|  |  |
| --- | --- |
| Java Coding Standard | |
| Purpose | To guide implementation of Java programs |
| Program Headers | Begin all programs with a descriptive header. |
| Header Format | // File: [File name]  // Created: [yyyy/mm/dd creation date]  // Last Changed: $Date: 2000/08/09 15:15:25 $  // Author: <A HREF="mailto:[Email address]">[Name]</A>  //  // License  //  // History:  // $Log: javaCodingStd.html,v $  // Revision 1.1.1.1 2000/08/09 15:15:25 adamkmy  // initial impound  // |
| Listing Contents | * Provide the package name * The import list |
| Contents Example | com.java.program1  Import Arrays  Import Javac |
| Reuse Instructions | * Class description. The first sentence should be a meaningful summary of the class * Declaration * Other notes, including errors, limitations, usage instructions, examples and reminders of desired improvements |
| Reuse Instruction Example | // ClassName is a class to do something  // object ClassName (object theObject)  // This class trigger an error when you are a bad boy.  // ClassName var = new ClassName( n );  // var something = var.theMethod()  // You have to use this variable with a BR Tree |
| Identifiers | * Use /\* ... \*/ comments to describe algorithmic details, notes, and related documentation that spans more than a few code statements. * When declaring try to use complete names for the variables, NEVER use just “int c;” and so on. * Use Running // comments to clarify non-obvious code. But don't bother adding such comments to obvious code; instead try to make code obvious! |
| Identifier Example | /\*  \* Strategy:  \* 1. Find the node  \* 2. Clone it  \* 3. Ask inserter to add clone  \* 4. If successful, delete node  \*/  int index = -1; // -1 serves as flag meaning the index isn't valid |
| Comments | * Use /\* ... \*/ comments to describe algorithmic details, notes, and related documentation that spans more than a few code statements. * Use Running // comments to clarify non-obvious code. But don't bother adding such comments to obvious code; instead try to make code obvious! |
| Good Comment | /\*  \* Strategy:  \* 1. Find the node  \* 2. Clone it  \* 3. Ask inserter to add clone  \* 4. If successful, delete node  \*/  int index = -1; // -1 serves as flag meaning the index isn't valid |
| Bad Comment | Not following the last example |
| Major Sections | This is override with the Class Declaration Comment block. Look the reuse instruction section. |
| Example | // ClassName is a class to do something  // object ClassName (object theObject)  // This class trigger an error when you are a bad boy.  // ClassName var = new ClassName( n );  // var something = var.theMethod()  // You have to use this variable with a BR Tree |
| Blank Spaces | * Write programs with sufficient spacing so they do not appear crowded. * Separate every program construct with at least one space. |

|  |  |
| --- | --- |
| Indenting | * Use of tabs or spaces for indenting. * Number of spaces to indent: One before father. * Left-brace (``{'') placement at end of line or beginning of next line. * Maximum line length of 70% of the ide screen. * Spill-over indentation for breaking up long lines. |
| Indenting Example | for (int i = 0; i < 10; i++){  crazyClass varCrazy = 12; // This is a crazy variable  varCrazy.method1().method2( i ).method3().method4().method5()  .method6();  } |
| Capitalization | * Capitalize all defines. * Lowercase all other identifiers and reserved words. * To make them readable, user messages may use mixed case. * CamelCase for naming convention in classes * Follow java Naming Convention |
| Capitalization Examples | * packages   + lowercase.  All package names should start with "vanderbilt." * files   + The java compiler enforces the convention that file names have the same base name as the public class they define. * classes:   + CapitalizedWithInternalWordsAlsoCapitalized * Exception class:   + ClassNameEndsWithException. * Interface. When necessary to distinguish from similarly named classes:   + InterfaceNameEndsWithIfc. * Class. When necessary to distinguish from similarly named interfaces:   + ClassNameEndsWithImpl * constants (finals):   + UPPER\_CASE\_WITH\_UNDERSCORES * private or protected:   + myVar (i.e. prefix with my) * static private or protected:   + ourVar * local variables:   + firstWordLowerCaseButInternalWordsCapitalized * methods:   + firstWordLowerCaseButInternalWordsCapitalized() * method parameters:   + inParam, outParam, or inOutParam. If the method reads the parameter but does not modify its contents, prefix the parameter with in. If the method receives an empty parameter and sets its contents as a return value, prefix the parameter with out. If the method reads the contents of a parameter and modifies them, prefix the parameter with inOut. * factory method for objects of type X:   + newX * converter method that returns objects of type X:   + toX * method that reports an attribute x of type X:   + X getX() * method that changes an attribute x of type X:   + void setX(X value) |

https://www.mc.vanderbilt.edu/infocntr/infointgr/AppDevelopment/javaCodingStd.html