

April 26, 2024

Lint report

Repository: https://github.com/C1-010/Acme-SF-D03

Group: C1.010

García Rivero, Andrés - [andgarriv@alum.us.es](mailto:andgarriv@alum.us.es)

López Cubiles, Antonio José - [antlopcub@alum.us.es](mailto:antlopcub@alum.us.es)

Nieto Vicioso, Javier - [javnievic@alum.us.es](mailto:javnievic@alum.us.es)

Díaz Ordoñez, Pablo -  [pabdiaord@alum.us.es](mailto:%20pabdiaord@alum.us.es)

Sevillano Barea, Alejandro - [alesevbar@alum.us.es](mailto:alesevbar@alum.us.es)

Content Table

[1. Executive Summary 2](#_Toc165055220)

[2. Revision Table 2](#_Toc165055221)

[3. Introduction 2](#_Toc165055222)

[4. Contents 3](#_Toc165055223)

[4.1 Bad Smell: Replace this assert with a proper check 3](#_Toc165055224)

[4.2 Bad Smell: Rename this package name to match the regular expression '^[a-z\_]+(.[a-z\_][a-z0-9\_])$'. 3](#_Toc165055225)

[4.3 Bad Smell: Define a constant instead of duplicating this literal "identification" 4 times. 4](#_Toc165055226)

[5. Conclusions 4](#_Toc165055227)

[6. Bibliography 4](#_Toc165055228)

# 1. Executive Summary

This Lint report provides an in-depth analysis of the bad smells detected by Sonar's Lint tool within the project. It includes a detailed list of reported bad smells, along with justifications explaining why they are considered innocuous. Notably, bad smells identified within the Acme Framework can be safely disregarded. In cases where Sonar's Lint has incorrectly flagged an innocuous bad smell, consulting the lecturer for clarification is recommended. Alternatively, correcting the identified bad smell and excluding it from the report is advised. This report aims to ensure thorough examination and resolution of reported bad smells, ultimately improving the overall quality of the project.

# 2. Revision Table

|  |  |  |
| --- | --- | --- |
| Revision number | Date | Description |
| 1 | 26/04/2024 | Created and finished this report |

# 3. Introduction

This Lint report presents a comprehensive analysis of the bad smells detected by Sonar's Lint tool within our project. It aims to provide insights into identified issues and offer justifications for considering them innocuous. In instances where Sonar's Lint may inaccurately flag a bad smell as innocuous, consulting the lecturer for clarification is encouraged.

The report is structured to facilitate a thorough examination and resolution of reported bad smells, contributing to the overall enhancement of the project's quality. It begins with an Executive Summary summarizing the report's objectives and key findings. Following that, a Revision Table outlines the document's revision history for transparency and accountability. The main body of the report comprises detailed analyses of specific bad smells, providing insights into their nature and implications. Each bad smell is categorized based on severity and accompanied by an explanation of why it is considered innocuous. Finally, the report concludes with a concise conclusion summarizing the key insights and a bibliography section.

# 4. Contents

## Bad Smell: Replace this assert with a proper check

Type: Medium

Classes: AnyContractListService, AnyContractShowService, AnyProgressLogListService, AnyProgressLogShowService, AuthenticatedClientCreateService, AuthenticatedClientUpdateService**…**

Explanation:

The use of assert statements in the mentioned classes suggests that the code relies on certain conditions being true at runtime. However, assert statements are typically used for debugging purposes and should not be relied upon for ensuring program correctness in production code. Instead, proper checks should be implemented to handle potential error conditions gracefully.

For example, in the AnyContractShowService class, the assert statement assert object != null is used in the unbind method. This assert checks if the object being passed to the method is not null. However, relying **only** on assert statements for null checks can lead to unexpected behavior if assertions are disabled in the runtime environment. Therefore, it is recommended to replace assert statements with proper null checks using conditional statements like if-else.

## Bad Smell: Rename this package name to match the regular expression '^[a-z\_]+(.[a-z\_][a-z0-9\_])$'.

Type: Low

Classes: AnyProgressLogController, AnyProgressLogListService, AnyProgressLogRepository, AnyProgressShowService**…**

Explanation: The package names mentioned do not adhere to the specified regular expression pattern. Package names in Java should follow standard conventions to ensure code readability and maintainability. By adhering to the specified regular expression pattern, it helps maintain consistency across the codebase and makes it easier for developers to understand the purpose of each package.

For example, in the AnyProgressLogController class, the package name is specified as acme.features.any.progressLog, which does not match the regular expression pattern. To rectify this, the package name should be adjusted to match the specified pattern, such as acme.features.any.progress\_log.

The primary purpose of package naming conventions is to provide a consistent structure for organizing code within a project, and the deviation from the pattern does not introduce any significant issues or risks to the codebase.

## Bad Smell: Define a constant instead of duplicating this literal "identification" 4 times.

Type: Major

Classes: AuthenticatedClientCreateService

Explanation:

Duplicating literal values multiple times throughout the codebase can lead to maintenance issues and make it harder to update values in the future. It's generally considered a best practice to define constants for such values to improve code readability, maintainability, and to avoid introducing errors due to typos or inconsistencies.

For instance, in the AuthenticatedClientCreateService class, the literal value "identification" is duplicated multiple times within the bind and validate methods. Instead of duplicating the literal, a constant should be defined at the class level, and then referenced wherever needed. This approach centralizes the definition of constants, making it easier to update them if necessary and reducing the likelihood of errors.

Therefore, while defining a constant for the literal value "identification" is a good practice in general, its absence in this specific context does not pose a significant risk or detriment to the codebase.

# 5. Conclusions

**I**n summary, the identified bad smells, while flagged by Sonar's Lint, do not significantly impact the project's quality or maintainability. Assert statements, package naming inconsistencies, and literal value duplication are minor issues that can be easily addressed without compromising code integrity. Moving forward, a balanced approach to code review and maintenance will ensure continuous improvement while minimizing disruptions to development workflows.

# 6. Bibliography

Intentionally blank