卷　　号	
卷内编号	
密　　级	

中央研究院 Java 编程规范

Version 2.0

作者:**张玮**

分类:COD
使用者:**设计员、实施员**

文档信息

标题:**中央研究院** Java **编程规范**
作者:**张玮**
创建日期:2001-09-04
上次更新日期:2001-09-30
版本:1.0

部门名称:**中央研究院**

文档状态

文档状态	□**草稿** ■**正式**		
文档评审人员	SEPG	**评审时间**	2001/9/30

修订文档历史记录

日期	**版本**	**说明**	**作者**
2001-9-4	**初稿**		**张玮**
2001-09-17	1.0		**张玮**
2001-09-30	2．0		**张玮**

目录

中央研究院 Java 编程规范

1. 简介

本文提供一整套编写高效可靠的 Java 代码的标准、约定和指南。它们以安全可靠的软件工程原则为基础，使代码易于理解、维护和增强。而且，通过遵循这些程序设计标准，你作为一个 Java 软件开发者的生产效率会有显著提高。经验证明，若从一开始就花时间编写高质量的代码，则在软件开发阶段，对代码的修改要容易很多。最后，遵循一套通用的程序设计标准将带来更大的一致性，使软件开发团队的效率明显提高。

1.1 目的

本文档用于指导编码，并确立起一个易于理解、维护，并同软件过程及过程工具紧密集成的编码约定。

1.2 范围

本文档对中央研究院所有采用 Java 作为开发语言的项目适用。

1.3 定义、首字母缩写词和缩略语

注释率　注释在代码中所占比例。
语句
注释　　在源文件中
C 语言风格注释
行内注释
文档注释

1.4 参考资料

RUP2000 中文版的《Java 编程指南》
Draft Java Coding Standard　Doug Lea
Javasoft coding standards
The Infospheres Java Coding Standard

1.5 概述

本文档共分为六部分：
第一部分为简介；
第二部分程序设计的一些普遍标准；为
第三部分为对 Java 源文件的分结构说明；
第四部分为描述与工具的集成；
第五部分为本文档所规定的一些数据；
第六部分为建议的最佳实践。

1.6 原则

本节描述描述制作本文档时所采取的原则，这些原则是一开始就确定的，并且贯穿在整个文档中。以下的原则的排列是按优先级进行的，这也就是说，在原则之间有冲突时，应该优先考虑排在前面的项目。

1.6.1 便于阅读

从软件工程的角度上来看，代码让人更容易看懂远比让机器看懂更有价值得多。首先代码永远不进行维护似乎是很少发生的，其次，就算代码永远不进行维护，至少为了代码重用的目的也需要其他人能够很容易地看懂代码。这里有一条 30 秒原则，即其他程序员能在 30 秒内完全读懂你的方法，理解做什么，为什么及怎样做的。

1.6.2 与工具集成

鉴于开发过程大量地集成了工具，因此要与整个开发过程结合就必须考虑怎样使代码与工具和平相处并配合密切。与工具集成同时可以增加生产力，并减少由于人工转换过程中引入的错误。

1.6.2.1 JavaDoc

与 JavaDoc 的集成可使文档的说明自动生成并可表现为多种形式。

1.6.2.2 Rose

考虑到整个开发过程使用了 RUP，因此与其所提供的工具 Rose 的集成是很有价值的。与 Rose 的集成是指保存在 Rose 中的设计模型可正向生成符合本规范的代码，也可从符合本规范的代码中反向生成 Rose 的设计模型。

1.6.2.3 配置工具

与配置工具的集成是指代码规范与配置工具的要求不冲突。

1.6.2.4 编码工具

与编码工具的集成是指代码可用编码工具自动完成一部分内容，这也是提高生产力的一种努力。

1.6.3 便于调试

便于调试是指在规范的约束下，应该不影响调试。

1.6.4 便于写作

这条原则是说明不要花时间去进行格式的编排，如将注释换一行书写，而不是花时间去调整让注释右对齐。让实施员集中精力于该干的事，而诸如注释的好看与否等等则不应该是编码人员关心的事。

1.6.5 减少重复

同样的一个信息不应该在多个地方出现。

原则	约定
便于阅读	缩进 适当地增加括号 折行 行宽限制
与工具集成	对 JavaDoc 的注释支持 对 Rose 的注释支持 JBuilder 的模板文件及配置
便于调试	一行一条语句
便于写作	注释与语句不同行
减少重复	同样的内容不在多处重复

表格 1原则及约定对应表

2. 程序设计标准

本节的大部分内容是从参考资料 1中剪裁而来，如果需要更详细的内容，请参见参考资料 1中的程序设计标准的一节。

Java 的程序设计标准很重要，原因在于它将提高开发团队各成员的代码的一致性。一致性的提高会使代码更易理解，这意味着它更易开发和维护。从而降低了应用程序的总开发成本。

2.1 通用命名约定

我们将在整个标准中讨论命名约定，所以让我们先讨论几个基本点：

2.1.1 采用该领域的术语

如果用户称他们的“客户”(clients) 为“顾客”(customers)，那么就采用术语 Customer 来命名这个类，而不用 Client。许多程序开发者会犯的一个错误是，不去使用工业或领域里已经存在着很完美的术语时，却生造出一些普通词汇。

2.1.2 大小写

常量的字母全部大写，单词之间用一个下划线字符(_)进行分隔。

除常量外的命名采用大小写混合，提高名字的可读性。一般采用小写字母，但是类和接口的名字的首字母，以及任何中间单词的首字母应该大写。

2.1.3 使用可以准确说明的英文描述符

例如，采用类似 firstName，grandTotal 或 CorporateCustomer 这样的名字。虽然象 x1，y1 或 fn 这样的名字很简短，输入起来容易，但是我们难以知道它们代表什么、结果是什么含义，因而使代码难以理解、维护和改进。

2.1.4 下划线的使用

避免使用下划线作为名字的首末字母，以下划线为首末字母的名字通常为系统保留，除预处理定义之外，一般不用作用户命名。更重要的是，下划线经常造成麻烦而且难输入，所以尽量避免使用。因此建议下划线只用在常量的命名的单词中间作分隔符。

2.1.5 避免使用长名字

命名不要超过标识符最大字符数，标识符字符数请参见数据一节。

虽然 PhysicalOrVirtualProductOrService 看起来似乎是个不错的类名，但是这个名字太长了，应该考虑重新给它起个短一点的名字，比如象 Offering。

2.1.6 标识符的命名

标识符类型	命名规则	Examples
Packages	每一个包的名称总是小写，规则采用类似于 TCP/IP 中域名的反序，即以 com, edu, gov, mil, net, org, 或 ISO 3166,1981 中定义的两位国家代码开始，接着是组织名称，在此处以 com.topgroup 开始，接着是<项目英文简写>，其后的部分由项目自已定义。	com.topgroup.eng com.topgroup.report.v2 com.topgroup.jdm
Classes	类名必须是名词，大小写遵照前面定义的大小	class Raster;

标识符类型	命名规则	Examples
	写规则（即每个单词首字母大写）。类名应该简单清晰。不要使用除通用的计算机缩写或通用的领域缩写外的其它缩写。	class ImageSprite;
Interfaces	同类名规则。	interface RasterDelegate; interface Storing;

Methods	方法应该是动词，或以动词开始的动宾结构短语，大小写遵照前面定义的大小写规则(即除第一个单词外，每个单词的首字母大写)。	run();
		runFast();
		getBackground();

Variables	变量名应该短而准确并便于记忆。大小写遵照前面定义的大小写规则(即除第一个单词外，每个单词的首字母大写)。除了临时使用的如循环变量等以外，不要使用单字符的变量名。对于使用单字符的临时变量名，建议循环变量名用 i,j,k，n 表示整数，c,d,e 表示字符。	int i; char c; float myWidth;

Constants	全部大写，单词之间用下划线分隔。	static final int MIN_WIDTH = 4;
		static final int MAX_WIDTH = 999;
		static final int GET_THE_CPU = 1;

2.1.7 界面元素的命名

界面元素是指如 Form，控件等变量及类的命名。界面元素命名的一条基本原则是界面元素的命名应简单而不产生混淆，一般情况下，界面元素的命名直接参照变量命名规则，但若发生了混淆或估计会发生混淆的界面元素，则可以在该元素的后加元素类型的后缀，后缀的生成规则是：

1、 若元素类型是标准的 Swing 类型，则去掉类名"J"字符后的部分为元素类型名
2、 若元素类型是标准的 AWT 类型，则直接使用类名
3、 如果元素类型不是标准的 Swing 或 AWT 类型，元素类型的后缀使用其 Swing 或 AWT 的基类进行命名。如：自定义了 class MyPanel extends JPanel，则若有 MyPanel 的元素，则元素后缀为 Panel。

下表给出了基本的元素类型后缀：

类型	后缀	例
JPanel	Panel	listPanel
JButton	Button	delButton
JFrame	Frame	mainFrame
JInternalFrame	Frame 或 InternalFrame	welcomeFrame,welcomeInternalFrame
JCheckBox	CheckBox	
J		

2.1.8 非强制的一些命名规则

类型	规则	例
抽象类	以 Abstract 开始	AbstractDjinn, AbstractCat, AbstractClass, AbstractPlayer
Factory 类	以 Factory 结束	DjinnFactory, CatFactory, ClassFactory, PlayerFactor
实现类	以 Impl 结束	DjinnImpl, CatImpl, ClassImpl, PlayerImpl
Exception	以 Exception 结束	DjinnException, CatException, ClassException, PlayerException
Interface	以 Interface 结束或加-able 后缀	DjinnInterface, CatInterface, Runnable, RemoteLoadable

2.2 通用格式

一种提高代码可读性的方法是给代码分段，换句话说，就是在代码块内让代码缩进。所有在括号{和}之内的代码，构成一个块。基本思想是，块内的代码都应统一地缩进去一个单位。

2.2.1 圆括号

在表达式、方法调用及方法声明中圆括号“(”后及“)”前不应该有空格。

2.2.2 缩进

同级之间在同一个缩进位置。

下一级与上一级之间需要缩进。

每一级缩进空格数请参见数据一节。

2.2.3 空白行

在代码中使用空白。在 Java 代码中加入几个空行，也叫空白，将代码分为一些小的、容易理解的部分，可以使它更加可读。建议采用一个空行来分隔代码的逻辑组，例如控制结构，采用两个空行来分隔方法定义。没有空白的代码很难读，很难理解。

2.2.3.1 **包和引入之间加一个空行**

2.2.3.2 **不同可见性的变量定义之间加一个空行**

2.2.3.3 **方法定义之间加一个空行**

2.2.3.4 **不同可见性的方法组定义之间多加一个空行**

2.2.4 空格

关键字与其后紧挨的括号之间应有一个空格进行分隔。

参数列表的逗号之后应有一个空格进行分隔。

大括号开始之前应有一个空格进行分隔。

所有的二元操作符的前后均应有一个空格，二元操作符的例子有加号，除号，等于符号，赋值符号等。

for **的每一个语句之间应有空格，即** for (expr1; expr2; expr3)

强制类型转换的括号之后应有一个空格。如

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

```
                                        + 1);
```

2.2.5 行宽

行宽不得超过最大行宽，超过最大行宽就需要折行，最大行宽请参见数据一节。

原理：行宽超过最大行宽往往需要用横向滚动条，影响阅读，而最大行宽之内的字符数在大多数阅读器中均能方便地进行阅读及打印。

2.2.6 折行

当一行装不下内容时，需要对这些内容进行折行，折行时遵守以下规则：

2.2.6.1 在分号处折行

2.2.6.2 在操作符前折行

操作符包括+，-，*，/，左括号，逗号，”&&”，”||”，问号,冒号，例外的情况是在逗号后折行。例：

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

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

2.2.6.3 有多个可以选择的地方时，选择层次较高的进行折行

同一层次是指在表达式树中同一级（同一个括号内），层次较高是指在表达式树中较高的一级，以下两个例子可以说明这个规则。

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

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

2.2.6.4 将新行与上一行的同一级的表达式的开始处对齐

此处同一级的含义同上。

2.2.6.5 方法定义的折行

如果方法定义太长，可以在参数处进行折行，下一行可以遵守同级开始处对齐原则。

如：

```
//同级开始处对齐原则
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
                Object andStillAnother) {
   ...
}
//比上一行缩进一级原则
private static synchronized horkingLongMethodName(int anArg,
   Object anotherArg, String yetAnotherArg,
   Object andStillAnother) {
   ...
}
```

2.2.6.6 简单地缩进一级

在使用以上原则时，如果出现太深的缩进，如括号开始处可能在行末附近，这时，只是简单地在上一行的开始处缩进一级。

2.3 注释

2.3.1 Java 注释语句类型

Java 有三种注释语句风格：以 /** 开始， */ 结束的文档注释，以 /* 开始，以 */ 结束的 C 语言风格注释，以及以 // 开始，代码行末尾结束的单行注释。下表是对各类注释语句建议用法的一个概括，也给出了几个例子。

注释语句类型	用法	示例
文档注释	在接口、类、方法和字段声明之前紧靠它们的位置用文档注释进行说明。文档注释由 javadoc 处理，为一个类生成外部注释文档，如下所示。	/** Customer（顾客）顾客是指作为我们的服务及产品的销售对象的任何个人或组织。 @author S.W. Ambler */
C 语言风格注释	采用 C 语言风格的注释语句将无用的代码注释掉。保留这些代码是因为用户可能改变想法，或者只是想在调试中暂时不执行这些代码。	/* 这部分代码已被它前面的代码替代，所以于 1999 年 6 月 4 日被 B. Gustafsson 注释掉。如果两年之后仍未用这些代码，将其删除。 . . .（源代码） */

单行注释	在方法内部采用单行注释语句对业务逻辑、代码片段和临时变量声明进行说明。	// 因为让利活动
		// 从 1995 年 2 月开始，
		// 所以给所有超过 $1000 的
		// 发货单 5% 的折扣。

2.3.2 快速浏览 javadoc

Sun 公司的 Java Development Kit (JDK) 中有一个名为 javadoc 的程序。它可以处理 Java 的源代码文件，并且为 Java 程序产生 HTML 文件形式的外部注释文档。

Javadoc 支持一定数目的标记，标识注释文档中各段起始位置的保留字。详情请参考 JDK javadoc 文档。

本规范支持并且要求有 Javadoc 的注释标记，该标记可由工具自动生成也可由手工制作。

标记	用于	目的
@author name	类、 接口	说明特定某一段程序代码的作者。每一个作者各有一个标记。
@deprecated	类、 方法	说明该类的应用程序编程接口 (API) 已被废弃，因此应不再使用。
@exception name description	方法	说明由方法发出的异常。一个异常采用一个标记，并要给出异常的完整类名。
@param name description	方法	用来说明传递给一个方法的参数，其中包括参数的类型/类和用法。每个参数各有一个标记。
@return description	方法	若方法有返回值，对该返回值进行说明。应说明返回值的类型/类和可能的用途。
@since	类、方法	说明自从有 JDK 1.1 以来，该项已存在了多长时间。
@see ClassName	类、接口、方法、字段	在文档中生成指向特定类的超文本链接。可以并且应该采用完全合法的类名。
@see ClassName#memberfunctionName	类、接口、方法、字段	在文档中生成指向特定方法的超文本链接。可以并且应该采用完全合法的类名。
@version text	类、接口	说明特定一段代码的版本信息。

2.3.3 按目的分的注释类型

注释可按其目的分为外部注释、业务逻辑注释及其它注释。业务逻辑注释是指在类的方法中用于解释业务逻辑的注释，外部注释是指不在类的方法中，是用于对外解释类或类的接口的注释。变量注释是指不用于外部注释中，对局部变量或私有类变量等变量的注释。

2.3.3.1 业务逻辑注释

在代码块（如类的方法，静态方法等）中用于解释业务逻辑的注释称为业务逻辑注释，除特殊情况外，业务逻辑注释一般采用单行注释的方式。

代码数，代码数即语句数，不包括变量定义，与代码行数不同。代码数可用自动化工具获得。

注释数为针对于语句的注释数。注释数与注释的行数不同，在两条语句之间的注释，不管行数多少，都算作一个注释。

注释率=注释数/代码数*100%。

每个方法的注释率的下限请参见数据一节，但语句数少于 3 句的除外。

2.3.3.2 外部注释

不在代码块中，用于对外解释类或类的接口的注释称为外部注释。外部注释包括类的内容注释、公共方法及变量注释、保护方法及变量注释、包方法及变量注释。除非特殊情况，外部注释最好采用内容注释的方式。

外部注释的注释率=内容注释数/定义数*100%。

其中定义数是指公用的类或类的接口的数量。

外部注释的注释率的下限请参见数据一节，如为 100%，则每一个公共的、保护的、包的方法和变量都应加一个内容注释。

原理：每一个公共、保护及包方法及变量都可能被其它人用到，因此关于该项的说明必须有。

2.3.3.3 变量注释

不用于外部注释中，对局部变量或私有类变量等变量的注释称为变量注释。变量注释的注释率=注释数/变量数*100%。

变量注释的注释率的下限请参见数据一节。

3. 源文件内容

每个 Java 源文件只包含一个公共的类或接口，这是 Sun 公司的规定。当该公共类附带有私有类或接口时，可以把它们放在该公共类的同一个文件，但公共类必须出现在它们前面。

源文件由以下三部分组成：

文件头注释部分

包及引入部分

类及接口定义部分

3.1 文件头注释部分

每个 Java 源文件必须以一个 C 语言风格的注释开始，该注释包括文件名，版本信息，日期，变更记录及版权声明。参考格式如下：

```
/*
 * <源文件名>
 *
 * Version information
 *
 * Date
 *
 * Copyright notice
```

*/

3.1.1 文件名

记录源文件名（不包括路径）。

3.1.2 版本信息

版本信息可选，但如果有的话，必须与配置系统的版本信息兼容。

3.1.3 变更记录

变更记录包含变更日期、变更人和变更内容。

3.1.4 版权声明

版权声明是必须的，其内容必须包括：
公司
年份
可用如下格式：
Copyright 2000-2001 Topgroup. All Rights Reserved.

3.2 包及引入部分

Java **源文件的第一条非注释行为** package **语句，包后加一空行，之后可以是一个或多个** import **语句，例如：**

```
package java.awt;

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

3.3 类及接口定义部分

类及接口定义部分开始于一个内容注释以说明类及接口的功能、环境、使用方法等说明。
类或接口与类或接口之间用一个空行间隔。

3.3.1 变量定义

变量定义包括类变量定义及局部变量定义。

3.3.1.1 位置

这两种变量定义都应该在其任用域的一开始处定义，即对于类变量，在类的开始处的大括号之后就对它进行定义，对于方法内的局部变量，在方法定义开始处的大括号之后进行定义。尽量不要在方法内某一块中定义局部变量。
静态变量在最前，按可见性从高到低排列。若有静态块，则静态块在其后，其后是非静态的类变量，按可见性从高到低排列，即按公共、保护、包、私有的顺序进行排列，不同可见性的类变量组之间应用一个空行进行分隔。

3.3.1.2 格式

公共的类变量前必须有一个内容注释说明它。
每个变量定义应该占据一行，并且应该对该变量进行注释，但建议不要进行行尾注释，最好在变量定义之前进行注释。如：

```
// indentation level
int level;
// size of table
```

int size;
就好于
int level, size;
两个同样可见性的变量定义之间不要有空行。
类型不同的变量不要在同一行中，如：
int foo, fooarray[];//**避免**
尽量在定义处对变量进行初始化

3.3.2 方法定义

两个方法定义之间用一个空行分隔。
方法名与参数引导符圆括号之间不要有空格，
方法顺序为：
构造方法
finalize **方法**
初始化方法
static **方法**
公共方法
保护方法
包方法
私有方法
方法定义部分开始于一个内容注释以说明该方法的功能、环境、使用方法等所有重要的有助于理解方法的信息，尽可能在其中加入 Javadoc **标记，当然，这件事有一部分也可能是由设计人员完成的。有返回值的必须加**@return **标记，有参数的必须对每一个参数加**@param **标记，有错误的必须加**@exception **标记，尽量加**@see **标记。这些信息包含但不仅仅局限于以下内容：**

3.3.2.1 **方法说明**

方法说明对方法整体进行说明，包括以下内容：

3.3.2.1.1 成员函数做什么以及它为什么做这个

通过给一个方法加注释，让别人更加容易判断他们是否可以复用代码。注释出方法为什么做这个可以让其他人更容易将你的代码放到程序的上下文中去。也使其他人更容易判断是否应该对你的某一段代码加以修改（有可能他要做的修改与你最初为什么要写这一段代码是相互冲突的）。

3.3.2.1.2 如何在适当情况下调用方法的例子

最简单的确定一段代码如何工作的方法是看一个例子。考虑包含一到两个如何调用方法的例子。

3.3.2.1.3 可用的前提条件和后置条件

前提条件是指一个方法可正确运行的限制条件；后置条件是指一个方法执行完以后的属性或声明。前提条件和后置条件以多种方式说明了在编写方法过程中所做的假设，精确定义了一个方法的应用范围。

3.3.2.2 @param

如果带参数，那么什么样的参数必须传给方法，以及方法将怎样使用它们。
原理：这个信息使其他程序员了解应将怎样的信息传递给一个方法。在 (“**快速浏览** javadoc”)

中讨论的 *javadoc @param* 标识便用于该目的。

3.3.2.3 @return

如果方法有返回值，则应注释出来。

原理：这样可以使其他程序员正确地使用返回值/对象。在 (“**快速浏览** javadoc”) **里讨论的** *javadoc @return* **标识便用于此目的。**

3.3.2.4 @bug

方法中的任何突出的问题都应说明。

原理：以便让其他程序开发者了解该方法的弱点和难点。如果在一个类的多个方法中都存在着同样的问题，那么这个问题应该写在类的说明里。

3.3.2.5 @exception

应说明方法抛出的所有异常。

原理：以便使其他程序员明白他们的代码应该捕获些什么。在 (“**快速浏览** javadoc”) **中讨论的** *javadoc @exception* **标识便用于此目的。**

3.3.3 语句

3.3.3.1 **业务逻辑注释**

除方法注释以外，在方法内部还需加上注释语句来说明你的工作。目的是使方法更易理解、维护和增强。

建议对业务逻辑采用单行注释，因为它可用于整行注释和行末注释。采用 C **语言风格的注释语句去掉无用的代码，因为这样仅用一个语句就可以容易地去掉几行代码。此外，因为** C **语言风格的注释语句很象文档注释符。它们之间的用法易混淆，这样会使代码的可理解性降低。所以，应尽量减少使用它们。**

在方法内，一定要说明：

3.3.3.1.1 控制结构

说明每个控制结构，例如比较语句和循环。你无须读完整个控制结构内的代码才判断它的功能，而仅需看看紧靠它之前的一到两行注释即可。

3.3.3.1.2 代码做了些什么以及为什么这样做。

通常你常能看懂一段代码做了什么，但对于那些不明显的代码，你很少能判断出它为什么要那样做。例如，看完一行代码，你很容易地就可以断定它是在定单总额上打了 5% **的折扣。这很容易。不容易的是为什么要打这个折扣。显然，肯定有一条商业法则说应打折扣，那么在代码中至少应该提到那条商业法则，这样才能使其他开发者理解你的代码为什么会是这样。**

3.3.3.1.3 局部变量

在一个方法内定义的每一个局部变量都应采用一个注释说明它的用法。

3.3.3.1.4 难或复杂的代码

若发现不能或者没有时间重写代码，那么应将方法中的复杂代码详细地注释出来。一般性的经验法则是，如果代码并非显而易见的，则应说明。

3.3.3.1.5 处理顺序

如果代码中有的语句必须在一个特定的顺序下执行，则应保证将这一点注释出来。没有比下面更糟糕的事了：你对一段代码做一点简单的改动，却发现它不工作，于是花了几个小时查找问题，最后发现原来是搞错了代码的执行顺序。

在闭括号后加上注释。常常会发现你的控制结构内套了一个控制结构，而在这个

控制结构内还套了一个控制结构。虽然应该尽量避免写出这样的代码，但有时你发现最好还是要这样写。

原理：问题是闭括号 } 应该属于哪一个控制结构这一点就变得混淆了。一个好消息是，有一些编辑器支持一种特性：当选用了一个开括号后，它会自动地使相应得闭括号高亮显示；一个坏消息是，并非所有的编辑器都支持这种属性。将类似 //end if，//end for，//end switch，& 这样的注释加在闭括号所在行的行后，可以使代码更易理解。

3.3.3.2 单语句

每行最多只允许一条单语句。

原理：便于阅读及调试。

3.3.3.3 复合语句

复合语句是指在两个大括号之内的语句列表。复合语句遵守下列格式要求：

复合语句内部的语句必须比复合语句自身所在的位置上缩进一级。

原理：便于阅读及分清层次关系。

开始大括号（即左大括号）必须在一行的最后一个字符以开始一个复合语句。行尾的开始大括号前跟一个空格。

结束大括号（即右大括号）必须新开始一行并与该复合语句缩进在同一级上，结束大括号后可以有其它内容，但必须在其间加入一个空格。

在 if-else、 for 或 while 语句中，即使只有一条语句，也应该用大括号把它们括起来做为一个复合语句。

原理：这样以后在复合语句中增加语句时也不会因为忘记加大括号而出错了。

3.3.3.4 return 语句

一般情况下在 return 语句中不要使用圆括号，除非它们使 return 语句看起来更清楚，如在三元操作符外加括号以使 return 语句看起来更清楚。如：

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

3.3.3.5 if 语句

if 语句采用下列格式：

```
if (condition) {
    statements;
}

if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

3.3.3.6 for 语句

for 语句采用下列格式：

```
for (initialization; condition; update) {
    statements;
}
```

3.3.3.7 while 语句

while 语句采用下列格式：

```
while (condition) {
    statements;
}
```

3.3.3.8 do-while 语句

do-while 语句采用下列格式：

```
do {
    statements;
} while (condition);
```

3.3.3.9 switch 语句

switch 语句采用下列格式：

```
switch (condition) {
case ABC:
    statements;
    /* falls through */

case DEF:
    statements;
    break;

case XYZ:
    statements;
    break;

default:
    statements;
    break;
}
```

3.3.3.10 try-catch 语句

try-catch 语句采用下列格式：

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

4. 与工具的集成

4.1 Rose

@Roseid ID

4.2 Javadoc

与 JavaDoc 的集成可使文档的说明自动生成并可表现为多种形式。本规范对 Javadoc 的支持有类和接口说明，公共变量说明，方法说明三种说明。

请参见JavaDoc 支持说明。

4.3 配置工具

当前对配置工具的集成未实现。

5. 数据

数据项	单位	数据值
每级缩进	字符数	4
最大行宽	字符数	80
标识符最大字符数	字符数	30
业务逻辑注释的注释率下限	%	30
外部注释的注释率下限	%	100
变量注释的注释率下限	%	80

6. 最佳实践

本节描述在编码中的最佳实践，这些实践并不强制使用，但它们是由技术人员使用多年并被证明行之有效的方法。当然，这些实践的使用有些可能是不适用的，因此需要根据实际情况酌情使用。

6.1 先写注释再写程序

写代码注释的最好方法是在写代码之前就写注释。这使你在写代码之前可以想想代码的功能和运行。而且这样确保不会遗漏注释。另一种方法是边写代码边写注释。因为注释可以使代码更易理解，所以在程序开发的过程中，也可以利用这一点。如果打算花些时间写注释，那么至少你应从这个过程中获得些什么

6.2 在文档注释中加入 HTML 标记

在文档注释中加入 HTML 标记可使最后生成的技术文档的格式丰富而变得更有表现力，因此强烈推荐采用在文档注释中加入 HTML 标记的做法，具体做的时候可在任一有 HTML 编辑功能的编辑器（如 FrontPage）中编辑后再将 HTML 源代码拷贝进源代码编辑器中。但是，由于 Javadoc 可在多种媒体中发布，可能会在一个不支持 HTML 的媒体上进行发布，因此在文档注释中加入 HTML 标记也会带来一些潜在的问题，但在当前环境中不会出现问题。

原理：在文档注释中加入 HTML 标记可使最后生成的技术文档的格式丰富，表达力强。

6.3 尽量在源文件的主类中加入 main 方法

main 方法可以是一个简单的单元测试或类使用的例子，这样有助于帮助别人使用这个类，请随时记住，类写出来首先是给他人阅读，其次是供他人使用的。

原理：可以通过这样提供一个单元测试的驱动程序或可以是一个使用的例子。

6.4 对每种类型的类写出一个模板

这些类型可能是：Applet，Application，Form 等。这一个实践也可能由 IDE 环境直接提供，如 JBuilder 直接在 New 时会提示采用某种模板。

原理：因为不必每次都重写，而模板是遵循规范的，因此可以与规范更加符合。

6.5 尽量不要使用公共的类变量

原理：因为直接暴露类变量会影响类对自身结构的控制。另外因为外部可以直接修改类变量，类的方法也不能对暴露的类变量作任何假设。另一方面从效率上来讲，由于一些现代的虚拟机会自动优化直接访问类变量的 getXXX，setXXX 类型的函数，效率上也不会产生问题。

6.6 使用 Type[] arrayName 而不要使用 Type arrayName[]

这一点可能 C 或 C++接受有一点困难。

原理：类型名可以更准确地反映类型。

6.7 对于对象的比较慎用==，使用 Object.equals()方法

原理：多数情况下 equals 才能工作正常。