2.1 Coding Conventions

/2.0 Conventions/2.1 Coding Conventions

Java Code Conventions

Introduction

The coding standards will following the Check Style and SonarLint conventions with some modifications describe here.

The maven build plugin **route66-code-quality** is used to enforce thse standards. Any conflict between checkstyle and SonarLint will be resolved by accepting SonarLint rule and the checkstyle rule will be modified to match the SonarLint rule.

Coding Standards

All projects and artifacts must adhere to Clean Code principles, built on SOLID OOP principles and Design Patterns. Design and build tools will be used to ensure designs and code adhere to standards (See Development Environment).

File Organization

Class files will be organized as described in 3 - File Organization.

Class declarations should generally follow 11.1 Java Source File Example and organize members and fields by the following scope order as specified in the JVM Specification.

1. public
2. protected
3. package
4. private

Indention

Two spaces should be used as the unit of indentation.

Line Length

Avoid lines longer than 120 characters.

Comments and Documentation

Documentation comments generally follow the How to Write Doc Comments for the Javadoc Tool.

Comments should be written in complete sentences and should *not state the obvious.* For example, documentation for the method **getImage()** should not state *this method will get the image*. That fact is stated in method name. Instead comments should provide informative and useful information about the method. If the method's purpose is obvious by it's name then the method need not be commented. This may be the case in Jave Bean classes where setters and getters do nothing more the pass object references.

See Requirements for Writing Java API Specifications description and examples of each of the following comment types:

What should be documented

Classes are required to have accompanying documentation

- *public* methods should be documented
- *protected* methods should be documented
- *private* methods do not need to be comment since private methods in JavaDoc are not generated

Package Documentation

```
/**
 * { Executive summary: brief description of the purpose.}
 *
 * { OS/Hardware Dependencies. }
 *
 * { External specifications }
 *
 */
```

Class Documentation

```
/**
 * { Executive summary: brief one line description of the purpose of the
class}
 *
 * { State Information:}
 *
 * { OS/Hardware Dependencies: }
 *
 * { Allowed Implementation Variances: }
 *
 * { Security Constraints: }
 *
 * { Serialized Form: }
 *
 * { External Secifications : }
 *
 * { Example of using the class }
 *
 * @author { author name }
 *
 * @since { version of the library the class was intruduced }
 */
```

Field Documentation

```
/**
 * { What this field models }
 *
 * { Range of valid values }
 *
 * { Null Value }
 *
 * @since { version of the library the method was intruduced }
 */
```

Method Documentation

```
/**
 * { Expected Behavior: }
 *
 * { State Transitions: }
 *
 * { Null Argument Values: }
 *
 * { Range of Return Values: }
 *
 * { Algorithms Defined: }
 *
 * { OS/Hardware Dependencies: }
 *
 * { Allowed Implementation Variances: }
 *
 * { Cause of Exceptions: }
 *
 * { Security Constraints: }
 *
 * { Example of using the class }
 *
 * @since { version of the library the method was intruduced }
 */
```

Naming Conventions

Naming conventions will follow the checkstyle Java Naming Conventions with the exeptions:

All package names will begin with "**admitto.unum.**" followed by the project and module id's. For example the route66 library Maybe interface would be in the **admitto.unum.route66.expressive** package.

For method, field and variable names see **1.3 Method Naming Conventions** for list and dictionary of standarized names.

References

- https://google.github.io/styleguide/javaguide.html
- http://www.oracle.com/technetwork/java/codeconventions-150003.pdf