CS18000 Java Coding Standards

The following standards are to be followed in writing Java programs. They are important enough to be part of the grading criteria for all programs. Any constructs not covered in this standard are to be implemented in a consistent and readable fashion.

Purpose

The purpose of these rules are (1) to introduce you to a representative set of coding standards that are typical in professional programming and (2) to help you develop good style as a habit, something you will find necessary to successfully complete large programs. There is no doubt that you can write Java programs to solve the small problems in this course without attention to style. But, you must keep in mind that you are building professional skills, not merely finding the answers to problems handed out as exercises.

File Names and Java Compiler

In this course each Java file name should clearly represent what it is and each should have the `.java` suffix. Files names must match the class name. For example, a class named `Robot` should be a file named `Robot.java`. A lab assignment that determines the payroll of a small company could be in a file named `Payroll.java` (or even something like `Lab3.java`).

Introductory Documentation

At the beginning of each Java class there should be a JavaDoc block that contains:

1. The program name, your name, and recitation/lab section.
2. A brief description of the program that describes the method or algorithm used to obtain the solution, the major data structures (if any) in the program, and other general comments including extra features.
3. The date the program was completed.

The format to be used is as follows:

```
// import statements here

/**
 * Program Name
 *
 * brief description of the program
 *
 * @author your name, section number
 *
 * @version date of completion
 *
 */
public class MyClass {
```

The following is an example of an introductory comment block that follows the format above:

```
import java.util.Scanner;

/**
 * Project 1 -- Video Calculator
 *
 * This program inputs the total number of minutes,
 * the number of minutes for commercials, and the
 * number of minutes for episodes.  The program
 * computes the total number of minutes for videos,
 * the number of videos, and the number of seconds
 * left over.
 *
 * @author Riya Thomas, lab sec 817
 *
 * @version September 17, 2017
 *
 */
public class VideoCalculator {
```

Declarations

1. You should use mnemonic names for all identifiers. These are names that have meaning in the problem. The names should suggest the use of the identifier.
2. Variable identifiers should be in all lower-case letters, except for capital letters used to delimit multiple words in a variable identifier. For example, shipRates, idEntry, numItems, and minScoreRequired.
3. Constant values used in your program will be given in final declarations. Constant identifiers should always appear in all-capital letters; underscore characters delimit multiple words. For example:

   ```
   final float BASE = 2.0;        // divisor to obtain base 2
   final char HYPHEN = '-';       // signals word continued on next line
   final int NAME_LENGTH = 20;    // names can be 20 characters
   ```

4. Classes should begin with a capital letter, and multiple words should be indicated by additional capital letters (for example, Date, VehicleNumber, SoftwareEngineer, BankAccount).
5. Methods should begin with a lowercase letter, and have any multiple words indicated by capital letters (for example, isValidNumber(), area(), setWidth(), etc). Note: Constructors are methods which cannot follow this convention, because their names must exactly match the class name.
6. Each identifier declaration (including method declarations) must be accompanied by a comment that explains its use.
7. Avoid declaring globally-visible variables. Globally-visible constants, enumerated types, classes, and file names are okay, but do not use global variables unless the project assignment

specifically tells you to do so.

General Rules

1. Each constant and variable declaration should be on a separate line that includes a comment that explains what each is. For example:

```
float volts;                        // voltage in the circuit
int   amps;                         // amperage in the circuit
char  circuitName[NAME_LENGTH];     // name of the circuit
```

2. Each statement should be on a line by itself. For example, instead of

```
left = nextLeft;     right = nextRight;
```

use

```
left  = nextLeft;      // move to the left
right = nextRight;     // move to the right
```

3. Each method should be identified by a comment block describing its purpose, parameter(s), return (if not void), and the method(s) it calls. The format to be used is as follows:

```
/**
 * Calculates and returns the charge for shipping cargo
 * between two planets.
 *
 * @param distance distance in miles between two planets
 * @param weight weight in pounds of item being shipped
 * @return A double representing the cost to ship the item in space
 */
public double findSpaceCost(long distance, float weight) {
  // implement here
}
```

4. Comments should be set apart from the code by at least two spaces, making it easy to see where the code ends and comment begins.

Formatting of Specific Java Statements

Alignment and indentation of lines of Java statements add much to the readability of programs. The following guidelines are general so that you may develop your own programming style. Use indentation and blank lines as you see fit to increase the readability of your programs. The body of a control structure, such as an if, a switch, a while, a do, or a for statement, should be indented. The following examples illustrate the only formatting style that is to be used in CS18000. Each level of indenting is 4 spaces. (All of the below examples should have 4 spaces for indentation, but not all browsers display it correctly.)

If you make minor changes to these formatting conventions, be sure you do so consistently.

if statement

```
if (expression) {
    statements
}
```

if-else statement

```
if (expression) {
    statements
} else {
    statements
}
```

switch statement

```
switch (selector variable) {
    case case-1-value:
        case 1 statements;
        break;
    case case-2-value:
        case 2 statements;
        break;
    ...
    case case-n-value:
        case n statements;
        break;
    default:
        default statements;
        break;
}
```

while statement

```
while (expression) {
    statements
}
```

do-while statement

```
do {
    statements
} while (expression);
```

for statement

```
for (initialization; condition; increment) {
    statements
}
```

Program Design

1. Use stepwise refinement. The programming assignments given are ones that can be solved in a step-by-step manner. The method of solution used in your program should exhibit this kind of structure.
2. Modularize your program. When a problem solution readily splits into distinct tasks, your program should handle each task in a separate function and in a straightforward manner.
3. Make your program efficient. Your algorithm should be direct and not unnecessarily complicated. Proper use of data structures will often simplify an algorithm.
4. Use parameters appropriately. All variables used in a method should either be declared as parameters or be declared locally within the method.

Output

1. All input values should be echoed in a readable manner unless otherwise indicated.
2. The program output should be readable and clearly labelled. Headings should be used when appropriate.
3. Finally, of course, your program should produce correct results!

Code Style Checking Tool

And you can follow the instructions on submitting to Vocareum to run the style checker