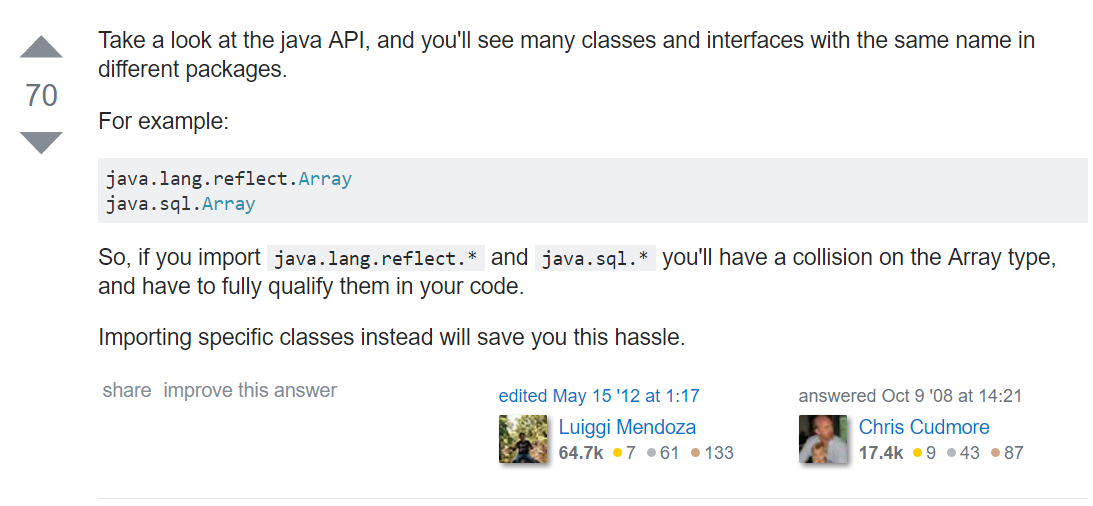
**Reference:** [**莫希爾(Mosil)手札**](http://blog.mosil.biz/2014/05/java-style-guide/)

**3.3.1 Import 不用使用萬用符號 (No wildcard imports)**

Import 敘述區裡的每一個 import，**皆不要**使用萬用符號 (\*) 或是 static 這類的修飾子。

**譯注：**像是下方這樣的 import 語句都不要使用  
萬用符號：import java.util.\*  
靜能修飾子：import static java.util.Arrays.sort;

**REASON**:



**3.3.2 不換行 (No line-wrapping)**

Import 敘述區裡的每一個 import 不換行 (not line-wrapped)。 意即每一行的最大字數限制 (請見 4.4 每行字數上限) 不適用於此。

## 4.4 每列字數限制：80 或 100 (Column limit: 80 or 100)

**我們是80！**

**4.5.2 每行縮排至少要有一組( 4 個)空白 (Indent continuation lines at least +4 spaces)**

**請確定各位TAB為4個空白**

每個接續在斷行後的第一行開始，都要比原行加上一組( 4 個)以上的空白來縮排。

當有連續多行時，其縮排可能會超過一組( 4 個)空白。一般來說，若是連續的兩行都是斷於同級別語法元素時，其縮排階層會是相同的。

#### **4.8.3.2 宣告方式不要用 C 語言風格 (No C-style array declarations)**

中括號是型別的一部份，而非變數的一部份： Srting[] args 而不是 String args[] 。

#### **4.8.4.3 default 一定要有** (The default case is present)

每個 switch 敘述的群組中都一定包含 default ，就算裡面沒有程式碼。

**5.2.1 Package Naming (Package names)**

Package 名稱全部小寫，連續單字直接寫在一起(不用下底線(\_))。例如， com.example.deepspace ，不要這樣 com.example.deepSpace 或com.example.deep\_space 。

**5.2.2 類別命名 (Class names)**

類別名稱採用大寫開始的駝峰命名法(UpperCamelCase)。

類別名稱為名詞或是名詞片語。例如， Character 或 ImmutableList，介面(interface)也使用名詞或是名詞片語(如：List)，但有時候會用形容詞或是形容詞片語取而代之(如：Readable)。

**5.2.3 函式命名 (Method names)**

函式名稱採用小寫開始的駝峰命名法(lowerCamelCase)。

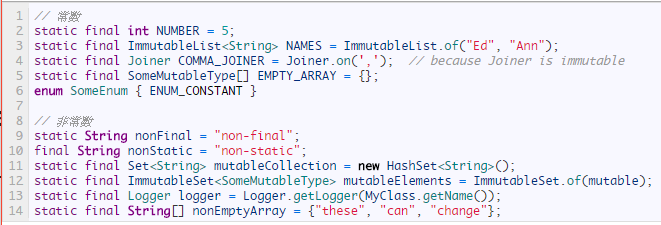
函式名稱通常都是動詞或是動詞片語。例如， sendMessage 或是stop 。

下底線可以做為 JUnit 的測試函式名稱與邏輯元件的分隔。一個典型的模式 test<MethodUnderTest>\_<state> ，範例：testPop\_emptyStack 。這邊並沒有唯一的正確方式去命名測試函式。

**5.2.4 常數命名 (Constant names)**

常數名稱採用 CONSTANT\_CASE ，全部採用大寫字母，使用下底線分隔。但究竟什麼是常數呢？

所以有常數的屬性 (Field) 皆是 static final， 但並非所有 static final 屬性的變數皆為常數。在確定變數是為常數前，需要先考慮他是否真的像一個常數。舉例來說，當所有在實作的觀察階段會改變時，那這個變數幾乎就可以肯定不是一個常數。而通常只是打算永不改變狀態是不夠的。

範例

**5.2.5 非常數屬性命名 (Non-constant field names)**

非常數屬性的名稱 (static 或是 其他)採用小寫開頭的駝峰命名法 (lowerCamelCase) 。

名稱通常為名詞或是名詞片語。

**5.2.6 參數命名 (Parameter names)**

參數名稱使用採用小寫開頭的駝峰命名法 (lowerCamelCase) 。

要避免只有一個字符的名稱。

**5.2.7 區域變數命名 (Local variable names)**

區域變數名稱採用採用小寫開頭的駝峰命名法 (lowerCamelCase) 。可以使用縮寫這種較為其他名稱寬鬆的命名方式。

但仍要避免單字符的名稱，除了泛型與迴圈變數。

即便是 final 或是不可變的，區域變數是不可為常數的，當然，也不該使用常數變數的風格。

**5.2.8 型別變數命名 (Type variable names)**

每個型別的命名方式可在下面二法中擇一：

* 一個大寫單字，其後用數字可接續(諸如： E 、 T 、 X 、T2 )。
* 可採用類別命名的方式(請見 5.2.2)，其後再接一個大寫字母T 。例如： RequestT 、 FooBarT 。

**5.3  駝峰命名定義 (Camel case: defined)**

英文詞彙有時並非只有一種合理的駝峰命名表示方式，也有像「IPv6」或是「iOS」這樣的縮寫或是不尋常的表示法。為了改善使其規律，Google Style 將使用下方(幾乎)確定的方法。

以散文格式 (prose form) 為名稱的開頭：

1. 字詞皆改為 ASCII 碼並移除所有單引號，例如，「Müller’s algorithm」可以改變為「Muellers algorithm」。
2. 上述步驟的結果，再依其中的空白以及其餘的符號(通為會連字符號)做為拆分點，拆成逐一的單字。
   * 建議：若所有字都已經有其慣用的駝峰命名用法，仍是將其拆開(例：「AdWords」變成「ad words」。注意，像「iOS」這個並不是一個駝峰命名的形式，這個建議就不適用於這樣的例子。
3. 現在，將每個字母全部變成小寫(包含縮寫)，接著，只要將第一個字母改為大寫：
   * … 每個單字都改，為大寫開頭的駝峰 (upper camel case) 命名。
   * … 每個單字除了第一個單字不改，則為小寫開頭的駝峰 (lower camel case) 命名。
4. 最後，將所有單字連成一個識別符。

需要注意的是，這邊的大小寫幾乎是已經無視原來的單字。範例：

* “XML HTTP request"
  + 正確：XmlHttpRequest
  + 錯誤：XMLHTTPRequest
* “new customer ID"
  + 正確：newCustomerId
  + 錯誤：newCustomerID
* “inner stopwatch"
  + 正確：innerStopwatch
  + 錯誤：innerStopWatch
* “supports IPv6 on iOS?"
  + 正確：supportsIpv6OnIos
  + 錯誤：supportsIPv6OnIOS
* “YouTube importer"
  + 正確：YouTubeImporter 或 YoutubeImporter\*
  + 錯誤：

\*允許， 但不建議。

**Note:**有些字在英語中，有無帶著連字符號都沒有錯，舉例來說「nonempty」以及「non-empty」二者皆對，所以方法若是命名成 checkNonempty 以及 checkNonEmpty 都是正確的。

# 6. 程式碼慣例(Programming Practices)

**6.1 @Override: always used (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s6.1-override-annotation)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s6.1_override_annotation)**)**  
**6.2 Caught exceptions: not ignored (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s6.2-caught-exceptions)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s6.2_caught_exceptions)**)**  
**6.3 Static members: qualified using class (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s6.3-static-members)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s6.3_static_members)**)**  
**6.4 Finalizers: not used (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s6.4-finalizers)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s6.4_finalizers)**)**

## 6.1 @Override：一定要使用 (always used)

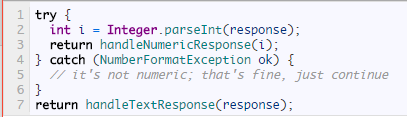
當一個被合法的標註了 @Override 的方法(method)，他一定是覆寫了其父類別 (superclass) 的方式、實作了介面方法(interface mthod) 以及介面中重新指定了其父介面 (superinterface) 的方法。

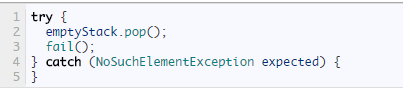
**例外： @Override 當可以被省略時，代表其父類別的方法已被標示為 @Deprecated**

## 6.2 (Caught exceptions: not ignored)

下面這個異常 (Except) 描述，他是少數在發生異常時，可以不做回應的異常處理。(以標準的異常回應，是需要記錄下來的，或若是被視為「不可能(impossible)」時，則重新用 AssertionError  拋出。

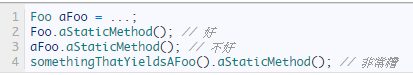
**當在 catch 區塊中，確實沒有任何動作的話，便用註解明確的說明其原委。**



例外：**在測試中，若其 catch 的異常被命名為 expected ，則其註解可以被省略**。下方這是一個常見的狀況，用以確保在測試時會拋出期望中的異常，所以這邊是不需要註解的。

6.3 **靜態成員：適當的搭配類別用 (Static members: qualified using class)**

引用靜態成員必需搭配著類別 (class) 名稱才是適當的用法，不是和一個物件類型或是描述句來使用。



6.4 Finalize 不要使用 (Finalizers: not used) Reference:

[Is overriding Object.finalize() really bad?](http://softwareengineering.stackexchange.com/questions/288715/is-overriding-object-finalize-really-bad)

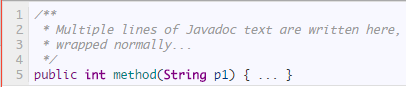
# 7. Javadoc

**7.1 Formatting (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s7.1-javadoc-formatting)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s7.1_formatting)**)**  
**7.2 The summary fragment (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s7.2-summary-fragment)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s7.2_the_summary_fragment)**)**  
**7.3 Where Javadoc is used (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s7.3-javadoc-where-required)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s7.3_where_javadoc_is_used)**)**

## 7.1 格式 (Formatting)

**7.1.1 General form (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s7.1.1-javadoc-multi-line)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s7.1.1_javadoc_multi_line)**)**  
**7.1.2 Paragraphs (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s7.1.2-javadoc-paragraphs)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s7.1.2_javadoc_paragraphs)**)**  
**7.1.3 At-clauses (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s7.1.3-javadoc-at-clauses)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s7.1.3_javadoc_at_clauses)**)**

### 7.1.1 通用格式 (General form)

基本的 Javadoc 區塊格式如下範例：

只有單行的範例：



這是個永遠可以被接受的基本格式。當整個 Javadoc (包含註解) 沒用使用到 at-子句時，若是可以被容量在一行的話，就採用單行的格式。

### 7.1.2 段落 (Paragraphs)

空行－在段落之間，以星號(\*) 為起始的一行空白；若是有「at-子句」，就要將之擺在這個群組之前。若是每個段落在一開始有 <p>時，他跟第一個字之間不會有空格。

### 7.1.3 At-子句 (At-clauses)

當全部的「at-子句 」都出現時，其標準的使用順序為 @param 、@return 、 @throws 、 @deprecated ，而這四種類型都不會為空。當一個「at-子句」無法以單行描述完畢時，其續行該要以 @ 為基準做四個(或更多)空白的縮排。

## 7.3 在哪裡使用 Javadoc (Where Javadoc is used)

**7.3.1 Exception: self-explanatory methods (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s7.3.1-javadoc-exception-self-explanatory)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s7.3.1_javadoc_exception_self_explanatory)**)**  
**7.3.2 Exception: overrides (**[**原文**](http://google-styleguide.googlecode.com/svn/trunk/javaguide.html#s7.3.2-javadoc-exception-overrides)**、**[**譯文**](http://blog.mosil.biz/2014/05/java-style-guide/#s7.3.2_javadoc_exception_overrides)**)**

在最小的限度下，所有公開類別 (public class) 以及每個類別 (class) 中公開 (public) 或被保護的(protected) 成員都該寫上 Javadoc，當然也有少數如下列出的例外狀況。

若是被拿來做為定義類別 (class)、方法 (method) 或是屬性 (field) 在實作時該有的整體目的或是行為，就該寫上註解 (comment) 而非 Javadoc。(這樣做將更趨一致性並更讓工具看起來親切些 (tool-friendly))。

### 7.3.1 例外：不言自明的方法 (Exception: self-explanatory methods)

有些「簡單、明確」的方法也不一定要寫上 Javadoc，像是 getFoo 這種簡明的案例，好像除了寫上「返回 foo 值」也什麼好寫的。

**Important:**  
當今天是一個必需要讓看的人知道的狀況下，這個例外就不該拿出來做為忽略不寫相關資訊的理由。比方說，有一個方法被命名為getCanonicalName，這種狀況下，看得人可能並不了解「canonical name」這是什麼意思，在這種狀況下就不要忽略它的說明(包含只寫上「返回 canonical name」)。

### 7.3.2 例外：覆寫 (Exception: overrides)

在子物中覆寫的父類別的方法就不一定要寫上 Javadoc。