Programming Guidelines for Java
CS 1101

This handout discusses four techniques which help to make a program more intelligible: descriptive identifiers, white space, indentation, and documentation. **You must follow the guidelines given here. There will be penalties for not doing so.**

1 The Use of Descriptive Identifiers

The variables, constants, and methods in your programs must all be given names, or *identifiers*. Choosing names that suggest what the variable, constant, or method does helps a reader of your code to understand how your program works. There are Java rules for identifiers that *must* be followed in that identifiers that do not comply with them will be flagged as errors and your code will not compile. There are also style guidelines for identifiers that *must* be followed in that identifiers that do not comply with them will be flagged as errors by the graders and points will be taken off. Here are the Java rules:

- The first character of an identifier must be a letter, an underscore (`_`), or a dollar sign (`$`).
- The rest of the characters in the identifier can be a letter, digit, underscore, or dollar sign. Note, in particular, that spaces are not allowed.
- Identifiers are case-sensitive. This means that `square` and `Square` are different identifiers.
- Identifiers cannot be the same as any of Java's *reserved words*:

```
Abstract assert    boolean   break  byte     case  catch      char         class
const    continue  default   do     double   else  enum       extends      false
final    finally   float     for    goto     if    implements import       instanceof
int      interface long      native new      null  package    private      protected
public   return    short     static strictfp super switch     synchronized this
throw    throws    transient true   try      void  volatile   while
```

A widespread style, *which you are required to use*, does not use the dollar sign at all in identifiers and does the following:

1. for variables and methods, use only letters and digits, separating words in the identifier by capitalizing the first letter of each word, *except for the first word*,
2. for constants, use all upper case letters, separating words with an underscore,
3. for class names, use only letters and digits, separating words in the identifier by capitalizing the first letter of each word, *including the first word*.

1.1 Variables and Constants

Here are some additional guidelines on creating identifiers. Noun phrases are good choices for variables and constants that describe *things*. For example:

```
private FramedRect sign = new FramedRect(285, 60, 80, 80, canavs);
private Text clickingText = new Text("CLICKING", 295, 95, canvas);
private Line slash = new Line(302, 77, 348, 126, canvas);

private static final double SALES_TAX_RATE = 0.065;
private int numReservedCustomers;
private double wagePerHour;
private char reply;
```

(Note that these are all variables and assignments as they would appear if declared globally (outside all methods). If they were declared locally (in a single method), you would not use the `private` or `static` modifiers.

Note that you do not need to use named constants for values that are obvious and general, e.g. dividing by 2 to calculate an average. You should name a constant (and use the name) when it means something special, e.g. the sales tax rate above, or the dimensions of an object (e.g. `BOX_WIDTH`).

Boolean variables often represent *conditions*, and their names should reflect that, e.g.:

```
boolean done;
```

Identifiers such as `x` and `y` should only be used if the value has no significance other than the fact that it is a real number, such as in a mathematical formula:

```
double y = a * x + b;
```

Identifiers such as `i`, `j`, and `k` are typically used to stand for arbitrary integer values, especially those used as loop control variables.

1.2 Methods

A method that returns a boolean value can often be named by the yes/no question that it answers, such as `isWithinRange` or `wantsToPlay`. Other times a method returns a non-boolean value (e.g. a numerical value) and the name of the method should indicate what that value is, as in `greatestCommonDivisor` or `absValueDifference`. Finally, a method might not answer a question or return a value, but, instead, perform some job on its inputs. A natural choice for naming this kind of method is a description of what it does, such as `playGame` or `runCalculator`.

2 The Use of White Space

Use white space (blank spaces and lines) freely to make your code easier to read. Observe the following rules:

- Don't put more than one statement on a line.
- Put spaces between operands and operators of expressions.
- Put blank lines between groups of statements that perform well-defined tasks (e.g. drawing some graphics that require you to create several graphical objects, doing some calculations that are related to each other).

3 Indentation

Each line in the body of a method (such as the `begin` method or the `onMousePress` method, is indented the same amount. Within a method, indentation is used to indicate what statements belong in an `if` or `else` statement, or a `while` or `for` loop. I will provide details in class when we talk about these constructs. Use the tab key to keep your indentation consistent. In particular, be sure that blocks of code at the same level in your program are indented the same amount. *We will take points off for indentation that is not consistent or that makes it difficult to follow what you are doing.*

A related issue is the placement of curly braces that enclose a block of code. The opening curly brace should be on the same line as the construct that starts that block (`if`, `else`, `while`, or `for`) and the closing curly brace should be lined up with the `if`, `else`, `while`, or `for`, e.g.

```
if (x < 20) {
    x = y + z;
    System.out.println(x);
}
```

4 Documentation

Documentation refers to comments placed in your program that explain what the program is doing. *We will take points off for insufficient documentation.* For this course, there are two kinds of documentation: summary documentation and in-line documentation.

4.1 Program Summary

The file containing your program should begin with a comment that includes the name and number of the lab, your name, the date, and a summary description of what the program

does, including any assumptions that the program makes about how it is going to be used, including particular cases that it can or cannot handle (particularly if there are cases that might go against reasonable user expectations).

For example:

```
/*
    Crosshairs
    Sean Barker
    1 October 2014
    CS 1101B

    This program allows the user to drag the center of two
    crosshairs around a window with the mouse.  When the
    mouse enters the window, text appears that tells the
    user what to do.  When the user starts dragging the
    mouse in the window, the crosshairs appear and the
    crosspoint follows the mouse arrow.  Note that the
    program is written to that the window can be resized.
    When the user resizes the window, and enters the window
    the text appears again and the user can drag the mouse
    to make the crosshairs move as they did in the original
    window.
 */
```

Initially, when your programs are quite simple, the program description can be quite brief. As your programs increase in complexity, the length of the summary will increase as well.

4.2 In-line Documentation

Comments placed between lines or blocks of code in a program are called *in-line* comments. Often it is helpful to use comments to explain blocks of code in your program. For example:

```
    // create a new random color
    int redLevel =  randomColorElement.nextValue();
    int greenLevel =  randomColorElement.nextValue();
    int blueLevel =  randomColorElement.nextValue();
    circleColor = new Color(redLevel, greenLevel, blueLevel);
```

In-line comments are generally necessary in all but the simplest programs, but don't go overboard! Not every line of code needs a comment.

4.3 Method Summaries

Methods should sometimes have summaries, similar to program summaries, just above their header statements. For predefined methods, such as `begin` and `onMouseDrag`, where the inputs and outputs are already specified, a summary is not necessary if the method has relatively few lines of code with an in-line comment or two describing what the code does. If the predefined method contains a lot of code, in-line comments should be used to document what it does, and a short summary just above the method header should be used to give a high-level explanation of what the method does.

For methods that *you* write, you should always have a summary above the header that includes a description of the inputs to the method, what the method does, and the return value (if any) of the method. It should also include any assumptions that the method makes about the input. For example:

```
//   This method accepts as input a double value and, after
//   checking to ensure that the value is positive (> 0), it
//   calculates and returns the log base 2 of the input;
//   otherwise it prints out an error message and returns -1.
//
public double logBase2 (double n) {

  double logBase2Value = -1;

  if (n > 0.0)
       logBase2Value = Math.log(n) / Math.log(2);
  else
       System.out.println("Error: Cannot compute the log of zero!");

  return logBase2Value;

}
```