2022

TAHS – Assignment 3 Mutation Testing

GROUP 13A

AUTHORS:

- ANDY LI
- TANESHWAR PARANKUSAM
- MEHMET SÖZÜDÜZ
- MIHNEA TOADER
- JEGOR ZELENJAK
- JEROEN BASTENHOF

Table of Contents

1 Mutation testing tool selection

We have chosen PIT[1] to perform mutation testing on our microservices. PIT is a popular mutation testing tool that introduces mutations (small changes in code) to existing source code and runs these against the original test suite. PIT offers a Gradle plugin that allows the tool to be easily integrated in the current project setup, which also uses Gradle to facilitate the building process.

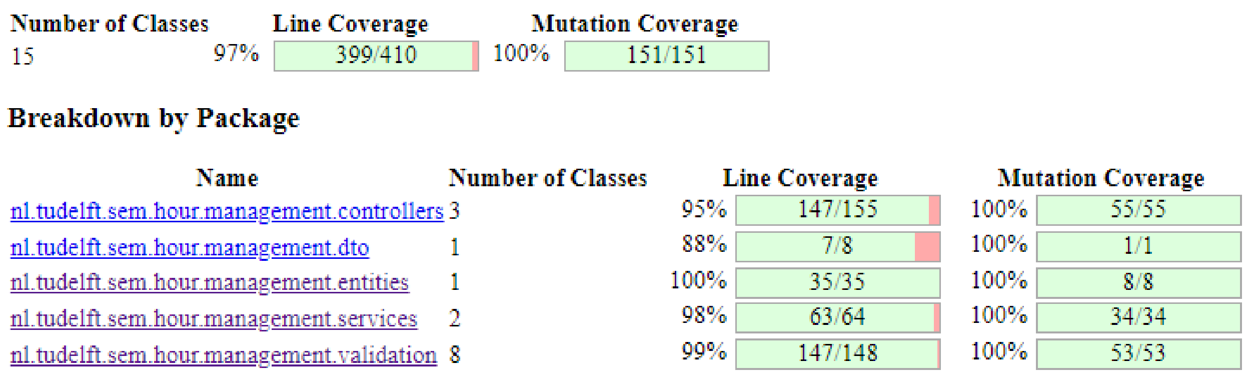
Note: PIT does not ignore the line coverage for Lombok annotated variables. Hence, the line coverage might turn out slightly lower than expected. JaCoCo ignores variables annotated with a Lombok annotation by default and is therefore more accurate in terms of line/branch coverage.

[1] See https://pitest.org/.

2 Identification of classes that require better mutation score

To find all classes that require a better mutation score, we ran PIT against each microservice individually. Based on this, we found that the gateway microservice, the users microservice as well as the hour management microservice, already achieved a mutation coverage score of *100%*, see images below. This narrowed our search down to the following microservices.

- Authentication microservice;
- Course microservice;
- Hiring procedure microservice.

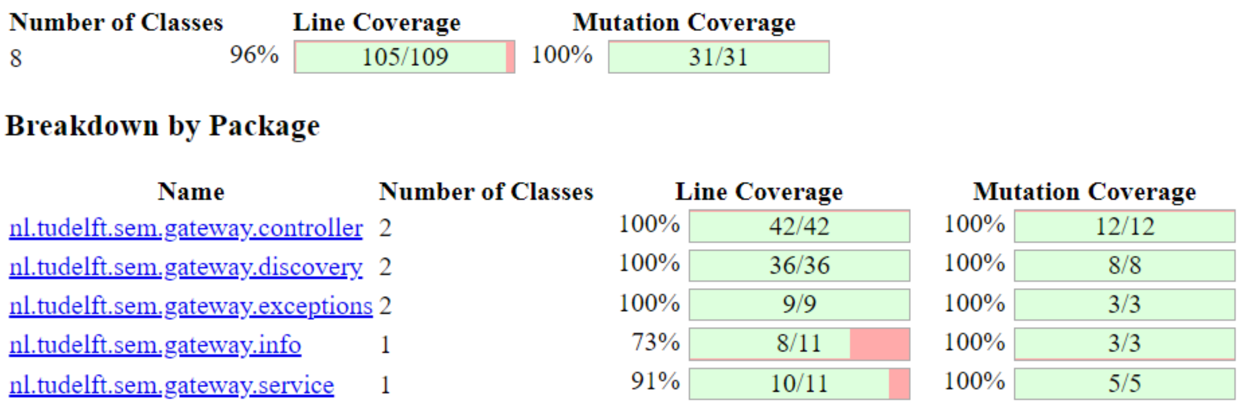


Number of Classes	Line Coverage		Mutation Coverage	
15	97%	399/410	100%	151/151

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.sem.hour.management.controllers	3	95%	147/155	100%	55/55
nl.tudelft.sem.hour.management.dto	1	88%	7/8	100%	1/1
nl.tudelft.sem.hour.management.entities	1	100%	35/35	100%	8/8
nl.tudelft.sem.hour.management.services	2	98%	63/64	100%	34/34
nl.tudelft.sem.hour.management.validation	8	99%	147/148	100%	53/53

Figure 1: Mutation coverage for the hour management microservice.

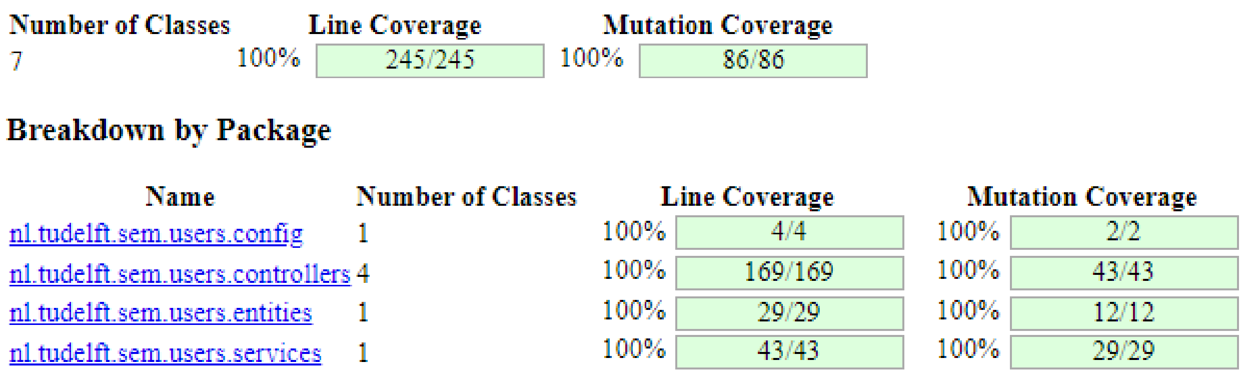


Number of Classes	Line Coverage		Mutation Coverage	
8	96%	105/109	100%	31/31

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.sem.gateway.controller	2	100%	42/42	100%	12/12
nl.tudelft.sem.gateway.discovery	2	100%	36/36	100%	8/8
nl.tudelft.sem.gateway.exceptions	2	100%	9/9	100%	3/3
nl.tudelft.sem.gateway.info	1	73%	8/11	100%	3/3
nl.tudelft.sem.gateway.service	1	91%	10/11	100%	5/5

Figure 2: Mutation coverage for the gateway microservice.



Number of Classes	Line Coverage		Mutation Coverage	
7	100%	245/245	100%	86/86

Breakdown by Package

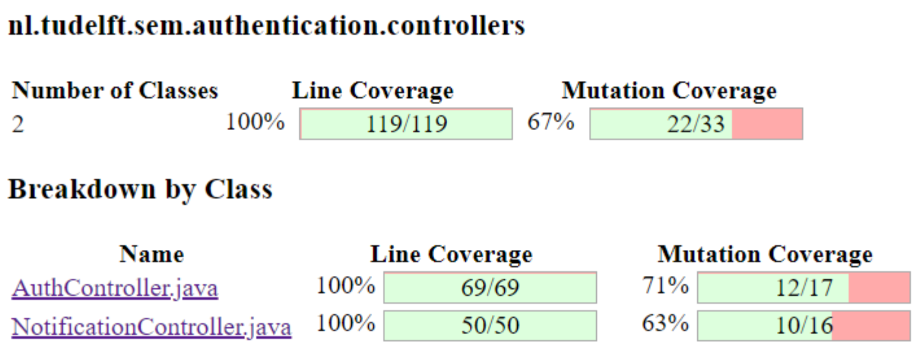
Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.sem.users.config	1	100%	4/4	100%	2/2
nl.tudelft.sem.users.controllers	4	100%	169/169	100%	43/43
nl.tudelft.sem.users.entities	1	100%	29/29	100%	12/12
nl.tudelft.sem.users.services	1	100%	43/43	100%	29/29

Figure 3: Mutation coverage for the users microservice.

We mostly focused on the classes that had the lowest mutation score. Using the metrics generated by PIT, we came across certain classes that require a better mutation score. The selected classes, as well as their mutation coverage score, can be found in the subsections below.

2.1 [Authentication] AuthController

As we can see in the image of the report generated by PIT below, the mutation score of the *AuthController* class is at *71%*, this needs to be increased to at least *91%*.



nl.tudelft.sem.authentication.controllers

Number of Classes	Line Coverage		Mutation Coverage	
2	100%	119/119	67%	22/33

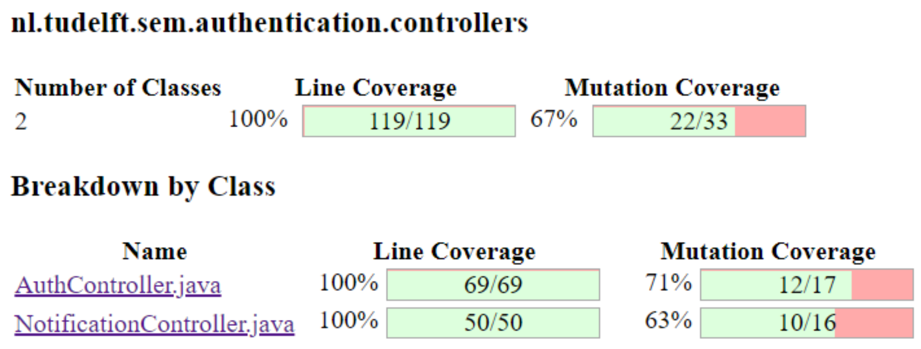
Breakdown by Class

Name	Line Coverage		Mutation Coverage	
AuthController.java	100%	69/69	71%	12/17
NotificationController.java	100%	50/50	63%	10/16

Figure 4: Mutation coverage score of the AuthController class.

2.2 [Authentication] NotificationController

As we can see in the image of the report generated by PIT below, the mutation score of the *NotificationController* is at *63%*, this needs to be increased to at least *83%*.



nl.tudelft.sem.authentication.controllers

Number of Classes	Line Coverage		Mutation Coverage	
2	100%	119/119	67%	22/33

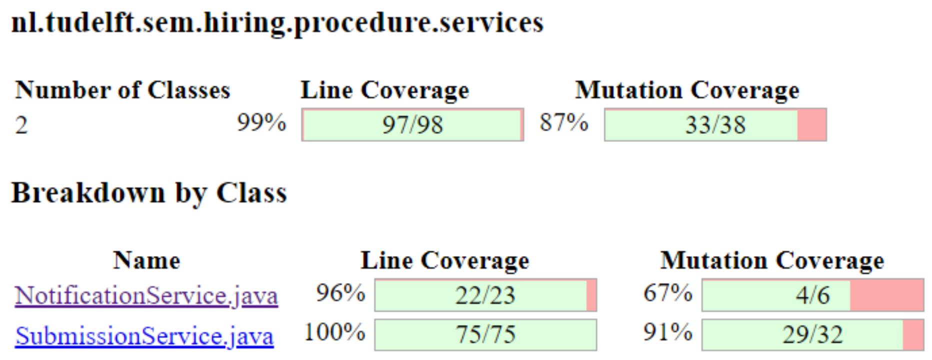
Breakdown by Class

Name	Line Coverage		Mutation Coverage	
AuthController.java	100%	69/69	71%	12/17
NotificationController.java	100%	50/50	63%	10/16

Figure 5: Mutation coverage score of the NotificationController class.

2.3 [Hiring-Procedure] NotificationService

As we can see in the image of the report generated by PIT below, the mutation score of the *NotificationService* class is *67%*, this needs to be increased to at least *87%*.



nl.tudelft.sem.hiring.procedure.services

Number of Classes	Line Coverage		Mutation Coverage	
2	99%	97/98	87%	33/38

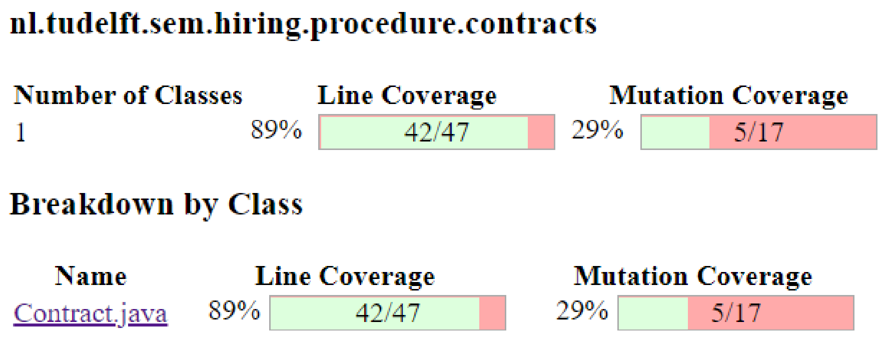
Breakdown by Class

Name	Line Coverage		Mutation Coverage	
NotificationService.java	96%	22/23	67%	4/6
SubmissionService.java	100%	75/75	91%	29/32

Figure 6: Mutation coverage score of the NotificationService class.

2.4 [Hiring-Procedure] Contract

As we can see in the image of the report generated by PIT below, the mutation score of the *Contract* class is at *29%*, this needs to be increased to at least *49%*.



nl.tudelft.sem.hiring.procedure.contracts

Number of Classes	Line Coverage		Mutation Coverage	
1	89%	42/47	29%	5/17

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Contract.java	89%	42/47	29%	5/17

Figure 7: Mutation coverage score of the Contract class.

3 Improving mutation score

This section describes the improvements that have been done to each of the identified classes in chapter 2. We do this by showing the mutation report *before* the mutation score was improved, as well as the changes that were performed to the respective classes. At last, we show the improved mutation score for the class which is incremented by **at least** 20%.

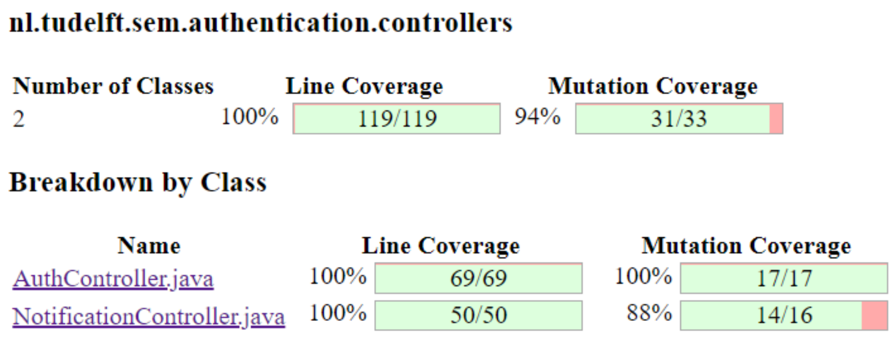
3.1 [Authentication] AuthController

See this commit for the changes and this merge request (!120) for more information.

We managed to increase the mutation score from 71% to 100% by modifying the following 3 tests:

- *testLoginSuccessfullyWithNotifications()*
- *testLoginSuccessfullyWithNoNotifications()*
- *testChangePasswordSuccessfully()*

And killing 5 mutants in the process, which required tests to fail when a password was successfully changed, but not actually verified. As well as checking if the response contained the necessary messages upon login (such as the notifications or a message displaying that there were no new notifications). See the Pitest report below.



nl.tudelft.sem.authentication.controllers

Number of Classes	Line Coverage		Mutation Coverage	
2	100%	119/119	94%	31/33

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
AuthController.java	100%	69/69	100%	17/17
NotificationController.java	100%	50/50	88%	14/16

Figure 8: Mutation coverage score of the AuthController class after improving the test suite.

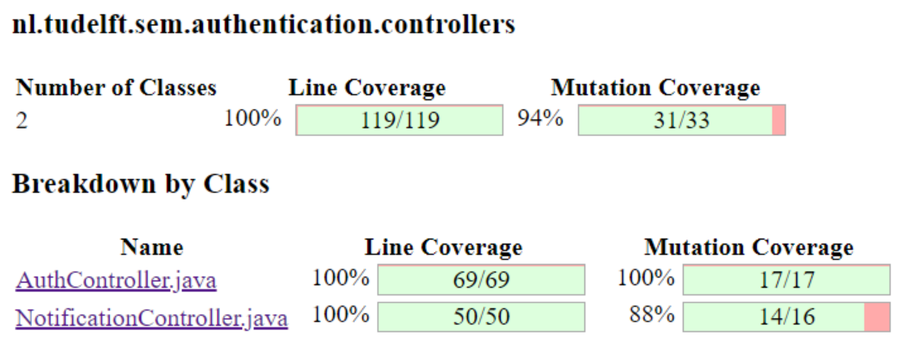
3.2 [Authentication] NotificationController

See this commit for the changes and this merge request (!120) for more information.

We managed to increase the mutation score from 63% to 88% by adding the following 4 tests:

- *testChangeUserFromNotificationAsStudentFailed()*
- *testChangeMessageFromNotificationAsStudentFailed()*
- *testDeleteExistingNotificationByIdAsStudentFailed()*
- *testDeleteExistingNotificationByUserIdAsStudentFailed()*

And killing 4 mutants in the process, which required the tests to fail if the user was a student and tried to make use of the changing notifications and deletion of notifications endpoints. See the Pitest report below.



nl.tudelft.sem.authentication.controllers

Number of Classes	Line Coverage		Mutation Coverage	
2	100%	119/119	94%	31/33

Breakdown by Class

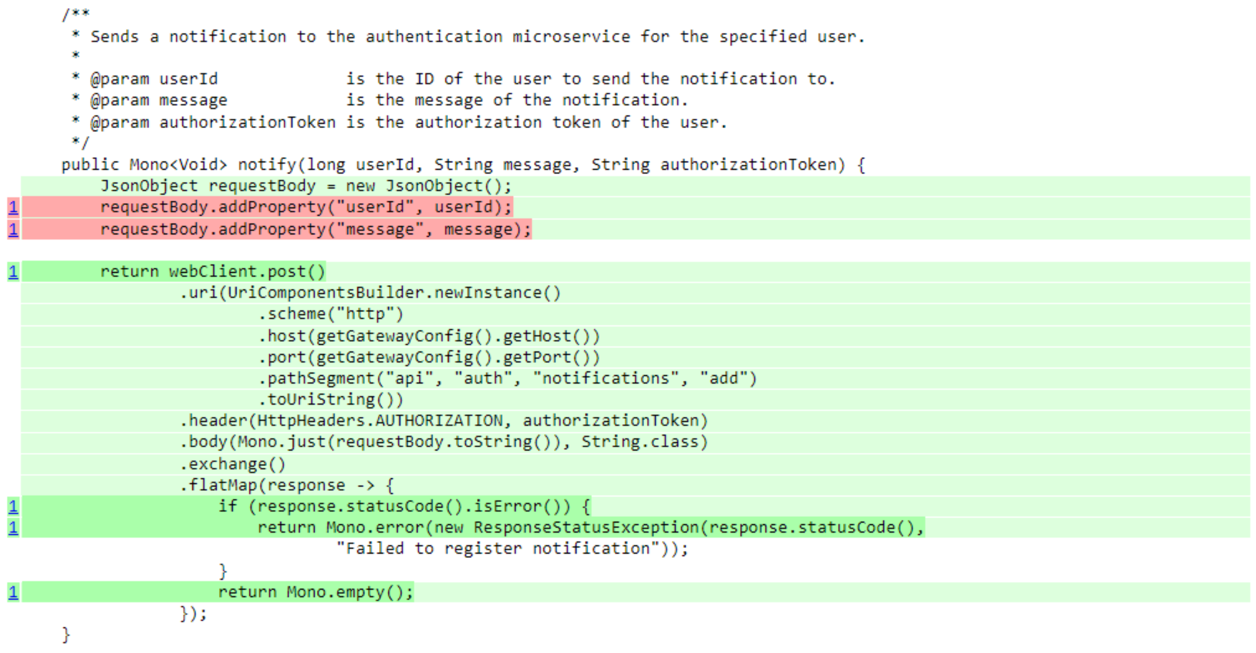
Name	Line Coverage		Mutation Coverage	
AuthController.java	100%	69/69	100%	17/17
NotificationController.java	100%	50/50	88%	14/16

Figure 9: Mutation coverage score of the NotificationController class after improving the test suite.

3.3 [Hiring-Procedure] NotificationService

See this commit for the changes and this merge request (!121) for more information.

As can be seen in Figure 6, the mutation score for the *NotificationService* class **before** refactoring is *4/6 (67%)*. The mutants that are not caught by our test suite can be found in Figure 10. The lines of code that are highlighted with red are mutants created by PIT where both calls to the *addProperty* method got removed (separately per test suite iteration). The removal of these calls affects the body that is sent to the endpoint for registering notifications for a particular user. This mutation can be tackled by verifying that the body contains the correct attributes.



```
    /**
     * Sends a notification to the authentication microservice for the specified user.
     *
     * @param userId             is the ID of the user to send the notification to.
     * @param message            is the message of the notification.
     * @param authorizationToken is the authorization token of the user.
     */
    public Mono<Void> notify(long userId, String message, String authorizationToken) {
        JsonObject requestBody = new JsonObject();
1       requestBody.addProperty("userId", userId);
1       requestBody.addProperty("message", message);

1       return webClient.post()
                .uri(UriComponentsBuilder.newInstance()
                        .scheme("http")
                        .host(getGatewayConfig().getHost())
                        .port(getGatewayConfig().getPort())
                        .pathSegment("api", "auth", "notifications", "add")
                        .toUriString())
                .header(HttpHeaders.AUTHORIZATION, authorizationToken)
                .body(Mono.just(requestBody.toString()), String.class)
                .exchange()
                .flatMap(response -> {
1                   if (response.statusCode().isError()) {
1                       return Mono.error(new ResponseStatusException(response.statusCode(),
                                "Failed to register notification"));
                    }
1                   return Mono.empty();
                });
    }
```

Figure 10: Survived mutants of the notify method in the NotificationService class.

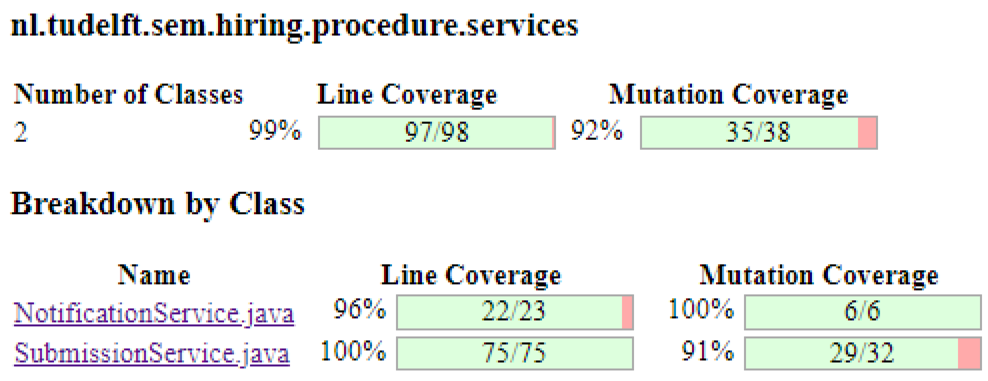
In order to kill the survived mutants above, we modified **existing** tests for the *notify* method. These tests now verify that the request body of the recorded requests of our mocked web server matches the expected request body, which is a JSON object consisting of a user ID and a message (see Code block 1).

```json
{
    "userId": 1234567,
    "message": "my new notification message"
}
```

Code block 1: Example of a JSON request body to create a new notification.

The details of the changes to the *NotificationService* test suite can be found in the aforementioned merge request and commit.

After having implemented the changes, PIT now reports a mutation coverage score of *100%* for the *NotificationService* class, see Figure 11.



nl.tudelft.sem.hiring.procedure.services

Number of Classes	Line Coverage		Mutation Coverage	
2	99%	97/98	92%	35/38

Breakdown by Class

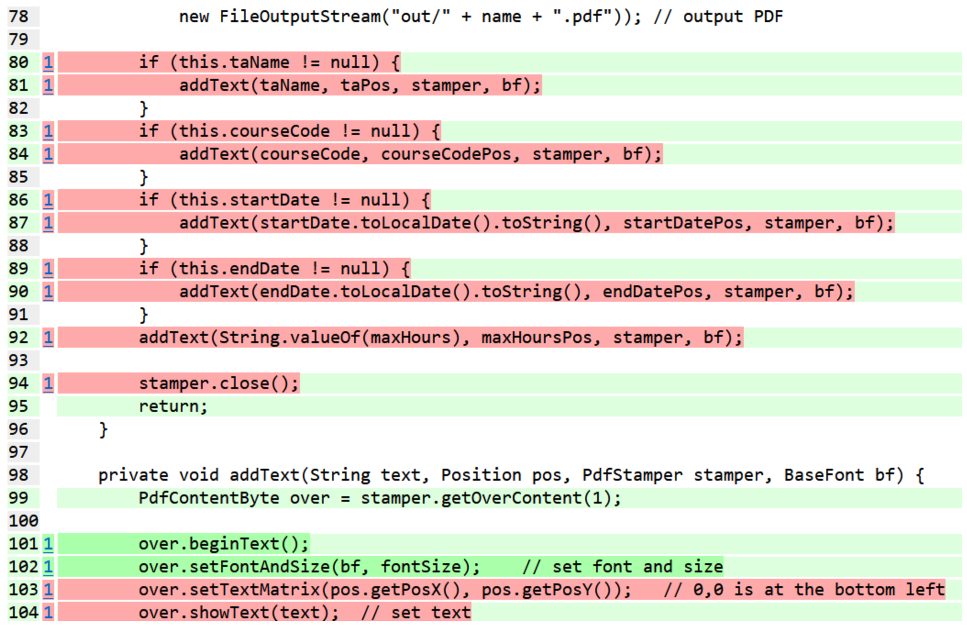
Name	Line Coverage		Mutation Coverage	
NotificationService.java	96%	22/23	100%	6/6
SubmissionService.java	100%	75/75	91%	29/32

Figure 11: Mutation coverage score of the NotificationService class after improving the test suite.

3.4 [Hiring-Procedure] Contract

See this commit for changes and this merge request (!123) for more information.

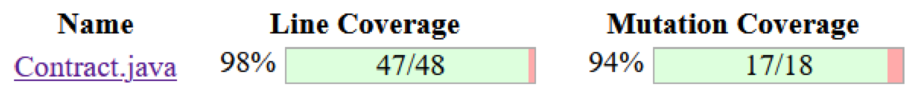
Figure 7 shows the poor mutation scores for the *Contract* class. As can be seen in Figure 12, the main cause of this is not checking whether the requested text got added or not to the generated pdf. Consequently, the mutations where the *addText()* calls are removed or the conditionals in the if-statements are negated, are not killed.



```
                new FileOutputStream("out/" + name + ".pdf")); // output PDF

        if (this.taName != null) {
            addText(taName, taPos, stamper, bf);
        }
        if (this.courseCode != null) {
            addText(courseCode, courseCodePos, stamper, bf);
        }
        if (this.startDate != null) {
            addText(startDate.toLocalDate().toString(), startDatePos, stamper, bf);
        }
        if (this.endDate != null) {
            addText(endDate.toLocalDate().toString(), endDatePos, stamper, bf);
        }
        addText(String.valueOf(maxHours), maxHoursPos, stamper, bf);

        stamper.close();
        return;
    }

    private void addText(String text, Position pos, PdfStamper stamper, BaseFont bf) {
        PdfContentByte over = stamper.getOverContent(1);

        over.beginText();
        over.setFontAndSize(bf, fontSize);    // set font and size
        over.setTextMatrix(pos.getPosX(), pos.getPosY());   // 0,0 is at the bottom left
        over.showText(text);  // set text
```

Figure 12: Survived mutants of the Contract class before improvement.

To solve this issue, a new *PdfReader* object must be created from the generated byte array, which is used to extract all the text in the document. After extracting the text, a simple *contains()* check for all the variables of the *Contract* object will kill the mutants, as it validates their existence in the generated PDF. These changes raise the mutation score from *29%* to *94%*:



Name	Line Coverage		Mutation Coverage	
Contract.java	98%	47/48	94%	17/18

Figure 13: Mutation coverage score of the Contract class after improving the method and test suite.

3.5 Score after improvements

Below is a quick overview of the mutation scores per class before and after the changes to the code/test suite were made.

Table 1: Mutation coverage score before and after changing the code/test suite.

Class	Mutation Score Before	Mutation Score After
AuthController.java [authentication]	71%	100%
NotificationController.java [authentication]	63%	88%
NotificationService.java [hiring-procedure]	67%	100%
Contract.java [hiring-procedure]	29%	94%