Coding conventions

Class and Interface Declatations

|  |  |  |
| --- | --- | --- |
|  | **Part of Class/Interface Declaration** | **Notes** |
| 1 | Class/interface documentation comment ( /\*\*...\*/) | See ["Documentation Comments" on page 9](#16838) for information on what should be in this comment. |
| 2 | class or interface statement |  |
| 3 | Class/interface implementation comment ( /\*...\*/), if necessary | This comment should contain any class-wide or interface-wide information that wasn't appropriate for the class/interface documentation comment. |
| 4 | Class ( static) variables | First the public class variables, then the protected, then package level (no access modifier), and then the private. |
| 5 | Instance variables | First public, then protected, then package level (no access modifier), and then private. |
| 6 | Constructors |  |
| 7 | Methods | These methods should be grouped by functionality rather than by scope or accessibility. For example, a private class method can be in between two public instance methods. The goal is to make reading and understanding the code easier. |

Line Length

-80 characters, can be slightly longer if no logical place to break

-indent 2 tabs or 8 spaces when wrapping lines

Comments

Block Comments

\* on every line of a comment

Example:

/\*

\* Here is a block comment.

\*/

Line Comments

// for any comment that takes up the whole line, or any end of line comment

/\* code \*/ for temporarily commenting out code

- comments explaining code always get a whole line

Variable Declarations

-one declaration per line, write a trailing comment describing variable’s purpose and limits on its use

-groups of similar variables should be declared without spaces between them, a single space should be added between groups of related variables

>int houseColor //some comment

>int houseHeight //come comment

>int houseLocation //come comment

>int carColor //some comment

>int carSpeed //some comment

Variable Initialization

-always initialize a variable if its initial value is known before compile time

-initialize with a value any other time it is reasonably possible

Variable Declaration Placement

-declare all variables at the top of a block except the counter for a for loop

Bracket usage

-always use brackets around a block including 1 line blocks

-all brackets get their own lines

-do this:

Method foo()

{

}

-Not this:

Method foo(){

}

Statements

-1 simple statement per line ex:

Foo++; //some comment

Not

Foo++; Bar++; //some comment

Return statements

A return statement with a value should not use parentheses unless they make the return value more obvious in some way. Example:

return;

return myDisk.size();

return (size ? size : defaultSize);

Loops

A for statement should have the following form:

for (initialization; condition; update)

{

statements;

}

If there is no initialization or update use a while loop

while (condition)

{

statements;

}

A do-while statement should have the following form:

do

{

statements;

} while (condition);

**Switch Statements**

A switch statement should have the following form:

switch (condition)

{

case ABC:

statements;

/\* falls through \*/

case DEF:

statements;

break;

case XYZ:

statements;

break;

default:

statements;

break;

}

Every time a case falls through (doesn't include a break statement), add a comment where the break statement would normally be. This is shown in the preceding code example with the /\* falls through \*/ comment.

Every switch statement should include a default case. The break in the default case is redundant, but it prevents a fall-through error if later another case is added.

\*\*\*Do not use labels and gotos outside of a switch statement \*\*\*

### try-catch Statements

A try-catch statement should have the following format:

try

{

statements;

}

catch (ExceptionClass e)

{

statements;

}

A try-catch statement may also be followed by finally, which executes regardless of whether or not the try block has completed successfully.

try

{

statements;

}

catch (ExceptionClass e)

{

statements;

}

finally

{

statements;

}

### Blank Lines

Blank lines improve readability by setting off sections of code that are logically related.

Two blank lines should always be used in the following circumstances:

* Between sections of a source file
* Between class and interface definitions

One blank line should always be used in the following circumstances:

* Between methods
* Between the local variables in a method and its first statement
* Between logical sections inside a method to improve readability

### Blank Spaces

Blank spaces should be used in the following circumstances:

* A keyword followed by a parenthesis should be separated by a space. Example:

while (true)

{

...

}

Note that a blank space should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.

* A blank space should appear after commas in argument lists.
* All binary operators except . should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands. Example:

a += c + d;

a = (a + b) / (c \* d);

while (d++ = s++)

{

n++;

}

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

* The expressions in a for statement should be separated by blank spaces. Example:

for (expr1; expr2; expr3)

* Casts should not be followed by a blank space. Examples:

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

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

Naming Conventions

|  |  |  |
| --- | --- | --- |
| **Identifier Type** | **Rules for Naming** | **Examples** |
| Packages | \*\*\*actual usage to be determined\*\*\*  The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981.  Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names. | com.sun.eng  com.apple.quicktime.v2  edu.cmu.cs.bovik.cheese |
| Classes | Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML). | class Raster;  class ImageSprite; |
| Interfaces | Interface names should be capitalized like class names. | interface RasterDelegate;  interface Storing; |
| Methods | Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. | run();  runFast();  getBackground(); |
| Variables | Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore \_ or dollar sign $ characters, even though both are allowed.  Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters. | int i;  char c;  float myWidth; |
| Constants | The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("\_"). (ANSI constants should be avoided, for ease of debugging.) | static final int MIN\_WIDTH = 4;  static final int MAX\_WIDTH = 999;  static final int GET\_THE\_CPU = 1; |