多維陣列 (Multi-Dimensional Array)

- 陣列的每一個元素都可以看做是單獨的變數, 如果每一個元素自己本身也指向一個陣列, 就會形成一個指向陣列的陣列。
- 這種陣列, 稱為 **2 維陣列 (2 - Dimensional Array)**, 而前面所介紹儲存一般資料的陣列, 就稱為**1 維陣列 (1 - Dimensional Array)**
- 依此類推, 您也可以建立一個指向 2 維陣列的陣列, 其中每一個元素都指向一個 2 維陣列, 這時就稱這個陣列為 3 維陣列。

多維陣列 (Multi-Dimensional Array) (Cont.)

- 這些大於 1維的陣列, 統稱為**多維陣列**, 每多一層陣列, 就稱是多一個**維度** (Dimension)
- 假設有某高職資料處理科有46位同學，每一位同學修9門課，若要以陣列將同學的成績記錄下來，應如何做?
- **多維陣列的宣告**
  - 舉例：宣告一個2維的整數 (int) 陣列

```
int[][] scores;
```

多維陣列 (Multi-Dimensional Array) (Cont.)

- 陣列配置

```
scores = new int[46][9];
```

```
int[][] scores = new int[46][9];
```

- 在電腦裡數位的彩色影像由許多像素點組成，例如一張128＊128像素點的影像，其中每一個像素點是由紅色、綠色及藍色所組成，同時，每一原色的值最小為0最大為255，問，如何利用陣列來存一張影像的像素資料?
- 利用3維陣列宣告一個3維的浮點數 (double) 陣列

```
int[][][] colImg;
```

Height Width Plane

多維陣列 (Multi-Dimensional Array) (Cont.)

- 3維陣列配置

colImg = new int[128][128][3];

int[][][] colImg = new int[128][128][3];

colImg[12][1][2];

- 問

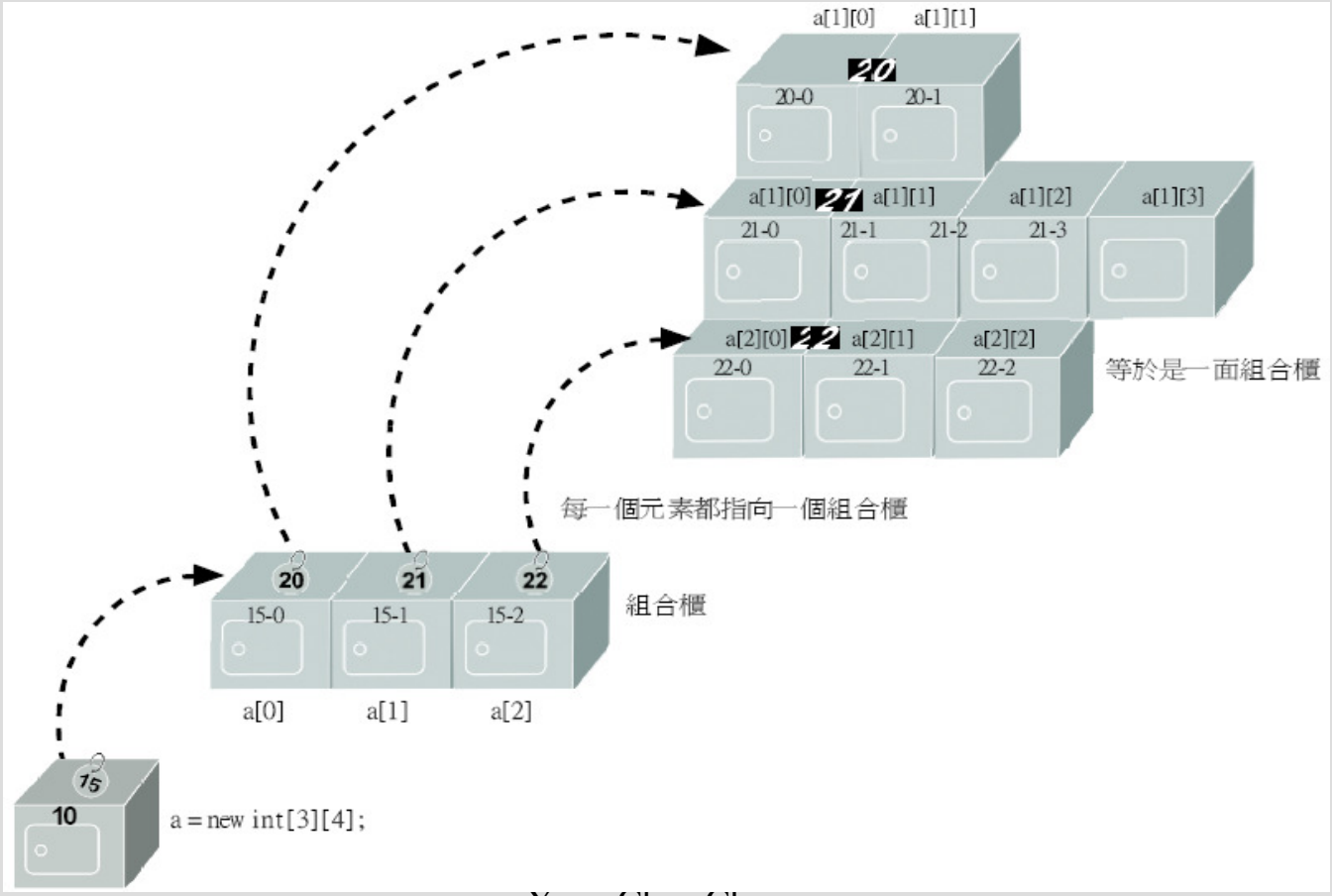
- 分層配置
  - 宣告之後的陣列，不一定必須都在同一時間配置，甚可分不同時候進行記憶體配置

程式 TwoDimArrayAlloc.java 個別配置第2維的陣列

```
01 public class TwoDimArrayAlloc {
02   public static void main(String[] argv) {
03     int[][] a;
04     a = new int[3][];
05
06     for(int i = 0;i < a.length;i++) {
07       a[i] = new int[4]; // 個別配置第 2 維的陣列
08     }
09
10     System.out.println("a 共有 " + a.length + " 個元素。");
11
12     for(int i = 0;i< a.length;i++) {
13       System.out.println("a[" + i + "] 共有 " +
14         a[i].length + " 個元素。");
15     }
16   }
17 }
```

多維陣列 (Multi-Dimensional Array) (Cont.)

- 非矩形的陣列宣告與配置



多維陣列 (Multi-Dimensional Array) (Cont.)

- 非矩形的陣列宣告與配置　舉例

程式　NonRectangular.java 非矩形的多維陣列

```
01 public class NonRectangular {
02   public static void main(String[] argv) {
03     int[][] a = new int[3][];
04
05     a[0] = new int[2]; // 有2個元素
06     a[1] = new int[4]; // 有4個元素
07     a[2] = new int[3]; // 有3個元素
08
09     System.out.println("a共有 " + a.length + "個元素。");
10
11     for(int i = 0;i< a.length;i++) {
12       System.out.println("a[" + i + "] 共有 " +
13         a[i].length + "個元素。");
14     }
15   }
16 }
```

執行結果

```
a共有 3個元素。
a[0] 共有 2個元素。
a[1] 共有 4個元素。
a[2] 共有 3個元素。
```

a 指向一個擁有 3 個元素的陣列

配置給 a[0] 2 個元素的陣列

配置給 a[1] 4 個元素的陣列

配置給 a[2] 3 個元素的陣列

多維陣列 (Multi-Dimensional Array) (Cont.)

- Java 允許建立這樣的多維陣列。如果以組合櫃的方式來描繪這種陣列, 就會發現這種陣列組合起來的組合櫃並不是矩形, 因此稱為**非矩形陣列 (Non -Rectangular Array)**

多維陣列 (Multi-Dimensional Array) (Cont.)

- 直接配置與設定多維陣列
  - 多維陣列也和1維陣列一樣, 可以同時配置並且設定個別元素的內容。只要使用多層的大括號, 就可以對應到多維陣列的各個維度, 直接指定要配置的元素個數與元素內容。

程式 MultiArrayInit.java 宣告同時設定多維陣列的內容

```
01 public class MultiArrayInit {
02   public static void main(String[] argv) {
03     // 直接配置與設定元素
04     int[][] a = {{1,2,3,4},{5,6,7,8}};
05
06     System.out.println("a 共有 " + a.length + " 個元素。");
07
08     for(int i = 0;i< a.length;i++) {
09       System.out.println("a[" + i + "] 共有 " +
10         a[i].length + " 個元素。");
11
12       for(int j = 0;j < a[i].length;j++) {
13         System.out.println(
14           "a[" + i + "][" + j +  "] : " + a[i][j]);
15       }
16     }
17   }
18 }
```

執行結果

```
a 共有 2 個元素。
a[0] 共有 4 個元素。
a[0][0] : 1
a[0][1] : 2
a[0][2] : 3
a[0][3] : 4
a[1] 共有 4 個元素。
a[1][0] : 5
a[1][1] : 6
a[1][2] : 7
a[1][3] : 8
```

多維陣列 (Multi-Dimensional Array) (Cont.)

- 如果維度較多, 或者是元素數量較多時, 建議將大括號的配對依據陣列的維度排列, 例如：

程式 MultiArrayInitLayout.java適當排列大括號

```
01 public class MultiArrayInitLayout {
02   public static void main(String[] argv) {
03     int[][] a = {{1,2,3,4},
04                  {5,6,7,8}}; // 排列成 2X4 陣列的樣子
05     int[][][] b = {{{1,2},
06                     {3,4},
07                     {5,6}}, // 3X2 陣列
08                    {{7,8},
09                     {9,10},
10                     {11,12}} // 3X2 陣列
11                    }; // 排列成 2X3X2 陣列的樣子
12   }
13 }
```

☆ 注意☆
最後必須要有 ';'

多維陣列 (Multi-Dimensional Array) (Cont.)

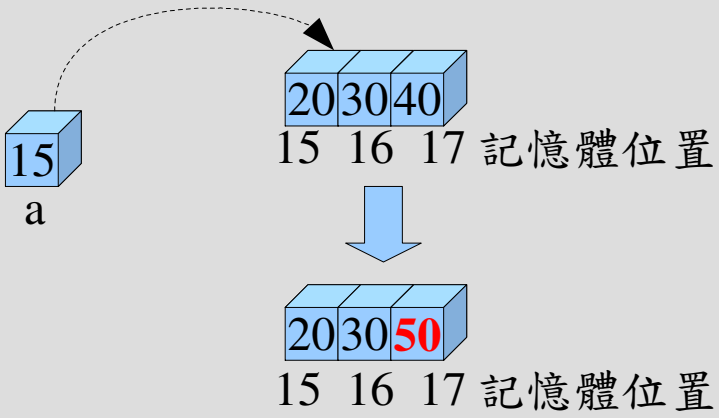
- 多維陣列的使用也和 1 維陣列一樣, 只要在**各維度指定索引碼**, 就可以將指定的元素取出。舉例來說, a[2][3] 就是將 a 指向的多維陣列中第 3 排的組合櫃中的第 4 個保管箱的內容取出來。
- 如果是 a[2], 那就相當於是把 a 指向的多維陣列的第 3 排組合櫃取出來, 而您就可以將 a[2] 視為是指向一個 1 維陣列的變數來使用。

參照型別 (Reference Data Type)

- 特色 1:
  - 參照型別的第一個特性是**間接存取資料**，而不是直接使用變數的內容。

```
int[] a = {20, 30, 40};
```

```
a[2] = 50;
```



參照型別 (Reference Data Type) (Cont.)

- 正因為在存取資料時, 實際上是**參照**變數所記錄的記憶體位址去存取真正的資料, 因此稱之為是參照型別。由於必須透過間接的方式存取資料, 因此存取的時間一定會比使用直接存取資料的基本型別要久。
- 如果是多維陣列, 維度越多, 間接參照的次數也越多, 所花費的時間也就越久。

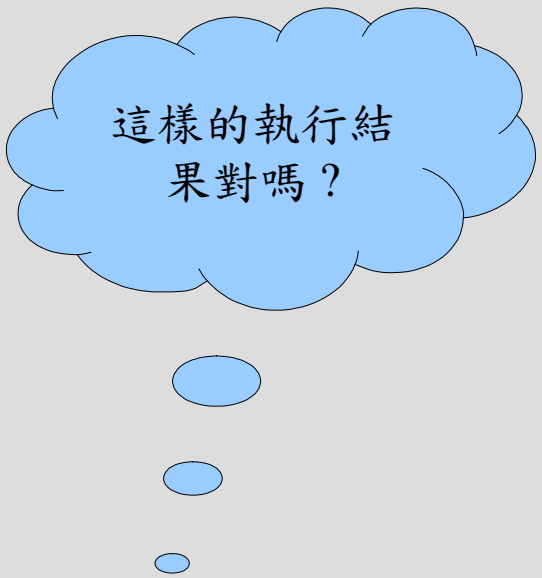
指定運算不會複製資料

程式 ArrayAssignment.java 測試陣列變數的指派運算

```
01 public class ArrayAssignment  {
02    public static void main(String[] argv) {
03       int[] a = {20,30,40};
04       int[] b;
05
06       b = a; // 將 a 的內容放到 b 中
07       b[2] = 100; // 更改陣列 b 的內容
08
09       System.out.println("陣列 a 的元素如下：");
10
11       for(int i : a) { // 顯示陣列 a 的所有元素
12          System.out.print("\t" + i);
13       }
14
15       System.out.println("\n 陣列 b 的元素如下：");
16
17       for(int i : b) { // 顯示陣列 b 的所有元素
18          System.out.print("\t" + i);
19       }
20    }
21 }
```

執行結果

```
陣列 a 的元素如下：
    20      30      100
陣列 b 的元素如下：
    20      30      100
```

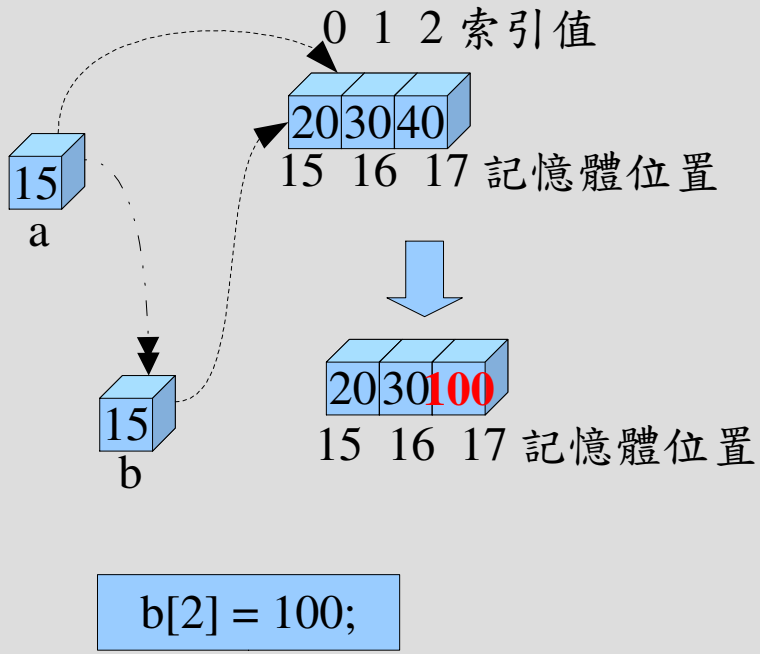


指定運算不會複製資料(Cont.)

- 特色 2:
  - 參照型別最特別但也最需要注意的地方
  - 參照型別的變數所儲存的是存放真正資料的記憶體位址, 因此在第5 行的指定運算中, 等於是把變數 a 所儲存的記憶體位址複製一個放到變數 b 中, 而不是把整個陣列的元素複製一份給變數 b。
  - 這也就是說, 現在 b 和 a 都擁有同樣號碼的號碼牌, 也就等於是 b 和 a 都指向同一個組合櫃：

指定運算不會複製資料(Cont.)

- 因此，接下來透過 b 對陣列的操作，就等於是對 a 所指陣列的操作，而現在 a 和 b 根本就是指向同一個陣列，所以更改的是同一個陣列。



陣列重配置

程式 NewArray.java 重新配置陣列

```
01 public class NewArray  {
02   public static void main(String[] argv) {
03     int[] a = {20,30,40};
04
05     System.out.println("陣列a的元素如下：");
06
07     for(int i : a) { // 顯示陣列a的所有元素
08       System.out.print("\t" + i);
09     }
10
11     a = new int[2]; // 重新配置陣列
12     a[0] = 100;
13     a[1] = 200;
14
15     System.out.println("\n陣列a的元素如下：");
16
17     for(int i : a) { // 顯示陣列a的所有元素
18       System.out.print("\t" + i);
19     }
20   }
21 }
```

執行結果

```
陣列a的元素如下：
20      30      40
陣列a的元素如下：
100     200
```

重新幫 a 配置陣列，這個新的陣列和一開始所配置的陣列大小不一樣，完全是新的陣列。事實上，這並非重新配置陣列，而只是捨棄了原本的陣列，再配置一個新的陣列而已。

陣列重配置（Cont.）

- 注意事項:
  - 在重新配置陣列的時候, 有一點要注意的是, 不能使用同時配置與設定元素內容的方式, 因為這種方式只能用在宣告陣列變數的時候。也就是說, 您不能撰寫如下的敘述：

```
a = {100,200};
```

  - 因為這並不是一個宣告陣列變數的敘述。

課堂練習　(Mar. 24, 2009)

- 二元搜尋
  - 讓使用者輸入要資料個數
  - 進行排序
  - 進行二元搜尋

二分搜尋法 (Binary Search)

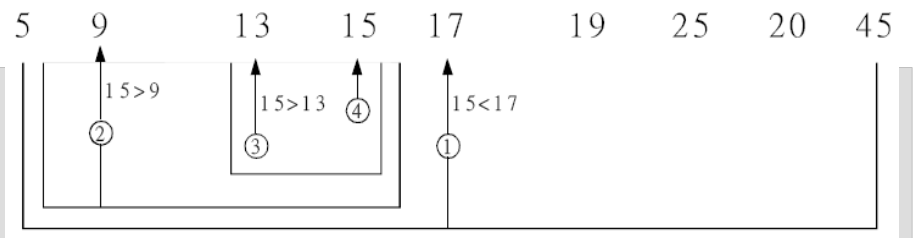
- 在已排好順序的資料中找尋資料, 最常使用的一種方法稱為**二分搜尋法**, 可以一次捨棄剩餘資料的一半, 大幅提升搜尋的效率。
- 也正因為其每次過濾一半資料的特性, 所以稱為二分搜尋法。
- 舉例來說, 假設陣列中有以下 9 項已排序的資料：

```
5,9,13,15,17,19,25,30,45
```

二分搜尋法 (Binary Search)

程式 Binarysearch.java 二分搜尋法

```
01 import java.io.*;
02
03 public class Binarysearch {
04   public static void main(String[] argv)
05     throws IOException{
06     int[] data = {5,9,13,15,17,19,25,30,45}; // 已排序資料
07
08     System.out.println("請輸入要找尋的資料");
09     System.out.print("→");
10
11     BufferedReader br =
12       new BufferedReader(new InputStreamReader(System.in));
13
14     String str = br.readLine();
15
16     // 轉換爲 int
17     int target = java.lang.Integer.parseInt(str);
18
19     int high = data.length - 1; // 範圍區間右邊位置
20     int low = 0; // 範圍區間左邊位置
21     int middle = 0;
22     int times = 0; // 搜尋次數
23
24     while(low <= high) { // 還沒找完
25       times++; // 累計次數
26       middle = (low + high) / 2; // 找出中間值
27
28       System.out.println("尋找區間：" + low + "(" +
29           data[low] + ").." + high + "(" + data[high] +
30           "),中間：" + middle + "(" + data[middle] + ")");
31
32       if(target == data[middle]) { // 找到了
33         break;
34       } else if (target < data[middle]) { // 在左半邊
35         high = middle - 1;
36       } else { // 在右半邊
37         low = middle + 1;
38       }
39     }
40
41     System.out.println("經過 " + times + " 次的尋找");
42
43     if(target == data[middle]) {
44       System.out.println("您要找的資料在陣列中的第" +
45         middle + "個位置");
46     } else {
47       System.out.println(target + "不在陣列中");
48     }
49   }
50 }
```



資源回收系統 (Garbage Collection System)

- 瞭解了上述參照型別的特色後, 您可能已經想到了一個問題, 若不斷的配置陣列, 也就是不斷的向作業系統索取記憶體, 則很可能全部的記憶體都被佔用, 沒有閒置的記憶體空間可用的情況？
- 為了避免這樣的狀況, Java 設計了一個特別的機制, 可以將已經閒置不再需要使用的記憶體空間回收, 以便供後續需要時使用。這個機制就稱為**資源回收系統**。

資源回收系統 (Garbage Collection System) (Cont.)

- **參照計數 (Reference Count)**
  - 為了讓資源回收系統能夠運作, 就必須要有一套監控記憶體空間使用狀況的機制, 這樣才能在發現有閒置的記憶體空間時, 自動將之回收。Java 的作法很簡單, 它會監控對應於每一個記憶體的位址個數, 以剛剛看過的

程式 ArrayAssignment.java

```
01 public class ArrayAssignment  {
02   public static void main(String[] argv) {
03     int[] a = {20,30,40};
04     int[] b;
05     b = a; // 將a的內容放到b中
.....
```

變數 a 和 b 都擁有對應同一個陣列的記憶體位址, 也就是該陣列目前已被參照2次。這個數目稱為**參照計數 (Reference Count)**, 因為它記錄了目前有多少變數還參照到這塊記憶體, 也就是還有多少變數可能會用到這塊記憶體空間。

資源回收系統 (Garbage Collection System) (Cont.)

- 有了參照計數後, 資源回收系統就可以在某個記憶體的參照計數為 0 時, 認定不再有需要使用該記憶體空間的可能, 因而回收該記憶體。那麼參照計數在甚麼狀況下才會減少呢？可分成 3 種狀況：
  - 參照型別的變數**自行歸還**記憶體空間：只要將參照型別的變數指定為**字面常數 null**, 亦即：

```
int[] a = {10,20,30};
.....
a = null;//表示 a 不再需要使用 {10,20,30} 這個陣列
```

資源回收系統 (Garbage Collection System) (Cont.)

- 等於告訴資源回收系統該變數不再需要使用，表示該記憶體空間被參照次數減 1。
- 參照型別變數被重新給了其他的記憶體位址：參照型別變數在同一時間僅能對應到一個記憶體位置, 如果指派給它另一個記憶體位址, 像是重新配置陣列, 它將放棄參照原來的記憶體位址，因此對應記憶體的參照計數也會減 1。例如：

```
int[] a = {10,20,30};
int[] b = {100,200};
......
a = b; // 取得新的號碼牌, 必須歸還原來的號碼牌
......
```

資源回收系統 (Garbage Collection System) (Cont.)

- 參照型別的變數離開有效範圍, 自動失效時。有關這一點, 會在下一章說明。
- 有了這樣的規則, 資源回收系統就可以知道哪些記憶體空間還可能會再用到, 而哪一些記憶體空間不可能會再用到了。

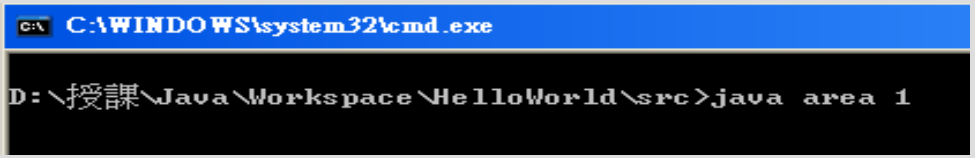
- 回收的時機
  - 一旦發現有閒置的記憶體時, 資源回收系統並不會立即進行回收的動作, 而是先將這個記憶體空間的位址記錄下來。

資源回收系統 (Garbage Collection System) (Cont.)

- 這是因為回收記憶體空間的工作並不單單只是將其收回, 可能還必須將記憶體拆開, 或是與其他的記憶體集中放置等搬移動作, 這些動作都必須耗費時間。
- 如果資源回收系統在發現閒置的記憶體空間的同時便立即回收, 就可能會影響到程式正在執行的重要工作。
- 因此, 資源回收系統會等到發現您的程式似乎沒有在執行繁重的工作, 像是等待網路連線的對方回應時, 才進行回收的工作。

命令列參數：argv 陣列

- 假設你寫了一支程式可以分別計算正方形、梯形、三角型及圓形的面積，而你又希望能在執行該程式時一併將參數加入，例如:



- Java 虛擬機器就會配置一個字串陣列, 然後將程式名稱 (以此例來說就是 area) 之後的字串, 以空白字元分隔切開成多個單字 , 依序放入陣列中，然後, 將這個陣列傳遞給 main() 方法, 而 argv 就會指向這個陣列

命令列參數：argv 陣列 (Cont.)

程式 ShowArgv.java 顯示命令列傳入的參數

```
01 public class ShowArgv {
02   public static void main(String[] argv) {
03     for(int i = 0;i < argv.length;i++) {
04       System.out.println("第 " + i +" 個參數：" + argv[i]);
05     }
06   }
07 }
```

程式

```
java ShowArgv test.html readme.txt
```

命令列執行

執行結果

```
第 0 個參數：test.html
第 1 個參數：readme.txt
```

命令列參數：argv 陣列 (Cont.)

- 如果要傳遞的資訊本身包含有空白, 可以使用一對**雙引號** (“) 將整串字括起來, 例如：

```
java ShowArgv this is a text 測試檔名
```

- 其中的 this is a text 會被拆解為 4 個單字，如下：

執行結果

```
第 0 個參數：this
第 1 個參數：is
第 2 個參數：a
第 3 個參數：text
第 4 個參數：測試檔名
```

- 如果 “this is a text” 是單一項資訊, 就得使用一對雙引號括起來：

```
java ShowArgv "this is a text" 測試檔名
```

執行結果

```
第 0 個參數：this is a text
第 1 個參數：測試檔名
```

argv 陣列內容的處理

- 由於 argv 指向的是一個字串陣列, 因此不論您在指令行中鍵入甚麼資訊, 實際上在 argv 中都只是一串文字。
- 如果您希望傳遞的是數值資料, 那麼就必須自行將由數字構成的字串轉換成為數值, 這可以透過 parseInt 指令來達成。
- 舉例來說, 如果您想要撰寫一個程式, 將使用者傳遞給main( ) 方法的整數數值算出階乘值後顯示出來

java Factory 5

argv 陣列內容的處理 (Cont.)

程式 Factory.java 計算階乘值

```
01 public class Factory {
02   public static void main(String[] argv) {
03     int i = 0 ,fact = 1;
04
05     i = java.lang.Integer.parseInt(argv[0]); // 轉換爲 int
06
07     for(;i > 0;i--) { // 計算 i!
08       fact *= i;
09     }
10
11     System.out.println(argv[0] + "!=" + fact);
12   }
13 }
```

執行結果

```
> java Factory 5
5!=120
```

Eclipse 中的程式執行參數設定