While most people think of computer programs as a tool for telling computers what to do, programs are actually much more than that. Computer programs are written in human readable language for a reason – because human beings need to know what is in the code that the computer is executing. High-level computer programming languages (like Java) were developed specifically to make it easier for human beings to understand the code they write.

However, even high-level programming languages can often be cryptic to people trying to understand the code. Often a programming language is not expressive enough to be able to really get across to a human being what the code is supposed to be doing – which is an important thing when a human being is trying to debug or modify a piece of code (especially when that code was written by another person). Because of this, every computer language possesses the ability to embed natural language descriptions called comments into the code. These comments allow programmers to make notes to themselves and to others about what the code they are writing is supposed to do.

In this class it is important that your code have good comments and good structure. The Eclipse IDE helps you with some of the structure (whitespace, bracket placement, etc.), and good choices for variable names and method names goes a long way to having good structure. But what makes for good comments?

This document describes the commenting conventions for this class (CSE 214). Your labs will be graded with the following conventions in mind. Be aware that if you deviate from these conventions you may be assessed a penalty on your final lab score. It is also important to note that if we have problems understanding your code and you leave no comments to help us figure out what your code is doing, your final grade on your lab may be impacted – good comments provide a way for us to figure out where you might have gone wrong with your code (and assess partial credit accordingly).
Example Program

The following program, computeArea.java, shows an example of the expected commenting conventions used for this class. These conventions make use of the Java “DocBook” commenting format, which you can feel free to read more about if you are interested. However, we will not be using all of the features of the DocBook commenting format in this class.

```java
/**
 * computeArea.java
 * A simple program that computes the area of rectangles
 * Used to demonstrate the proper use of comments for CSE 214
 *
 * @author Jeremy Morris
 * @version 20110328
 */
import java.util.*;

public class ComputeArea {
    /*
    * prompt
    * Asks the user to provide an integer value. If the user does not provide
    * an integer value, responds with an error and asks the user again until
    * a valid integer is provided.
    *
    * @param in - Scanner object that input is taken from
    * @param message - Message to display to the user to prompt for input
    * @return - An integer value provided from the Scanner object
    */
    private static int prompt(Scanner in, String message) {
        int response=0;
        // Loop until a valid response is obtained
        boolean validResponse = false;
        while (!validResponse) {
            try {
                // If the user provides an integer at the prompt, the loop ends
                System.out.print(message);
                response = in.nextInt();
                validResponse=true;
            }
            catch(InputMismatchException e) {
                // If the user provides anything except an integer, catch the
                // InputMismatchException and display an error message
                System.out.println("ERROR! Only integer values are allowed!");
                System.out.println("Please try again.");
            } // Since the "nextInt()" above had an exception, the bad data
            // is still in the Scanner. Use a "nextLine()" to clear it out
            // so the user can provide a valid response.
            if (in.hasNext())
                in.nextLine();
            }
            return response;
        }
```
Let's examine each of these pieces in turn:
Program Header

/**
 * computeArea.java
 * A simple program that computes the area of rectangles
 * Used to demonstrate the proper use of comments for CSE 214
 * @author Jeremy Morris
 * @version 20110328
 */

The program header comment should be at the very top every piece of Java code you write. It needs to contain the name of your program and a short description of what the program does in your own words. This should be a short summary of the lab assignment, not just a cut-and-pasting of the lab requirements into the program and it should be no more than 5-6 sentences even for a complex lab.

Following the short description are two special tags. The @author tag indicates the name of the person who wrote the code – this should be your name and it needs to be in the Program Header for every piece of Java code you write. The second tag is the @version tag – normally this tag would contain a version number for your code. For the purposes of this class the version number should be a date in YYYYMMDD format (so the above date indicates that this code was written on March 28, 2011). This date can either be the date you started to code the lab, the date you finished coding the lab, or the due date for the lab. But both your name and the date need to be in the program header of your code.

Main Method Header

/**
 * main
 * Provides the main input loop, taking information about rectangle length
 * and width from the user. Program ends when the user gives a rectangle
 * with at least one side with a length of 0.
 * @param args - command line arguments (ignored in this program)
 */

The main method header sits above your main method. The first line should contain the name of the method (main), followed by a short description of what the method is supposed to do. Again this is a short description – no more than a few sentences.

The main method will include a single tag – the @param tag. This tag indicates what parameters the method takes. For a main method, the “args” parameter is always taken and indicates arguments passed to the code on the command line. For labs that do not use command line arguments, this parameter is ignored and that should be noted in your comment (as in the example above).
Other Method Headers

/*
 *  Asks the user to provide an integer value. If the user does not provide
 *  an integer value, responds with an error and asks the user again until
 *  a valid integer is provided.
 *
 *  @param in - Scanner object that input is taken from
 *  @param message - Message to display to the user to prompt for input
 *  @return - An integer value provided from the Scanner object
 */

Other method headers follow the same format as the main method – the first line is the name of the method, followed by a short description of what the method is supposed to do. Unlike the main method, other methods may contain multiple @param tags. There should be one @param tag for each parameter passed into your method, and following the name of the parameter you should provide a short description of what the parameter is and how the method is supposed to use it.

Any method that is a function (rather than a procedure) will return a value – and this value should be noted with the @return tag. Again, provide a short description of what is returned by the method to the calling program.

Embedded Comments

// If the user provides anything except an integer, catch the
// InputMismatchException and display an error message
System.out.println("ERROR! Only integer values are allowed!");
System.out.println("Please try again.");

// Since the "nextInt()" above had an exception, the bad data
// is still in the Scanner. Use a "nextLine()" to clear it out
// so the user can provide a valid response.
if (in.hasNext())
    in.nextLine();

Embedded comments are comments made in the code to make it clear to people looking at the code what particular lines (or groups of lines) are supposed to do. Note that these should be fairly short comments – if you’re writing more than a few sentences to explain what you’re doing you might be making things too complicated, either in your code or in your explanation.

Following these commenting conventions will make your code more readable and more understandable. In the short term this helps us grade your labs appropriately – good comments are often the only way we can tell what you were thinking when you wrote a piece of code and be able to assess partial credit appropriately. But a good commenting style is probably even more important outside of the classroom – often good comments make the difference between code that is easy to maintain in a production environment and code that is a nightmare to work with. Good commenting styles are something that good programmers come to appreciate.