Gareth landy

C15735901@mydit.ie

Software Quality Assurance

Table of Contents

[Introduction 2](#_Toc6000156)

[Guide Notes 2](#_Toc6000157)

[Java File Standards 3](#_Toc6000158)

[Variables 3](#_Toc6000159)

[Method Standards 4](#_Toc6000160)

[Structure of Source Files 5](#_Toc6000161)

[Comments 5](#_Toc6000162)

[Packages 5](#_Toc6000163)

[Unit Testing 6](#_Toc6000164)

[Reasons for Unit Testing 6](#_Toc6000165)

[Unit Testing Examples 6](#_Toc6000166)

[Test Method Requirements 7](#_Toc6000167)

[Write Tests to Pass 8](#_Toc6000168)

[Culture of Unit Testing 8](#_Toc6000169)

[Version Control 9](#_Toc6000170)

[Branching Strategies 9](#_Toc6000171)

[Git Feature Branch Workflow 9](#_Toc6000172)

[Feature Branch in motion 9](#_Toc6000173)

[Merge Strategies 10](#_Toc6000174)

[Test before Commit 10](#_Toc6000175)

[Avoid Committing Half Completed Work 11](#_Toc6000176)

[Agreed Workflow 11](#_Toc6000177)

[Best Practices 11](#_Toc6000178)

[Conclusion 11](#_Toc6000179)

## Introduction

This document serves as a definition of the coding standards that can be used when coding in the programming language of Java. Throughout this document there will be a reference to a GitHub link that will show the user of the coding practices that are examples but are to illustrate the style guide that is being implemented.

## Guide Notes

All coding examples in this document are found here : <https://github.com/garethlandy/SoftwareQualityAssuranceAssignment>

Each example that is in a diagram will have a relevant link to direct GitHub page to view.

As new members join the team it is important that there is one “right way” to do things. This approach will benefit the applications that we are producing by keep the same strategy across all platforms. In doing this we can create a culture of inclusion while also setting the highest standards within software development.

[Section 1](#_Java_File_Standards) will include an introduction to the coding standards that are required by all developers to maintain the highest standards when dealing with code across all platforms.

[Section 2](#_Unit_Testing) is concerned with Unit Testing. Here you will find examples of code that have been written and shared on the repository. Each test that has be configured has examples where the test has passed or failed for comparison.

[Section 3](#_Branching_Strategies) Version Control: The branching and merging strategies that will be implemented will be found within this document which contain links to GitHub annotations that will be used for future reference.

## Java File Standards

Each new Class file will have a .java extension and will start with a capital Letter and each new concatenated word will start with an upper-case letter EG:

SoftwareAssurance.java

Below are some variations but discouraged .java file class names:

Software\_Assurance.java

softwareAssurance.java

Each new class should be named after its purpose to improve and maintain its readability. This will enable any user to quickly find a class without having to scan through the class to view its contents. If the class is a Test class and not yet ready for production, please ensure the what is being tested is added before the test classes definition IE: UserTest.java

## Variables

Each new variable entered should be given a short and meaningful name where possible. They must also be on a separate line to make it easier to find and read. Each new variable should have these characteristics:

* Do not start with an underscore (‘\_’) or dollar ‘$’ characters.
* The user should be able to read it with an indication of its use.
* One-character names should be avoided unless its use is for a temporary variable.
* Some common temporary variables include ‘i’, ‘j’ for integers and ‘c’ for String Characters .
* Always avoid using acronyms of words.
* Variable character maximum length should be less than 80.

When declaring a constant variable in any program all characters must be in uppercase with each new word separated by an underscore (‘\_’).

* private static final String ROOT\_URL = “https://123.321.23.2/mysql /users.php”;

## Method Standards

Each new method should begin with a lowercase letter and each subsequent word that is contained in the method name should start with an uppercase which is typically described as [lowerCamelCase](https://techterms.com/definition/camelcase). A verb is typically used here to perform a task on a method EG:

* void print ()
* void run (int start)
* void myMethod (String name)

Java Bean coding standards which are java classes that include private properties with public getter and setter methods are to be set in the following ways,

Getter methods should:

* be a public method
* always be prefixed with the word “get”
* not take in any arguments

Setter methods should:

* be a public method
* be given a return type of void
* always be prefixed with the word “set”
* take in some argument

<https://github.com/garethlandy/SoftwareQualityAssuranceAssignment/commit/5d21a4f1e7c57c2b41a8463863f073e61030acfa#diff-54cf17d8cf2f2e0b9f25b0a168512f06>

## Structure of Source Files

Any new java source files must include in order:

Comments on the author or the licence agreement holder of the product.

The package name (if applicable) must be named as the top line of the class/interface structure.

All necessary import statements to use throughout the viewed class only.

Only one top level class.

All source file sections must be at least separated by one line. This approach helps to keep the code as readable as possible in larger class files.

## Comments

Here are 3 different types of comments that can be used across all java classes. Block comments are used to give the user additional information or to obstruct a piece of code to be complied at run time. That said any piece of code that is not reachable should be garbaged collected unless it is intended for future use within the program. While it is always ok to leave comments on a section of code within any class, it is imperative that the comment is readable as too help any end user understand why that section is being left out or what is happening in the system at a particular moment in time. Precise and definite language is encouraged here.

/\*

\* This is multi-line // Another /\* Comment

\* Comment // Comment \* 3 \*/

\*/

## Packages

All package declarations are to be set in lowercase with words concatenated together to simply define its current state. Do not use the camelCase naming convention or underscores when naming your package.

EG: com.tudublin.qualitysoftware , not com.tuDublin.quality\_software

Under the previous six heading we have outlined the basic coding standards that should be adhered too at all times. These standards are to ensure that each developer, writing source code when in large scale projects, is consistent in their approach which will make it easier to understand. Each large software firm will have a set of standards that their developers must use and by insisting on these standards it makes the source code easier to read, line by line.

Further reading @ <https://blog.fossasia.org/why-coding-standards-matter/>

## Unit Testing

Definition:

To test individual components of a software program during the lifecycle of the application. Its objective is to pinpoint or isolate a section of this application and to verify that the intended results match what the user was expecting. In the Software Development Life Cycle (SDLC)unit tests are the first tests that a developer will implement.

## Reasons for Unit Testing A close up of a device Description automatically generated

1. Early bug fixes which will save on both time and costs

2. Alter the code in early stages of the life cycle so to understand the developers underlying code base and thinking

3. When using documentation, runnable unit tests can be supported for a better understanding of the project

4. By making small alterations this enables the developer to use the tests in different programs for better code reuse

## Unit Testing Examples

Unit test requires on a set of objects that must be created by the user that are not currently part of the application (Substitutions). Theses substitute variables or objects are used to create test methods or test cases where by the user can test the return from a method to get desired results.

In this case we will be using the jUnit Testing Tool. It is used in Java programming and provides an uncomplicated way of handling and running repeatable tests. jUnit also helps to find bugs early in the code before much larger application dependencies are incorporated into the program. The user will get two results from a test, Passed or Failed.

A screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generated

In the first diagram the test has failed as the expect value from the test did not meet the actual value that was expected when the method was called. The stack trace at the end of the screen shows what the expected value should be, and which method has failed.

See : <https://github.com/garethlandy/SoftwareQualityAssuranceAssignment/commit/87f244228ae072d165e1a3b5da65bb7a4bcb318b#diff-64e2794d289146ccc167adf61928e6e6>

The second diagram show that the test has passed. The completed code with other test cases can be found here : <https://github.com/garethlandy/SoftwareQualityAssuranceAssignment/commit/1ded6b42fb4affc72298ccb5cfbf44e40eda41fb>

## Test Method Requirements

A test method should conform to these 3 standards.

* These methods should never be private or static
* These methods should be preceded by the @Test annotation
* These methods should be void where nothing is returned

Other annotations that can be used to streamline the process of understanding this concept at run time are @BeforeClass, @AfterClass, @Before, @After. These annotations can be used to check the current state of a function before or after the test has been run. It will help you find errors much faster as the functions state can be used to find the desired results once the test has finished.

While unit testing is focusing on an isolated unit of code, it is imperative that the basics are done right to ensure that the developer reduces the amount of time spend on a program in the future. By writing unit tests before writing the code itself, it will reduce the number of errors thus greatly improving the process within the SDLC.

## Write Tests to Pass

The main idea behind unit testing code in a development environment is to write the unit test before writing the code for production. What you are trying to achieve here is to write the code to pass the test. Each time test has failed you will change the functionality to ensure that all your tests pass. There are certain parts of code that do not need to be tested (mock objects) but anything that is complex in its logic must almost certainty need to be tested. These tests will provide all developers the time needed to produce the product required.

## Culture of Unit Testing

Unit testing within your team is a habit that is developed over time. Some rules that should be applied are repository changes and code reviews. Each time a developer wants to commit to the production repository it must be tested beforehand. If a commit has not been tested first, it should not go into the repository. It can be used as an assurance of quality and will give each developer a feeling of satisfaction in knowing that each iteration over the test code is in fact the ‘definition of done’.

## Version Control

Version control relates to a system why by the records of each change that is made to a file or a complete set of files are recorded over time. Each record can then be used to track the state of a file as it is developed. All the source code that is transferred to the repository can be reverted to its previous state if any errors have occurred.

## Branching Strategies

Branches from a repository can be used for collaboration or by a single user. A good branching strategy is when teams develop changes to the code while simultaneously not affecting code that any one team member has developed. Only when tests are carried out and when the code is merged, at this point each developer get to see the effects that may be caused the underlying source-code. We will be using Git as our as our repository for branching and merging our source code.

## Git Feature Branch Workflow

[Git is an open source](https://git-scm.com/about) distributed version control system designed to handle everything from small to very large projects with speed and efficiency. The essence behind the feature branch workflow is where all the development that is created around this feature should take place on this feature branch without having to go near the master branch. This concept will allow multiple developers to maintain their code that is being developed for a new feature instead of contributing to the main production codebase. This is fundamental in continuous integration environments as the master branch will never feature code that is only at the testing stage.

## Feature Branch in motion

These branches assume a central repository and the master branch represents the history of the project that is in production. Feature branching is ideal for collaboration between developers by creating a new branch every time that you are working on a new feature. Make sure you use a unique or very descriptive name when creating a new branch. The name could relate to the current Jira ticket that you are working on. All new feature branches are created from the latest version of code from the master branch in production. Make sure you [check out](https://git-scm.com/docs/git-checkout) the branch locally so that any changes that are made will be on that branch.

A screenshot of a cell phone

Description automatically generated

## Merge Strategies

A merge is when you must combine two branches and Git will attempt to find a common base between them. This is what is called a merge strategy. I strongly recommend that you use the terminal instead of the built-in support within your IDE. This will enable you to fully understand the commands and it will keep all your projects more organized. Also, to be able to view the list of commits that are ordered by the actual time stamp which can be found and reverted, if needed, in a more efficient manner.

## Test before Commit

Tests should always be carried out on your code before it is committed to the central repository. What is generally conceived to be good code, could turn out to be something different when other developers get to review it. To minimise mistakes and rollbacks each new feature should be tested which should also include messages before any pushing or sharing of code with your co-workers.

## Avoid Committing Half Completed Work

Committing is an important part of the development process. Each logical new feature implementation can be divided into smaller parts to commit if needed. Here is where you can take advantage of Git’s [Stashing](https://git-scm.com/book/en/v1/Git-Tools-Stashing) tool that stages your uncommitted code for further development without the need to switch branches or to commit half completed code.

## Agreed Workflow

The whole idea with having a shared workflow to follow is to have a stable branch where you can work with your team in confidence. As each new feature branch is added, each developer can pull down the latest product feature from the repository to make changes. This workflow will allow collaboration with team members where by pull requests and merge reviews can be managed in a more productive fashion.

## Best Practices

Committing often will allow you to keep your new code to a minimum and it also helps to solely concentrate on the related changes. It will also allow you to share your new code with other members of the teams more often. Your team can also give you feedback on your changes more often. Any large commit that shows conflicts can become much harder to trace when trying to debug. This approach will over time reduce the amount of [merge conflicts](https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts) that can happen when developers introduce large amounts of new code.

## Conclusion

By following the above steps when developing new tests or writing code for a new feature, these guidelines if followed are trying to implement the best practices from within this team. We are all working to improve the features of the product. As always feedback is important as we are always trying to find ways of improving these processes.