

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

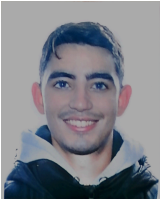
DP2-Reporte de Linting Student 5 D3



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-D03
<u>Student #5</u>	
	
UVUS:	dangalcan
Name:	Galván Cancio, Daniel
Email:	dangalcan@alum.us.es
<u>Date:</u>	Sevilla Abril 20, 2024

Índice de contenido

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

1. Versiones

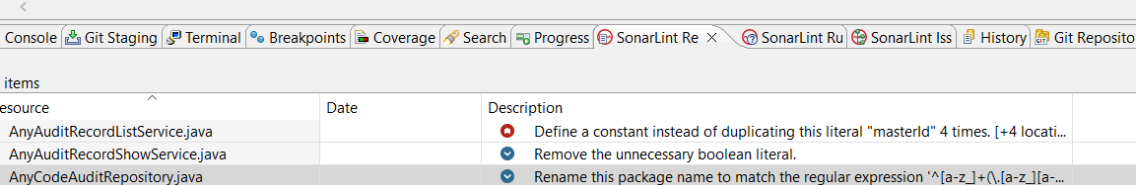
Versión	Fecha	Autor
1.0	20/04/2024	Daniel Galván Cancio

2. Lista de bad smells

- Renombrar archivos:

En los paquetes, aparece muchas veces 'Rename this package name to match the regular expression '[a-z_]+(\.[a-z_][a-z0-9_]*)*\$'.' Esto más que un badSmell sería el incumplimiento de las prácticas comunes y estándares de nombrado de paquetes. No afecta a la comprensión del código ni a su funcionamiento, por ello, es algo que no es importante y no he decidido modificar.

```
1
2 package acme.features.any.codeAudit;
3
4 import java.util.List;
```

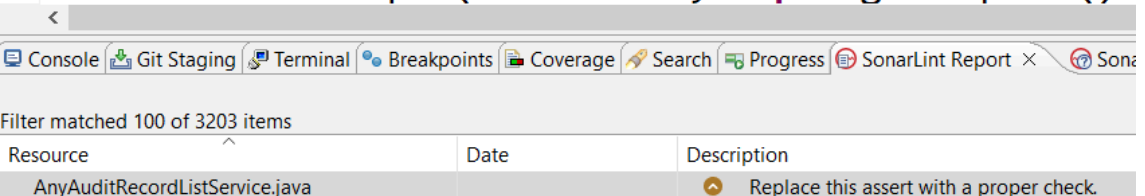


Resource	Date	Description
AnyAuditRecordListService.java		Define a constant instead of duplicating this literal "masterId" 4 times. [+4 locati...
AnyAuditRecordShowService.java		Remove the unnecessary boolean literal.
AnyCodeAuditRepository.java		Rename this package name to match the regular expression '[a-z_]+(\.[a-z_][a-z0-9_]*)*\$'.

- Evitar usar assert:

'Replace this assert with a proper check'. Este bad smell no es dañino porque el asset sigue cumpliendo con el objetivo de asegurar que el objeto no sea nulo, y por tanto, también he decidido ignorarlo. Además esa instrucción forma parte de la manera en la que se nos indica que hagamos los servicios al usar el framework, por lo que no lo cambiaré.

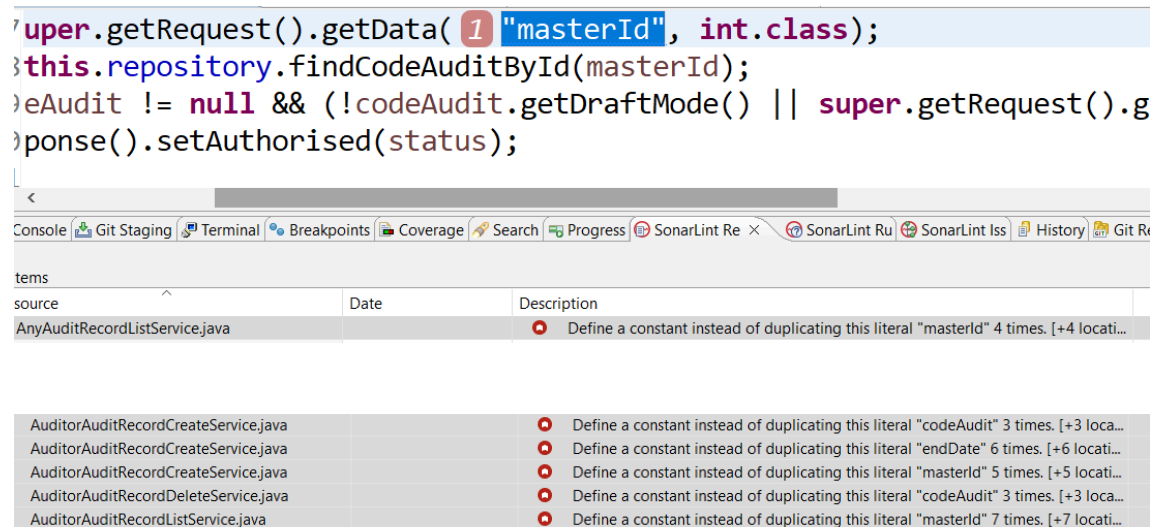
```
46 assert object != null;
47 Dataset dataset;
48 dataset = super.unbind(object, "code", "sta
49 dataset.put("masterId", super.getRequest().
```



Resource	Date	Description
AnyAuditRecordListService.java		Replace this assert with a proper check.

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

Este code smell tiene la siguiente descripción: ‘Define a constant instead of duplicating this literal "masterId" 4 times. [+4 locations]’. Aparece más veces en varios archivos. Me ha ocurrido con masterId más veces, pero también con las variables endDate y codeAudit. Realmente este bad smell tampoco es dañino porque de nuevo ocurre algo parecido a lo anterior, en los servicios nos explicaron que hiciéramos así las cosas y por ende, es la estructura que he seguido. De todos modos, esto no afecta ni a la comprensión, ni al funcionamiento del código, por lo que no es influyente.



- Usar ! en lugar de == false:

Este bad smell tiene la siguiente descripción: ‘Remove unnecessary boolean literal’, y hace referencia a que en lugar de poner “== false”, ponga “!” en la expresión booleana que quiero negar, en este caso auditRecord.getDraftMode(). Si bien es cierto que a priori suena más cómodo (y lo es), cuando estoy haciendo condiciones booleanas que son complejas (hay más de una condición), como los nombres de los métodos y de las variables suelen ser algo largos, para evitar confundirme al leer y que se me olvide visualizar el “!”, prefiero poner == false. Personalmente creo que ayuda a la comprensión y que aunque sea algo un poco más largo de escribir hace que quede más claro. Me ha saltado más veces este bad smell en distintos archivos, en la misma circunstancia. Como en lo personal me ayuda a entender mis expresiones lógicas y no es algo que afecte de forma negativa al funcionamiento del código, he decidido dejarlo así.

```
auditRecord = this.repository.findAuditRecordById(masterId);
status = auditRecord != null && auditRecord.getDraftMode() == false;
super.getResponse().setAuthorised(status);
```

	Date	Description
litRecordShowService.java		Remove the unnecessary boolean literal.

- Sobreescibir el método equals():

Su descripción es 'Override the "equals" method in this class'. Este bad smell me indica que sería una buena idea sobreescibir el método equals() en CodeAudit y en AuditRecord para poder tener un método que indique cuándo 2 instancias de esas clases sean iguales. No obstante, como sabemos que lo serán cuando el id sea el mismo, he decidido ignorar el consejo. Además, ninguna entidad tiene su método equals() sobreescrito.

```
5 public class AuditRecord extends AbstractEntity {
7
3     private static final long serialVersionUID = 1L;
}
```

items	Date	Description
AuditRecord.java		Override the "equals" method in this class.

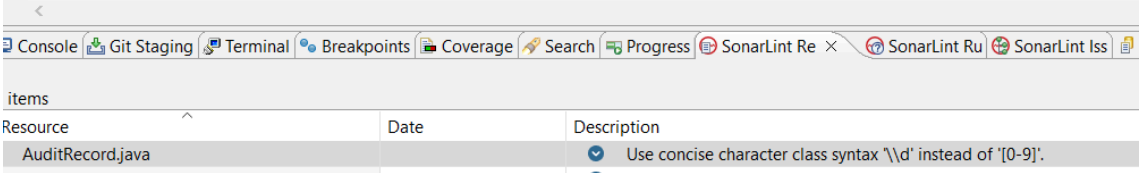
- Poner \\d en lugar de [0-9] en expresiones regulares.

Su descripción es 'use concise character class syntax '\\d' instead of [0-9]'. Realmente nos quiere decir que en lugar de decir que puede haber cualquier dígito del 0 al 9 ([0-9]), sería mejor idea poner que hubiera dígitos numéricos (\\d). No obstante, por claridad de la expresión regular y porque es la que se nos proporcionó en el listado de requisitos, he optado por ignorarlo. No afecta al funcionamiento del código, y además dejarlo así aumenta su claridad.

```

33 @NotBlank
34 @Column(unique = true)
35 @Pattern(regexp = "AU-[0-9]{4}-[0-9]{3}")
36 private String code;
37
38 @Past

```



Resource	Date	Description
AuditRecord.java		Use concise character class syntax '\\d' instead of '[0-9]'.

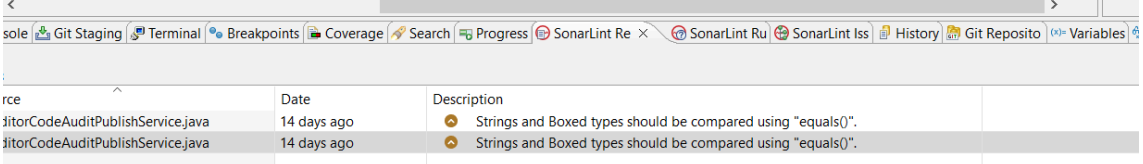
- Usar .equals() en lugar de != al comparar strings

La descripción dice: 'Strings and Boxed types should be compared using equals()'. Es cierto que es una buena práctica usar el método .equals() en lugar de != o ==, no obstante, al hacer las validaciones con los enumerados tuve problemas y por eso tuve que parsear sus valores a string (.toString()). En un inicio tenía puesto Mark.ValorDelEnum.equals("ValorDelEnum"). No obstante, no funcionaba y estuve probando hasta que encontré que haciendo la condición así sí que surtía efecto. Es por tanto por lo que decidí mantener así el código. Lo bueno es que de esa forma sí que funciona de manera correcta.

```

...s().hasErrors("mark")) {
    ...findCodeAuditMark(object.getId()).get(0);
    mark != null && mark.toString() != "F_MINUS" && mark.toString() != "F";
    "mark", "auditor.codeAudit.form.error.invalidMarkForPublish");
}

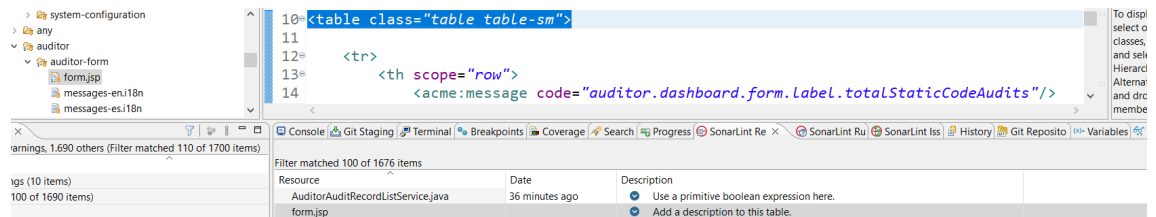
```



Resource	Date	Description
auditorCodeAuditPublishService.java	14 days ago	Strings and Boxed types should be compared using "equals()".
auditorCodeAuditPublishService.java	14 days ago	Strings and Boxed types should be compared using "equals()".

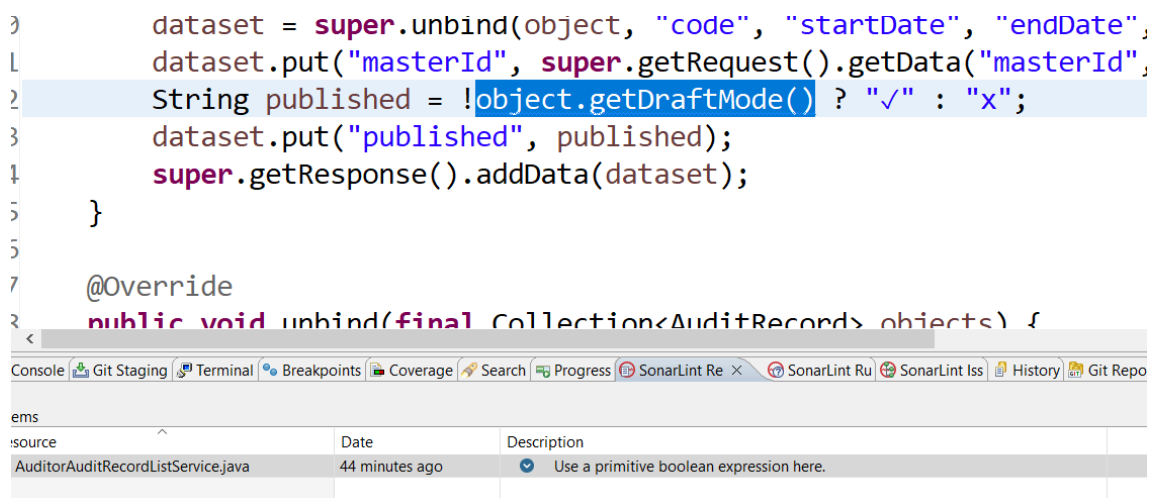
- No tener añadida una descripción en la tabla

La descripción es “Add description to this table”. El bad smell te sugiere añadir una descripción a la tabla. En este caso no es necesario añadir una descripción, por lo que a pesar de que sonarLint me sugiera añadirle una, no lo haré. No es un bad smell que suponga un problema ya que es algo puramente estético, y no empeora ni altera la funcionalidad del código. Además tampoco dificulta su comprensión.



- Utilizar una expresión booleana primitiva

La descripción es “Use a primitive boolean expression here.” En este caso draftMode nunca puede ser nulo, porque siempre lo inicializamos a true al crear el objeto, y siempre verificamos que no sea nulo, por lo que esa expresión no puede dar nunca error puesto que object.getDraftMode() siempre tendrá un valor distinto de nulo. Es por esto por lo que no es necesario modificar el código. Por consiguiente, podemos concluir que el bad smell no afecta negativamente al funcionamiento del código, ni empeora su legibilidad, por lo que es inocuo.



3. Conclusión

Podemos concluir que a pesar de que haya algunos bad smells, el código generado no es de mala calidad y funciona de manera correcta, siendo además, fácilmente entendible y legible.

4. Bibliografía

No hay bibliografía presente para esta entrega. (intentionally blank)