OurGuidelines

General Developing Guidelines

SVN Guidelines

(1) Binary files (e.g. *.class, *.jar) must not be checked in to the repository. If in some place binary files are to be created during the development process they should be ignored using the svn:ignore property.

Wrong:

```
$ javac HelloWorld.java
$ svn status
> ?   HelloWorld.class
$ svn add *
$ svn commit -m “Compiled HelloWorld and checked in the binary.”
> ...
```

Right:

```
$ svn propset svn:ignore “*.class”
$ javac HelloWorld.java
$ svn status
# won’t report anything since *.class is ignored
```

(2) Any file should contain information on the creator (the author who created the resource) and, unless otherwise specified, no one but the creator should author in that file. If another person wants to edit a foreign file she should ask one of the current authors and become an author herself.

(3) A commit is an atomic action, i.e. every thing you do should be commited as a single commit.

Wrong:

```
$ vim MyFile
...
$ vim andNowForSomethingCompletelyDifferent
...
$ svn commit -m "Done a lot of things."
```

Right:

```
$ vim MyFile
$ svn commit MyFile -m "Implemented XY"
$ vim andNowForSomethingCompletelyDifferent
$ svn commit andNowForSomethingCompletelyDifferent -m "Implemented Z"
```

Java Syntax & Naming Conventions

Name it what it is and what it does!

A TEACHER OF MINE

Indentation

(1) Use tabs, not spaces.

Note: You can adjust the length of a tab individually in most text editors.

Wrong (tabs are denoted by –, spaces by •):

```
class X {
••••public int member = 3;
}
```

Right:

```
class X {
–     public int member = 3;
}
```

(2) A case label should line up with its switch statement. The case statement is indented. (WebKit[1], Indentation, 4)

Wrong:

```
switch (size) {
      case 1:
            return "okay";
}
```

Right:

```
switch (size) {
case 1:
      return "okay";
}
```

[1] WebKit Coding Style Guidelines (http://WebKit.org/coding/coding-style.html)

(3) When breaking an expression into multiple lines, the next line should be indented to the same level as the previous line using tabs, and than by spaces to reach the current context.

Note: This is since tabs may have different widths on different platforms and are not granular enough to reach the desired point in any case without using spaces (as can be seen in the example below)

Wrong (tabs are denoted by –, spaces by •):

```
class X {
-     public static void doSomething(int one, int two,
-     -     -     -     -     -     int three, int four) {
-     -     // ...
-     }
}
```

Right:

```
class X {
-     public static void doSomething(int one, int two,
-     •••••••••••••••••••••••••••••••int three, int four) {
-     -     // ...
-     }
}
```

Spacing

(1) Do not place spaces around unary operators[2].

Wrong:

```
for (int i = 0; i < x; i ++) {
      // ...
}
```

Right:

```
for (int i = 0; i < x; i++) {
      // ...
}
```

(2) Do place spaces around binary and ternary operators[3].

Wrong:

```
x += (3+4)/7*8;
z=3 *5;
System.out.println(Math.random(7*4+5));
```

Right:

```
x += (3 + 4) / 7 * 8:
z = 3 * 5;
System.out.println(Math.random(7 * 4 + 5));
```

(3) Do not place spaces before comma and semicolon[4]

Wrong:

```
for (int i = 0 ; i < max; i++) {
      callSomeFunc(i / max , i);
}
```

Right:

```
(int i = 0; i < max; i++) {
      callSomeFunc(i / max, i);
}
```

[2] (WebKit, Spacing, 1)

[3] (WebKit, Spacing, 2)

[4] (WebKit, Spacing, 3)

(4) Place spaces between control statements and their parentheses[5]

Wrong:

```
if(false) { }
```

Right:

```
if (false) { }
```

(5) Do not place spaces between a function and its parentheses, or between a parenthesis and its content[6]

Wrong:

```
Math.round (3.5);
```

Right:

```
Math.round(3.5);
```

[5] (WebKit, Spacing, 4)

[6] (WebKit, Spacing, 5)

Line breaking

(1) Each statement should get its own line[7].

Wrong:

```
i++; j++;
```

Right:

```
i++;
j++;
```

(2) An else statement should go on the same line as a preceding close brace[8].

Wrong:

```
if (isValid) {
      // ...
}
else {
      // ...
}
```

Right:

```
if (isValid) {
      // ...
} else {
      // ...
}
```

[7] (WebKit, Line breaking, 1)

[8] (WebKit, Line breaking, 2)

Braces

(1) Anywhere: place the open brace on the line preceding the code block; place the close brace on its own line[9].

Wrong:

```
class X
{
      void doIt(boolean isValid)
      {
            if (isValid)
            {
                  System.out.println("I'm doing it!");
            }
      }
}
```

Right:

```
class X {
      void doIt(boolean isValid) {
            if (isValid) {
                  System.out.println("I'm doing it!");
            }
      }
}
```

(2) One-line control clauses should always use braces even if they could be omitted.

Wrong:

```
if (isToBeDone) doIt();
```

Right:

```
if (isToBeDone) {
      doIt();
}
```

(3) Control clauses without a body should use empty braces (WebKit, Braces, 4)

Wrong:

```
for (int i = n; i != 1; i = (i % 2 == 0) ? i / 2 : 3 * n + 1);
```

Right:

```
for (int i = n; i != 1; i = (i % 2 == 0) ? i / 2 : 3 * n + 1) { }
```

[9] (WebKit, Braces, 2.)

Naming

(1) Precede boolean values with words like "is" and "did"[10].

Wrong:

```
boolean valid = false;
boolean dataSent = true;
```

Right:

```
boolean isValid = false;
boolean didSendData = true;
```

(2) Precede setters with the word "set". Use bare words for getters. Setter and getter names should match the names of the variables being set/gotten[11].

Wrong:

```
class X {
	private String value;

	public String myString() {
		// ...
	}
}
```

Right:

```
class X {
	private String value;

	public String getValue() {
		// ...
	}
}
```

[10] (WebKit, Names, 5)

[11] (WebKit, Names, 6)

(3) Use descriptive verbs in function names[12].

Wrong:

```
interface X {
      void toASCII(String string);
}
```

Right:

```
interface X {
      void convertToASCII(String string);
}
```

(4) Constants are members declared final static and should be named using upper case letters and words are separated by underscores.

Wrong:

```
public class X {
      public final static pi = 3.14;
      final static gravitationalConstant = 6.678;
```

Right:

```
public class X {
      public final static PI = 3.14;
      final static GRAVITATIONAL_CONSTANT = 6.678;
}
```

(5) The prefix for our packages is "de.fu" in accordance to the naming conventions for java packages according to Oracle Sun[13]

(6) 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).

Wrong:

```
class arrange { }
```

Right:

```
class Arrangement { }
```

---

[12] (WebKit, Names, 7)

[13] Code Conventions for the Java TM Programming Language: 9 – Naming Conventions (http://www.oracle.com/technetwork/java/codeconventions-135099.html#367)

(7) Interface names should be capitalized like class names.

Wrong:

```
interface toilett { }
```

Right:

```
interface Toilett { }
```

(8) Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. No underscores.

Wrong:

```
interface X {
      Run(); // starting letter upper case
      emptied(); // not a verb
      the_background() // underscore, not a verb
}
```

Right:

```
interface X {
      run();
      isEmpty();
      getBackground();
}
```

(9) Variable names are in mixed case starting with a lower case letter.

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.

Programming practices[14]

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

Wrong:

```
anObject.classMethod();
```

Right:

```
classMethod();
AClass.classMethod();
```

(2) Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.

(3: Parenthesis) It is generally a good idea to use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems. Even if the operator precedence seems clear to you, it might not be to others.

Wrong:

```
if (a == b && c == d)
```

Right:

```
if ((a == b) && (c == d))
```

(4) If the sole use of an if-statement is to return true or false, return the boolean expression.

Wrong:

```
if (a || b) {
      return true;
} else {
      return false;
}
```

Right:

```
return a || n;
```

[14] Code Conventions for the Java TM Programming Language: 10 – Programming practices (http://www.oracle.com/technetwork/java/codeconventions-137265.html#529)

(5) If an expression containing a binary operator appears before the ? in the ternary ?: operator, it should be parenthesized. Example:

Wrong:

```
x >= 0 ? x : -x;
```

Right:

```
(x >= 0) ? x : -x;
```

(6: Special Comments) Use `XXX` in a comment to flag something that is bogus but works. Use `FIXME` to flag something that is bogus and broken. Use `TODO` to flag something that is missing.