CECS 277 – Coding Standards

Coding Standards – are a set of guidelines used when writing software. There are many different standards and different ones will be used at different software companies, on different languages, and even different projects. Standardizations are created so that everyone working on a single project writes the code the same way. This makes the code uniform, which makes it easier to read, understand, and maintain.

The following is in no way a complete list, but when in doubt, try to make your code as readable and understandable as possible.

Naming – Classes, methods, variables, constants, all need to be named. It is often tedious to come up with different names for everything, but please don't resort to giving variables names like num1 and var2, it is a bad practice to fall into, and can make your code difficult to understand. Your variables represent something, so give them appropriate names that describe the data they are holding.

- Variable names may consist of upper and lower case letters, dollar signs ($), underscores (_), and digits 0-9, but cannot begin with a digit, and cannot contain spaces.
- Class, method, and variable names should all be descriptive. You should be able to come back to your code months later and still be able to understand it.
- Try to keep them brief, two or three words (more if the meaning still isn't clear). Use comments to describe anything that might still be unclear.
- Use camel case or underscores to separate words in a variable name.
    - Ex. employeeIdNumber or employee_id_number vs. empidnum
- Variable and method names should begin with a lower case.
- Class names should begin with an upper case.
- Constants should be in all upper case with underscores separating words.
- Create boolean variable names so that they help you decipher how the value is used: (ex. `isValid` tells you that it is valid when the value is true, where `test` is ambiguous).
- Declare and initialize your variables just before you are going to use them.
- Abbreviations are sometimes permissible if the meanings are universally known, but otherwise spell them out.
- Constants & Magic Numbers – avoid using mysterious values in your calculations. Instead, create constants to store the values to make it easier to modify and debug your program later.

```
double pay = rate * 40; //what is 40?
```

-vs-

```
final int HOURS_IN_WORKWEEK = 40;
double pay = rate * HOURS_IN_WORKWEEK;
```

    - Mathematical formulas are usually ok. Just state in a comment what formula you are using so that it can be easily verified.

```
double volume = (4.0 / 3) * PI * r * r * r;
```

Spacing – makes your code easier to read. Densely packed code saves nothing and often hinders readability.

- Space out your code as you would paragraphs. Add line breaks around sections of code that go together. Comment these sections appropriately and even consider making it into a method, especially if it can be reused.
- Nested code should be inset by 4 columns. Use either tabs or spaces consistently.
- Use spaces to separate keywords, variables, operators, parenthesis, etc.
  - Ex. `if( x < 10 )` vs. `if(x<10)`
- A statement should be on its own line. Don't put multiple statements on one line.
- If a statement is too long to fit on a single line, break it up into multiple lines and indent appropriately.
- Use Eclipse's auto code formatter. Select the code, and hit Ctrl-Shift-F.

Curly Braces { } – are used to define areas of code, such as: classes, methods, and control structures (ifs and loops).

- Curly braces line up vertically, and the code contained inside is indented.

```
public class ProgramName
{
    public static void main( String [] args )
    {
         for( int i = 0; i < 10; i++ )
         {
              if( i % 3 == 0 )
              {
                   System.out.println( i );
              }
         }
    }
}
```

- If you prefer to have the opening brace at the end of the line, then line up the closing brace with the beginning of the first line. Whichever way you choose, use it consistently throughout your program.

```
if( test == 3 ) {
     System.out.println( test );
}
```

- Even though most control structures allow single line statements without needing curly braces, it is usually preferred to put them in anyway. It is too easy to forget to add them if additional statements are added later on.

Methods – a block of code that can be called throughout your program. If you find yourself copying and pasting parts of your code, you might be able to make it a method.

- Method names should be descriptive of what it does and begin with a lower case.
- All methods should be documented with the following in Javadoc style:
  - A brief description of what the method does.
  - A brief description for each parameter
  - A brief description for its return value.
  - A description of each exception it might throw.

Comments – use comments whenever necessary to aid in the understanding of your program. Having good variable and method names is a good place to start in making your code understandable, but it doesn't make up for having good comments.

- If you are using a formula, a set of logic, or a block of code, describe what it is doing or what it is for.
- Avoid useless comments and over commenting code.
    - Don't comment each line of code. Instead, comment sections of code.
    - Don't write a comment saying that you are declaring a variable, instead describe what the variable will be used for.
    - Be brief, but understandable.
- Use /* */ for multi-line comments and // for single line comments.
- Comments should go above the code you are describing, not below or to the right.
- Javadocs should be used in addition to normal commenting.

Javadocs – is a tool built into the JDK that can be used to generate Java Documentation pages in HTML format. When your code is correctly tagged and commented, the Javadoc tool creates easy to read web pages that list all of the information about each of your java classes and their methods. Using Javadocs also allows you to view that information when you hover over a variable or method name elsewhere in the code. This keeps you from having to scroll through your code to find the variable or method you need to know about.

- Create the method header first (some IDE's auto populate the Javadoc block).
- Use the /** */ comment tag to create a Javadoc comment block.
- Use the @param tag to document each method parameter.
    - @param parameterName description of parameter
    - You do not need to identify the parameter's data type. It already knows it from the method's parameter list. Just describe what the parameter is for.
- Use the @return tag to document the value the method returns.
    - You do not need to identify the data type of the return value.
    - You do not need to give the return value a name, just describe what the function will return.
- Use a @throws tag to describe each type of exception thrown from the method.
    - Create additional @throws for each exception.

```
/**
 * Method description.
 * @param x description of the first parameter
 * @param y description of the second parameter
 * @return description of the value returned
 * @throws Exception description of when the exception occurs
 */
public double methodX( int x, int y ) throws Exception
{
   //description of code logic
   return x * y;
}
```