Java Coding Style Review

Qingyu Song, *Qiao Xiang*

School of Informatics

Xiamen University

Email: qingyusong@xmu.edu.cn,

qiaoxiang@xmu.edu.cn

Credit to Richard Yang @ Yale University

Coding Style

- “A set of rules or guidelines used when writing the source code for a computer program” – Wikipedia
- Make source code more readable and understandable
- Avoid errors

This review borrows largely from:

- Java Style Guide@Google:
  - https://google.github.io/styleguide/javaguide.html#s5.2-specific-identifier-names
- Java Coding Style Guide@w3resource:
  - http://www.w3resource.com/slides/java-coding-style-guide.php
- Java Coding Conventions@Oracle:
  - http://www.oracle.com/technetwork/java/codeconventions-150003.pdf
- Java Code Style Guidelines@Cornell:
  - https://www.cs.cornell.edu/courses/JavaAndDS/JavaStyle.html

Table of Contents

Table of Contents

Source File and File Name

- Files containing more than 2000 lines should be avoided.
  - Long files are hard to understand.
- The file must be named after the class it represents.
  - Both file name and class name are case-sensitive.
  - Each top-level class resides in a source file of its own.

Table of Contents

A Java source file has the following structure

1. Beginning comments
   - Used for licensing and copyright information.
2. Package statement
   - Not line-wrapped
3. Import statements
   - Not line-wrapped
4. Exactly one top-level class

Table of Contents

Class Layout

- Indentation
  - 4-space, 2-space or 1-tab
  - Either is acceptable, but need to be consistent across the whole project
- Line length
  - Typically no more than 100 columns

Variable Declarations

- One variable per declaration
- Declare local variables just before they are needed.

This one is preferred because it declares variables with minimal scope.

```
public static void main() {
   final int N = 10;
   int i;
   for (i = 1; i <= N; i++) {
      System.out.print( N+1-i + " " );
   }
   System.out.println();
   for (i = 1; i <= N; i++) {
      System.out.print( N+1-i + " ");
   }
}
```

```
public static void main() {
   final int N = 10;
   for (int i = 1; i <= N; i++) {
      System.out.print( N+1-i + " " );
   }
   System.out.println();
   for (int i = 1; i <= N; i++) {
      System.out.print( N+1-i + " ");
   }
}
```

Line Wrapping

- When an expression will not fit on a single line, break it according to these general principles:
  - break after a comma;
  - break before an operator;
  - prefer higher level breaks to lower level breaks;
  - indent the new line;
  - if the above rules lead to confusing code or to code that's squished up against the right margin, please use common sense.

```
public static void drawBook(int x0, int y0, int size) {
    // draw Yale blue background
    StdDraw.setPenColor(0, 53, 107);
    StdDraw.filledSquare(x0 + size / 2, y0 + size / 2, size / 2);

    // draw text StdDraw.setPenColor(Color.WHITE);
    StdDraw.textLeft(x0 + size / 2.0, y0 + 4 * size / 5.0, "CS112");

    // draw strips
    final int N = 10;
    double unit = (double)size / N;
    for (int line = 1; line <= N; line++) {
        double xLeft = x0;
        double length = (N + 1 - line) * unit;
        double yLow = y0 + (line - 1) * unit;
        double height = unit - 2; // leave 2 points as space

        StdDraw.setPenColor(192, 128, 64);
        StdDraw.filledRectangle(xLeft + length / 2.0,
                                yLow + height / 2.0,
                                length / 2.0,
                                height / 2.0);
    }
}
```

Blank Lines

- Blank lines improve readability by setting of sections of code that are logically related. One blank line should always be used in the following circumstances:
  - between methods;
  - before a block or single line comment;
  - between logical sections inside a method to improve readability.

```
public class Mirror {
    public static void main(String[] args) {
        line();

         /*this methods draws the top half of the mirror */
        topHalf();
        bottomHalf();
        line();
    }

    public static void topHalf() {
        for (int line = 1; line <= 4; line++) {
            // ...
        }
    }

    public static void bottomHalf() {
        for (int line = 1; line <= 4; line++) {
            // ...
        }
    }

    public static void line() {
        // ...
    }
}
```

Blank Spaces

- A keyword followed by a parenthesis should be separated by a space
- The expressions in a for statement should be separated by blanks

```
System.out.print("T-minus ");
for (int i = 10; i >= 1; i--) {
    System.out.print(i + “, “);
}
System.out.println("blastoff!");
```

- Blanks should appear after commas in argument lists;
- Blanks should **NOT** be used between a method call and its opening parenthesis. This helps to distinguish keywords from function calls.

```
public static void main(String[] args) {
    drawCar(80, 100, 100);
}
```

Blank Spaces

- All binary and ternary operators except "." should be separated from their operands by spaces.
- Blanks should never separate unary operators such as unary minus, increment("++") and decrement("--") from their operands

```
System.out.println("blastoff!");
a = 2 + 10 / 3 - 6 / 4;
a++;
```

- Casts should be followed by a blank

```
double result = (double) 19 /5;
```

Naming

- Package:
  - All lower cases
- Class:
  - UpperCamelCase

```
public class BookCover {
    // ...
}
```

- Method, parameter and local variable:
  - lowerCamelCase

```
public static void drawBook(int x0, int y0, int size) {
    // ...
}
```

- Constant
  - All uppercase letters, separated by underscore

```
public static final int DAYS_IN_WEEK = 7;
```