**Java编码规范**

# 序言

本规范的目的在于编写出可靠的，一致的，便于理解的JAVA 源代码，并且写出的代码容易被其他人阅读，所有的JAVA 代码保持良好的一致性，容易在以后的阶段中进行维护和 升级。同时， 也是为了通过一定时间的实践，增加软件产品的投入产出量，以提高工作效率。

# 代码命名一般原则

在 JAVA 代码中，所有的代码命名的总原则是：

* 1. 用 全 英 文 单 词 命 名 的 方 式 ， 准 确 地 描 述 变 量 、 属 性 、 类 等 。 如 ： 使 用firstName,grandTotal 等命名就比x1,y1,fn 等更容易让人理解其含义，尽管它们的长度要大一些。
  2. 采用一些更加准确的术语来命名。例如，如果我们的用户称他们的clients为customers，那么我们就应该采用customer 来命名，而不是采用client 来命名。这是一些细微的地方，但也希望能注意。
  3. 采用大小写混合的方式来命名，以便命名有很好的可读性。在JAVA 代码中，将采用如下原则： 类或接口命名中每个单词的首字母均大写，而单词的剩余部分均小写。其它像变量、成员方法、属性等除第一个单词全部小写以外，其它单词的首字母均大写，而单词的剩余部分均小写。
  4. 尽量少用单词的缩写形式，但如果一定要用，则必须要选择大家通用的缩写方式， 并且要在本 JAVA 源代码中坚持用它，而不要一会用这种缩写方式，一会用那种缩

写方式。比如，如果要用 “number”的缩写方式，则可用 “no”或“num”两种缩写方式，而不要用 “nu”这种大家不常用的缩写方式，并且要保持不变。

* 1. 尽量避免太长的命名，一般以少于 20 个字符为宜。
  2. 尽量避免使用这样的命名：两个或多个命名仅仅是其中的有些字符大小写不一样，

或 者 仅 仅 是 其 中 有 些 单 词 是 单 复 数 之 区 别 。 例 如 ： persistentObject与persistentObjectsanSqlDatabase 与 anSQLDatabase 等 。

* 1. 尽量避免使用下划线。 在 JAVA 中，一般比较少的采用下划线的命名方式。(注：在使用 CORBA 时，系统会自动的生成一些带有下划线的属性，类名，接口名等等，

或者是 IDL 接口在定义时会带有下划线的属性，类名，接口名等，在此种情况下， 该条件可以适当放宽 )

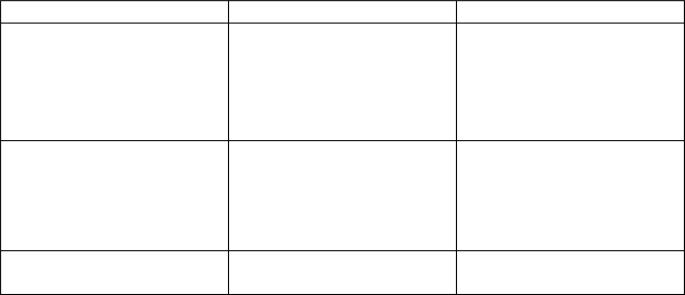
# 代码注释使用的一般原则和类型

在 JAVA 代码中， 我们经常要使用代码注释的方式来帮助理解代码的含义。代码注释的一般原则主要有以下几个方面：

* 1. 代码中的注释应该以让人一目了然为目标。我们之所以要增加代码注释，其目的就是为了让编写代码的人、同一项目组的成员或以后的开发人员能很容易的理解其代码的意图。
  2. 避免使用标语式的、实际毫无用处的代码注释。
  3. 尽量采用简洁、易懂的注释代码，而不要长篇大论。
  4. 有些特殊的代码中要加以适当的注释，以说明其为什么要这样做。例如，一些特殊的商业规范的计算等等。

在 JAVA 代码中，经常使用三种类型的注释：文档型的注释，其一般格式是：以“/\*\* ”开头，以 “\*/ ”结尾；C 语言类型的注释，它通常是以“/\* ”开头，以 “\*/ ”结尾；还有一种是通 常用的单行注释的方法，即以“//”开头，对一个单行进行注释。下面以表格的方式，分别说

明这三种方式在 JAVA 中如何更好的使用。

注释类型 使用方式 例程

文档型注释 文档型注释经常用在定义接 口，类，成员方法， 域的定义之前，不且这种注释可以被通过javadoc 直接生成帮助文

/\*\*

\*/

Customer- A customer is any person or

@author S.W.Ambler

档。

C 风格型的注释 当要一次对多行代码进行注释，或者这些代码将不被执行，或者对这些行进行调试时，本注释是一种比较合适的

注释方法。

单行注释 通常在成员方法内部注释商 业逻辑， 一小节代码， 或者临

public class Customer{}

/\*

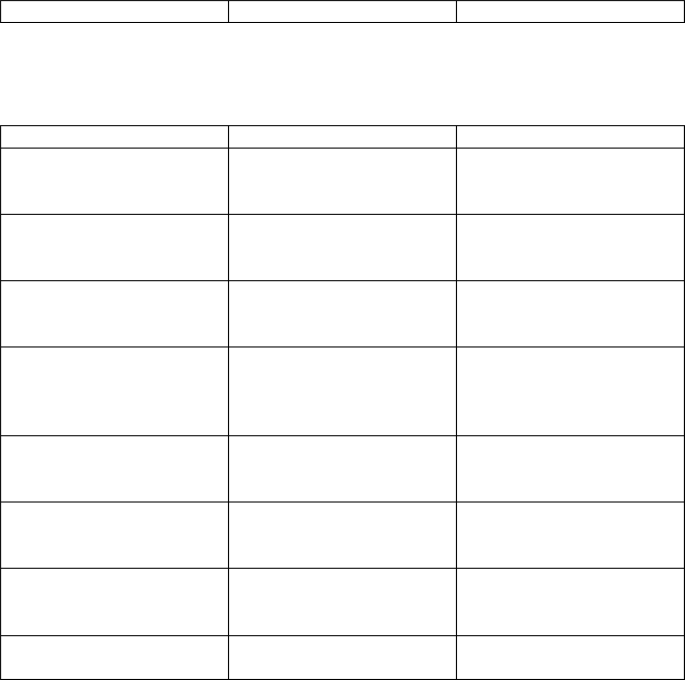
This code was commented ..

(the source code)

\*/

//Apply a 5% discount to all

//invoices over $1000 as

时变量的声明。 //defined by the ..

同时，为了编程人员所做的文档型注释，能更有效的生成 HTML 格式的帮助文件，将提供以下一些更多的内容：

标志 使用 目的

@author authorName 一般在类、接口定义之前的文

档型注释里面使用。

@deprecated 一般在类，成员方法定义之前的文档型注释里面使用。

@exception name description 成员方法定义之前的文档型

注释里面使用。

@param name description 成员方法定义之前的文档型

注释里面使用。

@return description 成员方法定义之前的文档型注释里面使用。

@see ClassName 一般在类，接口，成员方法， 域定义之前的文档型注释里 面使用。

一般是指明编写本接口或本 类的作者，当然可以有多个， 每一个作者用 @author 标明。它一般用于声明一个类或者 一个成员方法被废弃， 不再使用。

用于声明本成员方法将可能 会抛出的异常。 每一个异常用一个 @exception 标明。

用于说明传递给该成员方法的参数，包括其类型和它的使

用，每一个参数用 @param 标明。

用于说明本成员方法的返回 值类型， 如果可能， 适当的描述其可能的返回值

用于建立一个超文本链接到 指定的类， 接口。一般应该详细的描述其类名。

@see ClassName#member functionsName

一般在类，接口，成员方法， 域定义之前的文档型注释里 面使用。

用于建立一个超文本链接到 指定的成员方法。 一般应详细描述类名。

@version text 一般在类，接口定义之前的文档型注释里使用。

用于指定相应的版本信息

# 方法的命名和注释规范

## 4． 1．成员方法的命名

成员方法的命名应该采用全英文单词且是大小写混合的方式来进行。方法名的第一个单词均小写，其它单词的首字母大写。而且方法名中第一个单词最好是一个动词形式的。如： openAccount() ,printMailingLabel() , save() , delete()等。

通过这样的命名规范， 很容易看出这个方法是干什么用的，它的目的是什么。尽管有时

方法名会比较长，但它能很容易让其他人理解本方法的意图。

get 方法，它一般返回一个属性值，一般应将get 放在方法名中第一个单词的位置。而

当它返回的是判断其结果是true 或者false的时候，一般用”is”来取代 ”get”。例如：getFirstName() ,getAccountNumber() ,isPersistent() ,isAtEnd()等等。

通过这种命名规范，就可以很清楚的知道本方法是返回一个对象的属性或返回一个boolean 值。

同样，对于 set 方法，它一般是用于设置对象的属性值，同样也将set 放在方法名中的第 一 个 单 词 的 位 置 。 如 ： setFirstName(StringaName),setAtEnd(boolean isAtEnd), setPersistent(booleanisPersistent)等等。

## 4． 2．构造方法的命名

构造方法一般是用来创建一个对象时要执行的初始化操作。构造方法必须具有与类名完全一致的名字，其大小写也必须完全一致。这种命名规范是由sun 公司确定，且必须遵守。

## 3．成员方法的头部注释文档规范

每一个成员方法的前面必须要有相应的注释文档说明，它主要包括以下内容，但不仅限于以下内容：

* 1. 本成员方法的目的是什么。
  2. 本成员方法要传递的参数是什么。它们是如何被使用的。
  3. 本成员方法的返回值是什么。
  4. 本成员方法有什么潜在的 bugs，它们在被调用时要注意些什么。
  5. 本成员方法会抛出什么样的异常。
  6. 访问权限的控制说明。
  7. 代码更改的一些历史信息，如什么时候被更改过，谁更改的，目的是什么等等一些信息。

## 4．成员方法的内部注释文档规范

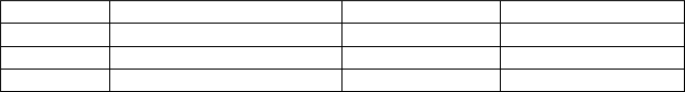
为了更好的描述成员方法的目的，还会在成员方法的内部给予适当的注释，一般有两种格式： /\* ..\*/ 和// .。一般而言，内部注释文档应该说明以下一些要素：

* 1. 控制结构，描述每一个控制结构是什么样的。
  2. 在某些特殊的地方说明本代码为什么要这样编写。例如，在某些特殊的商业规范的处理代码中，给予说明。
  3. 局部变量的说明，通常用单行的注释说明它们是如何使用的，意义是什么。
  4. 对难以理解的，复杂的代码要有适当的说明，以便于其他开发人员理解。
  5. 对一些必须按照特定的顺序被执行，否则出错的代码段，对其处理顺序进行适当的文档说明。

## 4． 5．成员方法的访问权限控制

成员方法的访问权限控制的基本原则是尽量的限制对成员方法的访问权限。如果没有必要设置为 public ，则将其设置为protected ，否则设置为默认方式或者private。

## 6． getXxxx方法和setXxxx方法的使用

getXxxx 方法和 setXxxx 方法的名称一般是：get + 属性名 /set + 属 性名或is +属性名。 例如：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 类型 | get 方法 | set 方法 |
| FirstName | String | getFirstName() | setFirstName( 参数表 ) |
| Persistent | Boolean | isPersistent() | setPersistent(参数表 ) |
| OrderItems | Array of OrderItems Objects | getOrderItems() | setOrderItems(参数表 ) |

getXxxx 方法和 setXxxx 方法一般对其有如下的使用方式：

* 1. 设置对象属性的初始值；
  2. 更新对象的属性值；
  3. 得到属性值；
  4. 控制对属性值的访问和修改，特别是修改；
  5. 访问常量；
  6. 访问数组， vector 等多值数据变量；
  7. 降低父、子类之间的耦合程度。例如：

// Answer the branch number, which is the leftmost four digits of the full account number protected int getBranchNumber(){

if (branchNumber = = 0){

//The default branch number is 1000,which is the main branch in downtown Bedrock setBranchNumber(1000);

}

return branchNumber;

}

getXxxx 方法和 setXxxx 方法的访问权限的控制：

一般情况，将它们设置为 “protected”，这样，只有同包类或子类的对象可以访问它们。

但是通常将 getXxxx 方法设置为 “public ”方式， 而 setXxxx 方法则是 “protected”方式。 只有在极少数情况下才将 setXxxx 方法设置为 “public ”方式。当然，有时也将 setXxxx 方法设置为 “private”方式。

# 属性变量的命名和注释规范

## 5． 1．属性变量的命名规范

属性变量的命名采用全英文单词的混合命名方式，第一个单词全部小写，以后单词的

第一个字母大写， 其余小写。 以便让它易于理解。 而对于那些像数组或动态数组等一些多值 的数据类型属性，则应该采用单词复数的方式命名（ s 加在最后一个单词上面），它的主要优点是可以方便的指明它们有多个值。例如：firstName, zipCode, orderItems等。

## 5． 2．属性变量的访问权限控制

为了是对象具有更好的封闭性，一般不把属性的访问权限设置为public ，建议一般把所有的属性访问权限设置为private，以防止其它类的对象直接访问本类对象的属性。而对属性的访问可以用 getXxxx 方法和 setXxxx 方法进行。

## 3．属性变量的注释

每一个属性变量都要给予注释，以便其他开发人员更好的理解其含义。一般从以下几个方面给予注释：

* 1. 描述属性变量，以便其他开发人员知道如何使用它；
  2. 访问权限的说明。如果你的属性变量的访问权限不是private 修饰的，则必须给予说明其为什么是这样的；
  3. 对于某些特殊取值（范围）的属性，其取值（范围）给予适当的说明，以便其他开发人员理解和使用；
  4. 适当的例子说明。对与某些特殊情况相关的属性，可适当的加以例子说明，以便于别人更好的理解和使用。

# 组件变量的命名规范

对于组件的命名， 一般采用全英文单词的命名方式， 但是采用将组件类型名后缀的方法。这样， 就很容易识别这个组件变量的目的， 从它的后缀类型就可以方便的看出。 也很容易在列表中找到每一个组件 （在很多可视化编程环境中， 提供列表的方式来快速显示所有用到的 组件变量）。如： okJButton, fileJMenu, newFileMenuItem 等等。

# 常量命名规范

常量命名采用全英文单词的方式进行，且全是大写的字母，单词之间用下划线连接。如：

MINIMUM\_BALANCE,DEFAULT\_START\_DATE等等。 它的主要优点是可以很容易的与变量区分开来。

# 局部变量的命名和注释规范

一般而言， 局部变量的命名规范遵循属性变量的命名规范，即全用英文单词命名，且除第一个单词以外，其它单词的首字母均大写。

但是，对于下面所列举的几种特殊类型的局部变量的命名有些特殊的规定：

* 1. streams

当一个单一的输入或输出流对象被定义和使用时，一般用in,inputStream表示输入流，而用out,outputStream 表示输出流。而当一个流同时被使用为输入流和输出流的时候，则用 inOut 或者 ioStream 来命名。

* 1. 循环计数器的使用

由于在成员方法中，我们经常会用一些循环结构，同时也就会使用循环计数器。在

以前的 C， C++ 中，已经习惯于使用像 i,j,k 等变量作为循环计数器，在JAVA 编码中，也采用常用的，单一的小写字母来命名循环计数器。例如for(i=0;i<10;i++)

* 1. 例外对象的命名规范

在 JAVA 中，经常会碰到一些例外情况的处理，一般用一个字母 ”e”来命名相应的例外对象。局部变量的文档注释一般涉及到以下几个方面：

1. 说明局部变量是如何被使用的；
2. 局部变量的声明与使用尽量靠近，也即等到要使用的时候才声明局部变量；
3. 一个局部变量尽量只用在一件事物上，而不要用在多件事物上，以便于理解；
4. 一般采用单行注释的方式。

# 成员方法参数命名和注释规范

成员方法参数的命名规范基本上与局部变量的命名规范相同。如：customer, inventoryItem, in , e等。

成员方法参数的文档注释：

成员方法参数的文档注释在成员方法的头部注释文档中，它应包括以下内容：

* 1. 该参数将被如何使用；
  2. 该参数的一些约束或其它前提条件；
  3. 如果仅凭以上的描述还不足以让人容易理解该参数是什么样的，可以举一个或多个例子加以说明。

# 类的命名和注释规范

类的命名也是采用全英文单词描述的方法，但它的第一个单词的首字母必须大写，其它单词的首字母也要大写。例如：public class Customer{ .}public class OrderItems{ }

类的注释文档一般位于类的定义之前，它将包含如下内容：

* 1. 本类的目的。其他开发人员常常需要知道一个类的通常目的，以便决定其是否是他所需要的；
  2. 一些预知的 bugs。一般情况， 如果有 bugs 存在， 最好是对其进行相应的处理。 但有时，我们没有更多的时间，或在本类中没有必要对其进行处理，在目前而言不是太

重要。在这种情况下，应文档给予说明。比如，类中的一个成员方法在传递一个负 数的参数时不能正常工作，而正好应用程序有不会传递负的参数过来，在这种情况 下，就有一个预知的 bugs，但没有相应的处理，则应文档说明，以便其他开发人员清楚；

* 1. 对本类的一些开发 /维护的历史信息的说明。 比如日期、 作者以及主要改变的内容等；
  2. 如果是线程类的话，应对其使用的并发控制策略作适当的描述，以及为什么采用这种策略而不是其它策略等信息。

# 接口的命名和注释规范

接口的命名，其第一个字母必须是大写的 “I”，以表明它是一个接口，与其它类进行区 别。剩下的部分与类的命名规范一样， 采用全英文单词描述的方法， 第一个单词首字母大写， 其它单词的首字母也大写。例如： public interface IOrderItems{ ..}

public interface ICustomer{ ..}

接口的文档注释必须在接口被定义前给予说明，主要包括以下内容：

* 1. 本接口定义的目的。当其他开发人员使用一个接口的时候，他们常常需要了解本接口所封装的内容以及其目的；
  2. 它将被怎样的使用或这不能被怎样的使用。其他开发人员想知道本接口能被如何使用以及它不能被如何使用；
  3. 接口内的抽象成员方法的注释，参照类的成员方法的注释。

# 包的命名和文档说明规范

包的命名有以下几个规则：

* 1. 包与子包之间或包与类之间用“ ”来加以区别；
  2. 包名全部用小写字母来命名，如：persistence.mapping.relational 等 ；
  3. 如果一个包有可能在多个组织中使用，则使用本组织域名的逆转方式来命名包。如：

[cn.com.mcm.www.persistence.mapping.relational](http://www.persistence.mapping.relational/) 等。

对于每一个包， 应该有一个或多个外部文档来描述这个包。一般而言，文档中应描述如下内容：

1. 描述这个包是一个什么样的包，它有一些什么样的相关性等等；
2. 包中的类和接口的简单描述。应该有一个包中的类和接口的简单列表，并对每一个类和接口有相应的简单描述，以便其他开发人员都清楚的知道其包含的内容。

# 编写测试代码

* 1. 凡是用于临时测试的所有成员方法均冠以前缀“test”，这样将有利于快速的找到临时用于测试的所有成员方法；
  2. 所有用于测试其它成员方法的成员方法，必须按以下规则命名：test+首字母大写的成员方法名+For+ 测试的某一方面（首字母大写）。例如：用来测试成员方法withdrawFunds() 的成员方法的命名为：testWithdrawFundsForInsufficientFunds()或者

是： testWithdrawFunds() 。

# 文件的命名

文件名必须与包含有public 关键字的类名或接口名完全一致，即大小写也必须一致。并且在一个文件中只能包含有一个public 关键字的类或者接口，文件名的后缀必须是java， 这是由 sun 公司规定的，必须遵守。

文件中还应包括以下内容：

* 1. 在文件中简要的列出各个类和接口；
  2. 在文件的顶端，说明文件名，这主要是以后打印的时候，知道本代码属于哪个文件的；