

AMPLIACIÓN DE INGENIERÍA DEL SOFTWARE

3º INGENIERÍA INFORMÁTICA

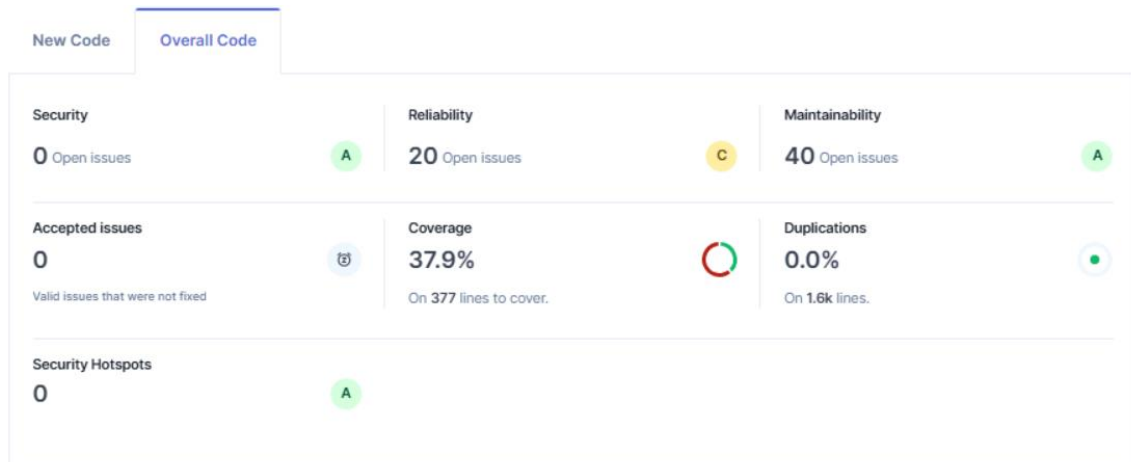
PRÁCTICA 2: PRUEBAS DE SISTEMA Y ANÁLISIS ESTÁTICO DE CÓDIGO

Por: Zaira Ruiz Fernández

ÍNDICE

1.Overview inicial con las principales métricas.....	3
2.Issues encontradas.	3
2.1.Inyección de dependencias por campo.	3
2.2.Declaración de una excepción no generada nunca.	5
2.3.Uso de un tipo de retorno no especificado, ambiguo.	6
2.4.Cadena de if/else y variable no utilizada.....	6
2.5.getFieldError() puede devolver null.....	7
2.6.Literal repetido tres veces.	8
2.7.El código de estado devuelto no refleja el estado de la operación.....	10
2.8.Cast innecesario.	11
2.9.Nombres de campos y métodos que no siguen el formato adecuado (camelCase).	12
2.10.film puede ser null.	13
2.11.Excepción poco clara.....	13
2.12.Lista mutable.	14
2.13.Visibilidad pública innecesaria en clases de test y sus métodos.....	15
3.Overview final con las principales métricas y la evaluación del nuevo código.....	16

1. Overview inicial con las principales métricas.



2. Issues encontradas.

2.1. Inyección de dependencias por campo.

The screenshot shows a detailed view of a SonarQube issue. The issue title is "Remove this field injection and use constructor injection instead." with a link to [java:S6813](#). It indicates the line affected is L21, with an effort of 5min and introduced 7 days ago. The issue is categorized as "Open", "Not assigned", and has "No tags".

On the right, it shows "Software qualities impacted": Reliability (orange icon) and Maintainability (orange icon). Below that, it lists "Clean code attribute": Consistency (blue icon) and Not conventional (blue icon).

The issue is linked to a code snippet in a Java file:

```
18 @Component
19 public class DatabaseInitializer {
20
21     @Autowired
22     private FilmService filmService;
23
24     @Autowired
```

Este problema se debe a que, al inyectar dependencias directamente en un atributo, a la hora de programar algún test, te obliga a tener las dependencias implementadas, ya que no se puede hacer un mock de ellas. En cambio, si inyectas las dependencias mediante el constructor de la clase, se podría hacer un mock de ellas y pasarle estos mocks al constructor, permitiendo ahora sí tests solitarios.

Este problema se daba varias veces, pero, por claridad y legibilidad en el documento, no voy a poner todos los casos, sólo el primero. Sin embargo, las soluciones a continuación sí son las de todos los casos.

```

private UserRepository userRepository;

@Autowired
public UserService(UserRepository userRepository){
    this.userRepository=userRepository;
}

private FilmService filmService;

private ReviewService reviewService;

@Autowired
public FilmRestController(FilmService filmService, ReviewService reviewService) {
    this.filmService=filmService;
    this.reviewService=reviewService;
}

```

```

private UserService userService;

private UserMapper userMapper;

@Autowired
public UserRestController(UserService userService, UserMapper userMapper){
    this.userService=userService;
    this.userMapper=userMapper;
}

```

```

private FilmService filmService;

private FavoriteFilmService favoriteFilmService;

private ReviewService reviewService;

@Autowired
public FilmWebController(FilmService filmService, FavoriteFilmService favoriteFilmService,
    ReviewService reviewService) {
    this.filmService = filmService;
    this.favoriteFilmService = favoriteFilmService;
    this.reviewService = reviewService;
}

```

```
private UserComponent userComponent;

private UserService userService;

@Autowired
public UserWebController(UserComponent userComponent, UserService userService) {
    this.userComponent = userComponent;
    this.userService = userService;
}
```

```
private UserComponent userComponent;
UserRepository userRepository;

@Autowired
public UserModelAttributes(UserComponent userComponent){
    this.userComponent=userComponent;
}

@Autowired
public UserComponent(UserRepository userRepository){
    this.userRepository=userRepository;
}
```

2.2.Declaración de una excepción no generada nunca.

Remove the declaration of thrown exception 'java.io.IOException', as it cannot be thrown from method's body. [java:S1130](#)

Exceptions in "throws" clauses should not be superfluous [java:S1130](#)

Line affected: L100 • Effort: 5min • Introduced: 8 days ago

☐ Open ☐ Not assigned ☐ error-handling ☐ Quick fix

Where is the issue? Why is this an issue? Activity

```

99
100     private FilmDTO saveFilmWithURLImage(CreateFilmRequest film, String localpath) throws IOException {
101
102         return filmService.save(film, imageUtils.localImageToBlob(localpath));
103     }

```

Remove the declaration of thrown exception 'java.io.IOException', as it cannot be thrown from method's body.

Software qualities impacted
Maintainability
Clean code attribute
Intentionality | Not clear

Dado que la excepción no se lanza nunca, se puede eliminar (hay otro caso más con este problema además de éste cuya solución se ve debajo).

```
private FilmDTO saveFilmWithURLImage(CreateFilmRequest film, String localpath) {
    return filmService.save(film, imageUtils.localImageToBlob(localpath));
}
```

```
@PostConstruct
public void init() {

    if (!isRunningTest()) {
        User michel = new User(name:"Michel", email:"michel.maes@urjc.es");
        User raul = new User(name:"Raúl", email:"raul@urjc.es");
    }
}
```

2.3. Uso de un tipo de retorno no especificado, ambiguo.

The screenshot shows a SonarLint issue titled "Remove usage of generic wildcard type." with a severity of "Error". The description states: "Generic wildcard types should not be used in return types [java:S1452](#)". The issue is located on line 21 of the code. The code snippet is as follows:

```
19  ~/  
20  @ExceptionHandler({ FilmNotFoundException.class, IllegalArgumentException.class, BindException.class })  
21  public ResponseEntity<?> handleException(Exception ex) {  
22  
    if (ex instanceof MethodArgumentNotValidException manvExp) {  
        // ...  
    }  
}
```

The issue is categorized under "Software qualities impacted" as "Maintainability". The "Clean code attribute" is "Intentionality | Not clear". The issue is "Open", "Not assigned", and has a "pitfall" tag. The "Where is the issue?" tab is selected, showing the code context. The "How can I fix it?" tab suggests: "Replace the chain of if/else with a switch expression."

Al no especificar el tipo de retorno, el código es mucho menos claro y se vuelve mucho más difícil saber qué tipo de datos se espera exactamente como respuesta. En este caso se devuelve o un String o nada (en el segundo caso), así que se puede cambiar ? por String.

```
@ExceptionHandler({ FilmNotFoundException.class, IllegalArgumentException.class, BindException.class })  
public ResponseEntity<String> handleException(Exception ex) {  
    if (ex instanceof MethodArgumentNotValidException manvExp) {  
        return ResponseEntity.badRequest().body(manvExp.getFieldError().getDefaultMessage());  
    } else if (ex instanceof FilmNotFoundException fnfExp) {  
        return ResponseEntity.notFound().build();  
    } else {  
        return ResponseEntity.badRequest().body(ex.getMessage());  
    }  
}
```

2.4. Cadena de if/else y variable no utilizada.

The screenshot shows a SonarLint issue titled "Replace the chain of if/else with a switch expression." with a severity of "Error". The description states: "Use switch instead of if-else chain to compare a variable against multiple cases [java:S6880](#)". The issue is located on line 22 of the code. The code snippet is as follows:

```
22  if (ex instanceof MethodArgumentNotValidException manvExp) {  
    // ...  
    return ResponseEntity.badRequest().body(manvExp.getFieldError().getDefaultMessage());  
23  }  
}
```

The issue is categorized under "Software qualities impacted" as "Maintainability". The "Clean code attribute" is "Intentionality | Not clear". The issue is "Open", "Not assigned", and has a "Quick fix" tag. The "Where is the issue?" tab is selected, showing the code context. The "How can I fix it?" tab suggests: "Replace the chain of if/else with a switch expression." and "A 'NullPointerException' could be thrown; 'getFieldError()' can return null."

Remove this unused "fnfExp" local variable. [java:S1481](#)

Unused local variables should be removed [java:S1481](#)

Line affected: L24 • Effort: 5min • Introduced: 8 days ago

☐ Open ☐ Not assigned ☐ unused ☐ Quick fix

Where is the issue? Why is this an issue? How can I fix it? Activity

```

23         return ResponseEntity.badRequest().body(manvExp.getFieldError().getDefaultMessage());
24     } else if (ex instanceof FilmNotFoundException fnfExp) {
25         return ResponseEntity.notFound().build();
26     }
  
```

A "NullPointerException" could be thrown; "getFieldError()" can return null.

Remove this unused "fnfExp" local variable.

Software qualities impacted: Maintainability

Clean code attribute: Intentionality | Not clear

Hay una cadena larga de if/else, por lo que la claridad del código sería mayor al sustituirlo por una expresión switch (que va a permitir que el contenido de dichos if/else se visualice mejor).

```

@ExceptionHandler({ FilmNotFoundException.class, IllegalArgumentException.class, BindException.class })
public ResponseEntity<String> handleException(Exception ex) {
    return switch (ex) {
        case MethodArgumentNotValidException manvExp ->
            ResponseEntity.badRequest().body(manvExp.getFieldError().getDefaultMessage());
        case FilmNotFoundException fnfExp -> ResponseEntity.notFound().build();
        default -> ResponseEntity.badRequest().body(ex.getMessage());
    };
}
  
```

Al utilizar el switch, la variable pasa a ser necesaria aunque no se utilice, así que ya no se puede quitar.

2.5.getFieldError() puede devolver null.

A "NullPointerException" could be thrown; "getFieldError()" can return null. [java:S2259](#)

Null pointers should not be dereferenced [java:S2259](#)

Line affected: L24 • Effort: 10min • Introduced: 5 minutes ago

☐ Open ☐ Not assigned ☐ cwe ... ☐ +

Where is the issue? Why is this an issue? How can I fix it? Activity More info

```

19     */
20     @ExceptionHandler({ FilmNotFoundException.class, IllegalArgumentException.class, BindException.class })
21     public ResponseEntity<String> handleException(Exception ex) {
22         return switch (ex) {
23             case MethodArgumentNotValidException manvExp ->
24                 ResponseEntity.badRequest().body(1 2 manvExp.getFieldError().getDefaultMessage());
  
```

A "NullPointerException" could be thrown; "getFieldError()" can return null.

Software qualities impacted: Reliability

Clean code attribute: Intentionality | Not logical

Como el método getFieldError() puede devolver null, intentar llamar al siguiente método (getDefaultMessage) podría provocar una excepción (ya que no se puede

llamar al método desde null). Para corregirlo, primero se hace una comprobación de si el objeto devuelto por getFieldError() es nulo o no y luego, en caso de no serlo, se llama al otro método. El aviso en sonar sigue saltando, pero como ahora es imposible que se lance la excepción, se puede clasificar el aviso como un falso positivo.

```
case MethodArgumentNotValidException manvExp ->
    ResponseEntity.badRequest()
        .body(manvExp.getFieldError() != null ? manvExp.getFieldError().getDefaultMessage() : "");
// ...
}
```

2.6.Literal repetido tres veces.

The screenshot shows a SonarLint warning in an IDE. The warning message is "Define a constant instead of duplicating this literal 'message' 3 times." It includes a link to "java:S1192" and states "Line affected: L22", "Effort: 8min", and "Introduced: 10 days ago". On the right, it shows "Software qualities impacted" with "Maintainability" highlighted in red, and "Clean code attribute" with "Adaptability" and "Not distinct" listed. Below the warning, there are tabs for "Where is the issue?", "Why is this an issue?", "How can I fix it?", and "Activity". The code snippet shows a Java method where the string literal "message" is used three times in the addObject method calls of a ModelAndView object.

```
21 ModelAndView modelAndView = new ModelAndView();
22 modelAndView.setViewName("message");
23 modelAndView.addObject("error", true);
24
25 if(ex instanceof MethodArgumentNotValidException manvExp){
26     modelAndView.addObject("message", manvExp.getFieldError().getDefaultMessage());
27 }else{
28     modelAndView.addObject("message", ex.getMessage());
29 }
30
31 return modelAndView;
```

Tener "message" tres veces va a aumentar el número de sitios en los que hay que modificar la cadena si se quiere cambiar por otra, lo que hace el código más largo y difícil de mantener (además es más fácil cometer un error al hacerlo).

Se soluciona añadiendo una variable que contenga "message" y que sea la que se use en lugar de la cadena directamente. Este problema también se daba en varias partes:


```

@ExceptionHandler({FilmNotFoundException.class, IllegalArgumentException.class, BindException.class})
public ModelAndView handleException(Exception ex){
    String message="message";
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName(message);
    modelAndView.addObject("error", true);

    if(ex instanceof MethodArgumentNotValidException manvExp){
        modelAndView.addObject(message, manvExp.getFieldError().getDefaultMessage());
    }else{
        modelAndView.addObject(message, ex.getMessage());
    }
}

```

```

private String filmNotFound="Film not found";
private String error="error";
private String action="action";
private String filmForm="filmForm";
private String redirectFilms="redirect:/films/";

return film;
} else {
    throw new RuntimeException(HttpStatus.NOT_FOUND, filmNotFound);
}

```

```

FilmDTO removedFilm = op.get();
model.addAttribute(error, false);
model.addAttribute("message", "Film " + removedFilm.title() + " deleted");
return "message";
} else {
    throw new RuntimeException(HttpStatus.NOT_FOUND, filmNotFound);
}

@GetMapping("/films/new")
public String newFilm(Model model) {
    model.addAttribute(action, "/films/new");
    model.addAttribute("film", new Film());
    model.addAttribute("ageRatings", AgeRating.values());
    return filmForm;
}

```

```

public String newFilmProcess(CreateFilmRequest film, MultipartFi
    FilmDTO newFilm = null;

    try {
        newFilm = filmService.save(film, imageField);
    } catch (IllegalArgumentException e) {
        model.addAttribute(error, true);
        model.addAttribute("errors", List.of(e.getMessage()));
        model.addAttribute(action, "/films/new");
        model.addAttribute("film", film);
        return filmForm;
    }

    return redirectFilms + newFilm.id();

@GetMapping("/films/{id}/edit")
public String editFilm(Model model, @PathVariable long id) {
    Optional<FilmDTO> op = filmService.findOne(id);
    if (op.isPresent()) {
        FilmDTO film = op.get();
        model.addAttribute(action, "/films/" + id + "/edit");
        model.addAttribute("film", film);
        model.addAttribute("ageRatings", AgeRatingOptionsUtils.getAgeRatingOptions());
        return filmForm;
    } else {
        throw new RuntimeException(HttpStatus.NOT_FOUND, filmNotFound);
    }
}

```

```

public class FilmWebController {
    public String editFilmProcess(Model model, @PathVariable long id) {
        FilmDTO updatedFilm = null;

        try {
            updatedFilm = filmService.update(id, film, imageField);
        } catch (RuntimeException e) {
            model.addAttribute(error, true);
            model.addAttribute("errors", List.of(e.getMessage()));
            model.addAttribute(action, "/films/" + id + "/edit");
            model.addAttribute("film", film);
            return filmForm;
        }

        model.addAttribute("film", updatedFilm);
        return redirectFilms + film.id();
    }

    public String addReview(@PathVariable
        FilmDTO film = reviewService.addReview(film);
        return redirectFilms + film.id();
    }

    @PostMapping("/films/{filmId}/reviews/{reviewId}")
    public String removeReview(@PathVariable
        FilmDTO film = reviewService.deleteReview(filmId, reviewId);
        return redirectFilms + film.id();
    }

    @PostMapping("/films/{filmId}/addFavorite")
    public String addFavorite(@PathVariable
        favoriteFilmService.addToFavorites(filmId);
        return redirectFilms + filmId;
    }
}

```

```

@PostMapping("/films/{filmId}/removeFavorite")
public String removeFavorite(@PathVariable
    favoriteFilmService.removeFromFavorites(filmId);
    return redirectFilms + filmId;
}

```

```

private String errorProcessingImage="Error at processing the image";

```

```

} catch (IOException e) {
    throw new RuntimeException(HttpStatus.BAD_REQUEST, errorProcessingImage);
}

} catch (IOException e) {
    throw new RuntimeException(HttpStatus.BAD_REQUEST, errorProcessingImage);
}

} catch (IOException e) {
    throw new RuntimeException(HttpStatus.BAD_REQUEST, errorProcessingImage);
}

```

2.7.El código de estado devuelto no refleja el estado de la operación.

Set a HttpStatus code reflective of the operation.

Set appropriate Status Codes on HTTP responses [java:S6863](#)

Line affected: L166 • Effort: 5min • Introduced: 10 days ago

☐ Open ☐ Not assigned ☐ spring ... +

Where is the issue? Why is this an issue? How can I fix it? Activity More info

nitflex-web-bbdd-rest > src/.../web/nitflex/controller/web/FilmWebController.java [Open in IDE](#) [See all issues in this file](#)

```

161         try {
162             poster = new InputStreamResource(FilmService.getPosterFile(id));
163         } catch (Exception e) {
164             ClassPathResource resource = new ClassPathResource("static/images/no-image.png");
165             byte[] imageBytes = resource.getInputStream().readAllBytes();
166             return ResponseEntity.ok().header(HttpHeaders.CONTENT_TYPE, "image/jpeg").body(imageBytes);

```

Set a HttpStatus code reflective of the operation.

Software qualities impacted

- Reliability

Clean code attribute

- Adaptability | Not distinct

Aquí se devuelve siempre el código 200 OK, pero esto puede no ser correcto, porque la operación podría fallar al no encontrar lo que se busca. Sin embargo, aquí lo que se busca es una imagen que está cargada en una carpeta del proyecto, por lo que siempre la encontrará, devolviendo un código 200 OK. Debido a esto, lo podemos considerar un falso positivo.

2.8.Cast innecesario.

The screenshot shows a SonarQube issue report for a Java code snippet. The issue title is "Replace this instanceof check and cast with 'instanceof Film f'". The description states: "Pattern Matching for 'instanceof' operator should be used instead of simple 'instanceof' + cast [java:S6201](#)". It also provides metadata: "Line affected: L147", "Effort: 5min", and "Introduced: 10 days ago". On the right, a sidebar indicates "Software qualities impacted" with a yellow circle for "Maintainability" and a blue circle for "Clean code attribute" with "Intentionality" marked as "Not clear". Below the issue title, there are tabs: "Where is the issue?", "Why is this an issue?", "Activity", and "More info". The "Where is the issue?" tab is selected, showing a code editor with lines 142 to 153. The code is as follows:

```
142  
143  
144 @Override  
145 public boolean equals(Object o){  
146     if(o == null) return false;  
147     if(o == this) return true;  
148     if(o instanceof Film){  
149         Film f = (Film) o;  
150         return f.getId().equals(this.id);  
151     }  
152     return false;  
153 }
```

A red box highlights the issue on line 147, with the message "Replace this instanceof check and cast with 'instanceof Film f'".

En este código el cast es innecesario y redundante porque ya se está haciendo en la comprobación instanceof, por lo que se puede declarar la variable Film f directamente ahí y usarla después.

```
if(o == this) return true;  
if(o instanceof Film f){  
    return f.getId().equals(this.id);  
}
```

2.9. Nombres de campos y métodos que no siguen el formato adecuado (camelCase).

The screenshot shows a SonarQube issue page. At the top, the title reads: "Rename this field 'created_at' to match the regular expression '[a-z][a-zA-Z0-9]*\$'". Below the title, it states: "Field names should comply with a naming convention [java:S116](\"#\")". Further down, it indicates: "Line affected: L39 • Effort: 2min • Introduced: 10 days ago". On the right side, there are two panels: "Software qualities impacted" showing "Maintainability" with a yellow smiley face icon, and "Clean code attribute" showing "Consistency" with a blue "Not identifiable" label. Below these panels are four tabs: "Where is the issue?", "Why is this an issue?", "Activity", and "More info". The "Where is the issue?" tab is selected, showing a code editor with the following Java code:

```
34  
35 @ManyToOne  
36 private Film film;  
37  
38 @CreationTimestamp  
39 private Date created_at;  
  
40  
41 public Review(){  
42     this.id = 0L;  
43 }  
44
```

A red box highlights the field `created_at` on line 39, with the message: "Rename this field 'created_at' to match the regular expression '[a-z][a-zA-Z0-9]*\$'".

Para solucionarlo, simplemente se eliminan las barras bajas y se pone la primera letra de la segunda palabra en mayúscula, pasando el nombre a camelCase. Este problema también se da en el nombre de un método de la misma clase.

```
@CreationTimestamp  
private Date createdAt;  
  
public Date getCreatedAt() {  
    return createdAt;  
}
```

2.10.film puede ser null.

A "NullPointerException" could be thrown; "film" is nullable here. [java:S2259](#)

Null pointers should not be dereferenced

Line affected: L111 • Effort: 10min • Introduced: 10 days ago

Software qualities impacted: Reliability

Clean code attribute: Intentionality | Not logical

Where is the issue? Why is this an issue? How can I fix it? Activity More info

```
106
107
108     if (!id.equals(review.id)) return false;
109     if (score != review.score) return false;
110     if (!text.equals(review.text)) return false;
111     if (!user.equals(review.user)) return false;
112     return ((1) film == null) && (review.film == null) || 2 film.equals(review.film);
113
114 }
115
116 @Override
117 public int hashCode() {
```

Como la variable film puede ser null, intentar llamar al método equals desde ella (o desde review.film, que también podría ser null), podría provocar una excepción (ya que no se puede llamar al método desde null). Para corregirlo, primero se hace una comprobación de si la variable film es nula o no y luego, en caso de no serlo, se llama al método equals (aquí se puede hacer mediante &&, ya que no pasa a la segunda condición si la primera no se cumple).

```
117 if (!user.equals(review.user)) return false;
118 return ((film == null) && (review.film == null)) || ((film != null) && film.equals(review.film));
```

2.11.Excepción poco clara.

Define and throw a dedicated exception instead of using a generic one. [java:S112](#)

Generic exceptions should never be thrown

Line affected: L53 • Effort: 20min • Introduced: 10 days ago

Software qualities impacted: Maintainability

Clean code attribute: Intentionality | Not complete

Where is the issue? Why is this an issue? How can I fix it? Activity More info

```
48
49
50     .orElseThrow(() -> new FilmNotFoundException(id));
51     Blob blob = film.getPosterFile();
52     try {
53         return blob.getBinaryStream();
54     } catch (SQLException e) {
55         throw new RuntimeException("Error getting image from database", e);
56     }
57 }
```

La excepción RuntimeException no es muy específica, lo que podría dificultar la

comprensión del código y de los problemas que surjan. Para solucionarlo, se cambia por una excepción más específica.

```
} catch (SQLException e) {  
    throw new ImageFromDatabaseException(message:"Error getting image from database", e);  
}  
  
package es.codeurjc.web.nitflex.service.exceptions;  
  
public class ImageFromDatabaseException extends RuntimeException {  
    public ImageFromDatabaseException(String message, Throwable cause) {  
        super(message, cause);  
    }  
}
```

2.12.Lista mutable.

Replace this usage of 'Stream.collect(Collectors.toList())' with 'Stream.toList()' and ensure that the list is unmodified. ⓘ

"Stream.toList()" method should be used instead of "collectors" when unmodifiable list needed [java:S6204](#) ⓘ

Line affected: L36 • Effort: 5min • Introduced: 10 days ago

☐ Open ▾ ☐ Not assigned ▾ ☐ java16 +

Where is the issue?	Why is this an issue?	Activity
32 33 34 35 36 37 38 39 40	<pre>ageRating -> new AgeRatingOption(ageRating.getDescription(), ageRating.getDescription().equals(selectedAgeRating))) .collect(Collectors.toList());</pre> <p>Replace this usage of 'Stream.collect(Collectors.toList())' with 'Stream.toList()' and ensure that the list is unmodified.</p>	

.collect(Collectors.toList()) devuelve una lista que se puede modificar, lo que es poco seguro y puede dar lugar a errores. Para hacerla inmutable, basta con utilizar .toList().

```
return Stream.of(AgeRating.values())  
    .map(  
        ageRating -> new AgeRatingOption(  
            ageRating.getDescription(),  
            ageRating.getDescription().equals(selectedAgeRating)  
        )  
    )  
    .toList();
```


2.13. Visibilidad pública innecesaria en clases de test y sus métodos.

The screenshot shows a SonarQube issue report for the rule 'Remove this 'public' modifier.' The issue is located in a Java file, specifically in a test class. The code snippet shows imports and the start of a test class: `import es.codeurjc.web.nitflex.repository.UserRepository;`, `import es.codeurjc.web.nitflex.service.FilmService;`, `import es.codeurjc.web.nitflex.service.exceptions.FilmNotFoundException;`, `import es.codeurjc.web.nitflex.utils.ImageUtils;`, and `public class FilmServiceTest {`. A red box highlights the `public` keyword in the class declaration, with a tooltip that says 'Remove this 'public' modifier.' The right sidebar shows 'Software qualities impacted' with 'Maintainability' at level 2, and 'Clean code attribute' with 'Intentionality' as 'Not clear'.

Al ser un test, se asume directamente la visibilidad pública, por lo que especificarlo no es necesario. Esto se da en las clases de test de integración y unitarios y no es realmente un problema, pero basta con quitar la visibilidad de las clases y métodos para solucionarlo.

The image displays four code snippets from a dark-themed editor. The first snippet shows `@SpringBootTest` above `class FilmServiceTest {`. The second snippet shows `@BeforeEach` above `void setup(){`. The third snippet shows `@Test` above `void save_film_without_image_and_with_valid_title(){`. The fourth snippet shows `@Test` above `void update_title_and_synopsis_of_film_with_image(){`. In all cases, the `public` modifier has been removed.

Esto es sólo la solución en una de las clases, pero con la otra es exactamente igual (no está puesta por claridad de la memoria).

3. Overview final con las principales métricas y la evaluación del nuevo código.

