As documentation is a large portion of your graded programming assignments, please read and apply this information about identifier names for classes, methods, class constants, class wide variables (fields), method parameter names, local variables, and documentation standards using multi-line, single line, and javadoc documentation.

Always include the following in the order listed (refer to pages 4-5 for a simple complete example program):

- ✓ You will use multiline comments at the top of each file which includes the course, the lecturer, the assignment name, the due date, and any other relevant information.
- ✓ All import statements follow the multiline comments (no need to write comments on import statements)
- ✓ Javadoc Class comment immediately follows import statements and precedes the Class heading(details below)
- ✓ Javadoc comment listed just prior to each Class wide Constants(details below)
- ✓ Javadoc comment listed just prior to each Class wide variable or "Instance Field"(details below)
- ✓ Javadoc comment just prior to ALL method headings. This documentation should include information about each parameter listed on separate lines AND information about the return value (details below)

- ➢ Continue to use single or multiline comments within you code when necessary for clarity (Javadoc should NOT appear in the body of a method)
- ➢ All class constants should be in all uppercase letters (use underscore to separate words), e.g. TOTAL_STUDENTS
- ➢ All class instance fields should begin with the word "my" followed by camelCase, e.g. myInterestRate
- ➢ All method parameters should begin with the word "the" followed by camelCase, e.g. thePurchasePrice (more detail on identifier naming conventions will be discussed during weeks 1 and 2 lectures)
- ➢ No line should exceed 72-76 characters in length.

Details On Comments and Javadoc:
We will use Javadoc tags in the Javadoc comments. There are several but, we will only use the following:

Order of Tags
Include tags in the following order:

- `@author` (classes and interfaces only, required)
- `@version` (classes and interfaces only, required)
- `@param` (methods and constructors only)
- `@return` (methods only)

Each described below:

Each file begins with:

```
/*
 * The file name of your program, e.g. Assign7.java
 *
 * TCSS 142 – Summer 2013
 * Assignment 7
 */
```

Next are your import statements

Next is your Javadoc comments that describe your program immediately before the class heading (NOTE the /**):

```
/**
 *  This program ...(describe the purpose of your program here)
 *
 *  @author Your Name
 *  @version A date or a version number
 */
public class MyClassName {
```

continued . . .

Class Constants:

If you have class constants (first listed immediately following the class header), include a Javadoc comment for each:

```
/**
 *  Explain the purpose of the constant here...
 */
  public static final int MY_CONSTANT_NAME;  // Assign the value inside the constructor or here.
```

Class Instance Variables (Class Fields):

If you have class variables (instance fields) each is preceded with a javadoc description of the variable followed by the variable declaration. Class variable names should begin with the word "my" with access of private (unless specified otherwise):

```
/**
 * Indicates if the customer has a store membership.
 */
private boolean myMembership;
```

You will not use class fields in the first two programming assignments. Your first use of class fields will begin with programming assignment 3.

Javadoc for each method:

- ✓ Include a Javadoc comment at the start of each method to explain the purpose of the method.
- ✓ Include @param (one per line per each parameter in the order they are listed in your method heading each followed by the parameter name and a brief description of the parameter purpose-NOTE: parameter names should begin with the word "the" followed by camelCase for the remaining words)
- ✓ Include @return tags if the method returns a data type or object

```
/**
 *  Explain the method's purpose. (Prompts for...,           Calculates..., etc)
 *
 *  @param theFirstParameterName (Explain the purpose of this parameter here)
 *  @param theSecondParameterName (Explain the purpose of this parameter here)
*    etc.
 *  @return (Explain what the method returns here)
 */
public static <returnType> methodName(<parameterType> <parameterName>  etc.) {
```

Other rules of documentation:

- ✓ All identifier names (class name, class fields, method names, local variables) should be meaningful
- ✓ Local variables (declared within a method) should use meaningful names and camelCase if multiple words. Local variables do not need any special prefix, i.e. **do not** begin with "my" or "the."
- ✓ blocks of code (method bodies, loops, selection structures, etc.) should be consistently indented
- ✓ Always use a space:
  - ☺ between operands and operators

    - ☺ following keywords (while, if, for, etc.)
    - ☺ after a comma
    - ☺ before a left brace (if on the same line as a method, selection, or loop header)
    - ☺ after the two semi-colons in a for loop header, e.g. `for (int num = 0; num < total; num++) {`
- ✓ When writing an executable class (a class that contains a main method), the main method should be the first method listed in the class
- ✓ When writing an executable class, do not use class wide fields, i.e. all variables should be declared inside methods and passed to other methods that may need access to them

Continued . . .
At first sight, this may seem daunting but, once you have created your first file that includes Javadoc, you can save a "skeleton" version of your code (primarily just the Javadoc followed by headings or constant declaration only, i.e. no other code). This skeleton file can be used to copy and paste all relevant Javadoc into other programs you write. You would then simply edit necessary specifics such as parameter names, file names, etc.

Once you have included Javadoc into your files and compile them, you can run Javadoc to create all the Javadoc files.

Everything you need to know about javadoc but were afraid to ask is here
http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

Later this quarter a portion of a lab session will be devoted to javadoc and its advantages. In this Lab you will be given existing files that include everything discussed here (to save time) and will modify only the necessary information, such as your name.

See the following two pages for an example program . . .

```java
/*
 * TCSS 143 - Spring 2017
 * Instructor:  David Schuessler
 * Documentation of code demo
 */

import java.util.Scanner;

/**
 * Generates a class average on a series of student test grades.
 *
 * @author David Schuessler dschues1@uw.edu
 * @version 31 March 2017
 */
public class Documentation_Demo {

  // constants (static final fields)
  /** A default value for the number of students. */
  public static final int TOTAL_STUDENTS = 10;

  /**
   * Driver method of the class average processing.
   *
   * @param theArgs is used for command line input.
   */
  public static void main(String[] theArgs) {
    Scanner console = new Scanner(System.in);   // Scanner-keyboard input.

    // The following is an array - to be discussed formally in week 2.
    int[] testScores = new int[TOTAL_STUDENTS]; // Self-documenting name.
    double classAverage = 0.0;                  // Same as above which means
                                                // no need for these comments.
    int totalPassed = 0;

    getTestScores(testScores, console);
    classAverage = getTestAverage(testScores);
    totalPassed = getTotalPassed(testScores);
    displayResults(testScores, classAverage, totalPassed);
  }

  /**
   * Reads all the test scores from the console.
   *
   * @param theTestScores is the array to hold all the scores.
   * @param theConsole is a Scanner to read from the keyboard.
   */
  public static void getTestScores(int[] theTestScores, Scanner theConsole) {

    for (int testNumber = 1; testNumber <= TOTAL_STUDENTS; testNumber++) {
      System.out.printf("Enter test Score #%2d: ", testNumber);
      theTestScores[testNumber - 1] = theConsole.nextInt();
    }
    System.out.println();
  }



// Continued next page ---->
```

```java
  /**
   * Generates and returns the average of the test scores.
   *
   * @param theTestScores is the array of test scores.
   * @return the average of all the test scores.
   */
  public static double getTestAverage(int[] theTestScores) {
    double sum = 0.0;

    for (int testIndex = 0; testIndex < TOTAL_STUDENTS; testIndex++) {
      sum += theTestScores[testIndex];
    }
    return sum / TOTAL_STUDENTS;
  }

  /**
   * Counts and returns the total of passing scores. Instead of using the preferred
   * for loop to process an array, this method does the same using a while loop.
   *
   * @param theTestScores is the array of test scores.
   * @return a count of how many scores were above 75%
   */
  public static int getTotalPassed(int[] theTestScores) {
    int totalPassingSum = 0;
                                              // 3 sections of for loop header:
    int testIndex = 0;                        //   1. Initialize
    while (testIndex < TOTAL_STUDENTS) {      //   2. Test condition
      if (theTestScores[testIndex] >= 75) {
        totalPassingSum++;
      }
     testIndex++;                             //   3. Increment
    }
    return totalPassingSum;
  }

  /**
   * Displays the results (class average and number or passing scores).
   *
   * @param theTestScores is the array of test scores.
   * @param theClassAverage contains the average of all scores.
   * @param theTotalPassed contains how many scores were >= 75.
   */
  public static void displayResults(int[] theTestScores,
                                    double theClassAverage,
                                    int theTotalPassed) {
    String output = "The class average = " + theClassAverage + "%\n" +
                    "The total Passing = " + theTotalPassed + "\n" +
                    "All the test scores are:\n";

    // Concatenate all test scores to output, one per line.
    for (int testIndex = 0; testIndex < TOTAL_STUDENTS; testIndex++) {
      output += String.format("\t Test #%2d:%4d\n",
                              testIndex + 1,
                              theTestScores[testIndex]);
    }

    // Display the results to the console.
    System.out.println(output);
  }
}
```