目錄

1. 導論

1.1. 爲甚麼要有程式碼慣例

對程式設計師來說，程式碼慣例很重要的原因有下列幾點：

- 一套軟體的生命期 80%都是花費在維護上。
- 任何軟體都很難從頭到尾全由原來的作者來維護。
- 程式碼慣例改善軟體的可讀性，讓工程師可以更快速完整的了解新程式碼。
- 如果你將你的原始碼當作產品，你必須確定它跟你創造的其他產品包裝的一樣好，一樣乾淨。

爲了方便工作，每一個寫軟體的人都必須遵循程式碼慣例。每一個人。

1.2. 致謝

這份文件反映出在 Sun MicroSystem, Inc 的 Java Language Specification 中的 Java 語言程式寫作標準。主要的貢獻者是 Peter King, Patrick Naughton, Mike DeMoney, Jonni Kanerva, Kathy Walcath 和 Scott Hommel.

這份文件是由 Scott Hommel 維護。有建議請寄到 shommel@eng.sun.com

2. 檔案名稱

這一節列出普遍使用的檔案副檔名和名稱。

2.1. 副檔名

Java 軟體使用下列的副檔名：

檔案型態	副檔名
Java 原始碼	`.java`
Java bytecode	`.class`

2.2. 一般檔案名稱

常用檔案名稱包括：

檔案名稱	使用
`GNUmakefile`	makefiles 最好的名稱。我們使用 gnumake 來建立我們的軟體。
`README`	特定目錄摘要文件的最佳名稱。

3. 檔案組織

由許多小節組成的檔案應該用"空白行"以及"用來辨認每個小節的選擇性註解"分隔。

超過 2000 行的檔案是很累贅的，應該避免。

Java 程式適當格式的範例請參考 11.1 的 "Java 原始碼檔案範例"。

3.1. Java 原始碼檔案

每一個 Java 原始碼檔案都包含了一個單一的 public class/interface。如果 private classes/interfaces 和 public class 相關，那你應該把他們跟 public class 原始碼檔案放在一起。public 應該是這個檔案中的第一個 calss/interface。

Java 原始碼檔案有下列順序：

- 起始註解(參見 3.1.1 "起始註解")
- package 和 import 敘述
- class 和 interface 宣告(參見 3.1.3 "class 和 interface 宣告")

3.1.1. 起始註解

所有的原始碼檔案都應該以 C 語言型態的註解起始，列出類別名稱，版本資訊，日期和版權宣告。

```
/*
 * 類別名稱
 *
 * 版本資訊
 *
 * 日期
 *
 * 版權宣告
 */
```

3.1.2. package 和 import 敘述

大多數 Java 原始碼檔案的第一行非註解行都是 `package` 敘述。在這之後跟著 `import` 敘述。例如：

```
package java.awt;

import java.awt.peer.CanvasPeer;
```

3.1.3. class 和 interface 宣告

下列表格描述了一個 class/interface 的宣告部份，以他們應該出現的順序排列。參考 11.1 "Java 原始碼檔案範例"來看包含註解的範例。

	class/interface 宣告部份	備註
1	class/interface 文件註解 (/**...*/)	參考 5.2 "文件註解"來查看這註解中應該有甚麼資訊。
2	class/interface 敘述	
3	class/interface 實作註解 (/*...*/)，如果需要的話	這註解應該包含任何不適合做爲 class/interface 文件註解的 class/interface 廣度的資訊。
4	類別(static)變數	首先是 public 的類別變數，然後是 protected，再接著是 package 層級(沒有存取修正子)，最後是 private。
5	實體變數	首先是 public，然後 protected，然後 package(沒有存取修正子)，最後是 private。
6	建構子	
7	方法	方法應該以功能來分群，而不是以視野或是存取權限分。例如 private 的類別方法可以在兩個 public 實體方法中間。目的是爲了讓程式碼更容易閱讀和了解。

4. 縮排

應該以四個空白來當一個縮排的單位。縮排的確切構成物(空白 vs. Tab)並沒有特別指定。Tabs 必須設定成每 8 個空白(不是 4 個)。

4.1. 行長度

避免行長度超過 80 字元，因爲很多終端機和工具無法將這個處理的很好。

注意：用在文件上的範例應該有更短的行長度 -- 通常不會超過 70 個字元。

4.2. 換行

當表示式無法放入一個單行時，根據下列的一般原則斷行：

- 在逗號後面斷行。
- 在操作子前面斷行。
  (譯註：操作子指運算符號，如 + & = 等等)
- 較高階的斷行比較低階的斷行好。
  (譯註：也就是斷行後，第二行縮排比較少，越靠近上一行起始位置的越好)
- 將新一行的表示式起始位置排列在和前一行相同的階層。
- 如果上述的規則造成令人困擾的程式碼或是破壞了右側邊界，那就用 8 個空白代替。

這裡有一些斷行的方法呼叫的範例：

```
someMethod(longExpression1, longExpression2, longExpression3,
          longExpression4, longExpression5);

var = someMethod1(longExpression1,
                  someMethod2(longExpression2,
                              longExpression3));
```

下列兩個範例切斷一個算術表示式。第一個比較好，因爲它在括號表示式外側切斷，這個切斷會比較高階。

```
longName1 = longName2 * (longName3 + longName4 - longName5)
            + 4 * longname6; //較佳

longName1 = longName2 * (longName3 + longName4
                         - longName5 + 4 * longname6 ; //避免
```

下列是兩個是將方法宣告縮排的範例。第一個是慣例的狀況。第二個如果他用慣例縮排來處理，會將第二和第三行轉的太過於右側，所以用只有 8 個空白來代替。

```
//慣例縮排
someMethod(int anArg, Object anotherArg,String yetAnoterArg,
           Object andStillAnother){
       ...
}

//以 8 個空白縮排來避免太深的縮排
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother){
       ...
}
```

一般應該用 8 個空白的規則來換行 `if` 敘述，因爲慣例(4 個空白)縮排會造成主體不容易看見。例如：

```
// 不要使用這種縮排
if (( condition1 && condition2 )
    || ( condition3 && condition4)
    ||!( condition5 && condition6)) { //不好的換行
   doSomethingAboutIt();              //讓這一行容易被錯過
}

// 使用這種縮排來代替
```

```
if (( condition1 && condition2 )
          || ( condition3 && condition4)
          || !( condition5 && condition6)) {
    doSomethingAboutIt();
}

// 或是用這個
if (( condition1 && condition2 ) || ( condition3 && condition4)
          || !( condition5 && condition6)) {
    doSomethingAboutIt();
}
```

這邊有三種可以接受的方法來格式化三元運算表示式：

```
alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

alpha = (aLongBooleanExpression)
        ? beta
        : gamma;
```

5. 註解

Java 程式可以有兩種註解：實作註解和文件註解。實作註解是可以在 C++中看的到，以/*...*/和//分隔的。文件註解(又稱爲"doc 註解")是只有 Java 有，而且是用/**...*/分隔的。doc 註解可以被用 javadoc 工具萃取成 HTML 檔案。

實作註解是用來將程式碼註解掉或是說關於特定實作。doc 註解是要描述程式碼的規格，必須以要讓一個手上沒有原始碼的開發者閱讀的實作不相關的觀點來寫。

註解應該用來給予程式碼概觀，提供閱讀程式碼本身無法獲得的額外資訊。註解應該只包含與閱讀和了解這程式有關的資訊。例如這一致的套件是如何被建立，或是它存在於甚麼目錄都不應該被包含在註解中。

對不瑣碎和無法觀察到的設計決定的討論是適當的，但是避免在程式碼中出現重複的資訊。冗長的註解太容易過期了。通常，避免任何程式碼改進後好像會過期的註解。

注意：註解的頻率有時候反映出貧乏的程式碼品質。當你覺得必須強迫加入註解時，請考慮重寫程式碼來讓它更清楚。

註解不應該被用星號或其他字元畫成的大格子包起來。
註解不能包含特殊字元，如 form-feed 和 backspace。

5.1. 實作註解格式

程式可以有四種實作註解的型態：區塊，單行，尾隨，行結尾。

5.1.1. 區塊註解

區塊註解用來提供檔案，方法，資料結構和演算法的描述。區塊註解可以用在每一個檔案起始處，或是在每個方法前面。它們也可以用在其他的位置，如在方法內部。在函式或方法內的區塊註解應該縮排到和他們描述的碼相同的層級。

區塊註解應該在前面有一行空白行，來將它與程式碼結構分離。

```
/*
 * 這是區塊註解。
 */
```

區塊註解可以用/*-開始，這個在區塊註解起始不能重定格式時，會被當成是縮排(1)。例如：

```
/*-
 * 這裡是含有我想要縮排(1)去忽略的一些非常特別的格式。
 *
 *    一
 *        二
 *            三
 */
```

注意：如果你不使用縮排(1)，那你不需要在你的程式碼中使用/*-或是做其它讓步來讓某些其它人可以在你的程式碼執行縮排(1)。

參考 5.2 "文件註解"。

5.1.2. 單行註解

簡短的註解可以以縮排到和隨後的碼同樣層級的單行模式出現。如果註解無法寫成單行，那他應該遵循區塊註解格式(參見 5.1.1 小節)。單行註解前面應該有一空白行。這是 Java 程式碼的單行註解範例：

```
if (condition) {
    /* 處理這個狀況。 */
    ...
}
```

5.1.3. 尾隨註解

非常短的註解可以在他們描述的程式碼同一行出現，但是應該離開足夠的距離來讓他們與敘述分

隔。如果有超過一個短註解在大區塊程式碼中出現，那他們應該被縮排到同一個 tab 設定。

這裡是一個 Java 程式碼中的尾隨註解範例：

```
if (a == 2) {
    return TRUE;              /* 特別狀況  */
} else {
    return isPrime(a);        /* 只在 a 爲奇數時有作用 */
}
```

5.1.4. 行結尾註解

這//的註解分隔子可以註解一整行或是行的一部分。他不能用在連貫的多行文字註解中；但是他可以用在連貫的多行中來註解掉一區塊的程式碼。下列是這三種形式的範例：

```
if ( foo > 1) {

    // 做出雙空翻。
    ...
}
else {
    return false;  // 在這裡解釋爲甚麼。
}

// if (bar > 1) {
//
//     // 做三重空翻。
//     ...
//}
//else{
//    return false;
//}
```

5.2. 文件註解

注意：參考 11.1 "Java 原始碼檔案範例"來看這邊描述的註解格式範例。

進一步的細節請看包含 doc 註解標籤(@return, @param, @see) 資訊的"How to Write Doc Comments for JavaDoc"：

```
http://java.sun.com/products/jdk/javadoc/writingdoccomments.html
```

進一步的 doc 註解和 javadoc 細節，請看 javadoc 首頁：

```
http://java.sun.com/products/jdk/javadoc/
```

doc 註解描述了 Java 類別,介面,建構子,方法,和欄位。每一個 doc 註解都設定成放在/**...*/分隔子中，每一個類別，介面或是成員都有一個註解。這些註解應該就在在宣告的前面出現。

```
/**
 * 這個 Example 類別提供了 ...
 */
public class Exampke { ...
```

注意頂層的類別和介面並沒有縮排，而他們的成員有。這類別和介面的 doc 註解的第一行(/**)是沒有縮排的；隨後的 doc 註解行都有 1 個縮排的空格(垂直對齊星號)。成員(包括建構子) 的第一個 doc 註解行則有 4 個空白，以及之後的有五個空白。

如果你需要給予不適合出現在文件中的關於類別，介面，變數，或是方法的資訊，使用馬上接在宣告之後的區塊註解(見 5.1.1 小節)或是單行註解(見 5.1.2 小節)來實作。例如，關於類別實作的細節應該在跟著類別敘述的區塊註解實作中，而不是在類別的 doc 註解中。

doc 註解不應該被放在方法或是建構子的定義區塊中,因爲 Java 會將文件註解和註解後的第一個宣告關聯起來。

6. 宣告

6.1. 每一行的數目

建議每一行一個宣告，因爲他可以支援註解。換句話說，

```
int level;  // 縮排層級
int size;   // 表格大小
```

比下列好

```
int level, size;
```

不要將不同的型態放在同一行，例如：

```
int foo,  fooarray[];    //錯誤
```

注意：上面的範例在型態和變數名稱之間使用一個空白。另一個可以接受的替代方法是用 Tabs，例如：

```
int     level;          // 縮排層級
int     size;           // 表格大小
Object  currentEntry;   // 目前選擇的表格項目
```

6.2. 初始化

試著在區域變數宣告的地方初始化它們。唯一不在他們宣告的地方初始化它們的理由是初始值要

先根據某些計算才能獲得。

6.3. 佈置

只將宣告放在區塊開始的地方。(區塊是被大括號 "{" 和 "}" 圍起來的任何程式碼。)不要等到它們第一次被使用時才宣告變數,這可能會困擾不夠小心的程式設計師並阻礙程式碼在這個視野中的移植性。

```
void myMethod() {
  int int1 = 0;          // 方法區塊的起始位置

  if (條件) {
     int int2 = 0;       // "if" 區塊的起始位置
     ...
  }
}
```

這個規則的例外是 for 迴圈的索引值，Java 可以在 for 的敘述中宣告：

```
for ( int i = 0; i < maxLoops; i++) { ... }
```

避免將宣告藏在高層級的區域宣告。例如，不要在一個內部區塊中宣告同名的變數名稱：

```
int count;
...
myMethod() {
   if (條件) {
      int count;       // 避免
      ...
   }
   ...
}
```

6.4. 類別和介面宣告

當寫 Java 類別和介面程式碼時，應該遵循下列格式規則：

- 不要在方法名稱和它的引數起始的小括號"("之間使用空白
- 左大括號 "{" 放在宣告敘述同一行的末端
- 右大括號 "}" 自己放在一行，縮排到符合他的左括號敘述的位置，除了空敘述的 "}" 應該緊接著 "{" 之後出現之外。

```
class Sample extends Object {
     int ivar1;
```

```
        int ivar2;

        Sample(int i, int j) {
            ivar1 = i;
            ivar2 = j;
        }

        int emptyMethod() {}
        ...
    }
```

- 方法以一行空白行隔開

7. 敘述

7.1. 簡單敘述

每一行應該最多都只有一個敘述。例如：

```
argv++;          // 正確
argc++;          // 正確
argv++; argc--; // 避免!
```

7.2. 複合敘述

複合敘述是被括號 "{ 敘述 }" 包住的一堆敘述。參考下一小節中的例子。

- 被包住的敘述應該比複合敘述還多縮排一個層級。
- 左大括號應該在複合敘述開始那行的結尾；右大括號應該自成一行，而且應該縮排到複合敘述起始的位置。
- 所有的敘述外面都要有大括號，即使在它們是如 `if-else` 或是 `for` 敘述的控制架構的單行敘述時也是一樣。這會讓要加上敘述時會更容易，而不會因爲忘了加上大括號而導致意料之外的錯誤。

7.3. return 敘述

一個帶著值的 `return` 敘述不應該使用小括號，除非它們以某種方式讓傳回的值更加的明顯。例如：

```
return;
return myDisk.size();
return ( size ? size : defaultSize );
```

7.4. if, if-else, if else-if else 敘述

這 if-else 類的敘述應該有下列形式：

```
if (條件) {
    敘述;
}

if (條件) {
    敘述;
} else {
    敘述;
}

if (條件) {
    敘述;
} else if (條件) {
    敘述;
} else {
    敘述;
}
```

注意：總是在 if 敘述上使用大括號{}。避免下列有錯誤傾向的形式：

```
if (條件)  // 避免!這個忽略了大括號{}!
    敘述;
```

7.5. for 敘述

for 敘述應該有下列形式：

```
for ( 初始化; 條件; 更新 ) {
      敘述;
}
```

而空的 for 敘述(所有的工作都在初始化，條件式，和更新的子句中完成的)應該有下列形式：

```
for ( 初始化; 條件; 更新);
```

當在初始化或更新子句的 for 敘述中使用逗號操作子時，應該避免超過三個變數的複雜使用。如果有需要，在 for 迴圈之前(對初始化子句) 或是在迴圈的結尾(對更新子句)使用分開的敘述。

7.6. while 敘述

while 敘述應該有下列的形式：

```
while (條件) {
    敘述;
}
```

空的 while 敘述應該有下列形式：

```
while (條件);
```

7.7. do-while 敘述

do-while 敘述應該有下列形式：

```
do {
    敘述;
} while (條件);
```

7.8. switch 敘述

switch 敘述應該有下列形式：

```
switch (條件) {
case ABC:
    敘述;
    /* 往下穿過 */
case DEF:
    敘述;
    break;

case XYZ:
    敘述;
    break;

default:
    敘述;
    break;
}
```

每次有往下穿過的 case 時(不包含 break 敘述者)，在一般會有敘述的地方增加一個註解。這個以上面程式範例中的/* 往下穿過 */註解來代表。

每一個 switch 敘述都應該包含 default case。這 default case 的 break 是多餘的，但是它可以防止如果之後加入了另一個 case 時，可能會造成的往下穿過的錯誤。

7.9. try-catch 敘述

try-catch 敘述應該有下列形式：

```
try {
    敘述;
} catch {ExceptionClass e} {
    敘述;
}
```

try-catch 敘述也可能跟著不管 try 區塊有沒有完全成功都會執行的 finally 敘述而有如下列所示的形式。

```
try {
    敘述;
} catch (ExceptionClass e) {
    敘述;
} finally {
    敘述;
}
```

8. 空白

8.1. 空白行

空白行利用區隔出邏輯上相關的程式碼區段來改善可讀性。

雙行的空白應該在下列的狀況中使用：

- 在原始碼檔案的小節之間
- 在類別和介面的定義之間

單行的空白應該在下列狀況中使用：

- 在方法之間
- 在方法中的區域變數和它的第一條敘述之間
- 在區塊註解(見 5.1.1 小節)或是單行註解(見 5.1.2 小節)之前
- 在方法的邏輯小節之間來改善可讀性

8.2. 空白

空白應該用在下列狀況中：

- 後面接著小括號的關鍵字應該以一個空白分開。例如：

```
while (true) {
    ...
```

```
}
```

注意空白不應該用在方法名稱和它的左小括號之間。這會幫助你分辨關鍵字和方法呼叫。

- 在引數列表中的逗號後面應該出現一個空白。
- 所有除了 . (dot operation)之外的所有二元操作子應該用空白和它們的運算元分隔。空白不應該分隔 unary 運算子和它的運算元，如 unary minus, 遞增("++")和遞減("==")等。例如：

```
a += c + d;
a = (a + b) / (c * d);

while (d++ = s++) {
    n++;
}
prints("size is " + foo + "\n");
```

- 在 for 敘述中的表示式應該以空白分隔。例如：

```
for (expr1; expr2; expr3)
```

- 轉型應該接著一個空白。例如：

```
myMethod((byte) aNum, (Object) x);
myMethod((int) (cp + 5), ((int) (i + 3))
                         + 1);
```

9. 命名慣例

命名慣例讓程式更容易被理解和閱讀。他們也可以給予關於識別子功能的資訊 -- 例如，是否爲常數，套件，或是類別 -- 這可以在瞭解程式碼時很有幫助。

變數型態	命名規則	範例
套件	獨一無二的套件名稱的前置字都是使用所有都是小寫的 ASCII 字母而且應該是頂層 domain name 名稱，目前有 com, edu, gov, mil, net, org 或是 1981 年 ISO 標準 3166 中用來辨識國家的兩字母碼。 接下來的套件名稱元件根據組織擁有的內部命名慣例而不同。這種慣例會指定某種被分隔的目錄名稱元件，部門，計畫，機器，或是登入名稱。	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese

變數型態	命名規則	範例
類別	類別名稱應該是名詞，由每個內部單字開頭字母皆爲大寫的混和字組成。試著讓你的類別名稱是個很簡單的敘述。使用整個字 -- 避免用頭字語或是縮寫 (除非縮寫比長字的形式被用的更廣泛。例如 URL 或 HTML)。	`class Raster;` `class ImageSprite;`
介面	介面名稱應該像類別名稱的大寫法一樣。	`interface RasterDelegate;` `interface Storing;`

方法	方法應該爲動詞，混和第一個字母小寫和內部單字的第一個字母大寫的狀況。	`run();`
		`runFast();`
		`getBackground();`

變數	除了變數之外，所有的實體，類別，以及類別常數都是第一個字母爲小寫的混和狀況。內部單字則以大寫開頭。變數字不應以底就算這兩個都是被 Java 允許的。
	
	變數名稱應該是短但仍有意義的。這變數名稱的選擇應該是幫助記憶的 -- 也就是說設計來讓不經意的觀察者都知道它使用的意圖。除了暫時使用後就被"丟開"的變數名稱之外，應該避免使用單字元的變數。普遍使用的暫時變數名稱爲用在整數的 i, j, k, m, 和 n 以及用在字元的 e。
		`int  i;`
		`char  c;`
		`float  myWidth;`

常數	宣告類別常數的變數名稱以及 ANSI 常數的變數名稱應該是以 ("_") 底線分隔的全大寫字。(爲了易於除錯，應該避免 ANSI 常數。)	`static final int MIN_WIDTH = 4;`
		`static final int MAX_WIDTH = 999;`
		`static final int GET_THE_CPU = 1;`

10.程式習慣

10.1. 提供實體變數和類別變數的權限

沒有好理由的話，不要讓任何的實體變數或是類別變數是 public 的。通常，實體變數不需要被直接的設定或是獲得 -- 通常是作爲方法呼叫的邊際效應。

一個適當的 public 實體變數的例子是當這個類別是一個沒有 method 的基本的資料結構時。換句話說，如果你可能會使用 struct 來代替類別的話(如果 Java 支援 struct 的話) ，那讓類別的實體變數是 public 就是適當的。

10.2. 引用類別變數和方法

避免使用物件來存取類別(static)變數和方法。使用類別名稱來代替。例如：

```
classMethod();           // OK
AClass.classMethod();        // OK
anObject.classMethod(); // 避免!
```

10.3. 常數

會出現在 for 迴圈中當作計數值的數值常數(直譯字)，除了-1，0，和 1 之外，不應該直接編寫。

10.4. 變數指定

避免在單一敘述中指定好幾個變數爲同一個值。那會不容易閱讀。例如：

```
fooBar.fChar = barFoo.lchar = 'c';  // 避免!
```

不要在可能很容易會跟相等運算子搞混的地方使用指定運算子。例如：

```
if (c++ = d++) {        // 避免！ (Java 不允許)
    ...
}
```

應該寫成

```
if ((c++ = d++) != 0) {
    ...
}
```

不要爲了改善執行效率而使用嵌入式的指定。這是編譯器的工作。例如：

```
d = ( a = b + c) + r;      // 避免!
```

應該寫成

```
a = b + c;
```

```
d = a + r;
```

10.5. 雜項慣例

10.5.1. 小括號

在表示式中包含了混和的操作子時，使用大量的小括號來避免優先權的問題是一個不錯的作法。即使操作子的優先權對你來說看來很清楚，對別人來說可能不是如此 -- 你不應該假設其他的程式設計師對優先權的瞭解跟你一樣好。

```
if (a == b && c == d)        // 避免!
if ((a == b) && (c == d))    // 使用
```

10.5.2. 傳回值

試著讓你程式的架構符合意圖。例如：

```
if (booleanExpression) {
    return true;
} else {
    return false;
}
```

應該以下列取代

```
return booleanExpression;
```

相同的，

```
if (條件) {
    return x;
}
return y;
```

應該寫成這樣

```
return (條件 ? x : y);
```

10.5.3. 在條件運算子中 "?" 之前的表示式

如果在「?:」運算子的?之前出現的表示式中包含了二元運算子，那它應該被小括號刮起來。例如：

```
(x >= 0) ? x : -x;
```

10.5.4. 特別註解

在註解中使用 XXX 來標出一些假的但是仍有用的東西。使用 FIXME 來標示出一些有問題而且壞掉的東西。

11.程式碼範例

11.1. Java 原始碼範例

下列範例顯示了如何格式化一個包含單一 public 類別的 Java 原始碼檔案。介面也以相似的方法格式化。更多的資訊請參閱 3.1.3 小節的"class 和 interface 宣告"和 5.2 小節的"文件註解"。

```
/*
 * @(#)Blah.java       1.82 99/03/18
 *
 * Copyright (c) 1994-1999 Sun Microsystems, Inc.
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of Sun
 * Microsystems, Inc. ("Confidential Information").  You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Sun.
 */

package java.blah;

import java.blah.blahdy.BlahBlah;

/**
 * 類別描述在這裡。
 *
 * @version  1.82 18 Mar 1999
 * @author  姓名
 */
public class Blah extends SomeClass {
    /* 類別實作註解在此。 */

    /** classVar1 文件註解 */
    public static int classVar1;
```

```
    /**
     * 會超過一行長度的 classVar2
     * 文件註解
     */
    private static Object classVar2;

    /** instanceVar1 文件註解 */
    public Object instanceVar1;

    /** instanceVar2 文件註解 */
    protected int instanceVar2;

    /** instanceVar3 文件註解 */
    private Object[] instanceVar3;

    /**
     * ...Blah 建構子文件註解...
     */
    public Blah() {
        // ...實作在此...
    }

    /**
     * ...doSomething 方法文件註解...
     */
    public void doSomething() {
        // ...實作在此...
    }

    /**
     * ...doSomethingElse 方法文件註解...
     * @param someParam 描述
     */
    public void doSomethingElse(Object someParam) {
        // ...實作在此...
    }
}
```