

# **Fundamentals of Java Programming**

**Student Guide**

**SL-100 Rev A**

D67336TC10  
Edition 1.0  
July 2010  
D68068

**ORACLE®**

**Copyright © 2005, 2010, Oracle and/or its affiliates. All rights reserved.**

**Disclaimer**

This document contains proprietary information, is provided under a license agreement containing restrictions on use and disclosure, and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except as expressly permitted in your license agreement or allowed by law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

**Sun Microsystems, Inc. Disclaimer**

This training manual may include references to materials, offerings, or products that were previously offered by Sun Microsystems, Inc. Certain materials, offerings, services, or products may no longer be offered or provided. Oracle and its affiliates cannot be held responsible for any such references should they appear in the text provided.

**Restricted Rights Notice**

If this documentation is delivered to the U.S. Government or anyone using the documentation on behalf of the U.S. Government, the following notice is applicable:

**U.S. GOVERNMENT RIGHTS**

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

**Trademark Notice**

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd.



**SL100**  
**Java 入門基礎**



# 目錄

<b>第一單元 認識 Java 技術 .....</b>	1-1
<b>單元目標.....</b>	1-1
問題與討論.....	1-2
<b>相關資料.....</b>	1-3
<b>什麼是 Java .....</b>	1-4
<b>什麼是 Java SE 、Java ME 、Java EE .....</b>	1-9
<b>Java SE 架構.....</b>	1-12
<b>Java 語言的特性 .....</b>	1-16
<b>如何開發 Java .....</b>	1-19
<b>複習題.....</b>	1-22
<b>第二單元 準備 Java 開發環境 .....</b>	2-1
<b>單元目標.....</b>	2-1
問題與討論.....	2-2
<b>相關資料.....</b>	2-3
<b>下載、安裝 JDK .....</b>	2-4
<b>認識 JDK 、JRE 提供的資源.....</b>	2-11
<b>設定 Path 環境變數 .....</b>	2-16
<b>重新安裝 JDK 、JRE .....</b>	2-22
<b>複習題.....</b>	2-23
<b>第三單元 開發、執行 Java 程式 .....</b>	3-1
<b>單元目標.....</b>	3-1
問題與討論.....	3-2
<b>相關資料.....</b>	3-3
<b>撰寫 Java 程式 .....</b>	3-4
<b>編譯 Java 程式 .....</b>	3-12
<b>執行 Java 程式 .....</b>	3-16
<b>設定 Classpath 環境變數 .....</b>	3-19
<b>複習題.....</b>	3-22
<b>第四單元 註解、資料、變數 .....</b>	4-1
<b>單元目標.....</b>	4-1
問題與討論.....	4-2
<b>相關資料.....</b>	4-3

為程式加入註解.....	4-4
認識資料型態.....	4-9
宣告、使用變數.....	4-15
跳脫字元.....	4-22
複習題.....	4-24
<b>第五單元 運算子 .....</b>	<b>5-1</b>
<b>單元目標.....</b>	<b>5-1</b>
問題與討論.....	5-2
<b>相關資料.....</b>	<b>5-3</b>
<b>算術運算子.....</b>	<b>5-4</b>
<b>關係運算子、條件運算子.....</b>	<b>5-13</b>
<b>位元運算子、位移運算子.....</b>	<b>5-18</b>
<b>指定運算子.....</b>	<b>5-24</b>
<b>複習題.....</b>	<b>5-25</b>
<b>第六單元 流程控制 .....</b>	<b>6-1</b>
<b>單元目標.....</b>	<b>6-1</b>
問題與討論.....	6-2
<b>相關資料.....</b>	<b>6-3</b>
取得使用者輸入.....	6-4
建立 if 和 if/else 陳述.....	6-8
使用 switch 陳述.....	6-17
<b>複習題.....</b>	<b>6-22</b>
<b>第七單元 迴圈控制 .....</b>	<b>7-1</b>
<b>單元目標.....</b>	<b>7-1</b>
問題與討論.....	7-2
<b>相關資料.....</b>	<b>7-3</b>
使用 while 與 do while 迴圈.....	7-4
使用 for 迴圈.....	7-10
break 與 continue.....	7-14
<b>複習題.....</b>	<b>7-19</b>
<b>附錄 A Java 程式語言的關鍵字.....</b>	<b>A-1</b>
<b>附錄 B Java 程式語言命名慣例.....</b>	<b>B-1</b>

---

# 第一單元

---

## 認識 Java 技術

---

### 單元目標

單元目標



當完成本單元後，您將能學習到：

- 從生活中了解什麼是 Java
- 什麼是 Java SE、Java ME 與 Java EE
- Java SE 基本架構
- Java 的幾個重要特性
- 首次開發 Java 所必須的流程

這個單元提供 Java 技術概念、Java 三大應用版本，和開發 Java 程式的基礎流程概念。

## 問題與討論



討論一下列的問題能夠幫助您了解何謂 Java 技術：

- 請您最直接的疑問請問講師，什麼是 Java？
- Java 在開發程式所提供的三個應用版本為何？討論它們所應用的領域與常見的產品。
- Java SE 架構中包括四個部份，其擔任的角色各為何？



## 相關資料



相關資料一 下列參考資料能夠對本單元所討論的話題提供更詳細的說明。

- Java Technology: An Early History  
<http://java.sun.com/features/1999/05/birthday.html>  
描述 Java 技術開發並述說當時參與開發人員的故事。
- The Java Tutorial  
<http://java.sun.com/docs/books/tutorial/>  
- 針對程式設計者很實用的指引，有數百個完整、可運作的範例。



## 什麼是 Java

是的！現在的您迫切的想了解 H 諸是 Java，您已經在大多地方看到 Java 這個名詞，在「手機」、「網頁」，在書報中有一大堆的電腦工具書都是在介紹 Java，這一篇的目標在於解答您的疑問，到底 H 諸是 Java？

有幾個常見的問題需要先回答！

常見的  
問題



- 為何電腦跟手機都有 Java？
- Java 是 H 諸是？是電腦的設計程式還是 ...
- 請問 Java 程式是 H 諸是遊戲嗎？是手機必備功能嗎？
- 書報常常看到有一堆 Java 書，還是不清楚它的作用！
- Java 是 H 諸樣的語法呢？Applet 和 Script 的差別是 H 諸？
- 我不是專業人士，大專家的答案，我可能看不懂！

### 為何電腦跟手機都有 Java？

電腦'、手  
機中的  
Java



如果您的電腦或是手機可以支援 Java 技術，並且精陞一點的說，有一個執行環境可以執行 Java 所撰寫的程式，那麼您就可以在電腦或手機上運行 Java 應用程式，包括遊戲、動畫。

例如在「網頁、瀏覽器」，如果您的瀏覽器支援 Java 技術，您就可以看到 Java Applet 所表現的動畫效果。

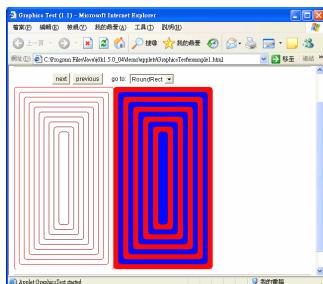
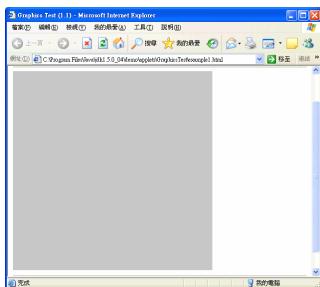


圖 1-1 Java Applet 表現的繪圖效果

**什麼是 Java**

如果您開啟了某個使用了 Java Applet 技術的網頁，卻只看到圖 1-2 中灰白的畫面，那樣您的電腦中並沒有安裝支援 Java 技術的執行環境，通俗的說法是「您沒有安裝 Java」。



**圖 1-2 沒有安裝 Java 執行環境，無法看到 Applet 效果**

同樣的，如果手機中有支援 Java 技術的執行環境，那樣您就可以下載為手機設計的 Java 應用程式、遊戲，然後在您的手機中執行這些程式。

## Java 是什麼東西？是電腦的設計程式還是...

是程式語言嗎？



是的！Java 最先開始的時候是一門程式語言！您也許聽過 C 語言、Visual Basic 語言或其它的語言，Java 在程式語言這方面的定義與它們是類似的，目的是讓程式設計人員用這種語言來設計程式。

**Java 程式語言 (Java programming language)** 起源於 1991 年，最先被稱為 “Oak”，是當時 Green team 軟體開發團隊中用來撰寫 Star 7 手持設備上應用程式的- 個程式語言。



**圖 1-3 Star 7 手持設備**



**重點提示**一當時專案的主持人 James Gosling 稱這新語言為“Oak”的原因，是因為看到窗外面的橡樹，但後來發現 Oak 這個名稱已經被註冊使用了，J 程師們邊喝咖啡邊討論新的名稱，才靈機一動的將之改名為“Java”。

不過，Star 7 這項產品後來並沒有成功，Green team 與他們的專案面臨停止的命運，所幸的是當時全球資訊網（World Wide Web, WWW）正在興起，Green team 仿照了當時（1993 年）的 Mosaic 瀏覽器，開發了一個以 Java 技術為基礎，支援 Java Applet 的瀏覽器 WebRunner，後來改名為 HotJava。



**註**—關於 Green team 與他們的專案，可以看看這邊的簡介：  
<http://java.sun.com/people/jag/green/index.html>

由於全球資訊網（WWW）開始被廣泛使用，並且 Green team 認為 Oak 語言有助於開發網路多媒體元件，以提升網頁的品質。這種稱為 **Applets** 的小型應用程式就成為 Oak 語言最初的目的，因此促使網路上的程式設計師採用，而到最後演變成為我們所熟悉的 Java。



**註**—相關 Java 程式語言的歷史可參考：  
<http://java.sun.com/features/1998/05/birthday.html>

## 請問 Java 程式是什麼？遊戲嗎？是手機必備功能嗎？

遊戲？  
手機必備功  
能？



Java 應用程式是使用 Java 語言所撰寫出來的程式，這個程式必須在安裝有 Java 執行環境的設備中才可以執行，當然了，Java 可以撰寫遊戲，而這是支援 Java 技術的手機的一大賣點，所以如果某品牌的手機有支援 Java 技術，它們一定會在產品廣告中特別標註，為它們的產品增加賣相。

手機中的 Java 程式讓大家對 Java 已熟能詳，當然！並不是所有的手機都支援 Java，而 Java 也不只是為了撰寫遊戲，有些手機中也肖像是行動電話、股票行情分析、電子郵件等軟體等，也都可以用 Java 來撰寫執行。

## 書局常看到賣一堆 Java 書，還是不清楚它的作用！

對 Java 的使用者來說，Java 是一個很簡單的名詞，但對於程式設計人員來說，要使用 Java 來開發程式就需要一點學習的時間了。

Java 無所



Java 隨著應用領域的逐漸增加，在功能與架構上越來越複雜，除了您看到的到手機、電腦上的 Java 程式之外，其實有很多您沒有注意到的地方也使用了 Java 的技術，例如應用領域的歡迎，針對某個領域探討如何使用 Java 來撰寫程式，滿足使用者的需求的介紹書籍就越來越多。

您看到書店賣 Java 的書越多，表示生活中越來越多東西使用 Java 技術，生活中無處不是 Java，這是真的！



圖 1-4 即使您沒察覺，這個網站用了 Java 技術撰寫與運行

## Java 是什麼樣的語法呢？Applet 和 Script 的差別

### 是什麼？

Java 是一個支援物件導向程式設計 (Object-oriented Programming) 的程式語言，語法與關鍵字上類似 C/C++，別急著問物件導向是什麼，要了解它並不容易，之後的課程會慢慢引導您了解什麼是物件導向。

您常常聽到 Java Applet 與 JavaScript 這兩個名詞，甚至您

亂說 Java 技術

1-7

Copyright 2005 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services, Revision A

什麼是 Java

可能還聽過 Java Servlet，簡單的說，Java Applet 與 Java Servlet 是 Java 技術下的兩種實作規範，Java Applet 會被下載到您的電腦中執行，而 Java Servlet 會直接在網站上執行，再將結果傳給您的電腦。

Java 跟  
JavaScript  
沒有關係



對於 JavaScript！請一定要記得，除了名字上有“Java”之外，它跟 Java 技術一點關係也沒有，這純粹是當初 LiveScript (JavaScript 的前名) 重新命名時開的玩笑，JavaScript 與 Java 是完全不同的技術，運行原理也完全不同。

## 我不是專業人士，太專業的答案，我可能看不懂！

沒關係！慢慢的從課程中，您會了解更多 Java 的細節，您可以先看看以下幾個答案，了解 Java 是什麼：：

- 是一種技術（這東西叫 Java）
- 是一種程式語言（您可以用 Java 寫個程式）
- 可撰寫應用程式（這程式是由 Java 開發的）
- 是手機上的一个賣點（這款手機有支援 Java 呢）
- 是一種誤導工具（我會 Java，我想找工作）
- 可能是程式設計人員滿懷的對象（Shit...我再也不碰 Java 了）



### ◎ 球測驗—配對以下的名詞與定義

#### 定義

- Java 可以寫遊戲  
這手機有 Java  
書房中有很多介紹 Java 的書  
JavaScript 與 Java

#### 名詞

- Java 的應用領域廣泛  
Java 是一個程式語言  
具支援 Java 的執行環境  
不同的技術

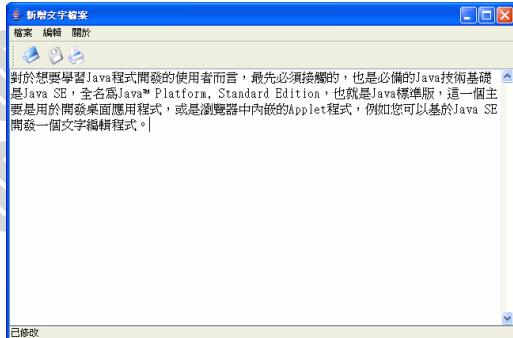
## 什麼是 Java SE、Java ME、Java EE

Sun Microsystems 提供了完整的 Java 技術產品，依市場需求主要區分為三個不同的種類，包括了 Java SE、Java ME 與 Java EE。

### Java SE

對於想要學習 Java 程式開發的使用者而言，最先必須接觸的，也是必備的 Java 技術基礎是 Java SE，全名為 Java™ Standard Edition，也就是 Java 標準版，這一個主要是用於開發桌面應用程式，或是瀏覽器中內嵌的 Applet 程式，例如您可以基於 Java SE 開發一個文字編輯程式。

Java  
標準版



電 1-5 什麼 Java SE 開發的文字編輯程式



註—Applet 和應用程式有些地方不同。主要的分別是 Applet 運行於 Web 瀏覽器上，而應用程式直接執行在作業系統上。這個課程暫時只教應用程式的開發，但在此課程中大多數的單元目標都可應用在 Applet 的開發上。

H 號是 Java SE、Java ME、Java EE

## Java ME



對於資源受限的消費性電子產品，例如手機或是 PDA，像這類的設備本身擁有的資源（像強大的 CPU 與充足的記憶體），若要開發這類設備上的應用程式，可以使用 Java ME 這個版本，全名為 **Java™ Micro Edition**，也就是 **Java 微型版**，現在手機上若聲稱有支援 Java，多半是由這個版本所開發出來的程式。

## Java EE



對於大型企業級網站而開發的應用程式，Java 提供了 Java EE 這個版本讓設計人員來進行開發，全名為 **Java™ Enterprise Edition**，也就是 **Java 企業版**，Java EE 以 Java SE 為基礎，在架構上與開發的規模上都比 Java SE 龐大許多，在所應用的技術上，比較為人所耳熟能詳的像是 JSP、Servlet、EJB 等。

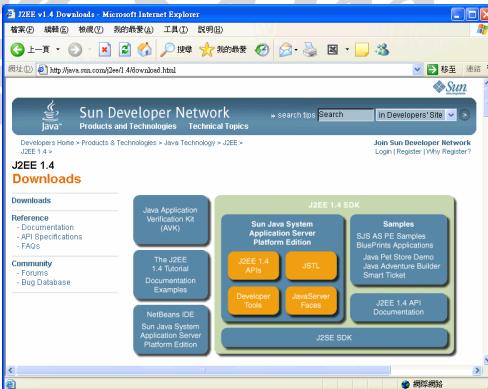


圖 1-6 開發企業級網站應用程式時，可使用 Java EE 技術



記—您應該也聽過 J2SE (Java™ 2 Platform, Standard Edition)、J2ME (Java™ 2 Platform, Micro Edition) 與 J2EE (Java™ 2 Platform, Enterprise Edition)，這些名詞將並非過式，在接下來的版本中，會分別命名為 Java SE、Java ME 與 Java EE，也就是不再有 Java 2 這個名稱了。

上標是 Java SE、Java ME、Java EE

在不同的 Java 技術中，每一 個版本都包含了 [軟體開發工具箱 \(Software Developer's Kits, SDK\)](#)，使程式設計者能夠在特定的平台製作、編譯和執行 Java 程式。



• [我建議](#)一隻月正確的版本 (Java SE、Java EE、Java ME)  
開發以下的應用程式

應用程式  
Java Applet  
手機遊戲  
網站應用程式  
小畫家

開發版本



## Java SE 架構

儘管許多 Java 程式設計者最後會專門研究特定目標產品的開發應用程式，但所有的程式設計者通常是以桌上型電腦製作應用程式或 Applet 做為開端。這也就是說，當學習 Java 程式語言時，Java 標準版本 (Java SE) 是最多程式設計者所選用的產品種類。

來看一々 Java SE 的基本架構：



圖 1-7 Java SE 基本架構

註一：圖 1-7 是個簡化過的 Java SE 架構圖，如果想要看看完全完整的 Java SE 架構圖，可以參引：  
<http://java.sun.com/j2se/overview.html>



即使是 Java SE，當中所包括的架構也是相當龐大的，對於初學者來說，只要先了解到 Java SE 可以分作幾個主要的部份：Java 語言、JDK、JRE 與 JVM。

## Java 語言

要使用 Java 技術來開發應用程式，首先就要學習如何使用 Java 程式語言，然而其被叫作 程式語言（Programming language）？



舉個例子來說，您要與懂得中文的外國人溝通，要懂得中文與懂得英文的外國人溝通，如果您要與電腦溝通，就要用電腦可以了解的語言與電腦溝通。

Java 程式語言就是用來與電腦溝通的語言，您使用這個語言命令電腦來為您作所指定的動作，程式語言的目的就是在此。

```
// 這個程式語言目的在告訴電腦
// 在螢幕上顯示"哈囉！Java！"

public class Main {
    public static void main(String[] args) {
        System.out.println("哈囉！Java！");
    }
}
```

圖 1-8 程式語言的目的在於與電腦溝通

## JDK

Java 程式語言是高階的程式語言，所謂高階的程式語言，指的是其語法與語義是人類比較容易的理解方式，然而電腦只懂得低階的機器語言，也就是 0 與 1 這樣的語言，Java 在 JDK 中提供有工具程式，幫您將高階語言轉換為低階語言。

### Java 開發工具箱



JDK 的全名是 Java SE Development Kits，中文名為 Java 標準版開發工具箱，除了提供編譯器（Compiler）轉換高階語言為較低階的語言之外，還提供有其它相關的開發、執行、測試等工具。

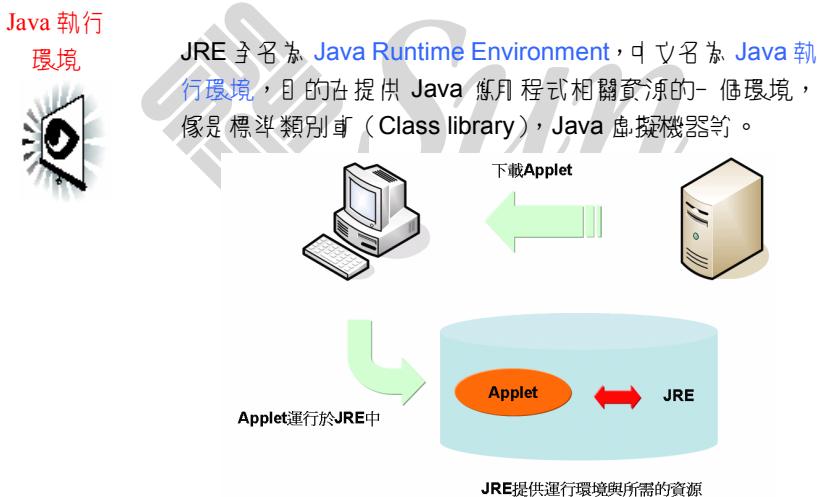
簡單的說，作為一名 Java 開發人員，在撰寫好 Java 程式的原始碼（Source code）之後，還必須安裝 JDK，才可以進行接下來的編譯、測試等工作。

## JRE

如果您在網路上遇到一個含有 Java Applet 的網頁，卻只看到以前圖 1-2 所示灰白的畫面，或者是您無法執行一個 Java 所撰寫的應用程式，您的電腦中可能沒有安裝 JRE。

在您實際開發 Java 程式時，您會發現所編譯出來的檔案其實容量並不大，像 Applet 這樣的應用，Java 程式是從網路上下載的，所以編譯出來的檔案可能很大，以節省程式下載時間，為了縮小檔案的體積，有一些 Java 標準的元件，必須要能在運行程式時直接從 JRE 中取得。

簡單的說，如果一個 Applet 程式需要使用 100 個資源，而其中 80 個資源是標準常見的元件，若這些元件直接安裝在您的電腦上，則下載 Applet 程式時只需下載 20 個資源，以節省下載的時間，剩下的 80 個標準元件可從執行環境中取得。



■ 1-9 要執行 Java 程式，必須有 Java 執行環境

簡單的說，您的電腦如果要運行 Java 程式，JRE 是必須安裝的基本要求。

## JVM

使用 Java 寫寫程式的最大好處是，您可以讓 Java 程式在安裝有 Java 執行環境的設備上直接執行，而不必理會該設備

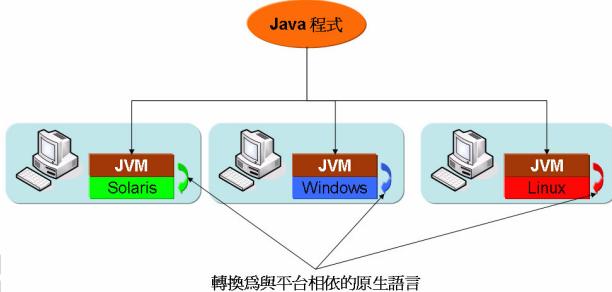
## Java SE 架構

上運行的是什麼樣的平臺，像是 Linux、Macintosh、Windows、Solaris 等，其背後的功臣就是 JVM。

### Java 虛擬 機器



JVM 全名為 **Java Virtual Machine**，包括在 JRE 之中，您所撰寫並編譯好的 Java 程式會將 JVM 當作一台真實的機器，而它會知道它是運行於哪一個作業系統之上，JVM 會將編譯過的 Java 程式轉換為與平臺相依的 **原生碼 (Native code)**，讓 Java 程式可以運行於各個平臺之上。



### 1-10 JVM 實現 Java 程式跨平臺的可能性



**重點提示**：一虛擬機器這個名字的由來是因為它是一台軟體，可以執行：通常由 CPU 或是“硬體機器”所完成的工作。



**知識測驗** 填寫以下所需的名詞 (JDK、JRE、JVM、Java 語言)

#### 定義

- 執行 Java 程式
- 提供標準類別庫
- 撰寫 Java 程式
- 讓跨平臺成為可能
- 提供 Java 編譯器

#### 名詞

## Java 語言的特性

作為一個程式語言來說，Java 有許多重複的特性：

- 簡單的 (Simple)
- 物件導向的 (Object-oriented)
- 安全的 (Secure)
- 多執行緒的 (Multi-thread)
- 跨平台的 (Platform-independent)

### 簡單的 (Simple)

**簡單的**



Java 語言在設計的時候，參考了一些程式語言（例如 C/C++），移除了某些程式語言中較為複雜或不容易掌控的部份，例如 Java 中沒有指標（Pointer），不允許您直接對記憶體位址進行操控，因為指標的使用相當複雜，程式設計人員必須小心的使用指標。

Java 程式語言僅允許程式設計師利用 **物件參考** (Object reference) 來操控物件，和使用 **垃圾收集** (Garbage collector) 來監視和移除已經不再被參考的物件，另一方面，Java 中對字串的處理更為簡單，而一些語法與 C/C++類似，這使得在學習 Java 語言時更易於上手。

### 物件導向的 (Object-oriented)

**物件導向**



物件導向是一種思考如何解決問題的方式，Java 的語言模型可以支持使用物件導向的思考方式來撰寫程式，例如您可以使用 Java 語言定義一個類別，使用 Java 語言來實現繼承的機制等。

了解物件導向的精神並以物件導向的方式來設計、撰寫程式並不容易，而且需要經驗的累積，在學習 Java 語言的時候，必須同時學習物件導向的思考方式，如此學來才可以事半功倍。初學者可藉由練習教材中的程式，並從有經驗的開發人員所撰寫的程式中觀察學習，慢慢了解物件導向的精髓。

### 安全的 (Secure)

## Java 語言 的特性

安全的



Java 所撰寫的程式必須運行於 Java 執行環境中，Java 執行環境採取了一些安全措施來保護程式不受攻擊，例如：

- 禁止利用指標操控記憶體。
- 禁止 Applet 這類的程式對電腦的硬碟直接進行讀取或寫入的動作，才不會下載 Applet 時，就不知不覺的被寫入某個資料或被竊取資料。
- 檢查所有 Java 程式是否擁有合法的程式碼，即使 Java 程式被使用其它程式惡意修改，Java 執行程式也會檢查出來。
- 類別載入器 (Class loader) 的階層架構設計，避免了載入惡意 (Malicious) 來源的類別。

## 多執行緒的 (Multi-thread)

要使用 Java 撰寫具多執行緒功能的程式是簡單的，也就是讓您的程式可以同時處理多件以上的異常，例如您可以一邊排列印文件，一邊從資料庫中讀出資料，另一方面繼續從網路上传送檔案。

## 跨平台 (Platform-independent)

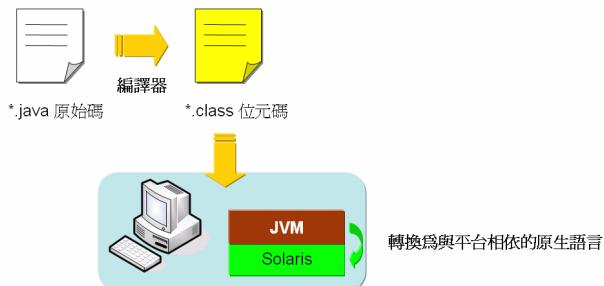
大多數的語言在撰寫程式時，假如要在不同類型的電腦平臺 (Computing platform) (像是不同的 CPU 或不同的作業系統) 上執行時，就必須要在程式中作大幅的修正。平台相依 (Platform-dependence) 是因為大多數的語言要求你必須在不同平臺上寫適合的程式碼。

跨平臺



Java 程式在撰寫時要編寫一個副檔名為 \*.java 的純文字檔案，之後透過編譯器 (Compiler) 將之編譯為 \*.class 的位元碼 (byte-code) 檔案，位元碼檔案與平臺沒有相依性，為了執行並執行這個位元碼檔案，您的電腦 (或設備) 上必須有 Java 虛擬機器 (JVM)，JVM 負責載入相關的 Java 類別、直譯 Java 位元碼為平臺相依的原生碼 (Native code)、及執行 Java 的程式。

## Java 語言 的特性



1-11 Java 跨平台原則



**註**—Java 程式在跨平臺時是需要稍微的修改。舉個例子：目錄名稱也許需要修改，例如：根據這的作業系統，而使用適當的分隔符號（斜線或是 \ 斜線）。



**重點提示**—Java 技術通常會被當作是一個平臺，因為它具有完備的作業系統和 CPU 所需工作的能力。這就是為何有些人提議電腦的 CPU 只需要了解位元碼。Sun Microsystems 曾經開發過這種電腦的原型，稱為 JavaStation™。

microsystems



**◎ 我測驗**—使用您自己的方式，簡單的描述一下 Java 的這幾個重要特性：

- 簡單的 (Simple)
- 物件導向的 (Object-oriented)
- 安全的 (Secure)
- 多執行緒的 (Multi-thread)
- 跨平臺 (Platform-independent)

## 如何開發 Java

是的！既然您已經打算開始學習 Java，我想您最關心的，就是如何起步開發 Java，以下 是首次開發 Java 程式所必須的幾個流程：

- 下載、安裝 JDK
- 設定 Path 與 Classpath 環境變數
- 使用純文字檔案撰寫副檔名 \*.java 的程式
- 使用 javac 工具程式編譯原始碼為 \*.class 檔案
- 使用 java 工具程式運行、測試 Java 程式是否符合功能
- 從 API Specification 中查詢所需的 API 之功能

詳細的設定與操作步驟，將會在下一個單元中介紹，在這邊介紹的目的在於給您一個整體概念，了解如何開發出第一個 Java 程式。

### 下載、安裝 JDK

#### 安裝 JDK



想使用 Java 技術來開發程式的第 - 步，就是從 Sun Microsystems 的 Java 官方網站 (<http://java.sun.com>) 下載 JDK，如前所介紹的，JDK 包括了所需的工具程式與 Java 執行環境，您必須在您的電腦上安裝 JDK 才能進行 Java 程式開發。

### 設定 Path 與 Classpath 環境變數

#### 設定 環境



很不幸的，並不是安裝好 JDK 就算 OK！您還要提供一些資訊給您的作業系統，Path 環境變數的設定在於告知作業系統 JDK 工具程式的位置，這樣作業系統才能找到您所想要執行的工具程式。

另一方面，您的 Java 程式可能放置在任意的位置，所以您要告訴 JVM，當您指定某個 Java 程式時，該到哪個位置去找出所指定的程式，Classpath 環境變數的設定目的就在這邊。

## 如何開發 Java

Java 初學者很常卡在 Path 與 Classpath 設定上，在下一個單元中將會介紹如何設定這兩個環境變數。

### 使用純文字檔案撰寫副檔名\*.java 的程式

撰寫 Java 程式最基本的，就是使用純文字檔案，在 Windows 上的話只要使用「記事本」就可以編寫，但是必須將副檔名改成 \*.java 。

### 使用 javac 工具程式編譯原始碼為\*.class 檔案

javac 工具  
程式

在您使用「整合開發環境」( Integration Development Environment)的情況下，您必須從「文字模式」下執行 javac 工具程式，例如在 Windows 中的「命令提示字元」。

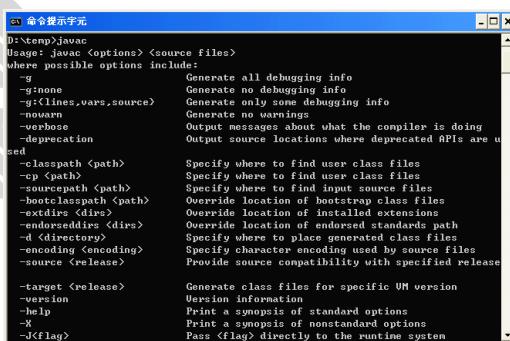


圖 1-12 在 Windows 的命令提示字元下執行工具程式

javac 工具程式負責將\*.java 編譯為\*.class 檔案，它也會幫您做一些基本的語法與錯誤檢查，即所謂的「編譯時期檢查」，了解如何使用 javac 工具程式，並看懂編譯器所提供的相關錯誤訊息是學習 Java 所必備的基本功夫。

### 使用 java 工具程式運行 Java 程式

java  
工具程式

在您使用「整合開發環境」的情況下，您必須從「文字模式」下執行 java 工具程式，java 工具程式會檢查 Java 程式的合法性（例如確定程式沒有被惡意修改）、確定相關類別位置並載入、執行 Java 程式。

## 從 API Specification 中查詢所需的 API 之功能

使用 Java 開發的最大好處之一，就是 Java 本身提供了數量龐大且功能豐富的標準類別庫（Class library），除了幾個常見的類別之外，您不必花費時間去記憶這些 API（Application programming interface）的功能與細節，而只要在使用到相關功能時查詢 API Specification。



**註 - 應用程式介面（Application programming interface, API）及類別庫的使用是可以替換的。進一步來說，API 可以使用在類別庫的程式碼內。**

就 J2SE 5.0 的 API Specification 而言，您可以線上直接查詢相關的類別功能與使用方式，網址是：  
<http://java.sun.com/j2se/1.5.0/docs/api/>



Java 2 Platform Standard Edition 5.0 API Specification	
<a href="#">java.awt</a>	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet content.
<a href="#">java.awt.event</a>	Contains all of the classes for creating user interfaces and for painting graphics and images.
<a href="#">java.awt.color</a>	Provides classes for color spaces.
<a href="#">java.awt.datatransfer</a>	Provides interfaces and classes for transferring data between and within applications.
<a href="#">java.awt.dnd</a>	Drag and drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
<a href="#">java.awt.event</a>	Provides interfaces and classes for dealing with different types of events.

■ 1-13 J2SE 5.0 API Specification



**註 - 在 Java 標準版 5.0 仍沿用 J2SE 的名稱，所以線上描述仍稱為 J2SE 5.0。**



◎ **步驟三** 嘗試先從 Java 官方網站 (<http://java.sun.com>) 中找出 JDK 下載網站並下載 JDK 安裝程式，API Specification 也有提供下載版本，請試著找出並下載吧。

## 複習題

- 、 Java 有什麼特點可以跨平臺？
- 二 、 Java SE、Java ME 與 Java EE 的應用範圍各為何？
- 三 、 簡略描述JDK、JRE 與 JVM 的功能。
- 四 、 在進入下一個單元之前，對於 Java 這個名詞，您是否有大致的瞭解，您還有哪些疑問嗎？請寫下來！並試著與其他同學互相討論。



---

## 第二單元

---

### 準備 Java 開發環境

---

#### 單元目標

單元目標



當完成本單元後，您將能學習到：

- 從 Java 官方網站下載 JDK 並進行安裝
- 了解 JDK 中安裝了哪些相關工具與程式
- 設定 Path 環境變數

這個單元以 Step by step 的方式告訴您如何下載、安裝 JDK，了解如何設定 Path 環境變數及其目的。





討論—下列的問題能夠幫助您了解 JDK :

- 為何要安裝 JDK 才能開發程式？它提供了什麼工具？
- JDK 為何要附帶自己專用的 JRE？
- 設定 Path 環境變數的目的為何？



## 相關資料



**相關資料**－下列參考資料能夠對本單元所討論的話題提供更詳細的說明。

- Installation Notes

<http://java.sun.com/j2se/1.5.0/install.html>

Solaris、Microsoft Windows、Linux 的 JDK 安裝需知



下載、安裝 JDK

---

## 下載、安裝 JDK

要編譯、執行、測試 Java 應用程式，您需要一些工具程式與環境資源來協助您，所以您必須安裝 JDK，在這個小節，將一步一步的操作示範來告訴您如何完成以下的動作：

- 下載 JDK
- 安裝 JDK

### 下載 JDK

要下載 JDK，首先打開瀏覽器連接至 Java 官方網站，網址是 <http://java.sun.com>，在這邊假設您是使用 Microsoft Internet Explorer 瀏覽器。

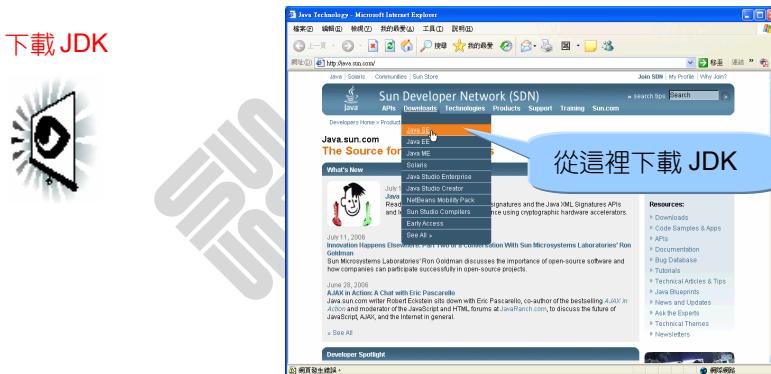


圖 2-1 Java 官方網站

在 Java 官方網站的右邊有 [Popular Downloads](#) 鏈結，這些是最常被下載的 Java 相關產品，直接按滑鼠上的左鍵選擇 [J2SE 5.0](#)（撰寫本單元時的 Java SE 版本為 5.0），就可以進入 Java SE 相關產品的下載網頁。

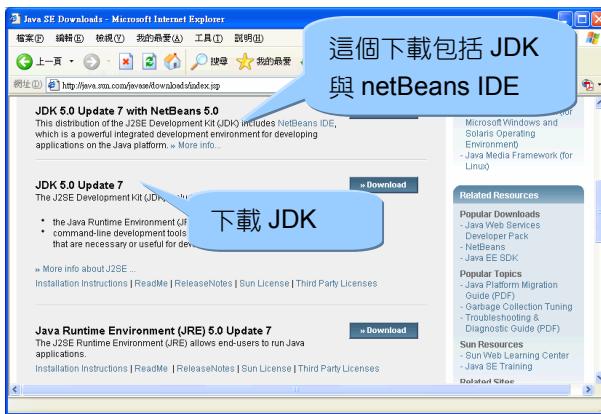


圖 2-2 Java SE 相關產品下載

在 Java SE 相關產品下載頁面中，您可以發現幾個可供下載的檔案，以下說明三個您要注意的下載檔案：

### JDK 5.0 with NetBeans 5.0



這個下載檔案包括了 JDK 及 NetBeans IDE 的安裝檔案，JDK 提供的是開發 Java 程式時所必須的基本工具，而 NetBeans 是開發 Java 應用程式時的**整合開發環境**（*Integration Development Environment, IDE*），整合開發環境提供開發人員更多的輔助功能，像是語言關鍵字提示、程式碼管理、除錯功能等，NetBeans IDE 是眾多 Java 程式開發人員所愛用的 IDE 之一。

Java 初學者通常建議先不使用 IDE，而建議先熟悉 JDK 相關工具程式的操作，以求對 Java 的運行機制有個充分的了解，之後再學習 IDE 工具的使用，所以這邊先不下載 NetBeans+JDK 的版本。

### JDK 5.0



這個下載檔案包括了 JDK 以及公用 JRE（Public JRE），JRE 也就是 Java 執行環境，是運行 Java 程式所必要的，通常建議初學者下載這個版本的檔案並進行安裝。

## 下載、安裝 JDK



註－在稍後會介紹到，JDK 本身也附帶了運行相關工具程式時所必須的 JRE，這個 JRE 是 JDK 自己擁有的，功能上與公用 JRE 有些許的不同。

## JRE



## JRE

如果只是要運行開發好的 Java 程式，則電腦上只需安裝 Java 執行環境，例如您若希望朋友們可以使用您所開發的 Java 程式，則您可以幫他們安裝這個版本的下載檔案。



註－在圖 2-2 中您可以看到還有 Update 7 的字樣，這表示 Java SE 版本發表後第七個修正版本，每隔一段時間，Java 都會推出修正版本，以修正一些程式中的錯誤或是安全性問題。

對於初學 Java 程式設計的人，建議下載不包括 IDE 的 JDK 安裝檔案，所以在圖 2-2 中，請按下滑鼠左鍵選擇 [JDK 5.0 Update 7 的 Download](#)，準備開始下載 JDK 安裝檔案。

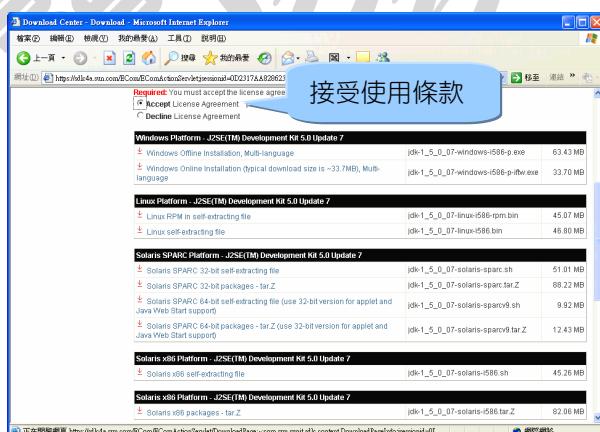


圖 2-3 各種平台的 JDK 安裝檔案

由於這邊的 JDK 安裝檔案是 Sun Microsystems 公司的產品，所以您必須同意使用條款才可以下載安裝檔案，使用滑鼠左鍵選擇 [Accept](#) 即同意使用條款，接著就可以開始進行檔案下載。

## 下載、安裝 JDK

您可以在下載頁面中發現，JDK 提供了各種作業平台的安裝檔案，包括了 Windows、Linux、Solaris 等平台，以提供不同平台的使用者開發程式的需求，您也可以發現到，每個平台都提供了 Offline Installation 與 Online Installation 的下載檔案。

Offline 或  
Online



Offline Installation 的下載檔案包括了 JDK 安裝過程中所必要的資源，所以下載這個檔案之後，無須連接到網路也可以進行 JDK 的安裝；Online Installation 則會依您所選擇的安裝元件，從網站上下載必要的安裝檔案，安裝過程中必須連接網路。

對於初學者，建議完整安裝所有的 JDK 資源，所以這邊建議下載 Offline Installation 這個版本的檔案，在這裏假設初學者所使用的是 Windows 作業系統，所以請在圖 2-3 中顯示的網頁中，使用滑鼠左鍵選擇 Windows Offline Installation, Multi-language，開始進行檔案下載，作業系統會請您確認這是您要求下載的檔案。

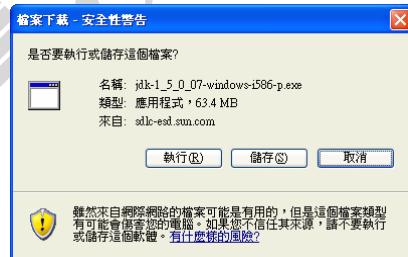


圖 2-4 作業系統確定檔案下載

按下「儲存」鍵之後，請選擇您的檔案儲存位置，這邊假設您的檔案儲存至 C:\Downloads，下載的檔案為 jdk-1\_5\_0\_07-windows-i586-p.exe。



**註**－這邊用來示範的作業系統是 Windows XP Service Pack 2，與您所使用的作業系統版本可能不同，所以示範畫面與您實際操作時的畫面會略有不同，但基本的操作流程是類似的。

下載、安裝 JDK

## 安裝 JDK

假設您已經按照之前的介紹下載了 JDK 的 Windows Offline Installation, Multi-language 安裝檔案，請開啟您的「Windows 檔案總管」瀏覽 C:\Downloads，在之前下載的 JDK 安裝檔案 jdk-1\_5\_0\_07-windows-i586-p.exe 上使用滑鼠左鍵連按兩下開始進行 JDK 的安裝。



圖 2-5 作業系統確定檔案下載

安裝的第一步是接受使用條款，選擇「I accept the terms in the license agreement」後，按下「Next」進入安裝的下一步驟。

確認 JDK  
安裝位置

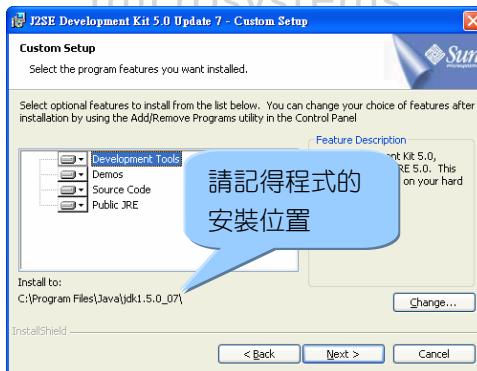


圖 2-6 選擇安裝元件並確認安裝位置

在這個步驟中，必須選擇您所想要的安裝元件與程式的安裝位置，請記下 JDK 的安裝位置，在圖 2-6 中是安裝至 C:\Program Files\Java\jdk1.5.0\_07\，這個資訊很重要，稍後設定環境變數時必須使用到這項資訊。

在可選擇的安裝元件中，**Development Tools** 安裝的是 **JDK** 的工具程式，像是編譯器、Applet 檢視器等；**Demos** 會安裝一些 Java 範例程式，可以作為開發程式時的參考對象；**Source Code** 則安裝 Java SE 標準類別庫的原始程式碼，Java 開發人員有時必須了解標準類別庫中某個類別程式是如何撰寫的，所以建議安裝這個項目；**Public JRE** 則是執行 Java 程式所必要的執行環境，建議一併安裝。

在確認安裝元件與 JDK 安裝路徑之後，按下圖 2-6 中的「**Next**」按鈕就會開始安裝 JDK，安裝完成後，會進行 Public JRE 的安裝。

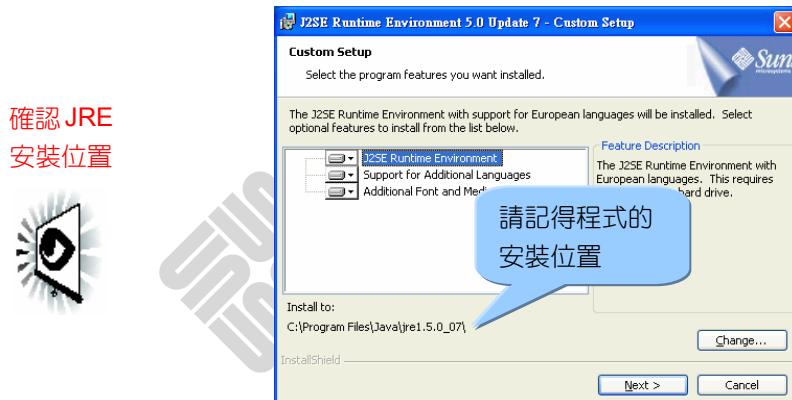


圖 2-7 選擇安裝元件並確認安裝位置

同樣的也請您記下 JRE 的安裝位置，稍後討論環境變數時，會需要使用到這項資訊，這邊假設 JRE 的安裝位置為 **C:\Program Files\Java\jre1.5.0\_07\**，按下「**Next**」按鈕之後，安裝程式會請您確認是否讓瀏覽器支援 Java 技術，像是在瀏覽器中執行 Java Applet。

## 下載、安裝 JDK

讓瀏覽器  
支援 Java

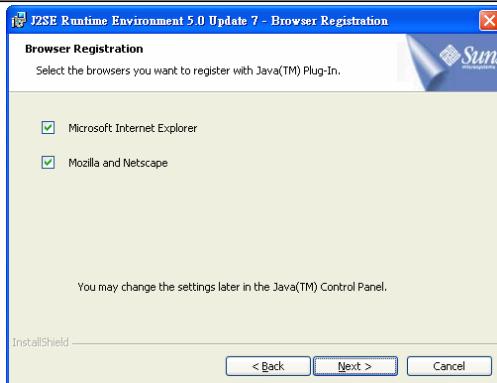


圖 2-8 讓瀏覽器支援 Java 技術

選擇要支援 Java 技術的瀏覽器之後，按下「[Next](#)」按鈕就會開始安裝 JRE，接下來就只要等待安裝完成即可。



**自我測驗**—嘗試不看教材，獨力完成整個 JDK 的安裝過程，詢問自己是否了解每一個安裝步驟的目的。



## 認識 JDK、JRE 提供的資源

假設您的 JDK、JRE 已經安裝完成，接下來的第一個疑問就是如何使用它們所提供的資源？要了解如何使用 JDK、JRE，首先必須先知道 JDK 提供了哪些資源。

### JDK 提供的資源

請開啟您的「檔案總管」來瀏覽安裝 JDK 的資料夾，假設是 [C:\Program Files\Java\jdk1.5.0\\_07](C:\Program Files\Java\jdk1.5.0_07)，看看裏頭安裝了哪些檔案？

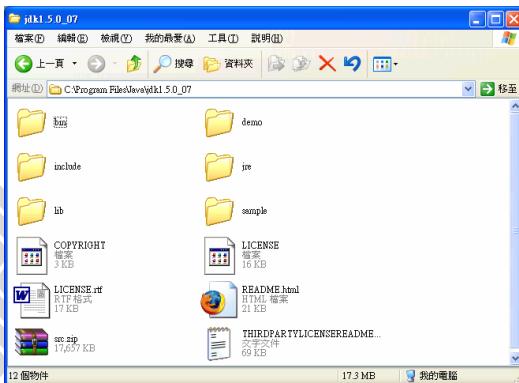


圖 2-9 JDK 提供的檔案

除了版權聲明、授權條款及 Readme 等檔案之外，有幾個重要的資料夾與檔案必須了解：

#### bin 資料夾



在這個資料夾下提供了開發 Java 程式時所必需的一些工具程式，像是 [javac](#) 編譯器、[java](#) 執行程式、測試 Applet 的 [appletviewer](#) 程式、製作說明文件 [javadoc](#) 程式、製作 JAR (Java Archive File) 檔的 [jar](#) 程式等。

## 認識 JDK、JRE 提供的資源

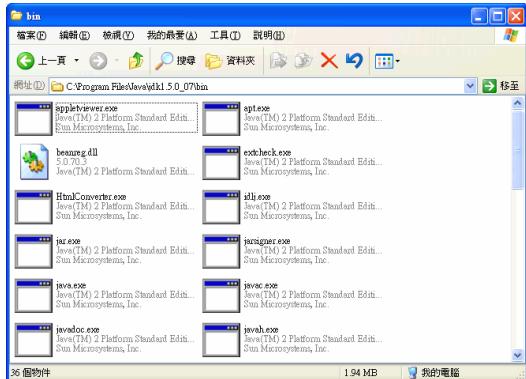


圖 2-10 JDK 提供的工具程式



**重點提示**－事實上，在 bin 資料夾中看到的程式只不過是個包裝器（Wrapper），JDK 所提供的工具程式大多是由 Java 所撰寫而成，它們被放在 lib 目錄的 tools.jar 中，bin 資料夾中的這些可執行程式，其目的是呼叫 tools.jar 中的 Java 程式，讓開發人員可以少輸入一些指令。

demo 資料夾



在 demo 資料夾中，JDK 提供了一些由 Java 撰寫而成的應用程式範例，有興趣的話，您可以執行一下這些程式範例，看看 Java 可以展現什麼樣的效果，這些範例都附有原始程式，您可以看看這些範例是如何撰寫的。

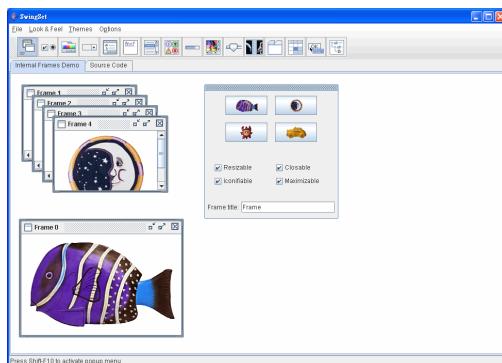


圖 2-11 demo\jfc\SwingSet2 範例程式

## jre 資料夾

jre

資料夾



在 JDK 安裝資料夾下的 jre 資料夾，是 JDK 自己要使用的 Java 執行環境，因為 JDK 所提供的工具程式多數是由 Java 撰寫而成，而執行 Java 程式就需要 Java 執行環境，因而 JDK 有必要自己附帶 Java 執行環境。

JDK 所附帶的 Java 執行環境，與公用 JRE (Public JRE) 略有不同，稍後介紹 JRE 提供的資源時，會一併進行比較。

## lib 資料夾

lib

資料夾



在 lib 資料夾下放置了一些由 Java 撰寫而成的 JDK 工具程式，這些 JDK 工具程式被封裝為 [JAR \(Java Archive File\)](#) 檔案，JAR 檔案專門用來封裝 Java 程式，它採用與 zip 壓縮檔案同樣的格式，如果您想要看看 JAR 檔案實際上封裝了哪些 Java 工具程式，可以使用解壓縮軟體來開啟這些檔案。

## src.zip 壓縮檔

src.zip



在這個壓縮檔中提供 Java SE 標準類別庫的原始程式檔案，Java 開發人員有時會想要了解標準類別庫中的某個類別或方法是如何實作的，這時就可以開啟這個檔案中的相對應程式碼來觀看。



圖 2-12 Java SE 標準類別庫的實作原始碼

## JRE 提供的資源

JRE 是執行 Java 所必要的執行環境，您可以單獨安裝 JRE，而 JDK 本身為了執行所提供的工具程式，也會附帶自己所使用的 JRE。

JDK 自己附帶的 JRE 是在 JDK 安裝資料夾下的 jre 資料夾，而在這邊假設您的公用 JRE 是安裝在 C:\Program Files\Java\jre1.5.0\_07 下，這邊介紹這兩個 JRE 的主要差異，請分別開啟這兩個資料夾下的 bin 資料夾。

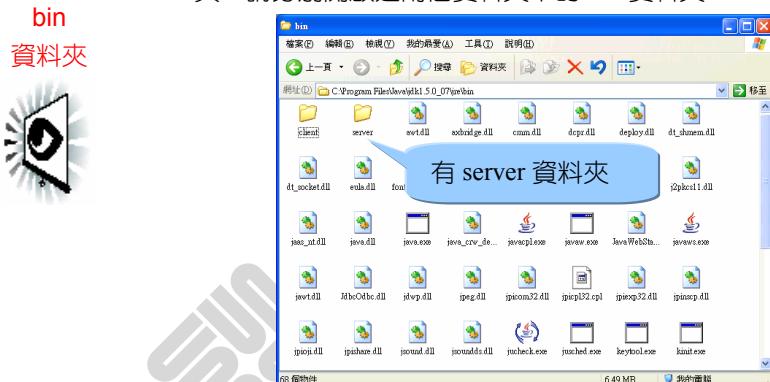


圖 2-13 JDK 中的 `ire/bin` 資料夾

與圖 2-14 中公用 JRE 的 bin 資料夾比較，您會發現公用 JRE 中沒有 server 資料夾。

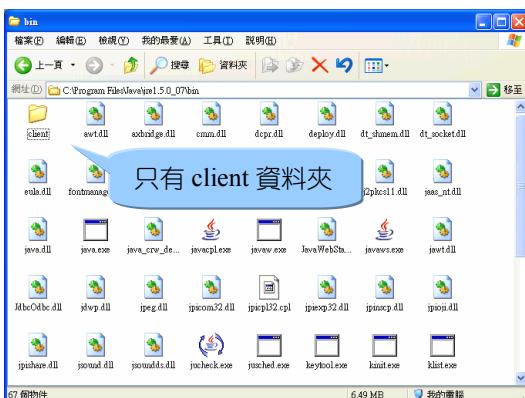


圖 2-14 JRE 安裝資料夾中 bin 資料夾

在執行 Java 程式時，您可以選擇 **client** 或 **server** 的選項來啟動 Java 程式，兩者的主要差別在於 **client** 選項的啟動時間較快，對於畫面的回應較為靈敏，主要用於桌面應用程式，客戶端預設是使用 **client** 選項；**server** 選項的啟動時間較慢並耗用較多的記憶體，通常用於網站應用程式，因為網站伺服器並不在乎啟動時多花一些時間，它需要的是夠多的資源來服務網路上的客戶，重視對來自網路請求的回應速度。



**重點提示**—無論是在 **client** 或是 **server** 資料夾中，您都可以找到 **jvm.dll** 這個檔案，這就是 Java 虛擬機器（JVM）之所在。

**rt.jar**



Java SE 編譯過的標準類別庫被放置在 JRE 中，無論是 JDK 自己附帶的 JRE 或是公用的 JRE，都可以在 JRE 資料夾下的 **lib** 資料夾中，找到 **rt.jar** 這個檔案，同樣的您可以使用解壓縮軟體開啟這個檔案，就可以發現已編譯完成的 Java SE 標準類別庫。

**lib/ext  
資料夾**



在 **lib** 資料夾下有一個 **ext 資料夾**，這是用來放置功能擴充套件（package）程式的資料夾，有些廠商所撰寫的 Java 程式，會將它們所開發的功能擴充套件放置到這個資料夾中。



### 自我測驗—配對 JDK 提供的檔案與作用

#### 定義

- JDK 下 **src.zip**
- JDK 下 **demos** 資料夾
- JRE 的 **lib** 資料夾下 **rt.jar**
- JDK 下 **bin** 目錄

#### 名詞

- JDK 範例程式
- Java SE 標準類別庫
- Java SE 標準類別庫原始碼
- JDK 工具程式

## 設定 Path 環境變數

對於 Java 的初學者，建議從文字模式下來操作各種 Java 工具程式，以增加對 JDK 的了解，然而今日多數的初學者已習慣圖型模式的操作方式，由於對文字模式工作原理與操作方式的不了解，不知道如何進行相關的環境變數設定，往往造成初學者對入門 Java 產生挫折感。

在這個小節中，將介紹以下的內容，這些內容與文字模式的操作是相關的：

- 如何在文字模式下切換工作路徑
- 文字模式下設定 Path 變數
- 設定 Path 系統環境變數

### 如何在文字模式下切換工作路徑

在這邊假設您所使用的是 Windows XP 作業系統，要開啟 Windows XP 的文字模式視窗，請執行「開始」功能表中的「執行」，在「開啟」的文字方塊中鍵入 cmd 指令（command，也就是進入指令模式）後按下「確定」即可開啟文字模式視窗。

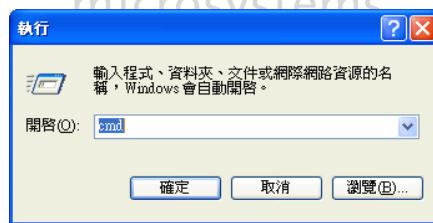


圖 2-15 執行 cmd 指令以開啟文字模式視窗

文字模式下要切換工作路徑是使用 cd 指令，使用時要指定切換目的地，以下的幾個指令是常用的：

cd \	切換至根目錄
cd ..	切換至上一層目錄
cd .\目錄名	切換至目前目錄下的相對目錄
cd 磁碟名:\目錄名	切換至所指定絕對路徑的目錄
磁碟名:	切換至指定磁碟



註—目錄就是資料夾的意思，例如在 Windows 下開啟一個目錄，就是指開啟一個資料夾。

假設您的 C 磁碟機下有 workspace 與 workspace\JNotePad 兩個資料夾，並且有 D 磁碟機，則以下是幾個簡單的指令示範與結果畫面：

```
C:\>cd workspace
C:\workspace>cd .\JNotePad
C:\workspace\JNotePad>cd \
C:\>cd c:\workspace\JNotePad\
C:\workspace\JNotePad>d:
D:>C:
C:\workspace\JNotePad>_
```

圖 2-16 文字模式下簡單的目錄切換示範

## 文字模式下設定 Path 變數

在已開啟的文字模式操作視窗中，試著鍵入 `java` 指令，以執行 `java` 工具程式，結果應該會出現以下的畫面：

```
C:\>>java
Usage: java [-options] class [args...]
              (to execute a class)
or  java [-options] -jar jarfile [args...]
          (to execute a jar file)

where options include:
  -client           to select the "client" VM
  -server           to select the "server" VM
  -hotspot          is a synonym for the "client" VM [deprecated]
                    The default VM is client.

  -cp <class search path of directories and zip/jar files>
  -classpath <class search path of directories and zip/jar files>
             ; separated list of directories, JAR archives,
             and ZIP archives to search for class files.
  -D<name>=<value>
                set a system property
  -verbose:[class|gc|jni]
                enable verbose output
  -version
                print product version and exit
  -version:<value>
                require the specified version to run
  -showversion
                print product version and continue
```

圖 2-17 執行了 `java` 工具程式

每個工具程式都會有個檔案名稱，當您 在文字模式下鍵入指令名稱，則作業系統會去執行相對應的程式檔案，現在請您鍵入 `javac` 指令，看看會有什麼樣的結果。

## 設定 Path 環境變數



圖 2-18 作業系統找不到指定的工具程式

結果有點令人意外，作業系統找不到 `javac` 工具程式，所以無法執行您所指定的工具程式，因而回報如圖 2-18 的錯誤。

### 查看 Path 變數



您要告訴作業系統，應當至什麼位置找到您所指定的工具程式，事實上作業系統在執行您所指定的程式時，會在 `Path` 變數所設定的路徑中尋找對應的檔案，您可以執行 `echo %Path%` 指令來看看目前的 `Path` 變數設定。

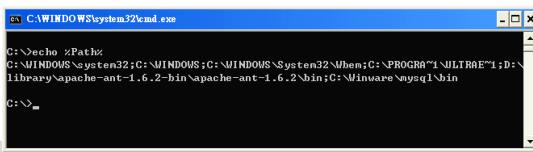


圖 2-19 查詢 Path 變數的設定

如果您是第一次安裝 JDK，並且從未修改過 `Path` 變數，則理論上您可以發現 `Path` 變數中並沒有包括 `JDK` 目錄下的 `bin` 資料夾，也就是 `JDK` 工具程式所在的資料夾，所以當您下 `javac` 指令時，作業系統不知道該至 `bin` 目錄下找 `javac` 工具程式。

您可能覺得有點奇怪！為什麼之前還沒設定 `Path` 變數，作業系統就找的到 `java` 工具程式？原因在於 `JDK` 會複製一份 `java` 工具程式的檔案至 `C:\WINDOWS\system32` 下，而 `Path` 變數中包括了這個路徑，因而您之前執行 `java` 指令時，作業系統可以順利找到 `java` 工具程式並執行。

## 設定 Path 環境變數



圖 2-20 JDK 會複製一份 java 工具程式至 system32 中

為了讓作業系統找到 JDK 的工具程式，您要設定 Path 變數包括 JDK 的 bin 資料夾，假設 JDK 的 bin 資料夾是在 C:\Program Files\Java\jdk1.5.0\_07\bin，則執行以下指令：



```
C:\>Path=C:\Program Files\Java\jdk1.5.0_07\bin;%Path%
C:>echo %Path%
C:Program Files\Java\jdk1.5.0_07\bin;C:Program Files\Java\jdk1.5.0_06\bin;c:\Winware\ruby\bin;C:\WINDOWS\System32;C:\WINDOWS\System32\Wbem;C:\PROGRA~1\ULTRAE\;D:\library\apache-ant-1.6.2-bin\apache-ant-1.6.2\bin;C:\Winware\mysql\bin;C:\Program Files\MySQL\MySQL Server 5.0\bin

C:>javac
Usage: javac <options> <source files>
where possible options include:
  -g                           Generate all debugging info
  -g:none                      Generate no debugging info
  -g:(lines,vars,source)         Generate only some debugging info
  -nowarn                       Generate no warnings
  -verbose                      Output messages about what the compiler is doing
  -deprecation                  Output source locations where deprecated APIs are u
sed
  -classpath <path>             Specify where to find user class files
  -cp <path>                    Specify where to find user class files
  -sourcepath <path>            Specify where to find input source files
  -bootclasspath <path>         Override location of bootstrap class files
  -extdirs <dirs>               Override location of installed extensions
  -endorseddirs <dirs>          Override location of endorsed standards path
```

圖 2-21 設定 Path 路徑包括 JDK 之 bin 資料夾

在 Path=C:\Program Files\Java\jdk1.5.0\_07\bin;%Path% 的設定中，等號後表示要設定的路徑，首先設定 JDK 的 bin 資料夾路徑，接著用分號區隔，而為了不影響原先 Windows 已有的 Path 變數設定，將%Path%加在後頭，原先的 Path 變數就仍然有效，而您可以在圖 2-21 中看到，作業系統這次成功的找到 javac 工具程式。

設定 Path 環境變數

## 設定 Path 系統環境變數

在文字模式下直接設定 Path 變數，只在當次開啟文字模式視窗有效，下一次重新開啟文字模式視窗時，就需要重新設定 Path 變數。

如果您覺得每次都要重新設定 Path 變數很麻煩，則您可以在系統環境變數中直接設定 Path 變數，每一次開啟文字模式視窗，系統環境變數都會自動套用。

要設定系統環境變數，在「開始」功能表中開啟「控制台」，接著在「系統」上按兩下滑鼠左鍵。



圖 2-22 在「系統」中設定環境變數

在「系統內容」對話框中，切換至「進階」頁籤，按一下「環境變數」按鈕。



圖 2-23 設定「環境變數」

## 設定 Path 環境變數

在「環境變數」對話框中，於下方的「系統變數」中選擇「Path」，並按下「編輯」按鈕。



圖 2-24 編輯 Path 系統變數

在「編輯系統變數」對話框中，將 JDK 的 bin 路徑加入，之後跟隨著一個分號，接著按下「確定」按鈕。

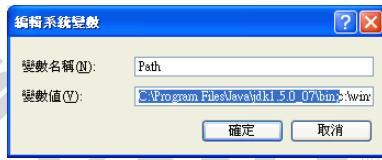


圖 2-25 加入 JDK 的 bin 路徑

接下來的步驟就是返回一個一個的視窗中按下「確定」按鈕，直到所有的視窗都設定完成。



**註** – Windows 2000 的設定方式與以上所介紹的類似，至於在 Windows 95/98 作業系統中，可以使用文字編輯器編輯 C 磁碟機的 autoexec.bat，在當中加入 JDK 的 bin 路徑：  
`SET PATH= C:\Program Files\Java\jdk1.5.0_07\bin`  
 接著重新開機即可。



**自我測驗** – 試著不看教材的指引，自行嘗試在文字模式下設定 Path 變數，並且嘗試在系統變數中設定 Path。

重新安裝 JDK、JRE

## 重新安裝 JDK、JRE

如果您已經安裝有 JDK 程式並想要移除它，例如移除舊版的 JDK 程式後重新安裝新的版本，則您可以執行「控制台」的「新增或移除程式」中，選擇 JDK 或 JRE 的項目並將之移除。



圖 2-26 移除 JDK、JRE



**自我測驗—如果不移除舊版本的 JDK、JRE，而直接安裝新版本的 JDK 或 JRE，會有什麼樣的結果發生？**

## 複習題

- 一、 JDK 的工具程式放置在哪一個資料夾中？
- 二、 如果打算放入廠商的擴充套件，可以直接放至 JRE 的哪一個資料夾中？
- 三、 JDK 工具程式多數是由 Java 所撰寫而成，它們實際上被放置在哪一個檔案中？
- 四、 為什麼 JDK 需要自己附帶 JRE？





---

## 第三單元

---

# 開發、執行 Java 程式

---

### 單元目標

單元目標



當完成本單元後，您將能學習到：

- 如何撰寫 Java 原始程式
- 編譯 Java 原始碼為位元碼
- 執行 Java 程式
- 設定 Classpath 環境變數

這個單元以 Step by step 的方式告訴您如何撰寫 Java 程式、編譯並執行，了解如何設定 Classpath 環境變數及其目的。

microsystems



討論—下列問題能夠幫助您了解 Java 程式的編譯與執行：

- 編譯過後的 Java 程式為什麼是位元碼？
- 設定 Classpath 環境變數的目的為何？
- Path 與 Classpath 的相同與不同之處有哪些？



## 相關資料



**相關資料**－下列參考資料能夠對本單元所討論的話題提供更詳細的說明。

- Setting the class path

<http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/classpath.html>  
設定 Classpath 的目的與方式



## 撰寫 Java 程式

撰寫 Java 程式時，所使用的是純文字檔案，在 Windows 下開啟的純文字檔案其副檔名為\*.txt 的檔案，不過 Java 原始碼檔案的副檔名是\*.java，在這個小節中，要教導您如何：

- 開啟純文字檔案並重新命名副檔名為\*.java
- 撰寫第一個 Java 程式

### 開啟純文字檔案並重新命名副檔名為\*.java

使用純文字檔案



要撰寫 Java 程式的原始碼時，所使用的是**沒有任何格式設定的純文字檔案**，如果是在 Windows 作業系統下，純文字檔案為副檔名為\*.txt 的檔案，可使用「**記事本**」開啟純文字檔案並進行編輯。

假設您的 C 磁碟機下已經有一個 workspace 資料夾，以後撰寫 Java 程式的示範都在這個資料夾中進行，在 Windows 下要新增一個純文字檔案最簡單的方式，就是按滑鼠右鍵執行「**新增**」、「**文字文件**」。

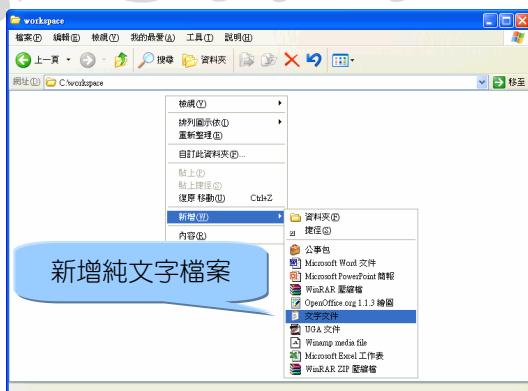


圖 3-1 新增文字文件

## 撰寫 Java 程式

新增文字文件完成後，您會發現多了一個圖示為 的檔案，在檔案上按兩下滑鼠左鍵即可開啟文字文件，Windows 下的文字文件預設副檔名為 \*.txt，您必須將之「另存新檔」，重新儲存為副檔名為 \*.java 的檔案，請執行「檔案」、「另存新檔」，然後在「存檔類型」選擇「所有檔案」，並重新命名檔案為副檔名為 \*.java 的檔案。

重新命名  
為 \*.java



圖 3-2 另存為 \*.java 的檔案



**註**－在 Windows 作業系統下，並不區分檔案名稱的大小寫，然而在其它作業系統下，通常區分檔案名稱的大小寫，建議您平時就養成區分檔案名稱大小寫不同的習慣，在日後使用其它作業系統撰寫程式時，就不至於在檔案名稱大小寫上發生困擾。

每一次都要透過另存新檔的方式來重新命名副檔名有點麻煩，最快的方式是直接點取一下 的檔案名稱後重新命名檔案，不過要注意的是有些 Windows 作業系統預設是不顯示副檔名的，這會造成檔案命名時的困擾，例如在不顯示副檔名的情況下，您也許會以為圖 3-3 中的檔案副檔名為 \*.java。

## 撰寫 Java 程式



圖 3-3 未顯示副檔名

在不顯示副檔名的情況下，事實上檔案名稱是 `HelloWorld.java.txt`，副檔名仍舊是`.txt`，可在文字模式下使用目錄查詢指令 `dir` 就看出來了。

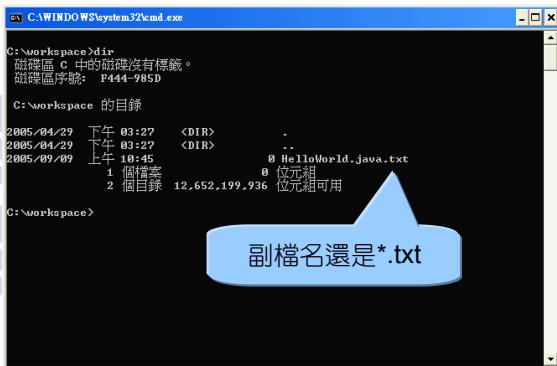


圖 3-4 文字模式下會顯示副檔名

為了避免檔案命名時的困擾，建議將 Windows 設定為顯示副檔名，這可以在「檔案總管」上執行「工具」、「資料夾選項」，在「檢視」頁籤中將「隱藏已知檔案類型的副檔案」選項取消。

顯示  
副檔名

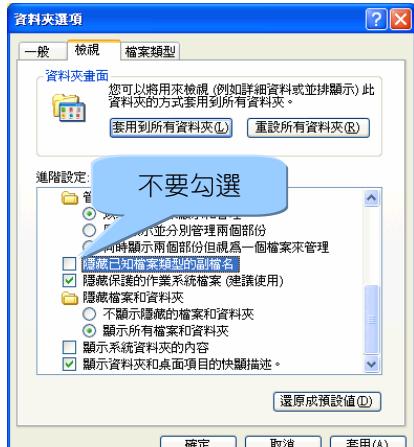


圖 3-5 文字模式下會顯示副檔名

如此設定之後，您就可以直接在 下的檔案名稱上點選，以直接修改副檔名，修改副檔名後，作業系統會請您確認是否修改副檔名，按「確定」即可。

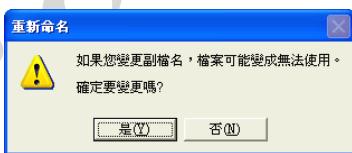


圖 3-6 確定修改副檔名

## 撰寫第一個 Java 程式

假設您在 C 磁碟機的 workspace 資料夾下開啟了 HelloWorld.java，接著就可以開始撰寫您的第一個 Java 程式了，請按照圖 3-7 中的程式進行撰寫，注意每個字的大小寫與標點符號，並注意空白不可以是全形空白字元。

第一個  
Java 程式



```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello ! World !");
    }
}
```

圖 3-7 第一個 Java 程式

## 撰寫 Java 程式

---

檔案中鍵入的文字其主要目的，就是跟電腦打交道，這個程式使用 Java 程式語言告訴電腦：「我是一個 [HelloWorld 類別](#)，請您按照我規定的[主要流程 \(main\)](#)來作事，流程中只有一個工作，在螢幕上顯示 Hello ! World ! 」。

了解這個程式的作用之後，接下來正式開始學習第一個 Java 程式語言，如同英文有單字、文法等結構，Java 程式語言也有關鍵字、語法等規範。

### 定義類別

#### 定義類別



撰寫 Java 程式的第一步都是從定義類別（`class`）開始，一個類別是一個 Java 程式的基本單位，定義類別時使用 `class` 關鍵字，後面跟隨著一個類別名稱。



**重點提示** – 關鍵字指的是程式語言中已經使用的一些單字，例如 `class` 關鍵字用來表示定義一個類別，當您為類別或程式中的一些元素取名時，不可以使用到這些關鍵字。

在類別名稱定義前加上一個 `public` 關鍵字，表示所定義的類別是[公開的](#)類別，公開的類別有兩種意義，第一個意義表示，檔案名稱在命名時，主檔名必須與類別名稱同名，就我們的例子而言，由於公開類別名稱為 `HelloWorld`，則檔案名稱必須取名為 `HelloWorld.java`，另外，一個 Java 原始檔案中可以定義數個類別，但是只能有一個公開的類別。



**重點提示** – 定義類別名稱時建議使用英文，雖然中文也可以被允許的，但由於並不是每個系統都可以順利顯示或讀取中文檔名，所以建議仍以英文取名，另外命名的慣例是類別名稱的首字大寫，並在名稱上表示出類別的作用。

公開類別的第二個意義為權限的管理，一個類別被定義為公開的類別，則表示其它的 Java 程式都可以使用這個類別，如果沒有定義類別為公開，則類別只能被同一個[套件 \(package\)](#) 管理下的其它 Java 程式所使用。

撰寫 Java 程式

可以把套件想成管理多個類別的單位，套件與套件間的類別檔案要相互使用的話，必須設定好相關權限，要讓目前的類別可以被其它套件中的類別使用，則目前類別必須設定為公開的（public）。

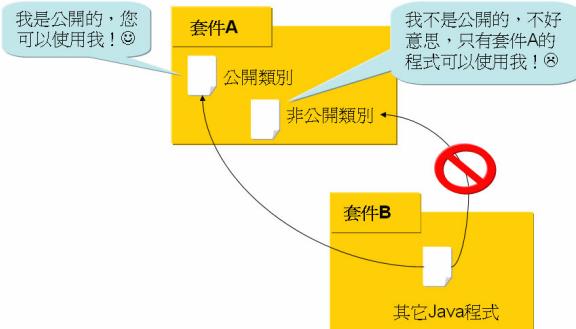


圖 3-8 公開類別可以被其它套件中的程式所使用

類別的定義範圍（Scope）由大括號 {} 包括，在括號之間稱為一個區塊（Block），在區塊之間的定義都屬於同一個類別。

### 定義主流程

main 方法



假設一個 Java 程式已經啟動，那麼該從哪邊開始執行呢？Java 程式的執行起點（Entry point）是從 main 方法（Method）開始，main 方法的寫法規定必須是：

`public static void main(String[] args)`

可以看到，有三個關鍵字來修飾 main 方法。必須是 `public`，因為一個公開的方法才可以被 Java 虛擬機器執行；必須是 `static`，Java 虛擬機器才可以直接呼叫 main 方法，而不用為該類別產生實例；`void` 表示方法執行完畢不傳回任何值。



註—Java 初學者要詳細了解這幾個關鍵字需要一些物件導向的基礎，建議初學者先將這種寫法當作一個規定即可。

在 main 方法後的括號定義了參數列（Parameter list），在呼叫方法時可以一併給方法一些執行過程中所必要的資訊，這些資訊稱之為引數（Argument），參數列中的參數可以儲存引數，以供方法使用。

main 括號中的 args 用來儲存啟動程式時，使用者一併給程式的命令列引數（Command line argument），目前雖然還沒必要使用命令列引數，但是仍要撰寫 String[] args，這是規定。

方法也是以花括號 { } 來定義作用範圍，括號間為方法區塊，在區塊之間的定義都屬於同一個方法。



註—對於 main 為何要這麼設計，可以進一步參考 Java™ Tutorial 中對 main 的介紹：

<http://java.sun.com/docs/books/tutorial/getStarted/application/main.html>

### 定義陳述句



陳述句

程式一開始執行 main 方法之後，就會執行您所指定的動作，在您的第一個 Java 程式中，只有一行 陳述句（Statement），您只要求電腦一件事：

```
System.out.println("Hello ! World !");
```

這一行的意思是使用 System 類別的 out 成員（Member），要求使用 out 上的 println()方法，在螢幕上顯示 Hello ! World ! 。

System 是 Java SE 提供的類別，一個類別中可以管理多個成員，其中 out 成員負責標準輸出（Standard output），只要將所要輸出的文字以雙引號"包括，並告知 println()方法，即可將文字顯示在螢幕上。

簡單的說，System.out.println()表示：請使用 System 的 out 上之 println()方法，顯示您所指定的文字。在 Java 中每一個陳述句結束時，記得加上分號。

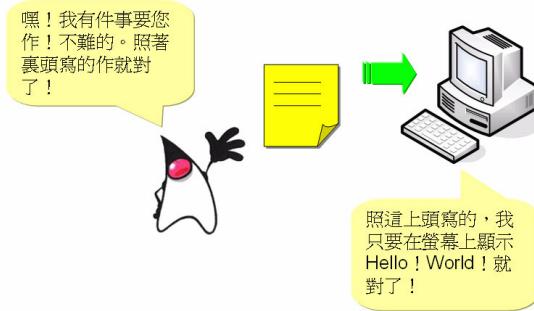


圖 3-9 用 Java 程式語言告訴電腦該作些什麼事



**自我測驗**－撰寫一個程式，在電腦螢幕上顯示您的名字與“您好”，別忘了替類別取個適當的名稱，另外留意在檔案名稱上需要注意的事項。



## 編譯 Java 程式

Java 語言屬於高階語言，但電腦只懂得低階的機器語言，在撰寫好 Java 原始程式之後，您需要一個翻譯員將原始程式翻譯為電腦所能理解的機器語言，這個翻譯員就是**編譯器（Compiler）**，編譯器不只幫您進行翻譯，它還會協助您檢查出程式中的一些語法或邏輯錯誤。

這個小節中將告訴您：

- 使用 `javac` 編譯器工具程式
- 看懂編譯器所提供的錯誤訊息

### 使用 `javac` 編譯器工具程式

#### 使用 `javac`



```
cmd C:\WINDOWS\system32\cmd.exe
C:\>cd c:\workspace
C:\workspace>javac HelloWorld.java
C:\workspace>
```

圖 3-10 使用 `javac` 編譯\*.java 檔案

您切換至 `C:\workspace` 資料夾下，執行 `javac` 工具程式並指定要編譯的`*.java` 檔案，如果指定的 Java 原始碼檔案沒有任何的錯誤，則編譯完成後不會有任何錯誤訊息出現，這時您的 `C:\workspace` 下應該會出現一個 `HelloWorld.class` 檔案。

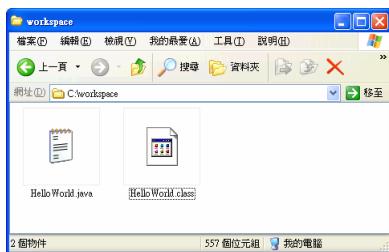


圖 3-11 編譯完成後會出現\*.class 檔案

編譯 Java 程式

對於每一個 Java 類別，編譯完成後都會出現一個與類別名稱相同，而副檔名為`*.class` 的檔案，這個檔案是位元碼（byte-code）格式，在執行 Java 程式時，會被 Java 虛擬機器轉換為目標平台所認識的原生碼（Native code）。

那麼 `javac` 工具程式到底作了什麼事情？您可以在執行 `javac` 指令的時候，同時使用 `-verbose` 選項，則執行編譯時的每一個步驟都會被顯示出來，例如：



-verbose  
選項

```
...\\Windows\\System32\\cmd.exe

C:\\workspace\\javac -verbose HelloWorld.java
[parsing started HelloWorld.java]
[parsing completed 78ms]
[search path for source files: ... c:\\workspace]
[search path for class files: [C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\rt.jar
  C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\jss.jar C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\ext\\dnsns.jar C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\ext\\jndi.jar C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\ext\\sunec.jar C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\ext\\sunpkcs1.jar ... c:\\workspace]]
[loading C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\rt.jar(java/lang/Object.class)]
[loading C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\rt.jar(java/lang/String.class)]
[checking HelloWorld]
[loading C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\rt.jar(java/lang/System.class)]
[loading C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\rt.jar(java/io/PrintStream.class)]
[loading C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\rt.jar(java/io/FilterOutputStream.class)]
[loading C:\\Program Files\\Java\\jdk1.5.0_07\\jre\\lib\\rt.jar(java/io/OutputStream.class)]
[wrote HelloWorld.class]
```

圖 3-12 使用`-verbose` 選項顯示編譯細節

編譯器首先會剖析\*.java 原始檔案，進行一些基本的語法檢查，接著搜尋原始碼路徑（可使用-sourcepath 選項設定）查看有無指定的相關類別之原始碼，然後搜尋類別路徑（可使用-classpath 選項設定）查看有無指定且已編譯完成的類別檔案，接著載入相關類別（例如 System、String 類別）以獲取類別資訊，編譯完成後將編譯結果寫至\*.class 檔案。



註 - 關於 `-sourcepath` 與 `-classpath` 的使用，稍後在介紹 Classpath 時會一併介紹。

初學者可記得這個基本流程，了解編譯器為您代辦了哪些事情，基本上如果程式原始碼沒有寫錯，則不必在編譯時加上 `-verbose` 選項，直接使用 `javac` 指令並指定原始檔案，接下來的工作交給編譯器即可。

## 編譯 Java 程式

**-d 選項**

有時將程式的\*.java 檔案與編譯完成的\*.class 檔案分開放置，在程式的檔案管理上是個不錯的方式，假設您的\*.java 檔案是放置在 C:\workspace\src 下，而您打算將編譯完成的\*.class 檔案放置在 C:\workspace\classes 下，則您可以使用**-d** 選項指定編譯完成後\*.class 的置放路徑，例如：

```
C:\Windows\System32\cmd.exe
C:\workspace>javac -d ./classes ./src/HelloWorld.java
C:\workspace>=
```

圖 3-13 使用-d 指定\*.class 的目的地

編譯時路徑的指定使用點號 . 開始表示相對路徑，您也可  
以指定絕對路徑，如果不指定-d 選項，則預設編譯完成的  
.class 是與\*.java 的所在位置放在一起。

## 看懂編譯器所提供的錯誤訊息

**常見錯誤  
訊息**



之前總是假設您所有的操作都是正確的，然而對於 Java 的初學者來說，有時並不會那麼的順利，這邊假設您犯了某些錯誤，有一些錯誤訊息提供給您，讓我們來看看問題出在哪邊？

**'javac' 不是內部或外部命令、可執行的程式或批次檔。**

事實上這並不是編譯器的錯誤訊息，而是作業系統告知的訊息，它告訴您找不到 javac 工具程式的所在，請重新複習第二單元，了解如何設定 Path 環境變數。

**error: cannot read: HelloWorld.java**

編譯器找不到您所指定的檔案，請確定 HelloWorld.java 存在，並且路徑沒有指定錯誤；還有一種可能性就是由於您的作業系統沒有顯示出副檔名，您的檔案事實上可能是 HelloWorld.java.txt，請看看前一個小節，了解如何正確的設定原始檔案的副檔名為\*.java。

---

**class HelloWorld is public, should be declared in a file  
named HelloWorld.java**

您定義了一個公開（public）類別，但是該類別的名稱與檔案的主檔名不同，Java 規定公開類別必須與所在檔案的主檔名相同。

**cannot find symbol**

這通常是打錯了方法或變數名稱所導致，或者是沒注意到大小寫問題，在 Java 中英文名稱的大小寫是有所區別的，請檢查看看每一個字母的大小寫都是正確的。

**package system does not exist**

找不到指定的類別或套件（package）名稱，請確定類別名稱沒有打錯，再次強調，Java 中對於字母大小寫是有所區分的（例如在這邊，您也許該將 system 改成 System）。

**HelloWorld.java:4: ';' expected**

每個陳述句結束時必須要加上分號，顯然的您忘記了這點。

**illegal character: \12288**

撰寫程式的過程中輸入了不合法的字元，通常的原因是您輸入了全型字元，例如全型空白，請去除或修正這些不合法的字元。



**自我測驗**－看懂編譯器所提供的訊息，對於撰寫程式而言是必備的功夫，以上僅列出一些常見的錯誤訊息，您還遇到了哪些錯誤訊息，寫下來與同學討論訊息的內容與解決方式。

執行 Java 程式

## 執行 Java 程式

在將 Java 原始碼編譯為位元碼之後，接下來您可以開始執行 Java 程式了，要執行 Java 程式必須使用 **java** 工具程式，並指定所要載入的類別，在這個小節中將告訴您：

- 使用 **java** 工具程式
- 看懂執行錯誤時的訊息

## 使用 **java** 工具程式

使用 **java**  
工具程式



假設您編譯完成的類別是放置在 C:\workspace，類別的檔案名稱是 **HelloWorld.class**，則您可以如圖 3-14 載入該類別並執行：

```
C:\WINDOWS\system32\cmd.exe
C:\workspace>java HelloWorld
Hello! World!
C:\workspace>
```

圖 3-14 使用 **java** 工具程式

程式成功執行了您所指定的動作：顯示"Hello ! World !"。要注意的是，使用 **java** 工具程式時只要指定類別名稱即可，**不需要加上\*.class 副檔名**。



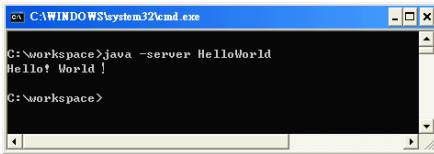
**註**—使用 **java** 工具程式時，同樣也可以使用**-verbose** 選項，來看看 **java** 工具程式在執行時進行了哪些動作，這是進階議題，初學者尚不用深入了解。。

這邊有個問題值得討論，您所使用的 **java** 工具程式使用了哪個 Java 執行環境？是 **JDK** 本身附帶的 **JRE**？還是公用 **JRE**？

如果您將 **JDK** 的 **bin** 資料夾之路徑設定於 **Path** 變數時，是放在最前面，則執行 **java** 時，作業系統會依 **Path** 變數中設定的路徑順序來找尋目標程式，所以啟動的 **java** 工具程式會執行 **JDK** 自己附帶的 **JRE**。

**執行 Java 程式**

測試所啟動的執行環境是 JDK 的 JRE 或是公用的 JRE，有一個簡單的方法，還記得公用的 JRE 並沒有`-server`選項所需的資料夾嗎？如果您可以順利使用`-server`選項，則表示所使用的是 JDK 的 JRE，例如：



**圖 3-15 使用的是 JDK 的 JRE**

如果執行時使用`-server`選項，卻出現 [Error: no `server' JVM](#) 的資訊，表示所使用的是公用 JRE。



**重點提示**－當您的系統中不只安裝一套 JRE 時，了解所使用的是哪一個 java 工具程式，以及是所啟用的是哪一套 JRE 是很重要的。

## 看懂執行錯誤時的訊息

### 常見錯誤

#### 訊息



假設您在之前的介紹的操作過程中，不小心犯了某些錯誤，那麼您可能無法順利執行 Java 程式，當 Java 執行環境發現執行過程中有錯誤時，會回報相關的錯誤訊息。

#### `java.lang.NoClassDefFoundError`

java 工具程式找不到您所指定的類別，有幾個可能的原因，您可能指定了錯誤的類別名稱，或是您所指定的類別檔案不存在。



**重點提示**－更具體來說，java 工具程式無法在類別路徑（Class path）中找到您所指定的類別，關於類別路徑的設定會在下一個小節進行說明。

執行 Java 程式

---

**java.lang.NoSuchMethodError: main**

您所指定的類別沒有 `main` 方法，或者是 `main` 方法撰寫錯誤，要注意程式執行的進入點（Entry point）`main` 方法必須是：[public static void main\(String\[\] args\)](#)。



**自我測驗**—有沒有辦法在別的路徑下，例如在 C:\ 路徑下，使用 `java` 工具程式執行 C:\workspace 下的類別？



## 設定 Classpath 環境變數

如果您的類別檔案是放置在 C:\workspace，而您打算在 C:\ 下執行這個檔案，則以下的執行方式是無效的。

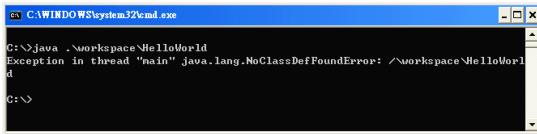


圖 3-16 直接指定類別檔案的路徑是無效的

Java 程式是執行於 Java 虛擬機器，也就是 JVM 之上，JVM 對於 Java 程式而言，就是一台虛擬的電腦設備，在第二單元討論 Path 變數的設定時，您知道要讓作業系統找到您所指定的程式，則必須在 Path 變數中告知程式所在的路徑，對於 Java 程式而言，JVM 就相當於是它的作業系統，為了讓 JVM 找到您所指定的 Java 程式，您要告訴 Java 程式的所在，您要提供（設定）Classpath 變數。

簡單的說，Path 變數之於作業系統的作用，就相當於 Classpath 變數對於 JVM 的作用。

在這個小節中，您將會學習到：

- -classpath 與 -sourcepath 選項的作用
- 如何設定 Classpath 環境變數
- 如何在系統變數中加入 Classpath 變數

## 如何使用-classpath 選項

使用  
-classpath



如果您的類別檔案是放置在 C:\workspace，而您打算在 C:\ 下執行這個檔案，您可以使用 -classpath 選項來提供類別所在的路徑，例如：

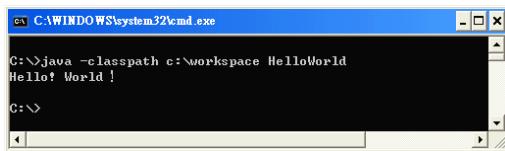


圖 3-17 使用-classpath

## 設定 Classpath 環境變數

您使用-classpath 選項告訴 JVM，可以在 C:\workspace 找到 HelloWorld 類別，然而您有疑問，為什麼在 C:\workspace 下執行 java 工具程式時，不必使用-classpath 選項呢？因為預設上 Classpath 就是當前的工作路徑，無需額外設定。

事實上 javac 工具程式也有-classpath 選項可以使用，當您的程式中使用到某個類別，在編譯時若該類別不是在當前的工作路徑下，則您必須使用-classpath 選項提供該類別的所在路徑。

### -sourcepath 選項



在使用 javac 工具程式時，還有一個-sourcepath 選項，這個選項的作用與-classpath 選項類似，只不過-classpath 所指定的是已經編譯完成的類別檔案（也就是\*.class）之路徑所在，而-sourcepath 所指定是尚未編譯完成的類別原始碼程式（也就是\*.java）之所在。

編譯器在編譯原始碼時，如果遇到所指定的類別，會先搜尋 sourcepath 路徑下是否有指定的類別，然後搜尋 classpath 路徑下是否有指定的類別，可以使用-verbose 選項來看到這個過程（參考圖 3-12），預設上 sourcepath 與 classpath 的設定是相同的。

## 如何設定 Classpath 環境變數

### Classpath 環境變數



與設定 Path 環境變數時類似，您也可以在當前開啟的文字模式視窗下設定 Classpath 變數，例如：

```
C:\>set Classpath=.;C:\workspace
C:\>java HelloWorld
Hello! World!
C:\>
```

圖 3-18 設定 Classpath 環境變數

在設定 Classpath 時包括了當前路徑，也就是使用 . 設定，每個設定以分號區隔，設定好 Classpath 環境變數之後，可以不再使用-classpath 選項，java 工具程式會讀取 Classpath 環境變數中的路徑設定。

## 設定 Classpath 環境變數

如果您已經設定了 Classpath 環境變數，而執行時又使用 -classpath 選項提供路徑的話，那麼-classpath 選項所提供的路徑會覆蓋 Classpath 環境變數的設定，而不會有加成的作用，例如若故意如圖 3-19 執行，則無法順利執行程式。

圖 3-19 以-classpath 選項提供的路徑為主

## 如何在系統變數中加入 Classpath 變數

Classpath  
系統變數



您也可以在系統變數中直接新增 Classpath 變數，如此每次開啟文字模式視窗時，就會自動套用變數內容，您可以先按照第二單元中，設定 Path 系統變數的操作步驟中，先操作至開啟「環境變數」對話框，在該對話框中按「系統變數」中的「新增」，並如圖 3-20 新增 Classpath 系統變數：

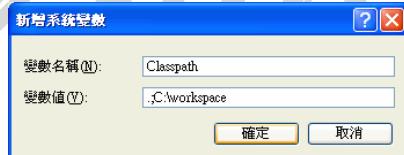


圖 3-20 新增 Classpath 系統變數

建議您在設定 Classpath 系統變數時，保留當前工作路徑為 Classpath 的設定之一。



**註** – Windows 2000 的設定方式與以上所介紹的類似，至於在 Windows 95/98 作業系統中，可以使用文字編輯器編輯 C 磁碟機的 autoexec.bat，在當中加入 Classpath 設定：

**SET CLASSPATH=.;C:\workspace**  
接著重新開機即可。



**自我測驗** – 說明 Path 與 Classpath 的設定目的之相同點與不同點。

複習題

---

## 複習題

- 一、 定義類別時使用哪一個關鍵字？
- 二、 如果類別是公開類別，名稱為 HelloJava，則檔案名稱該如何命名？
- 三、 如果編譯時要指定編譯完成的類別檔案置放路徑，該使用哪一個選項？
- 四、 設定 Classpath 的目的為何？



---

## 第四單元

---

# 註解、資料、變數

---

### 單元目標

**單元目標**



當完成本單元後，您將能學習到：

- 如何撰寫註解
- 列出 Java 程式語言的八個基本型態
- 變數的宣告與使用
- 使用晉升（Promotion）與轉型（Casting）

這個單元以程式實例告訴您如何在 Java 程式中加入註解，以供程式人員了解程式的目的，並了解資料型態、變數的種類與作用。

microsystems



討論—下列問題能夠幫助您了解資料型態與變數：

- 為什麼程式中要加入註解？註解如何才能簡單明瞭？
- 為什麼要規定不同種類的資料型態？
- 如何宣告變數？變數的指定有哪些必須注意的？



## 相關資料



**相關資料**－下列參考資料能夠對本單元所討論的話題提供更詳細的說明。

- The Java<sup>TM</sup> Tutorial. [Online]. Available:  
<http://java.sun.com/docs/books/tutorial/>  
Java 官方教材
- Variables  
<http://java.sun.com/docs/books/tutorial/java/nutsandbolts/variables.html>  
您可以從這邊查到更多有關資料型態與變數的文件
- Java 學習筆記, 畢峰出版社  
這是一本適合剛入門的書籍, 它解釋的觀念甚至超過更進階的書籍。



## 為程式加入註解

基本上程式的目的是告知電腦，按照您所指定的流程來運作，程式語言的目的是在與電腦溝通，然而另一方面，撰寫程式的是人類，您也必須看懂您程式的意圖，必要的時候，您可以為您的程式加入說明，在日後可以幫助您了解當初是如何撰寫程式的，如果有別的程式設計人員打算維護您的程式，也可以藉由程式中的說明迅速了解程式的作用。

您為程式所加入的說明，以程式設計的術語來說稱之為「[註解](#)」（Comment），在這個小節您可以了解：

- 註解的結構
- 註解的風格

### 註解的結構

您應該在您撰寫的程式中加入註解，這樣會使程式更具有可讀性。對於團體開發的大型程式而言，因為多個程式設計人員都必須閱讀程式碼，所以註解顯得更加重要。當在維護程式時，註解也能夠幫助新的程式設計人員了解程式碼的作用。

有兩種主要的註解種類可以使用：

#### 單行註解

#### 單行註解（Single-line comment）



下面的程式範例告訴您如何加入單行註解：

##### 程式碼 4-1 SingleCommentDemo.java

```

1 public class SingleCommentDemo {
2     public static void main(String[] args) {
3         // 顯示 Hello ! World !
4         System.out.println("Hello ! World ! ");
5     }
6 }
```

單行註解

標記“`/*`”告訴編譯器忽略此標記之後到此行結尾之間的內容，例如有 `SingleCommentDemo` 的第 3 行包括了單行註解，編譯器不會處理該行。

### 傳統註解(Traditional comment)

傳統註解



有時候註解文字必須分作好幾行來撰寫，一行一行的使用單行註解標記“`/*`”並不方便，您可以使用傳統的註解方式，在“`/*`”與“`*/`”之間撰寫註解文字，例如：

#### 程式碼 4-2 TraditionalCommentDemo.java

```

1  /*
2   * 名稱：第一個 Java 程式練習
3   * 目的：在螢幕上顯示 Hello ! World !
4  */
5
6 public class TraditionalCommentDemo {
7     public static void main(String[] args) {
8         System.out.println("Hello ! World ! ");
9     }
10 }
```

傳統註解



標記“`/*`”與“`*/`”告訴編譯器忽略“`/*`”標記之後到下一個“`*/`”之間的內容，例如編譯器不會處理 `TraditionalCommentDemo` 類別中第 1 行至第 4 行的內容。



註－之所以稱之為傳統註解，是因為這個註解方式是從 C 語言開始就有的，Java 也採用同樣的註解方式，因為這種註解可以多行，所以也俗稱「多行註解」。

要注意的是傳統註解是以“`/*`”作為註解開始，以遇到“`*/`”作為註解結束，所以不可以使用巢狀 (Nested) 的方式來撰寫傳統註解，例如下頁的註解方式是錯誤的：

## 為程式加入註解

```

/*
    名稱：第一個 Java 程式練習
    /*
        目的：在螢幕上顯示 Hello ! World !
*/

```



編譯器遇到第一個“`/*`”符號時，就會以為註解範圍已經結束，而在下一次遇到“`/*`”時，就會視它為不合法的程式撰寫，因而發出編譯錯誤的訊息。



**重點提示—第三種註解的方式**，稱為**文件註解 (Document comment)**。這種方式的註解是藉由 JDK 內附的工具 `javadoc` 來產生文件。事實上，在 Java Standard Edition SDK 上所有的類別函式庫說明文件都是由 `javadoc` 工具所產生。文件註解必須以斜線和兩個星號開始 (`/**`)，而由一個星號和一個斜線結束 (`*/`)。之前傳統註解的例子也是合法的文件註解。

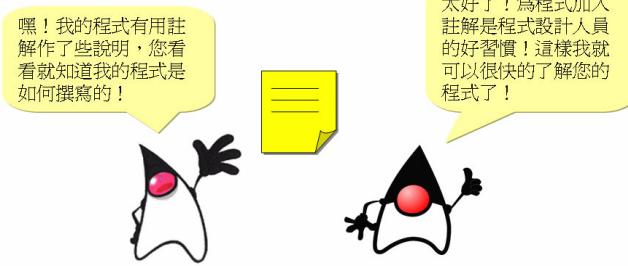


圖 4-1 為程式加入註解是個很好的習慣

## 註解的風格

撰寫註解就像是在撰寫文件，註解在於輔助程式設計人員閱讀程式，如何將註解撰寫的簡單明瞭、易於閱讀也是很重要的，這邊介紹幾種常見的註解風格。

## 為程式加入註解

許多的程式設計者喜歡在變數（Variable）宣告時用註解來  
明變數的作用，例如：

```
int numberOfRowsInSection; // 學生的學號
```

## 註解風格



有的開發人員會在每一個類別和方法的最後一行加上單行  
註解，使得區塊範圍顯而易見，例如：

```
public class HelloWorld {
    public static void main(String[] args) {
        ...
    } // main 結束
} // HelloWorld 結束
```

遇到暫時不想執行的陳述句（Statement），可以使用單行  
註解將之暫時註銷，編譯器就不會去處理該行，例如下面的  
程式不會顯示"Hello ! ?"：

```
public class HelloWorld {
    public static void main(String[] args) {
        // System.out.println("Hello ! ?");
        System.out.println("World ! ?");
    }
}
```

遇到大範圍的程式片段不想執行，可以用傳統註解將該段程  
式碼設為註解文字，如此編譯器就不會去處理，例如：

```
public class HelloWorld {
    public static void main(String[] args) {
        /* System.out.println("Hello ! ?");
        System.out.println("Good ! ?");
        ....一大堆程式碼
        */
        System.out.println("Hello ! ?");
    }
}
```

使用註解的  
另一種方式



## 為程式加入註解

有些程式開發人員喜歡在程式一開始時撰寫程式名稱、目的等文字，並用許多 "\*" 讓註解看了像是標題文字，例如：

```
*****  
* Variable Declaration Section      *  
* 變數宣告區域                  *  
*****
```



**重點提示**—有機會的話，可以打開 JDK 中 src.zip 檔案的原始程式來看看，了解一下官方程式碼中是如何撰寫註解。

記得要養成使用  
註解的好習慣！



圖 4-2 養成使用註解的習慣



**自我測驗**—底下哪一個是合法的註解開始符號？如果不合法，試說明不合法的原因。

選擇：/\*

選擇：\*/

選擇：//

選擇：/

## 認識資料型態

### 資料型態



程式在運行時，必須先將相關的資料載入至記憶體中執行，然而記憶體的空間是有限的，不同需求的資料必須配給它們不同長度的記憶體空間，而另一方面，配給適當的記憶體空間，也可以讓 CPU 在處理資料時更有效率，這就是為什麼您必須為不同資料使用不同的的資料型態（Data type）。

如果將資料比喻為生產材料的話，大樣的生產材料您必須使用大的箱子裝，小件的生產材料則使用小箱子裝，如果您分配大箱子裝小件的生產材料，不但浪費空間，運送這些箱子時還很麻煩。

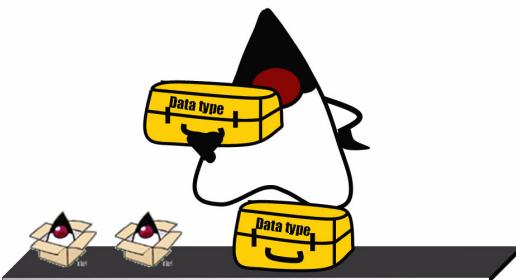


圖 4-3 為資料分配適當的資料型態

在 Java 語言中有八種基本型態（Primitive Type），可分作四個大類：

- 整數型態（Integral type）—byte、short、int 和 long
- 浮點數型態（Floating point type）—float 和 double
- 字元型態（Textual type）—char
- 邏輯型態（Logical type）—boolean

## 整數基本型態

### 整數型態



在 Java 程式語言中共有四個整數基本型態，分別以 byte、short、int 和 long 四個關鍵字來識別。這些型態以不使用小數點的方式來儲存數字。

## 認識資料型態

整數值有大有小，所以整數中才又區分有四種不同的資料型態，以儲存人的年齡為例，byte 的變數型態就可以滿足儲存的要求，因為 byte 型態的範圍已涵蓋了人們年齡的範圍。

表 4-1 的表格列出所有整數型態、包含它們所使用的記憶體容量大小以及所有可能值的範圍。

表 4-1 整數型態

型態	長度	範圍	範例
byte	8 bits	$-2^7$ 到 $2^7 - 1$ ( -128 到 127 共 256 個)	2 -114
short	16 bits	$-2^{15}$ 到 $2^{15} - 1$ ( -32,768 到 32767 共 65,535 個)	2 -32699
int	32 bits	$-2^{31}$ 到 $2^{31} - 1$ ( -2,147,483,648 到 2,147,483,647 共 4,294,967,296 個)	2 147,334,778
long	64 bits	$-2^{63}$ 到 $2^{63} - 1$ ( -9,223,372,036,854,775,808 到 9,223,372,036,854,775,808 共 18,446,744,073,709,557,616 個)	2 -2,036,854,775,808L 08L 1L



**重點提示**一位元 (bit) 是記憶體計量的最基本單位，如果 8 個位元則稱為一個位元組 (byte)。

直接在程式中寫下一個數值，該數值稱之為**字面值 (Literal value)**，編譯器會自動為該數值配給記憶體空間，以整數值為例，在一般情況下如果您寫下一個整數，編譯器將自動設定為 int 型態，除非您自行指定或是使用 L，例如寫下 10L 的話，則編譯器會為該數值配給 long 型態的空間。

您可以直接將數值指定給標準輸出 (Standard output)，在螢幕上顯示數值，例如：

`System.out.println(10);`

配給 int 型  
態空間

但是如果您的程式中撰寫了以下的程式片段：

`System.out.println(2036854775808);`



則您在編譯時會發生以下的錯誤訊息：

```
integer number too large: 2036854775808
    System.out.println(2036854775808);
^
1 error
```

之前說過，當您在程式中寫下一個整數值時，編譯器預設是配給 int 型態的記憶體空間，然而 2036854775808 已經超出了 int 型態的記憶體空間所能儲存的範圍，才會出現以上的錯誤訊息，簡單的說，您硬是要將大件貨物塞入小箱子中，當然就會有問題發生。

在撰寫整數時，可以用 L 加於整數之後，這是在告知編譯器：「請為這個數值配給 long 型態的記憶體空間」，如此才可以順利編譯，例如：

```
System.out.println(2036854775808L);
```



圖 4-4 有時需要告知編譯器資料型態的資訊

在撰寫整數值時，也可以使用 16 進位或 8 進位的方式來撰寫，以 16 進位撰寫時，開頭要寫 0x，例如寫作 0xFF（10 進位即 255），以 8 進位方式撰寫時，開頭要寫 0，例如寫作 017（10 進位即 15），當使用標準輸出顯示在螢幕時，則都以 10 進位方式顯示，例如以下的程式碼片段會在螢幕上顯示 10 進位的 255 與 15：

```
System.out.println(0xFF);           16 進位
System.out.println(017);            8 進位
```

## 浮點數基本型態

### 浮點數



`float` 與 `double` 為浮點數的兩個型態。它們是用來儲存右邊有出現小數點的數值，如 12.24 或 3.14159，表 4-2 說明這兩個浮點數型態的資訊。

表 4-2 浮點數型態

型態	長度	範例
<code>float</code>	32 bits	99F -327,456.99.01F 4.2E6F(工程標記記為 $4.2 \times 10^6$ )
		-1111 2.1E12
<code>double</code>	64 bits	999,701,327,456.99.999

當您 在程式中寫下一個浮點數值，編譯器預設會配給 `double` 型態的記憶體空間給該數值，如果您只打算配給浮點數 `float` 型態的記憶體空間，則必須使用 `F` 來指定，例如：

```
System.out.println(3.14F);
```



註—`double` 型態可以使用在需要更大範圍及準確性的情況下。

## 字元基本型態

### 字元型態



另一個您必須儲存與運用的資料型態是單一字元資料。這個 `char` 基本型態是用來儲存 16 bits 的單一字元，例如'Y'字元，在 Java 程式語言中，[字元必須使用單引號括住](#)，例如：

```
System.out.println('Y');
```



`char` 型態並不會真正儲存像'Y'這樣的字元，而是將字元轉換成一連串的位元儲存。而位元與字元的對應是根據程式語言所使用的字元集所對照而成。

## 認識資料型態

大多數的電腦語言都是使用 ASCII 碼 (American Standard Code for Information Interchange)，ASCII 是一個 8 位元的字元集，對所有的英文字元、標點符號、數字等都有所對應。

### Unicode



在 Java 程式語言中所使用的是 16 位元的 Unicode 碼，Unicode 能夠儲存在現今世界的各種語言中大部份必須顯示的字元。因此您可以用任何語言撰寫程式，程式就能夠正常地運作，並正確地顯示該語言。

為了與 ASCII 碼相容，Unicode 碼內包含有 ASCII 碼的子集合（前 128 個字元），例如您寫了一個'Y'字元，實際上該字元在 Java 中是以 00000000 00111011 的方式儲存，編碼數值是 59，對應 ASCII 編碼表的'Y'字元。

 註—您可以在下列的網址中找到 ASCII 編碼表：

<http://www.lookuptables.com/>

在 Java 中無論是中文字元或是英文字元，一律佔 16 bits 長度的記憶體空間，所以您也可以這麼顯示中文字元：

```
System.out.println('中');
```

您也可以使用 Unicode 編碼的方式來指定字元，只要以 \u 開始並指定字元編碼即可，例如下面的程式片段會顯示 H、e、I、l、o 五個字元：

```
System.out.println('\u0048');
System.out.println('\u0065');
System.out.println('\u006C');
System.out.println('\u006C');
System.out.println('\u006F');
```



H  
e  
l  
l  
o

## 邏輯基本型態

### 邏輯型態



電腦程式最常執行的動作是做決策。決策的結果—不論是正確 (true) 或錯誤 (false)—在 Java 中都能夠以 boolean 型態的數值來表示，在 Java 程式語言中：

- boolean 型態有兩種數值 true 或 false
- 應用於條件運算時，運算式的結果可為 true 或 false

## 認識資料型態



自我測驗—將左列的資料型態搭配相對應的位元大小。

資料型態	位元數
double	1
boolean	8
char	16
int	32
byte	64



## 宣告、使用變數

在 Java 程式中寫下一個數值，則編譯器會為該數值配給一個記憶體空間，然而在程式設計時，您經常會需要先擁有記憶體空間，該空間中要放些什麼數值您事先並無法得知，必須要等程式開始執行並運算之後，才會將運算結果儲存至該空間中。

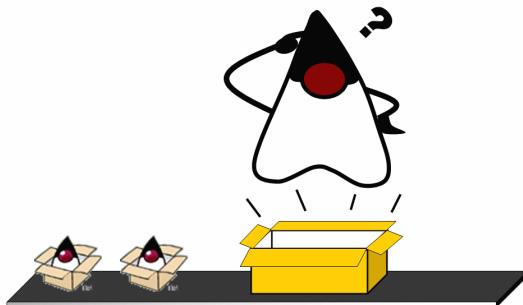


圖 4-5 您需要一個空間，但還不知道要放些什麼！

變數



您可以先宣告一個記憶體空間，告知編譯器這個空間將儲存的資料是什麼型態，並且這個記憶體空間會有個名稱，稍後您執行程式時，可以使用這個名稱，將指定資料存入預先宣告的記憶空間之中，這樣的名稱與空間叫作「[變數](#)」([Variable](#))，而宣告的這個動作稱之為[變數宣告](#)。

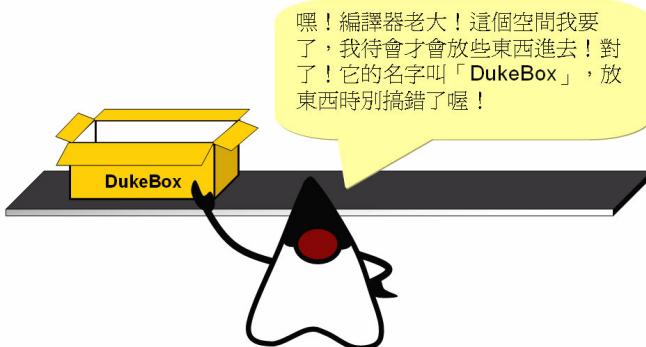


圖 4-6 變數宣告為您保留記憶體空間

宣告、使用變數

## 變數的宣告與使用

### 宣告變數



在方法中進行變數宣告的語法如下：

```
type identifier = [value];
```

其中 **type** 用來宣告變數的資料型態，可使用關鍵字 **byte**、**short**、**int**、**long**、**float**、**double**、**boolean** 等來宣告，主要看您需要多大的空間來儲存資料。

**identifier** 是您要對變數取的名稱，變數的名稱必須能表示出該變數的作用，例如用 **ageOfStudent** 這樣的名稱來表示，這個變數中儲存的資料是學生的年齡。

=是指定運算子(**Assignment operator**)，用來將右邊的 **value** 指定給左邊的變數來儲存。

一個宣告變數與使用變數的例子如下：

### 程式碼 4-3 VariableDemo.java

```
1 public class VariableDemo {
2     public static void main(String[] args) {
3         int ageOfStudent = 18;
4         char sexOfStudent = 'M';
5
6         // print() 方法在顯示文字後不換行
7         System.out.print("學生年齡: ");
8         System.out.println(ageOfStudent);
9         System.out.print("學生性別: ");
10        System.out.println(sexOfStudent);
11
12        ageOfStudent = 20;
13        sexOfStudent = 'F';
14
15        System.out.print("學生年齡: ");
16        System.out.println(ageOfStudent);
17        System.out.print("學生性別: ");
18        System.out.println(sexOfStudent);
19    }
20 }
```

宣告並  
指定值

取得變  
數的值

## 宣告、使用變數

注意在第 3 行中，宣告 char 變數在指定值時，字元值必須使用單引號包括，在宣告變數並指定值之後，您可以直接使用變數名稱來取得變數所儲存的值，如第 8 行與第 10 行所示範的，而如果您打算修改變數中所儲存的值，則再使用指定運算子來指定，如第 12 行與 13 行所示範的，不必再重新宣告變數，範例 4-3 的編譯與執行結果如下：

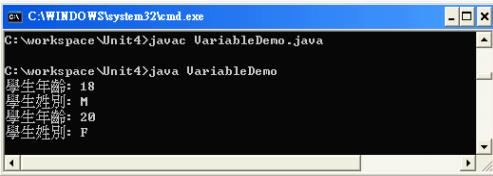


圖 4-7 範例 4-3 的執行結果

變數的大  
小寫區分



在 Java 中，變數的名稱是有大小寫之分的，也就是說 `ageOfStudent` 與 `ageofstudent` 這樣的名稱是兩個不同的變數。

您也可以指派一個變數的值給其它變數，右邊變數的值會複製一份給左邊變數，例如以下的程式片段中，`age` 變數的值會是 `ageOfStudent` 變數的值，所以程式會在螢幕上顯示 18：

```
int ageOfStudent = 18;
int age = ageOfStudent; // 指定變數給另一個變數
System.out.println(age);
```

`final`  
關鍵字



有時候您想要指定一個值給變數，之後不再改變它的內容，目的是為了方便使用名稱來取用某個數值，例如用 `PI` 這個變數名來取用 3.14159 的值，則您可以在宣告變數時，一併使用 `final` 關鍵字，例如：

```
final double PI = 3.14159;
```

使用 `final` 關鍵字宣告的變數，在指定值之後就不能改變它的值，如果您試圖改變，則編譯器會告訴您不可以這麼作，舉個例子來說：

```
final double PI = 3.14159;
PI = 3.14; // 試圖改變 final 所宣告的變值之值
```

如果程式中撰寫了以上的程式片段，則編譯時就會出現 `cannot assign a value to final variable` 的錯誤訊息。



**重點提示**－使用 `final` 宣告的變數，通常變數名稱會以全部大寫來表示，這可以提醒程式設計人員所使用的變數可能是使用 `final` 宣告的變數。

宣告變數會使得編譯器提供一塊記憶體空間，供您未來儲存數值，空間被宣告之後，當中可能含有未知的數值，在未指定數值給變數之前，您不可以直接取用變數的值，事實上編譯器也會發生錯誤訊息，例如：

```
int ageOfStudent; // 宣示變數但不指定值
System.out.println(ageOfStudent); // 顯示變數
```



您還沒指定  
值給它呢

上面的程式片段在編譯時將會出現 `variable ageOfStudent might not have been initialized` 的錯誤訊息，您必須在指定值之後才能取用變數，例如：

```
int ageOfStudent = 18;
System.out.println(ageOfStudent);
```

## 晉升 (Promotion) 與轉型 (Casting)

在宣告變數時必須使用關鍵字 `byte`、`short`、`int`、`long`、`float`、`double`、`boolean` 等告知編譯器，您將儲存什麼資料型態的資料至變數中，而您在程式中寫下一個字面值時，編譯器會預設它的資料型態，如果在宣告變數並指定值時，這樣寫不會有什麼問題。

```
int age = 18;
double score = 90.5;
```

因為編譯器會為數值 18 分配 `int` 型態的記憶體空間，為數值 90.5 分配 `double` 型態的記憶體空間，但如果這這麼宣告的話，編譯器會出現錯誤訊息：

```
int age = 2036854775808;
float score = 90.5;
```

因為 2036854775808 超過了 int 型態變數所能儲存的範圍，編譯器會出現 **integer number too large** 的錯誤訊息；90.5 編譯器預設為它提供了 double 型態的空間，而您卻指定這個值給 float 型態的變數，編譯器會出現 **possible loss of precision** 的錯誤訊息。

當提供的數值其資料型態與變數的資料型態不符時，我們必須考慮數值的**晉升（Promotion）**與**轉型（Casting）**問題。

### **晉升（Promotion）**

**晉升**



如果變數的資料型態其儲存範圍比指定的數值之資料型態為大時，則會發生數值型態晉升的動作，簡單的說，就是數值的型態會自動提昇為符合變數之資料型態，舉例來說，下面的程式碼會發生晉升的動作：

```
long money = 10;
```

10 被指定給 money，由於變數 money 是 long 型態，資料型態自動從 int 變為 long，資料型態晉升動作發生時，編譯器不會有任何的訊息出現。

### **轉型（Casting）**

如果數值的資料型態之儲存範圍比變數的資料型態為大時，由於原數值無法將所有的容量儲存至變數中，因而編譯器會提出錯誤訊息，例如：

```
long money = 10;
int smallMoney = money;
```

在上面的程式片段中，money 儲存了 long 型態的數值（10 被晉升為 long 型態），而您將 money 指定給 smallMoney 時，由於 smallMoney 變數的型態是 int，其容量不足以容納 long 型態數值，因而會出現以下的錯誤訊息：

```
possible loss of precision
found : long
required: int
int smallMoney = money;
```

## 宣告、使用變數

## 轉型



如果您確定要將資料型態大的值指定給資料型態小的變數，則您必須明確進行轉型（Casting）的動作，方法是使用括號()包括目標資料型態的關鍵字，例如以下的程式片段就不會有錯誤：

```
long money = 10;
int smallMoney = (int) money;
```



將 long 轉型成 int 吧！

同樣的方式也適用於 float 與 long 的轉型，例如以下的程式片段是沒有問題的：

```
double pi = 3.14159;
float piAlso = (float) 3.14; // 轉型為 float
```

當然，轉型會縮小記憶體空間，原來資料有可能會失去某些訊息，例如失去數值上的一些精確度。

Oh ! God ! 這箱子  
這麼小，看來我得  
減肥才進的去了。



圖 4-8 轉型會可能讓資料失去一些資訊



#### 自我測驗—選擇正確的變數宣告與初始化。

- a. \_\_\_\_ int myInteger=10;
- b. \_\_\_\_ long myLong;
- c. \_\_\_\_ long=10;
- d. \_\_\_\_ float = 3.14;

---

**自我測驗**－以轉型的方式修正程式的錯誤。

```
float score = 90.5;
```

**自我測驗**－下面的程式會在螢幕上顯示的數值為何？與

money 變數的值有差別嗎？討論其原因。

```
long money = 2036854775808L;  
int smallMoney = (int) money;  
System.out.println(smallMoney);
```

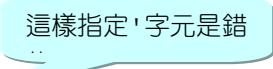


跳脫字元

**跳脫字元**

在介紹字元資料型態時，曾經介紹到可以使用 Unicode 碼來指定字元，像是用'\u0048'指定的字元為'H'，事實上，這種指定方式稱之為**跳脫字元（Escaped character）**的指定方式，所謂的跳脫字元，指的是字元本身的直接撰寫已經被程式語言用來作為語法的一部份，例如若要指定單引號，由於單引號已經被用來包括字元值，所以您不可以直接如下作單引號的指定：

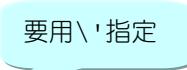
```
char single = '';
```

這樣指定'字元是錯



為了讓編譯器知道您要指定單引號字元，您可以在單引號前加上一個反斜線\，編譯器只要看到反斜線，就知道下一個字元是您要指定的字元，要跳脫不作語法上的處理，所以下面的指定才是正確的：

```
char single = '\'';
```

要用\'指定



同樣的，像雙引號"、反斜線\，若要在程式中當作數值，都必須使用同樣的方法，另外還有一些**不可見字元**，像是換行字元、跳格字元等，您無法直接在程式中輸入，也都必須使用跳脫字元，讓編譯器替您作處理。

表 4-3 列出了幾個常用的跳脫字元：

**表 4-3 跳脫字元**

字元	範例
\\	反斜線
'	單引號'
"	雙引號"
\uxxxx	以 16 進位數指定 Unicode 字元輸出
\xxx	以 8 進位數指定 Unicode 字元輸出
\b	倒退一個字元
\f	換頁
\n	換行
\r	游標移至行首
\t	跳格(一個 Tab 鍵)

---

舉個例子來說，下面的程式片段可以顯示包括雙引號的  
Hello ! World ! 文字。

```
System.out.println("\\"Hello ! World ! \\\"");
```



自我測驗—試著使用跳脫字元顯示如表 4-3 的格式與內容。



## 複習題

- 一、 在 Java 程式語言中內建八種基本資料型態為何？
- 二、 在 Java 程式語言中有多少個整數基本型態？
- 三、 當其他人閱讀您的程式碼時，您如何使變數名稱讓別人清楚明瞭？
- 四、 在 Java 程式設計語言中，變數名稱 intgrade 與 intGrade 是否相同。
- 五、 哪一個是對以下的變數宣告最好的敘述：  
`final double PI = 3.14159 ;`  
選擇：這是指派值給予整數基本型態。  
選擇：變數名稱是 Java 程式設計語言的關鍵字。  
選擇：這變數是在指定值之後，不可以再指定值給它。  
選擇：經由前面的準則，這變數命名錯誤。



---

## 第五單元

---

# 運算子

---

### 單元目標

**單元目標**



當完成本單元後，您將能學習到：

- 數學相關的算術運算子
- 決策相關的關係、條件運算子
- 處理數位邏輯的位元運算子
- 簡便寫法的指定運算子

這個單元以程式實例告訴您如何在 Java 程式運用各種運算子，以處理程式執行過程中的各種運算需求。

Sun®  
microsystems



討論一下列問題能夠幫助您了解運算子的作用：

- 資料型態對算術運算會造成什麼樣的影響？
- 運算子的優先順序為何？



## 相關資料



**相關資料**－下列參考資料能夠對本單元所討論的話題提供更詳細的說明。

- Learning the Java Language  
<http://java.sun.com/docs/books/tutorial/java/TOC.html>  
Java 程式設計人員的實用手冊，具有許多完整且能夠正常運作的範例。
- Operators  
<http://java.sun.com/docs/books/tutorial/java/nutsandbolts/operators.html>  
您可以在這邊找到更多有關運算子的資料
- Java 學習筆記, 莫峰出版社  
這是一本適合剛入門的書籍，它解釋的觀念甚至超過更進階的書籍。



## 算術運算子

程式會執行從簡單到複雜等難度不一的數學計算，數學中有加 (+)、減 (-)、乘 (\*)、除 (-) 等基本的運算，在程式語言中對應這些運算作用的符號稱之為 **算術運算子** (Arithmetic operator)。

在這個小節中將區分三個主題來討論算術運算子：

- 標準數學運算子
- 遞增與遞減運算子
- 算術運算與晉升、轉型的問題

### 標準數學運算子

二元  
運算子



標準的數學運算子，通常稱為 **二元運算子** (Binary operators)，因為在運算子的左右都必須提供 **運算元** (Operand) 方可進行運算，例如：

`System.out.println(1 + 2);`

+ 運算子

+ 表示加法運算子，兩邊提供的 1 與 2 數值即運算元。在 Java 程式語言中所使用到的數學運算子列於表 5-1 內。

表 5-1 數學運算子

用途	運算子	範例
加法	+	<code>num1 + num2</code> 若 num1 等於 10、num2 等於 2，運算結果等於 12。
減法	-	<code>num1 - num2</code> 若 num1 等於 10、num2 等於 2，則結果等於 8。
乘法	*	<code>num1 * num2</code> 若 num1 等於 10、num2 等於 2，則結果等於 20。
除法	/	<code>num1 / num2</code> 若 num1 等於 10、num2 等於 2，則結果等於 5。
模數（餘除）	%	<code>num1 % num2</code> 若 num1 等於 31、num2 等於 6，則結果等於 1。

模數運算就是取兩數相乘後的餘數；程式碼 5-1 示範了如何如何宣告變數並指定值，之後使用數學運算子對變數進行各種數學運算。

### 程式碼 5-1 ArithmeticDemo.java

```

1  public class ArithmeticDemo {
2      public static void main(String[] args) {
3          // 宣告變數與值
4          int x = 10;
5          int y = 5;
6
7          System.out.println("變數值...");  

8          System.out.println("    x = " + x);
9          System.out.println("    y = " + y);
10
11         // 加法示範
12         System.out.println("加法...");  

13         System.out.println("    x + y = " + (x + y));
14
15         // 減法示範
16         System.out.println("減法...");  

17         System.out.println("    x - y = " + (x - y));
18
19         // 乘法示範
20         System.out.println("乘法...");  

21         System.out.println("    x * y = " + (x * y));
22
23         // 除法示範
24         System.out.println("除法...");  

25         System.out.println("    x / y = " + (x / y));
26
27         // 模數運算
28         System.out.println("計算餘數...");  

29         System.out.println("    x % y = " + (x % y));
30     }
31 }
```

## 算術運算子

在使用 `println()` 進行輸出時，也可以使用 + 來串接想要輸出的內容，而您也可以看到程式中在運算式上加上了()括號，這表示您要先執行  $x$  與  $y$  運算，再將結果輸出，程式的編譯與執行結果如圖 5-1 所示：

```
C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit5>javac ArithmeticDemo.java
C:\workspace\Unit5>java ArithmeticDemo
變數值...
    x = 10
    y = 5
加法...
    x + y = 15
減法...
    x - y = 5
乘法...
    x * y = 50
除法...
    x / y = 2
計算餘數...
    x % y = 0
```

圖 5-1 程式碼 5-1 的編譯與執行結果

### 變數的運算與指定



如果要將運算的結果指定給某變數，則使用 **指定運算子** (**Assignment operator**) =，例如：

```
int x = 10;
int y = 5;
int sum = x + y;
System.out.println(sum);
```

先處理+運算子  
再處理=運算子

在第 3 行中， $x$  與  $y$  會先相加，之後再將運算結果指定給  $sum$ ，所以最後會顯示 15。

## 遞增與遞減運算子

將變數值加 1 或減 1 在程式中是很常見的指令。您可以簡單的使用加法運算子 (+) 來達成，例如：

```
int age = 10;
age = age + 1;
System.out.println(age); // 顯示 11
```

### 單元運算子



然而遞增與遞減 1 個單位是常見的動作，所以有特定的 **單元運算子** (**Unary operators**) 來表示：**++** 與 **--** 運算子。之所以稱為單元運算子，是因為運算時只需提供一個運算元，例如下頁的程式片段也是顯示 11：

```
int age = 10;
age++;
System.out.println(age); // 顯示 11
```

 使用++遞增運算子

## 遞增、遞減 運算子



++運算子稱之為 (Increment operator) 遷增運算子，而遞減運算子 (Decrement operator) 則是--，作用是將指定的變數值減 1，例如：

```
int age = 10;
age--;
System.out.println(age); // 顯示 9
```

### 前置寫法



像++與--這些運算子可以放在變數的前面（前置遞增及前置遞減）或後面（後置遞增及後置遞減），例如：

```
int age = 10;
++age;
System.out.println(age); // 顯示 11
--age;
System.out.println(age); // 顯示 10
```

### 前置、後置



事實上遞增與遞減運算子的前置與後置在使用時，兩個的作用是有差別的，使用後置遞增 (Post-increment) 與後置遞減 (Post-decrement) 時，必須一直等到這行程式碼的其它運算都執行後才作遞增與遞減的動作。所以假使此運算子在前面 (Prefix)，那這個運算子會在運算式執行之前就做遞增或遞減的動作。而假使此運算子在後面 (Post)，那這個運算子會在原來的數值被使用後才做遞增或遞減的動作。

舉個實際的例子來看，如果遞增運算子與指定運算子同時使用時，看看程式碼 5-2 會輸出什麼結果：

#### 程式碼 5-2 UnaryOperatorDemo.java

```
1 public class UnaryOperatorDemo {
2     public static void main(String[] args) {
3         int age = 10;
4         int var = age++;
5         System.out.println(age);
6         System.out.println(var);
7     }
8 }
```

### 運算子

5-7

## 算術運算子

由於 `age++` 是後置的寫法，所以 `age` 的值會先指定給 `var`，之後 `age` 才進行遞增的動作，也就是說程式中的第 4 行，其實相當於：

```
var = age;
age = age + 1;
```

所以範例 5-2 的編譯與執行結果會如下所示：

```
C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit5>javac UnaryOperatorDemo.java
C:\workspace\Unit5>java UnaryOperatorDemo
11
10
```

圖 5-2 程式碼 5-2 的編譯與執行結果

如果改變程式碼 5-2 為前置寫法，那麼結果會是如何呢？

### 程式碼 5-3 UnaryOperatorDemo2.java

```
1 public class UnaryOperatorDemo2 {
2     public static void main(String[] args) {
3         int age = 10;
4         int var = ++age; 改成前置寫法
5         System.out.println(age);
6         System.out.println(var);
7     }
8 }
```

由於 `++age` 是前置的寫法，所以 `age` 的值會先遞增，之後再指定給 `var`，也就是說程式中的第 4 行，其實相當於：

```
age = age + 1;
var = age;
```

所以範例 5-2 的編譯與執行結果會如下所示：

```
C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit5>javac UnaryOperatorDemo2.java
C:\workspace\Unit5>java UnaryOperatorDemo2
11
11
```

圖 5-3 程式碼 5-3 的編譯與執行結果

## 運算子的優先順序

或許您已經注意到程式在處理運算時，對於運算子的處理是有其先後順序的，在複雜數學表示式中，同一行中有多個運算子，電腦應該如何選擇最先使用的運算子呢？

### 運算子優先順序



#### 運算子優先順序的規則

為了讓數學運算一致，Java 使用底下的標準數學運算規則來做為運算子優先順序的規則：

1. 有一對圓括弧的運算子
2. 遞增及遞減運算子
3. 乘法與除法運算子是由左而右做計算
4. 加法與減法運算子是由左而右做計算

假如一樣優先權的運算子在一個陳述句中連續地出現，則它們都是由左而右做計算。

#### 需要運算子優先順序規則的範例

底下的範例驗證了運算子優先順序規則存在的必要性：

```
c = 25 - 5 * 4 / 2 - 10 + 4;
```

### 使用括號



不管是從數學表示式或者是從任何所提供的註解，程式所表達的意思都不很明確，如果根據優先順序的規則（由圓括弧所指），其計算方式應該如下：

```
c = 25 - ((5 * 4) / 2) - 10 + 4;
```

您的數學運算式會以優先順序的規則做計算；然而，使用圓括弧來說明您所想要表達的意思是個不錯的方式，例如：

```
c = (((25 - 5) * 4) / (2 - 10)) + 4;
c = ((20 * 4) / (2 - 10)) + 4;
c = (80 / (2 - 10)) + 4;
c = (80 / -8) + 4;
c = -10 + 4;
c = -6;
```

使用括號吧！這樣就不會搞錯了！



## 算術運算與晉升、轉型的問題

在進行算術運算時，需注意資料型態的問題，先來看看下面這個程式片段會顯示什麼結果？

```
System.out.println(10 / 3);
```

結果顯示 3 !  
不會吧！



這個程式片段會在螢幕上顯示 3，Java 程式運行時出錯了嗎？不是的！在解釋之前，再來看一個程式片段：

```
System.out.println(10.0 / 3);
```

3.3333... ? !



這次將被除數改以浮點數的方式撰寫，進行除法時，程式就顯示正確的值了，或許聰明的您已經想到，當您在程式中直接寫下 10.0 時，編譯器會自動配給它 double 型態的記憶體空間，跟這個有關嗎？

答案是正確的！在進行算術運算時，必須注意到資料型態的 **晉升 (Promotion)** 與 **轉型 (Casting)** 問題。

### 算術運算時資料型態的晉升

運算時的  
型態晉升



當您直接在程式中寫下  $10 / 3$  這樣的運算時，由於編譯器會配給 int 型態的空間給數值 10 與 3，當兩者進行除法運算時，最後的結果仍是 int 型態的數值，所以浮點數的資料無法容納，因而結果會只顯示 3。

在第四單元中曾經談到過型態晉升的議題，晉升是編譯器將較小的資料型態升級為較大的資料型態，使得它可以支援特定範圍的數值或運算，當運算式中有資料型態較大的數值時，所有其它的數值都會自動提升為符合該資料型態，接著再進行運算，如此才足以容納運算的結果。

所以當您進行  $10.0 / 3$  的運算時，int 型態的數值 3 也會轉換為浮點數 3.0 再進行運算，結果才會顯示浮點數的運算結果。

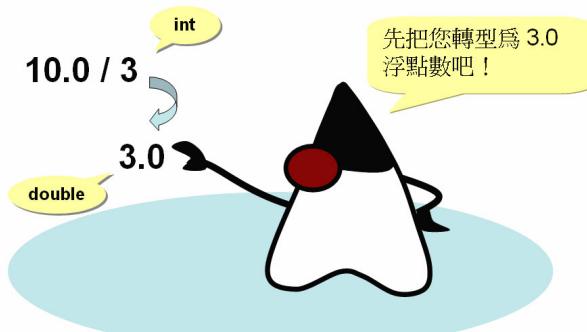


圖 5-4 算術運算時的自動型態晉升

型態的晉升並不只發生於除法，只要運算式中有某個資料型態較大，所有的資料都會晉升為該型態。

#### 算術運算時資料型態的轉型

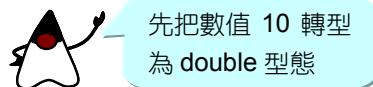
##### 運算時的 轉型



您宣告了兩個 int 型態的變數並進行運算，並在某個時間點將兩個進行相除，根據之前的討論，兩個 int 型態的數值相除，結果只會顯示 int 型態的運算結果，然而您想得到的是浮點數的運算結果時該如何作？<sup>®</sup>

您可以將某個數值進行轉型，「手動」將之晉升為較大的資料型態，方法是使用括號()並指定轉型的目標型態關鍵字，例如：

```
System.out.println((double) 10 / 3);
```



將運算式中的某個數值提升為較大的資料型態之後，進行運算時，所有的資料就會以該資料型態的大小進行運算，以上例來說，運算的結果就會 3.3333...，而不只是 3。

## 算術運算子

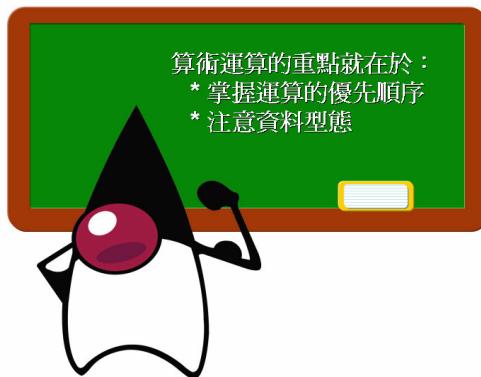


圖 5-5 算術運算的注意事項



**自我測驗**—底下何者正確地指出下列運算式的運算子優先權？

$x = 1 + 2 * 5 - 8 + 6 / 7 * 4$

選擇 :  $(1 + 2) * (5 - 8) + (6 / (7 * 4))$

選擇 :  $1 + (2 * 5) - 8 + ((6 / 7) * 4)$

選擇 :  $(1 + 2) * (5 - ((8 + 6) / (7 * 4)))$

選擇 :  $1 + (2 * (5 - 8)) + ((6 / 7) * 4)$

microsystems



**自我測驗**—查看底下的程式碼，並選擇執行後的正確輸出。

`int a = 5 ;`

`int b = ++a ;`

`System.out.println("a = " + a + " and b " + b) ;`

選擇 :  $a = 5$  and  $b = 5$

選擇 :  $a = 5$  and  $b = 6$

選擇 :  $a = 6$  and  $b = 6$

選擇 :  $a = 6$  and  $b = 5$

## 關係運算子、條件運算子

在程式中經常要對數值進行比較，查看兩數值的大小關係，例如「A 是否比 B 大？」，或是「A 是否等於 B？」，[關係運算子（Relational operator）](#)是用來比較兩個數值，並判定兩者之間的大小關係。

另一方面，對於複合式的條件在程式設計中也是經常遇到的，例如「是否 A 比 B 大並且 A 比 C 大？」，[條件運算子（Conditional operator）](#)讓您對兩個條件進行「並且」、「或」等判斷。

關係運算子與條件運算子的運算結果為 boolean 型態，也就是只有兩個結果：[true](#)、[false](#)。這一個小節將分別介紹這兩個運算子。

### 關係運算子

關係運算子是用來比較兩個數值，並判定兩者之間的大小關係，表 5-2 中列出關係運算子能夠測試的各種狀態。

**表 5-2 關係運算子**

狀況	運算子	範例
相等	<code>==</code>	<code>int i = 1 ;(i == 1 )</code>
不等	<code>!=</code>	<code>int i = 2 ;(i != 1 )</code>
小於	<code>&lt;</code>	<code>int i = 0 ;(i &lt; 1 )</code>
小於等於	<code>&lt;=</code>	<code>int i = 2 ;(i &lt;= 1 )</code>
大於	<code>&gt;</code>	<code>Int i = 1 ;(i &gt; 1 )</code>
大於等於	<code>&gt;=</code>	<code>int i = 1 ;(i &gt;= 1 )</code>

程式碼 5-4 示範了如何使用關係運算子：

#### 程式碼 5-4 RelationalDemo.java

```
1 public class RelationalDemo {
2     public static void main(String[] args) {
```

運算子

5-13

## 關係運算子、條件運算子

```

3      // 宣告變數
4      int i = 37;
5      int j = 42;
6      int k = 42;
7      System.out.println("變數值... ");
8      System.out.println(" i = " + i);
9      System.out.println(" j = " + j);
10     System.out.println(" k = " + k);
11
12     // 大於
13     System.out.println("大於... ");
14     System.out.println(" i > j = " + (i > j));
15     System.out.println(" j > i = " + (j > i));
16     System.out.println(" k > j = " + (k > j));
17
18     // 大於或等於
19     System.out.println("大於或等於... ");
20     System.out.println(" i >= j = " + (i >= j));
21     System.out.println(" j >= i = " + (j >= i));
22     System.out.println(" k >= j = " + (k >= j));
23
24     // 小於
25     System.out.println("小於... ");
26     System.out.println(" i < j = " + (i < j));
27     System.out.println(" j < i = " + (j < i));
28     System.out.println(" k < j = " + (k < j));
29
30     // 小於或等於
31     System.out.println("小於或等於... ");
32     System.out.println(" i <= j = " + (i <= j));
33     System.out.println(" j <= i = " + (j <= i));
34     System.out.println(" k <= j = " + (k <= j));
35
36     // 等於
37     System.out.println("等於... ");
38     System.out.println(" i == j = " + (i == j));
39     System.out.println(" k == j = " + (k == j));
40

```

小心！是兩個等號！



## 關係運算子、條件運算子

```

41      // 不等於
42      System.out.println("不等於... ");
43      System.out.println(" i != j = " + (i != j));
44      System.out.println(" k != j = " + (k != j));
45  }
46 }

```

所有關係運算子的運算結果都是一個 boolean 型態數值，不是 true 就是 false，程式的編譯與執行結果如下所示：

## 關係運算



```

C:\Windows\System32\cmd.exe
C:\workspace\JUnit5>javac RelationalDemo.java
C:\workspace\JUnit5>java RelationalDemo
變數值...
    i = 37
    j = 42
    k = 42
大於...
    i > j = false
    j > i = true
    k > j = false
大於或等於...
    i >= j = false
    j >= i = true
    k >= j = true
小於...
    i < j = true
    j < i = false
    k < j = false
小於或等於...
    i <= j = true
    j <= i = false
    k <= j = true
等於...
    i == j = false
    k == j = true
不等於...
    i != j = true
    k != j = false

```

圖 5-6 程式碼 5-4 的編譯與執行結果

必須特別注意等於的關係運算子，它是兩個連續的等號==而不是一個，新手很容易只寫一個等號就以為它是等於的關係運算子，[一個等號=是指定運算子的寫法](#)。

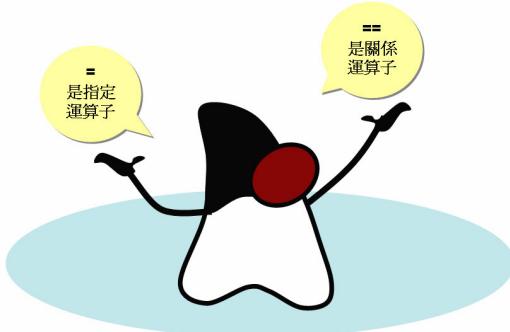


圖 5-7 注意=與==的差別

## 條件運算子

您或許需要有為多種狀況做決定的能力。在這種情況下，您可以使用條件運算子來評估複雜的狀況，表 5-3 列出 Java 程式語言中的條件運算子。

**表 5-3 條件運算子**

運算	運算子	範例
如果一個狀況與另一 狀況同時發生	&&	int i = 2 ; int j = 8 ; (( i < 1 ) && ( j > 6 ))
如果一個狀況或另一 狀況其中一個發生		int i = 2 ; int j = 8 ; (( i < 1 )    ( j > 10 ))
非	!	int i = 2 ; int j = 8 ; ( ! ( i < 3 ))

程式碼 5-5 示範了如何使用條件運算子：

**程式碼 5-5 ConditionalDemo.java**

```

1 public class ConditionalDemo {
2     public static void main(String[] args) {
3         int i = 2;
4         int j = 3;
5         int k = 4;
6
7         System.out.println("變數值... ");
8         System.out.println(" i = " + i);
9         System.out.println(" j = " + j);
10        System.out.println(" k = " + k);
11
12        // && 運算
13        System.out.println("且... ");
14        System.out.println(" i < j 且 i < k "
15                           + ((i < j) && (i < k)) );
16
17        // || 運算
18        System.out.println("或... ");
19        System.out.println(" i < j 或 k > j "
20                           + ((i < j) || (k > j)) );

```



小心！兩個&喔！

```

21
22      // ! 運算
23      System.out.println("非... ");
24      System.out.println(" i < j 的非 "
25                  + !(i < j));
26  }
27 }
28

```

要注意`&&`與`||`，是連續的兩個`&`與`|`；關係運算子的運算結果是 boolean 型態數值，也就是 true 或 false，程式碼 5-5 的編譯與執行結果如下：

### 條件運算



```

C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit5>javac ConditionalDemo.java
C:\workspace\Unit5>java ConditionalDemo
變數值...
    i = 2
    j = 3
    k = 4
且...
    i < j 且 i < k true
或...
    i < j 或 k > j true
非...
    i < j 的非 false

```

圖 5-8 程式碼 5-5 的編譯與執行結果



**自我測驗**－下段短文中，表達哪些關係運算子和條件運算子？

如果交通號誌是紅燈時，那我會停車。然而，如果變成黃燈的時間不到 2 秒，則我會加速。如果變成黃燈的時間超過 2 秒，我會減速。如果交通號誌是綠燈時，我會以目前的速度通過路口。

## 位元運算子、位移運算子

電腦中的數值其實全部都是以 0 與 1 的數值儲存，[位元運算子（Bitwise operator）](#) 與 [位移運算子（Shift operator）](#) 可以讓您直接對數值的位元進行運算。

### 位元運算子

在 Java 中提供 &、|、!、~ 四個位元運算子，這些運算子又稱 [邏輯運算子（Logical operator）](#)，因為它們分別提供數位邏輯處理中的 And、Or、Xor 與補數運算，表 5-4 提供位元運算子的介紹。

**表 5-4 位元運算子**

運算子	操作	範例
&	$op1 \& op2$	將 $op1$ 與 $op2$ 的數值逐位元進行 And 運算
	$op1   op2$	將 $op1$ 與 $op2$ 的數值逐位元進行 Or 運算
^	$op1 ^ op2$	將 $op1$ 與 $op2$ 的數值逐位元進行 Xor 運算
~	$\sim op1$	將 $op1$ 數值取其補數

所謂的逐位元運算，指的是對數值真正於記憶體中的二進位表示法，一個位元一個位元進行運算，這在以下針對每個運算子分別說明。

#### & 運算子

對於 & 運算，兩個位元都必須為 1，其結果才是 1，否則運算結果為 0，表 5-5 列出 & 運算子對單一位元的運算方式。

**表 5-5 & 運算**

op1	op2	結果
0	0	0
0	1	0
1	0	0
1	1	1

舉個例子來說，當 13 與 12 兩個數值進行 & 運算時，由於 13

## 位元運算子、位移運算子

的二進位表示為 1101，12 的二進位表示為 1100，逐位元運算的結果會是 1100，也就是 12，運算的方式與結果如圖 5-9 所示。

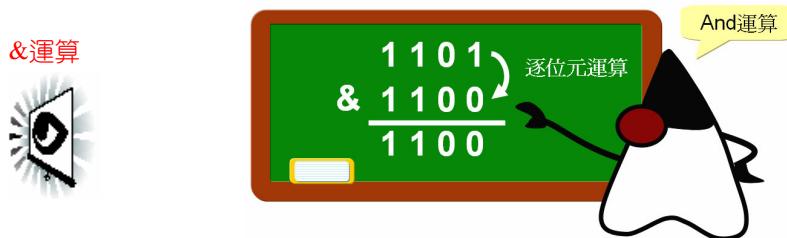


圖 5-9 逐位元進行 &amp; 運算

## | 運算子

對於 | 運算，只要有一個位元都必須為 1，其結果就是 1，只有在兩個位元都為 0 的情況下，結果才會是 0，表 5-6 列出 | 運算子對單一位元的運算方式。

表 5-6 | 運算

op1	op2	結果
0	0	0
0	1	1
1	0	1
1	1	1

舉個例子來說，當 13 與 12 兩個數值進行 | 運算時，逐位元運算的結果會是 1101，也就是 13，運算的方式與結果如圖 5-10 所示。

## | 運算

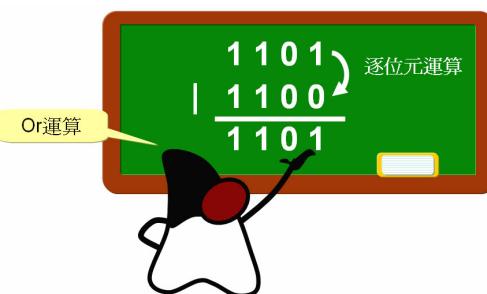


圖 5-10 逐位元進行 | 運算

## ^ 運算子

## 運算子

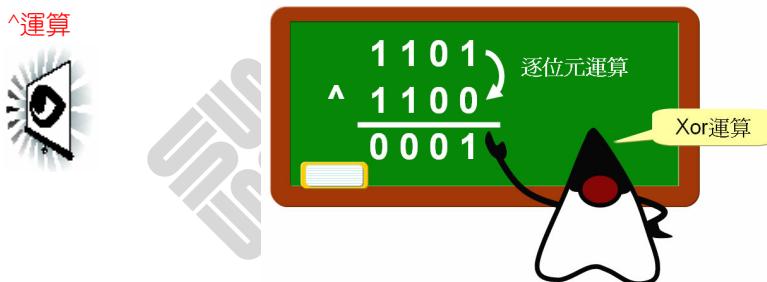
## 位元運算子、位移運算子

對於  $\wedge$  運算，進行的是 Xor (Exclusive or) 運算，也就是兩個位元都為 0 或都為 1 時結果為 0，兩個位元非同時為 0 或同時為 1 時結果為 1，如表 5-7 所示。

表 5-7  $\wedge$  運算

op1	op2	結果
0	0	0
0	1	1
1	0	1
1	1	0

舉個例子來說，當 13 與 12 兩個數值進行  $\wedge$  運算時，逐位元運算的結果會是 0001，也就是 1，運算的方式與結果如圖 5-11 所示。

圖 5-11 逐位元進行  $\wedge$  運算 $\sim$  運算 $\sim$  運算

$\sim$  運算子為補數運算子 (Complement operator)，簡單的說，就是將數值的每一個位元都反相，也就是 1 改為 0, 0 改為 1，不過在運算的時候必須注意，在 Java 中的正整數其二進位為 0 開始，例如 int 型態 13 的位元值應該是 00000000 00000000 00000000 00001101 (32 位元長度)，經補數運算後變成 11111111 11111111 11111111 11110010，以 1 開頭在 Java 中為負數，運算後的值為 -14。



**重點提示**一位元值全為 1 在電腦中表示 -1，再減 1 就是 -2，  
11111111 11111111 11111111 11110010 為減去 13 後的值，  
所以就是 -14，而另一個快速得知所表示負數值的方式，為  
取其補數再加 1 即可，上例取補數加 1 後，10 進位值表示  
為 14，所以這個數就是 -14。

程式碼 5-6 示範了如何使用位元運算子：

### 程式碼 5-6 BitwiseDemo.java

```

1 public class BitwiseDemo {
2     public static void main(String[] args) {
3         int i = 13;
4         int j = 12;           
5
6         System.out.println("變數值...");
7         System.out.println(" i = " + i);
8         System.out.println(" j = " + j);
9
10        System.out.println("位元運算...");
11        System.out.println(" i & j = " + (i & j));
12        System.out.println(" i | j = " + (i | j));
13        System.out.println(" i ^ j = " + (i ^ j));
14        System.out.println(" ~i = " + (~i));
15    }
16 }
```

編譯與執行的結果如下所示，您可以與之前的說明相印證：

```

C:\Windows\system32\cmd.exe
C:\workspace\Unit5>javac BitwiseDemo.java
C:\workspace\Unit5>java BitwiseDemo
變數值...
i = 13
j = 12
位元運算...
i & j = 12
i | j = 13
i ^ j = 1
~i = -14

```

圖 5-12 程式碼 5-6 的編譯與執行結果

## 位移運算子

### 運算子

## 位元運算子、位移運算子

位移運算子有左移運算與右移運算，可以將指定數值的位元進行左移或右移，並可指定位移單元，其使用方式與作用如表 5-8 所示：

表 5-8 位移運算子

運算子	範例	作用
<<	op1 << op2	將 op1 的位元左移 op2 個單位，右邊補上 0
>>	op1 >> op2	將 op1 的位元右移 op2 個單位，左邊補上原來最左邊的位元值
>>>	op1 >>> op2	將 op1 的位元右移 op2 個單位，左邊補上 0

舉個例子來說，將數值 1 進行左移，每次移動一個單位的話：

## 位移運算

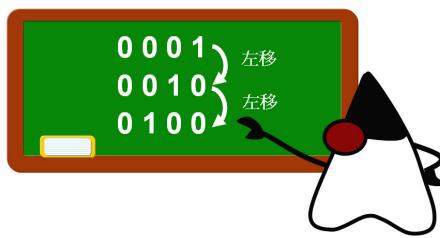


圖 5-13 左移運算

圖 5-13 中的三個數值由上而下表示為 10 進位的話，分別為 2、4、8，可由程式碼 5-7 進行驗證。

## 程式碼 5-7 ShiftDemo.java

```

1  public class ShiftDemo {
2      public static void main(String[] args) {
3          int i = 1;
4
5          System.out.println("變數值...");
6          System.out.println(" i = " + i);
7
8          System.out.println("位移運算...");
9          System.out.println(" i << 1 = " + (i << 1));
10         System.out.println(" i << 2 = " + (i << 2));
11         System.out.println(" i << 3 = " + (i << 3));
12     }

```

程式碼 5-7 的編譯與執行結果如下所示：

The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The command 'javac ShiftDemo.java' is run, followed by 'java ShiftDemo'. The output shows the value of variable 'i' being shifted left by 1, 2, and 3 positions.

```
C:\workspace\Unit5>javac ShiftDemo.java
C:\workspace\Unit5>java ShiftDemo
變數值...
    i = 1
位移運算...
    i << 1 = 2
    i << 2 = 4
    i << 3 = 8
```

圖 5-14 程式碼 5-7 的編譯與執行結果



自我測驗—撰寫一程式，執行 253 與 134 的&、|、^運算，並討論執行結果是如何得來的。



自我測驗—撰寫一程式，可以列出&、|、^的真值表（Truth table）。



## 指定運算子

## 指定運算子

當您要將某個數值或變數指定給另一個變數時，您使用的**指定運算子 (Assignment operator)**是`=`，事實上，Java 程式語言還提供了其它方便的指定運算子，讓您可以同時進行運算與指定的動作。

舉個例子來說，假如您有個程式片段是這麼撰寫的：

```
int i = 10;
i = i + 5;
```

**指定運算**

那麼您可以使用`+=`這個指定運算子將之改寫為：

```
int i = 10;
i += 5;
```



嗯...還有這種  
簡便寫法！

表 5-9 列出了 Java 中所提供的指定運算子與實際的作用：

**表 5-9 指定運算子**

運算子	範例	作用
<code>+=</code>	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
<code>-=</code>	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>
<code>*=</code>	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
<code>/=</code>	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
<code>%=</code>	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>
<code>&amp;=</code>	<code>op1 &amp;= op2</code>	<code>op1 = op1 &amp; op2</code>
<code> =</code>	<code>op1  = op2</code>	<code>op1 = op1   op2</code>
<code>^=</code>	<code>op1 ^= op2</code>	<code>op1 = op1 ^ op2</code>
<code>&lt;&lt;=</code>	<code>op1 &lt;&lt;= op2</code>	<code>op1 = op1 &lt;&lt; op2</code>
<code>&gt;&gt;=</code>	<code>op1 &gt;&gt;= op2</code>	<code>op1 = op1 &gt;&gt; op2</code>
<code>&gt;&gt;&gt;=</code>	<code>op1 &gt;&gt;&gt;= op2</code>	<code>op1 = op1 &gt;&gt;&gt; op2</code>



自我測驗—`i = i + 1` 可以寫成哪兩種簡短的形式。

## 複習題

- 一、 在下面一段短文中，會用到哪些關係運算子及狀態運算子？

如果現在是 8 : 00，則是工作及喝咖啡的時間。如果時間超過 12 : 00，則是吃午餐及喝茶的時間。如果現在時間是 5 : 00 或是更晚，就是該回家的時間了。

選項： == , > , >=

選項： == , & & , <

選項： == , & & , >

選項： == , | | , >

- 二、 下面的程式片段最後會顯示什麼？為什麼？

```
System.out.println(30.0 * 5);
```

- 三、 下面的程式片段最後會顯示什麼？為什麼？

```
System.out.println(32 / 5);
```

- 四、 某數與 1 進行&運算若結果為 1，表示該數為奇數，若結果為 0，表示該數為偶數，請問原因何在？





---

## 第六單元

---

# 流程控制

---

### 單元目標

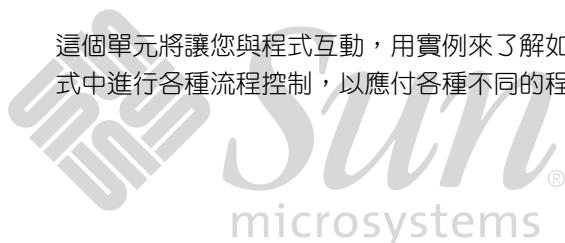
#### 單元目標

當完成本單元後，您將能學習到：

- 取得使用者的輸入
- 建立 if 和 if/else 陳述
- 使用 switch 陳述



這個單元將讓您與程式互動，用實例來了解如何在 Java 程式中進行各種流程控制，以應付各種不同的程式設計需求。





### 討論—下列問題能夠幫助您了解流程控制的作用：

- 當您為一個可能有多種不同解法的事件做決定時，您最後如何在全部的解法中選擇其中一個？
- 舉個例子，當您打算購買一件商品的時候，在您的腦袋中是如何盤算的？



## 相關資料



**相關資料**－下列參考資料能夠對本單元所討論的話題提供更詳細的說明。

- Learning the Java Language  
<http://java.sun.com/docs/books/tutorial/java/TOC.html>  
Java 程式設計人員的實用手冊，具有許多完整且能夠正常運作的範例。
- Java 學習筆記，碁峰出版社  
這是一本適合剛入門的書籍，它解釋的觀念甚至超過更進階的書籍。



取得使用者輸入

## 取得使用者輸入

為了讓您更了解各種流程控制語法的作用，並實際對程式輸入資料，讓程式根據各種輸入資料作出不同的反應與輸出結果，在這個小節中，要先告訴您如何從文字模式中取得使用者的輸入。

### 使用標準輸入取得資料

#### 標準輸入



要取得使用者於文字模式下的輸入，最基本（底層）的方式就是使用**標準輸入 (Standard input)** `System.in` 的 `read()` 方法，然而 `read()` 方法一次只從標準輸入取得一個位元組的資料，且取得的資料是以 `int` 整數值傳回，程式設計人員必須再作轉換處理，才可以將之轉換為字串或是整數值，而且必須處理 `java.io.IOException` 例外，對於初學者來說，使用 `System.in` 的 `read()` 並不方便。



對新手來說好像太麻煩了

### 使用 `java.util.Scanner` 取得輸入

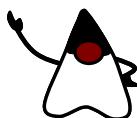
如果您所使用的是 JDK 5.0，則取得使用者輸入最簡便的方式，就是使用 5.0 中新增的類別：`java.util.Scanner`。雖然現在還沒有正式介紹如何在 Java 程式中使用已定義好的**類別 (Class)** 並新增**物件 (Object)**，不過使用 `Scanner` 的方式是固定的，初學者暫且無需理會語法的細節。

#### 使用 `Scanner`



`java.util.Scanner` 是 Java SE 中的標準類別，初學者可以先將類別看作是一個產品製造書，如果您要使用某個類別，就根據該類別建立一個**實例 (Instance)**，就好比您根據產品製造書製造一個實際的產品，要建立 `java.util.Scanner` 的實例，必須使用以下的語法：

```
java.util.Scanner scanner =
    new java.util.Scanner(System.in);
```



建立一個  
Scanner 實例

## 取得使用者輸入

在建立 Scanner 實例時，必須給它一個 System.in 作為參數，這是因為實際取得輸入的工作，仍是 System.in 在進行的，至於 System.in 實際如何取得輸入，而 Scanner 如何進行轉換等細節，您不用理會，如果您要取得一個整數，只要告訴 Scanner，Scanner 則命令 System.in 取得輸入，並將輸入作轉換，再將結果傳給您。

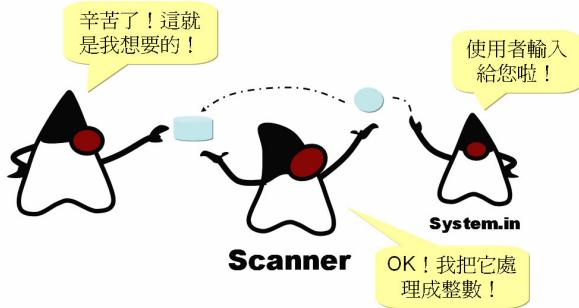


圖 6-1 建立 Scanner 物件協助您取得輸入



**註—**詳細了解 Scanner 物件的建立與運作原理並不是初學者現在學習的重點，現階段的學習重點在於學會如何取得使用者的輸入。

在建立 Scanner 實例之後，您可以使用各種 nextXXX()方法（Method）取得各種型態的資料，例如 nextInt()可將使用者輸入轉換為 int 型態，nextShort()可取得 short 型態的輸入，nextDouble()可取得 double 型態的輸入。



使用對應的方法  
取得對應的型態



**重點提示—**關於某個類別上可使用的方法說明，可查詢線上 API 文件，網址是：

<http://java.sun.com/j2se/1.5.0/docs/api/>

例如 Scanner 類別的相關說明，可以在以下的網址找到：

<http://java.sun.com/j2se/1.5.0/docs/api/java/util/Scanner.html>

程式碼 6.1 示範了如何取得 int 整數、double 浮點數與 boolean 型態的輸入，並示範如何使用變數來儲存輸入，並重新顯示這些輸入：

取得使用者輸入

**程式碼 6-1 ScannerDemo.java**

```

1  public class ScannerDemo {
2      public static void main(String[] args) {
3          java.util.Scanner scanner =
4              new java.util.Scanner(System.in);
5
6          System.out.print("輸入整數：");
7          int input1 = scanner.nextInt();
8
9          System.out.print("輸入浮點數：");
10         double input2 = scanner.nextDouble();
11
12         System.out.print("輸入布林數：");
13         boolean input3 = scanner.nextBoolean();
14
15         System.out.println("整數輸入：" + input1);
16         System.out.println("浮點數輸入：" + input2);
17         System.out.println("布林數輸入：" + input3);
18     }
19 }
```

如果要儲存 int 型態的輸入，則要使用 int 型態的變數來儲存取得的輸入，要儲存 double 型態的輸入，則要使用 double 型態的變數來儲存取得的輸入，程式碼 6-1 的執行結果如下所示：

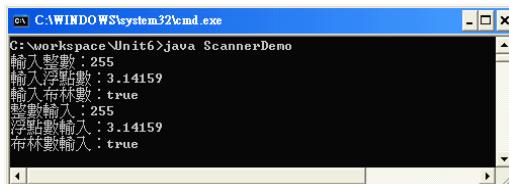


圖 6-2 程式碼 6-1 的執行結果

如果 Scanner 無法將使用者的輸入轉換為指定的資料型態，則程式就會發生 `java.util.InputMismatchException` 錯誤，現階段您還沒學到如何處理這個錯誤，所以程式處理這個錯誤的方式就是直接終止程式。

注意型態要符合喔！





**自我測驗**－嘗試撰寫一個程式取得使用者的兩個 int 整數輸入，並顯示兩數相加的結果。



**自我測驗**－嘗試寫一個程式取得使用者的名稱，並重新顯示在螢幕上。



建立 if 和 if/else 陳述

## 建立 if 和 if/else 陳述

在生活中經常會有這麼的對話：「如果我中了樂透，就可以去環遊世界！」、「如果我...就...，要不然就...」。

撰寫程式的目的之一，就是在應付生活中的一些條件假設與動作執行，例如「如果使用者輸入 0，就停止執行程式。」、「如果使用者密碼輸入錯誤，請顯示重新輸入畫面。」對於這樣的需求，在 Java 程式語言中，您可以使用 if、if/else 陳述來為程式進行各種條件判斷與流程控制。



### if 結構

基本的 if 結構很簡單，您給它一個條件運算式，根據運算結果為 true 或 false 來決定是否執行下一句陳述句，最簡單且最常見的格式為：

#### if 結構



```
if ( boolean_expression )
    statement;
// 繼續往下執行程式
```

直接以程式實例來說明，程式碼 6-2 可以判斷使用者的輸入是奇數還是偶數。

#### 程式碼 6-2 OddTest.java

```
1 public class OddTest {
2     public static void main(String[] args) {
3         java.util.Scanner scanner =
```

## 建立 if 和 if/else 陳述

```

4         new java.util.Scanner(System.in);
5
6         System.out.print("輸入整數：");
7         int input = scanner.nextInt();
8
9         if(input % 2 == 0) // 如果餘數為 0
10            System.out.println(input + " 是偶數");
11
12        if(input % 2 != 0) // 如果餘數不為 0
13            System.out.println(input + " 是奇數");
14    }
15 }
```

程式碼 6-2 將使用者的輸入除以 2，看看餘數是不是為 0 來判斷使用者的輸入為奇數還是偶數，只有在 if 括號中的條件式結果為 true 時，才會執行緊跟著 if 後的陳述句，一個執行的結果如下所示：

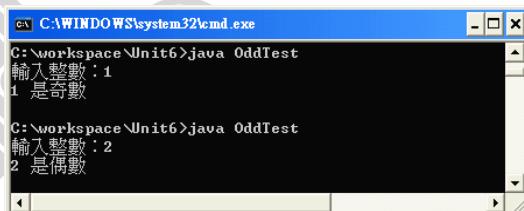


圖 6-4 程式碼 6-2 的執行範例

如果在 if 判斷為 true 時，有數個想要執行的陳述句，也就是執行所謂的**複合陳述句（Compound statement）**，那麼要使用花括號來定義出執行區塊（Block），使用的格式如下：

```

if ( boolean_expression ) {
    code_block;
    ...
} // if 結尾
// 繼續往下執行程式
```

區塊之中可以撰  
寫多個陳述句

### 複合陳述



## 建立 if 和 if/else 陳述

當 if 括號中的運算式結果為 true 時，會執行使用花括號所定義出來的區塊中所有的陳述句，一個使用範例如下：

```
if (input == 36) {
    System.out.println("恭喜您猜對了！");
    System.out.println("數值為 " + 36);
}
```

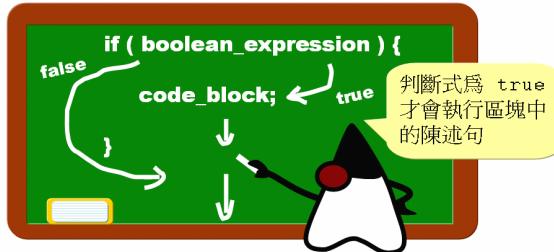


圖 6-5 根據判斷式來決定是否執行陳述句

## 巢狀 if 結構

有些時候您可能需要把 if 陳述當成另一個 if 陳述中的一部份，即形成所謂的**巢狀 if 結構**，例如：

### 巢狀 if 結構



```
if ( boolean_expression1 ) {
    code_block1;
    ...
    if ( boolean_expression2 ) {
        code_block2;
    } // if 結尾
    ...
} // if 結尾
// 繼續往下執行程式
```



`boolean_expression1` 為 true，才有可能執行到這邊的 if 判斷式

當 `boolean_expression1` 判斷為 true 時，才會執行 `code_block1` 定義的內容，此時執行到第二個 if 判斷式時，如果 `boolean_expression2` 為 true，才會又執行 `code_block2`，如果 `boolean_expression1` 一開始就判斷為 false，則 `code_block1` 中的內容就不會被執行，更不會執行到當中的第二個 if 判斷式。

## if/else 結構

## 建立 if 和 if/else 陳述

通常您想要在判斷式為 true 時，執行程式的一個區塊，而在運算式為 false 時，執行程式的另一個區塊。此時您可用一個 if 敘述來包含判斷式成立必需執行的程式碼，以及一個 else 敘述，包含只有在判斷式不成立時才能執行的程式碼。

if/else 的使用方式為：

## if/else 結構



```
if ( boolean_expression ) {
    code_block_1;
} // if 結構結尾
else {
    code_block_2;
} // else 結構結尾
// 程式由此繼續
```



依判斷式選擇區塊執行



**重點提示**—如果區塊中只有一行陳述句，則可以省略花括號，不過也有開發人員建議，即使只有一行陳述句，最好也用花括號標示區塊，這樣可以讓程式容易閱讀。

在程式碼 6-2 中，您使用了兩個 if 判斷式來判別使用者輸入的數值是否可以被 2 整除，程式執行時，無論如何都會執行那兩個判斷式，您應該改用 if/else 結構，以增進程式執行的效率，如程式碼 6-3 所示：

## 程式碼 6-3 OddTest2.java

```
1 public class OddTest2 {
2     public static void main(String[] args) {
3         java.util.Scanner scanner =
4             new java.util.Scanner(System.in);
5
6         System.out.print("輸入整數：");
7         int input = scanner.nextInt();
8
9         if(input % 2 == 0) {
10             System.out.println(input + " 是偶數");
11         }
12         else {
13             System.out.println(input + " 是奇數");
14         }
15     }
}
```

## 流程控制

6-11

建立 if 和 if/else 陳述

16 }

如果 `input%2` 的餘數為 0，則會執行第 10 行，否則就執行第 13 行，兩個區塊只會擇一執行。

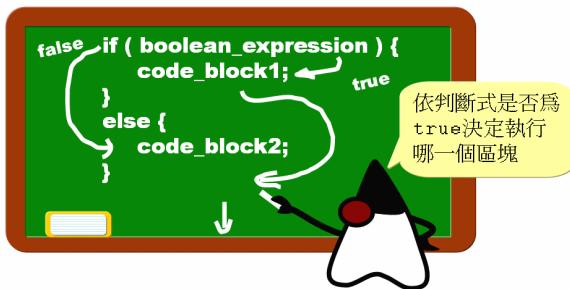


圖 6-6 if/else 結構依判斷式決定要執行的區塊

程式碼 6-3 的執行結果如下所示：

```

C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit6>java OddTest2
輸入整數：10
10 是偶數

C:\workspace\Unit6>java OddTest2
輸入整數：15
15 是奇數

```

圖 6-7 程式碼 6-3 的執行結果

## 鏈狀 if/else 結構

您可在 if/else 結構中的 else 區塊再撰寫 if/else，例如：

```

if ( boolean_expression_1 ) {
    code_block_1;
} // if 結構結尾
else {
    if ( boolean_expression_2 ) {
        code_block_2;
    } // if 結構結尾
    else {
        code_block_3;
    } // else 結構結尾
}

```

多個 if/else 的組合，可進行更複雜的流程控制

由於 else 後直接撰寫 if 陳述句，所以可以不使用括號，因而常常會撰寫為以下的形式，看起來比較容易閱讀：

## 鏈狀 if/else

結構



```

if ( boolean_expression_1 ) {
    code_block_1;
}
else if ( boolean_expression_2 ) {
    code_block_2;
}
else {
    code_block_3;
}

```



嗯！這麼寫好  
像清楚多了！

舉個實際的例子來說，您可以使用上面的結構，撰寫一個成績分級程式，如程式碼 6-4 所示範的：

## 程式碼 6-4 ScoreLevel.java

```

1 public class ScoreLevel {
2     public static void main(String[] args) {
3         java.util.Scanner scanner =
4             new java.util.Scanner(System.in);
5
6         System.out.print("輸入分數：");
7         int score = scanner.nextInt();
8
9         if(score >= 90)
10            System.out.println("得 A");
11        else if(score >= 80 && score < 90)
12            System.out.println("得 B");
13        else if(score >= 70 && score < 80)
14            System.out.println("得 C");
15        else if(score >= 60 && score < 70)
16            System.out.println("得 D");
17        else
18            System.out.println("得 E(不及格)");
19    }
20 }

```

程式執行時，會一個一個比對判斷式中的條件是否符合，如果遇到判斷式為 true 時，就執行對應的區塊，之後的判斷式

流程控制

6-13

建立 if 和 if/else 陳述

就不再進行比對，執行的結果如下所示：

```
C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit6>java ScoreLevel
輸入分數：88
得B

C:\workspace\Unit6>java ScoreLevel
輸入分數：66
得D
```

圖 6-8 程式碼 6-4 的執行結果

在撰寫 if/else 結構時，需注意到如果不使用花括號定義出區塊的話，每一個 if 是與最接近的 else 配對，例如初學者可能會這麼撰寫：

```
if ( boolean_expression_1 )
    if ( boolean_expression_2 )
        code_block_1;
else
    code_block_2;
```

初學者可能會以為 else 是與 boolean\_expression\_1 的 if 判斷式配對，但事實上，else 是與 boolean\_expression\_2 的 if 判斷式配對，為了結構清楚避免錯誤，您還是多使用花括號定義出區塊比較保險，例如這麼撰寫就不會有疑慮：

```
if ( boolean_expression_1 ) {
    if ( boolean_expression_2 )
        code_block_1;
else
    code_block_2;
```

寫程式還是要清楚易懂，別偷懶喔！

if/else 配對



## 關於三元運算子?:

在 Java 語言中提供了一個與 if/else 類似功能的三元運算子?:，它的使用方式如下所示：

```
boolean_expression ? result_1 : result_2;
```

之所以稱之為三元運算子，是因為您必須提供三個運算元作為它運算時所需的數值，如果 boolean\_expression 的運算

## 建立 if 和 if/else 陳述

結果為 true 的話，就傳回 result\_1 的值，如果 boolean\_expression 的運算結果是 false 的話，就傳回 result\_2 的值。

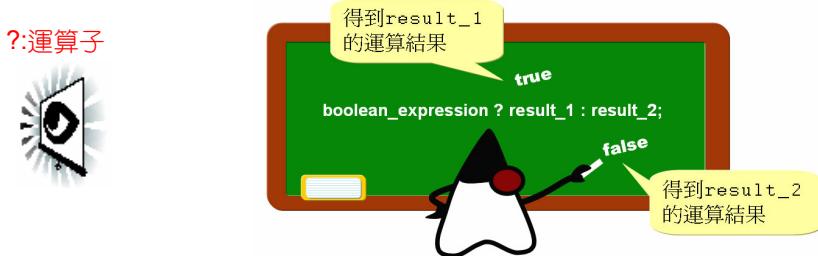


圖 6-9 三元運算子?:的使用

可以使用這個三元運算子來改寫一下程式碼 6-3，判斷使用者輸入的是奇數還是偶數：

**程式碼 6-4 OddTest3.java**

```

1  public class OddTest3 {
2      public static void main(String[] args) {
3          java.util.Scanner scanner =
4              new java.util.Scanner(System.in);
5
6          System.out.print("輸入整數：");
7          int input = scanner.nextInt();
8
9          int remain = input % 2;
10
11         System.out.println(input + " 是" +
12             ((remain == 0) ? "偶數" : "奇數"));
13     }
14 }
```

如果(`remain == 0`)運算結果為 true，則傳回"偶數"的字串，如果運算結果為 false，則傳回"奇數"的字串，程式的執行結果如圖 6-10 所示：

## 建立 if 和 if/else 陳述

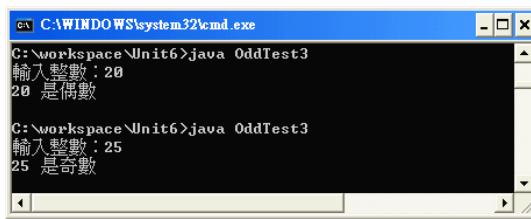


圖 6-10 程式碼 6-4 的執行結果



圖 6-11 開車時的結策判斷



**自我測驗**—下面這個程式如果條件式一不成立，會執行哪一條陳述？

```
if(條件式一)
    if(條件式二)
        陳述句一;
else
    陳述句二;
```



**自我測驗**—寫一個程式，依您所輸入的分數來判斷學生成績是否不小於 60 分，以決定其是否及格，如果是則傳回字元 '是'，否則傳回字元 '否'。



**自我測驗**—寫一個程式，判斷輸入的整數為是否為 3 跟 2 的倍數。

## 使用 switch 陳述

對於「若運算結果等於 1...，執行...，若運算結果等於 2，執行...，若運算結果等於 3...，執行...」這樣的條件分支判斷，您可以使用 switch 陳述來完成。

### switch 語法結構

switch 的語法結構如下所示：

**switch 結構**



```
switch ( variable ){
    case literal_value:
        code_block;
        [break;]
    case another_literal_value:
        code_block;
        [break;]
    [default:]
        code_block;
}
```

這邊是冒號，  
別看錯了！



**switch  
執行流程**



每一個 case 下的 break 可以省略，如果撰寫了 break，則執行完對應的 block 之後，遇到 break 就會離開整個 switch 定義的區塊。

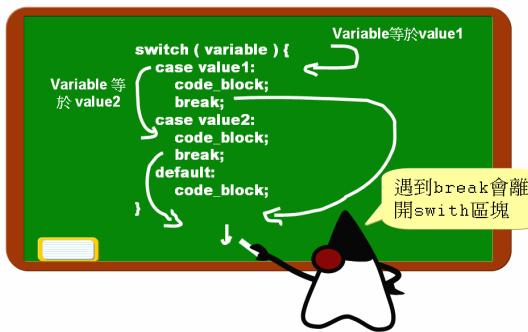


圖 6-12 switch 的執行流程

## 使用 switch 陳述

**default** 的設定可有可無，如果設定了 **default**，則當所有的數值比對都沒有符合的時候，就會執行 **default** 中定義的陳述句，如果沒有設定 **default** 的話，則直接離開 **switch** 定義的區塊。

來看個實際的例子，可以使用 **switch** 改寫一下程式碼 6-4，將當中的鏈狀 **if/else** 結構用 **switch** 結構取代。

**程式碼 6-5 SwitchDemo.java**

```

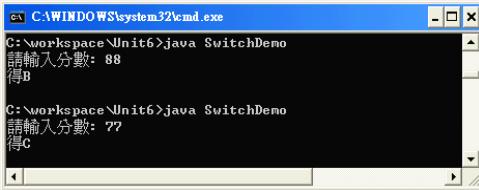
1  public class SwitchDemo {
2      public static void main(String[] args) {
3          java.util.Scanner scanner =
4              new java.util.Scanner(System.in);
5
6          System.out.print("請輸入分數: ");
7          int score = scanner.nextInt();
8          int level = (int) score/10;
9
10         switch(level) {
11             case 10:
12             case 9:
13                 System.out.println("得 A");
14                 break;
15             case 8:
16                 System.out.println("得 B");
17                 break;
18             case 7:
19                 System.out.println("得 C");
20                 break;
21             case 6:
22                 System.out.println("得 D");
23                 break;
24             default:
25                 System.out.println("得 E(不及格)");
26         }
27     }
28 }
```

咦？這邊沒有  
寫 **break**？



## 使用 switch 陳述

程式碼 6-5 運作的方式，是將使用者輸入的數值除以 10，看看商數是多少，再將之丟入 switch 中比對，由此得到分數等級，一個執行的結果如下：



```
cmd C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit6>java SwitchDemo
請輸入分數: 88
得B

C:\workspace\Unit6>java SwitchDemo
請輸入分數: ???
得C
```

圖 6-13 程式碼 6-5 的執行範例

您也許注意到了，程式碼中的第 11 行與第 12 行之間並沒有撰寫 break，case 與 case 之間沒有設定 break 時，執行流程會一路往下執行，而不理會之後的 case 比對值是否符合，所以在程式碼 6-5 中，無論是 100 分或 90 分以上的輸入，都會顯示等級為 A。

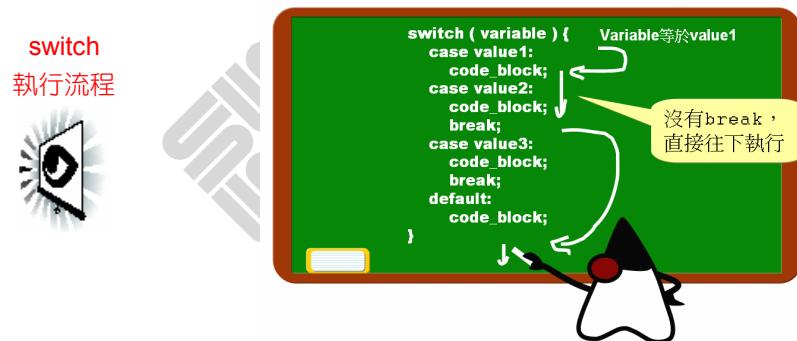


圖 6-14 沒有 break 的話會繼續往下執行



註—您可以在 switch 陳述中使用其它的巢狀迴圈結構。

## 何時使用 switch 結構

switch 結構只能用來檢測相等的狀況，並無法判斷變數值是否大於或小於另一個值。您不能使用 switch 敘述來判斷一個以上的值，而且您只能使用整數或字元的資料型態。

## 使用 switch 陳述

何時使用  
switch



底下是具有使用 switch 敘述資格的一些例子：

- 任何等式的檢測。
- 單一變數的判斷，如 month。
- 判斷值的型態為 int、short、byte 或 char。

在某些應用場合，使用 switch 來替代 if/else 鏈狀結構，還可以得到一些效率上的增進，例如在以下的 if/else 結構中，每一次都要重新取出 score 的變數值以進行比對：

```
if(score >= 90)
    System.out.println("得 A");
else if(score >= 80 && score < 90)
    System.out.println("得 B");
else if(score >= 70 && score < 80)
    System.out.println("得 C");
else if(score >= 60 && score < 70)
    System.out.println("得 D");
else
    System.out.println("得 E(不及格)");
```

而在下面的程式碼片段中，level 變數值只取出一次，之後就直接拿取得的值與 case 所設定的值進行比對：

```
switch(level) {
    case 10:
        System.out.println("得 A");
        break;
    case 9:
        System.out.println("得 B");
        break;
    case 8:
        System.out.println("得 C");
        break;
    case 7:
        System.out.println("得 D");
        break;
    case 6:
        System.out.println("得 E(不及格)");
        break;
    default:
        System.out.println("得 F(不及格)");
}
```



只在這邊取出  
變數值



自我測驗－撰寫一程式，讓使用者輸入字元 R、L、U、D，分別顯示"向右走"、"向左走"、"向上走"、"向下走"。



自我測驗－可以使用 if/else 改寫上面的程式嗎？哪種寫法比較好？



複習題

---

## 複習題

一、 **if/else** 結構中的 **else** 部分是做什麼用的？

選項：包含方法剩餘部分的程式碼。

選項：當 **if** 敘述中的運算式為 **false** 時，所執行的程式碼。

選項：測試運算式是否為 **false**。

二、 在 **switch** 敘述句中，是用以下哪一個關鍵字作為 **case** 敘述的結束？

選項：**default**

選項：**case**

選項：**switch**

選項：**break**

三、 哪一個敘述適合利用 **switch** 結構來作判斷？( p20 )

選項：**switch** 結構能夠判斷變數值是否大於或小於。

選項：**switch** 結構能夠對於一個變數做判斷。

選項：**switch** 結構能夠判斷 **float**、**double**、或 **boolean** 資料型態的值。



## 第七單元

## 迴圈控制

### 單元目標

#### 單元目標



當完成本單元後，您將能學習到：

- while 迴圈與 do while 迴圈的使用
- for 迴圈的使用
- break 與 continue 的使用

這個單元以實例講解，讓您了解如何使用迴圈來進行重複性工作的執行。





討論—下列問題能夠幫助您了解迴圈控制的作用：

- 在哪種情況下，只要某個特定的狀態成立時，您就必須繼續執行某些特定動作？
- 哪種情況下，您會需要建立一個不停止的迴圈，直到程式結束為止？



## 相關資料



**相關資料**－下列參考資料能夠對本單元所討論的話題提供更詳細的說明。

- Learning the Java Language  
<http://java.sun.com/docs/books/tutorial/java/TOC.html>  
Java 程式設計人員的實用手冊，具有許多完整且能夠正常運作的範例。
- Java 學習筆記，碁峰出版社  
這是一本適合剛入門的書籍，它解釋的觀念甚至超過更進階的書籍。



使用 while 與 do while 迴圈

## 使用 while 與 do while 迴圈

程式中常會有一些必須重複執行的指令或動作，例如輸入一串文字，讓程式在文字中尋找符合的字元，此時程式必須對輸入的文字依字元重複進行比對的動作，直到找到符合的字元後結束這個重複性的動作，在 Java 中，您可以使用 [while](#) [迴圈](#)或是 [do while](#) [迴圈](#)來為您完成重複性的動作。

### while 迴圈

**while** [迴圈](#)



先來看看 while 迴圈的語法：

```
while (boolean_expression) {
    code_block;
} // while 結尾
// 程式從這裡繼續
```

**while(true) {**

...

就形成無窮迴圈

所有的迴圈是由下面的部分所構成的：

- `boolean_expression` 是會產生 `true` 或 `false` 的運算式。在每一次執行迴圈之前都必須先判斷此運算式所產生的布林值。
- 假如 `boolean_expression` 判斷出值是 `true`，則會重複執行程式碼區塊（`code_block`）。

**前測式**

**迴圈**



由於進入 `while` 結構時，一開始就會先進行運算式的判斷，以決定要不要執行迴圈本體（Body），所以 `while` 迴圈又稱為**前測式迴圈**，來看個實際的例子，程式碼 7-1 可以讓您輸入一個整數值，程式會依您輸入的值來輸出對應次數的 "Hello ! World !"。

#### 程式碼 7-1 WhileDemo.java

```
1 public class WhileDemo {
2     public static void main(String[] args) {
3         java.util.Scanner scanner =
4             new java.util.Scanner(System.in);
5
6         System.out.print("輸入執行次數：");
7         int input = scanner.nextInt();
```

## 使用 while 與 do while 迴圈

```

8
9     int count = 0; // 計數用
10    while(count < input) { // 判斷次數
11        System.out.println("Hello ! World ! ");
12        count++; // 每次執行後遞增
13    }
14 }
15 }
```

當計數器 `count` 的值小於 `input` 的值時，`while` 迴圈的本體才會執行，每次執行完迴圈前都會遞增 `count`，直到 `count` 的值大於 `input` 的值，此時 `count < input` 的運算結果為 `false`，程式離開迴圈。

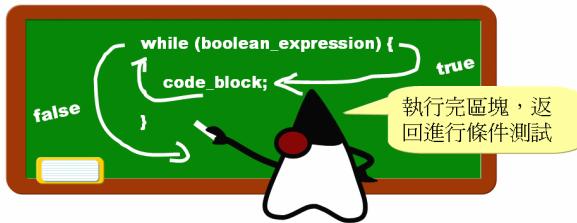


圖 7-1 while 迴圈會先判斷是否執行迴圈本體

程式碼 7-1 的執行範例如下所示：

```

C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit7>java WhileDemo
輸入執行次數：3
Hello ! World !
Hello ! World !
Hello ! World !

C:\workspace\Unit7>java WhileDemo
輸入執行次數：2
Hello ! World !
Hello ! World !

```

圖 7-2 程式碼 7-1 的執行結果

## 巢狀 while 迴圈

思考如果想要畫一個如下的矩形，並且一次畫一個字元，則該如何實現：

## 使用 while 與 do while 迴圈

```
@@@@@@@  
@@@@@@@  
@@@@@@@
```

## 巢狀迴圈



您可以利用 `while` 迴圈畫出矩形的一行。然後，將迴圈放至於另外一個迴圈內連續畫出三行。這樣的迴圈結構稱為 **巢狀 while 迴圈** (*Nested while loop*)。

程式碼 7-2 讓您可以輸入兩個整數，之後根據這兩個數值為長寬，並利用@符號印出矩形：

## 程式碼 7-2 NestedWhileDemo.java

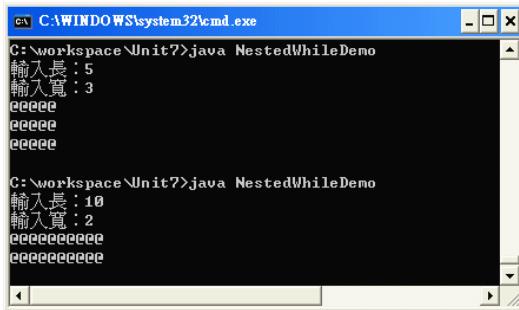
```
1 public class NestedWhileDemo {  
2     public static void main(String[] args) {  
3         java.util.Scanner scanner =  
4             new java.util.Scanner(System.in);  
5  
6         System.out.print("輸入長：");  
7         int length = scanner.nextInt();  
8  
9         System.out.print("輸入寬：");  
10        int width = scanner.nextInt();  
11  
12        int widthCount = 0;  
13        while(widthCount < width) {  
14            int lengthCount = 0;  
15            while(lengthCount < length) {  
16                System.out.print("@");  
17                lengthCount++;  
18            }  
19            System.out.println(); // 換行  
20            widthCount++;  
21        }  
22    }  
23 }  
24 }
```

外層 while

內層 while

## 使用 while 與 do while 迴圈

在進入外層迴圈之後，會再遇到內層迴圈，內層迴圈會一直執行到 `lengthCount < length` 為 `false` 後離開迴圈，並繼續完成外層迴圈的本體，之後外層迴圈的 `widthCount < width` 若還是為 `true`，就會再進行相同的流程，直到外層迴圈的 `widthCount < width` 為 `false` 而離開迴圈，執行的範例結果如下所示：



```
C:\> C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit>java NestedWhileDemo
輸入長：5
輸入寬：3
eeee
eeee
eeee
eeee

C:\workspace\Unit>java NestedWhileDemo
輸入長：10
輸入寬：2
eeeeeeeeee
eeeeeeeeee
```

圖 7-3 程式碼 7-2 的執行結果

**do while** 迴圈

來考慮一個問題，在每次執行迴圈之前，您會想要請使用者輸入一些資訊，之後再根據使用者輸入的資訊來執行迴圈，並依據執行結果來判斷是否該離開迴圈。像這種**先執行、後判斷**的迴圈，您可以使用 **do while** 迴圈來完成，它的語法結構如下所示：

**do while**

迴圈



```
do {
    code_block;
} while (boolean_expression);
```

一定要寫分號



其中：

- 若 `boolean_expression` 的陳述句為 `true` 時，則 `code_block` 就會被再次執行。
- `boolean_expression` 是會產生 `true` 或 `false` 的運算式。在執行 `do` 後而每一次重新執行迴圈之前都必須先判斷此運算式的執行結果。

## 使用 while 與 do while 迴圈

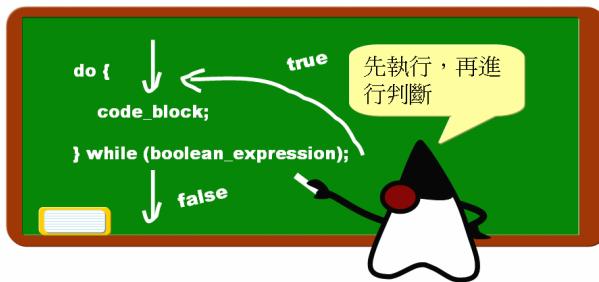


圖 7-4 do while 會先執行迴圈本體



註 – boolean\_expression 之後的分號是必要的。因為這 boolean\_expression 是在這迴圈末端。相反地，在 while 迴圈中並不在結尾加上分號，因為 while 迴圈是在一開始就決定是否要進入 code\_block。

由於先執行，後判斷，所以 do while 又稱之為後測式迴圈。程式碼 7-3 使用 do while 迴圈設計一個簡單的猜數字程式，do 區塊中每次會先請使用者輸入一數字，遇到 while 時再依使用者的輸入判斷是否要執行下一次迴圈。

## 程式碼 7-3 DoWhileDemo.java

```

1  public class DoWhileDemo {
2      public static void main(String[] args) {
3          java.util.Scanner scanner =
4              new java.util.Scanner(System.in);
5
6          int input = 0;
7          do {
8              System.out.print("輸入數字：");
9              input = scanner.nextInt();
10         } while(input != 30);
11
12         System.out.println("恭喜您！猜對了！");
13     }
14 }
```

程式的執行結果如下所示：

## 使用 while 與 do while 迴圈

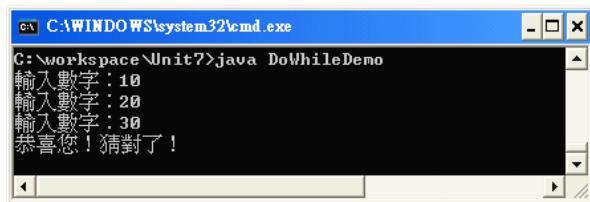


圖 7-5 程式碼 7-3 的執行結果

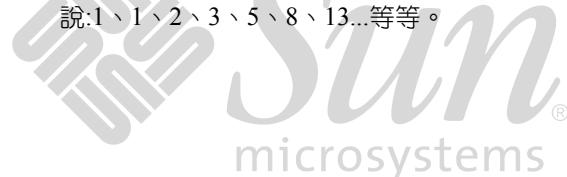
**do while** 迴圈同樣也可以像 **while** 迴圈一樣形式巢狀結構，也就是 **do while** 中再撰寫 **do while** 迴圈。



**自我測驗**—寫一個名為 Counter 的類別，利用 **while** 迴圈從 1 計數到 MAX\_COUNT，其中 MAX\_COUNT 是您自己定義的變數，並顯示計數值。



**自我測驗**—撰寫一個名為 Sequence 的類別，其會顯示由 1、1 開始的數列。下一個數字為上兩個數字的和。舉個例子來說:1、1、2、3、5、8、13...等等。



使用 for 迴圈

## 使用 for 迴圈

for 迴圈也是 Java 所提供的迴圈結構之一，for 迴圈允許您的程式事先決定迴圈重複執行的次數。它運作的方式與 while 迴圈相同，包括 while 迴圈從零到多次重複執行的特性，但會用比較集中的結構來計算數值的範圍。

### for 迴圈

for 迴圈語法為：

for 迴圈



```
for (initialize[, initialize];
      boolean_expression;
      update[, update]) {
    code_block;
}
```

注意要使用分號  
或逗號的時機



- `initialize[, initialize]` 初始化在迴圈內使用的變數（例如：迴圈的計數值）。這部分只會在進入迴圈之前執行一次。如果有多个變數則用逗號分隔。
- `boolean_expression` 是會產生 true 或 false 的運算式。在每一次執行迴圈之前都必須先判斷此運算式所產生的結果。
- `update[, update]` 部分是在負責變數(迴圈的計數器)適時的更新（增加或減少）。這部分處理程序是在本體執行完之後，但是在 `boolean_expression` 測試之前。如果有多个變數則用逗號分隔。
- 當 `boolean_expression` 的陳述為 true 時，則 `code_block` 內的程式碼就會被執行。

for 迴圈也是會執行零次到多次的迴圈：條件式會在執行迴圈本體之前先被執行，假如條件式判斷出來是 false，則迴圈本體部份將不再被執行。

## 使用 for 迴圈

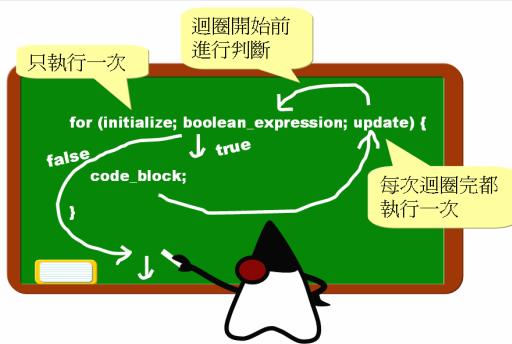


圖 7-6 for 迴圈的執行流程

感覺上 for 迴圈流程似乎比較複雜，實際來寫個例子體會，可以使用 for 迴圈來改寫程式碼 7-1 的 while 迴圈：

## 程式碼 7-4 ForDemo.java

```

1 public class ForDemo {
2     public static void main(String[] args) {
3         java.util.Scanner scanner =
4             new java.util.Scanner(System.in);
5
6         System.out.print("輸入執行次數：");
7         int input = scanner.nextInt();
8
9         for(int i = input; i > 0; i--) {
10             System.out.println("Hello ! World ! ");
11         }
12     }
13 }
```

程式碼中的第 9 行所使用的是遞減式，您也可以用遞增式，例如：

```

for(int i = 0; i < input; i++) {
    System.out.println("Hello ! World ! ");
}
```

程式碼 7-4 與程式碼 7-1 的作用是相同，執行結果可參考圖 7-2 的畫面。

使用 for 迴圈

## 巢狀 for 迴圈

巢狀 for

迴圈



for 迴圈當中也可以再包括 for 迴圈，也就是形成巢狀 for 迴圈，直接以實例說明，可以使用 for 迴圈來改寫程式碼 7-2 並達到相同的作用。

### 程式碼 7-5 NestedForDemo.java

```

1  public class NestedForDemo {
2      public static void main(String[] args) {
3          java.util.Scanner scanner =
4              new java.util.Scanner(System.in);
5
6          System.out.print("輸入長：");
7          int length = scanner.nextInt();
8
9          System.out.print("輸入寬：");
10         int width = scanner.nextInt();
11
12         for(int i = width; i > 0; i--) {
13             for(int j = length; j > 0; j--) {
14                 System.out.print("@");
15             }
16             System.out.println(); // 换行
17         }
18     }
19 }
```

執行結果可以參考圖 7-3 的畫面。



想想看！那麼 for 迴圈跟 while 迴圈要選哪個？

## 比較迴圈的結構

迴圈比較



for、while 與 do/while 迴圈的運作方式都非常類似。然而在任何情況下都會有一個迴圈執行效率比其它的還要好。以下提供了一些使用準則，教您如何決定採用哪一種迴圈結構：

- 當重複的次數不確定並且必須先測試迴圈條件時，就使用 while 迴圈。
- 當重複的次數不確定並且必須先執行某個動作後測試迴圈條件，就使用 do while 迴圈。
- for 迴圈比 while 迴圈更簡潔及容易閱讀，原因是因為它被設計成以分開的數字來計數。因此使用 for 迴圈能夠以預先定義的執行次數來一步一步達成我們賦予它們的工作。



自我測驗—試著使用 for 迴圈來顯示九九乘法表。



break 與 continue

## break 與 continue

在迴圈執行的本體中，如果打算離開迴圈的執行流程，您可以使用 **break** 或 **continue** 來進行控制，在這個小節中將分別說明。

### 使用 break

使用 break



**break** 可以離開目前正在執行的 while、do while、for 迴圈區塊，在這邊直接舉個實例來說明，程式碼 7-6 是個計算成績平均的程式，使用 while 迴圈來執行接受使用者的輸入，當輸入為-1 時終止迴圈並計算結果。

程式碼 7-6 BreakDemo.java

```

1  public class BreakDemo {
2      public static void main(String[] args) {
3          java.util.Scanner scanner =
4              new java.util.Scanner(System.in);
5          int score = 0;
6          int sum = 0;
7          int count = -1;
8          while(true) {
9              count++;
10             sum += score;
11             System.out.print("輸入分數(-1 結束) : ");
12             score = scanner.nextInt();
13
14             if(score == -1)
15                 break;
16         }
17         System.out.println("平均 : "
18                         + (double) sum/count);
19
20     }
21 }
```

中斷！離開  
while 迴圈



## break 與 continue

當使用者輸入-1 時，第 14 行運算結果為 true，所以執行第 15 行的 break，因而離開 while 迴圈。

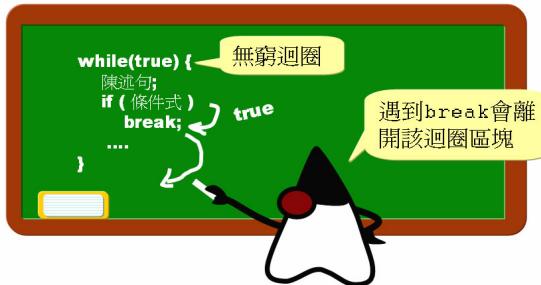


圖 7-7 使用 break 離開迴圈執行流程

對於 do while 或 for 迴圈，break 的作用也是相同的，程式碼 7-6 的執行結果如下所示：

```
C:\WINDOWS\system32\cmd.exe
C:\workspace\JUnit7>java BreakDemo
輸入分數<-1結束> : 10
輸入分數<-1結束> : 20
輸入分數<-1結束> : -1
平均 : 15.0
```

圖 7-8 程式碼 7-6 的執行結果

## 使用 continue

使用 continue



continue 主要使用於 for 迴圈，continue 會略過 continue 該行之後的 for 迴圈本體，直接繼續迴圈循環，直接來看個實際的例子，由於 continue 的關係，程式碼 7-7 的執行結果將不會顯示 "i = 5"。

### 程式碼 7-7 ContinueDemo.java

```
1 public class ContinueDemo {
2     public static void main(String[] args) {
3         for(int i = 1; i < 10; i++) {
4             if(i == 5)
5                 continue;           // Directly continues to the next iteration
6             System.out.println("i = " + i);
7         }
8     }
9 }
```

#### 迴圈控制

直接繼續下一個迴圈

7-15

## break 與 continue

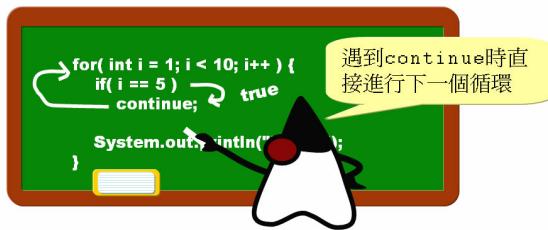


圖 7-9 continue 會略過之後的迴圈本體

程式碼 7-7 的執行結果如下，注意到確實沒有顯示 "i = 5" 的輸出：

```
C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit7>java ContinueDemo
i = 1
i = 2
i = 3
i = 4
i = 6
i = 7
i = 8
i = 9
```

圖 7-10 程式碼 7-7 的執行結果

**break 搭配標籤**

**break  
加標籤**



break 可以搭配標籤 (Label) 來使用，藉由設定標籤區塊，break 不僅是離開迴圈，而是離開整個標籤區塊的範圍 (Scope) 而繼續往下執行，程式碼 7-8 是個簡單的示範。

**程式碼 7-8 BreakLabel.java**

```
1 public class BreakLabel {
2     public static void main(String[] args) {
3         label1 : {
4             for(int i = 0; i < 10; i++) {
5                 if(i == 9) {
6                     System.out.println(
7                         "break label1");
8                     break label1;
9                 }
10            }
11            System.out.println("for 迴圈之後執行");
12        } // label1 區塊結尾
}
```



其實這種用  
法很少見

```
13    }
14 }
```

當“`i == 9`”成立為時執行至第 8 行，結果不只會離開迴圈，還會離開整個 `label1` 所設定的區塊。

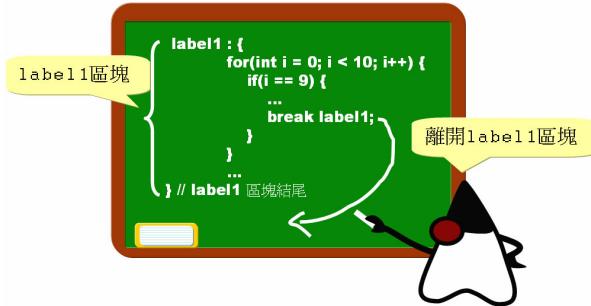


圖 7-11 break 搭配標籤的執行流程

由於離開了整個 `label1` 區塊，所以程式碼 7-8 不會執行第 11 行，執行結果只會顯示 `break label1`。

## continue 搭配標籤

`continue`  
加標籤



`continue` 可以搭配標籤來使用，標籤設定在 `for` 迴圈之前，在 `continue` 該行之後，會回到指定標籤處的 `for` 迴圈重新執行循環，直接以程式碼 7-9 作示範。

### 程式碼 7-9 ContinueLabel.java

```
1 public class ContinueLabel {
2     public static void main(String[] args) {
3         label1:
4             for(int i = 0; i < 5; i++) {
5                 System.out.println();
6             label2:
7                 for(int j = 0; j < 5; j++) {
8                     if(j == 3) {
9                         continue label1;
10                }
11                System.out.print("\tj = " + j);
12            }
13        }
14    }
15 }
```

這種用法  
也很少見



迴圈控制

7-17

## break 與 continue

```

13      }
14      }
15  }

```

在程式中，如果"*j == 3*"為 true，則執行第 8 行，即直接進行下一個 label2 下的 for 循環，所以執行結果不會顯示"*j = 3*"，如下圖所示：

```

C:\> C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit7>java ContinueLabel
j = 0    j = 1    j = 2    j = 4
j = 0    j = 1    j = 2    j = 4
j = 0    j = 1    j = 2    j = 4
j = 0    j = 1    j = 2    j = 4
j = 0    j = 1    j = 2    j = 4

```

圖 7-12 程式碼 7-9 設定 continue label2 的執行結果

如果您將第 8 行改為 continue label1，則執行時直接跳至 label1 的 for 迴圈重新循環，所以結果是"*j = 3*"、"*j = 4*"都不會執行，如下圖所示：

```

C:\> C:\WINDOWS\system32\cmd.exe
C:\workspace\Unit7>java ContinueLabel
j = 0    j = 1    j = 2
j = 0    j = 1    j = 2
j = 0    j = 1    j = 2
j = 0    j = 1    j = 2
j = 0    j = 1    j = 2

```

圖 7-13 程式碼 7-9 設定 continue label1 的執行結果



**自我測驗**－有必要使用 break 或 continue 嗎？討論一下使用它們的缺點。

## 複習題

- 一、何者可以讓您重複執行判斷式，並且反覆地執行程式本體？  
選擇：物件  
選擇：類別  
選擇：迴圈  
選擇：方法
- 二、在大部份的情況中，哪一種迴圈允許您宣告變數？  
選擇：`do/while` 復圈  
選擇：`while` 復圈  
選擇：巢狀式 `while` 復圈  
選擇：`for` 復圈
- 三、使用 `for` 復圈可以事先決定迴圈重複執行的次數。  
選擇：真  
選擇：假





## 附錄 A

### Java 程式語言的關鍵字

此附錄列出了 Java 程式語言所有的關鍵字

在 Java 程式語言裡 **關鍵字 (Keywords)** 是為編譯器特別保留的指令。關鍵字不能當作類別、方法及變數的識別字。

表 A-1 Java 關鍵字

abstract	do	implements	protected	throws
boolean	double	import	public	transient
break	else	instanceof	return	true
byte	extends	int	short	try
case	false	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
const	for	null	synchronized	
continue	goto	package	this	

 註一 在 Java 程式語言中，true、false 或是 null 都是依照字面值 (Literal value) 的意思。



## 附錄 B

# Java 程式語言命名慣例

這附錄中統一了在這課程中所提到的命名規則（Rules）及準則（Guidelines）。規則是由 Java 編譯器或直譯器所規定的。準則是用來提高您程式品質的建議。

以下的規則規定了內容及識別字的結構：

- 識別字的第一個字元必須符合下面要求：
  - 大寫字母 (A–Z)
  - 小寫字母 (a–z)。
  - 底線字元 (\_)
  - 美元字元 (\$)
  - Unicode 通用字元
- 識別字不能與 Java 關鍵字相同。

以下的準則設計用來幫助您命名良好的識別字：

- 每一個變數或方法從小寫字母開始；之後所連接的字母再使用大寫，例如 myVariable 或者 getSettings。
- 每一個類別從大寫字母開始；之後所連接的字母再用大寫，例如 ShirtTest。
- 常數識別字通常使用大寫字母、和分離的字之間加上底線，例如 SALES\_TAX。
- 選擇容易記憶的名稱及設計來粗略地指出變數所使用的目的。

良好的識別字範例：

- customerID (變數識別字)。
- isClosed (布林變數識別字)。
- play (方法識別字)。
- playMusic (方法識別字)。
- Order (類別識別字)。

## Java 程式語言命名慣例



註 – Java 程式語言是一種有大小寫之分的程式語言。[Case sensitivity](#) 表示每一個字母是有區分大小寫的。Java 程式語言視兩個大小寫相異的識別字是不同的。例如：如果您建立一個稱為 “order” 的變數，那麼之後您不能用 “Order” 來表示它。

