JavaMusician

编码规范

JavaMusician 编码规范修改表

版本	修改时间	修改的部分	修改概要说明	修改人
1.0	2010/11/9	全部	创建规范表	吴卓豪
1.0.1	2010/11/10	资源文件存放说明	建立存放说明	吴卓豪
1.0.2	2010/11/12	文档排版	进行文档最后排版	吴卓豪

JavaMusician 编码规范

一. 目的

本文是测试 JavaMusician 项目的规范文件。本文的目的是统一编码风格、提高代码质量。

二. 范围

本文的适用范围是项目的编码阶段，同时也适用于项目的设计阶段和测试阶段。

三. JavaMusician 编码规范概要

JavaMusician 代码风格概要说明如下。

1 代码风格

1.1 格式说明

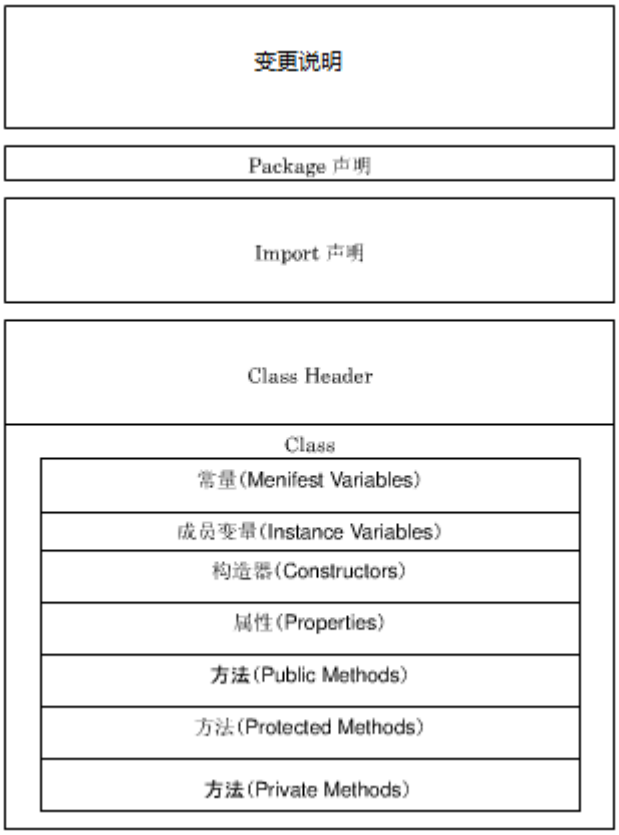
（1）除字符串和注释内以外，代码中的空格一律为半角；

（2）代码应该有缩进，缩进为 1 个 TAB 字符；

（3）任何可以省略大括号的地方都不应省略大括号。

1.2 基本结构

一个类/接口的基本结构应该是这样的：



图片 1 类/接口的基本结构

2 代码结构

2.1 变更说明

范例：

```
/**
 *Example.java
 *
 *功能：（用一句话描述类的功能）
 *
 *版本        变更日          修改人          变更内容
 *-------------------------------------------------------------------------
 *
 */
```

说明：

（1）变更说明的第一行是正文，即在它之前没有空行；

（2）当小组成员修改代码时，必须按照上述填写相关的内容，即使是使用 SVN 也要填写。

2.2 Package 声明

分包的命名规则为：

团队名称.项目名称.模块名称.子模块.子模块...

范例：

```
JavaMusician.MusicMan.Instruments.Piano
```

说明：

（1）Package 名中禁止使用除英语字母以外的任何字符。

2.3 Import 声明

范例：

```
Import java.util.Map;
Import java.util.HashMap;
```

说明：

（1）Import 声明内部没有空行；

（2）Import 需要指出 import 哪一个类，禁止 import 整个 package，例如：

```
Import java.util.*;   //错误
Import java.util.HashMap;    //正确
```

（3）通常程序内部使用的 package 都是用 import 语句声明在程序头部；原则上，程序内部不再出现完整类名，例如：

```
Java.io.File file=new java.io.File(“test.txt”);    //错误
File file=new File(“test.txt”);      //正确
```

（4）程序中要到的类应该在 Import 中声明，但是程序中没有用到的

类不要出现在 Import 中。

2.4 Class Header

代码的文件头应该遵循以下标准写法：

```
/*
*描述是一个什么样的类，格式：本类是 XXXXXXX
*描述这个类的功能
*-可选  介绍本类中涉及到的相关信息
*-可选  介绍本类在系统中角色，以及如何和其他类交互
*-可选  介绍本类的使用方法
*-可选  使用本类的注意事项
*
*@authour    作者名字
*@version     版本信息
*@since       创建信息
*/
```

说明：

（1）“Class Header”可以分为 2 个部分——说明部分和版本信息，中间使用空行分割；

（2）程序中的每一个类都必须有 Class 注释。根据需要，Class 注释中除了说明部分不能省略外，版本信息可以省略；

（3）版本信息要上下对齐，如上范例所示；

（4）@version 信息的机构是”Ver<版本><日期>”，其中”版本”是项目版本，”日期”为此文件最后一次修改的日期；

（5）@since 信息的机构是”<项目名称>Ver<版本>”，其中”项目名称”可以是项目的全称，也可以是项目的简称，但是要注意在整个项目中这个名称应该不变，”版本”是这类第一次被创建时的项目版本。

2.5 Class

范例：

```
Public class ExampleClass extends SuperClass throws Exception{

    //Class Body....Do sth

}///:~
```

说明：

（1）Class Body 应该缩进 4 格；

（2）Class 的最后一行规定以”}///:~”结尾，这样做的目的有：标明到了文件最后一行；防止不小心删除最后一个字符（大括号）；

（3）Class 声明部分可以分为 4 部分——Class 前缀、Class 名、扩展信息和例外信息；

（4）Class 的命名遵循以下规则：

1. Class 的名称应该是一个名词短语。可以是”形容词/名词+名词”；

2. Class 的名称由 1 个或 1 个以上的英语句单词组成，其中每一个英语句单词的首字母应该大写，其余字母小写；

3. 规定 Class 名称只能有 26 个英文字母组成；

4. 表示复数的 Class 可以命名为”单数+s”的形式。

2.6 常量

范例：

```
/**
```

```
*用一句话描述变量表示什么
*/
Public static final int TEMP=0;
```

说明：

（1）类中常量的声明位于类的最前面，和类的生命之间空一行；

（2）在声明常量的时候给它赋初值；

（3）常量修饰符如 static、final 可以省略，但是 public、private、protected 不能省略（默认属性 package 例外）；

（4）常量的名称使用 26 个英语句大写字母和下划线”_”组成。

2.7 成员变量

范例：

```
/**
*用一句话描述变量表示什么
*/
private int mTemp=0;
```

说明：

（1）在声明变量时给它赋初值；

（2）一般地，类成员变量声明为 private，如果需要访问，则提供相应的 setter 和 getter 方法，规定不能声明为 friendly；

（3）类成员变量的名称使用 26 个英文字母组成，并规定前缀为小写字母 m、单词的首字母大写。

2.8 构造函数

范例：

```
/**
*描述这个构造函数的作用
*/
Public Example(){
    //本体处理
}
```

说明：

（1）缺省的构造函数必须显示的声明；

（2）如果缺省的构造函数中没有任何代码，则应使用”//null”标注，

例如：

```
Public Example(){}      //错误
Public Example(){
//null
}
```

2.9 属性

范例：

```
/**
*获取名称
*
*@return String  名称
*/
Public String getName(){
    Return mName;
}

/**
*设置名称
*
*@param    name    名称
*/
Public void setName(String name){
```

```
        This.mName=name;
    }
```

说明：

（1）在类中做 setter 方法、getter 方法，方法的命名遵循 JavaBean 属性的命名规范；

（2）属性名使用 26 个英文字母组成；

（3）与属性对应的类成员变量的名字同属性名相同；

（4）属性的 getter 和 setter 方法应该为 public。

2.10 方法

范例：

```
/**
*描述方法的作用
*-可选 描述方法的使用条件
*-可选 描述方法的执行流程
*-可选 描述方法的使用方法
*-可选 描述方法的注意事项
*
*@param      String      名称
*@return     String      名称
*/
Public String parse(String name){
    //1.取得处理要求的字符串的有效性
    //2.处理本体
    //3.返回/保存结果
}
```

说明：

（1）方法部分中，每个方法之间有一空行；

（2）一个方法的注释中，包含这个方法的说明、参数说明、返回值说明、例外说明，规定，这几个部分在方法中存在的情况下，必须给

出相应的说明；

（3）方法本体缩进 4 个空格；

（4）关于方法内部的实现：

1. 一个方法只完成一个特定的功能，要求是可以用一句话描述这个方法的作用，如果用两句或两句以上的话才能描述这个方法的作用，说明这个方法过于复杂

方法内部的写法：

//如果不满足条件 1，那么返回或异常

//如果不满足条件 2，那么返回或异常

//如果不满足条件 3，那么返回或异常

//如果所有的不满足的条件都处理过，那么就执行

//返回

例如：

```
If   (name != “John”) {
      Return false;
}

If   (password != “123”) {
      Return false;
}

//执行方法的主体

Return true;
```

2.在方法内部的注释要写明方法的操作的简要说明；

3. For 循环采用 i、j 或 k 作为循环变量；

4. 正确使用空格，例如：

```
String mName = “default” + “name”; //操作符两边留一个空格
```

```
if (a == b)
for (int I = 0; I < 10; i++)//if、for 等后面留一个空格，分号后面留一个空格
```

5. 调整代码规范，主要是操作符处在同一列，例如：

```
String mName = “name”;
String mPassword = “123”;
```

6. 使用 try-catch-finally 结构来处理容易出问题的代码。

2.11 项目目录结构说明

（1）外部引用包统一放置在/lib 文件夹下；

（2）图像统一放置在/res/drawable 目录下；

（3）Xml 文件统一放置在/res/xml 目录下；

（4）音频和视频文件统一放置在/res/raw 目录下；

（5）输入文件统一放在/input 文件夹下；

（6）输出文件统一放在/output 文件夹下。

四. 代码规范检查工具

采用 Checkstyle 的 Eclipse 插件作为每个编码小组成员的本地代码规范检查方式，由于 Checkstyle 默认是按照 SUN 公司定制的规范进行检查，难免有部分代码不能通过，因此，过滤掉某些比较严格的规定来进行检查。

五. 代码评审

项目开发是采用敏捷开发方式进行方法的，经常开会是极佳的沟通方式，通过开会，小组成员之间的各种想法就可以得到很好的传达，

所以，在代码评审的过程中，各小组成员可以针对现在的代码进行发表意见，其中，较为主要的几个有：

（1）编码规范问题

每位编码成员在编写的过程中，难免为了自己的方便而忽略掉代码中变量和方法的编码规范。通过小组之间的代码编码规范检查就可以减少一部分的不规范。

（2）代码结构问题

分析代码的重用性，方法或者类的分层，个类的松耦合程度。

（3）实现问题

是否存在错误验证、异常处理、代码可读性不佳的情况。

（4）测试问题

1. 测试覆盖度够不够、可测试性好不好。

2. 代码评审的好处

提高代码质量，在项目的早期发现缺陷，将损失降至最低。评审的过程也是重新梳理思路的过程，双方都加深了对系统的理解，促进团队沟通、知识共享、共同提高。

3. 代码评审的实现

采用交叉评审，即任意两个组员，或开发组长分别与每个组员结对进行。代码作者讲解如何以及为何这样实现、评审者提出问题和建议，每次解决的问题要记录到 SVN 注释。每次评审不要贪多，要注重质量，例如，可以每编写 500 行代码就执行一次评审，这样能尽早的发现缺陷。

4. 会前准备工作

组织者应通知各参与者本次评审的范围，参与者阅读源代码，列出发现的问题、亮点，汇总给组织者，准备工作要细致，需要给出问题详细描述以及相关代码的位置；架构师提供开发规范，为代码评审提供依据，建立单元测试规范，否则无法达到测试覆盖度的要求，难以修改发现的问题。

5. 评审代码的选择

系统关键模块、业务较复杂的模块或容易出错的模块。

6. 会议议程

如果是第一次会议，先由该项目开发组长做整体介绍，参加者一次发言，结合代码讲解发现的问题，每讲完一个问题，针对其展开讨论，如果问题不多，可以针对项目做评审，如性能测试。

7. 会后总结

把会上提出的所有问题、亮点及最终结论详细的记录下来，未能讨论清除的问题，会后解决，并做好相应的记录，如评审到什么程度了、还存在哪些方面的问题、已经评审好等各方面内容。