DP2 2021-2022

Informe Lint

Proyecto Acme Toolkits

https://github.com/Alex-GF/Acme-Toolkits.git

Miembros:

- VICENTE CAMBRÓN TOCADOS (viccamtoc@alum.us.es)
- FRANCISCO JAVIER CAVERO LÓPEZ (fracavlop@alum.us.es)
- IRENE XIANG DOMÍNGUEZ GAROZ (iredomgar4@alum.us.es)
- ALEJANDRO GARCÍA FERNÁNDEZ (alegarfer4@alum.us.es)
- JOSÉ LUIS GARCÍA MARÍN (josgarmar31@alum.us.es)
- MARTA REYES LÓPEZ (marreylop4@alum.us.es)

GRUPO E1-03

Versión 1.0

23/05/22

Índice

Índice....2
Resumen ejecutivo....3
Tabla de revisiones....3
Introducción....4
Contenido....4
Lint Report....4
Private constructor....5
Call “Optional#isPresent()” before accesing the value....6
Use a StringBuilder instead....6
Save and re-use “Random”....6
Replace lambda method....6
Cambiar “replaceAll()” por "replace"....7
Situación final....7
Conclusión....7
Bibliografía....8

Resumen ejecutivo

El objetivo de este documento es inspeccionar el código realizado y así encontrar errores y malos olores reportados por Lint con respecto al proyecto. Se solucionarán los malos olores reportados y en caso de ser un mal olor no real, se proporcionará una clara justificación de este.

Tabla de revisiones

Fecha	Versión	Descripción	Entregable
23/05/2022	1.0	Versión inicial	4

Introducción

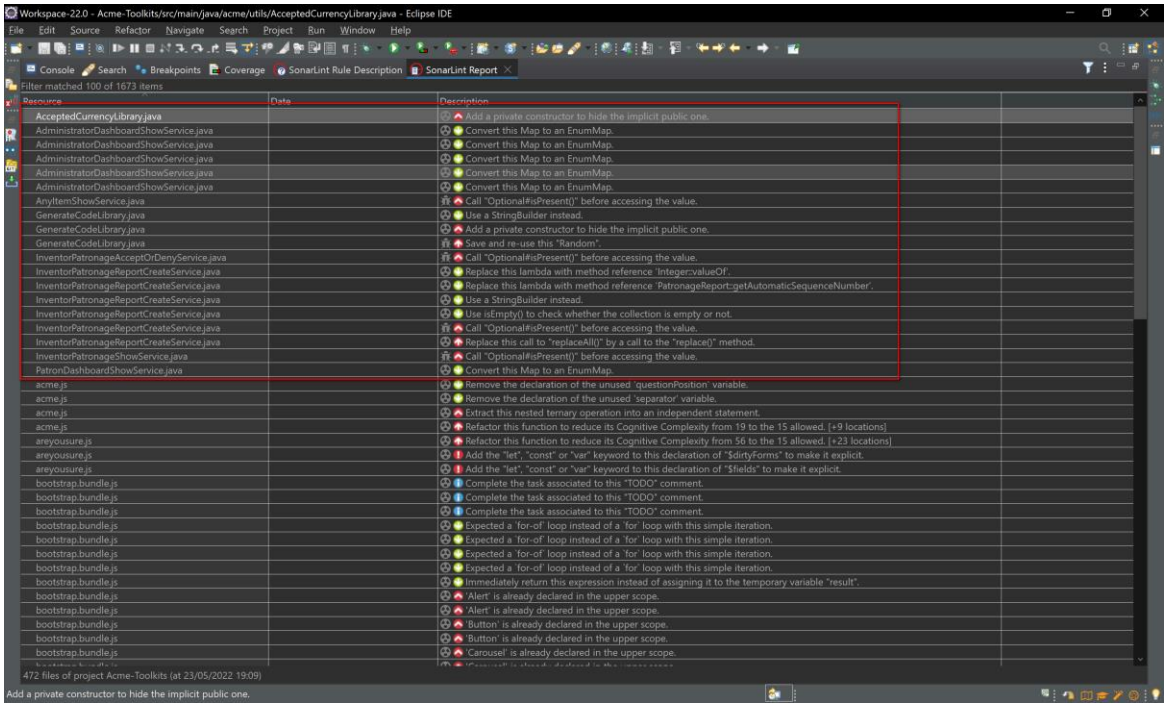
En este documento se enumeran y detallan todos los malos olores encontrados en el proyecto tras la realización de las distintas pruebas. Por tanto, una vez realizadas y pasado las pruebas se realizó con el SonarLint plug-in que nos proporciona Eclipse la detección de los malos olores.

Todos los malos olores reportados en el LintReport se solucionarán refactorizando hasta que deje de ser reportado por SonarLint. Si por cualquier caso nos reportase un error no real se proporcionará una justificación clara y definida del porqué de este.

Contenido

Lint Report

A continuación, se muestra una captura con los distintos errores en nuestro proyecto reportados en el Lint Report:



Las clases señaladas dentro de la sección roja son aquellas implementadas por nuestro grupo que contienen algún tipo de mal olor (leve o grave) o bug. Fuera de esta sección se encuentran archivos correspondientes al framework que estamos utilizando: Acme; que se encuentra completamente testeado.

Dado que en el Lint Report del Sprint 3 ubicado en la carpeta "reports/Entregable_3/Informe_Lint.pdf" ya explicamos las entradas referentes a EnumMaps (en AdministratorDashboardShowService.java y PatronDashboardShowService), Acme.js, Areyousure.js y Bootstrap; no las abordaremos en este informe.

A continuación, enumeramos y comentamos los errores reportados:

Private constructor

Mal olor grave ubicado en las clases AcceptedCurrencyLibrary y GenrateCodeLibrary. Estos archivos contienen funciones estáticas que utilizamos para realizar todos los cambios de divisa y generación de códigos en la aplicación.

No se trata de un repositorio, servicio, etc., si no de librerías que usamos para albergar funciones que utilizamos en repetidas ocasiones y así no repetir bloques de código innecesariamente.

Para solucionar este error, seguimos el consejo que nos provee el propio Lint, crear un constructor privado dentro de las clases para que no sean públicos.

Este mal olor se marca como "solucionado".

Call “Optional#isPresent()” before accesing the value

Este bug aparece en las clases AnyItemShowService, InventorPatronageAcceptOrDenyService, InventorPatronageReportCreateService e InventorPatronageShowService. Surge por invocar el método .get() de la clase Optional<?> de java en la misma línea que crea dicho objeto.

En nuestro caso, decidimos invocarlo en la misma línea porque se tratan de servicios que proveen a una vista de tipo “show”. Esto quiere decir que, para acceder hasta ella, ha sido necesario hacer click sobre la entrada correspondiente al Patronage/Item en la vista de tipo “list” correspondiente. En este tipo de vistas, sólo se listan (filtrados o no) elementos que existen en la BD.

Debido a esto, nos aseguramos de que, si se accede a la vista show, el elemento con el id indicado por el campo id en la query de la URL existe y, por tanto, no estaremos en la situación de que el Optional<?> no contenga nada. En este caso (una entrada forzosa en la vista show de un elemento), el error lo reportaría la aplicación antes.

No será necesario hacer ningún cambio. Estos bugs se marcan como “no arreglar”.

Use a StringBuilder instead

Mal olor que aparece en las clases GenerateCodeLibrary e InventorPatronageReportCreateService. Se genera por el uso del operador += para concatenar strings. Dado que la funcionalidad es similar y el rendimiento no se ve prácticamente afectado, optamos por no modificar el código.

Este mal olor se marca como “no arreglar”.

Save and re-use “Random”

Bug localizado en la clase GenerateCodeLibrary. Se ocasiona por crear un objeto Random dentro de un bloque while numerosas veces. Para el uso que le dábamos al objeto, no era necesario tenerlo dentro. Por tanto, decidimos sacarlo fuera del bucle para solucionar el bug.

Este mal olor se marca como “solucionado”.

Replace lambda method

Mal olor leve en la clase InventorPatronageReportCreateService generado por dos funciones lambda que podía ser sustituidos por la notación para este tipo de funciones con “::”, en particular, hemos sustituido:

- x-> getAuthomaticSequenceNumber(x)**->** PatronageReport::getAutomaticSequenceNumber

- x-> Integer.valueOf(x)**->** Integer:valueOf

Este mal olor se marca como “solucionado”.

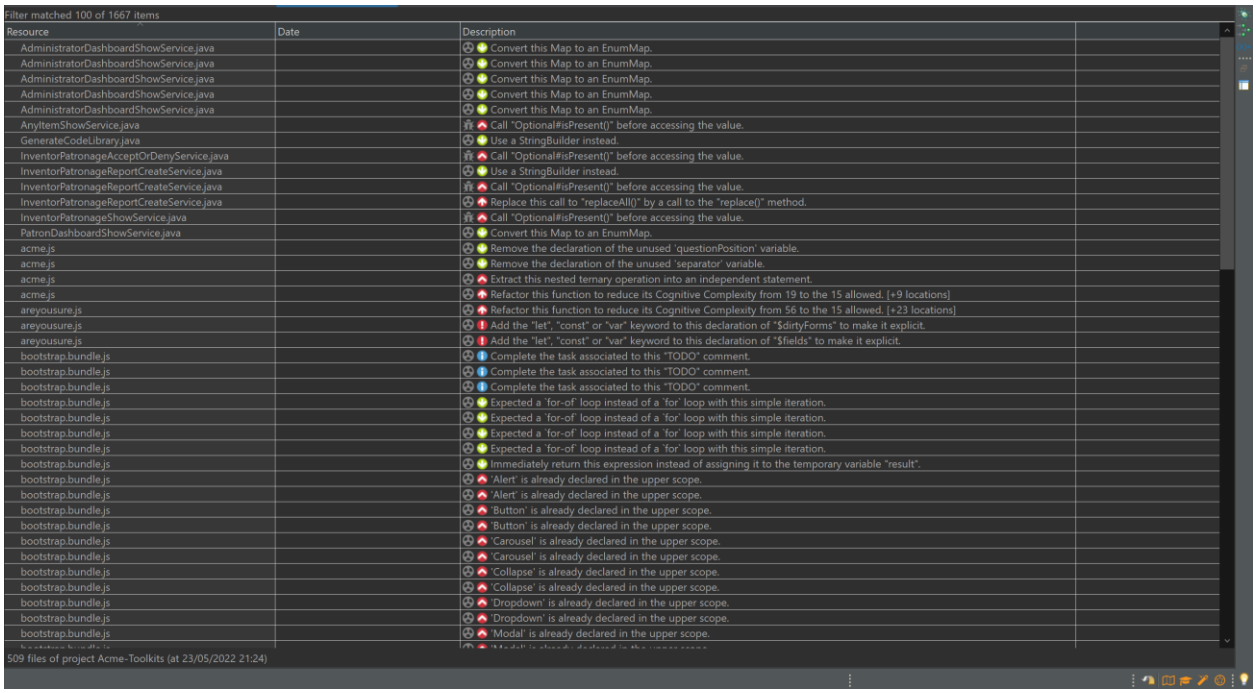
Cambiar “replaceAll()” por "replace"

A pesar de que está anotado como mal olor grave, es necesario utilizar replaceAll() es este punto, dado que tenemos que cambiar todos los 0s de cada sub-string por “”.

Este mal olor se marca como “no arreglar”.

Situación final

Tras realizar todas las modificaciones indicadas anteriormente, el resultado del Lint Report generado es el que sigue:



Conclusión

Observamos que, aunque desde el principio se ha intentado realizar un código muy estructurado y limpio, para que así esta parte tan tediosa de refactorizar y arreglar errores y malos olores nos la se pueda evitar, se ha visto que a lo largo del desarrollo esta refactorización ha sido inevitable.

A diferencia del sprint anterior, en esta ocasión el Informe Lint que nos ha generado SonarLint encontraba errores y malos olores bastante más relevantes para el proyecto. Se ha ido analizando uno por uno y decidiendo si se decidía a su arreglo o, por el contrario, se decidía no arreglarlo. Todas las decisiones han sido tomadas de forma justificada y defendida, atendiendo siempre a las recomendaciones de los profesores e incluso del propio SonarLint.

No obstante, considerando todo el código desarrollado a lo largo de este sprint, consideramos que los errores reportados han sido en menor cantidad. Por ende, seguimos manteniendo una estructura organizada y definida, permitiéndonos ahorrar tanto tiempo como esfuerzo.

Bibliografía

No aplica.