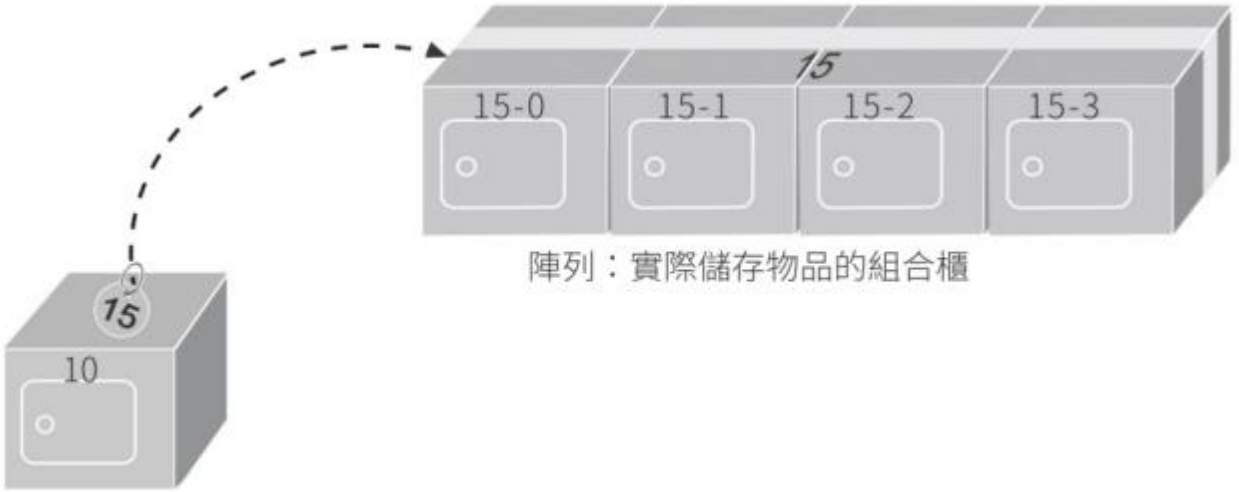
第 7 章

陣列（Array）

7-1 甚麼是陣列？

- 可以儲存多項資料的變數，
  而且隨時知道所儲存資料的數量，
  同時還能依據數量，
  一一取出各項資料進行處理



甚麼是陣列？

程式 使用組合櫃計算平均成績的演算法

```
01  依據學生人數,配置一個組合櫃
02  將學生分數依序填入組合櫃中的各個保管箱
03  sum = 0;
04  從組合櫃中一一取出各個保管箱的值 {
05      sum += 保管箱的值 // 加總
06  }
07  average = sum / 組合櫃中保管箱的數目
08  顯示平均成績
```

7-1-1 陣列的宣告與配置

- 使用陣列必須分成 2 個步驟, 第 1 個步驟是宣告陣列變數, 第 2 個步驟是配置陣列。

程式 ArrayAverage.java 使用陣列計算平均值

```
01 public class ArrayAverage {
02   public static void main(String[] argv) {
03     double[] students; // 宣告陣列
04     students = new double[5]; // 配置陣列
05
06     students[0] = 70; // 指派第1個保管箱的內容
07     students[1] = 65; // 指派第2個保管箱的內容
08     students[2] = 90; // 指派第3個保管箱的內容
09     students[3] = 85; // 指派第4個保管箱的內容
10     students[4] = 95; // 指派第5個保管箱的內容
```

陣列的宣告與配置

```
11
12      double sum = 0;
13      for(int i = 0;i < students.length;i++) {
14        sum += students[i]; // 加總
15      }
16
17      double average =  sum / students.length; // 計算平均
18
19      System.out.println("平均成績：" + average);
20    }
21  }
```

執行結果

```
平均成績：81.0
```

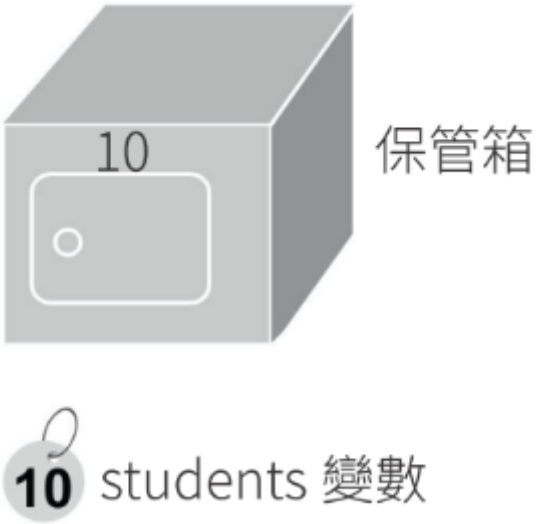
陣列變數的宣告

- 在 ArrayAverage.java 的第 3 行就宣告了一個陣列變數：

```
double[] students; // 宣告陣列
```

陣列變數的宣告

- 只要在型別名稱的後面加上一對中括號 [], 就表示要宣告一個指向可以放置該型別資料陣列的變數
- students 這個變數會指到一個儲存 double 資料的陣列



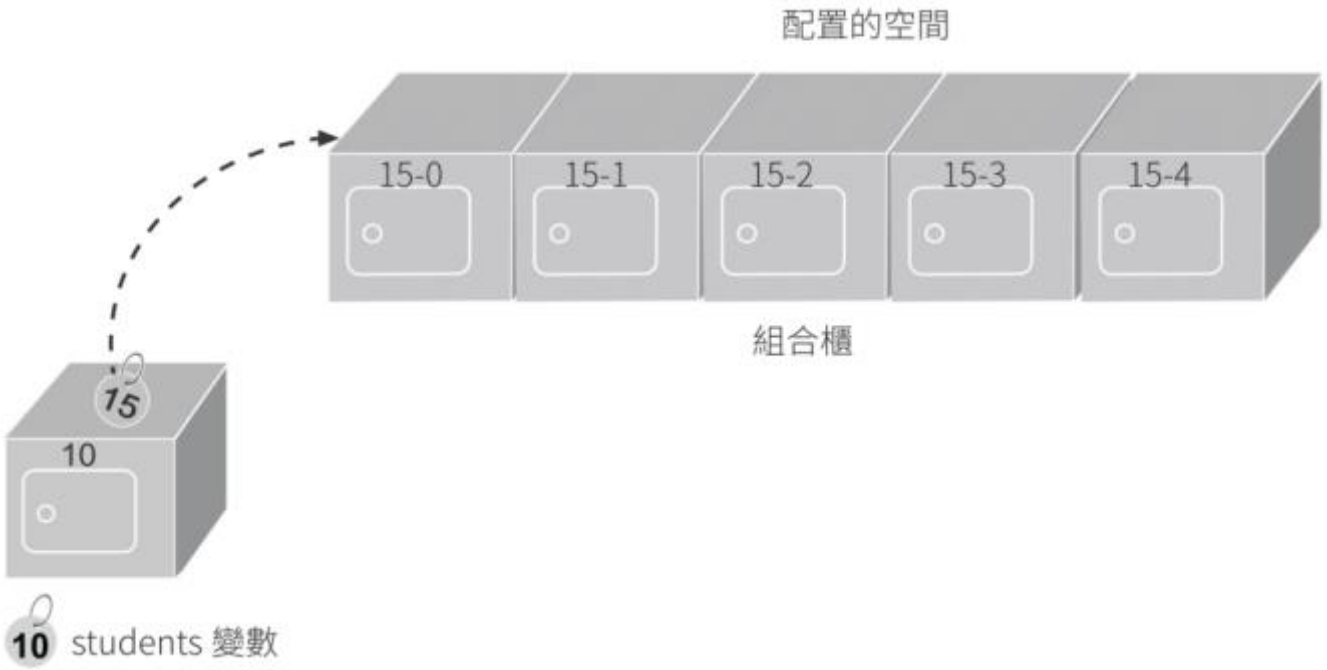
陣列的配置

- 在 ArrayAverage.java 的第 4 行中，就是配置陣列的動作：

```
students = new double[5]; // 配置陣列
```

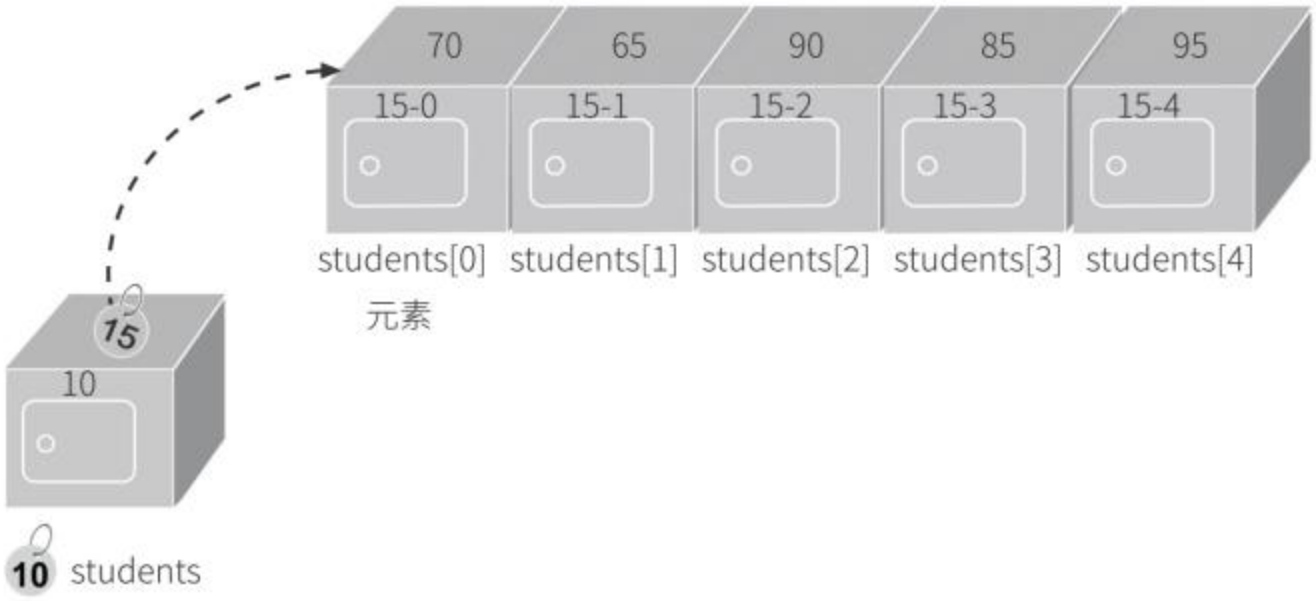
陣列的配置

- 要配置陣列, 必須使用 new 算符。
- new 算符的運算元就表示了所需要的空間大小。



使用陣列元素

```
06      students[0] = 70; // 指派第1個保管箱的內容
07      students[1] = 65; // 指派第2個保管箱的內容
08      students[2] = 90; // 指派第3個保管箱的內容
09      students[3] = 85; // 指派第4個保管箱的內容
10      students[4] = 95; // 指派第5個保管箱的內容
```



使用陣列元素

- 陣列還提供許多關於該陣列的資訊，稱為屬性 (Attributes)。
- 其中有一項 length 屬性，代表該陣列中元素的數量。
- 只要在陣列變數的名稱之後加上 .length 即可取得這個屬性的內容。

使用陣列元素


```
12      double sum = 0;
13      for(int i = 0;i < students.length;i++) {
14        sum += students[i]; // 加總
15      }
```

- 最後再利用陣列的 length 屬性值當除數, 以算出平均值, 如第 17 行：

```
double average =  sum / students.length; // 計算平均
```

7-1-2 使用陣列的好處

- **只需宣告一個陣列變數，**
  而不需要宣告和學生人數相同數量的變數
- 資料是從檔案中循序讀入的話，
  就**可以使用迴圈依序放入陣列的元素中**
- **可以使用索引碼存取陣列元素，**
  不論陣列多大，都一樣適用，
  而不需要去修改程式

7-2 陣列的配置與初值設定

7-2-1 宣告同時配置

程式 DeclareAndNew.java 宣告並同時配置陣列空間

```
01 public class DeclareAndNew {
02   public static void main(String[] argv) {
03     double[] students = new double[5]; // 宣告並配置陣列
04
05     students[0] = 70; // 指派第1個保管箱的內容
06     students[1] = 65; // 指派第2個保管箱的內容
       .......
```

宣告同時配置

程式 ArrayWithExpr.java 使用變數與運算式配置陣列空間

```
01 public class ArrayWithExpr {
02   public static void main(String[] argv) {
03     int i = 4,j = 8;
04     int[] a = new int[i];       // 使用變數
05     int[] b = new int[j - i]; // 使用運算式
06   }
07 }
```

- 由於元素個數必須為整數，因此只有運算結果為整數值的運算式才能用來指定陣列的元素個數。

7-2-2 設定陣列的初值

程式 DeclareAndInit.java 宣告同時配置與設定陣列初值

```
01 public class DeclareAndInit {
02   public static void main(String[] argv) {
03     // 宣告並指派陣列內容
04     double[] students = {70, 65, 90, 85, 95};
05     double sum = 0;
06
07     for(int i = 0;i < students.length;i++) {
08       sum += students[i]; // 加總
09     }
10
11     double average =  sum / students.length; // 計算平均
12
13     System.out.println("平均成績：" + average);
14   }
15 }
```

設定陣列的初值

1. 要直接設定陣列的初值,
   必須使用一對大括號,
   列出陣列中每一個元素的初值。
2. 記得要在右大括號後面加上分號 (;),
   表示整個宣告敘述的結束。

設定陣列的初值

程式 ArrayInitWithExpr.java 使用運算式設定陣列初值

```
01 public class ArrayInitWithExpr {
02   public static void main(String[] argv) {
03     int i = 4,j = 8;
04     int[] a = {4, i, i + j};  // 使用運算式
05
06     for(int k = 0;k < a.length;k++) {
07       System.out.println("a[" + k + "] : " + a[k]);
08     }
09   }
10 }
```

執行結果

```
a[0] : 4
a[1] : 4
a[2] : 12
```

7-2-3 配合陣列使用迴圈

程式 ArrayLoop.java 使用特殊的 for 迴圈

```
01 public class ArrayLoop {
02   public static void main(String[] argv) {
03     double[] students = {70, 65, 90, 85, 95};
04     double sum = 0;
05
06     for(double score : students) { // 使用特殊的for迴圈
07       sum += score; // 加總
08     }
09
10     double average =  sum / students.length; // 計算平均
11     System.out.println("平均成績：" + average);
12   }
13 }
```

7-2-3 配合陣列使用迴圈

- for(:) 的作用是迴圈進行時,
  每一輪就從“:”後面所列的陣列取出下一個元素,放到冒號前面所指定的變數中,
  然後執行迴圈本體的敘述。
- ":" 前面的變數必須和陣列的型別一致,
  否則編譯時就會發生錯誤。

配合陣列使用迴圈

程式 ArrayLoop2.java 將 for(:) 拆解開來

```
06      for(int i = 0;i < students.length;i++) {
07        double score = students[i];
08        sum += score; // 加總
09      }
```

7-3 多維陣列 (Multi-Dimensional Array)

- 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)。

7-3-1 多維陣列的宣告

```
int[][] a;    //宣告 2 維陣列
```

- 如果依循前面對於 1 維陣列宣告時的瞭解, 可以把 int[] 當成是一種新的資料型別, 表示一個用來儲存 int 型別資料的陣列。這樣一來, 上述的宣告就可以解讀為：

```
(int[])[]  a;
```

多維陣列的宣告

- 宣告一個3維陣列

```
int[][][]   b;
```

7-3-2 多維陣列的配置

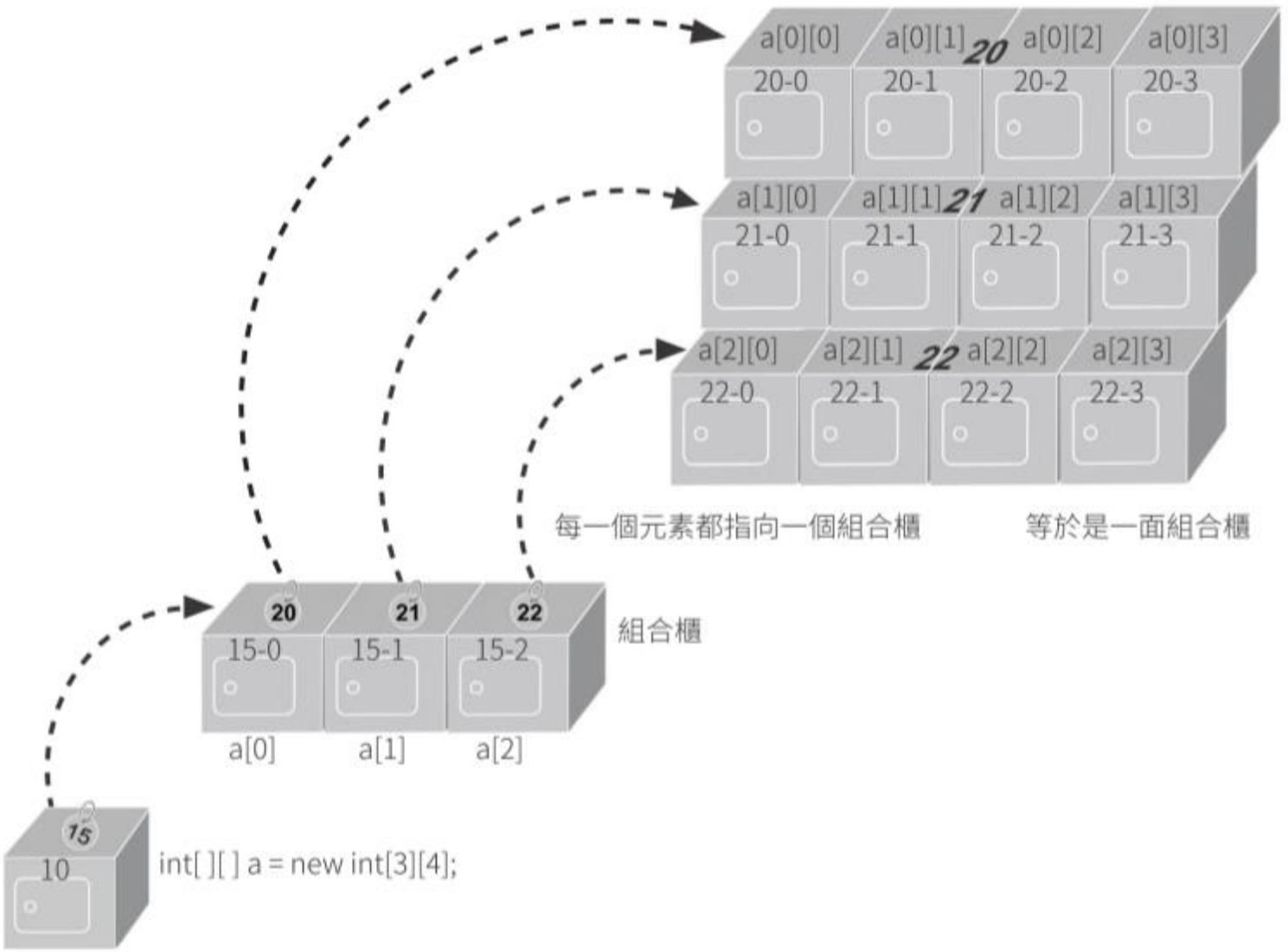
程式 TwoDimArray.java 2 維陣列的使用

```
01 public class TwoDimArray {
02   public static void main(String[] argv) {
03     int[][] a= new int[3][4]; // 宣告2維陣列並配置空間
04
05     System.out.println("a共有 " + a.length + "個元素。");
06
07     for(int i = 0;i< a.length;i++) {
08       System.out.println("a[" + i + "] 共有 " +
09         a[i].length + "個元素。");
10     }
11   }
12 }
```

執行結果

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

多維陣列的配置


a[0][0]
a[0][1]
20
a[0][2]
a[0][3]
20-0
20-1
20-2
20-3
a[1][0]
a[1][1]
21
a[1][2]
a[1][3]
21-0
21-1
21-2
21-3
a[2][0]
a[2][1]
22
a[2][2]
a[2][3]
22-0
22-1
22-2
22-3
每一個元素都指向一個組合櫃
等於是一面組合櫃
20
21
22
15-0
15-1
15-2
組合櫃
a[0]
a[1]
a[2]
15
10
int[ ][ ] a = new int[3][4];

分層配置

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 = new int[3][];
04
05     for(int i = 0;i < a.length;i++)
06       a[i] = new int[4]; // 個別配置第 2 維的陣列
07
08     System.out.println("a共有 " + a.length + "個元素。");
09
10     for(int i = 0;i< a.length;i++) {
11       System.out.println("a[" + i + "] 共有 " +
12         a[i].length + "個元素。");
13     }
14   }
15 }
```

分層配置

- 如果使用這種配置方式,請特別注意只有右邊維度的元素數目可以留空,最左邊的維度一定要指明。

程式 錯誤的多維陣列配置方式

```
01 public class TwoDimArrayAllocErr {
02   public static void main(String[] argv) {
03     int[][] a;
04     int[][][]b;
05     a = new int[][4];
06     b = new int[][3][];
07   }
08 }
```

分層配置

程式 高維度的元素個數可以先空著

```
03      int[][] a;
04      int[][][]b;
05      a = new int[3][];
06      b = new int[4][][];
```

非矩形的多維陣列

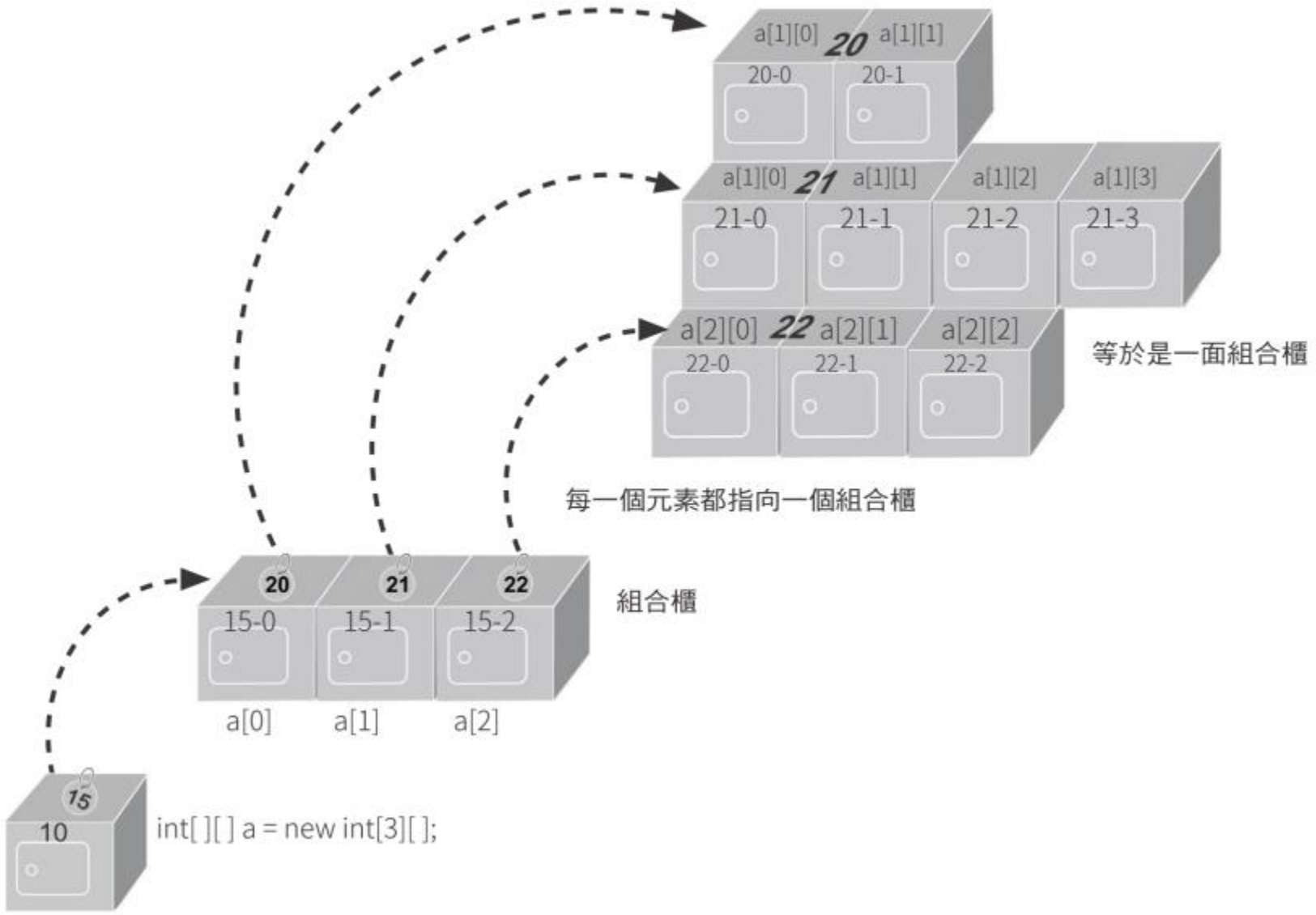
程式 NonRectangular.java 非矩形的多維陣列

```
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。
```

非矩形的多維陣列


a[1][0] 20 a[1][1]
20-0
20-1
a[1][0] 21 a[1][1] a[1][2] a[1][3]
21-0
21-1
21-2
21-3
a[2][0] 22 a[2][1] a[2][2]
22-0
22-1
22-2
等於是一面組合櫃
每一個元素都指向一個組合櫃
20
21
22
15-0
15-1
15-2
組合櫃
a[0]
a[1]
a[2]
15
10
int[ ][ ] a = new int[3][ ];

直接配置與設定多維陣列

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值
04     int[][] a = {{1,2,3,4},    // 可排列成 2x4
05                  {5,6,7,8}};   // 的型式以方便閱讀
06
07     System.out.println("a共有 " + a.length + "個元素。");
08
09     for(int i = 0;i< a.length;i++) {
10       System.out.println("a[" + i + "] 共有 " +
11               a[i].length + "個元素。");
12
```

直接配置與設定多維陣列

```
13          for(int j = 0;j < a[i].length;j++)
14            System.out.println(
15                    "a[" + i + "][" + j +  "] : " + a[i][j]);
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
```

直接配置與設定多維陣列

- 多維陣列也可以和 foreach 迴圈搭配，EX:

程式 MultiArrayForeach.java 在多維陣列上使用 for-each 迴圈

```
01 public class MultiArrayForeach {
02   public static void main(String[] argv) {
03     int[][] a = {{1,2,3,4},{5,6,7,8}};
04
05     for(int[] i : a) { // 使用for-each
06       for(int j : i) { // 使用for-each
07         System.out.print(j + "\t");
08       }
09        System.out.println("");
10     }
11   }
12 }
```

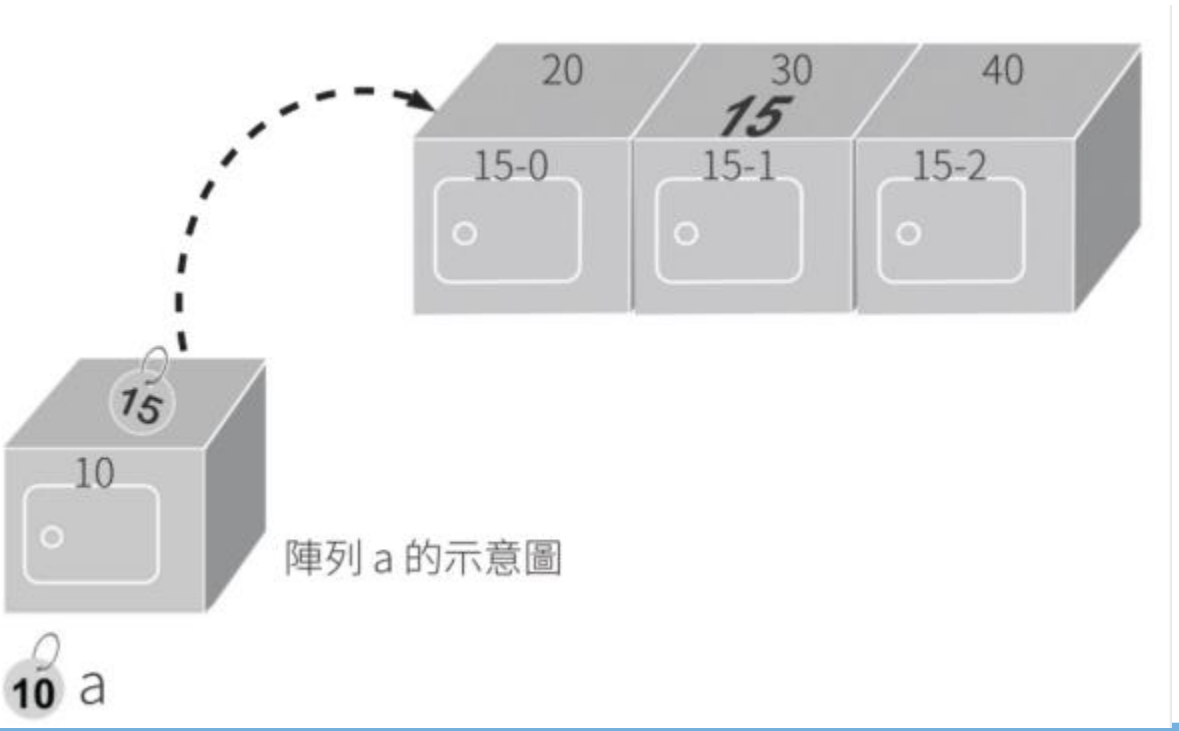
執行結果

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

7-4 參照型別（Reference Data Type）

7-4-1 參照型別的特色

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

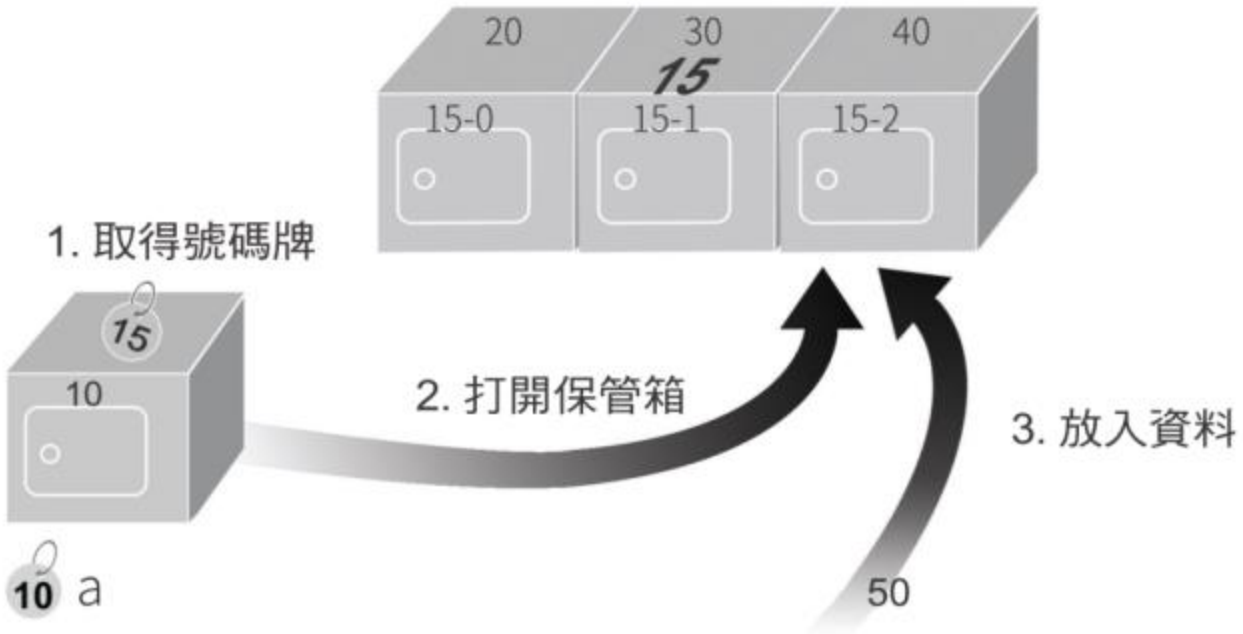


陣列 a 的示意圖

間接存取資料

- 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。

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



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

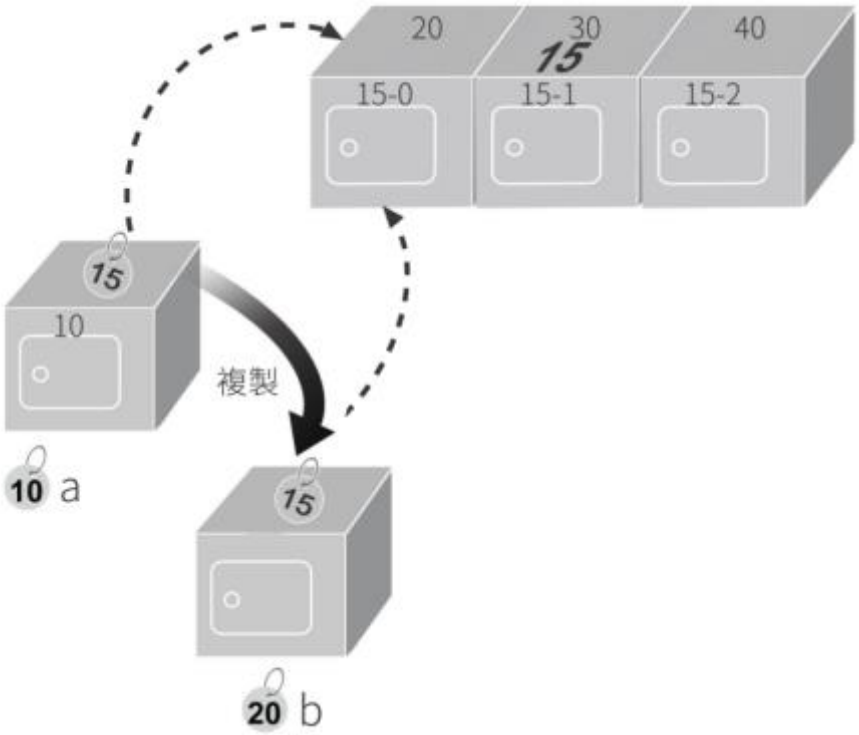
```
01 public class ArrayAssignment  {
02   public static void main(String[] argv) {
03     int[] a = {20,30,40};
04     int[] b = a; // 將a的內容放到b中
05
06     b[2] = 100; // 更改陣列b的內容
07
08     System.out.print("陣列a的元素：");
09     for(int i : a)   // 顯示陣列a的所有元素
10           System.out.print("\t" + i);
11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

指定運算不會複製資料

```
13      for(int i : b)   // 顯示陣列b的所有元素
14              System.out.print("\t" + i);
15    }
16 }
```

執行結果

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



指定運算不會複製資料

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};  // 原本是 3 個元素的陣列
04
05     System.out.print("陣列a：");
06     for(int i : a)    // 顯示陣列a的所有元素
07       System.out.print("\t" + i);
08
09     a = new int[5]; // 重新配置陣列
10     a[0] = 100;
11     a[1] = 200;
```

指定運算不會複製資料

```
12
13      System.out.print("\n重新配置陣列a：");
14      for(int i : a)    // 顯示陣列a的所有元素
15        System.out.print("\t" + i);
16    }
17 }
```

執行結果

```
陣列a：  20      30      40
重新配置陣列a：  100     200     0       0       0
```

7-4-2 資源回收系統（Garbage Collection System）

- Java 有一個特別機制, 會將不需要使用的組合櫃回收, 以供後續使用, 這個機制就是資源回收系統

7-4-2 資源回收系統（Garbage Collection System）

- 參照計數（Reference Count）記錄了目前有多少變數握有號碼牌

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
       ..........
```

參照計數 (Reference Count)

- 參照計數為 0 時，
  資源回收系統就可以認定不再有需要使用該組合櫃的可能，因而回收該組合櫃。

  那麼參照計數在甚麼狀況下才會減少呢？
  這可以分成 3 種狀況：

參照計數 (Reference Count)

- 參照型別的變數**自行歸還**號碼牌：

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

參照計數 (Reference Count)

- 給予參照型別變數其他組合櫃的號碼牌：

```
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
......
```

- 參照型別的變數離開有效範圍，自動失效時

7-5 命令列參數： argv 陣列

- 程式的 main() 方法前的括號中, 都有 String[] argv, 用途是？

```
public static void main(String[] argv) {
```

7-5-1 argv 與 main() 方法

- main( ) 方法是 Java 程式的起點, 當在命令提示符號下鍵入指令要求執行 Java 程式時, 例如：

```
java ShowArgv
```

- Java 虛擬機器就會載入 ShowArgv 程式, 並且從這個程式的 main( ) 方法開始執行。此時就可以在命令提示符號下鍵入的指令後面加上額外的資訊, 例如：

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

argv 與 main() 方法

- 因此在程式中就可以透過 argv 取出使用者執行程式時附加在程式名稱之後的資

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 }
```

argv 與 main() 方法

- 如果使用以下指令執行這個程式：

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

執行結果

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

argv 與 main() 方法

- 若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
```

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
```

argv 與 main() 方法

- 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
```

7-5-2 argv 陣列內容的處理

- 如果要撰寫一個程式,
  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, 像是這樣：

```
java Factory 5
```

argv 陣列內容的處理

程式 Factory.java 計算階乘值

```
01 public class Factory {
02   public static void main(String[] argv) {
03     double fact = 1;
04     int i = 5;                          // 設定預設值 5
05     if(argv.length > 0) // 如果有設定命令列參數
06       i = Integer.parseInt(argv[0]); // 將參數轉換為 int
07
08     System.out.print(i + "!=");   // 輸出訊息開頭
09     for(;i > 0;i--)   // 計算 i!
10       fact *= i;
11     System.out.println(fact);     // 輸出計算結果
12   }
13 }
```

argv 陣列內容的處理

執行結果 1

```
...>java Factory ◄── 未加參數
5!=120.0
```

執行結果 2

```
...>java Factory 55
55!=1.2696403353658264E73
```

argv 陣列內容的處理

- 若要將命令列參數字串轉換成浮點數, 則可改用：

```
double d = Double.parseDouble(argv[0]);
```

7-6 綜合演練

- 將陣列運用在查表上
- 搜尋 (Search) 資料
- 找出最大與最小值
- 排序 (Sorting)

7-6-1 將陣列運用在查表上

停車時數	費率（元 / 時）
超過 6 小時	100
4~6（含）小時	80
2~4 （含）小時	50
2（含）小時以下	30

- 如果停車 5 小時, 停車費就是：

```
(5 - 4) * 80 + (4 - 2) * 50 + 2 * 30 = 240
```

使用多層的條件敘述

程式 ParkFeeIf.java 以多條件 if 撰寫停車費程式

```
01 public class ParkFeeIf {
02   public static void main(String[] argv) {
03     int hours = 0;
04     int fee = 0;
05
06     // 轉換為 int
07     hours = Integer.parseInt(argv[0]);
08
09     if(hours > 6) { // 先計算超過6小時的部分
10       fee += (hours - 6) * 100;
11       hours = 6;
12     }
13
```

使用多層的條件敘述

```
14      if(hours > 4) { // 計算4~6小時的時段
15        fee += (hours - 4) * 80;
16        hours = 4;
17      }
18
19      if(hours > 2) { // 計算2~4小時的時段
20        fee += (hours - 2) * 50;
21        hours = 2;
22      }
23
24      if(hours > 0) { // 計算2小時內的時段
25        fee += (hours - 0) * 30;
26        hours = 0;
27      }
```

使用多層的條件敘述

```
28
29      System.out.println("停車時數：" + argv[0] + "小時");
30      System.out.println("應繳費用：" + fee + "元整");
31    }
32 }
```

執行結果 1

```
> java ParkFeeIf 5
停車時數：5小時
應繳費用：240元整
```

執行結果 2

```
> java ParkFeeIf 4
停車時數：4小時
應繳費用：160元整
```

使用多層的條件敘述

- 但如果業者要改停車費率：

表 7-2

停車時數	費率 (元 / 時)
超過 7 小時	100
3 ~ 7 (含) 小時	60
3 (含) 小時以下	30

- 那就得更改程式,
  甚至需要移除或是新增 if 敘述,
  平白增加寫錯程式的機會。
  如果善用陣列, 就可以避免這個缺點。

使用陣列

程式 ParkFeeArray.java 使用陣列撰寫多條件的程式

```
01 public class ParkFeeArray {
02   public static void main(String[] argv) {
03     int[] hourTable = {0,2,4,6}; // 時段
04     int[] feeTable = {30,50,80,100}; // 時段費率
05     int hours = Integer.parseInt(argv[0]); //停車時數
06     int fee = 0; //停車費用
07
08     int i = hourTable.length - 1;
09     while(i > 0) {// 先找出最高費率區段
10       if(hourTable[i] < hours)
11         break;
12       i--;
13     }
```

使用陣列


```
14
15      while(i >= 0) { // 由最高費率區段往下累加
16        fee += (hours - hourTable[i]) * feeTable[i];
17        hours = hourTable[i];
18        i--;
19      }
20
21      System.out.println("停車時數：" + argv[0] + "小時");
22      System.out.println("應繳費用：" + fee + "元整");
23    }
24  }
```

使用陣列

- 雖然看起來似乎沒有使用 if 的版本簡單，可是因為採用了查表的方式，即便停車費率的時段或是價格異動，也只需修改陣列中的資料，程式的邏輯部分完全不需要更改。

```
03      int[] hourTable = {0,3,7}; // 時段
04      int[] feeTable = {30,60,100}; // 時段費率
```

7-6-2 找出最大與最小值

- 陣列經常用來存放大量供程式處理的資料，所以常見的操作就是搜尋與排序。
- 搜尋就是在陣列中找出符合特定條件的資料；排序則是將陣列元素依由大到小或由小到大的順序重新排列。

找出最大與最小值

程式 FindMinMax.java 找出最低與最高溫度

```
01 public class FindMinMax {
02
03   public static void main(String[] argv) {
04     int[] temp = {21,18,21,23,25,25,24,22,22,16}; // 溫度
05     int min = temp[0]; // 先將最低溫度設為任一個元素
06     int max = temp[0]; // 先將最高溫度設為任一個元素
07
08     for(int i : temp) { // 一一比較每個元素值
09       if(i < min){
10         min = i; // 更新最低溫度
11       }
12       if(i > max) {
```

找出最大與最小值

```
13            max = i; // 更新最高溫度
14          }
15        }
16
17        System.out.println("全台目前最低的溫度是：" + min + "度");
18        System.out.println("全台目前最高的溫度是：" + max + "度");
19    }
20 }
```

執行結果

```
全台目前最低的溫度是：16度
全台目前最高的溫度是：25度
```

7-6-3 搜尋二維陣列

程式 RainArray.java 在二維陣列中搜尋

```
01 public class RainArray {
02   public static void main(String[] argv) {
03     String[] city= {"臺北", "基隆", "宜蘭"};
04     double[][] rain= // 月平均雨量
05           // 一月    二     三     四     五     六
06            {{83.2 , 170.3, 180.4, 177.8, 234.5, 325.9},  // 臺北
07             {331.6, 397.0, 321.0, 242.0, 285.1, 301.6},  // 基隆
08             {147.0, 182.3, 127.5, 138.4, 211.7, 214.2}}; // 宜蘭
09     int indexMin=0, indexMax=0;  // 最低、高的城市索引先設為 0
10
11     // 找各月份雨量最低、最高者
12     for(int month=0; month<6; month++){
13       for(int i=0; i<rain.length; i++) { // 找最低、最高平均雨量
14         if(rain[i][month] < rain[indexMin][month])
15           indexMin = i; // 更新平均雨量最低的城市索引
16
```

搜尋二維陣列

```
17            if(rain[i][month] > rain[indexMax][month])
18              indexMax = i; // 更新平均雨量最高的城市索引
19          }
20
21          System.out.println((month+1)+"月平均雨量最低："
22                    + city[indexMin] + "\t最高：" + city[indexMax]);
23        }
24      }
25  }
```

執行結果

```
1月平均雨量最低：臺北    最高：基隆
2月平均雨量最低：臺北    最高：基隆
3月平均雨量最低：宜蘭    最高：基隆
4月平均雨量最低：宜蘭    最高：基隆
5月平均雨量最低：宜蘭    最高：基隆
6月平均雨量最低：宜蘭    最高：臺北
```

7-6-4 排序（ Sorting ）

- 排序也是常見的資料處理。
  氣泡排序法 (Bubble Sort)是一種簡單的排序方法。假設陣列中有 n 個元素：

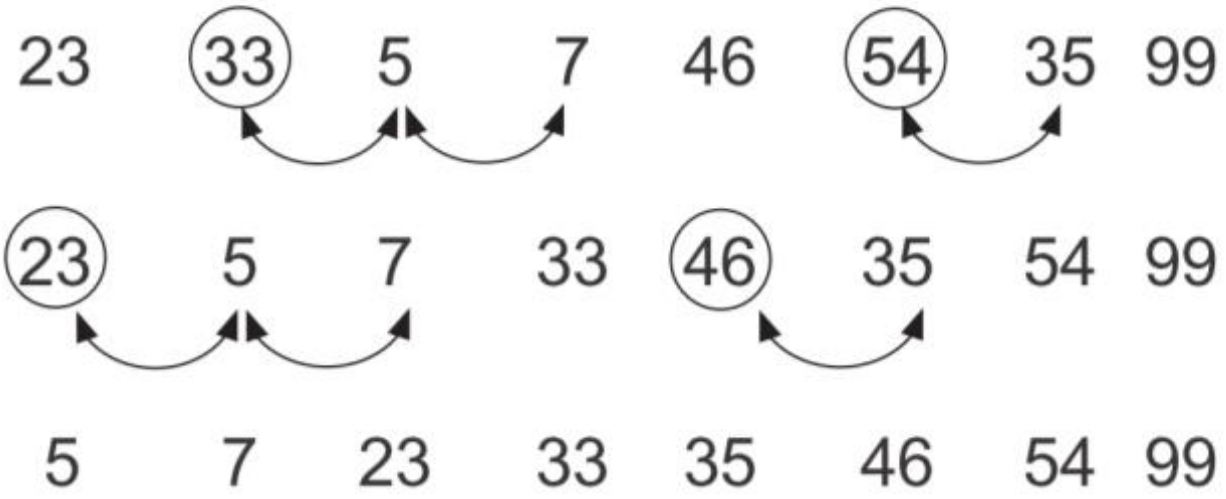
排序（ Sorting ）

1. 第 1 輪先從索引碼為 0 的元素開始，往後兩兩相鄰元素相比，如果前面的元素比後面的元素大，就把兩個元素對調。一直比對到陣列最後，索引碼為 n-1 的這個元素 (也就是陣列的最後一個元素) 必定是最大的元素。
2. 重複上述的步驟，依序將第 2 大、第 3 大、....、第 i 大的元素移到正確的位置。第 i 輪僅需比對到第 n-i 個元素即可，因為後面的元素已經依序排好了。

排序（ Sorting ）


```
23  33   5   7  46  54  35  99
```

排序的過程如下：



排序（ Sorting ）

程式 BubbleSort.java 氣泡排序法

```
01 public class BubbleSort {
02
03   public static void main(String[] argv) {
04     int[] data = {23,54,33,5,7,46,99,35}; // 未排序的資料
05     int temp; // 用來交換元素的暫存變數
06
07     for(int i = 0;i < data.length - 1;i++) {
08       // 共需進行元素個數-1輪
09       for(int j = 0;j < data.length - 1 - i;j++ ) {
10         // 第i輪比對到倒數第i+1個元素
11         if(data[j] > data[j + 1]) {
12           temp = data[j];
13           data[j] = data[j + 1];
14           data[j + 1] = temp;
```

排序（ Sorting ）

```
15            }
16          }
17
18          for(int k:data) {
19            System.out.print(" " + k);
20          }
21          System.out.println("");
22        }
23      }
24  }
```

執行結果

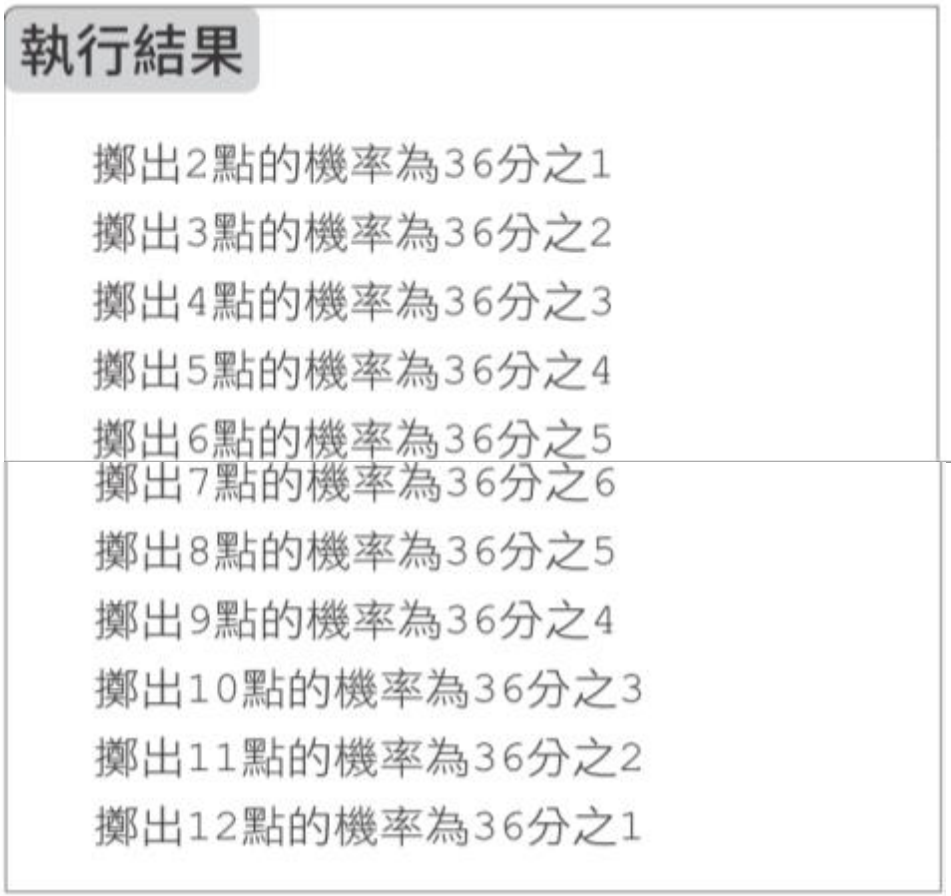
```
23 33 5 7 46 54 35 99
23 5 7 33 46 35 54 99
5 7 23 33 35 46 54 99
5 7 23 33 35 46 54 99
5 7 23 33 35 46 54 99
5 7 23 33 35 46 54 99
5 7 23 33 35 46 54 99
```

7-6-5 利用陣列儲存計算結果

程式 PlayDice.java 統計擲骰的點數出現機率

```
01 public class PlayDice {
02   public static void main(String[] argv) {
03     int[] data = new int[13]; // 儲存擲骰點數出現次數
04     int base=0;
05     for(int i=1;i<=6;i++)      // 2 個迴圈分別代表 2 個骰子
06       for(int j=1;j<=6;j++) { // i+j 就是擲出的點數
07         data[i+j]++;          // 將代表次數的元素加 1
08         base++;               // 加總擲骰組合次數
09       }
10
11     for(int point=0;point<data.length;point++)
12       if(data[point]>0)
13         System.out.println("擲出"+ point + "點的機率為" +
14                            base+ "分之" + data[point]);
15   }
16 }
```

利用陣列儲存計算結果


執行結果
擲出2點的機率為36分之1
擲出3點的機率為36分之2
擲出4點的機率為36分之3
擲出5點的機率為36分之4
擲出6點的機率為36分之5
擲出7點的機率為36分之6
擲出8點的機率為36分之5
擲出9點的機率為36分之4
擲出10點的機率為36分之3
擲出11點的機率為36分之2
擲出12點的機率為36分之1