Coding Guideline

Contents

File Organization

Java Source Files

- Should be avoided the file longer than 900 lines
- Java source files have the following ordering:
  - Beginning comments
  - Package and import statements
  - Class and Interface declarations

Beginning Comments

```
/*
* Classname
*
* Version info
*
* Copyright notice
*/
```

Class and Interface Declarations

- Should allow the following order:
  - Class/interface documentation comment (/**...*/)
  - `class` or `interface` statement
  - Class (static) variables (public, protected, private)
  - Instance variables (public, protected, private)
  - Constructors
- Should use “public static final” instead of “public final static”

Indentation

Indentation & Line Lengths

- Four spaces should be used as the unit of indentation.
- Tabs must be set exactly every 4 spaces
- Avoid line longer than 140 characters

Wrapping Lines

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

Example 01

```
function(longExpression1, longExpression2, longExpression3,
         longExpression4, longExpression5);
var = function1(longExpression1,
                function2(longExpression2,
                          longExpression3));
```

Example 02

- The high-level breaks is preferred:

```
longName1 = longName2 * (longName3 + longName4 - longName5)
            + 4 * longname6; // PREFER
longName1 = longName2 * (longName3 + longName4
                         - longName5) + 4 * longname6; //
  AVOID
```

Example 03

- Should avoid very deep indents:

```
//INDENT 8 SPACES
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother) {
    ...
}
```

Example 04

```
//DON'T USE THIS INDENTATION
if ((condition1 && condition2)
    || (condition3 && condition4)
    ||!(condition5 && condition6)) { //BAD WRAPS
    doSomethingAboutIt();            //EASY TO MISS
}
//USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
//OR USE THIS
if ((condition1 && condition2) || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

Example 05

- Here are three acceptable ways:

```
alpha = (aLongBooleanExpression) ? beta : gamma;
alpha = (aLongBooleanExpression) ? beta
                                 : gamma;
alpha = (aLongBooleanExpression)
        ? beta
        : gamma;
```

Comments

Block Comments

- Block comments are used to provide descriptions of files, methods, data structures and algorithms.
- Should be preceded by a blank line.
- Example

```
/*
 * Here is a block comment.
 */
```

End-of-Line Comments

- Short comments can appear on the line as the code they describe.
- Example

```
if (a == 2) {
     // special case
    return TRUE;
} else {
     // works only for odd a
    return isprime(a);
}
```

Declarations

Number per Line

- One declaration per line is recommended.
- Example:

```
int level; // indentation level
int size;  // size of table
```

Placement

- Put declarations only at the beginning of blocks.
- Example:

```
void MyMethod() {
    int int1;
    if (condition) {
        int int2;
        ...
    }
}
```

Initialization

- Try to initialize local variables where they're declared.
- Not to initialize a variable where it's declared is if the initial value depends on some computation occurring first.

Class and Interface Declarations

- The following formatting rules should be followed:
  - No space between a method name and the parenthesis "(" starting its parameter list
  - Open brace "{" appears at the end of the same line as the declaration statement
  - Methods are separated by a blank line

Statements

if-else Statements

```
if (condition) {
   statements;
}
if (condition) {
   statements;
} else {
   statements;
}
if (condition) {
   statements;
} else if (condition) {
   statements;
} else if (condition) {
   statements;
}
```

- ***Note**: if statements always use braces {}.*

for Statements

- A for statement should have the following form:

```
for (initialization; condition; update) {
  statements;
}
```

- An empty for statement should have the following form:

```
for (initialization; condition; update);
```

while Statements

- A while statement should have the following form:

```
while (condition) {
  statements;
}
```

- An empty while statement should have the following form:

```
while (condition);
```

do-while 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 switch statement should include a default case.

try-catch Statements

- A try-catch statement should have the following form:

```
try {
  statements;
} catch (ExceptionClass e) {
  statements;
  m_logger.debug(error message);

}
```

<u>Note:</u> *Should m_logger instead of e*

White Space

Blank Lines

- 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
  - Before a block

Blank Spaces

- A keyword followed by a parenthesis should be separated by a space.
  - a blank space should not be used between a method name and its opening parenthesis.
- A blank space should appear after commas in argument lists.
- All binary operators except “.”, “++”, “—” should be separated from their operands by spaces.
- Casts should be followed by a blank. Example:

```
myMethod((byte) aNum, (Object) x);
myFunc((int) (cp + 5), ((int) (i + 3)) + 1);
```

Naming Conventions

Class/Interface

- Class/Interface names should be nouns, in mixed case with the first letter of each internal word capitalized.
- Use whole words—avoid acronyms and abbreviations

Method

- Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

Variables

- Should have the m_ prefixes for fields and static fields.
- A lower-case first letter. Internal words start with capital letters.
- Variable names should be short yet meaningful.
- 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.

Constants

- Should be all uppercase with words separated by under scores (“_”).

Programming Practice

Static Variable & Method

- Avoid using an object to access a class (static) variable or method. Use a class name instead.
- For example:

```
classMethod();              //OK
AClass.classMethod();       //OK
anObject.classMethod();     //AVOID!
```

Variable Assignments

- Avoid assigning several variables to the same value in a single statement.
- Do not use embedded assignments in an attempt to improve run-time performance. Example:

```
d = (a = b + c) + r;
```

Returning Values

- Should avoid

```
if (condition) {
    return x;
}
return y;
```

- Should be written as

```
return (condition ? x : y);
```

Compare a String with a Constant

- *Should use* ***constant.equals(ref)*** instead of ***ref.equals(constant)***.
- Generally, we should use this way to compare a Constant with an Object that support ***equals()*** method.
- Should use ***String.valueOf(ref)*** instead of ***ref.toString()***

Check Null

- Always ***check null*** an instance before invoking its methods. And ***check null*** ***first*** in conditions
- Example:

```
if (a != null && a.isEmpty()) {
  // implement
}
```

Put/Get a Value of Map

- This is a good idea from a. Truc, when caching data from server:

```
Value valueA = map.get(keyA);
if (valueA == null) {
    valueA = new A();
    map.put(keyA, valueA);
}
return valueA;
```

In AxS Project

Issues – Who care of updating

- AxS Naming Convention for project, package, class (Thuan)
- AxS JUnit (Dung)
- AxS performance: always thinking about performance when using any services from server. (Buu)
- Always avoid duplicating codes.(Lan)
- Concurrent problem.(Khai)
- Cache resources. (Tam)
- Sort. (Khanh)

References

- http://www.javapractices.com/home/HomeAction.do