Code Review CheckList-Java

序号	分类	检查项	备注
1	定义	每个Java 文件只包含一个public 类或者接口	
2		第一行非注释的语句应该为一个package 语句(如果存在的话). 之后应该是 import 语句	
3	布局	一行不要超过120个字符，源代码一般不会超过这个宽度, 并导致无法完整显示, 但这一设置也可以灵活调整。在任何情况下, 超长的语句应该在一个逗号或者一个操作符后折行。 一条语句折行后, 应该比原来的语句再缩进4个字符。	
4		一个表达式不能被放在一行中时，将它分成多行例如： foo(long_expression1, long_expression2, long_expression3);	
5		函数代码行数的限制：200行	

6		左括号和后一个字符之间不应该出现空格, 同样, 右括号和前一个字符之间也不应该出现空格。不要在语句中使用无意义的括号. 括号只应该为达到某种目的而出现在源代码中。下面的例子说明错误和正确的用法：
		if ((i) = 42) // 错误 - 括号毫无意义
		if (i == 42) // 正确 - 的确需要括号	

7	注释	所有的 Java文件都应该在类定义前编写类注释， 注释包括了类名, 版本 , 日期和版权提示	
8		比较复杂的代码段需要描述它的实现原理，且不要以直接翻译代码的形式描述。	

9		恰当的使用四种样式的实现注释: 块, 单行, 尾, 和行尾注释
		例如：
		块注释
		/*
		* This is a block comment.
		* Each line is preceded by an asterisks.
		*/
		单行注释
		if (condition) {
		/* Short description about the following statments */
		statements;
		}
		尾注释
		if (condition) {
		stateme	

10		每个声明应该单独占一行并必须由一个尾注释. 使用tabs分隔开来数据类型,标识符,和注释字段例如：
		int level; // indentation level
		int size; // size of table
		Object currentEntry; // currently selected table entry	

11	命名惯例	类名要简单准确. 避免使用不易理解的缩写词和缩写符,由名词组成, 每个单词的第一个字母大写,例如：StringBuffer，Vector。 建议在某写特定类型的类名后加上该类型的说明，以便让类的使用者一目了然。 例如：Applet——UnitDetachApplet Servlet——UnitDetachServlet Dialog——CardAssQueryDialog	
12		接口名使用与类名一样的单词首字母大写规则	
13		方法名应该为动词或动词短语,使用多个单词场合的时候第一个单词的第一个字母小写,其他单词的第一个字母大写. 例: run(); isUpperCase(); getBackground(); findTotalCost(int Years);	

14		参数的名字必须和变量的命名规范一致。使用有意义的参数命名，如果可能的话，使用和要赋值的字段一样的名字。例如：
		public void setCounter(int size)
		{
		this.size = size;
		}	

15		static final基本类型常量名中的所有字母都应该大写(这样便可标志出它们属于编译期的常数)。	
16		变量名的大小写规则与方法名的大小写规则相同，变量名应该充分的反映出所定义变量的意义，由名词组成	
17		包的名字应该都是由小写单词组成。它们全都是小写字母，即便中间的单词亦是如此。对于域名扩展名称，如com，org，net或者edu等，全部都应小写。 例如：package com.neusoft.unieap.ui； 包名必须能概括包内所有成员类的共有特性。 建议：在多级包名中，下级包名不要重复包含上级包名的含义。 例如：有这样一个包： com.neusoft.elarp.business.alteration.employeealteration，它的命名就不是十分合理，因为在三级包名中已经声明了alteration，所以在四级包名中就没有必要重复定义它。合理的定义应为com.neusoft.elarp.business.alteration.employee，这与多重路径的命名规则是一样的。	
18	**编码规则**	禁止用一条语句对几个变量赋值	
19		对于复杂的表达式使用圆括号表示它们之间的逻辑关系	
20		boolean类型之间的变量或表达式禁止进行比较，例如： 错误：if (返回boolean类型的表达式 == true) 正确：if (返回boolean类型的表达式)	
21		import中禁止引入 * 的形式 保证所有引入的类都被使用。原因：如果不引入确切的类将很难理解当前类的上下文关系及其相关性。	
22		异常的捕获形式：将连续的小的“try-catch”块合并到一起。由于这些块将代码分割成小的、各自独立的片断，所以会妨碍编译器进行优化。但是，若过份热衷于删除异常处理模块，也可能造成代码健壮程度的下降，所以在合并的同时需要考虑程序的稳定性。	
23		运算语句的性能优化：见下表（标准时间 = 语句执行时间/本地赋值时间） 运算 示例 标准时间 本地赋值 i=n; 1.0 实例赋值 this.i=n; 1.2 int增值 i++; 1.5 byte增值 b++; 2.0 short增值 s++; 2.0 float增值 f++; 2.0 double增值 d++; 2.0 空循环 while(true) n++; 2.0 三元表达式 (x<0)?-x:x; 2.2 算术调用 Math.abs(x); 2.5 数组赋值 a[0]=n; 2.7 Long增值 l++; 3.5 方法调用 funct(); 5.9 Throw 或者catch 违例 try{throw e;} 320 同步方法调用 synchMethod(); 570 新建对象 new Object(); 980 新建数组 new int[10]; 3100	
24		线程的使用： 1 防止过多的同步 不必要的同步常常会造成程序性能的下降。因此，如果程序是单线程，则一定不要使用同步。 2 避免同步整个代码段 对某个方法或函数进行同步比对整个代码段进行同步的性能要好。因为代码段的同步牵涉的范围比对某个方法或函数进行同步广。 3 对每个对象使用多”锁”的机制来增大并发 一般每个对象都只有一个”锁”，这就表明如果两个线程执行一个对象的两个不同的同步方法时，会发生”死锁”。即使这两个方法并不共享任何资源。为了避免这个问题，可以对一个对象实行”多锁”的机制。	
25		break： 遍历数组、向量或者树结构时，如果满足条件的元素找到，一定要使用 break 语句退出循环。	

26		循环体内避免构建新对象： 如果在循环体内用到新对象，需要在循环体开始以前构建好该对象。由标准时间表可以看出构建对象有很大的系统消耗，并且在一次循环中还要清除掉该对象，下次循环再重新构建。	
27		数据库连接、数据库事务对象、输入输出流在使用后一定要关闭。	
28		异常的处理要全面，不要将影响到业务操作的异常抛给容器来处理。	
29		静态方法的使用: 错误： XXXFactory f = new XXXFactory(); f.getXXX(); 正确： XXXFactory.getXXX();	

30		循环语句的使用：
		错误：
		for(int i = 0; i < list.size(); i++)，list.size()要参与size()次循环，影响性能。
		正确：
		for(int i = 0, n = list.size(); i < n; i++)	

31		对声明的本地变量进行初始化，除非变量的初始化值不能够被直接给出。	
32		声明放在代码块的开始部，对所有要用到的变量进行集中声明，for循环可以例外。	
33		对于2个日期进行减法，不能使用字符串的转数值相减，而应该使用专门的日期减法方法。	