168 Coding Guidelines

This handout specifies coding guidelines for programs in ITK 168. You are expected to follow these guidelines precisely for all lecture programs, and for lab programs beginning in week 3. Sections of the guidelines will not apply to your earliest programs, but you will be expected to apply all portions that are relevant from the beginning.

Commenting
All code files should contain 2 opening comment blocks that specify information about the class or program defined in that file. The first opening comment block holds general information that is of interest only to someone looking at the code (and will appear green in Eclipse). The second block will be a Javadoc comment (and will appear blue in Eclipse). This will contain information for the external documentation of the program or class. This comment block will appear at the top of the class API. You can get Eclipse to generate a starting comment by checking this box (at the bottom of the new class window):

```
/*
 * Created on August 31, 2004
 * ULID: mecaliff
 * Course: ITK 168
 * Section: 7
 * Instructor: White
 */

//Place import statements between these 2 comment blocks
import something.somethingElse.*;

/**
 * This is a very simple program to demonstrate
 * the basic structure of a Java program and how
 * to print to the console window.
 * @author Mary Elaine Califf
 */
```
Notes: The import statements must appear above the class Javadoc comment block. Also, within the Javadoc comment block the @author tag (and @version tag if used) must follow the class description at the top of the block. The class description is what users will read to understand how and why to use your classes, so make them as clear and complete as possible.

In addition to opening comments, each public method (or service) except main must have a Javadoc comment unless it is overriding an existing method from its superclass. The Javadoc comment needs a good description of the method (fully explain how to use it and what it is used for, but do not discuss how it is coded). If the method has one or more parameters your Javadoc needs an @param clause for each parameter. The name following this tag must match the parameter name and is followed by a description of the expected contents. Javadoc comments start with /** and end with */. Here is an example:

```java
/**
 * Constructor for objects of class Employee
 * @param id The employee’s id
 * @param name The employee’s name in last, first format
 * @param pay The employee’s salary in whole dollars
 */
public Employee(String id, String name, int pay) {
    ...
}
```

If the method has a return type (other than void) use @return to describe the returned value. If the return value could be one of a few choices, you should describe all possible return values – however you can only code @return one time. Here is an example:

```java
/**
 * compareTo method compares 2 Student objects based on their GPA
 * @param stu Student object to compare to the calling object
 * @return A negative number if the calling Student has a lower GPA; zero
 * if the 2 Students have the same GPA; or a positive number if the calling
 * Student has a higher GPA
 */
public int compareTo(Student stu) {
    ...
}
```

Inside your methods you should add internal comment lines to clarify your code. These comments are used to help explain complex code or divide code into blocks. The best programmers write comments before writing code! Single line comments start with // and do not need a closing tag, but these comments cannot wrap to multiple lines. If the comment covers multiple lines, each line must start with //. There are many examples of inline commenting throughout your text.
Braces and Indentation of code
Curly braces are used in Java to enclose blocks of code. You’ve seen them thus far as part of the class definition and as part of the definition of the main method. There are two acceptable ways to format these braces – although we prefer the first. Regardless of the placement of the braces, everything inside the braces should be indented one level.

Method 1:
```java
public class ClassName {
    // indent everything between the braces
}
```

Method 2:
```java
public class ClassName {
    // indent everything between the braces
}
```

You can and should let Eclipse work for you in maintaining clean code. When you add and remove code from your programs you will often end up with unaligned code. Eclipse will automatically clean up your code for you using the following steps:

- Right click anywhere in your document
- Choose source -> format
- -or- simply use ctrl-shift-f as a single keystroke shortcut

Make sure you clean up your code often. Do not wait until you are ready to submit (but be sure to do it just prior to submitting also). Keeping your code clean and properly aligned will help you debug your control structures. It is easy to miss a closing curly brace, and that one missing brace will completely mess up your code!