1.1 Java 编程规范

本编程规范建立在标准的 Java 编程规范的基础上，如和标准的 Java 编程规范有冲突，以本编程规范为准。

1.1.1 程序结构

包名
引入包/类名
类注释
类
常量//常量注释
构造器注释
构造器
构造器注释
构造器
方法注释
方法
方法注释
方法

1.1.2 命名规范

命名规范使得程序更易理解，可读性更强。并且能够提供函数和标识符的信息。

- **文件命名规范**

java 程序使用如下的文件名后缀：

文件类型	后缀
Java 源文件	.java
Java 字节码文件	.class

对系统的文件命名方式有待于根据实际业务决定。

- **包命名规范**

包名应该唯一，它的前缀可以为任何小写的ASCII字符但必须是顶级域名，目前包括com, edu, gov, mil, net, org,或者ISO标准3166,1981中两个字符的国别代码。包名接下来的部分按照公司内部的命名规范，这些规范指出了某种目录名，主要包括部门，项目，机器，或者登录名。

命名规则为：

com.sinosoft.系统名[.模块名].xxx.xxx　具体参**错误！未找到引用源。**

包命名举例：

com.sinosoft.platform.bl.facade
com.sinosoft.platform.dto.domain

■ **类命名规范**

类名应该是名词，并且是大小写混合的。首字母要大写。尽量保证类名简单并且描述性强。避免使用只取单词首字母的简写或者单词的缩写形式，除非缩写形式比单词的完整形式更常用（例如：URL 或者 HTML）。

文件名必须和 public 的类名保持一致，注意大小写（JBuilder 等一些编译器可以忽略大小写，要特别注意）.

类命名举例：

class PolicyOverviewDto;

class DBPrpDuser.java;

■ **接口命名规范**

接口命名方式与类命名方式相同。

接口命名举例：

interface PolicyOverview;

interface PolicyOverviewSessionHome;

■ **方法命名规范**

方法名应该为动词，并且是大小写混合的。首字母要小写，方法名的第二个单词的第一个字母大写。

方法命名举例：

String getNoticeNo();

Collection findByCondition(String)

■ **变量命名规范**

变量，以及所有的类实例应为首字母小写的大小写混合形式。变量名的第二个单词的首字母大写。变量名的首字母不能为下划线或者$符。

变量名应该尽可能的短小，但要有意义。变量名应该便于记忆，也就是说变量名应该尽可能的做到见名知意。除了暂时使用的变量外（一般用于循环变量），应该避免使用只有一个字母的变量名。对于临时变量一般说来：i，j，k，m，n 代表整型变量。c，d，e 代表字符型变量。

变量命名举例：

String dataType;

String name;

int i;

char c;

■ **常量命名规范**

声明为类常量的变量或者 ANSI 常量应该全部为大写字母，并且每个单词间用下划线“_”隔开。为了便于调试，应避免使用 ANSI 常量。

常量命名举例：

static final int MIN_WIDTH = 4;

1.1.3 注释规范

Java 提供了两种类型的注释：程序注释和文档注释。程序注释是由分隔符/*…*/，和// 隔开的部分，这些注释和 C++ 中的注释一样。文档注释（即“doc 注释”）是 Java 独有的。由分隔符/**…*/隔开。使用 javadoc 工具能够将文档注释抽取出来形成 HTML 文件。程序注释主要是对程序的某部分具体实现方式的注释。文档注释是对程序的描述性注释，主要是提供给不需要了解程序具体实现的开发者使用。注释应该是代码的概括性描述，提供不易直接从代码中得到的信息，并且只包含对阅读和理解程序有用的信息。例如注释中包含相应的包如何编译或者在哪个目录下，而不应该包括这个包驻留在哪儿的信息。注释中可以描述一些精妙的算法和一些不易理解的设计思想，但应该避免那些在程序代码中很清楚的表达出来的信息。尽可能的避免过时的信息。错误的注释比没有注释更有害。经常性的注释有时也反映出代码质量的低下。

■ **程序注释**

程序注释有四种格式：块注释格式，单行注释，跟随注释，行尾注释

➢ **块注释格式**

块注释主要用于描述：文件、方法、数据结构和算法。一般在文件或者方法定义的之前使用。也可以用在方法定义里面，如果块注释放在函数或者方法定义里，它必须与它所描述的代码具有相同的缩进形式。

块注释应该用一个空行开头，以便于代码部分区分开来。

块注释举例：

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

➢ **单行注释**

比较短的注释可以放在一行中，但必须与它所跟随的代码有相同的缩进。如果注释不可以放在一行，那么必须按照块注释的格式来写。单行的注释会被解释为一空行。

单行注释举例：

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

➢ **跟随注释**

非常短的注释可以和它所描述的代码放在同一行。但要保证代码和注释之间有足够的间隔。在同一块代码中不止一个这样的注释时它们应该对齐。

跟随注释举例：

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

➢ **行尾注释**

注释标记“//”能够注释一行或者该行由“//”开始直到行尾的部分。行尾注释不

能用在多行文本注释中。但它可以将多行代码注释掉。这三种注释方法举例如下。

```
if (foo > 1) {
    // Do a double-flip.
    ...
}
else{
    return false;          // Explain why here.
}
//if (bar > 1) {
//
//      // Do a triple-flip.
//      ...
//}
//else{
//      return false;
//}
```

■ **文档注释**

文档注释描述了 Java 类，接口，构造函数，方法和属性。每个文档注释放在文档注释符/**…*/中。

每个类、接口或者成员只在声明的地方之前有一个文档注释。

例如：

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

注意：最外层的类或者接口的文档注释不用缩进。但它的成员的文档注释与成员的声明具有相同的缩进格式。

文档注释从第二行开始与第一行相比缩进一个字符。即注释的每一行的星号要对齐。

如果关于类、接口、变量或者方法的注释信息不是为文档信息提供的，就应该使用块注释或者单行注释（这两类注释应该使用在声明之后）。关于类实现方法的详细信息应该放在类语句后的块注释中，而不应该放在类的文档注释中。

文档注释方法或者构造函数的定义块内。Java 会将文档注释后的第一个声明于文档注释关联起来。

方法类型为 protected、public 的必须提供方法注释

1.1.4 构造方法规范

工具类不允许有 public 或 default 构造方法。这可以避免不必要的创建实例。

一个类里面的所有 public 方法都是 static 的，则这个类就可以作为工具类。其它类使用工具类的方法时，无需创建其实例，直接使用其方法即可。

例：在 sysframework 包中存在着的一个典型的工具类的例子：

com.sinosoft.sysframework.common.util.StringUtils，它的所有 public 方法都是 static 的，里面只有一个 private 构造方法。

1.1.5 修饰符规范

按照 Java 语言规范，修饰符按如下顺序组织：

public
protected
private
abstract
static
final
transient
volatile
synchronized
native
strictfp

例：

```
public static final int NOT_FOUND = 100; //正确的顺序
final static private String policyNo; //不正确的顺序。
```

1.1.6 声明规范

■ 变量声明

➢ **每行定义变量数目**

每行定义的变量数目必须有且只有一个。

例如：

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

➢ **变量初始化**

在声明局部变量的时候就要初始化变量。

➢ **变量定义位置**

在 for 循环里的循环变量可以在 for 语句里面定义。

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

注意：应避免局部变量屏蔽了外层变量的作用范围。也就是说不要在内部块中声明一个与外部块某个变量同名的变量。

例如类似下面的情况应避免：

```
int count;
...
myMethod() {
    if (condition) {
        int count;      // 与外部的 count 变量重名，应该避免!
        ...
```

```
    }
    ...
}
```

- **数组的定义**

  数组的[]应该放在类型名的后面，而不是变量名的后面。

  例：

```
String[] dangerNo;   //正确的顺序
String[][][] bigArray;   //正确的顺序
String dangerAddress[] ;//错误的顺序
```


- **类和接口声明**

  类和接口的声明应该遵循以下规范：

  1. 在方法名和参数列表的圆括号以及括号后的第一个参数间都没有空格。
  2. 开括号“{”必须与声明语句放在同一行。
  3. 闭括号“}”必须与声明语句有相同的缩进格式。
  4. 如果类或者接口实现内容为空，则可以将“}”放在“{”后面。
  5. 方法之间要用一个空行隔开。

```
class Sample extends Object {
    int ivar1;
    int ivar2;
    Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int emptyMethod() {}
    ...
    }
}
```


1.1.7 语句规范

不允许出现空语句的情况，典型的错误案例是：

- 一行里只有一个分号
- 一个语句后面连续出现几个分号

不允许出现无意义语句，如 policyNo = policyNo;

- **简单语句**

每行最多包含一个语句。

例如：

```
argv++;         // 正确
argc++;         // 正确
argv++; argc--;      // 错误，应该避免这样写
```

- **组合语句**

组合语句使用大括号括起来的一串语句。

1. 大括号中的语句比组合语句多一级缩进。
2. 开括号“{”应该放在组合语句前的语句末尾。闭括号“}”应该放在新的一行并与组合语句开始前的第一个语句有相同的缩进。
3. 如果语句是控制语句的一部分时，所有的语句都要用大括号围起来，即使只有一个语句也要用括号，例如在 if-else 或者 for 语句中。这样避免在添加语句时忘记添加括号而导致程序产生 bug。

- **return 语句**

return 语句在有返回值时不需要使用圆括号，除非使用圆括号在某些特定的情况下能够提高代码的可读性。

return 语句举例：

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

- **if 语句**

if 语句使用格式如下：

```
if (condition) {
    statements;
}

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

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

}

注意：在 if 语句中必须使用大括号，避免使用下面的形式。

```
if ( condition)   //禁止使用！这种方式忽略了大括号{}，容易导致错误。
statement;
```

- **switch 语句**

switch 语句使用格式如下：

```
switch ( condition) {
    case ABC:
        statements;
        /*  不包括 break 语句  */
    case DEF:
        statements;
        break;
    case XYZ:
        statements;
        break;
    default:
        statements;
        break;
}
```

1. 每一个 case 语句都要包含 break 语句，如果不包含应该说明正常情况下应在什么时候退出 switch 语句。
2. 每一个 switch 语句都要包含 default 情况，default 语句后的 break 语句不是多余的，它能避免在以后添加其他的 case 语句时而可能导致的错误。
3. 一般 switch 语句中 condition 的值是整型数。

- **for 语句**

for 语句使用格式如下：

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

- **try-catch 语句**

try-catch 语句使用格式如下：

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

try-catch 语句后也可以跟随 finally。无论 try 块内的语句是否成功执行，finally 块

内的语句都会执行。
使用格式如下：

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

如果 catch 块中除 throw 异常外没有任何处理，则不应该写该 try/catch 块。

1.1.8 空白使用规范

可使用编辑工具（如 Eclipse）的代码格式化功能实现，注意设置。使用无代码格式化功能的编辑器的，必须严格按照此规范编写。

- **空行使用规范**

空行的正确使用能够提高源代码的可读性，使得程序的逻辑关系更清楚。

- **两行空行的使用**

在下列情况下使用两行空行：

1. 在一个文件的两部分（section）代码之间；
2. 在两个类或者接口的定义之间。

- **单行空行的使用**

在下列情况下使用单行空行：

1. 在两个方法定义之间；
2. 在一个方法中局部变量的变量定义语句和方法的第一个其他语句之间；
3. 在一个语句块或者单行注释之前；
4. 为了提高程序可读性，在一个方法的两个逻辑段之间。

- **空格使用规范**

在下列情况下使用空格：

1. 关键字后跟随圆括号时要在两者间添加空格；
   例如：

```
while (true) {
    ...
}
```

2. 函数调用时，函数名和圆括号间要添加括号；
   注意：在定义方法时的方法名和圆括号之间不加空格，这样便于区分方法调用和方法定义。
3. 参数列表中的逗号后面要添加空格；

4. 除了成员运算符（.）以外，所有的二元运算符应该用空格和操作数分开；在一元运算符（如：++、--、负数-）和操作数之间不用空格；

例如：

```
a = c + d;
a = (a + b) / (c * d);
while (n<10) {
      n++;
}
print("size is " + foo + "\n");
```

5. for 语句中的表达式要用空格隔开；

例如：

```
for (expr1; expr2; expr3)
```

6. 强制类型转换后面要用空格隔开；

例如：

```
myMethod ((byte) aNum, (Object) x);
myMethod ((int) (cp + 5), ((int) (i + 3)) + 1);
```

1.1.9 缩进规范

可使用编辑工具（如 Eclipse）的代码格式化功能实现，注意设置。使用无代码格式化功能的编辑器的，必须严格按照此规范编写。

每一级缩进都要再上一级的基础上缩进 4 个字符。

不允许使用 TAB 键来进行缩进。

1.1.10 代码长度

检查点	最大字符数或行数
可执行语句的最大行数	120
Java 源文件最大行数	2000
方法、构造方法最多字符数	120
匿名内部类的最大行数	120
参数的最大个数	7

1.1.11 其它说明

不允许使用 TAB 键