# Coding Standards for Java language

## Comment Rules

1. Every source file (.java and .class files) must include a comment at the beginning of the file that briefly describes the purpose of the source file and which lists the author and the date.

2. Every function, except the main function, must be preceded by a block comment that explains a) the purpose of the function; b) the purpose of each argument to the function; c) what the function returns (if anything).

3. Variables should be declared at the top of a function or block. Each variable should have a brief comment explaining its intended use.

4. Use comments within a function only when you feel that the code alone is not self-explanatory. More comments does not usually mean better style.

5. Use // for comments

## Naming Rules

1. Names for variables (as well as functions and other identifiers) should be meaningful, in order to help the reader understand the purpose of the variable.

2. Variable names should be formatted as follows

| Type | Format |
|---|---|
| Variables | camelCase |
| Class/Enum/Methods | PascalCase |
| Constants | UPPERCASE_SNAKE_CASE |

4. Avoid using single letters for variable names, except for loop variables or other temporary variables, or if you're implementing a formula where the equation variables are single letters.

5. Avoid really long variable names, which are also difficult to understand.

6. Avoid using abbreviations unless the abbreviation is commonly understood or obvious e.g. vat or gpa.

7. Plural form should be used on names representing a collection of objects.

8. Iterator variables can be called *i*, *j*, *k* etc.

Variables named *j*, *k* etc. should be used for nested loops only.

9. Associated constants should have a common prefix.

```
static final int COLOR_RED   = 1;

static final int COLOR_GREEN = 2;

static final int COLOR_BLUE  = 3;
```

## Code Format Rules

1. All code must use consistent indentation to show the logical structure. Basic indentation should be one tab. Indentation for wrapped lines should be two tabs.

2. Don’t use “running dog” format

3. Long code lines are hard to read and understand. If a line of code is longer than 90 characters, break it into two lines.

4. Long functions are hard to read and understand. If a function is longer than about 60 lines of code, extract parts of the code into sub-functions

5. If-else statement should have the following form:

```
if (condition)
{
   statements;
}
else
{
   statements;
}
```

6. For statements should have the following form:

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

7. The *while* and the *do-while* statements should have the following form:

```
while (condition)
{
  statements;
}

Do
 {
  statements;
}
while (condition);
```

8. The *switch* statement should have the following form

```
switch (condition)

{

        case ABC:

                statements;

                // Fallthrough

        case DEF:

                statements;

                break;

        case XYZ:

                statements;

                break;

        default:
                statements;

                break;
}
```

The explicit //Fallthrough comment should be included whenever there is a case statement without a break statement.

Rationale: Leaving out the break is a common error, and it must be made clear that it is intentional when it is not there.

9. Operators should be surrounded by a space character. ex. a = (b + c) * d;

10. Commas should be followed by a white space. ex. doSomething(a, b, c, d);

11. Colons should be surrounded by white space when used as a binary/ternary operator.
Does not apply to switch x:. Semicolons in for statements should be followed by a space character. Ex. for (i = 0; i < 10; i++)
{
}

12. Logical units within a block should be separated by one blank line.

## Other coding considerations

1. Use parentheses to clarify the meaning of your expressions.
For example:
Poor Style
if (age >= 65 || memberNumber > 0) ...
result = value1 + value2/count – value3 % 7;
Better Style
if ((age >= 65) || (memberNumber > 0))...
result = value1 + (value2/count) – (value3 % 7);

2. Think carefully before you use continue and break. Avoid using more than one break statement in a particular loop

## References

1. Oracle. (n.d.). *Java Code Conventions*. Retrieved from https://www.oracle.com/docs/tech/java/codeconventions.pdf
2. Google. (n.d.). *Google Java Style Guide*. Retrieved from https://google.github.io/styleguide/javaguide.html
3. SE-EDUCATION. (n.d.). *Java Coding Conventions: Intermediate*. Retrieved from https://se-education.org/guides/conventions/java/intermediate.html