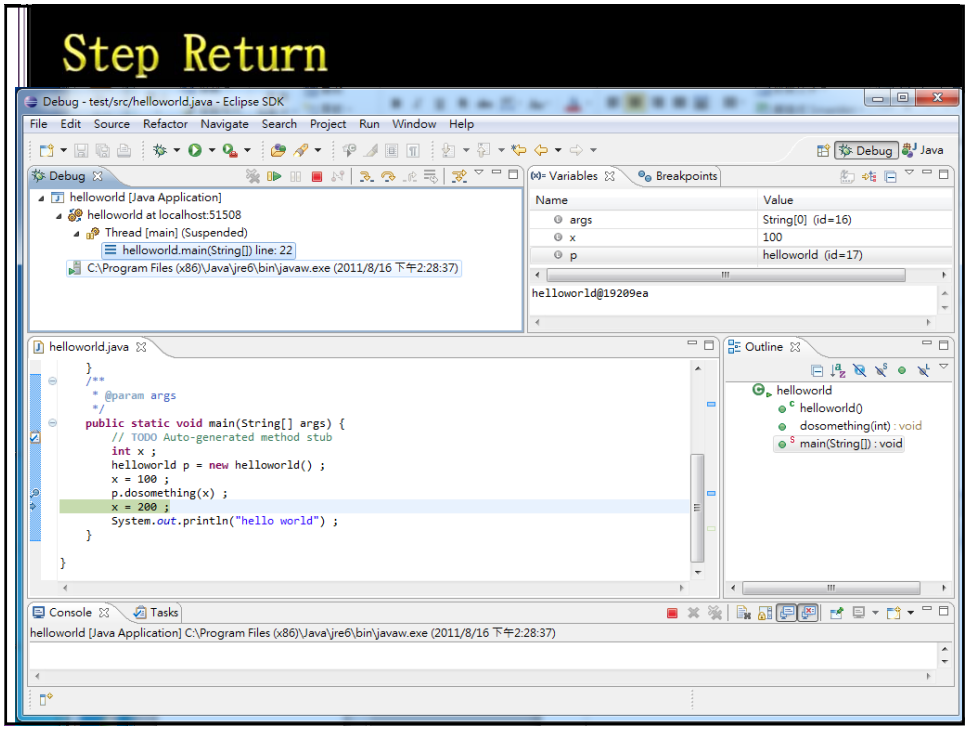
F8 Stop F5 F6 F7

53

Click resume to hit the next break point




resume
step
next
step return
Drop to frame
55


Step Return
Debug - test/src/helloworld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
helloworld [Java Application]
helloworld at localhost:51508
Thread [main] (Suspended)
helloworld.main(String[]) line: 22
C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2011/8/16 下午2:28:37)
Variables
Breakpoints
Name
Value
args
String[0] (id=16)
x
100
p
helloworld (id=17)
helloworld@19209ea
helloworld.java
Outline
helloworld
helloworld()
dosomething(int) : void
main(String[]) : void
Console
Tasks
helloworld [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2011/8/16 下午2:28:37)

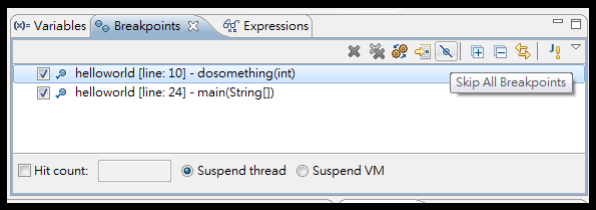
Drop to frame

- Drop to frame
  - Eclipse 特有的debugger 功能
  - 用途:你也許將某一個中斷點設在某一個 method 的中間，而且程式也來到這個中斷點。你想從method 的最前端重新開始看看怎麼樣到達這個中斷點
  - Drop to frame 可以讓你回到這個method 的最開始的指令
  - 某一種replay 的功能方便你除錯

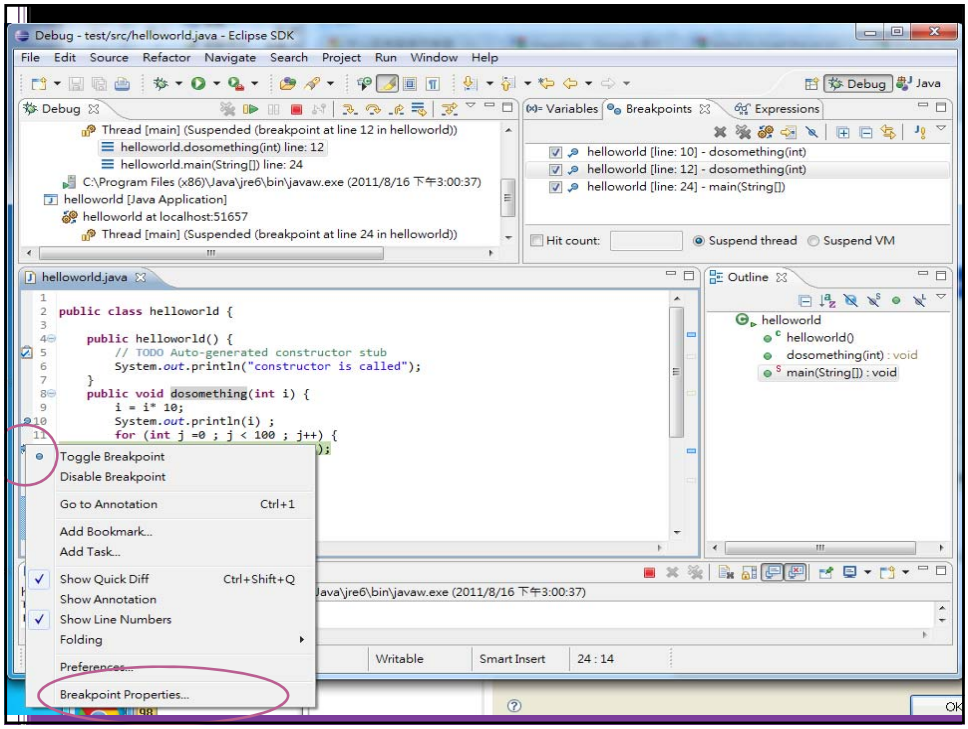
57

Skip All Breakpoints

- 如果你想暫時的取消 break points 的作用

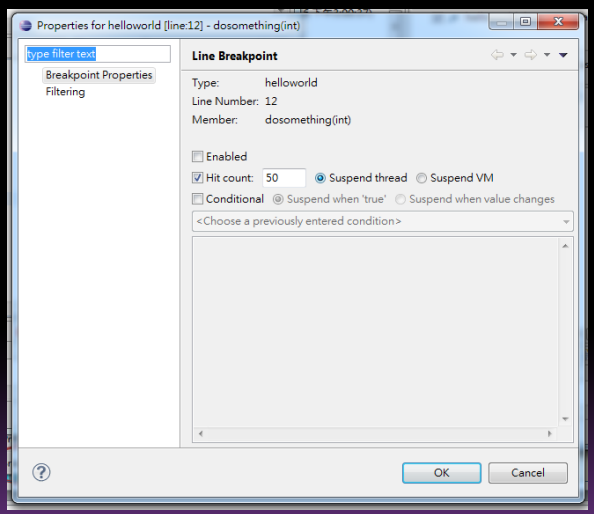


58



Hit Count

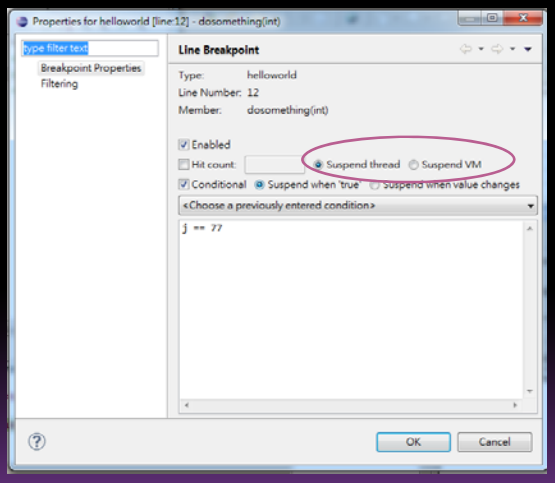
- Set the hit count to 50
- Resume the program
- Check the value of j in the for loop



60

Conditional Break Point

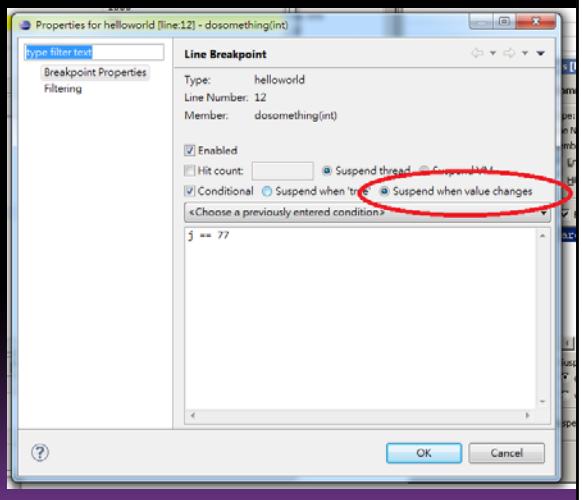
- Enter a Boolean expression
- The breakpoint will be hit when the condition is true



61

Conditional Break Point

- A break point is hit when the condition evaluation changes.
- So, the break point will be hit at j=0, 77, 78

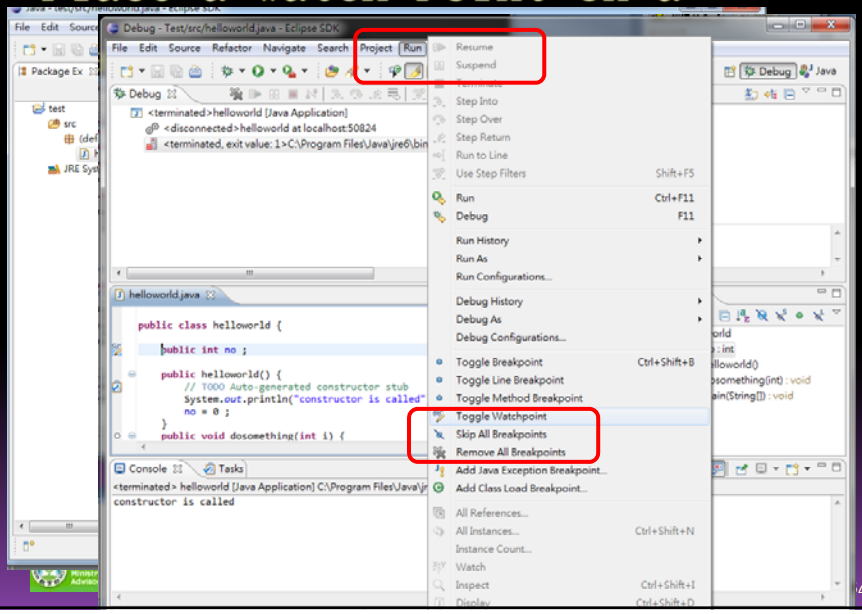


62

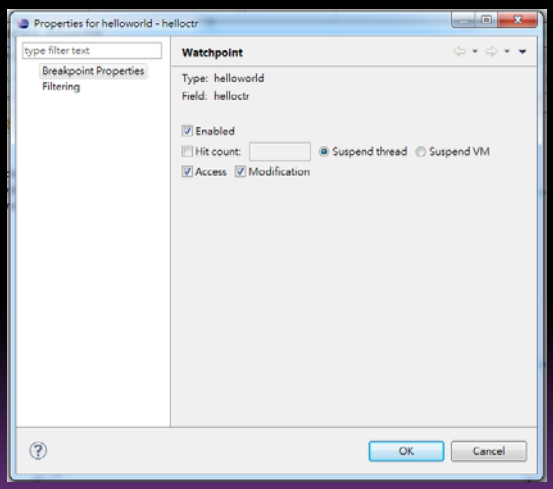
Watch Point

- 當你關心的某一樣變數的值被改變，程式會中斷
- 當系統複雜而且龐大，你不曉得到底是誰的程式或程式的哪一個部份將某個關鍵變數給改變了。所以你希望除錯器能夠停在變數被修改的那一行。
- 可以算是一種break point

Place a Watch Point on a



The Options of a Watch Point



Where the bugs are hidden?

- 當程式執行不正確時(不一定是 crash)，如何比較精準地猜測哪部分程式出了問題，然後開始在適當的地方插入中斷點?
- 當你的系統大又複雜，還牽涉到多人合作時又如何?
- 當你的程式沒有良好的模組化時，又如何?
- 通常程式發生錯誤的地方，只是結果，而不是因

Lab 2 - Debugging a Program using Debugger

- 請到 http://140.115.53.53/trac/bugreport
- 裡面有幾個簡單的Java 程式以及其功能的說明，但是這些程式都是錯的
- 請利用 Eclipse 的debugger 來找出錯誤


PART III - Defensive Programming


Why do we need defensive programming?

- In school, you mostly do your programming work alone.
- In real world, you work with others to complete a product
- The stupidest phenomenon
  - Commit source code
  - Build
  - Run
  - Crash
  - 大眼瞪小眼
  - Accuse each other
  - 拱出一人專責debugging
  - A great amount of time is spent until who is to blame is settled.

70

為甚麼debug很困難?

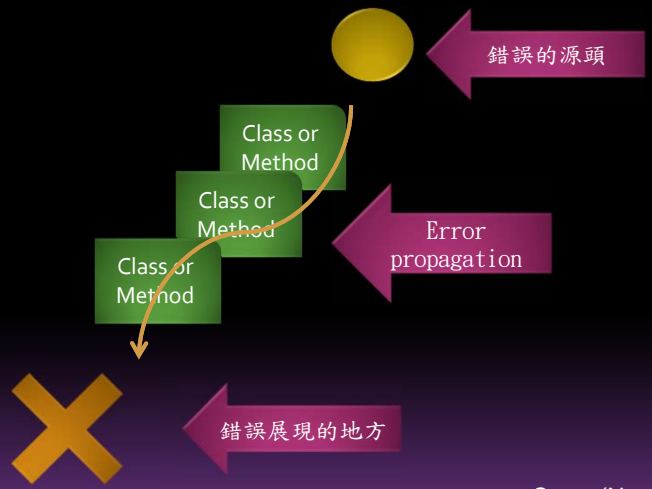
問題：如何節省修復以下的bug的時間？

1. 只知道有bug(例如crash)，而不知道原因在哪裡(哪裡導致crash)
2. 雖然知道bug的現象(例如NULL pointer導致crash)，但是，不知道為何出現這個現象
3. Bug的現象(表現)與bug的根源不同，找不到根源
4. 雖然知道出bug的原因，但是很難改，改來改去都不對，不知道怎麼改才對

- 1,2,3: 預防勝於治療，見下一頁
- 4: 改進程式邏輯或架構

71

Error propagation



So … (Next Page)

72

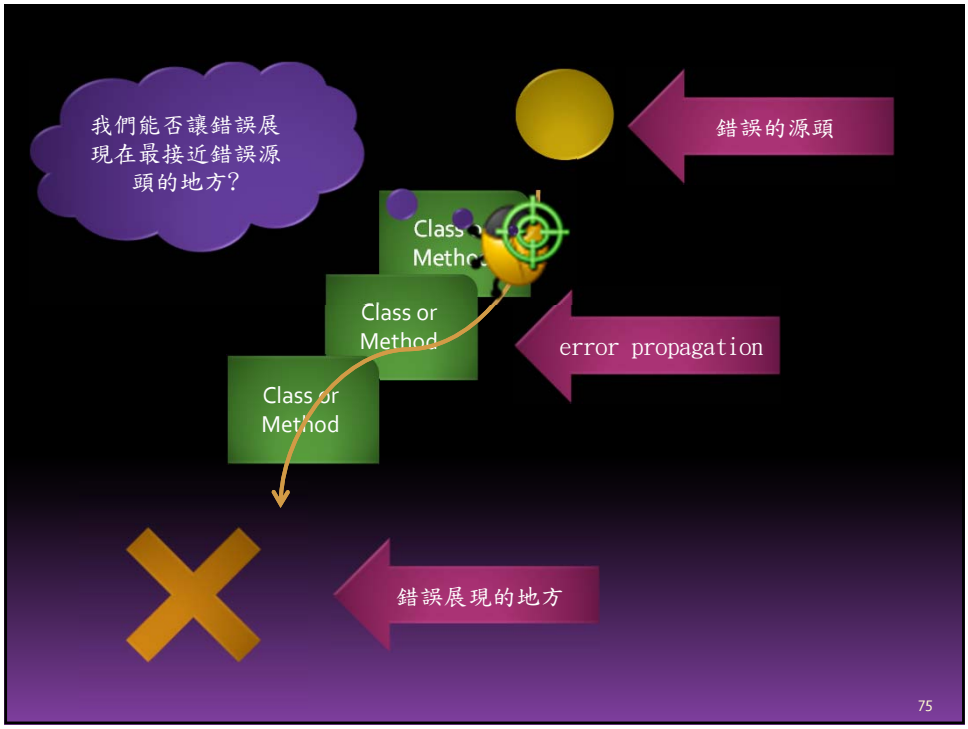
The Right Solution

- 如果你經常花很長的時間才能**找到**一個bug的**源頭**，這可能表示
  - 你的 **coding 習慣不良**，
  - 程式的模組性(或物件導向設計)太差所以**bug容易躲藏**
- 最好的除錯技巧是
  - **讓錯誤自己早一點跑出來**
  - But, how?


73

Best Debugging Strategy?

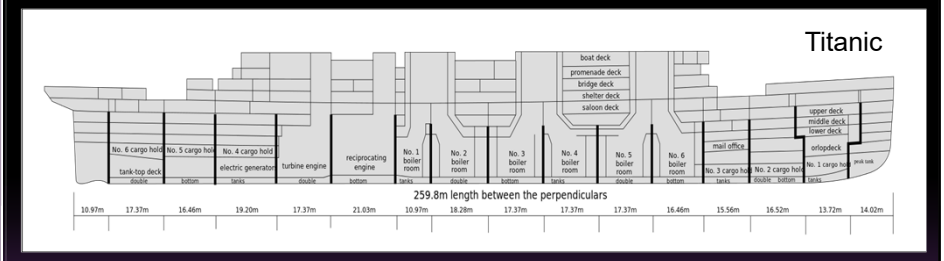
- Can you see the bug?



我們能否讓錯誤展現在最接近錯誤源頭的地方?
錯誤的源頭
Class or Method
Class or Method
error propagation
Class or Method
錯誤展現的地方
75


問題：太空船(或輪船)如何防止船體受損(爆炸、進水)時，擴散到其他部分?
隔火艙(門)、隔水艙(門)
76

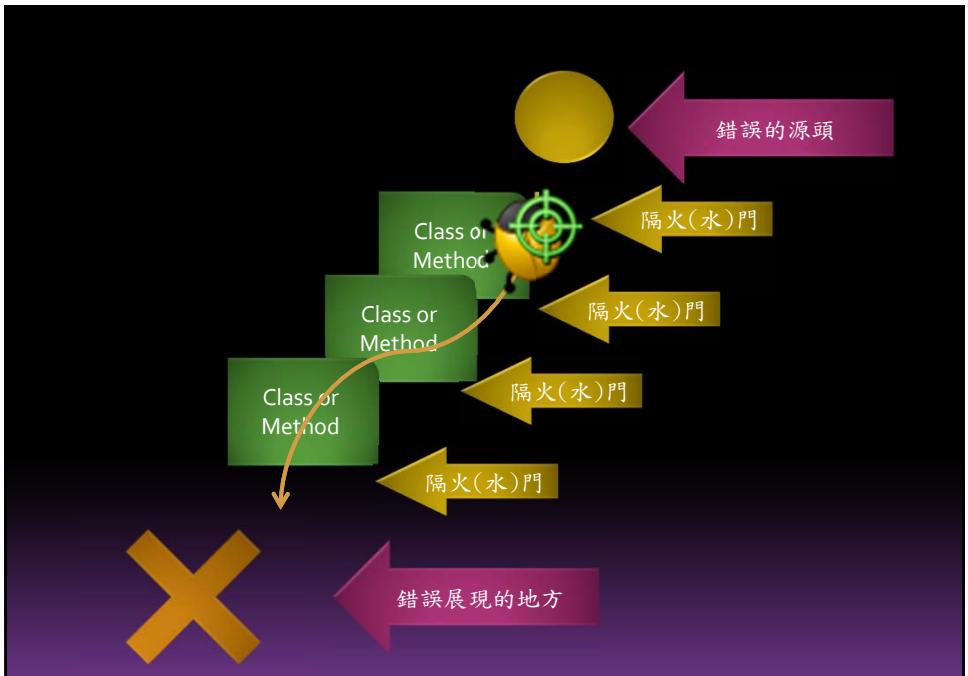
Watertight Door

- Titanic



Five of the ship's watertight compartments were breached (could not survive more than four compartments being flooded).

77



78

Watertight Door in Programming

- Inside production code
  - Assertion
  - Exception
- Outside production code
  - Unit Testing

79

Assertion

- Assertion
  - A condition that must be satisfied before continuing execution
  - Used during development – typically exists through alpha testing and beta testing
  - Removed in a released version
  - Useful in large, complicated programs and in creating high reliability programs

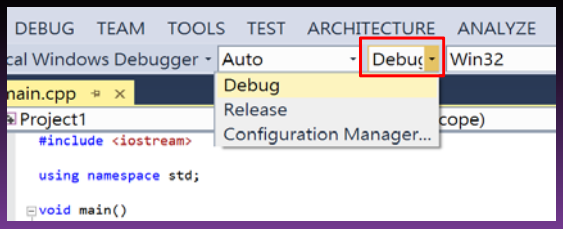
80

Assertion

- Assertion in C language

```
#include <assert.h>
...
void foo(int x) {
   ...
   assert(x > 0) ;
   ...
}
```

assert在debug mode會被執行，而在release mode會被省略



Assertion

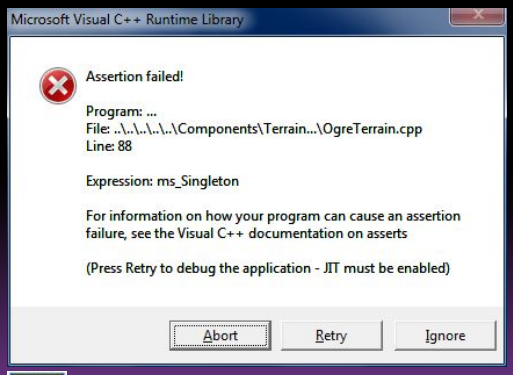
- Assertion in C# language

```
using System.Diagnostics;
...
public void foo(int x)
{
    Debug.Assert(x > 0);
    ...
}
```

assert在debug mode會被執行，而在release mode會被省略

Assertion

- 當assert failed時，IDE允許你觀察當時的變數，call stack等資訊，方便你來除錯




Assertion

- What to assert?
  - The precondition of executing a block of code

- 題目：寫一個程式，讓使用者輸入一個「檔名」，開啟檔案，從檔案讀入兩個整數，計算這兩個數目的平方和，最後把結果輸出至console。請問黃色部分的程式應該在那些地方作assert？


Assertion

- An assertion example in C

```
...
// C code: filename is a strin
assert(fileName != NULL);
// open the file
FILE *fp;
fp = fopen(fileName, "r");
// assert the file is successfully opened
assert(fp != NULL);
// read data from file
int data1, data2;
int n = fscanf(fp, "%d %d", &data1, &data2);
// assert two numbers are read
assert(n == 2);
...
```

fileName字串必須存在才能open檔案

如果不做assert呢？

讀到2個整數才能繼續下面的運算

85

Assertion

- The same example in C#
  - 會出現Exception的code，可省略Assertion
    (Exception本身就是隔水門)

```
// C# code: filename is a string
Debug.Assert(fileName != null);
// open the file
StreamReader file = new StreamReader(fileName);
string line = file.ReadLine();
string[] integerStrings =
         line.Split(new char[] { ' ', '\t', '\r', '\n' },
         StringSplitOptions.RemoveEmptyEntries);
// assert two strings are read
Debug.Assert(integerStrings.Length == 2);
int data1 = int.Parse(integerStrings[0]);
int data2 = int.Parse(integerStrings[1]);
...
```

assert

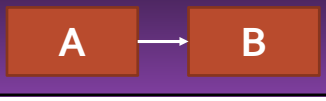
不用先assert

assert

不用先assert

Assertion

- 問題：到底是assert程式的precondition還是postcondition?
  - 一段程式A的postcondition是另一段程式B的precondition。
  - Assertion的目的是擔任「隔火(水)門」，避免在條件不對的情況下，盲目地進入B，所以是**B的precondition**。因此，assertion經常以precondition的形式存在。
  - 以效率而言，應該避免B之前驗證precondition，又重覆在A之後驗證postcondition。
  - 不限於precondition或postcondition，程式中有任何應該(或不應該)成立的條件，都可以寫成assertion。



Assertion Examples

- Function參數值的範圍落在合法的區間

```
void p(int dir, int velocity) {
        assert(dir >=0 && dir < 4 ) ;
        assert(velocity >= 0 && velocity < 10);
        ……
}
```

- 檔案已經成功開啟

```
void p(FILE *fp, string &msg) {
    assert(fp != NULL) ;
    assert(msg.getLength() >= 1 && msg.getLength() <= 255);
    …
}
```

Assertion Examples

- 某table中每個元素都有適當的值

```
void p(Object table[], int no) {
    assert(no != 0) ;
    for (int i = 0; i < no; i++) {
        assert(table[i] != NULL) ;
        assert(table[i].getX() >= -1 && table[i].getX() <= 1) ;
    }
    …
```

- File pointer所指定的位置在檔案的最前面(如果可能動過的話)
- 某個pointer 不是NULL
- 陣列的或容器的長度不是0，或至少有n個元素以上
- 陣列沒有用滿


89

Assertion Examples

- 某個不應該發生的情況

```
// Option is 'a', 'b', 'c'
switch (option)
{
case 'a':
   // a do something
   break;
case 'b':
   // b do something
   break;
default:
   // c do something
}
```

萬一option不是'c'呢？

```
// Option is 'a', 'b', 'c'
switch (option)
{
case 'a':
   // a do something
   break;
case 'b':
   // b do something
   break;
case 'c':
   // c do something
   break;
default:
   assert(false);
}
```

Better

90

Assertion

- ASSERT can also be used as a safety guard for work in progress

```
If (some condition) {
      // do some normal things
      ……
 } else {
      // right now, this condition should not happen
      // in case it does happen, "real" code is needed
      assert(false);
 }
```


Assertion

- 注意事項1

```
...
// assert two numbers are read
assert(fscanf(fp, "%d %d", &data1, &data2) == 2);
...
```

可以嗎？

```
...
int n = fscanf(fp, "%d %d", &data1, &data2);
// assert two numbers are read
assert(n == 2);
...
```

這樣才對

- 以上**兩段code**在debug mode與release mode行為**不相等**(release mode不執行assert，所以不作fscanf)


Assertion

- 注意事項2
  - 如果迴圈內有assertion，當迴圈大量重複執行時，可能會拖慢debug mode的執行速度
- 注意事項3
  - Assertion本身也會造成probe effect

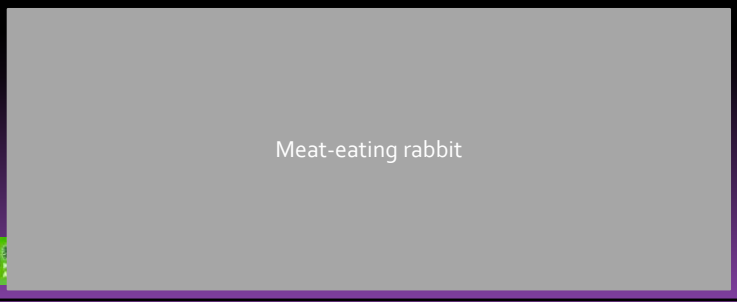
93

Assertion

94

Exception

- **Exception也可以擔任程式的隔火(水)門**
  - 在程式中throw exception可以終止執行(未被catch的話)，隔絕效果相當於assert
  - **這裡是討論的是throw exception，而不是exception handling(try/catch)**



Exception

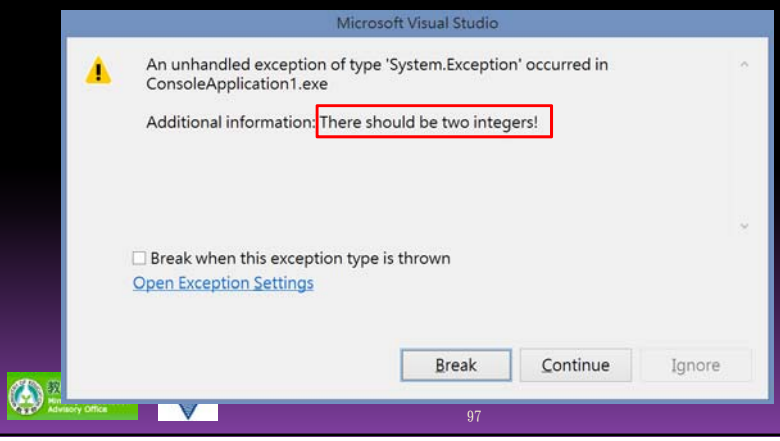
- Exception example in C#

```
// C# code
...
string line = file.ReadLine();
string[] integerStrings =
         line.Split(new char[] { ' ', '\t', '\r', '\n' },
         StringSplitOptions.RemoveEmptyEntries);
// assert two strings are read
if (integerStrings.Length != 2)
    throw new Exception("There should be two integers!");
int data1 = int.Parse(integerStrings[0]);
int data2 = int.Parse(integerStrings[1]);
...
```

Debug.Assert改寫為 throw exception

Exception

- 當出現exception時(未被catch)，IDE允許你觀察當時的變數，call stack等資訊，方便你來除錯



97

Exception

- 前面各assertion的例子都可以改寫為throw exception，達到隔絕的效果，例如

```
// C#
switch (option)
{
case 'a':
   // a do something
   break;
case 'b':
   // b do something
   break;
...
default:
   Debug.Assert(false);
}
```

```
// C#
switch (option)
{
case 'a':
   // a do something
   break;
case 'b':
   // b do something
   break;
...
default:
   throw new Exception("Bug!");
}
```

Exception

- Assertion
  - **僅做debug用**，測試時應該確認runtime不會(不可能)出現assert failed的情況 → **不影響release mode程式效能**
    - Assertion 的程式碼在 release 中會被移除
  - Release後，**runtime永遠不應該出現assert failed條件**
    - 例如：UI有防呆，保證輸入的某個數字合乎使用範圍，因此在runtime不用重複再檢查數字範圍。
- Exception
  - Throw exception程式碼是**系統本身的一部分**，不分Debug與release → **會影響release mode程式效能**
  - Release後，**當runtime出現不合理、無法繼續進行的條件時**，throw exception
    - 例如：UI雖然有防呆，只有檔案存在時才做讀檔，但是，該檔案仍然有可能在讀檔前被其他人或程式移除。

Exception

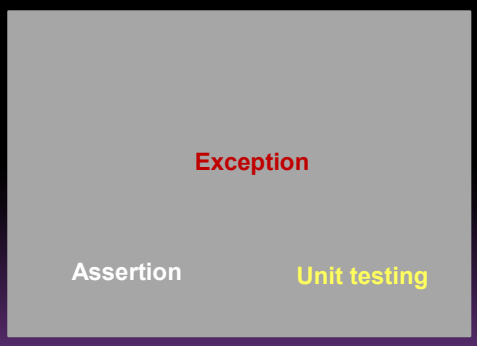
	Assertion	Exception
優點	1. 僅影響debug mode的執行效能，不影響release mode的效能。 2. 測試階段出現Assert failed時，程式一定會中止，有助於暴露問題，盡早修正。	1. Release mode與debug mode行為相同 2. 萬一在release mode出現exception的條件(且未catch)時程式會立刻當掉，不會延遲出現錯誤。 3. 出現exception時，其外層的code有機會catch exception，做補救或的額外的處理。
缺點	1. Release mode與debug mode行為不完全相同，可能出現怪bug。 2. 萬一在release mode出現assert failed的條件時，程式仍會盲目地做下去，可能產生莫名其妙的錯誤現象。 3. 出現Assert failed時，程式必定強迫中止，不適合作為共享的Library(或其他程式)使用。	1. 同時影響debug/release mode的執行效能。 2. 外層的code可能只做catch，卻不做適當的處理，隱藏了出現exception的事實，使得測試階段錯過修正問題的契機。

Exception

- 說人話，到底甚麼時候用Assertion，甚麼時候用Exception？
  - 用Assertion
    - 對於runtime效能有高度要求者
    - 充分Debug後runtime不可能再發生者
  - 用Exception
    - 在runtime判斷exception條件不至於影響效能者
    - 充分Debug後，runtime仍可能發生者
    - 寫作Class Library供他人使用者(enable try/catch)
    - 原始程式可能重複供其他專案利用者(enable try/catch)


Unit Testing

- **Unit Testing也可以擔任程式的隔火(水)門**
  - 如果method(function)的單元測試不通過，根本就不用run程式了，所以有隔離的效果，避免Error Propagation。



想從這裡過去嗎？


Unit Testing

- Assertion/Exception與Unit Testing的差別？

	Assertion/Exception	Unit Testing
地點	寫在production code裡面	寫在production code外面
目的	確定某段程式的precondition是正確的	1. 確定某method的實作是正確的。 2. 亦可測試當method的precondition不對時，method會做出正確的反應
Assert對象	Assert Precondition固定不變的條件(例如某參數不能是負的)，而不是特定的輸入或輸出值	通常使用許多組不同的input test data(值)測試該method，並Assert輸出為不同的output data(值)
Method中間	一個method中間可以寫很多個Assertion/Exception	通常將整個method視為一個單元，不管中間的執行過程

103

Unit Testing

- 選擇題：在程式執行時，確定某輸入資料檔的格式與其內容，均完全正確
  - (A) Assertion
  - (B) Exception
  - (C) Unit Testing
  - (D) 其他

104

Unit Testing

- 選擇題：假設UI有做防呆，當程式執行時，在Model Code確定某輸入資料(整數)值大於0
  - (A) Assertion
  - (B) Exception
  - (C) Unit Testing
  - (D) 其他


Unit Testing

- 選擇題：確定某一段高度最佳化的程式A(比較快，但是很長，很難懂)，與另一段程式B(比較慢，但是寫得很清楚易懂)，其執行結果完全相同
  - (A) Assertion
  - (B) Exception
  - (C) Unit Testing
  - (D) 其他


Conclusions on Defensive Programming

- Assert/exception可以確保元件之間的互動，符合contract，早一步攔截有問題的run，避免錯誤傳遞，排除整合所發生的複雜難解的bug
- 單元測試(unit testing)的施行雖然不能確保整合之後沒有 bug，不過它可以先期把關元件的品質，進而減少複雜而難解bug 的發生，讓許多 bug 早期發現。
- 多多益善，**有賴習慣的養成**
- 都是儘量讓錯誤早期發現，使得除錯能更有效率


除錯練習的正確解答在

- 3個練習題的正確答案在 http://140.115.53.53/trac/bugreport/wiki/HereWeAre