

# 修平技術學院

## 資訊網路技術系

### SCJP6.0 認證專題

指導老師：高國峰 老師

班 級：四技資網系 四年甲班

學生姓名：BN95035 張登凱

中華民國 99 年 6 月

## 摘要

台灣大學錄取率接近百分之百，在幾乎人人都是大學生的情況下。當我們要出社會並且求職的時候，沒有工作經驗的畢業生是比較吃虧的。所以擁有一張國際證照在求職時對我們會有所幫助。

SCJP 認證是許多 JAVA 語言的技術當中都是以此為基礎的。並且範圍相當廣泛。包含網頁 JAVA 應用程式；行動手機遊戲當中的 JAVA 遊戲軟體；企業電子商務...等。因此擁有一張 SCJP 認證國際證照，對於將來的就業，升遷，都具有一定的加分幫助。我也將在閱讀過程當中所了解到的考試重點與考試相關事項紀錄下來。方便日後同學在找尋 SCJP 認證考試的時候，能夠事半功倍。

目錄	
第一章 簡介	6
第二章 研讀書籍參考	7
第三章 研讀方法	8
第四章 讀書重點與考古題	9
1. Java 程式語言的基本觀念	10
1.1 Java 程式語言的關鍵字	11
1.1.1 存取修飾詞	12
1.1.2 類別、方法與變數保留字	12
1.1.3 流程控制保留字	13
1.1.4 錯誤處理保留字	13
1.1.5 套件控制項保留字	14
1.1.6 基本資料型別保留字	14
1.1.7 變數存取關鍵字	14
1.1.8 void 關鍵字	14
1.2 基本資料型別與字面常數(Literals)的範圍	15
1.2.1 基本資料型別的字面常數的範圍	15
1.2.2 所有基本資料型別的字面常數	15
1.3 陣列宣告、配置與初始化	17
1.3.1 宣告陣列	17
1.3.2 配置陣列	17
1.3.3 初始化陣列	18
2. 宣告與存取控制	20
2.1 宣告與修飾詞	21
2.1.1 類別宣告與修飾詞	21
2.1.2 方法與變數的宣告和修飾詞	25
2.2 宣告規則	26
2.2.1 原始檔、套件宣告和匯入敘述	26
2.2.2 main( )方法的格式要求	28
2.3 介面與抽象類別	29
2.3.1 介面 (interface)	29
2.3.2 抽象類別 (abstract class)	30
2.3.3 介面、抽象類別之注意事項	31
3. 運算子與指定運算	39
3.1 基本運算子	40
3.1.1 指定運算子	40
3.1.2 條件運算子	43
3.1.3 instanceof 比較	44

3.1.4	算數運算子 .....	<u>45</u>
3.1.5	位移運算子 .....	<u>47</u>
3.1.6	位元運算子 .....	<u>49</u>
3.1.7	位元補數運算子 .....	<u>50</u>
3.1.8	基本資料型別強制轉型.....	<u>50</u>
3.2	邏輯運算子 .....	<u>51</u>
3.2.1	短迴路邏輯運算子.....	<u>51</u>
3.2.2	邏輯運算子 (非短迴路運算子).....	<u>52</u>
3.3	方法的基本定義 .....	<u>52</u>
4.	流程控制、例外與斷言(Assertions).....	<u>61</u>
4.1	使用 if 與 switch 編寫程式碼.....	<u>62</u>
4.1.1	if-else .....	<u>62</u>
4.1.2	switch .....	<u>65</u>
4.2	使用迴圈寫程式碼 .....	<u>71</u>
4.2.1	while 迴圈.....	<u>72</u>
4.2.2	do-while 迴圈.....	<u>73</u>
4.2.3	for 迴圈.....	<u>73</u>
4.2.4	在 for 迴圈中使用 break 與 continue .....	<u>75</u>
4.2.5	無標籤的敘述 .....	<u>76</u>
4.2.6	帶標籤的敘述 .....	<u>76</u>
4.3	例外處理機制(Exception Handling).....	<u>78</u>
4.3.1	使用 try 與 catch 建立例外.....	<u>78</u>
4.3.2	finally .....	<u>79</u>
4.3.3	傳遞未捕捉到(catch)的例外.....	<u>81</u>
4.3.4	定義例外 .....	<u>81</u>
4.3.5	例外層級 .....	<u>82</u>
4.3.6	處理例外類別的階層 .....	<u>83</u>
4.3.7	例外對應(Exception matching).....	<u>83</u>
4.3.8	例外宣告與公用介面 .....	<u>84</u>
4.3.9	重新丟出相同例外 .....	<u>84</u>
4.4	Assertion(斷言)機制 .....	<u>85</u>
4.4.1	Assertion 概述.....	<u>86</u>
4.4.2	Assertion 語法.....	<u>88</u>
4.4.3	Assertion 應用 .....	<u>89</u>
5.	物件導向、過載與覆寫、建構子與回傳類型.....	<u>100</u>
5.1	封裝的效益.....	<u>101</u>
5.1.1	IS-A 與 HAS-A 的關係.....	<u>101</u>
5.2.1	覆寫的方法 .....	<u>103</u>

5.2.2 過載的方法 .....	103
5.3 建構子與實體化 .....	104
5.3.1 建構子的基本概念 .....	104
5.3.2 判斷是否會產生預設的建構子.....	107
5.3.3 過載的建構子 .....	108
5.4 合法的回傳型態 .....	109
5.4.1 回傳型態的宣告 .....	109
5.4.2 回傳數值 .....	109
6. Math 類別、字串及 StringBuffer 類別 .....	118
6.1 String 類別的用法 .....	119
6.1.1 字串是不可改變內容的物件.....	119
6.1.2 字串與記憶體 .....	119
6.1.3 String 類別中重要的方法 .....	120
6.1.4 StringBuffer 類別 .....	122
6.2 Math 類別的用法 .....	122
6.2.1 java.lang.Math 類別中的方法 .....	122
6.3 equals( ) 的用法 .....	125
6.3.1 = = 與 equals( ) 方法的比較 .....	125
7. 物件與集合 .....	133
7.1 覆寫 hashCode( ) 與 equals( ) .....	134
7.1.1 實作 toString( ) 方法 .....	134
7.1.2 覆寫 equals( ) 的方法 .....	135
7.1.3 覆寫 hashCode( ) 的方法.....	138
7.2 記憶體回收 .....	140
7.2.1 記憶體回收與記憶體管理的概論 .....	140
7.2.2 Java 的記憶體回收行程概論 .....	141
7.2.3 撰寫讓物件明確合適記憶體回收的程式 .....	141
7.2.4 在執行記憶體回收行程前的清理工作—Finalize( ) 方法 .....	143
8. 內部類別 .....	150
8.1 內部類別 .....	151
8.1.1 設計正常的內部類別 .....	151
8.1.2 從內部類別之內參考內部或外部物件實體 .....	152
8.2 方法區域內部類別 .....	154
8.3 匿名內部類別 .....	155
8.4 靜態巢狀類別 .....	158
8.4.1 實體化靜態巢狀類別 .....	159
9. 執行緒(Thread) .....	167
9.1 執行緒的定義、實體化與啟動.....	168

9.1.1 定義執行緒的作法 .....	<u>169</u>
9.2 防止執行緒的執行 .....	<u>174</u>
9.2.1 休眠狀態 .....	<u>174</u>
9.3 程式的同步化 .....	<u>175</u>
9.3.1 同步化與鎖定的作業 .....	<u>175</u>
9.4 執行緒的互動 .....	<u>176</u>
9.4.1 notifyAll( ) .....	<u>178</u>
第五章 經驗與心得分享 .....	<u>187</u>
第六章:結論 .....	<u>188</u>
附錄 .....	<u>189</u>

## 第一章 簡介

自從 1995 年 Sun 提出 Java 技術與解決方案開始，至今 Java 已是企業在開發重要專案與系統時所選擇的技術。因此不管是未來從事任何相關的 IT 職務，如程式設計師、網頁開發師、系統分析師或是 J2EE 系統架構師等，都必須充分了解並熟悉運用以 Java 程式語言為基礎的系統開發技術。而 SCJP 證照所認證的技術，正是這項技術。在 Java 認證範疇中，若想再考取其它進階認證（例如 SCWCD、SCJD、SWBCD 等）之前，都必須先取得 SCJP，因此，SCJP 認證等同於 Java 領域的「基本學歷」。

### 證照優勢:

- 國際：Linux 、 JAVA 為國際性技能，跟隨產業發展找工作不愁。
- 優先：資訊相關工作有 42% 以『相關認證』作為優先錄取條件。
- 高薪：年薪比一般工程師多 10%~30%。
- 除了徵才加薪參考之外，也是升學、推甄、留學、移民重要佐證。

## 第二章 研讀書籍參考

### 1. Java 認證 SCJP 6.0/5.0--猛虎出關

作者：段維瀚

出版社：基峰資訊

出版日：2008.09.30

### 2. SCJP 6.0 認證教戰手冊（附光碟）

作者：黃彬華

出版社：基峰

出版日期：2008 年 10 月 31 日

### 3. SCJP · SCJD 專業認證指南

原文書名：Sun Certified Programmer & Developer for Java 2

作者：Kathy Sierra · Bert Bates[譯者：吳品清 · 張世敏]

出版社：學貫行銷股份有限公司

### 4. JavaWorld@TW 論壇

<http://www.javaworld.com.tw/jute/>(首頁)

此論壇內容包括 Java 相關的所有範圍統籌，也提供考試方面的應試資訊。

<http://www.javaworld.com.tw/jute/post/page?bid=17&sty=3&age=0&tpg=1>

(SCJP 相關資訊)

在此討論區內，有相當多 SCJP 考試資訊，亦可在此網站內提出相關疑問，

將會有 Java 專業人員出來解答，是個相當不錯的論壇。



### 第三章 研讀方法:

1. 各章標題:參考書籍要清楚了解各章節的標題，考試中所考的相關題目大多都包含多種相關的標題，因此了解關鍵字的相關知識與技能是很重要的。
2. 習題:相關考古題型會放於各章的內容後面，這些都是需要逐步完成的練習，這些習題可以協助我們在考 SCJP 時，要我們所要了解的知識與技巧。從當中完成這些習題可以確實的幫助我們了解自己本身知識的不足。當我們遇到了問題時，回各章節研讀相關的知識，並且充分的了解才是最快速的方法。
3. 各章節的重點回顧:考前研讀各章重點回顧是很重要的，內容都是該張重點的檢核表，當作臨考前的複習之用。
4. 自我評量:書籍後面所提供的模擬考題，與考 SCJP 時所要遇到的主題結構與本質都類似。在閱讀完各章節的內容之後如果能練習這些自我評量的考題都可以強化我們在該章所學到的內容與結構，評估自己對考題的熟悉度。

#### 第四章 讀書重點與考古題

1. Java 程式語言的基本觀念。
2. 宣告與存取控制。
3. 運算子與指定運算。
4. 流程控制、例外、Assertion 機制。
5. 物件導向、過載與覆寫、建構子與回傳類型。
6. Math 類別、字串及 StringBuffer 類別。
7. 物件與集合。
8. 內部類別(Inner Classes)。
9. 執行緒(Thread)。

## 1. Java 程式語言的基本觀念

### 1.1 Java 程式語言的關鍵字

#### 1.1.1 存取修飾詞

#### 1.1.2 類別、方法與變數保留字

#### 1.1.3 流程控制保留字

#### 1.1.4 錯誤處理保留字

#### 1.1.5 套件控制項保留字

#### 1.1.6 基本資料型別保留字

#### 1.1.7 變數存取關鍵字

#### 1.1.8 void 關鍵字

### 1.2 基本資料型別與字面常數的範圍

#### 1.2.1 基本資料型別的數字範圍

#### 1.2.2 所有基本資料型別的字面常數

### 1.3 陣列宣告、配置與初始化

#### 1.3.1 宣告陣列

#### 1.3.2 配置陣列

#### 1.3.3 初始化陣列

## 1.1 Java 程式語言的關鍵字

「什麼是關鍵字呢？」關鍵字就是在程式語言中，語言本身已經定義並代表特殊意義的字，使用者（程式設計師）不能將其做為別的用途。

另外，有些特別的字稱為保留字（reserved word），系統保留但不做為識別字（identifier）使用。

在 Java 1.4，有 47 個關鍵字和 5 個保留字。下表列出了 Java 所有的關鍵字及保留字。

Java 的關鍵字				
abstract	assert	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	enum(1.5)
extends	final	finally	float	for
if	implements	import	instanceof	int
interface	long	native	new	package
private	protected	public	return	short
static	strictfp	super	switch	synchronized
this	throw	throws	<b>transient</b>	try
void	<b>volatile</b>	while		

Java 的保留字				
const	goto	true	false	null

const 和 goto 為 C/C++ 的關鍵字，在 Java 則是保留而不使用。const 在 C/C++ 是用來定義常數，Java 則是使用 final 修飾字來定義常數。goto 經常會破壞 C/C++ 程式的結構化，為避免混淆，在 Java 保留但不使用。

true 和 false 為布林值型別的字面常數，故為保留字面常數（Reserved Literals）。null 可作為參照變數（reference variable）的值，亦為保留字面常數，代表「空物件」。assert 和 strictfp 是最晚加入的關鍵字。assert 和程式的除錯有關，而 strictfp 和浮點數的計算有關。所有的關鍵字和保留字全部都是小寫字母，故某識別字有大寫字母出現時，該識別字為非關鍵字。

### 1.1.1 存取修飾詞

類別成員的存取修飾詞(包括屬性與方法)代表的意義：

- private：定義只能在自己的類別中存取的方法或變數。
- protected：定義只有在同一套件中的類別或類別的子類別才能存取的方法或變數。
- public：定義其它類別皆可以存取的類別、方法或變數。

### 1.1.2 類別、方法與變數保留字

- abstract：可宣告無法實體化的類別或抽象方法。
- class：定義類別的關鍵字。
- extends：表示子類別繼承超類別。
- final：使類別無法被繼承、方法無法被覆寫或變數無法重新賦值。
- implements：表示類別要實作介面。
- interface：用來定義介面的關鍵字。
- native：用來表示方法是使用不同平台的語言編寫而成。
- new：可叫用建構子來建立物件。
- static：定義方法或變數屬於靜態的。
- strictfp：在方法或類別前使用，表示所有運算式中的浮點數會遵守 FP 規格。
- synchronized：定義同步化區塊。
- transient：定義不需要序列化的欄位。

- volatile：定義欄位存取時，必須每次都需重新讀取(memory)以免資料在多執行緒環境下有不一致的問題。

### 1.1.3 流程控制保留字

- break：結束程式碼區段。
- case：根據 switch 的測試來決定執行的程式碼區段。
- continue：忽略敘述後面的程式碼，並開始執行下一次迴圈。
- default：如沒有符合 switch-case 敘述，則執行此預定程式碼區段。
- do：先執行程式碼，然後結合 while 敘述執行測試，來判定是否繼續執行該區段。
- else：如果其 if 測試結果為 false，就執行 else 程式碼區塊。
- for：迴圈控制指令，用來執行一次以上的指令控制。
- if：條件邏輯測試，執行結果 true 或 false。
- instanceof：判定物件是否為某一類別、超類別或介面的實體。
- return：從方法回傳結果。而 return 敘述後面的程式碼就不會被執行。
- switch：多重條件判斷控制，搭配 case，default。
- while：當某條件為 true 時，重複執行程式碼區段。

### 1.1.4 錯誤處理保留字

- catch：宣告處理特定例外時所用的程式碼區段。
- finally：在處理例外時，不管在什麼樣的程式流程中都必須執行的程式碼區塊。
- throw：將例外傳送給呼叫該方法的指令。

- throws：宣告某方法可能會回傳例外物件給呼叫者。
- try：試圖執行可能產生例外的程式碼區塊。
- assert：斷言條件表示式，以驗證程式設計人員的假設。

#### 1.1.5 套件控制項保留字

- import：將套件或類別匯入到程式碼中，以方便程式寫作。
- package：定義原始檔案中所有類別所屬的套件。

#### 1.1.6 基本資料型別保留字

- boolean：表示 true 或 false 的值。
- byte：8 位元的整數(有正負號)。
- char：16 位元的 unicode 字元(無符號)。
- double：一個 64 位元的浮點數(有正負號)。
- float：一個 32 位元的浮點數(有正負號)。
- int：一個 32 位元的整數(有正負號)。
- long：一個 64 位元的整數(有正負號)。
- short：一個 16 位元的整數(有正負號)。

#### 1.1.7 變數存取關鍵字

- super：指向超類別物件的系統預設的參考變數。
- this：指向觸發方向執行的物件的系統預設參考變數。

#### 1.1.8 void 關鍵字

- void：表示方法沒有回傳值，且只用在定義方法宣告的回傳值。

## 1.2 基本資料型別與字面常數(Literals)的範圍

### 1.2.1 基本資料型別的字面常數的範圍

類型	位元	位元組	最小範圍	最大範圍
byte	8	1	$-2^7$	$2^7-1$
short	16	2	$-2^{15}$	$2^{15}-1$
int	32	4	$-2^{31}$	$2^{31}-1$
long	64	8	$-2^{63}$	$2^{63}-1$
float	32	4	IEEE 754 規格	
double	64	8	IEEE 754 規格	

### 1.2.2 所有基本資料型別的字面常數

#### 整數字面常數

目前有三種方法可以讓你表示 JAVA 語言中的整數數值。十進位、八進位、十六進位。大部分與整數有關的考題，幾乎都是使用十進位的表示方式，較少使用八進位或十六進位的表示法。

十進位表示法：以數字開頭的方式表示。如下：

```
int x = 123; //相當於數字 123
```

八進位表示法：以數字 0 開頭的方式表示。如下：

```
int y = 06; //相當於十進位的 6  
int y = 011; //相當於十進位的 9
```

十六進位表示法：以 0x 開頭的方式表示。十六進位的 1~15 的值分別為 0 1 2 3 4 5 6 7 8 9 a b c d e f，程式中可支援大小寫。如下：

```
int z = 0x0008 //相當於十進位的 8  
int z = 0x0025 //相當於十進位的 37
```



### 浮點字面常數

浮點數字是由一個整數、一個小數點符號以及許多數字所組成。

```
double d = 11301874.9881024;
```

以上這段程式碼 11301874.9881024 代表常數值。JAVA 浮點字面常數屬於 double 型別(64 位元)，如果你要將浮點字面常數指派給 float 型別(32 位元)的變數，後方必須加上 f 或 F，編譯器才不會出現錯誤。

### 布林字面常數

布林字面常數是 boolean(邏輯-布林函數)。boolean 值只能為 true 或 false。

### 字元字面常數

一般 char 字元的表示方式是以單一字元表示，前後加上單引號。

```
char a = 'a' ;  
char b = '@' ;
```

你也可以鍵入字元的 Unicode 值，做法是在數值前面加上 Unicode 標記 (\u)。如下面程式：

```
char letterN = '\u004E' ; // The letter 'N'
```

如果字元無法以一般字面常數來表示，你也可以使用 escape 碼(跳脫字元 (escapes))來表示。如下表：

字元	用法
'\n'	換行
'\b'	退格
'\t'	定位 (tab 符號)

<code>'\r'</code>	游標移至本行之首
<code>'\''</code>	單引號 ‘
<code>'\"'</code>	雙引號 “
<code>'\\'</code>	反斜線 \

## 1.3 陣列宣告、配置與初始化

### 1.3.1 宣告陣列

陣列的宣告方式是先敘述陣列要包含的資料類型(可能是物件或基本資料型別)，接著在識別子的左邊或右邊使用括號。宣告要包含基本資料型別的陣列：

```
int [ ] key ; //中括號位於名稱前面
int key [ ] ; //中括號位於名稱後面(也是正確的，但比較容易被誤解)
```

宣告要包含物件參考的陣列：

```
Thread [ ] threads ;
Thread threads [ ] ; //同上
```

也可宣告為多維陣列，程式如下：

```
String [ ] [ ] [ ] occupantName ; //第一個
String [ ] ManagerName [ ] ; //第二個
```

第一個範例是三維陣列，第二個是二維陣列，第二個範例中變數名稱前方與後方各有一個中括號，這是可以被編譯器所接受的。

### 1.3.2 配置陣列

配置陣列就是建立陣列物件。在建立陣列物件時，JAVA 必須知道要分配多少空間，因此你必須在配置期間指定陣列大小。

## 配置一維陣列

配置陣列最簡單的方法，就是使用 `new` 關鍵字，後接陣列型別及中括號，中括號中指明陣列要容納的陣列元素數量。下面程式是建立 `int` 型別的陣列範例：

```
int [ ] testScores ; //宣告 int 型別的陣列
testScores = new int [4] ; //配置陣列並儲放參考值(記憶體位置)到
testScores
```

## 配置多維陣列

多維陣列只是陣列的陣列，因此 `int` 型別的二維陣列實際上是 `int` 型別陣列的物件，而該陣列中的每個元素都包含一個 `int` 陣列的參考。第二維則包含實際的 `int` 基本資料型別。如下面程式：

```
int [ ] [ ] ratings = new int [3] [ ] ;
```

上述程式只有第一組中括號提供陣列大小，由於 JVM 只需要知道指定給 `ratings` 變數的物件大小，因此這程式在 JAVA 中是可以被接受的。

### 1.3.3 初始化陣列

初始化陣列表示在陣列中放入東西。陣列裡的東西稱為陣列的元素，它們可以是基本資料型別或參考資料型別。若陣列包含的是物件，這並不表示陣列真的含有這些物件，而是存放對物件的參考值，就像其它非基本資料型別變數是沒有包含物件實體一樣。

#### 使用一行程式碼完成宣告、配置與初始化

透過兩種陣列專用的語法捷徑，你可以在一個敘述中完成初始化與配置。如下面程式碼：

```
int x = 9 ; //程式碼這邊進行宣告、建立與初始化
int [ ] dots = {3, 6, x, 8 } ; //這邊第二行牽涉到以下四點
```

- 宣告名為 dots 的 int 陣列參考變數
- 建立長度為 4 的 int 陣列
- 在元素中放入 3、6、9 跟 8 的值
- 將新的陣列物件存放 dots 參考變數

## 2. 宣告與存取控制

### 2.1 宣告與修飾詞

#### 2.1.1 類別宣告與修飾詞

#### 2.1.2 方法與變數的宣告和修飾詞

### 2.2 宣告規則

#### 2.2.1 原始檔、套件宣告和匯入敘述

#### 2.2.2 main( )方法的格式要求

### 2.3 介面與抽象類別

#### 2.3.1 介面 (interface)

#### 2.3.2 抽象類別 (abstract class)

#### 2.3.3 介面、抽象類別之注意事項

### ◎ 考題分析

## 2.1 宣告與修飾詞

宣告類別、巢狀類別、方法、實體變數與自動變數，以上這些宣告或方法，可讓你使用所有修飾詞(如：public、final、static、abstract 等)。

如果你是用 Java 語言寫程式碼，寫的是類別，類別中又包含了變數與方法。在使用類別、方法和變數的宣告時，會與程式碼息息相關。例如，你可以透過應用程式的任何程式碼去存取 public 方法，但它如果宣告成 private，其它類別或程式碼都無法使用。

### 2.1.1 類別宣告與修飾詞

宣告的基本規則如下：

- 每個原始碼檔案只能有一個 public 類別。
- 檔案名稱必須與 public 類別的名稱相同。
- 若檔案屬於某套件，該套件敘述必須放在原始碼檔案中的第一行。
- 若有任何匯入敘述，必須將其放在套件敘述與類別宣告之間。如果沒套件敘述，則匯入敘述必須放在原始碼檔案中的第一行。如果沒有套件敘述或匯入敘述，類別宣告必須置於原始碼檔案中的第一行。
- 匯入敘述與套件敘述可套用在原始碼檔案中的所有類別。

下列程式碼是基本類別宣告：

```
class MyClass { }
```

此程式碼可以成功編譯，不過也可在類別宣告之前加入修飾詞。

修飾詞可分成兩類：

- 存取修飾詞：public、protected、private
- 非存取修飾詞：strictfp、final、abstract

這邊先討論存取修飾詞。Java 有四種存取權限，但只有三個存取修飾詞，

因此 JAVA 的存取控制有點麻煩。如果你沒有使用這三個存取修飾詞，便會得到第四種的存取控制層級(稱為「預設」或「套件」存取)。也就是說，你宣告的每個類別、方法和實體變數都會有一個存取權限(不管你有沒有指定)。另外，雖然這四的存取權限都適用於大多數的方法和變數宣告，但在類別中只能使用 public 或預設存取權限來宣告。

#### 2.1.1.1 類別的存取修飾詞

類別存取就是，當我們說某類別(類別 A)的程式碼可以存取另一個類別(類別 B)時，這表示類別 A 可以做以下三件事情：

- 建立類別 B 的物件實體。
- 繼承類別 B (就是成為類別 B 的子類別)。
- 存取類別 B 中的某些方法和變數。存取的內容會隨方法和變數的存取控制層級而異。

事實上，存取就表示「可見度 (visibility)」，若類別 A 看不到類別 B，類別 B 之方法和變數的存取控制層級便無法發揮作用，而類別 A 將無法存取這些方法和變數。

#### ◎ 預設存取權限 (default)

在宣告時，具有「預設」存取權限的類別前面並沒有修飾詞。也就是說，如果你宣告類別中沒有鍵入修飾詞，將會得到此存取權限。由於「在同一套件的類別中才能看到預設存取權限的類別」，因此你可以將預設存取想像成套件層級的存取，例如，若類別 A 與類別 B 位於不同套件，而類別 A 具有預設存取層，則類別 B 便無法建立類別 A 的物件實體，甚至無法宣告類別 A 的變數回傳值。事實上，類別 A 對類別 B 來說是不存在的，如果類別 B 想存取類別 A，編譯器便會發生錯誤。

以下是範例原始碼：

```
package cert;  
class Beverage{  
}
```

再看看下面原始碼：

```
package exam.stuff;  
import cert.Beverage;  
class Tea extends Beverage{ }
```

在此範例中，超類別(Beverage)與子類別(Tea)分別位於不同的套件中，Tea 檔案上的 import 敘述試圖匯入 Beverage 類別。Beverage 檔案雖然可以正常編譯，但是看看我們如果試圖去編譯 Tea 檔案，則會發生 Tea 的超類別 Beverage 具有預設存取權限，而且位於不同的套件，因此，Tea 無法進行編譯。

### ◎ 公用存取權限 (public)

使用 public 關鍵字宣告的類別就是所謂的公用類別。在 JAVA 所有類別都可以存取公開類別。在上面範例中，如果不想將子類別和超類別放在同一套件中，但又要讓程式碼順利執行，你必須在超類別宣告的前面加上 public 關鍵字，如下：

```
package cert;  
public class Beverage{ }
```

這樣所有套件的所有類別都能夠看到 Beverage 類別。



### 2.1.1.2 非存取修飾詞

你可以使用 `final`、`abstract` 或 `strictfp` 等關鍵字宣告類別。不管類別的存取權限為何，這些修飾詞都必須加在存取修飾詞的後面，因此你可以將類別同時宣告為 `public` 與 `final`。但並非每一個非存取修飾詞都可以混合使用。例如，你可以同時使用 `strictfp` 與 `abstract` 或 `strictfp` 與 `final`，但是你絕對不能同時將類別宣告成 `final` 與 `abstract`。

由於我們不需要知道 `strictfp` 的作業方式(系統內部處理)，因此我們只著重在如何將類別修飾成 `final` 或 `abstract`。這邊，只需要知道 `strictfp` 是關鍵字，而且可以用來修飾類別或方法，但絕不能修飾變數。

#### `final` 類別

在宣告中使用 `final` 關鍵字表示該類別無法再定義類別。換句話說，其它類別都無法成為 `final` 的子類別，而且如果硬要這樣做，編譯器則會發生錯誤。

為何要將類別宣告成 `final` 呢？基本上，只有在需要確保他人無法覆寫該類別中的方法時才能將類別宣告成 `final`。也就是說，`final` 關鍵字可以確保該方法不會遭到他人修改。以下情況非常適合用來說明 `final` 類別的特性；假設你在目前使用的類別中發現任何與方法有關的錯誤，但你手邊沒有原始碼，以至於無法藉由修改原始碼來更正方法的錯誤，此時你可以延伸類別並覆寫新子類別中的方法，然後用子類別取代原有的超類別。可是如果該類別是 `final`，就無法這樣使用了。

## 2.1.2 方法與變數的宣告和修飾詞

### ◎ 公用存取權限 (public)

- 若成員被宣告為 public，則所有其它類別均可以存取該成員(不論是否在同一套件內)。
- 子類別可完全繼承父類別的所有公開成員，並且可直接使用這些成員，例：父類別有一 doRooThings( ) 函式，子類別可直接呼叫 doRooThings( )。

### ◎ 私有存取權限 (private)

- 除了它所屬的類別外，沒有其它類別可以取用。
- 子類別亦無法繼承父類別的私有成員。此外，當在子類別宣告一個與父類別私有成員一樣名稱的成員也是合法的。

### ◎ 預設存取權限

- 只有同一套件內之類別方法可存取。

### ◎ 保護存取權限 (protected)

- protected 和預設存取權限幾乎是相同，但有一關鍵點不同，就是保護存取除了只能讓同套件內的類別存取外，也可以被子類別存取(即使這個子類別在另外一個套件內)。因此，不同套件的子類別只能透過繼承的方式來存取 protected 成員。

決定類別組件的存取權限：

可見性	public	protected	default	private
同一個類別	Yes	Yes	Yes	Yes
同一個套件內的任何類別	Yes	Yes	Yes	No
同一個套件內的子類別	Yes	Yes	Yes	No

不同套件的任何非子類別	Yes	No	No	No
不同套件的子類別	Yes	Yes	No	No
不同套件的子孫類別	Yes	Yes	No	No
與父不同套件但與子同套件的類別，透過子類別的參考，存取父類別	Yes	No	No	No
與父同套件但與子不同套件的類別，透過子類別的參考，存取父類別	Yes	Yes	No	No

## 2.2 宣告規則

### 2.2.1 原始檔、套件宣告和匯入敘述

#### ◎宣告的規則

- 每個原始碼檔案只能有一個公用類別。
- 檔案名稱必須和公用類別名稱一致。
- 若類別屬於某套件，該套件敘述必須位於原始碼檔案中的第一行。
- 匯入與套件敘述都可套用於原始碼檔案中的所有類別。
- 若是匯入敘述，它們必須放在套件敘述與類別宣告之間。若沒有套件敘述，則匯入敘述必須位於原始碼檔案中的第一行。

#### 原始檔結構

正確的宣告方法，例：com.geekdanonymous 套件中宣告 Foo 類別正確的程式碼：

```
package com.geeksanonymous; //此行須注意分號
class Foo { }
```

以下為錯誤的範例，因為每個原始檔只能有一個套件敘述：

```
package com.geeksanonymous;
package com.wickedlysmart; //這邊錯誤，只能有一個套件宣告
class Foo{ }
```

若 Foo 需增加匯入敘述，必須位於套件宣告下方以及類別宣告上方：

```
package com.geeksanonymous;
import java.util.*; //萬用字元套件的匯入敘述
import com.wickedlysmart.Foo; //特定類別的匯入敘述
```

反之，若 Foo 類別沒有套件宣告，則匯入敘述是必須位於類別宣告上方：

```
import java.util.*; // wildcard 套件的匯入敘述，
//考試須注意.*;之前的項目需為套件名稱
import com.wickedlysmart.Foo; // explicit 類別的匯入敘述
```

每個原始碼檔案只能有一個公用類別，以下是錯誤範例：

```
package com.geeksanonymous;
public class Foo {}
public class Bat {} //此為錯誤的，因只能有一個公用類別
```

以下為正確的程式碼：

```
package com.geeksanonymous;
class Foo {}
public class Bat {}
```

使用匯入敘述

匯入敘述會有兩種:wildcard 匯入敘述與 explicit 類別匯入敘述。將類別放在套件內時，會賦予一個較長的名稱。例如，若 Foo 位於 com.geekcanonymous 套件中，則 Foo 類別的名稱仍為 Foo，但是它有一個完整名稱 com.geekcanonymous.Foo。當你將 Foo 放在套件內時，如果你參照的是其它程式碼中的 Foo 類別，則必須告訴編譯器要指向哪一個 Foo 類別。以下有兩個正確用法：

● 使用匯入敘述：

```
import com.wickedlysmart.Foo;
class Bar {
    void dostuff( ) {
        Foo f = new Foo( ) ;
    }
}
```

- 程式碼中使用完整名稱：

```
class Bar {
    void dostuff( ) {
        com.wickedlysmart.Foo f = new com.wickedlysmart.Foo ( )
    }
}
```

### 2.2.2 main( )方法的格式要求

正確定義 main( )方法之基本要求：

- 它必須為 static。
- 它須宣告為 public。
- 它必須具有 void 回傳型別。
- 它必須具有一個 String 陣列參數。
- 你可以隨心所欲地命名該參數。

main( )的使用

定義錯誤的 main( )方法是屬於執行錯誤，而不是編譯錯誤，若不是類別沒有定義正確的 main( )，便無法執行該類別，不過還是可以透過其它程式碼進行實體化。考試時，只須注意無法正常執行的 main( )方法屬於執行錯誤，而不是編譯錯誤。

## 2.3 介面與抽象類別

### 2.3.1 介面 (interface)

- 介面內部的方法絕對不能有實作部分，而且一定是公開的方法。
- 介面內部的變數宣告，系統都會自動轉為公開且共用的常數，也就是沒有物件實體變數。

介面宣告

```
public interface NIC {  
    //共用常數宣告，一定是 public, static, final  
    //抽象方法宣告，一定是 public  
    void send(Packet data);  
    Packet receive( );  
    void resend( );  
}
```

實作類別(具體類別，Concrete Class) : implements NIC

```
public class FiberNIC implements NIC {  
    private String producer;  
    public void send(Packet data) {  
        //光纖實際傳輸方式  
    }  
    public Packet receive( ) {  
        //光纖實際接收方式  
    }  
    public void resend( ) {  
        //光纖重新傳輸方式  
    }  
    public class WlanNIC implements NIC {  
        private String producer;  
        public void send(Packet data) {  
            //RF 實際傳輸方式  
        }  
        public Packet receive( ) {
```

```

//RF 實際接收方式
}
public void resend( ) {
//RF 重新傳輸方式
}

```

### 2.3.2 抽象類別 (abstract class)

- 在實作過程發現：每個實作類別中的重新傳送的處理指令都相同，也有共同屬性(producer)希望把共同的部分提升到上層 NIC 中；這樣會牴觸原有的 interface 原則，所以，就有另一種東西被定義出來 抽象類別。
- 所謂抽象類別就是實作不完全的類別，部分方法在定義時還不知道該如何完成，但有部分已經決定實作內容，此時就可以利用抽象類別來定義。未定義實作部份的方法，就宣告為抽象方法，方法宣告前方多了一個修飾詞 abstract，有實作區塊，直接以分號「；」結尾。

抽象類別 AbsNIC 的宣告

```

public abstract class AbsNIC {
//共用都有的變數宣告
private String provider;

//可以有建構子宣告，但不可以呼叫抽象方法！
//抽象方法宣告，不一定是 public，但絕對不可以是 private。
public abstract void send(Packet data);
public abstract Packet receive( );
//一般的方法
public void resend( ) {

```

```
//通用的重傳機制
}
}
```

子類別宣告：extends AbsNIC

```
public class FiberNIC extends AbsNIC {
    public void send(Packet data) {
        //光纖實際傳輸方式
    }
    public Packet receive( ) {
        //光纖實際接收方式
    }
}
```

### 2.3.3 介面、抽象類別之注意事項

類別內部只要有一個以上的抽象方法，該類別就無法被拿來產生物件。因此，編譯器會要求該類別一定要宣告為抽象類別，否則會有編譯錯誤訊息。

- 若程式中利用抽象類別產生物件，如：`new AbsNIC( )`，則編譯器會產生編譯錯誤的訊息。因為，介面之中的方法都無實作方法區塊，因此，有人說”介面就是完整的抽象類別”。也因為這樣，當然就無法利用介面去產生物件。

- 可以利用既有介面擴充定義為新的介面，如同物件繼承物件一樣。

```
interface2 extends interface1 {
    void newAbstractMethod( );
}
```

- 類別繼承抽象類別或實作介面時，若無法實作所有的抽象方法，該類別必



宣告為抽象類別，否則會有編譯錯誤訊息。

- Java 只支援單一繼承，而不支援多重繼承：利用介面的機制可以達成虛擬的多重繼承 extends 的後面只能接一個類別或抽象類別。implements 的後面則可以有一個以上的介面。

### ◎ 考題分析

1. 下列哪一個是條件最多的存取修飾詞。該修飾詞只允許類別的組件存取同一套件之其它類別的組件。
  - A. public
  - B. abstract
  - C. protected
  - D. synchronized
  - E. 預設存取權限

解：E. 預設存取權限 default access 是套件導向的存取修飾詞。A. 與 C. 是錯誤的答案，因為 public 跟 protected 的限制較少。B. 與 D 是錯誤答案，因為 abstract 跟 synchronized 不是存取修飾詞。

2. 以 protected 類別中的方法為例，下列哪一個修飾詞可以限制只有同一類別的其它組件可以存取此方法？
  - A. final
  - B. static
  - C. private
  - D. protected
  - E. volatile
  - F. 預設存取權限

解：C. private 存取修飾詞會限制只能存取同一類別的組件。A 跟 B 跟 E 雖是修飾子，但並非存取修飾詞，答案 D. 跟 F. 則是錯誤的存取修飾詞。

3. 若
1. abstract class A {
  2.     abstract short m1( ) ;
  3.     short m2( ) { return (short) 420; }
  4.     }
  - 5.
  6. abstract class B extends A {
  7.     //程式碼遺失了嗎?
  8.     short m1( ) { return (short) 42; }
  9.     }

下列哪三項敘述是對的？（選出三項）

- A. 不需變更程式碼亦可正常編譯
- B. 類別 B 必須將 m2( ) 方法宣告為 abstract 或實作 m2( ) 方法，否則程式碼無法編譯。
- C. 類別 B 必須將 m2( ) 方法宣告為 abstract 或實作 m2( ) 方法，但這不是必要的。
- D. 只要有第 8 行，類別 A 就必須宣告 m1( ) 方法。
- E. 若將第 6 行取代之為「class B extends A{ }」，程式碼便可編譯。
- F. 若類別 A 不是 abstract，且實作第 2 行的 m1( ) 方法，則程式碼將無法正常編譯。

解：A 跟 C 跟 E. 是正確答案。

為 A 跟 C 中 abstract 類別不需要在實作超類別的其它方法。E. 的語法是為了繼承類別 A 的用法。B 的用法相當於將 abstract 實作超類別，而 abstract 並不需要實作超類別，D 錯誤是因為，如果要繼承另一個類別的類別可以隨意新增方法。F 是錯誤答案，因為你可以繼承具體類別的 abstract 類別。

4. 對於非巢狀類別與介面而言，下列哪兩項是正確的宣告？(選出兩項)
- A. final abstract class Test { }

- B. `public static interface Test {}`
- C. `final public class Test {}`
- D. `protected abstract class Test {}`
- E. `protected interface Test {}`
- F. `abstract public class Test {}`

解:C F 為正確的類別宣告，A. 是因為類別宣告不可同時宣告為 `final` 與 `abstract`，B. 是因為介面與類別不可宣告為 `static`，D. 錯誤是不可將類別宣告為 `protected`，E. 錯誤是因為不可將介面宣告為 `protected`。

5. 下列哪幾項是正確的方法宣告？

- 1 - `protected abstract void m1 ( )`;
- 2 - `static final void m1 ( )`;
- 3 - `transient private native void m1 ( ) {}`
- 4 - `synchronized public final void m1 ( ) {}`
- 5 - `private native void m1 ( )`;
- 6 - `static final synchronized protected void m1 ( ) {}`

- A. 1
- B. 2
- C. 3
- D. 4
- E. 5
- F. 以上皆是

解: 1. 2. 4. 5. 6 是正確的宣告，3 是因為 `transient` 只能用在變數宣告。

6. 若

- 1. `package testpkg.pl;`
- 2. `public class ParentUtil {`
- 3.     `public int x = 420;`
- 4.     `protected int doStuff ( ) { return x; }`
- 5. `}`

```

1. package testpkg.p2;
2. import testpkg.p1.ParentUtil;
3. public class ChildUtil extends ParentUtil {
4.     public static void main (String [ ] args) {
5.         new ChildUtil ( ) .callStuff ( );
6.     }
7.     void callStuff ( ) {
8.         System.out.print( "this" +this.doStuff( ) );
9.         ParentUtil p = new ParentUtil ( );
10.        System.out.print( "parent" +p.doStuff ( ) );
11.    }
12.    }

```

下列哪一個敘述是正確的？

- A. 程式碼可以編譯與執行，而且會產生 this 420 parent 420 的結果。
- B. 只要移除第 8 行，程式碼便可編譯與執行。
- C. 只要移除第 10 行，程式碼便可編譯與執行。
- D. 第 8 行與第 10 行都必須移除，否則程式無法正常編譯。
- E. 在執行期間會拋出例外。

解：C，無法用 ParentUtil 的物件實體 p 來存取 doStuff ( ) 方法，因為 doStuff ( ) 具有 protected 存取權限，而 ChildUtil 類別與 ParentUtil 類別分別位於不同的套件中，因此只能透過 ChildUtil 類別的實體存取 doStuff ( )。

7.

```

1. interface Count {
2.     short counter = 0;
3.     void countUp( );
4. }
5. public class TestCount implements Count {
6.
7.     public static void main(String [] args) {
8.         TestCount t = new TestCount ( );
9.         t.countUp( );
10    }
11.     public void countUp( ) {
12.         for (int x = 6; x> counter; x--, ++ counter) {

```

```
13.         system.out.point( “ ” + counter);
14.     }
15. }
16. }
```

會產生怎樣的結果？

- A. 0 1 2
- B. 1 2 3
- C. 0 1 2 3
- D. 1 2 3 4
- E. 編譯失敗
- F. 在執行期間拋出例外

解：E 為正解。

由於 counter 變數是介面變數，其預設為 final static，因此程式碼無法編譯。

當第 2 行的程式碼在增加 counter 時，編譯器便會發生錯誤。

8. 若

```
1. import java.util.*;
2.     public class NewTreeSet2 extends NewTreeSet {
3.         public static void main(String [] args) {
4.             NewTreeSet2 t = new NewTreeSet2( );
5.             t.count( );
6.         }
7.     }
8.     protected class NewTreeSet {
9.         void count( ) {
10            for (int x = 0; x < 7; x++, x++) {
11                system.out.point( “ ” + x);
12            }
13        }
14    }
```

會產生怎樣的結果？

- A. 0 2 4
- B. 0 2 4 6
- C. 第 4 行編譯失敗
- D. 第 5 行編譯失敗
- E. 第 8 行編譯失敗
- F. 第 10 行編譯失敗

解:E. 為正解。

非巢狀的類別無法宣告為 protected(或 final)，因此編譯在第 8 行時會發生錯誤。

\*內部類別指的是非 static 的巢狀類別。

巢狀類別分為兩種，一種是 static 的，稱為 static 巢狀類別；另一種是非 static 的，稱為內部類別。

\*protected：只有繼承的子類別可使用，不同套件也可以(可用於變數方法)

9. 若

```
1.
2.     public class NewTreeSet extends java.util.TreeSet {
3.         public static void main(String [] args) {
4.             java.util.TreeSet t = new java.util.TreeSet ( );
5.             t.count( );
6.         }
7.     public void clear( ) {
8.         TreeMap m = new TreeMap( );
9.         m.clear( );
10        }
11.    }
```

則必須在第一行加入下列哪個敘述，程式碼才能正常編譯？(選擇兩項)

- A. 以下皆非
- B. Import java.util.\*;
- C. Import java.util.Tree\*;
- D. Import java.util. TreeSet;
- E. Import java.util. TreeMap;

解:B. 與 E. 為正解。

TreeMap 是唯一必須匯入的類別。由於 TreeSet 使用完整名稱，因此不需要匯入敘述。

A. 則是錯在它必須匯入 TreeMap。

D. 也是沒匯入 TreeMap。

C. 則是匯入的語法錯誤。

10. 下列哪兩個是介面中正確的宣告?(選出兩項)

A. `public static short stop = 23;`

B. `protected short stop = 23;`

C. `transient short stop = 23`

D. `final void madness(short stop);`

E. `public Boolean madness(long bow);`

F. `static char madness(double duty);`

解:A. 與 E. 為正確的介面宣告。

B.、C.、D. 與 F. 為錯誤答案。protected 與 transient 不可為介面變數。final 與 static 不可為介面方法。

11. 下列哪一個類別存取權限的(非區域)變數宣告會導致編譯失敗?

A. `protected int a;`

B. `transient int b = 3;`

C. `public static final int c;`

D. `volatile int d;`

E. `private synchronized int e;`

解:E 無法編譯。synchronized 只能用在方法上

### 3. 運算子與指定運算

#### 3.1 基本運算子

##### 3.1.1 指定運算子

##### 3.1.2 條件運算子

##### 3.1.3 instanceof 比較

##### 3.1.4 算數運算子

##### 3.1.5 位移運算子

##### 3.1.6 位元運算子

##### 3.1.7 位元補數運算子

##### 3.1.8 基本資料型別強制轉型

#### 3.2 邏輯運算子

##### 3.2.1 短迴路邏輯運算子

##### 3.2.2 邏輯運算子(非短路運算子)

#### 3.3 方法的基本定義

#### ◎ 考題分析



## 3.1 基本運算子

判定對各種型別、類別、範圍、可存取性或任一組合的運算元使用運算子。Java 運算子會從一個或數個運算元產生新的數值。大多數運算結果不是布林值就是數字。當+運算子套用在String上時，它會連接右邊與左邊的運算元。

### 3.1.1 指定運算子

指定運算子的功能就是將數值（或運算式結果）指定給變數，讓數值存放在變數所佔用的記憶體內。不過在程式裡「=」不是相等而是指定的意思，「=」的左邊一定是變數，不能是數值。變數只是位元的容器，它具有特定的型別。你可以宣告成int、double、Button，甚至String[ ]。

#### 指定運算子

當你替變數指定數值時，必須用到等號( = )。等號又稱為：指定運算子。目前有 12 種指定運算子，但其它 11 種都是等號和其它數學運算子的組合，如下表所示，這些稱：複合指定運算子，有隱含的強制型別轉換。

=	*=	/=	%=
+=	-=	<<=	>>=
>>>=	&=	^=	=

這邊可以使用字面常數或運算式的結果來指定基本資料型別變數，如下：

```
int x = 7;    //使用字面常數指定
int y = x + 2; //使用運算式指定（包含文字常數）
int z = x + y; //使用運算式指定
```

以上，字面常數(假設常數值 7)會自動轉成 int。int 是 32 位元值，你可以將數值指定給 int 或 long 變數，但是如果將數值指定給 byte 變數，則會發生錯

誤，畢竟，byte 比 int 小，因此無法容納跟 int 一樣大位元數的整數。例子如下：

```
byte b = 27;
```

以上編譯器會強制轉型，如下：

```
byte b = (byte) 27; //自行將 int 文字常數轉成 byte
```

這樣編譯器，就可正常執行。除了文字常數永遠都會是 int 外，等於或小於 int 的運算式運算結果也會是 int。所以，如果我們可以自行轉型，編譯器便可正常編譯。

### 指定浮點數

浮點數的指定與整數型別有些不同。首先，每個浮點常數都會自動轉換成 double(64 位元)，而不是 float 型別。以 float f = 32.3; 來看，32.3 表面上看來應該可以放入 float 大小的變數中，但編譯器不允許你這樣做。若要將浮點常數指定給浮點變數，你必須轉換數值或在文字常數後面加上 f。如下：

```
float f = (float) 32.3;  
float g = 32.3f;  
float h = 32.3F;
```

### 替變數指定過大的文字常數

如果替變數指定一個過大的的數值，編譯器也會發生錯誤。如下：

```
byte a = 128; //byte 最多只能容量 127
```

但可以透過轉換來修正此問題。如下：

```
byte a = (byte) 128;
```

將某基本資料型別變數指定給另一個基本資料型別變數

當你將某基本資料型別變數指定給其它基本資料型別變數時，你必須複製右邊的內容，如下：

```
int a = 6;  
int b = a;
```

此程式碼可讀做：「將數字 6 的位元樣式指定給 int 變數 a。」這會複製 a 的位元樣式並將它放入變數 b 中。

指定參考變數

你可以將剛建立的物件指定給物件參考變數，如下：

```
Button b = new Button( );
```

此段程式碼會執行下列動作：

- 將參考變數命名為 b，且屬於 Button 型別
- 在堆疊中建立新的 Button 物件
- 將剛建立的 Button 物件指定給參考變數 b

你也可以將 null 指定給物件參考變數，這表示變數不會參照至任何物件：

```
Button c = null;
```

此程式碼會替 Button 參考變數製造空間(參考值的位元容器)，但不會建立真正的 Button 物件。

將某參考變數指定給其它參考變數

有了基本資料型別變數，當你將某變數指定給其它變數時，表示將變數的內容複製到其它變數中。物件參考變數也一樣。參考變數的內容也是位元樣式，當你將參考變數 a 指定給參考變數 b 時，a 的位元樣式便複製到 b 中。如果將物件既有的物件實體指定給新的參考變數，這兩個參考變數便具有相同的位元樣式，即指向堆疊中特定物件的位元樣式。

### 3.1.2 條件運算子

比較運算子永遠只會產生布林值(true 或 false)。布林值最常出現在 if 測試中，如下：

```
int x = 8;
if (x < 9) {
    //執行某件工作
}
```

但是最後的數值也可以直接指定給布林基本資料型別。

以下比較運算子可供比較整數、浮點數或字元的組合時使用：

- > 大於
- >= 大於等於
- < 小於
- <= 小於等於
- == 等號 (或稱「等於」)
- != 不等號 (或稱「不等於」)

運算子會比較兩邊的物件，並回傳布林值。

以下四種型別是可以互相比較的：

- 數字
  - 字元
  - 布林基本資料型別
  - 物件參考變數
- 基本資料型別的等號

大多數程式都知道比對基本資料型別值。以下程式碼測試與基本資料型別變

數上的等號有關：

```
class ComparePrimitives{
    public static void main (String [ ] args) {
        System.out.println( "5.0 == 5L? " + (5.0 == 5L));
    }
}
```

結果如下：

```
5.0 == 5L? true //比對浮點與 int
```

若比較浮點數與整數，會發現兩個值相同，而==運算子應回傳 true。

參考變數的等號

兩個參考變數可以參照至同一物件，如下：

```
Button a = new Button( "Exit" );
Button b = a;
```

在執行此程式碼後，變數 a 與變數 b 便會指向同一物件(標示 Exit 的 Button)。你可以使用==運算子來測試參考變數，以判定它們是否指向同一物件。

### 3.1.3 instanceof 比較

Instanceof 運算子只能用在物件參考變數。你可以藉此檢查物件是否屬於特定型別。此處的型別是指類別或介面型別，也就是說，運算子左邊變數所參照的物件是否通過右邊類別或介面型別的 IS-A 測試(這邊 IS-A 會在後面做介紹)。

即使測試的物件並不是運算子右邊類別的實體化，若將物件的指定與右邊的项目相互比對，則 instanceof 仍會傳回 true。範例如下：

```
class A {
class B extends A {}
    public static void main (String [ ] args)
        B b = new B( );
```

```
    if (b instanceof A) {
        System.out.print( "b is an A");
    }
}
```

由以上程式碼可知，b 是一個 a。因此你可以根據物件參考的類別型別或超類別來測試物件參考。這表示當你再 Object 型別中使用運算子時，所有物件參考的結果都會是 true。

### 3.1.4 算數運算子

以下有 4 個基本的算術運算子：

- + 加
- - 減
- \* 乘
- / 除

還有一個 % 餘數運算子，餘數運算子會將左邊的運算元除以右邊運算元，結果就是餘數。

算術運算子有一個特別要注意的地方，當除數為 0 時會有以下規則：

- 將整數除以 0 會違反熱力學的重要規則，並產生 ArithmeticException(無法除以 0)。
- 右邊的運算元為 0，則使用餘數運算子(%)會產生 ArithmeticException(無法除以 0)。
- 浮點數除以 0，不會產生 ArithmeticException，而且所有內容都會保持不變。
- 浮點數的右邊運算元為 0，則用餘數運算子不會產生

ArithmeticException。

### 字串連接運算子

字串部分，加號可以用來連接兩個字串：

```
String animal = "Grey " + "elephant" ;
```

當你將含有 String 數字連接在一起時，會有以下特點：

```
String a = "String" ;  
int b = 3;  
int c = 7;  
System.out.println(a + b + c);
```

以上這段程式，結果是 String37。以 a 字串開始，先是字串，然後加一個 3(就是 b)，形成新的字串「String3」，然後再加一個 7(就是 c)，即成新字串「String37」，最後產生此字串。

如果在程式中加入兩個括號，如下：

```
System.out.println(a + (b + c));
```

 會有 String10 此結果

使用括號會使程式碼先計算(b + c)，得此結果。

### 遞增與遞減

Java 有兩個運算子，分別會增加 1 與減少 1，這兩個運算子是由兩個加號(++)

或兩個減號(-- )所組成：

- ++ 加 1 (前置與後置)
- -- 減 1 (前置與後置)

運算子位於變數前面或後面，不論運算子在運算元前面或後面，它都一定會改變運算式的結果。

### 3.1.5 位移運算子

以下是位移運算子：

- `>>` 右移
- `<<` 左移
- `>>>` 無符號右移 (zero-filled 右移)

位移運算子會將數字移到左邊或右邊，而產生新數字。位移運算子只有再整數時才會用到(而不是浮點數)。要判定位移的結果，你必須將數字轉換成二進位制。以下是例子：

```
8 >> 1;
```

首先，將此數字轉成二進位制：

```
0000 0000 0000 0000 0000 0000 0000 1000
```

`int` 是 32 位元數因此必須顯示所有 32 位元。若使用 `>>` 運算子，讓位元向右移動一位，則新的位元為：

```
0000 0000 0000 0000 0000 0000 0000 0100
```

此時，位元已經右移一個位置。

不管整數值的基底為何(八進位、十進位或十六進位)，所有整數數值都可以使用在位移運算。左移也一樣，不過位元是向左移動。若使用右移運算子(`>>`)移動負數的位元，則符號位元向右移，此時會使用該符號位元填補最左邊的空位。因此最下面一行是「在右移運算子(`>>`)中，負數仍為負數。」下面以 16 位元進位值來當例子：



1000 0000 0000 0000 0000 0000 0000 0000

現在使用 >> 將位元向右移動一個位置

1100 0000 0000 0000 0000 0000 0000 0000

符號位元也跟著右移一格，而最左邊的位元則使用原本的符號位元補上。以

下是程式碼：

```
class BitShift {
    public static void main (String [ ] args) {
        int x = 0x80000000;
        System.out.println( "Before shift x equals " +x);
        int = x >> 4;
        System.out.println( "After shift x equals " +x);
    }
}
```

以上程式，第五行的地方會向右移 4 格，執行結果為：

Before shift x equals -2147483648

After shift x equals -134217728

下面適用位元表示數字的結果：

1111 1000 0000 0000 0000 0000 0000 0000

這邊左邊的四個右移的位元已經使用原本的符號補上了。

若不想保留符號位元，可以使用特殊的位移運算子 >>> (無符號右移運算子)。下面是程式碼：

```
class BitShift {
    public static void main (String [ ] args) {
        int x = 0x80000000;
        System.out.println( "Before shift x equals " +x);
    }
}
```

```
x >>>= 4;
System.out.println( "After shift x equals " +x);
}
}
```

會產生下面的結果：

```
Before shift x equals -2147483648
After shift x equals 134217728
```

由於負數位元已經不存在，因此新數字為正數，下面是新數字的表示法：

```
0000 1000 0000 0000 0000 0000 0000 0000
```

上面這邊原符號為 1，但新的數字變成了 0，這也是無符號位移的結果，不管原本的符號位元是什麼，若要让數字永遠為正，就必須使用 `>>>` 而不是 `>>`。

### 3.1.6 位元運算子

位元運算子會用到兩個位元，然後根據各個位元使用 AND / OR 來判定結果，以下是位元運算子：

- `&` AND
- `|` OR
- `^` Exclusive OR

`|` 運算子跟 `&` 運算子不同。只有當兩個運算元都是 0 的時候相對應的執會是 0，也就是說，當運算原有 1 時，OR 計算的結果一定會是 1。`^` 運算子會比較兩位元並判定兩個位元是否不同，若是兩個不同位元，則結果會是 1。`&` 運算子會比較兩個數字對應的位元，若兩個位元都是 1，最後位元表示一定是 1。

下面是計算時的真值表：

X	Y	& (AND)	(OR)	^ (XOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

### 3.1.7 位元補數運算子

運算子是位元補數運算子。它會將所有 1 變成 0。如下面程式：

```
class Bitwist {
    public static void main (String [ ] args) {
        int x = 5;
        System.out.println( "x is initially " +x);
        x >>>= 4;
        System.out.println( "x is equal to" +x);
    }
}
```

程式結果如下：

```
x is initially 5
x is equal to -6
```

以位元表示：

```
~0000 0000 0000 0000 0000 0000 0000 0101
```

會轉換成：

```
1111 1111 1111 1111 1111 1111 1111 1010
```

### 3.1.8 基本資料型別強制轉型

轉型可以分為「隱式轉型」或「顯式轉型」。自動轉型表示你不需要編寫轉

型的程式碼，它會自動轉換，通常在進行放大轉換時便會自行進行隱式轉型。就是說，將較小的東西放到較大的容器中，前面有提過，如果將 int 放到 byte 中，將會發生錯誤。而這樣就必須執行顯式轉型，告訴編譯器，你確定要這麼做。以下是範例：

```
int a = 100;
long b = a //隱式，int 值可以放在 long 前面

float a = 100.001;
int b = (int) a; //顯式，float 變成 int 時可能會遺失一些資訊
```

## 3.2 邏輯運算子

目前有四種邏輯運算子。前面有介紹過 & 跟 | 位元運算子可用在布林運算式中，另外現在要提到的則是短路邏輯運算子：

- && 短路 AND
- || 短路 OR

### 3.2.1 短迴路邏輯運算子

&& 運算子跟 & 運算子相同，不過它只會計算布林值，而且無法當作位元運算子使用，記住，如果要讓 AND 運算式結果為 true，則兩個運算元都必須為 true。如下面例子：

```
if ((2 < 3) && (3 < 4)) { }
```

因為前面  $2 < 3$  是正確的，後面  $3 < 4$  也是正確的，所以結果會是 true。

`||` 運算子和 `&&` 運算子一樣，會先運算左邊，並檢查是否為 true，若 OR 運算時第一個位元是 true，結果一定是 true，此時短路的 `||` 邏輯運算元，就不會繼續執行下面的動作，若第一個運算元為 false，`||` 運算元會繼續運算下面的運算元。

### 3.2.2 邏輯運算子（非短迴路運算子）

`&` 與 `|` 這兩個位元運算子也可以用在邏輯運算式中，不過因為它們不是短路運算子，因此必須運算運算式的兩邊，這樣很沒有效率。所以說，短路運算子（`&&` 與 `||`）只能用在邏輯（非位元）運算式中。位元運算子（`&` 與 `|`）可以用在邏輯和位元運算式中，不過因效率不佳，因此很少在邏輯運算式中使用。

## 3.3 方法的基本定義

定義方法(method)的注意事項：

- 必須宣告有回傳資料型別的宣告；若方法執行後無法回傳任何值，就必須宣告為 void。
- 若回傳資料型別宣告，必須利用 `return xyz` 來傳回執行結果。xyz 的資料型別必須和方法的回傳資料型別的宣告匹配。
- 方法內部可以搭配條件判斷與 `return`；(或 `return xyz`) 提早結束方法的執行。

- return 之後的指令若沒有機會執行，會有編譯錯誤的情況發生。
- 區域變數：方法內部可以宣告暫時需要的變數以儲放臨時的資料，此變數稱之為區域變數(Local variable , automatic variable)。
  - 區域變數的有效範圍：在宣告的區塊內是可見的，離開區塊後就消失了（由 stack 堆疊區所配置）。
  - 區域變數不可重複宣告：
    - 也就是同一方法內的區域變數名稱要是唯一的。
  - 可以和類別成員的名稱相同：
    - 但是，若有相同變數名稱時，存取類別成員變數要記得加上 this，否則，無法得到正確的結果。

◎ 考題分析

1. 下列哪兩項是相等的？（選出兩項）

- A.  $32 / 4$  ；
- B.  $(8 \gg 2) \ll 4$  ；
- C.  $2 \wedge 5$  ；
- D.  $128 \ggg 2$  ；
- E.  $(2 \ll 1) * (32 \gg 3)$  ；
- F.  $2 \gg 5$  ；

解：答案 B. 跟 D. 是相等的。

- A.  $32/4=8$  B. 8 右移 2=2，在左移 4=32 C. 2 跟 5 做 XOR 運算=7
- D. 128 做無符號右移 2 =32 E. 2 左移 1\*32 右移 3 =16 F. 2 右移 5 =0

2. 若

- 1. `import java.awt.*;`
- 2. `class Ticker extends Component {`
- 3. `public static void main (String [] args) {`
- 4. `Ticker t = new Ticker( ) ;`
- 5.
- 6. `}`
- 7. `}`

此程式碼的第 5 行可分別插入下列哪兩個敘述？（選出兩項）

- A. `boolean test = (Component instanceof t) ;`
- B. `boolean test = (t instanceof Ticker) ;`
- C. `boolean test = t instanceof (Ticker) ;`
- D. `boolean test = (t instanceof Component) ;`
- E. `boolean test = t instanceof (Object) ;`
- F. `boolean test = (t instanceof String) ;`

解：答案 B. 跟 D. 是正確的。instanceof 正確的基本宣告方式（變數 instanceof 變數），也可以測試 null 值（null instanceof 變數），假設，程式中有 `null instanceof String` 其中的 String 會被當作物件類別，此時 instanceof 會回傳 false。

綜合以上幾點，A. C. E. F. 是錯誤的。

3. 若

```
1. class Equals {  
2.     public static void main (String [] args) {  
3.         int x = 100 ;  
4.         double y = 100.1 ;  
5.         Boolean b = (x = y) ;  
6.         System.out.println (b) ;  
7.     }  
8. }
```

會產生什麼結果？

- A. true
- B. false
- C. 編譯失敗
- D. 在執行期間丟出例外

解：答案是 C。編譯失敗，其中程式第 5 行處， $x = y$  的用法只會替變數產生數值，可將程式改為  $x == y$  才會回傳布林值。

所以，A. B. D. 都是錯誤的答案，如果將程式第 5 行處改成  $x == y$ ，程式將可成功執行，並秀出 false。

4. 若

```
1. import java.awt.Button ;  
2. class CompareReference {  
3.     public static void main (String [] args) {  
4.         float f = 42.0f ;  
5.         float [] f1 = new float [2] ;  
6.         float [] f2 = new float [2] ;  
7.         float [] f3 = f1 ;  
8.         long x = 42 ;  
9.         f1 [0] = 42.0f ;  
10.    }  
11. }
```



下列哪三個是正確的敘述？（選出三項）

- A. `f1 == f2`
- B. `f1 == f3`
- C. `f2 == f1 [1]`
- D. `x == f1 [0]`
- E. `f == f1 [0]`

解：正確答案是 B. D. 跟 E. 。B. `f1` 跟 `f3` 參考同一陣列的物件，所以是正確的。

D. 整數跟浮點數型別比較，可以，是正確的。E. 整數也可以跟陣列元素比較，所以也是對的。C. 錯誤因為 `f2` 是陣列物件，`f1[1]`是陣列元素，無法做比較。

5. 若

```
1. class BithShift {
2.     public static void main (String [ ] args) {
3.         int x = 0x80000000;
4.         System.out.print (x + " and ");
5.         x = x >>> 31;
6.         System.out.println(x);
7.     }
8. }
```

會產生什麼結果？

- A. -2147483648 and 1
- B. 0x80000000 and 0x00000001
- C. -2147483648 and -1
- D. 1 and -2147483648
- E. 以上皆非

解：為 A，一開始為 1000 0000 0000 0000 0000 0000 0000 0000 所以第 4 行印

出值為-2147483648，>>>經過此運算符號後，所有位元將會向右移 31 個位元，並使用 0 填補最左邊的位元也就是(符號的位子)，因此將會變成

0000 0000 0000 0000 0000 0000 0000 0001，因此第 6 行的 x 值為 1。

6. 若

```
1. class Bitwise {
2.     public static void main (String [ ] args) {
3.         int x = 11 & 9;
4.         int y = x ^ 3;
5.         System.out.println( y | 12 );
6.     }
7. }
```

會產生什麼樣的結果？

- A. 0
- B. 7
- C. 8
- D. 14
- E. 15

解：答案為 D.  $1011 \& 1001 = 1001 \Rightarrow 9 = x$

$y = 9 \wedge 3 \Rightarrow 1001 \wedge 0011 \Rightarrow 1010 = 10$

$10 | 12 = 1010 | 1100 = 1110 = 14$

7. 下列何者是正確的語法？(選出所有正確的選項)

- A. `int w = (int)888.8;`
- B. `byte x = (byte)1000L;`
- C. `long y = (byte)100;`
- D. `byte z = (byte)100L;`

解：此為指定基本資料型別值題型，A. 將 double 強制轉為 int 小數點將會不見，

B. 將 long 強制轉為 byte 但是超過 long 127 將會遺失最左邊的位元，C. 不用轉型因為 long 可以容納 byte，D. 和 B. 一樣只是沒超過 long 127。

8. 若

```
1. class Test {
2.     public static void main(String [ ] args) {
3.         int x = 0;
```

```

4.     int y = 0;
5.     for ( int z = 0; z < 5; z++) {
6.         if (( ++x > 2 ) || ( ++y > 2 )) {
7.             x++;
8.         }
9.     }
10.    System.out.println ( x+ " " + y );
11.    }
12. }

```

會產生什麼樣的結果？

- A. 5 3
- B. 8 2
- C. 8 3
- D. 8 5
- E. 10 3
- F. 10 5

解：答案是 B.，第 5 行迴圈，所以 z 值從 0 開始，z++(執行過後 z 再+1)，所以執行到 z=5 時就停止。第 6 行第一次運算左邊此時 x=1 不符合所以為 false 運算左邊此時 y=1 也不符。z=1, x=2 為 false, y=2 為 false。z=2, x=3 為 true, y 不執行跳第 7 行 x=4。z=3, x=5, y=2, 第 7 行 x=6。z=4, x=7, y=2, 第 7 行 x=8, z=5 停止迴圈。所以最後結果 x=8 y=2。

9. 若

```
1. class Test {
2.     public static void main (String [] args) {
3.         int x = 0;
4.         int y = 0;
5.         for (int z = 0; z < 5 ; z++)
6.             if (( ++x >2 ) && ( ++y >2 )) {
7.                 x++;
8.             }
9.         }
10.    system.out.println( x + " " + y);
11. }
12. }
```

會產生怎樣的結果？

- A. 5 2
- B. 5 3
- C. 6 3
- D. 6 4
- E. 7 5
- F. 8 5

解：C. 在前面兩迴圈中，x 會遞增一次，而&&運算子會不讓 y 遞增。

在第三次與第四次迴圈中，x 與 y 都會遞增。

而在第五次迴圈，x 會遞增兩次，而 y 只會遞增一次。

10. 若

```
1. class SSBool {  
2.     public static void main (String [] args) {  
3.         boolean b1 = true;  
4.         boolean b2 = false;  
5.         boolean b3 = true;  
6.         if ( b1 & b2 | b2 & b3 | b2 )  
7.             system.out.println( "ok " );  
8.         if ( b1 & b2 | b2 & b3 | b2 | b1 )  
9.             system.out.println( "dokey " );  
10.    }  
11. }
```

會產生怎樣的結果？

- A. ok
- B. dokey
- C. ok dokey
- D. 無輸出
- E. 編譯錯誤
- F. 執行期間丟出例外

解：B.&的運算子比 | 運算子來的優先。

因此第 6 行的 b1 跟 b2 會當成 b2 & b3 來做。

第 8 行的 b1 是導致 if 會是 true 的原因。

## 4. 流程控制、例外與斷言(Assertions)

- 4.1 使用 if 與 switch 編寫程式碼
    - 4.1.1 if-else
    - 4.1.2 switch
  - 4.2 使用迴圈寫程式碼
    - 4.2.1 while 迴圈
    - 4.2.2 do-while 迴圈
    - 4.2.3 for 迴圈
    - 4.2.4 在 for 迴圈中使用 break 與 continue
    - 4.2.5 無標籤敘述
    - 4.2.6 帶標籤的敘述
  - 4.3 例外處理機制
    - 4.3.1 使用 try 與 catch 建立例外
    - 4.3.2 finally
    - 4.3.3 傳遞未捕捉(catch)到的例外
    - 4.3.4 定義例外
    - 4.3.5 例外層級
    - 4.3.6 處理例外類別的階層
    - 4.3.7 例外對應(Exception matching)
    - 4.3.8 例外宣告與公用介面
    - 4.3.9 丟出相同例外
  - 4.4 Assertion(斷言)機制
    - 4.4.1 Assertions 概述
    - 4.4.2 Assertion 語法
    - 4.4.3 Assertion 應用
- ◎ 考題分析

此章節主要教你使用 if 及 for 迴圈的應用，而 Java 提供許多基本流程控制的指令，當然，Java 也有其它機制—例外(exception)還有 assertion，此章節都會有詳細的敘述，包含考試時常出現的應用。

if 與 switch 都是屬於條件/判斷的方法，可以根據邏輯測試的結果來決定使用哪一種方式。Java 可使用三種迴圈—for、while 與 do-while，若條件判斷為 true，則重複執行相同的程式碼。

而「例外處理機制」可以更簡單地編寫異常處理程式碼，用來解決執行期間出現的問題。

Assertion 功能就是用來解決編寫程式碼時，將可能出現問題的條件進行偵錯檢查。

## 4.1 使用 if 與 switch 編寫程式碼

使用 if 與 switch 編寫程式碼並找出這些敘述的有效條件，if 與 switch 通常被稱為判斷/決策(decision)。注意：需在程式中使用決策時，必須先評估特定的運算式，找出要採用 if 或是 switch 來編寫程式碼。

### 4.1.1 if-else

if 的基本格式：

1. if (booleanExpression) {
2. System.out.println("Inside if statement");
3. }

第一行小括弧中的運算式必須是 true 或是 false 的條件布林值。一般的做法是測試某個條件，判定它是否為 true，如果為 true 則執行某程式碼區段(此

包含一個或數個敘述)，如果不是 true 則執行另一個程式碼區段(選擇性的執行)。

下面為正確的 if 敘述：

```
1. if (x>3) {
2.   System.out.println("x is greater than 3");
3. } else {
4.   System.out.println("x is not greater than 3");
5. }
```

else 區段是選擇性的，因此也可使用下列程式碼：

```
1.  if(x >3) {
2.    y = 2;
3.  }
4.    z += 8;
5.    a = y + x;
```

上述如果 x 的確大於 3，則前一段的程式碼會指派 2 給 y，但不管測試結果怎樣，其它兩行程式碼都會執行，若條件式區塊的主體只有一個敘述需要執行，則可以省略大括弧「{}」。

以下是含有兩個條件的程式碼，當第一個測試成功時，先執行第二個測試再決定後續動作：

```
1. if (price < 300) {
2.   buyProuct( );
3. } else {
4.   if (price < 400) {
5.     getApproval( );
```



```
6. }
7. else {
8.     dontBuyProduct( );
9. }
10. }
```

以下例子可能會不知道哪個 if 搭配哪個 else：

```
1. if (exam.done ( ))
2. if (exam.getScore ( ) < 0.61)
3. System.out.println("Try again.");
4. else System.out.println("Java master!"); //此行 else 對應第 2 行
if
```

以上程式碼 else 對應於最靠近自己的 if 敘述。

以下是使用大括弧的一般表示法，此種敘述較不易混淆：

```
1. if (exam.done( )) {
2.     if (exam.getScore( ) <0.61) {
3.         System.out.println("Try again.");
4.     } else { //此行 else 對應第 2 行 if
5.         System.out.println("Java master!");
6.     }
7. }
```

### 敘述的合法參數

只能根據一個條件布林值來測試 if 敘述。任何能夠解析成布林值的運算式都是有效的運算式。下列程式碼，doStuff( ) 傳回的是 true

```
int y = 5;
int x = 2;
if (((x > 3) && (y < 2)) | doStuff( )) {
    System.out.println("true");
}
```

結果會是 true。

由上列程式碼得知如果(x>3)與(y<2)為 true，或是 doStuff( )的結果為 true，則印出 true，所以只有 doStuff( )為 true，還是會印出 true，如果 doStuff( )為 false，則(x>3)與(y<2)必須為 true 才能印出 true。

在指定布林變數以測試變數值時，需注意：

```
boolean boo = false;
if (boo = true) { }
```

上述程式碼可正常編譯與執行，由於 if 參數中的 boo 設定為 true(而不是測試是否為 true)，因此 if 測試成功。

在指定變數時只能指定布林值，若指定其它非布林值的結果會無法編譯。例如：

```
int x = 3;
if (x = 5) { } //由於 x 不是布林值所以無法編譯
```

#### 4.1.2 switch

以下為 if-else 較為複雜的程式碼：

```
int x = 3;
if(x == 1) {
    System.out.println("x equals 1");
}
else if (x == 2) {
    System.out.println("x equals 2");
}
else if (x == 3) {
    System.out.println("x equals 3");
}
else {
```

```

    System.out.println("No idea what x is");
}

```

以下使用 switch 來傳達相同功能的程式碼：

```

int x = 3;
switch (x) {
    case 1 :
        System.out.println("x is equal to 1");
        break;
    case 2 :
        System.out.println("x is equal to 1");
        break;
    case 3 :
        System.out.println("x is equal to 1");
        break;
    default :
        System.out.println("Still no idea what x is");
}

```

**if 的參數用法：**

if 的非法參數	if 的合法參數
int x = 1; if(x) { }	int x = 1; if(x == 1) { }
if(0) { }	if(false)
if(x = 6)	if(x == 6 )

### switch 與 case 的合法參數

switch 只支援與 int 相容的基本類型，這表示它只接受能夠自動轉換成 int 的類型：byte、short、char、int，如果裡面有其它類型像是 long、float 與 double 等類型，這些會使程式碼無法成功編譯。

case 支援的參數和 switch 屬於同一個類型，不過還有一個限制，case 參數必須為 final，由於 case 參數必須在編譯期間解析，因此，只能使用「一個常數」或是 final 變數。

### switch 與 case 的用法

```
final int a = 1;
final int b;
int x = 0;
switch(x) {
    case a:
    case b: //此處編譯錯誤，case 只接受一個常數或是 final 變數
}
```

此外，switch 只會檢查等式，因此，「大於」之類的關係運算子都無法在 case 中使用。

以下為 switch 敘述的有效運算式，注意：如果要建立有效的程式碼，物件參考需回傳一個與 int 相容的數值。

```
String s = "xyz";
switch (s.length( )) {
    case 1:
        System.out.println("length is one");
        break;
    case2 :
        System.out.println("length is two");
        break;
    case3 :
        System.out.println("length is three");
        break;
    default :
        System.out.println("no match");
}
```

以下範例在 case 中使用了 final 變數，注意：省略了 final 關鍵字，程式則無法成功編譯。

```
final int one = 1;
final int two = 2;
int x = 1;
switch (x) {
    case one : System.out.println("one");
                break;
    case two : System.out.println("two");
                break;
}
```

若使用比 int 小的變數，則無法編譯：

```
byte g = 2;
switch(g) {
    case 23:
    case 128: //此參數對 byte 太大，所以無法編譯
}
```

另外，每個 case 的標籤都必須使用不同的數值，例如：不得將兩個 case 的值都設為 80，會使程式碼編譯錯誤。以下為考試時常出現的無效程式碼：

```
Integer in = new Integer (4);
switch (in) { } //無法轉換 Integer 物件，只能使用 int 物件。
```

```
switch (x) {
    case 0 { //case 使用了大括弧且省略了冒號。
        y = 7;
    }
}
```

```
switch (x) {
    0:{ } //此程式碼省略了 case 關鍵字
    1:{ }
}
```

**switch 中的 default、break 與 fall-through**

若 switch 敘述中遇到 break 關鍵字，則程式會立刻跳出該 switch 區段並立刻執行後方的敘述，如果沒有 break 關鍵字，則程式會繼續執行不同的 case 區段，直到遇到 break 或是 switch 結束。

```
int x = 1;
switch(x) {
    case 1: System.out.println("x is one");
    case 2: System.out.println("x is two");
    case 3: System.out.println("x is three");
}
System.out.println("one of the switch");
```

此程式會產生下列結果

```
x is one
x is two
x is three
out of the switch
```

因為程式碼中沒有 break，因此，會執行每個 case 直到該敘述結束，此結果稱為 fall-through。如果只需要其中一段符合的程式，則必須在程式中參入 break：

```
int x = 1;
switch (x) {
    case 1:
        System.out.println("x is one");
        break;
    case 3:
        System.out.println("x is three");
        break;
}
System.out.println("out of the switch");
```

此程式碼結果為：

```
x is one  
out of the switch
```

上述結果：case 1 進入 switch 區段，由於它與 switch() 參數一致，因此，得到 println 敘述，接著遇到 break，並立即跳到 switch 的尾端。

另一個思考 fall-through 的方法如下：

```
int x = someNumberBetweenOneAndTen;  
switch (x) {  
    case 2 :  
    case 4 :  
    case 6 :  
    case 8 :  
    case 10 : {  
        System.out.println("x is an even number");  
        break;  
    }  
}
```

此 switch 程式碼根據數字介於 1 與 10，且為奇數或偶數產生「x is an even number」的結果或沒有結果，若 x 為 4 則會從 case 4 開始執行，然後執行 6、8 與 10，它會執行這些區段時產生結果，之後中斷執行並跳到下一個區段執行，因此，case 10 已經是 switch 敘述的尾端了，則不需要 break 了。

### default case

在執行上個範例的程式碼時，若沒有符合的 case 偶數，又必須產生「x is an even number」的結果，此時就必須使用 default 關鍵字。如下範例：

```
int x = someNumberBetweenOneAndTen;  
switch (x) {  
    case 2 :  
    case 4 :
```

```
case 6 :
case 8 :
case 10 :{
    System.out.println("x is an even number");
    break;
}
default : System.out.println("x is odd number");
}
```

default case 不一定會位於 switch 的尾端，它也可以在其它位置出現：

```
int x = 2 ;
switch (x) {
    case 2 : System.out.println( "2" );
    default : System.out.println( "default" );
    case 3 : System.out.println( "3" );
    case 4 : System.out.println( "4" );
}
```

此程式碼會產生：

```
default
3
4
```

## 4.2 使用迴圈寫程式碼

使用各種迴圈寫程式碼，包含帶標籤(labeled)與無標籤(unlabeled)及使用 break 與 continue 的應用。Java 的迴圈有三種:while、do-while 與 for，只要執行條件結果為 true，這三種迴圈便會重複執行此段程式碼或執行限定重複次數的程式碼。



### 4.2.1 while 迴圈

若不知道要重複幾次程式碼區段的迴圈，但又只希望某條件為 true 便重複該區段，這時可選擇使用 while 迴圈。以下為 while 程式碼：

```
int x = 2;
while (x == 2) {
    System.out.println(x);
    ++x;
}
```

此處和所有迴圈都一樣，運算式測試的結果都必須為布林值，其中在 while 迴圈的運算式中，所有變數須在運算式開始前宣告完成。只有當條件為 true 時，while 迴圈才會執行，迴圈便會執行到條件為 false 時停止。

while 迴圈的重點是它可能永遠都無法執行，因此，while 運算式的結果一開始就為 false，則程式會跳過迴圈，並執行該 while 迴圈之後的第一個敘述。

以下為範例：

```
int x = 8;
while(x>8) {
    System.out.println("in the loop");
    x = 10;
}
System.out.println("past the loop");
```

此程式執行結果：

```
past the loop
```

雖然 while 迴圈中的測試變數 x 會遞增，但是程式並不會自己察覺，這和

do-while 迴圈不同，因為 do-while 會先執行迴圈一次，然後進行第一次測試。

#### 4.2.2 do-while 迴圈

以下是執行 do-while 範例：

```
do {  
    System.out.println("Inside loop");  
} while(false);
```

即使運算式為(false)，System.out.println()仍產生一次結果，須注意在 do-while 運算式的尾端使用「；」分號。

以下為正確與錯誤的 while 運算式範例：

```
int x = 1;  
while (x) { } //x 不是布林值所以無法編譯  
while (x = 5) //無法編譯，此將 x 指派為 5 非等式  
while(x ==5) //正確，等號測試  
while(true) { } //正確，結果為布林值
```

#### 4.2.3 for 迴圈

若得知要執行幾次迴圈程式碼區段中的敘述，則使用 for 迴圈，而除了迴圈主體外，for 迴圈宣告有三個部分：1. 變數宣告與初始化；2. 布林運算式(測試條件)；3. iteration 運算式，這三個 for 迴圈都使用分號做分隔。以下為 for 迴圈範例：

```
for( /* initialization */ ; /* condition */ ; /* iteration */  
{ }
```

```
for(int i = 0;i < 10; i++) {  
    System.out.println("i is" + i);  
}
```

for 敘述的第一部分可以讓你在後面的括弧中宣告變數與初始化 0 個、1 個

或數個同類別的變數。如果宣告數個同類別的變數，需使用逗號區隔。

如下所示：

```
for (int x = 10, y = 3; y > 3; y++) { }
```

宣告與初始化必須在 for 迴圈的最前端，宣告與初始化只會在迴圈內一開始時執行一次，for 迴圈內宣告的變數必須與 for 迴圈一起結束，如下範例：

```
for ( int x = 1; x < 2; x++) {  
    System.out.println(x);  
}  
System.out.println(x); //此行錯誤，x 不在迴圈內，無法存取。
```

### 條件(布林)運算式

它的運算結果必須為布林值，且只能有一個邏輯運算式，以下為範例：

```
for(int x = 0; (x > 5), (y < 2); x++) { } //錯誤，運算式太多。
```

### iteration 運算式

在執行完 for 迴圈的主體後，接著會執行 iteration 運算式，此處是每次重複執行迴圈時執行的動作，此運算式會等到迴圈主體執行完成之後才開始執行，此處只須注意 iteration 運算式是最後才執行的區段。

### 造成迴圈中斷的原因

迴圈中的程式碼	結果
break	立即跳到 for 迴圈後的第一個敘述並執行該敘述。
return	立即跳回呼叫方法並執行該方法。
System.exit( )	停止執行所有程式，並關閉 VM

必須知道 for 迴圈內宣告的變數不能在 for 迴圈的範圍外使用，不過可以使

用只在 for 敘述內初始化的變數，但必須在一開始就宣告。

以下為正確的 for 迴圈程式碼：

```
int x =3;
for(x = 12; x < 20; x++) { }
System.out.println(x) ;
```

#### 4.2.4 在 for 迴圈中使用 break 與 continue

break 關鍵字會中斷整個迴圈而 continue 關鍵字會中斷目前執行的迴圈，這兩者之間的差異在於，要繼續執行新迴圈或是跳到迴圈下方的第一個敘述，然後從該敘述繼續執行。break 會讓程式停止執行最裡面的迴圈，並開始執行該區段後方的程式碼。

continue 會中斷最內層迴圈目前正在重複的區段，並從同一迴圈的下一個重複區段開始。以下為 continue 程式碼：

```
for (int i = 0; i < 10; i++) {
    System.out.println("Inside loop");
    continue;
}
```

上述程式並不會無止盡的迴圈，即使遇到 continue，iteration 運算式仍會繼續執行，它會把目前重複的區段當作自然結束的區段。以下為 continue 在 if 的敘述：

```
for (int i = 0; i < 10; i++) {
    System.out.println("Inside loop");
    if (foo.doStuff( ) == 5) {
        continue;
    }
    //更多迴圈程式碼，當上述 if 測試為 true 時，這些程式碼便不會執行。
}
```

continue 必須位於迴圈內，否則會得到編譯錯誤，break 必須在迴圈或

switch 敘述內使用。

#### 4.2.5 無標籤的敘述

break 與 continue 可以不帶標籤或帶標籤，break 會中斷最內層迴圈的執行，並跳到迴圈區段外的第一行程式碼，然後繼續執行。以下為 break 無標籤的做法：

```
boolean problem = true ;
while (true) {
    if(problem) {
        System.out.println("There was a problem");
        break;
    }
}
```

以下為無標籤的 continue 程式碼：

```
while(!EOF) {
    //讀取檔案的欄位
    if(three was a problem) {
        //移到檔案的下一個欄位
        continue;
    }
}
```

上述程式每次會從一個欄位讀取一個檔案，若發生錯誤，程式便會移到檔案的下一個欄位，並使用 continue 回到迴圈，繼續讀取各個欄位。若是使用 break 指令，一遇到錯誤，程式碼便會停止讀取檔案，並移到下一行程式碼。

#### 4.2.6 帶標籤的敘述

需了解到帶標籤與無標籤的 break 與 continue 兩者的差異，只有在巢狀迴圈的情況下才須知到 labeled 的變化，且需要指出要中斷哪一個巢狀迴圈，或是要從哪一個巢狀迴圈繼續執行。

若 break 關鍵字結合了標籤，則 break 會離開帶標籤的迴圈而不是最內層的迴圈，以下為範例：

```

foo:
  for (int x = 3; x < 20 ;x++) {
    while (y > 7) {
      y--;
    }
  }

```

使用標籤名稱與 break 的語法是 break 關鍵字，break 後方接標籤名稱，再加上分號，以下為帶標籤的 break 範例：

```

outer:
  for (int i = 0; i < 10; i++) {
    while (y >7) {
      System.out.println("Hello");
      break outer;
    } //內層 for 迴圈的結尾
    System.out.println("Outer loop."); //無法產生結果
  } //外層 for 迴圈的結尾
  System.out.println("Good-Bye");

```

此程式碼結果：

```

Hello
Good-Bye

```

以上範例中，一次會產生一個 Hello，接著執行帶標籤的 break，然後程式會離開標示為 outer 的迴圈，接著下一行產生 Good-Bye。

使用 continue 而不是 break 時，以下範例：

```

outer:
  for (int i = 0; i < 10; i++) {
    for (int j = 0 ;i < 5; j++) {
      System.out.println("Hello");
      continue outer; //跳回 outer 標籤
    } //內層迴圈的結尾
  }

```

```
        System.out.println("outer"); // 沒有結果
    }
    System.out.println("Good-Bye");
```

此程式碼執行結果 Hello 會重複 10 次，當 continue 執行之後，流程會繼續執行下一個標籤的迴圈，等到外層迴圈的條件運算結果為 false 時，i 迴圈便會結束並 Good-bye。

帶標籤的 continue 與 break 必須位於標籤名稱相同的迴圈內，否則無法編譯。

### 4.3 例外處理機制(Exception Handling)

本節主要是瞭解在程式碼某處出現的例外所造成的影響，例外可能是執行時的例外或是檢查(check)時發生的例外或錯誤。程式碼的有效方法中可能包含 try、catch 或 finally 等敘述。

#### 4.3.1 使用 try 與 catch 建立例外

例外是指異常狀況，而且會改變程式正常的流程，導致例外的原因包括硬體錯誤、資源耗盡以及一些常見的錯誤，當 Java 發生例外事件時，便會丟出例外，負責處理例外的程式碼稱為「例外處理機制」，它會捕捉丟出的例外。

例外處理的方式是在發生例外時，將程式的執行轉送給適當的例外處理機制。例如，呼叫一個開啟檔案方式，但檔案無法開啟，該方法便會停止執行，此時便會執行處理此狀況的程式碼。

try 會定義可能發生例外的程式碼區段，此區段稱為「監控區域」(guarded region)，一個或數個 catch 分別會將例外的類別對應到該例外處理機制的程式碼區段，以下為範例：

1. try {
2. //這是監控區域的第一行

```

3.    //它是由 try 關鍵字控制
4.    //此處是會導致某種例外的程式碼
5.    //此程式碼可能很多行可能只有一行
6.    }
7.    catch(MyFirstException) {
8.        //此處是處理此例外的程式碼
9.        //這是另一行的例外處理機制
10.       //這是例外處理機制的最後一行
11.    }
12.    catch(MySecondException) {
13.        //此處是處理此例外的程式碼
14.    }
15.
16.    //此處是其它未監控的程式碼

```

此虛擬碼中 2 到 5 行是屬於 try 分句的監控區域，需注意的是 catch 區段一定得緊跟在 try 區段之後，此外每一個 catch 區段都必須緊跟在彼此後面，期間不可有其它敘述區段。

例外從第 2 行開始，若程式一直執行到第 5 行都沒有例外產生，則會跳到第 15 行繼續執行，但若 2 至 5 行期間跳出 MyFirstException 類別的例外，此時會跳到第 8 行繼續執行，程式會執行第 8 行至第 10 行的 catch 區段，然後在跳到第 15 行繼續執行。

### 4.3.2 finally

try 與 catch 提供了非常好的例外處理機制，不過還有一個清除的問題，一但丟出例外時，程式便會離開 try 區段，因此，不能將清除碼放在 try 區塊尾端，try 區段程式碼後面的例外處理機制並不好清理，因為每個例外處理機制都需要個別的清除碼，Java 使用 finally 區段來解決例外處理時所產生不必要的程式碼。

不管有沒有丟出例外，finally 區塊會將永遠在 try 區塊後某處執行程式碼



包覆起來，即使在 try 區塊中有 return，finally 區塊也會在 return 之後立即執行，以下為另一個範例：

```
1.  try {
2.    //這是監控區域的第一行
3.  }
4.  catch(MyFirstException) {
5.    //此處是處理此錯誤的程式碼
6.  }
7.  catch(MySecondException) {
8.    //此處是處理此錯誤的程式碼
9.  }
10. finally {
11.   //此處程式碼會釋放
12.   //在 try 區塊中配置的資料
13. }
14.
15. //此處是其它程式碼
```

以上執行都是從 try 區塊的第一行開始，若在 try 區塊中沒有發生任何例外，程式便會跳到第 11 行繼續執行，若 try 區塊在執行時丟出 MySecondException，則程式會跳到例外處理機制的的第一行繼續執行，也就是 catch 區段的第 8 行，等到 catch 區塊中的所有程式碼都執行完成後，程式便會移到第 11 行也就是 finally 區塊的第一行，不管發生什麼事 finally 都會執行。

如果沒有 catch 區塊或 finally 區塊，則不能使用 try 區塊。try 區塊在執行時，將有可能發生編譯錯誤，所有 catch 區塊都必須緊跟在 try 區塊之後，則 finally 區塊也必須緊跟在最後一個 catch 區塊之後。可以單一省略 catch 區塊或是 finally 區塊，但是不能兩者都沒有。注意：try 與 catch 區塊之間不能加入任何程式碼。

### 4.3.3 傳遞未捕捉到(catch)的例外

方法未針對特定例外提供 catch 碼，該方法便稱為「迴避」例外，以堆疊的例外來說明，若堆疊在傳遞中掉到最底層稱為例外的傳遞(exception propagation)，可能會導致程式無法執行，未捕捉到的例外會導致應用程式停止執行，此時，應該讓程式知道丟出的例外、丟出例外方法以及當時的堆疊狀態，以方便偵錯。

### 4.3.4 定義例外

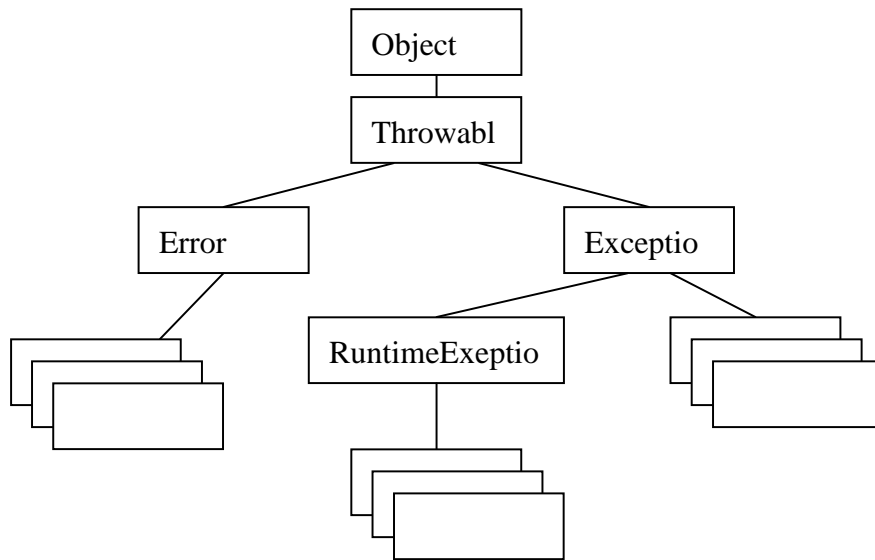
當例外丟出時，特定 Exception 子類別的物件便會實體化，並傳送給例外處理機制，當作 catch 區段的參數。catch 區段如下：

```
try {  
    //部分程式碼  
}  
catch (ArrayIndexOutOfBoundsException e) {  
    e.printStackTrace( );  
}
```

此範例中，e 是 ArrayIndexOutOfBoundsException 類別的實體，和其它物件一樣，也可以呼叫此方法。

### 4.3.5 例外層級

所有例外層級都是 Exception 類別的子類別，該類別是從 Throwable 類別衍生出來的。



圖：例外類別層級

由圖可知，Throwable 會衍生兩個子類別：Exception 與 Error。從 Error 衍生而來的類別代表非程式錯誤所導致的異常狀態或程式執行時一般事件所導致的異常狀態。而 Error 並不是例外，因為它們不是從 Exception 類別衍生而來的。

當某些資源無法取得或正確執行所需的條件不存在時，才會導致例外發生，因此例外不是程式設計錯誤所導致的結果，例如：應用程式必須與無法回應的應用程式或電腦通訊，這便是非錯誤所導致的例外。

在此部分考試時只須知道使用 throws 關鍵字可以丟出 Exception、Error、RuntimeException 與 Throwable 類別，並檢查所有類別。

### 4.3.6 處理例外類別的階層

catch 關鍵字可指定要捕捉的例外類別，且可以使用一個 catch 程式碼區段中捕捉數種類別的例外，若 catch 中指定的例外類別沒有子類別，則只會捕捉指定的例外類別，不過在 catch 區段中指定的類別包含子類別，則會一併捕捉該特定類別的所有例外物件，例如，IndexOutOfBoundsException 有兩個子類別：ArrayIndexOutOfBoundsException 與 StringIndexOutOfBoundsException，寫一個例外處理機制來處理這兩種型別的邊界錯誤所導致的例外，如果不在乎有的是哪一個例外，此時可以寫出下列 catch 區段：

```
try {
    //可丟出邊界例外的程式碼
}
catch (IndexOutOfBoundsException e) {
    e.printStackTrace( );
}
```

### 4.3.7 例外對應(Exception matching)

若例外層級包含一個超類別例外和許多子類別，而想要以特殊的方式處理其中一個子類別，再一起處理其它的類別，只需要寫兩段 catch 程式碼。當丟出例外時，Java 會找到例外類別的 catch 區段，若找不到，它便會在處理機制中尋找例外的超類別，如果找不到可以和例外的超類別對應的 catch 區段，它便會在呼叫堆疊中向下傳遞此例外，此過程稱為例外對應。

### 4.3.8 例外宣告與公用介面

如何知道方法丟出的哪些例外是必須檢查的?正如同方法必須指定它支援的類別和參數數目，方法所丟出的例外也必須經過宣告，丟出的例外清單是方法公用介面的一部分。throws 關鍵字會列出方法並丟出例外。注意：每個方法都必須提供 catch 區段來處理所有已檢查的例外 (checked exception)，或將每個未處理的已檢查例外，表示成已丟出的例外，此稱為 Java 的處理或宣告條件。

### 4.3.9 重新丟出相同例外

可以從 catch 區段中丟出新例外，也可以丟出剛捕捉的同一個例外，以下是範例 catch 的程式碼：

```
catch(IOException e) {  
    //執行，若不能處理...  
    throw e;  
}
```

它會忽略與同一個 try 有關的其它 catch 區段，然後將例外丟回呼叫的方法。若從 catch 區段丟出已檢查的例外，而不是處理或宣告例外，以下為錯誤範例：

```
public void doStuff ( ) {  
    try {  
        //可能有問題的 IO 作業  
    } catch (IOException ex) {  
        //無法處理  
        throw ex; //除非宣告此例外，否則無法丟出該例外  
    }  
}
```

此程式碼的 doStuff( )方法能夠丟出已檢查的例外，此處是 IOException，如果想要重新丟出捕捉到的 IOException，就必須宣告它。

#### 4.4 Assertion(斷言)機制

使用 assertion 的程式碼，並區分 assertion 的不當使用，判別正確的 assertion 機制。

以下範例把假設放在註解中：

```
if(x > 2 && y) {
    //執行某個動作
} else if ( x < 2 || y) {
    //執行某個動作
} else {
    //x 必須為 2，執行其它動作
}
寫入 print 敘述
while (true) {
    if (x > 2) {
        break;
    }
    System.out.print ( "If we got here something went horribly
wrong" );
}
```

此版本的 assertion 機制可在寫程式期間測試假設，而不需要替程式開發完後或部署完後便不會出現的例外編寫例外處理機制。

#### 4.4.1 Assertion 概述

如果假設傳遞給方法的數字一定不是負數，在測試與偵錯期間，此時又需要驗證假設，但又不想在開發完成後拿掉 print、執行期間例外處理機制或 if/else 測試，如果保留這些，又會影響執行效能，此時就可以使用 assertion 來解決。

以下為預設的 assertion 程式碼：

```
private void methodA(int num) {
    if (num >= 0) {
        //執行
    } else { //num 必須小於 0
        //永遠都用不到此程式碼
        System.out.println("Yikes! num is a negative number! "+num);
    }
    useNum(num + x) ;
}
```

若假設可以成立，而又不想花時間寫例外處理機制的程式碼，在執行期間，也不想程式碼中使用 if/else 區段，因為當符合 else 條件時，表示先前的邏輯是有缺點的，Assertion 可以在開發期間驗證假設，但是 Assertion 碼會在部署程式時消失，因此不需要追蹤或移除的 overhead 或偵錯碼。

以下為 methodA( )來驗證參數不是負數的假設：

```
private void methodA(int num) {
    pssert ( num >= 0); //若測試結果不是 true
        //丟出 AssertionError
    useNum(num + x) ;
}
```

Assertion 不僅可讓程式碼正整潔、簡單，還可以讓程式碼像下列這種方式執行，因為必須另外開啟，否則 Assertion 是不會發生作用的：

```
private void methodA(int num) {
    useNum(num + x); //此部分已經測試過
    //至此一切正常
}
```

Assertions 的作業方式很簡單，假設某件事情的條件絕對為 true，若驗證的結果證實該條件是真的，程式可以正常作業，程式碼會繼續執行，Assertion 有兩種：簡單版和非常簡單版，如下：

**非常簡單版：**

```
private void doStuff( ) {
    assert(y > x) ;
    //假設 y 大於 x 的其它程式碼
}
```

**簡單版：**

```
private void doStuff( ) {
    assert ( y > x): "y is "+y+"x is "+x;
    //假設 y 大於 x 的其它程式碼
}
```

兩者差異在於簡單版的會加入第二個運算式，並使用冒號與第一個運算式區隔，這兩版都會即時丟出 `AssertionError`，但是簡單版會提供較多的偵錯資訊，而非常簡單版只會告訴你假設條件是 `false`。



## 4.4.2 Assertion 語法

圖為正確與錯誤的 assert 運算式：

運算式 1		運算式 2	
正確	錯誤	正確	錯誤
assert(x= =2)	assert(x = 2)	: "x is"+x	:void
boolean z = true; assert(z)	int z = 0; assert(z)	public int go( ) {erturn;}:go( ) ;	public void go ( ) {} :go( );
assert false	assert 1	:new Foo ( );	:Foo f;

使用 Assertion 編譯：

```
int assert = getIntiialValue( );
if (assert == getActualResult( ) ) {
    //執行某作業
}
```

以上程式碼中的 assert 只是識別字，須注意的是可以將 assert 作為關鍵字或識別字，但不能兼具兩種身分。

### 執行使用 Assertion 的程式碼

在編寫 Assertion 的程式碼時，可以選擇是否要啟用 Assertion，預設狀態下 Assertion 是停用的。選擇性地啟用與停用，啟用與停用 Assertion 的指令可以有許多種方式：

- 不使用參數:啟用或停用所有類別中的 Assertion，但系統類別除外。
- 使用套件名稱:啟用或停用特定套件以及同一目錄階層中此套件下任何套件中的 Assertion。
- 使用類別名稱:啟用或停用特定類別中的 assertion。

Assertion 的指令行：

指令行範例	代表意義
java -ea java -enableassertions	啟用 Assertion
java -ea java -disableassertions	停用 Assertion
java -ea:com. foo. Bar	啟用 com. foo. Bar 類別中的 Assertion

<code>java -ea;com.foo</code>	啟用 com.foo 套件以及其所有子套件中的 Assertion
<code>java -ea -dsa</code>	啟用一般類別中的 Assertion，但停用系統類別中的 Assertion
<code>java -ea -da:com.foo</code>	啟用一般類別中的 Assertion，但停用 com.foo 套件極其子套件中的 Assertion

#### 4.4.3 Assertion 應用

在考試時，如果看到適當(appropriate)這個字，別把它與有效(legal)搞混了，appropriate 是指根據機制的開發人員或 Sun 認可的方式來使用某樣東西的方式。

##### 不要使用 Assertion 驗證公有(public)方法的參數

下列是不當使用 Assertion 的範例：

```
public void doStuff(int x) {
    assert (x > 0);
    //處理與 x 有關的事情
}
```

如果要驗證公用方法的參數，可能要使用例外丟出 `IllegalArgumentException`。

##### 不要使用 Assertion 驗證私有(private)方法的參數

如果寫的是私有方法，必須編寫呼叫該方法的程式碼，以下為範例：

```
private void doMore(int x) {
    assert(x > 0);
}
```

此範例與前一個範例差別在於存取修飾詞，在私有參數上必須強制使用條件，但在公用方法上不要強制使用條件。

##### 不要使用 Assertion 驗證指令行參數

```

switch(x) {
    case 2: y = 3;
    case 3: y = 17;
    case 4: y = 27;
    default : assert false; //程式不會執行到此處的程式碼
}

```

若認為 x 必須為 2、3 或 4，此時可以用 `assert false`，讓該程式碼執行時丟出 `AssertionError`。在 `switch` 範例中並不是執行布林測試，因為已經知道永遠都不會執行改程式碼區段，表示假設有問題。

### 不要使用會引起副作用的 `assert` 運算式

以下是不好的做法：

```

public void doStuff( ) {
    assert (modifyThings( ));
    //繼續
}
public boolean modifyThings( ) {
    x++ = y;
    return true ;
}

```

此處的規則是：程式必須保持與使用 `assert` 運算式之前一樣的狀態

◎ 考題分析

1. 若

```
1. public class Switch2 {
2.     final static short x = 2 ;
3.     public static int y = 0;
4.     public static void main(String [] args) {
5.         for (int z=0; z<3; z++) {
6.             switch(z) {
7.                 case y : System.out.print( "0 ");
8.                 case x-1 : System.out.print( "1 ");
9.                 case x : System.out.print( "2 ");
10.            }
11.        }
12.    }
13. }
```

則會產生下列何種結果？

- A. 012
- B. 012122
- C. 7 行的編譯失敗。
- D. 8 行的編譯失敗。
- E. 9 行的編譯失敗。
- F. 行期間丟出例外。

解：答案是 C。第七行錯誤，case 運算是必須為常數，或標示為 final 變數，y

部分不是設定為 final，所以執行到第 7 行會編譯失敗。

2. 若

```
1. public class Switch2 {
2.     final static short x = 2 ;
3.     public static int y = 0 ;
```

```

4.   public static void main(String [] args) {
5.       for (int z=0; z<3; z++) {
6.           switch(z) {
7.               case x : System.out.print( "0  ");
8.               case x-1 : System.out.print( "1  ");
9.               case x -2: System.out.print( "2  ");
10.          }
11.      }
12.  }
13. }

```

則會產生下列何種結果？

- A. 012
- B. 012122
- C. 210100
- D. 212012
- E. 第 8 行的編譯失敗。
- F. 第 9 行的編譯失敗。

解：答案是 D。第一次 z=0 進來，對應到 case x -2 印出 2；第二次 z=1 進來，對應到 case x -1 印出 1，因為沒遇到 break，所以繼續往下做印出 2；第三次 z=2 進來，對應到 case x 印出 0，往下走再印出 1 跟 2，所以答案是 212012。

3. 若

```

1.   public class If1 {
2.       static boolean b ;
3.       public static void main(String [] args) {
4.           short hand = 42 ;
5.           if (hand < 50 & !b ) hand++;
6.           if (hand > 50 ) ;

```

```

7.     else if (hand > 40 ) {
8.         hand += 7 ;
9.         hand++ ;          }
10.    else
11.        --hand ;
12.    System.out.println(hand) ;
13. }
14. }

```

則會產生下列何種結果？

- A. 41
- B. 42
- C. 0
- D. 51
- E. 第 5 行的編譯失敗。
- F. 第 6 行的編譯失敗。

解：答案是 D. 程式第 5 行處( `hand < 50 & !b` )，前面成立 true 後面 b 原設定為 false 反向後變成 true，成立往下做 `hand+1=43`，往下做第 6 行判斷是否大於 50，否跳下面第 7 行判斷有沒有大於 40，有往下做 `hand += 7`；此時 `hand=50`，再往下+1 答案為 51。

4. 若

```

1. public class Switch2 {
2.     final static short x = 2 ;
3.     public static int y = 0;
4.     public static void main(String [] args) {
5.         for (int z=0; z<4; z++) {
6.             switch(z) {
7.                 case x : System.out.print( "0 " );
8.                 default : System.out.print( "def " );
9.                 case x-1 : System.out.print( "1 " ); break ;

```

```

10.         case x-2 : System.out.print( "2  ");
11.         }
12.     }
13. }
14. }

```

則會產生下列何種結果？

- A. 0 def 1
- B. 2 1 0 def 1
- C. 2 1 0 def def
- D. 2 1 def 0 def 1
- E. 2 1 2 0 def 1 2
- F. 2 1 0 def 1 def 1

解：答案是 F. 當 z=0 時，印出 2；當 z=1 時，印出 1，遇到 break 跳出；當 z=2 時，印出 0 def 1 跳出；當 z=3 時，沒有符合的 case，它會直接跳到 default 關鍵字做執行，印出 def 1 ；最後的答案 2 1 0 def 1 def 1。

5. 若

```

1. public class If2 {
2.     static boolean b1,b2;
3.     public static void main (String [ ] args) {
4.         int x=0;
5.         if ( !b1 ) {
6.             if ( !b2 ) {
7.                 b1=true;
8.                 x++;
9.                 if ( 5 > 6 ) {
10.                    x++;
11.                }
12.                if ( !b1 ) x = x + 10;
13.                else if ( b2 = true) x = x+100;
14.                else if ( b1 | b2 ) x = x+100;

```

```

15.         }
16.         }
17.         System.out.println (x);
18.     }
19. }

```

會產生下列何種結果？

- A. 0
- B. 1
- C. 101
- D. 111
- E. 1001
- F. 1101

解：答案為 C，b1、b2 都為 boolean 值所以預設值為 false，第 5、6 行!(相反值)，所以 5、6 行都為 true 皆可繼續執行第 7 行 b1 設定為 true 所以第 8 行 x 遞增+1，第九行為 false 所以第 10 行不執行，12 行為 false，所以執行 13 行，將 b2 值設為 true 結果為 true 所以執行  $x = x + 100 \Rightarrow x$  帶 1， $x=101$

6. 若

```

1. public class While {
2.     public void loop ( ) {
3.         int x = 0;
4.         while( 1 ) {
5.             System.out.print("x plus one is " +(x+1));
6.         }
7.     }
8. }

```

下列哪一個敘述是正確的？

- A. 第 1 行有語法錯誤。
- B. 第 1 行與第 4 行有語法錯誤。
- C. 第 1、第 4 與第 5 行有語法錯誤。
- D. 第 4 行有語法錯誤。
- E. 第 4 行與第 5 行有語法錯誤。
- F. 第 5 行有語法錯誤。



解：答案是 D.，因為 while 運算式不能使用整數 1，會使編譯器發生錯誤(這是 C 語言的語法)，第一行因為 Java 語法可以區分大小寫所以 While 是正確的，第 5 行正確的因為 String 中可以放置一個等式。

7. 若

```
1. class For {  
2     .public void test ( ) {  
3.  
4.         System.out.println("x = "+ x);  
5.     }  
6. }  
7. }
```

且輸出如下

```
x = 0  
x = 1
```

下列哪兩行程式碼(分別插入)會導致此輸出(選出兩項)

- A .for (int x = -1;x <2; ++x) {
- B. for (int x = 1; x<3 ;++x) {
- C. for (int x = 0; x>2 ;++x) {
- D. for (int x = 0; x<2 ;x++) {
- E. for (int x = 0; x<2; ++x) {

解：答案為 D.、E. 因為此運算解果先遞增後遞增結果都一樣，它會在迴圈執行

完之後遞增，並在遞增運算式之前運算，A.、B. 的 x 迴圈起始值應為 0，C. 結果為 false 直接跳出迴圈。

8. 若

```
1. public class Test {  
2.     public static void main (String [ ] args) {  
3.         int I = 1;  
4.         do {while ( I < 1 )  
5.             System.out.print("I is " + I);  
6.         }while( I > 1) ;  
7.     }  
8. }
```

則會產生下列何種結果？

- A. I is 1
- B. I is 1 I is 1
- C. 沒有輸出。
- D. 編譯錯誤。
- E. I is 1 I is 1 I is 1 是無止盡的迴圈。

解：答案為 C，因為 4~7 行是一個 do while 迴圈，do while 內至少會執行一次，while 在 do while 迴圈內所以執行一次 while 迴圈結果為 false 跳出 while 執行 do while 結果也為 false，所以沒有任何輸出。(do while 主題，只有一個敘述所以不需括弧)

9. 若

```
1. int I = 0;
2. outer:
3.   while (true) {
4.     I++;
5.     inner:
6.     for (int j = 0; j < 10 ; j++) {
7.       I += j;
8.       if ( j==3 )
9.         continue inner;
10.      break outer;
11.    }
12.    continue outer;
13.  }
14. system.out.println(I);
```

會產生怎樣的結果？

- A. 1
- B. 2
- C. 3
- D. 4

解：A. 此程式過程如下：在進入 while 迴圈之後，I 會遞增(I++)。

接著進入 for 迴圈時，I 的值都不會增加。

if ( j==3 )的結果為 false。因此不會去執行到 continue inner。

所以直接執行 break outer，break 敘述會叫 JVM 離開 outer 迴圈。

跳到最外圍，執行 I 的輸出。

10. 若

```
1. int I = 0;
2. label:
3.   if ( I < 2 ) {
4.     system.out.print( "I is " +I);
5.     I++;
6.     continue label;
7.   }
```

會產生怎樣的結果？

- A. I is 0
- B. I is 0 I is 1
- C. 編譯錯誤
- D. 以上皆非

解：C. 由於 continue(繼續)只出現在迴圈結構中，因此是無法編譯的。此語法理論上是可行，但結合 continue 與 if 的敘述都在同一迴圈內，但其實不具任何意義，因此編譯器會強制您編寫彼此結構更整潔的程式。

## 5. 物件導向、過載與覆寫、建構子與回傳類型

### 5.1 封裝的效益

#### 5.1.1 IS-A 與 HAS-A 的關係

### 5.2 覆寫與過載的方法

#### 5.2.1 覆寫的方法

#### 5.2.2 過載的方法

### 5.3 建構子與實體化

#### 5.3.1 建構子的基本概念

#### 5.3.2 判斷是否會產生預設的建構子

#### 5.3.3 過載的建構子

### 5.4 合法的回傳型態

#### 5.4.1 回傳型態的宣告

#### 5.4.2 回傳數值

### ◎ 考題分析

## 5.1 封裝的效益

物件導向程式設計簡單地說就是一種抽象的程式設計，所謂的抽象，是形容物件中的屬性及方法被完整地封裝。一個良好的物件，它的屬性與方法以及存取範圍都是經規範且有著清楚的描述。

在物件導向概念中，所謂的封裝是透過統一方法或介面實作(interface implements)來取得那些類別中的不允許被外部直接存取的內部資料，以維護物件資源的完整性與存取安全。

封裝的程式碼有這兩個特徵：

- 實體變數被保護住（通常是使用 `private`(私有的)修飾詞)。
- 取回(getter)和設定(setter)方法是存取這些實體變數的管道。

### 5.1.1 IS-A 與 HAS-A 的關係

物件導向程式在實作上有 2 種方式，分別是 "is a" 與 "has a"。

- "is a": 在字面上解釋為「是一個」的意思，在 Java 語言中是利用 `extends` 關鍵字來實作延伸類別，也就是繼承的關係。例如：電腦是一種電子機械產品，電腦類別繼承了電子機械產品類別，所以電子機械產品就成了電腦的父類別而子類別則是電腦這個類別。
- "has a": 在字面上解釋為「有一個」的意思，是用來表示類別(class)中的成員變數(member variable)。例如：電腦中有 1 顆 CPU、RAM、HD... 等，CPU、RAM 與 HD 便成了電腦的成員變數。

依上述的例子我們可以利用 Java 語言來實作，如下：

“is a” 與 “has a” 的關係

```

1. class machine
2. { ← 電子機械產品類別
3. }
4. public class computer extends machine ← “is a” 電腦類別繼承電子
   機械產品類別
5. {
6.     CPU theCPU;    □
7.     RAM theRAM;   ⊥ ← “has a” 電腦類別的成員變數
8.     HD the HD;    ⊥
9. }

```

5.2 覆寫與過載的方法

overloading(過載)與 overriding(覆寫)的觀念：

overloading	名字一樣，給不同的條件(參數，多寡或順序不同即可)做不同的事。譬如上班(){}，帶參數”SR”時是在執行”趕工寫程式”，不帶參數時是在執行”摸魚”。注意，回傳值不同不可代表 overload，且會造成 compiler error。
overriding	名字一樣，且條件(參數及回傳值)也要一模一樣。主要是覆寫掉繼承來的東西。譬如繼承了超認真的做人態度，就一定會變得超認真(預設繼承來是不能選擇的)只要可以做到接觸的比爸媽更多(access modifier 要更大)且犯更少的錯(throws exceptions 範圍要更小)，就懂得用另一種態度來生活。

overloading 與 overriding 的區別：

	意義	存取修飾詞	回傳型態	方法名稱	丟出例外
overloading	reuse the method name	don't care	don't care	相同	don't care
overriding	redefine the method	不小於	相同	相同	subset

### 5.2.1 覆寫的方法

『覆寫(overriding)』就是子類別自行實作一個方法取代父類別所提供的方法，子類別的物件在執行時，會使用子類別中重新改寫的方法。主要效益在於重新定義行為的能力，通常這些行為都與特定的子類別型態有關。子類別中也可以定義和父類別相同的資料成員。在子類別中定義和父類別中相同的資料成員時，子類別中的資料成員會「隱藏 (hide)」父類別中的資料成員的內容。

#### 覆寫的基本原則：

- 覆寫是發生在有繼承關係的類別中。
- 子類別中，方法的名稱、參數的型態、個數及擺放順序必須和父類別中的方法相同。
- 回傳資料的型態必須相同(相容)。
- final 的方法無法被覆寫。
- 覆寫時，子類別中的方法的存取權限不能小於父類別中的方法。
- 覆寫時，子類別中的方法不能丟出比父類別中的方法更廣泛或是新的例外。

### 5.2.2 過載的方法

『過載』其實就是『覆載(overloading)』。是指同一種方法名稱根據其不同的參數型別以對應執行到不同的實作，通常發生在同一個類別中。



過載又可分為方法的過載與建構子的過載：

方法過載(method overloading)	建構子過載(constructor overloading)
<pre>public void amethod( ) {...} public void amethod(int i) {...} public void amethod(String s, int i) {...} public void amethod( ) {... return X}</pre>	<pre>public Base( ) {...} public Base(int i) {...} public Base(int I, String s) {...} public Base(String s, int i) {...}</pre>

參數列的順序也會影響，即使傳入型態都有 int 與 string，因順序不同，視為不同的過載方法。

**過載的基本原則：**

- 過載的方法必須有不同的參數清單。
- 過載的方法可以有不同的回傳型態。
- 過載的方法可以有不同的存取修飾詞。
- 過載的方法可以宣告新的、或是範圍更廣的可控制例外。
- 過載的方法可以在相同的類別、或是在子類別中。

## 5.3 建構子與實體化

### 5.3.1 建構子的基本概念

建構子是用來在建立物件時，給予類別中所有的物件或變數一個初始值，如果你沒有指定所給予的初始值，則會自動指派該變數型態的預設值給這一個變數。在一般正常的情形下，不會在建構子中使用「System.out.\*」這一個 package 來顯示任何的資訊。

**建構子有 2 個特性：**

- 建構子(Constructor)沒有回傳值(no return value)。
- 建構子(Constructor)名稱須與類別(class)名稱相同。

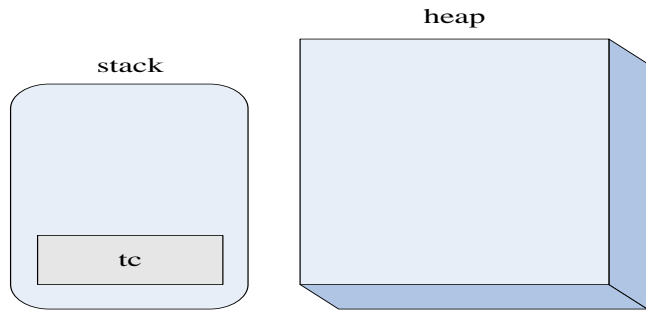
建構子的宣告名稱必須要和所屬的類別「完全相同」，並且不需要加上任何的回傳型態。

例如以下的「TestClass( )」即是「TestClass」這一個類別的建構子：

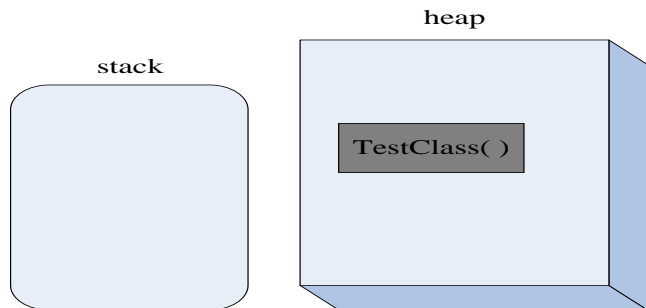
```
public class TestClass {  
    String value;  
  
    public TestClass( ) {  
        value = "字串值";  
    }  
}
```

當我們要建立物件時，必須使用「new」這個關鍵字在 heap 記憶體中要求配置一個區域，如下圖：

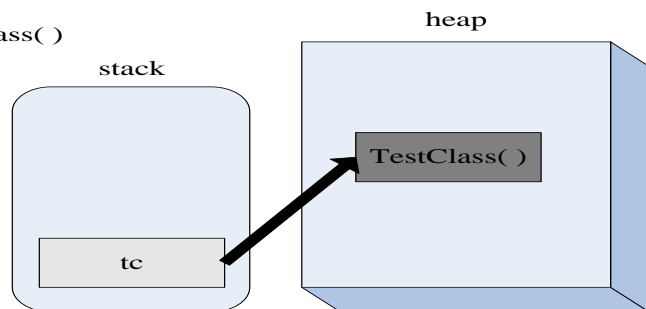
1. `TestClass tc`



2. `new TestClass()`



3. `TestClass tc = new TestClass()`



在第二個步驟中，你所呼叫的就是建構子，而不是類別。在你的程式當中，如果沒有加上自己所寫的可建構子時，則編譯器會自動幫你產生一個預設的可建構子。如果類別的存取等級為 `public`，則預設的可建構子的存取級就必須是 `public`。

### 建構子規則

- Constructor 第一行必須是 `this(..)` 或 `super(..)`，否則 JVM 會自己在第一行插入 `super()`。`this(...)` 是用來呼叫另一個 overloading constructor，使用時必須是 Constructor 的第一行。`super(...)` 是用

來呼叫父類別的某個 Constructor，使用時必須是 Constructor 的第一行。

- Constructor 可以用 public、private、protected、(default) 修飾。
- 使用 private 可以避免被實體化 (instantiates)。
- 可以 overloading constructors 來定義一個以上的 Constructors。
- Constructor 可以丟出 Exception。
- Constructor 不能有回傳值、也不能夠被繼承。
- 不能 overriding constructors。
- Constructor 內不可以同時有 this(...)或 super(...)。
- Constructor 不會被繼承，如果自己沒有寫 Constructor，Compiler 時會加入 default constructor，也就是一個 no parameters「無參數」 and throws no exceptions「沒例外」，其存取權限將與 class 相同。特別注意，若已定義 Constructor 後，compiler 就不會再給 default Constructor。
- 在繼承的情況下，如果 superclass 的 Constructor 有定義會丟出 Exception，那麼 subclass 的建構子也必須定義會丟出 Exception。

### 5.3.2 判斷是否會產生預設的建構子

類別 (class) 裡面一定要有建構子，所以在撰寫 class 時必須定義該類別建  
構子，程式中若沒有明確定義建 Java 在編譯時期會自動幫該類別加上建構子，  
所加上的建構子就稱為預設建構子(Default Constructor)。

預設建構子有下列 5 個特點：

- new 一個物件的時候會自動執行(無參數時)。
- 每一個 class 最多只能有一個預設建構子。
- 程式中若沒有定義建構子，在編譯時期會自動加入，所加入的就稱之為預設建構子。
- 預設建構子沒有參數列(no arguments)。
- 除了初始成員變數或繼承時 super( )的定義外，預設建構子基本上是沒有其它的程式敘述(no body statement)。

範例：

```
public class myTest {  
    myTest( ) {} // 通常不用撰寫因為 Java compile 於 compile time  
                時會自動撰寫  
}
```

建構子不會由繼承產生，它們不是方法，它們無法覆寫，因此超類別所擁有的建構子型態，無法決定將取回的預設建構子的型態。雖然建構子無法被覆寫，但是它們可以過載。

### 5.3.3 過載的建構子

意義在於讓建構子有多個版本，但是參數不同，建構子可以相互呼叫。規則是：建構子的第一行，不是呼叫 super(…)；就是呼叫 this(…)；所以建構子過載只是建構子呼叫超類別的手法。假設我們要建立 Animal 這個類別的實體，必須給這個動物一個名字，在呼叫建構子時放在參數 => new Animal("dog")；但也可以呼叫建構子時，不給參數 => new Animal( )；此時，Animal 必須提

供兩個建構子供呼叫，否則會出錯。

過載建構子的效益在於，可以從類別中提供彈性的方式，對物件進行實體化的處理。以一個建構子啟動另一個過載建構子的效益，則是能避免重複程式碼。

## 5.4 合法的回傳型態

### 5.4.1 回傳型態的宣告

- **過載方法的回傳型態**：過載的方法一定要更改參數清單，回傳型態不一定要改。

- **覆寫方法的回傳型態**：覆寫的方法參數與回傳型態要完全相等。

### 5.4.2 回傳數值

回傳的六個規則：

- 以物件參考作為回傳型態時，可以用 null 回傳。

```
public Button doStuff( ) {  
    return null;  
}
```

- 陣列可以作為回傳型態。

```
public String[ ] go( ) {  
    return new String[ ] { "  
Fred ", " Barney " , " Wilma " } ;  
}
```

- 以物件參考作為回傳型態時，可以回該類別的子孫類別的物件，或實作類別的物件

```
public Animal getAminal( ) {  
    return new Horse( ); } //假設 Horse 繼承 Animal
```

```

-----
public interface Chewable { }
    public class Gum implements Chewable{ }
public class TestChewable{
    public Chewable getChewable{ return new
Gum( ); }
}
//回傳介面實作

```

- 以基本資料型別為回傳型態時，可以回傳強迫轉型為宣告的回傳型態。

```

    public int foo( ){
        float f = 32.5f ;
        return (int) f ;
    }

```

- 同上也可以回傳隱式轉型為宣告的回傳型態。

```

public int foo ( ){
    char f = c;
    return c ;
} // char 與 int 相容

```

- 不可以在 void 回傳型態中的方法，回傳任何內容。

◎ 考題分析

1.

```
1. public class Barbell {
2.     public int getWeight ( ) {
3.         return weight;
4.     }
5.     public void setWeight(int w) {
6.         weight = w;
7.     }
8.     public int weight;
9. }
```

上面所描述的類別中，何者是正確的？

- A. Barbell 類別被緊密地封裝起來。
- B. 第 2 行會與封裝發生衝突。
- C. 第 5 行會與封裝發生衝突。
- D. 第 8 行會與封裝發生衝突。
- E. 第 5 與第 8 行會與封裝發生衝突。
- F. 第 2—5 與 8 行會與封裝發生衝突。

解：答案是 D。如要正確的執行封裝，第八行的地方會有錯誤，因為正確的封裝，該類別裡面不能有標示為 public 的實體變數。

2.

```
1. public class B extend A {
2.     private int bar;
3.     public void setBar(int b) {
4.         bar = b;
5.     }
6. }
7. class A {
8.     public int foo;
9. }
```



上面所描述的類別中，何者是正確的？

- A. 類別 A 被緊密地封裝起來。
- B. 類別 B 被緊密地封裝起來。
- C. 類別 A 與 B 都被緊密地封裝起來。
- D. 類別 A 與 B 都未被緊密地封裝起來。

解：答案是 D。程式部分看到第 7 8 9 行，可以得知 A 並沒有被封裝起來；另外 B 的部份，因為它繼承了 A 類別裡面 foo 的關係，所以還是不能封裝。

3. 下列敘述何者為真？

- A. 被緊密封裝起來的類別通常比較容易重複使用。
- B. 被緊密封裝起來的類別通常比未封裝的類別更常使用繼承。
- C. 在被緊密封裝起來的類別中的方法不能被覆寫。
- D. 在被緊密封裝起來的類別中的方法不能被過載。
- E. 被緊密封裝起來的類別通常不會使用 HAS-A 的關係。

解：答案 A. 是正確的。封裝基本上可以得到三大效益，第一讓程式易於維護，第二讓程式具備擴充性，第三提高程式清晰度。綜合以上觀念，封裝起來的類別，通常比較容易被重複使用。

B. 錯的原因，封裝和繼承是不同的概念，所以這邊提到的不管有沒有封裝，跟繼承沒有絕對的關係。C. 跟 D. 的答案提到，不能被覆寫跟過載，是錯的，在封裝的觀念裡，不會限制覆寫和過載的使用。E. 是錯的，封裝和 HAS-A 的觀念也是分開獨立的關係，沒有絕對的相關性。

4. 底下的敘述中，哪兩個不是封裝的效益？

- A. 程式的清晰度。
- B. 程式的效率。
- C. 稍後增加新功能的能力。

- D. 在需要修改時，只需要比較少的程式。
- E. 存取修飾詞變成具有可選擇性。

解：答案是 B 跟 E。封裝的效益，跟程式的效率沒有關係。且存取修飾詞也只有 private 可供選擇。

5.

```
1. class B extends A {
2.     int getID ( ) {
3.         return id;
4.     }
5. }
6. class C {
7.     public int name;
8. }
9. class A {
10.    C c = new C ( );
11.    public int id;
12. }
13.
```

上面所描述的類別實體中，哪兩個是正確的？

- A. A is-a B
- B. C is-a A
- C. A has-a C
- D. B has-a A
- E. B has-a C

解：答案是 C 跟 E。A 答案，從第一行看下來，可以得知 B 繼承 A 的超類別，所以 B is-a A。B 答案可看到程式中，兩個類別是分開的，並沒有提到誰繼承誰的問題，所以也不會有 C is-a A 或是 A is-a C 的東西出現。C 答案正確，是因為類別 A 中有一個 c 參考了 C 類別的物件，所以 A 跟 C 有 HAS-A 的關係。D 答案是錯的，B 跟 A 的關係是 B 繼承了 A，所以不會有 B has-a A 的東西出現。E 答案正確，因為 B 類別繼承 A 類別，所以它跟 C 類別也產生

了 HAS-A 的關係。

6.

```
1. class A {  
2.     public void baz ( ) {  
3.         System.out.println( "A" );  
4.     }  
5. }  
6. public class B extends A {  
7.     public static void main(String [ ] args) {  
8.         A a = new B ( );  
9.         a.baz( );  
10.    }  
11.     public void baz ( ) {  
12.         System.out.println( "B" );  
13.     }  
14. }
```

請問執行後會得到什麼結果？

- A. A
- B. B
- C. 編譯失敗
- D. 在執行期間產生例外

解: 答案是 B. , B 是 A 的繼承, 第 8 行參考變數 a 是屬於類型 A, A 的參考對象是 B 物件, 第 9 行是一個多行程式的呼叫, 多型的定義所有 A 的子類別型態應該都有 baz ( ) 方法, 呼叫者 B 可以在 A 的參考上啟動 baz ( ), 但是實際執行期的物件, 所執行的則是它自己特定的 baz ( ) 方法。

7.

```
class Foo {  
    String doStuff(int x) { return "hello" ;}  
}
```

在 Foo 的子類別中，哪個方法是不合法的？

- A. String doStuff (int x) { return "hello" ;}
- B. int doStuff (int x) { return 42; }
- C. public String doStuff (int x) { return "Hello" ; }
- D. protected String doStuff (int x) { return "Hello" ;}
- E. String doStuff (String s) { return "Hello" ;}
- F. int doStuff(String s) { return 42; }

解: 答案是 B，因為回傳型態改變，不能為覆寫，參數清單沒有改變，所以也不能過載。A、C、D 為過載。E、F 為覆寫。

8.

```
1. class ParentClass {  
2.     public int doStuff (int x) {  
3.         return x * 2;  
4.     }  
5. }  
6.  
7. public class ChildClass extends ParentClass {  
8.     public static void main (String [ ] args) {  
9.         ChildClass cc = new ChildClass( );  
10.        long x = cc.doStuff(7);  
11.        System.out.println( "x = " + x);  
12.    }  
13.  
14.    public long doStuff (int x) {  
15.        return x * 3;  
16.    }  
17. }
```

請問執行會得到什麼結果？

- A.  $x = 14$
- B.  $x = 21$
- C. 在第 2 行發生編譯失敗
- D. 在第 11 行發生編譯失敗
- E. 在第 14 行發生編譯失敗
- F. 在執行期產生例外

解：答案為 E.，在第 14 行是 ParentClass 中 doStuff ( ) 方法的非覆寫與第 2 行比較可得知，當覆寫某個方法時，必須同時維持相同的參數與回傳型態，如果 14 行回傳的是 int 則執行結果為 B  $x=21$ 。

9.

- 1. class Over {
- 2.     int doStuff (int a, float b) {
- 3.         return 7;
- 4.     }
- 5. }
- 6.
- 7. class Over2 extends Over {
- 8.     //在這插入程式碼
- 9. }

如果將底下的程式個別插入第 8 行，請問哪兩個方法無法編譯？

- A. public int doStuff (int x, float y) { return 4; }
- B. protected int doStuff (int x, float y ) { return4; }
- C. private int doStuff (int x, float y) {return4; }
- D. pribate int doStuff (int x, double y) { return 4; }
- E. long doStuff (int x, float y) { return 4; }
- F. int doStuff (float x, int y) { return4; }

解：答案是 C.、E.，C. 是非法覆寫因為在類別 Over 中，private 修飾詞比 doStuff ( ) 方法預設的修飾詞有更多的限制，E 是非法覆寫因為不能改變回傳型態，也是非法過載因為沒有改變參數。A.、B. 都是覆寫，D.、F. 都是過載。

10.

```
1. public class TestPoly {
2.     public static void main (String [] args) {
3.         parent p = new Child( );
4.     }
5. }
6.
7. class parent {
8.     public parent( ) {
9.         super( );
10.        system.out.println( "instantiate a parent" );
11.    }
12. }
13.
14. class Child( ) extends parent {
15.     public Child( ) {
16.         system.out.println( "instantiate a child" );
17.     }
18. }
```

會產生怎樣的結果？

- A. instantiate a child
- B. instantiate a parent
- C. instantiate a child  
instantiate a parent
- D. instantiate a parent  
instantiate a parent
- E. 編譯失敗
- F. 在執行期間產生例外

解: D. 在 Child 建構子中的程式執行之前，類別 Child 的建構子會以隱式的方式呼叫類別 Parent 的建構子。當類別 Parent 建構子的程式執行時，會列印輸出結果的第一行、結束執行、並將控制項目回傳給 Child 的建構子，這樣會讓它列印出結果，並結束程式的執行。對 super( ) 的呼叫是多餘的。

## 6. Math 類別、字串及 StringBuffer 類別

### 6.1 String 類別的用法

6.1.1 String 是不可改變的物件

6.1.2 String 與記憶體的重要事實

6.1.3 String 類別中重要的方法

6.1.4 StringBuffer 類別

### 6.2 Math 的用法

6.2.1 java.lang.Math 類別中的方法

### 6.3 equals( ) 的用法

6.3.1 == 與 equals( ) 方法比較

### ◎ 考題分析

## 6.1 String 類別的用法

### 6.1.1 字串是不可改變內容的物件

字串中的每個字都是 16 位元的標準萬國碼 (Unicode) 編碼在 JAVA 中每個字串都是物件，就跟其它物件一樣都可以使用 new 關鍵字來生成字串物件。

```
例：String s = new String( )
```

String 類別也有不加參數的建構子，但是你可以採取更有效的用法

```
例：String s = new String( "abcdef" );
```

一但你為某個字串宣告數值之後，它的內容就無法再做改變

```
例：String s = "abcdef" ;  
String s2 = s;  
s = s.concat( "more stuff" )
```

虛擬機器 (VM) 會先取得字串 s 的數值，並在它的後面加上 "more stuff"，於是產生 "abcdef more stuff" 這個內容，由於字串物件是不可改變的，虛擬機器 (VM) 無法將這新字串硬塞給變數 s 所參考的舊字串，所以它會產生一個新的字串物件，讓它內容等於 "abcdef more stuff"，並讓 s 參考它。

### 6.1.2 字串與記憶體

應用程式不斷成長，字串的文字佔用大量的程式記憶體，在這些文字當中，有許多是不必要的重複內容，為了讓 java 更有效率的使用記憶體，JVM 特別在記憶體中撥出一塊叫做 "String constant pool" 的區域，當編譯程式碰到某字串文字的時候，它會先檢查這塊區域，檢查是否已存有相同的字串，對新文字會被指向參考現有的字串，不會再產生新的字串物件，這也是讓字串有不可變更的這個優點。



### 6.1.3 String 類別中重要的方法

- `public char charAt(int index)`

這個方法是回傳位於字串上的位元

```
String x = "airplane" ;  
System.out.println(x.charAt(2));  
//輸出結果為" r"
```

- `public String concat(String s)`

這個方法是回傳一個結合後的字串，就是把方法中所傳遞的字串數值，  
加在啟動方法字串的後面。

```
String x=" taxi" ;  
System.out.println(x.concat( "cab" ));  
//輸出結果為" taxi cab"
```

- `public boolean equalsIgnoreCase(String s)`

這個方法是回傳布林值(true 或 false)，判斷參數中的字串數值是否與  
該字串內容相同，不論字串內容為大寫或小寫。

```
String x =" Exit" ;  
System.out.println(x.equalsIgnoreCase( "EXIT" ));//回傳  
值" true"  
System.out.println(x.equalsIgnoreCase( "exit" ));//回  
傳" false"
```

- `Public int length( )`

這個方法是回傳字串長度

```
String x =" 01234567" ;
```

```
System.out.println(x.length( ));//回傳值 8
```

● **public String replace(char old, char new)**

這個方法是回傳一個經過替換後的字串，此做法是根據第二個參數中的 char 替換成該字串中所有與第一個參數中相同的 char 字元。

```
String x = "oxoxoxox" ;
```

```
System.out.println(x.replace( "x" ," X" ));//輸出為 oXoXoXoX
```

● **public String substring(int begin) 、 public String substring(int begin, int end)**

這個方法是回傳該方法的字串的部分，第一個參數代表字串的起始位置，第二個字串代表結束的地方。

```
String x = "0123456789" ;
```

```
System.out.println(x.substring(5));//輸出為 56789
```

```
System.out.println(x.substring(5,8));//輸出為 567
```

● **public String toUpperCase( )**

這個方法是將所有小寫字元轉換成大寫

```
String x = "A New Moon" ;
```

```
System.out.println(x.toUpperCase( ));//輸出為" A NEW MOON"
```

● **public String trim( )**

這個方法是將字串中頭尾的空格都刪除

```
String x = "hi  " ;
```

```
System.out.println(x+" x" );//輸出為 hi x
```

```
System.out.println(x.trim()+" x" );//輸出為 hix
```

## 6.1.4 StringBuffer 類別

當你想對字串進行大量修改時，應該要使用 StringBuffer 類別。字串物件是不可改變的。所以，如果你要對字串物件執行大量處理的話，最後就會在字串的 pool 中，留下一大堆被棄置的字串物件。但是，類別為 StringBuffer 的物件則可以重複修改，而不會留下一大堆被棄置的字串物件。

## 6.2 Math 類別的用法

### 6.2.1 java.lang.Math 類別中的方法

#### ● abs( )

abs( )方法會回傳參數的絕對值，底下是它的例子：

```
x = Math.abs(99); //輸出是 99
```

```
x = Math.abs(-99); //輸出是 99
```

#### ● ceil( )

ceil( )方法會以 double 的資料型態回傳，回傳的參數值相當於直接進位法，負數值就回傳絕對值後，直接省略小數點後數字。換言之，它會將參數化簡為最接近的整數值。

```
Math.ceil(9.0); //輸出是 9.0
```

```
Math.ceil(8.8); //輸出是 9.0
```

```
Math.ceil(8.02); //輸出是 9.0
```

```
Math.ceil(-9.8); //輸出是 -9.0
```

#### ● floor( )

floor( )方法會以 double 的資料型態，回傳的參數相當於直接省略法，負數相當於絕對值後，直接進位法。它的處理方式剛好跟 ceil( )相反。

```
Math.floor(9.8); //輸出為 9.0
```

```
Math.floor(9.0); //輸出為 9.0
```

```
Math.floor(-8.8); //輸出為-9.0
```

### ● `max()`

`max()`方法可以接受兩個數值型態(可接受 `int`, `long`, `float`, `double`)的參數，並回傳比較後較大的數值。

```
x = Math.max(1024, -5000); //輸出為 1024
```

### ● `min()`

`min()`方法剛好跟 `max()`方法相反，它也可以接受兩個數值型態(可接受 `int`, `long`, `float`, `double`)的參數，但是回傳比較後較小的數值。

```
x = Math.min(0.5, 0.3); //輸出為 0.3
```

### ● `random()`

`random()`方法會回傳一個隨機的數值，這個數值的資料型態為 `double`，回傳值會大於或等於 0.0、小於 1.0。`random()`方法不需要參數。

```
public class RandomTest {  
    public static void main(String [] args) {  
        for (int x=0; x < 15; x++)  
            system.out.print((int)(Math.random() * 10) + " ");  
    }  
}
```

`println()`方法會將呼叫 `Math.random()`的結果乘以 10，然後將結果的 `double` 數值強迫轉型為整數。底下為程式實際跑過的樣本。

```
6 3 3 1 2 0 5 8 4 7 5 6 3 2 5
```

## ● `round( )`

`round( )`方法會回傳最接近參數的整數值。它的演算法如下：先將參數加上 0.5，然後去掉小數的部分，得出最接近它的整數值(就是四捨五入法的應用)。這個方法具有過載的能力，可處理 `float` 或 `double` 型態的參數。

```
Math.round(-10.5); //輸出為-10
```

```
Math.round(9.5); //輸出為 10
```

## ● `sin( )`

`sin( )`方法會回傳某個角度的正弦。它的參數為 `double` 資料型態，代表的是以弧度表示的角度。你也可以透過使用 `Math.toRadians( )`，將角度轉換成弧度。

```
Math.sin(Math.toRadians(90.0)); //回傳為 1.0
```

## ● `cos( )`

`cos( )`方法會回傳某個角度的餘弦。它的參數為 `double` 資料型態，代表的是以弧度表示的角度。

```
Math.cos(Math.toRadians(0.0)); //回傳為 1.0
```

## ● `tan( )`

`tan( )`方法會回傳某個角度的正切。它的參數為 `double` 資料型態，代表的是以弧度表示的角度。

```
Math.tan(Math.toRadians(45.0)); //回傳為 1.0
```

## ● `sqrt( )`

`sqrt( )`方法會回傳某個資料型態為 `double` 的數值的平方根。

```
Math.sqrt(9.0); //回傳為 3.0
```

## 6.3 equals( ) 的用法

判斷將 equals 方法套用在所有類別組合而成的物件上，這些類別指的是：  
java.lang.String、java.lang.Boolean、java.lang.Object。

- 如果要比較基本資料型別變數，使用 == 。
- 如果要判斷兩個參考變數是否參考同一個物件，使用 == 。
- == 比較的是位元樣式，可能是基本資料型別的位元，或是參考的位元。
- 如果要判斷兩個物件是否在意義上相等，使用 equals( ) 。
- 字串與 Wrapper 類別會覆寫 equals( ) 方法，來檢查其中的數值。
- 不同 Wrapper class 的物件，equals( ) 比較值永遠是 false。

### 6.3.1 == 與 equals( ) 方法的比較

#### 變數間的比較

從基本資料型別與參考變數開始討論，使用 == 比較基本資料型別變數，但 equals( ) 方法很明顯的就無法用在基本資料型別上。== 運算子會回傳一個 boolean 函數值：如果變數等值的話，回傳值為 true 否則為 false。不管 java 程式在那邊執行，只要在單一虛擬機器上執行所有的參考變數具有同樣的大小（以位元為單位）與格式。當使用 == 運算子比較兩個參考變數的時候，實際上執行的是測試看看這兩個參考變數是否為相同的物件，在比較變數的時候（不管是基本資料型別或參考變數），實際上是在比較兩組位元模式。

#### 比較變數時的重點：

- 參考變數與基本資料型別變數所使用的規則是相同的：如果兩組位元樣式相等的話，== 會回傳 true。

- 基本資料型別變數必須使用 ==，它們不能使用 equals( ) 方法。
- 對參考變數來說，== 的意思是指兩個參考變數的參考相同的物件。

### 物件間的比較

當想要判斷兩物件在意義上是否相等時，就需要用到 equals( ) 方法，equals( ) 方法也會回傳 true 或 false。

### equals( ) 方法的重點提示

Object 這個類別是所有類別開始往下延伸的起點，它具有 equals( ) 的用法，就是說每個其它的 java 類別(包括在應用程式介面或你所建立的類別中)都會繼承一個 equals( ) 方法。

### equals( ) 方法相關的重點

- equals( ) 方法只能用來比較物件。
- equals( ) 方法會回傳布林值。
- StringBuffer 類別並無覆寫 equals( ) 方法。
- 字串與 Wrapper 類別都標記為 final，而且已經有覆寫了 equals( ) 方法。

◎ 考題分析

1.

```
1. public class SteingRef {  
2.     public static void main (String [ ] args) {  
3.         String s1 = "abc" ;  
4.         String s2 = "def" ;  
5.         String s3 = s2;  
6.         s2 = "ghi" ;  
7.         System.out.println (s1 + s2 + s3);  
8.     }  
9. }
```

請問執行後會得到什麼結果？

- A. abcdefghi
- B. abcdefdef
- C. abcghidef
- D. abcghighi
- E. 編譯失敗
- F. 在執行期產生例外

解：答案是 C。當執行到第 5 行時，s3 會參考到 s2 的地方，兩個都會是 def  
到第六行時，s2 會建立一個新的字串物件，為 ghi，s2 會參考它，結局會  
變成 abcghidef'。

2.

```
11. String x = "xyz" ;  
12. x.toUpperCase ( );  
13. String y = x.replace( 'Y' , 'y' );  
14. y = y + "abc" ;  
15. System.out.println(y);
```

請問執行後會得到什麼結果？



- A. abcXyz
- B. abcxyz
- C. xyzabc
- D. Xyzabc
- E. 編譯失敗
- F. 在執行期間產生例外

解：答案是 C。程式 12 行處會建立一個新的字串物件，會指向第 11 行處的 xyz，可是它沒有參考變數會參考它，所以這個物件會直接不見。到 13 行處再建立一個新的物件會被 y 參考，物件內容直接抓到 x 下面的 xyz，replace 的意思是用前面的 Y 取代後面的 y，可是 Y 沒有指向任何物件，所以 13 行會直接建立一個字串 xyz，第 14 行的地方，直接將 xyz 放在 abc 前面，輸出 xyzabc。

3.

- 13. String x = new String ( “xyz” );
- 14. y = “abc” ;
- 15. x = x + y;

請問執行後會建立幾個字串物件？

- A. 2
- B. 3
- C. 4
- D. 5
- E.

解：答案是 C。第 13 行處會建立兩個新的物件，一個是 x 物件，另外一個是 xyz 物件可是沒變數參考它，但是它也算是一個物件，不會消失不見。13 行執行完會產生兩個，14 行一個，15 行一個，結局會有 4 個物件。

4.

14. String a = "newspaper" ;
15. a = a.substring(5,7);
16. char b = a.charAt(1);
17. a = a + b;
18. System.out.println(a);

請問執行後會得到什麼結果？

- A. apa
- B. app
- C. apea
- D. apep
- E. papp
- F. papa

解：答案是 B。第 15 行 substring 秀出第 5 個位置到第 7 個位置的字元秀出 ap，第 16 行處，b 參考到 a，charAt 秀出 a 中的第 1 個字元，秀出 p，所以結局是 app。

5.

4. String d = "bookkeeper" ;
5. d.substring( 1 , 7 );
6. d = " w " + d;
7. d.append ( " woo " );
8. System.out.println(d);

請問執行後得到什麼結果？

- A. wookkeewoo
- B. wbookkeeper
- C. wbookkeewoo
- D. wbookkeeperwoo
- E. 編譯失敗
- F. 在執行期產生例外

解:答案是 E. , 第 7 行的 append( ) 會對字串呼叫一個 StringBuffer 方法, 可是程式中並沒有 StringBuffer 方法, 所以編譯失敗。

6.

```
1. public class Example {
2.     public static void main (String [ ] args) {
3.         double values [ ] = {-2.3 , -1.0 , 0.25 , 4 };
4.         int cnt = 0;
5.         for(int x=0; x <values.length; x++) {
6.             if (Math.round(values[x] + .5) == Math.ceil(values[x]))
{
7.                 ++cnt;
8.             }
9.         }
10.        System.out.println( "same results" + cnt +
"time(s)" );
11.    }
12. }
```

請問執行後會得到什麼結果?

- A. 相同的結果 0 次
- B. 相同的結果 2 次
- C. 相同的結果 4 次
- D. 編譯失敗
- E. 在執行期產生例外

解:答案是 B, 第 6 行 Math.round( ) 會將參數+0.5, 但在 round( ) 方法前就先+了 0.5 所以參數總共+1 在執行 floor。

7. 底下何者是對 Math.max 的有效呼叫?(選擇所有適合者)

- A. Math.max (1, 4)
- B. Math.max (2.3 , 5)
- C. Math.max(1, 3, 5, 7)
- D. Math.max(-1.5, -2.8f)

解:答案是 A、B、D，max 可以接受型態是 int、long、float、double，但只能接受 2 個參數，所以 C 錯了。

8. 底下的敘述中，哪兩個是呼叫 Math.random( )後所得到的正確結果?(請選擇兩個答案)

- A. 結果小於 0.0
- B. 結果大於或等於 0.0
- C. 結果小於 1.0
- D. 結果大於 1.0
- E. 結果大於或等於 1.0
- F. 結果小於或等於 1.0

解:答案為 B、C，random( )方法會回傳一個隨機的亂數值，這個值的型態為 double，而且會大於或等於 0.0、小於 1.0。另外 random( )方法不需要接任何的參數。

9.

1. public class SqrtExample {
2. public static void main (String [] args) {
3. double value = -9.0;

```
4.     system.out.println( Math.sqrt(value));
5.     }
6.     }
```

會產生怎樣的結果？

- A. 3.0
- B. -3.0
- C. NaN
- D. 編譯失敗
- E. 在執行期間產生例外

解：C。當參數小於零的時候，`sqrt()`方法會回傳NaN(不是一個數字)。

10.

```
1. public class Degrees {
2.     public static void main (String [] args) {
3.         system.out.println( Math.sin(75) );
4.         system.out.println( Math.toDegrees (Math.sin(75) ) );
5.         system.out.println( Math.sin(Math.toDegrees (75) ) );
6.         system.out.println( Math.toRadians (Math.sin(75) ) );
7.     }
8. }
```

請問哪行會輸出 75 度正弦值？

- A. 第 3 行
- B. 第 4 行
- C. 第 5 行
- D. 第 6 行
- E. 第 3 行、第 4 行、第 5 行或第 6 行中的任何一行
- F. 以下皆非

解：C。Math 類別的三角函數方法接受的都是弧度參數，而不是度數。第 5 行內容可解釋為：將 75 度轉成弧度，然後求該值。

## 7. 物件與集合

### 7.1 覆寫 hashCode( )與 equals( )

#### 7.1.1 實作 toString()方法

#### 7.1.2 覆寫 equals( )的方法

#### 7.1.3 覆寫 hashCode( )的方法

### 7.2 記憶體回收

#### 7.2.1 記憶體回收與記憶體管理的概論

#### 7.2.2 Java 的記憶體回收行程概論

#### 7.2.3 撰寫讓物件明確合適記憶體回收的程式

#### 7.2.4 在執行記憶體回收行程前的清理工作—finalize( )方法

### ◎ 考題分析

## 7.1 覆寫 hashCode( )與 equals( )

你將基本資料型別排除在外的話，Java 中的任何東西都可當做物件看待，也包括以大寫「O」開頭的 Object 在內。從 java.lang.Object 繼承而來的每項例外、每個陣列、每個事件，都是一個物件。

方法	說明
Boolean equals(Object obj)	判斷兩個物件在意義上是否同等。
Void finalize ( )	當記憶體回收行程發現某個物件無法被參考時，則這個物件會被該行程所呼叫。
int hashCode()	回傳雜湊碼的 int 數值給某個物件，讓該物件可以在使用雜湊方法的 Collection 類別中應用，這些類別包括 Hashtable、HashSet。
final void notify()	喚醒某個正等待將這個物件鎖住的執行緒。
final void notifyAll()	喚醒某個正等待將這個物件鎖住的執行緒。
final void wait()	讓目前的執行緒處於等待的狀態，直到另一個執行緒呼叫這個物件上的 notify 或 notifyAll 為止
String toString()	回傳該物件「文字敘述」的內容。

### 7.1.1 實作 toString( )方法

**toString( )方法** 當你想閱讀某個類別中的某個物件內容時，請覆寫 toString( )方法。例如：某物件參考傳給 System.out.println( )方法時，該物件的 toString( )方法會被呼叫，而回傳的內容就是：

```
public class HardToRead {
    public static void main (String [ ] args) {
        HardToRead h = new HardToRead( );
        System.out.println(h); //此行會自動呼叫 toString( )
    }
}
```

執行 HardToRead 類別後，結果會出現雜湊碼：

```
% java HardToRead
HardToRead@47e0
```

覆寫類別中的 toString( )方法，底下就是範例：

```
public class BobTest {
    public static void main (String [ ] args) {
        Bob f = new Bob( "GoBobGo" , 19);
        System.out.println(f);
    }
}
class Bob {
    int shoeSize;
    String nickname;
    Bob (String nickname, int shoeSize) {
        this. shoeSize = shoeSize;
        this. String nickname = nickname;
    }
    public String toString( ) {
        return( "I am a Bob, but you call me" + nickname + ", My shoe size
is" + shoeSize);
    }
}
```

這段程式的輸出結果比較有意義(顯示字串)：

```
%java BobTest
I am a Bob, but you call me GoBobGo. My shoe size is 19
```

### 7.1.2 覆寫 equals( )的方法

當你需要知道兩個物件參考是否相等時，可使用 == 運算子。但，當你需要知道這些物件的變數是否相等時，則必須使用 equals( )方法。假設有一個叫 Car



的類別，變數指的是車子的屬性，如製造商、款式、年份、組態等。你當然不希望只因為另一部車子具有一樣的屬性，就將這兩部車子視為相同。如果兩個參考對象都是某一部車子，你會知道它們所指的是同一部車，而不是兩部具有相同屬性的車輛。

### 不覆寫 equals( ) 的方法的意義

這裡隱藏著一個限制條件。你不覆寫 equals( ) 的方法的話，將無法在雜湊碼中將該物件當做索引值使用。除非你覆寫 equals( ) 的方法，否則當兩個參考的對象都是相同的物件時，它們才會被視為相等。

當然，覆寫 Car 的 equals( ) 的方法也有不好的地方。此時，可能會有多個的物件，同時代表某部車子。在你設計程式時，這可能不是安全的做法。

### equals( ) 的方法的實作

```
public boolean equals(Object o) {
    if ((o instanceof Moof) && ((Moof)o).getMoofValue() ==
this.moofValue)) {
        return true;
    } else {
        return false;
    }
}
```

首先，必須觀察所有與覆寫相關的規則。第一行程式裡，宣告了一個有效的覆寫 equals( ) 的方法，是由 Object 繼承來的。

第二行程式可說是程式作業的核心。我們必須執行兩件事，才能進行有效的

相等關係比較。

1. **確定要測試的物件屬於正確的型態**：這指的就是 Object 型態。所以你須對它執行 instanceof 的測試。
2. **比較屬性**：只有開發人員可決定能將兩個物件實體視為相等的條件。

### equals( )的方法

請特別記住，equals( )、hashCode( )與 toString( )等方法都是公用的 (public)。底下的程式範例並不是合法覆寫 equals( )的方法，雖然看起來好像是對的：

```
class Foo {  
    boolean equals(Object o) { }  
}  
public void run( ) { }
```

另外，也請注意其中的參數型態。底下是過載 equals( )的方法的例子，而不是覆寫 equals( )的方法：

```
class Boo {  
    public boolean equals(Boo b) { }  
}
```

請務必熟悉覆寫規則，因此可分辨出從 Object 繼承來的方法，是否被覆寫、過載或在類別中被非法重新宣告。

equals( )方法的合約具有底下特性：

- **反身性(reflexive)**：對任何的參考數值 x 來說，x.equals(x)回傳的值應該是 true。
- **對稱性(symmetric)**：對任何的參考數值 x 與 y 來說，若 y.equals(x)回傳

的值是 true 的話，則 `x.equals(y)` 回傳的值應該是 true。

- **遞移性(transitive)**：對任何的參考數值 `x`、`y` 與 `z` 來說，如果 `x.equals(y)` 回傳的值是 true，且 `y.equals(z)` 也回傳的值是 true 的話，則 `x.equals(z)` 回傳的值應該也要為 true。
- **一致性(consistent)**：對任何的參考數值 `x` 與 `y` 來說，假設在該物件上進行相等關係的比較時，使用的資訊未做修改的話，對 `x.equals(y)` 進行多次出的呼叫回傳也應該都要為 true。

這份合約指出，如果兩個物件以 `equals()` 方法判斷為相等的話，則它們必須擁有相等的雜湊碼。所以，如果你覆寫了 `equals()` 方法，也一定要覆寫 `hashCode()` 方法。

### 7.1.3 覆寫 `hashCode()` 的方法

#### 認識雜湊碼的意義

地板上擺著一排收藏盒，有人將紙條交給你，上面寫著一個名字。你根據 A 是 1、B 是 2... 的法則，將這名子中的每個字母換算成一個整數值。然後，再把所有整數值加總。特定的名子經過這樣計算後，每次都會產生同樣結果。下圖中我們可以看到幾個計算例子。我們所採用的法則是，當給定某個輸入內容時，它永遠依照相同的邏輯執行。

好的雜湊碼擷取作業可分為兩個步驟：

1. 找到正確的收藏盒。
2. 從這收藏盒中找尋正確元素。

#### 實作 `hashCode()`

```
class HasHash {  
    public int x;
```

```

    hashCode(int xVal) {
        x = xVal;
    }
    public boolean equals(Object o) {
        hashCode h = (hashCode) o;
        if (h.x == this.x) {
            return true;
        } else {
            return false;
        }
    }
    public int hashCode() {
        return (x * 17);
    }
}

```

因為 equals( ) 方法在兩個物件擁有相同的 x 值時，會將它們視為相等，所以我們必須確定，帶有相同 x 值的物件，會回傳相等的雜湊碼。

對於不管是否相等的所有物件實體，都回傳相同數值的 hashCode( )，依然是合法的 hashCode( ) 方法！舉例說：

```

public int hashCode( ) {
    return 1492;
}

```

這段程式碼並不違反合約。X 值都為 8 的兩個物件，會有相同的雜湊碼。不過回傳唯一的雜湊碼的方法仍會被視為正確的，因為它並沒有違反規定。再次提醒，語法正確並不表示它就是好語法。

### hashCode( ) 的方法的合約

- 在執行 Java 應用程式過程中，每當對同一個物件啟動一次以上的 hashCode( ) 的方法時，必須前後一致回傳相同整數值。
- 如果根據 equals(Object) 方法判定兩個物件不相等的話，任何的物件在呼叫

hashCode( )的方法時，必須產生相同結果。

- 根據 equals(java.lang.Object)方法判定兩個物件不相等的話，任何的物件在呼叫 hashCode( )的方法時，不一定要產生不同整數結果。

以上所指的意思是說：

條件	必要的	不一定需要的
x.equals(y) == true	x.hashCode() == y.hashCode()	
x.hashCode() == y.hashCode()		x.equals(y) == true
x.equals(y) == false		不需要 hashCode()
x.hashCode() != y.hashCode()	x.equals(y) == false	

所以，在看看還有哪些因素，會造成 hashCode()方法的失敗：

- 提供一些狀態給某個物件(數值指派給它的變數)。
- 該物件放入 HashMap 中，並將該物件當作索引值使用。
- 使用物件序列化的方式，將該物件儲存到某個檔案中，但並不改變它的狀態。
- 透過還原序列化的作業，從該檔案中擷取這個物件。
- 使用序列化的物件(將它帶回堆疊之上)，將它從 HashMap 中取出。

過渡變數真的會弄亂你在 hashCode( )和 equals( )方法上的實作。解決方法可以將變數宣告為非過渡變數性質，或者，必須標示為過渡，則不要使用它來決定物件的雜湊碼。

## 7.2 記憶體回收

### 7.2.1 記憶體回收與記憶體管理的概論

對許多類型的應用程式來說，記憶體管理都是很重要的元素。典型的

設計方式會先將資料讀入記憶體中的某種元件集合內，在這些資料執行一些作業，最後再將這些資料存入記憶體中。將資料寫入後，在處理下一批資料之前，將舊資料清空、刪除、或進行重建的作業，這些作業可能會做好幾千次。如果這些邏輯中出了小問題的話，可能會讓數量不多的記憶體，發生不當回收或遺漏的現象。這稱為記憶體遺漏，再好幾千次的迭代之後，可能會讓許多的記憶體無法再次被讀取，最後可能會照成程式當掉的情況。

### 7.2.2 Java 的記憶體回收行程概論

所有的記憶體回收的討論，都會以確定堆疊是否擁有足夠的可用空間為中心。要點就是刪除任何不再被執行中的 Java 程式所讀取的物件。當記憶體回收行程執行時，它的目的是要找出並刪除無法被讀取的物件。

#### 記憶體回收行程會在什麼時候執行？

Java 虛擬機器控制會決定何時需要執行記憶體回收行程。通常它會在偵測到可執行的記憶體數量降低時，開始執行記憶體回收行程。

#### 記憶體回收行程是如何作業的？

記憶體回收行程使用的是一套「標記清掃」的法則。現在可以說，當沒有活著的執行緒可以存取某個物件的時候，它就合適執行記憶體回收行程。

#### Java 程式是否有可能用光記憶體呢？

答案是肯定的。物件不再被使用的時候，記憶體回收系統會將它們從記憶體中移除。假如你保留太多活著的物件時，可能會將記憶體用完！

### 7.2.3 撰寫讓物件明確合適記憶體回收的程式

### 將參考變成 null 的做法

要將某個物件的參考移除的話，第一種方式就是設定參考變數，讓這個物件參考 null。

```
public class GarbageTruck {
    public static void main (String [ ] args) {
        stringBuffer sb = new StringBuffer ( "hello" );
        system.out.println(sb); //此時，StringBuffer 物件並不合適進行
        記憶體回收。
        sb = null; //現在 StringBuffer 則適合記憶體回收。
    }
}
```

為了讓它適合回收，我們會將參考變數 sb 設為 null，這樣會將對象為 StringBuffer 物件的參考變數移除掉。

### 重新指定參考變數

也可以透過更改變數的參考對象，解除參考變數與某個物件間的關聯。

```
class GarbageTruck {
    public static void main (String [ ] args) {
        stringBuffer s1 = new StringBuffer ( "hello" );
        stringBuffer s1 = new StringBuffer ( "goodbye" );
        system.out.println(s1);
        //此時，數值為「hello」的 StringBuffer 物件還不合適進行記憶體回
        收。
        s1 = s2; //重新設定 s1，讓它參考數值「goodbye」的物件。
        //現在，數值為「hello」的 StringBuffer 則適合記憶體回收。
    }
}
```

一旦這個方法回傳結果之後，它所配置的物件就適合做記憶體的回收。

### 孤立參考的做法

另外一種方式，即使它還存在有效的參考也不例外！

```

public class Island {
    Island i;
    public static void main (String [ ] args) {
        Island i2 = new Island();
        Island i3 = new Island();
        Island i4 = new Island();
        i2.i = i3;
        i3.i = i4;
        i4.i = i2;
        i2 = null;
        i3 = null;
        i4 = null;
        // 執行複雜且密集使用記憶體的工作
    }
}

```

三個 Island 物件都有實體變數，讓它們彼此參考。不過，它們與外界的聯繫已變成 null。

#### 7.2.4 在執行記憶體回收行程前的清理工作—Finalize( )方法

這段程式在名為 Finalize( )方法之中，所有類別都從類別 Object 的地方繼承這個方法。你在類別中已被覆寫的 Finalize( )方法內加入的任何程式，都不保證一定會被執行。一般來說，你根本不要去覆寫 Finalize( )的方法。

##### Finalize( )方法的一些小技巧

- 對任何特定的物件來說，記憶體回收行程只會呼叫一次 Finalize( )方法。
- 實際上，呼叫 Finalize( )方法可以讓某物件免於被刪的命運。

問題	對策
我想分配某物件，並確定它永遠不會被解除分配。我可以告知記憶體回收行程忽略某物件嗎？	不行！Java 沒有提供這樣機制，可以讓某物件不被刪除。
我的程式執行不如我預期的好。我認為記憶體回收行程佔太多時間，我可以怎樣改善它呢？	首先，程式中一定建立了許多對象為暫時性的物件參考，又在程式中把它刪除。嘗試重新設計這段程式，讓它能重覆使用物件，或只需要更少的暫時性物件。。



我正在某個方法中建立物件，並將它傳出做為這個方法的結果。我該如何確定在這個方法回傳結果之前，該物件不會被刪除？	再最後一個對象為該物件的參考物件被刪除，這物件都不會被除去。
某個物件被我的類別的組件參考的話，我該如何除去對這物件的參考呢？	將該組件設定為 null。或你對一個新物件設定參考，舊物件就會失去參考。
只要物件不影響的分配，我都希望讓物件繼續保留下來。我是否可以要求 Java，如果可用的記憶體變少時，對我提出警告呢？	在 Java1.2 之前，你必須自己檢查可使用的記憶體數量。

### ◎ 考題分析

1. 假設你看到底下這段程式

- ```

11. x = 0 ;
12. if( x1.hashCode( ) != x2. hashCode( ) ) x = x+1 ;
13. if( x3.equals(x4) ) x = x+10 ;
14. if( !x5.equals(x6) ) x = x+100 ;
15. if( x7.hashCode( ) == x8. hashCode( ) ) x = x+1000 ;
16. System.out.println ( "x = " + x );

```

同時假設 equals( ) 與 hashCode( ) 方法都做了適當的實作，如果輸出結果是「x = 1111」，則底下的敘述何者是永遠正確的？

- A. x2.equals(x1)
- B. x3.hashCode( ) == x4.hashCode( )
- C. x5. hashCode( ) != x6. hashCode( )
- D. x8.equals(x7)

解：答案是 B。程式中寫明輸出結果皆為 true，在此狀態下什麼是永遠正確。

看到 A. 當 x1.hashCode( ) != x2. hashCode( ) 則 x2.equals(x1) 會不相等，所以 A. 答案錯誤。C. 答案 x5.equals(x6) 時必須會有 x5. hashCode( ) == x6. hashCode( ) 這邊即使是 !x5.equals(x6)，答案不見得會是 x5. hashCode( ) != x6. hashCode( )。D. 答案錯誤，當 x7.hashCode( ) == x8. hashCode( ) 時，不見得要有 x8.equals(x7)。

2.

```
Class Test1 {
    Public int value ;
    Public int hashCode( ) { return 42 ; }
}
Class Test2 {
    Public int value ;
    Public int hashCode( ) { return (int) (value^5) ; }
}
```

底下的敘述中何者是正確的？

- A. 類別 Test1 將無法編譯。
- B. Test1 的 hashCode( ) 方法會比 Test2 的 hashCode( ) 方法更有效率。
- C. Test1 的 hashCode( ) 方法會比 Test2 的 hashCode( ) 方法更沒有效率。
- D. 類別 Test2 將無法編譯。
- E. 這兩個 hashCode( ) 方法的效率會是一樣的。

解：答案是 C。類別 Test1 所回傳的值會是直接回傳 42，這邊會將雜湊碼的紀錄放在一個收藏盒內，因此會是最沒有效率的方法。

3. 假設已經適當地覆寫了 equals( ) 與 hashCode( ) 方法，則關於比較同類別的兩個物件實體時，底下的敘述中哪兩個是正確的？（請選擇兩個答案）

- A. 如果 equals( ) 方法回傳的是 true，則 hashCode( ) 方法的比較運算子 == 一定會回傳 true。
- B. 如果 equals( ) 方法回傳的是 false，則 hashCode( ) 方法的比較運算子 != 一定會回傳 true。
- C. 如果 hashCode( ) 方法的比較運算子 == 回傳的是 true，則 equals( ) 方法一定會回傳 true。
- D. 如果 hashCode( ) 方法的比較運算子 == 回傳的是 true，則 equals( ) 方法有可能會回傳 true。
- E. 如果 hashCode( ) 方法的比較運算子 != 回傳的是 true，則 equals( ) 方法有可能會回傳 true。

解：答案是 A. 跟 D. 。B. 答案 equals( ) 方法回傳的是 false，則 hashCode( ) 方法的比較運算子 != 不一定會回傳 true。C. 答案如果 hashCode( ) 方法的比較運算子 == 回傳的是 true，則 equals( ) 方法不一定會回傳 true。如果 hashCode( ) 方法的比較運算子 != 回傳的是 true，則 equals( ) 方法有可能會回傳 true，這邊敘述是錯誤的，當 hashCode( ) 方法的比較運算子 != 回傳的是 true，equals( ) 方法會回傳 false。

4. 底下的類別都從來類別 Object 直接繼承而來，它們之中的哪個類別沒有覆寫 equals( ) 與 hashCode( )？

- A. java.lang.String
- B. java.lang.Double
- C. java.lang.StringBuffer
- D. java.lang.Character
- E. java.util.ArrayList

解：C.，在答案中，java.lang.StringBuffer 是唯一有用到類別 Object 所提供的預設方法的類別。

5. 底下有關適當覆寫 equals( ) 與 hashCode( ) 方法的敘述中，哪兩個是正確的？

- A. 如果 equals( ) 方法被覆寫的話，則 hashCode( ) 方法並不需要被覆寫。
- B. 如果 hashCode( ) 方法被覆寫的話，則 equals( ) 方法並不需要被覆寫。
- C. hashCode( ) 方法可以永遠回傳相同值，不管啟動它的物件為何。
- D. 如果兩個意義不等價的物件，同時啟動 hashCode( ) 方法，則 hashCode( ) 方法對這兩個呼叫不能同時回傳相同數值。
- E. 在比較不同物件的時候，equals( ) 方法可以回傳 true。

解：C. 與 E.，A 與 B 不正確是因為根據合約內容，除非 equals( ) 與 hashCode( )

都覆寫，否則，兩者都不能被覆寫。D. 錯誤是因為則是因為對不相似的物件進行雜湊作業時，hashCode( )往往回傳相同數值。

6. 哪個元件集合的類別可讓你增加或減少它的大小，並對它的元素提供索引式的存取，但它的方法不是同步的？

- A. java.util.HashSet
- B. java.util.LinkedHashSet
- C. java.util.List
- D. java.util.ArrayList
- E. java.util.Vector

解：答案為 D.，所有元件集合的類別都可讓你增加或減少它的大小。ArrayList 會提供索引值給它的元素，比較新的類別傾向不提供同步方法。Vector 是 ArrayList 比較舊的方法，它具有同步，會比 ArrayList 慢一些。

7. 哪個元件集合的類別可讓你透過將索引值與元素的數值產生關聯，而存取它的元素，並提供同步化的功能？

- A. java.util.SortedMap
- B. java.util.TreeMap
- C. java.util.TreeSet
- D. java.util.HashMap
- E. java.util.Hashtable

解：答案是 E，此選單中僅有 Hashtable 提供同步化的類別。

8.

- 12. TreeSet map = new TreeSet( );
- 13. map.add( "one" );
- 14. map.add( "two" );
- 15. map.add( "three" );
- 16. map.add( "four" );
- 17. map.add( "one" );

```

18. Iterator it = map.iterator( );
19. while ( it.hasNext( ) ) {
20.     System.out.print( it.next ( ) + " ");
21. }

```

請問執行後互得到什麼結果？

- A. one two three four
- B. four three two one
- C. four one three two
- D. one two three four one
- E. one four three two one
- F. 列印的順序不一定。

解：答案為 C。TreeSet 為排序性且不重複，會依字母的順序進行排序。

9. 哪個元件集合的類別可讓你將它的元素數值與索引值產生關聯，並讓你先進先出的順序擷取物件？

- A. java.util.ArrayList
- B. java.util.LinkedHashMap
- C. java.util.HashMap
- D. java.util.TreeMap
- E. java.util.LinkedHashSet
- F. java.util.TreeSet

解：答案為 B，LinkedHashMap 提供快取的功能，先進先出是另一種表達快取行為的方式，若要以快取順序擷取 LinkedHashMap 元素的話，需用 values() 方法，並對結果的元件集合執行迭代的作業。

10.

```

1. public class X {
2.     public static void main(String [ ] args) {
3.         X x = new X( );
4.         X x2 = ml(x);
5.         X x4 = new X( );
6.         x2 = x4;
7.         doComplexStuff( );
8.     }

```

```
9.      static X m1(X mx) {
10.          mx = new X( );
11.          return mx;
12.      }
13.  }
```

在第 6 行執行之後，會有多少物件適合進行記憶體回收的作業？

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

解：正確答案為 B。在第 6 行執行之後，為一沒有參考的物件。就是第 4 行程式所產生的結果，參考變數 x 不會受到 m1( ) 方法的影響。

## 8. 內部類別

### 8.1 內部類別(Inner Class)

#### 8.1.1 設計正常的內部類別

#### 8.1.2 從內部類別之內參考內部或外部實體物件

### 8.2 方法區域內部類別

### 8.3 匿名內部類別

### 8.4 靜態巢狀類別

#### 8.4.1 實體化靜態巢狀類別

### ◎ 考題分析

## 8.1 內部類別

### 8.1.1 設計正常的內部類別

- 1、在這裡，我們將「正常的」用來表示不屬於底下這幾種的內部類別：
  - 靜態
  - 方法區域
  - 匿名
- 2、在定義內部類別時，你會把它放在外部類別的大括號之內。
- 3、使用內部類別的好處在於可以直接存取外部類別的私有成員。
- 4、內部類別會在另一個類別的大括號內。
- 5、內部類別是外部類別完全合法的成員之一，所以它能使用存取修飾詞與 `abstract` 或 `final` 修飾詞。
- 6、內部類別也是外部類別的成員，所以其它成員可以存取或呼叫內部成員變數和方法，就算宣告成 `private` 也一樣可以；反之，內部類別的方法也可以直接存取其它成員變數和呼叫成員方法。
- 7、內部類別也可以使用 `public`，`protected`，`private` 修飾詞。
- 8、外部類別之內的程式，你可以只使用內部類別的名稱將它實體化，如下：

```
MyInner mi = new MyInner();
```

#### 從內部類別之內參考外部或外部物件實體的作法

最常見到的情形是，由外部類別產生內部類別的物件實體，因為通常是外部類別想要將內部的物件實體當作協助使用者使用，這完全是私人使用的性質。我



們將修改 MyOuter 類別，產生 MyInner 的某個物件實體，

如下：

```
Class MyOuter {
    private int x = 7;
    public void makeInner() {
        MyInner in = new MyInner();
        in.seeOuter();
    }
    class MyInner{
        public void seeOuter() {
            System.out.println(“Outer x is “ + x);
        }
    }
}
```

### 8.1.2 從內部類別之內參考內部或外部物件實體

通常，某個物件會怎樣參考它的本身？答案是使用 this 參考，底下是對 this 的快速複習：

- 關鍵字 this 只可以從物件實體程式之內使用。換言之，不能從靜態程式之內使用。
- this 是對目前執行中的物件所做的參考。換言之，該物件的參考會用來啟動目前執行中的方法。
- this 參考物件可以將參考方法的參數，傳給本身或其它程式的方法：

```
public void myMethod() {
    MyClass mc = new MyClass();
    mc.doStuff(this); //將參考傳給執行 myMethod 的物件。
```

在內部類別程式之內，this 參考的對象會是內部類別的物件實體，因為 this 參考的永遠都是目前執行中的物件。不過，萬一內部類別程式想要明確參考這個

內部物件實體所結合的外部類別物件實體時，會發生什麼結果呢？換言之，你該如何參考「外部的 this」呢？在正常的情況下，內部類別程式並不會需要參考外部類別，因為它已經有一個隱式的參考，可以用來讀取外部類別的組件。不過，如果它需要將這個參考傳給其它程式，就需要參考外部類別，如下所示：

```
class MyInner {
    public void seeOuter() {
        System.out.println( "Outer x is " + x);
        System.out.println( "Inner class ref is " + this);
        System.out.println( "Outer class ref is" + MyOuter.this);
    }
}
```

如果我們執行底下這整段的程式之後：

```
class MyOuter {
    private int x = 7;
    public void makeInner() {
        MyInner in = new MyInner();
        in.seeOuter();
    }
}
class MyInner {
    public void seeOuter() {
        System.out.println( "Outer x is " + x);
        System.out.println( "Inner class ref is " + this);
        System.out.println( "Outer class ref is" + MyOuter.this);
    }
}
public static void main (String[] args) {
    MyOuter.MyInner inner = new MyOuter().new MyInner();
    inner.seeOuter();
}
```

輸出為：

```
Outer x is 7
```

```
Inner class ref is MyOuterMyInner@113708
Outer class ref is MyOuter@33fld7
```

所以當內部類別需要參考本身，或外部物件實體的時候，相關規定如下：

- 如果要參考內部類別物件實體的本身，要從內部類別的程式之內執行，並應該使用 `this`。
- 如果要從內部類別的程式之內參考「外部的 `this`」(外部類別的物件實體)，則應該使用 `<NameOfOuterClass>.this`(例如，`MyOuter.this`)。

## 8.2 方法區域內部類別

如果你確實想要使用這個內部類別的話(假設是要啟動它的方法)，則必須在這個方法的內部，但是要在內部類別定義的底下，建立它的一個物件實體。底下這段程式是合法的，它所顯示的是如何實體化並使用該方法區域的內部類別：

```
class MyOuter2 {
    private String x = "Outer2" ;
    void doStuff() {
        class MyInner {
            public void seeOuter() {
                System.out.println( "Outer x is " + x);
            } //結束內部類別方法的定義。
        } //結束內部類別的定義
        MyInner mi = new MyInner(); //這行程式必須放在類別之後
        mi.seeOuter();
    } //結束外部類別方法 doStuff()的定義。
} //結束外部類別的定義。
```

### 方法區域內部類別可執行與不可執行的工作

只有在定義內部類別的方法之內，才可以實體化該方法區域內部類別。換言之，在任何其它方法中所執行的程式，不管它是在內部類別之內或之外，都不能

實體化該方法區域內部類別。就像正常的內部類別物件一樣，方法區域內部類別物件也與外部類別物件，共享一種特殊的關係，而且也可以讀取私有(或任何其它的)組件。不過內部類別物件不能使用內部類別所在方法中的區域變數。

### 8.3 匿名內部類別

前面已經看過在外部類別之內，以及某個方法之內定義類別的做法。這是在 Java 中看過最不尋常的用法，也就是不以任何類別名稱宣告的內部類別(因此取名「匿名」)，你甚至不只可以在方法中定義這些類別，而改在方法的參數中定義。(版本分為兩種，下面將會介紹)

#### 匿名內部類別 第一種類型

```
class Popcorn {
    public void pop() {
        System.out.println( "popcorn" );
    }
}
class Food {
    popcorn p = new Popcorn() {
        public void pop() {
            System.out.println( "anonymous popcorn" );
        }
    };
}
```

以上這段程式，有下面這些奇特的地方：

- 程式中定義了兩個類別，Popcorn 跟 Food
- Popcorn 擁有一個方法，叫做 pop()
- Food 則擁有一個實體變數，以 Popcorn 的型態宣告

● Food 的內容就是這些，它並沒有任何方法

以下是程式的解析：

Popcorn 參考變數所參考的對象不是 Popcorn 的物件實體，而是 Popcorn 匿名子類別的物件實體。以下是有關匿名的程式碼：

```
2. popcorn p = new Popcorn() {  
3.     public void pop() {  
4.         System.out.println( "anonymous popcorn" );  
5.     }  
6. };
```

第二行：這段程式以實體變數的宣告開始，這個變數的型態為 Popcorn。程式解釋大約如下：宣告一個名為 p 的參考變數，它的型態為 Popcorn。然後再宣告一個新的類別，這個類別沒有名稱，但它是 Popcorn 的一個子類別。接著出現一個大括號，用來開啟類別的定義。

第三行：實際上，這行程式是新類別定義中的第一個敘述。它的作用是覆寫超類別的 Popcorn 的 pop() 方法。這是設計內部匿名類別時的重點，也就是覆寫超類別的一個或多個方法。

第四行：這行程式是覆寫 pop() 方法的第一個敘述，其中並沒有任何特殊的地方。

第五行：這行程式是結束 pop( ) 方法的大括號。

第六行：這行的部分，是最需要留意的地方。這行程式包含了一個大括號，用以結束匿名類別的定義。不過，其中還有其它的內容。第六行部份還有一個分號，用來結束從第二行程式開始的敘述，這個敘述的作用式宣告 Popcorn 參考變數，並將它初始化。

## 匿名內部類別 第二種類型

第一種與第二種類型之間的唯一差別，就在於產生特定類別型態的匿名子類別的方法，第二種類型會建立特定介面型態的匿名實作。在前面的例子中，我們會以底下的方式，定義型態為 Popcorn 的新匿名子類別：

```
Popcorn p = new Popcorn( ) {
```

但是，如果 Popcorn 是介面型態，而不是類別型態的話，新的匿名類別會變成介面的實作，而不是該類別的子類別。

```
interface Cookable {
    public void cook( );
}
class Food {
    cookable c = new Cookable( ) {
        public void cook( ) {
            System.out.println( "anonymous cookable implementer" );
        }
    };
}
```

以上這段程式，就像 Popcorn 的範例一樣，前面這段程式可以產生匿名的內部類別物件實體，只是這次所建立的會是 Cookable 介面的實作。

```
cookable c = new Cookable( ) {
```

這邊可以將它解釋成：宣告型態為 Cookable 的參考變數。它會參考來自於某個實作 Cookable 的類別，所以我們現在就要在這邊建立一個這樣的類別。我們不需要幫這個類別取名，但它會是實作 Cookable 的類別，同時，這個大括號會開始新實作類別的定義。

這邊還有一個與匿名介面實作有關的重點，那就是：它們可以實作一個介面。但，一個匿名內部類別甚至無法同時延伸某個類別與實作某個介面。內部類別必須選擇成為具名類別的子類別，而不直接實作任何的介面，或單一介面。簡單來說，就是如果這個匿名內部類別是某個類別型態的子類別，則它會自動轉成由該超類別實作的任何介面的實作之一。

### 由參數定義的匿名內部類別

以下，假設你要輸入一些內容，目的是要建立 Perfect 類別，當你撰寫了呼叫 Bar 物件上的方法時，而且這個方法接受型態為 Foo 的物件。

```
class MyWonderfulClass {
    void go() {
        Bar b = new Bar();
        b.doStuff(AckWeDon'tHaveAFoo!);
    }
}
interface Foo {
    void foof();
}
class Bar {
    void doStuff(Foo f) { }
}
```

這程式，除了無法得到來自於實作 Foo 的類別物件之外。你也無法實體化這樣的物件，因為你甚至沒有實作 Foo 的類別，更別說是它的物件實體。所以，首先需要的是實作 Foo 的類別，然後你還需要這樣類別的一個物件實體，將它傳給 Bar 類別的 doStuff() 方法。再來，只要在引參數內定義一個匿名內部類別，就可解決問題。

## 8.4 靜態巢狀類別

有時候，你會聽到別人將靜態巢狀類別稱為頂級巢狀類別，或是靜態內部類別。不過，根據內部類別的標準定義來看，它們實際上一點也不是內部類別。內

部類別都會與外部類別具有特殊關係，但是靜態巢狀類別則沒有這樣的特性。它只是在另一個類別範圍裡面內部的類別(也稱為「頂層」類別)。靜態巢狀類別只是外部類別的靜態組件中的一個類別而已，如下：

```
class BigOuter {
    static class Nested {}
}
```

這個類別的本身並不是真正「靜態的」，其實並沒有靜態類別這樣的東西。在這種狀態中，static 修飾詞的意思是說，巢狀類別是外部類別的靜態組件。這表示它可以像其它靜態組件一樣被讀取，而不需要具備外部類別的物件實體。

#### 8.4.1 實體化靜態巢狀類別

實體化靜態巢狀類別的語法，與正常的內部類別有點不同，它的寫法類似底下的程式：

```
class BigOuter {
    static class Nested { }
}
class Broom {
    public static void main (String [ ] args)
        BigOuter.Nested n = new BigOuter.Nested();
    }
}
```



◎ 考題分析

1.

```
public class MyOuter {  
    public static class MyInner {  
        public static void foo() { }  
    }  
}
```

如果放在 MyOuter 或 MyInner 以外的類別中，底下哪個敘述可以實體化這個巢狀類別的物件實體？

- A. `MyOuter.MyInner m = new MyOuter.MyInner() ;`
- B. `MyOuter.MyInner mi = new MyInner() ;`
- C. `MyOuter.MyInner m = new MyOuter() ;`  
`MyOuter.MyInner mi = m.new MyOuter.MyInner() ;`
- D. `MyInner mi = new MyOuter.MyInner() ;`

解：答案是 A。MyInner 是屬於靜態巢狀類別，它使用時必須使用完整的名稱

MyOuter.MyInner，才能進行實體化的作業。

B. 錯誤是因為它只有 new 內部類別，導致 MyOuter 外部類別部份無法使用。

C. 錯誤是因為，第 2 行部分只需要對內部類別作實體化作業就好，所以程式

部份就有錯誤。D. 錯誤是因為，前面變數宣告中，沒有宣告到外部類別的名

稱。

2. 底下有關靜態巢狀類別的敘述中，請問哪兩個是正確的？

- A. 你必須參考外部類別的某個物件實體，才能將它實體化。
- B. 它無法讀取外部類別中非靜態的組件。
- C. 它的變數與組件必須是靜態的。
- D. 它可以使用 `new MyOuter.MyInner()`；進行實體化的作業。
- E. 它必須延伸外部類別。

解：答案是 B. 跟 D。B. 是因為靜態巢狀類別並沒有跟外部類別結合，所以讀取

不到外部類別中的非靜態組件。D. 實體化靜態類別的語法，正確。

A. 錯誤的地方，是因為靜態類別中不需要參考外部類別的實體。C. 錯誤是因為，靜態類別中，宣告變數或組件可以是非靜態的。E. 錯誤是因為，不一定要延伸到外務類別，也就是說它可以不必延伸到外部類別也可使用。

3. 底下的敘述中，請問哪個可以建立匿名內部類別的實體？

- A. `Runnable r = new Runnable(){ };`
- B. `Runnable r = new Runnable(public void run() { });`
- C. `Runnable r = new Runnable{public void run() { }};`
- D. `Runnable r = new Runnable(){public void run{ }};`
- E. `System.out.println(new Runnable(){public void run(){ }})`
- F. `System.out.println(new Runnable{public void run(){ }})`

解：答案是 E。它定義的是匿名內部類別的實體，這意思說，它同時建立了一個新的匿名內部類別的實體。匿名類別是 `Runnable` 的實作器，所以它必須覆寫 `Runnable` 的 `run()` 的方法。  
A 錯是因為它沒覆寫 `Runnable` 的 `run()` 的方法。  
B C 與 D 錯都是因為語法錯誤。

4.

```
1. class Boo {
2.   Boo(String s) { }
3.   Boo() { }
4. }
5. class Bar extends Boo {
6.     Bar() {}
7.     Bar(String s) {super (s);}
8.     void zoo() {
9.         //在這裡插入程式碼
10.    }
11. }
```

請問哪兩行程式可以從類別 Bar 之內建立匿名內部類別?(請選兩個)

- A. Boo f = new Boo(24) { };
- B. Boo f = new Boo() { };
- C. Boo f = new Boo() { String s ;};
- D. Boo f = new Boo(String s) { };
- E. Boo f = new Boo.Bar(String s) { };

解：答案為 B 與 C。B 正確是因為當提及多型時候，匿名內部類別與其它並沒有什麼不同。這意思說你永遠可以宣告超類別型態的參數，並將參數的對象指派為超類別型態的某個實體。在這例子中 Bar 就是匿名子類別。因為 Bar 是 Boo 的子類別，所以可以正常執行。A 錯在它將 int 傳給 Boo 的建構子，但 Boo 沒有匹配的建構子。D 錯在它違反了多型的規則，不可以用子類別參考超類別的參考變數。E 錯在它的語法不正確。

5.

```
1.class Foo {  
2.  class Bar { }  
3. }  
4.class Test {  
5.  public static void main (String [ ] args) {  
6.      Foo f = new Foo ( );  
7.      //在這裡插入程式碼  
8.  }  
9. }
```

如果將底下的敘述個別插入第 7 行，請問哪個敘述可以建立 Bar 的實體?

- A. Foo.Bar b = new Foo.Bar ( );
- B. Foo.Bar b = f .new Bar ( );
- C. Bar b = new f . Bar ( );
- D. Bar b = f . new Bar ( );
- E. Foo.Bar b = new f.Bar ( );

解：答案為 B，它是正確的語法，在參考的宣告中，同時使用兩個名稱，然後使用對象為外部類別的參考，啟動這個內部類別的 new。

A 錯是因為它沒有使用對象為外部類別的參考，且在 new 中加了兩著的名稱，C 錯是因為沒有使用外部類別名稱，且 new 語法錯誤，D 是因為沒有使用外部類別名稱，E 錯是因為 new 語法錯誤。

6. 底下有關方法區域內部類別的敘述中，請問哪兩個是正確的？

- A. 它必須標示為 final
- B. 它可以標示為 abstract
- C. 它可以標示為 public
- D. 它可以標示為 static
- E. 它可以讀取外部類別的私有組件。

解：答案是 B 與 E，B 正確是因為方法區域內部類別可以是抽象的，如果這抽象類別需被使用的話(有可能都不會被使用到)，必須產生內部類別的子類別，E 正確是因為方法區域內部類別可以像任何其它的內部類別般作業，它與外部類別的物件實體間也有特殊關係，所以可以讀取外部類別所有組件。

A 錯是因為不一定要宣告為 final，C 與 D 錯是因為不能宣告為 public 或 static(無法將任何區域變數標示為 public)。

7. 底下有關匿名內部類別的敘述中，請問哪一個正確的？

- A. 它只可以延伸一個類別，並只可以實作一個介面。
- B. 它只可以延伸一個類別，但可以實作多個介面。
- C. 它只可以延伸一個類別，或只可以實作一個介面。
- D. 不管是可以延伸一個類別，它都可以實作多個介面。
- E. 如果不是延伸一個類別，則它都可以實作多個介面。

解：答案為 C 匿名內部類別無法同時執行這兩種作業，無法同時延伸某個類別，又實作某個介面，另外，它也不能實作一個以上的介面。

8.

```
public class Foo {
    Foo( ) {System.out.print( "foo" );}
    class Bar{
        Bar( ) {System.out.print( "bar" );}
        public void go( ) (System.out.print( "hi" );)
    }
    public static void main (String [ ] args) {
        Foo f = new Foo( );
        f.makeBar( );
    }
    void makeBar( ) {
        (new Bar( ) { }).go( );
    }
}
```

請問執行後會得到什麼結果？

- A. 編譯失敗
- B. 在執行期間產生錯誤
- C. foobarhi
- D. barhi
- E. hi

解：答案為 C，當第一個 Foo 的物件實體被建立，Foo 建構子可以執行，並印出「foo」，makeBar( ) 方法被啟動，產生一個 Bar，Bar 建構子可以執行，並印出「bar」，最後，在新的 Bar 實力上啟動 go( ) 方法，並印出「hi」。

9.

```
1. public class TestObj {
2.     public static void main (String [ ] args)
3.         Object o = new Object( ) {
4.             public boolean equals(Object obj) {
5.                 return true;
6.             }
7.         }
```

```
8.         System.out.println(o. equals ( "Fred" ));
9.     }
10. }
```

請問執行後會得到什麼結果？

- A. 在執行期產生例外
- B. turn
- C. fred
- D. 因為第 3 行程式的錯誤造成編譯失敗
- E. 因為第 4 行程式的錯誤造成編譯失敗
- F. 因為第 8 行程式的錯誤造成編譯失敗
- G. 因為第 3、4 或 8 行以外程式的錯誤造成編譯失敗

解：答案為 G 第 3 行程式是一個敘述，直到第 7 行才結束，而敘述必須以分號結束。

10.

```
1. public class HorseTest {
2.     public static void main (String [] args) {
3.         class Horse {
4.             public String name;
5.             public Horse(String s) {
6.                 name = s;
7.             }
8.         }
9.         Object obj = new Horse( "Zippo" );
10.        Horse h = (Horse) obj;
11.        System.out.println(h.name);
12.    }
13. }
```

請問執行後會有什麼結果？

- A. 在執行期中第 10 行程式會產生例外
- B. Zippo

- C. 因為第 3 行程式的錯誤造成編譯失敗
- D. 因為第 9 行程式的錯誤造成編譯失敗
- E. 因為第 10 行程式的錯誤造成編譯失敗
- F. 因為第 11 行程式的錯誤造成編譯失敗

解：答案為 B 在 HorseTest 類別中的程式內容是完全合法的。第 9 行程式會使用宣告為 Object 型態的參考變數，建立方法區域內部類別 Horse 的一個實體。

第 10 行程式會將 Horse 物件，強迫轉型為 Horse 參考變數，這可以讓第 11 行程式的編譯不會產生問題。如果第 10 行程式被刪除的話，HorseTest 的程式將無法編譯，因為類別 Object 並沒有一個叫做 name 的變數。

## 9. 執行緒(Thread)

### 9.1 執行緒的定義、實體化與啟動

#### 9.1.1 定義執行緒的作法

### 9.2 防止執行緒的執行

#### 9.2.1 休眠狀態

### 9.3 程式的同步化

#### 9.3.1 同步化與鎖定的作業

### 9.4 執行緒的互動

#### 9.4.1 notifyAll( )

### ◎ 考題分析



## 9.1 執行緒的定義、實體化與啟動

能同時使用 `java.lang.Thread` 與 `java.lang.Runnable`，撰寫定義、實體化與啟動新執行緒的程式。

### 執行緒的意義

在 java 中「執行緒」指的是兩件不同的事情：

- 類別 `java.lang.Thread` 的一個實體
- 自己的執行緒(`thread of execution`)

其實，`Thread` 的實體就是一個物件。就像 Java 中其它的物件一樣，它也有變數與方法，並且可以在堆疊上啟動。不過，自己的執行緒則是一個獨立的處理程序，它擁有自己的呼叫堆疊。在 Java 中，每個呼叫堆疊上都存在一個執行緒。或者，你也可以從它的反面來思考，就是每個執行緒都有一個呼叫堆疊。即使你在程式中並未建立任何新的執行緒，它們還是會啟動。

### 建立一個執行緒

在 Java 中，執行緒會以 `java.lang.Thread` 的物件實體開始作業。你可以在 `Thread` 類別中找到一些方法，用來管理建立、啟動與暫停這些執行緒的工作。

以上這些動作都會從 `run()` 方法開始。可以在個別執行緒中處理的程式，想像成是「要進行的工作」。換言之，你有一些工作必須要完成，像是：當程式中其它工作正在進行的同時，以背景方式下載其它資料。所以你真正想要完成的事情稱為工作的話，則執行緒就是拿來啟動它的程式碼。這些工作都會以 `run()` 方法開頭，如下：

```
public void run( ) {
```

```
//此處放插入工作的程式碼  
}
```

永遠都必須在個別執行緒中處理的程式碼，放在一個 `run()` 方法中。當然，這個 `run()` 方法會呼叫其它的方法。但是自己的執行緒，也就是新的呼叫堆疊，永遠都要透過啟動 `run()` 才能開始作業。因此，`run()` 方法在定義執行緒的工作時，你可以使用兩個類別的其中之一。

你可以採用下列兩個方式，來定義與實體化某個執行緒：

- 延伸 `java.lang.Thread` 類別
- 實作 `Runnable` 介面

### 9.1.1 定義執行緒的作法

為了讓程式在執行期間執行多樣工作，必須利用其它方法產生額外的執行緒來執行程式。可透過延伸 `Thread` 類別，或是實作 `Runnable` 介面的方式達成這樣的目標。

Java 程式語言中提供了二種方法來撰寫執行緒程式：

- 延伸 `java.lang.Thread` 類別
- 實作 `java.lang.Runnable` 介面

#### 延伸 `java.lang.Thread` 類別

利用延伸 `Thread` 類別方式建立執行緒是較簡易的方式，而當中 `Thread` 類別是繼承自 `java.lang.Object` 而來。

常用的 `Thread` 類別建構子：

|                                      |                                             |
|--------------------------------------|---------------------------------------------|
| Thread 建構子                           | 簡要說明                                        |
| Thread( )                            | 建立一個新的執行緒                                   |
| Thread(String name)                  | 建立一個新的執行緒，並設定名稱為 name                       |
| Thread(Runnable, target)             | 建立一個新的執行緒，並呼叫 target 物件的 run 方法             |
| Thread(Runnable target, String name) | 建立一個新的執行緒，並呼叫 target 物件的 run 方法，並設定名稱為 name |

Thread 類別提供三個屬性，可以用來決定執行緒的優先權：

| 欄位                       | 作用                 |
|--------------------------|--------------------|
| static int MAX_PRIORITY  | 執行緒的最高優先權值，為整數值 10 |
| static int MIN_PRIORITY  | 執行緒的最低優先權值，為整數值 1  |
| static int NORM_PRIORITY | 執行緒的預設優先權值，為整數值 5  |

常用的 Thread 類別方法：

| Thread 方法名稱        | 回傳值         | 簡要說明                                    |
|--------------------|-------------|-----------------------------------------|
| destroy( )         | void        | 銷毀一個執行緒                                 |
| getName( )         | String      | 此方法是一個 final，將會傳回執行緒的名稱                 |
| getPriority( )     | int         | 傳回執行緒的優先權                               |
| run( )             | void        | 執行執行緒的工作                                |
| sleep(long millis) | static void | 暫停目前正在執行中的執行緒                           |
| start( )           | void        | 將執行緒丟入 Runnable pool 中等待被 run( ) 方法呼叫執行 |
| stop( )            | void        | 停止目前正在執行的執行緒                            |
| suspend( )         | void        | 停止目前正在執行的執行緒                            |
| toString( )        | String      | 利用 String 資料型態傳回執行緒名稱，優先權以及執行緒群組名稱      |
| yield( )           | static void | 將目前正在執行中的執行緒先行暫停，讓與其它執行緒執行              |

當建立了 Thread 的子類別後，您必需實作 run( ) 方法。run( ) 方法是執行

緒開始執行時的起始點，以這點而言，它很類似 main 方法，您可以在 run( ) 方法中撰寫需要執行的程式。以下的範例程式為如何建立一個執行緒：

```
public class Thread1 {
    public static void main(String[] args) {
        Thread1 t1 = new Thread1( );
        t1.start( );
    }
}
class Thread1 extends Thread {
    public void run( ){
        System.out.println("執行緒啟動了!!!");
    }
}
```

編寫執行緒的程式時，您也可以直接將 start( ) 方法放在類別的建構子中。那麼，在該類別實體化後，執行緒會自動的執行，例如：以下的範例程式顯示不同的執行緒程式的寫法：

```
public class Thread2 {
    public static void main(String[] args) {
        Thread2At = new Thread2A( );
        System.out.println("主程式結束!!!");
    }
}
class Thread2A extends Thread {
    Thread2A( ){
        //如果有，初始化這邊的資料
        start( );
    }
    public void run( ){
        System.out.println("執行緒啟動了!!!");
    }
}
```

## 實作 java.lang.Runnable 介面

除了可以延伸 Thread 類別物件之外，也可以利用實作 Runnable 介面的方式來撰寫。

Runnable 介面的 Thread 類別建構子呼叫方式：

| 使用 Runnable 介面的 Thread 建構子           | 簡要說明                                                   |
|--------------------------------------|--------------------------------------------------------|
| Thread(Runnable target)              | 分派一個新的執行緒，target 物件實作 Runnable 介面以建立執行緒物件              |
| Thread(Runnable target, String name) | 分派一個新的執行緒，target 物件實作 Runnable 介面以建立執行緒物件，name 為執行緒的名稱 |

Thread 類別也就是實作 Runnable 介面，Thread 類別本身只是實作空的 run() 方法，程式中實作 Runnable 介面必須呼叫 Thread 類別中所屬的建構子，才能建立執行緒物件。

建立執行緒的方法是用實作 Runnable 介面的方式建立新的類別，在該類別之中實作 run 方法。以下的範例程式示範如何用這種方法建立執行緒：

```
public class Thread3{
    public static void main(String[] args){
        new Thread3A( );
        System.out.println("主程式結束!!!");
    }
}
class Thread3A implements Runnable{
    Thread t;
    Thread3A( ){
        t = new Thread(this);
        t.start( );
    }
    public void run( ){
```

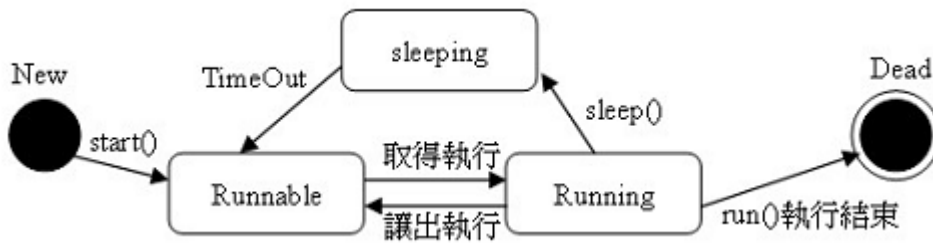
```
        System.out.println("執行緒啟動了!!!");
    }
}
```

## 執行緒的狀態

任何一個執行緒都只可以處於五種狀態之一：

- **新增(New)**：當記憶體中新增(new)了 Thread 物件實體時，此時在記憶體中的 Thread 物件只是先配置好一個區塊，還未分配到 CPU 時間。
- **準備執行(Runnable)**：當 Thread 執行緒物件呼叫 start( )方法時，便是將 Thread 物件丟入 Runnable pool 中準備執行。另一種讓執行緒處於等待、準備的狀態是呼叫 yield( )方法。在多執行緒程式中，藉由呼叫 yield( )方法可讓本身執行緒暫停執行，多出的 CPU 時間給其它執行緒執行。呼叫 yield( )時，雖然執行緒本身暫不執行，但仍屬可執行的狀態。當執行緒呼叫 yield( )方法時將會給予其它在 Runnable pool 中等待的執行緒有執行的機會。
- **執行中(Running)**：呼叫 run( )方法，將 Runnable pool 中準備執行的 Thread 物件轉到執行狀態，以執行 run( )方法中的程式。
- **等待/閉鎖/休眠(waiting/ blocked/ sleeping)**：非執行狀態是指執行緒的狀態為暫停。例如在程式中呼叫 sleep( )、wait( )及 suspend( )等方法都是讓本身執行緒暫停的方法，其運作方式都是讓執行緒先暫停一段時間之後才再回到執行狀態。
- **死亡(dead)**：Java 程式中下列 2 種情況會讓執行緒處於 dead 狀態：
  1. 執行緒將其工作執行完成(正常結束)；

2. 呼叫 `stop()` 方法，中止執行緒目前的動作(非正常結束)。  
當執行緒處於 `dead` 狀態後將無法再重新執行。



## 9.2 防止執行緒的執行

### 能辨識出防止執行緒執行的條件

將執行緒變成非執行的狀態，而不是停止某個執行緒。通常，某個執行緒已經停止，也就是它已經變成 `dead` 狀態。我們該了解的是：

- 休眠狀態
- 等待狀態
- 因為需要鎖定物件所產生的閉鎖狀態(`blocked state`)

### 9.2.1 休眠狀態

`sleep()` 是 `Thread` 類別的一個靜態方法。在程式中，你可以使用它「讓某個執行緒減速」，透過的方式是在回復為可執行緒狀態前，強制讓它進入休眠狀態。

你可透過執行靜態的 `Thread.sleep()` 方法，達到前述效果，假設以千分之一為單位的話，可使用底下的方式撰寫這段程式：

```
try {  
    Thread.sleep( 5*60*1000); //讓執行緒休息五分鐘  
} catch (InterruptedException ex) { }
```

## 9.3 程式的同步化

### 9.3.1 同步化與鎖定的作業

同步化作業的進行是使用物件鎖定(lock)的機制在運作，Java 中每個物件都有一把鎖，只有在這個物件使用同步化的方法時，它才會發揮作用，因為每個物件也都只有一把鎖，如果一個執行緒取得這把鎖的話，其它執行緒就無法再進入同步化的程式。

以下為鎖定與同步化的相關重點：

- 可以同步化的對象只有方法，而不包括變數
- 每個物件只有一把鎖
- 不是類別中的所有方法都必須同步化
- 兩個方法在某個類別中被同步化，則只能有一個執行緒可讀取這兩個方法的其中之一
- 類別同時包含同步化與非同步化的方法，多執行緒還是可以讀取其中的非同步化方法
- 執行緒變成休眠狀態的話，會帶著鎖一起進入休眠狀態
- 一個執行緒可以取得一把以上的鎖
- 可以對一段程式進行同步化，而不是針對特定的方法

### 靜態方法(static methods)



靜態方法也可以被同步化，需要保護的只是一份靜態資料，所以每個類別只需要一把鎖，就能將靜態方法同步化，這把鎖可以用在整個類別之內。

如果執行緒嘗試進入同步化的方法，而鎖已經被取走的話，稱為執行緒在物件上的鎖被閉鎖，基本上，該執行緒會進入這個特別物件的集中區，在裡面等待，直到這把鎖解開，這個執行緒才會再次回到可執行的狀態。

執行緒鎖死(deadlock)的狀況：當兩個執行緒被閉鎖，在有一方釋出鎖前，兩者都不能執行。

**鎖定狀態分別定義方法的類別：**

| 釋出鎖     | 保留鎖       | 定義方法的類別            |
|---------|-----------|--------------------|
| wait( ) | notify( ) | java. lang. Object |
|         | join( )   | java. lang. Thread |
|         | sleep( )  | java. lang. Thread |
|         | yield( )  | java. lang. Thread |

以上表格，釋出鎖：呼叫後，會立刻釋放所佔用的所有物件鎖，然後進入等待通知(waiting queue)。

保留鎖：呼叫後，並不會釋放所佔用的任何物件鎖。

#### 9.4 執行緒的互動

java. lang. Object 類別中有三個方法，分別是 wait( )、notify( )與 notifyAll( )，可以協助執行緒通知事件的狀態，等待與通知的機制可讓執行緒進入暫停狀態，直到其它執行緒通知它，回復到可執行狀態為止。注意：等待和通知機制中，必須從同步化的方法或程式段落內，呼叫 wait( )、notify( )與

notifyAll()，但執行緒不能在某個物件上啟動等待或通知的方法，除非它擁有這個物件的鎖。

wait()與 notify()都是 Object 的實體方法，與每個物件都有一把鎖一樣，每個物件也可以擁有一系列的執行緒，並讓它處於等待該物件發出訊息的狀態。

以下為一個物件等待另一個物件通知的例子：

```
1. class ThreadA {
2.     public static void main (String [] args) [
3.         ThreadB b = new ThreadB( );
4.         b.start( );
5.
6.         synchronized(b) {
7.             try {
8.                 System.out.println("Waiting for b to complete...");
9.                 b.wait( );
10.            } catch (InterruptedException e) { }
11.        }
12.        System.out.println("Total is:" +b.total) ;
13.    }
14. }
15.
16. class ThreadB extends Thread {
17.     int total;
18.
19.     public void run( ) {
20.         synchronized(this) {
21.             for(int i=0;i<100;i++) {
22.                 total += i;
23.             }
24.             notify( );
25.         }
26.     }
27. }
```

此段程式包含兩個帶有執行緒的物件:ThreadA 包含的是主執行緒，ThreadB 可算出從 0 到 99 的所有數字的合計，第 4 行呼叫 start( )後，ThreadA 會繼續處理在它自己類別中下行程式的內容，注意第 6 行程式物件 b 將自己同步化，這是為了要呼叫該物件上的 wait( )方法，ThreadA 必須擁有物件 b 的所有權。

```
synchronized(this) {  
    notify( );  
}
```

這段程式會通知任何目前正在等候這個物件的執行緒，需注意的是，如果呼叫 wait( )的這個執行緒不持有這把鎖的話，則會產生 IllegalMonitorStateException 的例外。

第 7 到 10 行的程式內容，其中 wait( )方法的外為有一個 try/catch 的區塊，等待中的執行緒也可能像休眠中的執行緒一樣被中斷。

當在某個物件上執行 wait( )方法的時候，執行這段程式的執行緒會立刻釋出這個物件上的鎖，不過當 notify( )被呼叫的時候，並不表示這個執行緒會在那個時後釋出它的鎖，如果這個執行緒還在處理同步化的程式，則這把鎖將不會被釋放出來，直到這個執行緒離開同步的程式為止，所以只有呼叫 notify( )方法，並不代表這把鎖會立刻可以使用。

#### 9.4.1 notifyAll( )

在大多數的案例中，對等待某個特定物件的所有執行緒，做統一呼叫可能會是比較好的做法，如果是的話可以在這個物件上使用 notifyAll( )，讓所有執行緒都離開等待區，並回復到可執行的狀態。

所有執行緒都會接到通知，並開始使用這把鎖，由於這把鎖會被每個執行緒

使用並釋出，但是這把鎖不會進一步的通知下一個執行緒。

許多執行緒有可能正在等待同一個物件，而且使用 `notify()` 只會影響其中之一，至於誰會收到這通知就不一定，要藉由 Java 虛擬機器的實作來決定。在可能有多個執行緒處於等待狀態的情況中，最好的做法就是使用 `notifyAll()`，以下為範例：

```
1. class Reader extends Thread {
2.     Calculator C;
3.
4.     public Reader (Calculator calc) {
5.         c = calc;
6.     }
7.
8.     public void run( ) {
9.         synchronized(c) {
10.            try {
11.                System.out.println("Waiting for calculation...");
12.                c.wait ( );]
13.            } catch (InterruptedException e) { }
14.        }
15.        System.out.println("Total is: " + c.total);
16.    }
17.
18.    public static void main (String [] args) {
19.        Calculator calculator = new Calculator( );
20.        new Reader(calculator).start( );
21.        new Reader(calculator).start( );
22.        new Reader(calculator).start( );
23.        calculator.start( );
24.    }
25. }
26. class Calculator extends Thread {
27.     int total;
28. }
29.     public void run( ) {
30.         synchronizer(this) {
```

```
31.         for(int i = 0; i < 100; i++) {
32.             total +=i;
33.         }
34.         notifyAll( );
35.     }
36. }
```

◎ 考題分析

1.

```
1. class MyThread extends Thread {
2.
3.     public static void main (String [ ] args) {
4.         MyThread t = new MyThread( );
5.         t.run( );
6.     }
7.
8.     public void run ( ) {
9.         for (int i=1;i<3;++i) {
10.            System.out.print(I + “. .” );
11.        }
12.    }
13. }
```

請問執行後會得到什麼結果？

- A. 由於第 4 行程式的內容，所以這段程式編譯失敗。
- B. 由於第 5 行程式的內容，所以這段程式編譯失敗。
- C. 1..2..
- D. 1..2..3..
- E. 在執行期產生例外

解：答案是 C。程式執行到第 5 行處，呼叫 run 方法，執行 run 方法，輸出結果

1..2..。由於，程式第 4 行、第 5 行處，宣告皆為正確，所以會正常執行。

2. 底下的方法中，哪兩個是在 Thread 類別定義的？

- A. start( )
- B. wait( )
- C. notify( )
- D. run( )
- E. terminate( )

解：答案是 A 跟 D。Thread 類別只定義到 start( ) 跟 run( )。

3. 底下的程式區塊會使用 Runnable 的物件建立一個 Thread

```
Runnable target = new MyRunnable( );  
Thread myThread = new Thread (target);
```

底下的類別中，哪個可以用來建立這個目標，讓前面的程式可以正確編譯？

- A. public class MyRunnable extends Runnable { public void run( ) {} }
- B. public class MyRunnable extends Object { public void run( ) {} }
- C. public class MyRunnable implements Runnable { public void run( ) {} }
- D. public class MyRunnable implements Runnable { void run( ) {} }
- E. public class MyRunnable implements Runnable { public void start( ) {} }

解：答案是 C，C 是正確的語法。A 錯誤是因為，Runnable 介面只能實作。B 錯誤是因為，你是要針對 Runnable 介面來建立 Thread，這邊它不是實作 Runnable 介面，所以當你編譯時，會產生錯誤。D. 錯誤是因為，run( ) 方法必須要宣告為 public。E 錯誤是因為實作 Runnable 介面，要的是實作 run( ) 方法，而不是 start( ) 方法。

4 .

```
1. class MyThread extends Thread {  
2.  
3.     public static void main (String [ ] args) {  
4.         MyThread t = new MyThread( );  
5.         t.start( );  
6.         System.out.print( "one.  " );  
7.         t.start( );  
8.         System.out.print( "two.  " );  
9.     }  
10.  
11.     public void run ( ) {  
12.         System.out.print( "Thread  " );  
13.     }  
14. }
```

請問執行後會得到什麼結果？

- A. 編譯失敗
- B. 在執行間產生例外
- C. Thread one. Thread two.
- D. 無法決定輸出結果

解：答案是 B。在 Thread 物件上啟動 start() 方法時，只能允許執行一次，當第二次啟動時，會產生例外，所以說就算它已完成它所交代的工作，可是當你在呼叫 start() 方法的時候，它還是不合法的。

5.

- 1. public class MyRunnable implements Runnable {
- 2. public void run( ){
- 3. //在這裡插入程式碼
- 4. }
- 5. }

請問何者可以產生這個執行緒並啟動它？

- A. new Runnable(MyRunnable).start( );
- B. new Thread(MyRunnable).run( );
- C. new Thread(new MyRunnable( )).start( );
- D. new Runnable( ).start( );

解：C 因為該類別實作 Runnable，實體化後傳給 Thread 建構子，然後 Thread 需要被啟動。A 與 B 錯的原因 Runnable 是一個介面，所以不能將類別或介面名稱傳給任何建構子。D 是因為 MyRunnable 沒有 start() 可以使用，這裡唯一可以使用 start() 執行緒只有 Thread。



6.

```
1. public class MyThread extends Thread {
2.
3.     public static void main(String [] args) {
4.         MyThread t = new MyThread( );
5.         Thread x = new Thread(t);
6.         x. start( );
7.     }
8.
9.     public void run( ){
10.        for(int i = 0; i<3; ++i )
11.            System.out.print(i + " . . " )
12.        }
13.    }
```

請問執行後得到什麼結果？

- A. 編譯失敗
- B. 1 . . 2 . . 3 . .
- C. 0 . . 1 . . 2 . . 3 . .
- D. 0 . . 1 . . 2 . .
- E. 執行期間發生例外

解:答案為 D，MyThread 會執行，並執行三次。

7.

```
1. class Test {
2.
3.     public static void main(String [] args) {
4.         printAll(args);
5.     }
6.
7.     public static void printAll (String[] args) {
8.         for(int i = 0; i<lines.length; ++ ) {
9.             System.out.print(lines[i]);
```

```

10.      Thread.currentThread( ).sleep(1000)
11.      }
12.  }
13.  }

```

靜態方法 `Thread.currentThread( )` 會回傳一個參考，它的對象是目前執行中的 `Thread` 物件。請問執行後得到什麼結果？

- A. 在陣列 `lines` 中的每個字串都有輸出，其間有一秒的暫停時間。
- B. 在陣列 `lines` 中的每個字串都有輸出，其間沒有暫停時間，因為這方法不是在 `Thread` 中執行的。
- C. 在陣列 `lines` 中的每個字串都有輸出，但不能保證其間是否有暫停時間，因為 `currentThread( )` 可能不會擷取這個執行緒。
- D. 編譯失敗

解：答案為 D，`sleep( )` 方法並需放在 `try/catch` 的區塊裡，否則 `printAll( )` 方法必須宣告它會產生 `InterruptedException`。A 錯在沒針對 `InterruptedException` 做處理。B 錯在因為所有 Java 的程式，包括 `mail( )` 在內，都在執行緒中處理。C 錯在 `sleep( )` 方法是靜態，所以即使它在物件實體上被呼叫，還是會影響目前執行中的執行緒。

8. 假設你有一個類別，其中包含兩個私有的變數：`a` 與 `b`。底下那對敘述可以防止這類別中發生同時讀取的問題？(請選出所有正確)

- A. 

```
public int read(int a, int b) { return a+b;}
public void set(int a, int b) {this.a = a ; this.b = b;}
```
- B. 

```
public synchronized int read(int a, int b) { return a+b;}
public synchronized void set(int a, int b) {this.a = a ; this.b = b;}
```
- C. 

```
public int read(int a, int b) { synchronized(a){ return a+b;}}
public void set(int a, int b) { synchronized(a){this.a = a ; this.b = b;}}
```
- D. 

```
public int read(int a, int b) { synchronized(a){ return a+b;}}
public void set(int a, int b) { synchronized(b){this.a = a ; this.b = b;}}
```
- E. 

```
public synchronized(this) int read(int a, int b) { return a+b;}
public synchronized(this) void set(int a, int b) {this.a = a ;
```

```
    this.b = b;}
F. public int read(int a, int b) { synchronized(this){ return a+b;}}
    public void set(int a, int b) { synchronized(this){this.a = a ;
    this.b = b;}}
```

解:B 與 F。透過將方法標示為 `synchronized`，這些執行緒會在處理之前，就取得這個物件的鎖。在任何特定的時間上，只有一個執行緒可以設定或讀取，因此可以確信 `read()` 永遠會回傳一對有效數字的和。

9. 哪個類別或介面可以定義 `wait()`、`notify()` 與 `notifyAll()` 方法？

- A. Object
- B. Thread
- C. Runnable
- D. Class

解: 答案為 A，僅需知道 `Object` 類別可以定義這些與執行緒有關的方法。

10. 底下敘述中，哪兩個是正確的？

- A. 靜態方法無法被同步化。
- B. 如果某個類別中包含同步化的程式，多個執行緒仍然可以讀取非同步化的程式。
- C. 以關鍵字 `synchronized` 標示後，變數就可以防止同時讀取的問題。
- D. 當某個執行緒進入休眠狀態的時候，它會釋出它的鎖。
- E. 當某個執行緒啟動 `wait()` 的時候，它會釋出它的鎖。

解：答案為 B、E，A 錯是因為靜態方法可以同步化，C 錯是因為只有方法可以標示 `synchronized` 而變數不行，D 錯是因為休眠時執行緒還是會持有它的鎖。

## 第五章 經驗與心得分享

學生: BN95035 張登凱

這個時代找工作除了學歷之外，對自己本身有所幫助的就是擁有一張證照在身邊證照可以表示你有這方面相關技能的能力證明。也可比別人多點機會錄取工作。

當初在考取這張證照的時候，我花了很多時間在研讀 SCJP 5.0 猛虎出關這本書上，另外也花上許多時間在研讀那些考古題。對於那些考古題如果我有不懂得地方我就會去找尋相關書本上面的題型解析。因為考試內容實在廣泛，況且有許多類似的題型，因此充分的了解題目才是最快的捷徑，也是最大的幫助。

在考取 SCJP 證照的這段過程中，我深刻的體會到人只要有心肯努力，沒有做不到的事情，考到證照就是對於自己能力最佳的證明。另外考試的時候盡量保持放輕鬆的心情，在不慌張不緊張的情況下才不會影響腦袋的思考，對於解題時的幫助也是相當大的。

## 第六章 結論

在研讀 SCJP 的這段路上，我相信大家都是從零開始的。對此學校在於每學期 JAVA 的課程教學都是準備的很充足，而各個老師在教導學生時講解課程內容也都相當認真，經過多個學期的經驗累積，各個老師教導課程的經驗，我覺得在考取證照的難度上相對就降低許多。因此我在準備證照考試的時候所要花費的時間相對的就減少許多，我想每個學期老師所教導的充分內容就是幫助我能夠順利通過這次 SCJP 考試的主要原因了。

在這段考試過程當中，我最感謝的就是每個學期教導我們 JAVA 程式相關課程的老師。當我們在學習上遇到了瓶頸或者是對於在寫程式中許多的小細節與重要觀念時，各個老師都不厭其煩的花費了許多的時間與精力來為我們講解課程，也用心的為我們學生講解在於寫程式當中時常所犯的錯誤並且教導我們正確的觀念。

高國峰 老師 是我很尊敬的指導老師，每當我有許多問題時，老師總是很用心的在幫助我解決問題。如果沒有高國峰 老師 的幫助，我想我也是無法考取到這張證照的。真的非常感謝老師的幫忙。

## 附錄

### a. 專有名詞中英文對照表

#### CH1

- |                        |        |
|------------------------|--------|
| 1. reserved word       | 保留字    |
| 2. primitive data type | 基本資料型別 |
| 3. range               | 範圍     |
| 4. default value       | 預設值    |
| 5. array               | 陣列     |
| 6. initialized value   | 初始值    |
| 7. operator            | 運算子    |
| 8. operand             | 運算元    |
| 9. elements            | 元素     |
| 10. arguments          | 參數     |

#### CH2

- |                |       |
|----------------|-------|
| 1. public      | 公開    |
| 2. private     | 私有    |
| 3. protected   | 保護    |
| 4. default     | 預設    |
| 5. final       | 完整、最終 |
| 6. abstract    | 抽象    |
| 7. super class | 超類別   |
| 8. sub class   | 子類別   |

#### CH3

- |                                  |              |
|----------------------------------|--------------|
| 1. operator                      | 運算子          |
| 2. array                         | 陣列           |
| 3. constructor                   | 建構子          |
| 4. logical operator              | 邏輯運算子        |
| 5. precedence                    | 優先權          |
| 6. associative                   | 連結性          |
| 7. operand types                 | 運算元型別        |
| 8. String Concatenation Operator | 字串連接運算子      |
| 9. primitive data types          | 基本資料型別       |
| 10. cast operator(type)          | 強制型別轉換(type) |
| 11. pass-by-reference            | 傳址           |

12. pass-by-value 傳值

#### CH4

1. exception 例外  
2. assertion 斷言  
3. decision 判斷、決策  
4. label 標記  
5. iteration 迭代  
6. guarded region 監控區域  
7. stack trace 堆疊追蹤  
8. exception propagation 例外傳遞  
9. unchecked exception 未檢查的例外  
10. checked exception 已檢查的例外  
11. appropriate 適當  
12. legal 有效、合法  
13. correct 正確  
14. recursion 遞迴

#### CH5

1. getter method 取回方法  
2. setter method 設定方法  
3. access 存取  
4. instance variable 物件實體變數  
5. variable 變數  
6. extends 繼承  
7. overloaded 過載  
8. override 覆寫  
9. composition 合成  
10. utility methods 公用方法  
11. helper methods 援助方法  
12. abstract 抽象  
13. control flow 流程控制  
14. encapsulation 封裝  
15. constructor 建構子

#### CH6

1. String 字串  
2. empty string 空白字串  
3. delimiter 分界符號

- |                    |             |
|--------------------|-------------|
| 4. character       | 字元          |
| 5. escape sequence | 跳脫串列        |
| 6. concatenation   | 串接          |
| 7. literal         | 字面常數(數字、文字) |
| 8. wrapper         | 外覆          |

## CH7

- |                       |       |
|-----------------------|-------|
| 1. sorted             | 已排序的  |
| 2. ordered            | 有順序的  |
| 3. reachable          | 可到達的  |
| 4. garbage collection | 記憶體回收 |
| 5. Object             | 物件    |
| 6. List               | 串列    |
| 7. Set                | 集合    |
| 8. Map                | 對應表   |

## CH8

- |                          |        |
|--------------------------|--------|
| 1. override              | 覆寫     |
| 2. inner class           | 內部類別   |
| 3. outer class           | 外部類別   |
| 4. anonymous inner class | 匿名內部類別 |
| 5. static inner class    | 靜態內部類別 |

## CH9

- |                 |      |
|-----------------|------|
| 1. thread       | 執行緒  |
| 2. synchronized | 同步化  |
| 3. block        | 區塊   |
| 4. code         | 程式碼  |
| 5. constructor  | 建構子  |
| 6. runnable     | 可執行  |
| 7. running      | 執行中  |
| 8. wait         | 等待   |
| 9. blocked      | 閉鎖   |
| 10. sleep       | 休眠   |
| 11. dead        | 死亡   |
| 12. feature     | 特點   |
| 13. instance    | 物件實體 |