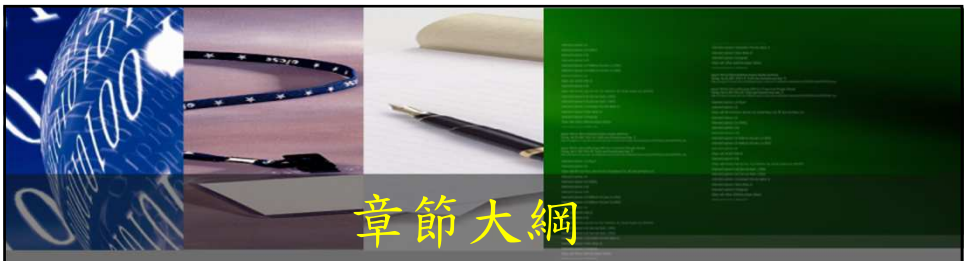
第六章　函式(Method；方法)

課前指引

在設計程式時，我們會將一個較大的程式切割為許多個子功能。我們通常將這些小功能獨立寫成一個「函式」，當程式需要運用該功能時，就可以直接呼叫函式，使得主要程式的長度變短而有助於日後的維護。

在物件導向程式設計中，這些函式(function)被稱為方法(method)，並且隸屬於某一個類別，這些方法又可以分為兩種，一種是可以由類別直接執行的靜態方法(static method)，另一種則是必須由類別產生物件實體後，由物件執行的方法。



章節大綱

- 6.1 認識函式
- 6.2 自行定義函式
- 6.3 好用的亂數函式
- 6.4 引數串列與引數傳遞
- 6.5 回傳陣列
- 6.6 搜尋演算法【補充】
- 6.7 main函式的參數串列
- 6.8 函式的final參數
- 6.9 遞迴函式
- 6.10 多載(overloading)

6.1 認識函式

- 函式在物件導向中，由於隸屬於某一類別，故稱為**成員函式(member function)**，又稱為**方法(method)**。
  - 本書可能以函式、成員函式、方法、成員方法等名詞來解說，其實指的都是method。

3

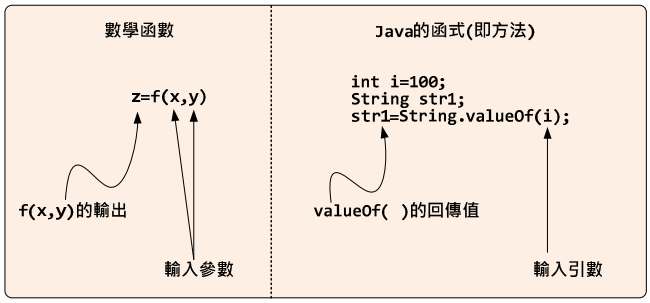
6.1.1什麼是函式

- Java提供的函式功能與數學的函數類似
  - 數學函數
    - 輸入函數的參數，可得到函數的輸出結果。
  - 在Java的函式中
    - 傳遞引數(Argument)給函式處理，經過函式的處理之後，可以獲得一個輸出結果（即函式回傳值）
  - 例如：valueOf()隸屬於String類別
    - 用來將數值轉換為字串的函式。
  - 兩者的比較如下圖示意。

4

6.1.1什麼是函式

圖6-1 數學函數與Java函式比較圖



- 我們必須限制Java函式輸入引數的資料型態，例如：i必須為int型態。
- 我們必須在函式宣告時就定義回傳值的資料型態，也允許函式沒有回傳值。

5

6.1.1什麼是函式

Coding 偷撇步

事實上，還有另一種函數式程式語言（functional programming language）更像數學的函數，例如R語言，這類語言甚至提供了如數學上的合成函數功能。雖然Java並非函數式程式語言，但Java 也在8.0 版提供了lambda運算式來對應這種功能，我們將於後面章節中介紹。

6

6.1.2函式的優點與特性

- 程式語言的函式雖然與數學的函數設計的目的略有不同
  - 程式語言的函式可以視為【一群敘述的集合】，需要使用時，直接呼叫該函式，如此便可以有效重複利用程式碼。
- Java函式的特點整理如下：
  - (1)Java的函式隸屬於某一個類別，除非透過多載(overload)技術，否則不允許宣告兩個相同名稱的函式。
  - (2)函式內宣告的變數為『區域變數』
    - 不同函式內可以使用相同的變數名稱
    - 該變數只會在該函式中生效。

7

6.1.2函式的優點與特性

- (3)函式最好具有特定功能，如此才能夠提高可讀性並有利於除錯與日後的維護。
  - （函式應以它的功能來命名函式，這個功能應該以動作或操作來思考其效用，並且盡可能以動詞或動詞+受詞的方式來命名）
- (4)對於發展大型類別而言，將物件可能使用的功能製作為方法（即函式），可以交由許多程式設計師分工撰寫，如此可以加快類別的開發速度
  - 不過在切割功能及實際撰寫方法之前，必須討論出一定的規格，以免發生不協調的狀況。
- (5)類別內撰寫的函式分為static method與一般method
  - static method內的敘述只能呼叫static method。

8

6.1.3呼叫函式的執行流程

- 直接使用別人已經撰寫好的函式，只要import引入函式所屬之類別即可。
  - 例如：引入java.lang.String類別，就可以直接使用valueOf( )函式，而不必自行撰寫valueOf( )函式。
  - 如果該函式為static，則不用產生物件即可使用。
- 當主程式呼叫函式時
  - 程式的控制權將會轉移到函式開頭處，然後執行函式中的程式碼
  - 函式的程式碼執行完畢後，控制權將重新回到主程式碼（呼叫敘述）的下一個敘述，繼續往下執行。

9

6.1.3呼叫函式的執行流程

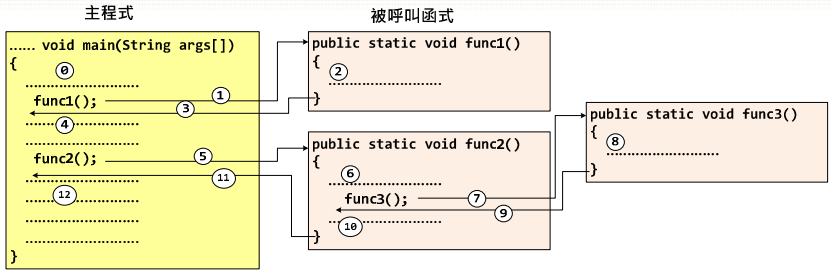


圖6-2 函式呼叫與返回示意圖（程式控制權的轉移）

- 【觀念範例6-1】：藉由範例說明函式呼叫與返回的程式流程控制權之轉移。
  - **範例6-1：**ch6_01.java（隨書光碟myJava\ch06\ch6_01.java）

10

6.1.3呼叫函式的執行流程

```
1   /*  檔名:ch6_01.java       功能:呼叫static method   */
2
3   package myJava.ch06;
4   import java.lang.*;
5   import java.lang.Math;
6
7   public class ch6_01          //主類別
8   {
9       public static void main(String args[])
10      {
11          double i=7.0,j=4.0;
12          double powNum,logNum;
13
14          powNum = Math.pow(i,j);
15          System.out.println(i + "的" + j + "次方=" + powNum);
16          logNum = Math.log10(powNum);
17          System.out.println(powNum + "取10的對數=" + logNum);
18      }
19  }
```

pow,log10所屬類別，由於屬於java.lang.* 類別庫，故可省略。

11

6.1.3呼叫函式的執行流程

- 執行結果：

```
7.0的4.0次方=2401.0
2401.0取10的對數=3.3803921600570273
```

- 範例說明：
  - (1)第5行：引入類別java.lang.Math，該類別中包含有pow( )與log10( )兩個static method。這兩個static method的宣告原型如下：

所屬類別：java.lang.Math
語法：**static double pow(double a, double b)**
功能：回傳a的b次方

所屬類別：java.lang.Math
語法：**public static double log10(double a)**
功能：回傳a的對數（以10為底）

12

6.1.3呼叫函式的執行流程

- 範例說明：
  - (2)第14行：呼叫pow函式，計算i的j次方，回傳值由powNum接收。
  - (3)第16行：呼叫log10函式，計算powNum的對數（以10為底），回傳值由logNum接收。
  - (4)由於第4行已經引入所有java.lang的所有類別，因此，第5行可以省略。事實上，java.lang.Math並不會被引入兩次，因為編譯器會替我們檢查。
  - (5)整個程式的流程如下圖所示。

13

6.1.3呼叫函式的執行流程

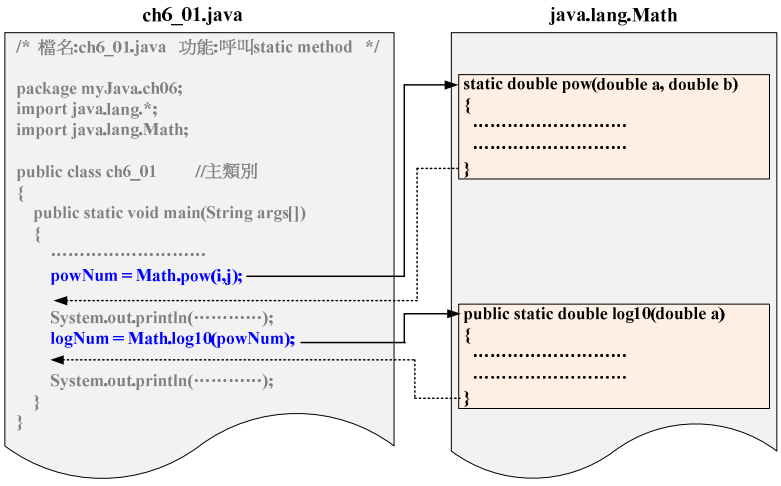


圖6-3 範例6-1的程式流程

14

6.2自行定義函式

- 我們也可以自行發展類別與函式。
  - 因此本章內容以發展主類別的函式為主。
  - 至於其他類別的函式則由下一章開始介紹。

15

6.2.1函式定義

- 在使用函式之前，必須先定義函式內容（也就是函式的定義）。
  - 函式的內容決定了該函式究竟提供了什麼樣的服務。
- 【定義主類別其他函式的語法】

```
[封裝等級] static 回傳值型態  method名稱(資料型態 參數1,資料型態 參數2,……)
{
  ……函式主體（程式碼）……
  [return … ;]
}
```

16

6.2.1函式定義

- 【語法說明】：
  - (1)封裝等級：下一章會進行介紹，此處使用public即可。
  - (2)static關鍵字：static method內的程式碼只能呼叫static method，因此為了要讓主函式main呼叫，故必須宣告為static method。
  - (3)回傳值型態：若函式沒有回傳值，宣告為void。
  - (4)參數群：
    - 1.每宣告一個輸入參數，都必須宣告資料型態，也同時宣告該名稱。而參數的資料型態將會限制住傳入引數之資料型態。
    - 2.命名規則與變數的命名規則相同。
    - 3.輸入參數在函式主體內屬於合法的資料變數。
    - 4.若無引數需要傳遞，則參數列可以為( )。

17

6.2.1函式定義

- (5)return敘述：
  - 1.具有回傳值的函式，在函式主體內應該包含一個以上的return敘述，以便傳回資料。不具回傳值的函式則可以沒有return敘述。
  - 2.具有回傳值的函式，其return後面應該接上與回傳值相容資料型態的常數、變數或運算式。
- (6)【合法的函式定義範例】

函式定義範例	解說

public static void showWelcome(int print_times)
{
for(int a=1;a<=print_times;a++)
System.out.println ("您好,歡迎光臨");
}
	1. 可以在函式主體內使用輸入參數print_times。
	2. 函式無回傳值。

18

6.2.1函式定義

函式定義範例	解說

public static int Mul(int a,int b)
{
int result;
result = a*b;
return result;
}
	1. 可在函式主體內使用輸入參數a,b。
	2. 函式回傳值的資料型態為int。
	3. 使用return回傳資料，result為Mul函式的回傳值。
	4. return敘述執行完畢，控制權將立刻返回原呼叫函式的下一個敘述。

public static double Add(double a,double b)
{
return (a+b);
}
	1.可在函式主體內使用輸入參數a,b。
	2.函式回傳值的資料型態為double。
	3.使用return回傳資料，(a+b)運算式的結果為Add函式的回傳值。
	4.執行完畢return敘述，控制權將立刻返回原呼叫函式的下一個敘述。

19

延伸學習：**引數與參數**

在第2章中，我們提及到Java有四種變數，也就是實體變數**(Instance Variables)**、類別變數**(Class Variables)**、區域變數**(Local Variables)**、**參數(Parameters)**，其中參數與引數是相互對應的。

通常，一個程式語言如果同時出現Parameter與Argument兩詞，則Argument代表的是呼叫方傳遞的值，並且它必須是單向的，也就是不會受到被呼叫方的影響。而被呼叫方宣告的對應變數則稱為Parameter，它是可以改變的值（因為對被呼叫方來說，它是一個變數）。

換句話說，**在Java的函式定義列中宣告的變數稱為參數(Parameter)。而呼叫函式敘述中小括號內出現的叫做引數(Argument)**。在不同場合中，這些名詞有不同的稱呼，因此，您可能在不同的書籍中，看到不同的名稱。我們將之整理如下，**為了避免讀者混淆，本書將只使用引數與參數來區別兩者。**

	Java語言/C語言	程式語言
在呼叫敘述小括號內	引數 (argument)	實參數(actual parameter)
在被呼叫函式定義或宣告第一列	參數(parameter) 實引數(actual argument) 正式引數(formal argument)	正式參數(formal parameter)

在本書中，還有另一個名詞命令列引數，則是代表在Console Mode環境中，使用者輸入給主函式的資料項目。

20

6.2.2函式呼叫

- 函式經由定義後，必須透過**函式呼叫**才能實際應用該函式。
  - 函式呼叫可以視為一種**控制權轉移**的敘述。
    - 當遇到函式呼叫時，控制權將被轉移到被呼叫函式的起始點，並執行該函式的程式碼（即函式定義）
    - 當程式碼被執行完畢後（或遇到return敘述時），將會把控制權再交還給原來發生函式呼叫的程式執行點，繼續執行下一個敘述。

21

6.2.2函式呼叫

- 在Java中，呼叫端與被呼叫端若隸屬同一類別，則呼叫函式的語法如下：
  - 【語法1】（函式無回傳值）：

```
method名稱(傳入引數串列);
```

  - 【語法2】（函式有回傳值）：

```
變數 = method名稱(傳入引數串列);
```

  - 【語法說明】：
    - (1)若無資料需要傳遞，則只需要使用『method名稱();』來呼叫即可。否則，必須一一輸入對應的引數。
    - (2)若函式有回傳值，則可以使用一個相容資料型態的變數來接收這個函式回傳值。

22

6.2.2函式呼叫

- (3)函式呼叫敘述必須與函式宣告的名稱相同，但引數與參數的名稱可以不同。
  - 若呼叫者(Caller;Calling Program)有資料要傳遞給被呼叫者(Callee;Called Program)，則必須藉由傳入引數串列將資料傳遞給函式的參數
  - 並且『傳入引數串列』的傳入變數會由『函式定義的參數串列』的相對參數來接收
  - 順序、個數、資料型態必須相同（不會做自動轉型）但引數名稱可以與參數名稱不同。

23

6.2.2函式呼叫

- 如下圖示意：（圖中3.14必須指定為float型態的3.14f，否則會被視為double，而由於不會自動轉型，故會發生錯誤）

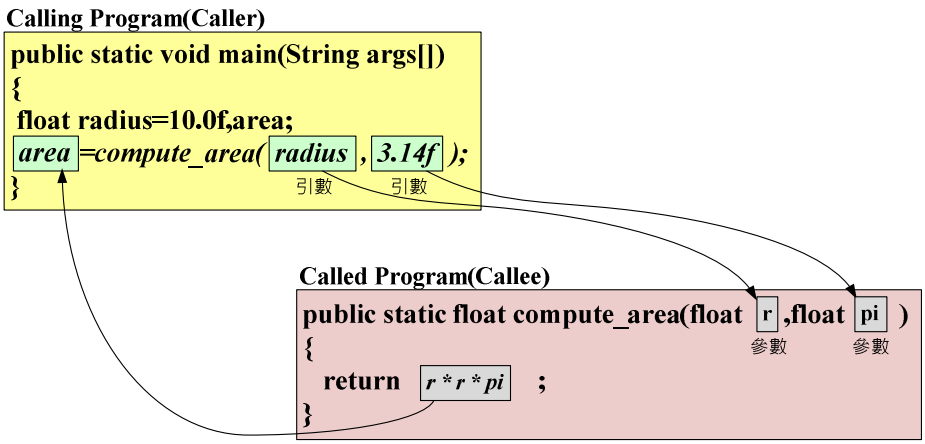


圖6-4 引數與參數的對應

24

6.2.2函式呼叫

- 【觀念範例6-2】：不使用Math類別的pow函式，自行製作一個函式（power函式），功能為計算X^n。（X為double型態、n為int型態）。

```
1   /*  檔名:ch6_02.java       功能:定義,呼叫函式    */
2
3   package myJava.ch06;
4   import java.lang.*;
5   import java.util.Scanner;
6
7   public class ch6_02          //主類別
8   {
9       public static void main(String args[])
10      {
11          int k;
12          double Ans;
13          Scanner keyboardInput = new Scanner(System.in);
14
15          System.out.print("計算3.5的k次方?請輸入k=");
16          k = Integer.parseInt(keyboardInput.nextLine());
17          Ans = power(3.5,k);
18          System.out.println("3.5的" + k + "次方=" + Ans);
19      }
```

函式呼叫

6.2.2函式呼叫

```
20
21      public static double power(double X,int n)
22      {
23          int i;
24          double powerXn=1;
25          for(i=1;i<=n;i++)
26              powerXn = powerXn * X;
27          return powerXn;
28      }
29  }
```

函式定義

- **執行結果：**

```
計算3.5的k次方?請輸入k=5
3.5的5次方=525.21875
```

6.2.2函式呼叫

- 範例說明：
  - (1)第21~28行：power函式的定義，用來計算X^n。回傳值的資料型態是double，接受兩個傳入引數，資料型態分別是double, int，而第27行return後面的變數powerXn是回傳值。
  - (2)第17行：呼叫power函式
    - 傳入的引數為3.5, k，與函式定義的參數名稱不相同，只要傳入符合資料型態的數值或變數即可
    - （您也可以將引數與參數名稱設為相同，但這兩個仍是不同的變數，我們將在後面介紹傳值呼叫時詳加說明）。
    - 使用Ans變數來存放函式回傳值。

27

6.2.2函式呼叫

- (3)函式呼叫之引數傳遞與回傳值如下圖。

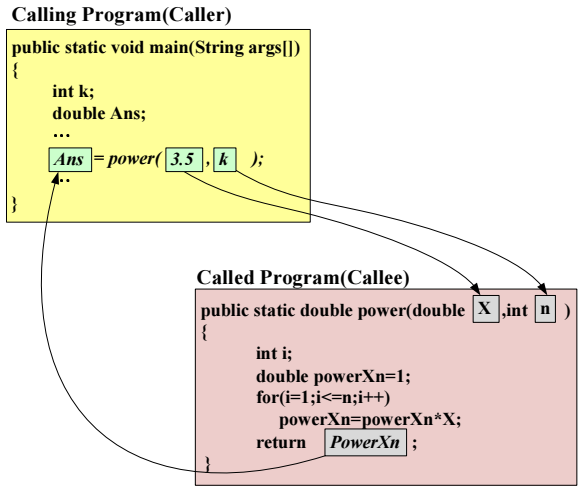


圖6-5 範例6-2的函式呼叫與回傳值

28

6.2.2函式呼叫

- 【實用及觀念範例6-3】：製作三個函式（Odd、Even、TotalSum函式），功能分別為計算奇數和、偶數和、整數和。（其中的整數和請使用奇數和與偶數和之函式）。
  - **範例6-3：**ch6_03.java（隨書光碟 myJava\ch06\ch6_03.java）

```
1  /*   檔名:ch6_03.java        功能:函式應用     */
2
3  package myJava.ch06;
4  import java.lang.*;
5  import java.util.Scanner;
6
7  public class ch6_03          //主類別
8  {
9      public static void main(String args[])
10     {
11         int n,sum;
12         char addChoice;
13         Scanner keyboardInput = new Scanner(System.in);
14
15         System.out.print("1+2+...+n=?請輸入n=");
16         n = Integer.parseInt(keyboardInput.nextLine());
17         System.out.print("請問要做奇數和(O),偶數和(E),還是整數和(I)?");
18         System.out.print("請選擇:");
19         addChoice = keyboardInput.nextLine().charAt(0);
20
```

29

6.2.2函式呼叫

```
21
22         switch(addChoice)
23         {
24             case 'O': sum = odd(n);
25                       break;
26             case 'E': sum = even(n);
27                       break;
28             case 'I': sum = totalSum(n);
29                       break;
30             default:  System.out.println("選擇錯誤");
31                       return;
32         }
33         System.out.println("總和為" + sum);
34     }
35
36     public static int odd(int U)
37     {
38         int i,total=0;
39         for(i=1;i<=U;i++)
40             if(i%2 == 1)
41                total = total + i;
42         return total;
43     }
```

odd函式的定義，用來計算1+3+…+U的奇數和。（total是回傳值）

30

6.2.2函式呼叫

```
44      public static int even(int U)
45      {
46          int i,total=0;
47          for(i=1;i<=U;i++)
48              if(i%2 == 0)
49                  total = total + i;
50          return total;
51      }
52
53      public static int totalSum(int U)
54      {
55          return odd(U) + even(U);
56      }
57  }
```

even函式的定義，用來計算2+4+…+U的偶數和。（total是回傳值）

透過呼叫其他函式來完成這個函式的功能，可提高程式碼的再利用性

- **執行結果：**

```
1+2+...+n=?請輸入n=10
請問要做奇數和(O),偶數和(E),還是整數和(I)?請選擇:I
總和為55
```

31

6.2.2函式呼叫

- 範例說明：
  - (1)第53~56行：totalSum函式的定義，用來計算1+2+…+U的整數和。其中呼叫了odd及even函式，幫忙做奇數和與偶數和，合起來就是整數和（函式應盡量重覆使用）。
  - (2)第21~31行：依據不同的選擇，決定呼叫不同的函式，完成不同的功能。
  - 請特別注意，當使用者輸入非「O」、「E」、「I」時，將會執行default的程式，使用return結束main( )的執行，此時第32行不會被執行

```
1+2+...+n=?請輸入n=10
請問要做奇數和(O),偶數和(E),還是整數和(I)?請選擇:A
選擇錯誤
```

32

6.2.2函式呼叫

- (3)假設我們輸入的是『10』與『0』，則整個程式的執行流程如下：

```
public static void main(String args[])
{
    int n,sum;
    char addChoice;
    Scanner keyboardInput = new Scanner(System.in);

    System.out.print("...");
    n = Integer.parseInt(keyboardInput.nextLine());
    System.out.print("...");
    System.out.print("請選擇:");
    addChoice = keyboardInput.nextLine().charAt(0);

    switch(addChoice)
    {
       case 'O': sum = odd(n);
                 break;
       case 'E': sum = even(n);
                 break;
       case 'T': sum = totalSum(n);
                 break;
       default: System.out.println("選擇錯誤");
                return;
    }
    System.out.println("總和為" + sum);
}
```

```
public static int odd(int U)
{
    int i,total=0;
    for(i=1;i<=U;i++)
       if(i%2 == 1)
          total = total + i;
    return total;
}
```

圖6-6 範例6-3的程式執行流程（輸入『10』與『O』）

6.2.2函式呼叫

- 【實用範例6-4】：製作階層函式（factorial函式），功能為計算某一正整數的階層k!。並利用該函式求出C_n^m 的值，m、n為任意正整數，$C_n^m = \frac{m!}{n!(m-n)!}$ 。
  - **範例6-4：**ch6_04.java（隨書光碟myJava\ch06\ch6_04.java）

```
1   /*  檔名:ch6_04.java        功能:函式應用    */
2
3   package myJava.ch06;
4   import java.lang.*;
5   import java.util.Scanner;
6
7   public class ch6_04          //主類別
8   {
9       public static void main(String args[])
10      {
11          int m,n;
12          long ans;
13          Scanner keyboardInput = new Scanner(System.in);
14          long temp[]=new long[3];
```

執行結果

```
求排列組合C(m,n)
m = 10
n = 8
C(10,8)=45
```

6.2.2函式呼叫

```
16          System.out.println("求排列組合C(m,n)");
17          System.out.print("m = ");
18          m = Integer.parseInt(keyboardInput.nextLine());
19          System.out.print("n = ");
20          n = Integer.parseInt(keyboardInput.nextLine());
21
22          temp[0] = factorial(m);      //  計算 m! 的值
23          temp[1] = factorial(n);      //  計算 n! 的值
24          temp[2] = factorial(m-n);    //  計算 (m-n)! 的值
25          ans = (temp[0])/(temp[1]*temp[2]); //C(m,n)= (m!)/(n!*(m-n)!)
26          System.out.println("C(" + m + "," + n + ")=" + ans);
27      }
28
29      public static long factorial(int p)  /* 函式定義 */
30      {
31          long result = 1L;
32
33          for(int count=1;count<=p;count++)
34              result = result * count;
35          return result;
36      }
37  }
```

- 在這個範例中，factorial()函式一共被呼叫了3次（第22、23、24行），充分利用函式，提高程式碼的重複使用率。

35

6.2.2函式呼叫

- 【觀念範例6-5】：製作一個專門用來列印九九乘法表的函式（print99函式），該函式不接受任何傳入引數，也不回傳任何資料。
  - **範例6-5：**ch6_05.java（隨書光碟myJava\ch06\ch6_05.java）

```
1   /*   檔名:ch6_05.java       功能:無傳入引數及回傳值的函式     */
2
3   package myJava.ch06;
4   import java.lang.*;
5
6   public class ch6_05          //主類別
7   {
```

36

6.2.2函式呼叫

```
8     public static void main(String args[])
9     {
10       print99();   // 函式呼叫 */
11    }
12
13    public static void print99()  // 函式定義
14    {
15       for(int i=1;i<=9;i++)
16       {
17         for(int j=1;j<=9;j++)
18             System.out.print(i + "*" + j + "=" + i*j + "\t");
19         System.out.println();
20       }
21    }
22 }
```

雖然沒有引數，但仍須寫上()。

雖然沒有參數，但仍須寫上()。

- 執行結果：（同範例4-17）
  - 範例說明：
  - (1)第13行，由於沒有傳入引數，因此參數串列為空。由於函式不需要回傳值，所以設為void。
  - (2)第10行為函式呼叫，雖然不必傳入引數，但( )仍不可省略。
  - (3)print99()函式中，並無return敘述，所以函式會執行到最後一行（第21行），然後才返回。

37

6.2.3 return敘述

- return敘述一共有2個功用：
  - (1)回傳函式資料
  - (2)函式返回。
- 使用return回傳資料的語法如下：

```
return 常數、變數、運算式或其他具有結果值的敘述；
```

38

6.2.3 return敘述

- 【語法說明】：
  - (1)回傳函式資料的資料型態必須和函式定義的回傳值資料型態相容（需可進行自動轉型）。
    - 範例：

```
int func1(……)
{
  int a;
  ……
  return a;
}
```

```
double func2(……)
{
  float c=3.0f;
  ……
  return c;
}
```

  - (2)函式回傳值資料型態若被宣告為void，則不用return回傳值。
  - (3)若回傳值為運算式或其他具有結果值的敘述，則會先計算其運算結果，然後才傳回該值。

39

6.2.3 return敘述

- 使用return返回
  - 使用return返回呼叫函式敘述如下：

```
return;
```

或

```
return  函式回傳值;
```

  - 【語法說明】：
    - (1)使用return將返回呼叫函式處，並由呼叫函式的下一個敘述開始執行。
    - (2)return並不限定為一個，一但執行return敘述後，其餘函式內未執行敘述不會被執行。
    - (3)無回傳值的函式，不需要用return敘述返回，此時函式將被執行完畢後，才會返回呼叫函式處。
  - 【觀念範例6-6】：設計一個包含有多個return敘述的函式，觀察其執行過程。
    - 範例6-6：ch6_06.java（隨書光碟 myJava\ch06\ch6_06.java）

40

6.2.3 return敘述

```
1   /*  檔名:ch6_06.java       功能:return返回    */
2
3   package myJava.ch06;
4   import java.lang.*;
5
6   public class ch6_06          //主類別
7   {
8       public static void main(String args[])
9       {
10          int k;
11
12          k=funcl();
13          System.out.println("k=" + k);
14      }
15
16      public static int funcl()
17      {
18          int a=5,b=7;
19          a++;
20          if (a>0) return a+b;
21          a++;
22          if (a>0) return a+b;
23          a++;
24          return a+b;
25      }
26  }
```

執行結果

```
k=13
```

41

6.2.3 return敘述

- 範例說明：
  - (1)程式執行的行數順序為9、10、12（函式呼叫）、17、18、19、20（返回）、13、(14)。亦即程式的21~24行將不會被執行。
  - (2)執行第12行時，會先計算a+b的值（6+7=13），然後回傳13給呼叫函式的敘述，並返回函式呼叫處，因此k為13。

Coding 注意事項

Java的編譯器頗為聰明，如果您刪除第20行與第22行的if(a>0)，僅留下return a+b;，則它會指出第21行執行不到，提醒程式設計師，程式可能有某些邏輯上的錯誤。但若如本範例，加上一些條件判斷後，它就不一定能夠找出程式邏輯上的錯誤。

42

6.2.3 return敘述

Coding 偷撇步

如果您將第24行改寫如第20行，則Java的編譯器會顯示出找不到return的錯誤（因為該函式的宣告處已經註明要回傳一個int型態的回傳值），雖然這樣子的修改，程式實際上還是會回傳一個值，但Java的編譯器並沒有那麼聰明。此時，可以在最後加入return 0;之類的敘述以避開編譯錯誤，不過您必須先確定該行絕對不可能被執行到。

43

6.3好用的亂數函式

- 在程式設計中，我們可以使用亂數函式來模擬大量的隨機資料，例如統計與實驗、電腦遊戲、樂透開獎等等。
  - 若使用他人完成且可信賴的函式將可縮減程式的開發時程
    - 取亂數的函式，只要使用 java.lang.Math類別所提供的random( )函式即可。

44

6.3 好用的亂數函式

- random( )【取亂數】

所屬類別：java.lang.Math
語法：**public static double random()**
功能：回傳一個介於0.0~1.0（不含1.0）的亂數

- 【語法說明】：
  - 回傳值為隨機產生的一個double浮點數。
- 【觀念範例6-7】：使用Math.random()亂數函式產生6個隨機亂數。
  - **範例6-7：**ch6_07.java（隨書光碟myJava\ch06\ch6_07.java）

45

6.3 好用的亂數函式

```
1   /*  檔名:ch6_07.java        功能:亂數函式random  */
2
3   package myJava.ch06;                    random所屬類別,可省略
4   import java.lang.*;
5   import java.lang.Math;
6
7   public class ch6_07          //主類別
8   {
9       public static void main(String args[])
10      {
11        for(int i=1;i<=6;i++)
12           System.out.println("第" + i + "個隨機亂數為" + Math.random());
13      }
14  }
```

- 執行結果：

```
第1個隨機亂數為0.5324784887116961
第2個隨機亂數為0.02382472609536701
第3個隨機亂數為0.45666998933475733
第4個隨機亂數為0.1490267800925611
第5個隨機亂數為0.3327052029025074
第6個隨機亂數為0.3953424926107535
```

46

6.3 好用的亂數函式

- 範例說明：
  - (1)迴圈執行6次，使用random函式產生6個隨機亂數。每次重新執行本範例，將會得到隨機的6個亂數。（通常是不同的6個亂數）
  - (2)由於產生的亂數是一個小數，進行乘法或加法，然後透過型態轉換可取得應用所需要的整數亂數值。

- 【實用範例6-8】：使用亂數函式產生6個1~49的整數，並存放於整數陣列中（不要求數字不重覆）。
  - **範例6-8：**ch6_08.java（隨書光碟 myJava\ch06\ch6_08.java）

47

6.3 好用的亂數函式

```
1  /*  檔名:ch6_08.java          功能:產生6個1~49的隨機亂數  */
2
3  package myJava.ch06;
4  import java.lang.*;
5  import java.lang.Math;
6
7  public class ch6_08
8  {
9      public static void main(String args[])
10     {
11         int lottos[]=new int[6];
12
13         for(int i=1;i<=6;i++)
14         {
15             lottos[i-1] = (int)((Math.random()*49)+1);
16             System.out.println("第" + i + "個隨機亂數為" + lottos[i-1]);
17         }
18     }
19 }
```

轉換為整數資料型態

執行結果

```
第1個隨機亂數為13
第2個隨機亂數為27
第3個隨機亂數為14
第4個隨機亂數為14
第5個隨機亂數為34
第6個隨機亂數為35
```

- 由於Math.random()*49可產生的亂數將介於0.0~49.0（不含49.0），故將之加1後為1.0~50.0（不含50.0），強制型態轉換為int整數類型，則小數部分會被去除，就可以得到『1~49』的亂數。
- （在後面，將修改為6個數字不重複，並加上特別號，符合真正的樂透開獎）

48

6.4引數串列與引數傳遞

- 當需要回傳超過一個回傳值的時候，該怎麼辦呢？基本上，大多數的程式語言都是靠『傳址呼叫』，藉由共用同一塊記憶體來達到目的。
  - 同樣的問題若遇到物件導向程式語言時，解法就更多了。
  - 基本上，物件導向程式語言有兩個方法可以解決這個問題
    - 第一種方法是使用『欄位』（例如將資料宣告為類別的欄位，則類別內的所有函式都可以取用），但是這個方法並非適用於所有場合，因為除了呼叫方與被呼叫方可以使用這些欄位外，類別內的其他函式也可以使用這些欄位。

49

6.4引數串列與引數傳遞

- 第二種方法是利用物件來達成。
  - 利用物件來達成又分為回傳一個物件與傳遞一個物件兩種方式。
    - 回傳一個物件是把所有想要回傳的資料塞進那個物件中（由於各位讀者尚未具備自行宣告類別的能力，因此本章暫且不說明此法）。
    - 傳遞一個物件則是在未提供傳址呼叫的情況下，利用物件實體的參考（這也是一個位址）來達成目的。

50

6.4引數串列與引數傳遞

- 在Java中，呼叫端與被呼叫端的串列分別稱為『引數串列』與『參數串列』
  - 如下圖中的a, b為引數，x, y為參數。

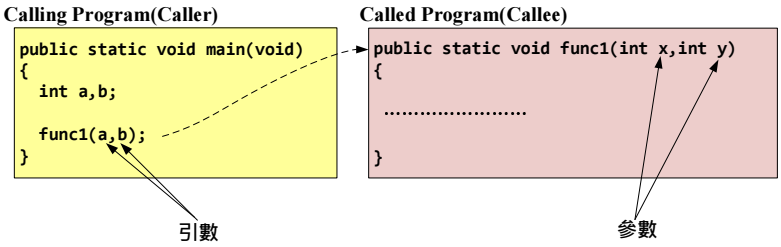


圖6-7 引數與參數

51

6.4引數串列與引數傳遞

- 關於資料傳遞，一般程式語言將之分為傳值呼叫(Call by value)與傳址呼叫(Call by address)兩種，其個別意義如下：
  - 傳值呼叫：
    - 效果：接收端（被呼叫端）如何改變參數值，都不會影響傳遞端（呼叫端）的引數值。
    - 作法：只將引數的值傳遞給參數，亦即在被呼叫端建立一個複本來接收此值，因此引數與參數使用不同的記憶體空間，故而各自獨立，不互相影響。
  - 傳址呼叫：
    - 效果：接收端（被呼叫端）若改變參數值，則傳遞端（呼叫端）的引數值也會跟著改變。
    - 作法：將引數的位址傳遞給參數，使得引數與參數使用了相同的記憶體空間來存放內容，故而參數值一改變（該記憶體內容被改變），引數內容就會跟著改變，彼此互相影響。

52

6.4引數串列與引數傳遞

- 當被呼叫端有一個以上的資料想要和呼叫端溝通時，程式語言必須提供「傳址呼叫功能」或「類似傳址呼叫效果」的機制。
  - 有些高階語言直接提供了傳址呼叫功能
    - 直接提供傳址呼叫(Call by address)功能的程式語言
      - （例如C++）
      - 大多把傳址呼叫以傳參考呼叫(Pass by reference）來實現。
  - 有些高階語言則只提供了類似的功能。
- 而傳值呼叫由於可以避免邊際效應（亦即呼叫者與被呼叫者並不相互影響）
  - 因此，所有的高階程式語言都直接提供了此一機制
    - 有些（例如VB.NET）還被設定為預設機制。

53

6.4引數串列與引數傳遞

- **對於上述兩種傳遞引數的機制，Java僅支援其中的傳值呼叫(Call by Value)**
- 在Java中將之稱為**傳值呼叫(Pass by value)**
  - 但為了共用同一塊記憶體，因此Java對於傳址呼叫，則是利用傳『參考值』來達成類似的效果。
    - 當然有些書籍將Java傳遞物件或陣列時，視為傳參考呼叫(Pass by reference)，但這是不正確的說法
    - Java傳遞物件或陣列時，嚴謹的說法應該是傳參考值呼叫(Pass by value of reference)。

54

6.4引數串列與引數傳遞

筆者的話

每年在程式設計討論區都會引起爭議的兩個問題是(1)Java是否支援傳參考呼叫(Pass by reference)？(2)Java到底有沒有像C語言的指標？

關於Java究竟有沒有**傳參考呼叫(Pass by reference)**機制？嚴格來說，Java確實只有**傳值呼叫(Pass by value)**，只不過這個值在引數為原始資料型態時，是一個數值、字元或布林值，而當引數為陣列或物件時，它將會是一個參考(reference)，這是因為陣列名稱與物件名稱原本就只是個參考，而非實體。

少數書籍對這個問題有明確的說明，以O'Reilly's Java in a Nutshell by David Flanagan一書為例，它的原文解釋如下：（我們將於介紹物件時，透過一個範例研究這個問題）

"Java manipulates objects 'by reference', but it passes object references to methods 'by value'." As a result, you cannot write a standard swap method to swap objects.

55

6.4引數串列與引數傳遞

筆者的話

關於Java到底有沒有像C語言的指標？答案是有的，Java的參考就是C語言的指標；不過它和C語言不同之處在於，Java不可以對參考進行加減法任意移動所指向的記憶體位址，但可以將它指向另一個合法物件實體的記憶體位址。事實上，這個問題與前一個問題有高度相關，Java的傳參考值呼叫(Pass by value of reference)事實上就是C語言的傳指標呼叫(Pass by pointer)！您或許會問，C語言的傳指標呼叫不就是傳址呼叫(Call by address)嗎？那Java不也支援了傳址呼叫(Call by address)了？事實上不是這樣的。其實，不論是C語言的傳指標呼叫或Java的傳參考值呼叫都屬於傳值呼叫(Call by value)，會有這樣的誤解，代表您可能對於C語言的傳指標呼叫有錯誤的認知。有興趣的讀者，可以參閱筆者所著之C語言初學指引第四版的說明。由於本書非講解C語言的書籍，因此，在此僅提示C語言傳遞陣列是傳遞一個指標常數，能夠將引數指定為常數，必定不是傳址呼叫(Call by address)，因為傳址呼叫的被呼叫方參數，可以影響呼叫方的引數，而常數是不允許被改變的，故而只要是傳遞常數類的引數，必定使用的是傳值呼叫(Call by value)。

56

6.4引數串列與引數傳遞

筆者的話

針對Java究竟有沒有**傳參考呼叫(Pass by reference)** 機制的最終解答，筆者認為是沒有。至於在某些Java文件中出現的reference一詞，與C++的reference在定義上，是有些許不同的，它更像是C 語言的pointer，只不過它只接受設定運算，不接受加減法運算而已。

57

6.4.1傳值呼叫(Pass by value)

- Java的**傳值呼叫(Pass by value)**
  - 傳值呼叫就是在呼叫函式時，只會將引數『數值』傳遞給函式中相對應的參數，作為函式啟動時的初始值
  - 參數實際上會產生一個引數的複本，呼叫方的引數與被呼叫方的參數將佔用不同的記憶體空間
  - 不論被呼叫函式在執行過程中如何改變參數的變數值，都不會影響原本呼叫方引數的變數值，這種引數傳遞方式稱為「傳值呼叫」。

58

6.4.1傳值呼叫(Pass by value)

- 在Java中，如果傳遞的引數是原始資料型態的變數（例如int, float, char, boolean）
  - 則採用傳值呼叫傳送，亦即只把『值』傳送過去，並沒有把記憶體位址也傳送過去。
- 我們藉由下面一個範例重新闡述何謂**傳值呼叫(Pass by value)**。
  - 【觀念範例6-9】：透過觀察變數內容，了解傳值呼叫的引數傳遞原理。
    - **範例6-9：**ch6_09.java（隨書光碟myJava\ch06\ch6_09.java）

59

6.4.1傳值呼叫(Pass by value)

```
1  /*   檔名:ch6_09.java       功能:傳值呼叫   */
2
3  package myJava.ch06;
4  import java.lang.*;
5
6  public class ch6_09
7  {
8      public static void main(String args[])
9      {
10         int m=1,n=1;
11
12         func1(m,n);
13         System.out.println("main( )的m=" + m);
14         System.out.println("main( )的n=" + n);
15     }
16
17     public static void func1(int a,int b)
18     {
19         a = a + 10;
20         b = b + 100;
21         System.out.println("func1()的a=" + a);
22         System.out.println("func1()的b=" + b);
23     }
24 }
```

執行結果

```
func1()的a=11
func1()的b=101
main( )的m=1
main( )的n=1
```

60

6.4.1傳值呼叫(Pass by value)

- 範例說明：
  - (1)當第12行呼叫func1函式後，不論func1函式如何運算，影響到的變數只有a,b，而不會影響呼叫它的m,n變數，因為兩者的記憶體位置不同。
  - (2)本範例的引數傳遞如圖：（引數傳遞完畢後，參數與引數就相互不干擾）

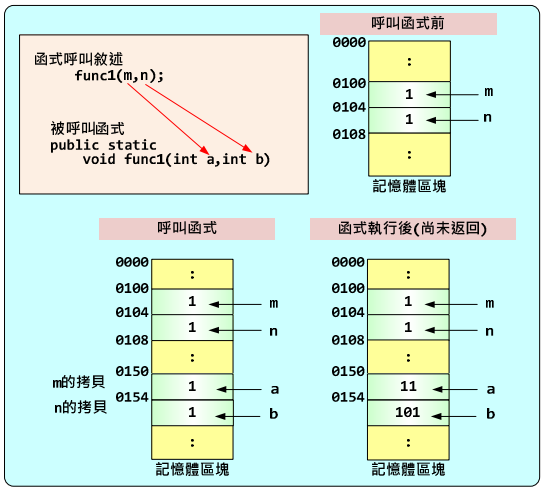


圖6-8 傳值呼叫示意圖

61

6.4.1傳值呼叫(Pass by value)

- (3)本範例即使將main()函式的變數m,n也命名為a,b，仍舊不會影響引數傳遞的過程與結果，因為它們仍將佔用不同的記憶體位置。
- (4)由於a,b是func1的區域變數，故當函式返回後，上圖中a,b所佔用的記憶體將被釋放。

小試身手6-1

請將範例6-9的第10行改為final int m=1,n=1;，然後重新編譯與執行，證明傳值呼叫可將引數設定為不可變動之變數。

小試身手6-2

請在範例6-9的第12~13行間，加入func1(2,4);敘述，然後重新編譯與執行，證明傳值呼叫可將引數設定為常數。

62

6.4.2 傳遞陣列

- 傳遞陣列
  - 將參數的資料型態設定為陣列，並且維度必須正確
    - 傳遞一維整數陣列應該宣告為int[]
    - 傳遞二維整數陣列應該宣告為int[][]，依此類推。
- 當傳遞的引數被宣告物件或陣列時
  - 由於列名稱本身只是一個實體的參考(reference)，故它只會將該參考的值傳遞給被呼叫端相對應的參數
  - 被呼叫端函式可以透過這個參考修改原呼叫端的物件或陣列實體內容。
    - 有些書籍將這種引數傳遞視為傳參考呼叫(pass by reference)，但其實這並不精確
    - 本書將傳遞陣列或物件視為傳「參考值」呼叫(pass by value of reference)。

63

6.4.2 傳遞陣列

- 我們透過下面這個範例，觀察傳遞陣列時的引數傳遞原理。
- 【觀念範例6-10】：利用傳「參考值」呼叫傳遞陣列。
  - **範例6-10：**ch6_10.java（隨書光碟myJava\ch06\ch6_10.java）

```
1   /*   檔名:ch6_10.java          功能:傳參考值呼叫(傳遞陣列)   */
2
3   package myJava.ch06;
4   import java.lang.*;
5
6   public class ch6_10          //主類別
7   {
8       public static void main(String args[])
9       {
10          int lottos[]=new int[6];
11
```

64

6.4.2 傳遞陣列

```
12          generate_lottos(lottos);
13          System.out.println("樂透號碼如下.....");
14          for(int i=0;i<6;i++)
15              System.out.print(lottos[i] + "\t");
16      }
17
18      public static void generate_lottos(int[] arr)
19      {
20         for(int i=0;i<arr.length;i++)
21         {
22               arr[i] = (int)((Math.random()*49)+1);
23             System.out.println("第" + (i+1) + "個隨機亂數為" + arr[i]);
24         }
25      }
26  }
```

- 執行結果：

```
第1個隨機亂數為31
第2個隨機亂數為15
第3個隨機亂數為21
第4個隨機亂數為32
第5個隨機亂數為39
第6個隨機亂數為16
樂透號碼如下.....
31      15      21      32      39      16
```

65

6.4.2 傳遞陣列

- 範例說明：
  - (1)從執行結果中，我們可以得知lottos陣列與arr陣列都參考到同一個陣列實體，所以在generate_lottos函式中對arr陣列元素的修改都將會影響main函式的lottos陣列元素值（共用同一塊記憶體空間）。
  - (2)第18行的參數也可以修改為int arr[]，只要是正確的陣列變數宣告方式即可。
  - (3)事實上，當呼叫generate_lottos函式，記憶體的變化如下：

小試身手6-3

請在範例6-10第10行改為final int lottos[]=new int[6];，然後重新編譯與執行，證明傳遞陣列時，雖然是傳遞參考並且改變了陣列內容，但陣列參考並不會被改變，因為我們已經宣告為final陣列變數，因此lottos不會再參考到其他的陣列實體。

66

6.4.2 傳遞陣列

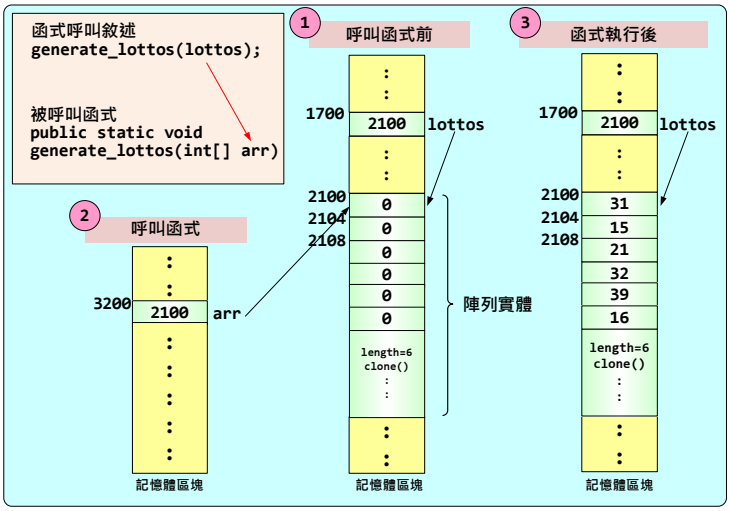


圖6-9 傳遞陣列示意圖（傳參考值呼叫）

67

6.5 回傳陣列

- 既然陣列變數是一個參考，當我們要回傳陣列時，是否需要回傳整個陣列實體，還是只要回傳變數（參考）即可。我們透過下面這個範例來說明：
  - 【觀念範例6-11】：改寫範例6-10，將產生陣列實體敘述放置於被呼叫函式內，並回傳該陣列。
    - **範例6-11：**ch6_11.java（隨書光碟 myJava\ch06\ch6_11.java）

```
1  /*   檔名:ch6_11.java        功能:回傳陣列    */
2
3  package myJava.ch06;
4  import java.lang.*;
```

68

6.5 回傳陣列

```
5
6   public class ch6_11          //主類別
7   {
8       public static void main(String args[])
9       {
10          int lottos[];
11
12          lottos = generate_lottos();
13          System.out.println("樂透號碼如下.....");
14          for(int num : lottos)
15              System.out.print(num + "\t");
16      }
17
18      public static int[] generate_lottos()
19      {
20          int arr[]=new int[6];
21          for(int i=0;i<arr.length;i++)
22          {
23              arr[i]= (int)((Math.random()*49)+1);
24              System.out.println("第" + (i+1) + "個隨機亂數為" + arr[i]);
25          }
26          return arr;
27      }
28  }
```

69

6.5 回傳陣列

- 執行結果：（同範例6-10，但因亂數緣故，值可能不同）
- 範例說明：
  - (1)第18行的回傳值資料型態為int[]，代表要回傳一個整數一維陣列。
  - (2)從執行結果中，我們可以得知lottos陣列變數可以存放arr所回傳的陣列參考，使得將之指向同一個陣列實體
  - 然而，我們在前面提過，在函式內宣告的為區域變數，而arr也是一個區域變數，這意味著當函式執行完畢時，它將被釋放。但釋放的只是陣列的參考而非陣列實體，因此在函式內產生的陣列實體仍可以保留，並由第15行取出其元素。

70

6.5 回傳陣列

- (3)萬一該函式在宣告時並不回傳陣列變數，那麼當函式執行完畢時，陣列參考被釋放了，而陣列實體並未被釋放，可是又沒有任何參考變數可以指向它，那麼陣列實體豈不成為記憶體中的垃圾而浪費記憶體空間嗎？
  - 沒錯，但您不用煩惱，因為Java提供記憶體垃圾收集(Garbage Collection)機制，對於沒有被任何變數參考的陣列或物件實體，JVM將會定期地自動清除。
  - 我們將於後面章節說明Java的Garbage Collection機制。
- (4)本範例的記憶體變化如下：

71

6.5 回傳陣列

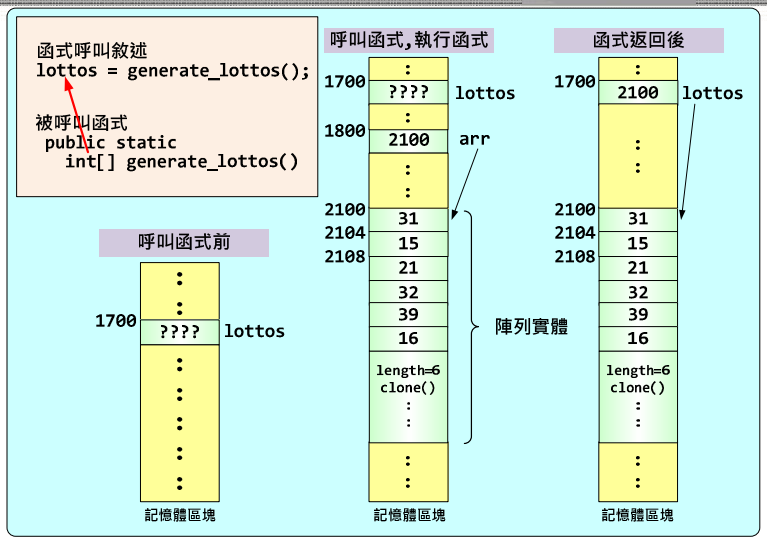


圖6-10 回傳陣列示意圖

72

6.6搜尋演算法【補充】

- 排序的主要目的通常是方便於搜尋資料，搜尋的方法分成許多種，每一種的難度與效率皆不相同，以下是兩種常用的搜尋法：
  - 1. 循序搜尋法
  - 2. 二分搜尋法

73

6.6搜尋演算法【補充】

- 循序搜尋法
  - 『循序搜尋』是從第一筆資料開始尋找，然後是第二筆資料、、、一直到找到所要的資料或全部資料被找完為止。
  - 因此，假設有N筆資料，則最差需要作N次比較，而平均則需要N/2次比較。
  - 通常，我們會在資料量比較少或資料未經排序的狀況下使用『循序搜尋法』。
  - 【實用範例6-12】：使用循序搜尋法在未排序的資料中，尋找所需要的資料『57』。
    - **範例6-12：**ch6_12.java（隨書光碟myJava\ch06\ch6_12.java）

74

6.6搜尋演算法【補充】

```
1   /*   檔名:ch6_12.java       功能:循序搜尋法  */
2
3   package myJava.ch06;
4   import java.lang.*;
5   import java.util.Scanner;
6
7   public class ch6_12          //主類別
8   {
9       public static void main(String args[])
10      {
11          int workArr[]={43,23,67,27,39,15,39,37,57,26,14};
12          int findNum,location;
13          Scanner keyboardInput = new Scanner(System.in);
14
15          System.out.print("請輸入您要找的數值:");
16          findNum = Integer.parseInt(keyboardInput.nextLine());
17          location = seqSearch(findNum,workArr);
18          if(location==-1)
19             System.out.println("在陣列中找不到要找的數值");
20          else
21             System.out.println("數值" + findNum +
                                    "位於work[" + location + "]");
22      }
23
```

執行結果

```
請輸入您要找的數值:57
數值57位於work[8]
```


6.6搜尋演算法【補充】

```
24      public static int seqSearch(int target,int[] arr)
25      {
26          for(int i=0;i<arr.length;i++)
27             if(target == arr[i])       // 找到了
28                return i;
29          return -1;                  // 完全找不到
30      }
31  }
```

- 範例說明：
  - (1)第24~30行是循序搜尋函式。可以接受一個尋找目標（整數int資料型態）及一個工作陣列。語法如下：

所屬類別：主類別
語法：**public static int seqSearch(int target,int[] arr)**
功能：循序搜尋。
引數：target為尋找目標。arr[ ]為工作陣列。
回傳值：若target位於arr[ ]陣列中，則回傳索引值。若不位於arr[ ]陣列中，則回傳-1。

  - (2)第17行使用『傳值呼叫』傳遞findNum→target參數。使用『傳參考值呼叫』，傳送workArr →arr陣列變數。

6.6搜尋演算法【補充】

- 二分搜尋法
  - 二分搜尋法比循序搜尋法來得有效率許多，平均只需要做$\log_2 N+1$次比較即可找到資料。雖然速度比較快
  - 必須先將資料經過排序之後，才可以使用二分搜尋法。以下是二分搜尋法的原理及步驟：

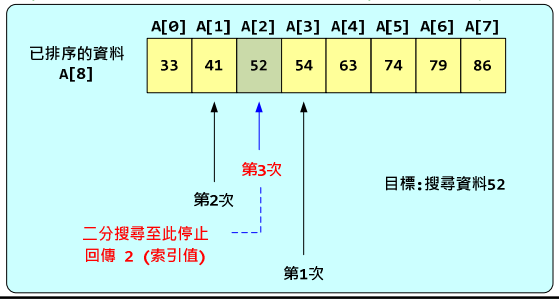


圖6-11 二分搜尋法

77

6.6搜尋演算法【補充】

- 【二分搜尋法原理】
  - 先從記錄中央開始搜尋，若該記錄比目標還小，則往大的剩餘另一半搜尋
  - 若記錄比目標還大，則往小的剩餘另一半搜尋；
  - 相等，則代表找到資料。
    - 重覆此步驟直到找到資料為止，或者發現要搜尋的資料不存在。
  - 因此，每次會剩下1/2、1/4、1/8、、、，在第k次比較時，最多只剩下『$n/2^k$』筆記錄未搜尋，在最壞的狀況下，只剩單一記錄$n/2^k=1$，也就是$k=\log_2 n$，所以最多的比較次數為$\log_2 n$。

78

6.6搜尋演算法【補充】

- **演算法（使用非正式但較接近Java語法的虛擬碼）：**

```
輸入：已排序的資料X[0]~X[n-1]、要找尋的目標資料k
輸出：目標資料的索引值middle
left←0
right←n-1
while (left ≦ right) do
{
    middle ← (left+right) div 2     //除2取商
    case
    {
        k > x[middle]: left←middle+1   // 放棄左半部
        k = x[middle]: return middle
        k < x[middle]: right←middle-1  // 放棄右半部
    }endcase
}endwhile
return -1     // 沒有符合的記錄
```

79

6.6搜尋演算法【補充】

- 【實例說明】：
  - 8個陣列元素A[0]~A[7]為『33,41,52,54,63,74,79,86』，尋找目標為52，使用二分搜尋法搜尋，則以下是詳細步驟：
  - 1. 計算中間位置為(0+7)/2=3.5取整數為3。
  - 2. A[3]=54>52，所以right=3-1=2。
  - 3. 計算中間位置為(0+2)/2=1。
  - 4. A[1]=41<52，所以left=1+1=2。
  - 5. 計算中間位置為(2+2)/2=2。
  - 6. A[2]=52，所以找到了。
- 【不實用範例6-13】：使用二分搜尋法在已排序的資料中，尋找所需要的資料『52』。
- 範例6-13：ch6_13.java（隨書光碟myJava\ch06\ch6_13.java）

80

6.6搜尋演算法【補充】

```
1   /*   檔名:ch6_13.java      功能:二分搜尋法  */
2
3   package myJava.ch06;
4   import java.lang.*;
5   import java.util.Scanner;
6
7   public class ch6_13          //主類別
8   {
9       public static void main(String args[])
10      {
11          int workArr[]={33,41,52,54,63,74,79,86};
12          int findNum,location;
13          Scanner keyboardInput = new Scanner(System.in);
14
15          System.out.print("請輸入您要找的數值:");
16          findNum = Integer.parseInt(keyboardInput.nextLine());
17          location = binarySearch(workArr,findNum);
18          if(location==-1)
19             System.out.println("在陣列中找不到要找的數值");
20          else
21             System.out.println("數值" + findNum +
                                   "位於work[" + location + "]");
22      }
23
```

執行結果

```
請輸入您要找的數值:52
數值52位於work[2]
```

81

6.6搜尋演算法【補充】

```
24      public static int binarySearch(int[] x,int k)
25      {
26          int left,right,middle;
27          left = 0;
28          right = x.length-1;
29
30          while(left<=right)
31          {
32             middle = (left + right) / 2;
33             if(k==x[middle])  return middle;
34             if(k>x[middle])   left = middle + 1;    // 放棄左半部
35             else              right = middle - 1;   // 放棄右半部
36          }
37          return -1;
38      }
39  }
```

- 範例說明：
  - (1)第24~38行是二分搜尋函式。可以接受一個尋找目標（整數）及一個已排序的工作陣列。語法如下：

所屬類別：主類別
語法：**public static int binarySearch(int[] x,int k)**
功能：二分搜尋。
引數：**k**為尋找目標。**x[ ]**為已排序的工作陣列。
回傳值：若**k**位於**x**陣列中，則回傳索引值。若不位於**x**陣列中，則回傳**-1**。

82

6.6搜尋演算法【補充】

- (2)第17行使用『傳值呼叫』傳遞findNum→k參數。使用『傳參考值呼叫』，傳送已排序的陣列workArr→x陣列變數。
- (3)workArr陣列一定要先經過排序完成，否則無法使用二分搜尋法完成工作。

- 二分搜尋法已經被Java納入Arrays類別的方法，名稱為binarySearch。故上一個範例，我們將之歸類為**不實用範例**。
- Java提供的二分搜尋法，可以針對各種資料型態進行搜尋，甚至還可以指定要搜尋的陣列範圍。
  - 以下，我們僅提供整數陣列的二分搜尋法之語法，其餘請自行參閱Java說明文件。

83

6.6搜尋演算法【補充】

所屬類別：java.util.Arrays
語法：**public static int binarySearch(int[] a,int key)**
功能：二分搜尋。
引數：**key**為尋找目標。**a[ ]**為已排序的工作陣列。
回傳值：若**key**位於**a**陣列中，則回傳索引值。若不位於**a**陣列中，則回傳負值。

- 【實用範例6-14】：改寫範例6-12、6-13，使用Arrays類別的sort與binarySearch對未排序的陣列進行搜尋，判斷資料『57』是否位於陣列中。
  - **範例6-14：**ch6_14.java（隨書光碟myJava\ch06\ch6_14.java）

84

6.6搜尋演算法【補充】

```
1  /*  檔名:ch6_14.java        功能:sort與binarySearch方法    */
2
3  package myJava.ch06;
4  import java.lang.*;
5  import java.util.Scanner;
6  import java.util.Arrays;
7
8
9
10 public class ch6_14          //主類別
11 {
12     public static void main(String args[])
13     {
14         int workArr[]={43,23,67,27,39,15,39,37,57,26,14};
15         int findNum,location;
16         Scanner keyboardInput = new Scanner(System.in);
17
18         Arrays.sort(workArr);
19         System.out.print("請輸入您要找的數值:");
20         findNum = Integer.parseInt(keyboardInput.nextLine());
21         location = Arrays.binarySearch(workArr,findNum);
22         if(location<0)
23            System.out.println("在陣列中找不到要找的數值");
24         else
25            System.out.println("數值存在於陣列中");
26     }
27 }
```

sort與binarySearch隸屬於該類別

對陣列排序。

對陣列進行二分搜尋。

執行結果

```
請輸入您要找的數值:57
數值存在於陣列中
```

85

6.6搜尋演算法【補充】

- 範例說明：
  - 在第18行先使用sort進行排序，然後在第21行進行二分搜尋
  - 如果您想要印出57所在的索引，光是印出location是無法達成的，因為陣列已經被排序過
    - 由於傳遞陣列必定是傳參考值呼叫，故陣列實體內容已經被改變

- 【實用範例6-15】：使用亂數函式、搜尋函式，完成樂透開獎遊戲（產生6個1~49的整數號碼並存放於整數陣列以及1個特別號，且這7個號碼不得重覆）。
  - **範例6-15：**ch6_15.java（隨書光碟myJava\ch06\ch6_15.java）

86

6.6搜尋演算法【補充】

```
1  /*   檔名:ch6_15.java        功能:設計樂透開獎遊戲  */
2
3  package myJava.ch06;
4  import java.lang.*;
5
6  public class ch6_15         //主類別
7  {
8      public static void main(String args[])
9      {
10         int special;              //存放特別號
11         int lottos[]=new int[6];   //存放六球
12
13         special = generate_lottos(lottos);
14         System.out.println("樂透號碼如下.....");
15         for(int num : lottos)
16            System.out.print(num + "\t");
17         System.out.println();
18         System.out.println("特別號:" + special);
19     }
20
21     public static int generate_lottos(int[] arr)
22     {
23         int generateNum;
24
```


6.6搜尋演算法【補充】

```
25         for(int i=0;i<arr.length;i++)
26         {
27           generateNum = (int)((Math.random()*49)+1);
28           while (seqSearch(generateNum,arr)!=-1)
29           {
30             generateNum = (int)((Math.random()*49)+1);
31           }
32           arr[i] = generateNum;
33         }
34         generateNum = (int)((Math.random()*49)+1);
35         while (seqSearch(generateNum,arr)!=-1)
36         {
37            generateNum = (int)((Math.random()*49)+1);
38         }
39         return generateNum;
40     }
41
42     public static int seqSearch(int target,int[] arr)
43     {
44         for(int i=0;i<arr.length;i++)
45            if(target == arr[i])       // 找到了
46               return i;
47         return -1;                  // 完全找不到
48     }
49 }
```

是否與陣列元素重複

特別號是否與其他號碼重複

6.6搜尋演算法【補充】

- 執行結果：

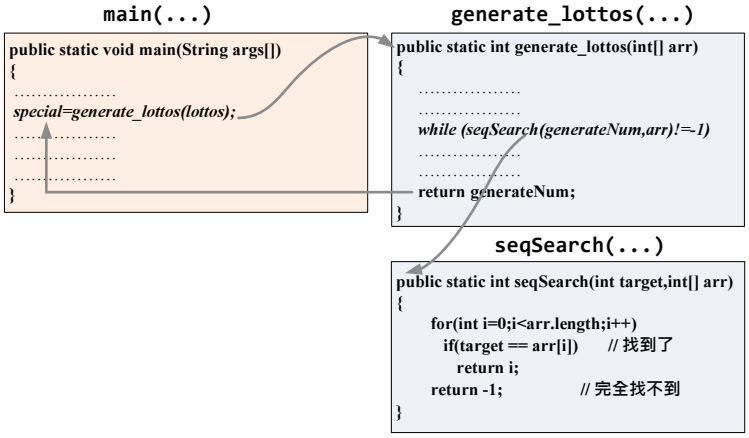
```
樂透號碼如下.....
49    19    29    9    37    3
特別號:35
```

- 範例說明：
  - (1)這個程式使用了前面範例介紹的兩個函式，分別是
    - [1]亂數產生1~49的數字
    - [2]使用循序搜尋決定開出的號碼是否重複。
  - (2)我們修改了generate_lottos()函式，使得它能夠回傳一個整數值，以代表特別號。
  - (3)由於陣列自動初始化會將整數陣列元素設定為0，而呼叫循序搜尋時，由於輸入的球號必定為1~49（不會為0），故仍可適用。
  - (4)本範例的兩個函式可用於不限定樂透開獎球數的遊戲中，例如可以用來製作大樂透（49選7球）、國外的超級樂透（49選8球，只要在陣列宣告時多宣告1個元素即可）。

89

6.6搜尋演算法【補充】

- (5)函式呼叫關係如下圖。（本範例未將6個樂透開獎號碼排序顯示）



90

6.7 main函式的參數串列

- 那麼main函式也可以接收呼叫者傳入的引數值嗎？
  - 答案是肯定的，不過main的參數串列已經在Java中詳細定義，使用者不得更改。
- main函式的參數是用來接收JVM的命令引數
  - 例如：我們在Dos命令列中
    - 若輸入『java test This is a book』
    - 則代表要求JVM執行test類別，並將『This is a book』等四個字串傳入test類別的main函式中。

91

6.7 main函式的參數串列

- main函式的參數被宣告為字串陣列String[] args或String args[]。
  - 我們可以在main()函式中取用這些字串，做其他進一步的應用。
  - 【觀念範例6-16】：接收由JVM傳送過來的命令列引數。
    - **範例6-16：**ch6_16.java（隨書光碟 myJava\ch06\ch6_16.java）

```
1   /*   檔名:ch6_16.java      功能:main函式的參數   */
2
3   package myJava.ch06;
4   import java.lang.*;
5
6   public class ch6_16          //主類別
7   {
8       public static void main(String args[])
9       {
10          System.out.println("本程式共接受到" + args.length + "個輸入引數");
11          for(int i=0;i<args.length;i++)
12             System.out.println("args[" + i + "]字串為" + args[i]);
13      }
14  }
```

92

6.7 main函式的參數串列

- 執行結果：

```
C:\>java myJava.ch06.ch6_16 This is a book
本程式共接受到4個輸入引數
args[0]字串為This
args[1]字串為is
args[2]字串為a
args[3]字串為book
```

- 範例說明：
  - 在執行結果中，我們先要求JVM執行myJava.ch06 Package的ch6_16類別，並傳送『This』、『is』、『a』、『book』等四個額外引數給ch6_16類別的main()函式。

93

6.7 main函式的參數串列

- 【實用及觀念範例6-17】：由於範例6-15可以修改為『其他球數的樂透遊戲』，因此我們將範例6-17改寫為由使用者在輸入命令列引數時決定要開出的球數（球數當然不應該超過48球，因為必須保留一個球作為特別號）。
  - **範例6-17：**ch6_17.java（隨書光碟 myJava\ch06\ch6_17.java）

```
1   /*  檔名:ch6_17.java        功能:設計樂透開獎遊戲  */
2
3   package myJava.ch06;
4   import java.lang.*;
5
6   public class ch6_17         //主類別
7   {
8       public static void main(String args[])
9       {
10          int special,balls=6;              //存放特別號
11
12          if(args.length>0)
13             balls = Integer.parseInt(args[0]);
14          if((balls>48)||(balls<0)) return;       //球數錯誤，離開
15          int lottos[]=new int[balls];             //存放balls球
16          special = generate_lottos(lottos);
```

94

6.7 main函式的參數串列

```
17          System.out.println("樂透號碼如下.....");
18          for(int i=0;i<balls;i++)
19          {
20             if((i%6==0)&&(i!=0))
21                 System.out.println();
22             System.out.print(lottos[i] + "\t");
23          }
24          System.out.println();
25          System.out.println("特別號:" + special);
26      }
27      public static int generate_lottos(int[] arr)
28      {
29          …同範例6-15的第23~39行之generate_lottos函式內容…
:       }
47
48      public static int seqSearch(int target,int[] arr)
49      {
50           …同範例6-15的第44~47行之seqSearch函式內容…
:       }
55  }
```

95

6.7 main函式的參數串列

- 執行結果：

```
C:\>java myJava.ch06.ch6_17 48
樂透號碼如下.....
11      9       33      32      44      5
3       39      4       21      49      38
19      42      34      43      17      46
22      41      47      31      40      20
8       7       36      1       48      29
16      6       35      37      23      24
45      2       13      27      12      25
26      15      10      18      28      30
特別號:14
```

- 範例說明：
  - (1)我們修改了main()函式，讓使用者可以指定開出的球數（若使用者輸入超過48或其他文字，則會執行return提前結束程式）。
  - (2)第13行，由於輸入的引數會被存放到字串陣列，因此，必須將字串轉換為數值。

96

6.8 函式的final參數

- 由於函式參數列的參數可以被當作區域變數使用，因此，也可以被宣告為final，而宣告為final的參數，在函式內完全不能被重新設定
  - 因為當函式呼叫發生而傳遞引數時，參數就被設定了某個初值。

97

6.8 函式的final參數

- 通常將參數宣告為final，是為了避免函式不小心修改了該參數，因此你可以把該參數視為「特殊的」常數來使用。
  - 如果您只是為了避免呼叫端的引數被修改，則使用final來宣告是不必要或無意義的，請見下列的分析。
  - Case1：
    - 呼叫端的引數為原始資料型態變數，此時，由於是傳值呼叫，因此被呼叫的函式並不會改變原呼叫端的引數內容。所以不需要宣告為final。

98

6.8 函式的final參數

- Case2：
  - 呼叫端的引數為常數（例如3,5.7,'r',"abc"），此時，由於是呼叫端為常數，所以不可能被更改。所以不需要宣告為final。
- Case3：
  - 呼叫端的引數為陣列變數或物件變數（參考），由於採用傳參考值呼叫，因此，即使您使用了final來宣告參數，代表的只不過是陣列或物件變數不會被改變，而非所指實體不會被改變，所以想要藉此達成陣列實體內容或物件實體內容不會被改變是行不通的。請見下一個範例。
- 【觀念範例6-18】：釐清函式呼叫的引數傳遞與final參數的效果。
  - **範例6-18：**ch6_18.java（隨書光碟myJava\ch06\ch6_18.java）

99

6.8 函式的final參數

```
1  /*    檔名:ch6_18.java          功能:final陣列參數  */
2
3  package myJava.ch06;
4  import java.lang.*;
5
6  public class ch6_18
7  {
8      public static void main(String args[])
9      {
10         int orgArr[] = new int[3];
11         for(int i=0;i<orgArr.length;i++)
12             orgArr[i] = i;
13         func1(orgArr);
14         for(int i=0;i<orgArr.length;i++)
15             System.out.println("orgArr[" + i + "]=" + orgArr[i]);
16     }
17
18     public static void func1(final int[] arr1)
19     {
20         int arr2[]={5,10,15};
21         for(int i=0;i<arr1.length;i++)
22             arr1[i] = arr1[i] * arr1[i];
23         //arr1=arr2;
24     }
25 }
```

執行結果

```
orgArr[0]=0
orgArr[1]=1
orgArr[2]=4
```

修改陣列實體的內容

不合法的敘述

100

6.8 函式的final參數

- 範例說明：
  - (1)第11~12行，已經將orgArr的陣列實體內容設定為{0,1,2}。
  - (2)第13行呼叫func1，並將orgArr的陣列參考傳遞給arr1。因此arr1被第一次設定。
  - (3)由於arr1被宣告為final且設定過一次，因此第23行欲將arr1改為參考arr2的陣列實體時，代表修改arr1陣列參考，將會出現編譯錯誤，因為arr1被宣告為final，不可再被設定一次。
  - (4)在第21~22行，修改陣列實體內容，由於並非修改陣列參考，因此是合法的敘述。同時，由於陣列實體內容被改變了，因此當函式返回時，orgArr的陣列實體內容也是被修改過的內容，即{0,1,4}。所以我們不能透過final來保證原陣列內容不被更動。

101

6.9遞迴函式

- 當一個函式呼叫自己的時候會發生什麼狀況呢？
  - 這就是所謂函式的**遞迴呼叫(recursive call)**。其實『遞迴呼叫』應該明確定義如下：
- **遞迴呼叫：**
  - 一個函式經由直接或間接呼叫函式本身，稱之為函式的『遞迴呼叫』。
    - 例如：func1()呼叫func1()為直接遞迴呼叫
    - func1()呼叫func2()且func2()呼叫func1()為間接遞迴呼叫。

102

6.9遞迴函式

- Java允許函式的遞迴呼叫，通常遞迴函式可以輕鬆解決一些資訊領域常見的問題（例如：樹狀圖的相關演算法），而且相當簡潔使人易懂，但執行效率則略遜一疇。
- 通常初次介紹遞迴函式時，大多以著名的費氏數列或河內塔等問題來解釋及示範遞迴函式，在此我們就先來解釋何謂『費氏數列』：

103

6.9遞迴函式

- 『費氏數列』：
  - 一個無限數列，該數列上的任一元素值皆為前兩項元素值的和，數列如下所示：
    - 0，1，1，2，3，5，8，13，21，34，55，89，144……
  - 當我們使用數學函數來表示費氏數列時，費氏數列的定義本身就是一個遞迴定義如下：

費氏數列的遞迴定義式	
F(0) = 0	n = 0
F(1) = 1	n = 1
F(n) = F(n-1)+F(n-2)	n ≧ 2

104

6.9遞迴函式

- 在實際計算時【例如計算F(10)】，數學的遞迴函數必須不斷地重複呼叫自己，直到遇上非遞迴定義式為止，才能夠求出答案。
- 同樣地，當我們使用Java設計遞迴函式時，也必須對該函式做出某些限制條件，以避免函式無窮的執行下去，通常一個遞迴函式需符合下列兩個限制條件：
  - (1)遞迴函式必須有邊界條件，當函式符合邊界條件時，就應該返回（可使用return 強制返回）函式呼叫處，在費氏數列中，F(0)=0與F(1)=1就是函式的邊界條件。
  - (2)遞迴函式在邏輯上，必須使得函式漸漸往邊界條件移動，否則該函式將無法停止呼叫，而無窮地執行下去，由於每次的函式呼叫都會使用一些記憶體堆疊，最終將造成記憶體不足的問題。

105

6.9遞迴函式

- 【實用及觀念範例6-19】：使用遞迴，求出費氏數列第0~25項的元素值。
  - 範例6-19：ch6_19.java（隨書光碟 myJava\ch06\ch6_19.java）

```
1   /*  檔名:ch6_19.java          功能:遞迴函式求費氏數列  */
6   public class ch6_19           //主類別
7   {
8       public static void main(String args[])
9       {
10          System.out.print("費氏數列如下:");
11          for(int i=0;i<=25;i++)
12          {
13             if(i%8==0)
14                System.out.println();
15             System.out.print(Fib(i) + "\t");
16          }
17          System.out.println("......");
18      }
```

106

6.9遞迴函式

```
19
20      public static int Fib(int n)
21      {
22          if((n==1) || (n==0))
23            return n;
24          else
25            return Fib(n-1) + Fib(n-2);
26      }
27  }
```

- 執行結果：

```
費氏數列如下:
0       1       1       2       3       5       8       13
21      34      55      89      144     233     377     610
987     1597    2584    4181    6765    10946   17711   28657
46368   75025   ......
```

107

6.9遞迴函式

- 範例說明：
  - (1)為節省篇幅，本範例之後的所有範例，如果只是依循著與前一範例相同的Package宣告，本書可能會將之省略不顯示出來，如果是引入一些常見的函式庫，也會省略不顯示出來。請讀者自行參閱光碟內容。
  - (2)在main函式中，大多數的程式碼都是為了處理列印的問題，實際上最重要的程式出現在第15行的呼叫Fib(i)，以便計算費氏數列的元素值。
  - (3)簡潔有力，只是將數學定義式轉換為Java程式語法而已。這就是遞迴函式的優點。
  - 舉例來說，當main()的函式呼叫敘述Fib(4)執行時，函式呼叫與返回狀況如下圖：

108

6.9遞迴函式

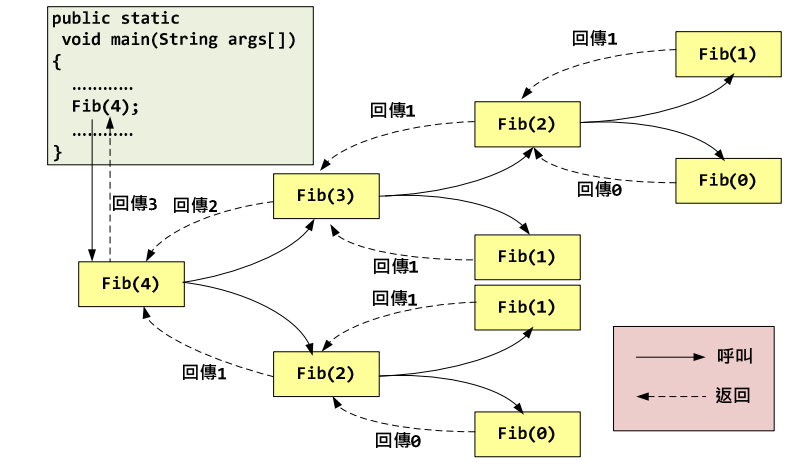


圖6-12 Fib(4)的遞迴呼叫與返回

109

6.9遞迴函式

- 執行遞迴函式時，可能會呼叫函式很多次，而每一次呼叫都必須將相關資料疊入(push)堆疊中
  - 因此，當呼叫的層次越多時，就必須使用到非常大的堆疊記憶體空間，同時也會耗費不少時間來處理程式的呼叫與返回，如此一來將會使得程式非常沒有效率
  - 因此程式設計師有時會將遞迴函式轉換成普通的迴圈結構，節省記憶體空間並提高執行效率。舉例來說，我們可以將上面的遞迴範例改寫為下面範例的迴圈結構：

110

6.9遞迴函式

- 【觀念範例6-20】：使用迴圈，求出費氏數列第0~25項的元素值。
  - **範例6-20：**ch6_20.java（隨書光碟 myJava\ch06\ch6_20.java）

```
1   /*   檔名:ch6_20.java      功能:使用迴圈求費氏數列   */
6   public class ch6_20          //主類別
7   {
8       public static void main(String args[])
9       {
10          System.out.print("費氏數列如下:");
11          for(int i=0;i<=25;i++)
12          {
13             if(i%8==0)
14                System.out.println();
15             System.out.print(Fib(i) + "\t");
16          }
17          System.out.println("......");
18      }
19
```

111

6.9遞迴函式

```
20      public static int Fib(int n)
21      {
22          int n1=0,n2=0,sum=1;
23
24          if((n==1) || (n==0))
25             return n;
26          else
27          {
28             for(int i=2;i<=n;i++)
29             {
30                n1 = sum;
31                sum = sum + n2;
32                n2 = n1;
33             }
34             return sum;
35          }
36      }
37  }
```

- 執行結果：（同範例6-19）
- 範例說明：
  - 使用迴圈來解決此類問題顯得複雜許多（如第28~33行），不過將會比遞迴函式快了許多（您可以將n值擴大，就會感覺到兩者之間的效率差異越來越大），並且不必擔心記憶體空間不足的問題。

112

6.10多載(overloading)

- 在傳統的結構化程式設計中（例如傳統C程式），一般都不允許出現同名函式，但這會發生兩個問題：
  - 1.由不同人設計的函式，可能出現同名的現象。
    - 例如，可能有一個Search函式是由A函式庫引入，功能是用來找出字串的某個字元。而另一個由B函式庫引入的Search函式則是用來找出整數陣列的某項資料。
    - 由於兩個函式庫由不同人發展，因此，在開發函式庫時，無法預測其他人會使用哪個函式名稱，而一般人對於函式命名的原則大多相同，因此容易發生同名的現象。

113

6.10多載(overloading)

  - 2.針對同樣功能的函式，必須對於不同資料型態，取不同的函式名稱。
    - 例如，可能有一個函式名稱為abs(int x)，用來取整數的絕對值，它接收的引數為整數型態。但如果要適用於取浮點數絕對值時，則必須取不同的名稱，例如absFloat(float x)。
- 上述兩項問題，對於發展大型程式非常不利，而物件導向語言利用一些機制改善此問題。

114

6.10多載(overloading)

- 針對這兩種問題，物件導向程式設計（例如C++或Java）採用下列方式來解決。
  - 1.改以類別來區分，只要在類別內不出現同名函式即可（不同類別的函式同名則無妨），當然如此做還是可能出現類別名稱相同的問題，此時，不同的程式語言則採用不同的機制。例如：C++採用的是namespace機制、Java採用的是Package機制來管理。
  - 2.在上述解決方案中，所謂類別內不允許出現「同名函式」，實際上的規範則是不允許出現「同署名的函式」，因此，同名但不同署名的函式是允許出現的，這就是所謂**多載(orverload)**的機制。

115

6.10多載(overloading)

- 所謂函式署名則包含了下列三項元素：
  - 1.函式名稱
  - 2.參數資料型態
  - 3.參數的個數與順序

```
public static int func1(int n,float b,...)
```

116

6.10多載(overloading)

- Java提供了多載功能，故一個類別內的函式署名不能相同，意即函式名稱可以相同，只要參數的資料型態或個數或順序其中有一項不同即可。如下範例：

```
class MyClass
{
   void show(){}
   void show(int n){}
  //void show(int a){}      不合法,因只有參數名稱不同,但那並非署名的一部分

  //int show(int n){}       不合法,因只有回傳值資料型態不同,但那並非署名的一部分

  //int show(int a){}       不合法,原因同上述兩點
   void show(double n){}
   void show(int m,int n){}
  //void show(int n,int m){}  不合法,因只有參數名稱不同,但那並非署名的一部分
   void show(double m,int n){}
   void show(int m double m){}
   ……
}
```

117

6.10多載(overloading)

筆者的話

法律沒有規定兩個人不能有相同的名字，所以不同的父母親可能為自己的子女取了相同的名字。不過即使是相同名字的兩個人，其簽名的樣子也不會相同（故文件上的簽名就可以作為區別），而函式利用**署名(signature)**來區分，大概也是由此而來。

118

6.10多載(overloading)

- 多載的用意
  - 多載免除了相同功能卻必須使用不同函式名稱開發的困擾，如同第三章所提到String的valueOf(原始資料型態 引數)函式，它的引數可以輸入不同的原始資料型態，其用意都是將該引數轉換為字串，故功能相同，不應以不同的函式名稱來命名，而多載提供了這種功能。
    - 例如在String類別的說明文件中，我們可以發現下列關於valueOf method的宣告語法：

```
static String valueOf(boolean b) { ... }
static String valueOf(char c) { ... }
static String valueOf(char[] data) { ... }
static String valueOf(char[] data, int offset, int count) { ... }
static String valueOf(double d) { ... }
static String valueOf(float f) { ... }
static String valueOf(int i) { ... }
static String valueOf(long l) { ... }
static String valueOf(Object obj) { ... }
```

119

6.10多載(overloading)

- 我們不應隨意「濫用」多載，而忽略了函式名稱所代表的重要性。
  - 例如我們有一些關於幾何圖形的成員變數 length, width, radius, …等等，並且有計算面積的函式如下，則可以透過下列兩種方式命名：

```
class MyClass
{
   int calArea(int a,int b){....}
   int calArea(double a,double b){....}
   double calArea(int r){...}
   double calArea(double r){...}
}
```

合法

```
class MyClass
{
   int calRectArea(int a,int b){....}
   int calRectArea(double a,double b){....}
   double calCircleArea(int r){...}
   double calCircleArea(double r){...}
}
```

合法且較有擴充性

120

6.10多載(overloading)

- 第一種命名方式，四個函式共同發生多載現象，而第二種方式則是上兩個函式發生多載現象，下兩個函式也發生多載現象。
  - 雖然兩者都合法，但第一種方法，可能無法再被擴充為計算正方形的面積，例如再宣告一個double calArea(double edge){…}來計算正方形面積就不合法了。
  - 當然，在這個範例中，光是由幾何圖形作為類別可能是不足夠的，我們也可以透過繼承機制將幾何圖形分為更細的矩形、正方形、圓形等子類別來解決相同問題，由於牽涉的技巧較為複雜，我們於後面章節再行介紹。

121

6.10多載(overloading)

- 「多載」是實踐物件導向設計的「多型」設計，多型(Polymorphism)又稱為同名異式
  - 它代表的是使用相同函式名稱，但可以有不同的函式內容，所謂『名』指的是函式名稱，而『式』則是函式內容的敘述群。
- 由於函式內容的敘述群可以不同，因此功能也可能不同。因為程式語言規範的只能是語法而很難檢測出函式內容究竟提供了什麼樣的服務。
- 但我們一般在使用多載時，仍以同功能的函式為主要目的，但同功能並非指的是只有資料型態的不同，而程式碼完全相同；如果是僅僅資料型態不同，但程式碼完全相同，某些程式語言則提供了「樣板」或「泛型」來解決。
- 物件導向設計的「多型」設計，除了多載之外，還有改寫(override)以及介面等等，我們將於後面陸續介紹。

122

6.10多載(overloading)

- 多載的函式之間，是否可以互相呼叫呢？
  - 答案是可以的，因為它們仍是不同的函式，並且當彼此發生呼叫時，仍不算是遞迴呼叫，除非，它們間接或直接呼叫回原本同署名的函式。
  - 事實上，為了切割程式的主要設計，我們有時也會在多載的函式間進行呼叫，請見下面的範例：

123

6.10多載(overloading)

- 【觀念範例6-21】：釐清多載的設計。
  - **範例6-21：**ch6_21.java（隨書光碟myJava\ch06\ch6_21.java）

```
1   /*   檔名:ch6_21.java        功能:函式的多載  */
2
3   package myJava.ch06;
4   import java.lang.*;
5
6   public class ch6_21
7   {
8       public static void main(String args[])
9       {
10          printHello();          //呼叫第18~21行的函式
11          System.out.println("----------");
12          printHello(2);         //呼叫第23~37行的函式
13          System.out.println("----------");
14          printHello("three");   //呼叫第38~44行的函式
15          //printHello(2.1);
16      }
```

錯誤的函式呼叫

執行結果

```
Hello Java
----------
Hello Java
Hello Java
----------
Hello Java
Hello Java
Hello Java
```

124

6.10多載(overloading)

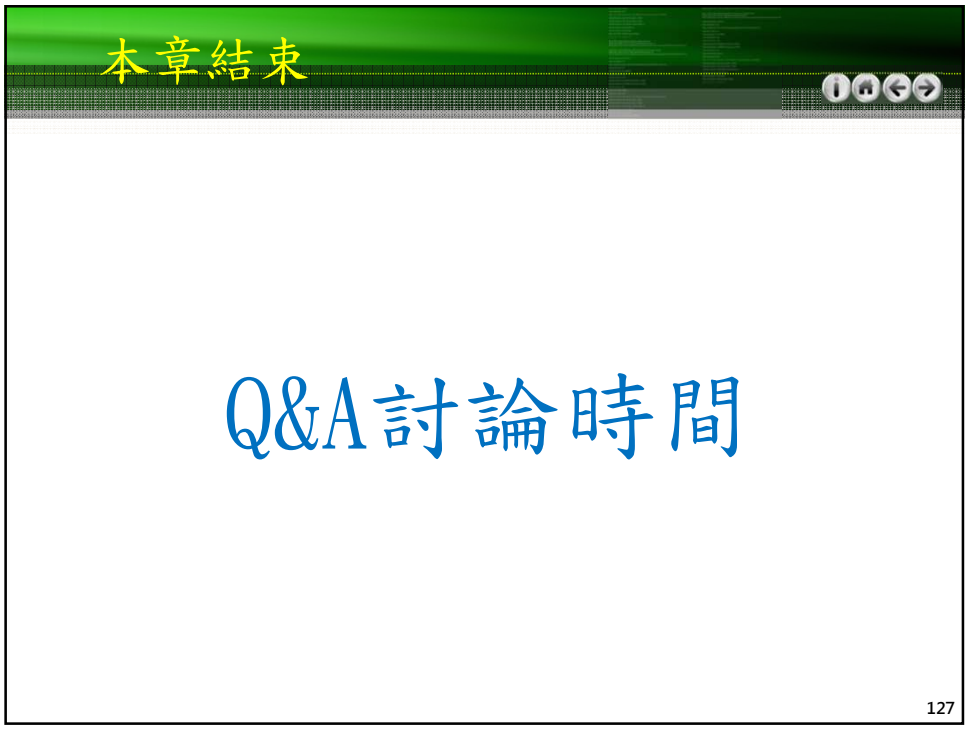
```
18  public static void printHello()
19     {
20         System.out.println("Hello Java");
21     }
22
23     public static void printHello(int n)
24     {
25         if(n>3)
26         {
27            System.out.println("sorry,more than 3!");
28            return;
29         }
30         else if(n<0)
31         {
32            System.out.println("sorry,bad command!");
33            return;
34         }
35         for(int i=0;i<n;i++)
36            printHello();       //呼叫第18~21行的函式
37     }
38     public static void printHello(String str1)
39     {
40         if(str1=="one") printHello();          //呼叫第18~21行的函式
41         else if(str1=="two") printHello(2);    //呼叫第23~37行的函式
42         else if(str1=="three") printHello(3); //呼叫第23~37行的函式
43         else System.out.println("sorry,more than 3 or bad command!");
44     }
45  }
```

125

6.10多載(overloading)

- 範例說明：
  - 函式呼叫會執行的函式如註解所示。而第15行的函式呼叫會發生編譯錯誤，因為編譯器找不到printHello(double 參數)的函式可以進行呼叫。

126


本章結束
Q&A討論時間
127