SUNGARD 金仕达

JAVA 编码规范

组织标准软件过程文档

文档标识

文档名称	JAVA 编码规范
版本号	<>
状况	☐ 草案 ☐ 评审过的 ☐ 更新过的 ☑ 定为基线的

文档修订历史

版本	日期	描述	文档所有者
	<yyyy-mm-dd>	<简要说明在此版本中的发生的变化以及发生变化的原因。同时列出此版本评审人和核准人的姓名。>	

此版本文档的正式核准

姓名	签字	日期

分发控制

副本	接受人	机构

目 录

1. 排版

1-1：程序块要采用缩进风格编写，缩进的空格数为 4 个。

说明：通常不要使用制表符，不同的编辑工具对制表符的解释可能不一样；对于由开发工具自动生成的代码可以有不一致。

1-2：相对独立的程序块之间、变量说明之后必须加空行。

示例：如下例子不符合规范。

```
if (someMethod(cond)){
    ... // program code
}
index = data[index].index;
content  = data[index].content;
```

应如下书写

```
if (someMethod(cond)){
    ... // program code
}

index = data[index].index;
content  = data[index].content;
```

1-3：尽量避免一行的长度超过 80 个字符，因为很多终端和工具不能很好的处理。当一个表达式无法容纳在一行内时，可以依据如下一般规则断行：

- 在一个逗号后面断开
- 在一个操作符前面断开
- 宁可选择较高级别(higher-level)的断开，而非较低级别(lower-level)的断开
- 新的一行应该与上一行同一级别表达式的开头处对齐
- 如果以上规则导致你的代码混乱或者使你的代码都堆挤在右边，那就代之以缩进 8 个空格。

示例：

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

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

以下是两个断开算术表达式的例子。前者更好，因为断开处位于括号表达式的外边，这是个较高级

别的断开。

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

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

以下是两个缩进方法声明的例子。前者是常规情形。后者若使用常规的缩进方式将会使第二行和第三行移得很靠右，所以代之以缩进 8 个空格

```
//CONVENTIONAL INDENTATION
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
               Object andStillAnother) {
    ...
}

//INDENT 8 SPACES TO AVOID VERY DEEP INDENTS
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother) {
    ...
}
```

if 语句的换行通常使用 8 个空格的规则，因为常规缩进(4 个空格)会使语句体看起来比较费劲。比如：

```
//DON' T USE THIS INDENTATION
if ((condition1 && condition2)
    || (condition3 && condition4)
    ||!(condition5 && condition6)) { //BAD WRAPS
     doSomethingAboutIt();             //MAKE THIS LINE EASY TO MISS
}

//USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

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

1-4：不允许把多个短语句写在一行中，即一行只写一条语句。

示例：如下例子不符合规范。

```
rect.length = 0;  rect.width = 0;
```

应如下书写

```
rect.length = 0;
rect.width  = 0;
```

1-5：if、for、do、while、case、switch、default 等语句自占一行，且 if、for、do、while 等语句的执行语句部分无论多少都要加括号{}。

示例：如下例子不符合规范。

```
if (userName == null) return;
```

应如下书写：

```
if (userName == null){
    return;
}
```

1-6：函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格，case 语句下的情况处理语句也要遵从语句缩进要求。

1-7：程序块的分界符‘{’应紧随在引用程序块的语句后面；分界结束符‘}’独占一行，且与引用程序块的语句左对齐。在函数体的开始、类的定义、以及 if、for、do、while、switch、case 语句中的程序都要采用如上的缩进方式。

示例：如下例子不符合规范。

```
for (...)
{
    ... // program code
}

if (...)
    {
    ... // program code
    }

public void example_fun(){
```

```
    ... // program code
    }
```

应如下书写。

```
for (...) {
    ... // program code
}

if (...) {
    ... // program code
}

public void example_fun(){
    ... // program code
}
```

1-8：在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格。

说明：采用这种松散方式编写代码的目的是使代码更加清晰。

由于留空格所产生的清晰性是相对的，所以，在已经非常清晰的语句中没有必要再留空格，如果语句已足够清晰则括号内侧(即左括号后面和右括号前面)不需要加空格，多重括号间不必加空格。

在长语句中，如果需要加的空格非常多，那么应该保持整体清晰，而在局部不加空格。给操作符留空格时不要连续留两个以上空格。

2. 注释

2-1：Java 有两种注释：实现注释(implementation comments)和文档注释(document comments)。实现注释是使用/*...*/或者//界定的注释。文档注释是 Java 独有的，并由/...*/界定。文档注释可以通过 javadoc 工具转换成 HTML 文件。实现注释用以注释代码或者实现细节。文档注释从实现自由的角度描述代码的规范，用于文档描述。**

说明：注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。

2-2：类头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、功能、与其它文件的关系、修改日志等，头文件的注释中还应有函数功能简要说明。

示例：下面这段头文件的头注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

```
/*
* Copyright (C), 2000-2003, Kingstar Co., Ltd.
* File name:        // 文件名
* Date:             //完成日期
* Description:     // 用于详细说明此程序文件完成的主要功能，与其他模块
                    // 或函数的接口，输出值、取值范围、含义及参数间的控
                    // 制、顺序、独立或依赖等关系
* Modify History:  // 修改历史记录列表，每条修改记录应包括修改日期、修改
                      // 者及修改内容简述
*/
/**
* @author           //作者
* @version          //版本
*/
```

在必要的时候，可以选择添加：

```
@see className      //参考的相关类
@see className#functionName        //参考相关类的某个特定方法
```

2-3：成员函数头部应进行注释，列出：函数的目的/功能、输入参数、返回值、异常、修改历史等。示例：下面这段函数的注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

```
/**
 * Description:          // 函数功能、性能等的描述
 * @param                 // 输入参数说明，包括每个参数的类型、作
                          // 用、取值说明及参数间关系
```

```
 * @return              //函数返回值的说明
 * @exception           //异常的说明（如果有异常抛出则使用，不是必须的）
 * Modify History:    //修改历史
*/
```

在注释中还可以包括已经发现的 Bug 描述。由于有些 Bug 可能暂时不影响操作和程序的执行，但是以后会完善，这些信息也可以增加在成员函数的注释中。

2-4：注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

示例：如下例子不符合规范。

例 1：

```
/* get replicate sub system index and net indicator */


index = data[index].index;
indicator = data[index]. indicator;
```

例 2：

```
index = data[index].index;
indicator = data[index]. indicator;
/* get replicate sub system index and net indicator */
```

应如下书写

```
/* get replicate sub system index and net indicator */
index = data[index]. index;
indicator = data[index].indicator;
```

2-5：对于所有有物理含义的变量、常量，如果其命名不是充分自注释的，在声明时都必须加以注释，说明其物理含义。变量、常量的注释应放在其右方，且常用‘//’的形式。

示例：

```
final static int  MAX_ACT_TASK_NUMBER = 1000 // active task number
```

2-6：注释与所描述内容进行同样的缩排。

说明：可使程序排版整齐，并方便注释的阅读与理解。

示例：如下例子，排版不整齐，阅读稍感不方便。

```
void exampleFunction()
{
/* code one comments */
    CodeBlock One
```

```
        /* code two comments */
    CodeBlock Two
}
```

应改为如下布局。

```
void exampleFunction()
{
    /* code one comments */
    CodeBlock One

    /* code two comments */
    CodeBlock Two
}
```

2-7：对变量的定义和分支语句（条件分支、循环语句等）必须编写注释。

说明：这些语句往往是程序实现某一特定功能的关键，对于维护人员来说，良好的注释帮助更好的理解程序，有时甚至优于看设计文档。

1：避免在一行代码或表达式的中间插入注释。

说明：除非必要，不应在代码或表达中间插入注释，否则容易使代码可理解性变差。

2：通过对函数或过程、变量、结构等正确的命名以及合理地组织代码的结构，使代码成为自注释的。

说明：清晰准确的函数、变量等的命名，可增加代码可读性，并减少不必要的注释。

3：在代码的功能、意图层次上进行注释，提供有用、额外的信息。

说明：注释的目的是解释代码的目的、功能和采用的方法，提供代码以外的信息，帮助读者理解代码，防止没必要的重复注释信息。

示例：如下注释意义不大。

```
/* if receiveFlag is TRUE */
if (receiveFlag)
```

而如下的注释则给出了额外有用的信息。

```
/* if mtp receive a message from links */
if (receiveFlag)
```

4：在程序块的结束行右方加注释标记，以表明某程序块的结束。

说明：当代码段较长，特别是多重嵌套时，这样做可以使代码更清晰，更便于阅读。

示例：参见如下例子。

```
if (...)
{
    // program code

    while (index < MAX_INDEX)
```

```
    {
        // program code
    } // end of while (index < MAX_INDEX)  // 指明该条 while 语句结束
} // end of  if (...) // 指明是哪条 if 语句结束
```

5：注释应考虑程序易读及外观排版的因素，使用的语言若是中、英兼有的，建议多使用中文，除非能用非常流利准确的英文表达。

说明：注释语言不统一，影响程序易读性和外观排版，出于对维护人员的考虑，建议使用中文。

3. 标识符命名

3-1：标识符的命名要清晰、明了，有明确含义，同时使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。

说明：较短的单词可通过去掉“元音”形成缩写；较长的单词可取单词的头几个字母形成缩写；一些单词有大家公认的缩写。

示例：如下单词的缩写能够被大家基本认可。

temp 可缩写为 tmp;

message 可缩写为 msg;

amount 可缩写为 amt;

3-2：命名中若使用特殊约定或缩写，则要有注释说明。

说明：应该在源文件的开始之处，对文件中所使用的缩写或约定，特别是特殊的缩写，进行必要的注释说明。

3-3：自己特有的命名风格，要自始至终保持一致，不可来回变化。

说明：个人的命名风格，在符合所在项目组或产品组的命名规则的前提下，才可使用。（即命名规则中没有规定到的地方才可有个人命名风格）。

3-4：包的命名：一个唯一包名的前缀总是全部小写的 ASCII 字母并且是一个顶级域名，通常是 com，edu，gov，mil，net，org，或 1981 年 ISO 3166 标准所指定的标识国家的英文双字符代码。包名的后续部分根据不同机构各自内部的命名规范而不尽相同。这类命名规范可能以特定目录名的组成来区分部门(department)，项目(project)，机器(machine)，或注册名(login names)。

示例：常见的包名开头多为公司域名的逆序，比如：

```
com.kingstar
com.kingstar.bank
```

3-5：类的命名：类名是个名词，采用大小写混合的方式，每个单词的首字母大写。尽量使类名简洁而富于描述。接口的命名规则与类相同。

示例：

```
class Raster
class ImageSprite

interface RasterDelegate
interface Storing
```

3-6：方法的命名：方法名是一个动词，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。

示例：

```
run()
runFast()
```

getBackground()

3-7：变量的命名：变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。变量名应简短且富于描述。变量名的选用应该易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量，比如：循环变量。临时变量通常被取名为 i，j，k，它们一般用于整型；c，d，e，它们一般用于字符型。

示例：如下的命名是错误的。

```
char i;
int _width;
int $height;
double MyWidth;
```

而如下的命名是正确的。

```
int width;
doule myWidth;
for(int i=...)
```

3-8：常量命名：常量的声明，应该全部大写，单词间用下划线隔开。

示例：

```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
static final int GET_THE_CPU = 1;
```

1：除非必要，不要用数字或较奇怪的字符来定义标识符。

示例：如下命名，使人产生疑惑。

```
Static final EXAMPLE_0_TEST;
Static final EXAMPLE_1_TEST;
void setSleep12(int value);
```

应改为有意义的单词命名

```
static final EXAMPLE_UNIT_TEST;
static final EXAMPLE_ASSERT_TEST;
void setMsgSleep(int value);
```

4. 声明

4-1：推荐每行声明一个变量，因为这样有利于注释。

示例：

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

4-2：尽量在声明局部变量的同时初始化。唯一不这么做的理由是变量的初始值依赖于某些先前发生的计算。

4-3：只在代码块的开始处声明变量。（一个代码块是指任何被包含在大括号‘{’和‘}’中间的代码。）不要在首次用到该变量时才声明之。这会把注意力不集中的程序员搞糊涂，同时会妨碍代码在该作用域内的可移植性。

示例：

```
void myMethod() {
      int int1 = 0;          // beginning of method block
      if (condition) {
          int int2 = 0;      // beginning of "if" block
       ...
      }
}
```

该规则的一个例外是 for 循环的索引变量

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

5. 编程惯例

5-1：注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认优先级。

说明：防止阅读程序时产生误解，防止因默认的优先级与设计思想不符而导致程序出错。

示例：下列语句中的表达式

```
if ((a | b) && (a & c))
```

如果书写为

```
if （a | b && a & c）
```

虽然不会出错，但语句不易理解；

5-2：避免使用不易理解的数字，用有意义的标识来替代。涉及物理状态或者含有物理意义的常量，不应直接使用数字，必须用有意义的常量来代替。

示例：如下的程序可读性差。

```
if (Trunk[index].trunkState == 0)
{
    Trunk[index].trunkState = 1;
    ...  // program code
}
```

应改为如下形式。

```
static final TRUNK_IDLE 0;
static final TRUNK_BUSY 1;

if (Trunk[index].trunkState == TRUNK_IDLE)
{
    Trunk[index].trunkState = TRUNK_BUSY;
    ...  // program code
}
```

5-3：避免在一个语句中给多个变量赋相同的值。它很难读懂。

示例：避免使用类似下面语句中的表达式

```
fooBar.rightChar = barFoo.leftChar = 'c';
```

1：源程序中关系较为紧密的代码应尽可能相邻。

说明：便于程序阅读和查找。

示例：以下代码布局不太合理。

```
rect.length = 10;
```

```
charPosition = str;
rect.width = 5;
```

若按如下形式书写，可能更清晰一些。

```
rect.length = 10;
rect.width = 5; // 矩形的长与宽关系较密切，放在一起。
charPosition = str;
```

2：不要使用难懂的技巧性很高的语句，除非很有必要时。

说明：高技巧语句不等于高效率的程序，实际上程序的效率关键在于算法。

6. 代码范例

下面的范例展示了如何合理布局一个包含单一公共类的 Java 源程序；例子来源为 SUN 公司。

```
/*
 * @(#)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;

/**
 * Class description goes here.
 *
 * @version  1.82 18 Mar 1999
 * @author   Firstname Lastname
 */
public class Blah extends SomeClass {
    /* A class implementation comment can go here. */

    /** classVar1 documentation comment */
    public static int classVar1;
```

```
/**
 * classVar2 documentation comment that happens to be
 * more than one line long
 */
private static Object classVar2;

/** instanceVar1 documentation comment */
public Object instanceVar1;

/** instanceVar2 documentation comment */
protected int instanceVar2;

/** instanceVar3 documentation comment */
private Object[] instanceVar3;

/**
 * ...constructor Blah documentation comment...
 */
public Blah() {
    // ...implementation goes here...
}

/**
 * ...method doSomething documentation comment...
 */
public void doSomething() {
    // ...implementation goes here...
}

/**
 * ...method doSomethingElse documentation comment...
 * @param someParam description
 */
public void doSomethingElse(Object someParam) {
    // ...implementation goes here...
```

```
        }
    }
```