Java Style Guide

This document is intended to guide you in formatting your code for this class. While code formatting might seem arbitrary, it is important for communicating your understanding of your program; the purpose of your variables, the control flow of your program's execution. Additionally, as you program more, you will hopefully come to appreciate the clarity good style can bring to your understanding of your code, particularly for debugging purposes.

Consider writing an essay for a class. It is important to communicate to the reader a clear structure of your argument through the use of section headers and paragraphs. You strive to arrange these in a way that makes sense, grouping similar ideas into the same paragraph. Similarly, when writing code, we aim to achieve good style to aid in communicating our ideas for solving some problem.

As you write your programs, strive to be intentional in your style as you write each line of code. While there is still value in fixing your style at the end of your program, developing these habits early and through constant practice will serve you well.

This document is structured according to the different components of the Java programming language: comments, identifiers, control flow statements, and spacing. There are examples in each section, as well as two longer examples (one good, one bad) at the end of the document.

Comments

Comments should be descriptive given the current context. For example, a comment for the next two or three lines should describe just those lines, while the comment for an entire code block (e.g., method, class) should describe that entire code block:

```
// Creates a new Scanner variable to read in the user's input
```

```
/*
 * Calculates the average age of the adults based on an array of ages. Ages 17
 *   and younger are not considered.
 */
```

When in doubt, comment! Commenting might help me give you partial credit if I can better understand what you were thinking.

Also, remember to fill out the header comment at the top of your file (just above the class) with the required information: program description, your name, and the date last modified:

```
/**
 * My tip calculator program. Takes a given total price and a
 * tipping rate (in percent) and calculates the tip and the new
 * price with tip included.
 *
 * Last Modified: Sept. 8, 2015
 * @author Allie Sauppe
 **/
```

Identifiers

Below is information on naming various identifiers in Java, such as variables, methods, and classes. As a general rule, all identifiers should be descriptive of what they are identifying.

Variables

All variable identifiers should be descriptive, with their name reflecting the data they contain. Variable identifiers should be written in camelcase with the first letter lowercase. Variables with the `final` descriptor should be written in all capital letters with underscores between words. See the examples below:

```
int total;

String currentWeather;

final TWO_PI;

// This code snippet reflects proper naming of a counter variable for a loop
for(int index = 0; index < 10; ++index) {
    System.out.println(index);
}
```

Classes

All class identifiers should be descriptive, with their identifier reflecting the real-world object or concept contained in the class. Class identifiers should be written in camelcase with the first letter uppercase. See the examples below:

```
public class Professor

public class HelloWorld
```

Methods

All method identifiers should be descriptive, with their identifier reflecting the purpose of the code in the method. Method identifiers should be written in camelcase with the first letter lowercase. Parameters should follow the conventions outlined for variables above. See the examples below:

```
public static void printStatements(String[] stmts)

public static void calculateTip(int total, double tippingRate)
```

Control Flow

Control flow statements (i.e., loops, conditionals) should be formatted consistently through a program. There are multiple accepted styles for control flow statements, two of which are demonstrated below with conditionals:

```
if (age < 18) {
    System.out.println("You must have an adult accompanying you to see this
    movie.");
} else if (age > 62) {
    System.out.println("You can receive a discount at this movie!");
} else {
    System.out.println("You will pay standard pricing for this movie.");
}
```

```
if (age < 18)
{
    System.out.println("You must have an adult accompanying you to see this
    movie.");
}
else if (age > 62)
{
    System.out.println("You can receive a discount at this movie!");
}
else
{
    System.out.println("You will pay standard pricing for this movie.");
}
```

Key things to note in the above examples:

- They keyword for the conditional (e.g., `if`, `else if` should be on the same line as the condition associated with it.
- Opening curly brackets are either on the same line as the conditional statement (e.g., `if`) or are on their own line, indented at the same level as the conditional statement.
- Closing curly brackets should be at the same level of indentation as the conditional block, and are on the line after the last line of code in the conditional.
- Any code contained in the conditional block is indented inside the block.

Similar guidelines apply to loops:

```
for (int index = 0; index < 10; ++index) {
    System.out.println(index);
}
```

```
for (int index = 0; index < 10; ++index)
{
    System.out.println(index);
}
```

Spacing

Line breaks should be inserted in the code to break it up into logical chunks; there should never be more than one blank line in a row. A good rule of thumb is to have a line break, a comment, and then a chunk of code that comment describes. See the examples below:

```
// Variable creation
int age;
String name;
Scanner scan = new Scanner(System.in);

// Reads in the data from the user
System.out.print("What is your name? ");
name = Scanner.nextLine();
System.out.print("What is your age? ");
age = Scanner.nextInt();

// Displays the information to the user
System.out.println("This is " + name + " who is " + age + " years old.");
```

Formatting

All code should be properly indented. I would recommend doing this by hand to better understand how Java is structured, but you can also have Eclipse automatically format your code by selecting **Source** → **Format**. Below are two examples, one good and one bad, of formatting for a variety of Java components.

Good Example

```
import java.awt.Color;
import java.util.Scanner;
import galapagos.Turtle;

/**
 * Style Example
 *
 * @author Allie Sauppe
 */
public class TurtleGraphics {

    /**
    * Main method for executing the program.
    * @param args
    */
    public static void main(String[] args) {
        //creates a new Turtle object
        Turtle myTurtle = new Turtle();

        //changes the pen color to black
        myTurtle.penColor(Color.MAGENTA);

        //increases the pen size to 10
        myTurtle.penSize(10);

        //increases the turtle's speed
        myTurtle.speed(100);

        //draws some lines
        boolean isPenDown = true;
        for(int index = 0; index < 10; ++index) {

            //if we are still drawing, pick up the pen
            if(isPenDown) {
                isPenDown = !isPenDown;
            }

            myTurtle.move(10*index);
            myTurtle.turn(45);
        }
    }
}
```

Bad Example

```
import java.awt.Color;



import java.util.Scanner;
import galapagos.Turtle;

public class TurtleGraphics {

    /**
    * Main method for executing the program.
* @param args
    */
   public static void main(String[] args) {
        //creates a new Turtle object


        Turtle t = new Turtle();
                t.penColor(Color.MAGENTA);
        t.penSize(10);

        tspeed(100);

        boolean isPenDown = true;
    for(int index = 0; index < 10; ++index) {
            if(isPenDown) { isPenDown = !isPenDown; }

t.move(10*index);
            t.turn(45);
}}}
```