**Supplement D: Expanded Guidelines on Programming Style and Documentation**

For Introduction to Java Programming

Y. Daniel Liang

[**y.daniel.liang@gmail.com**](mailto:liang@armstrong.edu)

**Introduction**

Programming style deals with the appearance of your program. If you were to write an entire program on one line, it would compile and run properly, but doing this would be bad programming style because the program would be hard to read. Documentation consists of explanatory remarks and comments for the program. Programming style and documentation are as important as coding. Good programming style and appropriate documentation reduce the chance of errors and make programs easy to read. This handout provides more detailed guidelines on programming style and documentation that supplements the brief guidelines presented in Chapter 2, "Elementary Programming." First, here is a sample code:

/\*\*

\* Class: CSCI1301-03 Introduction to Programming Principles <br />

\* Instructor: Y. Daniel Liang <br />

\* Description: (Give a brief description for Exercise 1) <br />

\* Due: 1/18/2013 </ br>

\* I pledge that I have completed the programming assignment independently. <br />

I have not copied the code from a student or any source. <br />

I have not given my code to any student. <br />

<br />

Sign here: \_\_\_\_\_\_\_\_\_\_ <br />

\*/

**public** **class** Exercise1 {

/\*\* The main method displays three message \*/

**public** **static** **void** main(String[] args) {

// Use the println statemetns to display three messages

System.out.println("Programming is fun");

System.out.println("Welcome to Computer Programming");

System.out.println("Java is a programming language");

}

}

/\*\*

\* Class: CSCI1301-03 Introduction to Programming Principles

\* Instructor: Y. Daniel Liang

\* Description: (Give a brief description for Exercise 2)

\* Due: 1/18/2013

\* I pledge that I have completed the programming assignment independently.

I have not copied the code from a student or any source.

I have not given my code to any student.

Sign here: \_\_\_\_\_\_\_\_\_\_

\*/

**public** **class** Exercise2 {

/\*\*Main method\*/

**public** **static** **void** main(String[] args) {

**double** radius;

**double** area;

// Prompt the user to enter radius

System.out.print("Enter radius: ");

radius = MyInput.readDouble();

// Compute area

area = radius \* radius \* 3.14159;

// Display results

System.out.println("The area for the circle of radius " +

radius + " is " + area);

}

}

NOTE: This guideline is consistent with the Java API source code.

**Appropriate Comments and Comment Styles**

Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses. In a long program, you should also include comments to introduce each major step and to explain anything that is difficult to read. It is important to make your comments concise so that you do not crowd the program or make it difficult to read.

Use the javadoc comments (/\*\* ... \*/) for commenting an entire class and an entire method. These comments must precede just before the class or the method header and they can be extracted in a javadoc HTML file. For commenting the steps inside a method, use line comments (//). For information on javadoc comments, see http://java.sun.com/j2se/javadoc/.

Eclipse TIP: Use Ctrl+/ as a line comment toggle. Ctrl+/ adds comment tags (//) to the line of code the cursor is on or removes existing comment tags. Highlight blocks of code and use Ctrl+/ to comment or remove existing comments on an entire block.

**Naming Conventions**

Make sure that the meanings of the descriptive names you choose for variables, constants, classes, and methods are straightforward. Names are case-sensitive. Listed below are the conventions for naming variables, methods, classes, and packages.

       For variables and methods, always use lowercase. If the name consists of several words, concatenate them into one, making the first word lowercase and capitalizing the first letter of each subsequent word in the name; for example, the variables radius and area and the method readDouble.

       For class names, capitalize the first letter of each word in the name; for example, the class name ComputeArea.

       All letters in constants should be capitalized, and underscores should be used between words; for example, the constant PI and constant MAX\_VALUE.

       Use singulars for variables representing single items such as student and count. Use plurals for arrays or collections. For example,

      Student[] students = new Student[4]; and Count[] counts = new Count[10];

TIP: It is important to become familiar with the naming conventions. Understanding them will help you to understand Java programs. If you stick with the naming conventions, other programmers will be more willing to accept your program.

TIP: Do not choose class names that are already used in the Java standard packages. For example, since the Math class is defined in Java, you should not name your class Math.

**Package-Naming Conventions**

Packages are hierarchical, and you can have packages within packages. For example, java.lang.Math indicates that Math is a class in the package lang and that lang is a package within the package java. Levels of nesting can be used to ensure the uniqueness of package names.

Choosing a unique name is important because your package might be used on the Internet by other programs. Java designers recommend that you use your Internet domain name in reverse order as a package prefix. Since Internet domain names are unique, this avoids naming conflicts. Suppose you want to create a package named mypackage.io on a host machine with the Internet domain name liang.cs.armstrong.edu. To follow the naming convention, you would name the entire package edu.armstrong.cs.liang.mypackage.io.

Java expects one-to-one mapping of the package name and the file system directory structure. For the package named edu.armstrong.cs.liang.mypackage.io, you must create a directory, as shown in the Figure 1. In other words, a package is actually a directory that contains the bytecode of the classes.

[](http://www.cs.armstrong.edu/liang/intro9e/supplement/coding2.jpg)

Figure 1:  The package edu.armstrong.cs.liang.mypackage.io is mapped to a directory structure in the file system.

**Proper Indentation and Spacing**

A consistent indentation style makes programs clear and easy to read. Indentation is used to illustrate structural relationships among the program’s components or statements. Java can read the program even if all of the statements are in a straight line, but it is easier to read and maintain code that is aligned properly. You should indent each subcomponent or statement *two* spaces more than the structure within which it is nested.

Use a space to separate parameters in a method. Do not leave spaces  before or after parentheses in a method. For example, aMethod(a1, a2) is preferred, whereas aMethod ( a1, a2 ) is not a good style.

A single space should be added on both sides of a binary operator, as shown in the following statement:

boolen b = 3 + 4 \* 4 > 5 \* (4 + 3) - ++i;

A single space line should be used to separate segments of the code to make the program easier to read.

**Block Styles**

A block is a group of statements surrounded by braces. A block can be written in many ways. For example, the following are equivalent:

public class Test

{

  public static void main(String[] args)

  {

    System.out.println("Block Styles");

  }

}

public class Test {

  public static void main(String[] args) {

    System.out.println("Block Styles");

  }

}

The former is referred to as the *next-line* style, and the latter, as the *end-of-line* style. The end-of-line block style is used in this book.

**if-else Style**

  if (cond1) {

// Statements

}

else if (cond2) {

// Statements

}

else if (cond3) {

// Statements

}

else {

    // Statements

  }

**for loop Style**

  for (int i = 1; i < n; i++) {

    // Statements

  }

**while loop Style**

  while (i < n) {

    // Statements

  }

**do-while loop Style**

  do {

    // Statements

  } while (i < n);

**try-catch Style**

  try {

    // Statements

  }

catch (Exception ex) {

// Handler

}

**Summary**

* Use javadoc comments at the beginning of the program to state your name, class, and give a brief description for the program.
* Use javadoc comments to describe an entire method.
* Use line comments to describe steps inside a method.
* Indent your statement two spaces.
* Use Next-Line style.
* Leave a blank line before a comment line or a comment paragraph.