**1 命名规范(Naming Conventions)**

命名规范使程序更易读，从而更易于理解。它们也可以提供一些有关标识符功能的信息，以助于理解代码，例如，不论它是一个常量，包，还是类。

标识符类型 命名规则 例子

包(Packages) 一个唯一包名的前缀总是全部小写的ASCII字母并且是一个顶级域名，通常是com，edu，gov，mil，net，org，或1981年ISO 3166标准所指定的标识国家的英文双字符代码。包名的后续部分根据不同机构各自内部的命名规范而不尽相同。这类命名规范可能以特定目录名的组成来区分部门(department)，项目(project)，机器(machine)，或注册名(login names)。 com.sun.eng

com.apple.quicktime.v2

edu.cmu.cs.bovik.cheese

类(Classes) 命名规则：类名是个一名词，采用大小写混合的方式，每个单词的首字母大写。尽量使你的类名简洁而富于描述。使用完整单词，避免缩写词(除非该缩写词被更广泛使用，像URL，HTML) class Raster;

class ImageSprite;

接口(Interfaces) 命名规则：大小写规则与类名相似 interface RasterDelegate;

interface Storing;

方法(Methods) 方法名是一个动词，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。 run();

runFast();

getBackground();

变量(Variables) 除了变量名外，所有实例，包括类，类常量，均采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。

变量名应简短且富于描述。变量名的选用应该易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。临时变量通常被取名为i，j，k，m和n，它们一般用于整型；c，d，e，它们一般用于字符型。 char c;

int i;

float myWidth;

实例变量(Instance Variables) 大小写规则和变量名相似，除了前面需要一个下划线 int \_employeeId;

String \_name;

Customer \_customer;

常量(Constants) 类常量和ANSI常量的声明，应该全部大写，单词间用下划线隔开。(尽量避免ANSI常量，容易引起错误) static final int MIN\_WIDTH = 4;

static final int MAX\_WIDTH = 999;

static final int GET\_THE\_CPU = 1;

**2 文件名(File Names)**

这部分列出了常用的文件名及其后缀。

**2.1 文件后缀(File Suffixes)**

Java程序使用下列文件后缀：

文件类别 文件后缀

Java源文件 .java

Java字节码文件 .class

**2.2 常用文件名(Common File Names)**

常用的文件名包括：

文件名 用途

GNUmakefile makefiles的首选文件名。我们采用gnumake来创建（build）软件。

README 概述特定目录下所含内容的文件的首选文件名

**3 开头注释(Beginning Comments)**

所有的源文件都应该在开头有一个C语言风格的注释，其中列出类名、版本信息、日期和版权声明：

  /\*

   \* Classname

   \*

   \* Version information

   \*

   \* Date

   \*

   \* Copyright notice

   \*/

**4 缩进排版(Indentation)**

统一使用TAB

**5. 实现注释的格式(Implementation Comment Formats)**

程序可以有4种实现注释的风格：块(block)、单行(single-line)、尾端(trailing)和行末(end-of-line)。

**5.1 块注释(Block Comments)**

块注释通常用于提供对文件，方法，数据结构和算法的描述。块注释被置于每个文件的开始处以及每个方法之前。它们也可以被用于其他地方，比如方法内部。在功能和方法内部的块注释应该和它们所描述的代码具有一样的缩进格式。

块注释之首应该有一个空行，用于把块注释和代码分割开来，比如：

  /\*

   \* Here is a block comment.

   \*/

块注释可以以/\*-开头，这样indent(1)就可以将之识别为一个代码块的开始，而不会重排它。

  /\*-

    \* Here is a block comment with some very special

    \* formatting that I want indent(1) to ignore.

    \*

    \*    one

    \*        two

    \*            three

    \*/

注意：如果你不使用indent(1)，就不必在代码中使用/\*-，或为他人可能对你的代码运行indent(1)作让步。

参见"文档注释"

**5.2 单行注释(Single-Line Comments)**

短注释可以显示在一行内，并与其后的代码具有一样的缩进层级。如果一个注释不能在一行内写完，就该采用块注释(参见"块注释")。单行注释之前应该有一个空行。以下是一个Java代码中单行注释的例子：

  if (condition) {

    /\* Handle the condition. \*/

    ...

  }

**5.3 尾端注释(Trailing Comments)**

极短的注释可以与它们所要描述的代码位于同一行，但是应该有足够的空白来分开代码和注释。若有多个短注释出现于大段代码中，它们应该具有相同的缩进。

以下是一个Java代码中尾端注释的例子：

  if (a == 2) {

      return TRUE;              /\* special case \*/

  } else {

      return isPrime(a);         /\* works only for odd a \*/

  }

**5.4 行末注释(End-Of-Line Comments)**

注释界定符"//"，可以注释掉整行或者一行中的一部分。它一般不用于连续多行的注释文本；然而，它可以用来注释掉连续多行的代码段。以下是所有三种风格的例子：

  if (foo > 1) {

      // Do a double-flip.

      ...

  }

  else {

      return false;          // Explain why here.

  }

  //if (bar > 1) {

  //

  //    // Do a triple-flip.

  //    ...

  //}

  //else {

  //    return false;

  //}

**6 声明(Declarations)**

**6.1 每行声明变量的数量(Number Per Line)**

推荐一行一个声明，因为这样以利于写注释。亦即，

  int level;  // indentation level

  int size;   // size of table

要优于，

int level, size;

不要将不同类型变量的声明放在同一行，例如：

  int foo,  fooarray[];   //WRONG!

注意：上面的例子中，在类型和标识符之间放了一个空格，另一种被允许的替代方式是使用制表符：

  int        level;         // indentation level

  int        size;          // size of table

  Object    currentEntry;  // currently selected table entry

**6.2 初始化(Initialization)**

尽量在声明局部变量的同时初始化。唯一不这么做的理由是变量的初始值依赖于某些先前发生的计算。

**6.3 布局(Placement)**

只在代码块的开始处声明变量。（一个块是指任何被包含在大括号"{"和"}"中间的代码。）不要在首次用到该变量时才声明之。这会把注意力不集中的程序员搞糊涂，同时会妨碍代码在该作用域内的可移植性。

  void myMethod() {

      int int1 = 0;         // beginning of method block

      if (condition) {

          int int2 = 0;     // beginning of "if" block

          ...

      }

  }

该规则的一个例外是for循环的索引变量

  for (int i = 0; i < maxLoops; i++) { ... }

避免声明的局部变量覆盖上一级声明的变量。例如，不要在内部代码块中声明相同的变量名：

  int count;

  ...

  myMethod() {

      if (condition) {

          int count = 0;     // AVOID!

          ...

      }

      ...

  }

**6.4 类和接口的声明(Class and Interface Declarations)**

当编写类和接口是，应该遵守以下格式规则：

- 在方法名与其参数列表之前的左括号"("间不要有空格

- 左大括号"{"位于声明语句同行的末尾

- 右大括号"}"另起一行，与相应的声明语句对齐，除非是一个空语句，"}"应紧跟在"{"之后

  class Sample extends Object {

      int ivar1;

      int ivar2;

      Sample(int i, int j) {

          ivar1 = i;

          ivar2 = j;

      }

      int emptyMethod() {}

      ...

  }

- 方法与方法之间以空行分隔

**7 语句(Statements)**

**7.1 简单语句(Simple Statements)**

每行至多包含一条语句，例如：

  argv++;       // Correct

  argc--;       // Correct

  argv++; argc--;       // AVOID!

**7.2 复合语句(Compound Statements)**

复合语句是包含在大括号中的语句序列，形如"{ 语句 }"。例如下面各段。

- 被括其中的语句应该较之复合语句缩进一个层次

- 左大括号"{"应位于复合语句起始行的行尾；右大括号"}"应另起一行并与复合语句首行对齐。

- 大括号可以被用于所有语句，包括单个语句，只要这些语句是诸如if-else或for控制结构的一部分。这样便于添加语句而无需担心由于忘了加括号而引入bug。

**7.3 返回语句(return Statements)**

一个带返回值的return语句不使用小括号"()"，除非它们以某种方式使返回值更为显见。例如：

  return;

  return myDisk.size();

  return (size ? size : defaultSize);

**7.4 if，if-else，if else-if else语句(if, if-else, if else-if else Statements)**

if-else语句应该具有如下格式：

  if (condition) {

      statements;

  }

  if (condition) {

      statements;

  } else {

      statements;

  }

  if (condition) {

      statements;

  } else if (condition) {

      statements;

  } else{

      statements;

  }

注意：if语句总是用"{"和"}"括起来，避免使用如下容易引起错误的格式：

  if (condition) //AVOID! THIS OMITS THE BRACES {}!

      statement;

**7.5 for语句(for Statements)**

一个for语句应该具有如下格式：

  for (initialization; condition; update) {

      statements;

  }

一个空的for语句(所有工作都在初始化，条件判断，更新子句中完成）应该具有如下格式：

  for (initialization; condition; update);

当在for语句的初始化或更新子句中使用逗号时，避免因使用三个以上变量，而导致复杂度提高。若需要，可以在for循环之前(为初始化子句)或for循环末尾(为更新子句)使用单独的语句。

**7.6 while语句(while Statements)**

一个while语句应该具有如下格式

  while (condition) {

      statements;

  }

一个空的while语句应该具有如下格式：

  while (condition);

**7.7 do-while语句(do-while Statements)**

一个do-while语句应该具有如下格式：

  do {

      statements;

  } while (condition);

**7.8 switch语句(switch Statements)**

一个switch语句应该具有如下格式：

  switch (condition) {

  case ABC:

      statements;

      /\* falls through \*/

  case DEF:

      statements;

      break;

  case XYZ:

      statements;

      break;

  default:

      statements;

      break;

  }

每当一个case顺着往下执行时(因为没有break语句)，通常应在break语句的位置添加注释。上面的示例代码中就包含注释/\* falls through \*/。

**7.9 try-catch语句(try-catch Statements)**

一个try-catch语句应该具有如下格式：

  try {

      statements;

  } catch (ExceptionClass e) {

      statements;

  }

一个try-catch语句后面也可能跟着一个finally语句，不论try代码块是否顺利执行完，它都会被执行。

  try {

      statements;

  } catch (ExceptionClass e) {

      statements;

  } finally {

      statements;

  }

**8 空白(White Space)**

**8.1 空行(Blank Lines)**

空行将逻辑相关的代码段分隔开，以提高可读性。

下列情况应该总是使用两个空行：

- 一个源文件的两个片段(section)之间

- 类声明和接口声明之间

下列情况应该总是使用一个空行：

- 两个方法之间

- 方法内的局部变量和方法的第一条语句之间

- 块注释（参见"5.1.1"）或单行注释（参见"5.1.2"）之前

- 一个方法内的两个逻辑段之间，用以提高可读性

**8.2 空格(Blank Spaces)**

下列情况应该使用空格：

- 一个紧跟着括号的关键字应该被空格分开，例如：

  while (true) {

      ...

  }

注意：空格不应该置于方法名与其左括号之间。这将有助于区分关键字和方法调用。

- 空白应该位于参数列表中逗号的后面

- 所有的二元运算符，除了"."，应该使用空格将之与操作数分开。一元操作符和操作数之间不因该加空格，比如：负号("-")、自增("++")和自减("--")。例如：

    a += c + d;

    a = (a + b) / (c \* d);

    while (d++ = s++) {

        n++;

    }

    printSize("size is " + foo + "\n");

- for语句中的表达式应该被空格分开，例如：

    for (expr1; expr2; expr3)

- 强制转型后应该跟一个空格，例如：

    myMethod((byte) aNum, (Object) x);

    myMethod((int) (cp + 5), ((int) (i + 3)) + 1);