# 软件编码规范说明书

# 优惠券管理系统

**团队：花开富贵!**

# **编码规范说明书**

# 1 介绍(Introduction)

## 规范存在的意义

1. 尽可能的减少一个软件的维护成本 , 并且几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护。
2. 可以改善软件的可读性，可以让开发人员尽快而彻底地理解新的代码。
3. 可以最大限度的提高团队开发的合作效率。

# 2命名规范

1. 尽量使用完整的英文描述符
2. 采用适用于相关领域的术语
3. 采用大小写混合使名字可读
4. 尽量少用缩写，但如果用了，必须符合整个过程中的统一定义
5. 避免使用长的名字（小于 15 个字母为正常选择）
6. 避免使用类似的名字，或者仅仅是大小写不同的名字
7. 避免使用下划线（除静态常量等）
8. 所有Pojo类与数据库表名对应
9. 所有Dao接口类继承自BaseDao，以I打头，以pojo类名称+Dao结束
10. 所有Daoimpl实现类以pojo类名称+DaoImpl结束
11. 所有业务接口类以I打头，以业务名称+Service结束

## 2.1 文件后缀(File Suffixes)

Java程序使用下列文件后缀：

|  |  |
| --- | --- |
| 文件类别 | 文件后缀 |
| Java源文件 | .java |
| Java字节码文件 | .class |

## 2.2 常用文件名(Common File Names)

常用的文件名包括：

|  |  |
| --- | --- |
| 文件名 | 用途 |
| GNUmakefile | makefiles的首选文件名。我们采用gnumake来创建（build）软件。 |
| README | 概述特定目录下所含内容的文件的首选文件名 |

# 3 标识符类型说明

## 3.1 包（ Package ）的命名：

采用完整的英文描述符，都是由一个小写单词组成

## 3.2类（ Class ）的命名:

类名应该是个一名词，采用大小写混合的方式，每个单词的首字母大写。尽量保证类名简洁而富于描述。使用完整单词，避免缩写词 ( 除非工程内有统一缩写规范或该缩写词被更广泛使用。

## 3.3接口（ Interface ）的命名:

基本与 Class 的命名规范类似。

在满足 Classd 命名规则的基础之上，保证开头第一个字母为”I”，便于与普通的 Class区别开

枚举（ Enum ）的命名

异常（ Exception ）的命名

在service中，操作数据库更新、写入、删除，方法名以update、save、edit、add、move、delet、remove打头，保证spring事务托管的有效；对数据的查询采用 list、get、find打头。

个Java源文件都包含一个单一的公共类或接口。若私有类和接口与一个公共类相关联，可以将它们和公共类放入同一个源文件。公共类必须是这个文件中的第一个类或接口。

|  |  |  |
| --- | --- | --- |
|  | 类/接口声明的各部分 | 注解 |
| 1 | 类/接口文档注释(/\*\*……\*/) | 该注释中所需包含的信息，参见"[文档注释](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html#52)" |
| 2 | 类或接口的声明 |  |
| 3 | 类/接口实现的注释(/\*……\*/)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，access modifier)，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。 |

# 4 缩进排版(Indentation)

4个空格常被作为缩进排版的一个单位。缩进的确切解释并未详细指定(空格 vs. 制表符)。一个制表符等于8个空格(而非4个)。

## 4.1 列宽

代码列宽控制在110字符左右。

## 4.2缩进

缩进应该是每行一个Tab(4个空格)，不要在代码中使用Tab字符.

## 4.3空行

空行是为了将逻辑上相关联的代码分块，以便提高代码可阅读性。

在代码中，不能包含多个空行。在以下情况下使用一个空行

1)方法与方法、属性与属性之间。

2)方法中变量声明与语句之间。

3)方法与方法之间。

4)方法中不同的逻辑块之间。

5)方法中的返回语句与其他的语创之间。

6)属性与方法、 属性与字段、方法与字段之间。

7)注释与它注释的语句间不空行，但与其他的语句间空一行。

## 4.4空格

在以下情况中要使用到空格

1)关键字和左括符“(”应该用空格隔开。如while (true)注意:在方法名和左括符“(”之间不要使用空格，这样有助于辨认代码中的方法调用与关键字。

2)多个参数用逗号隔开，每个逗号后都应加一个空格。

3)除了.之外，所有的元操作符都应用空格与它们的操作数隔开。一元操作符、 ++及-与操作数问不需要空格。

4)语句中的表达式之间用空格隔开。

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处理之。

注意：用于文档中的例子应该使用更短的行长，长度一般不超过70个字符。

## 4.5 换行(Wrapping Lines)

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开之：

- 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。

以下是断开方法调用的一些例子：

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();

}

这里有三种可行的方法用于处理三元运算表达式：

alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta

: gamma;

alpha = (aLongBooleanExpression)

? beta

: gamma;

# 5 注释(Comments)

**注释概述**

1. 在每个例程的开始，提供标准的注释样本以指示例程的用途、假设和限制很有帮助。注释样本应该是解释它为什么存在和可以做什么的简短介绍。
2. 避免在代码行的术尾添加注释:行尾注释使代码更难阅读。不过在批注变量声明时，行尾注释是合适的。
3. 避免在块注释的周围加上印刷框。这样看起米可能很漂亮，但是难于维护。
4. 在部署发布之前，移除所有临时或无关的注释，以避免在口后的维护工作中产生混乱。
5. 在编写注释时使用完整的句子。注释应该阐明代码，而不应该增加多义性。
6. 在编写代码时就注释，因为以后很可能没有时间这样做。
7. 为了防止问题反复出现，对错误修复和解决方法代码总是使用注释，尤其是在团队环境中。
8. 在所有的代码修改处加上修改标识的注释。
9. 用空白将注释同注释分隔符分开。在没有颜色提示的情况下查看注释时，这样做会使注释很明显且容易被找到。

### 5.1 块注释(Block Comments)

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。

块注释可以以/\*-开头，这样indent(1)就可以将之识别为一个代码块的开始，而不会重排它。

注意：如果你不使用indent(1)，就不必在代码中使用/\*-，或为他人可能对你的代码运行indent(1)作让步。

### 5.2 单行注释(Single-Line Comments)

短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](http://doc.javanb.com/code-conventions-for-the-java-programming-language-zh/index.html" \l "511)")。单行注释之前应该有一个空行。

### 5.3 尾端注释(Trailing Comments)

极短的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。若有多个短注释出现于大段代码中，它们应该具有相同的缩进。

### 5.4 行末注释(End-Of-Line Comments)

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。

### 5.5开头注释(Beginning Comments)

所有的源文件都应该在开头有一个C语言风格的注释，其中列出类名、版本信息、日期和版权声明：

/\*

\* Classname

\*

\* Version information

\*

\* Date

\*

\* Copyright notice

\*/

# 6 声明(Declarations)

## 6.1 每行声明变量的数量(Number Per Line)

推荐一行一个声明，因为这样以利于写注释。亦即，

int level; // indentation level

int size; // size of table

要优于，

int level, size;

不要将不同类型变量的声明放在同一行，例如：

int foo, fooarray[]; //WRONG!

注意：上面的例子中，在类型和标识符之间放了一个空格，另一种被允许的替代方式是使用制表符：

int level; // indentation level

int size; // size of table

Object currentEntry; // currently selected table entry

## 6.2 初始化(Initialization)

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。

## 6.3 布局(Placement)

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。

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++) { ... }

避免声明的局部变量覆盖上一级声明的变量。例如，不要在内部代码块中声明相同的变量名：

int count;

...

myMethod() {

if (condition) {

int count = 0; // AVOID!

...

}

...

}

## 6.4 类和接口的声明(Class and Interface Declarations)

当编写类和接口是，应该遵守以下格式规则：

- 在方法名与其参数列表之前的左括号"("间不要有空格  
- 左大括号"{"位于声明语句同行的末尾  
- 右大括号"}"另起一行，与相应的声明语句对齐，除非是一个空语句，"}"应紧跟在"{"之后

class Sample extends Object {

int ivar1;

int ivar2;

Sample(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int emptyMethod() {}

...

}

- 方法与方法之间以空行分隔