

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática


DP2-Reporte de Linting Student 3 D3



Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2023 – 2024

<u>Group:</u>	C1.016
<u>Repository:</u>	https://github.com/luchersol/Acme-SF-D02-24.1
<u>Student #5</u>	<div><p>UVUS: edurobrus Name: Robles Russo, Eduardo Email: edurobrus@alum.us.es</p></div>
<u>Date:</u>	Sevilla Febrero 14, 2024

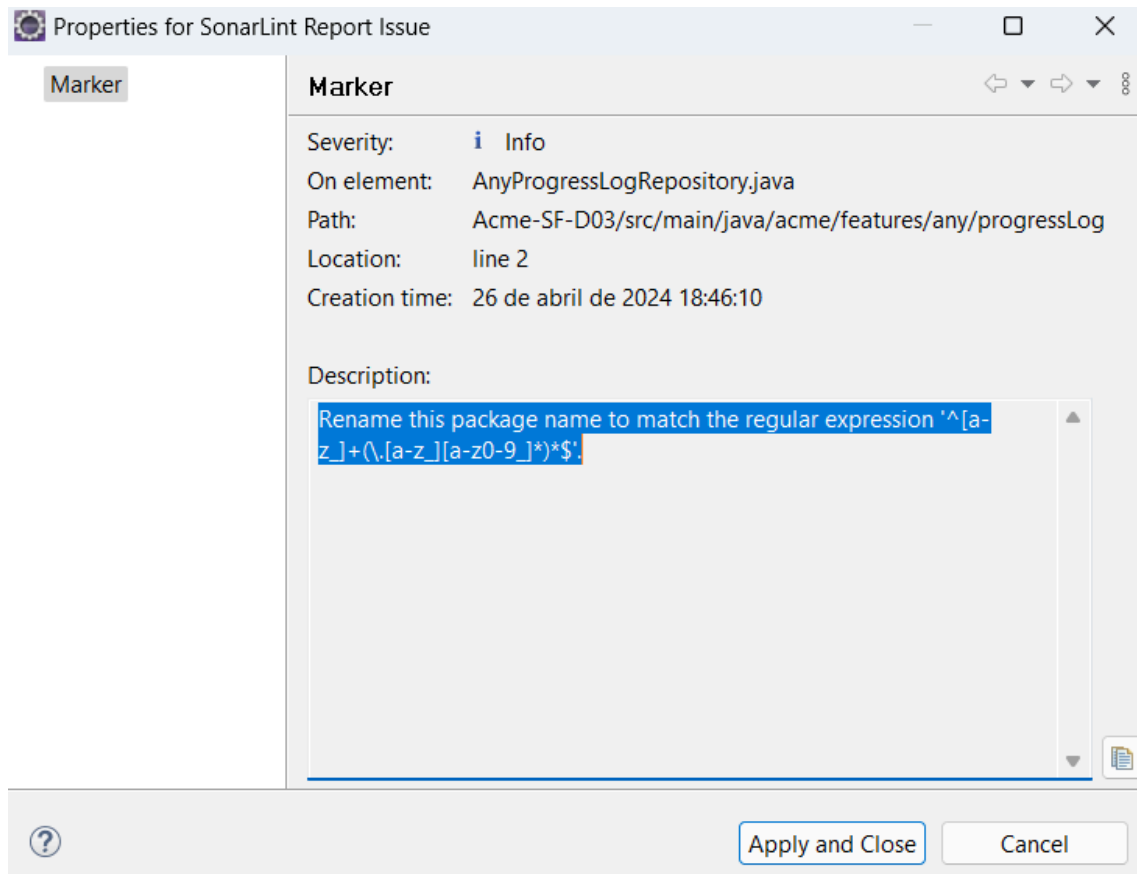
Índice de contenido

1. Versiones	2
2. Lista de bad smells	2
3. Conclusión	2
5. Bibliografía	2

1. Versiones

Versión	Fecha	Autor
1.0	23/04/2024	Eduardo Robles Russo

2. Lista de malos olores



- Rename this package name to match the regular expression `'^[a-z_]+(\.[a-z_][a-z0-9_]*)*$'`.

Este es un mal olor que sugiere cambiar el nombre del paquete para que coincida con la expresión regular proporcionada. Esto aseguraría que el nombre del paquete siga las convenciones de nomenclatura establecidas, lo que facilitaría la comprensión y el mantenimiento del código. Por lo tanto, se recomienda seguir esta sugerencia y realizar el cambio correspondiente en el nombre del paquete.

AnyTrainingModuleControll	⬇️ Rename this package name to match the regular expression <code>^[a-z_]+\.[a-z_][a-...</code>
AnyTrainingModuleListServi	⬇️ Rename this package name to match the regular expression <code>^[a-z_]+\.[a-z_][a-...</code>
AnyTrainingModuleListServi	⚠️ Replace this assert with a proper check.
AnyTrainingModuleReposit	⬇️ Rename this package name to match the regular expression <code>^[a-z_]+\.[a-z_][a-...</code>
AnyTrainingModuleShowSe	⬇️ Rename this package name to match the regular expression <code>^[a-z_]+\.[a-z_][a-...</code>

- Usar `\d` en lugar de `[0-9]` en expresiones regulares:

Este mal olor se refiere a "cuantificadores y clases de caracteres en expresiones regulares deberían usarse de forma concisa". Utilizar la expresión regular `[0-9]` para representar cualquier dígito del 0 al 9 podría simplificarse usando `\d` para indicar dígitos numéricos en general. Sin embargo, por motivos de claridad en la expresión regular y porque así se especificó en los requisitos, he optado por mantenerla tal como está. Esta decisión no afecta el funcionamiento del código y, además, contribuye a hacer la expresión más clara y fácil de entender.

- Eliminar código inservible:

Este mal olor se presenta como "asignaciones no utilizadas deberían ser eliminadas". Considero que este es un mal olor válido, ya que el código no utilizado puede causar problemas si el código cambia en el futuro. Por lo tanto, he decidido eliminar dicho código para solucionar este problema.

- Utilizar Guava en lugar de Java:

Este mal olor indica que se prefieren las características de Java a Guava. No estoy de acuerdo en considerarlo como un mal olor, ya que estoy utilizando las características proporcionadas por Java en lugar de la biblioteca Guava. No veo que esto cause problemas en el futuro.

- Evitar usar `assert`:

Se sugiere reemplazar el `assert` con una verificación adecuada. Sin embargo, como el código sigue cumpliendo su propósito asegurando que el objeto no sea nulo, y dado que esta instrucción forma parte de las pautas recomendadas para implementar servicios utilizando el framework que estamos utilizando, he decidido no abordarlo.

- Crear constantes genéricas en lugar de repetir la misma variable en varios sitios:

Este mal olor se refiere a "literales de cadena no deben ser duplicados". Aunque este problema no afecta críticamente la calidad del código y cumple su propósito sin afectar la comprensión ni el funcionamiento del código, quiero señalar que estas estructuras fueron recomendadas en la capacitación de servicios que recibimos. Por lo tanto, he seguido esta estructura en mi código y no planeo cambiarla.

2. Conclusión

Aunque existen algunos "malos olores" en el código, podemos concluir que no afectan significativamente su calidad y funcionamiento. El código es fácil de entender y leer, lo que es esencial para su mantenimiento y evolución.