CSE 11
Fall 2013
Style Guide

Introduction

Why are comments and style guidelines important? Comments help you and others read and understand your program, it's logic, and how it should be used. Unfortunately, many programmers ignore the value of good comments. It is also possible to put too many comments in your code (but very few programs suffer from this particular problem.)

Style guidelines are important for larger codes so that it is easier to figure out where various code segments begin and end, this is particularly important in methods, if statements, loops, and exception blocks.

You will develop your own style of indentation, white space and comments as you do more programming. These guidelines are intended to help you structure comments and programs to be more readable.

Comment Guidelines

It is reasonable to break comments into several key ideas

1. File Header Comments
2. Short Description/Purpose of the class being commented
3. Building, Running, and Dependency Comments
4. Methods defined with call signatures (constructors come first)
5. In-line comments to make code more understandable

The following example shows a reasonable set of comments for the first four points. The basic idea for these beginning comments to be an outline of the what the class is used for with some detail, but not so much detail that one gets lost. For class assignments, it is quite reasonable to put course and which assignment this is for after your student id.

```
/*********************************************************
* Sun.java
*
* Author: Philip Papadopoulos
* Student ID: 666
* Course: CSE11 Spring 2013
* Assignment: Program #1
*
* Creation Date: 9 April 2013
* Last Modified: 12 April 2013
* Description:  Creates a graphical Sun object that can be
*               moved, reset to it's original position
*               when first constructed, colored, and report current
*               location.
*
*               This is intended to be called by other classes.
*
```

```
* Build:   javac -classpath '*':'.' Sun.java
* Dependencies:  objectdraw.jar, java.awt.*
*
* Public Methods Defined:
*           Sun(Location, double, DrawingCanvas)
*           Sun(double, double, double, DrawingCanvas)
*           void resetLocation()
*           void setColor(Color)
*           void move(double)
*           Location getLocation()
*
* Public Class Variables:
*       None
*
*****************************************************************/
public class Sun {
// Class Variables
// Constructors
// Methods
}
```

The last section is for in-line comments. Some rough guidelines, for methods it is useful to give not only the signature but also what each parameter is called and its logical function. For example, the Sun constructor method could be commented as

```
public class Sun {
        /***** Constructor
         *
         * Location initial      Where to create the sun on
         *                       the canvas
         * double diam           diameter of the Sun object
         *                       ,in pixels
         * DrawingCanvas canvas  the objectdraw canvas
         *
         *****/
        public Sun(Location initial, double diam,
                                DrawingCanvas Canvas) {

        }

}
```

Use white space to make things more readable and to set of sections of code that are related. It is often useful to comment variables, or a code block; One might comment a variable as follows

```
Location initialLocation;  // remember where the object was first created
```

If you have a complicated block of code, write an English description of what the code block is supposed to do, and then write the code. Think of comments a human-readable summary of what the code is supposed to do.

Indentation

Indentation is critical to easily seeing where blocks of code begin and end. Blocks of code can be if

statements, for/while loops, methods, and classes. Let's look at statement blocks. A java statement block looks like

```
{
    java statement 1;
    java statement 2;
    ...
}
```

Let's put this in an if statement, there are two generally acceptable ways to indicate the beginning and end. The first way is where the opening brace is on the same line as the if statement.

```
if (condition) {
        java statement 1;
        java statement 2;
        ...
}
```

The other accepted way of doing this, is to put the opening brace on the first line under the if statements. e.g.,

```
if (condition)
{
        java statement 1;
        java statement 2;
        ...
}
```

Choose the method you like and stay with it.

Now, notice the indentation. Whenever you have block of statements, indent them. You can use tabs to indent, or spaces. I am a fan of using tabs. The only problem with tabs is by default they indent 8 spaces (as in the examples above). 4 spaces is more rational if you have many indent levels. In vim, you can change this with the following command

```
: set tabstop=4
```

Here's an example at tabstop=4

```
public class MyExample
{
    public void mutateMethod()
    {
        if (condition)
        {
            java statement 1;
            java statement 2;
        }
```

```
        else
        {
            if (nestedCondition2)
            {
                java statement 3;
                java statement 4;
            }
        }
    }
}
```

And the same at tabstop=8

```
public class MyExample
{
        public void mutateMethod()
        {
                if (condition)
                {
                        java statement 1;
                        java statement 2;
                }
                else
                {
                        if (nestedCondition2)
                        {
                                java statement 3;
                                java statement 4;
                        }
                }
        }
}
```

Choose for yourself which tabstop you prefer when editing. A final note on indentation, some prefer spaces to tabs. A great deal of code (e.g. the Linux kernel) is written using TABS. Do yourself a favor, use tabs for indentation, it's more universal. Setting a tabstop in vi allows you to quickly increase or decrease the tab depth to make the code more readable on your screen.

A helpful hint for vim, If your cursor is on an opening or closing brace, hit '%', this will take you the other brace for the code block (If you are at the end, it takes you to the beginning, at the beginning it takes you to the end). This very handy for quickly moving around a java source file. '%' also works for open/closing parenethesis “( )” and brackets “[ ]”

Capitalization Conventions

- Classes – First letter is capitalized, then mixed case.
  - Good: MyClass
  - Not so Good: myClass or myclass or MYCLASS or mYcLASS

- Instance variables – first letter lower case, then mixed case
  - Good: myInstanceVar
  - Not So Good: MyInstanceVar, myinstancevar. MYINSTANCEVAR
- Constants – All CAPS with Underscores between works
  - Good: MAXIMUM_PERCENTAGE_RATE
  - Not so good: maximumPercentageRate, maximumpercentagereate

Initialization

Java initializes all class and instance variables to “zero”. While this is convenient, it is not universal in other programming languages.

Java never initializes temporary variables

-- Programs should explicitly initialize (or assign) all variables irrespective of whether they are class, instance, or temporary variables BEFORE using them in expressions. This will keep subtle bugs from cropping up when you write in other languages.