國立屏東大學 資訊工程學系 物件導向程式設計

9. Arrays

- 參考型別reference type
- 一維陣列
- 多維陣列
- Reference of References

-何謂陣列?

陣列(Array)是程式設計時，相當簡單同時也相當重要的一種資料結構。基本上，陣列可視為一些相同型態的資料集合，同時在這個集合中每一筆資料都有一個唯一的編號，我們將這個編號稱為索引；透過索引，您就可以存取在集合中的特定資料。以下本章將就陣列的宣告及使用等細節做一說明，現在先讓我們來看看陣列在使用上需要瞭解的一些性質：

- 陣列中所存放的資料必須為同一種資料型態。
- 索引值的範圍是由0開始。意即第一筆資料的索引值為0、第二筆為1、第三筆為2、……。

現在讓我們以Java語言宣告一個陣列，存放　五個學生的成績(假設為整數)：

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

score[0]=80;
score[1]=60;
score[2]=100;
score[3]=80;
score[4]=90;
```

上述程式碼說明如下：

- int[] score = new int[5]; → 在記憶體中配置一塊連續的空間，以存放五個整數值。
- int[] score = new int[5]; → 會產生一個名為score的變數，其資料型態為參考型別(reference type)，也就是score本身並不是陣列，它只是一個4個bytes的空間，用以儲存一個記憶體空間。
- int[] score = new int[5]; → 將new int[5]所配置的記憶體空間的開始位址，儲存在score這個reference裡。
- score[i]=XX; → 將XX值寫入到一個記憶體位址，該位址的值可由score所儲存的位址(那是陣列的第一個元素所在的位址)，再加上ix4(因為一個整數佔4個bytes)計算出來。

figure 1顯示了在記憶體中陣列與score這個reference的關係：

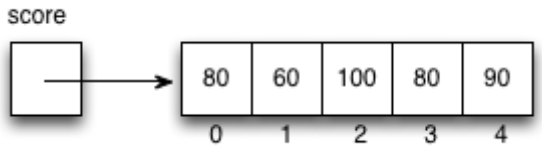

Fig. 1: 一個擁有5個元素的陣列

<note tip> ++++ Reference Type | Java語言的參考型別(reference type)類似C++語言中的reference。一個reference type的變數存放的不是數值，而是一個記憶體位置。回顧一下在C++語言中。reference type的使用：

```
int &r;
int x=5;
r=x;

r=6;   //現在x=6;
```

在這個例子中。r是一個指向x的reference。r所存放的是x的記憶體位址，我們對r進行的操作，其實是對r所指向的記憶體位址進行操作。因此r=6的結果，是讓r所指向的位址的值變成6，其結果就讓x的值成為6。++++ </note>

其實score應該稱為指向陣列的reference。不過一般的習慣還是將它稱為陣列。score陣列(score所指向的記憶體位址的那個陣列)含有五個元素，你只要透過陣列的索引(index。有時亦稱為subscript)就可以存取它們。例如score[0]與score[3]分別代表(score所指向的那個)陣列的第1個與第4個元素。

9.1 一維陣列

9.1.1 宣告

一維陣列的宣告語法如下：

```
DataType[] arrayName;

或

DataType arrayName[]; //雖然Java允許你這樣宣告，但這樣不容易讓你將arrayName視為一個指向DataType[]的reference
```

- DataType是指定存放在陣列中的資料為何種資料型態，只要是一般變數可使用的資料型態都可以使用。
- arrayName定義陣列的名稱。

請不要忘記前述的說明，其實這是一個reference的宣告。arrayName將會用來儲存一個記憶體位址。

<note important> 不要在宣告陣列時指定大小！和C語言非常不同的地方是。Java語言不允許這樣的宣告: int data[3]或是int[3] data。原因很簡單。data是一個 reference而不是陣列。 </note>

以下是一些正確的陣列宣告：

```
int[] score;
float[] data;
double dist[], x; //等於double[] dist; double x;
char[] x,y;       //等於char[] x; char[] y;
```

<note tip> 推薦閱讀int[ ] array vs int array[ ] </note>

9.1.2 初始化

如前述，DataType[] arrayName的宣告，其實並不是陣列的宣告，只是將arrayName宣告為指向DataType陣列的reference，我們可以使用new DataType[size]來真正地將陣列在記憶體中產生。您可以在宣告時順便產生陣列，也可以在宣告後才產生。例如：

```
DataType[] arrayName = new DataType[size];

或

DataType[] arrayName;
...
arrayName = new DataType[size];
```

然後再將陣列的值加以設定，例如搭配for迴圈來完成：

```
int[] data = new int[50];

Scanner sc = new Scanner(System.in);

for(int i=0;i<50;i++)
{
   data[i] = sc.nextInt();
}
sc.close();
```

當然你也可以自己指定數值：

```
int[] data = new int[50];

data[0] = 50;
data[1] = 38;
data[2] = 87;
...
```

```
data[49]=39;
```

您也可以在宣告時，直接給定數值。在這種情況下Java會自動幫你完成new DataType[size]的操作，並將其配置到的記憶體空間儲存於arrayName當中，然後將你給定的初始值放入陣列當中。例如：

```
int[] score = {80, 39, 24, 88, 95, 100, 39};
```

9.1.3 存取元素

要存取陣列內的元素，只要以陣列名稱加上一組中括號，並於其中指定索引值即可，其語法如下：

```
arrayName[indexing_expression]
```

其中indexing_expression是用以索引存取陣列特定元素的運算式，其運算結果必須為整數且小於陣列大小。以score陣列為例，下列都是正確的使用方式：

```
x=score[3];

score[2]=x;


score[x]=5;

score[x]=score[y]+2;

x=score[i>j?i:0];
```

9.1.4 陣列應用

在許多的應用裡，陣列時常都會被使用，且往往會配合迴圈來存取陣列的元素。例如使用for迴圈「從陣列的第一個元素開始，逐一進行資料處理，直到最後一個元素為止」，就是一個非常典型的應用。假設data為一整數陣列，且具有SIZE個元素，其程式如下：

```
for(int i=0;i<SIZE;i++)
{
   // do something for data[i];
}
```

但若是不知道該陣列有多少元素時，又該如何處理？事實上Java語言為陣列提供一個名為length的屬性，

我們可以據以獲得陣列的大小資訊，因此上述程式碼可修改如下：

```
for(int i=0;i<data.length;i++)
{
   // do something for data[i];
}
```

自JDK 1.5起，Java又提供了foreach的語法，供我們簡易地進行陣列的操作foreach是for迴圈的一種衍生語法，可以幫助我們在for迴圈中逐一取得陣列元素並加以處理，其語法可參考下面的說明：

```
for( DataType variableName : arrayName )
{
 // 在迴圈的處理敘述中，我們可以variableName代表目前'拜訪'到的元素
}
```

- for(DataType variableName : arrayName) → 表示要逐一存取arrayName陣列的每個元素
- for(DataType variableName : arrayName) → 每次所取回的元素為DataType型態，並放置於variableName變數中

<note tip>事實上，除陣列外Collection等集合物件也都可以使用foreach的語法，後續會再加以介紹</note>

下面的例子示範以foreach的方式，為陣列的每一個元素讀取使用者輸入的數值：

```
int[] data = new int[50];

Scanner sc = new Scanner(System.in);

for(int element : data)
{
   element = sc.nextInt();
}
```

而下面的例子，則是以foreach的方式將陣列內容印出:

```
for(int element : data)
   System.out.println(element);
```

我們也時常需要取得陣列中的最大值、最小值、數值總和與平均等資訊，請參考下面這個程式：

```
//假設data陣列有50個元素
int max, min, sum=0;
```

```
double average=0.0;
max=min=data[0];

for(int i=0;i<50;i++)
{
   if(max<data[i])
      max = data[i];
   if(min>data[i])
      min = data[i];
   sum+=data[i];
}
average = sum/50;
```

改用foreach後，同一個程式可修改如下：

```
int max, min, sum=0;
double average=0.0;
max=min=data[0];

for(int current : data)
{

   if(max<current)
      max = current;

   min = min>current? current : min;

   sum+=current;
}
average = sum/50;
```

最後，讓我們設計一個樸克牌洗牌的程式：

```
//設定一付撲克牌
// 0-12 spade
// 13-25 heart
// 26-38 diamond
// 39-51 club
// 0 for A, 1 for 2, ..., 8 for 9, 9 for 10, 10 for J, 11 for Q and 12 for K

public class Poker
{
    public static void main(String[] args)
    {
      char[] suits ={'\u2660', '\u2665', '\u2666', '\u2663'};
      int[] cards = new int[52];
      for(int i=0;i<52;i++)
```

```
            cards[i]=i;
        for(int i=0;i<52;i++)
        {
            int p = (int)(Math.round(Math.random()*100))%52;
            int temp = cards[i];
            cards[i]=cards[p];
            cards[p]=temp;
        }
        for(int i=0;i<52;i++)
        {
            System.out.print(suits[cards[i]/13]);
            int p = cards[i]%13;
            switch(p)
            {
                case 0:
                    System.out.print('A');
                    break;
                case 9:
                    System.out.print('T');
                    break;
                case 10:
                    System.out.print('J');
                    break;
                case 11:
                    System.out.print('Q');
                    break;
                case 12:
                    System.out.print('K');
                    break;
                default:
                    System.out.print( p+1 );
            }
            System.out.print(" ");
            if(i%13==12)
                System.out.println();
        }
    }
}
```

其中Math.round() method是將數值捨去小數的部份使其轉換成一個整數Math.random() method則是產生一個介於0.0~1.0間的隨機變數，我們將它乘上100後，就成了介於0.0~100.0間的數值，透過Math.round()取整數後再餘除52，即可得到介於0~51的隨機變數。

將數字0~12所代表的樸克牌點數事先以陣列宣告，則上述程式還可修改如下：

```
public class Poker
```

```
{
    public static void main(String[] args)
    {
      char[] suits ={'\u2660', '\u2665', '\u2666', '\u2663'};
      char[] points={'A', '2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J',
'Q', 'K'};
      int[] cards = new int[52];
      for(int i=0;i<52;i++)
          cards[i]=i;
      for(int i=0;i<52;i++)
      {
          int p = (int)(Math.round(Math.random()*100))%52;
          int temp = cards[i];
          cards[i]=cards[p];
          cards[p]=temp;
      }
      for(int i=0;i<52;i++)
      {
          System.out.print(suits[cards[i]/13]);
          System.out.print(points[cards[i]%13] + " ");
          if(i%13==12)
              System.out.println();
      }
    }
}
```

9.2 多維陣列

除了一維陣列外，Java語言也支援多維度的陣列(Multidimensional Array)。

9.2.1 宣告與初始化

以二維陣列為例，其宣告語法如下：

```
DataType[][] arrayName;

或

DataType arrayName[][];

或

DataType[] arrayName[];
```

上述的三種方式，都說明了一個事實：這是一個reference of references的宣告。假設宣告一個陣列用以記載5個學生的成績，其中每個學生有國文、英文與數學三個科目的成績，有兩種思考的方式：

1. int[][] score = new int[3][5]; *三個科目，每個科目有五個學生的成績 - int[][] score = new int[5][3];* 五個學生，每個學生有三個科目的成績

兩種方式都是正確地，要使用哪一種取決於您慣用的思考方式。我們以第一種方式為例，figure 2顯示了score是一個reference of references意即score是一個reference指向一些references這些所指向的references又指向真正地數值資料。圖中我們特別以黃色標明reference藍色標明數值。

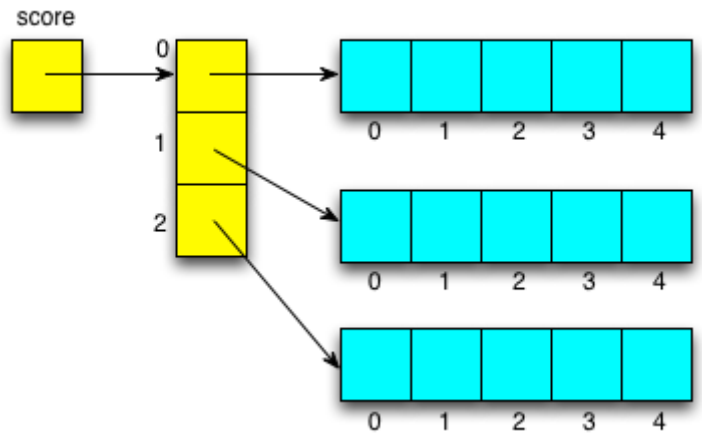

Fig. 2

當然，您也可以在宣告的同時，順便給定初始值：

```
int[][] score = { { 80, 60, 100, 80, 90 }, {100, 60, 90, 80, 75}, {60, 60, 95, 70, 53} };
```

二維陣列的操作通常會使用到雙重迴圈，例如以下的例子將score陣列的內容印出：

```
for(int i=0;i<3;i++)
{
   for(int j=0;j<5;j++)
   {
      System.out.print(score[i][j]);
   }
   System.out.println();
}
```

改用foreach語法，可改寫如下：

```
for(int[] s : score)
{
   for(int t : s)
```

```
    {
        System.out.print(t);
    }
    System.out.println();
}
```

其它更多維度的陣列也是類似的觀念，例如三維陣列是reference of references of references請自行研究。

9.2.2 應用

多維陣列的應用很廣泛，以下是一些例子：

- 成績處理(多個學生、多個科目、多個成績項目等)
- 棋奕遊戲(利用二維陣列來定義棋盤)
  - 井字遊戲(3×3陣列)
  - 五子棋(19×19陣列)
  - ...
- 影像處理
  - 以640×480解析度
  - 每個像素須包含R, G, B數值
  - 每個數值可能為0-255
- 還有更多…