Appendix

A Java 語言的基礎

本章學習目標

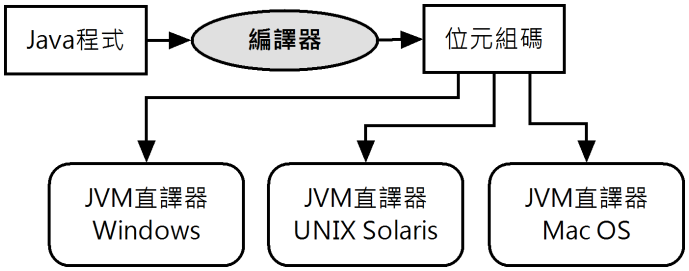
A-1 Java 的基礎

「Java」(爪哇)是一種物件導向程式語言,類似 C++ 語言的編譯語言,不過並不完全相同,因為它是結合編譯和直譯優點的一種新世代的程式語言。

Java 屬於一種高階程式語言,Java 程式是在 Java 語言的「平台」(Platform)上執行,平台是一種結合硬體和軟體的執行環境,因為 Java 屬於一種與硬體無關和跨平台的程式語言,所以 Java 平台是一種軟體平台,主要是由 JVM 和 Java API 兩個元件所組成。

JVM (Java Virtual Machine)

Java 編譯器可以將 Java 原始程式碼編譯成「位元組碼」(Bytecode),這種程式碼是一種虛擬的機器語言,這台電腦稱為「JVM」(Java Virtual Machine),所以,在作業系統需要安裝 JVM 的直譯器,才能夠直譯和執行位元組碼,如下圖所示:



上述圖例的 Java 原始程式碼(副檔名.java)在編譯成位元組碼(副檔名.class)後,就可以在 Windows、UNIX 或 Mac OS 作業系統上執行,只需作業系統安裝 JVM 直譯器,同一個位元組碼檔案,可以跨平台在不同作業系統上正確的執行。

Java API (Java Application Programming Interface)

Java API 是軟體元件的集合,也就是在 C/C++ 語言所謂的函數庫,提供集合物件、GUI 元件、檔案處理、資料庫存取和網路等相關的類別和介面,稱為「套件」(Packages)。

A-2 Java 程式的架構

Java 程式的架構與傳統程式語言 C/C++ 或 BASIC 不同，因為 Java 屬於一種真正的物件導向程式語言，其程式架構是「類別」(Class) 宣告，例如：ChA_2.java，如下所示：

```
01: /* 程式範例: ChA_2.java */
02: public class ChA_2 {
03:     // 主程式
04:     public static void main(String[] args) {
05:         // 顯示訊息
06:         System.out.println("第一個 Java 應用程式");
07:     }
08: }
```

上述 Java 程式架構是使用 public class 關鍵字和大括號括起的類別宣告，類別名稱是 ChA_2，這是 Java 程式的基本架構，其詳細說明如下所示：

- **程式註解**：第 1 列是程式註解，說明程式名稱等資料，如下所示：

```
01: /* 程式範例: ChA_2.java */
```

- **類別宣告**：在第 2~8 列是和檔案名稱 ChA_2 同名的類別宣告，如下所示：

```
02: public class ChA_2 {
03:     // 主程式
04:     public static void main(String[] args) {
05:         // 顯示訊息
06:         System.out.println("第一個 Java 應用程式");
07:     }
08: }
```

上述類別區塊是一個使用 public 關鍵字宣告的類別，請注意！檔案名稱需要和宣告成 public 類別的名稱相同，而且英文字母大小寫也需相同。

- **主程式**：第 4~7 列的 main()方法是 Java 程式的主程式，這是 Java 應用程式執行時的進入點，也就是說，執行 Java 程式是從此方法開始，如下所示：

```
04:    public static void main(String[] args) {
05:       // 顯示訊息
06:       System.out.println("第一個 Java 應用程式");
07:    }
```

上述 main() 方法宣告成 public、static 和 void，表示是公開、靜態類別和沒有傳回值的方法，在第 6 列使用 System 類別子類別 out 的 println()方法顯示參數的字串，這是 Java 語言的標準輸出，如下所示：

```
System.out.println("第一個 Java 應用程式");
```

上述程式碼的 println()方法將括號內的參數字串輸出到螢幕顯示，並且換行，字串是使用「"」號括起一組字元集合。如果不換行是使用 print()方法，如下所示：

```
System.out.print("第一個 Java 應用程式");
```

A-3 Java 語言的變數與資料型態

Java 資料型態分為「基本」（Primitive）和「參考」（Reference）兩種資料型態，如下所示：

- **基本資料型態**：變數共有 byte、short、int、long、float、double、char 和 boolean 八種資料型態。
- **參考資料型態**：變數值是一個記憶體位置，這個位置值是物件儲存的位置，例如：String 字串、陣列和物件。

A-3-1 Java 的基本資料型態

Java 基本資料型態依資料類型，可以分為整數、浮點數、布林和字元資料型態。

整數資料型態

「整數資料型態」(Integral Types)是指變數的資料為整數沒有小數點,依照整數資料長度的不同(即佔用的記憶體位元數),可以分為 4 種整數資料型態,如下表所示:

整數資料型態	位元數	範圍
byte	8	$-2^7 \sim 2^7-1$,即-128 ~ 127
short	16	$-2^{15} \sim 2^{15}-1$,即-32768 ~ 32767
int	32	$-2^{31} \sim 2^{31}-1$,即-2147483648 ~ 2147483647
long	64	$-2^{63} \sim 2^{63}-1$,即-9223372036854775808 ~ 9223372036854775807

浮點數資料型態

「浮點數資料型態」(Floating Point Types)是指整數加上小數,例如:3.14、100.567 等,依照長度的不同(即佔用的記憶體位元數),可以分為 2 種點數的資料型態,如下表所示:

浮點數資料型態	位元數	範圍
float	32	1.40239846e-45 ~ 3.40282347e38
double	64	4.94065645841246544e-324 ~ 1.79769313486231570e308

布林資料型態

「布林資料型態」(Boolean Type)的變數只有 2 種值 true 和 false,這不是變數名稱,而是 Java 保留字,布林變數主要是使用在邏輯運算式,如下所示:

```
rate >= .04
```

上述運算式的結果是布林資料型態,可以使用在條件和迴圈控制的條件判斷,以便決定繼續執行哪一區塊的程式碼,或是判斷迴圈是否結束。

字元資料型態

「字元資料型態」(Char Type)是「無符號」(Unsigned)的 16 位元整數所表示的 Unicode 字元,Unicode 字元使用 2 個位元組表示字元,這是用來取代 ASCII 字元單一位元組的表示方式。

A-3-2 變數宣告

變數的目的是儲存程式執行中的一些暫存資料，程式設計者只需記住變數名稱，而且知道名稱表示一個記憶體位置中的資料，至於這個記憶體位置到底有哪裡？並不用傷腦筋，因為這是編譯程式的工作。

Java 語言在宣告變數時，一定需要指定變數的資料型態。例如：整數變數 balance 宣告的範例，如下所示：

```
int balance;
```

上述程式碼宣告一個整數變數，資料型態為整數 int，名稱為 balance，如果需要同時宣告多個變數，請使用「, 」逗號分隔，如下所示：

```
int i, j, balance;
```

上述程式碼在同一列程式敘述宣告 3 個整數變數 i、j 和 balance。

A-3-3 常數宣告

「常數」（Named Constants）是指一個變數在設定初始值後，就不會變更其值，簡單的說，就是在程式中使用一個名稱代表一個固定值。

Java 常數宣告和指定初值的變數宣告相同，只需在前面使用 final 關鍵字，如下所示：

```
final double PI = 3.1415926;
```

上述程式碼宣告圓周率的常數 PI。請注意！在宣告常數時一定要指定常數值。

A-3-4 指定敘述

「指定敘述」（Assignment Statement）可以在程式碼存取變數值，如果在宣告變數時沒有指定變數初值，我們可以使用指定敘述即「=」等號指定變數值或更改變數值，如下所示：

```
int size, size1;
size = 35;
size1 = 57;
```

A-4 Java 語言的運算子

Java 指定敘述的「運算式」(Expressions) 都是由「運算子」(Operators) 和「運算元」(Operands) 組成，Java 語言擁有完整的算術、指定、位元和邏輯運算子，一些運算式的範例，如下所示：

```
a + b - 1
a >= b
a > b && a > 1
```

上述運算式變數 a、b 和數值 1 都屬於運算元，「+」、「-」、「>=」、「>」和「&&」為運算子，Java 語言的運算子是使用 1 到 3 個字元所組成的符號。

Java 語言各運算子說明的優先順序 (愈上面愈優先)，如下表所示：

運算子	說明
()	括號
!、-、++、--	條件運算子 NOT、算數運算子負號、遞增和遞減
*、/、%	算術運算子的乘、除法和餘數
+、-	算術運算子加和減法
<<、>>、>>>	位元運算子左移、右移和無符號右移
>、>=、<、<=	關係運算子大於、大於等於、小於和小於等於
==、!=	關係運算子等於和不等於
&	位元運算子 AND
^	位元運算子 XOR
|	位元運算子 OR
&&	條件運算子 AND
||	條件運算子 OR
?:	條件控制運算子
=、op=	指定運算子

條件控制運算子「?:」可以在運算式建立簡單的條件控制敘述，如同是一個 if 條件敘述。

A-5 Java 語言的流程控制指令

流程控制可以配合運算式條件來執行不同程式區塊，或重複執行指定區塊的程式碼，流程控制主要分為 2 類，如下所示：

- **條件控制**：條件控制是一個選擇題，可能為單一選擇、二選一或多選一，依照運算式的條件值，決定執行哪一個程式區塊的程式碼。
- **迴圈控制**：迴圈控制是重複執行程式區塊的程式碼，擁有結束條件可以結束迴圈的執行。

A-5-1 條件控制指令

Java 條件控制敘述是使用關係和條件運算式，配合程式區塊建立的決策敘述，可以分為選擇（if）、二選一（if/else）或多選一（switch）幾種方式，此外還提供條件敘述運算子（?:）可以建立單行程式碼的條件控制。

if 是否選條件敘述

if 條件敘述是一種是否執行的單選題，只是決定是否執行區塊內的程式碼，如果關係/條件運算結果為 true，就執行括號間的程式區塊，一個條件敘述的範例，如下所示：

```
if ( score >= 60 ) {
   System.out.println("成績及格......");
   System.out.println("分數: " + score);
}
```

if/else 二選一條件敘述

if 條件只是選擇執行或不執行程式區塊的單一選擇，更進一步，如果是條件是排它情況的 2 個執行區塊，只能二選一，我們可以加上 else 指令。此時如果 if 的關係/條件運算式為 true，執行 else 之前的程式敘述，false 就執行之後的程式敘述，一個條件敘述的範例，如下所示：

```
if ( score >= 60 && type == 'm' ) {
   System.out.println("課程: " + type);
   System.out.println("成績及格: " + score);
}
else
   System.out.println("課程不正確或成績不及格");
```

上述程式碼因為成績有排它性，只有當成績超過 60 分及格分數，且課程代碼為 m 時，條件才成立，可以顯示不同的文字內容。

?:條件運算式

Java 條件敘述運算子?:可以使用在指定敘述以條件指定變數值，如同一個 if/else 條件，使用「?」符號代替 if，「:」符號代替 else，如果條件成立，就將變數指定成「:」前的變數值，否則就是之後的變數值，一個條件敘述運算子的範例，如下所示：

```
hour = (hour >= 12) ? hour-12 : hour;
```

上述程式碼使用條件敘述運算子指定變數 hour 的值，如果條件為 true，hour 變數值為 hour-12，false 就是 hour。

if/else/if 多選一條件敘述

Java 程式如果需要多選一的條件敘述，也就是依照一個條件判斷來執行多個區塊之一的程式碼，在 Java 程式只需重複使用 if/else 條件，就可以建立多選一的條件敘述，如下所示：

```
if ( score >= 80 )
   System.out.println("學生成績 A");
else
   if ( score >= 70 )
      System.out.println("學生成績 B");
   else
      System.out.println("學生成績 C");
```

上述程式碼使用 if/else 條件，每次判斷一個條件，如果為 false 就重複使用 if/else 條件再進行下一次的判斷，這種多選一的條件敘述架構比較複雜。

switch 多選一條件敘述

Java 語言的 switch 多條件敘述比較簡潔，可以依照符合條件執行不同區塊的程式碼，在 switch 條件只擁有一個關係/條件運算式，每一個 case 條件的比較相當於一個「==」運算子，如果符合，就執行 break 指令前的程式碼，每一個條件需要使用 break 指令跳出條件敘述。

在 switch 多條件敘述最後的 default 指令並非必要指令，這是一個例外條件，如果 case 條件都沒有符合，就執行 default 程式區塊，一個條件敘述的範例，如下所示：

```
switch (grade) {
    case 'A':
       System.out.println("學生成績超過 80");
       break;
    case 'B':
       System.out.println("學生成績超過 70");
       break;
    case 'C':
       System.out.println("學生成績超過 60");
       break;
    default:
       System.out.println("學生成績不及格");
}
```

上述程式碼比較成績 A、B 和 C 以便顯示不同的成績範圍，

A-5-2 迴圈控制指令

迴圈控制能夠重複執行指定區塊的程式碼，Java 支援多種迴圈控制敘述，能夠在迴圈的開始或結尾測試迴圈的結束條件。

for 計數迴圈

Java 語言的 for 迴圈屬於一種簡化的 while 迴圈，可以執行固定次數的程式區塊，迴圈預設提供計數器，計數器每一次增加或減少一個固定值，直到迴圈的結束條件成立為止。

for 迴圈稱為「計數迴圈」(Counting Loop)，迴圈使用變數控制迴圈的執行，從一個最小值執行到最大值，例如：計算 1 加到 10 的總和，每次增加 1，如下所示：

```
for ( i = 1; i <= 10; i++ ) {
   System.out.print("|" + i);
   total += i;
}
```

上述迴圈的程式碼是從 1 加到 10 計算其總和。相反的情況，如果是從 10 到 1，此時 for 迴圈的計數器是使用 i--，表示每次遞減 1，如下所示：

```
for ( i = 10; i >= 1; i-- ) { …… }
```

前測式 while 迴圈敘述

while 迴圈敘述不同於 for 迴圈，需要在程式區塊自己處理計數器的增減，while 迴圈是在程式區塊的開頭檢查結束條件，如果條件為 true 才進入迴圈執行，例如：使用 while 迴圈計算階層 5!的值，如下所示：

```
while ( level <= 5 ) {
   n *= level;
   System.out.println(level + "!=" + n);
   level++;
}
```

上述 while 迴圈計算從 1!到 5!的值，變數 level 是計數器變數，如果符合 level <= 5 條件，就可以進入迴圈執行程式區塊，迴圈的結束條件為 level > 5。

後測式 do/while 迴圈敘述

do/while 和 while 迴圈敘述的差異是在迴圈的結尾檢查結束條件，因此 do/while 迴圈的程式區塊至少會執行一次，例如：使用 do/while 迴圈顯示攝氏轉華氏的溫度轉換表，如下所示：

```
do {
   f = (9.0 * c) / 5.0 + 32.0;
   System.out.println(c + "\t" + f);
   c += step;
} while ( c <= upper);
```

上述迴圈的第一次執行需要到迴圈的結尾，才會檢查 while 條件是否為 true，如果 true 就繼續執行迴圈，計算從 lower 到 upper 間的溫度轉換，變數 c 是計數器，迴圈每次的增量是 step 變數的值，迴圈的結束條件為 c > upper。

A-5-3 break 和 continue 指令敘述

Java 迴圈預設是在開頭或結尾測試結束條件，但是有些時候，我們需要在迴圈中測試迴圈條件，以便決定中斷或繼續迴圈的執行。

break 指令中斷迴圈

break 指令可以在指定的條件成立時，強迫終止迴圈的執行，如同 switch 條件敘述使用 break 指令敘述跳出程式區塊一般，如下所示：

```
do {
   System.out.println("|" + i);
   total += i;
   i++;
   if ( i > 10 )
     break;
} while ( true );
```

上述程式碼當計數器變數 i > 10 時跳出迴圈，所以，這個 do/while 是無窮迴圈，使用 break 指令敘述控制迴圈的結束，所以一樣可以計算 1 加到 10 的總和。

continue 指令繼續迴圈

continue 指令敘述對應 break 指令，可以馬上繼續下一次迴圈的執行，不過它並不會執行程式區塊位在 continue 指令敘述後的程式碼，如果使用在 for 迴圈，一樣會自動更新計數器變數，如下所示：

```
for ( i = 1; i <= 10; i++ ) {
   if ( (i % 2) == 0 )
     continue;
   System.out.println("|" + i);
   total += i;
}
```

上述程式碼是當計數器變數為偶數時，繼續迴圈的執行，所以，其後的 System.out.println()和 total += i 兩列程式碼並不會執行。

A-5-4 巢狀迴圈

巢狀迴圈是在迴圈內擁有其他迴圈，例如：在 for 迴圈擁有 for、while 和 do/while 迴圈，同樣的，while 迴圈內也可以擁有 for、while 和 do/while 迴圈。

Java 語言的巢狀迴圈可以有很多層，二、三、四層都可以，例如：一個二層的巢狀迴圈，在 for 迴圈內擁有 while 迴圈，如下所示：

```
for ( i = 1; i <= 9; i++ ) {
   …
   j = 1;
   while ( j <= 9 ) {
     …
     j++;
   }
}
```

上述迴圈共有兩層，第一層的 for 迴圈執行 9 次，第二層的 while 迴圈也是執行 9 次，兩層迴圈總共可執行 81 次，

A-6 Java 語言的方法

Java 程序是一種類別成員，稱為「方法」（Methods），簡單的說，在 Java 語言的程序或函數稱為方法。Java 方法分為屬於類別的「類別方法」（Class Methods）和物件的「實例方法」（Instance Methods）兩種。

A-6-1 建立 Java 的類別方法

Java 類別方法是由方法名稱和程式區塊組成，屬於一種「靜態方法」（Static Method），因為使用 static「修飾子」（Modifiers），例如：一個沒有傳回值和參數列的方法範例，如下所示：

```
private static void writeTriangle() {
   int i, j;
   for ( i = 1; i <= 5; i++) {
      for ( j = 1; j <= i; j++)
         System.out.print("*");
      System.out.print("\n");
   }
}
```

上述方法的傳回值型態為 void，表示沒有傳回值，方法名稱為 writeTriangle()，括號內定義傳入的參數列，不過這個方法並沒有任何參數，在「{」和「}」括號是方法的程式區塊，在最前面的「存取敘述」（Access Specifier）是一種修飾子，可以是 public 和 private，如下所示：

- **public**：這個方法可以在程式任何地方進行呼叫，甚至是其他類別。
- **private**：這個方法只能在同一個類別內進行呼叫。

Java 方法的呼叫需要使用類別名稱或方法名稱，因為上述 writeTriangle() 方法沒有傳回值和參數列，所以呼叫方法只需使用方法名稱，加上空的括號，如下所示：

```
writeTriangle();
```

A-6-2 類別方法的參數傳遞

Java 方法的參數列是資訊傳遞的機制，可以從外面將資訊送入程序的黑盒子，參數列是方法的使用介面。一個方法如果擁有參數列，在呼叫方法時，傳入不同的參數就可以產生不同的執行結果，一個擁有參數列 printTable()方法的範例，如下所示：

```
static void printTable(int lower, int upper) {
   int step = 10;
   int c = lower;
   double f;
   System.out.println("攝氏     華氏");
   do {
      f = (9.0 * c) / 5.0 + 32.0;
      System.out.println(c + "\t" + f);
      c += step;
   } while ( c <= upper);
}
```

上述 printTable()方法可以顯示指定範圍的溫度轉換表，因為方法擁有參數列，所以在呼叫時需要加上參數列，如下所示：

```
printTable(30, upper);
```

A-6-3 類別方法的傳回值

如果 Java 的方法的傳回值型態不是 void，而是資料型態 int 或 char 等，表示方法擁有傳回值，此時稱為「函數」（Functions）。因為方法在執行完程式區塊後，需要傳回一個值，一個擁有傳回值的方法範例，如下所示：

```
static int nAdd2N(int begin, int end) {
   int i;
   int total = 0;
   for ( i = begin; i <= end; i++ )
      total += i;
   return total;
}
```

上述 nAdd2N()方法的傳回值型態為 int，可以計算傳入參數 begin 到 end 的總和，在使用 for 迴圈計算總和後，以 return 指令傳回方法的執行結果。

如果方法擁有傳回值，在呼叫時可以使用指定敘述取得傳回值，如下所示：

```
total = nAdd2N(5, 15);
```

上述變數 total 可以取得方法的傳回值，而且此變數的資料型態需要與方法傳回值的型態相符。