



Escuela Técnica Superior de
Ingeniería Informática

Lint Report

Deliverable-03

Miembros del equipo

Apellidos, Nombre	Correo electrónico	Roles
Gallardo Roco, Raúl	raugalroc@alum.us.es	Desarrollador Tester
Gómez Arias, Nuria	nurgomari@alum.us.es	Jefe Desarrollador Tester
Ortiz Guerra, Juan Antonio	juaortgue@alum.us.es	Desarrollador Tester
Páez Páez, Jesús	jespaepae@alum.us.es	Analista Desarrollador Tester Operador
Piñero Calera, Borja	borpincal@alum.us.es	Desarrollador Tester
Ramos Berciano, Pablo	pabramber@alum.us.es	Desarrollador Tester

Repositorio: <https://github.com/Hesusu-ikie/Acme-Toolkits>

Grupo E2.06

Fecha: 01 de Marzo de 2022

Índice

Resumen Ejecutivo	3
Tabla de Revisión	3
Introducción	3
Contenidos	3
Rename field “quantity”:	3
Eliminar la asignación de una variable local sin utilizar:	4
Usar preferiblemente StringBuilder:	5
Cambiar el nombre de las clases de Test para que coincidan con un patrón determinado:	6
Eliminar un literal booleano innecesario:	6
Reemplazar el uso de Map por EnumMap:	7
Añadir una descripción a las tablas de las vistas:	7
Conclusiones	8
Bibliografía	8

Resumen Ejecutivo

En nuestro grupo E2.06 tenemos conocimientos sobre este tipo de proyectos debido a asignaturas como Diseño y Pruebas I e Introducción a la ingeniería del Software y Sistemas de la Información 1 y 2. Gracias a ello podemos desarrollar sistemas con características parecidas a las que se requieren para este proyecto.

Nuestra intención es completar todas las características de cada Sprint para intentar obtener la máxima calificación posible en la asignatura.

Tabla de Revisión

Versión	Fecha	Descripción
1	23/04/2022	Primera versión

Introducción

En este informe se planea realizar un análisis del código gracias a la herramienta Sonar Lint instalada en Eclipse. Gracias a esta, se analizará todo nuestro código y se nos devolverá un reporte de todos los malos olores y posibles bugs en nuestro código. Además, realizaremos un análisis de estos malos olores, indicando cuál es su causa, si es algo importante y por tanto a corregir y cómo lo corregiremos.

Contenidos

Rename field "quantity":

Captura del reporte de Sonar Lint



Captura del fragmento de código con problemas

```


15 @Entity
16 @Getter
17 @Setter
18 public class Quantity extends AbstractEntity {
19
20     // Serialisation identifier -----
21
22     protected static final long serialVersionUID = 1L;
23
24     // Atributes -----
25
26     @Min(0)
27     protected Double quantity;
28 }

```

Este mal olor se debe a que en ocasiones es confuso tener una propiedad que se llame igual que la clase que la contiene. Estas confusiones se pueden ocasionar al llamar a un método que devuelva esta propiedad, donde alguien que no haya escrito el código podría confundir sobre qué devuelve ese método. Esto lo consideramos un problema que debemos de solucionar debido a la problemas con la mantenibilidad o de cara a añadir nuevas funcionalidades. Se corregirá cambiando el nombre a la propiedad en cuestión por “*numberOfQuantity*”

Eliminar la asignación de una variable local sin utilizar:

Captura del reporte de Sonar Lint

AnyToolkitShowService.java	 Remove this useless assignment to local variable "inventions".
InventorToolkitShowService.java	 Remove this useless assignment to local variable "inventions".

Captura del fragmento de código con problemas

```

36 @Override
37 public void unbind(final Request<Toolkit> request, final Toolkit entity, final Model model) {
38     assert request != null;
39     assert entity != null;
40     assert model != null;
41     final Double totalPrice = this.repository.findTotalRetailPriceByToolkitId(entity.getId());
42     final Collection<Invention> inventions = this.repository.findManyInventionsByToolkitId(entity.getId());
43     model.setAttribute("totalPrice", totalPrice);
44     model.setAttribute("id", entity.getId());
45     request.unbind(entity, model, "code", "title", "description", "assemblyNotes", "link");
46
47
48 }


```

En este caso nos encontramos con que estamos creando una variable que no estamos usando, esto puede deteriorar el rendimiento de nuestra aplicación innecesariamente, ya que estamos consumiendo más memoria de la que deberíamos sin darle uso. Este problema aparece en dos ocasiones ya que el código de estos dos requisitos es prácticamente el mismo. Es un problema que consideramos grave ya que nuestra aplicación deberá de funcionar lo mejor posible y que corregiremos eliminando la variable.

Usar preferiblemente StringBuilder:

Captura del reporte de Sonar Lint

AnyUserAccountShowService.java

 Use a StringBuilder instead.

Captura del fragmento de código con problemas

```


41 @Override
42 public void unbind(final Request<UserAccount> request, final UserAccount entity, final Model model) {
43     assert request != null;
44     assert entity != null;
45     assert model != null;
46
47
48     final List<UserRole> roles=new ArrayList<>(entity.getRoles());
49     roles.sort(Comparator.comparing(UserRole::getAuthorityName).reversed());
50     String rolesSt="";
51
52     model.setAttribute("name", entity.getIdentity().getName());
53     model.setAttribute("surname", entity.getIdentity().getSurname());
54     model.setAttribute("email", entity.getIdentity().getEmail());
55
56     boolean firstIteration=true;
57
58     for (final UserRole r: roles) {
59         if (firstIteration==true) {
60             rolesSt=r.getAuthorityName();
61             firstIteration=false;
62         }else {
63             rolesSt=rolesSt+", "+r.getAuthorityName();
64         }
65     }
66     model.setAttribute("roles", rolesSt);
67
68 }

```

Este mal olor es debido a que, para mostrar los distintos roles de un usuario, usamos una concatenación de String, para un visualizado más cómodo. El problema deriva de que para hacer esta asignación, en cada iteración del bucle la nueva cadena no se añade a la anterior, sino que para hacerlo el String tiene que pasar a un objeto intermedio, posteriormente añadir la nueva cadena y tras esto transformarlo de nuevo a un String. Sonar Lint nos propone cambiar esta metodología por el uso de StringBuilder, que se salta todos esos pasos innecesarios y crea el String al final, cuando todas las cadenas están ya añadidas. Esto sería beneficioso para el rendimiento de la aplicación, sobre todo cuando se trabaje con un gran volumen de datos. Por lo que se solucionará de la forma que propone Sonar Lint.

Cambiar el nombre de las clases de Test para que coincidan con un patrón determinado:

Captura del reporte de Sonar Lint

AuthenticatedAnnouncementShow.java  Rename class "AuthenticatedAnnouncementShow" to match the regular expression: '^((Test|IT)[a-zA-Z0-9_]+|[A-Z][a-zA-Z0-9_]+)(Test|Tests|TestCase|IT|ITCase)\$'

Captura del fragmento de código con problemas

```

9 public class AuthenticatedAnnouncementShow extends TestHarness{
10     @ParameterizedTest

```

Este mal olor deriva de la importancia de los nombres de las clases en el desarrollo de una aplicación y donde, por comodidad y posterior mantenibilidad, se suele seguir una metodología a la hora de nombrar a estas clases. En nuestro caso seguimos el patrón de <Rol><Entidad><Acción>Test. En este caso al nombre de la clase le falta la palabra Test para indicar que, efectivamente, es una clase que sirve para el testeado. Este mal olor se solucionará renombrando la clase mejorando así la comprensión futura del código.

Eliminar un literal booleano innecesario:

Captura del reporte de Sonar Lint

AnyUserAccountShowService.java  Remove the unnecessary boolean literal.





Captura del fragmento de código con problemas

```
55
56     boolean firstIteration=true;|
57
58     for (final UserRole r: roles) {
59         if (firstIteration==true) {
60             rolesSt=r.getAuthorityName();
61         }
62     }
```

Este mal olor es causado por el uso de la comprobación de una variable booleana con un literal de este mismo tipo. Esto es innecesario ya que bastaría con poner la propia variable para comprobar la condición que tenemos. Esto se solucionará ya que puede derivar en un pequeño problema de rendimiento que preferimos evitar pero sobre todo para una mejor lectura del código.

Reemplazar el uso de Map por EnumMap:

Captura del reporte de Sonar Lint

AdministratorAdministratorDashboardShowService.java	 Convert this Map to an EnumMap.
AdministratorAdministratorDashboardShowService.java	 Convert this Map to an EnumMap.
AdministratorAdministratorDashboardShowService.java	 Convert this Map to an EnumMap.
AdministratorAdministratorDashboardShowService.java	 Convert this Map to an EnumMap.
AdministratorAdministratorDashboardShowService.java	 Convert this Map to an EnumMap.















































Captura del fragmento de código con problemas

```
51
52     final Map<PatronageStatus, Double> numberPatronages = new HashMap<>();
53
54     final Map<PatronageStatus, Double> averageBudgetPatronage= new HashMap<>();
55     final Map<PatronageStatus, Double> deviationBudgetPatronage= new HashMap<>();
56     final Map<PatronageStatus, Double> minimumBudgetPatronage= new HashMap<>();
57     final Map<PatronageStatus, Double> maximumBudgetPatronage= new HashMap<>();
58
```

En este caso nos encontramos ante un mal olor que puede causar una bajada de rendimiento considerable debido a que en este método se realizan numerosas consultas a los Map y tenemos este mal olor en cinco ocasiones, por lo que solucionándolo podríamos mejorar algo el rendimiento. Este es causado debido a que, en este caso, los Map tienen como clave un enumerado, por lo que usando EnumMap se mejora la eficiencia en las consultas, ya que la lista de claves es un simple Array, que en este caso solo contendría tres valores (los valores del PatronageStatus). Esto se solucionará a fin de mejorar el rendimiento del código.

Añadir una descripción a las tablas de las vistas:

Captura del reporte de Sonar Lint

form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.
form.jsp	  Add a description to this table.

Captura del fragmento de código con problemas:

```
<table class="table table-sm">
  <jstl:forEach var="entry" items="${minimumRetailPriceComponents}">
    <tr>
      <th class="text-left">
```

Este error nos lo encontramos en cada una de las tablas creadas para las vistas de los dashboards. Es causado debido a no introducir una descripción de esta tabla, lo que podría provocar un problema de accesibilidad en nuestra aplicación, ya que, en caso de no saber de qué trata dicha tabla, habría un campo que lo describiría. Este mal olor también se solucionará usando para ello la propiedad *aria-describedby* de los `<table>`.

Conclusiones

Como podemos comprobar, tenemos algunos malos olores en nuestro código, principalmente debidos al desconocimiento de ciertas partes de la tecnología y de ciertas malas prácticas a la hora de desarrollar. Tras este análisis hemos podido verificar algunos de nuestros errores y solución, haciendo que en un futuro sea menos probable que se den.

Bibliografía

- Reporte Sonar Lint en Eclipse