CSC 207 Java Style Guidelines

1 Naming

- Follow Oracle's naming convention guidelines:
  `http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html#367`

2 Javadoc and Comments

- Write a good Javadoc comment for each class, and within each class for all public members: variables, constructors, and methods. In your constructor and method comments, include `param` and `return` tags when appropriate. Note that Eclipse generates the tags for you.
- Your method comments must mention the purpose of each parameter, and must be grammatically correct.
- Javadoc is not required for private members, but commenting them is advisable. TAs marking your code will deduct marks if they are left confused about a member's purpose.
- Use 3rd person rather than 2nd person (e.g. "Gets the label" rather than "Get the label").
- Begin method descriptions with a verb phrase (e.g. "Gets the label" rather than "This method gets the label").
- Begin class, interface, and field descriptions by simply stating what the object is (e.g. "A button label" rather than "This field is a label").
- Use "this" rather than "the" when when referring to an object on which a method is invoked.
- Put comments above or next to what they are commenting on.
- Put a blank line before every comment that appears on a line by itself.

3 Spacing

- Each line must be less than 80 characters long *including tabs and spaces*. Many editors tell you what column you're in. Look in the bottom of the window, typically.

  Beware of "soft returns" — some word processors, like WordPad, wrap lines automatically. If you use such a program, make sure that you press the return key yourself.
- Use a tab width of 4, if you use tabs at all. The best way to make sure your program will be formatted correctly is never to mix spaces and tabs – use only tabs, or only spaces. If you use a tab width of less than 4, it is your responsibility to make sure that your lines are shorter than 80 characters when we look at your program, because we will use a tab width of 4. Most editors let you specify that tabs should be replaced by spaces, and also let you specify the number of spaces used.
- Put a blank space before and after every operator. For example, the first line below is good but the second line is not:

```
boolean b = 3 > x && 4 - 5 < 32;
boolean b = 3>x && 4-5<32;
```

  Note, however, that we do not recommend or require spaces before or after the '<' and '>' in class names with type parameters (using generics). For example, this is fine:

```
List<String> sList = new ArrayList<String>();
```

- When breaking up a long line, break it *before an operator*, not after. The first example below is good but the second is not:

```
boolean b = "Hello".charAt(3) + 3 > x
&& Integer.parseInt(s) < 32;

boolean b = "Hello".charAt(3) + 3 > x &&
Integer.parseInt(s) < 32;
```

- When breaking up a long line, break it *after a comma*, not before. The first example below is good but the second is not:

```
String[] letters = ["a", "b", "c",
"d", "e"];

String[] letters = ["a", "b", "c"
,"d", "e"];
```

4 Booleans

- Never compare a `boolean` expression to `true` or `false`. For example, if you find yourself writing code like this:

```
if (a.equals(b) == false) {
  ...
}
```

  replace it with code like this:

```
if (!a.equals(b)) {
  ...
}
```

- Similarly for assignments:

```
if (a < b) {
  ack = false;
} else {
  ack = true;
}
```

  Replace it with code like this:

```
ack = !(a < b);
```