****命名规范****  
定义这个规范的目的是让项目中所有的文档都看起来像一个人写的，增加可读性，减少项目组中因为换人而带来的损失。（这些规范并不是一定要绝对遵守，但是一定要让程序有良好的可读性）

* Package 的命名  
  Package 的名字应该都是由一个小写单词组成。
* Class 的命名  
  Class 的名字必须由大写字母开头而其他字母都小写的单词组成
* Class 变量的命名  
  变量的名字必须用一个小写字母开头。后面的单词用大写字母开头。
* Static Final 变量的命名  
  Static Final 变量的名字应该都大写，并且指出完整含义。
* 参数的命名  
  参数的名字必须和变量的命名规范一致。
* 数组的命名  
  数组应该总是用下面的方式来命名：

             byte[] buffer;

* 而不是：

             byte buffer[];

* 方法的参数  
  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：

SetCounter(int size){

    this.size = size;

}

****Java 文件样式****  
所有的 Java(\*.java) 文件都必须遵守如下的样式规则  
package 行要在 import 行之前，import 中标准的包名要在本地的包名之前，而且按照字母顺序排列。如果 import 行中包含了同一个包中的不同子目录，则应该用 \* 来处理。

package hotlava.net.stats;

import java.io.\*;

import java.util.Observable;

import hotlava.util.Application;

* 这里 java.io.\* 使用来代替InputStream and OutputStream 的。
* Class  
  接下来的是类的注释，一般是用来解释类的。

/\*\*

\* A class representing a set of packet and byte counters

\* It is observable to allow it to be watched, but only

\* reports changes when the current set is complete

\*/

* 接下来是类定义，包含了在不同的行的 extends 和 implements

public class CounterSet

        extends Observable

        implements Cloneable

* Class Fields  
  接下来是类的成员变量：

/\*\*

\* Packet counters

\*/

protected int[] packets;

* public 的成员变量必须生成文档（JavaDoc）。proceted、private和 package 定义的成员变量如果名字含义明确的话，可以没有注释。
* 存取方法  
  接下来是类变量的存取的方法。它只是简单的用来将类的变量赋值获取值的话，可以简单的写在一行上。

/\*\*

\* Get the counters

\* @return an array containing the statistical data.  This array has been

\* freshly allocated and can be modified by the caller.

\*/

public int[] getPackets() { return copyArray(packets, offset); }

public int[] getBytes() { return copyArray(bytes, offset); }

public int[] getPackets() { return packets; }

public void setPackets(int[] packets) { this.packets = packets; }

* 其它的方法不要写在一行上
* 构造函数  
  接下来是构造函数，它应该用递增的方式写（比如：参数多的写在后面）。  
  访问类型 ("public", "private" 等.) 和 任何 "static", "final" 或 "synchronized" 应该在一行中，并且方法和参数另写一行，这样可以使方法和参数更易读。

public

CounterSet(int size){

  this.size = size;

}

* 克隆方法  
  如果这个类是可以被克隆的，那么下一步就是 clone 方法：

public

Object clone() {

  try {

    CounterSet obj = (CounterSet)super.clone();

    obj.packets = (int[])packets.clone();

    obj.size = size;

    return obj;

  }catch(CloneNotSupportedException e) {

    throw new InternalError("Unexpected CloneNotSUpportedException: " + e.getMessage());

  }

}

* 类方法  
  下面开始写类的方法：

/\*\*

\* Set the packet counters

\* (such as when restoring from a database)

\*/

protected final

void setArray(int[] r1, int[] r2, int[] r3, int[] r4)

  throws IllegalArgumentException

{

  //

  // Ensure the arrays are of equal size

  //

  if (r1.length != r2.length || r1.length != r3.length || r1.length != r4.length)

    throw new IllegalArgumentException("Arrays must be of the same size");

  System.arraycopy(r1, 0, r3, 0, r1.length);

  System.arraycopy(r2, 0, r4, 0, r1.length);

}

* toString 方法  
  无论如何，每一个类都应该定义 toString 方法：

public

String toString() {

  String retval = "CounterSet: ";

    for (int i = 0; i < data.length(); i++) {

      retval += data.bytes.toString();

      retval += data.packets.toString();

    }

    return retval;

  }

}

* main 方法  
  如果main(String[]) 方法已经定义了, 那么它应该写在类的底部.

****代码编写格式****

* 代码样式  
  代码应该用 unix 的格式，而不是 windows 的（比如：回车变成回车+换行）
* 文档化  
  必须用 javadoc 来为类生成文档。不仅因为它是标准，这也是被各种 java 编译器都认可的方法。使用 @author 标记是不被推荐的，因为代码不应该是被个人拥有的。
* 缩进  
  缩进应该是每行2个空格. 不要在源文件中保存Tab字符. 在使用不同的源代码管理工具时Tab字符将因为用户设置的不同而扩展为不同的宽度.  
  如果你使用 UltrEdit 作为你的 Java 源代码编辑器的话，你可以通过如下操作来禁止保存Tab字符, 方法是通过 UltrEdit中先设定 Tab 使用的长度室2个空格，然后用 Format|Tabs to Spaces 菜单将 Tab 转换为空格。
* 页宽  
  页宽应该设置为80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2个字符.
* {} 对  
  {} 中的语句应该单独作为一行. 例如, 下面的第1行是错误的, 第2行是正确的:

if (i>0) { i ++ }; // 错误, { 和 } 在同一行

if (i>0) {

i ++

};       // 正确, { 单独作为一行

} 语句永远单独作为一行.

* 如果 } 语句应该缩进到与其相对应的 { 那一行相对齐的位置。
* 括号  
  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. 下面的例子说明括号和空格的错误及正确使用:  
    
  CallProc( AParameter ); // 错误  
  CallProc(AParameter); // 正确  
    
  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) or (J == 42) then // 正确 - 的确需要括号

****程序编写规范****

* exit()  
  exit 除了在 main 中可以被调用外，其他的地方不应该调用。因为这样做不给任何代码代码机会来截获退出。一个类似后台服务地程序不应该因为某一个库模块决定了要退出就退出。
* 异常  
  申明的错误应该抛出一个RuntimeException或者派生的异常。  
  顶层的main()函数应该截获所有的异常，并且打印（或者记录在日志中）在屏幕上。
* 垃圾收集  
  JAVA使用成熟的后台垃圾收集技术来代替引用计数。但是这样会导致一个问题：你必须在使用完对象的实例以后进行清场工作。比如一个prel的程序员可能这么写：

...

    {

        FileOutputStream fos = new FileOutputStream(projectFile);

        project.save(fos, "IDE Project File");

    }

    ...

* 除非输出流一出作用域就关闭，非引用计数的程序语言，比如JAVA，是不能自动完成变量的清场工作的。必须象下面一样写：

    FileOutputStream fos = new FileOutputStream(projectFile);

    project.save(fos, "IDE Project File");

    fos.close();

* Clone  
  下面是一种有用的方法：

  implements Cloneable

  public

    Object clone()

    {

      try {

        ThisClass obj = (ThisClass)super.clone();

        obj.field1 = (int[])field1.clone();

        obj.field2 = field2;

        return obj;

      } catch(CloneNotSupportedException e) {

        throw new InternalError("Unexpected CloneNotSUpportedException: " + e.getMessage());

      }

  }

* final 类  
  绝对不要因为性能的原因将类定义为 final 的（除非程序的框架要求）  
  如果一个类还没有准备好被继承，最好在类文档中注明，而不要将她定义为 final 的。这是因为没有人可以保证会不会由于什么原因需要继承她。
* 访问类的成员变量  
  大部分的类成员变量应该定义为 protected 的来防止继承类使用他们。  
  注意，要用"int[] packets"，而不是"int packets[]"，后一种永远也不要用。

    public void setPackets(int[] packets) { this.packets = packets; }

      CounterSet(int size)

        {

          this.size = size;

        }

****编程技巧****

* byte 数组转换到 characters  
  为了将 byte 数组转换到 characters，你可以这么做：  
    
  "Hello world!".getBytes();
* Utility 类  
  Utility 类（仅仅提供方法的类）应该被申明为抽象的来防止被继承或被初始化。
* 初始化  
  下面的代码是一种很好的初始化数组的方法：  
    
  objectArguments = new Object[] { arguments };
* 枚举类型  
  JAVA 对枚举的支持不好，但是下面的代码是一种很有用的模板：

class Colour {

  public static final Colour BLACK = new Colour(0, 0, 0);

  public static final Colour RED = new Colour(0xFF, 0, 0);

  public static final Colour GREEN = new Colour(0, 0xFF, 0);

  public static final Colour BLUE = new Colour(0, 0, 0xFF);

  public static final Colour WHITE = new Colour(0xFF, 0xFF, 0xFF);

}

* 这种技术实现了RED, GREEN, BLUE 等可以象其他语言的枚举类型一样使用的常量。 他们可以用 '==' 操作符来比较。  
  但是这样使用有一个缺陷：如果一个用户用这样的方法来创建颜色 BLACK  
    
  new Colour(0,0,0)  
    
  那么这就是另外一个对象，'=='操作符就会产生错误。她的 equal() 方法仍然有效。由于这个原因，这个技术的缺陷最好注明在文档中，或者只在自己的包中使用。

****Swing****

* 避免使用 AWT 组件
  + 混合使用 AWT 和 Swing 组件  
    如果要将 AWT 组件和 Swing 组件混合起来使用的话，请小心使用。实际上，尽量不要将他们混合起来使用。
  + 滚动的 AWT 组件  
    AWT 组件绝对不要用 JscrollPane 类来实现滚动。滚动 AWT 组件的时候一定要用 AWT ScrollPane 组件来实现。
  + 避免在 InternalFrame 组件中使用 AWT 组件  
    尽量不要这么做，要不然会出现不可预料的后果。
  + Z-Order 问题  
    AWT 组件总是显示在 Swing 组件之上。当使用包含 AWT 组件的 POP-UP 菜单的时候要小心，尽量不要这样使用。

****调试****

* 调试在软件开发中是一个很重要的部分，存在软件生命周期的各个部分中。调试能够用配置开、关是最基本的。  
    
  很常用的一种调试方法就是用一个 PrintStream 类成员，在没有定义调试流的时候就为 null，类要定义一个 debug 方法来设置调试用的流。

****性能****

* 在写代码的时候，从头至尾都应该考虑性能问题。这不是说时间都应该浪费在优化代码上，而是我们时刻应该提醒自己要注意代码的效率。比如：如果没有时间来实现一个高效的算法，那么我们应该在文档中记录下来，以便在以后有空的时候再来实现她。  
    
  不是所有的人都同意在写代码的时候应该优化性能这个观点的，他们认为性能优化的问题应该在项目的后期再去考虑，也就是在程序的轮廓已经实现了以后。
* 不必要的对象构造
  + 不要在循环中构造和释放对象
  + 使用 StringBuffer 对象  
    在处理 String 的时候要尽量使用 StringBuffer 类，StringBuffer 类是构成 String 类的基础。String 类将 StringBuffer 类封装了起来，（以花费更多时间为代价）为开发人员提供了一个安全的接口。当我们在构造字符串的时候，我们应该用 StringBuffer 来实现大部分的工作，当工作完成后将 StringBuffer 对象再转换为需要的 String 对象。比如：如果有一个字符串必须不断地在其后添加许多字符来完成构造，那么我们应该使用 StringBuffer 对象和她的 append() 方法。如果我们用 String 对象代替 StringBuffer 对象的话，会花费许多不必要的创建和释放对象的 CPU 时间。
* 避免太多的使用 synchronized 关键字  
  避免不必要的使用关键字 synchronized，应该在必要的时候再使用她，这是一个避免死锁的好方法。

****可移植性****  
Borland Jbulider 不喜欢 synchronized 这个关键字，如果你的断点设在这些关键字的作用域内的话，调试的时候你会发现的断点会到处乱跳，让你不知所措。除非必须，尽量不要使用。

* 换行  
  如果需要换行的话，尽量用 println 来代替在字符串中使用"/n"。  
  你不要这样：  
    
  System.out.print("Hello,world!/n");  
    
  要这样：  
    
  System.out.println("Hello,world!");  
    
  或者你构造一个带换行符的字符串，至少要象这样：  
    
  String newline = System.getProperty("line.separator");  
  System.out.println("Hello world" + newline);
* PrintStream  
  PrintStream 已经被不赞成（deprecated）使用，用 PrintWrite 来代替它。
* 代码：

1. **package** com.common.file;
3. **import** java.io.File;
4. **import** java.io.FileInputStream;
5. **import** java.io.FileNotFoundException;
6. **import** java.io.FileOutputStream;
7. **import** java.io.IOException;
8. **import** java.io.InputStream;
9. **import** java.io.OutputStream;
10. **import** java.text.DateFormat;
11. **import** java.util.Date;
12. **import** java.util.Iterator;
14. **import** javax.swing.text.html.HTMLDocument.HTMLReader.FormAction;
16. /\*\*
17. \*
18. \* 功能描述：
19. \*
20. \* @author Administrator
21. \* @Date Jul 19, 2008
22. \* @Time 9:46:11 AM
23. \* @version 1.0
24. \*/
25. **public** **class** FileUtil {
27. /\*\*
28. \* 功能描述：列出某文件夹及其子文件夹下面的文件，并可根据扩展名过滤
29. \*
30. \* @param path
31. \*            文件夹
32. \*/
33. **public** **static** **void** list(File path) {
34. **if** (!path.exists()) {
35. System.out.println("文件名称不存在!");
36. } **else** {
37. **if** (path.isFile()) {
38. **if** (path.getName().toLowerCase().endsWith(".pdf")
39. || path.getName().toLowerCase().endsWith(".doc")
40. || path.getName().toLowerCase().endsWith(".chm")
41. || path.getName().toLowerCase().endsWith(".html")
42. || path.getName().toLowerCase().endsWith(".htm")) {// 文件格式
43. System.out.println(path);
44. System.out.println(path.getName());
45. }
46. } **else** {
47. File[] files = path.listFiles();
48. **for** (**int** i = 0; i < files.length; i++) {
49. list(files[i]);
50. }
51. }
52. }
53. }
55. /\*\*
56. \* 功能描述：拷贝一个目录或者文件到指定路径下，即把源文件拷贝到目标文件路径下
57. \*
58. \* @param source
59. \*            源文件
60. \* @param target
61. \*            目标文件路径
62. \* @return void
63. \*/
64. **public** **static** **void** copy(File source, File target) {
65. File tarpath = **new** File(target, source.getName());
66. **if** (source.isDirectory()) {
67. tarpath.mkdir();
68. File[] dir = source.listFiles();
69. **for** (**int** i = 0; i < dir.length; i++) {
70. copy(dir[i], tarpath);
71. }
72. } **else** {
73. **try** {
74. InputStream is = **new** FileInputStream(source); // 用于读取文件的原始字节流
75. OutputStream os = **new** FileOutputStream(tarpath); // 用于写入文件的原始字节的流
76. **byte**[] buf = **new** **byte**[1024];// 存储读取数据的缓冲区大小
77. **int** len = 0;
78. **while** ((len = is.read(buf)) != -1) {
79. os.write(buf, 0, len);
80. }
81. is.close();
82. os.close();
83. } **catch** (FileNotFoundException e) {
84. e.printStackTrace();
85. } **catch** (IOException e) {
86. e.printStackTrace();
87. }
88. }
89. }
91. /\*\*
92. \* @param args
93. \*/
94. **public** **static** **void** main(String[] args) {
95. // TODO Auto-generated method stub
96. File file = **new** File("F:\\Tomcat");
97. list(file);
98. Date myDate = **new** Date();
99. DateFormat df = DateFormat.getDateInstance();
100. System.out.println(df.format(myDate));
101. }
103. }