



Escuela Técnica Superior de
Ingeniería Informática

Lint Report

Deliverable-04

Miembros del equipo

Apellidos, Nombre	Correo electrónico	Roles
Gallardo Roco, Raúl	raugalroc@alum.us.es	Desarrollador Tester
Gómez Arias, Nuria	nurgomari@alum.us.es	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/raugalroc/Acme-Toolkits>

Grupo E2.06

Fecha: 23 de Mayo 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	23/05/2022	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

Eliminar la asignación a “creationMoment”:

Captura del reporte de Sonar Lint



Captura del fragmento de código con problemas

```

public void bind(final Request<Chirp> request, final Chirp entity, final Error
    assert request != null;
    assert entity != null;
    assert errors != null;

    Date creationMoment;
    Calendar calendar;
    creationMoment = new Date();
    calendar = Calendar.getInstance();
    creationMoment = calendar.getTime();
    request.bind(entity, errors, "title", "author", "body", "email");
    entity.setCreationMoment(creationMoment);

```

Este mal olor se debe a que la variable “creationMoment” se asigna 2 líneas más abajo a partir de la variable “calendar”. Básicamente se realizan 2 asignaciones a la misma variable cuando la asignación que aparece en primer lugar con los atributos vacíos lo único que aporta es código prescindible y por tanto deteriora el aspecto del código.

Extraer el bloque de código anidado en un nuevo método:

Captura del reporte de Sonar Lint

InventorInventionCreateService.java		⚠️ Extract this nested code block into a method.
InventorInventionPublishService.java		⚠️ Extract this nested code block into a method.
InventorInventionUpdateService.java		⚠️ Extract this nested code block into a method.
InventorToolkitCreateService.java		⚠️ Extract this nested code block into a method.
InventorToolkitPublishService.java		⚠️ Extract this nested code block into a method.
InventorToolkitPublishService.java		⚠️ Extract this nested code block into a method.
InventorToolkitUpdateService.java		⚠️ Extract this nested code block into a method.

Resource	Date	Description
AdministratorAnnouncementCreateService.java		⚠️ Extract this nested code block into a method.
AdministratorAnnouncementCreateService.java		⚠️ Extract this nested code block into a method.

Captura del fragmento de código con problemas

```

    Boolean isSpam;

    isSpam = entity.isSpam(this.repository.getSystemConfiguration());

    errors.state(request, !isSpam, "*", "inventor.toolkit.form.error.spam");
}

```

(caso de las features InventorToolkit e InventorInvention)

```

{
    boolean confirmation;

    confirmation = request.getModel().getBoolean("confirmation");
    errors.state(request, confirmation, "confirmation", "javax.validation.constraints.AssertTrue.message");
}

{
    Boolean isSpam;

    isSpam = entity.isSpam(this.repository.getSystemConfiguration());

    errors.state(request, !isSpam, "*", "administrator.announcement.form.error.spam");
}




```

(caso de *AdministratorAnnouncement*)

En este caso nos encontramos con que estamos implementando un código dentro de un método entre llaves. Como podemos observar en las clases afectadas, el código entre llaves es un código redundante que se debería extraer en un nuevo método al que se le llame cada vez que se necesite en vez de repetir las mismas líneas continuamente en distintas partes de la aplicación.

Usar preferiblemente StringBuilder:

Captura del reporte de Sonar Lint

InventorInventionDeleteService.java		 Remove the unnecessary boolean literal.
InventorInventionPublishService.java		 Remove the unnecessary boolean literal.
InventorInventionUpdateService.java		 Remove the unnecessary boolean literal.
PatronPatronageDeleteService.java	35 minutes ago	 Remove the unnecessary boolean literal.
PatronPatronagePublishService.java	34 minutes ago	 Remove the unnecessary boolean literal.
PatronPatronageUpdateService.java	33 minutes ago	 Remove the unnecessary boolean literal.

Captura del fragmento de código con problemas

```

@Override
public boolean authorise(final Request<Invention> request) {
    assert request != null;

    boolean result;
    int inventionId;
    Invention invention;
    Inventor inventor;

    inventionId = request.getModel().getInteger("id");
    invention = this.repository.findOneInventionById(inventionId);
    inventor = invention.getInventor();
    result = invention.getPublished() == false && request.isPrincipal(inventor);

    return result;
}

```

(caso de InventorInvention)

```
@Override
public boolean authorise(final Request<Patronage> request) {
    assert request != null;
    boolean result;
    int patronageId;
    Patronage patronage;
    Patron patron;

    patronageId = request.getModel().getInteger("id");
    patronage = this.repository.findPatronageById(patronageId);
    patron = patronage.getPatron();
    result = patronage.getPublished() == false && request.isPrincipal(patron);

    return result;
}
```

(caso de PatronPatronage)

En este mal olor, no tenemos muy claro, qué es lo que SonarLint quiere que corrijamos. Suponemos que este mal olor se arreglaría poniendo "!" en vez de la comprobación "==" ya que eso promueve un código más limpio.

Conclusiones

En definitiva, podemos observar que tenemos muchos menos malos olores que en el entregable anterior y la mayoría de ellos son código redundante. Esto es debido al desconocimiento de la forma extraer dicho código en un método y que todo funcione correctamente. Tras este análisis nos hemos dado cuenta que el código redundante ensucia mucho la implementación y que deberíamos de haber dedicado más tiempo en arreglar ese mal olor.

Bibliografía

- Reporte Sonar Lint en Eclipse