MOVIETIME 电影售票系统

代码风格说明

系统分析与设计 MovieLight 小组

Java代码风格说明

1.前言

编程语言：Java。

注意：如果以下风格规则出现矛盾或者冲突，则根据规则的适用范围进行判定，**适用范围越小，优先级越高**。如3.3和4.4，关于package语句规则的冲突，这里须采用3.3规则。

1.1术语说明

在本文档中，除非另有说明：

- 术语class可表示一个普通类，枚举类，接口或是annotation类型(@interface)
- 术语comment只用来指代实现的注释(implementation comments)，我们不使用“documentation comments”一词，而是用Javadoc。
- 其他的术语说明会偶尔在后面的文档出现。

2.源文件基础

2.1 文件名

源文件以其最顶层的类名来命名，大小写敏感，文件扩展名为.java。

2.2 文件编码：

源文件编码格式为UTF-8。

2.3 特殊字符

2.3.1 空白字符

除了行结束符序列，ASCII水平空格字符(`0x20`，即空格)是源文件中唯一允许出现的空白字符，这意味着：所有其它字符串中的空白字符都要进行转义。

制表符不用于缩进。

2.3.2 特殊转义序列

对于具有特殊转义序列的任何字符(`\b`, `\t`, `\n`, `\f`, `\r`, `\“`, `\‘`及`\\`)，我们使用它的转义序列，而不是相应的八进制(比如`\012`)或Unicode(比如`\u000a`)转义。

2.3.3 非ASCII字符

对于剩余的非ASCII字符，是使用实际的Unicode字符(比如∞)，还是使用等价的Unicode转义符(比如`\u221e`)，取决于哪个能让代码更易于阅读和理解。

3.源文件结构

一个源文件包含(按顺序地)：

1. 许可证或版权信息(如有需要)
2. package语句
3. import语句
4. 一个顶级类(只有一个)

以上每个部分用一个空行隔开

3.1 许可证或版权信息

如果一个文件包含许可证或版权信息，那么它应当被放在文件最前面。

3.2 package语句

每个package语句独立成行。

3.3 import语句

3.3.1 import不要使用通配符

即不要出现类似这样的import语句：`import java.util.*;`

3.3.2 不要换行

每个import语句独立成行。

3.3.3 顺序和间距

对这一点不做任何要求。

3.4 类声明

3.4.1 只有一个顶级类声明

每个顶级类都在一个与它同名的源文件中(当然，还包含`.java`后缀)。

例外：`package-info.java`，该文件中可没有`package-info`类。

3.4.2 类成员顺序

类的成员顺序不存在唯一的通用法则。

需要注意的是：**开发者应当以某种逻辑去排序每个类的成员；维护者应当遵循这种逻辑而不得随意排序（如习惯性地添加到类的开头或结尾），或者以恰当地理由遵循另外一种逻辑**

3.4.2.1 重载：永不分离

当一个类有多个构造函数，或是多个同名方法，这些函数/方法应该按顺序出现在一起，中间不要放进其它函数/方法。

4.格式

4.1大括号

4.1.1多块语句

大括号与`if`, `else`, `for`, `do`, `while`等语句一起使用，即便是空语句或一条语句。

4.1.2非空块

- 左大括号前不换行
- 左大括号前空格
- 左大括号后换行
- 右大括号前换行
- 如果右大括号是一个语句、函数体或类的终止，则右大括号后换行; 否则不换行。例如，如果右大括号后面是`else`或逗号，则不换行。

`public void sample() { if (true) { //do something } else { //do somthing } };`

4.1.3空块

空块结构里什么都不含， 大括号可以写成`{}`，但如果是多块语句的一部分，即使是空块也要换行

```
void doSomething() {}
```

多块语句

```
try {
    //do something
} catch(Exception e) {
}
```

4.2块缩进

2个空格

4.3每个语句后要换行

4.4列限制：80或100

4.5自动换行

为了满足4.4，而被分为多行

4.5.1 断行的位置

1. 如果在非赋值运算符处断开，那么在该符号前断开(比如+，它将位于下一行)。 ``` a = b
    - c; ``` 这条规则也适用于以下“类运算符”符号：点分隔符(`.`)，类型界限中的`& (<T extends Foo & Bar>)`，catch块中的管道符号(`catch (FooException | BarException e)`)
2. 如果在赋值运算符处断开，通常的做法是在该符号后断开(比如=，它与前面的内容留在同一行)。 `a = b + c;`

3. 方法名或构造函数名与左括号留在同一行。

4. 逗号(,)与其前面的内容留在同一行。

4.5.2 缩进

自动换行时，第一行后的每一行至少比第一行多缩进4个空格。

5.命名约定

5.1标识符通用的规则

标识符只能采用ASCII字母和数字，即每个标识符都能用正则表达式`\w+`完全匹配。

5.2标识符类型的规则

5.2.1包名

包名全部小写，连续的单词只是简单地连接起来，不使用下划线。

5.2.2 类名

类名都以**UpperCamelCase**风格编写。 *类名通常是名词或名词短语，接口*名称有时可能是形容词或形容词短语。 *测试类*的命名以它要测试的类的名称开始，以Test结束。例如，`HashTest`或`HashIntegrationTest`。

5.2.3 方法名

方法名都以**lowerCamelCase**风格编写。方法名通常是动词或动词短语。 注意：下划线可能出现在JUnit测试方法名称中用以分隔名称的逻辑组件。一个典型的模式是：`test<MethodUnderTest>_<state>`，例如`testPop_emptyStack`。

5.2.4 常量名

常量名命名模式为CONSTANT*CASE，全部字母大写，用下划线分隔单词。* ***那，到底什么算是一个常量？*** *每个常量都是一个`static final`字段，但不是所有`static final`字段都是常量。在决定一个字段是否是一个常量时， 考虑它逻辑上是否真的像是一个常量，而不仅仅是对象不变。* ``` *// Constants static final int NUMBER = 5; enum SomeEnum { ENUM*CONSTANT }

// Not constants static final Set mutableCollection = new HashSet(); static final String[] nonEmptyArray = {"these", "can", "change"}; ```

5.2.5 非常量字段名

非常量字段名以**lowerCamelCase**风格编写。

5.2.6 参数名

参数名以**lowerCamelCase**风格编写。 *参数应该避免用单个字符命名。*

5.2.7 局部变量名

局部变量名以**lowerCamelCase**风格编写，比起其它类型的名称，局部变量名可以有更为宽松的缩写。 即使局部变量是final和不可改变的，也不应该把它示为常量。

5.2.8 类型变量名

类型变量可用以下两种风格之一进行命名： 单个的大写字母，后面可以跟一个数字(如：E, T, X, T2)。 以类命名方式(5.2.2节)，后面加个大写的T(如：RequestT, FooBarT)。

5.3 驼峰式命名法(CamelCase)

驼峰式命名法分大驼峰式命名法(UpperCamelCase)和小驼峰式命名法(lowerCamelCase)。 名字从散文形式开始: 1. 把短语转换为纯ASCII码，并且移除任何单引号。例如："`廖Lushen'sbug`"将变成"`LiaoLushensbug`"。 把这个结果分割数个单词（`Liao Lushens bug`） 2. 现在将所有字母都小写(包括缩写): `liao lushens bug` 3. 大驼峰式：每个单词的第一个字母都大写。`Liao Lushens Bug` 小驼峰式：除了第一个单词，每个单词的第一个字母都大写。`liao Lushens Bug` 4. 最后将所有的单词连接起来得到一个标识符。`LiaoLushensBug` 和 `liaoLushensBug`

推荐：如果某个单词已经有了常用的驼峰表示形式，在进行第1步时可以按组成拆分开(如"`AdWords`"将分割成"`ad words`")。

6.编程实践

6.1 @Override：能用则用

只要是合法的，就把@Override注解给用上。

6.2 不可忽视捕获的异常

6.3 使用类对静态成员进行调用

使用类名调用静态的类成员，而不是具体某个对象或表达式。

6.4 Finalizers: 禁用

6.5 谨慎使用recycle

7.Javadoc

7.1 格式

7.1.1 一般形式

Javadoc块的基本格式如下所示：
`/** * Multiple lines of Javadoc text are written here, * wrapped normally... */ public int method(String p1) { ... }` 或者是以下单行形式：
`/** An especially short bit of Javadoc. */`

7.1.2 段落

空行(即，只包含最左侧星号的行)会出现在段落之间和Javadoc标记(@XXX)之前(如果有的话)。 除了第一个段落，每个段落第一个单词前都有标签`<p>`，并且它和第一个单词间没有空格。

7.1.3 Javadoc标记

标准的Javadoc标记按以下顺序出现：`@param`, `@return`, `@throws`, `@deprecated`, 前面这4种标记如果出现，描述都不能为空。 当描述无法在一行中容纳，连续行需要至少再缩进4个空格。

7.2 摘要片段

每个类或成员的Javadoc以一个简短的摘要片段开始。这个片段是非常重要的，在某些情况下，它是唯一出现的文本，比如在类和方法索引中。 这只是一个小片段，可以是一个名词短语或动词短语，但不是一个完整的句子。它不会以`A {@code Foo} is a...`或`This method returns...`开头, 它也不会是一个完整的祈使句，如`Save the record...`。然而，由于开头大写及被加了标点，它看起来就像是个完整的句子。 Tip：一个常见的错误是把简单的Javadoc写成`/** @return the customer ID */`，这是不正确的。它应该写成`/** Returns the customer ID. */`。

7.3 哪里需要使用Javadoc

至少在每个public类及它的每个`public`和`protected`成员处使用Javadoc，以下是一些例外：

7.3.1 例外：不言自明的方法

对于简单明显的方法如`getFoo`，Javadoc是可选的(即，是可以不写的)。这种情况下除了写“`Returns the foo`”，确实也没有什么值得写了。 单元测试类中的测试方法可能是不言自明的最常见例子了，我们通常可以从这些方法的描述性命名中知道它是干什么的，因此不需要额外的文档说明。

7.3.2 例外：重载

如果一个方法重载了超类中的方法，那么Javadoc并非必需的。

7.3.3 可选的Javadoc

对于包外不可见的类和方法，如有需要，也是要使用Javadoc的。如果一个注释是用来定义一个类，方法，字段的整体目的或行为， 那么这个注释应该写成Javadoc，这样更统一更友好。