Programming Guidelines for Java
CS 107

Stephen Majercik

This handout discusses four techniques which help to make a program more intelligible: descriptive identifiers, white space, indentation, and documentation. **Please follow the guidelines given here, since there will be a substantial penalty 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.

1.1 Variables and Constants

Noun phrases are good choices for variables and constants that describe *things*. For example:

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

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

```
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.

A widespread style, illustrated in the examples above (and which I am requiring that you use), does not use the underscore in variable identifiers, separating words in the identifier by capitalizing the first letter of each word, except for the first word. Identifiers for constants, on the other hand, should be in upper case letters, with words separated by an underscore.

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 may 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`.

Use the same style in naming methods as that described above for variables—do not use the underscore and separate words in the identifier by capitalizing the first letter of each word, except for the first word.

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. getting the user inputs, doing some calculation).

3 Indentation

Each line in the body of a method (such as the `main` method, in which you will be writing your entire program in the beginning) 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. Xcode will often do the proper indentation for you; otherwise, use the tab key when editing files 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. I'm not going to sit there and count spaces when I'm grading your programs, but I *will* take points off for indentation 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. For this course, there are two kinds of documentation: summary documentation and in-line documentation.

4.1 Summary Documentation

4.1.1 Program Summary

The file containing your program should begin with a comment that includes a summary description of what the program does, your name, and the date. This should include a description of the inputs, the calculations that are done, and the outputs. It should also include any assumptions that the program makes about the input (implying inputs it can't handle), or particular cases that it can handle (that the person reading the program might not be sure whether it handles). For example:

```
//  This program reads in someone's age in years, months, and days and
//  calculates that age in hours, minutes, and seconds.  It ensures that
//  all inputs are non-negative, but does no other validity checks.
//
//  Stephen Majercik
//  24 September 2005
```

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. I will provide further guidelines in class as needed.

4.1.2 Method Summaries

Methods (except for the `main` method) should also have summaries just above their header statements. Method summaries are similar to the program summary. They should include a description of the inputs to the method, the calculations that are done, and the return value (if any) of the method. It should also include any assumptions that the method makes about the input.

```
//   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 static double logBase2 (double x) {

  double logBase2Value = -1;

  if (x > 0.0)
```

```
        logBase2Value = log(x) / log(2);
    else
        System.out.println("Error: Cannot compute the log of zero!");

    return logBase2Value;

}
```

4.2 In-line Documentation

Comments placed between lines or blocks of code in a program are called *in-line* comments. If your program is well-modularized (broken up into logical methods), your method summaries are likely to provide all the documentation that is needed. In fact, if you find yourself writing a lot of in-line comments, it is a good indication that you have done a poor job modularizing your code and should work on that. Sometimes, however, it is helpful to use comments to explain *blocks* of code in your program. For example:

```
// Get the user's age and reject values not greater than zero.
System.out.print("Enter your age in years:  ";
int ageInYears = r.readInt();
while (ageInYears <= 0) {
   System.out.print("Your age must be > 0.  Please re-enter:  ");
   ageInYears = r.readInt();
}
```

In general, however, in-line comments should be used sparingly.