CS 1103 - AY2019-T3: Style Guide

0. A program should be readable. All the other rules on this page can be overridden by this primary rule. Your programs should display good taste, which you can generally acquire only by paying attention to the practices of other, expert programmers.

Comments
1. Every class (except for anonymous inner classes) should have a Javadoc comment that specifies the purpose of the class and describes its public interface in general terms. Except for nested classes, the Javadoc comment should include an @author tag that lists your name.

2. Every method, unless it is declared private, should have a Javadoc comment explaining what it does. The comment should include a statement of any conditions that must hold when the method is called, such as restrictions on the acceptable values of its parameters. (These are called the "preconditions" of the method.) The purpose of each parameter should be clearly documented. If there is a return value, its meaning should be documented. If the method can throw any exceptions, it's a good idea to document those as well. You are encouraged to use @param, @return, and @throws tags for the relevant documentation.

3. Every variable that has a non-trivial role in the program should have a comment that explains its purpose. This includes both member variables and local variables. For non-private member variables, the comments should be in Javadoc format. For-loop variables and other local utility variables do not, in general, have to be commented.

4. Comments can be included in the body of a method when they are needed to explain the logic of the code. In general, well-written code needs few comments.

5. Comments should never be used to explain the Java language. A comment such as "declare an int variable named x" or "increment the variable ct" is worse than useless. Assume that your reader knows Java! (Note that such comments are sometimes used in programming textbooks or on the blackboard, but never in real programs.)

Formatting
6. Use indentation to display the structure of your program. The body of a class definition should be indented. The body of a method definition should be indented. When a statement is nested inside another statement, it should be indented one additional level. (Note that Eclipse can fix the indentation of any segment of code: Just hilite the code segment and hit Control-I.)

7. An ending "}" should be on a line by itself. The opening "{" can be at the end of a line, or it can be placed on a line by itself so that it lines up with the matching "}". I prefer the first of these two options, but whichever one you use, be consistent.

8. Don't put more that one statement on a line.

9. Avoid very long lines. In general, lines should not be longer than 80 characters. Longer statements should be broken up over several lines.

10. Avoid very deep nesting of statements. If you find yourself using more than two or three levels of nesting, think about defining some subroutines in order to break up your code into more manageable chunks.

11. Blank spaces and blank lines can make a program easier to read. In general, you should put some blank lines between method definitions. Leave spaces around operators such as =, ==, !=, and so on.

Naming
12. Use meaningful names for your variables, methods, and classes.

13. Use a consistent style of capitalization. It is strongly recommended that you follow the usual Java convention: Variable, method, and package names begin with lower case letters. Class names begin with upper case letters. When a name contains more than one word, capitalize the extra words, as in "interestRate".

14. Use "final static" to declare named constants to represent constant data. Consider using "enum" to create several related constants. Constants generally have names that are entirely in upper case, with words separated by underscore characters.

Methods
15. A method should have a clear, single, identifiable task.

16. An individual method definition should not be too long. In general, a function should not be longer than one printed page (and that is stretching it).

17. Instance methods can access the instance variables that represent the state of an object, but you should avoid using instance variables simply to pass information from one method to another -- for that, you should use parameters and return values.

Classes
18. A class should represent a clear, single, identifiable concept.

19. Use the modifiers public, protected, and private to control access to the variables and methods in a class.

20. Member variables should, in general, be declared to be private. Getter and setter methods can be provided to access and manipulate the private member variables.