ASM Linter - Development/Maintenance Guide

Libraries

This software was created in Java 1.8 with the following libraries:

Library name	Version
plantuml	8059
snakeyaml	2.0
ASM	9.2
JUnit	5.8.1

These libraries are organized and handled using Gradle 6.8. The project also uses Gradle for compilation and testing.

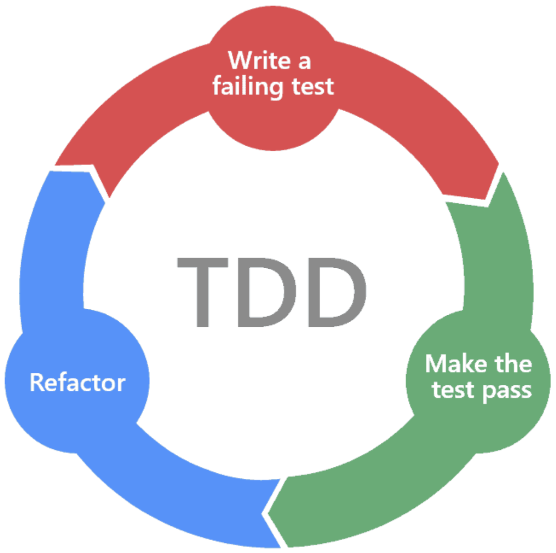
Development Process

The development process of our software project adheres to the principles of Test Driven Development (TDD), a widely adopted software development approach. TDD involves writing tests before implementing the actual software code, ensuring that the desired behavior and functionality of the software are thoroughly defined and verified.

At the core of our project lies JUnit, a powerful and popular Java testing framework that enables us to create and execute unit tests. These tests serve as executable documentation, specifying the expected outcomes and behaviors of different parts of our software.

The TDD process begins by identifying a specific requirement or functionality that needs to be implemented in the software. This requirement is typically derived from user stories, system specifications, or other sources of project requirements. Before any code is written, we focus on defining the expected behavior and outcomes of the functionality through test cases.

Each test case is designed to verify a specific aspect of the software's behavior. It encapsulates a set of inputs, the expected output, and the conditions that determine whether the functionality is correctly implemented. These test cases are written in JUnit, utilizing its comprehensive set of assertions and annotations to define and manage the tests effectively.



TDD encourages refactoring to improve the quality of the code while preserving the existing behavior. After a set of tests pass, we evaluate the codebase for any potential enhancements, such as eliminating duplication, improving readability, or optimizing performance. Refactoring allows us to continuously enhance the codebase without compromising the correctness of the software, thanks to the safety net provided by the comprehensive test suite.

Throughout the development process, the test suite remains an essential asset. It serves as a safety net, guarding against regression and alerting us to any unintended consequences of code changes. By rerunning the test suite regularly, we can quickly identify and fix any issues that may arise during the development cycle.

Testing Process

Within our project's GitHub repository, we have incorporated the utilization of GitHub Actions, a powerful automation tool provided by GitHub. GitHub Actions enables us to automate various tasks, including the execution of tests, by defining workflows that are triggered in response to specific events within our repository.

In our case, we have configured GitHub Actions to automatically run all tests whenever a branch attempts to merge into the main branch. This setup ensures that any changes or additions made to the codebase are thoroughly validated before being integrated into the main branch. By enforcing this requirement, we maintain the stability and reliability of the main branch, ensuring that it always contains code that passes all tests.

GitHub Actions

When a branch attempts to merge into the main branch, GitHub Actions is triggered and initiates the defined workflow. This workflow consists of a series of steps and actions that execute specific tasks. One crucial step in our workflow is the execution of our comprehensive test suite, which includes all the unit tests developed using JUnit. These tests exercise different components, functions, and scenarios of our software, validating the expected behaviors and outcomes.

If any of the tests fail during this automated execution, GitHub Actions detects the failure and reports it as a status check on the branch attempting to merge into the main branch. This failure acts as a gatekeeper, preventing the merge from proceeding until the failed tests are resolved. By enforcing this policy, we uphold the principle that any code merged into the main branch must pass all tests, maintaining the integrity of our software and safeguarding against introducing potential issues or regressions.

The use of GitHub Actions to enforce the passing of tests before merging into the main branch promotes a culture of continuous integration and quality assurance within our development team. It ensures that the main branch is always in a stable state, with all tests passing, thus reducing the risk of introducing bugs or breaking existing functionality. It also provides a transparent and automated mechanism for tracking and addressing test failures, fostering collaboration and accountability among team members.