Code Conventions

This handout is under development.

Introduction

English writers follow conventions of the English language. (Okay, not all writers follow all the convensions, but most follow most conventions.) These conventions dictate not only word meaning and order, but also punctuation. And following conventions makes text more readable. Think of how hard it is to read things that are missing spaces, have inconsistent indentation, or shove extra punctuation in the middle of lines. Similarly, programmers also have conventions to follow. Some come from the language (e.g., the meanings of braces), but many come from the local programming community. Conventions dictate indentation, the way loops are formatted, the way and place variables are declared, and so on and so forth.

I think it's difficult to read badly formatted code. I know it's difficult to read code that's formatted using multiple styles. I've observed that almost every FOSS community has coding conventions for their projects - how loops are formatted, how variables are declared, and so on and so forth. And I hear from our alums that most of the companies they work for require that their programmers follow local coding conventions when working on a project. Hence, *I require that students in this course follow coding conventions*.

My favorite set of code conventions is the GNU C Source Code Formatting Standards, which I use in all of my C coding projects (not least because I work on GIMP source code, which follows those conventions). But these have not been extended for Java.

There are a widely accepted set of Java coding conventions, Code Conventions for the Java Programming Language, published by the now-defunct Sun Microsystems, the original designers of the Java programming language. I find these conventions misguided, at best. I attempted to use them my first semester teaching this course and found them even more painful than I'd thought. It appears that Oracle, the new "owners" of Java have also found the doucment outdated.

What's the solution? I have a set of conventions that I like, which essentially extend the GNU standards in natural ways to Java. However, I have not been able to convince Eclipse to format code using these conventions. Hence, I will also provide some alternative conventions, along with an Eclipse formatter that helps you achieve those conventions.

How serious am I that I expect you to use these conventions? Serious enough that *I will impose a penalty on any work that has more than four violations of the these conventions*. I will not apply this policy to code written during class time (e.g., during an in-class examination).

I also require you to comment your end braces. In my too many years of programming and helping students debug programs, I've found that knowing what a brace is supposed to end is incredibly helpful (particularly since we often forget or add extra end braces).

Guiding Principles

Three basic principles undergird these guidelines.

First, code should be easy to trace visually. That is, it should be easy for the reader to match braces, to understand nesting, to find annotations, and so on and so forth.

Second, text should be easy to "process" with a text editor or text tools. For example, we often want to find procedure names. In the GNU C conventions, we know that function names start in the leftmost column, so we can just look for `^name`. It's not quite so easy in Java, since all functions are nested within classes, and so should be indented. But, if we follow similar guidelines, we should be able able to search for `'^ name'`.

Third, we follow GNU conventions as closely as possible because those conventions have been thoughtfully designed (and because I'm used to using them).

Ideally, those three principles would guide all of the design. Unfortunately, it is not possible to completely achieve the GNU conventions with any configuration of the Eclipse formatter. (In particular, the designers of that formatter don't seem to believe that function names should begin on their own line.) Hence, I will provide both ideal and alternate formatting for some conventions.

Conventions

Indentation. Indent to indicate nesting, particularly when nesting within braces. By default, indent by two spaces. Do not use tabs, because different environments may display tabs differently.

Braces. Align matching braces vertically so that the reader can quickly identify matching braces. Comment all end braces to clarify what the brace ends.

Spacing. Put a space before the open parenthesis in a function declaration or function call. Put spaces after keywords like `if`, `for`, and `while`. In general, follow the GNU conventions.

Line Width. Lines should be no more than 79 characters wide. Limiting lines to that width makes it easier to print your code, to view it in a typical terminal window, and to edit it in that terminal window. (In each case, you are likely to get unfortunate automatic wrapping if you use more than 79 characters.)

Alignment with Line Wrapping. Manually wrap your lines so that they do not exceed the specified line width. I prefer that you align the next line so that it matches the corresponding value on the previous line. However, in cases in which there is no corresponding value, or when such wrapping puts too much whitespace at the beginning of the line, you may indent by four spaces. You may also use the four-space indent as your default, since that's easier to get Eclipse to enforce.

Documentation. Use Javadoc documentation for every class, method, and field declaration.

Following Code Conventions in Eclipse

Eclipse lets let you set the code conventions to use, and will even reformat/reindent your code once for you. You can find the code conventions for this class in `/home/rebelsky/share/CSC207.xml`.

An Example

Forthcoming.

References

- Sun/Oracle Java Code Conventions
- GNU C Source Code Formatting Standards
- Google Java Style