**JAVA CODE CONVENTIONS**

**We need code conventions because of the following reasons:**

* helps in software maintainance
* improves code readibility
* helps to deliver clean and well packed source code if required.

**Types of files are:**

* .java
* .class

**Conventions related to a java source file**

* A source file contains either a single public class or an interface. More that one public class or an interface in a single file is not preffered.
* Its good to have a source file in the following order:

1. **beginning comment**; For Example:

/\*

\* Classname

\*

\* Version info

\*

\* Copyright notice

\*/

2. **Package and Import statements**; for example:

import java.applet.Applet;

import java.awt.\*;

import java.net.\*;

3**. Class and interface declarations**

1. 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.
2. Class (static) variables : First the public class variables, then the protected, and then the private.
3. Instance variables: First public, then protected, and then private

4. **Methods**

1. 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.

**Conventions related to Indentation**

* **Indentation**: Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 8 spaces (not 4).
* **Line Length**:Avoid lines longer than 80 characters, since they’re not handled well by many terminals and tools.
* **Wrapping Lines**: When an expression will not fit on a single line, break it according to these general principles:

1. Break after a comma.
2. Break before an operator.
3. Prefer higher-level breaks to lower-level breaks.
4. Align the new line with the beginning of the expression at the same level on the previous line.
5. If the above rules lead to confusing code or to code that’s squished up against the right margin, just indent 8 spaces instead.

**Conventions related to Comments**

* **Implementation comments**: meant for commenting out code or for comments about the particular implementation.

1. Block Comments format: Block comments are used to provide descriptions of files, methods, data structures and algorithms. Block comments should be used at the beginning of each file and before each method. They can also be used in other places, such as within methods. Block comments inside a function or method should be indented to the same level as the code they describe. For example:  
    /\*  
    \* Here is a block comment  
    \*/
2. Single-Line Comments format: Short comments can appear on a single line indented to the level of the code that follows. If a comment can’t be written in a single line, it should follow the block comment format. For example:  
    if (condition) {  
    /\* Handle the condition. \*/  
    ...  
    }
3. Trailing Comments format: Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements. If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting. Avoid the assembly language style of commenting every line of executable code with a trailing comment. For example:  
    return TRUE; /\* special case \*/  
    } else {  
    return isprime(a); /\* works only for odd a \*/  
    }
4. End-Of-Line Comments format: The // comment delimiter begins a comment that continues to the newline. It can comment out a complete line or only a partial line. It shouldn’t be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code. For example:  
    if (foo > 1) {  
    // Do a double-flip.  
    ...  
    }  
    else  
    return false; // Explain why here.  
    //if (bar > 1) {  
    // Do a triple-flip.  
    // ...  
    //}  
    //else  
    // return false;

* **Documentation comments**: Doc comments are meant to describe the specification of the code, from an implementation-free perspective. to be read by developers who might not necessarily have the source code at hand. For example:

<http://java.sun.com/products/jdk/javadoc/writingdoccomments.html>

1. The first line of doc comment (/\*\*) for classes and interfaces is not indented; subsequent doc comment lines each have 1 space of indentation (to vertically align the asterisks). Members, including constructors, have 4 spaces for the first doc comment line and 5 spaces thereafter.
2. Doc comments should not be positioned inside a method or constructor definition block, because Java associates documentation comments with the first declaration after the comment.

**Conventions related to Declaration**

Number Per Line:

1. One declaration per line is recommended since it encourages commenting.
2. In absolutely no case should variables and functions be declared on the same line.
3. Try to initialize local variables where they’re declared. The only reason not to initialize a variable where it’s declared is if the initial value depends on some computation occurring first.
4. Each line should contain at most one statement.
5. A return statement with a value should not use parentheses unless they make the return value more obvious in some way
6. The if-else class of statements should have the following form:

if (condition) {

statements;

}

if (condition) {

statements;

} else {

statements;

}

if (condition) {

statements;

} else if (condition) {

statements;

} else if (condition) {

statements;

}

**Naming Conventions**

1. 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.
2. Interface names should be capitalized like class names
3. sMethods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.
4. Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters.
5. 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.
6. The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores (“\_”).

**Variable Assignments:**

1. Avoid assigning several variables to the same value in a single statement.
2. Do not use the assignment operator in a place where it can be easily confused with the equality operator.  
   if (c++ = d++) { // AVOID! Java disallows  
   }
3. Do not use embedded assignments in an attempt to improve run-time performance.