CS 1331 Java Style Guide

Style is very important when writing code. Often code is not judged simply base on how effectively it performs a specific function, but also how easily it can be understood by a human reader. While rules of style vary, it is important to be able to learn to write code which conforms to a given standard.

Checkstyle

In this class, the style rules that you are asked to follow are crystallized into an automated tool, called Checkstyle. This tool is configurable to check for a large number of different stylistic errors. Additional details about how to run checkstyle on your code can be found here.

The file you need to run Checkstyle:

- checkstyle-6.2.2.jar

To run **checkstyle** on all Java files in the **current directory**:

```
java -jar checkstyle-6.2.2.jar *.java
```

To check the **Javadocs** in all Java files in the **current directory**:

```
java -jar checkstyle-6.2.2.jar -j *.java
```

To fully check your code you will need to run both checks above.

To see all available checkstyle options run `java -jar checkstyle-6.2.2.jar` without specifying a file to check.

Style Guidelines

The rest of this document summarizes and in some cases clarifies Sun/Oracle's Java Code Conventions (and in some places copies it directly). The Checkstyle web site also documents each of its standard checks here: http://checkstyle.sourceforge.net/checks.html

Names

- Class names should be capitalized camel case, as in `ClassName`.
- Method and variable names should be camel case with a lower case first letter, as in `variableName`. -Constants (`final` variables) should be all caplital letters with words separated by underscores, as in `CONSTANT_NAME`.
- Names should be descriptive, but not excessively long. Avoid single-letter variable names. Exception: loop variables are typically `i`, `j`.
- Example:

```
public class ClassName {
    public static final int SOME_CONSTANT;

    private int instanceVariable;

    public int getInstanceVariable() {
        return instanceVariable;
    }
}
```

Indentation and Line Length

- Each scope should be indented one level deeper than the previous level.
- The class declaration is not indented.
- Method names are indented one level, statements within the method by two levels, and so on.
- Indent with 4 spaces. Do not use TAB characters.
- Avoid lines longer than 80 characters.
- Use the following guidelines to break long lines:
  - Break after commas and before operators.
  - Align method parameters, or if the first line is too long indent the second and subsequent lines by 8 spaces. For example:

```
//CONVENTIONAL INDENTATION
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
           Object andStillAnother) {
    ...
}

//INDENT 8 SPACES TO AVOID VERY DEEP INDENTS
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother) {
    ...
}
```

- Use 8-spaces to indent subsequent line of multiple-line if statement conditions to help distinguish the condition from the body. For example:

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

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

- Indent ternary expressions as in these examples:

```
alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

alpha = (aLongBooleanExpression)
         ? beta
         : gamma;
```

Declarations and Statements

- Avoid putting more than one statement or declaration on a single line. Prefer

```
int width;
int height
```

to

```
int width, height;
```

- Avoid declaring variables of different types on the same line, as in:

```
int foo,  fooarray[]; //WRONG!
```

- If possible, initialize a variable where it is declared.
- Place declarations at the beginning of blocks.
- Do not put spaces between parentheses, "(" and ")", and parameter lists.
- Open brace "{" appears at the end of the same line as the declaration statement
- Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{"
- Separate methods with a blank line.
- Example:

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int emptyMethod() {}
    ...
}
```

- Format compound statements as in the following examples:

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

for (it i = 0; i < 10; i++) {
    statements;
}

while (condition) {
   statements;
}

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

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

Javadoc Comments

- Write Javadoc comments for the class and all public members of a class.
- Don't write Javadoc comments for private members.
- Write non-Javadoc comments for private members as necessary to explain aspects of the code that can't be expressed by the code itself. Such cases should be rare because your code should be self-documenting.
- A class comment describes the class's purpose.
- Inlcude an `@author` tag in all of your class comments with your Prism ID as your name.

Here's a small example demonstrating proper Javadoc commenting:

```
import java.time.LocalDate;
import java.time.Month;

/**
 * Represents an employee who is paid an hourly wage.
 *
 * @author your name
 * @version 13.31
 */
public class HourlyEmployee extends Employee {

    private double hourlyWage;
    private double monthlyHours;

    /**
     * Creates an HourlyEmployee with hourly wage of 20 and
     * monthly hours of 160.
     *
     * @param aName the employees offical full name
     * @param aHireDate the date on which the employee was hired
     */
    public HourlyEmployee(String aName, LocalDate aHireDate) {
        this(aName, aHireDate, 20.00, 160.0);
    }

    /**
     * Creates an HourlyEmployee with all required parameters.
     *
     * @param aName the employees offical full name
     * @param aHireDate the date on which the employee was hired
     * @param anHourlyWage this employee's non-overtime hourly wage
     * @param aMonthlyHours the number of hours this employee must work every
     *                      month
     */
    public HourlyEmployee(String aName, LocalDate aHireDate,
                          double anHourlyWage, double aMonthlyHours) {
```

```
        super(aName, aHireDate);
        ValidationUtils.disallowZeroesAndNegatives(anHourlyWage,
                                                   aMonthlyHours);
        hourlyWage = anHourlyWage;
        monthlyHours = aMonthlyHours;
    }

    /**
     * @return this employee's official full name
     */
    public String getName() {
        return "Hourly: " + super.getName();
    }

    /**
     * Calculates the monthly pay of this employee by mutiplying the
     * employee's hourly wage and monthly hours.
     *
     * @return this employee's monthly pay
     */
    public double monthlyPay() {
        return hourlyWage * monthlyHours;
    }
...
}
```