HOPECHART

Java 编码规范

编号：HQ-C-33-04　　版本：C/1　　发布号：

编制：　　日期：2021.12.28

审核：　　日期：2021.12.29

批准：　　日期：2021.12.29

保密等级：□绝密 ■机密 □敏感 □公开

受控盖章区	作废盖章区
受控文件	

基本信息

文档名称	HQ-C-33-04 Java 编码规范		
上级文档信息	HQ-B-33 软件控制程序		
文档属性	工作流程□ 技术规范■ 规章制度□	维护部门	研发中心

修改记录

序号	修改时间	修改人	主要修改	存档版本
1	2016.08.30	上官丹萍	ISO 9001-2015 转版	B/0
2	2018.01.22	俞月卿	将“杭州鸿泉数字设备有限公司”改为“杭州鸿泉物联网技术股份有限公司”	B/1
3	2019.09.24	叶小平	根据质量手册过程识别修改进行梳理修订	C/0
4	2021.12.23	陈玲	修改文件模版要求，增加文档属性	C/1
5				
6				
7				
8				
9				
10				

目录

1 目的

为了改善软件的可读性，可以让开发人员尽快而彻底地理解新的代码，确保每个软件开发人员必须一致遵守编码规范，特制定本规范。

2 范围

本规定适用于指导本公司所有产品的 Java 编码过程。

3 职责

3.1 研发中心 Java 开发工程师：按照本规范执行编码。

3.2 研发中心高级工程师：负责进行代码规范评审。

4 定义

无

5 流程图

无

6 内容

6.1 文件结构

6.1.1 Java 源文件

Java 源文件主要由三部分内容构成：

- 文件头注释（参见"文件头注释"）
- 包和引入语句（参见"包和引入语句"）
- 类和接口声明（参见"类和接口声明"）

一个 Java 源文件中，可以声明多个相关联的类或接口。其中只能有一个是公共的。公共类或接口必须是源文件中的第一个类。同时，源文件也要以该公共类的类名命名。

6.1.2 文件头注释

所有的源文件都应该在开头有一个“文件头注释”，其中列出文件名、内容、修改记录。修改记录包括日期(YYYY/MM/DD)、创建者/修改者、修改内容，新增的修改记录排在最前，如下：

```
/*
 * 模块:    [该类所处的模块功能说明] <br>
 * 用途:    [该类实现的功能]说明 <br>
 * 作者:    [作者名称] <br>
 * 日期:    [文件创建日期] <br>
 * 版权:    Copyright (c) 1999-2011 ChiTone,Corp. <br>
 */
```

6.1.3 包和引入语句

在 Java 源文件中，第一个非注释行是包语句。在它之后可以跟引入语句。

包语句的“包名称”指明了源文件所处的路径。“包名称”必须以 com.job5156 开头，以“文件所属模块名”为组合，例如营业缴费业务实现类源文件：

```
package com.job5156.sns.common;

import java.util.List;
```

其中：“com.job5156”为公共部分

“sns.common”为文件所属模块

6.1.4 类和接口声明

“类和接口声明”主要有以下几部分，它们出现的先后次序如下表。实例参见"Java 源文件范例"中一个包含注释的例子。

	类/接口声明的各部分	说明
1	类/接口文档注(/**……*/)	该注释中所需包含的信息，参见"文档注释规范"
2	类/接口的声明	
3	类/接口实现注释(/*……*/) 如果有必要的话	该注释应包含任何有关整个类或接口的信息，而这些信息又不适合作为类/接口文档注释。
4	类的(静态)变量	首先是类的公共变量，随后是保护变量，再后是包一级别的变量(没有访问修饰符)，最后是私有变量。
5	实例变量	首先是公共级别的，随后是保护级别的，再后是包一级别的(没有访问修饰符)，最后是私有级别的。
6	构造器	
7	方法	这些方法应该按“功能”，而非“作用域或访问权限”分组。例如，一个私有的类方法可以置于两个公有的实例方法之间。其目的是为了更便于阅读和理解代码。

6.2 程序的版式

版式虽然不会影响程序的功能，但会影响可读性。程序的版式追求清晰、美观，是程序

风格的重要构成因素。

可以把程序的版式比喻为“书法”。好的“书法”可让人对程序一目了然，看得兴致勃勃。差的程序“书法”如螃蟹爬行，让人看得索然无味，更令维护者烦恼有加。

6.2.1 空行

空行起着分隔程序段落的作用。空行得体（不过多也不过少）将使程序的布局更加清晰。空行不会浪费内存，虽然打印含有空行的程序是会多消耗一些纸张，但是值得。所以不要舍不得用空行。

- **【规则】**在每个类/接口声明之后、每个函数定义结束之后都要加空行。参见示例 6-2-1（a）
- **【规则】**在一个函数体内，局部变量定义和方法的第一条语句之间应加空行分隔。参见示例 6-2-1（b ）
- **【规则】**在一个函数体内，在两个逻辑段之间应加空行分隔。参见示例 6-2-1（b ）
- **【规则】**在块注释或单行注释之前，都要加空行。参见示例 6-2-1（a）

```
// 空行
/*
  *Here is a block comment.
  */
void Function1(…){
    …
}
// 空行
void Function2(…){
    …
}
// 空行
void Function3(…){
    …
}
```

示例 6-2-1 (a) 函数之间的空行

```
// 空行
while (condition){
  int var1;
  int var2;
  // 空行
  statement1;
  // 空行
  if (condition) {
      statement2;
  }
  else{
      statement3;
  }
  // 空行
  statement4;
}
```

示例 6-2-1 (b) 函数内部的空行

6.2.2 代码行

- **【规则】**一行代码只做一件事情，如只定义一个变量，或只写一条语句。这样的代码容易阅读，并且方便于写注释。
- **【规则】**if、for、while、do、swich 等语句自占一行，执行语句不得紧跟其后。这样可以防止书写失误。

示例 6-2-2（a）为风格良好的代码行，示例 6-2-2（b）为风格不良的代码行。

int width; // 宽度
int height; // 高度
int depth; // 深度
	int width, height, depth; // 宽度高度深度

x = a + b;
y = c + d;
z = e + f;
	X = a + b; y = c + d; z = e + f;

if (width < height) {
  dosomething();
}
	if (width < height) dosomething();

for (initialization; condition; update){
  dosomething();
}
// 空行
other();
	for (initialization; condition; update)
	  dosomething();
	other();

示例 6-2-2(a) **风格良好**的代码行　　示例 6-2-2(b) **风格不良**的代码行

✧ **【建议】**尽可能在定义变量的同时初始化该变量（就近原则）

如果变量的引用处和其定义处相隔比较远，变量的初始化很容易被忘记。如果引用了未被初始化的变量，可能会导致程序错误。本建议可以减少隐患。例如

```
int width = 10;      // 定义并初始化 width
int height = 10;     // 定义并初始化 height
int depth = 10;      // 定义并初始化 depth
```

6.2.3　代码行内的空格

- **【规则】**关键字之后要留空格。像 abstract、static、final、case 等关键字之后至少要留一个空格，否则无法辨析关键字。像 if、for、while 等关键字之后应留一个空格再跟左括号‘(’，以突出关键字。
- **【规则】**函数名之后不要留空格，紧跟左括号‘(’，以与关键字区别。
- **【规则】**‘(’向后紧跟，‘)’、‘,’、‘;’向前紧跟，紧跟处不留空格。
- **【规则】**‘,’之后要留空格，如 Function(x, y, z)。如果‘;’不是一行的结束符号，其后要留空格，如 for (initialization; condition; update)。
- **【规则】**赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=” “>=”、“<=”、“+”、“*”、“%”、“&&”、“||”、“<<”，“^”等二元操作符的前后应当加空格。
- **【规则】**一元操作符如“!”、“~”、“++”、“--”等前后不加空格。

- 【规则】像“[]”、“.”这类操作符前后不加空格。

✧ 【建议】对于表达式比较长的 for 语句和 if 语句，为了紧凑起见可以适当地去掉一些空格，如 for (i=0; i<10; i++)和 if ((a<=b) && (c<=d))

```
void Func1(int x, int y, int z);          // 良好的风格
void Func1 (int x,int y,int z);           // 不良的风格

if (year >= 2000)                         // 良好的风格
if(year>=2000)                            // 不良的风格
if ((a>=b) && (c<=d))                     // 良好的风格
if(a>=b&&c<=d)                            // 不良的风格

for (i=0; i<10; i++)                      // 良好的风格
for(i=0;i<10;i++)                         // 不良的风格
for (i = 0; I < 10; i ++)                 // 过多的空格

x = a < b ? a : b;                        // 良好的风格
x=a<b?a:b;                                // 不良的风格

array[5] = 0;                             // 不要写成 array [ 5 ] = 0;
a.Function();                             // 不要写成 a . Function();
```

示例 6-2-3 代码行内的空格

6.2.4 长行拆分

- 【规则】代码行最大长度宜控制在 70 至 80 个字符以内。代码行不要过长，否则眼睛看不过来，也不便于打印。
- 【规则】长语句通常在一个逗号后面断开，拆分成新行。
- 【规则】长表达式要在低优先级操作符处拆分成新行，操作符放在新行之首（以便突出操作符）。
- 【规则】拆分出的新行要进行适当的缩进，使排版整齐，语句可读。

```
if ((very_longer_variable1 >= very_longer_variable12)
    && (very_longer_variable3 <= very_longer_variable14)
    && (very_longer_variable5 <= very_longer_variable16)){
    dosomething();
}

virtual CMatrix CMultiplyMatrix (CMatrix leftMatrix,
                                 CMatrix rightMatrix);

for (very_longer_initialization;
     very_longer_condition;
     very_longer_update){
    dosomething();
}
```

示例 6-2-4 长行的拆分

6.2.5 对齐缩进

语句块是包含在"{ “和” }"中的语句序列。语句块的对齐缩进采用 Java 风格的对齐缩进。

- 【规则】 语句块中，左大括号"{"应位于语句起始行的行尾；右大括号"}"应另起一行并与语句首行对齐。
- 【规则】被括其中的语句应该较之缩进一个层次，同一层次的代码要有相同的缩进值，缩进统一为 4 个空格，不论任何位置的缩进，必须以 4 个空格为单位；TAB 键统一使用 4 个空格代替。

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

示例 6-2-5 对齐缩进

6.3 命名规范

命名规范使程序更易读，从而更易于理解，也利于记忆与阅读，减小沟通成本。

6.3.1 类(Class)

类名是一个名词，采用大小写混合的方式，每个单词的首字母大写。尽量使你的类名简洁而富于描述。使用完整单词，避免缩写词(除非该缩写词被更广泛使用，像 URL，HTML)。特殊地，系统表类，需要前跟“Sys”前缀。如：

```
    class Raster;
    class ImageSprite;
    class Operator;
    class SysOperator    //系统表类
```

注意：必须使用英文命名，切忌使用汉语拼音来命名。

6.3.2 接口(Interface)

“接口”命名规则与“类”相似。

6.3.3 方法

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。如：

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

```
    getBackground();
```

注意：必须使用英文命名，切忌使用汉语拼音来命名。

6.3.4 变量

变量名均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。变量名绝不允许以下划线或美元符号开头。变量名最好采用英文单词或其组合，便于记忆和阅读。切忌使用汉语拼音来命名。用词应当简短且准确。如：

```
    float width;          //良好的风格
    float myWidth;        //良好的风格
    float _width;         //不良的风格（使用下划线开头）
    float changdu;        //不良的风格（使用拼音）
```

尽量避免单个字符的变量名，除非是一次性的临时变量，如：for (int i=0; i<10; ++i)。临时变量通常被取名为 i，j，k，m 和 n，它们一般用于整型；c，d，e，它们一般用于字符型。

6.3.5 常量

常量名，应该全部大写，单词间用下划线隔开，并用 static final 修饰词修饰。

注意：绝不允许在代码中使用魔术数字，请定义成常量再使用常量的名称。特殊情况除外，如：for 循环中作为计数器值的数字常量-1,0 和 1。

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

6.4 编写规范

遵守“编写规范”，可使你的程序结构更加优美。

6.4.1 声明

6.4.1.1 每行声明一个变量

推荐一行一个声明，因为这样更清晰，也利于写注释。

示例 6-4-1-1（a）为风格良好的代码行，示例 6-4-1-1（b）为风格不良的代码行。

```
int width;    // 宽度
int height;   // 高度
int depth;    // 深度
```

示例 6-4-1-1(a) **风格良好**的代码行

```
int width, height, depth; // 宽度高度深度
```

示例 6-4-1-1(b) **风格不良**的代码行

6.4.1.2 在声明处初始化（就近原则）

如果变量的引用处和其定义处相隔比较远，变量的初始化很容易被忘记。如果引用了未

被初始化的变量，可能会导致程序错误。本建议可以减少隐患。

例如：

```
int width = 10;      // 定义并初始化 width
int height = 10;     // 定义并初始化 height
int depth = 10;      // 定义并初始化 depth
```

6.4.1.3 声明的布局

只在代码块的开始处声明变量。（一个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。

例如：

```
void myMethod() {
    int int1 = 0;              // beginning of method block

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

6.4.2 语句

6.4.2.1 简单语句

每行至多包含一条语句，例如：

```
argv++;          // Correct
argc--;          // Correct
argv++; argc--;          // AVOID!
```

6.4.2.2 复合语句

复合语句是包含在大括号中的语句序列，形如"{ 语句 }"。例如下面各段：

- 被括其中的语句应该较之复合语句缩进一个层次
- 左大括号"{"应位于复合语句起始行的行尾；右大括号"}"应另起一行并与复合语句首行对齐。
- 大括号可以被用于所有语句，包括单个语句，只要这些语句是诸如 if-else 或 for 控制结构的一部分。这样便于添加语句而无需担心由于忘了加括号而引入 bug。

6.4.2.3 返回语句

一个带返回值的 return 语句不使用小括号"()"，除非它们以某种方式使返回值更为显见。

例如：

```
return;

return myDisk.size();

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

6.4.2.4 if，if-else，if else-if else 语句

if-else 语句应该具有如下格式：

```
if (condition) {
      statements;
}

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

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

此外，if 语句还要遵循以下规则：

- 【规则】if 语句总是用"{"和"}"括起来，这样可以防止书写失误。

```
if (width < height) {
  dosomething();
}
```

示例 6-4-2-4(a) **风格良好**的代码行

```
if (width < height) dosomething();
```

示例 6-4-2-4(b) **风格不良**的代码行

- 【规则】对多条件的判断语句，每个条件语句和子条件要用上括号。

如：if((0 == a) && ((TRUE == b) || (c == d)))

- 【规则】在判断语句中，尽量将常量写在左值，而不是右值，如：

如：if (0 == count) // 即使写成 0 = count 也能在编译期发现

而不是写成：

if (count == 0) // 如果写成 count = 0 则不容易发

现问题

- 【规则】避免在判断语句中字符和整型变量的直接比较，要显式地进行类型转换。

6.4.2.5 for 语句

一个 for 语句应该具有如下格式：

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

此外，for 语句还要遵循以下规则：

- 【规则】在 for 语句的初始化或更新子句，避免因使用三个以上变量，而导致复杂度提高。若需要，可以在 for 循环之前(为初始化子句)或 for 循环末尾(为更新子句)使用单独的语句。

6.4.2.6 while 语句

一个 while 语句应该具有如下格式

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

6.4.2.7 do-while 语句

一个 do-while 语句应该具有如下格式：

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

6.4.2.8 switch 语句

一个 switch 语句应该具有如下格式：

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

case XYZ:
    statements;
    break;

default:
```

```
        statements;
        break;
    }
```

每当一个 case 顺着往下执行时(因为没有 break 语句)，通常应在 break 语句的位置添加注释。上面的示例代码中就包含注释/* falls through */。

6.4.2.9 try-catch 语句

一个 try-catch 语句应该具有如下格式：

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

一个 try-catch 语句后面也可能跟着一个 finally 语句，不论 try 代码块是否顺利执行完，它都会被执行。

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

注意：代码中通过 try/catch 捕捉异常，catch 所捕捉的异常顺序必须遵循类派生相反顺序，否则将会导致部分异常无法捕捉或捕捉处理的代码错误

6.4.3 函数设计

函数设计应遵循以下规则：

- **【规则】**每个函数的代码行数控制在 120 左右，最好不要超过 300 行。
- **【规则】**要显式地给出函数或过程的返回值类型。
- **【规则】**在一个函数的嵌套层次最好不要超过三层，最多不超过六层。
- **【规则】**不可继承或重载的函数用 final 修饰。

6.4.4 类设计

类设计应遵循以下规则：

- **【规则】**类的成员个数控制在 15 个以内，最好不要超过 30 个。
- **【规则】**尽量不要把类的成员变量声明为公有，以充分发挥面向对象的数据封装功能。
- **【规则】**不允许使用匿名类或内部类。

- 【规则】避免用一个对象访问一个类的静态变量和方法。应该用类名替代。

```
如：   classMethod();          //OK
       AClass.classMethod();   //OK
       anObject.classMethod(); //AVOID!
```

6.5 注释规范

Java 程序注释大致可分为两类：实现注释(implementation comments)和文档注释(document comments)。

- 实现注释：是那些在 C++中见过的，使用/*...*/和//界定的注释。“实现注释”用以说明代码或者实现细节（实现注释不生成 API 文档）。
- 文档注释：(被称为"doc comments")是 Java 独有的，并由/**...*/界定。通常可用来生成 API 文档。(可以通过 javadoc 工具转换成 API 文档，只能为 public（公共）和 protected（受保护）成员生成注释文档）

6.5.1 实现注释

“实现注释”通常是不适宜出现在 API 文档中的注释，如：程序的实现细节等。程序可以有 4 种实现注释的风格：

- 块注释(block)。
- 单行注释(single-line)。
- 尾端注释(trailing)。
- 行末注释(end-of-line)。

注意：频繁的注释有时反映出代码的低质量。当你觉得被迫要加注释的时候，考虑一下重写代码使其更清晰。

6.5.1.1 块注释

“块注释”通常用于提供对文件，方法，数据结构和算法的描述。块注释被置于每个文件的开始处以及每个方法之前。它们也可以被用于其他地方，比如方法内部。在功能和方法内部的块注释应该和它们所描述的代码具有一样的缩进格式。

块注释之首应该有一个空行，用于把块注释和代码分割开来，比如：

```
// 空行
/*
 * Here is a block comment.
 */
```

6.5.1.2 单行注释

“单行注释”可以显示在一行内，并与其后的代码具有一样的缩进层级。如果一个注释不能在一行内写完，就该采用块注释(参见"块注释")。单行注释之前应该有一个空行。以下是一个 Java 代码中单行注释的例子：

```
if (condition) {
   // 空行
   /* Handle the condition. */
   ...
}
```

6.5.1.3 尾端注释

“尾端注释”可以与它们所要描述的代码位于同一行，但是应该有足够的空白来分开代码和注释。若有多个“尾端注释”出现于大段代码中，它们应该具有相同的缩进。

以下是一个 Java 代码中尾端注释的例子：

```
if (a == 2) {
      return TRUE;                  /* special case */
} else {
      return isPrime(a);            /* works only for odd a */
}
```

6.5.1.4 行末注释

“行末注释”，使用注释界定符"//"，可以注释掉整行或者一行中的一部分。它一般不用于连续多行的注释文本；然而，它可以用来注释掉连续多行的代码段。以下是所有三种风格的例子：

```
if (foo > 1) {

      // Do a double-flip.
      ...
}
else {
      return false;               // Explain why here.
}

//if (bar > 1) {
//
//      // Do a triple-flip.
//      ...
//}
```

```
//else {
//      return false;
//}
```

6.5.2 文档注释

“文档注释”通常用于描述 Java 的类、接口、构造器，方法，以及字段(field)，并可通过 javadoc 工具生成相应 API 文档。每个文档注释都会被置于注释定界符/**...*/之中，一个注释对应一个类、接口或成员。该注释应位于声明之前：

```
/**
 * The Example class provides ...
 */
public class Example { ...
```

注意顶层(top-level)的类和接口是不缩进的，而其成员是缩进的。描述类和接口的文档注释的第一行(/**)不需缩进；随后的文档注释每行都缩进 1 格(使星号纵向对齐)。成员，包括构造函数在内，其文档注释的第一行缩进 4 格，随后每行都缩进 5 格。

若你想给出有关类、接口、变量或方法的信息，而这些信息又不适合写在文档中，则可使用实现块注释(见 5.1.1)或紧跟在声明后面的单行注释(见 5.1.2)。例如，有关一个类实现的细节，应放入紧跟在类声明后面的实现块注释中，而不是放在文档注释中。

文档注释不能放在一个方法或构造器的定义块中，因为 Java 会将位于文档注释之后的第一个声明与其相关联。

下面详细讲解下各种文档注释的使用。

6.5.2.1 类和接口

在每个类/接口的前头，可以进行以下方面的一些注释：

```
/**
 *[类或接口的功能描述].
 *
 * @author      [作者]
 */
```

6.5.2.2 方法

在每个方法的前头，要对方法进行以下方面的一些注释：

```
/**
 *[函数或过程实现功能描述]
 *
 * @param [参数名称]---参数注释说明
```

```
* @param [参数名称]---参数注释说明
* @return [函数或过程返回值说明，以及值的具体意义说明]
* @throws [抛出的异常类型] 异常说明
*/
```

说明：

1.“@param”项需要根据实际函数的参数个数进行添加，多个参数时，需要按参数的顺序进行添加，若函数没有参数，则该项不需要添加；

2.“@return”项为函数的返回值说明，若函数没有返回值，该项不需要添加；

3.“@throws”项为异常说明。

举例：

```
/**
 * 函数说明
 *
 * @param param1 参数一说明
 * @param param2 参数二说明
 * @return 返回值说明
 * @throws NullPointerException 异常说明
 */
public String test(String param1, int param2) throws NullPointerException{
 …… 
}
```

6.5.2.3 字段

在每个字段的前头，可以进行字段注释：

```
/** [字段描述]*/
public static int classVar1;
```

举例：

```
/**测试字段 1 说明*/
public static int test1;
```

6.6 其他约定规范

6.6.1 一般约定

1） 每个 Java 源文件不超过 2000 行，太长的程序难以阅读，应该尽量避免。

2） 如果一个包含二元运算符的表达式出现在三元运算符"？："的"?"之前，那么应该给表达式添上一对圆括号。

如：(x >= 0) ? x : -x;

3） 当不需要使用原变量值时，尽量使用前缀++或--而不是使用后缀++或--，例如：

for (int i = 0; i < 10; ++i)

4） 不同作用域变量的变量名不许相同。

5） 注释不允许嵌套。

6.6.2 日志记录规范

1）绝不使用 System.out.println()来记录日志；而是使用 Logger 或 Log4J 来记录

2）代码中生成的日志一定要可配置的，用于生产环境关闭日志信息

3）为不同日志消息选择不同级别

4）在输出日志时先进行级别判断，如：

```
if (logger.isDebuggable())
{
    logger.debug(…);
}
```

6.6.3 可移植规范

1） 尽量不要使用已经被标为不赞成使用的类或方法。

6.7 代码范例

6.7.1 Java 源文件范例

下面的例子，展示了如何合理布局一个包含单一公共类的 Java 源程序。接口的布局与其相似。更多信息参见"类和接口声明"以及"文挡注释规范"。

```
/*
 * 模块:  测试
 * 用途: 类或接口说明测试
 * 作者: 谭尤栋
 * 日期: 2011-07-05 10:55:04
 * 版权: Copyright (c) 1999-2011 ChiTone,Corp.
 */
package com.job5156.test.blah;

import java.blah.blahdy.BlahBlah;
```

```
/**
 * 类注释测试
 *@author 谭尤栋
 */
public class Blah extends SomeClass {
    /* A class implementation comment can go here. */

    /** 测试字段 1 说明 */
    public static int test1;

    /** 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...
    }

    /**
     * 函数说明
     *
     * @param param1 参数一说明
     * @param param2 参数二说明
     * @return 返回值说明
     * @throws NullPointerException 异常说明
```

```
     */
     public String test(String param1, int param2) throws NullPointerException{
          … …
     }

    /**
     * ...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...
    }

}
```

7 相关文件

无

8 相关记录

无

9 附录

无