A few notes: for the first two assignments, penalties for style are reduced by half (and some penalties will just get a warning). This is to give you a little time to get used to the language and its syntax, and to learn the style we want you to have. Also, this is not a comprehensive list. You may lose points for a style offense not explicitly listed here; this is just a guideline to try to help you. If you have specific questions about style, feel free to write to the staff list or see us in office hours.

Major offenses

- **Improper indentation.** You can use either tabs or spaces, as long as you’re consistent. If you’re using spaces, use 4 spaces per tab. This is a big offense because Eclipse will automatically indent your code for you (if you select some text and hit Ctrl-i). Do check over your code after you auto-indent with Eclipse; sometimes it can make small mistakes.

- **Print statements.** Do not leave debugging or other print statements in your code. This makes our grading scripts unhappy. A simple Ctrl-f will show you where all their print statements are, so this is a major offense too.

- **Submitting the correct files in the correct way.** This isn’t exactly style, but submitting the wrong things in your zip file – like zipping a whole folder or Eclipse project instead of just the Java file(s), or submitting our support code in addition to your own code – is a major pain for us. It, like extra print statements, makes the grading scripts unhappy. We won’t take off any points for this for the first two assignments, but after that, you’ve had more than enough time to figure out how to submit your homework properly.

- **Non-compiling code.** Again, not quite style, but if your code doesn’t compile, you will receive a penalty for it. How big that penalty is depends on how severe your mistake was and if we can fix it, but this happening at all shows us that you didn’t even run your code before you submitted it.

Medium offenses

- **Redundant code.** This includes repeated (i.e. copy/pasted) code for things that should be in helper methods and code like:

  ```java
  // wrong!
  public boolean isPositive (int x) { 
      if (x >= 0) return true;
      else return false;
  }
  ```

  which should be

  ```java
  // right!
  public boolean isPositive (int x) { 
      return x >= 0;
  }
  ```
Other examples include:

    // wrong!
    while (boolean_var == true) { ... }

instead of

    // right!
    while (boolean_var) { ... }

Note that testing something like `boolean_var == true` is bad practice in Java. In other languages, like Ruby, you may need to use `==` to check for equality, but doing so in Java is both unnecessary and poor style.

Another example of redundant code is:

    // wrong!
    if (x == 0) do_something();
    else if (x == 1) do_something();

instead of

    // right!
    if (x == 0 || x == 1) do_something();

- **Redundant comments.** Redundant comments are things like

    // bad!
    if (x > 0) return; // if x is positive, return

or

    // bad!
    x++; // increment x

Comments should explain details high-level algorithm being implemented, e.g.

    // good!
    // Binary search, returns -1 if element not found
    public int binarySearch (int[] arr, int x) { ... }

the general idea of a tricky/complicated/non-obvious bit of code, or descriptions beyond what the code already says, e.g.

    // good!
    // greedy search terminates when no improvement can be made
    if (best <= current) return;

Comments should not explain literally what the code is doing – we can read the code already.
• **Redundant testing.** Redundant testing is something like

    // bad!
    while (front != null) {
        if (front != null) do_something();
        ...
    }

Similarly, don’t surround lots of code by a statement meant for error checking; for example, don’t surround the code of an entire method with a `if (curr != null)`. Instead, you should check `if (curr == null)` in the beginning and handle that case there. This makes your code much cleaner and easier to read.

• **Curly braces.** Namely, having a closing brace anywhere but on its own in a new line (with the exception of do-while loops) is very bad style. Additionally, make sure you’re consistent with using or not using curly braces in one-line loops/conditionals, or putting the opening curly brace on the same line or on a new line for a loop/method/class/etc.

• **Leaving in commented-out code.** Whether it’s tester code or code that didn’t work or whatnot, this looks horrible and serves no purpose. The only exception is if it is one block of cleanly-written testing code (like inside a `main()` method, or in its own helper method(s)), but you will lose points if it’s just e.g. miscellaneous print statements scattered around.

**Minor offenses**

• **Bad variable names.** This includes one letter variable names outside of for loops and completely nonsensical variable names like `hello` or `thing`.

• **Lines longer than 80 characters.** We’re very flexible with this one. A couple of lines that are 85 characters are alright. Long method signatures are OK, and starter code that we give you that’s over 80 characters are okay. But lines that are 110+ characters long aren’t. Note that this includes comments!

• **Improper access level modifiers.** For example, except in very particular circumstances, helper methods and global variables should be private.

• **Leaving in the // remove this line when you complete this method comments.** This is just silly!