CSC 115: Fundamentals of Programming II

Coding Conventions

Revision: January, 2016 (L. Jackson & B. Bultena), January, 2015 (M. Zastre)

Please apply the following guidelines to all your Java source-code files. You will have the best success if you apply these style guidelines *as you code*. Be sure to be consistent. When making changes, deletions and additions, ensure the style is maintained.

Document Comments

The top of every submitted assignment must identify its owner (name and identification number), the date, the title and details of the file. This is done through the use of a comment at the top of the submitted file. See the following example:

```
/*
 * Name:  <your names goes here>
 * ID:    <your identification number goes here>
 * Date:  <date of last modification goes here>
 * Filename:  <Classname>.java
 * Details: CSC115 Assignment <number>
*/
```

In addition, every method requires information that clearly indicates its purpose, a list of its parameters and a description for each, and a description of what is returned. The public method commenting is part of the *specifications* of the method; to be used by developers who need not be concerned about the implementation details. The programmer only references the private method commenting. However, it is good practice to use similar formatting for both public and private methods.

Method header comments are placed on the line above the appropriate method. See the following example, which uses the *javadoc* style, detailed in Oracle's How to write Doc Comments page, to describe a method that sorts and array of integers and then returns the k^{th} element in the sorted array.

```
/**
 * Sorts an array of integers and then returns the
 * element in index position k.
 * @param array The array of integers.
 * @param k The position of the integer to be returned
 * in the sorted array.  Note that array indices
 * are from 0 to array.length-1 inclusive.
 * @return The value in the newly sorted array at
 * index k.
 * @throws ArrayIndexOutOfBoundsException if k is
 * out of bounds.
 */
```

You may elect to use regular block commenting for the method headers, in which case, you can substitute the initial /** for /* and change the @tags to a plain word followed by a colon. The beauty of using javadoc comments over block comments for method headers is that the javadoc tool can then create professional webpage documentation.

Implementation Comments

The programmer who is actually writing or modifying the source code reads implementation comments. The purpose of these is to add some clarity to the implementation or purpose of the source code statements. Implementation comments can either be *block comments* or *single-line comments*.

Block comments provide a brief description of data structures or algorithms. They should be preceded by a blank line to set it apart from the rest of the code, but not the code it is describing. For example:

```
<blank line>
/*
 * Here is a block comment that describes
 * the lines of code that follow this comment.
 */
<begin the first statement of the lines of code..>
```

Single-line comments are very short comments that take one line or can follow a short java statement on the same line: The following is a single-line comment:

```
/* this comment takes a single line */
```

The following is a single-line comment that shares the line with a java statement:

```
} /* end of while loop */
```

In the above cases for single-line comments, you can substitute the alternative, which consists of two forward slashes '//' followed by the text that makes up the actual comment. Because this comment style is recognized within block comments, it is preferred over the
/* . . . */ single-line comment, which cannot be contained within a block comment.

Identifier names

- Names for most variables should be descriptive, with the one exception of loop variables such as `i`, `j`, or `k` (or `temp` for temporary, `x` and `y` for coordinates).
- A convention for the name of Java variables consisting of several words together is to use capitalization for the second and successive words. For example, idiomatic Java would write `listHead` rather than `list head`.
- There is an exception for the naming of constants (i.e., `static final`) as the convention is to use all uppercase letters for the name. In this case we would use underscores. For example: `public static final MAX_LENGTH = 10;`
- Class and interface names must begin with a capital. For example: `BoxCar`, `Calculator`, `List`

- Package identifiers are lower case. For example: `package linkedlist;`

Coding Style

Although ***whitespace*** aids the eyes to read code, too much whitespace can reduce readability. The following situations indicate a specific use of whitespace:

- Add single blank lines to offset sections of code. For example, variable declarations are often set apart. Other locations suitable for single blank lines are: before the first line of a class or method definition, and after the closing brace for each; when separating methods as well as separating major code blocks within functions.
- Insert spaces to assist left-right scanning of syntax. For example, spaces around the assignment operator, relational logical operators, and other binary operators help highlight the appearance of an important symbol.
- Use parentheses in expressions in order to both ensure proper precedence and also guide the reader to an understanding of the expression's overall structure.

A good general practice is to have ***one statement per line***. However, introducing line breaks and then suitably indenting the rest of the line does help when reading the code left to right. Imagine what the code will look like on a standard size sheet of paper. If there is line wrapping, then manually create separate lines to fit on the page. The standard number of characters on a page is about 80 characters.

Curly braces also have their own convention:

- ***Opening braces*** (i.e., "`{`") appear at the end of the code line opening the block (for example, at the end of the line containing the if, else, switch, for, while, class, etc.). An opening brace on a line by itself is also acceptable. Each ***closing brace*** (i.e., "`}`") goes on a line by itself. Ensure it is vertically aligned with the start of the line opening the block.
- Braces used to create a list of initial values may share the same line in order to save vertical space. This is usually seen with shorter array initialization declarations.
- The text of an `else` statement may be put on the same line as the closing brace of its `if` statement. (See the two acceptable alternatives below.)

```
public static void main( ... ) {
      if ( ... ) {
            ...
      } else {
            ...
      }
}
```

```
public static void main( ... ) {
      if ( ... ) {
            ...
      }
      else {
            ...
      }
}
```

Indentation should be one tab (i.e., three to four spaces) within each layer of braces (i.e., methods, if/else, for, while, etc.) Use either tabs or spaces, but do not mix the two. Consistency is crucial for readability. One exception to the rule is when a statement is

longer than 80 characters: break up the statement into several lines, and use an extra tab to signal that a statement is continued from the previous line. If more space would be suitable to help with improved readability, then use it. (See example below.)

```
public static void main( ... ) {
      System.out.println("Please choose from the following options\n" +
                         "1. Input values\n" +
                         "2. View Previous Values\n" +
                         "3. Reset Current Session\n" +
                         "4. Save Session\n" +
                         "5. Exit Session\n");
      // Code to obtain input from user.
      // Further processing...
}
```

The proper use of commenting is an art, developed with repeated practice. For now we offer these guidelines on their formatting:

- Look at examples of good formatting and comment-writing.
- Develop your own style by keeping to the approved methods, and be consistent.
- Use implementation comments liberally while developing the code or debugging. Insert personal comments as placeholders for code that can be done 'later' in the process. When the code is complete, then remove all unnecessary commenting.
- Minimize the use of *superfluous,* or obvious comments. For instance, the following comment should never be part of completed code.

```
int x = 1; // 1 is assigned to x
```

Closing observations The coding conventions in this document are suitable for use in this and other first- and second- year computer-science courses at UVic and must be used throughout CSC 115. Each organization typically has its own conventions, so always check with your team for the expectations. (For an extremely detailed example see Google's Java Style).

Consistency of formatting with all source files making up your program is very important. This helps others understand your code and helps you and others identify issues with the code (and perhaps correction).