# 物件導向程式設計導論

Introduction to Object Oriented Programming

# 目錄

# 前言

這裡的內容主要是希望能夠提升各位的物件使用能力

如何學？如何用？給個學習的方向　能更快學會程式

也順便推廣各位參加程式能力競賽　讓大家更具備參加的實力

希望各位能夠互相學習自己不拿手的地方　彼此互補　達到更好的學習效果

等這個正式打完後　如果還有需要學習其他東西　如：C++

也是可以的～　現在主要目的就是希望各位能先會使用物件

只要會使用物件　大部分的ＡＣＭ基礎題幾乎都會了　解出來只是時間經驗問題

物件的使用能夠用的好　那麼再難的語言　如號稱很難學的Object-C

適應力應該也會很強的　所以下面主要是介紹物件的使用和介紹　快速學習

# 前言

這份電子書主要是希望增強與幫助大家學習程式　給定方向

用於１０５Ａ班同學們的程式學習上

若需要做其他用途或是班外之類的地方上　請先告知~

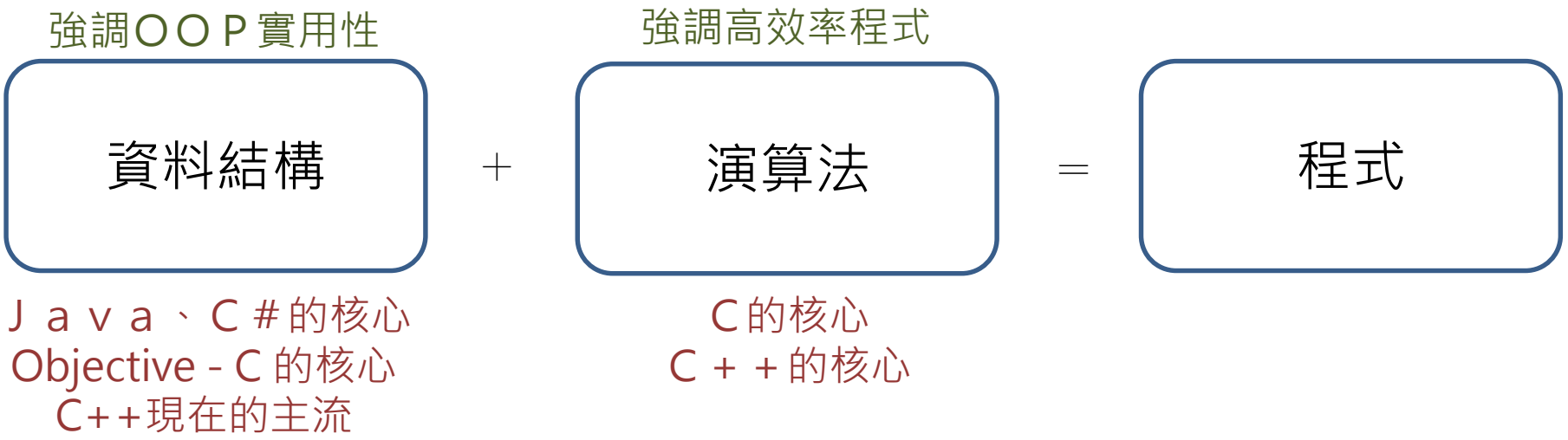
基本上有告知都會准許　至少要通知一下

如果不通知　而被抓到的話...　嘿嘿.....

# CH0 學習

# 如何學好程式？

多看　多寫　並且熟悉特性↓




Java提供了很多好用的類別，封裝好給使用者使用，不過缺點是速度較慢

所以對Java來說　演算法是蠻重要的　也就是說　資料結構和演算法都很重要

如：Android執行速度較耗時　硬體要求較高

Ｃ的速度是目前高階語言表現上優秀的語言之一，只是大部分的method需自行實作

# 如何學習**資料結構**？

熟悉函式庫或應用程式介面

C# library http://msdn.microsoft.com/library

C++ library http://www.cplusplus.com/reference/clibrary/

Objective-C library https://developer.apple.com/library/mac/docum

Java API http://download.java.net/jdk8/docs/api/

OOP不管是什麼語言 都是很重要的 下面只介紹Java的學習法

所以知道如何使用物件 也算學會其他語言的踏板

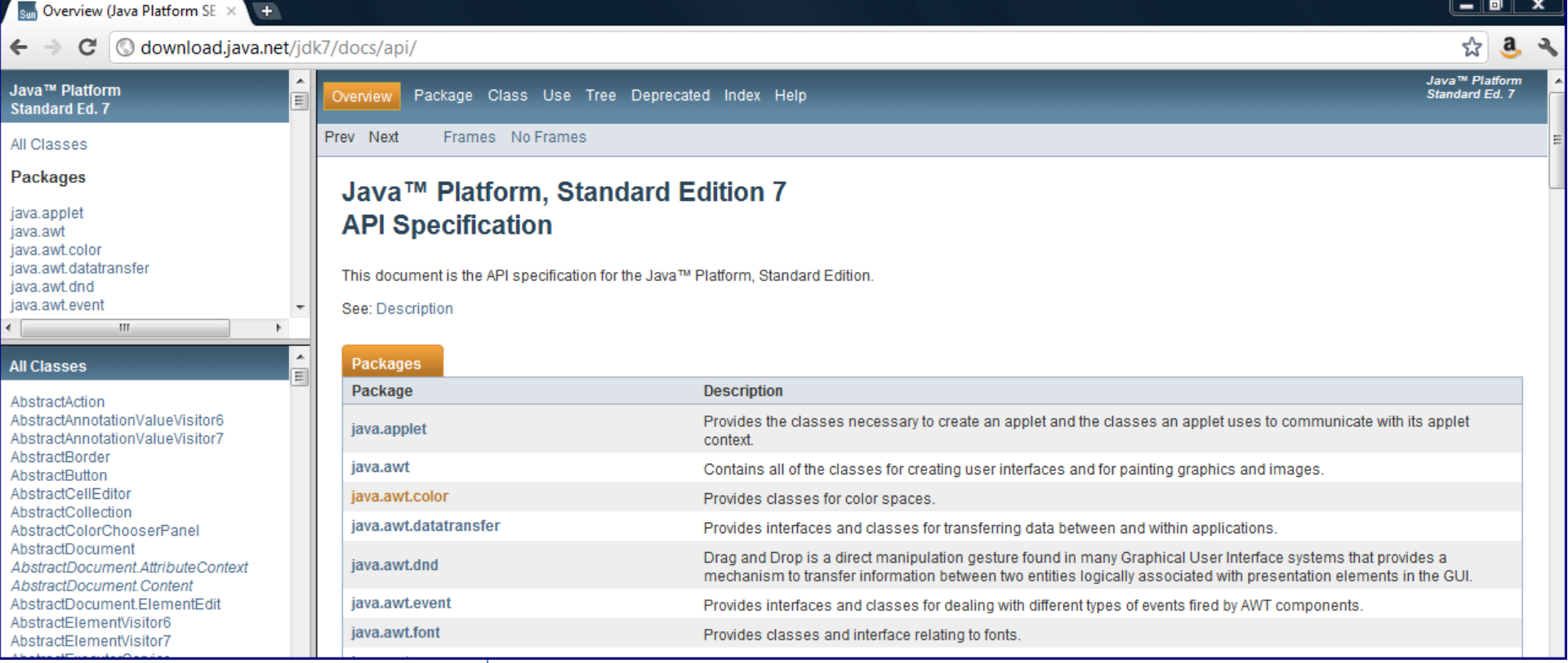
# 學好Ｊａｖａ的資料結構

熟悉ＡＰＩ

Ｊａｖａ　SE８　http://download.java.net/jdk8/docs/api/　（ｎｅｗ）

Ｊａｖａ　SE７　http://docs.oracle.com/javase/7/docs/api/

知道如何查詢




左上角
第一步
物件的封裝

左下角
第二步
封裝裡面的類別
（或介面、例外...）

內容
第三步
包含繼承族譜、method、繼承的method．．．

# 舉右邊的程式當例子

```
import java.lang.Integer;  // 常被IDE省略
public class Main
{
	public static void main(String[] args)
	{
		int a = 123;
		String s = Integer.toString(a);
	}
}
```

第一眼看到Integer.toString(a);你可能不知道是什麼功用

但是至少你知道這種宣告方式可以知道是使用Integer物件的method

所以

查到後學起來　再嘗試使用　你就會了！

easy　easy！

第一步　找java.lang

第二步　找Integer

第三步　找到toString的介紹　可以知道它的用法　還有載入和回傳的資料型態

| | |
|---|---|
| static **String** | **toString**(int i)Returns a String object representing the specified integer. |

# 如何學習**演算法**？

多看題型　多做題目　但不一定要打出程式

題目：

有一串葡萄和一顆柑橘，問有多少水果？

演算法：

1 + 1 = 2

程式：

```
a=1;
b=1;
ans=a+b;
```

通常打程式都是先想好演算法再打出演算法程式

如果直接打出演算法程式　可能會無從下手或是打到一半不知所打

所以最好先在紙上建構好雛型　再打出來比較好　除非已經很熟了

# 寫出演算法的技巧

程式的構造很簡單

只有判斷條件、迴圈、運算...　這些簡單的構造

但是要怎麼學好？

下面介紹幾個常見又實用的演算法！

# 枚舉法

枚舉法又叫做暴力破解法　是很常見也最常用的一種方法～　也是新手常用的

把所有可能性都列出來　盡可能的求出答案　（有惡補的意味．．．

必要時常被使用　如：ACM考試的時候針對測試資料的阻擋……

第一次不通過

```
for(.....)
{
    .....;
    .....;
}
```

第二次不通過

```
for(.....)
{
    .....;
    .....;
    if(n==100)
    {continue;}
}
```

第三次通過

```
for(.....)
{
    if(n==0)
    {break;}
    .....;
    .....;
if(n==100)
    {continue;}
}
```

# 枚舉法

當一個問題想不出要從何下手時　也是用枚舉法推出公式或是答案

如：　求２的１９９９次方的個位數是多少？

用枚舉法可慢慢推出正確答案

並且可以列出規律性的公式而寫出程式！！

# 貪心法

貪心法是演算法裡面最有名的一種方法　也很生活化

它主要不太在乎求出答案　而是只看見眼前的好處而投機取巧　接近答案

也是大家平常常做的事情　別說你沒做過喔～！

如：騎車只騎最短路徑、寫選擇題時，答案取近似值.....

舉例：

如果桌上有１００、５０、１０　１天只能拿１個　那你會怎麼拿？

一般人都是100 -> 50 -> 10

不懂？意思是　反正最後都會拿到160　所以我主要不在乎結果

因為結果都是一樣的　我是為了達成當下眼前的最大好處　這就是貪心法

# 貪心法使用原則：自以為可以求出答案

貪心法通常不是解決問題的主要方法　通常是輔佐

因為貪心法雖然可以以投機取巧的方式求出答案

但是並不是所有問題都可以這樣瞎搞出來　也並不是常常發生這種“巧合”

所以在貪心之前　也要有個原則在　考慮運氣、機率、是否自認為可以求出？.....

最常見的是　貪心法搭配枚舉法！

貪心法搭配遞增法

一開始沒有答案　但可以根據情況慢慢填滿答案

貪心法搭配遞推法

一開始先隨便設個答案　再一一修飾答案　直到求出漂亮的答案

# 遞增法

當一個問題很複雜的時候　最需要做的就是化簡

於是將問題變成一個一個慢慢做　也符合電腦本身處裡程式的特性

一一的慢慢填滿答案　雖然不是一次解決　但這種方法卻是很簡潔的

```
int sum=0;
int a[3]={1,2,3,4,5,6};
for(int i=0;i<a.length;i++)
{
   sum+=a[i];
}
```

變化前：　（需先知道有幾個變數

1+2+3+4+5+6

變化後：　（程序簡單　只要 2 數相加

0+1=1
1+2=3
3+3=6
……

# 遞推法

遞推法又被叫做迭代法（Iterative）

利用之前求到的數據來獲得新的數據

如：　很有名的數學問題　->　3n+1猜想

是偶數就除 2　是奇數就乘 3 再加 1　一直做下去　最後一定是 1

3　->　10　->　5　->　16　->　8　->　4　->　2　->　1

為了求到 1　計算過程不斷地用到之前的數據　這就是遞推法～

# 記憶法

1　預先處裡

電腦有個很大的特性就是有大量的空間可以儲存

所以我們可以先建一個資料庫　再給使用者輸入　到時候就不用再計算一次了

```
for(...)//資料庫
{
   成績紀錄;
}
for(...)//使用者使用
{
   查詢成績;
}
```

2　隨時記憶

計算的途中把資料儲存起來　以備不時之需

如上一個範例的sum

```
int sum=0;
int a[3]={1,2,3,4,5,6};
for(int i=0;i<a.length;i++)
{
   sum+=a[i];
}
```

# 遞迴法

遞迴法就是不斷地呼叫原本的程式來求出最後的答案

程式碼雖然很簡潔　但是耗時卻相當嚴重　並不是所有程式都適合遞迴

如：很有名的上帝問題　河內塔　給定幾個盤子　求出移動幾次

利用遞迴可以求出最後的答案（寫出過程也有點複雜

河內塔問題用遞迴法　電腦也要花好幾天才跑得玩

但是利用遞推法搭配記憶法可以馬上求出答案2^n-1

因為
N=1　ans=1
N=2　ans=3
N=3　ans=7

最後可以推出答案

# 分治法

分治法是把大問題切成很多小問題　如果小問題還是很難　那就繼續切

是一種精細度蠻高的演算法　主要分成 3 個階段：

1　原問題分割成小問題

2　解決所有小問題

3　整理出原問題的答案

如：排序法　　分成左右 2 堆來做排序 ( 方法之一

但是　也有不需要處裡所有小問題的題目

如：二元搜尋法　　找到答案後　剩下的數據就不用做了

上面介紹了這麼多演算法　都是常用而且實用的

這些學會後　自學也會很有效率的～

當然還有很多演算法　不過...先從簡單的學起吧～

在寫程式前　先想好適合解答問題的演算法

再來搭配ＯＯＰ引入　能讓程式變得更好！

希望上面的學習感想對大家的程式學習

和參加程式能力檢定會有很大的幫助！

有需要補充的我會再打進來

有問題再找我或是留言 . . .

# 練習程式的方法

下面推薦幾個學習程式的網頁　不必某天都做

每週看一點或是做一點　自然會對程式的直覺性很敏銳

| | |
|---|---|
| ＵＶａ學習網 | http://uva.onlinejudge.org/　(←推薦 |
| ＡＣＭ學習網 | https://icpcarchive.ecs.baylor.edu/ |
| ＣＰＥ學習網 | http://gpe.acm-icpc.tw/ |
| 高中生程式學習網 | http://zerojudge.tw/ |
| 大學生程式學習網 | http://judge.nccucs.org/ |

# 如何尋找適合自己的題目？

一顆星選集：http://cpe.cse.nsysu.edu.tw/environment.php#starList

雖然是一顆星　但是並不是每一題解起來都很輕鬆　所以不會也沒關係

同理　二星三星也並不是每一題都很難　也有很簡單的題目！

但是最重要的是訓練程式的直覺性　看到什麼問題會馬上想到“怎麼做”

目的並不是為了看到每個題目都會做　這樣到最後只會變成在學“題目”

全部星等（2010u.d.）：http://par.cse.nsysu.edu.tw/~advprog/advprog2010/star.htm

或參考考古題：http://cpe.cse.nsysu.edu.tw/history.php

考古題難易度分布：

一星３題　二星２題　三星１題　四星１題

題目都英文看不懂？：http://luckycat.kshs.kh.edu.tw/

# 練習程式與考試時常用的格式

```
import java.lang.*;   //考試時必須寫出來   因為常被使用   平常會自動被IDE引入
import java.util.Scanner;
public class Main   //cpe正式考時   M必須是小寫
{
    public static void main(String[] args)
    {
            Scanner scanner = new Scanner(System.in);

    }
}
```

請注意上傳的時候　名稱必須是Main！　其他就看你自己囉.....

下面我會示範一題來讓大家知道怎麼練習

示範題目是：**10055 - Hashmat the Brave Warrior**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem

# 10055 - Hashmat the Brave Warrior

Hashmat is a brave warrior who with his group of young soldiers moves from one place to another to fight against his opponents. Before fighting he just calculates one thing, the difference between his soldier number and the opponent's soldier number. From this difference he decides whether to fight or not. Hashmat's soldier number is never greater than his opponent.

**Input**

The input contains two integer numbers in every line. These two numbers in each line denotes the number of soldiers in Hashmat's army and his opponent's army or vice versa. The input numbers are not greater than 2^32. Input is terminated by End of File.

**Output**

For each line of input, print the difference of number of soldiers between Hashmat's army and his opponent's army. Each output should be in seperate line.

**Sample Input:**

10 12
10 14
100 200

**Sample Output:**

2
4
100

請自己先試著仔細了解題目...

找關鍵字　並不需要每個字都看

# 解釋

看懂了嗎？如果是英文問題　就表示很少看題目

常看題目就算英文不太好　也能看懂的　真的！

題目通常是嚇人用的　當你了解題目意思知道　你會發現比你想像中的簡單

讓我們來看看中文意思吧：

http://luckycat.kshs.kh.edu.tw/homework/q10055.htm

# 10055-Hashmat the brave warrior

Hashmat是一個勇敢的將領，他帶著年輕的士兵從這個城市移動到另一個城市與敵人對抗。
在打仗之前他會計算己方與敵方士兵的數目差距，來決定是要開打或不開打。
Hashmat的士兵數絕不會比敵人的士兵數大。

## Input
每組測試資料1列，有2個整數，代表Hashmat及敵人的士兵數或反之。
這些數不會超過$2^{32}$。

## Output
對每組測試資料請輸出Hashmat與敵人士兵數目的差（正數）。

## Sample input
10 12
14 10
100 200

## Sample Output
2
4
100

# 題目解釋

這題是在說明將領的故事　可是真正的問題是

叫你計算 2 方士兵數的差　也就是 2 數的差

但是請注意！！題目有陷阱“$2^{32}$”因為int的範圍被切一半

所以我們必須使用long　或是 unsigned int

所以是一個很簡單的題目！　竟然看懂題目了　那我們就來設計演算法吧！

演算法：

因為是隨便給 2 個數字　所以有可能相減是負號　但是士兵數必須是正的

所以算式應該是　｜ａ－ｂ｜　恆大於等於 0

所以這題的關鍵就是在絕對值了　那要怎麼寫呢？

# 演算法變成程式

方法一：

如果ａ＞ｂ　ａ－ｂ就直接輸出

如果ａ＜ｂ　ａｎｓ＝ａ－ｂ

然後０－ａｎｓ　就從負變正了

```
Scanner scanner = new Scanner(System.in);
long a = scanner.nextLong ();
long b = scanner.nextLong ();
If(a>b || a==b)
   System.out.println(a-b);
else if(a<b)
  System.out.println(0-(a-b));
```

# 演算法變成程式

方法二：

使用m a t h 物件裡面的method　使用絕對值

a b s ( ) ;

// import java.lang.Math;　被IDE省略

```
Scanner scanner = new Scanner(System.in);
long a = scanner.nextLong();
long b = scanner.nextLong ();
System.out.println(Math.abs(a-b));
```

通常引入物件可以讓自己解決程式的速度更快更容易

而且程式也會看起來簡潔有力　這也是ＯＯＰ重要的地方！！

# 上傳

先到這裡註冊會員　http://uva.onlinejudge.org/

註冊好後再回到題目網頁　http://uva.onlinejudge.org/index.php?option=com_onlinejudge&It
（不用到題目網頁也可以上傳　http://uva.onlinejudge.org/index.php?option=com_onlinejud
點選右上方的ｓｕｂｍｉｔ

之後選擇語言

然後貼上程式碼　或是直接載入檔案　然後送出

再來點選左方的　ｍｙ　ｓｕｂｍｉｓｓｉｏｎｓ　查看動態　（也會寄信給你

如果出現　ａｃｃｅｐｔｅｄ　表示通過～

答案不對　會出現　ｗｒｏｎｇ　ａｎｓｗｅｒ

如果出現其他狀態　有可能是其他問題

狀態參考：http://zerojudge.tw/

# 補充

但是要注意的是　題目有多少測試資料

如果題目沒有說要幾組測試資料　那你的格式必須是

```
Scanner scanner = new Scanner(System.in);
while(scanner.hasNext())
{
        //內容
}
```

如果題目有指定幾個測試資料　那格式就是

```
Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt();
for(int i=0;i<n;i++)
{
        //內容
}
```

# 補充

請注意時間限制！！尤其是使用ｊａｖａ撰寫題目

雖然ｊａｖａ的物件方便使用者　但是有些動作可能會增加編譯時間

有可能增加時間的情況　如：引入物件、卡在迴圈、多於資料的處裡.....

當物件不能達到題目的時間限制的要求時

這時候就要想更高效率演算法了　或是利用枚舉法慢慢除錯之類的

如剛剛那題

請注意　Time limit: 3.000 seconds　這就是時間限制

超時的話　上傳會出現ＴＬＥ（ｔｉｍｅ　ｌｉｍｉｔ　ｅｒｒｏｒ）

但是一般而言　一星和二星的題目很少會發生超出時間的問題

反而是三星四星的題目對高效率演算法的要求比較高

# 想練習卻看不懂題目？

可以參考：

http://luckycat.kshs.kh.edu.tw/

裡面提供翻譯過後的題目

# 分類題目

依章節分類：

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=118

# 參考題目（worm　up！）

**11172 - Relational Operator　　難度：一星**

http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem

提示：很智障的題目　比大小即可

# CH1 字串

# 學習java.lang.String 與 java.lang.StringBuilder

為什麼要先講字串呢？　因為字串很常用

而且也有很多學問　應該說是常被忽略的地方

因為這些常被忽略的地方沒注意到　所以常常導致很多人在用字串的時候

明明演算法是好的　可是用字串的時候　卻用錯方法

所以下面來介紹字串的重點！

# 數字比較 請自己試試看　想一下為什麼？

整數

```
int a=scanner.nextInt();
int b=scanner.nextInt();

 if(a==b)
 {
   System.out.println(true);
 }else if(a!=b)
 {
    System.out.println(false);
 }
```

輸入：
123　123
111　222

輸出：
true
false

字串

```
String a=scanner.next();
String b=scanner.next();

 if(a==b)
 {
    System.out.println(true);
 }else if(a!=b)
 {
    System.out.println(false);
 }
```

輸入：
123　123
111　222

輸出：
false
false

# 原因

因為在Ｊａｖａ中　他把字串視為物件的一種

所以建立字串等於是新增物件

但是一串字並不是數字　是一串字

一串字是不能比大小的

所以在Ｊａｖａ中　如果物件使用運算子

代表的意義就跟一般數字比較的方式就不同了

如：等等於

物件使用的話　就是比較物件的儲存位置

右邊是String完整的寫法

在Ｊａｖａ５以後的版本　String的寫法簡化了

所以String a=“123”；　常常被誤會成變數

```
//可縮寫成
//String a = new String(scanner.next());
String a=new String( “” );
a=scanner.next();
//可縮寫成
//String b = new String(scanner.next());
String b=new String( “” );
 b=scanner.next();
if(a==b)//錯誤的寫法！！
{
    System.out.println(true);
}else if(a!=b) //錯誤的寫法！！
{
    System.out.println(false);
}
```

輸入：
123　123
111　222

輸出：
false
false

# 正確的寫法

物件要互相比較內容　必須使用第一代祖先Ｏｂｊｅｃｔ的method

Ｊａｖａ只要是建立物件或是ｃｌａｓｓ　都會自動繼承第一代祖先Ｏｂｊｅｃｔ

```
public class String extends Object // extends Object常被省略
{
   ……;
}
```

所以我們就使用Ｏｂｊｅｃｔ裡面的equals()

補充：但是為了使用者方便找尋方法

所以你也可以在String裡面找到equals()

你會發現　同樣都輸入123

結果終於是true了

```
String a = new String(scanner.next());
String b = new String(scanner.next());

 if(a.equals(b))
    System.out.println(true);
else if(!(a.equals(b)))
    System.out.println(false);
```

輸入：
123　123
111　222
輸出：
true
false

# 系統輸入輸出與靜態成員

因為在Ｊａｖａ５以後的版本　加入了靜態成員的用法　靜態隨時儲存在記憶體中

所以可以讓使用者少打一些字　下面舉例：

```
import static java.lang.System.in;
import static java.lang.System.out;
public class Main
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(in);
      out.println(scanner.next());
   }
}
```

整數輸出

System.out.println( 內容 );

字元輸出

System.out.println( ‘內容’ );

字串輸出

System.out.println( “內容” );

# 字串的用法

一般變數互相轉換的用法

整數轉浮點數：

**int a=321;**
**float b=(float)a;　　（O）**

整數轉字串：

**int a=321;**
**String b=(String)a;　（X）**

別忘了剛剛提過字串是一個物件

並不是變數　所以不能互相轉換

所以我們要變數和物件互相轉換的話

必須使用物件裡面的method

剩下的還是請大家多多嘗試和練習了！

字串轉整數：

**String a=321;**
**int b=Integer.parseInt(a);　　（O）**

//已經建立過物件　就直接用物件變數的method

整數轉字串：

**int a=321;**
**String b=Integer.toString(a);　（O）**

//未建立過物件的話　就直接使用原始物件名稱

# 字串的進階用法

相信你經過上面的練習後　字串的使用一定栩栩如生了！！

那麼　我們來更進一步學習字串吧！

雖然String是很常用的物件　但是它有個缺點是

他的內容一旦建立後　就無法刪除　內部的method主要是建立固定資料

所以使用的時候　必須確保內容是固定的

雖然字串可以用運算子來做鏈結以增加內容

如：

String a = “123”；

a= a+”456”；

但是鏈結只是refrence位置

這樣做　頗耗執行時間的

也不會是個漂亮的程式

# 字串的進階用法

所以下面來介紹字串家族

| | | | | |
|---|---|---|---|---|
| String | Ｊａｖａ１以後的版本 | 資料最安全 | 速度慢 | 實用性不高 |
| StringBuffer | Ｊａｖａ２以後的版本 | 資料安全 | 速度快 | 實用性高 |
| StringBuilder | Ｊａｖａ５以後的版本 | 資料不安全 | 速度最快 | 實用性高 |
| StringJoiner | Ｊａｖａ８以後的版本 | 資料很安全 | 速度最快 | 僅拼裝用 |

我們會使用更有效率的字串物件來做運算之類的動作

現在一般常用的是StringBuilder　以前StringBuffer比較常用　２者用法是一樣的

StringBuffer是多執行緒　而StringBuilder是單執行緒　使用時可斟酌

最主要的是StringBuilder可以隨時變動資料內容

如：delete、insert、replace、……　　是String辦不到的

而且速度比String要快好幾倍　所以如果會使用的話　絕對是個很方便的物件

# 字串的進階用法

一般字串的寫法

String　a = “123”;
a+=“456”;

雖然只有加一次

可是會建立 3 個字串

123　456　123456

其他字串物件的寫法
//考試時記得寫出 import java.lang.*;
StringBuilder　a = new StringBuilder(“123”);
a.append(“4”);
a.append(“56”);

StringBuilder清空的方法：

1　a.delete(0,a.length());

2　a = new StringBuilder(“”);

StringBuilder還有很多很好用的方法可以用　剩下的就交給各位自己練習了

這裡主要是告訴大家那些地方是大重點　值得注意　和練習的方向

# 字串的實用法　取出所有元素做相加

題目：輸入一串數字做加總

```
String s = scanner.next();
 int sum=0;
 for(int i=0;i<s.length();i++)
 {
    sum+=s.charAt(i)-'0';
 }
```

題目：輸入一串小寫英文做加總

```
//a=0 , b=1 , ……
String s = scanner.next();
 int sum=0;
 for(int i=0;i<s.length();i++)
 {
    sum+=s.charAt(i)-  'a';
 }
```

請注意從字串取出來的型態是字元　所以必須減掉“初始值”

然後經過運算子的計算過後　型態會變成整數

不過有些題目可能不會這樣簡單　所以最好記一下ＡＳＣＩＩ　ｃｏｄｅ

不用全部記　只要記下面 3 個就等於記幾乎全部了　↓

０：４８　　Ａ：６５　　ａ：９７　　(不好記的話　看２進位會比較好記

# 注意載入型態和回傳型態

使用StringBuilder的method需要注意

StringBuilder sb = new StringBuilder(“123”);

String s = sb ;　（Ｘ）

StringBuilder的型態不是String　而是StringBuilder　（詳細參考ＡＰＩ

所以撰寫的時候要注意一下　可以寫成：

1　String s = new String(sb);

2　String s = sb.toString();

# 字串的英文大小寫轉換與子字串

簡單又實用的方法

```
String a = "AbCdE";

System.out.println(a.toLowerCase());          //abcde

System.out.println(a.toUpperCase());          //ABCDE

System.out.println(a.substring(1, 3));        //bC

System.out.println(a.subSequence(1, 3));    //bC
```

上面 2 個用法是一樣的　因為

```
public CharSequence subSequence ( int beginIndex ,  int endIndex )
{
   return this.substring(beginIndex, endIndex);
}
```

# 字串切割

有重複很多分隔符號　而且又想一一拿出來時　可以善用split()

```
String dot = "12/32/64/70";
String space = "12 32 64 70";

String[] s1 = dot.split("/");

for(String s:s1)
System.out.print(s+ “ ” );              //考試時請不要在輸出後面多放一個空白鍵！

System.out.println();

String[] s2 = space.split(" ");
for(String s:s2)
System.out.print(s+" ");
```

輸出：

12 32 64 70
12 32 64 70

來試試看你是不是對字串熟了呢？

這題很適合讓你驗收成果　也蠻簡單的　和上一頁的觀念有關

如果接下來的題目你已經有辦法解答了　代表你已經具備解題的實力了

# 參考題目

**12602 - Nice Licence Plates　　難度：一星**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=

提示：java.lang.Math; ->abs();　　**java.lang.String; -> charAt();**

＜延伸學習＞　**10921 - Find the Telephone**

**483 - Word Scramble　　難度：一星**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page

提示：練習字串反轉最好的題目->StringBuilder　　可搭配split()

**10018 - Reverse and Add　　難度：一星～二星**

http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem

提示：以StringBuilder來接收數字　反轉->reverse()　　運算時　轉為整數

# CH2 整數

# 學習java.lang.Integer

整數和其他變數比較起來　算是最重要的

常用到　而且只要用的好　再來就只需要擔心演算法的設計了

看到問題　一定是先設計演算法　再變成程式

如果設計好演算法　卻沒辦法變成程式

就表示是基本功沒做好　所以希望大家能夠紮穩基本功

以後在設計演算法的時候　才有辦法打出程式解決問題！

# 進制轉換

這個是以前計概的內容　如果不太熟的話　也務必練熟唷

有些問題會需要進制轉換　尤其是二進位最常用到

所以接下來　來介紹的就是進制運算

十進位變二進位的演算法：用遞推法一直除 2

所以變成程式可以寫成這樣：

這是十進位變二進位的演算法（只舉這個例子）

```
int a=scanner.nextInt();
StringBuilder sb = new StringBuilder("");
while(a!=1)
{
   if(a%2==1)
      sb.append("1");
   else if(a%2==0)
      sb.append("0");
   a/=2;
}
sb.append("1").reverse();
System.out.println( “二進制 : "+sb);
```

# 進制轉換

但是如果是用物件來寫十進位變二進位的話　非常簡單

前提當然是先知道物件裡面有哪些方法　不需要全部都記　至少要有印象

引物件的寫法：

```
int a=scanner.nextInt();
String s = Integer.toBinaryString(a);
System.out.println(  “二進制 : "+s);
```

轉二進位的步驟蠻常用到的　所以最好學起來囉

# 進制轉換

同理　求十進位轉成通用進位

```
int a=scanner.nextInt();

//二進位
String s1 = Integer.toBinaryString(a);

//八進位
String s2 = Integer.toOctalString(a);

//十進位
String s3 = Integer.toString(a);

//十六進位
String s4 = Integer.toHexString(a);
```

是不是超簡單的？

物件其實非常簡單　只要知道如何查

# 進制轉換

如果是倒過來　變成十進位呢？

```
//二進位變十進位
String s = scanner.next();  //e.g. 1010
String bs = Integer.valueOf(s, 2).toString();

//八進位變十進位
String s = scanner.next();  //e.g. 38
String os = Integer.valueOf(s, 8).toString();

//十進位變十進位
String s = scanner.next();  //e.g. 45
String ds = Integer.valueOf(s).toString();

//十六進位變十進位
String s = scanner.next();  //e.g. 5F
String hs = Integer.valueOf(s, 16).toString();
```

valueOf() 和 parseInt() 都可以

看你要怎麼寫了

下面幾張會介紹這 2 個的差別

# 進制轉換

十進位轉成其他進位的演算法：

```
int n = scanner.nextInt();//輸入十進位
int r = scanner.nextInt();//輸入要轉換的基底
StringBuilder sb = new StringBuilder("");
while(n!=0)
{
        sb.append( n%r );
        n /= r;
}
System.out.println(sb.reverse());
```

輸入：
127

輸出：
1111111

# 進制轉換

其他進位轉成十進位：

<整數>

```
int a=scanner.nextInt();  //使用者輸入的某進制
Int radix = scanner.nextInt();  //輸入要轉換的基底
String s = Integer.toString ( Integer.parseInt ( Integer.toString ( a ) , radix) ) ;
System.out.println(radix+“進制 : "+s);
```

<字串>

```
String s = scanner.next();  //使用者輸入的某進制
int radix = scanner.nextInt();  //輸入要轉換的基底
String rs = Integer.valueOf(s, radix).toString();
System.out.println(radix+"進制:"+rs);
```

# valueOf() 和 parseInt()的差別

當你在研究或是練習的時候　可能會很疑惑valueOf() 和 parseInt() 的差異

２個方法都是字串變成整數　但是不知道要用哪個　或是差在哪？

其實都可以！　因為在Ｊａｖａ５以後的版本

有了ａｕｔｏ-ｂｏｘｉｎｇ和ａｕｔｏ-ｕｎｂｏｘｉｎｇ的功能

當程式發現等號２邊的狀態一樣，但只差在 Primitive 和 Object 時　它會自動跟你封裝

下面的寫法是可行的　所以同理可知valueOf() 和 parseInt() 的用法可以說是一樣的～

當然必須是基本資料型態（Primitive Data Type）和物件資料型態（Object Data Type）才可

自動封裝（ａｕｔｏ-ｂｏｘｉｎｇ）：

```
int a = 123;
Integer b =a;
```

自動拆封（ａｕｔｏ-ｕｎｂｏｘｉｎｇ）：

```
Integer a = 123;
int b =a;
```

# hashCode()的作用

hashCode()是繼承第一代　java.lang.Object　就有的方法

他是雜湊碼　可以用來比較物件是否相同

但是equals()比較好用

而且因為是快速查詢　所以位置較不嚴謹

要找位置的話　最好另尋方法

可以用Ｏｂｊｅｃｔ下幾代的方法來求位置

如：Ｈａｓｈｔａｂｌｅ、ＨａｓｈＭａｐ.....

# compare() 和 compareTo()的作用

比較 2 個整數的長度　記住！是長度唷！！

```
Integer a = 12;
Integer b = 1234;
Integer c = 123456;
System.out.println(Integer.compare(a, b));  //    小於就輸出 - 1
System.out.println(Integer.compare(b, b));  //    等於就輸出    0
System.out.println(Integer.compare(c, b));  //    大於就輸出    1

Integer a = 12;
Integer b = 1234;
Integer c = 123456;

System.out.println(b.compareTo(a));  //    小於就輸出 - 1
System.out.println(b.compareTo(b));  //    等於就輸出    0
System.out.println(b.compareTo(c));  //    大於就輸出    1
```

# 補充

第一點大家一定都沒什麼問題

可是第 2 點和第 3 點大家一定會有疑問

在字串那個單元我已經介紹過

如果物件用等等於時　是比較記憶體位置

可是為什麼第 2 個是true呢？

因為只要範圍不超過　-128 ~ +127

系統會把資料放在緩存區

所以比較的位置當然會一樣囉 ~

大家可以自行實驗　蠻有趣的

< 1 >

```
int a = 1000;
int b = 1000;
System.out.println(a == b);
```

< 2 >

```
Integer c = 100;
Integer d = 100;
System.out.println(c == d);
```

< 3 >

```
Integer e = 1000;
Integer f = 1000;
System.out.println(e == f);
```

Output：

```
true
true
false
```

# 參考題目

**10019 - Funny Encryption Method　　難度：一星**

http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=9

提示：用　java.lang.Integer　可以做的很輕鬆～

**11185 - Ternary　　難度：一星**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page

提示：轉成三進位　簡單的演算法即可

**10473 - Simple Base Conversion　　難度：一星**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page

提示：也是很基本的進制轉換題目

# CH3 浮點數

# 學習java.lang.Double

浮點數在一般常見的計算可能不太常用

如：ＣＰＵ主要以整數運算為主

但是在繪圖有關的程式就有可能常用到

如：ＧＰＵ在３Ｄ呈現上需要大量的浮點運算讓圖形更細緻

（ｅ.ｇ.　ｎＶＩＤＩＡ的ＣＵＤＡ架構）

所以浮點運算也是有它的重要性

不過重點是　只要整數學得好　事實上浮點數的學習方向也大概會了！

而且２個可以使用的method事實上都差不多

不過還是有很多常被大家忽略的地方

所以接下來當然也是介紹一些重點了！

# 宣告方式

宣告方式事實上大家幾乎都會　課本上和網路上也能隨便查的到

所以我也不需要在這上面介紹什麼簡單的問題

不過Ｊａｖａ有很多小細節需要特別注意　這也是打這篇特別強調的

float　　a = 0.12;　　（Ｘ）
double b = 0.12;　　（Ｏ）

為什麼float不能這樣宣告？因為Ｊａｖａ預設只要建立浮點數

初始型態都是double　並不是float　如果double變成float可能會溢位

所以Ｊａｖａ是不允許的　右邊寫法也是錯的 float a = 0.12D;　　（Ｘ）

要２邊資料型態一樣　所以必須寫成　　float a = 0.12F;　　（Ｏ）

如果不可能發生溢位的話　當然是可行的->　double a = 0.12F;（Ｏ）

# 無限與ＮａＮ（Not A Number）

不知道大家還記不記得計概的浮點運算　是使用ＩＥＥＥ７５４規則運算

計算的時候　有幾個情況需要注意　－＞無限、負無限、ＮａＮ、０

正負無限：

```
double a = 100.0/0.0;
System.out.println(a);                          //輸出 Infinite
System.out.println(Double.isInfinite(a));       //輸出 true
 a = -(100.0/0.0);
System.out.println(a);                          //輸出 –Infinite
System.out.println(Double.isNaN(a));            //輸出 false
```

ＮａＮ：

```
double a = 0.0/0.0;
System.out.println(a);                          //輸出 NaN
System.out.println(Double.isNaN(a));            //輸出 true
System.out.println(Double.isInfinite(a));       //輸出 false
```

# 特殊值初始化與查看數值

回味一下ＩＥＥＥ７５４的格式：

正負號（ｓｉｇｎｅｄ）　指數（ｅｘｐｏｎｅｎｔ）　有效數字（ｍａｇｎｉｔｕｄｅ）

可以從物件直接取出final值　來做初始值

```
double a = Double.NEGATIVE_INFINITY;
double b = Double.POSITIVE_INFINITY;
double c = Double.NaN;
double d = Double.MIN_EXPONENT;
double e = Double.MAX_EXPONENT;

System.out.println(a);      //-Infinity
System.out.println(b);      //Infinity
System.out.println(c);      //NaN
System.out.println(d);      //-1022.0
System.out.println(e);      //1023.0
```

其中ＮａＮ需要注意

一旦計算過程有了NaN

那這串數據就整個壞了

```
double n = 0.0/0.0;
System.out.println((n-n));          //NaN
System.out.println((n-n)==0);       //false
```

# 進一步查看數值

仔細看API的敘述可以知道final值的原貌

負無限
double a = Double.NEGATIVE_INFINITY;
double a = 0xfff0000000000000L;

正無限
double b = Double.POSITIVE_INFINITY;
double b = 0x7ff0000000000000L;

N a N
double c = Double.NaN;
double c = 0x7ff8000000000000L;

# 字串與浮點數的轉換

浮點數轉字串：

```
double a = 0.12;
String s = Double.toString(a);
```

字串轉浮點數：

```
String s =  “0.12";
1   double a = Double.valueOf(s);
2   double a = Double.parseDouble(s);
```

# 補充

在Ｊａｖａ５以後的版本　除了多出Scanner物件方便使用者輸入輸出外

也新增了System.out的輸出方式　printf()　主要是Ｃ的輸出風格

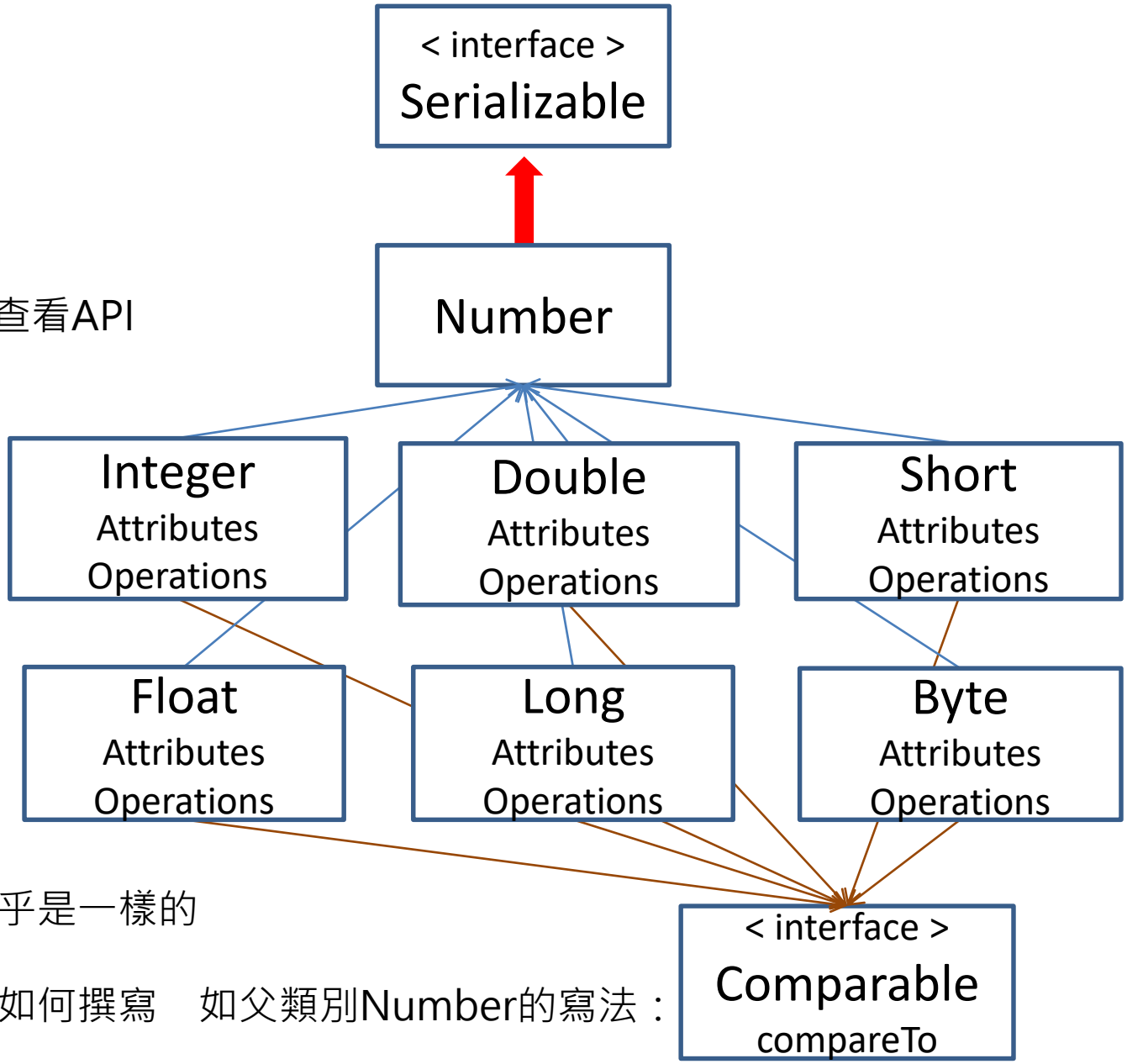
下面３種輸出方式都是一樣的：

```
double a = 0.12;
System.out.print("數值是 : "+a+"\n");
System.out.println("數值是 : "+a);
System.out.printf("數值是 : %f %n",a);
```

用法跟Ｃ幾乎一模一樣

# 補充

更詳細的內容請查看API




資料型態的繼承關係幾乎是一樣的

可以看出他們的類別要如何撰寫　如父類別Number的寫法：

public abstract class Number extends Object implements Serializable
{ …… }

# 參考題目

**11984 - A Change in Thermal Unit　　難度：一星**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page

提示：重新定義公式的題目　請忽視32

# CH4 字元與陣列

# 學習 java.lang.Character 和 java.util.Arrays

字元通常和字串的運算有很大的關係

通常陣列和字串、字元的使用常常是大家的絆腳石

所以這個單元特別把 2 個比較重要的地方都列出來給大家學習

# 編譯器蜜糖（compiler sugar）

編譯器蜜糖是讓使用者在撰寫程式的時候　能更方便編輯　就像吃到蜜糖一樣爽！！

整數單元介紹的ａｕｔｏｂｏｘｉｎｇ事實上也是編譯器蜜糖的一種

不過這個單元要介紹另外一種編譯器蜜糖

就是在Ｊａｖａ５以後的版本新增的ｆｏｒ－ｅａｃｈ

通常使用在陣列和 Java Collections framework 下面的物件（有迭代性<Iterative>的）

或是其他具有迭代性的物件

```
int a[] = {1,2,3};
for(int i : a)
{
      System.out.print( i +” ” );
}
```

輸出：
1 2 3

格式：

```
for( 變數宣告 : 陣列或物件 )
{
            //內容
}
```

“:”讀做ｉｎ

# 再次強調字串與字元的實用法

要知道如何拿字元做運算　這裡常常出錯　用的時候需要小心點　最好多練習這個部分

**輸入一串數字做加總**

```
String s = scanner.next();
 int sum=0;
 for(int i=0;i<s.length();i++)
 {
    sum+=s.charAt(i)-'0';
 }
```

**輸入一串小寫英文做加總**

```
//a=0 , b=1 , ……
String s = scanner.next();
 int sum=0;
 for(int i=0;i<s.length();i++)
 {
     sum+=s.charAt(i)- ‘a';
 }
```

**輸入一串大寫英文做加總**

```
// A =0 ,  B =1 , ……
String s = scanner.next();
 int sum=0;
 for(int i=0;i<s.length();i++)
 {
     sum+=s.charAt(i)- ‘A';
 }
```

從字串取出來的型態是字元　所以必須減掉“初始值”

然後經過運算子的計算過後　型態會變成整數

必記的３個ＡＳＣＩＩ　ｃｏｄｅ（Ｄｅｃｉｍａｌ）

０：４８　　Ａ：６５　　ａ：９７

# 另外一種字串與字元的實用法

如果遇到大寫和小寫混合的題目　也可以用下面的方法

（你可以試試用 StringBuilder 來寫出程式碼）

**輸入一串英文做加總，但統一以小寫運算**

```
//a=0 , b=1 , ……
String s = new String(scanner.next() .toLowerCase());
int sum=0;
 for(int i=0;i<s.length();i++)
{
   sum+=s.charAt(i)-  'a';
}
```

**輸入一串英文做加總，但統一以大寫運算**

```
//A=0 , B=1 , ……
String s = new String(scanner.next() .toUpperCase());
int sum=0;
 for(int i=0;i<s.length();i++)
{
   sum+=s.charAt(i)-  'A';
}
```

# 另外一種字串與字元的實用法

如果再加深難度　遇到小寫作小寫的　遇到大寫作大寫的運算呢！？

別怕　先冷靜下來想好演算法　你就能解開了！

雖然這題可能有其他偷吃步的方式可以寫出來

但是剛接觸新問題還是建議一步一腳印　才有辦法學到很多又不容易出錯！

( 你也可以試試看　再把數字相加的條件合在一起 )

```
String s = new String(scanner.next());
int sum=0;
for(int i=0 ; i<s.length() ; i++)
 {
        if('a' <= s.charAt(i) && 'z' >= s.charAt(i))
                sum+=s.charAt(i)-'a';
        if('A' <= s.charAt(i) && 'Z' >= s.charAt(i))
                sum+=s.charAt(i)-'A';
 }
System.out.println(sum);
```

# 字串和陣列

字串和陣列是語言中蠻重要的角色也蠻常用到的　在C中陣列是個大主角

可是在Ｊａｖａ卻變成2個重要的角色　　在運算過程中可能會常常轉換

所以這次我們來練習如何互相轉換

陣列變字串：

```
int a[]={1,2,3,4,5};
String s = Arrays.toString(a);
//import java.util.Arrays;
```

字串變陣列：

```
String s = "12345";
char a[]=s.toCharArray();
```

（需要和整數運算的話　配合 ASCII 即可）

# 判斷字元（數字）

有些題目可能只會用到數字　但是英文字或是其他符號卻是多餘的

又或是數字和英文字的處裡是需要分開的　所以我們有個方法可以判斷是否為數字：

```
String s = "1A2B3CD";
boolean result[] = new boolean[s.length()];
for(int i=0;i<s.length();i++)
{
	if(Character.isDigit(s.charAt(i)))
		result[i] = true;
	else
		result[i]=false;
}
for(boolean i : result)
	System.out.print(i+" ");
```

輸出：

true false true false true false false

# 判斷字元（字母）

下面的程式碼只是呈現方式不同　但是是一樣的

但是是用來判斷英文單字的method

```
String s = "1A2B3CD";
boolean result[] = new boolean[s.length()];
for(int i=0;i<s.length();i++)
{
	if(Character.isLetter(s.charAt(i)))
		result[i] = Character.isLetter (s.charAt(i));
	else if(!(Character. isLetter(s.charAt(i))))
		result[i]=Character. isLetter(s.charAt(i));
}
for(boolean i : result)
	System.out.print(i+" ");
```

輸出：

false true false true false true true

# 判斷陣列 - 排序法（ｓｏｒｔ）

陣列的運算另外比較常用的是　搜尋和排序　它有很多演算法　花費時間也不同

但是Ｊａｖａ有內建搜尋法和排序法給大家使用

但是使用搜尋法前需要注意　“通常”你搜尋的資料必須經過排序了！！

確定這個動作非常重要　必須牢記了　所以下面先介紹排序法

```
int a[] = {5,3,2,6,1,4};
System.out.println(“排序前：” +Arrays.toString(a));
Arrays.sort(a);
System.out.println(“排序後：” +Arrays.toString(a));
```

輸出：

```
排序前：[5, 3, 2, 6, 1, 4]
排序後：[1, 2, 3, 4, 5, 6]
```

注意：　System.out.println(a.toString());　會顯示雜湊碼　需自行實作toString()

# 判斷陣列 - 二元搜尋法（binarySearch）

它的搜尋方式很簡單　他會先把一串資料切成２半　如果是小於　就做左邊的資料

然後再切一半　小於大於哪邊的資料　依此類推下去.....

二元搜尋的結果會是你要找的資料的索引值　（　０　１　２　３　４　５・・・

如果找不到的話　就回傳　-（沒找到的位置）- １　　（詳細內容查看ＡＰＩ

```
int a[] = {5,3,2,6,1,4};
Arrays.sort(a);
System.out.println(Arrays.toString(a));
System.out.println(Arrays.binarySearch(a, 1));
System.out.println(Arrays.binarySearch(a, 4));
System.out.println(Arrays.binarySearch(a, 7));
```

輸出：

```
[1, 2, 3, 4, 5, 6]
0
3
-7
```

# 陣列複製

一樣使用 java.util.Arrays 的method

```
int a[] = {1,2,3,4,5};
int b[] = new int[a.length];
int c[] = new int[a.length];

b=Arrays.copyOf(a,a.length);
c=Arrays.copyOfRange(a, 1, 3);

System.out.println(Arrays.toString(b));
System.out.println(Arrays.toString(c));
```

輸出：

```
[1, 2, 3, 4, 5]
[2, 3]
```

# 陣列設初始值

有時候會需要讓陣列有初始值再做運算

數值初始化：

```
int a[] = new int[5];

Arrays.fill(a,1);
System.out.println(Arrays.toString(a));

Arrays.fill(a, 1, 3, 0);
System.out.println(Arrays.toString(a));
```

輸出：
[1, 1, 1, 1, 1]
[1, 0, 0, 1, 1]

真假值初始化：

```
boolean b[] = new boolean[5];

Arrays.fill(b,false);
System.out.println(Arrays.toString(b));

Arrays.fill(b, 1, 3, true);
System.out.println(Arrays.toString(b));
```

輸出：
[false, false, false, false, false]
[false, true, true, false, false]

# 忘記ＡＳＣＩＩ的補救辦法

一般推薦ＡＳＣＩＩ的記憶法是用２進位來記會比較好記

如果還不好記　那就記３個就好了（０、Ａ、ａ）

如果再記不起來　那在上機考的時候

可以把你想要知道字元的ＡＳＣＩＩ　ｃｏｄｅ顯示出來

```
int test = 'A';
System.out.println(test);
```

輸出：
65

# 如何實現無限空間的陣列？

每次使用陣列都要設空間　尤其是一開始不知道要多少空間時　很麻煩對吧？

Java Collection Framework內的 ArrayList 擁有模擬無限空間的能力

使用者不需要設太大的空間去占用記憶體

在後面的單元　會介紹

# 參考題目

12015 - Google is Feeling Lucky　　難度：一星

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&pa

提示：找最大值　再輸出最大值的網址即可

11942 - Lumberjack Sequencing　　難度：一星

http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_proble

提示：檢查有無排序的題目　不會很難

11917 - Do Your Own Homework　　難度：一星~二星

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&cat

提示：看起來很複雜　其實很簡單　注意字串的存放　和輸入順序

# CH5 特殊符號

有些運算符號蠻重要的

也有些大家常用的運算符號常被大家忽略

下面也會介紹其他運算符號的用法

# 編譯器蜜糖（compiler sugar）

編譯器蜜糖是讓使用者在撰寫程式的時候　能更方便編輯　就像吃到蜜糖一樣爽！！

整數單元介紹的ａｕｔｏｂｏｘｉｎｇ事實上也是編譯器蜜糖的一種

不過這個單元要介紹另外一種編譯器蜜糖

就是在Ｊａｖａ５以後的版本新增的ｆｏｒ－ｅａｃｈ

通常使用在陣列和 Java Collections framework 下面的物件（有迭代性<Iterative>的）

或是其他具有迭代性的物件

```
int a[] = {1,2,3};
for(int i : a)
{
      System.out.print( i +” ” );
}
```

輸出：
1 2 3

格式：

```
for( 變數宣告 : 陣列或物件 )
{
            //內容
}
```

“:”讀做ｉｎ

# 三元運算子　？：　（ｉｆ的特殊寫法）

ｉｆ有另外一種寫法　這種特殊寫法在很多語言上蠻常見的

尤其是Ｃ　所以算蠻重要的　不管是在解讀程式上或是考試之類的

剛開始看到莫名的程式碼　通常都會一頭霧水　如果能訓練到輕易解讀意思

當然是最好的～　廢話太多了！！　看下面吧……　不太好理解　不過常看就好了

題目：輸入２個數　再求２數的差　　格式

（判斷式）？（True則…）：（false則…）；

**一般寫法**

```
int a = scanner.nextInt();
int b = scanner.nextInt();
int c ;
if (a == b)
   c=0;
else
   c=Math.abs(a-b);
```

**特殊寫法**

```
int a = scanner.nextInt();
int b = scanner.nextInt();
int c ;

c = ( a==b ? 0 : Math.abs(a-b) );
```

# 產生物件

在Ｊａｖａ中　只要有ｎｅｗ　就是產生物件的動作

下面舉例：

```
Scanner scanner = new Scanner(System.in);

Integer a = new Integer(2);

//將StringBuilder的capacity設為１０　（其實系統會自動偵測　幫你增加capacity
StringBuilder sb = new StringBuilder(10);

sb.append("123");
System.out.println(sb);
System.out.println(a);
```

輸出：

```
123
2
```

# 1 的補數

如果使用運算子　~

會把１０進位轉成２進位後　進行１的補數　再轉成十進位

如果你計概學的有經驗的話　看到一個十進位數　通常可以很快的知道他的值

```
int a = 1227 ;
int b = ~a;
System.out.print(b);
```

輸出：
-1228

# 邏輯運算

做邏輯的動作一樣也是轉成 2 進位進行運算

&：ＡＮＤ　|：ＯＲ　^：ＸＯＲ　<<：左移　>>：右移　>>>：右移 ( 左邊補 0 )

```
int a = 8;
System.out.println("a&0 : "+ (a&0) );
System.out.println("a&8 : "+ (a&8) );
System.out.println("a|0 : "+ (a|0) );
System.out.println("a|8 : "+ (a|8) );
System.out.println("a^0 : "+ (a^0) );
System.out.println("a^8 : "+ (a^8) );
System.out.println("a<<1 : "+ (a<<1) );
System.out.println("a<<2 : "+ (a<<2) );
System.out.println("a>>1 : "+ (a>>1) );
System.out.println("a>>2 : "+ (a>>2) );
System.out.println("a>>>1 : "+ (a>>>1) );
System.out.println("a>>>2 : "+ (a>>>2) );
```

輸出：

```
a&0 : 0
a&8 : 8
a|0 : 8
a|8 : 8
a^0 : 8
a^8 : 0
a<<1 : 16
a<<2 : 32
a>>1 : 4
a>>2 : 2
a>>>1 : 4
a>>>2 : 2
```

# 字串遇到 + 時

在Ｃ中　是以陣列最後的資料為 / 0 來當作字串

但在Ｊａｖａ中　前面的單元也提過　字串視為物件而不是陣列的一種

而Ｊａｖａ在print的時候　如果字串另一邊遇到　+　的話

它會編譯成StringBuffer裡面的append()　所以下面 2 個輸出方式是相同的

```
int a = 3;
System.out.println("a="+a);
System.out.println(new StringBuffer("a=").append(a));
```

輸出：
a=3
a=3

注意唷　這是Ｃ沒有的寫法　Ｊａｖａ用+來省去麻煩的撰寫方式

所以我們平常在用Ｊａｖａ顯示出資料的時候　這也算是編譯器蜜糖的一種～

# 點

在程式中　不管是什麼語言

只要看到　.　都是讀做　**的**

如：　import java.util.Scanner;

# 補充

你看完上一張後　可能會想問一個很大的問題

上一章提過字串是一種物件　那陣列是不是物件的一種？

那我必須回答你　不要懷疑　是的！！

在Ｊａｖａ中　只要看到ｎｅｗ　你不用懷疑了　它就是物件！！

但是你可能常看到大家的寫法是　int a[] = new int[10];

但是這不是Ｊａｖａ中正規的寫法　正規寫法是　int[] a = new int[10];

如果你還不懂　那你可以再看看字串的宣告方式　String s = new String();

你會發現左邊和右邊必須是一模一樣的物件名稱　只是右邊的 [ ] 要設定capacity而已

變數宣告的讀法通常是右邊到左邊讀過去　所以這樣子想會更好理解

所以我們在Ｊａｖａ會稱　int[]　叫做　一維陣列物件

# 參考題目

**10469 - To Carry or not to Carry　　難度：一星**

http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem

提示：答案就在圖片上（注意carry）　輕鬆解決

# CH6 進階運算

# 學習java.lang.Math

在第 0 章的時候有請大家自行摸索裡面的方法

不過摸索過並不會馬上就知道怎麼用

所以下面來介紹一些常用而且容易搞混的地方

# 數學常數

java.lang.Math 內建常數包含圓周率和尤拉常數

這個內建的常數值　號稱比任何值還要來的接近

```
System.out.println(Math.E);
System.out.println(Math.PI);
```

輸出：

```
2.718281828459045
3.141592653589793
```

# 絕對值

之前有提到這個用法　用法也相當簡單

```
System.out.println(1-2);
System.out.println(Math.abs(1-2));
```

輸出：
-1
1

# 次方

需要注意的地方是　他的載入和回傳值都是ｄｏｕｂｌｅ

載入值系統會自動變成double　所以特別注意回傳值即可

int a = Math.pow(2,4);　　（Ｘ）
double b = Math.pow(2,4);　　（Ｏ）

System.out.println(Math.pow(2,4));

輸出：
16.0

# 三角函數

這裡介紹幾個簡單的三角函數用法就好　剩下比較難的就看大家要不要練習了

需要注意載入和回傳值都是double之外　還要小心載入的不是度　是弧度！

所以要輸入度的話　要寫成

```
double deg = scanner.nextDouble();
double rad = Math.PI * deg / 180.0;
System.out.println(Math.sin(rad));
```

你也可以利用反三角函數來是多少度

```
double d = scanner.nextDouble();
double Angle = Math.asin(d) / Math.PI * 180.0;
```

也提供了這種方法

```
double a = scanner.nextDouble();
System.out.println(Math.sin(Math.toRadians(a)));
System.out.println(Math.sin(Math.toDegrees(a)));
```

# 尤拉次方

也可以算出尤拉常數的多少次方

System.out.print(Math.exp(1));

輸出：
2.718281828459045

# 對數

也可以算出對數的值是多少

```
System.out.println(Math.log(Math.E));
System.out.println(Math.log10(10));
```

輸出：
1.0
1.0

# 亂數

產生 0 ~ 1 的亂數　常用於機率

System.out.println(Math.random());

輸出：
0.606635005829882

# 比大小

比較 2 數的大小後　回傳

```
System.out.println(Math.min(1, 3));
System.out.println(Math.max(1, 3));
```

輸出：

```
3
1
```

# 開根號

做開根號的動作

```
System.out.println(Math.sqrt(2));
System.out.println(Math.sqrt(4));
```

輸出：
1.4142135623730951
2.0

# 近似值

回傳最接近的整數值　型態不變

```
System.out.println(Math.rint(0.99999));
System.out.println(Math.rint(2.11111));
```

輸出：
1.0
2.0

回傳最接近的整數值　型態是整數

```
System.out.println(Math.round(0.99999));
System.out.println(Math.round(2.11111));
```

輸出：
1
2

# 上界與下界

離散數學的時候有上過上界與下界的定義　直接取上下界

```
System.out.println(Math.floor(5.856));
System.out.println(Math.ceil(5.856));
```

輸出：
5.0
6.0

# 參考題目

**10924 - Prime Words　　難度：一星**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&pa

**提示：會判斷質數與熟悉字元　即可輕鬆解答**

**10235 - Simply Emirp　　難度：一星～二星**

**提示：仔細看題目　質數題的難度加深版**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&pa

# CH7 大數運算

# 學習java.math.*

演算法不好+程式不太好的人　要做大數題可以說是比登天還難

但是Ｊａｖａ的大數物件福利到了Ｊａｖａ使用者

別人在那邊努力設計大數演算法的同時　Ｊａｖａ使用者已經用物件馬上解完題目了

所以如果對大數演算法沒把握學好的人　至少要會Ｊａｖａ的大數物件

# 大數初始宣告（一般）

我們先以大整數來做例子

因為大數是個物件　而且他又是數　所以它初始化常見下面

分別是 0 和空物件或是空節點

```
BigInteger a = new BigInteger("0");
System.out.println(a);

BigInteger b = null;
System.out.println(b);
```

輸出：

```
0
null
```

# 大數初始宣告（正統）

大數類別也內建初始變數給大家使用　如果你怕不保險　想嚴謹一點寫程式的話

這樣子寫　當然是最好的啦 ~

```
BigInteger n = null;
System.out.println(n);

BigInteger a = BigInteger.ZERO;
System.out.println(a);

BigInteger b = BigInteger.ONE;
System.out.println(b);

BigInteger c = BigInteger.TEN;
System.out.println(c);
```

輸出：

```
null
0
1
10
```

# 大數基本運算

大數運算內的花招蠻多的　下面介紹基本運算

```
BigInteger a = new BigInteger("6");
BigInteger b = new BigInteger("2");

//System.out.println(a+b);
System.out.println(a.add(b));
System.out.println(a.subtract(b));
System.out.println(a.multiply(b));
System.out.println(a.divide(b));
```

（Ｘ）千萬別忘記物件是不可以用運算子來計算的

輸出：

```
8
4
12
3
```

# 1 的補數與變負號

跟之前講的一樣　就只是邏輯運算　只是運算子要用在物件上　就要用method

效果等同於 ~

```
BigInteger a = new BigInteger("6");
System.out.println(a.not());
System.out.println(a.negate());        //變負號
```

輸出：

```
-7
-6
```

其他邏輯運算就給大家自己練習了

如：OR、左移、右移.....

# 取模數

Magic~不是啦　有一個運算子叫做%（ｍｏｄ）

要用在物件上面　方法是

```
BigInteger a = new BigInteger("7");
System.out.println(a.remainder(new BigInteger("1")));
System.out.println(a.remainder(new BigInteger("2")));
System.out.println(a.mod(new BigInteger("1")));
System.out.println(a.mod(new BigInteger("2")));
```

輸出：

```
0
1
0
1
```

２個方法都可以　看你要用哪個

# 大浮點數宣告

宣告方式和method使用方式也大同小異

一樣也交給大家自行練習

```
BigDecimal a = new BigDecimal("6");
System.out.println(a);

BigDecimal b = new BigDecimal("6.0");
System.out.println(b);

BigDecimal c = new BigDecimal("6.00000000");
System.out.println(c);
```

輸出：

```
6
6.0
6.00000000
```

# 參考題目

**10106 - Product　　難度：一星**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&cat

**提示：很簡單的大數相乘題目**

**ＵＶＡ題庫 - 大數運算使用Ｊａｖａ：**

http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&cat

# CH8 進階學習

如果你前面都有看的話　相信你的程度有了大大的進步

如果看得很挫折　那是正常的　程式經驗是累積來的

並不是１天之內苦幹完就是好的程式設計師　那已經叫做應付了事了

希望各位學程式不是為了應付學校或是怎樣的　最少最少要有一定的水準

Method也希望各位盡量不是用背的　因為考試的時候也可以查

所以只要知道方法就好了　程式用背的　老實說頗蠢的　也許是基礎不好

當然啦　如果程式基本功沒紮好　或是太久沒打程式　忘光光了（我曾經過...

我會建議你花１週找時間研究基礎　畢竟你已經學過了　看完保證靈感回來

看方法就好了　讀史書不如讀漢書嘛　是吧是吧！！

所以下面要幫各位建立更深的基礎　我覺得對大家有幫助　重要的　放上來提醒大家

如果你有覺得對大家重要的資料　也歡迎提供啦～　只要程式感建立了

你後面學程式也不需要花太多時間　因為方法感覺你都會了　加油啦

# 建構元（ constructor ）

有人叫他建構元、建構子、建構值、建構函數……　反正都一樣啦

你怕別人指正或是怎樣的　你就直接講constructor

我必須說　建構元　超～～～極重要！！　尤其是你在用物件的時候

會不會想　到底該在裡面打什麼？

如看到：StringBuilder sb = new StringBuilder( ???????? );

不知道要打什麼的時候　你就要去看ＡＰＩ裡面的介紹啦～

請打開找到　**Constructor Summary**　裡面都是建構元的overloading（後面介紹）

如果什麼都不打StringBuilder sb = new StringBuilder( );　就直接給16個空間

打整數StringBuilder sb = new StringBuilder( 8 );　直接設定 8 個空間

打字串StringBuilder sb = new StringBuilder( “1234” );　把字串直接丟進去

會看了吧　超簡單的是吧！！

# 多載（overloading）

多載的觀念可以說是設計師的小常識

意思是同樣的方法可以有很多的載入方式

如剛剛提到的StringBuilder就是一種多載

StringBuilder( )

StringBuilder( CharSequense  seq )

StringBuilder( int capacity )

StringBuilder( String str )

# 複載（overriding）

很多人懂多載　但是複載有些人就不太能理解　但是是很簡單的

就是父類別已經有的method　子類別繼承下來　但是把他的method改寫了

這個在Ｊａｖａ的ＡＰＩ上其實很常看到

如： equals()

父類別有equals()　但是子類別也有equals()

子類別直接用父類別的equals()可能不能用　需要經過改寫才能用

# 物件導向（ＯＯ）

ＯＯ是物件導向　不是表情符號　常看到的是ＯＯＰ　物件導向程式語言

但是　你可能會在一些地方看到ＯＯＤ　其實２個是差不多的

所以如果你想選ＯＯＰ卻沒得選　可以選ＯＯＤ　但是硬要說　這２個又不太一樣

也可以說　ＯＯＰ比較偏向Ｊａｖａ但Ｃ++勉強算　ＯＯＤ則用在Ｃ上面講比較洽當

當然　在Ｊａｖａ還沒正式問世的時候　ＯＯＰ當然是Ｃ++的代表了

至少要知道　**ＯＯＡ（分析）->ＯＯＤ（設計）->ＯＯＰ（實現）**

並不是倒過來喔！！當然你要倒過來設計也是可以啦～

可以這樣竄改程式的頻率會很高　就像我們寫程式都會先想好演算法再寫程式

這是一樣的道理～～

# 靜態（static）

Ｊａｖａ是一種很人性化的語言　也就是因為這樣才比Ｃ還好學

大部分的人會覺得不好學主要是因為英文問題　這習慣就好了

這裡要說的是　Ｊａｖａ不像Ｃ這麼複雜　有分什麼靜態動態CPU暫存變數...有的沒的

Ｊａｖａ很直接　靜態　　沒寫就是動態或是靜態省略（通常是動態）

如：(public) class Main{.....}　裡面的 (public) static void main(String[] args){.....}

為什麼要加static？　因為當你用主要類別去呼叫副類別的時候

你一定會跑去讀副類別　但是主類別怎麼辦？難道說讀到一半就要全部砍掉重來了嗎？

所以必須加上static！！　它會自動把資料暫時存在記憶體中　下次繼續用

補充：　Ｃ有個ＣＰＵ暫存變數是Ｊａｖａ沒有的　使用這個可以使程式跑更快

但是注意　這個變數是不可以多個宣告　通常２～３個是極限　畢竟ＣＰＵ記憶體有限

# 改寫（override）

上幾張提過複載的基本觀念　在ＩＤＥ上　有提供@override　的功能

只要在你要改寫的method上面加上指令　ＩＤＥ就會自動判斷是否改寫

通常用在改寫 interface 實作上面的類別　或是內建方法的加強版本等……

如：toString()　是正確的名稱　但是要改寫的話　就必須名稱一模一樣

```
public String tostring()　（ＩＤＥ判斷正確）
{……}

@override
public String tostring()　（ＩＤＥ判斷錯誤）
{……}
```

# ＡＤＴ（Abstract Data Type）

ＡＤＴ的中文翻譯是　抽象型資料型態

這個觀念非常重要　是學習資料結構前的必備基礎　計概也特別強調它的重要

後面在介紹集合的時候　會再強調一次這個的重要性

一般你常看到用來宣告變數的資料型態通常是：int、double……

但是為什麼有時候卻會看到其他型態在宣告變數之類的？如：String、Stack……

如果我這樣呈現　你已經看出是不是物件的差別　很好　你已經更進步了

但是為什麼要叫抽象型資料型態？　你可以想想　物件能當變數嗎？

應該不行吧？　為什麼可以？　因為是想像出來的嘛！！

所以才叫抽象型資料型態　　這樣子解釋　應該比課本好懂多了

如：String s = “123”；　　這就是一種ＡＤＴ

# 泛型（Generic Type）

以前的Ｊａｖａ在宣告抽象變數的時候　跟一般物件沒什麼２樣：

Stack stack = new Stack();

但是在Ｊａｖａ５以後的版本　增加了泛型的用法　這個用法跟Ｃ＋＋非常像

你可以定義一個抽象類別負責存取整數　甚至是字串：

Stack<Integer>　stack = new　Stack<Integer> ();

Stack<String>　stack = new　Stack<String> ();

以此類推　　雖然越後面的版本越來越抄襲Ｃ

但是複製過來的用法　也越來越人性化　也能使程式更嚴謹的呈現

# 泛型（Generic Type）

可是有些類別可能在設計的時候　並不是固定存取某些型態

而且宣告的時候　也不一定固定１個型態使用

所以在Ｊａｖａ５以後的版本　也加入了把泛型更加強化的代號可以使用

常常使用在實作上面和常見於 Java  Collection  Framework 上

當你看到<E>的時候　表示這個類別支援泛型　是給客戶端宣告的E型態

E只是一個型態代號　並沒有什麼特別的意思　也是一種例行習慣

當然　你高興的話　你也可以用T、V、K……

但是使用泛型的話　可能會造成很多語法上的麻煩

但是對客戶端來說　會有很大的好處

# 泛型簡化

這裡要介紹另外一種編譯器蜜糖

你會不會覺得很麻煩　前面都宣告過型態了　後面還要再打一次？

舉例　原本的寫法：

```
Stack<Integer>   stack = new   Stack<Integer> ();
```

但是在Ｊａｖａ７以後的版本　你可以這樣寫：

```
Stack< E >   stack = new   Stack<> ();
```

或是這樣：

```
Stack< E >   stack = new   Stack ();
```

當然……也可以寫這樣：

```
Stack  stack = new   Stack< E > ();
```

# 泛型簡化

我們再來介紹另一種編譯器蜜糖

過不了：　　（java.util.List）

```
List aList = new LinkedList();
aList.add("123");
String s = aList.get(0);    （X）
```

但是改成這樣　過了：

```
String s = (String) aList.get(0);    （O）
```

# 泛型

下面的方法是過不了的

使用者打的：

```
List<String> aList = new LinkedList<String>();
aList.add("123");
Integer a = aList.get(0);
```

但是編譯器會翻譯成：

```
List aList = new LinkedList();
aList.add("123");
Integer a = (String) aList.get(0);
```

很明顯的　是過不了的

# 泛型

Interface一旦支援泛型　實作起來會較方便

例如之前提到過的comparator

```
public  interface  Comparator<T>
{
        int  compare( T  o1 , T  o2 );
        ……
}
```

表示實作的時候　可以指定 T 的型態　所以compare()就可以套用T型態了

但是注意　不同代號不可以互用　如：指定 T 之後　就不可以給下面指定 E

# Interface實作改寫

舉例comparator介面被實作的一個例子

```
Class  StringComparator2  implements  Comparator<String>
{
        @override
        public int compare(String s1 , String s2)
        {
                return  s1.compareTo(s2);
        }
}

public class Main
{
        public static void main(String[] args)
        {
                List word = Arrays.asList( “B” ,” A” ,” C” );
                Collections.sort(word,new StringComparator2());
                System.out.println(word);
        }
}
```

# 泛型

當你打開ＡＰＩ的時候　看到<E>　就表示此類別支援泛型

而且　Java  Collection  Framework　所有的類別都支援泛型！！

對於泛型還有很多細節　但是對入門者來說　了解到這裡就已經很優秀了

再繼續下去　就已經到很高階的地段了　以後有機會再說啦～

之後會對集合再介紹一次　集合也是相當重要的介面

**java.util**
**Interface Collection<E>**

Type Parameters : E - the type of elements in this collection

All Super interfaces : Iterable<E>

All Known Subinterfaces :
BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>

# 抽象類別與介面（abstract class and interface）

很多人會有個疑問　為什麼Ｊａｖａ有了抽象類別　還要搞個介面出來攪局

雖然這2個都只是個雛型　並沒有真正的實作出來　但是卻有不少不同的地方

在Ｃ中　類別要怎樣繼承都行　但在Ｊａｖａ中　只能允許一個父類別

可是有些類別卻必須再有其他父類別繼承　於是就有了interface　就這樣誕生了

為什麼要這麼麻煩？又產生了一個多的東西　這是Ｊａｖａ和Ｃ＃的一個模糊點

身為最新的程式語言　雖然學習容易　但是很多地方卻比Ｃ++還要複雜化

所以國內才會有很多學者依然提倡Ｃ++　而不是現在最夯的新語言

介面說穿了其實是抽象類別的加強版　做了抽象類別辦不到的事情

其實不需要搞得這麼複雜讓自己搞混　只需要知道介面要怎麼繼承都可以　也是抽象

而抽象類別只能單一繼承　也是抽象

# 封裝與拆裝（boxing and unboxing）

前面的單元提過了　這裡再強調一下

在Ｊａｖａ５以後的版本　有了封裝和拆箱的功能

當程式發現等號２邊的狀態一樣，但只差在 Primitive 和 Object 時　它會自動跟你封裝

下面的寫法是可行的　所以同理可知valueOf() 和 parseInt() 的用法可以說是一樣的～

當然必須是基本資料型態（Primitive Data Type）和物件資料型態（Object Data Type）才可

自動封裝（ａｕｔｏ-ｂｏｘｉｎｇ）：

```
int a = 123;
Integer b =a;
```

自動拆封（ａｕｔｏ-ｕｎｂｏｘｉｎｇ）：

```
Integer a = 123;
int b =a;
```

# 補充字串單元未提到的新類別 StringJoiner

這是Ｊａｖａ８新增的字串新用法　位於java.util. StringJoiner

以較高的效率來拼裝一些代碼　而且新版本也在java.lang.String中新增 String.join()

網上有人測試它的處裡速度　速度跟StringBuilder幾乎一模一樣　可以說很快了

至於要什麼時候用上　目前還不知道了…　下面提供參考：

```
StringJoiner sj = new StringJoiner(",",  “[",  “]");
sj.add("A").add("B").add("C").add("D");
String s = sj.toString();
System.out.println( s );
```

輸出：
[A,B,C,D]

# CH9 例外與除錯

# 除錯

這是一個看似非常普通的除法運算

看似沒問題　其實卻隱藏了造成程式錯誤的輸入

```
int a = scanner.nextInt();
int b = scanner.nextInt();
System.out.println( a/b );
```

輸入：
10  2
10 0

輸出：
5
Exception in thread "main" java.lang.ArithmeticException: / by zero

你會發現　0 是不能當除數的　所以需要避免這種錯誤發生

# 除錯

修改了程式碼後　你會發現結果改善了　不會再拋出異常

```
int a = scanner.nextInt();
int b = scanner.nextInt();
//System.out.println(b!=0?a/b:"除數不可以是 0 ");

if(b!=0)
        System.out.println( a/b );
else
        System.out.println("除數不可以是 0 ");
```

輸入：
10  2
10 0

輸出：
5
除數不可以是 0

# 除錯

但是剛剛的除錯　是發現會產生錯誤的情況下　而除錯

如果能事先預測或是知道會發生什麼樣的錯誤　而先設置　當然最好

除了用if來除錯之外　ｊａｖａ有內建很多例外物件方便設計者

請回頭看上面的輸出　你會看到

Exception in thread "main" java.lang.ArithmeticException: / by zero

請仔細看　Exception是例外發生

他拋出了異常　例外物件就是　java.lang.ArithmeticException

物件名稱叫ArithmeticException　運算例外

發生了什麼事情造成異常？ / by zero

# Ｔｒｙ　ｃａｔｃｈ　ｆｉｎａｌｌｙ

Ｃ＋＋有Ｔｒｙ　ｃａｔｃｈ可以使用　而ｊａｖａ不只可以用

還多了ｆｉｎａｌｌｙ　只是一般都是使用Ｔｒｙ　ｃａｔｃｈ就夠了

```
Try
{
        正常情況下的程式 ( 也就是想要測試的程式碼
}
Catch(   例外物件變數宣告   )
{
        發生異常時   要執行的片段
}
Finally   //可有可無！！
{
        執行完上面 2 個區塊後   要執行的內容
}
```

# Ｔｒｙ　ｃａｔｃｈ的除錯

我們將上一個問題重新翻新一下

```
int a = scanner.nextInt();
int b = scanner.nextInt();
try
{
        System.out.println( a/b );
}
catch(ArithmeticException ex)
{
        System.out.println("Exception: 除數不可以是 0 ");
}
System.out.println("程式繼續執行中...");
```

輸入：
10 0

輸出：
Exception: 除數不可以是 0
程式繼續執行中...

# Ｔｒｙ　ｃａｔｃｈ的除錯

我們來修改一下catch裡面的內容　瞧瞧會發生什麼事

```
int a = scanner.nextInt();
int b = scanner.nextInt();
try
{
        System.out.println( a/b );
}
catch(ArithmeticException ex)
{
        throw new ArithmeticException("除數不可以是 0 ");
}
System.out.println("程式繼續執行中...");
```

輸入：
10 0

輸出：
Exception in thread "main" java.lang.ArithmeticException: 除數不可以是 0
at test.Main.main(Main.java:20)

# Ｔｒｙ　ｃａｔｃｈ的除錯

直接顯示異常的內容

```
int a = scanner.nextInt();
int b = scanner.nextInt();
try
{
        System.out.println( a/b );
}
catch(ArithmeticException ex)
{
        System.out.println(ex);
}
System.out.println("程式繼續執行中...");
```

輸入：
10 0

輸出：
java.lang.ArithmeticException: / by zero
程式繼續執行中...

# Ｔｒｙ　ｃａｔｃｈ的除錯

直接拋出異常時　可以直接使用　throw　new

throw new 例外物件(內容);

接下來還有很多例外物件

每個package都有專屬的例外物件可以使用（拉到最下面可以看到Exception物件）

可以去查看ＡＰＩ　上面都會寫出　使用什麼method時

遇到什麼情況會發生異常

這裡就只舉例這個例子了　其他就給大家自己去挖吧～

如：最常發生的例外　ArraysIndexOutOfBoundException

# 多重捕抓（multi catch）

Ｊａｖａ７新增了一個功能叫做多重捕抓

可以一次宣告多個類別物件　使用運算元　ＯＲ（｜）

原：

```
……
catch(例外１　ex)
{　…..　}
catch(例外２　ex)
{　…..　}
catch(例外３　ex)
{　…..　}
……
```

Ｊａｖａ７後：

```
…
catch(　例外１　|　例外２　|　例外３　……　)
{　…..　}
```

# throws

用來告知類別可能發生的例外

public static void main(String[] args) throws ArrayIndexOutOfBoundsException
{……}

解釋：這個程式可能會發生超出陣列範圍的例外

可以同時接多個例外　用　,　來接

public static void main(String[] args) throws ArrayIndexOutOfBoundsException , Illega
{
　　……
}

# CH10 I/O

# 學習 java.util.Scanner　java.io.*

Ｊａｖａ早期的輸入輸出是以ｉｏ來進行的

在Ｊａｖａ５以後　Ｓｃａｎｎｅｒ較易使用者輸入輸出

於是漸漸地取代ｉｏ　但是有些地方Ｓｃａｎｎｅｒ卻有些缺失

Ｉｏ是以行來進行動作的

但是Ｓｃａｎｎｅｒ主要是以空格做區隔

# Scanner

Scanner大家應該都很熟了　隨時都能在一般java課本上看到

nextInt()　nextDouble()　next()　這些method被稱為符記讀取method

都它讀取分隔字元分開的符記　一般情況下　分隔字元就是空白鍵

如果輸入的型態不是期望的型態　就會拋出InputMismatchException

其中next()和nextLine()雖然都是讀取字串　但是後者是行分隔符號做結尾

但是請注意啦！！　不同的作業系統　分隔符號是不同的

使用 Java 跨平台coding或是考試的時候需要注意一樣~

| | |
|---|---|
| Windows | \r\n |
| Unix ( Linux ) | \n |

# nextLine()的超自然現象

請先試試看這個

```
int n = scanner.nextInt();
String s = scanner.next();
System.out.printf("n=%d    s=%s",n,s);
```

請再自己執行一下下面的程式　看會發生什麼事情？

```
int n = scanner.nextInt();
String s = scanner.nextLine();
System.out.printf("n=%d    s=%s",n,s);
```

試了之後你一定會發現　s明明就是輸入數字　竟然輸出的是一個空白

空白哪來的？怎麼剛剛沒有現在有？　而且明明只有按1下enter　竟然直接跳過了

超　自　然！！

# nextLine()的超自然現象

因為nextInt()是讀到分隔字元才結束的　在這裡就是enter

nextLine()於是又讀到了　於是停止　所以裡面是空的

可是有些題目就是一定需要在後面來一個nextLine()押

那要怎麼解決？　看下面：

```
int n = scanner.nextInt();
String s;
while( (s = scanner.nextLine() ).equals("") );
```

利用迴圈一直跑　直到讀取到資料後　才跳出迴圈

早期的java輸入輸出是沒這個問題的　但是Ｊａｖａ５以後的方法

好用卻有些需要注意的地方～

# I / O

在早期的Ｊａｖａ中　是以ＢｕｆｆｅｒｅｄＲｅａｄｅｒ來讀取

每次讀取都是一行　也可以解決Scanner的nextLine()讀取問題

只是不能分段取元素　只能逐行輸入　變成了一個較麻煩的缺點

**BufferedReader  br = new BufferedReader(new InputStreamReader(System.in));**

上面的縮寫　分解之後：

```
InputStreamReader read = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(read);
```

# I / O

再來～我們來介紹基本數值要如何輸入　記得加上　**throws IOException**

```
public static void main(String[] args) throws IOException
{
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    int a = Integer.parseInt(br.readLine());
    double b = Double.parseDouble(br.readLine());
    String s = br.readLine();
    System.out.println(a);
    System.out.println(b);
    System.out.println(s);
}
```

輸入：

30
20
Just  test

輸出：

30
20
Just  test

# I / O　分割

但是如果要做題目的話　一行可能不只輸入 1 個數值　可能 2 個或更多

之前介紹過字串有split的method可以分割　讓我們來看一下這個例子：

```
public static void main(String[] args) throws IOException
{
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String s = br.readLine();
    String[] array = s.split(" ");
    System.out.println(Arrays.toString(array));
}
```

輸入：
42 62 28

輸出：
[42, 62, 28]

# I / O　實現Scanner的hasNext()

我們可以像模擬C的ＥＯＦ　來達成相同的方法：

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 String s ;
 while( (s = br.readLine())!=null )
 {
        ..................
 }
```

# I / O　讀取數值

有些題目是　每行有 2 個不同的形態　　如：字串和整數

可以這樣處理：

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String s ;
while( (s = br.readLine())!=null )
{
        String[] array = s.split(" ");
        String a = array[0];
        int   b = Integer.parseInt(array[1]);
        System.out.println(a+" "+b);
}
```

輸入：

test 111
End 213

輸出：

test 111
End 213

# I / O　StringTokenizer~（ split ）

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String s ;
while( (s = br.readLine())!=null )
{
	StringTokenizer st = new StringTokenizer(s);
	System.out.println("每行有多少數值："+st.countTokens());

	int[] a = new int[st.countTokens()];
	for(int i=0;i<a.length;i++)
	{
		a[i] = Integer.parseInt(st.nextToken());
		// a[i] = Integer.parseInt((String) st.nextElement()); // 與上一行一樣
	}
	System.out.println(Arrays.toString(a));
}
```

輸入：

2 3 4 5

輸出：

每行有多少數值：4

[2, 3, 4, 5]

**StringTokenizer比split的速度還快**

**位在import java.util.StringTokenizer;**

# I / O　StringTokenizer~（split）

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 String s ;
 while( (s = br.readLine())!=null )
 {
       StringTokenizer st = new StringTokenizer(s);
       System.out.println("每行有多少數值："+st.countTokens());
       System.out.println(st.hasMoreElements());
       System.out.println(st.nextElement());
       System.out.println(st.nextToken());
       System.out.println(st.nextElement());
       System.out.println(st.hasMoreElements());
 }
```

輸入：

2 3 4

輸出：

每行有多少數值：3
true
2
3
4
false

**因為boxing　所以上面其中 2 個method**

**是可以視為相同的用法**

# I / O　模仿next’s method

```
class Reader
{
    static BufferedReader reader;
    static StringTokenizer tokenizer;
    static void init(InputStream input)
    {
        reader = new BufferedReader( new InputStreamReader(input) );
        tokenizer = new StringTokenizer("");
    }
    static String next() throws IOException
    {
        while ( ! tokenizer.hasMoreTokens() )
        {
            tokenizer = new StringTokenizer( reader.readLine() );
        }
        return tokenizer.nextToken();
    }
    static int nextInt() throws IOException  {    return Integer.parseInt( next() );          }
    static double nextDouble() throws IOException  {  return Double.parseDouble( next() );  }
}
```

**scanner有nextInt()　這種方便的用法**

**不用像 B R 需要慢慢分解**

**如果要用的話　必須自行設計了**

**這是別人設計的程式碼　放上來給大家參考**

# I/O 的演化

之後 有人習慣撰寫成下面 用法跟Scanner一模一樣

而且對某些情況來說 輸入的速度上有些微的提升唷

```
Scanner scanner =
            new Scanner(new BufferedReader(new InputStreamReader(System.in)));
int n = scanner.nextInt();
System.out.println(n);
```

Ｊａｖａ６以後有ｃｏｎｓｏｌｅ類別可以使用

這裡就不介紹了

# I / O　模擬hasNext

BufferedReader沒有hasNext　不過我們可以這麼做

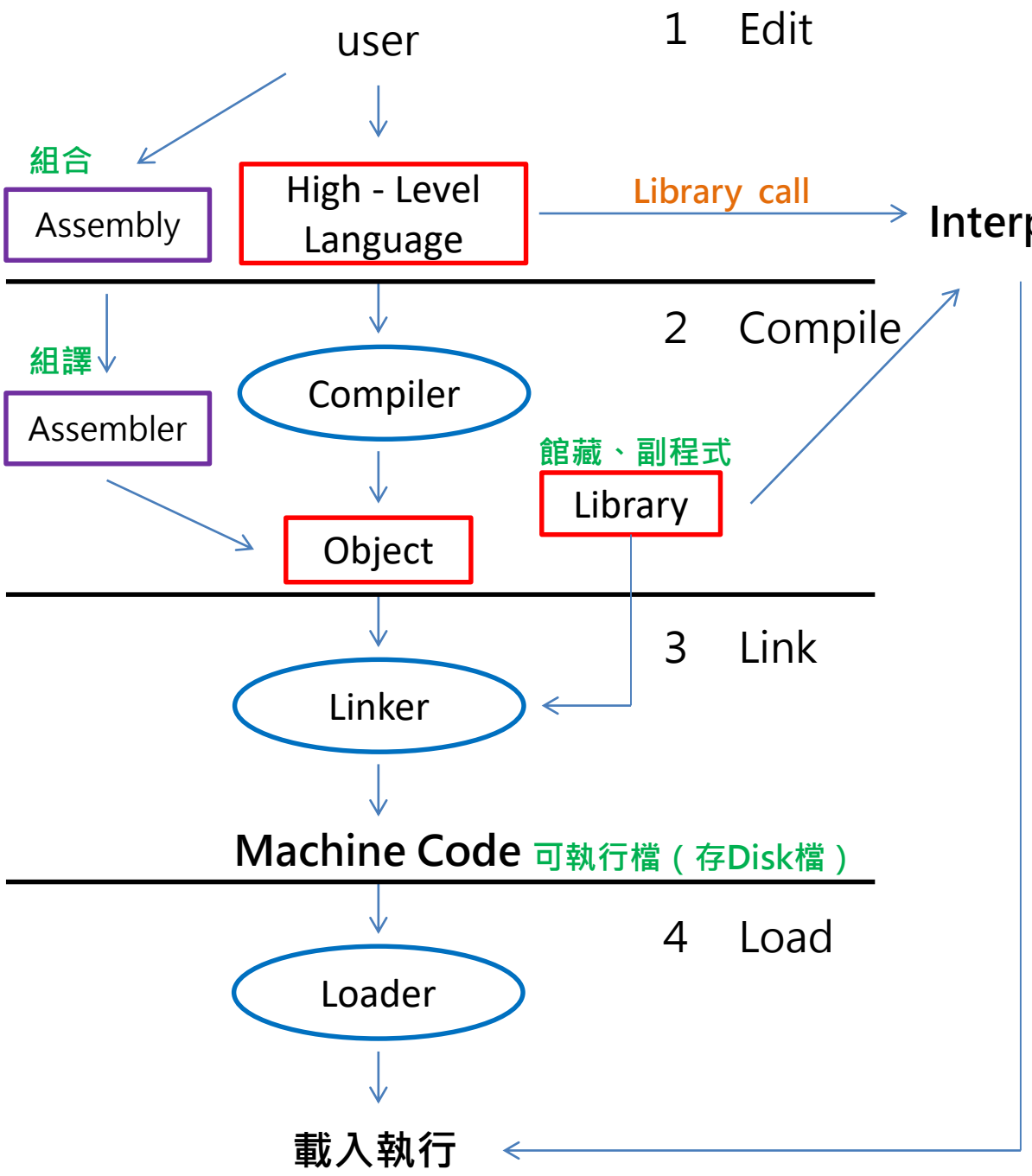
```
public static void main(String[] args) throws IOException
{
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String  s;
    while(( s = br.readLine() ) != null )
    {
        ..............
     }
}
```

# 程式的執行

這裡要介紹程式是如何從使用者到執行

為什麼同樣是程式碼　同樣的演算法　在不同的語言上　時間會是不同的？

# C語言的執行



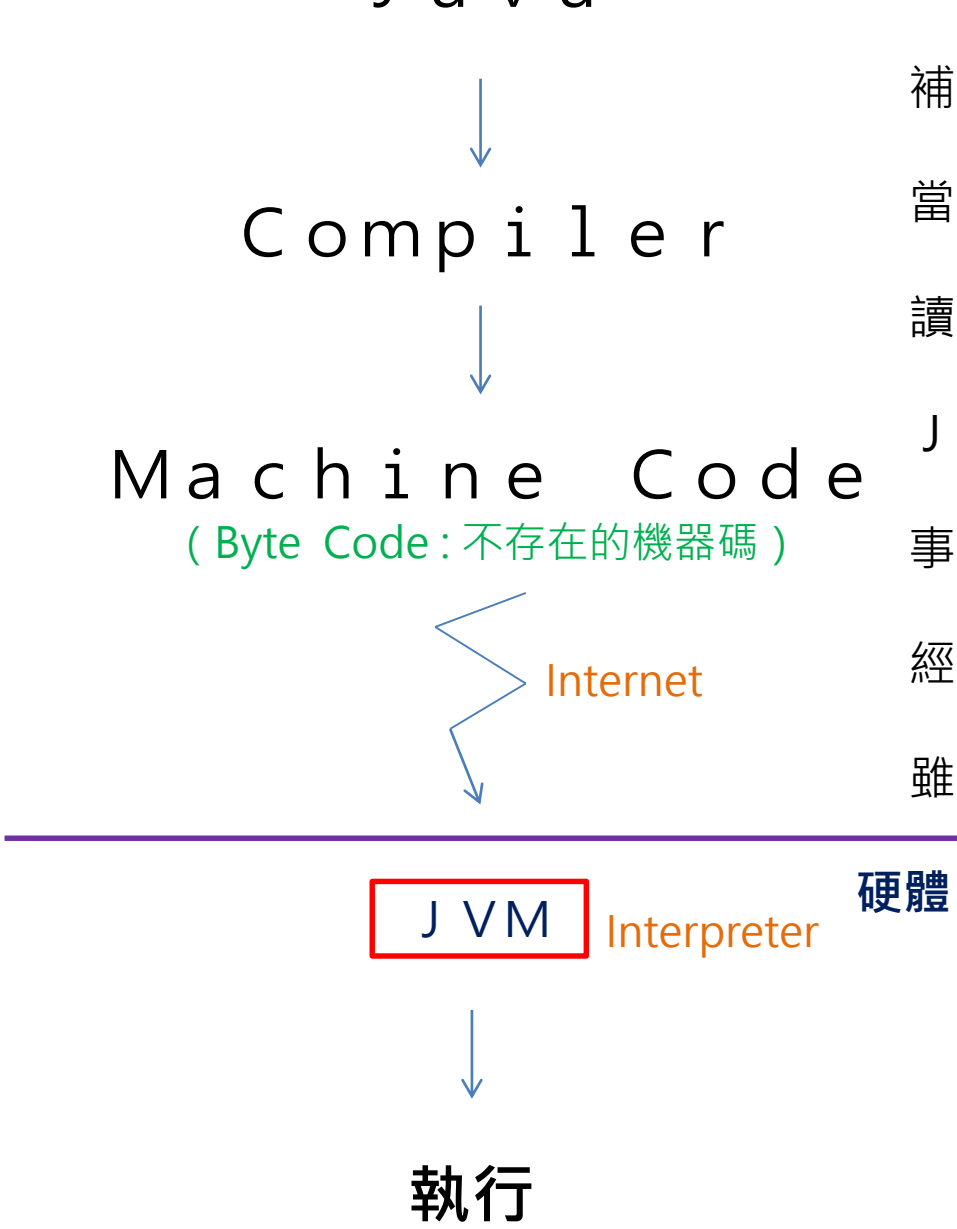

補充：

目前只有Ｂａｓｉｃ有直譯

Ｊａｖａ（跨平台）的ＶＭ

正是以直譯完成的！！

# Ｊａｖａ的執行




補充：

當你讀Ｊａｖａ的書籍時

讀到ＩＯ單元　你會常常看到一些作者在強調

Ｊａｖａ在input時　是讀取Byte

事實上對Ｊａｖａ來說　那正是一種機器碼

經過Java Virtual Machine直譯後執行

雖然有跨平台的優勢　執行速度慢卻是缺點

關於Ｉ／Ｏ時間的參考資料：

http://www.cpe.ku.ac.th/~jim/java-io.html

Ｃ與Ｊａｖａ的時間性能比較

# 參考題目

**10008 - What‘s Cryptanalysis?　難度：一星 ~ 二星**

http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_proble

**提示：注意nextLine()的問題**

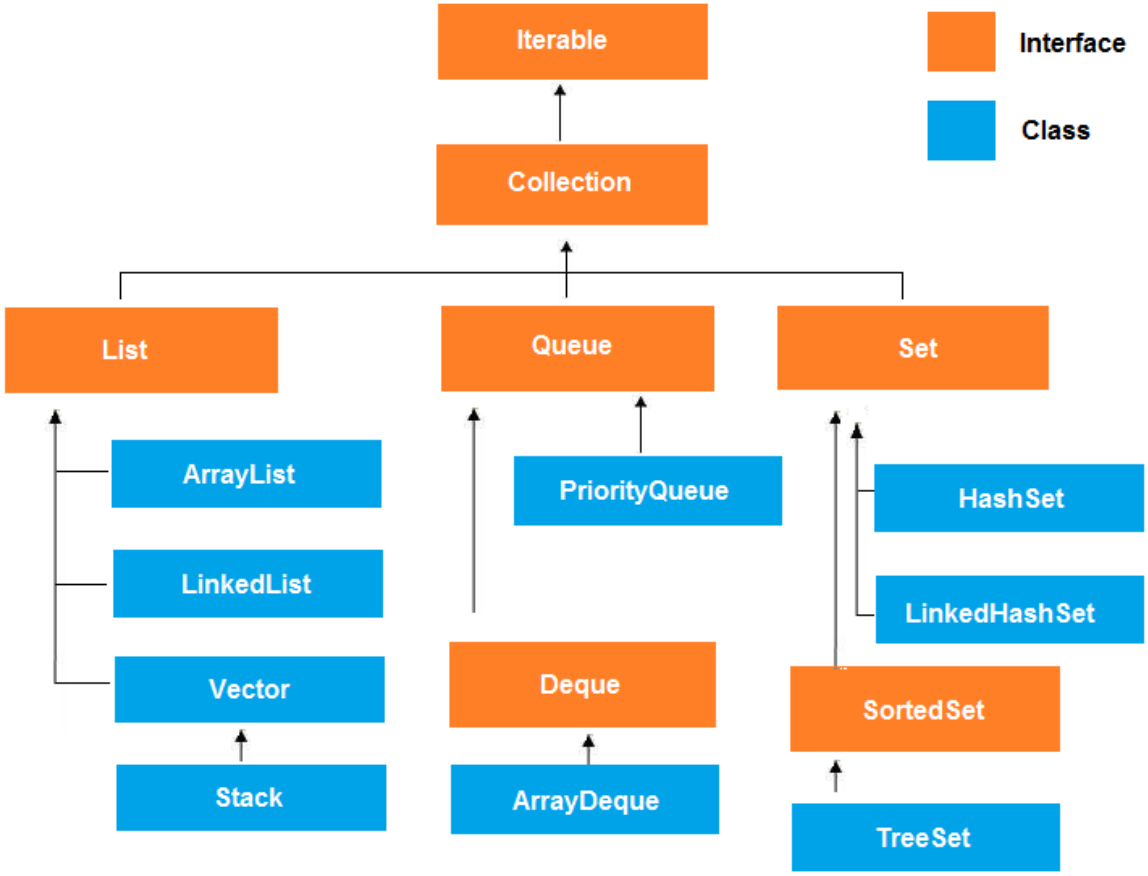
# CH11 集合與圖

# 學習Java Collection Framework與Map

沒學過集合別說你學過資料結構～　至少基本的堆疊或是串列這些都會

不過除了這些外　集合裡面還有很多好用的方法

下面只呈現基本架構和特性　剩下的就給大家自己摸索了



# 向量（vector<E>）

向量是很多實用類別的父類別　　如堆疊的父類別就是向量

所以使用起來會發現堆疊跟向量使用上　沒有什麼 2 樣的感覺

```
Vector<Integer> vc = new Vector<Integer>();
vc.add(100);
vc.add(10);
vc.add(1000);
System.out.println(vc);
```

也是可以這樣宣告：

```
Vector<Integer> vc = new Stack<Integer>();
```

但是Ｊａｖａ的向量表現上不如C的向量好用　　而且method名稱較長

下幾代的子類別比較好用　如：Stack

下幾代的類別並不是向量的改進　而比較像是獨立的幾個子類別　改進只是個錯覺

# 鏈結串列（ＬｉｎｋｅｄＬｉｓｔ<E>）

一個接著一個串　很單純的特性

```
LinkedList<Integer> List = new LinkedList<Integer>();
List.add(50);
List.add(5);
List.add(100);
System.out.println(List);
```

輸出：

[50, 5, 100]

# ＡｒｒａｙＬｉｓｔ<E>

ＡｒｒａｙＬｉｓｔ同時具有陣列和串列的特性　和陣列的特性最貼切

一般使用陣列的時候　必須要先給他ｉｎｉｔｉａｌ　ｃａｐａｃｉｔｙ

如：　int[]  a = new  int[  INNITIAL_CAPACITY  ];

如果不確定要設的空間大小時　可以考慮使用ＡｒｒａｙＬｉｓｔ

```
ArrayList<Integer> aList = new ArrayList<Integer>();
aList.add(30);
aList.add(68);
aList.add(24);
aList.add(70);
System.out.println(aList);
```

輸出：

[30, 68, 24, 70]

# 堆疊（Ｓｔａｃｋ<E>）

像堆硬幣一樣　擁有ＦＩＬＯ或是ＬＩＦＯ的特性

```
Stack<Integer> stack = new Stack<Integer>();
stack.push(10);
stack.push(30);
stack.push(20);
System.out.println(stack);
stack.pop();
System.out.println(stack);
```

輸出：

```
[10, 30, 20]
[10, 30]
```

# 佇列（Ｑｕｅｕｅ<E>）

就像買東西排隊一樣　擁有ＦＩＦＯ或是ＬＩＬＯ的特性

但是在Ｊａｖａ中　Ｑｕｅｕｅ並沒有被實作　只是interface

所以需要用子類別來接在這個接口上

```
Queue<Integer> queue = new LinkedList<Integer>();
queue.offer(10);
queue.offer(30);
queue.offer(20);
System.out.println(queue);
queue.poll();
System.out.println(queue);
```

輸出：

```
[10, 30, 20]
[30, 20]
```

# ＰｒｉｏｒｉｔｙＱｕｅｕｅ<E>

佇列大家都知道擁有ＦＩＦＯ的特性

而佇列的子類別有個叫做優先權佇列的物件　有時候意外的好用

好用的地方在於　他並不需要遵守ＦＩＦＯ的特性　poll的時候　從小到大

```
PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
pq.add(10);
pq.add(35);
pq.add(80);
pq.add(5);
System.out.println(pq);     pq.poll();
System.out.println(pq);     pq.poll();
System.out.println(pq);     pq.poll();
System.out.println(pq);
```

輸出：

```
[5, 10, 80, 35]
[10, 35, 80]
[35, 80]
[80]
```

# Ｄｅｑｕｅ<E>（ｄｏｕｂｌｅ　ｅｎｄｅｄ）

Ｄｅｑｕｅ是雙向佇列　一般而言　排隊是必須從排尾開始排的

但是Ｄｅｑｕｅ可以從排頭或是排尾進去或是出來

但是ｊａｖａ的Ｄｅｑｕｅ是個interface　要使用的話　要用Ｄｅｑｕｅ的子類別宣告

舉例：

```
Deque<Integer> dq = new ArrayDeque<Integer> ();
```

# ＡｒｒａｙＤｅｑｕｅ<E>

使用Ｄｅｑｕｅ的子類別　ＡｒｒａｙＤｅｑｕｅ

```
Deque<Integer> dq = new ArrayDeque<Integer> ();

dq.add(50);
dq.add(68);
dq.addFirst(52);
dq.addLast(64);

System.out.println(dq);
```

輸出：

[52, 50, 68, 64]

# Ｈａｓｈｔａｂｌｅ<Ｋ,Ｖ>

Ｈａｓｈｔａｂｌｅ是雜湊表　擁有較高效率　不重複相同的key

但是有個缺點就是　一旦執行　占用的記憶體空間太大了

但是子類別　ＨａｓｈＳｅｔ、ＨａｓｈＭａｐ　解決了這個問題

```
Hashtable<String,Integer> table = new Hashtable<String,Integer>();
table.put("sally",1);
table.put("sally",2);
table.put("sandy",2);
table.put("danny",3);
System.out.println(table);
System.out.println(table.get(1));
System.out.println(table.get("sally"));
```

輸出：

```
{danny=3, sandy=2, sally=2}
null
2
```

# Set<E>

Set也是interface　如果要使用　也必須和子類別做接口宣告

他主要的功能是：資料不重複

宣告方式舉例：

```
Set<String> set = new HashSet<String>();
```

# Ｈａｓｈｓｅｔ＜Ｅ＞

ＨａｓｈＳｅｔ＜Ｅ＞對有些題目真的很好用！！　而且速度很理想

他**不會重複**已經有的資料　但是他不會跟你的資料做輸入排序　是雜湊的！！

```
HashSet<Integer> hs = new HashSet<Integer>();
hs.add(2);
hs.add(1);
hs.add(-3);
hs.add(1);
System.out.println(hs);
```

輸出：

[1, 2, -3]

# Map<K,V>

Map也是interface　如果要使用　也必須和子類別做接口宣告

他主要的功能是：資料包含索引值（key）與資料（value）

專業的術語是　　鍵　與　映射值

宣告方式舉例：

```
Map< Integer , String > map = new HashMap< Integer , String >();
```

# ＨａｓｈＭａｐ<Ｋ,Ｖ>

Ｍａｐ是Ｃ＋＋使用者很喜歡用的一個類別

不過Ｊａｖａ中的Ｍａｐ中　子類別的ＨａｓｈＭａｐ蠻好用的

可以利用索引值（ｋｅｙ）　來找資料　但是這個也是雜湊的

```
HashMap< Integer , String > map = new HashMap< Integer , String >();
map.put(1, "danny");
map.put(2, "frank");
map.put(3, "lucy");
map.put(4, "funk");

System.out.println(map);
System.out.println(map.get(3));
```

輸出：

```
{1=danny, 2=frank, 3=lucy, 4=funk}
lucy
```

# LinkedHashMap<K,V>

Public class LinkedHashMap<K,V> extends HashMap<K,V> implements Map<K,V>

他的用法跟HashMap大同小異　但是他是有按順序存放的！！

```
LinkedHashMap< Integer , String > map = new LinkedHashMap< Integer , String >();
map.put(3, "lucy");
map.put(1, "danny");
map.put(4, "funk");
map.put(2, "frank");

System.out.println(map);
System.out.println(map.get(3));
```

輸出：

```
{3=lucy, 1=danny, 4=funk, 2=frank}
lucy
```

**另外也有LinkedHashSet　用法和上面大同小異 < E >　也符合原先的不重複特性**

# ＴｒｅｅＳｅｔ<Ｅ>

ＴｒｅｅＳｅｔ是以紅黑樹的方式來做儲存資料

但是紅黑樹是樹裡面最難的樹　很難輕易去理解他　也是研究所考試的大殺手

所以這裡就先不討論紅黑樹的地方了

下一張會用到　ＴｒｅｅＳｅｔ　用在別的接口上　２個方法都是一樣的

最主要的原因是SortedSet繼承TreeSet　所以使用上來差異不大

```
TreeSet<Integer> set = new TreeSet<Integer>();
set.add(12);
set.add(3);
set.add(50);
set.add(36);
System.out.println(set);
```

輸出：

[3, 12, 36, 50]

# SortedSet<E>

SortedSet有個很大的好處就是放進去的資料都會經過排序

省去了很多的麻煩　但事實上直接使用TreeSet即可

```
//SortedSet<Integer> set = new SortedSet<Integer>(); //錯誤

SortedSet<Integer> set = new TreeSet<Integer>();
set.add(12);
set.add(3);
set.add(50);
set.add(36);
System.out.println(set);
```

輸出：

[3, 12, 36, 50]

# ＴｒｅｅＭａｐ<Ｋ,Ｖ>

ＴｒｅｅＭａｐ<Ｋ,Ｖ>和Ｔｒｅｅｓｅｔ <Ｅ> 一樣以紅黑樹的方式排序

與Ｔｒｅｅｓｅｔ差在ＴｒｅｅＭａｐ是以key當作排序的依據

```
TreeMap<Integer,String> map = new TreeMap<Integer,String>();
map.put(4, "Funk");
map.put(1, "Danny");
map.put(3, "Lucy");
map.put(2, "Frank");

System.out.println(map);
System.out.println(map.get(3));
```

輸出：

```
{1=Danny, 2=Frank, 3=Lucy, 4=Funk}
Lucy
```

# ＳｏｒｔｅｄＭａｐ<Ｋ,Ｖ>

ＴｒｅｅＭａｐ 幾乎擁有ＳｏｒｔｅｄＭａｐ的特性

所以也可以直接用ＴｒｅｅＭａｐ宣告就好　不需要用到ＳｏｒｔｅｄＭａｐ

<索引值,資料>　以泛型定義各種型態

```
SortedMap<Integer,String> map = new TreeMap<Integer,String>();
map.put(4, "Funk");
map.put(1, "Danny");
map.put(3, "Lucy");
map.put(2, "Frank");

System.out.println(map);
System.out.println(map.get(3));
```

輸出：

```
{1=Danny, 2=Frank, 3=Lucy, 4=Funk}
Lucy
```

# 使用Ｃｏｌｌｅｃｔｉｏｎｓ

只要是Ｊａｖａ　Ｃｏｌｌｅｃｔｉｏｎ　Ｆｒａｍｅｗｏｒｋ上的類別

都可以使用父類別Ｃｏｌｌｅｃｔｉｏｎｓ的method　下面舉例：

```
ArrayList<Integer> aList = new ArrayList<Integer>();
aList.add(65);
aList.add(47);
aList.add(82);
aList.add(26);
System.out.println(aList);
Collections.sort(aList);
System.out.println(“排序後:”+aList);
Collections.shuffle(aList);
System.out.println(“亂排後:”+aList);
```

輸出：

```
[65, 47, 82, 26]
排序後:[26, 47, 65, 82]
亂排後:[65, 26, 47, 82]
```

# Ｍａｐ與Ｃｏｌｌｅｃｔｉｏｎｓ

你看了上面的　Ｓｅｔ、Ｌｉｓｔ、Ｍａｐ

一定會問一件事情　為什麼Ｓｅｔ、Ｌｉｓｔ有繼承集合　但是Ｍａｐ沒有？

原因非常單純

因為Ｓｅｔ、Ｌｉｓｔ的泛型結構是＜Ｅ＞

但是Ｍａｐ的泛型結構是　　＜Ｋ,Ｖ＞

就算Ｍａｐ繼承了集合　結構不一樣也是不能用的　有需要的話要另外實作

但是大部分的功能Ｍａｐ已經有了　不太需要擔心

# Ｍａｐ與Ｃｏｌｌｅｃｔｉｏｎｓ

接下來２張圖是Ｍａｐ與Ｃｏｌｌｅｃｔｉｏｎｓ的繼承圖

練習完上面幾個類別後

再來看繼承圖　有沒有更了解呢？

你會發現集合其實並沒有很難

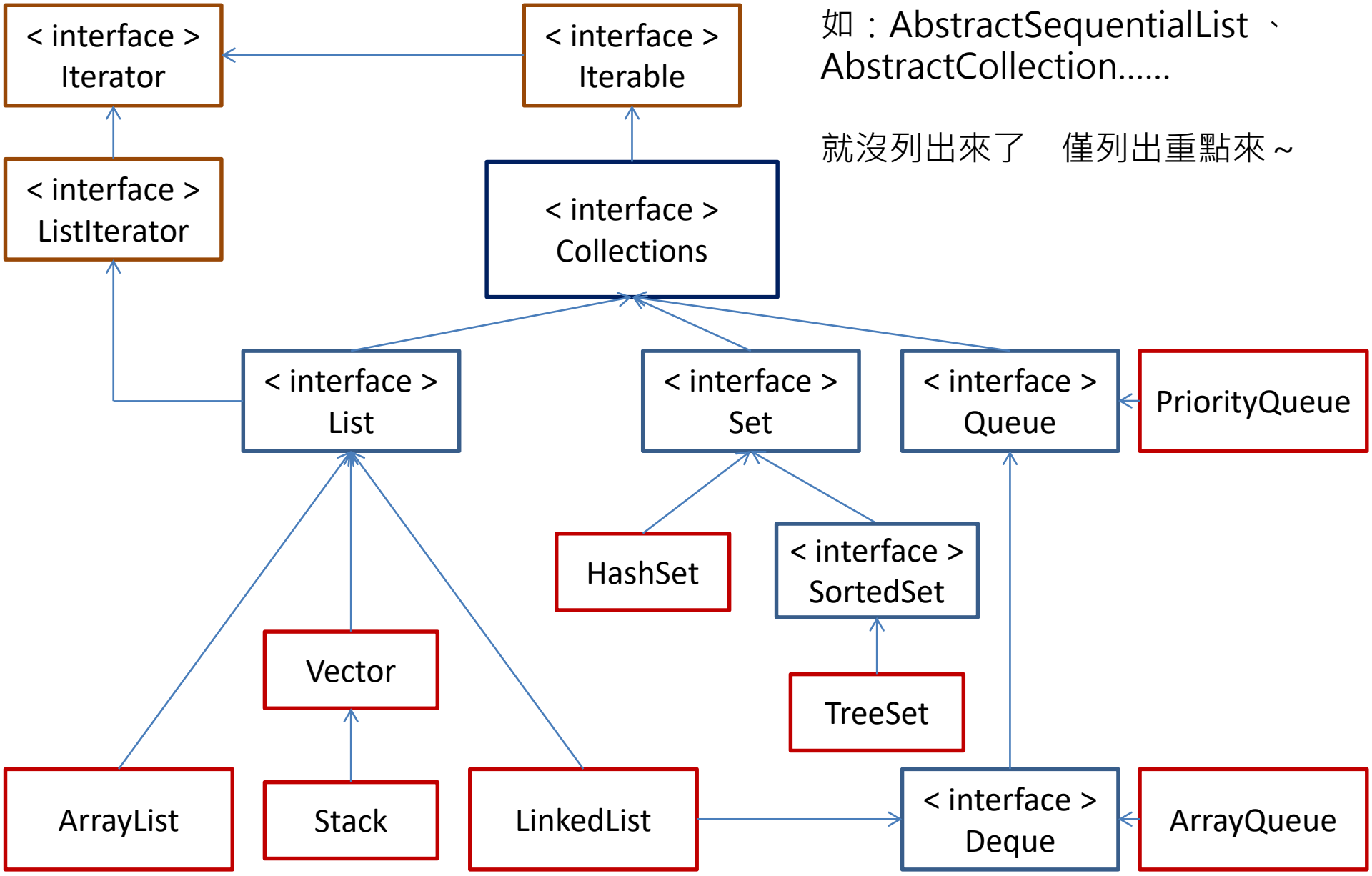
只要多熟悉就會了~

# Collections

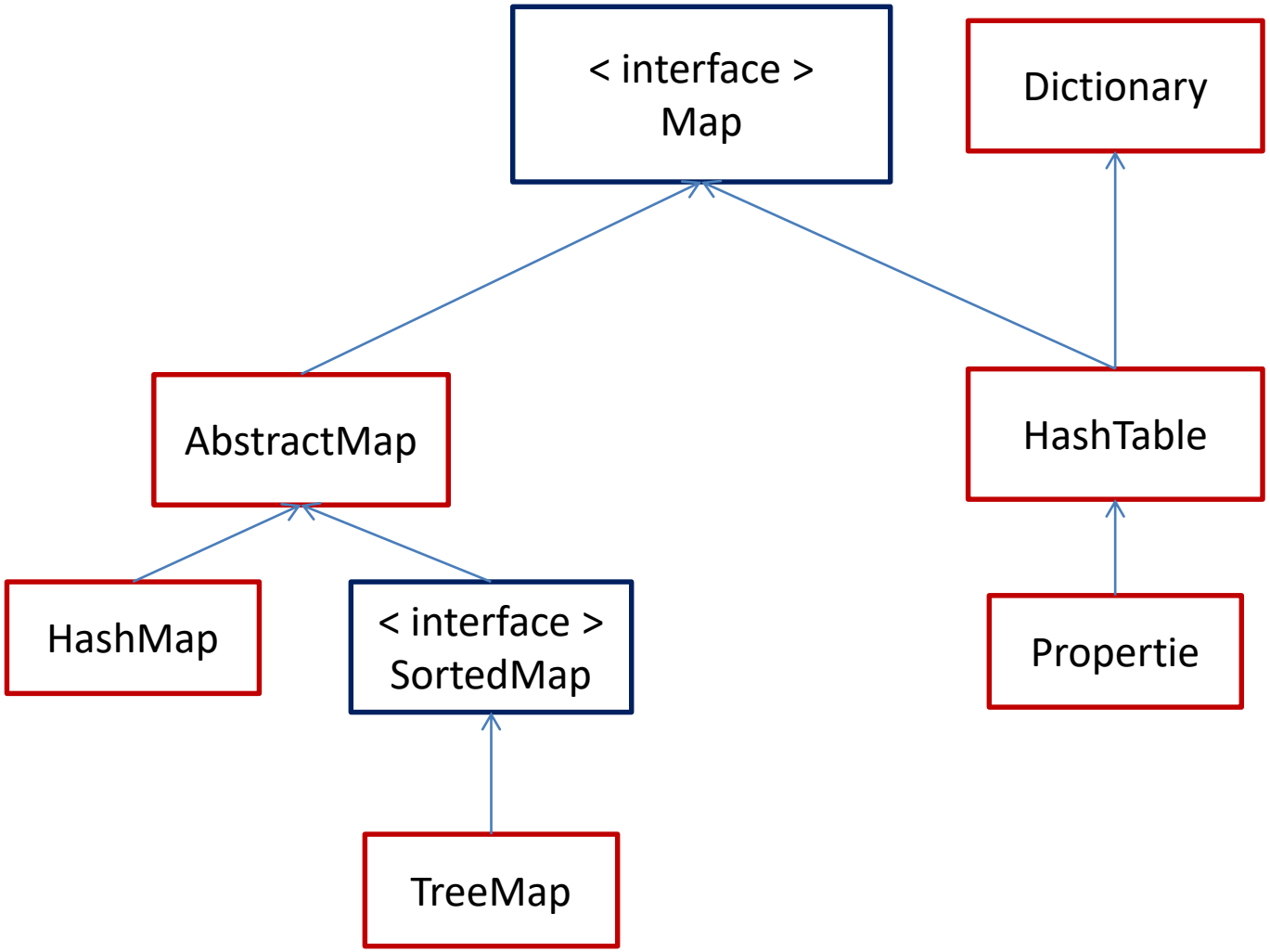
有些比較細節的interface和類別

如：AbstractSequentialList 、AbstractCollection……

就沒列出來了　僅列出重點來～



# Map


< interface >
Map
Dictionary
AbstractMap
HashTable
HashMap
< interface >
SortedMap
Propertie
TreeMap

# 參考題目

10107 - What is the Median?　難度：一星

提示：(　Array　or　ArrayList　)　+　遞推演算法　可輕鬆解決

http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_proble

10420 - List of Conquests　　難度：一星～二星

提示：Map < String , Integer > ~ < K , V >　較容易解

http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_proble

10954 - Add All　　難度：二星

提示：使用 PriorityQueue 可輕鬆解答

http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_proble

# Ｍａｐ的進階用法

不管是不是Ｍａｐ　都可以使用泛型讓程式做出很多的變化

下面舉Ｍａｐ搭配大家常用的Ａｒｒａｙ Ｌｉｓｔ來實現單鍵多空間

```
TreeMap<String,ArrayList<Integer>> map
                                        = new TreeMap<String,ArrayList<Integer>>();

   map.put("Sam",new ArrayList<Integer>());
   map.put("Coco",new ArrayList<Integer>());
   map.get("Sam").add(68);
   map.get("Sam").add(45);
   map.get("Coco").add(81);
   map.get("Coco").add(99);
   map.get("Coco").add(12);
   System.out.println(map.toString());
   System.out.println("Sam的數值有:"+map.get("Sam").toString());
   System.out.println("Coco的數值有:"+map.get("Coco").toString());
```

輸出：

```
{Coco=[81, 99, 12], Sam=[68, 45]}
Sam的數值有:[68, 45]
Coco的數值有:[81, 99, 12]
```

# Ｍａｐ的進階用法

程式碼繼續延續上一章　這張搭配集合的method讓程式更完美的呈現和整理

```
Collections.reverse(map.get("Coco"));
System.out.println("序列反轉過後："+map.get("Coco").toString());
System.out.println( “Coco最大值："+Collections.max(map.get("Coco")));
System.out.println( “Coco最小值："+Collections.min(map.get("Coco")));
Collections.swap(map.get("Coco"), 0, 1);
System.out.println("交換過後："+map.get("Coco").toString());
Collections.sort(map.get("Sam"));
Collections.sort(map.get("Coco"));
System.out.println("Sam的數值排序過後:"+map.get("Sam").toString());
System.out.println("Sam的數值排序過後:"+map.get("Coco").toString());
```

輸出：

序列反轉過後：[12, 99, 81]
Coco最大值：99
Coco最小值：12
交換過後：[99, 12, 81]
Sam的數值排序過後:[45, 68]
Sam的數值排序過後:[12, 81, 99]

# Ｍａｐ的進階用法

延續上一章的程式

讓指定的所有元素都顯示出來　可使用foreach

如果map的key是整數的話　也可以試著搭配巢狀迴圈　或是其他方法.....

```
for(Integer I : map.get("Coco"))
   {
            System.out.print(i+" ");
   }
```

輸出：

12 81 99

# Ｊａｖａ使用泛型

Ｊａｖａ還沒有泛型時　實作都需要把物件宣告成第一代Object的物件型態

所以在Ｊａｖａ在使用泛型的時候　你可以想像成它是Object

但是泛型算是一種多型嗎？　不是

Object是一種多型嗎？　　沒錯

多型針對的是函式，不同的型態進行相同的操作　泛型則是對於型態的多元化

早期是以物件的多型來達成類似Ｃ語言的泛型實作

但是Ｃ和Ｊａｖａ的泛型又是不同的２個ｃａｓｅ

Ｊａｖａ的泛型　只要是物件都可以接上去

Map< K , V >　　Map< Integer , String>

Ｃ語言的泛型則不太一樣
map< K , V >　　map< int , string >

# CH12 迭代器與泛型

# 回味foreach

之前談過這個可以用在陣列會是具有迭代性的物件

現在就來介紹　要如何用在具有迭代性的物件上

For ( 變數宣告：物件 )
{ ..... }

# 鏈結串列使用for

有時候　不需要用到迭代器也能輕易輕易顯示資料

```
LinkedList<Integer> List = new LinkedList<Integer>();

List.add(32);
List.add(18);
List.add(16);
List.add(62);

for( int i : List )
        System.out.println(i);
```

輸出：

```
32
18
16
62
```

# 鏈結串列使用while（Ｉｔｅｒａｔｏｒ）

```
LinkedList<Integer> List = new LinkedList<Integer>();

List.add(32);
List.add(18);
List.add(16);
List.add(62);

Iterator iter = List.iterator();

while(iter.hasNext())
{
        System.out.println(iter.next());
        //iter.remove();
}
```

輸出：

```
32
18
16
62
```

# 鏈結串列使用while（ＬｉｓｔＩｔｅｒａｔｏｒ）

```
LinkedList<Integer> List = new LinkedList<Integer>();

List.add(32);
List.add(18);
List.add(16);
List.add(62);

//Iterator iter = List.listIterator();                    //也可以這麼宣告
ListIterator iter = List.listIterator();
while(iter.hasNext())
{
        System.out.print(iter.next() + " ");
        iter.remove();
}
```

輸出：

32 18 16 62

# Ｉｔｅｒａｔｏｒ與ＬｉｓｔＩｔｅｒａｔｏｒ

２個使用上並沒有什麼太大的差別　但是要分的話

Ｉｔｅｒａｔｏｒ是ＬｉｓｔＩｔｅｒａｔｏｒ的父類別

Ｉｔｅｒａｔｏｒ對所有的集合物件都是可以共同使用的

ＬｉｓｔＩｔｅｒａｔｏｒ只有Ｌｉｓｔ才能使用

Ｉｔｅｒａｔｏｒ的method較少　但是ＬｉｓｔＩｔｅｒａｔｏｒ的method較多

可以從頭或從尾跑

# 鏈結串列使用while（ＬｉｓｔＩｔｅｒａｔｏｒ）

```
LinkedList<Integer> List = new LinkedList<Integer>();
List.add(32);
List.add(18);
List.add(16);
List.add(62);
ListIterator iter = List.listIterator();

while(iter.hasNext())
   {
      System.out.print(iter.next() + " ");
      //iter.remove();
   }

System.out.println();

while(iter.hasPrevious())
   {
      System.out.print(iter.previous() + " ");
      //iter.remove();
   }
```

從頭跑到尾

再從尾跑到頭

輸出：

32 18 16 62
62 16 18 32

# Ｍａｐ如何實現迭代？

Ｍａｐ並不是集合的成員　但是卻擁有相似的特性

如果要實現迭代的話　需要使用Ｍａｐ裡面的Entry

```
TreeMap<String,Integer> map = new TreeMap<String,Integer>();
map.put("7-11",85);
map.put("Familymart",90);
map.put("Hi-life",80);
map.put("OK",60);
map.put("Nikomart",90);

for(Map.Entry<String, Integer> i: map.entrySet())
    System.out.println(i.getKey() + " " + i.getValue());
```

輸出：

```
7-11 85
Familymart 90
Hi-life 80
Nikomart 90
OK 60
```

# 泛型實作

之前的單元介紹過泛型的使用方式　相信大家用過之後　都熟悉基本特性了

再來要介紹簡單的實作方式　我們以實作　swap()　來舉例

前提知識摘要：

因為Ｊａｖａ沒有指標　所以只能使用陣列的 Refrence 來進行交換的動作

１　Ｃａｌｌ　ｂｙ　Ｖａｌｕｅ　　只有傳值　並沒更改原始資料

２　Ｃａｌｌ　ｂｙ　Ｒｅｆｒｅｎｃｅ　　直接進去原始資料的位置動作

３　Ｃａｌｌ　ｂｙ　Ａｄｒｅｓｓ　　傳遞原始資料的位置（pointer）

注意！２和３很像　但是又不太一樣　因為現在的高階語言普及化

所以很多書都只寫１、２　　３已經慢慢被忽視了　但是意思卻不太一樣

事實上早期的語言只有１和３　　２算是後續衍生出來的新名詞

# 泛型實作

所以Ｊａｖａ要實作ｓｗａｐ　不能使用ｓｗａｐ（ｉｎｔ,ｉｎｔ）

必須使用ｓｗａｐ（陣列,索引值１,索引值２）

我們先不要管泛型該怎麼實作　我們先設計好基本的ｓｗａｐ程式

如下：

```
public static void swap( int[] array , int indexA , int indexB )
{
        int temp = array[indexA];
        array[indexA] = array[indexB];
        array[indexB] = temp;
}
```

# 泛型實作

我們設計完整數的ｓｗａｐ程式後　我們可能希望浮點數也能參與

或是範圍更大的長整數　又或者是字串陣列的資料互換

當你有這種想法時　就表示　你該考慮加入泛型了

所以可以改成這樣：

```
public static<T> void swap(  T[] array , int indexA  , int indexB  )
{
        T temp = array[indexA];
        array[indexA] = array[indexB];
        array[indexB] = temp;
}
```

# 泛型實作

測試的時候使用這個：

Integer[] a = new Integer[10];

不能使用：

Int[] a = new Int[10];

泛型是被定義為接收任何物件變數的一個接口

在實作時也可以強制轉換型態　如：

```
E[ ]  data;
data = ( E[ ] ) new Object[ capacity ];
```

在Ｊａｖａ中　陣列是一種變數　所以物件是可以拿來做為陣列的

如：

BigInteger[ ] a = new BigInteger[10];

# 泛型實作

注意程式邏輯

一般在實作泛型的時候　常常會犯下面的錯誤：

```
data = new E[ capacity ];                    //( X )
```

泛型非物件　他只是一種接口　不可new物件

下面的宣告方式是正確的：

```
data = ( E[ ] ) new Object[ capacity ];
```

# 小節

恭喜你已經把ＯＯＰ的精隨都吸收一遍了　也許還不是很熟

但是只要多練習　基本上１、２星的題目都有辦法做出來的

三星題則需要演算法或是高演才有辦法解決

之後的單元　介紹的跟程式沒什麼太大的關係　但是卻是不可不學的知識

對以後學演算法會有很大的幫助

這份電子書主要是介紹物件的使用　至於演算法就靠各位練習了~

# ＣＨ１３　樹

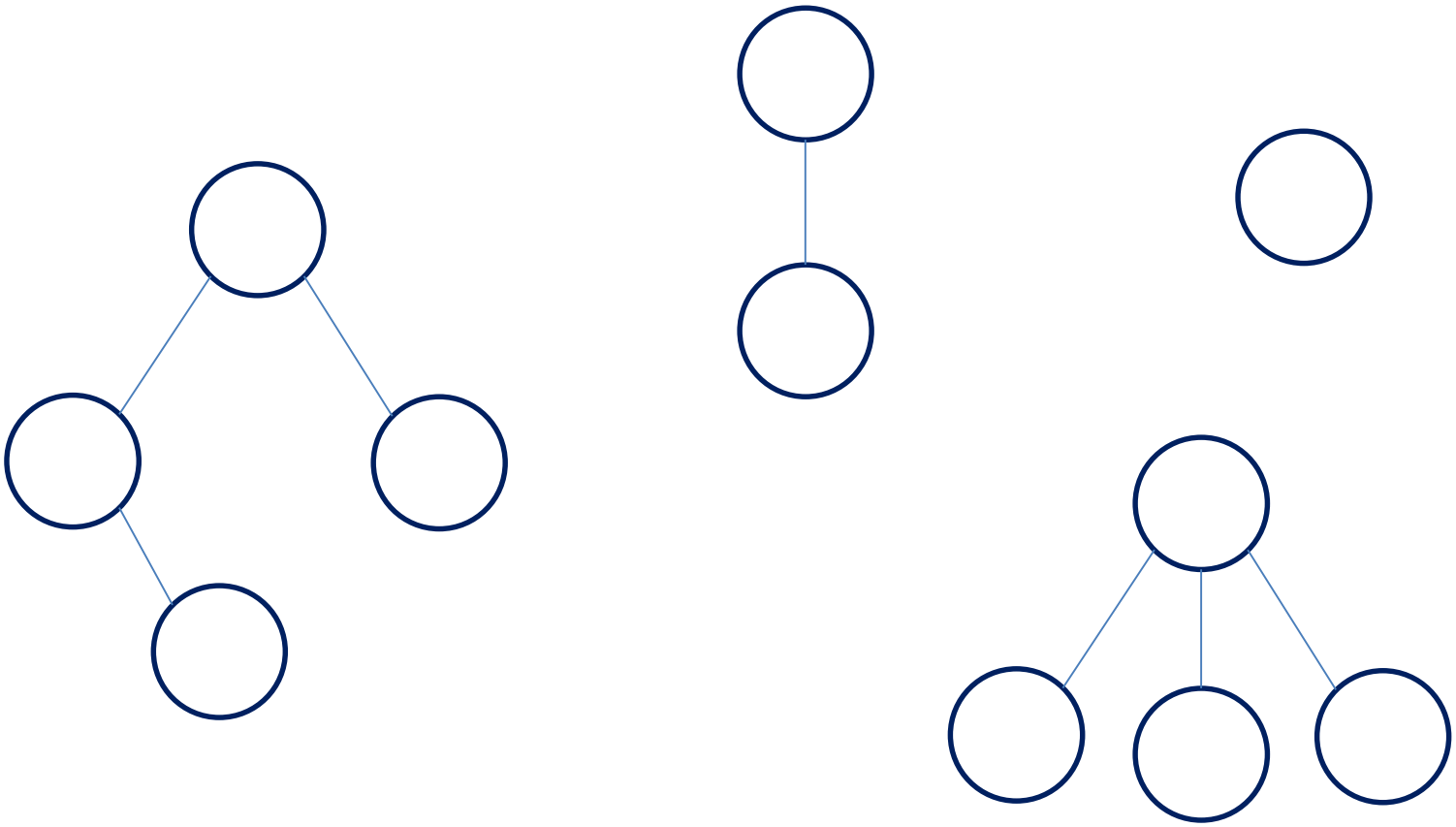
# 學習Ｔｒｅｅ

樹的定義很簡單　是一種擴散結構

任何由節點和分支構成的擴散結構都可以稱為樹

# 樹

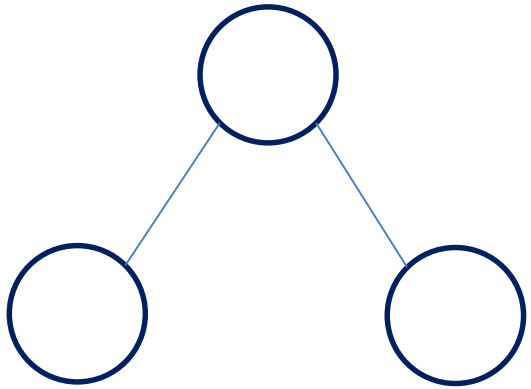
下面都是樹



# 二元樹

二元樹的每個節點最多有２個分支

有Ｎ個節點的話　則必有Ｎ＋１個ｎｕｌｌ　ｌｉｎｋ



# A　推薦書籍

# ＪＡＶＡ推薦書籍

下面介紹幾本我認為ＪＡＶＡ的經典書籍！！

因為市面上大部分都是Ｃ＋＋的參考書籍

所以ＪＡＶＡ的書可能會比較難找

於是下面就推薦給大家了

學校圖書館應該都會有才對　因為都是經典書籍

# ＪＡＶＡ聖經本 - 螞蟻書

Ｃ和ＪＡＶＡ的聖經本都是這本出名的螞蟻書　中文英文都有

因為封面以往的風格都是螞蟻做封面　所以大家都稱呼他螞蟻書

但是最新版似乎沒有螞蟻了　大家都在很熱烈地討論這件事 ( 不是吧...

裡面的內容從簡單到難都有　如果你要打基礎　可以參考這本 ~

你遇到的問題　上面應該都能得到解答

# ＪＡＶＡ７　技術手冊

**ＰＴＴ與ＪＡＶＡ程式設計討論區一致推薦**

這本是從進階到專家的ＪＡＶＡ書籍

我超級推薦這本書　寫的超讚！！

但是基本觀念只有打１個單元　如果你是第一次看到那些東西

剩下的對你來說都是震撼教育了　但是每個人都有第一次

有事沒事多翻多看幾眼　你會秒懂　這本也很容易懂

資料結構的學習重點都在上面

其實這本我覺得也很適合打基礎

用簡單的觀念去描述比較難的觀念　讚～

# ＪＡＶＡ７　初學指引

這本內容也很詳細　也適合基礎到進階的學習

很多常常被忽略的地方會提醒

也有介紹不少資料結構的基礎觀念

就算看不太懂　能翻完就翻完吧

隨便翻個幾遍　至少也有個印象　之後去看更容易了解

# ＪＡＶＡ７　基礎必修課

這本我只有看一次而已　不過這本講的方式很適合打基礎

也有講到一些常被忽略的地方　好書　推了~

# ＪＡＶＡ程式設計導論

這本寫得很詳細　而且又大又厚　也可以算是聖經了

內容也很詳細　解釋得很清楚　要從頭翻到尾應該不難

基礎到進階的觀念也都有　比較難的觀念會講得有點仔細

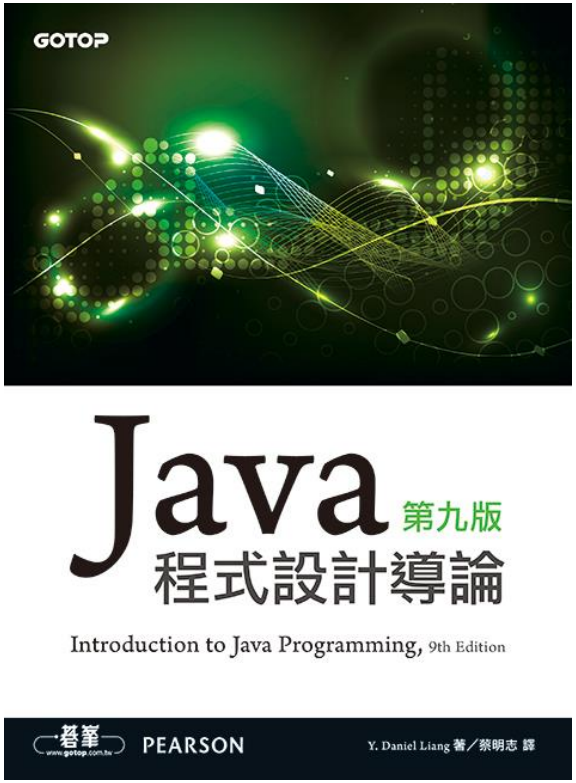
但是資料結構的部分比起其他課本　就比較沒有多談

但是卻很厚　就知道他基本觀念打得多仔細

這本主要是打地基　字串的部分也有講到StringBuilder

推薦給想衝向進階程式設計師的老朋友們～

（交通大學博士　蔡明志翻譯）



# Ivor Horton的ＪＡＶＡ初學指引

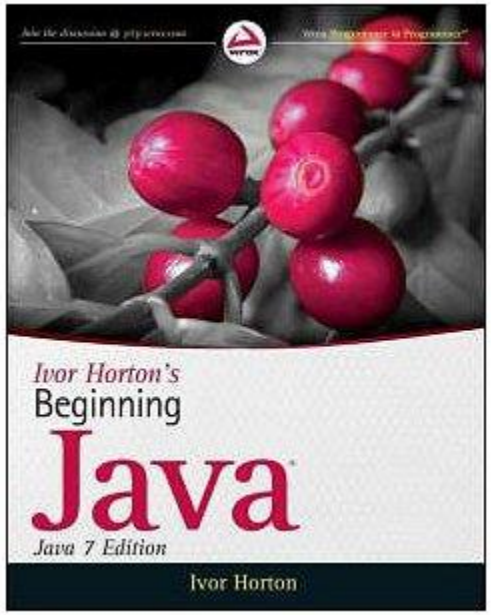
ＪＡＶＡ的程式設計書雖然不多　不過除了螞蟻書在全球很知名外

這本也是非常有名的！！　**Ivor Horton**是程式設計的掌權

以往他撰寫的Ｃ和ＪＡＶＡ書籍都有不錯的回應

有興趣的可以參考　只是**Ivor Horton**的書　目前只有Ｃ＋＋每一版在翻譯

以往都是蔡明志在翻譯　不知道以後會不會翻譯



# 最新ＪＡＶＡ７程式語言

這本書是施威銘研究室撰寫的

施威銘是台灣程式設計書籍很有名的一位作者

年代久遠的人　如果常讀程式設計的書籍　一定知道它

這本是很多老師推薦的書籍

老實說　我沒看過.....

會放上來推薦是因為很多老師和教授們大推

（逢甲大學、勤益科大、交通大學老教授推薦）

聽說是以前有和他在合作所以才推薦的...

（這不是重點...

# B　演算法學習資源

# 演算法筆記

這是臺灣師範大學的附屬網站

光這個網頁就夠自己學習演算法了

用簡單的觀念去解析演算法　還不錯　只可惜是用Ｃ＋＋寫

好像沒有ＪＡＶＡ版本　所以可以自己寫寫看　或是問同學了

http://www.csie.ntnu.edu.tw/~u91029/