Java Stylesheet

Alex Vondrak

Last Revised: January 11, 2012

1 Java Features

Integrated Development Environments (IDEs) are often excessive with their suggested uses of *annotations* like `@Override` or `@SuppressWarnings`, which are a special form of documentation that can be added to Java code. Annotations don't buy us much at this level, especially if we insert them just because the IDE told us to. For lack of usefulness in projects as small as those in CS 240, **don't use annotations**.

Additionally, you may have learned about Java's documentation system called *Javadoc*. If the source code includes specially-formatted comments, Javadoc generates HTML web pages like those seen at `http://docs.oracle.com/javase/7/docs/api/`. Again, though it's very useful, CS 240 projects aren't so complex that they need elaborate documentation. Thus, **do not use Javadoc**.

This is not to say you shouldn't comment your code. But even writing comments is an acquired skill: too many comments—especially trivial ones—don't really help the code's readability. As a rule of thumb, good comments document the *why*, not the *how*. Code is the most literal way to describe what's happening, so if the *how* isn't clear, you should rewrite your code until it is. If the code is clear, additional comments are unnecessary, except to clarify *why* the code's a certain way. Don't be that programmer who makes useless comments like `x += 1; // increment x by 1.`

2 Conventions

2.1 Structure

1. Lines should be a maximum of 80 characters long. You may need to put a linebreak in between long expressions to adhere to this rule.
2. Sort **`import`** statements lexicographically and place them at the top of the file.
3. Using the `.*` form of **`import`** should be avoided.
4. Using a static member import should be avoided (i.e., the **`import static`** statement).
5. Don't import
   - a class you don't use.
   - a class more than once.
   - a class from the `java.lang` package (this is redundant).

- a class from the `sun` package[1].

6. Always label instance variables, methods, and constructors with the proper modifiers. If more than one modifier applies, declare them in the following order:

   1) **`public`**
   2) **`protected`**
   3) **`private`**
   4) **`abstract`**
   5) **`static`**
   6) **`final`**

7. The parts of a **`class`** or **`interface`** declaration should appear in the following order:

   1) Class (**`static`**) variables. First the **`public`** class variables, then the **`protected`**, then the **`private`**.
   2) Instance variables (fields). First the **`public`** fields, then the **`protected`**, then the **`private`**.
   3) Constructors
   4) Methods
   5) Inner (nested) classes

8. In general, instance variables should be **`private`** and have accessor methods ("getters" and/or "setters").

9. In general, non-**`final`** instance variables should be initialized in constructors, *not* in their declarations. **`final`** variables should be initialized in their declarations, though.

2.2 Blocks

10. The bodies following **`do`**, **`else`**, **`if`**, **`for`**, and **`while`** constructs require the use of curly braces, even if they are only for a single statement.

11. Avoid empty blocks. Every block should have at least one statement.

12. Avoid nested

    - blocks.
    - **`for`** loops.
    - **`if`** statements.
    - **`try`** statements.

13. If the left brace will fit on the first line of the statement, then the brace must be at the end of the line. Otherwise the brace must be on a new line. For example:

[1] See `http://www.oracle.com/technetwork/java/faq-sun-packages-142232.html`.

GOOD

```
if (condition) {
```

```
if (condition1 && condition2 && condition3 && condition4
    && condition5)
{
```

BAD

```
if (condition)
   {
```

```
if (condition1 && condition2 && condition3 && condition4
    && condition5) {
```

14. The right brace of a block must be alone on a line. For example:

GOOD

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

BAD

```
if (condition) {
   // ...
} else { /* ... */ }
```

2.3 Whitespace

15. Indent each level[2] of code by **3 spaces**.

16. Do not use the TAB key to indent lines. There should be no TAB characters in your file. If there are, they'll be assumed to represent 3 spaces.

17. Classes should be separated by two blank lines.

18. Methods in a class should be separated by one blank line.

19. If a line is too long to fit an expression involving multiple infix operators, the operator should go on a new line and be indented to the proper level. For example:

[2]Basically, a new level is introduced by each block.

GOOD

```
someVariable = aBigVariableNameThatIsQuiteBig + "blah blah"
               + someOtherVariable;
```

BAD

```
someVariable = aBigVariableNameThatIsQuiteBig + "blah blah" +
               someOtherVariable;
```

20. There should be no space after a left parenthesis or before a right parenthesis. For example:

GOOD

```
object.method(1,␣2,␣"buckle",␣"my",␣"shoe");
```

BAD

```
object.method(␣1,␣2,␣"buckle",␣"my",␣"shoe"␣);
```

This includes typecasts. For example:

GOOD	**BAD**
`(int) x;`	`( int ) x;`

21. There should be no space between the identifier of a method definition, constructor definition, method call, or constructor invocation and its corresponding left parenthesis. For example:

GOOD

```
public static void main(String[] args) {
   methodCall();
   Constructor x = new Constructor(1, 2, 3);
}
```

BAD

```
public static void main (String[] args) {
   methodCall       ();
   Constructor x = new Constructor
   (1, 2, 3);
}
```

22. There **should** be whitespace after:

- a comma

- a semicolon
- a typecast (so `(int)␣x`, not `(int)x`)
- a keyword (**`catch`**, **`do`**, **`else`**, **`finally`**, **`for`**, **`if`**, **`return`**, **`try`**, **`while`**)

23. There **should not** be whitespace after:

    - a unary operator (`~`, `!`, prefix `--`, prefix `++`, unary `-`, unary `+`)
    - the left brace of an array literal (so **`int`**`[]␣{1,␣2,␣3}`, not **`int`**`[]␣{␣1,␣2,␣3␣}`)

24. There **should** be whitespace before a left curly brace. For example:

GOOD

```
public␣static␣void␣main(String[]␣args)␣{
```

BAD

```
public␣static␣void␣main␣(String[]␣args){
```

25. There **should not** be whitespace before:

    - a semicolon
    - the postfix `--` and `++` operators

26. There **should** be whitespace surrounding both sides of:

    - binary operators (`&`, `&&`, `|`, `||`, `>`, `>=`, `<`, `<=`, `==`, `!=`, `<<`, `>>`, `>>>`, `^`, `/`, `-`, `+`, `*`, `%`)
    - assignment operators (`=`, `&=`, `|=`, `<<=`, `>>=`, `>>>=`, `^=`, `/=`, `-=`, `*=`, `+=`, `%=`)
    - the parts of the ternary operator (`?`, `:`)

27. There **should not** be whitespace surrounding either side of:

    - a dot (`.`)
    - the angle-brackets of a generic type declaration (so `List<Integer>␣xs`, not `List␣<␣Integer␣>␣xs`)

2.4 Naming

28. Generic type parameters should use single uppercase letters (`A`–`Z`).

29. **`final`** variables should use `UNDERSCORE_CAPS`.

30. Non-**`final`** variables and method names should use `lowerCamelCase`.

31. **`class`** and **`interface`** names should use `UpperCamelCase`.

32. Variable names should reflect their purpose. As a rule of thumb, class names should be nouns and method names should be verbs/verb-phrases.

3 Readability

33. **long** constants should use an uppercase `L`, since a lowercase `l` looks too much like the number `1` (so `100L`, not `100l`).

34. Do not use the variable name `l` (a lowercase `L`), because it is very easily confused with the number `1` (one).

35. Array brackets should come after the type, not the variable (so `String[] args`, not `String args[]`).

36. Don't rely on the **this.** default for field names and methods; instead, spell it out in full (so **this.**`x`, not just `x`).

37. Except for constructors and setter methods, do not name a method parameter the same thing as a class field. That is, do not "hide" a field with a parameter. For example:

```
class Foo {
   private int x;

   // OKAY
   public Foo(int x) {
      this.x = x;
   }

   // OKAY
   public void setX(int x) {
      this.x = x;
   }

   // NOT OKAY
   public int normalMethod(int x) {
      return x + this.x;
   }
}
```

38. Each variable declaration should be on its own line.

39. Each statement should be on its own line.

40. **switch** statements should always have a **default** clause, and the **default** should come last.

41. Avoid unnecessary parentheses.

42. In general, avoid using the ternary operator.

43. Avoid empty statements (i.e., standalone `;`s).

44. Avoid inner assignments, wherein there's an assignment inside of a subexpression. For example:

GOOD

```
i = 2;
String s = Integer.toString(i);
```

BAD

```
String s = Integer.toString(i = 2);
```

45. Do not leave in "to do" comments or other comments automatically generated by your IDE.