CS 51 Lab Style Guide

The way you present your program is just as important as having a correct program. While having good comments or good variable names will not affect the correctness of your program, it will make it easier to read your program. It is important that others be able to understand your code so that they can modify your code if they have to. Half of your lab grade will be determined by your programming style.

This is a guide to help you better understand what we are looking for when we look at your labs. As the semester progresses there will be steeper penalties for styling mistakes. So, get into the habit of writing good labs from the beginning. After writing your lab be sure to look over your code and your style.

1 Commenting

Comments are extremely important. Take some care to write accurate yet concise comments.

- Be specific with your comments. What happens when the mouse is pressed? What is the `FilledRect` used for? Your comments should describe the intent of the code. What are you trying to accomplish with this piece of code?
- Do not make your comments too long. Any line of comment or code that is over 80 characters will not print. If it does not print, the TA will not be able to read it. Consider revising or writing it on multiple lines.
- *Do not overcomment!* Overcommenting can make the program hard to read just as much as under-commenting. Do not comment every line. If they are simple instructions, most people will understand them without your comments. If you think you need commenting, try commenting chunks of code under the same comment.
- Delete any extraneous code that is not used. You would not hand in an English paper with crossed out lines. Similarly, you should not hand in a lab with commented out code. Remove all code that you comment out.

You should write comments for:

Class (Program): At the top of each class, you should write your name, date of creation, and a brief description of what the class is for. If you decide to do more than the assignment requested, you should describe these "extra" features in the program under the class description. Sometimes it's hard to tell a bug from a feature.

Methods: Above each method heading there should be a brief description of what the method does. You should also describe what each parameter means and what the return result means. If the method code is simple you do not need to comment in the method body. If the method body is complicated, you might want to add some comments to certain areas that you deem to be complicated. Be careful not to overcomment!

Variables and constants: In general, variables and constants should all have comments as to what the variable is used for. For example a good comment for your variable `Line diagonal` in `NoClicking` would be: `// the slash in the prohibit sign`. Occasionally several closely related variables or constants can be grouped together under the same comment.

2 Blank Lines

Blank lines are used to delineate different areas of the code. The instance variable declarations at the top of your program should be separated from the header of the class and the header of the first method. There should always be space between methods. It is advisable to break up long method bodies and long declarations into logical pieces. Always start a new line after the semicolon. Always leave a blank line before a comment line.

3 Names

You should always choose names that suggest the meanings of the things being named. If the purpose of a method is to draw a rectangle, then a good name for that method is `drawRect`. If there is a variable of type `FilledRect` that is used as the stem of a stop sign, then a good name would be `stem` or `stemRect`.

- Names that have nothing to do with the program are very bad names! Ex. `FramedRect frodo`
- Names that are not specific or descriptive enough are generally bad names. Ex. `Line line` or `Line l` are not as good as `Line foulLine` if the line represents a foul line.
- Try not to name objects sequentially. You should only do this when the objects are related in some way that is not otherwise expressible. Ex., `FramedRect box1`, `FramedRect box2`, `FramedRect box3` are not as good as `darksBasket`, `whitesBasket`, and `colorsBasket` if the rectangles represent laundry baskets for clothes of specific colors.

By convention, constants (indicated by "`static final`" in Java) are all capital letters, classes begin with uppercase, variable and method names begin with lowercase, while uppercase is used to start new words in multi-word names.

Instance variables should (almost) never be declared to be public. Instance variables should only be used when a value must be saved for use after a constructor or method has been completed. Otherwise local variables should be used.

4 Format

Your program should be organized as neatly as possible. Always select Eclipse's format command from the Source menu frequently to keep your program readable. Be sure to select format to reformat your program before turning it in.