University of North Carolina at Greensboro
CSC 330: Advanced Data Structures
Prof. Stephen R. Tate

Handout 3
August 30, 2016

CSC 330 Java Coding Style Rules

These rules are based on the original standard *Code Conventions for the Java Programming Language*, produced by Sun (the original creators of Java) and available at

`http://www.oracle.com/technetwork/java/codeconvtoc-136057.html`

While these code conventions are no longer actively maintained, they form the basis of most professional style guidelines for Java, such as Google's Java Style Guide. You are expected to follow these style guidelines for all programming assignments in this class, although you will not lose points for following the style from either the CSC 230 book (see Appendix A) or the CSC 330 book (no style guidelines are explicitly stated in the book, but are implied through their code examples).

1. Required comments at the top of every file include: your name, assignment number, a brief (one line) description of what is in the file, the last date modified, and a statement that you followed the UNCG Academic Integrity Policy on the assignment. This is slightly different from the *Code Conventions*, because we're in a class setting.

2. *All* data elements in a class should be private, or possibly of package or protected scope if there is a compelling reason for this.

3. Proper indentation is an absolute must! Consistently use 2-4 characters for each indentation level — the exact number of spaces is up to your personal taste, but be consistent!

4. All lines should be at most 80 characters long.

5. Use whitespace and blank lines to increase readability.

6. Use meaningful but concise names for classes, methods, and variables. Class names should be nouns, begin with a capital letter, and each word of a multi-word name should start with a capital letter (Examples: `Tree`, `BinaryTree`, and `BinarySearchTree`). Method names should be verbs, start with a lower case letter, and start subsequent words with upper case letters (Examples: `setData()`, `isEmpty()`, and `getNext()`). Variables in a class should start with a lower case letter and capitalize subsequent words (like method names), but should be nouns

(Examples: `root`, `rightChild`, and `vertexList`). Local variables in a function should also have meaningful names, with an exception being for simple loop variables (`i` is ok for an integer loop, `x` for a variable in numeric code, etc.).

7. Should consistently order class definitions as follows: static variables, instance variables, constructors, and finally the methods (grouped by functionality).

8. Use modular, multi-file construction. Each source file should contain only *one* public class or interface (although a file may include private classes or nested classes in support of the main public class should be in the same file).

9. Provide constructors to make sure objects are initialized properly.

10. Variables should be declared with as narrow a scope as possible, including declaring loop variables in a for statement. If a variable is only used within a single block, it should be declared within that block. *Never* declare a variable outside of a method unless it is truly part of the object or class state (i.e., don't use an instance variable as a lazy way to pass values from one method to another). Local variables should be declared as close to their first use a possible, and should always be initialized.

11. Use parentheses for grouping expressions, especially precedence is not obvious.

12. Use braces around every block of code, including single-line blocks after an "if" statement or loop construct. I prefer "Egyptian style" brace placement, which puts open braces at the end of lines rather than on a line by itself (this is fairly universal in professional style guides, but almost every introductory programming book does it differently).

13. Put comments before each function or method, and at the beginning of each source file. Your comments should in "javadoc style," which allows a description of your class to be extracted and formatted in HTML. Make sure you clearly state the purpose of each method, what parameters the method expects, and the meaning of the return value (if any). Rule of thumb: I should have all the information I need to call a function and interpret the results by looking only at the function comments (I shouldn't have to look at the code at all).

14. Comment unclear parts of the code, although your goal should be to make code as clear as possible so that it is "self-documenting" and does not need additional explanation. Comments are very helpful with variable declarations, if there is information that is not clear from the variable name (how it's used, assumptions about legal values, etc.).

15. Don't improvise — if there's no rule stated, but all code you see is written a particular way, imitate that and don't make up your own non-standard style.