CS106/CS206 Code Formatting Standards and Guidelines

Disclaimer

Note that the code we create during class and in the notes is for illustrative purposes – it often will not adhere to these guidelines.

Naming Conventions

- Use meaningful names! For example, if your program needs a variable to represent the radius of a circle, call it radius, *not* r and *not* rad.
- Use single letter variables for simple loop indices and occasionally generic integers.
- The use of very obvious, common, meaningful abbreviations is permitted. For example, "number" can be abbreviated as "num" as in numStudents.
- Variable, instance variables, and method names in Java generally are written in camelCase, starting with a lower-case letter and putting the first letter of subsequent words in uppercase.
- Class names are written in PascalCase, starting with a capital letter.
- Constants (static final) are written in ALL_CAPS.

Whitespace

The most-readable programs are written with prudent use of whitespace (including both blank lines and spaces).

- Use blank lines to separate major parts of a source file or method. These are like paragraph breaks in English writing.
- After every {, indent by at least 2 spaces until the matching }. A good editor like emacs will help you with indentation - remember to keep pressing those tabs!
- Separate an operator from its operands by spaces.
- There should never be a need for 2 blank lines in a row or two spaces in a row.

Line Length

All lines should have length at most 100 characters. This makes it easier to read code on a potentially small screen. If you need to break a line in order to satisfy this, the rest of the line should be indented more than its current block. In cases where the line length is within a pair of parentheses, one good option is to indent the rest of the line to match with the opening parenthesis. For example:

```
if (here is a long condition for an if statement and
    the rest of the condition) {
  here is a long statement inside the if statement and
    the rest of the statement below it;
}
```

File header comments

Every source code file should contain a header comment that describes the contents of the file and other pertinent information. It must include the following information (different order is fine):

- A description of the contents of the file
- Your name
- Date
- The file name (optional)

For example:

```
/**
 * The main driver program for Assignment 1.
 *
 * This program reads the file specified by the first command line
 * argument, counts the number of words, spaces, and characters and
 * displays the results in the format specified in the project
 * description.
 *
 * @author: Dianna Xu
 * @version: February 7, 2020
 */
```

Variable comments

All instance variables must be commented (it is okay to use a single comment to refer to more than one instance variable though). Most local variables should be commented, too.

Method comments

All methods must be commented. The comments should explain what the method does, what its parameters are (not their types, we already know that), and what it returns. You should use the javadoc method comment style, which lists and comments all parameters and return values, shown below:

```
/**
 * Returns the sum of two integers
 * @param x The first integer
 * @param y The second integer
 * @return  The sum of x+y
 */
public static int sum(int x, int y) {
    ...
}
```

Imports

Eclipse will organize your imports for you, but they should not include the "*" wildcard. Each class should be imported separately to avoid confusion with the classes you're creating.

```
import java.util.ArrayList;
import java.util.Scanner;
```

Instead of

~~`import java.until.*;`~~

In-Line Comments

You should strive for your code to be self-explanatory. However, it is inevitable that some lines of code are more intricate. In these cases, a comment describing the code is well-advised. The comment should *not* simply translate the code to English, but should explain what's really going on. For example:

```
// Unhelpful comment:
starSides = 5; // set starSides to 5

// Helpful comment:
starSides = 5; // reset starSides to original value
```

Well-structured code will be broken into logical sections that each perform a simple task. Each of these sections of code (typically starting with an if statement or a loop) should be documented.

An in-line comment too long to appear to the right of your code appears above the code to which it applies and is indented to the same level as the code. For example:

```
// increment all the odd values in the array
for (int i = 0; i < n; i++) {
    // add 1 only to the odd values
    if (array[i] % 2 == 1) {
        array[i] = array[i] + 1;
    }
}
```

Remember, good comments tell us things we don't already know, or can't easily decipher among dense code blocks.

Indentation Styles

Choose one of the two styles and use it consistently (note how the braces are placed):

```
if (condition) {
    ...
} else if (condition) {
    ...
} else {
    ...
}
```

```
if (condition)
{
    ...
}
else if (condition)
{
    ...
}
else
{
    ...
}
```

```
for (control expressions) {
    ...
}
```

```
for (control expressions)
{
    ...
}
```

```
while (condition) {
   ...
}
```

```
while (condition)
{
    ...
}
```

```
do {
   ...
} while (condition);
```

```
do
{
   ...
} while (condition);
```

```
switch (variable) {
   case constant1:   ...
        break;
   case constant2:   ...
        break;
   case default:     ...
}
```

```
switch (variable)
{
    case constant1:   ...
        break;
    case constant2:   ...
        break;
    case default:     ...
}
```