Java Programming Style Guide

A guide to clean, consistent and well written Java Code

Dr. Martin O'Connor
www.computing.dcu.ie/~moconnor

Table of Contents

Introduction

- The Java language allows you to write code that would be very difficult for others to understand
- Java also allows you to write code that is very easy to understand
- Let’s write Java code that is very easy to understand! ☺
- This style guide provides a set of directions to empower you to write code that will be consistent, easier to read, easier to maintain and less prone to errors

Indentation

- **Indentation**
  - All indenting is done with spaces, not tabs.
  - All indents are two or four spaces. Pick either two or four and stick with it – important to be consistent!
  - *Reasoning: All programs work well with spaces. Most programs will mix tabs and spaces so that some lines are indented with spaces and some with tabs. If your tabbing is set to 4 and you share a file with someone that has tabbing set to 8, everything comes out goofy.*

Indentation (Continued)

How to Indent

– Matching braces should always line up vertically in the same column as their construct.

```
public class HelloWorld
{
....public void greetUser(int currentHour)
....{
........System.out.print("Good");
........if (currentHour < AFTERNOON)
........{
............System.out.println(" Morning");
........}
........else if (currentHour < EVENING)
........{
............System.out.println( "Afternoon");
........}
........else
........{
............System.out.println("Evening");
........}
....}
}

Note: The period char (.) is used to show indentation
```

Indentation (Continued)

How to Indent

- All if, while and for statements must use braces even if they control just one statement.

```
if (superHero == theTick) System.out.println("Spoon!");  // NO!

if (superHero == theTick)
    System.out.println("Spoon!");  // NO!

if (superHero == theTick) {
    System.out.println("Spoon!");
}                                                  // NO!

if (superHero == theTick)
{
    System.out.println("Spoon!");
}                                                  // YES!
```

Spacing

Spacing

- All method names should be immediately followed by a left parenthesis.

```
foo (i, j); // NO!
foo(i, j);  // YES!
```

- All array dereferences should be immediately followed by a left square bracket.

```
args [0];  // NO!
args[0];   // YES!
```

- Binary operators should have a space on either side.

```
a=b+c;           // NO!
a = b+c;         // NO!
a=b + c;         // NO!
a = b + c;       // YES!

z = 2*x + 3*y;           // NO!
z = 2 * x + 3 * y;       // YES!
z = (2 * x) + (3 * y); // YES!
```

Spacing (Continued)

Spacing

- Unary operators should be immediately preceded or followed by their operand.

```
count ++; // NO!
count++;  // YES!

i --;     // NO!
i--;      // YES!
```

- Commas and semicolons are always followed by whitespace.

```
for (int i = 0;i < 10;i++)    // NO!
for (int i = 0; i < 10; i++) // YES!

getPancakes(syrupQuantity,butterQuantity);  // NO!
getPancakes(syrupQuantity, butterQuantity); // YES!
```

Spacing (Continued)

Spacing

- All casts should be written with no spaces

```
(MyClass) v.get(3);   // NO!
( MyClass )v.get(3); // NO!
(MyClass)v.get(3);    // YES!
```

- The keywords if, while, for, switch, and catch must be followed by a space

```
if(hungry)   // NO!
if (hungry) // YES!

while(pancakes < 7)   // NO!
while (pancakes < 7) // YES!

for(int i = 0; i < 10; i++)   // NO!
for (int i = 0; i < 10; i++) // YES!

catch(TooManyPancakesException e)   // NO!
catch (TooManyPancakesException e) // YES!
```

Class Member Ordering

Class Member Ordering

```
class Order
{
    // fields (attributes)

    // constructors

    // methods
}
```

Maximum Line Length

Maximum Line Length

- Avoid making lines longer than 120 characters. If your code starts to get indented way to the right, consider breaking your code into more methods.

- *Reasoning: Editors and printing facilities used by most programmers can easily handle 120 characters. Longer lines can be frustrating to work with.*

Use of Parentheses

Parentheses

- Parentheses should be used in expressions not only to specify order of precedence, but also to help simplify the expression. When in doubt, parenthesize.

Identifier Naming Convention

Identifiers

- All identifiers use letters ('A' through 'Z' and 'a' through 'z') and numbers ('0' through '9') only. No underscores, dollar signs or non-ascii characters

1 - Classes and Interfaces

– All class and interface identifiers will use mixed case. The first letter of each word in the name will be uppercase, including the first letter of the name. All other letters will be in lowercase, except in the case of an acronym, which will be all upper case.

Examples:

Customer
SalesOrder
TargetURL
URLTarget

Identifier Naming Convention (Continued)

Identifiers

2 - Packages

– Package names will use lower case characters only. Try to keep the length under eight (8) characters. Multi-word package names should be avoided.

Examples:

common
core
lang

Identifier Naming Convention (Continued)

Identifiers

3 - All Other Identifiers

– All other identifiers, including (but not limited to) fields, local variables, constants, methods and parameters, will use the following naming convention

The first word in the name will always be in lowercase. The first letter of the second and subsequent words in the name will always be in uppercase. All other letters in the name will be in lowercase, except in the case of an embedded acronym, which will be all uppercase. Leading acronyms are always in lowercase.

Examples:

customer
salesOrder
addToTotal()
targetURL
urlTarget

Identifier Naming Convention (Continued)

Identifiers

4 – Naming Convention – Variable Names

- Use full English descriptions for names. **Avoid using abbreviations**. For example, use names like firstName, lastName, and middleInitial rather than the shorter versions fName, lName, and mi.

- **Avoid overly long names** (greater than 15 characters). For example, setTheLengthField should be shortened to setLength.

- Avoid names that are very similar or differ only in case. For example, avoid using the names product, products, and Products in the same program for fear of mixing them up.

Identifier Naming Convention (Continued)

Identifiers

4 – Naming Convention - Rationale

- You should write your code with the aim of making it understandable to others and yourself. Others will need to read and understand your code and one of the major keys to understanding is through the use of meaningful identifier names.

- By using meaningful names, you go a long way towards writing **self-documenting code**.

Identifier Naming Convention (Continued)

Identifiers

4 – Naming Convention - self-documenting code.

– That is, code that is understandable on its own without requiring accompanying comments.

```
double tax1;        // sales tax rate (example of poor variable name)
double tax2;        // income tax rate (example of poor variable name)

double salesTaxRate;       //no comments required due to
double incomeTaxRate;      //self-documenting variable names
```

Identifier Naming Convention (Continued)

Identifiers

4 –Constant Naming Convention

- Use **ALL_UPPER_CASE** for your named constants, separating words with the underscore character. For example, use TAX_RATE rather than taxRate or TAXRATE.

- **Avoid using magic numbers** in the code. Magic numbers are actual numbers like 3.142 that appear in the code that require the reader to figure out what 3.142 is being used for. Consider using named constants for any number other than 0 and 1.

Identifier Naming Convention (Continued)

Identifiers

4 –Constant Naming Convention - Rationale

- By using all upper-case for your named constants, others reading your code will immediately know that the identifier is a fixed, constant value that cannot be changed.

- Using meaningful names for constants instead of using magic numbers in your code makes the code **self-documenting**

```
day = (3 + numberOfDays) % 7;    //NO! uses magic numbers

final int WEDNESDAY = 3;
final int DAYS_IN_WEEK = 7;

day = (WEDNESDAY + numberOfDays) % DAYS_IN_WEEK;  //Yes, self-documenting
```

Method Naming Convention

Method Naming Convention

- Try to come up with **meaningful method names** that succinctly describe the purpose of the method, making your code self-documenting and reducing the need for additional comments.

- Compose method names using **mixed case letters**, beginning with a lower case letter and starting each subsequent word with an upper case letter.
  - applyBrakes(), getBalance(), printBanner(), calcTaxRate()

Method Naming Convention (Continued)

Method Naming Convention - Continued

- **Begin method names with a strong action verb**. If the verb is not descriptive enough by itself, include a noun (for example, addInterest). Add adjectives if necessary to clarify the noun (for example, convertToEuroDollars).
- Use the prefixes get and set for ***getter*** and ***setter*** methods
- Getter methods merely return the value of a instance variable; setter methods change the value of a instance variable
  - For example, use the method names getBalance and setBalance to access and change the instance variable balance

Method Naming Convention (Continued)

Method Naming Convention - Continued

- If the method **returns a boolean value**, use *is* or *has* as the prefix for the method name
- For example, use isOverdrawn or hasCreditLeft for methods that return true or false values
- Avoid the use of the word ***not*** in the boolean method name, use the ! operator instead
- For example, use !isOverdrawn instead of isNotOverdrawn

Commenting Convention

Commenting Conventions

- Comments provide yourself and other programmers with the information helpful to understanding your program
- Use comments to provide overviews or summaries of chunks of code (logical block of code) and to provide additional information that is not readily available in the code itself
- Comment the details of nontrivial or non obvious design decisions
- Avoid comments that merely duplicate information that is present in and clear from reading the code