The Choice Awards Style Guide
Group 4 - CSC 131

Java
Formatting

Spacing:

Each indent is to use 4 spaces. Set tab width to 4 spaces
There should be 1 space between end of parenthesis and brace.
When the block ends, the indent returns to the previous indent level. The indent level applies to both code and comments throughout the block

```
while (1) {
    while (1) {
        return;
    }
}
```

Spaces between operations such as + and = should have a space between variables and operators.

```
int a = b * c - d;
```

There is no requirement for horizontal spaces, just please make it logical and neat.

Braces:

Use of braces are hanging with one space after parenthesis

```
if (true) {
    do();
}
```

Braces are always used in statements even when optional: i.e. if, else, for, do, while.

```
//Do not do
      while (true)
            return;

//Do
      while (true) {
            return;
      }
```

```

```

Empty blocks can be concise if desired.

```
//This is fine
void foo() {}

//this is also fine
void foo() {
}

//this is also fine
void foo() {

}
```

Line Wrapping:

Line wrapping for if statements should generally use the 8-space rule

```
//DON'T USE THIS INDENTATION
if ((condition1 && condition2)
    || (condition3 && condition4)
    ||!(condition5 && condition6)) { //BAD WRAPS
    doSomethingAboutIt();           //MAKE THIS LINE EASY TO MISS
}

//USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
//OR USE THIS
if ((condition1 && condition2) || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

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.
- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that's squished up against the right
  margin, just indent 8 spaces instead.

Here are some examples of breaking method calls:

```
function(longExpression1, longExpression2, longExpression3,
         longExpression4, longExpression5);

var = function1(longExpression1,
                function2(longExpression2,
                          longExpression3));
```

Statements

Simple Statements

Each line should contain at most one statement.

```
argv++; argc--; // AVOID!
```

Do not use the comma operator to group multiple statements unless it is for an obvious reason.

```
if (err) {
    Format.print(System.out, "error"), exit(1); //do not do
}
```

Compound Statements

Compound statements are statements that contain lists of statements enclosed in braces “{ statements }”. See the following sections for examples.

• The enclosed statements should be indented one more level than the compound statement.

• The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement.

• Braces are used around all statements, even singletons, when they are part of a control structure, such as a if-else or for statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.

return Statements

A return statement with a value should not use parentheses unless they make the return value more obvious in some way.

```
return;
return myDisk.size();
return (size ? size : defaultSize);
```

if, if-else, if-else-if-else Statements

The if-else class of statements should have the following form:

```
if (condition) {
    statements;
}

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

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

Note: if statements always use braces {}. Avoid the following error-prone form:

```
if (condition) //AVOID! THIS OMITS THE BRACES {}!
    statement;
```

for Statements

A for statement should have the following form:

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

An empty for statement (one in which all the work is done in the initialization, condition, and
update clauses) should have the following form:

```
for (initialization; condition; update);
```

When using the comma operator in the initialization or update clause of a for statement, avoid
the complexity of using more than three variables. If needed, use separate statements before
the for loop (for the initialization clause) or at the end of the loop (for the update clause).

while Statements

A while statement should have the following form:

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

do-while Statements

A do-while statement should have the following form:

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

switch Statements

A switch statement should have the following form:

```
switch (condition) {
case ABC:
    statements;
    /* falls through */
case DEF:
    statements;
    break;
case XYZ:
    statements;
    break;
default:
    statements;
    break;
}
```

Every time a case falls through (doesn't include a break statement), add a comment where the break statement would normally be. This is shown in the preceding code example with the /* falls through */ comment.

Every switch statement should include a default case. The break in the default case is redundant, but it prevents a fall-through error if later another case is added.

try-catch Statements

A try-catch statement should have the following format:

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}
```

Naming

Variable names:

Variable names should avoid using abbreviations and use descriptive names for variables.

```
//Avoid
int a, d;
a = c(d);

//Do
int returnedData, date;
returnedData = functionMath(date);
```

Naming for variables and functions should be first name lowercase, trailing names uppercased.

```
int firstVariableName, secondVariableName;

void mathFunction();
void splitFunction();
```

Naming for classes should have each word of the class name capitalized.

```
public class ThisIsAClass {

}
```

Comments

Types of comments:

Single line comments can either be

```
// This is fine
Or
/* This is fine */
```

Block comment styles can either be

```
// This is
// fine

Or

/*
    This is
    fine
*/

Or

/* This is
 * fine    */

Or

/*
* This is
* fine
*/
```

Top of the file:

Please put the name of the programmer(s), date, and purpose of the file at the top of the file with a block comment.

```
/*
Author(s):
Date:
Purpose of file:
*/
```

Above line comment for functions and classes:

Above each function and class provide a quick description of what each function or class does

```
//This class is used to parse data
public class parsing {

    //This function will take a given string and split it at a certain spot
    public static void splitString(string values){
        . . . stuff . . .
    }
}
```

Inline comments:

If the code is difficult to follow or you want to mention where certain variables come from and go, in line comments at the end of a statement are appreciated.

```
if(true) {
    data[1] = getValue(exampleFunc(score)); //returns the value of score into array
}
```