Contents

[Software Quality Assurance 2](#_Toc5979721)

[Coding Standards 2](#_Toc5979722)

[Comments 2](#_Toc5979723)

[Package and Import Statements 2](#_Toc5979724)

[Directory structure 2](#_Toc5979725)

[Indentation 2](#_Toc5979726)

[Line Length 2](#_Toc5979727)

[Wrapping Lines 3](#_Toc5979728)

[Identifiers 3](#_Toc5979729)

[Classes 3](#_Toc5979730)

[Methods and Variables 3](#_Toc5979731)

[Simple Conventions Code Examples 3](#_Toc5979732)

[If and If-else Statements 3](#_Toc5979733)

[For Statements 3](#_Toc5979734)

[If, Else, While and For Statements Examples 4](#_Toc5979735)

[Switch Statements 4](#_Toc5979736)

[Parameters and Arguments 4](#_Toc5979737)

[Parameters and Arguments Coding Examples 4](#_Toc5979738)

[Further Reading for Coding Conventions 4](#_Toc5979739)

[Unit Testing 5](#_Toc5979740)

[Jenkins 5](#_Toc5979741)

[JUnit Naming Conventions 6](#_Toc5979742)

[Test Data 6](#_Toc5979743)

[Version Control 6](#_Toc5979744)

[Git 6](#_Toc5979745)

[Jira 7](#_Toc5979746)

# Software Quality Assurance

This document will be used for companywide coding standards, a guide for unit testing, and version control. It will be made using Java. Eclipse should be used for all coding to keep teams working on the exact same page.

## Coding Standards

### Comments

* At the opening of every class a comment should be made showing the class name, version and author.

/\*

\* Class name

\*

\* Version info

\*

\* Author

\*

\*/

* Comments in code should be used when necessary, they should only be used when the code isn’t specific enough to reads without them.
* All comments and blocks of comments should be separated from other code by one line.

### Package and Import Statements

* The package name should be the first thing written in the class after comments.
* Imports should be after the package name; one line should be skipped between them.

### Directory structure

* In directories project names must be capitalised and must be named related to what the project will be used for.
* Packages should be used to group relevant classes together.

### Indentation

* For indentation tabs will be used as a standard.
* All brackets will be inserted a single space, on the same line as the code they follow for opening brackets, closing brackets will be closed with the same indentation as they are opened with.
* All “(“ and “)” brackets will have a space in-between whatever is inside them.
* Semi colons will be used without a space between the code they come after. Examples of these can be seen below.

### Line Length

* Lines should be no longer than 75 characters in length.

### Wrapping Lines

* When lines are too long to be placed on a single line, they it should be broken up by a comma or before an operator.
* The new line should be aligned with where the previous line started. An example of this is in the repository below.

### Identifiers

* Identifiers should be descriptive of their use.
* Class names will usually be nouns explaining what the class is for.
* Variables should be formatted beginning with a lower-case letter, if it is two words it should be concatenated into one, the second word will have it’s first letter capitalised. For example, a variable should be named “secondName”.
* For the name of classes, they should be named the same way as variables but with a capitalised first letter like so “TheClass”.
* When declaring constant variables, they should be in all capitals with an underscore separating them: “private final String FIRST\_NAME”.

### Classes

* A class will be undented to the far left and brackets will be opened on the same line as it is needed, as seen in the code repository below.
* When using inheritance or an interface, it should be on the same line as the class name.

### Methods and Variables

* Methods and variables will follow the rule in which every time brace is used an indentation will follow as seen in the code below.
* There will also be a space before and after the “=” when variables are declared. Variables should not be declared on the same line.

### Simple Conventions Code Examples

<https://github.com/edwardmk/SoftwareQualityAssurance/blob/master/BasicConventions/src/simpleCodingConventions/SimpleConventions.java>

### If and If-else Statements

* If and if statements will follow previous instructions, else statements will begin on the next line after the if statement is finished. An example will follow below.

### For Statements

For statements should have the following layout:

for ( initialisation; condition; update ) {

statement;

}

### If, Else, While and For Statements Examples

<https://github.com/edwardmk/SoftwareQualityAssurance/blob/master/BasicConventions/src/simpleCodingConventions/IfWhileForStatments.java>

### Switch Statements

Switch statements should work with the same indentation as said before, like this:

**int** month = 2;

String monthString;

**switch** (month) {

**case** 1: monthString = "January";

**break**;

**case** 2: monthString = "February";

**break**;

}

<https://github.com/edwardmk/SoftwareQualityAssurance/blob/master/BasicConventions/src/simpleCodingConventions/SwitchStatment.java>

### Parameters and Arguments

* Arguments should be written: foo.car( 5 );
* and parameters should be written: public int add( int i );

### Parameters and Arguments Coding Examples

<https://github.com/edwardmk/SoftwareQualityAssurance/blob/master/BasicConventions/src/simpleCodingConventions/ParametersAndArguments.java>

### Further Reading for Coding Conventions

<https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

## Unit Testing

* When it comes to unit testing the main principals are a “test first” principal should be followed.
* JUnit 4 will be the framework to be followed throughout the testing process and tests should be automated as often as possible to increase efficiency.
* JUnit classes should only be used for tests and all tests inside the class should be annotated with the @Test annotation.
* Assert statements should provide useful messages to provide clarity when looking at the tests.
* Tests should be made in a separate package to classes that need to be tested.
* Apache Maven in conjunction with Jenkins will be used for build automation.
* In the pom.xml file add the following lines:

<dependency>  
<groupId>junit</groupId>  
<artifactId>junit</artifactId>  
<version>4.12</version>  
<scope>test</scope>  
</dependency>

For Gradle, add the following to the build.gradle:

apply plugin: 'java'  
  
dependencies {  
testCompile 'junit:junit:4.12'  
}

### Jenkins

* When building and rebuilding branches always make sure that everyone else is aware of the branch being built using the team google hangouts.
* Nightly builds will be built every night at midnight to get new code from the git repository.
* Any nightly conflicts will be resolved the following day.
* Any failure of builds should be reported to the appropriate person.
* All Jenkins components will be named following this convention: $namespace\_$service/$feature/$util.

$namespace, $service, $feature, $util can contain alphanumeric characters (lower case only) along with hyphen (-).

The naming of jobs will follow these examples:  
component\_user-services  
component\_build-pack-function  
component\_build-pack-website  
component\_build\_pack\_api  
component\_cleanup\_cloudfront\_distributions  
component\_create-service  
component\_delete-service  
component\_core-services

### JUnit Naming Conventions

* Tests will be named using MethodName\_StateUnderTest\_ExpectedBehavior as a guide to follow. For example: isNewCustomer\_NewCustomer\_True.
* The beginning should be the name of the method being tested, the middle should be what is being tested and the end should be the expected outcome of the test.

### Test Data

* Unit tests should all be able to run independently of each other, meaning one test’s result should not affect another test.
* A base set of tests should be created, all tests should rely on the base set of tests to be created. When a test needs to change data in the class it should create it itself, then delete the newly created data once the test is finished.

## Version Control

### Git

* A basic guide to Git can be found here <http://rogerdudler.github.io/git-guide/>
* Git will be used for version control.
* For naming files when committing they will follow the naming conventions of being all lower case and separating words by a hyphen. For example, purchase-rest-service.
* Every time a change is made it should be added to the repository, no matter how small.
* Git should be set up with Eclipse to make using git easier.

#### Committing

* When committing to git you must have your code peer reviewed by at least 3 people on your team.
* Before committing to the master branch you should fetch from the repository.
* Use git add before using git commit.
* When committing to the master branch you should inform the team so as to not interrupt their work.

#### Git Message

* When committing to the git repository a message must be committed with it, it must follow these seven conventions.
* Separate subject from body with a blank line.
* Limit the subject line to 50 characters.
* Capitalize the subject line.
* Do not end the subject line with a period.
* Use the imperative mood in the subject line.
* Wrap the body at 72 characters.
* Use the body to explain what and why vs. how.
* Further reading can be found here: <https://chris.beams.io/posts/git-commit/>

### Jira

* Jira should be used to track all bug fixes, bug and all tests that need to be made.
* Jira will house the documentation of each project; each project will have a set of tests that need to be developed for each project.
* Any time a document is changed it must be added to the Jira.
* Every test will be made into an issue on Jira, when it’s complete it can be resolved.