Coding Conventions for Team Wonderful Code

1. Introduction

1.1. Purpose

1.2. References

2. Formatting

3. Capitalization

4. Naming

4.1. Local Variables

4.2. Mutators & Accessors

4.3. Methods

4.4. Classes

4.5. Interfaces

5. Commenting

5.1. Formatting

5.1.1. Example

5.2. Block Tags

5.3. When to Comment

5.4. Comment Example

6. Testing

6.1. Test Package

6.2. Test Class

6.3. Test Method

1 Introduction

1.1 Purpose

This document is to establish a consistent coding standard for Team Wonderful Code. The following information will be specific to the Java language and will be agreed upon by all members of the team. For cases not covered by this document, refer to the guidelines given by Google’s Java guide (link available in the References section).

1.2 References

The following sources were used in compiling this document:

<https://google.github.io/styleguide/javaguide.html>

<https://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>

2 Formatting

Covers conventions of formatting.

2.1 Curly Brackets

Curly brackets will open on the same line, and close on the line following the content, unless specified otherwise.

Example:

public static void exampleMethod() {

if (true) {

} else {

}

}

3 Capitalization

3.1 Local Variables

All local variables will be formatted in camelCase.

3.2 Methods

All methods will be formatted in camelCase.

3.3 Classes

All classes will be formatted in PascalCase.

3.4 Packages

All packages will be lowecase\_seperated\_by\_underscores..

4 Naming

4.1 Local Variables

All local variables should be a noun describing what data they contain.

· Example: An integer that keeps track of a user’s location in an index should be named *int currentIndex*

4.2 Mutators & Accessors

All mutators and accessors will be prefaced by get or set respectively, followed by the variable that is being retrieved/modified.

4.3 Methods

The first word of all methods should be a verb describing its action followed by the objects which it will be acting upon.

· Example: A method that returns *string firstName* and *string LastName* concatenated should be named concatenateFirstNameAndLastName()

4.4 Classes

Class names should be a noun or noun phrase describing its purpose.

4.5 Interfaces

Interfaces will primarily be an adjective or adjective phrase, though may be a noun or noun phrase if applicable.

5 Commenting

5.1 Formatting

Comments will be formatted in Javadoc blocks. They are started with *\*//*

5.1.1 Examples

/\*\*

\* Comment blocks are typically detailed.

\*

\*/

/\*\* Sometimes they can be short, though. \*/

5.2 Block Tags

Use the following tags where applicable, ordered descending on the table.

|  |  |
| --- | --- |
| @param | Used to describe parameters |
| @return | Used to describe return values of method |
| @throws | Used to describe exceptions |
| @deprecated | Used to show an element should not be used |

5.3 When to Comment

Comments will only be used for classes and methods. Variables should be named so that a comment is not necessary, and comments describing the last time code was changed will be documented by the code repository (GitHub). If a comment is required inside a method to describe what it does, consider refactoring into a separate method or a simpler/more readable solution.

5.4 Comment Example

*/\*\* A Class that can perform mathematical functions.\*/*

*Public class MathFunctions{*

*/\*\**

*\* Adds two positive integers and returns the sum.*

*\**

*\* @param numOne the first integer to add*

*\* @param numTwo the second integer to add*

*\* @return the sum of the two numbers*

*\* @throws InvalidArgumentException if int1 is negative*

*\* @throws InvalidArgumentException if int2 is negative*

*\*/*

*public void addPositiveNumbers(int num1, int num2) {*

*return num1 + num2;*

*}*

*}*

6 Testing

6.1 Test Package

For each package, another test package will be created with the same name, followed by ‘.tests’

· Example: For the package business\_logic, a package business\_logic.tests will be created

6.2 Test Class

A test class will be the name of a class followed by Tests.

· Example: The test class for class *MathFunctions* will be named *MathFunctionsTests*.

6.3 Test Method

Each method test will be *test*, followed by the name of the method being tested.

· Example: The test method for method *addPositiveNumbers* will be *testAddPositiveNumbers*

If the method has multiple branches, use an underscore to describe the branch being tested.

· Example:

public boolean isPositive(int value) {

return i >= 0

}

There are two branches that would need to be tested in this case:

testIsPositive\_valueIsPositive()

testIsPositive\_valueIsNotPositive()