Java 编码标准（V2.0）

信息工程系　方宏

1.目的

此编码标准是在软件开发中，提供Java 编码时的规则、建议。标准制定的原则是写容易读、容易维护的代码。在实际的编码中，大家应遵守此规则，本编码标准的最终目的是希望大家能形成良好的代码风格习惯。

2. 文件构成

(1) 文件名

public 类名应与所在java文件名同名。

例：public class Point 要放到Point.java 中。

软件包内的非Public类应包含在主要使用该类的Public类的文件中(此时,要注意*.java 和*.class 可能不对应)。

(2) 文件的位置

根据Project的根目录，将软件包名的“.”放入到对应目录层次中。

例：myProject.framework 软件包配置在<ProjectRoot>/myProject/framework 目录中.

例：com.google.dto软件包配置在<ProjectRoot>/com/google/dto 目录中

(3) 测试类名

类ClassName 的综合测试类名为ClassNameTest .每个软件包测试为LastPackageNameTest .

例：Point2D 类的话，作成Point2DTest.java .

例：com.google.extremedomo软件包的话，写成ExtremeDomoTest.java .

理由：起一个有一贯性的名字. 测试代码为使用方法的样品.

其他方法：将ClassName 的综合测试类名作为ClassNameUt (Ut 为UnitTest之略)

(4) 测试类的位置

测试类配置在和被测试类相同目录，或者其副目录“test” 中。
理由：如果不是在物理上相近的位置的话,维护会被忘记. 关于和产品代码的分离，可以用其他的工具(makefile, Ant 的build.xml 等)调整.

注意：在集成开发环境如 eclipse、netbean 中会对上述要求做统一默认处理，一般应采纳。

3. 命名规则

(5) 软件包名

用“.” 隔开的文字.
com.google.domainname.projectname
junit.framework
建议在实际的项目中，包的命名由项目经理和开发组长来制定，在整个项目中使用统一的包名及类名。结合某公司的实际，制定一个参考如下(都位于<projectroot>/src目录下)：
com.google.domain//存放DTO，POJO，JAVABEAN
com.nemtarch.spring//存放Spring的相关文件
com.google.struts.actin//存放struts的ACTION类
com.google.struts.form//存放struts的FORM类
com.google.filter　　//存放过滤器，如session过滤器，字符过滤器等
com.google.listener　　//存放监听器，如访问流量监听器，
　　商业软件中的配置文件读取监听器
com.google.util　　//存放公用的类库，以后完善了我们公司的类库过后，就可以只用导入一个JAR包就可以啦
com.google.maps　　//存放iBATIS的XML配置文件
com.google.interface　//存放接口类
com.google.infaimpl　　//存放接口实现类(这个可以根据需要，也可以用
　　com.google.dao)
com.google.dao　　//实体操作类
com.google.abst　　//存放抽象类
com.google.hibernate//存放hibernate相关类
com.google.test　　//存放的测试类
com.google.service　　//业务接口外部调用类
com.google.common　　//基本配置类
com.google.excel　　//生成EXCEL的操作类
(根据项目的不同，下面可以再添加相关的包，这个可以由PM来决定)

(6) 文件名

Public类名要根据编译程序的规则，必须是和文件名相同(包含大写小写的区别).

(7) 类名

开头大写. 后面将隔开作为大写.

CapitalizedWithInternalWordsAlsoCapitalized

(8) 异常类名

将最后作为了Exception 的类名.

ClassNameEndsWithException

(9) 接口名

同类名. 但是，如果有和class 区别的必要的话，最开始加I .

InameOfInterface

(10) 实现类名

特别是有和interface 区别的必要的话，最后加Impl .

ClassNameEndsWithImpl

(11) 抽象类名

没有适合抽象类名的名字时，从Abstract 开始,起一个联想SubClass名的名字.

AbstractBeforeSubClassName

(12) 常数(static final)

将大写用“_” 连接的.

UPPER_CASE_WITH_UNDERSCORES

（*注：）static 和 final 的区别

final 注意要素：

1.abstract 和 final 不能同时修饰一个类。

2.修饰方法保证该方法不能被子类重载，已经被 private 修饰的方法以及所有包含在 final 类中的方法，都被缺省认为是 fianl 的。

3.如果是修饰 primitive 数据，那么这份数据是不可更改的，如果是修饰对象的话，那么其指向不可更改。但例如一个对象内部有另外一个对象，这另外一个对象可以变更。

两者重点不同：

static 强调只有一份

final 强调是常量

(13) 方法名

最开始是小写，后面隔开用大写.

firstWordLowerCaseButInternalWordsCapitalized()

(14) Factory方法(new Object的)

X newX()
X createX()

(15) Converter方法(将Object变换成其他的Object的)

X toX()

(16) 属性的取得方法
当然，如果用MYECLIPSE开发的话，属性的取得方法和设定方法都是可以自动生成的

X x()
在X getX() // JavaBeans 可以作为Proverty来用(推荐)
在boolean isEnabled() // JavaBeans 可以作为Proverty(标准)来用(推荐)

(17) 属性的设定方法
在void setX(X value) // JavaBeans 可以作为Proverty（标准）来用(推荐)
(18) 返回boolean 变量的方法

is + 形容词，can + 动詞，has + 过去分词，三单元动词，三单元动词+ 名词.

在boolean isEmpty() // JavaBeans 可以作为Proverty（标准）来用(推荐)
boolean empty() // 不行！因为能取’空’的动词的意思,所以不好.
boolean canGet()
boolean hasChanged()
boolean contains(Object)
boolean containsKey(Key)
理由: if, while 文等的条件会变为容易读. 还有true 是哪个意思容易懂.
(19) 英语和汉语

所有的标识符的名字都要以英语为基本,另外作成汉语对应用语辞典,在Project的整个生命周期维护.

(20) 名字的对称性

取类名，方法名时，注意以下英語的对称性.

add/remove
insert/delete
get/set
start/stop
begin/end
send/receive
first/last
get/release
put/get
up/down
show/hide
source/target
open/close
source/destination
increment/decrement
lock/unlock
old/new
next/previous

(21)局部循环变量

范围较小的循环变量，将i, j, k 的名字按顺序使用.

(22)范围窄的名字

范围窄的变量名使用省略了型名的为好.
例： ServeletContext sc = getServletContext();

(23)无意思的名字

叫做Info, Data, Temp, Str, Buf 的名字要再考虑.

坏例子: double temp = Math.sqrt(b*b - 4*a*c);
好例子: double determinant = Math.sqrt(b*b - 4*a*c);

(24)大写小写

大写和小写被作为其他的文字来用，但不能取仅用此区别的名字.

(25)其他

其他，因Project不同,有时会用以下的命名规则.

Local变量:

lower_case_with_underscore

private/protected 变量:

prefixUnderscore 或suffixUnderscore

static private/protected 变量:

__twoPrefixUnderscores 或twoSuffixUnderscores__

4. 编码指南

(26)程序文件形式

代码文件形式是以Sun Microsystems, Inc 的JDK 源程序为准. 编排基本上是和K&R 的C 语言形式相同。

```
/* COPYRIGHT ...
        * ...                                  文件开头有时有copywrite. 这里不是/** , 而是/* 的注释.
        */
        package myProject.util;
                                               然后, package, 1 行空,import 的罗列
        import java.util.Stack;
                                               在import java.util.Vector;类定义之前从关于类
                                               的/**开始的注释. 第1行短,明白说明类. 用半角句号结
                                               束. 从下一行开始详细说明. 继续行按照第 2 个*,将* 放在
                                               开头.
        /**************************************************
         * FileName:DaoConfig.java
         * Version: 1.00
         * Author: Bruce
         * Date: 2006-07-27
         * Last modify Date: 2006-07-26
         * Function: 读取配置文件以获取相关的信息
         *
         * CopyRight (c) Google Company 2006
         *          All rights reserved
         **************************************************/
      类定义开始的”{“ 不改行.
        public class Stack {定义开始的”{“ 不改行.

          /**                                  方法的注释也和类同样.
            *追加要素.                         @param, @return, @exception(如有)为

            * @param item 追加的要素           必须. 根据需要@see 等.
```

```
    */
    public void push(Object item) {                    方法定义开始的”{“ 也不改行.
      if (itemCapacity <= itemCount) {
          //...                                        if, while 等的关键词和“(“之间空一格.
字朝下    } else                                       (方法名后面的”(“无空格). ”(“的后
面不空,
1 TAB=4 SPACE.  {                                      运算符的两侧空格. ”(“的后面空格, 继
续
        // ...                                         “{“.
      }        注意用if/else 的“{“, “}”的位置.
    }
    /**
      *取得开头要素.开头要素被除去.
      * @return 开头要素
      */
    public Object pop() {
        // ...
        return top;        不用()包围return 值.
    }
  }
```

其他方法: 关于缩排，{} 的位置，为了不破坏各自的创造性，不敢规定.

当然在 MyEclipse 中开发的时候，先用“CTRL＋A”全部选中过后,可以使用快捷键”CTRL+SHIFT+O”来进行自动排版

(27)长的行

一行最大80 个字，超过时要分行. 分割的原则是：(1) 利用Local变量，(2)用回车键换行，(3)在优先低的运算符的前面改行.

例:

```
double length = Math.sqrt(Math.pow(Math.random(), 2.0) +
Math.pow(Math.random(), 2.0));

// 方针(1)
double xSquared = Math.pow(Math.random(), 2.0);
double ySquared = Math.pow(Math.random(), 2.0);
double length = Math.sqrt(xSquared + ySquared);

// 方针(2)
double length = Math.sqrt(Math.pow(Math.random(), 2.0,
                Math.pow(Math.random(), 2.0);

// 方针(3)
return this == obj
            || (this.obj instanceof MyClass
```

```
                    && this.field == obj.field);
```

(28)长的声明行

类，方法的声明很长时，(1) 用extends/implements/throws 节改行，(2) 用回车键改行.

例:

```
public class LongNameClassImplemenation
            extends AbstractImplementation,
            implements Serializable, Cloneable {
  private void longNameInternalIOMethod(int a, int b)
               throws IOException {
          // …
  }
  public void longMethodSignature(int a, int b, int c,
                                  int d, int e, int f) {
          // …
  }
  // …
}
```

(29)import

在import 尽量不使用* . 从同一个软件包import 3 个以上的类时，使用* .

理由: 将依存性明确化. 如有使用了多个*的import ，读起来辛苦. 但是，重复使用某个软件包时等(会用).

(30)abstract class vs. interface

抽象类(abstract class)尽量不使用，而多用interface 吧. abstract class 仅在有一部分安装，一部分是抽象方法时使用.

理由:interface 有多少都可以继承，但class 只有1 个. 从1 个开始继承的话，就再也不能继承了,没办法. Abstract class means isA ,interface means like A ;

(31)初始化

变量应明确的予以初始化，但不重复初始化。

坏例子：

```
class PoorInitialization {
    private name = “initial_name”;
    public Sample() {
    name = “initial_name”;//ERROR
}
```

理由: 将关于初始化的BUG最小化.

(32)避免static 变量

static 变量(类变量)要极力避免. (static final 常数除外)

(33) final 使用

如果实例变量在被作成之后绝对不变化的话，积极使用final . 还有，如果不变更方法的自变量的参照地的话，当作final 吧.

理由: final ，synchronization 、编译的效率化等容易被适用. 从内部类参照自变量时，有必要是final .

(34) private vs. protected

在对象的生命周期要锁定在一定范围时, 尽量用private ，使别人不能破坏 封装性.

理由: private 确实能够Shut Out从类外来的使用，但客户不能凭subclass 化进行细的Chaining. 喜欢使用 private 吧. protected 以后，发生了变更时继承了它的全类会产生影响

(35) get/set 方法

要避免过度作成到实例变量的Access方法getX()/setX() 后当做public . 讨论其必要性，做成更有意思的方法.

理由: 实例变量多数依存于其他的实例变量. 不能破坏类内部的整合性.

(36) 变量隐藏

使用和超级类的变量名相同的变量名要避免.

理由: 一般来说这是BUG. 如有意图要注释.

(37) 排列声明

排列的声明为Type[] arrayName .

理由: Type arrayName[] 不过是作为从C 那里剩留下的.

例:

```
static void main(String[] args); --- ○
static void main(String args[]); --- ×
```

当然，上面这种写法也不会错，但是建议还是养成好的编码规范吧！

(38) public 方法

类的public 方法以「自动销售机的接口」为目标. 设计成容易懂，即使搞错了使用方法, 内部的整合性也不会搞坏. 还有，如果可能,进行按合同的设计(Design by Contract)，用代码表现类的不变条件和方法的事前事后条件.

(43) this 之 return

即使考虑了使用上的方便，return this 也要尽量避免.

理由:叫做 a.meth1().meth2().meth3() 的连锁一般会成为synchronization上的问题之源

(44)方法的多重定义

因为自变量的类型，要尽量避免方法的OverLoad（自变量数不同为OK）. 特别是和继承一起使用的话较为麻烦.

例:

× : draw(Line), draw(Rectangle)
○ : drawLine(Line), drawRectangle(Rectangle)
○ : draw(Shape)

(45) equals()和hashCode()

若重载Object.equals()方法，同时hashCode()方法也重载，相反亦然。
理由: 因为对应Container类(Hashtable)等.

clone()

如果使用clone() 方法，需封装Cloneable并清楚标明。
例:

```
class Foo implements Cloneable {
        // ...
        public Object clone() {
                try {
                        Foo foo = (Foo)super.clone();
                        // Foo 类的属性的 (Clone)
                        // ...
                } catch (CloneNotSupportedException e) {
                        // 因为implements  Cloneable 所以不能发生
                        throw new InternalError();
                }
        }
}
```

理由: 在shallow copy里不好的Case很多.

(47) 缺省构造方法

如果可以的话，不管什么时候都准备缺省的构造方法(没有自变量的方法).
理由: 在Class.newInstance() 里从类名字符串可以动态创建该类.

(48) abstract method in abstract classes

在abstract 类，通过添加no-op方法，并明确声明为abstact 方法. 并且，如果可以准备可共有的缺省封装，将其做为protected，使SubClass可在1 行写处理.
理由: 因为java 编译程序在编译时可检出没被封装的abstract 方法,所以可以避免所谓忘记单个封装的Bug.

(49) Object的同值比较

Object的比较是使用equals()方法，不是使用== . 而String 的比较必须使用== .
理由: 如果封装者准备了equals()，应该是希望使用使用该方法才封装的.

equals()的缺省的封装，仅仅是==而已.
理由: 在单体测试里，因为assertEquals 使用了equals()，可以简单地写同值测试.

(50) 声明与初始化

Local变量与初始值一起声明.
理由: 最小化变量的假定值.
不好的例子:

```
        void f(int start) {
                int i, j; // 无初始值声明
                // 多的代码
                // ...
                i = start + 1;
                j = i + 1;
                // 使用i, j
        }
```

好例子:

```
void f(int start) {
        // 多的代码
        // ...
        // 使用前，首先声明与初始化
        int i = start + 1;
        int j = i + 1;
        // 使用i, j
}
```

(51) Local变量的再利用不好

通过反复使用Local变量，声明新的变量并初始化.
理由: 最小化变量的假定值.
理由: 有助于编译程序的优化.
不好的例子:

```
void f(int N, int delta) {
        int i; // 无初始值声明
        for (i = 0; i < N; i++) {
                //使用 i
        }
        for (i = 0; i < N; i++) {// 还使用i
                if (...) {
                        break;
                }
        }
        if (i != N) { // 判断是否回到最后时使用了i
                // ...
        }
        i = N - delta*2; // 再利用
        // ...
}
```

好例子:

```
void f(int N, int delta) {
        for (int i = 0; i < N; i++) {
                // 使用i
        }
        for (int i = 0; i < N; i++) {
                // 使用其他的i
                if (...) {
                        found = true;
                        break;
                }
        }
        if (found) {
                // ...
        }
        int total = N  -  delta*2; // 有其他含义的变量
        // ...
}
```

(52) if/while 条件中的“=”

if, while 的条件里，必须使用（代入）“=”.

理由: 大多情况下，是Bug。只要不是boolean 型，java 编译程序都可以捕捉该Bug.

(53) 大小比较演算子

尽量使用“<”, “<=”，请尽量避免使用”>”, “>=”.

理由: 这样统一了大小的方向，右侧总是大的一方，避免混乱。

(54) Cast

Cast尽可能用instanceof 的条件来包围.

```
C cx = null;
if (x instanceof C)
cx = (C)x;
else
evasiveAction();
```

理由: 在这里，经常会习惯性地考虑：「如果Object不是该实例?」. 不过，如果可以判断：不可能Cast时是Bug ，这种情况就不在此限制内。

(55) 异常类

异常类具有大域性的性格，难以读取多用Program流.

对于异常类，如果可以利用JDK 标准软件包已有的内容的话，最好是加以利用。而不是新创建一个异常类。

例: IOException, NoSuchFileException, IllegalArgumentException, 等是容易使用的标准异常.

新异常的创建，需做为该软件包全体的接口来研究.

(56) 方法自变量的变更是不好的。

做为原则，方法的自变量需输入，而不用做输出. 即在方法内部不调用变更自变量状态的

方法. 不在输出自变量里代入新Object(如果可能的话，声明为final).
不好的例子:

```
void moveX(Point p, int dx) {
        p.setX(p.getX()+dx); // 变更自变量(尽量避免)
}
void moveX(Point p, int dx) {
        p = new Point(p.getX()+dx, p.getY());
        // 这里不传递给调用方
}
```

例外: 注意性能的情况下

(57) 方法自变量的名字

用来使方法的自变量读取容易. 特别在与实例变量重复时，活用this，可以使自变量的读取较为容易.
不好的例子:

```
void reset(int x_, int y_) {
        x = x_;
        y = y_;
}
```

好例子:

```
void reset(int x, int y) { // 不将自变量名取为x_, y_等
        this.x = x;
        this.y = y;
}
```

(58) toString()

toString() 方法如可能要随时封装.
理由1: 用System.out.println(object)可随时打印.
理由2: 单元测试等失败时的显示比较易懂.

(59) switch,if/else 重复多了不好

在用switch 语法进行分支的处理时，要考虑这是不好设计的征兆，同时还要考虑用polymorphism能不能实现. 特别是有两个以上同样的switch 时，建议用polymorphism，**FactoryMethod，Prototype 形式**等进行Refactoring. 连续的if/else 也一样. 并且，进行null CHECK的同样的if如果很多时，请讨论是不是要用**NullObject 形式**.

(60) String 和基本型的变换

从int到String或其逆变换，如下(他的基本型也同样).

```
String s = String.valueOf(i);
int i = Integer.parseInt(s);
```

理由: 虽有其他的写法，但上述方法最易懂最有效.
其他方法: (不推荐)

```
String s = “” + i;
String s = new Integer(i).toString();
String s = Integer.toString(i); // 这样不好
int i = new Integer(s).intValue();
int i = Integer.valueOf(s).intValue();
```

(61) COLLECTION

如条件允许，请用JDK1.2 以后的COLLECTION类. 即，不用Vector,Hashtable, Enumeration 而用 List (ArrayList), Map(HashMap),Iterator.
理由1: 可更简洁地使用有逻辑，连贯性的方法名.
理由2: 通过List, Set, Map 接口，或不变更接口即能替换实现.
理由3: 同步化为OPTION，因此可写出更高速的代码(有可能性).

5. 注释

(62) javadoc 的活用
就多用/** 注释*/ . 该注释，可用javadoc或同樣的变換成HTML 形式的文档.
java 的注释有3 種.
/** ... */ javadoc 注释. 可以html 形式输出文档.
/* */ 通常的注释,内部的
// 通常的注释,内部的
public 类，方法，域必须要加/** */ 注释.

(63) 长注释
注释跨多行时，应在最初用一句概括，然后后面加长注释.

(64) javadoc TAG
/** */注释中，举从@ 开始的关键字(javadoc TAG).
@author author-name
@param paramName description
@return description of return value
@exception exceptionName description
@see string
@see URL
@see classname#methodname
param, return 中有特别要注意之处，要进行主旨注释. 例如，自变量为输出用，被变更时.
例1:

```
/**
* 取得边界框.
*
* @param b 边界框 (内存与速度效率用，输出自变量)
*/
void getBBox(BBox b) { b.min = this.min; b.max = this.max; }
```

(65) 类注释
/** */ 注释，用于机能概要，即外部的规格描述，写在类及方法的定义开始前. 该注释的第1 行有特别处理. 即，被html 的Method Index使用. 因此，最初的1 行应是注释对象的外部的机能的简短説明. 该行以，半角(.)，或
 HTML TAG结束. 接该第1 行，进行机能説明.
注释中，，写使用例等时，也可用<pre> </pre> 围住自動缩排，而避免改行.

(66) // vs. /* */
方法或类的内部注释要使用/* */ 或// ，可根据长度判断.
1 行注释最好用//.

例1:

```
/*
* 作用:
* 1. do something
* 2. do something
* 3. do somethng
*/
```

例2:

```
int index = -1; // -1 验证值的意思
```

参考: 这个方法最好.

```
static final int INVALID= -1;
int index = INVALID;
```

(67) Design by Contract(合同设计)

因为要按合同进行设计，所以请在Project设置Assert 类. 用Assert 类来表现，合同.

例:

```
class Stack {
        private int capacity;
        private int itemCount;
        public void push(Object o) {
                Assert.require(o != null); // 事前条件
                // ...
                // ...
                Assert.ensure(this.contains(o)); // 事后条件
        }
        public boolean invariant() { // 不变条件
                Assert.invariant(0 <= capacity);
                Assert.invariant(0 <= itemCount);
                Assert.invariant(itemCount <= capacity);
                return true;
        }
}
```

注意: 用户输入CHECK等不能assert. 为捕捉BUG请用assert.

补充: J2SE1.4 以后，请利用assert关键字.

6. 性能

(68) 首先检测

性能改善首先要从检测开始. 不能随意.

(69) new

java 中，new费时间. heavyweight中调用new时，如需要要用输出自变量.

```
X getX() {
        return new X(this.value);
}
```

比较慢时，让调用方new，如

```
void getX(X x) {
```

```
            x.setValue(this.value);
}
```

(70) synchronized

synchronized 较费時間. 请不要同步全部类，而请仅对必要部分进行synchronized. 另，Vector, Hashtable有默认同步化的重载. 请自愿地使用ArrayList, HashMap，并仅对必要部分进行同步化

(用Collections.synchronizedCollection进行外部同步).

(71) 向变量赋null

在出现大量不使用变量时，请主动给其赋null. 特别，对数组元素(性能要求较严时).

理由: 有助于Garbage Collection

7. 其他

(72) 在自已重新改变代码前协商

对其他人作成的类如需新的操作，若要自已extends该类来作成新类，或将该类作为实例变量作成类时,首先要与该类的作成者商谈.如通用形式满足其要求,则可将全体作成COMPACT.

(73) 复杂的设计不好

设计有迷惑时，多数情况是重视‘Simplicity’(简单) 者，与java 言語的特性良好一致. Java语言的设计原理是KISS(Keep It Small and Simple). 同时，随后的实现维护中‘Simplicity’很重要.

(74) 性能调整应在功能测试后

编码不能一开头就注重性能. 要优先保证易读，易维护. 性能应在功能正确的基础上改善.

(75) 过于精巧的代码不好

要写一般的Java的程序员都能理解的代码. 要假定对于运算符的顺序，初始化等有关的规则等，誰不能肯定有自信，要用()明确运算符顺序，进行明确的初始化这样才易读.

坏例子: return cond == 0 ? a < b && b < c : d == 1;

好例子: return (cond == 0) ? ((a < b) && (b < c)) : (d == 1);

坏例子:

```
// 作成单位行列，但费时间，谁也不好说.
for (int i = 1; i <= N; i++)
    for (int j = 1; j <= N; j++)
        M[i-1][j-1] = (i/j)* (j/i);
```