修 平 技 術 學 院

資 訊 網 路 技 術 系

SCJP 專業認證專題

指導老師：高 國 峰　老師

班　　級：四技資網系　四年甲班

組　　長：BN94035 陳柏佑

組　　員：BN94016 黃一峰

組　　員：BN94005 涂志承

中華民國　99 年 5 月

摘要

現在的社會，好像擁有一張國際認證或國內證照是不可或缺的，也是基本條件之一，對於畢業後出去找工作有相當大的幫助。不僅如此，在未來公司內的升遷也可以說佔有重要的關鍵之一。因為，你能多別人一張認證或證照就是比別人多一項能力或一分的努力。

而 SCJP 國際認證在現今的資訊產業上普及化，但對於學生的我們這是一項挑戰，不管將來的升學或就業，都是具有一定加分的效果，為了提升在往後社會上的競爭有利於比別人多一份的能力，所以考取認證是提升我們在這社會的競爭力的方法之一。因此，此次專題我們以考取 SUN（昇陽）所推出的 SCJP 國際認證為目標，將我們在此過程中的經驗、重點整理、注意事項分享給未來想考取此認證的同學們一個不錯的參考。

第一章　簡介

1-1 證照的重要性

大學窄門變成大門，學歷還值錢嗎？根據調查，四分之三民眾認為證照比學歷值得投資成本，若非取得博士高學歷，否則在學歷與證照之間，多數民眾會選擇較為實用的證照。

根據 Yahoo!奇摩新聞民調中心與 104 市調中心同步進行的網路問卷調查發現，在學歷要求上，上班族比一般大眾要求更高。超過八成五的上班族認為一般人應該要有大專以上的學歷，比一般大眾認知多了

一成。其中，約五成受訪者認為所謂的「高學歷」，必須取得博士學位。

高等教育入學門檻逐年下降，在一般大眾的心目中，對學歷的重視程度隨之下降，取而代之的是證照。若要花同樣的時間進修，約七成五的民眾認為取得證照，比學歷來得重要。即使是公認高學歷的博士，雖有三成六受訪者認為學歷比較重要，但超過六成仍傾向證照重要程度超過學歷。這個現象在 19~22 歲，正值就讀大學階段的年齡層，尤其明顯。

學歷似乎不再是上班族心目中的萬靈丹，證照對於上班族的影響力也日益增加，104 市調中心營運長蔡家昌分析，就算已經取得碩士以上學歷的上班族，仍有超過一半的比例認為證照的取得較學歷來得重要；他建議上班族和在學學生，當學歷已成為基本需求時，更應該思考取得相關證照以凸顯和證明個人的能力。(來至奇摩新聞)

1-2 SCJP 介紹

「SCJP」是 Java 技術的第一科認證考試，它的全名是「Sun Certified Programmer for the Java 2 Platform」

中文是「Sun 認證 Java 2 平台程式設計師」,通過這個認證表示具備 Java 程式語言基本的認識，與使用 Java 撰寫程式的能力

SCJP 是由昇陽提供的國際性專業認證,所以雖然是在台灣通過這個考試，全世界都承認你擁有的 SCJP 證照。

1-3 SCJP 認證概述

SUN 認證 Java 程式員 (Sun Certified Programmer for Java, SCJP) 是 Java 的基礎認證，以 Java 程式語言與基礎類別庫使用為考試的方向,並且作為 SCJD 與平台認證集的前置需求,此認證是目前 Sun 認證體系中人數最多的，SCJP 自 Java JDK 1.2 版開始，目前最新的版本為 Java SE 6.0。

SCJP 只要考過一科考試即可，同時也沒有任何前置需求。考試代號是 310-055/310-065。自 2000 年 SCJP 開辦以來，SCJP 已歷經四個版本：SCJP 6.0，以 Java SE 6.0 版為基礎，考試代碼為 Exam 310-065：Sun Certified Programmer for the Java 2 Platform. SE6.0。SCJP 5.0，以 J2SE 5.0 版為基礎，考試代碼為 Exam 310-055。SCJP 1.4，以 JDK 1.4 版為基礎，考試代碼為 Exam 310-035。SCJP 1.2，以 JDK 1.2 版為基礎，考試代碼為 Exam 310-025。

SCJP 的考試重點是着重在 Java 的程式語言本身，諸如運算子、陳述式、邏輯比較、屬性與方法、事件宣告與處理、變數與常數等等，以及 Java 本身所提供的內建基礎類別庫，例如 java.io、java.lang.Thread、java.lang.Runnable、java.lang.Comparable 與 java.lang.String 等類別與命名空間等的操作與使用，大多數的考試重點不會因為版本更動而改變，但在版本更替或是 Java 語言與類別庫的演進時，考試重點可能會有所變更（例如早期會考 Abstract Window Toolkit，現在已廢考）。以 SCJP 6.0 為例，目前考試主題分為七大項：程式語言：宣告、初始化及定義範疇。 程式語言：流程控制。 基礎類別庫：API 內容。 基礎類別庫：同時性。基礎類別庫：物件導向概念。基礎類別庫：集合與泛型。綜合：基本原理。

（來至 http://blog.roodo.com/killtest/archives/8902659.html）

1-4 SCJP 研讀參考書籍

在這此所準備的參考書籍有：

一、 SCJP・SCJD 專業認證指南

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

二、 JAVA 2 SCJP 專業認證大全

作　者：陳培勳

出版社：學貫行銷股份有限公司

三、 Java 認證 SCJP 5.0--猛虎出閘

作　者：段維瀚

出版社：碁峰

四、 SCJP 重點題庫

第二章　認證研讀要點

本章節中所有分析用的考題範例大部分引用自研讀書籍內所附之考題，版權為該書籍所有!!

2-1 程式語言的基本原則

2-1-1-1 分辨 keywords 以及 Java 所擁有的 Reserve Words

一、下表分別為 Java 的所有的 Keywords 和 Reserved Words：

Abstract	do	implements	protected	throws
Boolean	double	import	public	transient
Break	else	instanceof	return	true
Byte	extends	int	short	try
Case	false	interface	static	void
Catch	final	long	strictfp	volatile
Char	finally	native	super	while
Class	flaot	new	switch	assert
Const	for	null	synchronized	
Contine	qoto	packaqe	this	
Default	if	private	throw	

其中 null、true 及 false 並不是 keyword，而是使用中的保留字。雖然 java 不使用 const 及 goto，但仍然是保留字。

例題：

下列哪兩項為關鍵字?

A. interface

B. unsigned

C. float

D. this

E. string

ANS：A 與 D

2-1-2 所有 primitive data type 的 range 以及 default value

2-1-2-1 所有 primitive data type 的 range

所有 primitive data type 的 range：

基本型別	大小	最小值	最大值	外覆型別
boolean	--	--	--	Boolean
char	16 - bit	Unicode 0	Unicode $2^{16}-1$	Character

byte	8 - bit	- 128	+ 127	Byte
short	16 - bit	-2^{15}	$+2^{15}-1$	Short
int	32 - bit	-2^{31}	$+2^{31}-1$	Integer
long	64 - bit	-2^{63}	$+2^{63}-1$	Long
float	32 - bit	IEEE 754	IEEE 754	Float
double	64 - bit	IEEE 754	IEEE 754	Double
void	--	--	--	Void

2-1-2-1-2 primitive data type default value

一、primitive data type of default value(預設值)：

a、 int：

例如：12、45、10、1、8…等等

b、 double：

例如：12.451、45.152、10.745、1.258、8.152…

c、 float：

例如：12.451f、45.152f、10745f、1.258f、8.152f…等等

d、 char：

例 1：’a’ 、’@’

例 2：12、45….等等

例：也可以鍵入字元的Unicode值，作法是在數值前面加上

Unicode標記\u，如下所示:

char letterN = '\u004E'

二、Range（範圍）:

a、 byte：

-128 -127 ….-2 -1（負數）~0 1 2 3…126 127（正數）

b、 short：

-32768 -32767….-1（負數）~0 1 2 3.. 32766 32767（正數）

c、 int：

-2147483648…-1（負數）~0 1 2…2147483467（正數）

d、 char：

0~65535

【補充】無負數

e、 float：

1.40239846 乘以 10 的-45 次方（負數） ~

3.40282347 乘以 10 的 38 次方（正數）

(e-45 就是 10 的-45 次方)

f、double：

4.94065645841246544e-324（負數） ~

1.79769313486231570e308（正數）

例題一：

```
1.class A {
2.    public static void main( String[] args ) {
3.       short s1 = 1 ;          //1
4.       char c1 = 1 ;           //2
5.       byte b1 = s1 ;          //3
6.       byte b2 = c1 ;          //4
7.       final short s2 = 1 ;    //5
8.       final char c2 = 1 ;     //6
9.       byte b3 = s2 ;          //7
10.      byte b4 = c2 ;          //8
11.   }
12.}
```

What is the result of attempting to compile the program？

ANS:Compiler error at 3 & Compiler error at 4

【補充】Contains（內含）及長度：

a、boolean：表示 true or false 的值

b、byte：8 位元的整數（帶符號）

c、char：一個 Unicode 字元（6 位元，無符號）

d、double：一個 64 位元的浮點數（帶符號）

e、float：一個 32 位元的浮點數（帶符號）

f、int：一個 32 位元的整數（帶符號）

g、long：一個 64 位元的整數（帶符號）

h、short：一個 16 位元的整數（帶符號）

例題二：

```
1. class Maroon {
2.     public static void main ( String[] args )
3.         int I = 1;
4.         short s = 1;
5.         long l = 1，m = 2;
6.         i = l + i;
7.         l = s + i;
8.     }
9. }
```

What is the result of attempting to compile the program？

ANS： Compiler error at line 6

例題三:

```
1.class UltraViolet {
2.    public static void main (String[] args) {
3.       char a = '\u002a' ;  // Asterisk
4.       char b = '\u0024' ;  // Dollar Sign
5.       System.out.print(a + b);
6.       System.out.print(" ABC" + a + b);
7.    }
8.}
```

What is the result of attempting to compile and run the above program ?

ANS：Prints: 78 ABC*$

【解說】

一、將a、b轉換成String型式將會印出*S

二、將 a、b 轉換成數字時會印出 0x2a 與 0x24(十六進制)轉換成十進制為 42 與 36

2-1-3 如何宣告一個 array

一、陣列宣告方式是先陳述陣列要包含的元素類型（可能是類別或基本資料型別)，接著在識別子的左邊或右邊使用中括弧：

例如 1：int〔〕 key；//方括弧位於名稱前面

int key 〔〕；//方括弧位於名稱後面（可以

接受，但容易誤解）

例如 2：宣告要包含物件參考的陣列

Thread〔〕threads；//建議用法

Thread threads 〔〕；可以接受，但容易誤解

二、我們也可以宣告多維陣列，也就是陣列的陣列。你可用下列方式來完成宣告：

例如 1：String〔〕〔〕〔〕occupantName；

例如 2：String〔〕ManagerName 〔〕；

第一個範例是三維陣列，第二個範例是二維陣列。注意，第二個範例的變數名稱前方與後方各有一個方括弧。編譯器可以接受這樣的格式。

2-1-4 array 的 initialize value 與 new operator

Array variable 是用來儲存 array 的位置，即一個 array variable 指向一個 array object。它是由固定數量的 elements 所組成，但這些 elements 必須屬於同一種 data type。

建立矩陣有兩種方法:

一、array initializers(陣列 初始值)

二、使用 new operator(運算元)

2-1-4-1 array initializers

char[ ] ca = { ‘a’ , ‘b’ , ‘c’ };

為大小為 3 的字元 array， ‘a’ 、 ‘b’ 、 ‘c’ 是這個字元 array 內 3 個 elements 的 initial values。這 array 的大小就是 initial values 的數量。所以”ca.length”是 3。

例題：

```
1. class Black {
2.     public static void main(String args[]) {
3.         int[] i = {1，2，3，4，5};
4.         long[] l = new long[5];
5.         for (int j=0; j < l.length(); j++) {
6.             l[j] = i[j];
7.         }
8.     }
9. }
```

What is the result of attempting to compile and run

the above program ?

ANS：Compiler error at line 5.

【說明】陣列的長度屬性被取出作為

2-1-4-2 new operator

byte[ ] ba = new byte [3];

上式建立一個大小為3的byte array。ba是一個byte[ ] type的reference variable，它指向這個剛剛建立的array（ba所儲存的是這個array的address)。每個array object都有一個內建的field—length。length記錄著這個array object有多少個elements；”length”就是這個array的大小。例如，ba.length就是3

2-1-5 main method的宣告方法以及它的arguments用法

Java應用程式的進入點，是main() method。其特徵如下:

一、它是一個public method

二、它必須是一個static method。

三、它的method name 必須是main，且全為小寫。

四、它只有一個參數，參數的 data type 是 String[]且第一字元 S 為大寫。

例題：

下列選項中的 main() method，何者可當 Java 應用程式進入點？

A)public static void main(){/*…*/}

B)public static int main(String [] args){/*…*/}

C)public void main(String [] args){/*…*/}

D)public static void main(String[] args){/*…*/}

E)public static void main(String args[]){/*…*/}

F)static public void main(String abc[]){/*…*/}

G)public void static main(String abc[]){/*…*/}

H)public static void main(int i){/*…*/}

ANS：D E F

【解說】A、B、C、H 中的 main() method，只是一般的 method，不是 Java 應用程式的進入點；G 項會產生 compilation error，因為 return type—“void”，後面需緊跟著 method name—main

2-2 宣告與存取控制

2-2-1 類別宣告與修飾子

一、class 內通常可包含一個以上的建構子(constructor)。

class 內若無定義任何建構子(constructor)，Java 會自動提供 default constructor，只是這個建構子不帶

任何參數，也不做任何事情。

二、一個檔案內只允許出現一個 public 等級的 interface 或 class，且該檔案名稱必須等於該 public 等級的 interface 或 class 的名稱(再加上 java 副檔名)。

三、每個原始碼只能有一個 public 類別，檔名必須與 public 類別名稱相同。

四、第一行先後順序 package>>import>>class。

五、註解可以放在任何位置。

例題：

```
1.  package exam.stuff;
2.  import cert.Car;
3.  class Honda extend Car{
4.  }
```

2-2-1-1 類別宣告與修飾子

一、**預設存取** defaulet (只有同一套件內的類別宣告才能看到預設存取的類別)。

二、**公用存取** public 宣告的類別會讓所有的套件中所有的類別能夠讀取，不過如果公用類別和寫的類別在不同的套件中還是需要 import 進來。

三、final，abstract，strictft 這 3 種都可以搭配使用，例如 public 跟 final，strictft 和 abstract，strictft 和 final，但是不能 final 和 abstract(會造成抽象類別不能實作)。

四、strictft 可用在**方法**或是**類別**，表示該類別中的方法程式碼會遵守 IEEE754 浮點運算標準，因爲方法中的浮點運算可能會因爲平台不同而出現不同的作業方式。

五、abstract 類別，只要有一個抽象方法，整個類別就需要宣告 abstract，不過你可以將非抽象方法放在抽象類別中。

a、 抽象方法 abstract go();

b、 非抽象方法 go(){};

c、 抽象類別不能被實體化 XXa=new XX();

例題：

```
1. abstract class Car{
2.   private double price;
3.   private Color carColor;
4.   private String model;
5.   private String year;
6.   public abstract void goFast();
7.   public abstract void goUpHill();
8.   public abstract void impressNeighbors();
9. }
```

程式碼可以成功地編譯，如果要在另一段程式碼中實體化 Car 類別，編譯器邊會發生錯誤：

```
Car x =  new Car ( ) ；抽象類別不能被實體化
```

2-2-1-2 方法與變數的宣告和修飾子

一、類別中的方法是否能夠存取其他類別的套件：

```
1. class Zoo {
2.   public String coolMethod ( ) {
3.      return  “baby” ;
4.   }
5. }
6. class Moo {
7.    public void useAZoo ( ) {
8.       Zoo z = new Zoo ( ) ;
9.    // 若上一行編譯的 Moo 能夠存取
10.       // Zoo 類別
11.       // 但是是否能夠存取 coolMethod
12.      System.out.println( “A  Zoo
      says”  + z. coolMethod( ) ) ;
13.      // 上一行程式碼可以正常編譯 Moo 可以
      存取
14.       // 公用方法
15.     }
16.   }
```

二、子類別是否能夠繼承父類別：

```
1. class Zoo {
2.   public String coolMethod ( ) {
3.       return  “baby” ;
4.   }
5. }
6. class Moo extends Zoo {
7.       public void useMycoolMethod ( ) {
8.       // Moo 是否繼承 coolMethod
9.       System.out.println(“  Moo  says”
   + z. coolMethod( ) ) ;
10.          // Moo 可繼承公用方法，因此上一行
   程式馬可正常執行
11.          //Moo 實例是否在 Zoo 實例中用
   coolMethod( )
12.          Zoo z = new Zoo( ) ;
13.          System.out.println(“  Zoo
   says”  + z. coolMethod( ) ) ;
14.          // coolMethod ( ) 為公用方法，因
   此 Moo 能夠在 Zoo 中用此方法
15.      }
16.   }
```

三、存取修飾詞：

a. **預設**只有當存取套件的類別與預設套件位於相同套件時，才能存取預設套件。

b. **protected** 即使存取套件的子類別與預設套件分別位於不同的套件中，該子類別還是可以存取 protected 套件。

c. 如果子類別所處的套件與父類別不同，它只能透過繼承的方式來存取 protected 套件。一旦不同套件的子類別繼承了 protected 套件，子類別外的程式碼便無法存取該套件，例：A B C，BC 屬同個套件 A 與 BC 屬不同套件，B 繼承 A 可以使用套件，但是 C 不能存取 A 的套件。如圖-1

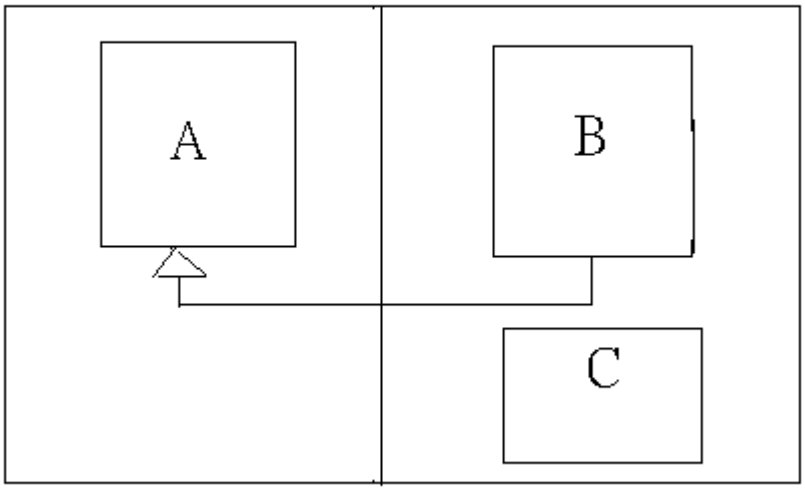


圖-1

四、**區域變數不能使用存取修飾子，除了 final 之外。**

例題：

```
class Foo{
    void doStuff() {
      private int x = 7;
      this.doMore(x);
    }
}
```

使用存取修飾子宣告的區域變數都無法成功編譯

五、**抽象方法**不能爲 static，final，private，protected，synchronized，strictfp，native。有抽象方法類別一定要宣告抽象類別，但是抽象類別不一定要有抽象方法。抽象類別第一個非抽象的子類別必須負責實作父類別的方法。

六、**synchronized** 修飾子只能用在方法上，不能用在類別與變數，可以搭配四種存取控制級中任一個，可以搭配 final 使用，不可以搭配 abstarct 使用，同步化是跟實作有關。

七、**native** 只能用在方法上，修飾子表示使用隨平台而異的程式語言來實作方法。

八、**strictfp** 只能用在方法或是類別，會強制浮點(任何浮點運算)遵守 IEEE754 規則。

九、**實例變數** (instance variables)是在類別內、方法外定義的，而且只有在例項化類別時才會初始化。

例題：

```
1.    class employee {
2.      //  定義實例變數
3.      private string name；
4.      private string title；
5.      private string manager；
6.      //
7.    }
```

十、**transient** 只能用在 instance variables，表示要 JVM 在序列化宣告該變數的物件時跳過此變數。

註：序列化是 JVM 最好的功能之一，它可讓物件的狀態(實例變數的值)寫入到特殊的 IO 資料中，以儲存該物件。透過序列化功能可將物件儲存到檔案中。

十一、**volatile** 變數只能用在 instance variables 告訴 JVM 存取變數的執行緒其私有變數必須與記憶體內的主要變數一致。

十二、**static 變數與方法** 宣告 static 的變數和方法都屬於類別，不需要該類別任何實例也可以使用靜態方法或存取靜態變數。宣告 static 的變數會自動初始化，宣告時可以不用給初始值。

十三、**static 不能宣告在建構子，類別，介面，Inner 類別，inner 類別的方法與區域變數**。

十四、靜態變數會賦予和實例變數相同的預設值。靜態方法無法存取非靜態(實例)變數，因爲它沒有實例。靜態方法無法直接叫用非靜態的方法。

十五、**存取靜態方法與變數：**

a. 非靜態變數或方法要利用實例來作存取

```
A f=NEW A(); f.X;
```

b. 靜態方法可使用 類別名稱.方法或是同上方法也可以

c. 靜態變數可使用 類別名稱.變數 或是同上方法也可以

d. **靜態方法不能被撤銷**。

2-2-2 宣告規則

2-2-2-1 原始檔套件宣告和匯入陳述式

一、 原始檔的套件宣告規則：

1. 每個原始檔案只能有一個公用類別。
2. 檔案名稱必須和公用類別名稱一致。
3. 類別屬於某套件，該套件的陳述式必須位於原始碼檔案的第一行。
4. 匯入與套件陳述式都可套用於原始碼檔案中的所有類別。
5. 若是有匯入陳述式，他們必須放在套件陳述式與類別宣告之間。若是沒有匯入陳述式，則匯入陳述式必須位於原始碼檔案的第一行。若是沒有套件或匯入陳述式，則

類別宣告必須位於原始碼檔案中的第一行。

例題：

```
1.  package exam.stuff；//注意分號
2.  package cert.Beverage；//只能有一個套件宣告
3.  class Foo()
```

匯入語法：

語法有兩種：wildcare 匯入語法與 explicit 類別匯入語法。匯入語法不是 include。匯入語法的用途有點像在鍵入程式碼時能夠少按幾個鍵。將類別放在套件內時，要賦予類別一個較長的名稱，我們稱之爲**完整名稱**。類別的完整名稱就像你的全名和名子的差異。

例題：

```
1. class  Bar{
2.    void  doStuff( ) {
3.       Foo f= new Foo( ) ;
4.   }
5. }
```

此編譯器無法知道你要哪一個類別

解決這問題，必須透過編譯器找到類別

a、匯入語法：

例題：

```
1. import com . wickedlysmart.Foo；
2.   class  Bar{
3.    void  doStuff( ) {
4.      Foo f = new Foo( ) ;
5.   }
6. }
```

此編譯器知道要使用哪一個類別

b、程式碼中使用完整名稱：

例題：

```
1. class  Bar{
2.    void  doStuff( ) {
3.     com . wickedlysmart.Foo f = new  com .
   wickedlysmart.Foo ( )
4.   }
```

5. }

還有兩個相同名稱，但位於不同套件的類別，如果想在同一個原始碼中使用這兩個類別，這時候必須在程式碼中使用完整名稱。就算在核心類別程式庫中，也會有數個名稱相同的類別，例如：在 java . awt 和 java . util 中各有一個 List 類別，如果要使用這兩個類別必須明確的告訴編譯器

2-2-2-2 初始化 main( )方法

一、 main 格式必須是 static。
二、必須具有 void 回傳型態(也可以有回傳值)。
三、必須有一個 String 陣列引數。
四、必須宣告是 public。
五、可以隨心所欲地命名該引數。
六、Public 跟 static 可以擺放不同順序。無法正常執行的 main() 方法屬於執編譯器錯誤
七、class 中如果沒有合法的 main method，在 compile 時不會有錯，但在 run-time 時會產生錯誤 main() method 可以宣告為 final。

Main( )並沒什麼特別之處，它只是類別中另一個靜態方法。和其他方法不一樣的，當呼叫 java 時 JVM 會在此方法裡尋找簽章：

Java MyClass

在指令中鍵入此簽章，JVM 便會尋找名為 MyClass 的類別檔案，如果找到該檔案便會尋找 main( )方法，此方法中就有一個 JVM 要找的簽章

例題：

```
1. class MyClass{
2.     public void main (String [] args) {}
```

2-2-3 抽象類別與介面實作

2-2-3-1 abstract class(抽象類別)宣告

一、一種物件導向語言的機制，用來建立專門的類別當作父類別。

二、抽象類別有點類似”範本”的作用，其目的是要您依據它各格式來修改並建立新的類別。因此，抽象類別裡頭的抽象函數並沒有定義處理的方式。

三、不能直接由抽象類別建立物件，只能透過抽象類別衍生出新的類別，再由它來建立物件。

定義： abstract class 類別名稱{

//資料成員宣告

// 函數成員宣告

修飾子 abstract 傳回值資料型態 方法名稱(引數，…);

}

注意:abstract 方法的修飾子不能宣告為 private

定義：由抽象類別衍生出的子類別？跟繼承一樣，使用 extends關鍵字

```
1. class CCircle extends CShape
2. {
3.   protected double radius;
4.   public CCircle(double r)
5.   {
6.    radius=r;
7.   }
8.   public void show()
9.   {
10.   System.out.print("color="+color+", ");
11.   System.out.println("area="+3.14*radius*r
        adius);
12.  }
13. }
```

2-2-3-2 interface class(介面類別)宣告

介面的使用 介面(interface)與抽象類別非常相似。

但有以下幾個不同的地方：

一、介面的資料成員必須初始化。(即設定出值)

二、介面裡頭的method必須全部宣告為abstract，也就是說，介面不能像抽象類別一樣保有一般的方法，而必須全部是抽象函數。

定義：

```
interface 介面名稱{
    final 資料型態 成員名稱 = 常數;
    修飾子 abstract 傳回值資料型態 方法名稱
    (引數，…);
}
```

註:

1. 關鍵字final有沒有都可以，因為你無法改變它的值。
2. 抽象成員只能宣告為public或是不做宣告。不能宣告為protected或是private。以便讓實作介面的類別都能取用到它。
3. 利用介面打造新的類別的過程叫做實作(implementation)。
4. 介面實作:

```
class 類別名稱 implements 介面名稱{
}
```

一、用介面型態的變數來存取物件：

例題：

```
1.  public class Foo
2.  {
3.    public static void main(String args[])
4.    {
```

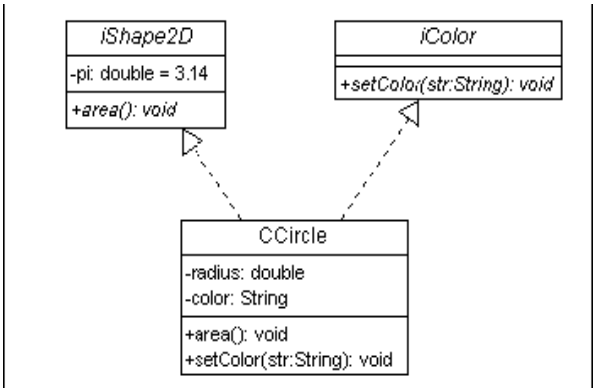
```
5.      iShape2D var1,var2; // 宣告介面型態的變數
6.      var1=new CRectangle(x,y); // 將介面型態的變數var1指向新建的物件
7.      var1.area(); // 透過介var1呼叫show() method
8.      var2=new CCircle(x.y); // 將介面型態的變數var2指向新建的物件
9.      var2.area(); // 透過介面 var2 呼叫 show() method
10.   }
11.  }
```

二、多重繼承

Java 不允許多的父類別的繼承。這點跟 C++不同。但藉由介面的機制，多重繼承的處理還是可以實現。

多重介面繼承:

```
class 類別名稱 implements 介面1，介面2，..{
//
}
```

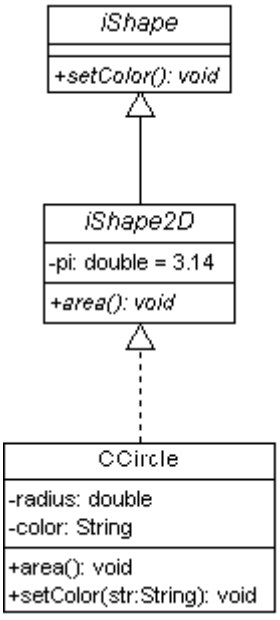


UML類別圖。

介面的延伸(extends)跟類別一樣，介面可以透過繼承的技術來衍生出新的介面。

例題：

```
1.  interface 子介面名稱 extends 父介面1, 父介面2, ..{
2.      //
3.  }
4.  // Foo
5.  interface iShape{
6.      final double pi = 3.14;
7.      abstract void setColor(String str);
8.  }
9.  interface iShape2D extends iShape{
10.     abstract void 2Darea();
11. }
```



UML 類別圖

2-3 運算子與指派

2-3-1 Java 運算子

了解對各種型別，類別，範圍，可存取性或任一組合的運算元使用運算子（包括指派運算子與 instanceof ）

Java 運算子會從一個或數個運算元(運算元式運算子左邊或右邊的東西)產生新的數值。

對於各種 operators，必須知道它的 precedence（優先權）、associative（結合性）、operand types（運算元型別）以及他所執行的 operation（操作）。

P 優先權 A 結合性 Operand types 運算元型別 Description 解說

P	A	Operator	Operand types	Description
15	L	++	numeric	unary post-increment 如 : n++
		--	numeric	unary post-decrement 如 : n--

14	R	++	numeric	unary pre-increment 如 : ++n
		--	numeric	unary pre-decrement 如 : --n
		+	numeric	unary plus 如 :+n
		-	numeric	unary minus 如 : -n
		~	inteqer	unary bitwise complement 如 : ~i
P	A	Operator	Operand types	Description
14	R	!	boolean	unary logic negation 如 : !b
13	R	new	class	object creation 如 : new MyClass()
		(type)	any	casting 如 : (int) |
12	L	*	numeric	multiplication 如 : n1 * n2
		/	numeric	division 如 : n1 / n2
		%	numeric	modulus 如 : n1 % n2
11	L	+	numeric	addition 如 : n1 + n2
			String	String concatenation 如 :"s1" + "n2"
		-	numeric	subtraction 如 : n1 - n2
10	L	<<	inteqer	bitwise left shift,filling

		>>	inteqer	with 0s 如 : i << 2 bitwise right shift,filling with the sugn bit 如 : i >> 2
		>>>	inteqer	bitwise right shift,filling with 0s 如 : i >>> 2
P	A	Operator	Operand types	Description
9	L	<	numeric	less than 如 : n1 < n2
		<=	numeric	less than or egual to 如 : n1 <= n2
		>	numeric	greater than 如 : n1 > n2
		>=	numeric	greater than or egual to 如 : n1 >= n2
		instanceof	numeric	greater than or egual to 如 : n1 >= n2
8	L	==	any	egual 如 : a1 == a2
		!=	any	not egual 如 : a1 != a2
7	L	&	inteqer	bitwise AND 如 : i1 & i2
			boolean	boolean AND 如 : b1 & b2

6	L	^	inteqer boolean	bitwise XOR 如 : i1 ^ i2 boolean XOR 如 : b1 ^ b2
5	L	|	inteqer boolean	bitwise OR 如 : i1 | i2 boolean OR 如 : b1 | b2
4	L	&&	boolean	conditional AND 如 : b1 && b2
3	L	||	boolean	conditional OR 如 : b1 || b2
P	A	Operator	Operand types	Description
2	R	?:	boolean,any,any	tenary condition 如 : b ? a1 : a2
1	R	= += -= *= /=	any numeric String numeric numeric numeric	assignment 如 : a1 = a2 assignment with addition 如 : n1 += n2 assignment with String concatenation 如 : s1 += s2 assignment with subtraction 如 : n1 -= n2 assignment with multiplication 如 : n1 *= n2 assignment with division 如 :

				n1 /= n2
		%=	numeric	assignment with modulus 如 : n1 %= n2
		&=	inteqer	assignment with bitwise AND 如 : i1 &= i2
P	A	Operator	Operand types	Description
1	R		boolean	assignment with boolean AND 如 : b1 &= b2
		^=	inteqer	assignment with bitwise XOR 如 : i1 ^= i2
			boolean	assignment with boolean XOR 如 : b1 ^= b2
		|=	inteqer	assignment with bitwise OR 如 : i1 |= i2
			boolean	assignment with boolean OR 如 : b1 |= b2

		<<=	inteqer	assignment with bitwise left shift,filling with 0s 如 : i1 <<= 3
		>>=	inteqer	assignment with bitwise right shift,filling with the sugn bit 如 : i1 >>= 3
P	A	Operator	Operand types	Description
1	R	>>>=	inteqer	assignment with bitwise right shift,filling with 0s 如 : i1 >>>= 3

integer types 包含 byte、short、int、long 及 char type。

numeric types 包含 integer types、floating point types。

any 包含 primitive types 及 reference。

上表說明 Java operators 的 precedence 及 operand types，只有少部分的 operators 是 right associative，其他 operators 大多是 left associative。

2-3-1-1 指定運算子

指定變數值非常簡單。只要將 = 右邊的東西指定給左邊的變數，但不行像下面一樣：

x= 6 ;

一、指定基本資料型別

當你替變數指派數值時，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。

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

表 複合指定運算子

可以使用文字與數學式的結果來指定基本資料型別變數　如下圖：

```
int x = 7 ;     // 使用文字指定
int y = x + 2 ; // 使用數學式指定(包含文字)
int z = x * y ; // 使用數學式指定
```

文字整數會自動轉換為 int，可以將數值指定給 int 或 long 變數，如果將數值指定給 byte 變數結果會是編譯錯誤，因為 byte 容器比 int 容器小，因此無法容納後者一樣多的位元數。右邊是錯誤示

範 byte b = 27; 編譯器會強制轉型 byte b = (byte) 27; //自行將 int 文字轉換為 byte

指定浮點數 是指整數加上小數，例如：3.1415926、123.567 等，依照長度的不同 （即變數佔用的記憶體位元數），分為 2 種浮點數的資料型態(float、double)如果浮點變數宣告的是 float，在指定浮點文字值時，需要在浮點數值的字尾加上字元”F”或”f”，將數值轉換成浮點數 float，如右所示 float i = 25.0F;

變數數指定過大的文字 如果替變數指派一個過大的數值，編譯器會發生錯誤。

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他基本資料型別變數時，必須複製右邊變數的內容。

int a = 6 ;

int b = a ;

將數字 6 的位元樣式指定給 int 變數 a，a 跟 b 都有 6 的位元樣式，但是如果變更 a 或 b 的內容，

對其他變數無影響。

例題

```
1. class T {
2.   public static void main(String args[]) {
3.     String a = "1";
4.     byte b = 2;
5.     short c = 3;
6.     char d = 4;
7.     int e = 5;
8.     float f = 2;
9.     a += b *= c += d *= e += f;
10.        System.out.print(a);
11.    }
12.  }
```

解釋

指派運算子都是從右到左，從 e += f 開始 e = e + f；e = 5 + 2，e = 7，再看 d *= e； d = d * e；d = 4 * 7，d = 28，再看 c += d；c = c + d；c = 3 + 28，c = 31，再看 b *= c；b = b * c；b = 2 * 31，b = 62，再看 a += b；a = a + b；a = “1” + 62，a = 162 最後 a 印出來是 162

二、指定參考變數

物件指定給物件參考變數

Button b = new button () ;

此程式碼會執行下列作業

a、將參考變數命名為 b，且屬於 Button 型

別

b、在堆積中建立新的 Button 物件

c、將剛建立的 Button 物件指定給參考變數

b

2-3-1-2　比較運算子

比較運算子會比較兩邊運算式的值然後傳回 true 或 false 。

目前有四種比較運算子可比較整數、浮點數和字元的組合時使用

a、＞　大於

b、＞＝ 大於或等於

c、＜ 小於

d、＜＝ 小於或等於

可將字元基本資料型別與任何數值相互比較。比較不同字元或比較字元與數字時，Java 會使用字元的 ASCII 或 Unicode 值來比較。

例：

1. 5 < 6　　// 結果為 true

2. 5 < 6 < 7 // compilation error 原式 →（ 5 <

6) < 7 → true < 5 // true 不是

numeric types

2-3-1-3 instanceof 比較

instanceof 只能用在參考變數上，它會檢查物件是否屬於特定型別。

instanceof 運算子根據同一類別階層內的類別型別來測試物件。

例題

```
1、class A {
2、    public static void main (String[] args)
   {
3、    System.out.print( (new Object( )
   instanceof Object )+",");
4、    System.out.print(( new Object( )
   instanceof String )+",");
5、    System.out.print(( new String( )
   instanceof Object ));
6、   }
7、}
```

解釋

一個新的物件的介面是物件是 true;

一個新的物件的介面是字串是 false;

一個新的字串的介面是物件是 true。

新的字串是一種物件，那新的物件並不是一種字串

2-3-1-4 等號運算子

可以使用等號和不等號運算子來測試相等性

一、== 等號（或稱 等於）

二、!= 不等號（或稱 不等於）

等號運算子會比較兩個「東西」並傳回「布林」值。每次比較都需要用到兩個數字（包括 char）、兩個布林值或兩個物件參考變數。但是不能比較不相容的型別。例如 boolean 是否等於 char。

下列四種型別是可以相互比較的：

數字	字元	布林基元	物件參考變數

比較浮點數與整數,會發現兩個值相同，而==運算子 應傳回 true

例題：

```
1.    9.0 == 9 // true
2.    0.0/ 0 == 0.0 / 0 // false (NaN != NaN)
3.    9 != 8 // true
```

例題：

```
1.  class W {
2.     public static void main (String[] args)
    {
3.        byte x = 3;
4.        byte y = 5;
5.        System.out.print((-x == ~x +
    1)+","+(-y == ~y +1));
6.     }
7.  }
```

解釋

bytes → 8bits

-3=11111101，~3=11111100，

~3+1=11111101，-3=~3+1。

-5=11111011，~5=11111010，

~5+1=11111011，-5=~5+1。

結果印出 true，true。

2-3-1-5　算術運算子

下例基本的算術運算子

+ 加	- 減	* 乘	/ 除	% 餘數

```
int x = 5 * 3;
int y = x - 4;
Syatem.out.println ( “x - 4 is “ + y );
   // 產生 11
```

一、字串連接運算子

```
System.out.println (” String” + 3 + 7 );
```

以 String(即 a)字串開始，先是字串，然後加一個 3(即 b)，形成新的字串”String3”，然後再 加一個 7（即 c)，即成為新字串”String37”。如果

把String字串放到最後面System.out.println(3 + 7 + ”String”)；會得到 10String。

※”＋”operator是left associative，所以兩行指令的結果不同。

二、遞增與遞減

a、++ 加1（前置與後置）

b、-- 減1（前置與後置）

可以將遞增或遞減運算子在變數之前或變數之後，但其實兩者是有差別的，將遞增（遞減）運算子撰寫在變數前時，表示先將變數的值加(減)1，然後再傳回變數的值，將遞增(遞減）運算子在變數之後，表示先傳回變數值，然後再對變數加（減）1。

例如

```
int i = 0;
int number = 0;
number =
++i; // 相當於 i = i + 1; number = i;
System.out.println(number);
```

在這段程式中，number的值會前後分別顯示為1與0，再看看下面這段：

```
number =
i--; // 相當於 number = i; i = i - 1;
System.out.println(number);
```

在這段程式中，number的值會顯示前後分別為

0 與 1

2-3-1-6　位移運算子

位移運算，就是把位元(bit)向左移(<<)或是向右移(>>)幾個位置， (>>>)是無符號右移。

1. 向左移 n 個位元，相當於乘以 2^n 次方倍。
2. 向右移 n 個位元，相當於除以 2^n 次方倍。
3. 將 int 移動 32 的倍數，或 long 移動 64 的倍數，則原本的數值會保持不變。

運算子	使用方式	功能敘述
>>	op1 >> op2	將 op1 向右移 op2 的位元
<<	op1 << op2	將 op1 向左移 op2 的位元

例題

```
1.  class N {
2.    public static void main (String[] s) {
3.      byte b = 5;
4.  System.out.println(b<<32);
5.     }
6.  }
```

解釋

byte 有 8 位元，int 有 32 位元。

因為沒有強制轉型，所以 Java 會自動看為 32 位元

b 向左移 32 位元，32%32=0 不做任何移位！還是印出 5。

2-3-1-7　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：

a、bitwise AND(&)

b、bitwise OR(|)

c、bitwise XOR(^)

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

& (AND)　0011 & 0101 = 0001　←　3 & 5 → 1

| (OR)　　0011 | 0101 = 0111 ←　3 | 5 → 7

^　(XOR)　0011 ^ 0101 = 0110 ←　3 ^ 5 → 6

2-3-1-8　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。

一、bitwise NOT(~)

~5 = ~(0101)→(1010)

例題

```
1.   class D {
2.    public static void main (String args[])
       {
3.         int i1 = ~1;
4.         int i2 = -1;
5.         int i3 = -2;
6.         System.out.print(Integer.
      toHexString(i1) + ",");
7.         System.out.print(Integer.
      toHexString(i2) + ",");
8.         System.out.print(Integer.
      toHexString(i3));
9.       }
10.  }
```

解釋

Integer.toHexString 以 16 進制，將數值表示為字串，~是 Not, Not 是把 0 變 1,1 變 0。

1 的 2 進制是 0001 , ~1 = 1110

1110 的 16 進制是 e,所以是 fffffffe

負號數值要先把正數數值 Not 之後再加 1，即為２的補數表示法

2-3-1-9 條件運算子

條件運算子是三元運算子(它有三個運算元)。其用途是運算布林數學式。

變數 = 判斷條件?指定值 1:指定值 2;

當 "判斷條件" 成立時，變數的內容便是 "指定值 1"，反之，則變數的內容則為 "指定值 2"。

?後的值為(若判斷條件為 true 則回傳)

:後的值為(若判斷條件為 false 則回傳)

例題

```
1.class iii{
2.   public static void main(String[] arg){
3.      int numOfpets = 4;
4.        String status = (numOfpets < 4)?"pet
           limit not  exceeded":"toomany pets";
5.        System.out.println("This prt status
            is " + status);
6.    }
7.}
```

解釋

這是在問 numOfpets 有沒有小於４，那是的話就會印出 pet limit not exceeded，不是的話就會印出 too many pets。

2-3-2 邏輯運算子

說明牽涉到&、|、&&與||等運算子的數學運算，以及以之數值的變數、計算運算元的條件何數學運算的數值。

2-3-2-1 短路邏輯運算子

一、AND(&&)

AND(&&) 跟 boolean AND(&)相似，不同的是，前者當第一個 operand 是 false 時，”&&”不會對第二個 operand 求值。後者(&)兩個都去求值。

二、OR(||)

OR(||) 跟 boolean OR(|) 相似，不同的是，

前者當第一個 operand 是 true 時，”||”不會對第二個 operand 求值。後者(|)兩個都去求值。

2-3-2-2　邏輯運算子(非短路運算子)

& 與 | 這兩種位元運算子也可以用在邏輯數學式中，不過因為它們不是短路運算子，因此必須運算數學式的兩邊！

一、boolean AND(&)

當兩個 operands 都是 true,運算結果才會是 true。

二、boolean OR(|)

當兩個 operands 都是 false，運算結果才會是 false。

X	Y	&(AND)	|(OR)
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

例題

```
1.  class B {
2.   static boolean m(boolean b) {
3.        System.out.print(b + " ");
4.        return b;
5.     }
6.    public static void main(String[] args)
          {
7.         boolean a = false;
```

```
8.          boolean b = false;
9.          boolean c = true;
10.         boolean d = false;
11.         m(m(a | b == c & d) == m((a | b) ==
             (c & d)));
12.     }
13. }
```

解釋

等號運算子的優先權比邏輯運算子要高，(a | b == c & d)的b跟c先比較了。b != c，所以是false。右邊a|b =false，c&d=false所以是true。左邊是false，右邊是true。

2-3-3 將變數傳送到方法中

方法的參數列是資訊傳遞的機制，可以從外面將資訊送入程序的黑盒子，參數列是方法的使用介面。

一個方法如果擁有參數列，在呼叫方法時，傳入不同的參數就可以產生不同的執行結果。

例如

```
static void printTriangle(char ch, int level)
{  int i, j;
   for ( i = 1; i <= level; i++)
   {
      for ( j = 1; j <= i; j++ )
         System.out.print(ch);
      System.out.println();
   }
}
```

解釋

printTriangle()方法傳入不同字元和層數，就可以顯示不同大小和字元的文字三角形，會印出 Null 之後 0 接著是 1*~xx 2*xx 3*xx，類似九九乘法表。

2-3-3-1 正式參數

一、printTriangle()方法定義的參數稱為「正式參數」（Formal Parameters）或「假的參數」(Dummy Parameters)。

二、正式參數是識別字，其角色如同變數，需要指定資料型態，並且可以在函數的程式碼區塊中使用，如果參數不只一個請使用”,”符號分隔。

2-3-3-2 呼叫參數的方法與實際參數

一、printTriangle()方法擁有參數列，所以在呼叫時需要加上參數列，如下所示：printTriangle('@', level);

二、上述方法呼叫的參數稱為「實際參數」(Actual Parameters)，這是參數值，例如：’@’，如果是運算式，例如：level 變數，其運算結果的值需要和正式參數定義的資料型態相同，方法的每一個正式參數都對應一個相同型態的實際參數。

2-4 流程控制、例外&Assertions

2-4-1 所有迴圈的使用方式

2-4-1-1 if-else 分之結構

只當條件式為 true 時，if 敘述式才會執行指定動作。

if-else 敘述式在條件為 ture 時，就會執行某項動作；若條件為否，則會執行另一動作，以下列程式為例：

if 陳述式的有效引數，if 測試的有效引數必須為布林值才行。由於 if 測試需要布林數學式，因此邏輯運算子與 if 測試語意都必須非常完

整。 if 陳述式必須使用至少一組的 “ {…}” 包覆所有的數學式。

布林代數式使用方法	
不正確	正確
Int x = 1; If (x) {}	Int x = 1;If (x == 1) {}
If (0) {}	If (false)
If (x = 6)	If (x == 6)

例題：

```
1.public class Test {
2.    public static woid main (String[ ] args )
  {
3.        int i = 15;
4.        if ( i > 10){
5.           if ((i % 2) == 0 )
6.              system.out.println( “i 大於
                10、而且是偶數” );
7.        }
8.        else {
9.              system.out.println( “i 小於或
                等於 10” )
10.          }
11.     }
12.   }
```

解答：

假如 i 大於 10 且除以 2 為 0 的數，程式會印出” I 大於 10、而且是偶數”

相反的則會印出 “i 小於或等於 10”

2-4-1-2 switch 陳述式

JAVA 提供 Switch 多重選擇敘述式，可依據整數變數或是運算式的可能數值，執行不同的動作，**每個動作都會與變數或運算式所表的整數** 型別為 byte. short. Int 或 char 的數值有關。但不是 long 型別的整數。switch & case 的有效引數 case 的引數必須為文字或 final 變數！

Case 無法包含不是 final 的變數或值域。switch 只支援 int 相容的基本類型! 只接收自動轉換的 int 型別 如果使用其他類型 如:long. Float. Double 等類型，程式就無法成功編譯。

此外，**switch 只會檢查等式**，因此“ 大於” 之類的關係運算子都無在 case 中使用。Switch 中的 default . Break & Fall-Through 程式在執行 switch 陳述式時遇到 break 關鍵字，該程式會立刻跳出該 switch 區段並立即執行其後方的陳敘式。

若無 break 關鍵字，程式便會繼續執行不同的 case 區段，直到遇到 break or switch 陳述式結束。這種現象又稱為 Fall Through 處理區段時先從符合的 case 下手!! 必須在每段符合的程式 case 中插入

break。

若是沒有符合條件值的 case 值，而你想執行某些程式碼，你必須在 switch 陳述式中使用 default 關鍵字。

例題：

```
1.    String s = ” xyz ” ;
2.       switch (s.length())  {
3.         case 1:
4.           System.out.println( “length is
               one” );
5.       break;
6.         case 2:
7.           System.out.println( “length is
               two” );
8.       break;
9.       defauit:
10.          System.out.println( “length is
               three” );
11.   }
```

解答：

因為 s.length();長度為 3，但是程式中並無 3 的選項，所以印出 defauit 的值。

2-4-1-3　while 迴圈

如果不知道要重複多少次執行程式碼區段與陳述

式，但是只要某條件為 true 時便重複執行該區段，可以使用 while 迴圈。while 迴圈結構如下：

例題：

```
1. int x = 2;
2. while ( x == 2 ) {    //執行結果只在條件為真時，while 迴圈才會執行。
3.     System.out.println ( x );
4.     ++x;
5. }
```

解答：

宣告 x = 2，while 迴圈判斷 x = 2，印出 x。

while 迴圈的重點是它**可能永遠無法執行**。若 while 數學式的結果一開始就是 false 則程式就會跳過迴圈，並執行 while 迴圈之後的第一個陳述式。

2-4-1-4 do-while 迴圈

do-while 迴圈一定會**至少執行一次迴圈。記得在 while 數學是尾端使用分號。**

例題

```
1.    do {
2.        System.out.println ( “Inside loop” );
3.    }  while ( false );  // while 後面要使用分號
```

解答：

即使運算結果為 false，System.out.println () 陳述式仍產生一次結果。

2-4-1-5 for 迴圈

如果你已知道要執行幾次迴圈程式碼區段中的陳述式，for 迴圈特別有用除了迴圈主體外， for 迴圈不需要在宣告中使用任何引數。但是 for 迴圈的宣告有三個部份：

一、變數宣告與初始化

二、布林數學式（條件測試）

三、iteration 數學式

若是 for 迴圈內的**變數經過增量或運算，必須在迴圈之前或在 for 迴圈宣告中進行宣告。**你**無法在 for 迴圈外面存取在 for 迴圈宣告中的變數**。可以在 for 迴圈宣告的第一個部分初始化數個變數，**每個變數的初始化都必須以逗號隔開**。不能將數字或是結果不是布林值的項目當做 if 陳述式或是迴圈的條件。

例題

```
1. public class A {
2.     public static void main(String[] args) {
3.         for (int i = 1; i <= 5; i++) {
4.         for (int j = 0; j < i; j++) {
5.           System.out.print(i);
6.               }
7.           System.out.println();
8.    }
9.   }
10.     }
```

解答：

結果會輸出如下面一樣的式子

1

22

333

4444

55555

2-4-2 break&continue 敘述式

2-4-2-1 break 敘述式

當我們在 whlie、for、do…while or switch 中執行 break 敘述式時，程式會立即離開該敘述式。 程式會繼續執行該控制結構之後的第一個敘述式。通常**使用 break 敘述式會造成提早離開迴圈，或者跳脫 switch 剩下的部份**。

2-4-2-2 continue 敘述式

當在 whlie、for、do…while or switch 中執行 continue 敘述式時，就會跳過該迴圈主體中剩下的敘

述式，然後進行下一次的迴圈操作。在 while 和 do…while 敘述式裡，當執行完 continue 敘述式後，**程式會立即測試迴圈繼續條件式。**

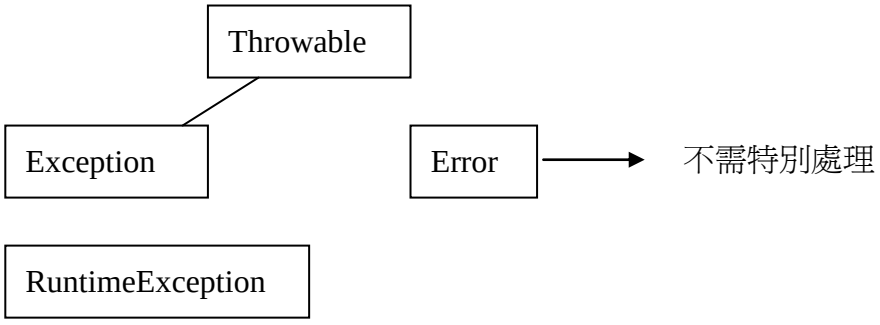
2-4-2-3　在 for 迴圈中使用 break 與 continue

無標籤的 break 陳述式會讓程式停止執行最裡面的迴圈，並處理該區段後方的程式碼。無標籤的 continue 陳述式會中段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。有標籤的 break or continue 陳述式 會在代標籤的迴圈而非最內層的迴圈上執行上述行為。若是在 for 迴圈內使用 continue 陳述式，則會執行遞回陳述式，並再次檢查條件。

2-4-3 labeled&unlabeled statements 的用法

標記陳述式 ，**提供陳述式的識別項。**label 必要項。參考標記陳述式 (Label Statement) 時使用的唯一識別項。statements 選擇項。與 label 有關的一或多個陳述式。標記供 break 和 continue 陳述式所使用，指定要將 break 和 continue 套用至哪個陳述式。

2-4-4 Exception 如何去 handle



在 Exception 的 subclass 中
只有 RuntimeException 不需特別處理
其餘皆需透過捕捉處理之。

一、try{…….. }→ 可能產生例外

二、catch{…….. } → 善後處理

三、finally{……..} → 不論是否產生例外，皆執行

Exception handle

1. 定義：→extends Exception。
2. 宣告：在可能發生處先宣告 throws….. （有 s，用 , 分隔多個)。
3. 丟出：在發生例外情況, 丟出自訂的例外物件. throw new … 。
4. 捕捉：在適當處捕捉自訂例外。

2-4-5 try/catch/finally 如何運作

使用 try 與 catch 捕捉例外

例外有兩種：**可檢查與不可檢查的例外**。可檢查的例外包含 Exception 的所有子類別，細分 RuntimeException 的類別除外。可檢查的例外必須遵守處理或宣告規則。任何“可能”丟出檢查例外的方法都必須使用 throws 關鍵字宣告例外或是使用適當的 try/catch 處理例外。

Error 或 RuntimeException 的子類別屬於不可檢查的例外，編譯器不能強制執行處理或是宣告規則。你可以選擇要不要處理或是宣告，不管你有沒有這樣做，編譯器都沒有影響。使用選擇性的 finally 區段，不管對應的 try 區段中是否丟出例外或是否捕捉到丟出的例外，都會叫住此 final 區段。“永遠會呼叫 finally 區段” 規則唯一的例外是當 JVM 關閉時，程式便不會叫用 finally。

如果 try 或 catch 區段的程式碼呼叫 System.exit ( )，便可能發這種現象，而 jvm 將無法開始執行 finally

區段。

雖然程式會叫用 finally ，但不一定會完成執行。Finally 區段中的程式碼可能會產生例外或丟出 System.exit ( )。未補捉到的例外會在呼叫堆疊中傳遞，從丟出例外的方法開始傳遞到第一個具有符合該例外型別的 catch 區段方法，直到 jvm 關閉，若例外抵達 main( ) 而 main ( )藉由宣告來傳遞例外，此時 jvm 便會關閉。所有的 catch 區段的排序必須從最狹義的區段到最廣義的區段。

一、try → catch → finally 順序

二、try & catch 之間不能夠下其他指令

三、一個 try 指令 可以擁有很多 catch 指令

四、一個 catch 指令可以再創一個 try 指令

catch 指令 但是要照上述順序

五、finally 的例外停止 System.return

六、finally 保證會執行

例題：

```
1. try {
2.    catch (E2 e) {
3.       try {
4.          catch (E3 e)
```

```
5.         }
6.     }
7. }
```

2-4-6 Assertion 的機制

assertions 概述：

assertion 可讓你在寫碼和偵測期間驗證假設。在測試期間通常會啟用 assertion ，並在部屬期間停用 assertion 。

你可以當 assert 當作關鍵字或識別子，但不能兼具兩種身分。

預設情況下，執行期間會停用 assertion。若要啟用 assertion，請使用-ea 或 - enableassertions 指

令行標籤。可以使用 -da 或 -disableassertions 標籤停用 assertion。若使用沒有引數的標籤啟用或停用 assertion，啟用\停用的是一般的 assertion。你可以結合啟用與停用的指令，讓程式在某些類別與套件中啟用 assertion，但在其他時候停用 assertion 。

你可以在具有 -esa 或 -dsa 標籤中的系統類別中啟用或停用 assertion。可使用下列語法，讓 assertion 根據不同類別啟用或停用：

java -ea -da：MyClass TestClass 。

讓assertion根據不同套件以及子套件啟用或是停用。不要使用 assertion 驗證的公用方法的引數。由於公用方法屬於顯露在外的介面，因此必須確定方法本身會強制執行引數上的任何條件。不要使用會造成不良後果的 assert 數學式。Assertion 不一定會執行，因此避免讓執行的作業隨 assetion 是否啟用而改變。

不要使用 assertion 驗證永遠不會執行的程式碼區段。可以對永遠不執行的程式碼使用，如此當程式執行 assert 陳述式時便會立即丟出 assertion 錯誤。不要使用會導致不良後果的 assert 數學式。

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
java –ea ;com.foo	啟用 com.foo 套件以及其所有子套件中的 assertion
java –ea –dsa	啟用一般類別中的 assertion，但停用系統類別中的 assertion
java –ea –da :com.foo	啟用一般類別中的 assertion，但停用 com.foo 套件以及其子套件中的 assertion

例題：

```
1.    public class Test {
2.      public void f( int x )    {
3.        if ( (x%2) == 0 )  {
4.           system.out.println( “even” );
5.        return;
6.          }
7.        else if ( ( x % 2) == 1)  {
8.           system.out.println( “odd” );
9.        return;
10.         }
11.       assert false;
12.       }
13.   public static void main (String [] args)  {
14.      Test t = new Test ( ) ;
15.        t.f (- 1 ) ;
16.      }
17.   }
```

解答：

當 x 除以 2 取餘數 = 0 時，印出 even，餘數 = 1 時，則印出 odd

2-5 封裝的效益、過載與覆寫、建構子以及回傳

2-5-1 封裝的效益

封裝的重點在於，你可以在日後改變心意，將更多的程式加入這些方法，而不會破壞你的應用程式介面，即使你不認為真的需要執行資料的驗證或處理，好的物件導向設計也規定，你必須未雨綢繆為將來做規劃。為了安全起見，你一定要強制呼叫的程式通過你的方法，而不是直接存取實例變數。而**封裝會讓程式更易於維護、更具有擴充能方、也更容易除錯。**

封裝程式有兩項特徵:

一、實例變數會受到保護(通常使用的是 private 這個修飾子)

二、取回與設定方法提供對實例變數的存取功能。

例題:

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

如果某個類別擁有標示為 public 的實例變數(第 8 行),則該類別不能進行封裝。

2-5-1-1 IS-A 與 HAS-A 的關係

一、IS-A 關係

IS-A 的概念是基於繼承的關係,意思就是說:「這個東西是那個東西的一種」。例如 :「馬”是”動物的一種」「馬”IS-A”動物」 在 JAVA 的語言中,需透過 extends 這個關鍵字,來表示 IS-A。如下所示:

```
public class A{…}
類別 A
public class B extends A{…}
「類別 B is-a 類別 A」 「B 是 A 的子類別」
public class C extends B{…}
「類別 C is-a 類別 B」「C 是 B 的子類別」「A 是 C 的超類別」
```

二、HAS-A 關係

HAS-A 的概念基於用法，而不是繼承。換言之，如果類別 A 中的程式參考了類別 B 中的某個實例，則我們說 :「類別 A　HAS-A　類別 B」。

例題:

```
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.  }
```

由於類別 A 有一個實例變數 c 並參考了類別 C 物件，因此 A has-a C，而類別 B 從類別 A 廷伸而來，類別 A 與類別 C 的參考有 has-a 的關係，因此類別 B 透過繼承的關係，所以 B has-a C。

2-5-2 覆寫(override)與過載(overloaded)的方法

方法可以進行過載或覆寫的處理，但建構子只能做過載的處理(建構子絕不會繼承，因此無法覆寫)。過載的方法與建構子可以讓你使用相同的方法名稱(或建構子)，但接受不同的引數清單，覆寫則是在需要與子類別相關的新行為時，讓你重新定義子類別的某個方法。

2-5-2-1 覆寫的方法

只要你的程式中有類別繼承了某個超類別的方法，就有機會覆寫這個方法(除非這個方法已經標示為

final)。覆寫主要的效益在於重新定義子類別中的某個方法。 **覆寫方法不能比要被覆寫的方法具有更多限制的存取修飾子(例如 不能將 public 覆寫改成 protected 或改為 private…等)**

例題1: 下述的程式是 A 的子類別 B 覆寫了 A 的 eat() 方法 。

```
1.    public class override{
2.       public static void main(String [] args){
3.            A a = new B();
4.            a.eat();
5.       }
6.    }
7.    class A{
8.         public void eat(){
9.              System.out.println("123456");
10.       }
11.    }
12.   class B extends A{
13.         public void eat(){
14.             System.out.println("789012");
15.       }
16.    }
```

因為類別 B 的 eat() 方法覆寫了 A 的 eat() 方法，因此執行結果為: 789012。如果將 B 的 eat()方

法改為 private 則程式將無法編譯。

例題 2:下述的程式是 P 的子類別 Q 覆寫了 P 的 printS1()與 printS2()方法 。

```
1.  class P{
2.     void printS1(){System.out.print("P.printS1 ");}
3.     void printS2(){System.out.print("P.printS2 ");}
4.     void printS1S2(){printS1();printS2();}
5.  }
6.  class Q extends P{
7.     void printS1(){System.out.print("Q.printS1 ");}
8.     void printS2(){System.out.print("Q.printS2 ");}
9.     public static void main(String [] args){
10.          new Q().printS1S2();
11.    }
12. }
```

在程式的第 7 行與第 8 行覆寫了父類別 P 的 printS1 與 printS2 方法。

覆寫方法的規則如下:

一、引數必須完全與被覆寫的方法吻合。

二、回傳型態必須完全與被覆寫的方法相同。

三、存取層級不能比被覆寫的方法更有限制性。

(例:將 private 改為 public)

四、存取層級可以比被覆寫的方法更少限制性。

五、覆寫方法不能比被覆寫的方法,提出新的或範圍更廣的可控式例外(除非是子類別)。

六、不能覆寫為 final 的方法 。

七、靜態方法不能覆寫非靜方法，反亦之。

八、如果某程式被繼承，你將無法覆寫它。如下述程式….

例題 3:

```
1.  public class TestAnimals{
2.     publid static void main (String [] args){
3.       Horse h = new Horse();
4.       h.eat();  // 這裡不合法，因為 Horse 未繼承 eat()方法
5.  }
6.  }
7.  class Animal{
8.    private void eat(){
9.      System.out.println( “Generic Animal Eating Generically” );
10.   }
11. }
12. class Horse extends Animal{}
```

2-5-2-2 過載的方法

過載的方法可以讓你在類別中**重複使用相同的方法名稱**，但是**後面接的是不同的引數(也可以是不同的回傳型態)**。

過載方法的規則如下:

一、過載的方法必須改變引數清單。

二、過載的方法可以改變回傳型態。

三、過載的方法可以改變存取修子。

四、過載的方法可以宣告新的、或是範圍更廣的可控式例外。

五、某個方法可以在相同的類別、或是在子類別中過載。

例題 1: 下述的程式是個合法的過載。

```
1.   class A {void m1(String s1){}}
2.   class B extends A{
3.   void m1(String s){}
4.   void m1(boolean b){}
5.   void m1(byte b)throws Exception{}
6.   String m1(short s){return new String();}
7.   private void m1(char c){}
8.   protected void m1(int i){}
```

例題 2: 下述的程式也是個合法的過載。

```
1.  public void changesize(int size, String
    name, float pattern) {}
2.  public void changesize(int size, String
    name) {}
3.  public int changesize(int size, float
    pattern) {}
4.  public void changesize(String name, float
    pattern) throws IOException {}
```

上述兩個程式都顯示了，只要**引數**不同，**回傳型別**不同或是**丟出例外方法**不同，都是一個合法過載的方法。

2-5-2-3 過載與覆寫方法中的多型

假設一個類別具有過載與覆寫的方法，如下題所示

例題: 假設類別 B 同時擁有過載與覆寫的方法

```
1.  public class A{
2.       public void eat(){
3.         System.out.println( “I’ m a A” );
4.       }
5.   }
6.    public class B extends A{
```

```
7.        public void eat(){//此行為覆寫的方法。
8.          System.out.println( “I’ m a  B” );
9.        }
10.     public void eat(String s){//此行為過載
                                    的方法。
11.       System.out.println( “I’ m a” +s);
12.     }
13.   }
```

呼叫方法的程式碼如下:

一、A a = new A();

a.eat(); //執行結果 I’ m a A

二、B b = new B();

b.eat(); //執行結果 I’ m a B

三、A ah = new B();

ah.eat(); //執行結果 I’ m a B

四、B b = new B();

eat(“C”); //執行結果 I’ m a C 此時執行的是類別 B 裡過載 eat(String s)的方法

五、A a1 = new A();

a1.eat(“C”); //編譯錯誤，編譯時會發現類別 A 裡並無可接受字串的 eat()方法

2-5-3 建構子與實例化

建構子指的是每當你使用關鍵字 new 時，所執行的那段程式。

2-5-3-1 建構子的基本概念

包括抽象類別在內的所有類別，都必須具有一個建構子(牢記)。建構子只能做過載的處理。而**建構子的名稱必順與類別名稱相同**。建構子一般內容如下:

```
1.  class Foo {
2.        Foo()  {  }  // Foo 類別的建構子
3.   }
```

建構子沒有回傳型態，而且它的名字必須與類別名稱相同，通常建構子都被用來初始化實例變數，如下程式所示:

```
1.  class Foo {
2.        int size;
3.        String name;
4.    Foo(String name, int size){
5.         this.name = name;
6.         this.size = size;
7.         }
8.  }
```

Foo 類別並沒有不需要接受引數的建構子，看看底下兩個程式:

Foo f = new Foo(); // 無法編譯，因為沒有匹配的建構子。

Foo f = new Foo(“A”, 43); // 編譯成功，因為引數符合Foo建構子的定義。

底下是建構子的規則(重點):

一、可以使用任何存取修飾子,包括 private 。

二、建構子名稱必須與類別名稱相同。

三、不能有回傳型態。

四、如果看到回傳型態(void)，就可以判斷它是方法，而不是建構子。

五、如果不在類別的程式碼中輸入建構子，編譯程式會自動產生預設建構子

六、預設的建構子是不帶引數的。

七、每個建構子的第一個陳述，必須是對過載建構子的呼叫(this())，或是對超類別的呼(super())。

八、super() 呼叫可以不帶引數，也可以包含要傳給超類別建構子的引數。

九、不帶引數的建構子，不一定就是預設的建構子。

十、在超類別建構子執行之前，你都無法呼叫實例方法或存取方法。

十一、可以存取靜態變數與方法。

十二、抽象類別也有建構子。

十三、介面則沒有建構子，介面不屬於物件繼承結構中的一部份。

十四、唯一能啟動建構子的方式，就是從另一個建構子的內部。

※五到八這四點極為重要，勿必牢記。

2-5-3-2　判斷是否會產生預設的建構子

建構子可以接受引數，就像方法一樣。如果你想啟

動一個有引數的方法，但未傳入任何內容，編譯就會失敗。

下述顯示是 Horse 類別，加上兩個建構子 :

```
1.  class Horse {
2.  Horse () {}
3.  Horse (String name) { }
4.  }
```

```
//編譯上述程式並不會加入預設建構子
```

```
1.  class Horse {
2.  Horse (String name) { }
3.  }
```

```
//編譯上述程式並不會加入預設建構子
```

```
1.   class Horse { }
```

```
//編譯上述程式會加入預設建構子 因為它沒有定義任何建構子
```

```
1.  class Horse {
2.  void Horse () { }
3.  }
```

```
//這並不是一個建構子而是一個方法 剛好與類別同名而已
```

如果你的超類別建構子帶有引數的話，你必須在對 super() 的呼叫中，提供適當的引數。如果超類別沒有不帶引數的建構子，則必須在你的子類別中輸入一個建構子，因為你需要有地方放置呼叫 super() 時，所需要傳送的引數。如下程式所示:

例題:

```
1.  class Animal{
2.  Animal(String name)
3.  }
4.  class Horse extends Animal{
5.  Horse(){
6.  super();   //編譯無法成功，因為超類別
     不接受不帶引數的建構子。
7.      }
8.  }
```

順帶一提的是，建構子不會由繼承產生，因此無法覆寫，只有方法才能覆寫。但建構子可以被過載，而預設的建構子永遠不會引數。

一、如何確定是否會建立預設的建構子?

因為你在類別中並沒有撰寫任何建構子的程式。

二、如何知道預設建構子的內容?

預設建構子與該類別具有相同的存取修飾子。

預設建構子沒有引數。

預設建構子中，包括一個對超類別建構子（ super（）） 的無引數呼叫。

三、萬一類別建構子帶有引數，會產生什麼結果?

建構子可以接受引數，就像方法一

樣。

四、如果超類別建構子帶有引數的話，那就必須在對 super() 的呼叫中，提供適當的引數。

五、如果超類別不帶引數建構子，則必需在子類別輸入一個建構子，因為你需要有地方放置呼叫 super() 所需要傳的引數。

六、如果超類別沒有不帶引數的建構子，則子類別無法使用編譯程式所提供的預設建構子。

2-5-3-3 過載建構子

將建構子過載的意思是說，建立該建構子的多個版本，每個版本都有不同的引數清單，如下程式範例:

```
1.  Class Foo{
2.        Foo(){ }
3.        Foo(String s){ }
4.   }
```

2-5-4 合法的回傳型態

一、過載方法可以改變回傳型態，但覆寫方法不行。

二、物件參考的回傳型態可以接受 null 當做回傳值。

三、陣列是合法的回傳型態，這對宣告與當做數值回傳都有效。

四、對以基元為回傳型態的方法來說，任何可以隱式轉換成回傳型態的數值都可以回傳。

五、從 void 不能回傳任何內容，不過你可以不回傳。你

允許在任何帶有 void 回傳型態的方法中，只寫上 return。不過，在不是以 void 為回傳型態的方法中，你磺不可以不回傳任何內容。

六、對以物件參考為回傳型態的方法來說，該型態的子類別可以被回傳。

七、對以介面為回傳型態的方法來說，該介面的任何實作都可以被回傳。

2-5-4-1　過載與覆寫方法上的回傳型態

一、過載的回傳型態

要過載某個方法，必須改變引數如下程式所述:

```
1.   pubic class Foo{
2.         void go(){ }
3.   }
4.   public class Bar extends Foo{
5.       string go(int x){
6.           return null;
7.       }
8.   }
```

不能將第 5 行的 go()方法裡的引數拿掉，因為不能只改變回傳型態。

二、覆寫的回傳型態

過載方法可以修改回傳型態，但覆寫方法則不行，引數與回傳型態都必須相等。

2-5-4-2 回傳數值

回傳數值的部份，需記住六大規則:

一、在以物件參考為回傳型態的方法中，可以回傳 NULL 值。

```
1. public Button doStuff() {
2.   return null;
3. }
```

二、陣列是完全合法的回傳型態。

```
1. public String [] go {
2.   return new String [] 
   ( “Fred”, ” Barney”, ” Wilma” );
3. }
```

三、不能從帶有 void 回傳型態的方法中回傳任何內容。

```
1. public void bar() {
2.   return “this is it” ;
3. }
```

四、在帶有以基元為回傳型態的方法中，你可以回傳任何數值或變數，只要它們都可以明確地強迫轉型為宣告的回傳型態。

```
1. public int foo () {
2. float f = 32.5f;
3. retutn (int) f;
4. }
```

五、在帶有以基元為回傳型態的方法中，你可以回

傳任何數值或變數，只要它們都可以明確地強迫轉型為宣告的回傳型態。

```
1.  public int foo () {
2.     char =   ' c '   ;
3.      retutn c ;     //char 與 int 是相容
4.   }
```

六、在以物件參考為回傳型態的方法中，你可以回傳任何的物件型態，只要它們都可以隱式地強迫轉型為宣告的回傳型態。

```
1.  public Animal getAnimal() {
2.     return new Horse(); //假設 Horse 是由 Animal 廷伸來的
3.     }
```

例題 1:假設看到底下程式,第 4 行不可插入下列哪行程式，將會無法編譯?

```
1.    int x;
2.    x = n.test();
3.    int test(){
4.
5.      return y;
6.    }
```

A. short y =7;(由於 short 比 int 較短的基元，因此可行。)

B. int y =(int) 7.2d;(型別為 double，會轉型為 int。)

C. Byte y= 7;(Byte 是 Wrapper 類別的物件，而

不是基元，因此不可插入。）

D. char y=’s’;（char 比 int 較短的基元，因此可行。）

E. int y=0xface0; 這是一個有效的整數值(十六進位)。

例題 2:假設看到底下程式，哪兩行不可插入第 2 行程式，將會無法編譯?

```
1.  Long test (int x, float y){
2.
3.  }
```

A. return x; （回傳為 int 值，編譯可成功。）

B. return (long)x / y;（因為 long 的強制轉行只適用於“x” ，不是”x/y”）。

C. return (long)y;（強制轉行為 long，編譯可成功。）

D. return (int) 3.14d;（將結果 double 轉行為

int 型別，編譯可成功。）

E. return (y / x); （編譯結果為浮點數，而不是 Long。）

F. return x / 7;(結果為 long 值，編譯可成功。）

2-6 Math 類別、字串與包裝類別

2-6-1 String 及 StringBuffer 類別的用法

2-6-1-1 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)在 JAVA 中每個字串都是物件就跟其他物件一樣 都可以使用 new 關鍵字

例題： String s = new String () ;

String 類別也有無數的建構子 你可以採取更有效率的捷徑

例題：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

例題：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,並在他的後面加上”more stuff”,於是產生的”abcdef more stuff”這個新內容 由於字串是不可變的 虛擬機器無法將這新字串硬塞給變數 S 所參考的舊字串 所以 他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

2-6-1-2 字串與記憶體的重要事實

應用程式不斷成長 字串文字佔用大量的程式記憶體，在這些文字當中，通常也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 特別在記憶體中播出一塊叫做 ”String constant pool ” 的區域，當編譯程式碰到某字串文字的時候，他會先檢查這塊區域，檢查是否已存有相同的字串，對新文字的

參考會被導向現有的字串,不會再產生新的字串文字物件，這也是讓字串有不可變更這個特性的優點。

2-6-1-3 String類別中重要的方法

一、Public char charAt(int index)

這個方法是回傳位於字串特定索引值上的位元

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

二、Public String concat(String s)

這個方法是回傳一個結合後的字串方式就是把方法中所傳遞的字串數值加在啟動開方法的字串後面。

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

三、Public boolean equalsIgnoreCase(String s)

這個方法是回傳布林值(true 或 false)判斷引數中的字串數值是否與啟動方法的字串內容相同 大小寫可以不同。

```
String x = ” Exit ”
System.out.println(x.equalsIgnoreCase(”
                EXIT” )); //回傳值 ”  true”
System.out.println(x.equalsIgnoreCase(”
                tixe” )); //回傳值 ”  flase
```

四、Public int length()

這個方法是回傳啟動該方法的字串長度

```
String x = ” 01234567” ;
System.out.println(x.length());
```

```
    //回傳值  8
```

五、Public String replace (char old , char new)

這個方法是回傳一個經過更新後的字串 更新的對象是啟動該方法的字串更新的做法是根據第二個引數中的 char 替換該字串中任何與第一個引數中的 cahr 相同的字元。

```
String x = ” oxoxoxox” ;
System.out.println(x.replace(” x” , ” X” ));
    //輸出為 oXoXoXoX
```

六、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一部份 第一個引數代表子字串的起始位置，第二個字串代表從他開始結束。

```
String x =  “0123456789” ;
System.out.println(x.substring(5));
     //輸出為 56789
System.out.println(x.substring(5, 8));
     //輸出為 567
```

七、Pubilc 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”
```

八、Public String trim()

這個方法是將字串中首尾的空格都刪除

```
String x =  “hi  “;
System.out.println(x+”  x” );
     //輸出為  “ hi x “
System.out.println(x.trim()+”  x” );
```

```
//輸出為 “ hix “
```

2-6-1-4 StringBuffer 類別

當你想對字串進行大量修改時，應該要使用 StringBuffer 類別。就像在前面討論過的，字串是不可變更的。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 StringBuffer 的物件可以重複修改，而不會留下這些後遺症。

2-6-1-5 StringBuffer 類別中重要的方法

一、Public synchronized StringBuffer

append(String s)

這方法會更新啟動物件內容，不管回傳值是否被派給某個變數。

```
StringBuffer sb = new StringBuffer(
                                “set  “);
sb.appwnd( “point” );
System.out.println(sb);
//輸出為  set point
```

二、Public stnchronized StringBuffer insert(int

offset , String s)

在兩種情況中，作為第二個引數的字串都會插入原始的 StringBuffer 物件而開始的位置則由第一個引數決定。

```
StringBuffer sb = new
          StringBudder( “01234567” );
sb.insert(4,  “- - - “);
System.out.println(sb);
//輸出為 0123- - -4567
```

三、Public synchronized StringBuffer reverse()

StringBuffer 中的字元會被倒置，也就是第一個字元會變成最後一個。

```
StringBuffer sb = new StringBuffer( “
          A man a plan a canal panama” );
sb.reverse();
System.out.println(sb);
//輸出為 amanap lanac a nalp a nam A
```

四、Public String toString()

這方法會以字串的型態，回傳啟動該方法的 StringBuffer 物件內容：

```
StringBuffer sb = new StringBuffer(“
                    test string” );
System.out.println(sb.toString());
   //輸出為 test string
```

2-6-2 Math 類別的用法

2-6-2-1 java.lang.Math 類別中重要的方法

一、Math 類別中的所有方法都被定義成 static 要建立 Math 類別的實例是不可能的 因為其中的建構子都是 private 你也無法擴充 Math 類別 因為它被標記為 final。

二、Math 類別中方法都是靜態的 讀取的方式也與任何靜態方法相同 就是透過類別名稱來讀取。

這些方法呼叫的一般形式：

```
result = Math.aStaticMathMethod();
```

Math 的方法與範例：

一、abs()

回傳引數的絕對值

```
x.Math.abs(99); //輸出為 99
x.Math.abs(-99); //輸出也是 99
```

二、ceil()

將引數化簡為最接近的整數值

```
Math.ceil(9.0)  //輸出為 9.0
Math.ceil(8.8)  //輸出會進位為 9.0
Math.ceil(8.02)  //輸出會進位為 9.0
```

負數的情況 (-9.0 大於 -10.0)

```
Math.ceil(-9.0)  //輸出為 -9.0
Math.ceil(-9.4)  //輸出進位為 -9.0
Math.ceil(-9.8)  //輸出進位為 -9.0
```

三、floor()

它的處理方式與 ceil()相反

```
Math.floor(9.0); //輸出為 9.0
Math.floor(9.4); //輸出會退位為 9.0
Math.floor(9.8); //輸出會退位為 9.0
```

負數的情況 (-9.0 小於 -8.0)

```
Math.floor(-9.0)  //輸出為 -9.0
Math.floor(-8.8)  //輸出進位為 -9.0
Math.floor(-8.1)  //輸出進位為 -9.0
```

四、max()

傳回兩個數值中 比較大的數值

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

五、min()

與 max() 相反 回傳最小的數值

```
x = Math.min(0.5 , 0.0); //輸出為 0.0
```

六、round()

回傳最接近引數的整數值。它的演算法則:先將引數加上 0.5，再去掉小數的部份，得出最接近它的整數值。這個方法具有超載的能力，可處理 float 或 double 型態的引數。

```
Math.round(-10.5);
        //結果是 -10 (引數為負數 加上
        0.5 會進位為更大的數)
```

七、sin()

會回傳某個角度的正弦，引數為 double 資料型態，代表以弧度顯示的角度，可透過 Math.toPadians()把角度換成弧度。

```
Math.sin(Math.toRadians(90.0))
 //輸出為 1.0
```

八、cos()

傳回某個角度的餘弦

```
Math.cos(Math.toRadians(0.0));
    //輸出為 1.0
```

九、tan()

傳回某個角度的正切

```
Math.tan(Math.toRadians(45.0));
 //輸出為 1.0
```

十、sqrt()

回傳某個資料型態為 double 的數值的平方根

```
Math.sqrt(9.0); //輸出為 3.0
```

十一、toDegrees()

這方法可接受一個以弧度表示的角度為引數 並回傳以度數表示的等值角度。

```
Math.toDegrees(Math.PI * 2.0)
    // 輸出為 360.0
```

十二、toRadians()

這方法可接受一個以度數表示的角度為引數 並回傳以弧度表示的等值角度

```
Math.toRadians(360.0)
    // 輸出為 6.283185 = 2*Math.PI
```

靜態的 Math 方法：

double ceil (double a)
double floor (double a)
double random ( )
double abs (double a)
float abs (float a)
int abs (int a)

long abs (long a)
double max (double a , double b)
float max (float a , float b)
int max (int a , int b)
double min (double a , double b)
float min (float a , float b) double sqrt (double a)
int min (int a , int b)
long min (long a , long b)
double toDegrees (double anglelnRadians)
double toRadians (double anglelnRadians)
double tan (double a)
double sin (double a)
double cos (double a)
double sqrt (double a)
int round (float a)
long round (double a)

2-6-3 Wrapper 類別的用法

一、提供”Wrapper“物件中基元數值的機制，讓這些基元可以加入為物件所保留，像是加入集合元件(collection)或是從某個帶有物件回傳數值的方法中執行回傳的作業。

二、提供基元的公用程式函數的分類。這些函數中大部分都與各式各樣的轉換。

有關 : 基元與字串物件間的轉換，以及將基元與字串

物件在不同基數間的相互轉換。

例如: 二進位，八進位，十六進位。

2-6-3-1 Wrapper 類別的概論

在 JAVA 中每個基元都有一個 Wrapper 程式類別。例如 : int 的 Wrapper 程式類別為 Integer floot 則是 Float ,要記得的是基元名稱就是 Wrappe 名稱的小寫。唯一例外的是 char 跟 int ,char 對應 Character，int 對應 Integer。

2-6-3-2 建立包裝物件的方法

Wrapper 的建構子 :

除了 Character 之外，所有 Wrapper 類別都提供兩個建構子 : 其中之一可接受被建構類型的基元。另一個則接受被建構類型的字串表述：

```
Integer.i1 = new Integer(42);
Integer.i2 = new Integer( “42” );
```

valueOf()

也是建立 Wrapper 物件的方法之一，兩種方法的第一個引數都是適當基元類型的字串表述。第二種方法(當提供的時候)則會加上另一個引數，也就是 int radix 標示第一個引數所使用的基數

Integer i2 = Integer.valueOf (“101011”,2);
//將 101011 轉換成 二進制 43
//並將數值 43 指派給 Integer 物件 i2

2-6-3-3 Wrapper 轉換公用程式的用法

xxxValue()

當你需要將被包裝的數值轉換成基元的時候可以使用許多 xxxValue()法中的一種。在這家族中的所有方法都不需要引數。

```
Integer i2 = new Integer(42);
    //建立一個新的 Wrapper 物件
byte b = i2.byteValue();
    //將 i2 的數值轉換成 byte 基元
short s = i2.shortValue();
     //另一個整數的 xxxValue 方法
double d = i2.doubleValue();
    //又另一個整數的 xxxValue 方法
```

或

```
Float f2 = new Float (3.14f);
    //建立一個新的 Wrapper 物件
Short s = f2.shortValue ();
     //將 f2 的數值轉換成 short 基元
System.out.println(s);
    //結果為 “3”(小數點後面的部份被截
       斷,而不是做進位的處理)
```

常用的 Wrapper 轉換方法

方法
s=靜態
n=NFE

例外	Boolean	Byte	Character	Double	Float	Integer	Long	Short
byteValue		x		x	x	x	x	x
doubleValue	x			x	x	x	x	x
floatValue	x			x	x	x	x	x
intValue	x			x	x	x	x	x
longValue	x			x	x	x	x	x

例外	Boolean	Byte	Character	Double	Float	Integer	Long	Short
shortValue	x			x	x	x	x	x
parseXxx s,n	x			x	x	x	x	x
parseXxx s,n (帶有基數)	x					x	x	x
valueOf s,n	x x			x	x	x	x	x
valueOf s,n (帶有基數)	x					x	x	x
toString	x x		x	x	x	x	x	x
toString s (基元)	x			x	x	x	x	x
toString s (基元,基數)						x	x	
toBinaryStrings						x	x	
toHexString s						x	x	
toOctalString						x	x	

s				

Wrapper 轉換方法的使用方式為
a. primitive xxxValue ( )
b. primitive parseXxx (String)
c. Wrapper valueOf (String)

2-6-4 equals()的用法

判斷將布林equals(Object)方法套用在任何類別組合而成的物件上的結果，這些類別指的是: java.lang.String java.lang.Booean 與 java.langObject。

一、如果要比較基元變數，使用==

二、如果要判斷兩個參考變數是否參考同一個物件，使用==

三、==比較的是位元樣式，可能是基元的位元，或是參考的位元

四、如果要判斷兩個物件是否在意義上相等，使用 equals()

五、字串與 Wrapper 類別會撤銷 equals()方法，來檢查其中的數值

六、StringBuffer 類別的 equals()沒有被撤銷，它會借用

七、如果類別不在相同的階層內，編譯程式將不允許使用

八、如果 Wrapper 位於不同的類別中，它們將不會傳送 equals()方法

2-6-4-1 ==與 equals()方法的概論

變數間的比較

從基元與參考變數開始討論，使用==比較基元變數，但 equals()方法很明顯的就無法用在基元上。==運算子會回傳一個布林函數值：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的所有參考變數都具有同樣的大小(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可相同的物件，在比較變數的時候(不管是基元或參考變數)，實際上是在比較兩組位元模式。

比較變數時的重點：

一、參考變數與基元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

二、基元變數必須使用==，它們不能使用 equals()

方法

三、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這個布林值，否則為 false。

equals()方法的重點揭示

Object 這個類別是所有類別開始往下延伸的起點，它具有一個 equals()方法的細節，就是說每個其他的 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 ：

一、equals()方法只能用來比較物件

二、equals()方法會回傳或的布林代數

三、StringBuffer 類別無法撤銷 equals()方法

四、字串與 Wrapper 的類別都標記為 final,而且已經撤銷了 equals()方法

2-7 物件與元件集合

2-7-1 overriding hashCode()與 equals() Method

本章節將深入討論關於 eqluals() method 與 hashcode()的關係，在前幾個章節曾提到關於 equals()method 與 ”==” 的差別，接下來讓我們來了解如何 overriding hashcode()吧。

2-7-1-1 overriding equals() Method

一、首先從 Java 文件說法，equals( ) method 的限制具有底下幾個特性：

a. **反身性（ reflexive ）**

b. **對稱性（ symmetric ）**

c. **遞移性（ transitive ）**

d. **一致性（ consistent ）**

二、特別需要注意的地方

equals( )、hashCode( )與 toString( ) 等方法都是**公開(public)**的。

例題 1：

```
1.   class Red{
2.       boolean equals(object o){ //code }
3.   }
```

上面 equal( ) method 看起來似乎是對的，但實際上並非是有效的 overriding。

三、當你想要比較兩個物件(object)的內容(content)是否相同，那就必須用到 overriding “equals( )” method，同時你也必須 overriding 定義於 object class 的”hashCode( )” method，否則 java collection API 操作這個 class 的 object 時，就會有錯誤，等一下再 7.1.2 overriding hashCode( ) method 的時候就會探討這個問題了。

四、不符合被Overriding equals( ) method的意義

這裡藏著一個潛在的限制條件：**如果不符合overriding equals( ) method的話，將無法再hasecode( )中將該物件當作索引值使用**。Object中的equals( ) method只能使用==運算子進行比較的工作，所以，除非你overriding equals( ) method，否則只能有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。

五、equals( ) method的實作

例題：分析equals( ) method中所發生的事情

```
1. public boolean equals (Object o){
2.    if( (o instanceof Moof) &&( ( (Moof)
         o).MoofValue()==this.moofValue)){
3.       return ture;
4.    } else {
5.       return false;
6.    }
7. }
```

首先，你必須觀察所有與overriding相關的規則。在第1行的程式中，我們宣告了一個有效的overriding equals( ) method，它是由Object繼承來的。

第二行程式可說是全部作業的核心。邏輯上，

我們需要執行兩件事情，才能進行有效的等價關係比較。

a、確定要測試的物件屬於正確的型態

從多型的角度來看，這指的就是 Object 型態。所以，你需要對它執行 instanceof 的測試。通常，將兩個屬於不同型態的物件視為相等，並不是一個好的想法。不過，這已經屬於設計的範疇，我們不打算在這裡討論這個部分的內容。除此之外，你還是需要執行 instanceof 的測試，確定你可以將物件的引數強迫轉型為正確的型態，讓你存取它的方法或變數，以實際進行比較的作業。

b、比較我們所關心的屬性

2-7-1-2 overriding hashCode() Method

認識 hashCode( ) method 的意義，為了理解什麼是適當與正確的意義，我們需要觀察某些元件集合的使用 hashCode( ) method。

當實際計算 hashCode( ) method 的時候，在一個陣列裡面，也許會有一筆以上相同數值元素存在，通

常，好的 hashCode( ) method 擷取作業可分為兩個步驟：

a. 找出正確的陣列。

b. 從正確的陣列裡面搜尋正確的元素。

一、hash code( ) method

一個 object 的 hash code value 是由它的 hashCode( ) method 來決定。在 Object class 已經定義了 hashCode( ) method，語法： int hashCode( )

例題 3：

```
1.    class HashHash{
2.       public int x;
3.       HasHash (int xVal){
4.          x = xVal;
5.    }
6.    public Boolean equals(Object o){
7.       HasHash h = (HasHash) o;  //在未進行
                        instanceof 測試的情況下
8.       If (h.x == this.x){   //千萬不要嘗試這
                                段程式
9.          return ture;
10.      }else{
11.         return false;
12.      }
13.   }
14.      public int hashCode(){
15.         return (x * 17);
16.      }
17.   }
```

因為 equals() method 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 hashCode( )。

通常，你所看到的 hashCode( ) method 都會執行一些以「或(XOR)」連結的運算式，或許也會讓它成上某個質數。不管是哪種情況，雖然他們的目標都是要將物件更廣泛且隨機地分配到不同的收藏盒之中。

二、hashCode( ) method 限制

在 Java 的 API 文件中，有關 Object 類別的內容部分，呈現 hashCode( ) method 的合約：

a、在執行某支 Java 應用程式的過程中，每當對同一個物件啟動一次以上的 hashCode( ) method 時，它必須前後一致地回傳相同的整數值，假設該物件上進行 equals( )比較時所用到得資訊未被修改的話。同一支程式在不同的執行過程中，回傳的整數值則不一定要相等。

```
x.equals(y) == ture
x.hashCode() == y.hashCode()
```

b、如果根據 equals(Object) method 判斷兩個物件相等的話，則其中任何一個物件呼叫 hashCode( )方法的時候，必須產生相同的整數結果。

```
x.hashCode() == y.hashCode()
```

c、如果根據 equals(java.lang.Object) method 判定兩個物件不相等的話，則其中任何一個物件呼叫 hashCode( )方法的時候，不一定要產生不同的整數結果。不過，程式設計人員應該要知道的是，讓不相等的物件產生不同的整數結果，將可以改善 hashCode( )的執行效能。

```
x.equals(y)==false
    //則不必要 hashCode( )
x.hashCode()!=y.hashCode( )
    //則必要 x.equals(y) == false
```

2-7-2 元件集合 Collection

2-7-2-1 什麼是 Collection？

首先談到 Collection 在 JDK1.2 版推出時成形，並在 1.4 版中有所擴充，它的功能相當強大，這樣說也許有點模糊不清，簡單來說，當有一天你的長官，有一份人員清單要建檔的時候，要求你幾日內要完成的時候，

通常人員清單資料，不外乎姓名、身分證號碼、電話、住址、職位…等等，通常程式設計人員不可能還在那邊慢慢寫套件，因此 Java 在 java.util 套件中建立一個公用程式，讓程式設計人員，直接套用，再加以修改，這不是方便省時許多。

一、元件 Collection 可以做些什麼

底下列出一些基本的作業，這些都是比較經常會使　用到的：

a. 將物件加入元件集合

b. 從元件集合將物件刪除

c. 確定某個物件(或一群物件)是否存在元件的集合之中。

d. 從元件集合擷取某個物件(但不將它移除)

e. 對元件集合的內容執行迭代(iteration)的作業，逐一檢視每個元素(物件)。

二、元件 Collection 架構

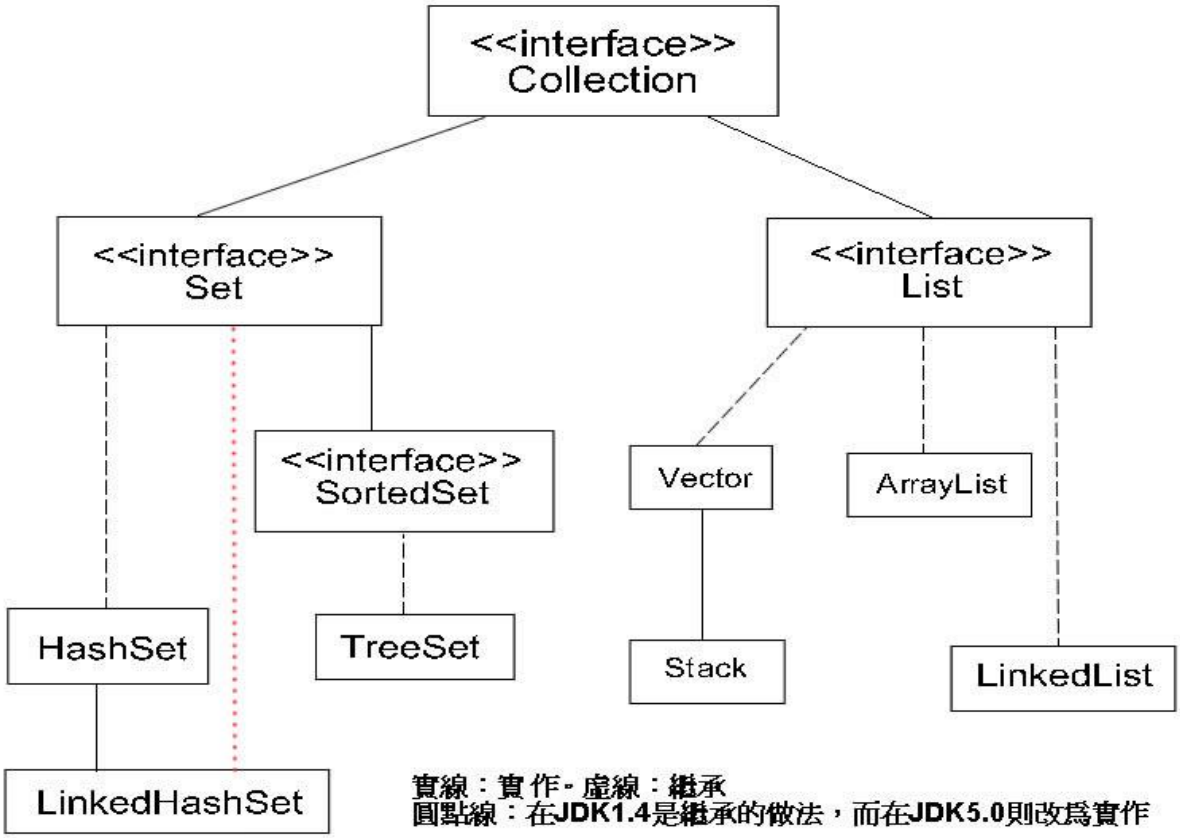
元件 Collection 六個核心介面

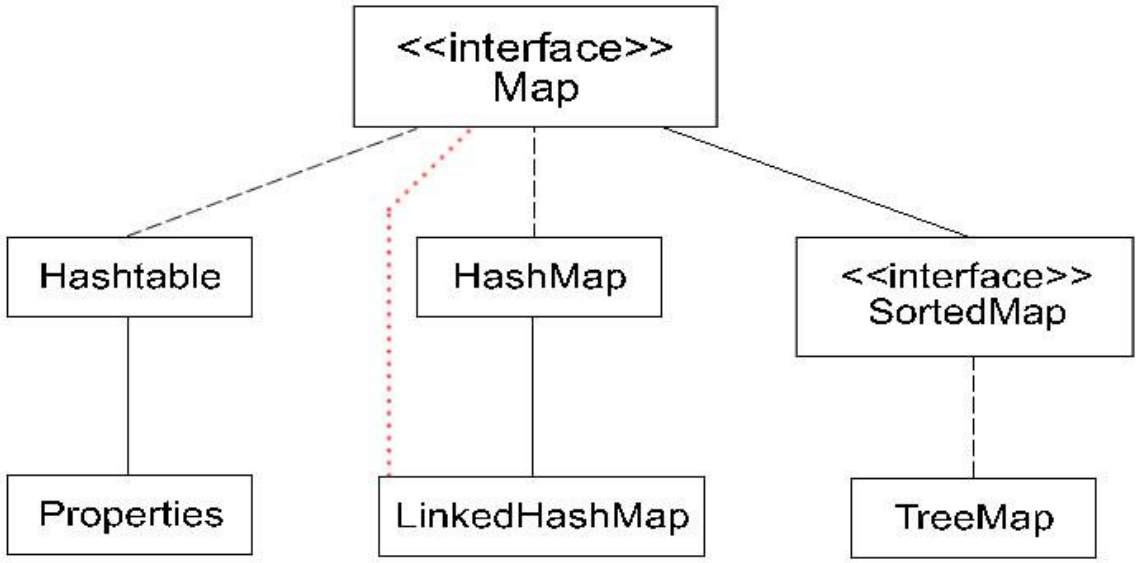
Collection	Set	Sorted Set
List	Map	Sorted Map

元件 Collection 十個核心的具體實作類別

<table>
<tr><th colspan="3">Collection</th></tr>
<tr><th>Set</th><th>List</th><th>Map</th></tr>
<tr><td>HashSet</td><td>ArrayList</td><td>HashMap</td></tr>
<tr><td>LinkedHashSet</td><td>Vector</td><td>Hashtable</td></tr>
<tr><td rowspan="2">TreeSet</td><td rowspan="2">LinkedList</td><td>TreeMap</td></tr>
<tr><td>LinkedHashMap</td></tr>
</table>

三、元件 Collection 的類別與介面階層圖





2-7-2-2　List、Set、Map 的特質是什麼？

一、元件 Collection 有三種基本的特質

Lists	資料索引，且有順序，內容會重複，但不會進行排序
Sets	唯一性，不允許重複內容存在
Maps	擁有唯一的識別碼(索引值)。

二、元件 Collection 介面的具體實作類別

<table>
<tr><th></th><th>類別</th><th>有順序</th><th>可重覆</th><th>Thread safe</th><th>排序</th></tr>
<tr><td rowspan="4">List</td><td>Vector</td><td colspan="2" rowspan="4">Yes</td><td rowspan="2">Yes</td><td rowspan="4">No</td></tr>
<tr><td>Stack</td></tr>
<tr><td>LinkedList</td><td rowspan="2">No</td></tr>
<tr><td>ArrayList</td></tr>
<tr><td rowspan="3">Set</td><td>HashSet</td><td>No</td><td rowspan="3">No</td><td rowspan="3">No</td><td rowspan="2">No</td></tr>
<tr><td>LinkedHashSet</td><td rowspan="2">Yes</td></tr>
<tr><td>TreeSet</td><td>Yes</td></tr>
<tr><td rowspan="5">Map</td><td>Hashtable</td><td rowspan="3">No</td><td rowspan="5">No</td><td rowspan="2">Yes</td><td rowspan="4">No</td></tr>
<tr><td>Properties</td></tr>
<tr><td>HashMap</td><td rowspan="3">No</td></tr>
<tr><td>LinkedHashMap</td><td rowspan="2">Yes</td><td rowspan="2">Yes</td></tr>
<tr><td>TreeMap</td></tr>
</table>

三、怎樣分辨 List，Map，Set？

透過上面的表單後，是不是有了眉目呢？基本上考題都會提示，這時如果你夠了解的話，將可以輕易地消去某些答案。舉例來說，當你需要名稱／數值配對的元件集合時，就不可能選擇 Map 做為答案，因為 Map 是一個介面，而不是一個具體的實作的類別。

例題 1：

a. Stores key/value pairs.

b. Allows null elements, keys, and values.

c. Duplicate entries replace old entries.

d. Entries are not sorted.

Which of these classes provides the specified features?

a. LinkedList　　b. TreeMap

c. TreeSet　　d. HashMap

e. HashSet　　f. Hashtable

g. All of the above　　h. None of the above

ANS：d

首先我們先看到這題到底再問些什麼，這題主要是

在問說，在以下這些類別之中哪一個許可上面幾項特徵，我們知道題目在問什麼了之後，接下來看到特徵那邊，A項索引值跟數值，這不是只有出現在Map裡，沒錯，這下子可以移除不是在Map類別元件，接下來看B項，允許空值或者數值，這下子因該很明顯的知道是HashMap了。

2-7-3 記憶體回收 Garbage Collector

2-7-3-1　JVM 的 Garbage Collector 如何運作？

Java 的記憶體回收行程，為記憶體管理提供了自動化的對策。在大多數的案例中，你根本不需要再應用程式中，加入任何管理記憶體的邏輯。自動化的記憶體回收行程也有一項缺點，那就是你無法完全控制它何時執行，以及何時不要執行。

記憶體回收行程是由 Java 虛擬機器控制的，Java 虛擬機器會決定何時執行記憶體回收行程。從你的 Java 程式中，你可以要求虛擬機器執行記憶體回收行程。不過，這並不保證在任何情況下，Java 虛擬機器都會接受你的要求。Java 虛擬機器會將執行的時機交給它自己的設備來決定，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。經驗顯示當你的 Java 程式要求執行記憶體回收行程的時候，Java 虛擬機器通常會馬上執行你的要求，不過並不保證一定會如此。當你覺得可以仰仗它的時候，Java 虛擬機器往往又會決定忽略你的要求。

2-7-3-2　符合被收走一個 object 的條件？

簡而言之，每一支 Java 程式都擁有從一個到多個的執行緒，每個執行緒都有它自己的執行堆疊。通常，一支 Java 程式中至少啟動一個執行緒，也就是 Main Thread 主要程式碼在 main()方法中。不過，當你在第九章學到極度細節的內容後，你會發現有許多很酷的理由，讓你從初始的執行緒中再啟動其它的執行緒。除了有它自己的執行堆疊外，每個執行緒也有它自己的生命週期。到目前為止，我們只需要知道執行緒可能是活著，也可能已經死亡。具備這些背景資訊之後，我們現在可以很清楚地說，**當沒有任何活著的執行緒可以存取某個物件的時候，它就符合執行記憶體回收**(Garbage Collector)。

例題 1：

```
1.    public class Garbage{
```

```
2.      public static void main(String[] args){
3.          StringBuffer tea = new
             StringBuffer( “Good Tea” );
4.          System.out.println(tea);
      //StringBuffer tea 還不適合回收
5.          tea = null;
      //StringBuffer tea 已經可以進行記憶體回收
```

當 tea 被放置 null 時，此時表示 tea 將沒有放置任何物件或者數值，此時完全符合被回收的條件了。透過此題，大家大概知道怎樣讓一個物件符合備回收的條件了吧。

2-7-3-3 finalize()被呼叫的時機

當你的物件被記憶體回收行程刪除前，執行某些程式。這段程式就放在 Finalize( ) method 之中，所有類別都從類別 Object 的地方繼承了這個方法。

finalize( ) method 的一些小技巧

a. 對任何特定的物件來說，Garbage Collector 只會呼叫一次 finalize( ) method。

b. 實際上，呼叫 finalize( ) method 可以讓某個物件免於被刪除的命運。

首先，要記得的是，finalize( )是一個方法。所以，你在一般方法中可以加入的指令，都可放進 finalize( ) method 之中。

2-8 內部類別

2-8-1 內部類別

2-8-1-1 設計「正常的」內部類別

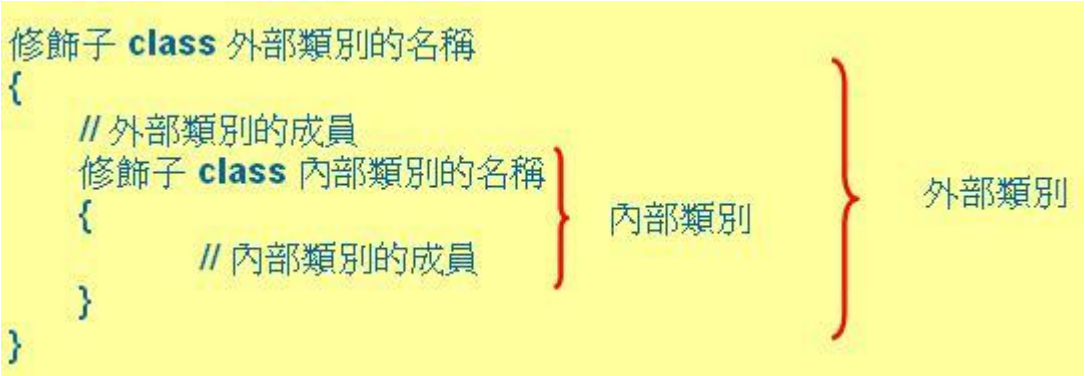
一、在這裡，我們將「正常的」用來表示不屬於底下這幾種的內部類別：

a. **靜態**

b. **方法區域**

c. **匿名**

二、在定義內部類別時，你會把它放在外部類別的大括號之內，如下：



三、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。

四、內部類別會在另一個類別的大括號內。

五、內部類別是外部類別完全合格的成員之一，所以它能使用存取修飾子與 abstract 或 final 修飾子。

六、內部類別也是外部類別的成員，所以其他成員可以

存取或呼叫內部類別的成員變數和方法，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他成員變數和呼叫成員方法。

七、內部類別也可以使用 pubilc，protected，private 修飾子。

八、從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例化，如下：

MyInner mi = new MyInner();

2-8-1-2 從內部類別之內參考內部或外部實例

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

a. 關鍵字 this 只可以從實例程式之內使用，換言之，不能從靜態程式之內使用。

b. This 是對目前執行中的物件所做的參考。

c. This 參考是物件能將參考以方法引數，傳給本身或其他程式的方法

方法：

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

2-8-2 方法內部區域類別

正常的內部類別，都會放在另一個類別的大括號之內，不過你也可以放在某個方法內定義

例題

```
1.  class MyOuter2 {
2.      private String x = “Outer2” ;
3.      void doStuff () {
4.         class MyInner {
5.            public void seeOuter()  {
6.            System.out.println( “Outer x is “ + x);
7.            } //結束內部類別方法的定義
8.         } //結束內部類別的定義
9.         MyInner mi = new MyInner();
10.        mi.seeOuter();
11.     } //結束外部類別方法 doStuff()的定義
12. } //結束外部類別的定義
```

解釋

一、方法區域內部類別會在外部類別的某個方法內定義

二、為了讓內部類別能被使用，你必須把它實例化。而且，這項實例化的作業必須發生在同一個方法之內，但是要在類別定義程式之後。

三、方法區域內部類別不能使用在該方法之內宣告的變數(這也包括參數在內)，除非這些變數標示為 final。

四、唯一可以套用在方法區域內部類別上的修飾子，就是 abstract 與 final，但不能同時使用。

2-8-3 匿名內部類別

不以任何類別名稱宣告的內部類別，因此取名為「匿名」。

主要目的是用來簡化程式碼,除了可繼承原類別的成員變數與成員函式，也可以另外新增變數與函式。

宣告的方式是直接在程式中以 new 關鍵字來建立類別實體，如下：

例題

```
1. class Popcorn  {
2.     public void pop()  {
3.       System.out.println( “popcorn” );
4.     }
5. }
6. class Food {
7.   Popcorn p = new Popcorn()  {
8.       public void pop()  {
9.           System.out.println( “anonymous
              popcorn” );
10.          }
11.     } ;
12.   }
```

解釋

我們定義了兩個類別，分別是 Popcorn 與 Food

Popcorn 擁有一個方法，叫做 pop ()

Food 則擁有一個實例變數，以 Popcorn 的型態宣告

Food 的內容就是這些，它並沒有任何的方法

Popcorn 參考變數所參考的對象不是 Popcorn 的實例，而是 Popcorn 的匿名子類別

第 7 行要注意結尾地方放的是大括號

第 8 行是匿名內部類別時的重點所在，也就是 overriding 超類別 Popcorn 的 pop（）的方法

第 11 行的分號是用來結束第 7 行程式開始的陳述句

一、匿名內部類別沒有名稱，而且它們的型態必須是具名型態的子類別，或是具名介面實作。

二、匿名內部類別永遠會以某個陳述式的一部份被建立，所以不要忘記在類別定義後，使用大括號將這個陳述式結束掉。這是在 java 中，少數可以看到大括號接在分號之後的情形。

三、由於多型的緣故，可以在匿名內部類別上呼叫的方法，只限於在參考類別中所定義者。即使這個匿名內部類別事實上是該參考變數型態的子類別或實作也不例外。

四、匿名內部類別可以延伸一個子類別，或實作一個介面。與非匿名類別不定的地方是，匿名內部類別無法同時執行這二種作業。換言之，它無法同時延伸某個類別，又實作某個介面。

五、引數區域內部類別會以方法呼叫的一部分，進行宣告、定義與自動實例化的作業。這種類別會在方法引數內定義，所以它的語法會以大括號結束類別的定義，然後再接上一個小括號，結束方法的呼叫，最後再接上一個分號！

2-8-4 靜態巢狀類別

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：

```
1.class BigOuter {
2.    static class Nested { }
3. }
```

這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。

在這種情況中，static 修飾子的意思是說，巢狀類別是外類別的靜態成員。

這表示它可以像其他靜態成員一樣被讀取，而不須要具備外部類別的實例。

2-8-4-1 實例化靜態巢狀類別

實例化靜態巢狀類別的語法，與正常的內部類別有點不同，它的寫法如下：

例題

```
1.    class BigOuter {
2.       static class Nested { }
3.    }
4.    class Broom {
5.       public static void main (Sting [] args)
            {
6.          BigOuter.Nested n = new
              BigOuter.Nested ();
7.          //同時使用兩個類別的名稱。
8.            }
9.    }
```

解釋

須要同時使用外部與巢狀的類別名稱，如下所示：

BigOuter.Nested n = new BigOuter.Nested ();

一、靜態巢狀類別指的是標上 static 修飾子的內部類別。

二、從技術的角度來看，靜態巢狀類別並不是內部類別，而應該視為項層的巢狀類別。

三、因為巢狀類別是靜態的，所以不會與外部類別的實例共享任何特殊的關係。

四、實例化靜態巢狀類別須要同時使用外部與巢狀的類別名稱，如下所示：

BigOuter.Nested n = new BigOuter.Nested ();

五、靜態巢狀類別無法讀取外部類別的非靜態成員。

2-9 執行緒

2-9-1 執行緒的定義、實例化與啟動

java 以 java.lang.thread 這個類別來表示 thread。

class thread 有兩個 constructor:

a. thread()

b. thread(runnable)

一、任何一個執行緒都只可以處於五種狀態之一：

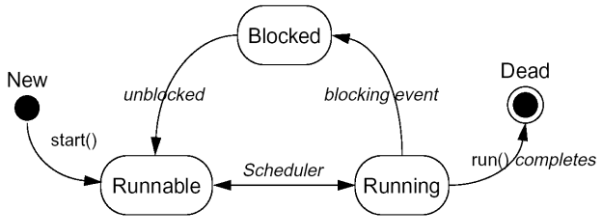
a. 新產生（ new ）

b. 可執行（ runnable ）

c. 執行中（ running ）

d. 等待/閉鎖/睡眠（ waiting/blocked/sleeping ）

e. 無作用 (dead)



新執行緒由 new thread() 所建立。執行 start() 方法時轉為可執行狀態 (runable)，若呼叫 yield() 方法則是將執行權轉交至其他的執行緒。可執行狀態若

執行阻塞型的輸入輸出、sleep()、wait()等方法時將從可執行狀態轉成不可執行狀態。可執行狀態若執行stop()方法，或 run() 方法執行完畢將轉成死亡的狀態。

二、方法

start( )、yield( )、sleep( )、run()，這些動作都會從 **run()**開始。

例題：

```
1. public void run(){
2.     //在這裡插入工作的程式碼
3. }
```

解釋：

你永遠都需要將準備在個別執行緒中處理的程式碼，放在一個 run()方法中

三、只可以在 thread 中呼叫 start()一次，一次以上會產生 RuntimeException。

四、同一個 runnable 物件為目標，建立多個 thread 物件是合法的。

2-9-2 防止執行緒的執行

能辨識出防止執行緒執行的條件

runnable 介面中只定義一個 run()方法,然後實例化一個 thread 物件時，傳入一個實作 runnable 介面的物件作為引數，Thread 物件會調用 runnable 物件的 run()方法，進而執行當中所定義的流程。

一、狀態

a. 睡眠狀態

b. 等待狀態

c. 需要鎖定物件產生的閉鎖狀態

二、睡眠狀態用來讓某個動作執行延遲一段時間。

三、sleep()是靜態方法會讓目前執行的執行緒進入睡眠狀態。一個執行緒無法讓另一個執行緒進入睡眠狀態。

四、如果存在優先權相同的可執行的執行緒，yield()方法有可能讓執行中的執行緒回到可執行狀態。

五、當一個執行緒呼叫另一個執行緒的 join()方法時，目前執行的執行緒會進入等待狀態，直到它聯結的執行緒完成工作。你可將 join()的方法想成：「某某執行緒，我想要加在你的後面執行。當你完成工作時，讓我知道一下，讓我進入可執行的狀態。

2-9-3 程式的同步化

2-9-3-1 如何使用 Synchronized method？

使用鎖定的機制。java 中每一個物件都內建一把鎖，只有使用在同步化方法時，才會發揮作用

一、可以使用"Synchronized"關鍵字來進行這個動作。

例題 1：

```
1. public synchronized void setNameAndID(String
                    name String id) {
2.     this.name = name;
3.     this.id = id;
4.     if(!checkNameAndIDEqual()) {
5.           System.out.println(count + "
                illegal name or ID.....");
6.     }
7.     count++;
8. }
```

解釋：

synchronized"的一個使用方式是用於方法上，讓方法作用範圍內都成為被同步化區域，就是有執行緒在執行 setNameAndID()時，會從物件上取得鎖定，其它執行緒必須等待它執行完畢，釋放鎖定之後，才會有機會競爭鎖定，取得鎖定的執行緒才可以執行 setNameAndID ()。

2-9-3-2　如何只 Synchronized 一個 block 的 code

同步化的設定不只可用於方法上，也可以用於某個程式區塊上，稱之為實例區塊同步化（instance block synchronized）

例題 2：

```
1.    public void setNameAndID(String name,
        String id) {
2.      synchronized(this) {
3.          this.name = name;
4.          this.id = id;
5.          if(!checkNameAndIDEqual()) {
6.              System.out.println(count + "
                    illegal name or ID.....");
7.          }
8.          count++;
9.      }
10.   }
```

解釋：

執行緒執行至"synchronized"設定的區塊時取得物件的鎖定，這麼一來其它執行緒暫時無法取得鎖定，因此無法執行物件同步化區塊，這個方式可以應用於您不想鎖定整個方法區塊，而只是想在共享資料在被執行緒存取時確保同步化時，由於只鎖定方法中的某個區塊，在執行完區塊後即釋放對物件的鎖定，以便讓其它執行緒有機會取得鎖定，對

物件進行操作，在某些時候會比較有效率。

一、同步化所犧性的自然就是在於執行緒等待時的延遲，所以同步化的手法不應被濫用，您不用將整個物件的方法都加上"synchronized"，有些方法只是單純的傳回某些數值，它並沒有對共用資料進行修改的動作，那麼它就不需要被同步化。

二、靜態方法可以被同步化，使用的是代表 java.lang.class 這個類別的實例的鎖。

三、當某個執行緒進入睡眠狀態時，它會帶著它的鎖一起入眠。

2-9-4 執行緒的互動

2-9-4-1 Thread 之間如何做互動

object 類別中有三個方法，分別是 wait()、notify()、notifyAll()，可以協助執行緒溝通某個事件的狀態。必須從同步化的方法或程式區塊內，呼吸這三種方法!執行緒不能在某個物件上啟動等待或通知的方法，除非它擁有這物件的鎖。

當多個執行緒共用相同的資料時，除了利用同步化的功能來解決問題，另一個方法是採用基礎類別 object 提供的 wait 與 notify 函式，建立執行緒之間的溝通機制，以決定執行緒是否可進入 runnable 狀態

2-9-4-2　了解 Thread 提供的 method 如何使用？

ms 是 long type、frature 是特點、instance 是實例

method name	feature	method type	class
yield()		static	thread
run()	instance	thread	
start()		instance	thread
sleep(ms)	會丟出 interrupexception	static	thread
join(ms)	會丟出 interrupexception	instance	thread
wait(ms)	會丟出 interrupexception	instance	object
notify()	須事先拿到 this object 的 lock	instance	object
notifyall()	須事先拿到 this object 的 lock	instance	object

2-9-4-3　wait、notify、notifyAll 使用時機、方法？

wait(wait()、notify()和notifyAll()方法只能使用於synchronized修飾的方法或synchronized敘述中。

a、wait()：使呼叫此方法的 thread 進入 blocking mode，並設為等待該 Object,該 thread 必須擁有該物件的 lock。blocking mode 下的 thread 必須釋放所有手中的 lock,並且無法使用 CPU。

例題 1：

```
1.    synchronized(abotherobject) {
      //它擁有 abotherobject 上的所有權
2.      try {
3.         abotherobject.wait();
4.     //這個執行緒釋出這把鎖，並進入等待狀態
5.     //如果要繼續處理，這個執行緒是需要這把鎖
6. //所以它可能會被閉鎖起來，直到取回這把鎖為止
7.      } catch(interruptedexception  e) {}
8.  }
```

解釋：

程式會一直的處於等待狀態，直到 abotherobject 上呼叫 notify()。當某物件啟動 wait()方法時，執行這段程式的執行緒會立刻釋出這個物件上的鎖。

b、notifyAll()：讓等待該 object 的所有 thread 進

入 runnable mode。

notifyAll(); //這會通知所有等待中的執行緒

所有的執行緒都會接到通知，而且開始競逐這把鎖。由於這把鎖會被每個執行緒使用並釋出，所以它們會立刻展開行動。當許多執行緒都在等待時使用 notifyAll()。

c、notify():讓等待該 object 的某一個 thread 進入 runnable mode。

例題 2：

```
1.    synchronized(this) {
2.       notify();
3.    }
```

解釋：

程式會通知任何目前正在等候這個物件的執行緒。當 notify()呼叫時，並不會馬上釋出它的鎖，而是等執行緒處理完同步化的程式為上。許多執行緒都在等待同一個物件，則會選其中一個來執行且選誰是不一定的。

一、 notify()方法作用是將訊號傳給一個，而且只有一個等待中的執行緒，同時這執行緒必須在同個物件的等待集散區。

二、 notifyAll()方法作用與notify()相同，只是它會將訊號傳給所有在等待這個物件的執行緒。

三、 這三個方法，必須從同步化的方法被呼叫!某個執行緒會在特定的物件上啟動wait()或notify()，而且這個執行緒目前必須持有這個物件的鎖。

四、 所謂Runnable Mode是指該Thread隨時可由作業系統分配CPU資源。blocking mode表示該Thread正在等待某個事件發生,作業系統不會讓這種thread取得CPU資源。

第三章 研讀心得

陳柏佑

證照考取對我來說相當的吃力，相較於國內專業證照來說國際專業證照，更是令我頭痛不已，但有題庫可以研讀對我來說可以放心了許多，因為是國際性的證照所以內容當然都是英文，所以這時團體合作就充分發揮了，起初先是各自稍微研讀，也會利用下課及專題時間找組員們互相討論，當然還是有些題目會不懂，這時就可以請教系上的老師們，以便我們更充分了解整個題庫的內容，所有組員裡我是最後考取證照的，聽到其他順利取得證照的組員分享後，對於考取證照的信心更是加倍，當然最後也順利取得證照。

由於整個社會開始大學化，要提高本身的競爭力就是從學歷以及證照下手，尤其是國際性的專業證照更是有用，希望學弟妹們大學生涯裡不是僅有取得大學學歷而已，考取許多證照更是重要的一環，趁著在學期間資源豐富，且有許多系上老師們可以請教，要是等到出社會後才想考取證照，光是心思跟金錢部分可以要花上好幾倍呢。

黃一峰

SCJP 這張證照的難度適中，剛拿到題庫的時候，對於題目的內容完全一竅不通，所以我們幾個組員決定開始利用下課時間一起討論題庫，一方面互相討論題目的內容，一方面互相勉勵不要放棄！但是，我們討論的過程當中，還是遇到許多不解的難題，當我們遇到困難的時候，就會在課後請教幾位專任老師，他們也會不願其煩的解釋並讓我們了解。之後，我們幾個組員分別不同的時間去面對考試，當我走進試場的時候，非常的緊張！但至少對於題庫已經非常熟悉，在作答的時候相對地也很快速的把題目答題完畢。

我覺得學弟妹可以在求學過程多為自己爭取更多證照，累積專業知識，以後出社會的時候，就可以把自己的專長表現出來讓業主知道！當然，考取證照的過程並不是很簡單，要付出相對的代價，也可以和同學互相切磋，並互相鼓勵！每個人都會為自己的理想去努力，年輕的時候就不要吝嗇的展開你的翅膀，努力之後的果實是最甜美的。

涂志承

升上了大三，開始找同學討論要做怎麼樣的專題或是考照，一開始本來是做專題可是發生了一些事，跟組員就決定考證照了，

決定考 SCJP 證照，學校在暑假的時候有開了幾門證照班，就去參加了 SCJP 的課程，都是英文的資料跟題目，真的很頭痛，有很多東西必須去做整理，翻譯題目阿找老師問一些比較不懂的地方，而且很多都是平常比較不常看到的東西，所以花了很多時間去了解題目的意思，跟組員常常互相找相關的題目資料來讀，還蠻辛苦的，因為要準備的東西很多，還好證照課老師有交一些秘訣，還有一些考試的方向，所以幫了很大的忙，準備差不多時就去報名考試了，說真的還好有做足了準備，不然真的不知道會怎麼樣，做專題真的會學到很多東西，不管在知識或是忍與人間的相互合作等等....

第四章 結論

學習的路途上，成員們都感到相當的棘手，基本到連講義上所提的重點，也必須思考很久，陌生的東西，讓我們學習的困難度增加了不少，還好成員們都有心想學，在遇到難題時，都能發揮團結的力量，一起思考解決並學習，我想這是讓我們能繼續閱讀並專心於認證考試的原動力，如果沒有大家的努力相互扶持，共同的監督、教學相長，我想這次的認證並不會如此的順利。

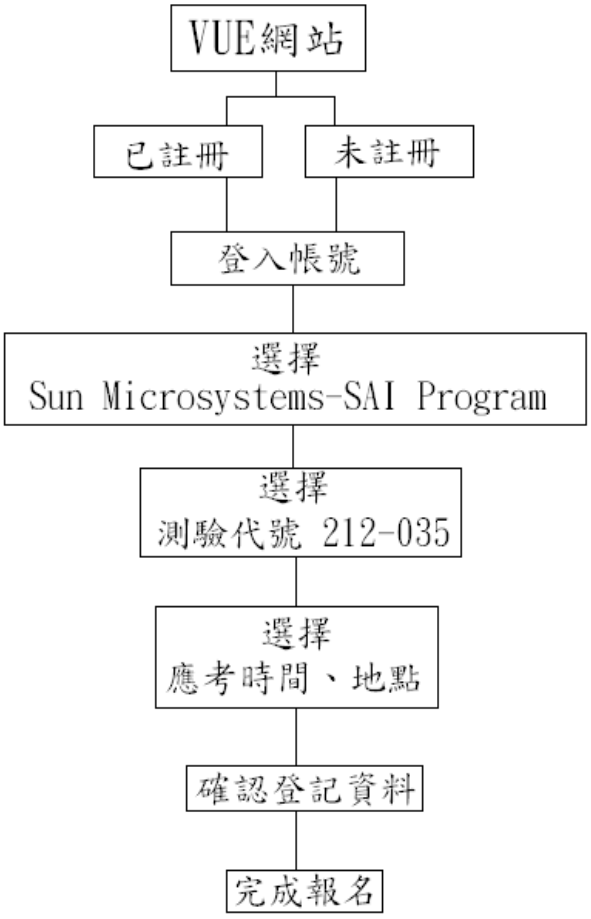
在這學習的過程中，除了成員們的努力，最重要的推手則是我們的指導老師，**高國峰** 老師。當我們在學習上的瓶頸或產生盲點時，肯花費時間、精力來指導我們，牽引我們走出疑惑的爪哇世界，讓我們在學習路程上能勇往直前更加的順利。沒有陳老師用心的指導，想順利取的認證的難度，就可能提高了許多。非常的感謝陳老師的指導，讓我們努力的目標能夠達成。

我們相信努力付出就一定有收穫，認證並不難考，最怕你不肯下定決心去取得認證。

附錄 A

1　、報名與考試程序

SCJP線上報考程序



2、參考資料

2-1 參考書籍

1. SCJP‧SCJD 專業認證指南

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

2. JAVA 2 SCJP 專業認證大全

作　者：陳培勳

出版社：學貫行銷股份有限公司

3. Java 認證 SCJP 5.0--猛虎出閘

作　者：段維瀚

出版社：碁峰

4. SCJP 重點題庫

附錄 B

1、專有名詞對照表

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	引數

Ch2

1.	public	公開
2.	private	私有
3.	protect	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.	preincrement operator	前置遞增運算子
6.	predecrement operator	前置遞減運算子
7.	postincrement operator	後置遞增運算子
8.	postdecrement operator	後置遞減運算子
9.	primitive data types	資料型別
10.	cast operator(type)	強制型別轉換
11.	pass-by-reference	傳址
12.	pass-by-value	傳值

Ch4

1. exception 例外
2. assertions 斷言
3. decision 判斷；決策
4. labeled 標記
5. iteration 迭代
6. guarded region 監控區域
7. stack tace 堆疊追蹤
8. exception propagation 例外傳遞
9. unchecked exception 未核對的例外
10. checked exception 以核對的例外
11. appropriate 適當
12. legal 有效
13. correct 正確
14. recursion 遞迴

Ch5

1. getter 取回方法
2. setter 設定方法
3. accessor 存取
4. instance variable 實例變數
5. variable 變數
6. extends 繼承
7. overloaded 過載
8. override 覆寫
9. composition 合成
10. accessor method 存取方法
11. predicate method 判斷方法
12. access modifier 存取修飾詞
13. utility methods 公用方法
14. helper methods 援助方法
15. abstract 抽象
16. control flow 流程控制
17. encapsulation 封裝
18. 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 脫序串列

6. word character 文字字元
7. concatenation 串接
8. literal 文字的
9. wrapper 外覆

Ch8

1. override 覆寫
2. inner class 內部類別
3. outer class 外部類別
4. anonymous inner calss 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 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 睡眠
11. dead 死亡
12. feature 特點
13. instance 實例

2、光碟資料

專題報告書(WORD 檔)

封面、書背、書面資料