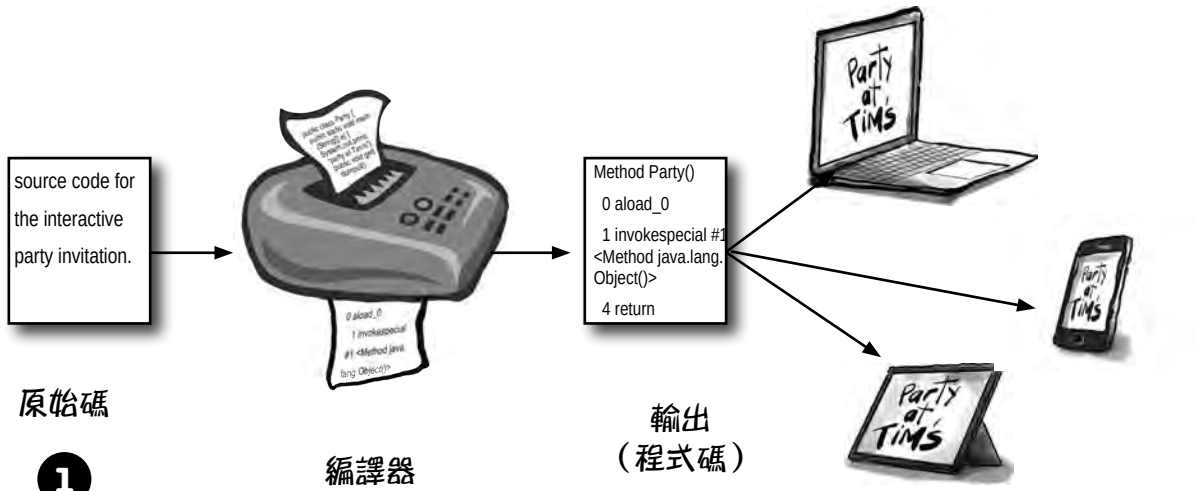
1 快速導覽

突破表面

Java 帶你進入新世界。從它初始的 1.02 版本向大眾公開以來，Java 就以其友善的語法、物件導向的功能、記憶體管理以及最出色的可攜性承諾，吸引著程式設計師。**write-once/run-anywhere（寫一次就能在任何地方執行）**的誘惑力實在是太強了。忠實的追隨者人數暴增，同時程式設計師也在與 bugs、限制以及，哦，沒錯，它的速度很慢的事實不停對抗。但那是很久以前的事了。如果你是剛開始學習 Java，**你很幸運**。過去，我們中的一些人不得不在雪地裡走五英里，而且都是上坡路（光著腳），才能讓最微不足道的應用程式執行。但你，為什麼，你可以駕馭今天**更時尚**、**更快速**、**更容易讀和寫**的 Java！

Java 的運作方式

目標是編寫一個應用程式（在此例中，是一個互動式的聚會邀請），並讓它在你朋友的任何設備上執行。



原始碼

1

創建一個原始碼檔案。使用一個確立的協定（在此為 Java 語言）。

編譯器

2

使用一個原始碼編譯器（source code compiler）處理你的檔案。編譯器會檢查是否有錯誤，在它確定一切都能正確執行之後，才會讓你進行編譯。

輸出（程式碼）

3

編譯器創建一個新的檔案，編譯為 Java ***bytecode***。任何能夠執行 Java 的裝置都將能夠直譯（interpret）或翻譯（translate）這個檔案，使其能夠執行。編譯後的 bytecode 獨立於平台。

虛擬機器

4

你的朋友們都有一個用軟體實作的 Java **虛擬**機器（Java ***virtual*** machine，JVM），在他們的電子裝置中執行。當你的朋友執行你的程式時，虛擬機器將讀取並執行 bytecode。

你會用 Java 做什麼

你要輸入一個原始碼檔案，用 javac 編譯器進行編譯，然後在 Java 虛擬機器上執行編譯後的 bytecode。

```
import java.awt.*;
import java.awt.event.*;

class Party {
 public void buildInvite() {
  Frame f = new Frame();
  Label l = new Label("Party at Tim's");
  Button b = new Button("You bet");
  Button c = new Button("Shoot me");
  Panel p = new Panel();
  p.add(l);
 } // more code here...
}
```

原始碼

1

輸入你的原始碼。

儲存為 ***Party.java***

編譯器

2

藉由執行 javac（編譯器應用程式）編譯 ***Party.java*** 檔案。如果沒有錯誤，你會得到名為 ***Party.class*** 的第二個檔案。

編譯器產生的 Party.class 檔案是由 *bytecodes* 組成的。

```
Method Party()
  0 aload_0
  1 invokespecial #1 <Method
java.lang.Object()>
  4 return
Method void buildInvite()
  0 new #2 <Class java.awt.Frame>
  3 dup
  4 invokespecial #3 <Method
java.awt.Frame()>
```

輸出（程式碼）

3

編譯後的程式碼：***Party.class***

虛擬機器

4

用 ***Party.class*** 檔案啟動 Java 虛擬機器（JVM）來執行該程式。JVM 會將 *bytecode* 翻譯成底層平台能夠理解的東西，並執行你的程式。

（注意：這並**不**是要作為一個入門教程而存在的…你很快就會開始寫真正的程式碼，但現在，我們只是想讓你感受一下全部是如何組合在一起的。

換句話說，這個頁面上的程式碼並不十分真實，不要試圖去編譯它。）

Java 簡史

Java 最初是在 1996 年 1 月 23 日發行的（有人會說是「脫逃出來的」）。它已經超過 25 歲了！在最初的 25 年裡，Java 作為一種語言持續演進，Java API 也大幅增長。我們的最佳估計值是，在過去的 25 年裡，已經有超過幾千幾百億行的 Java 程式碼被寫了出來。當你花時間用 Java 編寫程式時，你肯定會遇到一些相當老舊的 Java 程式碼，還有一些新得多的程式碼。Java 以其回溯相容性（backward compatibility）而聞名，所以舊的程式碼可以很愉快地在新的 JVM 上執行。

在本書中，我們一般會先使用較早的程式碼風格（記住，你很有可能會在「真實世界」中遇到這樣的程式碼），然後再介紹風格較新的程式碼。

藉由類似的方式，我們有時會向你展示 Java API 中較老的類別，然後再向你展示較新的替代品。

速度和記憶體用量

Java 剛發行時，它的速度很慢。但不久之後，HotSpot VM 就被創造出來了，其他效能增強器也是如此。雖然 Java 確實不是最快的語言，但它被視為一種非常快速的語言：幾乎與 C 和 Rust 等語言一樣快，而且比大多數其他語言**快得多**。

Java 有一個神奇的超級力量：JVM。Java Virtual Machine 可以在你程式碼執行的同時對其進行最佳化，因此，無須編寫專門的高性能程式碼就能建立非常快的應用程式。

但是，我們可以全然披露：與 C 和 Rust 相比，Java 使用大量的記憶體。

削尖你的鉛筆

→ 答案在第 6 頁

看看編寫 Java 有多麼容易

試著猜測每一行程式碼在做什麼…

（答案在下一頁）。

```
int size = 27;
String name = "Fido";
Dog myDog = new Dog(name, size);
x = size - 5;
if (x < 15) myDog.bark(8);

while (x > 3) {
  myDog.play();
}

int[] numList = {2, 4, 6, 8};
System.out.print("Hello");
System.out.print("Dog: " + name);
String num = "8";
int z = Integer.parseInt(num);

try {
  readTheFile("myFile.txt");
}
catch (FileNotFoundException ex) {
  System.out.print("File not found.");
}
```

宣告一個名為「size」的整數變數並賦予它 27 這個值

如果 x（值為 22）小於 15，就告訴狗狗叫 8 次

印出 "Hello"…大概是在命令列上

問：**Java 版本的命名慣例令人困惑。有 JDK 1.0，還有 1.2、1.3、1.4，然後跳到 J2SE 5.0，然後變成了 Java 6、Java 7，而我上次查看時，Java 已經到了 Java 18。這到底是怎麼回事？**

答：在過去 25 年多的時間裡，版本號碼有很大的變化！我們可以忽略字母（J2SE/SE），因為那些字母現在並沒有真正被使用。數字就比較麻煩了。

嚴格來講，Java SE 5.0 實際上是 Java **1**.5。6（1.6）、7（1.7）和 8（1.8）也一樣。理論上，Java 仍然在 1.x 版本上，因為新版本是回溯相容的，一直往回到 1.0 都是。

然而，版本號碼與每個人所用的名稱不同，這確實有點令人困惑，所以從 Java 9 開始的官方版號只是數字，沒有「1」的前綴，也就是說，Java 9 真的是版本 9，而非版本 1.9。

在本書中，我們將使用 1.0-1.4 的常見慣例，然後從 5 開始，我們將放棄「1」的前綴。

此外，從 Java 9 在 2017 年 9 月發行以來，每六個月就會有一個 Java 版本釋出，每個版本都有一個新的「主要（major）」版號，所以我們從 9 到 18 的變化非常快！

削尖你的鉛筆 解答

看看編寫 Java 有多麼容易

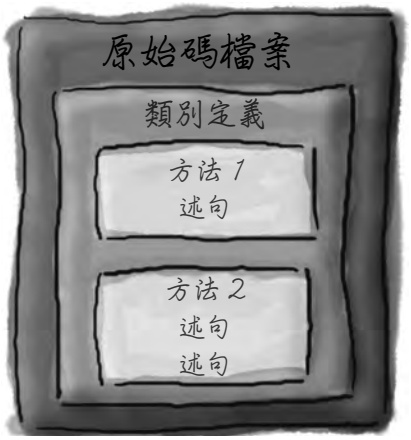
```
int size = 27;
String name = "Fido";
Dog myDog = new Dog(name, size);
x = size - 5;
if (x < 15) myDog.bark(8);

while (x > 3) {
  myDog.play();
}

int[] numList = {2, 4, 6, 8};
System.out.print("Hello");
System.out.print("Dog: " + name);
String num = "8";
int z = Integer.parseInt(num);

try {
  readTheFile("myFile.txt");
}
catch (FileNotFoundException ex) {
  System.out.print("File not found.");
}
```

先別煩惱你是否能理解這些東西！

這裡的一切在書中都會有非常詳細的解釋（大部分在前 40 頁內）。如果 Java 類似於你過去使用過的語言，其中的一些就會相當簡單。若非如此，那也不要擔心。我們會到達那裡的…

宣告一個名為「size」的整數變數並賦予它 27 這個值
宣告名為「name」的一個字元字串，並賦予它 "Fido" 這個值
宣告一個新的 Dog 變數「myDog」，並讓此新的 Dog 使用「name」和「size」
從 27（「size」的值）減去 5，並將其指定給一個名為「x」的變數
如果 x（值為 22）小於 15，就告訴狗狗叫 8 次
只要 x 大於 3 就繼續跑迴圈…
告訴狗狗去玩耍（不管那對狗狗而言代表什麼）
這看起來像是迴圈結尾：{} 內的所有東西都在迴圈中完成
宣告一個整數串列變數「numList」，並把 2、4、6、8 放到該串列中
印出 "Hello"…大概是在命令列上
在命令列上印出 "Dog: Fido"（「name」的值為 "Fido"）
宣告一個字元字串變數「num」並賦予它 "8" 這個值
將字元字串 "8" 轉換為實際的數值 8
試著做些事情…可能我們要嘗試的事情並不保證行得通…
讀取一個名為「myFile.txt」的文字檔案（或至少「試著」去讀該檔案…）
必定是「要嘗試的事情」之結尾，所以我猜你可以嘗試許多事情…
這必然是你發現你所嘗試的事情是否行得通的地方…
若是我們所嘗試的事情失敗了，就印出 "File not found" 到命令列
看起來 {} 內的所有東西就是在「try」行不通的時候要做的事情…

Java 的程式碼結構



在一個原始碼檔案中，放入一個類別（class）。

在一個類別中，放入方法（methods）。

在一個方法中，放入述句（statements）。

原始碼檔案中放些什麼？

一個原始碼檔案（延伸檔名為 *.java*）通常包含一個**類別**定義。這個類別代表了你程式的一個片段，雖然一個非常小型的應用程式可能只需要單一個類別。類別必須放在一對大括號（curly braces）內。

```
public class Dog {




}
```

class

類別中放些什麼？

一個類別會有一或多個**方法**。在 Dog 類別中，***bark*** 方法將保存 Dog 應該如何吠叫（bark）的指令。你的方法必須在類別內部宣告（換句話說，在類別的大括號內）。

```
public class Dog {
   void bark() {



   }
}
```

method

方法中放些什麼？

在一個方法的大括號內，寫下你對該方法應該如何進行的指令。方法程式碼（*code*）基本上是一組述句，就現在而言，你可以把一個方法看成是一個函式（function）或程序（procedure）。

```
public class Dog {
   void bark() {
     statement1;
     statement2;
   }
}
```

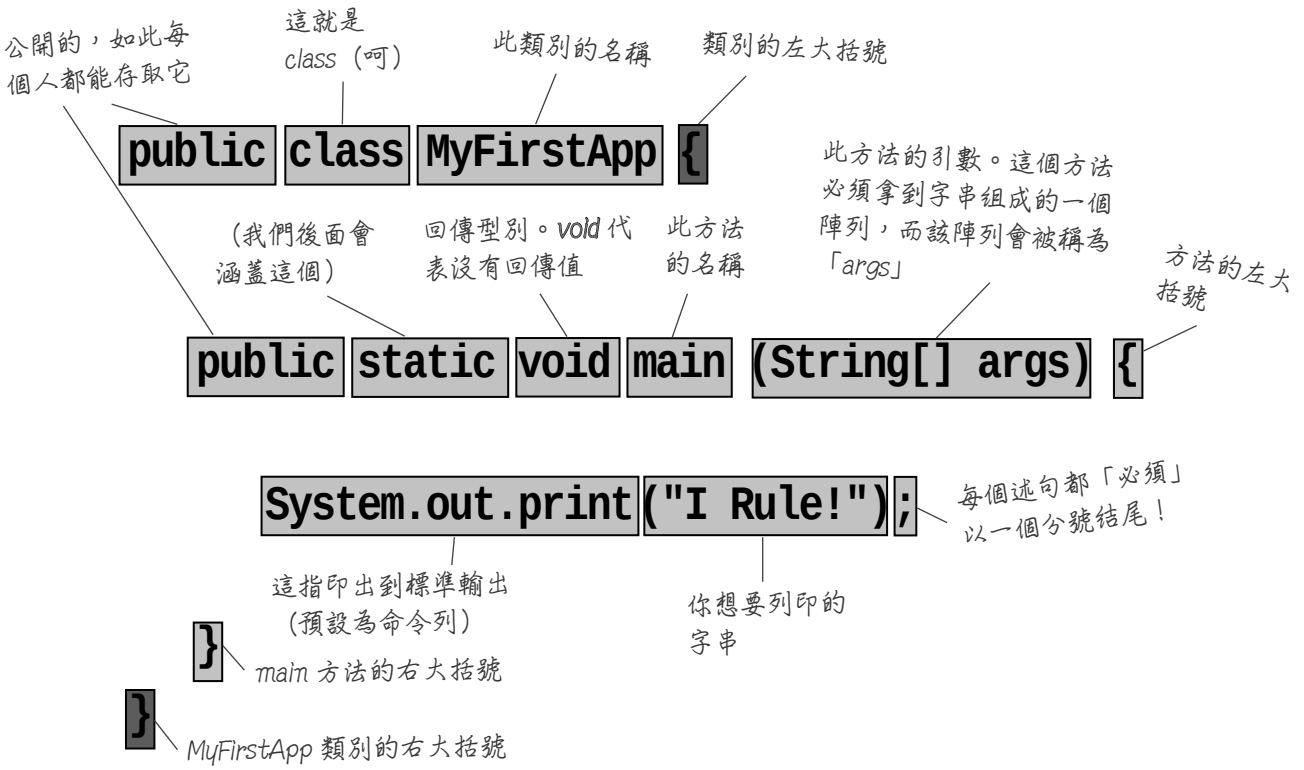
statements

解剖一個類別

當 JVM 開始執行時，它尋找你在命令列提供給它的類別。然後它開始尋找一個特別編寫的方法，這個方法看起來就像：

```
public static void main (String[] args) {
   // 你的程式碼在這裡
}
```

接下來，JVM 會執行你主要（main）方法的大括號 { } 之間的所有東西。每個 Java 應用程式（application）都必須至少有一個**類別**，並且至少有一個 **main** 方法（不是每個類別一個 main，而是每個應用程式一個 main）。



現在先別擔心是否要把這些全都背起來…本章只是帶你入門而已。

以一個 main() 撰寫一個類別

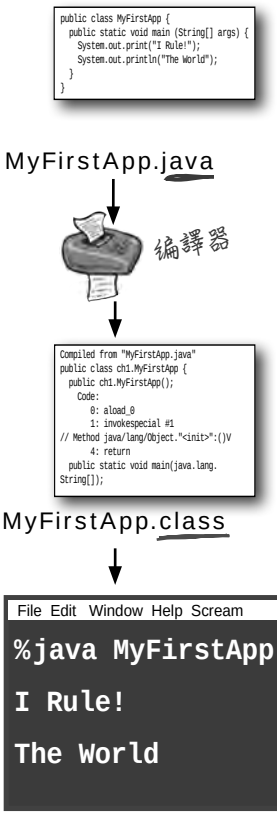
在 Java 中，所有的東西都要放到一個**類別**中。你要輸入你的原始碼檔案（延伸檔名為 *.java*），然後將其編譯成一個新的類別檔案（延伸檔名為 *.class*）。當你執行你的程式時，實際上是在執行一個類別。

執行一個程式意味著告訴 Java Virtual Machine（JVM）「載入 **MyFirstApp** 類別，然後開始執行其 **main()** 方法。持續執行，直到 main 中的所有程式碼都完成為止」。

在第 2 章「物件村之旅」中，我們將深入探討整個類別，但現在，你唯一需要問的是，***我如何編寫 Java 程式碼，使它能夠執行呢？***而這一切都要從 **main()** 開始。

main() 方法是你程式開始執行的地方。

不管你的程式有多大（換句話說，無論你的程式使用多少個類別），都必須有一個 **main()** 方法來啟動一切。



❶ 儲存

MyFirstApp.java

❷ 編譯

```
javac MyFirstApp.java
```

❸ 執行

```
java MyFirstApp
```

```
public class MyFirstApp {

   public static void main (String[] args) {
     System.out.println("I Rule!");
     System.out.println("The World");
   }

}
```

今晚主題：**編譯器和 JVM 在「誰更重要？」的問題上爭論不休**

Java 虛擬機器

什麼，你在開玩笑嗎？**哈囉**，我就是 Java。我才是真正讓程式執行的人。編譯器就只是給你一個檔案，僅此而已。只是一個檔案，你可以把它印出來，用來做壁紙、點火、襯托鳥籠，就那些事情，除非我在那裡執行它，否則那個檔案什麼也做不了。

這就是另一個點了，編譯器根本沒有幽默感。話說回來，如果你整天都得吹毛求疵地查驗那些微小的語法違規行為，會變成這樣也無可厚非啦…

我不是說你完全沒用，但真的要說的話，你所做的到底是什麼？我是認真不懂。程式設計師只要用手寫 bytecode，我就能接受。老兄，你可能很快就會失去工作了。

（果然沒有幽默感。）但你仍然沒有回答我的問題，你究竟都在做些什麼？

編譯器

我不欣賞那種語氣。

你說什麼？若是沒有我的話，你到底能執行什麼？如果你不知道的話，我可以告訴你，Java 被設計為使用 bytecode 編譯器是有原因的。如果 Java 是一種純粹的直譯語言（interpreted language），也就是說，執行的同時，虛擬機器還必須翻譯直接來自文字編輯器的原始碼，那麼 Java 程式的執行速度就會慢得令人髮指。

不好意思喔，但這是一個相當無知（更不用說是傲慢）的觀點。雖然從理論上講，你的確可以執行格式正確的任何 bytecode，即使那不是由 Java 編譯器所產生的，但在實務上這是很荒謬的。程式設計師用手寫 bytecode 就像你放假出遊，想要留念時，就用手繪畫而非拍照，當然，那是一門藝術，但大多數人更願意以不同的方式運用他們的時間。另外，如果你不用「老兄」稱呼我，我會很感激。

還記得 Java 是一種強定型語言（strongly typed language）嗎？那意味著我不能允許變數持有型別錯誤的資料。這是一項重要的安全功能，我能在絕大多數違規行為在你那邊發生之前阻止它們。而且，我還…

Java 虛擬機器

但有些還是跑過來了！我可能擲出 ClassCastException 例外，有時我還會發現某些人試圖把型別不對的東西放到陣列中，而那個陣列是被宣告為容納其他東西的，而且…

好的，當然是這樣。但安全性的問題呢？看看我所做的所有安全工作，而你在做什麼，檢查分號？哇啊！好大的安全風險啊！謝天謝地，有你真好呢！

不管怎麼樣，我還是要做同樣的事情，只為了確保沒有人在你之後潛入，在執行之前竄改 bytecode。

哦，你可以期待一下，好兄弟。

編譯器

對不起，我還沒說完呢。是的，有一些資料型別的例外確實會在執行時期出現，但其中一些必須被允許，以支援 Java 其他的重要功能：動態繫結（dynamic binding）。在執行時，一個 Java 程式可能包括新的物件，而這些物件甚至不為原本的程式設計師所知，所以我必須允許一定程度的彈性。但我的工作是阻止在執行時期永遠不會成功，也不可能成功的事情。通常，我可以判斷什麼東西不會成功，舉例來說，如果程式設計師意外用了 Button 物件作為 Socket 連線，我會偵測到，從而保護他們在執行時不會造成傷害。

不好意思，正如他們所說的，我是第一道防線。我之前說過的資料型別違規行為如果被允許出現，就會在程式中造成嚴重的破壞。我也是阻止違規存取的人，例如程式碼試圖呼叫一個私有方法（private method），或改變一個出於安全理由絕對不能更改的方法。我防止人們接觸到他們不應該看到的程式碼，包括試圖存取另一個類別關鍵資料的程式碼。要描述我工作的重要性，需要幾個小時，甚至幾天的時間。

當然，但正如我之前指出的，如果我不阻止可能相當於 99% 的潛在問題，你就會陷入停滯。看來我們的時間用完了，所以我們必須在之後的聊天中重新探討這個問題。

你可以在 main 方法中說些什麼？

一旦你進入 main（或任何方法），有趣的事情就開始了。你可以說在大多數程式語言中能說的正常事情，以**讓電腦做事**。

你的程式碼可以告訴 JVM 去：

1 做某些事情

述句（statements）：宣告（declarations）、指定（assignments）、方法呼叫等等。

```
int x = 3;
String name = "Dirk";
x = x * 17;
System.out.print("x is " + x);
double d = Math.random();
// 這是一個註解（comment）
```

2 重複做某些事

迴圈（loops）：*for* 和 *while*

```
while (x > 12) {
  x = x - 1;
}

for (int i = 0; i < 10; i = i + 1) {
  System.out.print("i is now " + i);
}
```

3 在此條件下做某些事

分支（branching）：*if/else* 測試

```
if (x == 10) {
  System.out.print("x must be 10");
} else {
  System.out.print("x isn't 10");
}
if ((x < 3) && (name.equals("Dirk"))) {
  System.out.println("Gently");
}
System.out.print("this line runs no matter what");
```

語法樂趣

★ 每個述句都必須以一個分號做結尾。

```
x = x + 1;
```

★ 單行註解以兩個斜線開頭。

```
x = 22;
// 這行讓我心煩意亂
```

★ 大多數的空白都不重要。

```
x      =      3  ;
```

★ 變數以一個**名稱（name）**和**型別（type）**來宣告（你將在第 3 章中了解到所有的 Java 型別）。

```
int weight;
// 型別：int，名稱：weight
```

★ 類別與方法必須定義在一對大括號內。

```
public void go() {
   // 神奇的程式碼在這裡
}
```

迴圈一圈又一圈…

Java 有很多迴圈構造：while、do-while 以及 *for*，是最古老的迴圈。你會在本書的後面得到完整的迴圈資訊，但不是現在。讓我們從 while 開始。

語法（更別說邏輯）是如此簡單，你可能已經快睡著了。只要某些條件為真，一切的事情就會在迴圈區塊（loop *block*）內進行。迴圈區塊由一對大括號（curly braces）界定，所以你想重複的東西必須放在該區塊內。

迴圈的關鍵是條件測試（*conditional test*）。在 Java 中，條件測試是一個運算式，其結果是一個 *boolean* 值，換句話說，即結果為**真（*true*）**或**假（*false*）**的東西。

如果你說「While *iceCreamInTheTub is true*, keep scooping（浴缸裡的冰淇淋還有的話，就繼續舀）」，你就有一個清楚的 boolean 測試。浴缸裡要麼有冰淇淋，要麼就沒有。但是如果你說：「While *Bob* keep scooping（當 Bob 繼續舀的時候）」，你有的就不是一個真正的測試。為了使之有效，你必須把它改為：「While Bob is snoring...（當 Bob 在打鼾時…）」或「While Bob is *not* wearing plaid...（當 Bob 穿的不是格紋時…）」。

簡單的 boolean 測試

你可以透過檢查一個變數的值來做簡單的 boolean 測試，使用一個像這樣的比較運算子：

<（小於）

>（大於）

==（等於）（你沒看錯，那是兩個等號）

注意指定（*assignment*）運算子（單一個等號）和相等運算子（兩個等號）之間的區別。很多程式設計師在想要 **==** 的時候不小心輸入了 **=**（但你不會）。

```
int x = 4; // 指定 4 給 x
while (x > 3) {
  // 迴圈程式碼會執行
  // 因為 x 大於 3
  x = x - 1; // 不然我們會永遠迴圈
}
int z = 27; //
while (z == 17) {
  // 迴圈程式碼不會執行
  // 因為 z 不等於 17
}
```

問：**為什麼所有東西都必須在一個類別裡？**

答：Java 是一種物件導向（object-oriented，OO）的語言。這不像以前那樣，你只有蒸汽驅動的編譯器，只能寫出一整個帶有一堆程序的原始碼檔案。在第 2 章「*物件村之旅*」中，你將了解到類別是物件的藍圖，而 Java 中幾乎所有的東西都是物件。

問：**我必須在我寫的每一個類別裡都放一個 main 嗎？**

答：不需要。一個 Java 程式可能會使用幾十個（甚至幾百個）類別，但你可能只有一個帶有 main 方法的類別，即啟動程式開始執行的那個。

問：**在我用的其他語言中，我可以對一個整數做 boolean 測試。在 Java 中，我可以像這樣說嗎？**

```
int x = 1;
while (x){ }
```

答：不可以。在 Java 中，*boolean* 和整數（*integer*）是不相容的型別。由於條件測試的結果必須是一個 boolean 值，所以你唯一能直接測試的變數（不使用比較運算子）是 ***boolean*** 值。舉例來說，你可以說：

```
boolean isHot = true;
while(isHot) { }
```

一個 while 迴圈範例

```
public class Loopy {
  public static void main(String[] args) {
    int x = 1;
    System.out.println("Before the Loop");
    while (x < 4) {
      System.out.println("In the loop");
      System.out.println("Value of x is " + x);
      x = x + 1;
    }
    System.out.println("This is after the loop");
  }
}
```

這是輸出

```
% java Loopy
Before the Loop
In the loop
Value of x is 1
In the loop
Value of x is 2
In the loop
Value of x is 3
This is after the loop
```

本章重點

- 述句以一個分號 **;** 做結。
- 程式碼區塊由一對大括號 **{ }** 所定義。
- 以一個名稱和型別宣告一個 *int* 變數：**int x;**。
- **指定**運算子是一個等號 **=**。
- **相等**運算子使用兩個等號 **==**。
- 只要條件測試為**真**，*while* 迴圈就會執行其區塊內的所有東西（由大括號所定義）。
- 如果條件測試為**假**，*while* 迴圈程式碼區塊就不執行，並會往下移到緊接於迴圈區塊之後的程式碼繼續執行。
- 將一個 boolean 測試放在括弧（parentheses）之內。
  `while (x == 4) { }`

條件分支

在 Java 中，*if* 測試基本上與 *while* 迴圈中的 boolean 測試相同，只不過不是說「***while*** there's still chocolate（**當**還有巧克力的時候）」，而是說「***if*** there's still chocolate...（**如果**還有巧克力…）」。

```
class IfTest {
  public static void main (String[] args) {
    int x = 3;
    if (x == 3) {
      System.out.println("x must be 3");
    }
    System.out.println("This runs no matter what");
  }
}
```

程式碼輸出

```
% java IfTest
x must be 3
This runs no matter what
```

只有當條件（*x* 等於 3）為真時，前面的程式碼才會執行印出「x must be 3」的那一行。不過，無論它是否為真，列印「This runs no matter what」的那一行都將被執行。因此，取決於 *x* 的值，可能會有一個或兩個述句被印出來。

但我們可以在條件中加入一個 *else*，這樣我們就能說：「*If*（如果）還有巧克力，就繼續寫程式，*else*（否則）就去找更多的巧克力，然後從這裡開始繼續…」。

```
class IfTest2 {
  public static void main(String[] args) {
    int x = 2;
    if (x == 3) {
      System.out.println("x must be 3");
    } else {
      System.out.println("x is NOT 3");
    }
    System.out.println("This runs no matter what");
  }
}
```

新的輸出

```
% java IfTest2
x is NOT 3
This runs no matter what
```

System.out.print vs. System.out.println

如果你一直很專心（你當然有），那麼你會注意到我們在 **print** 和 **println** 之間切換。

你發現區別了嗎？

System.out.***println*** 會插入一個 newline（把 print***ln*** 看作 **printnewline**），而 System.out.***print*** 會持續列印到相同的一行。如果你希望印出來的每樣東西都在自己的行上，就用 print**ln**。如果你想讓所有東西都集中在一行，就用 print。

削尖你的鉛筆

給定輸出：

```
% java DooBee
DooBeeDooBeeDo
```

請填入缺少的程式碼：

```
public class DooBee {
  public static void main(String[] args) {
    int x = 1;
    while (x < _____ ) {
      System.out._________("Doo");
      System.out._________("Bee");
      x = x + 1;
    }
    if (x == ______  ) {
      System.out.print("Do");
    }
  }
}
```

答案在第 25 頁。

編寫一個認真的商業應用程式

讓我們善用你所有的 Java 新技能，發揮一些實際的用處。我們需要帶有 *main()* 的一個類別，一個 *int* 和一個 *String* 變數，一個 *while* 迴圈，以及一個 *if* 測試。再做一點潤飾，你很快就能建立起商業後端。但在你看這一頁的程式碼之前，請先想一想你會如何編寫孩童最喜歡的那經典的「10 個綠瓶子（10 green bottles）」。

```
public class BottleSong {
  public static void main(String[] args) {
    int bottlesNum = 10;
    String word = "bottles";

    while (bottlesNum > 0) {

      if (bottlesNum == 1) {
        word = "bottle"; // 單數，就像「ONE bottle」中那樣
      }

      System.out.println(bottlesNum + " green " + word + ", hanging on the wall");
      System.out.println(bottlesNum + " green " + word + ", hanging on the wall");
      System.out.println("And if one green bottle should accidentally fall,");
      bottlesNum = bottlesNum - 1;

      if (bottlesNum > 0) {
        System.out.println("There'll be " + bottlesNum +
                           " green " + word + ", hanging on the wall");
      } else {
        System.out.println("There'll be no green bottles, hanging on the wall");
      } // 結束 else
    } // 結束 while 迴圈
  } // 結束 main 方法
} // end 類別
```

我們的程式碼中還有一個小缺陷。它可以編譯和執行，但輸出結果並不是 100% 完美。看看你是否能發現那個缺陷並加以修復。

問：**這不是以前的「99 Bottles of Beer（99 瓶啤酒）」嗎？**

答：是的，但 Trisha 希望我們使用這首歌的英國版本。如果你更喜歡 99 瓶的版本，那就把它當作一個有趣的練習吧。

週一早上在 Bob 有 Java 功能的房子裡

Bob 的鬧鐘在週一早上 8:30 響起，就像其他工作日一樣。但 Bob 度過了一個瘋狂的週末，所以他伸手去按 SNOOZE（貪睡）鈕。就在這時，行動開始了，支援 Java 的電器開始運作…

首先，鬧鐘向咖啡機發送一個訊息：「嘿，這個技客（geek）又睡回頭覺了，把咖啡推遲 12 分鐘」。

咖啡機向 Motorola ™ 烤麵包機發送一個訊息：「吐司待會再烤，Bob 又睡著了」。

然後，鬧鐘向 Bob 的 Android 手機發送一條訊息：「9 點打電話給 Bob，告訴他時間有點晚了」。

最後，鬧鐘向 Sam（Sam 是條狗）的無線項圈發送訊息，其中含有一個再熟悉不過的訊號，意思是「去拿報紙，但別指望能散步」。

幾分鐘後，鬧鈴再次響起。Bob 又一次按下了 SNOOZE 鈕，而那些家電開始咯咯作響。最後，鬧鐘第三次響起。但就在 Bob 伸手去按貪睡鈕時，時鐘向 Sam 的項圈發出了「跳過去吠叫」的信號。震驚到清醒之餘，Bob 站了起來，感謝他的 Java 技能和自發的網路購物，提升了他日常生活的品質。

他的吐司烤好了。

他的咖啡冒著熱氣。

他的報紙在等著他。

在具備 *Java* 功能的房子裡，又是一個美妙的早晨。

這個故事可能是真的嗎？大部分是的！在包括手機（特別是手機）、ATM、信用卡、家庭安全系統、停車計時器、遊戲機等裝置中都有某種版本的 Java 在運行，但你可能還找不到 Java 狗項圈就是了…

Java 有多種方式可以僅使用 Java 平台的一小部分，以在較小型的裝置上執行（取決於你所用的 Java 版本）。它在 IoT（Internet of Things，物聯網）開發中非常流行。當然，很多的 Android 開發工作是用 Java 和 JVM 語言來完成的。

好吧，所以瓶子歌並不是真正嚴肅的商業應用程式。還需要一些實用的東西來向老闆展示嗎？看看 Phrase-O-Matic 程式碼。

注意：當你在編輯器中輸入這些內容時，讓程式碼自己進行字詞或文字行的繞行動作！當你輸入一個 String（介於 "" 引號之間的東西）時，千萬不要按下 return 鍵，否則它將無法編譯。因此，你在本頁看到的連字號是真實的，你可以輸入它們，但請在你封閉一個字串「之後」再按 return 鍵。

```
public class PhraseOMatic {
  public static void main (String[] args) {

1     // 製作要從中挑選的三組字詞。加上你自己的！
      String[] wordListOne = {"agnostic", "opinionated",
"voice activated", "haptically driven", "extensible",
"reactive", "agent based", "functional", "AI enabled",
"strongly typed"};

      String[] wordListTwo = {"loosely coupled", "six sigma",
"asynchronous", "event driven", "pub-sub", "IoT", "cloud
native", "service oriented", "containerized", "serverless",
"microservices", "distributed ledger"};

      String[] wordListThree = {"framework", "library",
"DSL", "REST API", "repository", "pipeline", "service
mesh", "architecture", "perspective", "design",
"orientation"};

      // 找出每個串列中有多少字詞
2     int oneLength = wordListOne.length;
      int twoLength = wordListTwo.length;
      int threeLength = wordListThree.length;

      // 產生三個隨機數字
3     java.util.Random randomGenerator = new java.util.Random();
      int rand1 = randomGenerator.nextInt(oneLength);
      int rand2 = randomGenerator.nextInt(twoLength);
      int rand3 = randomGenerator.nextInt(threeLength);

      // 現在建置出一個片語（phrase）
4     String phrase = wordListOne[rand1] + " " +
wordListTwo[rand2] + " " + wordListThree[rand3];

      // 印出該片語
5     System.out.println("What we need is a " + phrase);
  }
}
```

Phrase-O-Matic 片語產生器

它的運作方式

簡而言之，該程式列出了三個字詞串列（lists of words），然後從這三個串列中各隨機抽取一個字詞，並列印出結果。如果你不完全明白每一行到底發生了什麼事，請不要擔心。看在老天的份上，有整本書等著你呢，所以請別緊張。這只是從三萬英尺高空外鎖定目標的快速觀察而已。

1. 第一步是建立三個 String 陣列，作為容納所有字詞的容器。

宣告和創建一個陣列很容易，這裡有一個小陣列：

```
String[] pets = {"Fido", "Zeus", "Bin"};
```

每個字詞都在引號中（所有好的字串都必須如此），並用逗號分隔。

2. 對於三個串列（陣列）中的每一個，我們的目標是挑選一個隨機字詞，所以我們必須知道每個串列中有多少個字詞。如果一個串列中有 14 個字詞，那麼我們就需要介於 0 到 13 之間的一個亂數（Java 陣列是從零起算，所以第一個詞在 0 的位置，第二個詞在 1 的位置，而最後一個詞則是在 14 元素陣列的位置 13）。相當方便的是，Java 陣列非常樂意告訴你它的長度。你只需要問一下。在寵物陣列（pets array）中，我們會說：

```
int x = pets.length;
```

而 **x** 現在持有的值為 3。

3. 我們需要三個隨機數字。Java 內建就有幾種生成亂數的方法，包括 java.util.Random（我們將在後面看到為什麼這個類別的名稱前綴是 java.util）。**nextInt()** 方法回傳介於 0 和我們給它的某個數字之間的一個亂數，不包括我們給它的數字。所以我們要給它正在使用的串列的元素數（陣列長度）。然後我們把每個結果指定給一個新的變數。我們也可以很輕易地要求介於 0 到 5 之間的一個亂數，不包括 5：

```
int x = randomGenerator.nextInt(5);
```

4. 現在我們要建立這個片語，從三個串列中各選一個字詞，然後把它們揉合在一起（在詞與詞之間也插入空格）。我們使用「**+**」運算子，將 String 物件串接（*concatenates*）在一起（我們偏好使用用更專業的術語 *smooshes*）。要從一個陣列獲得一個元素，你能用以下方式為陣列提供你想要的東西之索引號碼（位置）：

```
String s = pets[0]; // s 現在是字串 "Fido"
s = s + " " + "is a dog"; // s 現在是 "Fido is a dog"
```

5. 最後，我們將這段片語列印到命令列中，然後…瞧！我們可以去做行銷了。

這裡我們需要的是一個…

可擴充的
微服務管線

充滿主張的鬆散耦合
REST API

基於代理人的
微服務程式庫

具備 AI 功能的
服務導向方向

不可知論的
pub-sub DSL

函式型的 IoT
觀點

程式碼磁貼

一個可運作的 Java 程式被打亂分散在冰箱上。你能重新排列這些程式碼片段，製作出一個會產生下面所列輸出的可運作的 Java 程式嗎？有些大括號掉在地上，它們太細小了，很難撿起，所以請隨意添加你需要的程式碼！

```
if (x == 1) {
   System.out.print("d");
   x = x - 1;
}
```

```
if (x == 2) {
   System.out.print("b c");
}
```

```
class Shuffle1 {
  public static void main(String [] args) {
```

```
if (x > 2) {
   System.out.print("a");
}
```

```
int x = 3;
```

```
x = x - 1;
System.out.print("-");
```

```
while (x > 0) {
```

輸出：

```
File  Edit  Window  Help  Sleep
% java Shuffle1
a-b c-d
```

答案在第 25 頁。