

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Documentación del proyecto PetClinic

Refactoring




Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2019 – 2020


Fecha	Versión
03/06/2020	V2r00

Grupo de prácticas	G2-15 B.F.C.N
Autores	Rol
Franco Sánchez, Pablo	Desarrollador
Gutiérrez Prieto, Gabriel	Desarrollador
Lopez, Thibaut	Desarrollador
Rojas Gutiérrez, Rodrigo	Desarrollador
Romero Cáceres, Antonio	Desarrollador
Vidal Pérez, Antonio	Desarrollador

	Diseño y Pruebas II Documentación de refactoring
	Control de Versiones


Control de Versiones

Fecha	Versión	Descripción
30/05/2020	v1r00	Elaboración del primer borrador del documento.
01/06/2020	v1r10	Actualización puntos 1 y 2
03/06/2020	v2r00	Versión final

	<p>Diseño y Pruebas II Documentación de refactoring</p>
---	---

Índice de contenido

1. Refactoring Disease	2
2. Resultado de refactoring Disease	2
3. Refactoring de ChipController	6
4. Resultado del refactoring de ChipController	6
5. Refactoring de OpinionController	7
6. Resultado del refactoring de OpinionController.....	7

	<p>Diseño y Pruebas II Documentación de refactoring</p>
---	---

1. Refactoring Disease

Se han eliminados malos olores, para ello se ha hecho uso de SonarCloud, esta herramienta nos ha ayudado a identificarlos, unos de los más comunes eran los comentarios e imports innecesarios, y se han refactorizado los tests de servicio de enfermedad.

En estas refactorizaciones entra la eliminación de variables intermedias y uso del refactor de eclipse para crear un método llamado insert, que reutilizamos en los métodos:

- shouldInsertDisease
- shouldInsertNegativeDisease

2. Resultado de refactoring Disease

```
@Test
void shouldFindDiseaseById() {
    assertThat(this.diseaseService.findDiseaseById(1).getId()).isEqualTo(1);
}

@ParameterizedTest
@ValueSource(ints = {-10,-40} )
void shouldFindDiseaseByIdNegative(int argument){
    Disease disease = this.diseaseService.findDiseaseById(argument);
    Assertions.assertThrows(NullPointerException.class,()->{disease.getClass();} );
}

@Test
void shouldFindDiseaseAll() {

    assertThat(EntityUtils.getById(this.diseaseService.findAll(), Disease.class, 1).getCure())
        .isEqualTo("malisimo de la muerte");
    assertThat(EntityUtils.getById(this.diseaseService.findAll(), Disease.class, 4).getSeverity()).isEqualTo("MEDIUM");

}

public Disease Insert() {
    final Disease disease = new Disease();
    final Pet pet = new Pet();
    pet.setId(1);
    disease.setSeverity("LOW");
    disease.setCure("cure");
    disease.setSymptoms("esta mal");
    disease.setPet(pet);
    return disease;
}
```



```
@Test
void shouldInsertNegativeDisease(){

    final Disease d = Insert();
    d.setSeverity("");
    Validator validator = createValidator();
    Set<ConstraintViolation<Disease>> constraintViolations = validator.validate(d);
    assertThat(constraintViolations.size(),isEqualTo(1);
    ConstraintViolation<Disease> violation = constraintViolations.iterator().next();
    assertThat(violation.getPropertyPath().toString(),isEqualTo("severity");
    assertThat(violation.getMessage(),isEqualTo("must match LOW|MEDIUM|HIGH" );
}

@Test
@Transactional
void shouldUpdateDisease() {
    String oldSymptoms = this.diseaseService.findDiseaseById(1).getSymptoms();
    String newSymptoms = oldSymptoms + "XXXXXXXX";

    this.diseaseService.findDiseaseById(1).setSymptoms(newSymptoms);
    this.diseaseService.saveDisease(this.diseaseService.findDiseaseById(1));
    assertThat(this.diseaseService.findDiseaseById(1).getSymptoms(),isEqualTo(newSymptoms);
}

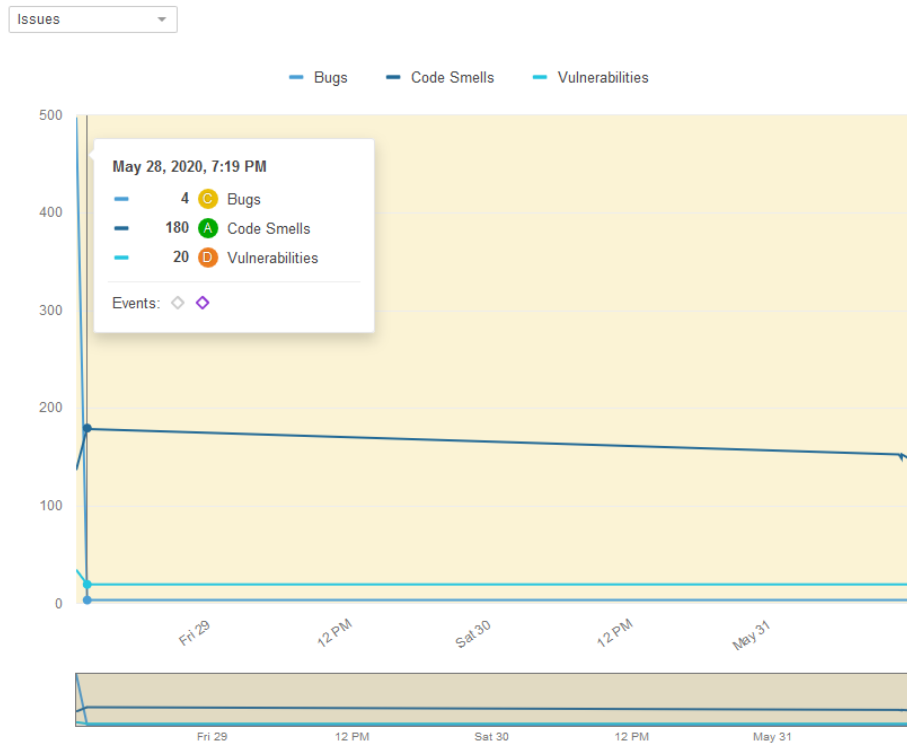
@ParameterizedTest
@ValueSource(ints = {1,4})
void shouldThrowExceptionUpdateDisease(int id){
    Disease disease = this.diseaseService.findDiseaseById(id);
    String newSymptoms = null;
    disease.setSymptoms(newSymptoms);
    this.diseaseService.saveDisease(disease);
    Assertions.assertThrows(NullPointerException.class, () -> {disease.getSymptoms().isEmpty();});
}

@Test
void shouldDeleteDisease() {
    Collection<Disease> found1 = this.diseaseService.findAll();
    this.diseaseService.delete(this.diseaseService.findDiseaseById(1));
    Collection<Disease> found2 = this.diseaseService.findAll();

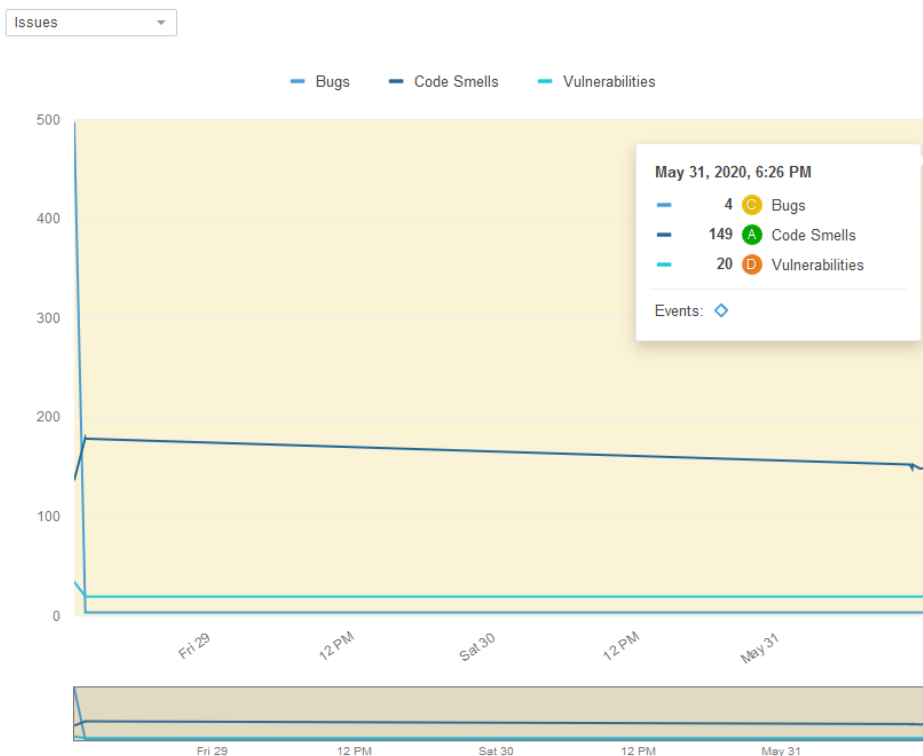
    assertTrue(found2.size() < found1.size());
}

@ParameterizedTest
@ValueSource(ints = {-16, -30})
void shouldThrowExceptionDeleteDisease(int id){
    Disease disease = this.diseaseService.findDiseaseById(id);
    Assertions.assertThrows(InvalidDataAccessApiUsageException.class, () -> {this.diseaseService.delete(disease);});
}
```

Esto sería antes de refactorizar:



Por último, esto sería el resultado obtenido después de haber refactorizado enfermedad:



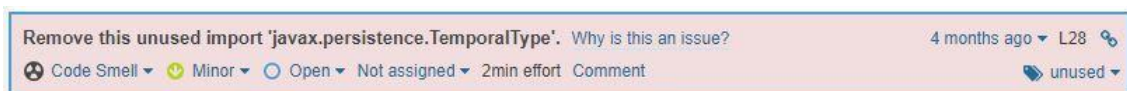
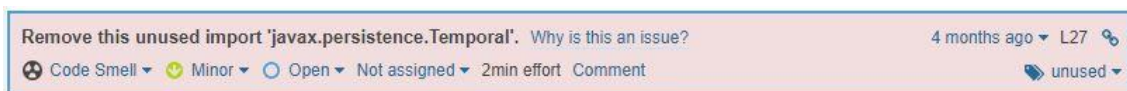
Como se puede apreciar hemos reducido en 31 los malos olores.

Estos son algunos tipos de malos olores encontrados en Enfermedad son de los siguientes tipos:

Dispensables: Se solucionaron eliminando dichas declaraciones en clases como DiseaseRepository.java, RoomRepository, entre otros.

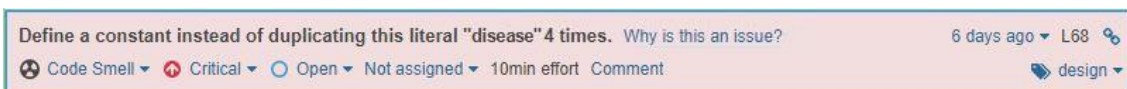


Dispensables: teníamos varios imports que no hacían falta en las clases, como los que se pueden observar en las imágenes. Se solucionaron rápidamente eliminándolos de Pet.java o Disease.java, entre otros.



Para solucionar este mal olor, se creó una constante Disease y se sustituyó los strings en los sitios donde indicaba SonarCloud.

```
private static final String Disease = "disease";
```



3. Refactoring de ChipController

Tras usar Sonar hemos detectado una serie de malos olores en la clase ChipController. Estos son los malos olores:

src/.../samples/petclinic/web/ChipController.java

<input type="checkbox"/>	Define a constant instead of duplicating this literal "petId" 3 times. Why is this an issue?	2 months ago	L47				Not assigned	8min effort	Comment	
<input type="checkbox"/>	Replace this use of System.out or System.err by a logger. Why is this an issue?	2 months ago	L58				Not assigned	10min effort	Comment	
<input type="checkbox"/>	Replace this use of System.out or System.err by a logger. Why is this an issue?	2 months ago	L60				Not assigned	10min effort	Comment	
<input type="checkbox"/>	Define a constant instead of duplicating this literal "redirect:/owners/{ownerId}" 3 times. Why is this an issue?	2 months ago	L63				Not assigned	8min effort	Comment	
<input type="checkbox"/>	Replace this use of System.out or System.err by a logger. Why is this an issue?	2 months ago	L96				Not assigned	10min effort	Comment	

Todos estos malos olores corresponden a la clase de dispensables. Los patrones que se usarán para solucionarlos son para los malos olores de tipo literal duplicado, crearemos una constante para sustituir los literales repetidos y para los de tipo System.out, la solución será eliminar esas llamadas debido a que se usaron mientras se depuraba el código mientras se realizaba y se abandonaron ahí.

4. Resultado del refactoring de ChipController

Las constantes creadas son: PET_ID para los "petId" y OWNER_REDIRECT para los "redirect:/owners/{ownerId}". Revisando manualmente hemos tomado la decisión de eliminar más código sin uso que quedaba tras eliminar los System.out debido a que eran para realizar pruebas.

```
private static final String PET_ID = "petId";
private static final String OWNER_REDIRECT = "redirect:/owners/{ownerId}";
```


5. Refactoring de OpinionController

Los siguientes malos olores han sido detectados en OpinionController:

src/.../samples/petclinic/web/OpinionController.java

<input type="checkbox"/>	Remove these unused method parameters. Why is this an issue?	2 months ago	L40		
	Code Smell Major Open Not assigned 5min effort Comment			cert, unused	
<input type="checkbox"/>	Define a constant instead of duplicating this literal "redirect:/opinions/listMine" 4 times. Why is this an issue?	last month	L84		
	Code Smell Critical Open Not assigned 10min effort Comment			design	
<input type="checkbox"/>	Replace this lambda with a method reference. Why is this an issue?	2 months ago	L105		
	Code Smell Minor Open Not assigned 2min effort Comment			java8	
<input type="checkbox"/>	Remove this useless assignment to local variable "ret". Why is this an issue?	last month	L125		
	Code Smell Major Open Not assigned 15min effort Comment			cert, cwe, unused	
<input type="checkbox"/>	Remove this unused "ret" local variable. Why is this an issue?	last month	L125		
	Code Smell Minor Open Not assigned 5min effort Comment			unused	
<input type="checkbox"/>	Remove this useless assignment to local variable "ret". Why is this an issue?	last month	L128		
	Code Smell Major Open Not assigned 15min effort Comment			cert, cwe, unused	

Todos pertenecen a la clase de dispensables, en dos casos son restos de código abandonado mientras se probaban distintas soluciones a errores encontrados (ret y parámetros), luego está la repetición de un literal y finalmente una expresión lambda que Sonar recomienda transformar en una referencia a método por pura comodidad.

6. Resultado del refactoring de OpinionController

Los malos olores se han resuelto de la siguiente manera: las repeticiones se han sustituido por la llamada a una constante, el código residual ha sido eliminado y la lambda expresión ha sido sustituida por una llamada a método tal y como recomienda Sonar.

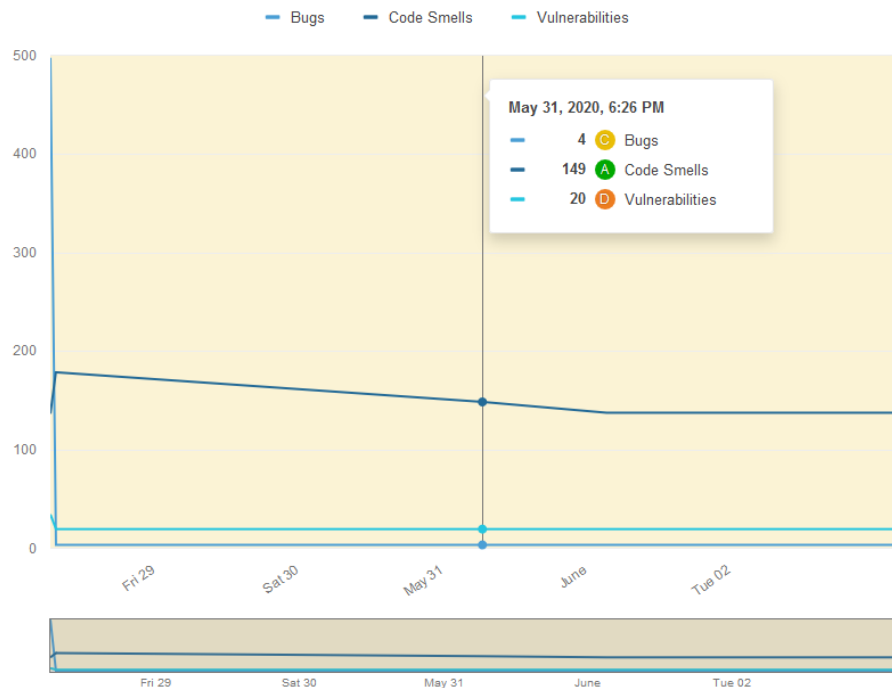
```

    opinions.forEach(opinionsList::add);

private static final String REDIRECT_LISTMINE = "redirect:/opinions/listMine";

```

En Sonar podemos ver la gráfica que nos muestra los cambios en la cantidad de errores y malos olores antes de los cambios realizados en OpinionController y ChipController:



Este es el gráfico después de las modificaciones, podemos observar que se han reducido los malos olores en 11, que son los corregidos en los apartados anteriores.

