



程式碼實作學習

Java Steps: Learning Java Step by Step

Release 1.0

董少桓、林彥宏

September 28, 2011

CONTENTS

1	簡介	1
1.1	Java 程式語言的特色	1
1.2	安裝 JDK	2
1.3	編譯及執行 Java 程式	2
2	Java 的變數、方法與型態	5
3	區域變數與基本資料型態	9
4	螢幕輸出及鍵盤輸入	11
5	算術運算式	13
6	類別變數與類別方法	15
7	運算式、句子與條件判斷句	21
7.1	關係運算式	21
7.2	邏輯運算式	21
7.3	運算子的優先順序	22
7.4	Java 條件判斷句的種類	23
8	迴圈與遞迴	33
8.1	迴圈	33
8.2	遞迴	43

簡介

1.1 Java 程式語言的特色

Java 程式語言起源於 1991 年，Green Team 軟體開發團隊用它來開發 Star 7 機器上的應用程式，當時設計此語言的 James Gosling 因為看見窗外的「橡樹 (oak)」，決定將新語言命名為 Oak。但是由於工程師們喜歡邊喝咖啡邊討論，隨後又將名稱改為 Java（一種咖啡的名稱），這個名稱就一直沿用到現在。

Java 原本是為了控制冰箱、冷氣、微波爐等家電用品而設計的程式語言。由於家電用品相當多樣，因此 Java 選用了一個與傳統的程式語言不一樣的執行模式：

- 傳統的程式語言在編譯後會產生 machine code（機器碼），然後直接在硬體上執行；
- Java 在編譯後則會產生 Byte Code 並間接的在 Java Virtual Machine (JVM) 上執行。這個 JVM (Java 虛擬機器) 其實是一個軟體，其功用是解譯並執行 Byte Code，而 JVM 仍然是在硬體上執行。

因為 JVM 是軟體，所以 Java 也有跨平台的特性：只要為不同的處理器或作業系統設計其專屬的 JVM，Java 的程式便可以不需改寫的在這些處理器或作業系統上執行。這便是「Write once, runs everywhere 或一次編譯、到處執行」的由來。

Java 也支援物件導向程式設計 (Object-Oriented Programming)，所謂「物件」，簡單的說有「屬性」也有「方法」，例如冷氣機的「屬性」可以包括：「開關」及「溫度」；而「方法」則可以包括：「開機」、「關機」及「設定溫度」等。為了讓程式設計師，可以比較容易的使用物件撰寫模擬、控制與應用電腦本身（如滑鼠與鍵盤等也是物件）和我們生活周遭的物件的程式，因此便有研究人員發明了支援物件導向程式設計的語言。

Java 的設計搭上了全球資訊網的順風車，原因是 Java 的設計團隊可以寫一個能夠在瀏覽器中執行的 JVM，而讓 Java 的程式可以透過網路下載至瀏覽器中執行。這個「網路」＋「物件導向」的特性讓 Java 瞬間爆紅。

除了跨平台、物件導向、可透過網路動態的載入及執行程式等功能之外，Java 還支援多執行緒、例外狀態處理與自動記憶體回收的功能：

- 多執行緒讓一個程式可以執行數個工作；
- 例外狀態處理讓處理例外的程式碼也能夠物件化；
- 自動記憶體回收則讓程式設計師免除了使用低階的指標（pointers）來設計資料結構及管理記憶體的負擔。這個特色成了 C 語言程式設計師的福音，因為它可以為程式設計師減少許多不容易 debug 的錯誤。

1.2 安裝 JDK

在編譯與執行 Java 程式前，你的電腦必須先安裝 JDK（Java Development Kit）：

- 可以從這裡下載新版的 JDK <http://java.sun.com/javase/downloads/index.jsp>

在安裝完成後，也需要完成 path 及 classpath 的設定：

```
1 path=C:\Program Files\Java\jdk1.6.0\bin;....  
2 classpath=.;C:\Program Files\Java\jdk1.6.0\lib;....
```

請注意：以上路徑中的 jdk1.6.0 會因版本的不同而異。此外，在設定 classpath 時要特別注意在一號的右邊要輸入這個 **【.】**。這個點的意義是目前的目錄（current directory），也是執行 Java 程式時用來搜尋執行檔的目錄。

1.3 編譯及執行 Java 程式

有兩種方式可以編譯及執行一個 Java 程式。第一種是使用程式開發環境（program development environment），例如：[Eclipse](#)；另一種則是使用一般的程式編輯器。以下是使用「記事本」寫 Java 程式時所需要進行的三個步驟：

1. 使用「記事本」輸入以下的程式並將檔案命名為 EnglishExam.java（注意：附檔名必須是.java 而不是.txt，而這個檔案的主檔名必須與 public class 後面的 EnglishExam 相同）：

```
1 public class EnglishExam {  
2     public static void main(String argv[]) {  
3         System.out.println("Your score is 97.");  
4     }  
5 }
```

2. 執行「命令提示字元」並將目錄切換至儲存 EnglishExam.java 的目錄，然後執行：

```
1 javac EnglishExam.java
```

執行這個 `javac` 指令就是執行 Java 的編譯器 (compiler)，其結果是在同樣的目錄產生一個 `EnglishExam.class` 的 byte code 檔。

3. 上面的步驟如果有編譯錯誤則繼續修改程式。如果沒有編譯錯誤則可以執行：

```
1 java EnglishExam
```

執行後 `Your score is 97`。便會顯示在螢幕上。

在開發 Java 程式的過程中，有可能發生編譯錯誤 (compile-time error)。這時便需要再次的使用編輯器修改錯誤，直到沒有任何的編譯錯誤為止。編譯完畢之後，在程式執行時也有可能發生 run-time error。同樣的，這時也需要使用編輯器修改、編譯、執行、除錯，直到沒有錯誤為止。

一般的 Java 程式都是由一或多個類別 (class) 所組成，其中的一個類別至少要有一個命名為 **`public static void main`** 的方法 (method)，而這個程式就是由 `main` 開始執行。(透過網路瀏覽器執行的 Java applet 不適用此規則。)

上例中的 `public class EnglishExam` 是指定 `EnglishExam` 這個類別是 `public` 是公用的，也就是可以被程式中其他的類別引用。而 `public static void main(String argv[])` 的意義是：

1. `public`：指定 `main` 為一個可以被其他類別使用的 `public method`；
2. `static`：指定 `main` 為一個類別方法 (static method)，一個類別方法隸屬於一個 `class`；
3. `void`：代表 `main` 執行完畢後回傳的型態，因為 `main` 沒有回傳任何數值，因此它的回傳型態是 `void`；
4. `String argv[]`：指這個方法的輸入參數是 `argv[]` 而 `String` 則是它的型態。`main` 的輸入參數 `String argv[]` 可以在執行一個 Java 程式時將字串 (String) 資料輸入這個程式。例如在編譯以下的程式之後：

```
1 public class HelloJava {  
2     public static void main(String argv[]) {  
3         System.out.println("Hello " + argv[0] + argv[1]);  
4     }  
5 }
```

以「命令提示字元」執行：

```
1 java HelloJava Basic C++
```

便會呼叫 `System.out.println` 並輸出：

```
1 Hello Basic C++
```

這個程式的 `argv[]` 代表 `argv` 這個變數是一個陣列，而 `argv[0]`、`argv[1]` 則取用 `argv` 內第 0、1 個儲存格的内容。

Java 程式中 **用大刮號 { } 標示的 Block (區塊)** 是用來組織程式層次關係的語法。

例如上例的程式就有兩個區塊，一組用來標示 class 的區塊，另一組則用來標示 main 的區域。區塊中可以包含其他的區塊，在撰寫程式時也應注意要把區塊的內容往右縮排。一組用來標示類別的區塊內，可以有數個變數與方法。而一組用來標示方法的區塊內可以有一或多句以「;」結束的程式碼。這些程式碼共同構成了這個方法的 body。

為 Java 程式中使用的名字命名，有一個不成文的規定：**類別名稱的第一個字母要用大寫。方法或變數的第一個字母則是小寫**，若有數個字合併時則**後續的字的第一個字母也習慣用大寫**。

動手練習：修正程式碼的錯誤

JAVA 的變數、方法與型態

一個電腦程式基本上是由兩部分組成：

1. 資料：這部分的程式碼要為所處理的資料命名、指定其型態、並存放於記憶體中；
2. 處理：這部分的程式碼要使用撰寫在方法（method）中的運算式、條件判斷式、迴圈，來存取資料、計算結果、然後輸出。

這兩部分的內容（程式碼），不但是機器要知道如何執行，更要讓人（程式設計師）容易寫，也容易讀。而讓程式碼易寫、易讀的最基本的方法，就像收拾家中的衣櫃一般：按照種類與性質，分別放置整齊。

Java 語言整理程式碼的最基本的單元稱做「類別」。一個類別可以有儲存資料的「變數」與負責處理資料的「方法」。

「類別」可以提供兩種整理程式碼的方式：一種是以程式的邏輯結構當作分類的方式（俗稱的結構化程式設計）；另一種則是以物件的種類來歸類。而有的時候，一個類別也可以同時提供這兩種方式。

以程式的邏輯結構（例如將迴圈的邏輯放入一個方法中，或將一個判斷分數高低的邏輯放入一個方法中）來分類時，所使用到的變數稱為類別變數或 static field，而所用到的方法稱為類別方法或 static method。稱做 static（靜態）的原因是，這類的變數與方法是在程式開始執行時便在記憶體中產生了，而且它們的壽命一直到程式結束時才結束。

以物件來歸類的話，則是將類別內的程式碼，看成是產生這種類別的物件的「規格」。由於只是規格，所以只有在使用這個類別製造物件時，所對應的記憶體才會產生。而這種在程式執行時「動態」產生的物件實例中的變數與方法，稱為實例變數（instance variable），與實例方法（instance method）。

類別方法或實例方法，也需要自己有儲存資料的地方，而在類別方法或實例方法中儲存資料的變數，都叫區域變數（local variable），意思是在一個方法所屬的區域內才可以使用的變數。

Java 如何分區呢？主要是使用大刮號，例如：

```
1 {  
2     {  
3         ...  
4     }  
5     {  
6         ...  
7     }  
8     ...  
9 }
```

程式語言的句子與一般說話的語言一樣是由基本的字詞（names）組合而成。Java 為這些字詞命名的規定是一個字詞可以包含一個或多個英文字母、數字、_ 及 \$ 所組成的字元，而第一個字元不可以是數字。

Java 語言有 public, class, void, if, while, for 等 **保留字**。除了開始認識這些保留字的意義與用法之外，程式設計師所要學習的第一件事，就是為儲存資料的 **變數** 與執行處理的 **方法** 命名。

變數（variable） 是程式中的一種字詞。一個變數有一個 **名字（name）**、一個 **資料值（value）**、一塊儲存資料值的 **記憶體** 以及這個資料值的 **型態（type）**（如 int、double 等）。由於一個變數的型態，定義了這個變數的值所需要的記憶體的大小，所以一個 Java 程式在編譯時，就可以知道如何為這些變數，在執行時，配置適當大小的記憶體空間，以存放這些變數的值。

整的來說，Java 有三種變數：

1. **區域變數（local variable）**：宣告在方法內或參數部分的變數；
2. **類別變數（class variable or static field）**：在一個類別中以 static 宣告的變數；
3. **實例變數（instance variable or non-static field）**：在一個類別中沒有使用 static 宣告的變數。

Java 也有兩種方法：

1. **類別方法（class method or static method）**：這種方法以 static 宣告。呼叫的方式是 C.m(...)，其中 C 是類別名稱，m 是方法名稱，... 則是 0 至多個傳入的參數。
2. **實例方法（instance method or non-static method）**：這種方法不以 static 宣告，隸屬於一個類別所產生的實例。呼叫的方式是 o.m(...)，其中 o 是這個類別或其子類別的實例，而 m 是其方法名稱，... 則是 0 至多個傳入的參數。

Java 之所以有種類這麼多的變數與方法，是因為 Java 同時支援結構化（例如：C 與 Basic）與物件導向兩種普遍使用的程式設計方式。撰寫結構化程式時需要使用類別變數與類別方法。類別變數在概念上與結構化程式語言的全域變數（global variable）一

致；而類別方法在概念上則與結構化程式語言的函式（function）或程序（procedure）一致。實例變數、實例方法，則屬物件導向程式設計的功能。一般的 Java 程式可以同時使用結構化與物件導向並存的方式設計程式。

這份講義介紹 Java 結構化程式設計的語法及語意，另一份講義《Java 物件導向程式設計》則介紹 Java 物件導向程式設計的功能。

此外，Java 變數的型態也有兩大類：

1. **primitive type**，包括：`int`、`double`、`boolean`、`char` 等。
2. **reference type**，包括：
 - (a) 類別型態：經由類別（class）的宣告而得到。如果 `Car` 是一個類別，而 `aCar` 是一個這個類別的變數，則 `Car` 便是 `aCar` 的型態（type）。之所以稱為 `reference type`，是因為 `aCar` 這個變數在記憶體中的位置，實際上是存著指向（reference）一個 `Car` 實例的地址。
 - (b) 介面型態：經由介面（interface）的宣告而得到。
 - (c) 陣列（array）型態。
 - (d) `enum` 型態：一種特別的類別宣告方式，用於宣告月份、一週的七天等。

區域變數與基本資料型態

區域變數是一個方法的參數或是宣告在一個方法的區塊中。以下是宣告區域變數的幾個範例，其中 `int` 代表整數，而 `double` 代表倍精準浮點數，宣告的意義是告訴編譯器一個變數的型態是什麼：

```
1 public class EnglishExam {
2     public static void main(String argv[]) {
3         // argv 是傳入參數，同時也是一種區域變數
4
5         // 宣告三個區域變數
6         int vocabulary;
7         int grammar;
8         int listening;
9         ...
10    }
11 }
```

多個變數的宣告，可以合併在一行：

```
1 public class EnglishExam {
2     public static void main(String argv[]) {
3         // 將以上三個區域變數，合併在一行
4         int vocabulary, grammar, listening;
5         ...
6     }
7 }
```

宣告變數時，也可以同時指定數值：

```
1 public class EnglishExam {
2     public static void main(String argv[]) {
3         // 在宣告時也將數值存入這三個區域變數中
4         int vocabulary = 24;
5         int grammar = 26;
6         int listening = 33;
7         ...
8     }
9 }
```

```
8     }
9 }
1 public class EnglishExam {
2     public static void main(String argv[]) {
3         // 也可以這樣寫：
4         double vocabulary = 22.5, grammar = 25.4, listening = 32.0;
5         ...
6     }
7 }
```

有了變數之後，可以使用設值運算符號（=），將數值存入區域變數中，如果一個區域變數尚未被存入數值，則其預設值（default value）會被存入，而數字的預設值是 0。例如：

```
1 public class EnglishExam {
2     public static void main(String argv[]) {
3         int vocabulary, grammar, listening;
4         int score;
5
6         vocabulary = 22;
7         grammar = 26;
8         score = vocabulary + grammar + listening;
9
10        System.out.print("The score of the exam is ");
11        System.out.println(score);
12        // listening 的預設值是 0，所以印出 48
13    }
14 }
```

以上程式碼執行的結果為：

```
1 The score of the exam is 48
```

Java 的註解是以 // 或 /* */ 表示，例如：

```
1 // 這是註解
2 /*
3     這也是註解
4     這還是註解
5 */
```

螢幕輸出及鍵盤輸入

螢幕輸出有幾種方式。第一種是前面章節已經使用過的 `System.out.print` 及 `System.out.println`。這兩種方法的差別是前者沒有換行，而後者有換行。如果有數個資料需要一起印出時，則可以使用 `+` 進行串接。例如：

```
1 public class Show {
2     public static void main(String argv[]) {
3         int design, acting;
4
5         design = 3;
6         acting = 5;
7         System.out.println( "Design is " + design + "and acting is " + acting );
8     }
9 }
```

第二種方式是在 J2SDK5 之後才支援¹。這個方式與 C 語言的 `printf` 功能類似。例如：

```
1 System.out.printf("Today is %s, %d.\n", "January", 18);
2 // %s 的位置替換成 January 這個 String
3 // %d 的位置替換成 18 這個整數
4 // \n 表示換行符號
```

顯示：

```
1 Today is January, 18
```

例如：

```
1 double score = 92.345
2 System.out.printf("My score is %.2f.\n", score);
3 // %.2f 的意義是小數點以下取兩位，並四捨五入。
4 System.out.printf("My score is %6.2f.%n", score);
5 // %6.2f 的意義是：包括小數點共 6 位，小數點以下取兩位，
6 // 並四捨五入。所以 9 的左邊多空一格。
```

¹ `System.out.printf()`, <http://www.java2s.com/Code/JavaAPI/java.lang/System.out.printf.htm>

顯示：

```
1 My score is 92.35.  
2 My score is 92.35.
```

鍵盤輸入則可以透過²。例如：

```
1 // 使用時先載入 Scanner 所屬的 package  
2 import java.util.*; // * 的意義是 java.util 內所有的類別  
3  
4 // 定義物件：  
5 Scanner scanner = new Scanner(System.in);  
6  
7 // 輸入字串：  
8 String name = scanner.nextLine();  
9  
10 // 輸入整數：  
11 int score = scanner.nextInt();  
12  
13 // 輸入 double  
14 double height = scanner.nextDouble();  
15  
16 // 輸入 float  
17 float weight = scanner.nextFloat();
```

以下是一個完整的範例：

```
1 import java.util.*;  
2  
3 public class EnglishExam {  
4     public static void main(String argv[]) {  
5         int vocab, grammar, listen, score;  
6  
7         Scanner scanner = new Scanner(System.in);  
8         String name = scanner.nextLine();  
9         vocab = scanner.nextInt();  
10        grammar = scanner.nextInt();  
11        listen = scanner.nextInt();  
12        score = vocab + grammar + listen;  
13        System.out.printf("The total score of %s is %d.%n", name, score);  
14    }  
15 }
```

² java.util.Scanner, <http://www.java2s.com/Code/JavaAPI/java.util/Scanner.htm>

算術運算式

使用算術運算式要注意的是運算的優先次序。例如：先乘除、後加減。如果不記得運算的優先次序，那麼最簡便的方法是使用（）也就是指定內層的括號先執行。以下是算術運算式的幾個範例：

使用算術運算式另一項要注意的是型態的轉換，也就是在一個算術式中同時有整數與浮點數時，Java 會將整數先轉換成浮點數然後再進行運算。

例如：

```
1 int i = (3 + 4) / **3** // 兩個整數相除，結果仍是整數，小數部分捨棄。傳回 2
2 double f;
3 f = (3 + 4) / **3.0** // 7 轉型成浮點數，然後與浮點數 3.0 相除。傳回 2.333...
4 System.out.println(i);
5 System.out.println(f);
```

印出：

```
1 2
2 2.3333333333333335
```

如果需要指定轉換的型態，則可以使用以下的方式：

至於 i、f 的原來的值還是不變。

如同一般的算術運算式，設值運算式（=）也會傳回值。例如：

```
1 x = 3 // 傳回 3
2 y = (x = 3) // 因為 x = 3 傳回 3，所以 y 的值也設成 3
```

自動將資料範圍較小的型態轉為資料範圍較大的型態，稱為自動轉型 (promotion)。Java 資料型態範圍之大小次序為：

```
1 byte < short < int < long < float < double
```

以下的例子會出現 possible loss of precision 的錯誤：

這個錯誤是因為 a 會自動轉型成較大的 long 再跟 b 相加，但是 int 型態的 c 放不下 long 的資料。此時就要透過強制轉型 (casting)，將 a 轉型成 int：

```
1 int c = (int)(a + b);
```

當資料型態由小轉為大時，會自動轉型；當資料型態由大轉為小時，則需強迫轉型。

類別變數與類別方法

非物件導向程式語言（例如：C），程式設計師主要是使用函式（function、全域變數與區域變數將一個大的程式分割成幾個小的部份，以簡化程式的撰寫。這些觀念在 Java 中仍然可以使用，而使用的方式是透過類別變數與類別方法。

舉例而言，如果要為英文檢定考試寫一個計算成績的程式，那麼這個程式應該有一個計算成績的方法。例如：

```
1 public class EnglishExam {  
2     public static int englishScore(int v, int g, int l) {  
3         return v + g + l;  
4     }  
5     public static void main(String argv[]) {  
6         System.out.print("The score of the exam is ");  
7         System.out.println(englishScore(24, 27, 32));  
8     }  
9 }
```

執行結果：

```
1 The score of the exam is 83
```

`public static int englishScore(int v, int g, int l)` 定義了 `englishScore` 這個類別方法，這個方法有三個命名為 `v`, `g`, `l` 的輸入參數，它們的型態都是 `int`。這個方法的輸出型態也是 `int`，也就是會使用 `return` 傳出一個 `int` 的值 (`v + g + l`)。Java 使用 `static` 這個保留字來定義類別方法。因為這種方法是靜態的，也就是在程式執行時，呼叫這個方法的程式碼，一定都會執行相同的方法。

在 `main` 中的 `System.out.println(englishScore(24, 27, 32))` 將 24, 27, 32 傳入 `englishScore` 中，並依序成為 `v`, `g`, `l` 三個輸入參數的值，而這三個數相加的結果 83 會繼續的傳入 `System.out.println`，然後顯示在螢幕上。一個方法的輸入參數也是那個方法的區域變數。所以 `v`, `g`, `l` 三個輸入參數也是 `englishScore` 的區域變數。

除了直接將數值傳入方法中以外，還可以將變數或其他也有傳回值的式子，寫在方法

呼叫中傳入的位置。例如：

```
1 public class EnglishExam {
2     public static int englishScore(int v, int g, int l) {
3         return v + g + l;
4     }
5     public static void main(String argv[]) {
6         int a = 3, b = 4;
7         System.out.print("The score of the exam is ");
8         System.out.println(englishScore( **a, b, a + b** ));
9     }
10 }
```

執行結果：

```
1 The score of the exam is 14
```

Java 會在得到 a, b, a + b 的數值後，才將 3, 4, 7 傳入 englishScore 中。也就是先得到數值再傳入，然後 v, g, l 便使用傳入的數值生成三個區域變數。這個特性稱為 **call-by-value 或傳值呼叫**。v, g, l 三個參數之所以也是區域變數，因為這三個變數的可見範圍（scope）只包含 englishScore 的區塊。

如果一個方法沒有傳回值，那麼這個方法的輸出型態便是 void。例如：

```
1 public class EnglishExam {
2     public static **void displayScore** (int v, int g, int l) {
3         System.out.print("The score of the exam is ");
4         System.out.println(v + g + l);
5     }
6
7     public static void main(String argv[]) {
8         displayScore(24, 27, 32);
9     }
10 }
```

執行結果：

```
1 The score of the exam is 83
```

displayScore 這個方法將字串顯示在螢幕上，不需要傳回值，因此它的輸出型態是宣告成 void，而 main 的輸出型態也是 void。

不同的類別中也可以定義同名的方法。這個功能稱做 **overloading**。而 Java 是以 **類別名稱. 方法名稱 (0 或多個參數)**；呼叫宣告在不同類別的類別方法。例如：

```
1 public class Exam {
2     public static void main(String argv[]) {
3         int voc = 3, grammar = 7, listen = 8;
4         System.out.print("The score of the english exam is ");
5         System.out.println(EnglishExam.displayScore(voc, grammar, listen));
```

```

6         System.out.print("The score of simple english exam is ");
7         System.out.println(SimpleEnglishExam.displayScore(voc, grammar, listen));
8     }
9 }
10
11 class EnglishExam {
12     public static int displayScore(int v, int g, int l) {
13         return v + g + l;
14     }
15 }
16
17 class SimpleEnglishExam {
18     public static int displayScore(int v, int g, int l) {
19         return v + g + 0;
20     }
21 }

```

執行結果：

```

1 The score of the english exam is 18
2 The score of simple english exam is 10

```

一個類別中也可以有同名的方法，但是他們必須有不同的輸出入型態。例如：

```

1 public class Exam {
2     public static void main(String argv[]) {
3         int voc = 3, grammar = 7, listen = 8;
4         System.out.print("The int score of the exam is ");
5         System.out.println( **EnglishExam.displayScore(3, 7, 8)** );
6         System.out.print("The double score of the exam is ");
7         System.out.println( **EnglishExam.displayScore(3.0, 8.0, 7.0)** );
8     }
9 }
10
11 class EnglishExam {
12     public static int **displayScore(int v, int g, int l)** {
13         return v + g + l;
14     }
15     public static double **displayScore(double v, double g, double l)** {
16         return v + g + l;
17     }
18 }

```

執行結果：

```

1 The int score of the exam is 18
2 The double score of the exam is 18.0

```

另一個 overloading 的例子是：+。+可以用來將數字相加，也可以將字串合併。例如：

```
1 int a = 4, b = 5;
2 System.out.print(3 + a + b);
```

執行結果：

```
1 12
```

例如：

```
1 .. code-block:: java

    String a = ``xy", b = ``Z"; System.out.print(``3" + a + b);
```

執行結果：

```
1 3xyz
```

使用類別方法在程式中有許多好處：

1. 增加程式碼的再用性：同樣的計算步驟，只需要透過呼叫類別方法便可以重複使用。
2. 讓程式碼的細節，被隱藏在類別方法中：程式設計師在完成類別方法的撰寫後，便只需要知道那個類別方法的輸入、輸出與功用即可，而不用擔心執行的細節。
3. 容易除錯：除錯的過程可以一個類別方法、一個類別方法的進行，容易找出錯誤的根源。
4. 容易擴充類別方法內程式碼的功能：只要在類別方法內擴充其功能，而不用在每次呼叫時都重複的擴充。例如以下的程式碼擴充了成績的計算方式，所有 `displayScore` 的呼叫的計算結果都同步改變：

除了使用 `static` 宣告類別方法外，還有也是使用 `static` 宣告的類別變數。以下是一個在程式中內建三筆考試成績的資料，呼叫 `displayScore` 計算成績後，將三筆資料加總並存入 `total` 這個類別變數中的範例：

```
1 .. code-block:: java

    public class Exam { public static int total = 0;

        public static void main(String argv[]) { total = displayScore(3, 4, 5); // total = 12
            total = total + displayScore(4, 5, 6); // total = 27 total = total + displayScore(1, 2, 3); // total = 33 System.out.print(``The total score is ``);
            System.out.println(total);

        }

        public static int displayScore(int v, int g, int l) { return v + g + l;

        }

    }
```

```
    }
```

執行結果：

```
1 The total score is 33
```

程式設計師也可以使用不是定義在自己類別中的類別變數，而 Java 是以 **類別名稱. 變數名稱** 使用定義在其他類別中的類別變數。以下便是一種將 total 宣告在另一個類別 EnglishExam 中的寫法是：

```
1 public class Exam {
2     public static void main(String argv[]) {
3         EnglishExam.computeScore(3, 4, 5);
4         EnglishExam.computeScore(4, 5, 6);
5         EnglishExam.computeScore(1, 2, 3);
6         System.out.print("The total score is ");
7         System.out.println(EnglishExam.total);
8     }
9 }
10
11 class EnglishExam {
12     public static int total = 0;
13
14     public static void computeScore(int v, int g, int l) {
15         total = total + (v + g + l);
16     }
17 }
```

執行結果：

```
1 The total score is 33
```

以下則是一個為考試成績的計算，加入權重的範例。在這個範例中是以 Exam.wV, Exam.wG, Exam.wL 來使用這三個類別變數：

```
1 public class Exam {
2
3     public static double wV = 0.3, wG = 0.3, wL = 0.4;
4
5     public static void main(String argv[]) {
6         int voc = 3, grammar = 7, listen = 8;
7         System.out.print("The score of the english exam is ");
8         System.out.println(EnglishExam.displayScore(voc, grammar, listen));
9         System.out.print("The score of simple english exam is ");
10        System.out.println(SimpleEnglishExam.displayScore(voc, grammar, listen));
11    }
12 }
13
14 class EnglishExam {
```

```
15     public static double displayScore(int v, int g, int l) {
16         return v * Exam.wV + g * Exam.wG + l * Exam.wL;
17     }
18 }
19
20 class SimpleEnglishExam {
21     public static double displayScore(int v, int g, int l) {
22         return v * Exam.wV + g * Exam.wG + 0;
23     }
24 }
```

執行結果：

```
1 The score of the english exam is 6.2
2 The score of simple english exam is 3.0
```

類別變數與區域變數，在變數的可用「區域」與存在的「時間」上都不相同。類別變數若是定義為 `public`，則它的可用區域便包括整個程式，而且在整個程式執行時都存在。區域變數則是在程式執行到一個區塊或方法內時，那個區塊或方法的區域變數才存在，一旦離開那個區塊或方法，便消失了。因此區域變數的可用區域，只在定義該區域變數的區塊或方法內。

以下是一個「計算蛋與水果總價」的程式及其執行過程的動畫：

```
1 class Market {
2     static int sEgg = 5, sFruit = 20;
3     static int getMoney(int nEgg, int nFruit) {
4         return sEgg * nEgg + sFruit * nFruit;
5     }
6 }
7
8 public class Ex1 {
9     public static void main(String argv[]) {
10         int egg = 20, fruit = 30;
11         System.out.print("Money:");
12         System.out.println(Market.getMoney(egg, fruit));
13     }
14 }
```

執行結果：

```
1 Money:700
```

Media:Ex1new.swf 觀看執行過程

運算式、句子與條件判斷句

運算式（expression）執行完畢會後傳回一個值；句子（statement）在執行之後，則不回傳值。一個運算式與句子在執行時都可能會改變電腦的狀態，例如：更改變數、陣列或檔案的內容。算術運算式已經介紹了，而最簡單的句子，包括：

1. expression statement：在運算式後加一個 `;"` 。
2. block statement：在 { } 內可以有 0 或多個變數宣告或句子。

這兩種句子，也都已經看過範例。這個單元將介紹關係運算式、邏輯運算式及 if、switch 等句子。

7.1 關係運算式

以下是在寫關係運算式（relational expression）時常常會使用到的關係運算子（relational operator），整理如下：

- 需注意關係運算式使用的 `==`，和寫設值運算式（assignment expression）的 `=` 是不一樣的。

7.2 邏輯運算式

Java 常用的邏輯運算子（logical operators）有三個，分別是 && (AND)、|| (OR)、! (NOT)。邏輯運算式使用邏輯運算子以連結二個或以上的關係運算式。

- && (AND)

&& 是「AND（且）」運算：左右二邊「都」為 true，結果為 true；其餘情況其結果都為 false。

如果左邊為 `false`，則系統就不會去執行右邊；因為結果必為 `false`；既然左邊已經為 `false`，不管右邊結果為何，結果一定為 `false`。

- `||` (OR)

`||` 是「OR（或）」運算，左右二邊只要有一邊為 `true`，結果為 `true`；只有左右二邊都為 `false`，結果才為 `false`。

如果左邊為 `true`，則系統就不會去執行右邊；因為結果必為 `true`；即然左邊已經為 `true`，不管右邊結果為何，結果一定為 `true`。

- `!` (NOT)

`!` 是「NOT（相反）」運算，其結果為! 右邊的相反。

7.3 運算子的優先順序

以下是在在寫運算式時常常會使用到的運算子的優先順序，整理如下：

優先順序	運算子
1	((左括號)) (右括號) ++ (左遞增) -- (左遞減)
2	+ (正) - (負) ! (NOT) ++ (右遞增) -- (右遞減)
3	* (乘) / (除) % (取餘數)
4	+ (加) - (減)
5	< (小於) <= (小於等於) > (大於) >= (大於等於)
6	== (等於) != (不等於)
7	&& (AND)
8	(OR)
9	?: (條件判斷)

上表雖然不易記憶，然而，由於 ``(" 與 `") 的優先順序是最優先，所以在程式的設計過程中，我們可以藉由使用左、右括號來指定運算式的優先順序。

立即練習：

```

1  a = 48
2  b = 32
3  c = 55
4
5  d = a>b?a+b++*--c:a-b--*++c

```

1. 請問 `d` 的值是多少？

上面的程式碼，讀者可能需要查閱運算子優先次序，才能計算輸出的值是多少，雖然對電腦來說，可以很快就按照規則計算出正確的數字，但畢竟程式碼是由「人」撰寫

和維護，要記住這些瑣碎的規則可是一點都不容易。

請再嘗試以下的練習：

1. 在上一個練習的程式碼加上 () 括號，讓你自己可以很容易看懂計算的優先次序。

7.4 Java 條件判斷句的種類

Java 的條件判斷句 (statement) 可以分成以下 3 種：

1. if...
2. if... else...
3. switch

7.4.1 if 條件判斷句

Java 語言 if (...) {...} 條件判斷句的流程圖及語法如下：

Image If-1.png

```
1  if (/* 條件判斷 */) {  
2    //條件判斷成立時執行的程式碼  
3  }
```

說明：

1. 如果第 1 行的條件判斷成立 (值 == true)，則會執行第 2 ~ 4 行的程式碼 (圖形藍色的路徑)。而如果條件判斷不成立 (值 == false)，則會跳過第 2 ~ 4 行程式碼不執行 (圖形黑色 False 的路徑)。
2. 如果第 1 行的條件判斷成立時，執行的程式碼只有一行，則區塊符號可以省略不寫。如果超過一行，其區塊符號 { } 不可以省略。

Example 1：請設計一個 Java 程式，讓使用者自行輸入一個成績，判斷輸入的成績是否「大於或等於」60，如果是就輸出「成績及格！」。

Image If-2.png

```
1  import java.util.Scanner;  
2  class if_loop {  
3    public static void main(String[] args) {  
4      if_condition();  
5    }  
6  }
```

```
7 public static void if_condition() {
8     System.out.print(" 請輸入成績:");
9     Scanner scanner = new Scanner(System.in);
10    int grade = scanner.nextInt();
11
12    if (grade >= 60) {
13        System.out.println(" 成績及格!");
14    }
15 }
16 }
```

執行結果：

```
1 請輸入成績:60
2 成績及格!
```

說明：

- 輸入的 grade 為 60，條件判斷的結果為 true (60 >= 60)，會執行條件判斷成立區塊內的程式碼，輸出「成績及格!」（圖形藍色的路徑）。
- 因為範例中條件判斷成立時，執行的程式碼只有一行：System.out.println(`成績及格!`); 所以區塊符號可以省略不寫。

7.4.2 if...else...條件判斷句

Java 語言 if (...) {...} else {...} 條件判斷句的語法如下：

Image If-else-1.png

```
1 if (條件判斷)
2 {
3     條件判斷成立時執行的程式碼;
4     ....
5 }
6 else
7 {
8     條件判斷不成立時執行的程式碼;
9     ....
10 }
```

說明：

- 如果第 1 行的條件判斷成立（值 == true），則執行第 2 ~ 5 行區塊符號內的程式碼（圖形藍色的路徑）。
- 如果第 1 行的條件判斷不成立（值 == false），則跳過第 2 ~ 5 行的程式碼，而執行第 7 ~ 10 行區塊符號內的程式碼（圖形紅色的路徑）。

- 如果 if 條件判斷成立或不成立時，執行的程式碼只有一行，則該區塊符號可以省略不寫。而如果超過一行，則區塊符號不可以省略。
- if 和 else 是彼此互斥的關係，二個條件區塊在程式執行的過程中，只會選擇一個條件區塊去執行。

Example 2：請設計一個 Java 程式，讓使用者輸入一個成績，判斷該輸入的成績是否「大於或等於」60，如果是就輸出「成績及格!」；如果不是則輸出「成績不及格!」。

Image If-else-2.png

```
1 import java.util.Scanner;
2
3 class if_else_loop {
4     public static void main(String[] args) {
5         if_else_condition();
6         if_else_condition();
7     }
8
9     public static void if_else_condition() {
10        System.out.print(" 請輸入成績:");
11        Scanner scanner = new Scanner(System.in);
12        int grade = scanner.nextInt();
13
14        if (grade >= 60) {
15            System.out.println(" 成績及格!");
16        }
17        else {
18            System.out.println(" 成績不及格!");
19        }
20    }
21 }
```

執行結果：

```
1 請輸入成績:88
2 成績及格!
3 請輸入成績:59
4 成績不及格!
```

說明：

- 第一次輸入的 grade 為 88，其條件判斷的結果為 True (88 >= 60)，執行 if 成立區塊內的程式碼，輸出「成績及格!」（圖形藍色的路徑）。
- 第二次輸入的 grade 為 59，其條件判斷的結果為 False (59 >= 60)，執行 else 區塊內的程式碼，輸出「成績不及格!」（圖形紅色的路徑）。
- 因為 if 條件判斷成立和不成立時，執行的程式碼都只有一行，所以二者的區塊符

號都可以省略不寫。

7.4.3 數個 if 的條件判斷

幾個接續在一起的 if 句子，可以寫成：if (...) {...} else if (...) {...} else {...}：

[[Image:If-elseif-else-1.png]]

```

1  if (條件判斷 1) {
2      條件判斷 1 成立時執行的程式碼;
3      ....
4  }
5  else if (條件判斷 2) {
6      條件判斷 2 成立時執行的程式碼;
7      ....
8  }
9  :
10 :
11 else {
12     上述條件判斷都不成立時執行的程式碼;
13     ....
14 }
```

說明：

- 如果在第 1 行的條件判斷 1 成立（值 == true），則執行第 2 ~ 5 行區塊符號內的程式碼（圖形藍色的路徑），並略過其餘的程式碼。
- 如果在第 6 行的條件判斷 2 成立（值 == true），則執行第 7 ~ 10 行區塊符號內的程式碼（圖形綠色的路徑），並略過其餘的程式碼。
- 如果所有的條件判斷都不成立（值 == false），則跳過第 1 ~ 12 行的程式碼，而執行第 14 ~ 17 行區塊符號內的程式碼（圖形紅色的路徑）。
- if 、else if 和 else 是彼此互斥的關係，所有條件區塊在程式執行的過程中，只會選擇一個條件區塊去執行。

Example 3：請設計一個 Java 程式，讓使用者自行輸入一個成績，判斷該輸入的成績是屬於 A, B, C, D 或 E。

[[Image:If-elseif-else-2.png]]

```

1  import java.util.Scanner;
2  class if_elseif_else_loop {
3      public static void main(String[] args) {
4          if_elseif_else_condition();
5          if_elseif_else_condition();
6          if_elseif_else_condition();
```

```
7         if_elseif_else_condition();
8         if_elseif_else_condition();
9     }
10
11     public static void if_elseif_else_condition() {
12         System.out.print(" 請輸入成績:");
13         Scanner scanner = new Scanner(System.in);
14         int grade = scanner.nextInt();
15
16         if (grade >= 90)
17             System.out.println(" 成績為 A");
18         else if (grade >= 80)
19             System.out.println(" 成績為 B");
20         else if (grade >= 70)
21             System.out.println(" 成績為 C");
22         else if (grade >= 60)
23             System.out.println(" 成績為 D");
24         else
25             System.out.println(" 成績為 E");
26     }
27 }
```

執行結果：

```
1 請輸入成績:90
2 成績為 A
3 請輸入成績:89
4 成績為 B
5 請輸入成績:70
6 成績為 C
7 請輸入成績:60
8 成績為 D
9 請輸入成績:59
10 成績為 E
```

說明：

- 第一次輸入的 grade 為 90，符合條件判斷 1，輸出 ``成績為 A" (圖形藍色的路徑)。
- 第二次輸入的 grade 為 89，符合條件判斷 2，輸出 ``成績為 B" (圖形綠色的路徑)。
- 第三次輸入的 grade 為 70，符合條件判斷 3，輸出 ``成績為 C" (圖形粉紅色的路徑)。
- 第四次輸入的 grade 為 60，符合條件判斷 4，輸出 ``成績為 D" (圖形淺藍色的路徑)。

- 第五次輸入的 `grade` 為 `59`，都不符合上面的條件判斷，會執行 `else` 區塊內的程式碼，輸出 ``成績為 E" (圖形紅色的路徑)。
- 因為 `if` 條件判斷、所有 `else if` 條件判斷或 `else` 成立時，執行的程式碼都只有一行，所以區塊符號都可以省略不寫。

7.4.4 switch 與 break

break

- `break`：在程式執行時，遇到 `break`，會跳過目前執行區塊後的程式碼，並跳出目前的區塊。

switch

當需要對一個 `int`、`short`、`char`、`byte` 或是 `enum` 型態值做多種不同的判斷時，可以使用 `switch`，以下是 `switch` 的語法：

[[Image:Switch-1.png]]

```
1  1  switch (變數或運算式) {
2      2      case 值 1:
3          3          符合值 1 執行的程式碼;
4          4          ....
5          5          break;
6      6      case 值 2:
7          7          符合值 2 執行的程式碼;
8          8          ....
9          9          break;
10     10     :
11     11     :
12     12     case 值 n:
13         13         符合值 n 執行的程式碼;
14         14         ....
15         15         break;
16     16     default:
17         17         都不符合上述值執行的程式碼;
18         18         ....
19     19 }
```

說明：

- 第 1 行：`Switch` 後面括號內的程式碼，可以是變數（例如：`grade`）或是運算式，然而其型態必需是 `int`、`short`、`char`、`byte` 或是 `enum` 型態（圖形藍色菱形的部

份)。

- 第 2、6、12 行：用以判斷 switch 後面括號內的變數或運算式的值是否符合值 1（第 2 行；圖形藍色的路徑）、值 2（第 6 行；圖形綠色的路徑）、值 n（第 12 行；圖形粉紅色的路徑）的條件判斷。值 1、值 2、值 n 必須是常數（compile-time constant）。
- 第 (3~4)、(7~8)、(13~14) 行：如果符合值 1（圖形藍色的路徑）、值 2（圖形綠色的路徑）、值 n（圖形粉紅色的路徑）時會執行的程式碼。
- 第 16 行：如果都不符合 case 值 1 到 case 值 n 的情況下，則會去執行 default 區塊內（第 17、18 行；圖形紅色的路徑）的程式碼，然後離開整個 switch 的條件判斷。
- 第 5、9、15 行：當程式執行遇到 break 敘述，會結束 switch 的執行。
- 在 Java 語言中，switch 條件判斷的 case 的值只能是單一的常數值（compile-time constant），不可以是範圍值（例如： ≥ 90 ）。

Example 4：請使用 switch 來設計一個 Java 程式，讓使用者自行輸入一個成績，判斷該輸入的成績是介於哪一個區間之內。

[[Image:Switch-2.png]]

```

1 import java.util.Scanner;
2 class Switch_Statement {
3     public static void main(String[] args) {
4         switch_statement();
5         switch_statement();
6         switch_statement();
7         switch_statement();
8         switch_statement();
9         switch_statement();
10    }
11
12    public static void switch_statement() {
13        System.out.print(" 請輸入成績:");
14        Scanner scanner = new Scanner(System.in);
15        int grade = scanner.nextInt();
16        grade = (int)grade / 10;
17
18        switch (grade) {
19            case 10:
20            case 9:
21                System.out.println("90~100");
22                break;
23            case 8:

```

```

24         System.out.println("80~89");
25         break;
26     case 7:
27         System.out.println("70~100");
28         break;
29     case 6:
30         System.out.println("60~69");
31         break;
32     default:
33         System.out.println("0~59");
34     }
35 }
36 }

```

執行結果：

```

1  請輸入成績:100
2  90~100
3  請輸入成績:90
4  90~100
5  請輸入成績:89
6  80~89
7  請輸入成績:74
8  70~59
9  請輸入成績:60
10 60~69
11 請輸入成績:59
12 0~59

```

說明：

- 第一次輸入的 grade 為 100，符合 case 10 的條件判斷，但 case 10 區塊內沒有 break 敘述（圖形藍色的路徑）會繼續往下執行到 case 9 區塊內的程式碼，輸出 ``90~100"（圖形綠色的路徑），之後遇到 break 敘述跳離開整個 switch。
- 第二次輸入的 grade 為 90，符合 case 9 的條件判斷，會執行 case 9 區塊內的程式碼，輸出 ``90~100"（圖形綠色的路徑），之後遇到 break 敘述跳離開整個 switch。
- 第三次輸入的 grade 為 89，符合 case 8 的條件判斷，會執行 case 8 區塊內的程式碼，輸出 ``80~89"（圖形粉紅色的路徑），之後遇到 break 敘述跳離開整個 switch。
- 第四次輸入的 grade 為 74，符合 case 7 的條件判斷，會執行 case 7 區塊內的程式碼，輸出 ``70~79"（圖形淺藍色的路徑），之後遇到 break 敘述跳離開整個 switch。
- 第五次輸入的 grade 為 60，符合 case 6 的條件判斷，會執行 case 6 區塊內的程式碼，輸出 ``60~69"（圖形橘色的路徑），之後遇到 break 敘述跳離開整個 switch。
- 第六次輸入的 grade 為 59，都不符合上面 case 的條件判斷，會執行 default 區塊

內的程式碼，輸出 ``0~59"（圖形紅色的路徑），之後離開 switch。

7.4.5 巢狀條件判斷

巢狀條件判斷（nested-if）：是指在一個 if 裡面還有 if。

Example 5：請寫一個判斷使用者輸入的數是否是 2 或 3 或 6 的倍數，或者都不是他們的倍數的 java 程式。

[[Image:nested_if.png|link=]]

```
1  import java.util.Scanner;
2  public class Times {
3      public static Scanner keyboard = new Scanner(System.in);
4
5      public static void main(String[] args) {
6          times();
7          times();
8          times();
9          times();
10     }
11
12     public static void times() {
13         System.out.print(" 請輸入一個整數：");
14         int num = keyboard.nextInt();
15
16         if ((num % 2) == 0) {
17             if ((num % 3) == 0) {
18                 System.out.println(num + " 是 2、3、6 的倍數。");
19             }
20             else {
21                 System.out.println(num + " 是 2 的倍數。");
22             }
23         }
24         else {
25             if ((num % 3) == 0) {
26                 System.out.println(num + " 是 3 的倍數。");
27             }
28             else {
29                 System.out.println(num + " 都不是 2、3、6 的倍數。");
30             }
31         }
32     }
33 }
```

輸出結果：

```
1 請輸入一個整數：12
2 12 是 2、3、6 的倍數。
3 請輸入一個整數：4
4 4 是 2 的倍數。
5 請輸入一個整數：9
6 9 是 3 的倍數。
7 請輸入一個整數：11
8 11 都不是 2、3、6 的倍數。
```

說明：

- 當輸入的 num 為 12 時，第一層（外層）可被 2 整除（執行 if 區塊），再往第二層（內層）可被 3 整除（執行 if 區塊），輸出 ``12 是 2、3、6 的倍數。"（圖形藍色的路徑）。
- 當輸入的 num 為 4 時，第一層（外層）可被 2 整除（執行 if 區塊），再往第二層（內層）不可被 3 整除（執行 else 區塊），輸出 ``4 是 2 的倍數。"（圖形粉紅色的路徑）。
- 當輸入的 num 為 9 時，第一層（外層）不可被 2 整除（執行 else 區塊），再往第二層（內層）可被 3 整除（執行 if 區塊內），輸出 ``9 是 3 的倍數。"（圖形綠色的路徑）。
- 當輸入的 num 為 11 時，第一層（外層）不可被 2 整除（執行 else 區塊），再往第二層（內層）不可被 3 整除（執行 else 區塊），輸出 ``11 都不是 2、3、6 的倍數。"（圖形紅色的路徑）。

7.4.6 ? : 條件判斷式

條件判斷？條件判斷成立時執行的程式碼：條件判斷不成立時執行的程式碼；

if 與 ? : 的差別在於 if 不傳回值；而 ? : 可以傳回值（見以下範例）

Example 6：請設計一個 Java 程式，可以判斷成績是否及格。

```
1 if (grade >= 60)
2     message = " 成績及格!";
3 else
4     message = " 成績不及格!";
```

上面 Example 6 可以改寫成下面只有一行的 ? : 條件判斷式。

```
1 message = (grade >= 60) ? " 成績及格!" : " 成績不及格!";
2
3 message 的值是上例執行 ? : 運算式之後傳回的結果。
```

迴圈與遞迴

8.1 迴圈

我們在撰寫程式時，常常需要用到迴圈。像是要讓使用者輸入 10 位學生的成績，如果沒有使用迴圈，就必須將輸入成績的程式碼寫 10 次，而這 10 次的程式碼卻完全一樣。如果使用迴圈，便只需要寫 1 次就可以了。使用迴圈讓這輸入成績的程式碼可以連續執行 10 次，以達到相同的效果，因此可以讓程式設計的更精簡、有彈性。

Java 用來寫迴圈的句子有以下三個：

1. for
2. while
3. do-while

8.1.1 for-loop

for-loop 的語法如下：

```
1 for (控制變數的初始值設定； 控制變數的條件判斷； 控制變數值的改變) {  
2     符合控制變數的條件判斷時，執行的程式碼區塊；  
3     ....  
4 }
```

說明：

- for-loop 主要由四個部份所組成，分別是「控制變數的初始值設定」、「控制變數的條件判斷」、「控制變數值的改變」與「符合控制變數的條件判斷時，執行的程式碼區塊」。
- 「控制變數的初始值設定」：只有在迴圈第一次執行時才會執行，其目的是用來設定控制迴圈變數（loop control variable）初始值的設定。例如：`int i = 0;` 表示 for

迴圈一開始執行時，宣告一個 `int` 的控制變數，其值為 0。

- 「控制變數的條件判斷」：每一次執行 { } 內的程式碼區塊前，會先執行「控制變數的條件判斷」。如果符合該條件判斷，才會去執行迴圈內的程式碼；不符合便結束迴圈的執行。例如：`i < 5`；表示如果此時迴圈的控制變數 `i` 的值小於 5 才會去執行迴圈內的程式碼，否則便離開迴圈。
- 「控制變數值的改變」：每一次 `for` 執行完在 { } 內的程式碼區塊之後，會執行的部份。它會改變迴圈控制變數的值。例如：`i++`；表示每一次迴圈執行完之後，將變數 `i` 的值加 1。
- 「符合控制變數的條件判斷時，執行的程式碼區塊」（第 2 行的 { 至第 5 行的 }）：符合控制變數的條件判斷時，會執行的程式碼。例如：`System.out.println(i)`；表示將迴圈的控制變數 `i` 的值輸出。

Example 1：請利用 `for-loop` 來設計一個程式，讓迴圈執行 5 次，每次執行時都輸出迴圈執行到第幾次。

```
1 public class for_loop {
2     public static void main(String[] args) {
3         for (int i=1; i<=5; i++)
4             System.out.println("for 迴圈執行第" + i + " 次。");
5     }
6 }
```

執行結果：

```
1 for 迴圈執行第 1 次。
2 for 迴圈執行第 2 次。
3 for 迴圈執行第 3 次。
4 for 迴圈執行第 4 次。
5 for 迴圈執行第 5 次。
```

- 由於區塊內的程式碼只有一個句子，所以 { } 可以省略不寫。

Example 2：請利用 `for-loop` 搭配 `if` 條件判斷來設計一個 Java 程式，將 1 到 10 中的偶數輸出。

```
1 public class for_loop2 {
2     public static void main(String[] args) {
3         for (int i=1; i<=10; i++)
4             if (i % 2 == 0)
5                 System.out.println("1 到 10 的偶數有：" + i);
6     }
7 }
```

輸出結果：

```
1 1 到 10 的偶數有：2
2 1 到 10 的偶數有：4
3 1 到 10 的偶數有：6
4 1 到 10 的偶數有：8
5 1 到 10 的偶數有：10
```

8.1.2 while-loop

while-loop 是前測式的迴圈，其語法如下：

```
1 //控制變數的初始值設定；
2
3 while (控制變數的條件判斷) {
4     //符合控制變數的條件判斷時，執行的程式碼區塊；
5     //....
6     //控制變數值的改變；
7 }
```

說明：

- while-loop 和 for-loop 同樣由四個部分所組成，分別是「控制變數的初始值設定」、「控制變數的條件判斷」、「符合控制變數的條件判斷時，執行的程式碼區塊」與「控制變數值的改變」。
- while-loop 是「前測式迴圈」，因為控制變數的條件判斷位於迴圈一開始的地方，因此 while-loop 執行次數可以為 0 次，也就是控制變數的條件判斷在一開始便不符合。

Example 3：請利用 while-loop 設計一個 Java 程式，讓迴圈執行 5 次，每次執行時都輸出迴圈執行到第幾次。

```
1 public class while_loop {
2     public static void main(String[] args) {
3         int i = 1;
4         while (i <= 5) {
5             System.out.println("while 迴圈執行第" + i + " 次。");
6             i++;
7         }
8     }
9 }
```

執行結果：

```
1 while 迴圈執行第 1 次。
2 while 迴圈執行第 2 次。
3 while 迴圈執行第 3 次。
```

- 4 while 迴圈執行第 4 次。
- 5 while 迴圈執行第 5 次。

Example 4：請利用 while-loop 設計一個 Java 程式，這個程式一直讓使用者從鍵盤輸入數字，直到輸入 -1 時為止。這個程式輸出這些數字的和。

```
1 import java.util.Scanner;
2 public class Sentinel {
3     static Scanner keyboard = new Scanner(System.in);
4     static int i = -1;
5
6     public static void main(String[] args) {
7         int total = 0;
8         int s = 0;
9
10        System.out.print("Please enter score or enter -1 to stop: ");
11        s = keyboard.nextInt();**
12        while (s != i) {
13            total = total + s;
14            **System.out.print("Please enter score or enter -1 to stop: ");
15            s = keyboard.nextInt();
16        }
17        System.out.println("The sum of scores: "+ total);
18    }
19 }
```

輸出結果：

```
1 Please enter score or enter -1 to stop: 3
2 Please enter score or enter -1 to stop: 5
3 Please enter score or enter -1 to stop: 7
4 Please enter score or enter -1 to stop: 9
5 Please enter score or enter -1 to stop: -1
6 The sum of scores: 24
```

- 這個程式的特性是輸入數字的程式碼，在進入迴圈前及迴圈尾各出現一次。
- 這種迴圈叫旗標控制迴圈（sentinel-controlled loop）。

8.1.3 do-while

do-while loop 是一種後測式迴圈，其語法如下：

說明：

- do-while loop 和 for-loop、while-loop 同樣由四個部分所組成，分別是「控制變數的初始值設定」、「控制變數的條件判斷」、「符合控制變數的條件判斷時，執行的程

式碼區塊」與「控制變數值的改變」。

- do-while loop 是「後測式迴圈」，因為控制變數的條件判斷位於最後面。
- do-while loop 最少會執行 1 次。因為一開始會先執行 do 區塊內的程式碼，之後才會進行 while 之後控制變數的條件判斷。
- 特別留意在最一行 while 控制變數的條件判斷後面，需加上「;」以表示 do-while loop 的結束。

Example 5：請利用 do-while loop 設計一個 Java 程式，讓迴圈執行 5 次，每次執行時都輸出迴圈執行第幾次。

```
1 public class do_while_loop {
2     public static void main(String[] args) {
3         **int i = 1;
4         do {
5             System.out.println("do while 迴圈執行第" + i + " 次。");
6             i++;
7         } while (i <= 5);**
8     }
9 }
```

執行結果：

```
1 do while 迴圈執行第 1 次。
2 do while 迴圈執行第 2 次。
3 do while 迴圈執行第 3 次。
4 do while 迴圈執行第 4 次。
5 do while 迴圈執行第 5 次。
```

Example 6：請利用 do-while loop 搭配 if 條件判斷來設計一個 Java 程式，將 1 到 10 中間的偶數輸出。

```
1 public class do_while_loop2 {
2     public static void main(String[] args) {
3         **int i = 1;
4         do {
5             if (i % 2 == 0)
6                 System.out.println("1 到 10 的偶數有：" + i);
7             i++;
8         } while (i <= 10);**
9     }
10 }
```

執行結果：

```
1 1 到 10 的偶數有：2
2 1 到 10 的偶數有：4
3 1 到 10 的偶數有：6
```

- 4 1 到 10 的偶數有：8
- 5 1 到 10 的偶數有：10

8.1.4 break & continue

====break====

- break：可以離開目前的 switch、for、while、do while 的區塊，並跳離至區塊後的下一行程式碼。在 switch 中主要用來離開；而在 for、while 與 do while 迴圈中，主要用於中斷目前迴圈的執行。

Example 7：下面是一個使用 break 敘述的 Java 程式，觀察程式碼及程式執行的過程。

執行結果：

- 1 迴圈執行第 1 次。
- 2 迴圈執行第 2 次。
- 3 迴圈執行第 3 次。
- 4 迴圈執行第 4 次。

- 此迴圈只會執行 4 次，因為當 i==5 時就會執行 break; 述敘而離開迴圈。

8.1.5 continue

- continue：作用與 break 敘述類似，主要使用於 for、while、do while 迴圈，所不同的是 break 敘述會結束迴圈區塊的執行，而 continue 只會結束目前執行中區塊的程式碼，並跳回迴圈區塊的開頭繼續下一迴圈，而不是離開迴圈。

Example 8：下面是一個使用 continue 敘述的 Java 程式，觀察程式碼及程式執行的過程。

```
1 public class continue1 {
2     public static void main(String[] args) {
3         for (int i=1; i<=10; i++) {
4             if (i == 5)
5                 continue;
6             System.out.println(" 迴圈執行第" + i + " 次。");
7         }
8     }
9 }
```

執行結果：

- 1 迴圈執行第 1 次。
- 2 迴圈執行第 2 次。
- 3 迴圈執行第 3 次。

- 4 迴圈執行第 4 次。
- 5 迴圈執行第 6 次。
- 6 迴圈執行第 7 次。
- 7 迴圈執行第 8 次。
- 8 迴圈執行第 9 次。
- 9 迴圈執行第 10 次。

- 此迴圈會執行 1 ~ 4 加 6 ~ 9 次，當 $i==5$ 時，會執行 `continue`；述敘而跳離此次迴圈，然後從區塊開頭執行下一次迴圈，所以 $i==5$ 並沒有被執行。

8.1.6 巢狀迴圈

- 巢狀迴圈（nested loop）是指一個迴圈裡面還存在一個以上的迴圈。
- 巢狀迴圈執行的總次數為每一個迴圈執行次數的乘積。例如：一個輸出九九乘法表程式的執行次數，就是外層的 8 次乘以內層的 9 次，因此得到 $8 \times 9 = 72$ 次。
- 巢狀迴圈裡面的每一個迴圈都不可以與其它迴圈重疊，只能是彼此包含的關係。

Example 9：請使用巢狀迴圈設計一個輸出九九乘法表的程式，並計算巢狀迴圈總共執行了幾次。

```

1 public class NineNineTable {
2     public static void main(String[] args) {
3         int count = 0;
4         for (int i=2; i<=9; i++) {
5             for (int j=1; j<=9; j++) {
6                 System.out.printf("%d*%d=%2d ", i, j, i*j);
7                 count++;
8             }
9             System.out.printf("\n");
10        }
11        System.out.printf(" 巢狀迴圈總共執行了%d 次 \n", count);
12    }
13 }
```

執行結果：

```

1 2*1= 2 2*2= 4 2*3= 6 2*4= 8 2*5=10 2*6=12 2*7=14 2*8=16 2*9=18
2 3*1= 3 3*2= 6 3*3= 9 3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27
3 4*1= 4 4*2= 8 4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36
4 5*1= 5 5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45
5 6*1= 6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54
6 7*1= 7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63
7 8*1= 8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72
8 9*1= 9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
9 巢狀迴圈總共執行了 72 次
```

Example 10：請使用巢狀迴圈設計一個 Java 程式，這個程式持續輸入兩個整數 m, n，利用 while-loop 計算 m 的 n 次方，直到輸入 999 為止。

```
1  import java.util.Scanner;
2  public class PowerOf {
3      public static Scanner input = new Scanner(System.in);
4
5      public static void main(String[] args) {
6          int n, m;
7
8          System.out.println("Input: ");
9          m = input.nextInt();
10
11         while (m != 999) {
12             n = input.nextInt();
13
14             System.out.print(m + "^" + n + "=");
15
16             int result = 1;
17             while (n > 0) {
18                 result *= m;
19                 n--;
20             }
21
22             System.out.println(result);
23
24             System.out.println("Input: ");
25             m = input.nextInt();
26         }
27     }
28 }
```

-----Result-----

```
1  Input:
2  3
3  4
4  3^4=81
5  Input:
6  4
7  5
8  4^5=1024
9  Input:
10 5
11 6
12 5^6=15625
13 Input:
14 999
```

說明：

- 外層的 while 迴圈控制程式執行到輸入 999 才結束。
- 內層的 while 迴圈計算 m 的 n 次方。

8.1.7 for-loop、while-loop 與 do-while-loop 的相互轉換

- for-loop、while-loop、do-while-loop 可以彼此相互轉換，前面曾經介紹這三種迴圈主要由 4 個部分所組成，我們可以藉由搬動這 4 個部分的程式碼來完成迴圈的相互轉換。
- for-loop -> while-loop 可以遵從下面三個步驟：
 1. 將「控制變數的初始值設定」從 for-loop 拉到 while-loop 前面。
 2. 將「控制變數的條件判斷」保留在 while-loop 的「控制變數的條件判斷」內。
 3. 將「控制變數值的改變」拉到 while-loop 的「符合控制變數的條件判斷時，執行的程式碼區塊」內的最後一行。
- while-loop -> do-while-loop：
 1. 將 while-loop 的「控制變數的條件判斷」拉到 do-while-loop 最後一行的「控制變數的條件判斷」內即可完成。

Example 11：請分別利用 for、while 與 do-while 設計計算 1 加到 100 的程式。

```
1 public class Loop_Transfer1 {
2     public static void main(String[] args) {
3         // for-loop
4         System.out.println("for 迴圈 1 加到 100 為" + for_loop());
5
6         // while-loop
7         System.out.println("while 迴圈 1 加到 100 為" + while_loop());
8
9         // do-while-loop
10        System.out.println("do-while 迴圈 1 加到 100 為" + do_while_loop());
11    }
12
13    public static int for_loop() {
14        int sum = 0;
15        for(int i=1; i<=100; i++)
16            sum += i;
17        return sum;
18    }
19
20    public static int while_loop() {
```

```

21     int sum = 0;
22     int i = 1;
23     while (i <= 100) {
24         sum += i;
25         i++;
26     }
27     return sum;
28 }
29
30 public static int do_while_loop() {
31     int sum = 0;
32     i = 1;
33     do {
34         sum += i;
35         i++;
36     } while (i <= 100);
37     return sum;
38 }
39 }

```

執行結果：

for 迴圈 1 加到 100 為 5050 while 迴圈 1 加到 100 為 5050 do-while 迴圈 1 加到 100 為 5050

Example 12：請分別利用 for、while 與 do-while 設計計算 1 到 100 中間 3 的倍數的總合。

```

1  public class Loop_Transfer2 {
2      public static void main(String[] args) {
3          // for-loop
4          System.out.println("for 迴圈 1 到 100 間 3 的倍數總合為" + for_loop());
5
6          // while-loop
7          System.out.println("while 迴圈 1 到 100 間 3 的倍數總合為" + while_loop());
8
9          // do-while-loop
10         System.out.println("do-while 迴圈 1 到 100 間 3 的倍數總合為" + do_while_loop());
11     }
12
13     public static int for_loop() {
14         int count = 0;
15         for (int i=1; i <=100; i++)
16             if (i % 3 == 0)
17                 count += i;
18         return count;
19     }

```

```

20
21 public static int while_loop() {
22     int count = 0;
23     int i = 1;
24     while (i <= 100) {
25         if (i % 3 == 0)
26             count += i;
27         i++;
28     }
29     return count;
30 }
31
32 public static int do_while_loop() {
33     int count = 0;
34     int i = 1;
35     do {
36         if (i % 3 == 0)
37             count += i;
38         i++;
39     } while (i <= 100);
40     return count;
41 }
42 }

```

-----Result-----

```

1 for 迴圈 1 到 100 間 3 的倍數總合為 1683
2 while 迴圈 1 到 100 間 3 的倍數總合為 1683
3 do-while 迴圈 1 到 100 間 3 的倍數總合為 1683

```

8.2 遞迴

- 遞迴是一個相當獨特，對訓練邏輯思考很有助益的程式設計方式，為了能循序漸進的闡述遞迴程式的撰寫，在本節中，我們將以 String 資料型態當作程式的輸入資料，並以 1 來表示 boolean 的 true；而以 0 來表示 boolean 的 false。
- 我們需要使用 Integer.parseInt(<String Variable>) 將 String 型態的資料轉換為 int 型態的資料。其轉換過後的值如果 == 1，則代表 true；如果 != 1，則代表 false。

[註] Java 的每一個 primitive type 都有一個與之對應的 reference type 或類別，例如：int 與 Integer。parseInt 則是 Integer 的一個類別方法。Java 這麼設計的原因，會在《Java 物件導向程式設計》中的「泛型與 Collection」部分說明。

8.2.1 簡易的遞迴程序

以下是呼叫 `and2` 的範例：

```
1 System.out.println(and2("11")); //=> true
2 System.out.println(and2("10")); //=> false
3 System.out.println(and2("01")); //=> false
4 System.out.println(and2("00")); //=> false
```

`and2` 的特性是，它的參數中的兩個資料值都要是 1 結果才是 `true`，要不然結果就是 `false`。

現在我們試著將 `and2` 的功能擴充，並將擴充後的方法命名為 `allTrue`。而 `allTrue` 可以判斷一個 `string` 中的值是否都是 `true`，這個 `string` 可以有不止兩個資料值。以下是呼叫 `allTrue` 的範例：

```
1 System.out.println(allTrue(""));      => true   輸入參數中沒有一個 false，所以結果是 true。
2 System.out.println(allTrue("111"));   => true
3 System.out.println(allTrue("1111"));  => true
4 System.out.println(allTrue("11010")); => false
```

以下是 `allTrue` 的寫法：

```
1 public class Recursive {
2     public static void main(String args[]) {
3         System.out.println(allTrue(""));      // => true   輸入參數中沒有一個 false 的值，所
4         System.out.println(allTrue("111"));   // => true
5         System.out.println(allTrue("1111"));  // => true
6         System.out.println(allTrue("11010")); // => false
7     }
8
9     public static boolean allTrue(String str) {
10        if (str.equals(""))
11            return true;
12        else if (Integer.parseInt(str.substring(0, 1)) == 1 )
13            return allTrue(str.substring(1));
14        else
15            return false;
16    }
17 }
```

- `Integer.parseInt()`：將 () 裡的 `string` 由 `String` 型態轉換為 `int` 型態，以提供 `if` 來判斷其為 `true(== 1)` 或 `false(!= 1)`。
- `str.substring(0, 1)`：取出 `str` 字串變數中的第 0 個位置 (index) 至第 1 個位置 (index) 間的資料。(例如："1234" => "1")
- `str.substring(1)`：取出 `str` 字串變數中第 1 個位置以後的資料。(例如："1234" =>


```
``234")
```

`allTrue(`111");` 的執行過程可 trace 如下：

```
1   allTrue("111")
2 => allTrue("11")
3 => allTrue("1")
4 => allTrue("")
5 => true
```

`allTrue(`1101");` 的執行過程可 trace 如下：

```
1   allTrue("1101")
2 => allTrue("101")
3 => allTrue("01")
4 => false
```

觀察 `allTrue` 這個程序以及它的執行過程，我們發現：

1. `allTrue` 內有三個執行的可能（if 條件判斷的三個條件情況）。
2. 其中一行呼叫 `allTrue` 自己。這個呼叫自己的遞迴呼叫，傳入少了頭一筆資料值的 `String(str.substring(1))`。另外兩行沒有遞迴呼叫，而程式執行到這兩行之後便傳回 `true` 或 `false`，然後結束。
3. if 條件判斷式，控制程式執行哪個式子。
4. `allTrue` 這個程式的輸出入型態是：`String -> boolean`，也就是傳入一個 `String`、傳出一個布林值的意思。

一般而言，遞迴程式都有類似的模式：

1. 有終止條件（termination condition）如：`str.equals("")`、`n == 0`。
2. 有遞迴呼叫，而遞迴呼叫所傳入的資料一定會趨近終止條件（例如使用 `substring(1)` 使 `string` 愈來愈縮短）。

這很合理，因為這樣程式才能結束。

這個單元的練習都是 `String -> boolean` 型態的程序。我們還會在後續的單元，陸續的介紹遞迴程式的其他型式。另外，在撰寫程式的過程中，最簡單的除錯工具是把變數的值印出來。`System.out.println(<expression>)` 與 `System.out.println()` 是可以顯示一個式子的傳回值與換行的兩個程序：

執行以上的程式碼會列印與傳回以下的內容：

```
1 str=111
2 str=11
3 str=1
4 str=
5 true
```

前面 4 行與最後 1 行分別顯示了每次呼叫 `allTrue` 的輸入參數的值與最後的執行結果。

8.2.2 累計答案值的遞迴程序

上一單元我們練習了幾個傳回 `true` 或 `false` 的程序。這些題目的答案不需要經過累計，例如：累加、累乘、或累積成字串的過程。這個單元我們將練習需要經過累計才能計算出答案的遞迴程序。

假設我們需要撰寫一個能夠計算一個 `string` 中有幾個 `true` 的程序。我們將這個程序命名為 `countTrue`：

```
1 countTrue("");      => 0
2 countTrue("1101"); => 3
```

很顯然的，在程序執行的過程中如果遇到 1(true) 則需要將 1 加入結果之中。寫法如下：

```
1 public class Recursive {
2     public static void main(String args[]) {
3         System.out.println(countTrue(""));      // => 0 輸入參數中沒有一個 false 的值，所以
4         System.out.println(countTrue("1"));      // => 1
5         System.out.println(countTrue("01"));      // => 1
6         System.out.println(countTrue("101"));      // => 2
7         System.out.println(countTrue("1101"));     // => 3
8     }
9
10    public static int countTrue(String str) {
11        if (str.equals(""))
12            return 0;
13        else if (Integer.parseInt(str.substring(0, 1)) == 1) // 1(true)
14            return 1 + countTrue(str.substring(1));
15        else // 0(false)
16            return countTrue(str.substring(1));
17    }
18 }
```

我們可以追蹤這個程式是怎麼執行的：
`countTrue("")`; `""` 使 `str.equals("")` 成立因此傳回 0 => 0

`CountTrue("1")`; `"1"` 使 `(Integer.parseInt(str.substring(0, 1)) == 1)` 成立因此傳回 1

=> `(1 + countTrue(""))`; => `(1 + 0)` => 1

`countTrue("01")`; `"01"` 使 `else` 成立因此傳回 1

=> `countTrue("1")`; => 1

`countTrue("`101");`101"` 使 `(Integer.parseInt(str.substring(0, 1)) == 1)` 成立因此傳回 2

$\Rightarrow (1 + \text{countTrue}("`01")); \Rightarrow (1 + 1) \Rightarrow 2$

接下來，我們可以追蹤將一個有四個資料值的 string 傳入 `countTrue` 的執行狀況：

```

1    countTrue("1101");
2    => (1 + countTrue("101");)
3    => (1 + (1 + countTrue("01");))
4    => (1 + (1 + countTrue("1");))
5    => (1 + (1 + (1 + countTrue("");)))
6    => (1 + (1 + (1 + 0)))
7    => 3

```

另外，`countTrue` 的型態是：`String -> int`。

8.2.3 將遞迴程序轉換成迴圈程序

1. 階乘（factorial）是一個常見的遞迴程序的範例。如果「!」代表階乘的符號，則

```

1    0! = 1
2    N! = N * (N - 1)!

```

2. 加總（sum）是計算一個 string 變數內數字和的範例。

```

1    例如：sum("1234") => 10

```

在這一個單元，我們將以階乘（factorial）與加總（Sum）兩個程式為範例，說明如何將一個遞迴程式逐步的改寫成迴圈程式。

內涵式遞迴

階乘函數可以用 Java 寫成如下的程式碼，我們將之命名為 `factorial`：

```

1    public class Recursive {
2        public static void main(String args[]) {
3            System.out.println(factorial(3));
4        }
5
6        public static int factorial(int n) {
7            if (n == 0)
8                return 1;
9            else
10               return n * factorial(n - 1);
11        }
12    }

```

Trace

```

1    factorial(3)
2 => return n * factorial(n - 1);    // n=3
3 => return 3 * factorial(2);
4 => return 3 * (n * factorial(n - 1));    // n=2
5 => return 3 * (2 * factorial(1));
6 => return 3 * (2 * (n * factorial(n - 1)));    // n=1
7 => return 3 * (2 * (1 * factorial(0)));
8 => return 3 * (2 * (1 * 1));    // n=0
9 => return 3 * (2 * (1 * 1));
10 => 6

```

觀察以上的執行狀況，呼叫 factorial 的遞迴呼叫是內涵 (embedded) 在乘法算式 ($n * \dots$) 之內。而程式的執行結果是在最後一個遞迴呼叫 factorial(0) 完成後才依序累乘 ($3 * (2 * (1 * 1))$) 而得的。這種類型的遞迴程序稱為內涵式遞迴 (embedded recursion)。

加總涵式 sum 的寫法如下：

```

1 public class Recursive {
2     public static void main(String args[]) {
3         System.out.println(sum("1234"));
4     }
5
6     public static int sum(String str) {
7         if (str.equals(""))
8             return 0;
9         else
10            return Integer.parseInt(str.substring(0, 1)) + sum(str.substring(1));
11    }
12 }

```

Trace

```

1    Sum("1234")
2 => return Integer.parseInt(str.substring(0, 1)) + sum(str.substring(1));    // str="1234"
3 => return 1 + sum("234");
4 => return 1 + (Integer.parseInt(str.substring(0, 1)) + sum(str.substring(1)));    // str="234"
5 => return 1 + (2 + sum("34"));
6 => return 1 + (2 + (Integer.parseInt(str.substring(0, 1)) + sum(str.substring(1))));    // str="34"
7 => return 1 + (2 + (3 + sum("4")));
8 => return 1 + (2 + (3 + (Integer.parseInt(str.substring(0, 1)) + sum(str.substring(1))));    // str="4"
9 => return 1 + (2 + (3 + (4 + sum(""))));
10 => return 1 + (2 + (3 + (4 + (Integer.parseInt(str.substring(0, 1)) + sum(str.substring(1))));    // str=""
11 => return 1 + (2 + (3 + (4 + 0)));
12 => 10

```

觀察以上的執行狀況，呼叫 sum 的遞迴呼叫也是內涵 (embedded) 在加總算式 ($\text{Integer.parseInt}(\text{str.substring}(0, 1)) + \dots$) 之內。而程式的執行結果是在最後一個遞迴呼叫

sum("")) 完成後才依序累加 (1 + (2 + (3 + (4 + 0)))) 而得的。這是內涵式遞迴的另一個例子。

尾端式遞迴

第二種形式的遞迴程序是尾端遞迴 (tail recursion)。尾端遞迴的特色是遞迴呼叫沒有內涵在任何一個尚未執行完成的式子內。以下面的例子而言，呼叫 tail-fac 的遞迴呼叫雖然是在 if 之內，但是該 if 的條件判斷式已經執行完畢，所以是尾端遞迴。

Trace

```

1   tail-fac(3, 1)
2 => tail-fac(2, 3)
3 => tail-fac(1, 6)
4 => tail-fac(0, 6)
5 => 6

```

以上程序的執行狀況是「平的」。尾端遞迴在執行的過程中不會像內涵式遞迴累積一層又一層如同樓梯般的式子，原因是前面遞迴呼叫時所產生的區域變數區內的變數值，並不需要被保留，因此下一次遞迴呼叫的區域變數區可以直接的覆蓋上一次遞迴呼叫時所使用的區域變數區。

將內涵式遞迴程序轉換成尾端遞迴程序的技巧是增加傳入的參數。以 fac2 來說我們增加了一個能夠儲存累乘值的參數 m。增加這個參數後便不需要以累積乘法算式的方式來計算階乘的值了。

而尾端遞迴的加總函式 sum，也可以經由增加一個參數及改變遞迴的回傳值得到：

```

1 public class Recursive {
2     public static void main(String args[]) {
3         System.out.println(tail-sum("1234", 0));
4     }
5
6     public static int tail-sum(String str, int n) {
7         if (str.equals(""))
8             return n;
9         else
10            return tail-sum(str.substring(1), (n + Integer.parseInt(str.substring(0, 1))));
11    }
12 }

```

Trace

```

1   tail-sum("1234", 0)
2 => tail-sum("234", 1)
3 => tail-sum("34", 3)
4 => tail-sum("4", 6)

```

```
5 => tail-sum("", 10)
6 => 10
```

迴圈

第三種計算階乘的方式是使用一般程式語言常常使用的「迴圈」。一個內涵式遞迴程序，在轉換成尾端式遞迴程序後，便可以改寫成迴圈程式了。

轉換的方式可以分成下面三個步驟：

1. 將尾端式遞迴的終止條件內的條件判斷拉到迴圈的條件判斷內，並將它轉換成相反（Not）的條件判斷（加上! 或!=）。
2. 將尾端式遞迴的參數拉到迴圈內並改寫成設值運算式，但是要注意的是設值運算式的前後次序。
3. 將尾端式遞迴的終止條件成立時的回傳值拉到迴圈的後面，讓程序結束後將結果值回傳回去。

使用 while-loop 將尾端式遞迴轉換成一般的程式語言常常使用的「迴圈」。

```
1 public class Recursive {
2     public static void main(String args[]){
3         System.out.println(fac3(3, 1));
4     }
5
6     public static int fac3(int n, int m) {
7         while (n != 0) {
8             m = m * n;
9             n = n - 1;
10            // 注意：上兩行次序不可顛倒。
11        }
12        return m;
13    }
14 }
```

Trace

```
1     fac3(3, 1)
2 => 6
```

```
1 public class Recursive {
2     public static void main(String args[]){
3         System.out.println(sum3("1234", 0));
4     }
5
6     public static int sum3(String str, int n) {
7         while (!str.equals("")) {
```

```

8         n = n + Integer.parseInt(str.substring(0, 1));
9         str = str.substring(1);
10        // 注意：上兩行次序不可顛倒
11    }
12    return n;
13 }
14 }

```

Trace

```

1    sum3("1234", 0)
2 => 10

```

同樣也可以使用 Java 語言的 for Loop 來將尾端式遞迴轉換成一般的程式語言常常使用的「迴圈」。

```

1 public class Recursive {
2     public static void main(String args[]){
3         System.out.println(fac3(3, 1));
4     }
5
6     public static int fac3(int n, int m) {
7         for (int i=m; i<=n; i++)
8             m = m * i;
9         return r;
10    }
11 }

```

Trace

```

1    fac(3, 1)
2 => 6

```

```

1 public class Recursive {
2     public static void main(String args[]){
3         System.out.println(sum3("1234", 0));
4     }
5
6     public static int sum3(String str, int n) {
7         for (; !str.equals(""); str = str.substring(1))
8             n = n + Integer.parseInt(str.substring(0, 1));
9         return n;
10    }
11 }

```

Trace

```

1    sum3("1234", 0)
2 => 10

```