**google Java编程规范**

**班级：信1604-2 学号：20163786 姓名：宋璋晗**

# **源文件基础**

### **2.1 文件名**

源文件以其最顶层的类名来命名，大小写敏感，文件扩展名为.java。

### **2.2 文件编码：UTF-8**

源文件编码格式为UTF-8。

### **2.3 特殊字符**

#### **2.3.1 空白字符**

除了行结束符序列，ASCII水平空格字符(0x20，即空格)是源文件中唯一允许出现的空白字符，这意味着：

1. 所有其它字符串中的空白字符都要进行转义。
2. 制表符不用于缩进。

例如：

String unitAbbrev = "μs"; | 赞，即使没有注释也非常清晰

String unitAbbrev = "\u03bcs"; // "μs" | 允许，但没有理由要这样做

String unitAbbrev = "\u03bcs"; // Greek letter mu, "s" | 允许，但这样做显得笨拙还容易出错

String unitAbbrev = "\u03bcs"; | 很糟，读者根本看不出这是什么

return '\ufeff' + content; // byte order mark | Good，对于非打印字符，使用转义，并在必要时写上注释

# **源文件结构**

一个源文件包含(按顺序地)：

1. 许可证或版权信息(如有需要)
2. package语句
3. import语句
4. 一个顶级类(**只有一个**)

以上每个部分之间用一个空行隔开。

### **3.1 许可证或版权信息**

如果一个文件包含许可证或版权信息，那么它应当被放在文件最前面。

### **3.2 package语句**

package语句不换行，列限制(4.4节)并不适用于package语句。(即package语句写在一行里)

### **3.3 import语句**

#### **3.3.1 import不要使用通配符**

即，不要出现类似这样的import语句：import java.util.\*;

#### **3.3.2 不要换行**

import语句不换行，列限制(4.4节)并不适用于import语句。(每个import语句独立成行)

#### **3.3.3 顺序和间距**

import语句可分为以下几组，按照这个顺序，每组由一个空行分隔：

1. 所有的静态导入独立成组
2. com.google imports(仅当这个源文件是在com.google包下)
3. 第三方的包。每个顶级包为一组，字典序。例如：android, com, junit, org, sun
4. java imports
5. javax imports

组内不空行，按字典序排列。

### **3.4 类声明**

#### **3.4.1 只有一个顶级类声明**

每个顶级类都在一个与它同名的源文件中(当然，还包含.java后缀)。

例外：package-info.java，该文件中可没有package-info类。

#### **3.4.2 类成员顺序**

类的成员顺序对易学性有很大的影响，但这也不存在唯一的通用法则。不同的类对成员的排序可能是不同的。 最重要的一点，每个类应该以某种逻辑去排序它的成员，维护者应该要能解释这种排序逻辑。比如， 新的方法不能总是习惯性地添加到类的结尾，因为这样就是按时间顺序而非某种逻辑来排序的。

##### **3.4.2.1 重载：永不分离**

当一个类有多个构造函数，或是多个同名方法，这些函数/方法应该按顺序出现在一起，中间不要放进其它函数/方法。

# **格式**

**术语说明**：块状结构(block-like construct)指的是一个类，方法或构造函数的主体。需要注意的是，数组初始化中的初始值可被选择性地视为块状结构(4.8.3.1节)。

### **4.1 大括号**

#### **4.1.1 使用大括号(即使是可选的)**

大括号与if, else, for, do, while语句一起使用，即使只有一条语句(或是空)，也应该把大括号写上。

#### **4.1.2 非空块：K & R 风格**

对于非空块和块状结构，大括号遵循Kernighan和Ritchie风格 ([Egyptian brackets](http://www.codinghorror.com/blog/2012/07/new-programming-jargon.html)):

* 左大括号前不换行
* 左大括号后换行
* 右大括号前换行
* 如果右大括号是一个语句、函数体或类的终止，则右大括号后换行; 否则不换行。例如，如果右大括号后面是else或逗号，则不换行。

示例：

return new MyClass() {

@Override public void method() {

if (condition()) {

try {

something();

} catch (ProblemException e) {

recover();

}

}

}

};

### **4.2 块缩进：2个空格**

每当开始一个新的块，缩进增加2个空格，当块结束时，缩进返回先前的缩进级别。缩进级别适用于代码和注释。(见4.1.2节中的代码示例)

### **4.3 一行一个语句**

每个语句后要换行。

### **4.4 列限制：80或100**

一个项目可以选择一行80个字符或100个字符的列限制，除了下述例外，任何一行如果超过这个字符数限制，必须自动换行。

例外：

1. 不可能满足列限制的行(例如，Javadoc中的一个长URL，或是一个长的JSNI方法参考)。
2. package和import语句(见3.2节和3.3节)。
3. 注释中那些可能被剪切并粘贴到shell中的命令行。

### **4.5 自动换行**

**术语说明**：一般情况下，一行长代码为了避免超出列限制(80或100个字符)而被分为多行，我们称之为自动换行(line-wrapping)。

我们并没有全面，确定性的准则来决定在每一种情况下如何自动换行。很多时候，对于同一段代码会有好几种有效的自动换行方式。

#### **4.5.1 从哪里断开**

自动换行的基本准则是：更倾向于在更高的语法级别处断开。

1. 如果在非赋值运算符处断开，那么在该符号前断开(比如+，它将位于下一行)。注意：这一点与Google其它语言的编程风格不同(如C++和JavaScript)。 这条规则也适用于以下“类运算符”符号：点分隔符(.)，类型界限中的&（<T extends Foo & Bar>)，catch块中的管道符号(catch (FooException | BarException e)
2. 如果在赋值运算符处断开，通常的做法是在该符号后断开(比如=，它与前面的内容留在同一行)。这条规则也适用于foreach语句中的分号。
3. 方法名或构造函数名与左括号留在同一行。
4. 逗号(,)与其前面的内容留在同一行。

#### **4.5.2 自动换行时缩进至少+4个空格**

自动换行时，第一行后的每一行至少比第一行多缩进4个空格(注意：制表符不用于缩进。见2.3.1节)。

当存在连续自动换行时，缩进可能会多缩进不只4个空格(语法元素存在多级时)。一般而言，两个连续行使用相同的缩进当且仅当它们开始于同级语法元素。

### **4.6 空白**

#### **4.6.1 垂直空白**

以下情况需要使用一个空行：

1. 类内连续的成员之间：字段，构造函数，方法，嵌套类，静态初始化块，实例初始化块。在函数体内，语句的逻辑分组间使用空行。

**例外**：两个连续字段之间的空行是可选的，用于字段的空行主要用来对字段进行逻辑分组。

1. 类内的第一个成员前或最后一个成员后的空行是可选的(既不鼓励也不反对这样做，视个人喜好而定)。
2. 要满足本文档中其他节的空行要求(比如3.3节：import语句)

多个连续的空行是允许的，但没有必要这样做(我们也不鼓励这样做)。

#### **4.6.2 水平空白**

除了语言需求和其它规则，并且除了文字，注释和Javadoc用到单个空格，单个ASCII空格也出现在以下几个地方：

1. 分隔任何保留字与紧随其后的左括号(()(如if, for catch等)。
2. 分隔任何保留字与其前面的右大括号(})(如else, catch)。
3. 在任何左大括号前({)，两个例外：

* @SomeAnnotation({a, b})(不使用空格)。
* String[][] x = foo;(大括号间没有空格，见下面的Note)。

1. 在任何二元或三元运算符的两侧。这也适用于以下“类运算符”符号：

* 类型界限中的&(<T extends Foo & Bar>)。
* catch块中的管道符号(catch (FooException | BarException e)。
* foreach语句中的分号。

1. 在, : ;及右括号())后
2. 如果在一条语句后做注释，则双斜杠(//)两边都要空格。这里可以允许多个空格，但没有必要。
3. 类型和变量之间：List list。
4. 数组初始化中，大括号内的空格是可选的，即new int[] {5, 6}和new int[] { 5, 6 }都是可以的。

### **4.8 具体结构**

#### **4.8.1 枚举类**

枚举常量间用逗号隔开，换行可选。

没有方法和文档的枚举类可写成数组初始化的格式：

private enum Suit { CLUBS, HEARTS, SPADES, DIAMONDS }

由于枚举类也是一个类，因此所有适用于其它类的格式规则也适用于枚举类。

#### **4.8.2 变量声明**

##### **4.8.2.1 每次只声明一个变量**

不要使用组合声明，比如int a, b;。

##### **4.8.2.2 需要时才声明，并尽快进行初始化**

不要在一个代码块的开头把局部变量一次性都声明了(这是c语言的做法)，而是在第一次需要使用它时才声明。 局部变量在声明时最好就进行初始化，或者声明后尽快进行初始化。

#### **4.8.3 数组**

##### **4.8.3.1 数组初始化：可写成块状结构**

数组初始化可以写成块状结构，比如，下面的写法都是OK的：

new int[] {

0, 1, 2, 3

}

new int[] {

0,

1,

2,

}

new int[] {

0, 1,

2, 3

}

new int[]

{0, 1, 2, 3}

#### **4.8.4 switch语句**

**术语说明**：switch块的大括号内是一个或多个语句组。每个语句组包含一个或多个switch标签(case FOO:或default:)，后面跟着一条或多条语句。

##### **4.8.4.1 缩进**

与其它块状结构一致，switch块中的内容缩进为2个空格。

每个switch标签后新起一行，再缩进2个空格，写下一条或多条语句。

switch (input) {

case 1:

case 2:

prepareOneOrTwo();

// fall through

case 3:

handleOneTwoOrThree();

break;

default:

handleLargeNumber(input);

}

##### **4.8.4.3 default的情况要写出来**

每个switch语句都包含一个default语句组，即使它什么代码也不包含。

#### **4.8.5 注解(Annotations)**

注解紧跟在文档块后面，应用于类、方法和构造函数，一个注解独占一行。这些换行不属于自动换行(第4.5节，自动换行)，因此缩进级别不变。例如：

@Override

@Nullablepublic String getNameIfPresent() { ... }

**例外**：单个的注解可以和签名的第一行出现在同一行。例如：

@Override public int hashCode() { ... }

应用于字段的注解紧随文档块出现，应用于字段的多个注解允许与字段出现在同一行。例如：

@Partial @Mock DataLoader loader;

参数和局部变量注解没有特定规则。

#### **4.8.6 注释**

##### **4.8.6.1 块注释风格**

块注释与其周围的代码在同一缩进级别。它们可以是/\* ... \*/风格，也可以是// ...风格。对于多行的/\* ... \*/注释，后续行必须从\*开始， 并且与前一行的\*对齐。以下示例注释都是OK的。

/\*

\* This is // And so /\* Or you can

\* okay. // is this. \* even do this. \*/

\*/

注释不要封闭在由星号或其它字符绘制的框架里。

# **命名约定**

### **5.1 对所有标识符都通用的规则**

标识符只能使用ASCII字母和数字，因此每个有效的标识符名称都能匹配正则表达式\w+。

在Google其它编程语言风格中使用的特殊前缀或后缀，如name\_, mName, s\_name和kName，在Java编程风格中都不再使用。

### **5.2 标识符类型的规则**

#### **5.2.1 包名**

包名全部小写，连续的单词只是简单地连接起来，不使用下划线。

#### **5.2.2 类名**

类名都以UpperCamelCase风格编写。

类名通常是名词或名词短语，接口名称有时可能是形容词或形容词短语。现在还没有特定的规则或行之有效的约定来命名注解类型。

测试类的命名以它要测试的类的名称开始，以Test结束。例如，HashTest或HashIntegrationTest。

#### **5.2.3 方法名**

方法名都以lowerCamelCase风格编写。

方法名通常是动词或动词短语。

下划线可能出现在JUnit测试方法名称中用以分隔名称的逻辑组件。一个典型的模式是：test<MethodUnderTest>\_<state>，例如testPop\_emptyStack。 并不存在唯一正确的方式来命名测试方法。

#### **5.2.4 常量名**

常量名命名模式为CONSTANT\_CASE，全部字母大写，用下划线分隔单词。那，到底什么算是一个常量？

每个常量都是一个静态final字段，但不是所有静态final字段都是常量。在决定一个字段是否是一个常量时， 考虑它是否真的感觉像是一个常量。例如，如果任何一个该实例的观测状态是可变的，则它几乎肯定不会是一个常量。 只是永远不打算改变对象一般是不够的，它要真的一直不变才能将它示为常量。

// Constants

static final int NUMBER = 5;

static final ImmutableList<String> NAMES = ImmutableList.of("Ed", "Ann");

static final Joiner COMMA\_JOINER = Joiner.on(','); // because Joiner is immutable

static final SomeMutableType[] EMPTY\_ARRAY = {};

enum SomeEnum { ENUM\_CONSTANT }

// Not constants

static String nonFinal = "non-final";

final String nonStatic = "non-static";

static final Set<String> mutableCollection = new HashSet<String>();

static final ImmutableSet<SomeMutableType> mutableElements = ImmutableSet.of(mutable);

static final Logger logger = Logger.getLogger(MyClass.getName());

static final String[] nonEmptyArray = {"these", "can", "change"};

#### **5.2.5 非常量字段名**

非常量字段名以lowerCamelCase风格编写。

这些名字通常是名词或名词短语。

#### **5.2.6 参数名**

参数名以lowerCamelCase风格编写。

参数应该避免用单个字符命名。

#### **5.2.7 局部变量名**

局部变量名以lowerCamelCase风格编写，比起其它类型的名称，局部变量名可以有更为宽松的缩写。

虽然缩写更宽松，但还是要避免用单字符进行命名，除了临时变量和循环变量。

即使局部变量是final和不可改变的，也不应该把它示为常量，自然也不能用常量的规则去命名它。

#### **5.2.8 类型变量名**

类型变量可用以下两种风格之一进行命名：

* 单个的大写字母，后面可以跟一个数字(如：E, T, X, T2)。
* 以类命名方式(5.2.2节)，后面加个大写的T(如：RequestT, FooBarT)。

### **5.3 驼峰式命名法(CamelCase)**

[驼峰式命名法](http://zh.wikipedia.org/wiki/%E9%A7%9D%E5%B3%B0%E5%BC%8F%E5%A4%A7%E5%B0%8F%E5%AF%AB)分大驼峰式命名法(UpperCamelCase)和小驼峰式命名法(lowerCamelCase)。 有时，我们有不只一种合理的方式将一个英语词组转换成驼峰形式，如缩略语或不寻常的结构(例如"IPv6"或"iOS")。Google指定了以下的转换方案。

名字从散文形式(prose form)开始:

1. 把短语转换为纯ASCII码，并且移除任何单引号。例如："Müller’s algorithm"将变成"Muellers algorithm"。
2. 把这个结果切分成单词，在空格或其它标点符号(通常是连字符)处分割开。
   * + 推荐：如果某个单词已经有了常用的驼峰表示形式，按它的组成将它分割开(如"AdWords"将分割成"ad words")。 需要注意的是"iOS"并不是一个真正的驼峰表示形式，因此该推荐对它并不适用。
3. 现在将所有字母都小写(包括缩写)，然后将单词的第一个字母大写：最后将所有的单词连接起来得到一个标识符。
   * + 每个单词的第一个字母都大写，来得到大驼峰式命名。
     + 除了第一个单词，每个单词的第一个字母都大写，来得到小驼峰式命名。

示例：

Prose form Correct Incorrect

-----------------------------------------------------------------------------------------------------

"XML HTTP request" XmlHttpRequest XMLHTTPRequest

"new customer ID" newCustomerId newCustomerID

"inner stopwatch" innerStopwatch innerStopWatch

"supports IPv6 on iOS?" supportsIpv6OnIos supportsIPv6OnIOS

"YouTube importer" YouTubeImporter

YoutubeImporter\*

# **编程实践**

### **6.1 @Override：能用则用**

只要是合法的，就把@Override注解给用上。

### **6.2 捕获的异常：不能忽视**

除了下面的例子，对捕获的异常不做响应是极少正确的。(典型的响应方式是打印日志，或者如果它被认为是不可能的，则把它当作一个AssertionError重新抛出。)

如果它确实是不需要在catch块中做任何响应，需要做注释加以说明(如下面的例子)。

try {

int i = Integer.parseInt(response);

return handleNumericResponse(i);

} catch (NumberFormatException ok) {

// it's not numeric; that's fine, just continue

}

return handleTextResponse(response);

**例外**：在测试中，如果一个捕获的异常被命名为expected，则它可以被不加注释地忽略。下面是一种非常常见的情形，用以确保所测试的方法会抛出一个期望中的异常， 因此在这里就没有必要加注释。

try {

emptyStack.pop();

fail();

} catch (NoSuchElementException expected) {

}