Style Guide | CS 61B Spring 2019

sp19.datastructur.es/materials/guides/style-guide.html

A note on this style guide: It is probably easier to get to know these rules by running the automated style checker. (More instructions on how to run the automated style checker will be given in the future.)

Whitespace

1. Each file must end with a newline sequence.

2. Files may not contain horizontal tab characters. Use blanks only for indentation.

3. No line may contain trailing blanks.

4. Do NOT put whitespace:
    1. Around the "<" and ">" within a generic type designation ("List", not "List ", or "List< Integer >").
    2. After the prefix operators "!", "−", "++", unary "-", or unary "+".
    3. Before the tokens ";" or the suffix operators "−" and "++".
    4. After "(" or before ")".
    5. After "."
    6. When you are lonely.
5. DO put whitespace:
    1. After ";", ",", or type casts (e.g., "(String) x", not "(String)x").
    2. Around binary operators (e.g., "*", "+") and comparison operators.
    3. Around assignment operators (e.g., "=", "+=").
    4. Around "?" and ":" in the ternary conditional operator ("x>0 ? x : -x").
    5. Around the keywords "assert", "catch", "do", "else", "finally", "for", "if", "return", "synchronized", "try", and "while".
6. In general, break (insert newlines in) lines before an operator, as in

    ```
    ... + 20 * X
        + Y;
    ```

7. Do not separate a method name from the "(" in a method call with blanks. However, you may separate them with a newline followed by blanks (for indentation) on long lines.

Indentation

1. The basic indentation step is 4 spaces.

2. Indent code by the basic indentation step for each block level (blocks are generally enclosed in "{" and "}"), as in

```
if (x > 0) {
    r = -x;
} else {
    r = x;
}
```

3. Indent 'case' labels an indent past their enclosing 'switch', as in

```
switch (op) {
    case '+':
        addOpnds(x, y);
        break;
    default:
        ERROR();
}
```

4. Indent continued lines by the basic indentation step.

Braces

1. Use { } braces around the statements of all 'if', 'while', 'do', and 'for' statements.

2. Place a "}" brace on the same line as a following "else", "finally", or "catch", as in

```
if (x > 0) {
    y = -x;
} else {
    y = x;
}
```

3. Put the "{" that opens a block at the end of a line. Generally, it goes at the end of the "if", "for", "while", "switch", "do", method header, or class header that contains it. If line length forces it to the next line, do not indent it, and put it alone on the line.

Comments

1. Methods should have javadoc comments explaining the behavior, parameters (using @param tags or otherwise), and return type.

2. Methods that return non-void values must describe them in their Javadoc comment either with a "@return" tag or in a phrase in running text that contains the word "return", "returning", or "returns".

3. Each Javadoc comment must start with a properly formed sentence, starting with a capital letter and ending with a period.

Names

1. Names of static final constants must be in all capitals (e.g., RED, DEFAULT_NAME).

2. Names of parameters, local variables, and methods must start with a lower-case letter, or consist of a single, upper-case letter.

3. Names of types (classes), including type parameters, must start with a capital letter.

4. Names of packages must start with a lower-case letter.

5. Names of instance variables and non-final class (static) variables must start with either a lower-case letter or "_".

Imports

1. Do not use 'import PACKAGE.*', unless the package is java.lang.Math, java.lang.Double, or org.junit.Assert. 'import static CLASS.*' is OK.

2. Do not import the same class or static member twice.

3. Do not import classes or members that you do not use.

Assorted Java Style Conventions

1. Write array types with the "[]" after the element-type name, not after the declarator. Write "String[] names", not "String names[]".

2. Write any modifiers for methods, classes, or fields in the following order:
   1. public, protected, or private.
   2. abstract or static.
   3. final, transient, or volatile.
   4. synchronized.
   5. native.
   6. strictfp.

3. Do not explicitly modify methods, fields, or classes where the modification is redundant:
   1. Do not label methods in interfaces or annotations as "public" or "abstract".
   2. Do not label fields in interfaces or annotations as "static", "public", or "final".
   3. Do not label methods in final classes as "final".
   4. Do not label nested interfaces "static".

4. Do not use empty blocks ('{ }' with only whitespace or comments inside) for control statements. There is one exception: a catch block may consist solely of comments having the form

```
/* Ignore EXCEPTIONNAME. */
```

5. Avoid "magic numbers" in code by giving them symbolic names, as in

```
public static final MAX_SIZE = 100;
```

Exceptions are the numerals -10 through 10, 0.5, -0.5, 0.25, -0.25.

6. Do not try to catch the exceptions Exception, RuntimeError, or Error.

7. Write "b" rather than "b == true" and "!b" rather than "b == false".

8. Replace

```
if (condition) {
    return true;
} else {
    return false;
}
```

with just

```
return condition;
```

9. Only static final fields of classes may be public. Other fields must be private or protected.

10. Classes that have only static methods and fields must not have a public (or defaulted) constructor.

11. Classes that have only private constructors must be declared "final".

Avoiding Error-Prone Constructs

1. If a class overrides "equals", it must also override "hashCode".

2. Local variables and parameters must not shadow field names. The preferred way to handle, e.g., getter/setter methods that simply control a field is to prefix the field name with "_", as in

```
public double getWidth() {
    return _width;
}

public void setWidth(double width) {
    _width = width;
}
```

3. Do not use nested assignments, such as "if ((x = next()) != null) …". Although this can be useful in C, it is almost never necessary in Java.

4. Include a "default" case in every "switch" statement.

5. End every arm of a "switch" statement either with a "break" statement or a comment of the form

```
/* fall through */
```

6. Do not compare String literals with "==". Write

```
if (x.equals("something"))
```

and not

```
if (x == "something")
```

There are cases where you really want to use "==", but you are unlikely to encounter them in this class.

Limits

1. No file may be longer than 2000 lines.

2. No line may be longer than 100 characters.

3. No method may be longer than 80 lines.

4. No method may have more than 8 parameters.

5. Every file must contain exactly one outer class (nested classes are OK).