

# CS32 Style Guidelines

Computer Science 32, Summer 2009  
Steven P. Reiss  
John Jannotti

## 1 Summation

Writing a program is like writing a book. Style is as important as substance. You are writing not just for yourself, but for others – making things readable is essential.

For the most part you are expected to follow the guidelines in “The Elements of Java Style”. You should read this book several times during the course of the semester. Deviations other than the ones listed below should be avoided and may cause you to receive a lower grade on the assignment.

While this is in no way an extensive list (you still must read the book!), the staff (i.e. the people grading your projects) would appreciate that you pay particular attention to the following rules.

- 8** If you’re going to use hard tabs, ensure that your tab stops are unix standard (every 8). Similarly, you should keep the length of your lines of code below 96 characters, though most should be shorter.
- 9, 19-21, 23, 24** Use meaningful names. The reader should be able to look at a class/method/variable and immediately have a general idea of its basic properties, functions, dreams, and aspirations. Though keep in mind rule **11** about excessively long names. Even “throw-away” variables described in rule **28** should be reasonably named.
- 56** In order for your code to be even slightly useful, people must know what it does and how to use it (i.e. knowing the input and output of your method without needing to know the actual inner workings). This documentation standard ensures that a reader will know what a function is supposed to do, regardless of whether it actually does.
- 57** It certainly won’t hurt your grade if you can provide specific information about a defect in your project, saving your frustrated/groggy/overworked grader time and effort by pointing him/her in the right direction. This will also help you keep track of what problems need to be fixed, which is exceedingly helpful when working in groups.

## 2 Edits

- 5** The book recommends indents of 2 spaces. Indenting 3 spaces is also ok; but more generally you should avoid complex code that requires lots of indentation.
- 15** As you are a student of this class and Brown University, the root for all your package names should be `edu.brown.cs.cs032`. You might want to use symbolic links to avoid typing com-

plex path names. For example, if you're working package is `edu.brown.cs.cs032.projectX`, your project directory might resemble

```
projectX
  edu
    brown
      cs ...
src => symbolic link to edu/brown/cs/cs032/projectX
```

- 25** We prefer differentiating between local variables in a function (including parameters) and fields and differentiating both from method names.

Hence, we suggest that all local variables be all lower case. Field names should contain an underscore character (`the_game`; `x_value`; ...), while parameters and local variables should not. While it is ok to have locals start or end with an underscore, it makes the program harder to read (and type) – making the name include two words is preferable.

An exception to this rule occurs with unused parameters. We like to document a method parameter that is not used inside method by putting an underscore in front of its name. This tells the reader right away that the parameter is not used in the method, useful information to know when reading the code.

- 30** Since the changes to **25** already describe the naming conventions for parameters, rule **30** is irrelevant.

- 39** Provide full javadoc for all public and protected methods of public classes and interfaces. Provide documentation of some sort for all other methods. Full javadoc is not required for trivial private methods (e.g. local accessor functions).

The public and protected methods of public classes of a class represent the interface that other people will be using if they make use of your class. Other people generally prefer to read documentation instead of code and hence having complete and accurate documentation (produced by javadoc) is a necessity for such methods when working in teams or when producing libraries or other reusable packages.

Package local classes and the private and package protected methods of public represent the private interface. Programmers interested in using these methods are generally going to be reading the code. It is important that these classes and methods be documented, but it is not essential that javadoc be used. Whatever form of documentation is used, it should be as complete as javadoc would be. My primary reason for not using javadoc in these cases is that javadoc looks good when outside the code, but is not the most readable documentation in its source form. Making the code readable is as important as using the tool.

- 43-44** Use html in comments only if you are going to produce javadoc for the code; it makes the actual comment harder to read.

- 73-74** You can use Java enums to avoid creating extra classes here.

**89-92** Use the `assert` statement in java. Always run your programs with the `-ea` (`-enableassertions`) and `-esa` (`-enablesystemassertions`) options.

### 3 Additions

**A** Use inner classes for any nontrivial helper class of a class rather than defining a new file. This keeps the support classes private and close to the implementation where they are used, making the code easier to read and simpler since there are fewer files. Generally, a class with under 50 lines of code is acceptable as an inner class, and an inner class of more than 100 lines should be made a separate class.

**B** Provide a `<Package>Constants` interface for each package. This interface should define all constants that are global to the package (i.e. that are used in more than one file or that control the behavior of the package in some way or other – e.g. a pathname). This interface can also contain very simple support classes or interfaces (e.g. enumerations). All classes (or interfaces) in the package should implement (or extend) this interface.

**C** Avoid anonymous classes. Use named inner classes instead. Anonymous classes are very hard to debug and can have unexpected semantics.

**D** Put an end-line comment on the line containing the close brace of a class or interface definition. This holds for both the main class in a file and any inner classes. For example,

```
    } // end of class Struggle          or          } // end of inner class Ism
```

**E** Don't put more than one top-level class in a file.

**F** Make sure your name is in the source documentation, either in an `@author` tag or separately (or both). If more than one person works on a given file, both names should be listed. It might be good to get into the habit of adding a copyright statement to the file with your name attached.