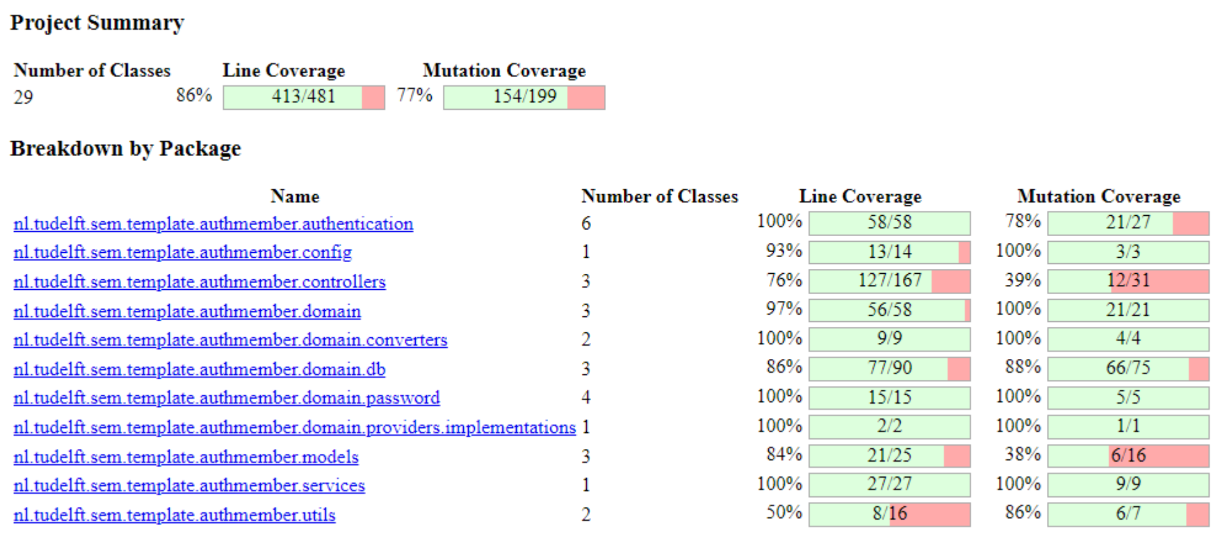
Group 22a Assignment 3 - Mutation Testing

Task 1: Automated Mutation Testing

We will start off by running the already integrated PiTest framework for each of our microservices.

AuthMember MS:



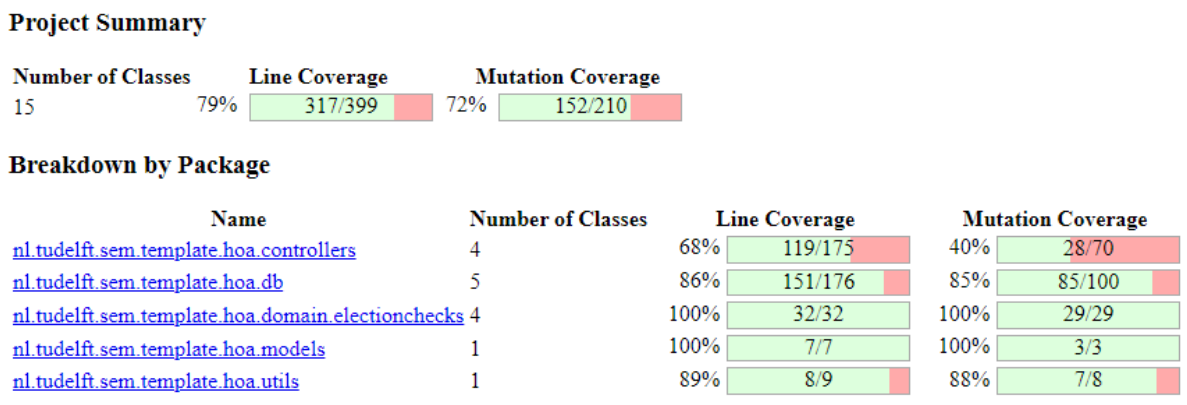
Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
29	86%	413/481	77%	154/199

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.sem.template.authmember.authentication	6	100%	58/58	78%	21/27
nl.tudelft.sem.template.authmember.config	1	93%	13/14	100%	3/3
nl.tudelft.sem.template.authmember.controllers	3	76%	127/167	39%	12/31
nl.tudelft.sem.template.authmember.domain	3	97%	56/58	100%	21/21
nl.tudelft.sem.template.authmember.domain.converters	2	100%	9/9	100%	4/4
nl.tudelft.sem.template.authmember.domain.db	3	86%	77/90	88%	66/75
nl.tudelft.sem.template.authmember.domain.password	4	100%	15/15	100%	5/5
nl.tudelft.sem.template.authmember.domain.providers.implementations	1	100%	2/2	100%	1/1
nl.tudelft.sem.template.authmember.models	3	84%	21/25	38%	6/16
nl.tudelft.sem.template.authmember.services	1	100%	27/27	100%	9/9
nl.tudelft.sem.template.authmember.utils	2	50%	8/16	86%	6/7

HOA MS:



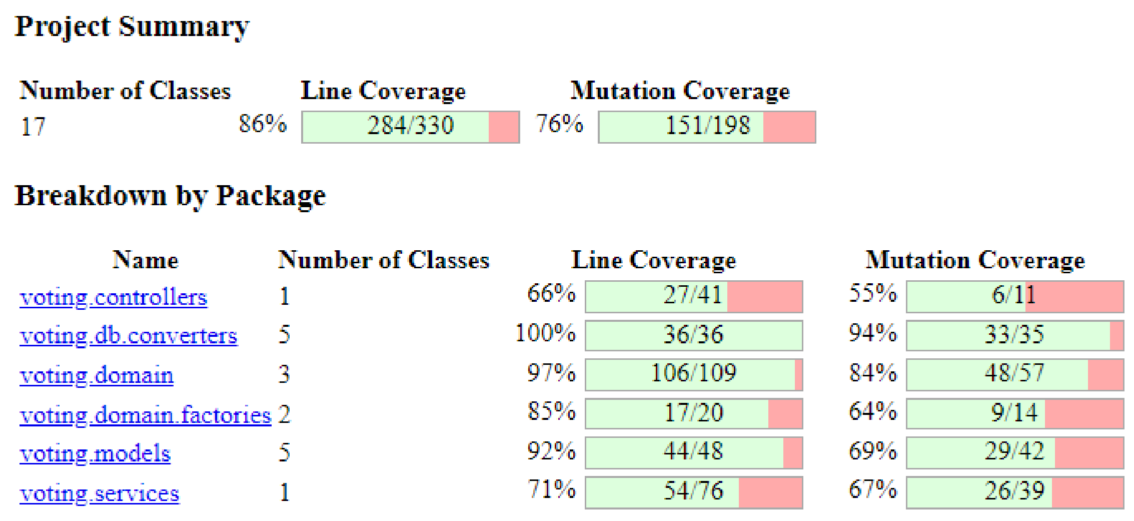
Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
15	79%	317/399	72%	152/210

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
nl.tudelft.sem.template.hoa.controllers	4	68%	119/175	40%	28/70
nl.tudelft.sem.template.hoa.db	5	86%	151/176	85%	85/100
nl.tudelft.sem.template.hoa.domain.electionchecks	4	100%	32/32	100%	29/29
nl.tudelft.sem.template.hoa.models	1	100%	7/7	100%	3/3
nl.tudelft.sem.template.hoa.utils	1	89%	8/9	88%	7/8

Voting MS:



Project Summary

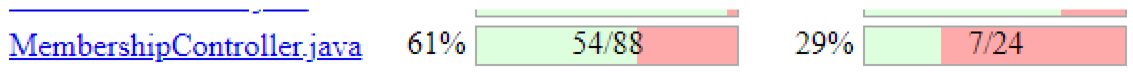
Number of Classes	Line Coverage		Mutation Coverage	
17	86%	284/330	76%	151/198

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
voting.controllers	1	66%	27/41	55%	6/11
voting.db.converters	5	100%	36/36	94%	33/35
voting.domain	3	97%	106/109	84%	48/57
voting.domain.factories	2	85%	17/20	64%	9/14
voting.models	5	92%	44/48	69%	29/42
voting.services	1	71%	54/76	67%	26/39

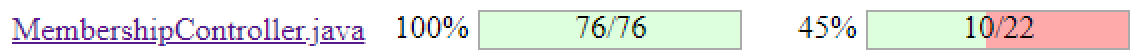
After running the fully automated mutation tests, we can see that there are many classes that have a mutation score below 70%. For this task, we will list the chosen classes as we go:

1. MembershipController/AuthMember MS



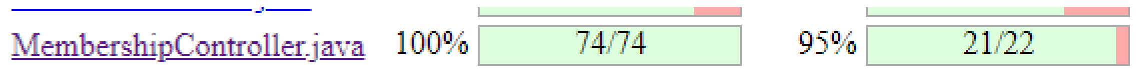
MembershipController.java	61%	54/88	29%	7/24

As you can see, it's initial line coverage was insufficient to begin with, so many mutants survived by the sheer fact that we do not have any tests to kill them. Therefore, our first job was to make sure that the controller had regular full test coverage.
After this was done, the mutant score got just slightly higher:



MembershipController.java	100%	76/76	45%	10/22

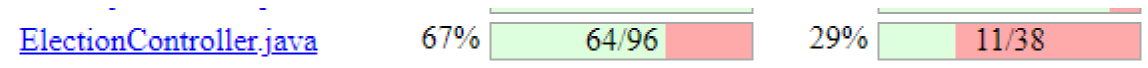
This, however, is still not fully sufficient to consider this test suite immune to mutations. Therefore we had to write a few more unit tests to catch any potential mutants we might have. Most of the surviving mutants for this class ended up being *VOID_CALL* mutants, which just means that we need more negative test cases for validation, or have additional assertions to validate the response during execution. After going with the latter approach and refactoring the tests, we managed to kill all but one of the tested mutants, as seen below.



MembershipController.java	100%	74/74	95%	21/22

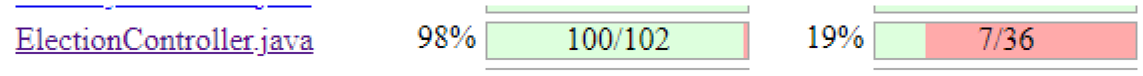
SHA1 of relevant commit: 714c505458c0233c5e2dc770761b8f440e67de32

2. ElectionController/HOA MS



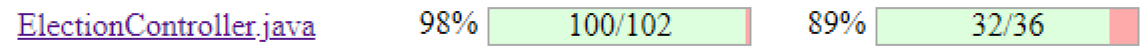
ElectionController.java	67%	64/96	29%	11/38

Similarly to the previous class, the line coverage is quite low to begin with. After dealing with that and running PITtest once more we get the following coverage:



ElectionController.java	98%	100/102	19%	7/36

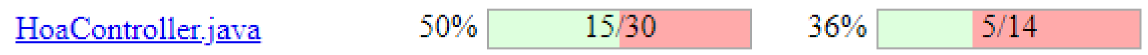
Once again, most mutations that survive the tests are VOID_CALL mutants, but this time we also have a few NEGATE_CONDITIONAL mutants.



ElectionController.java	98%	100/102	89%	32/36

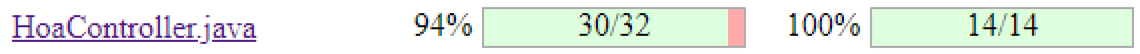
SHA1 of relevant commit: 294dfb0aedb503439e7e916e2b17b04b58920a9c

3. HOAController/HOA MS



HoaController.java	50%	15/30	36%	5/14

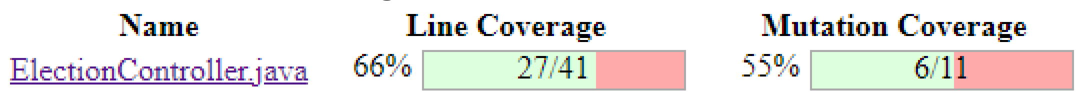
Here the mutation testing proved to be especially useful, because it allowed us to spot a bug in the implementation of one of the methods. After fixing the method, writing tests for it, and augmenting a few of the existing tests to have assertions for the response, we get the following metrics:



HoaController.java	94%	30/32	100%	14/14

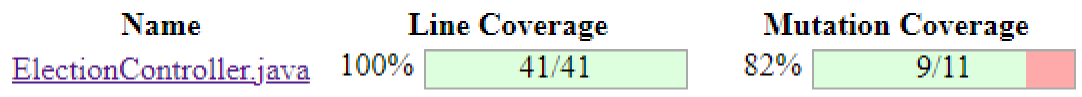
SHA1 of relevant commit: b08d2e3ff3e68f22ffac39fc16bdb12f46e4e2b7

4. ElectionController/Voting MS



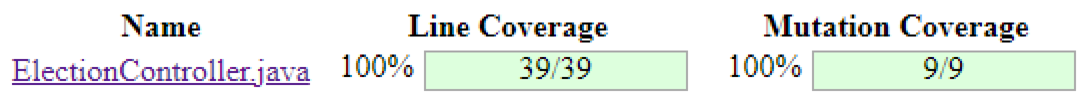
Name	Line Coverage		Mutation Coverage	
ElectionController.java	66%	27/41	55%	6/11

The mutants for this class were much clearer compared to the other classes - here they were mostly of the NULL_RETURN_VALUES variety, so all that it required was to add a few verifiers after asserting the response. Once again we first do full test coverage, with making sure we verify response in new tests, and we get the following metrics:



Name	Line Coverage		Mutation Coverage	
ElectionController.java	100%	41/41	82%	9/11

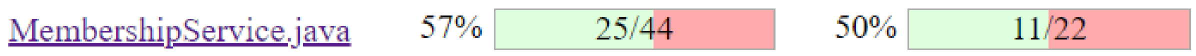
Then we can see that only two mutants remained, both being NULL_RETURN_VALUE mutants. These can be easily resolved by one more assert per method to verify the response for the old tests. After that was done, we got full mutation coverage:



Name	Line Coverage		Mutation Coverage	
ElectionController.java	100%	39/39	100%	9/9

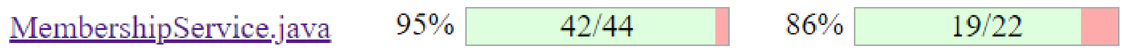
SHA1 of related commit: 850fef22dac2aa0ae6ed5d4951dc2716738bf40d

5. MembershipService/AuthMember MS



MembershipService.java	57%	25/44	50%	11/22

Here the low line coverage again led to low mutation coverage. After dealing with the line coverage there were some NULL_RETURN_VALUE mutants left. We easily resolved these by improving the tests so that they check all the functionalities in the methods.
The three mutants that survived are all *VOID_CALL mutants* which we were unable to kill because this part of the program was mocked and it cannot be changed during the execution of the method.



MembershipService.java	95%	42/44	86%	19/22

SHA1 of relevant commit: a4cadbd344ec8b27cc21b82a56ad3fb39b253c06

Task 2: Manual Mutation Testing

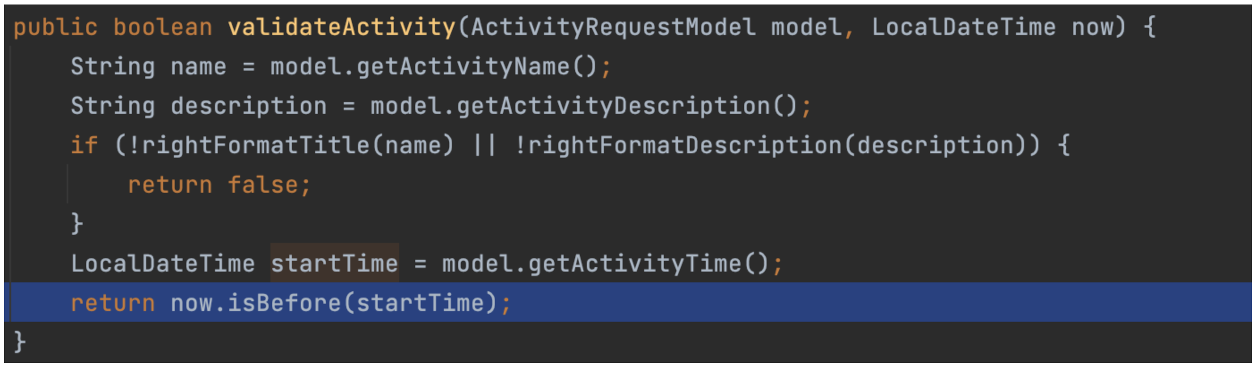
The core domain we have selected is HOA. We thought this was the core one because it has to deal with members, but also with elections and activities. The classes from this core domain that we have selected and that we deem as critical will be listed right after. The structure in which we present how we did manual mutation testing is the following: we list the crucial class in the core domain we selected, we motivate the importance of the class, we pick a method where we will introduce a mutant and we motivate the choice for both the mutant and the method we picked, then we will write a test case that kills it, followed by the SHA1 of the commit where the test that kills that mutant was created.

1. ***<u>ActivityService</u>***

The public board functionality in every homeowner's association depends heavily on this class, as this class contains methods that create activities, that assign members of associations to activities, that allow members of associations to leave an activity. It ensures that anomalies do not happen, for instance, an activity that would begin in the past compared to the moment of creating it, or a non-member of association X joining an activity in that association.

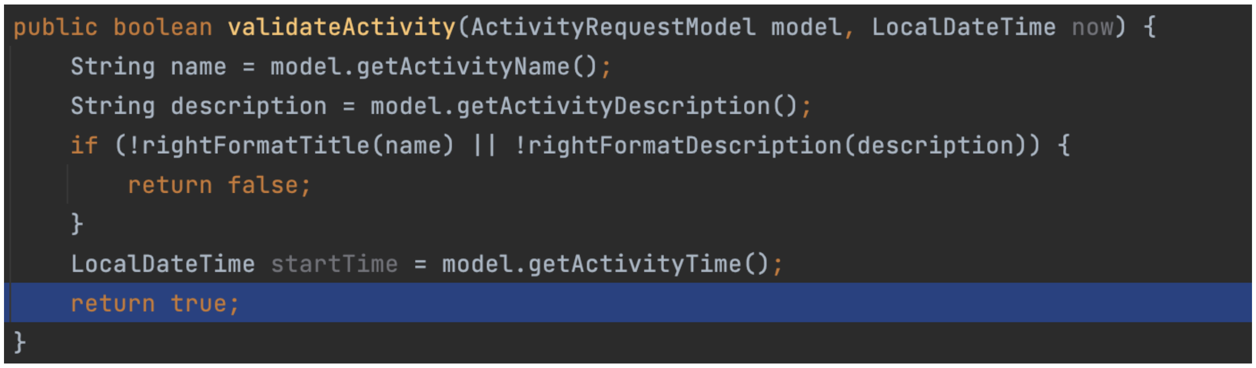
Method, where we will introduce a mutant and the mutant, introduced:

Before bug:



```java
public boolean validateActivity(ActivityRequestModel model, LocalDateTime now) {
    String name = model.getActivityName();
    String description = model.getActivityDescription();
    if (!rightFormatTitle(name) || !rightFormatDescription(description)) {
        return false;
    }
    LocalDateTime startTime = model.getActivityTime();
    return now.isBefore(startTime);
}
```

After bug:



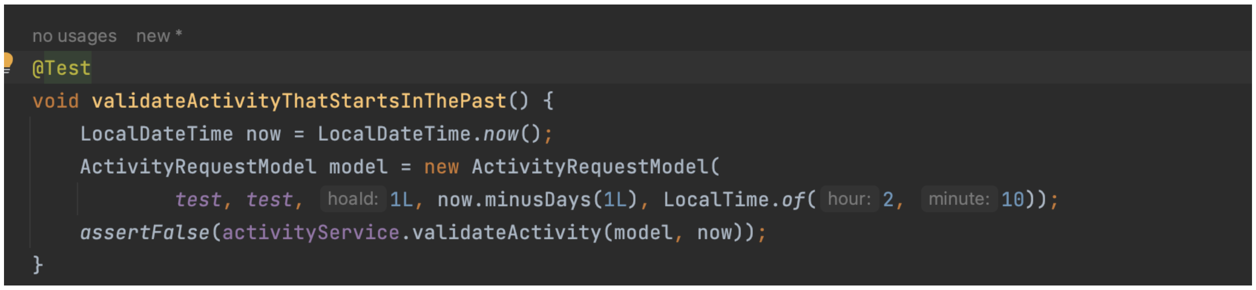
```java
public boolean validateActivity(ActivityRequestModel model, LocalDateTime now) {
    String name = model.getActivityName();
    String description = model.getActivityDescription();
    if (!rightFormatTitle(name) || !rightFormatDescription(description)) {
        return false;
    }
    LocalDateTime startTime = model.getActivityTime();
    return true;
}
```

We chose to manually introduce a mutant in this method because it’s crucial in how activities are created. They first have to respect a format in the name and description they have, and then the time when it begins must be in the future at the time of creation.

The mutant we introduced is of type BOOLEAN_TRUE_RETURN, as we replaced the original return value of the method “now.isBefore(startTime)” with “true”. This means that if the activity satisfies the name and description format, it will be regardless of the start time. This is certainly something we do not want to happen, and this is why we chose to introduce this mutant.

SHA1 of the commit where we introduced the bug (proving that it survives our test suite): *C01387782113a7ce72895a7b5c4e76fa981c52ae*

The test we made to kill this bug:



```
no usages   new *
@Test
void validateActivityThatStartsInThePast() {
    LocalDateTime now = LocalDateTime.now();
    ActivityRequestModel model = new ActivityRequestModel(
            test, test,  hoald: 1L, now.minusDays(1L), LocalTime.of( hour: 2,  minute: 10));
    assertFalse(activityService.validateActivity(model, now));
}
```

The test effectively kills the mutant as the method “validate activity” has to return false every time, as the activity is always created in the past.

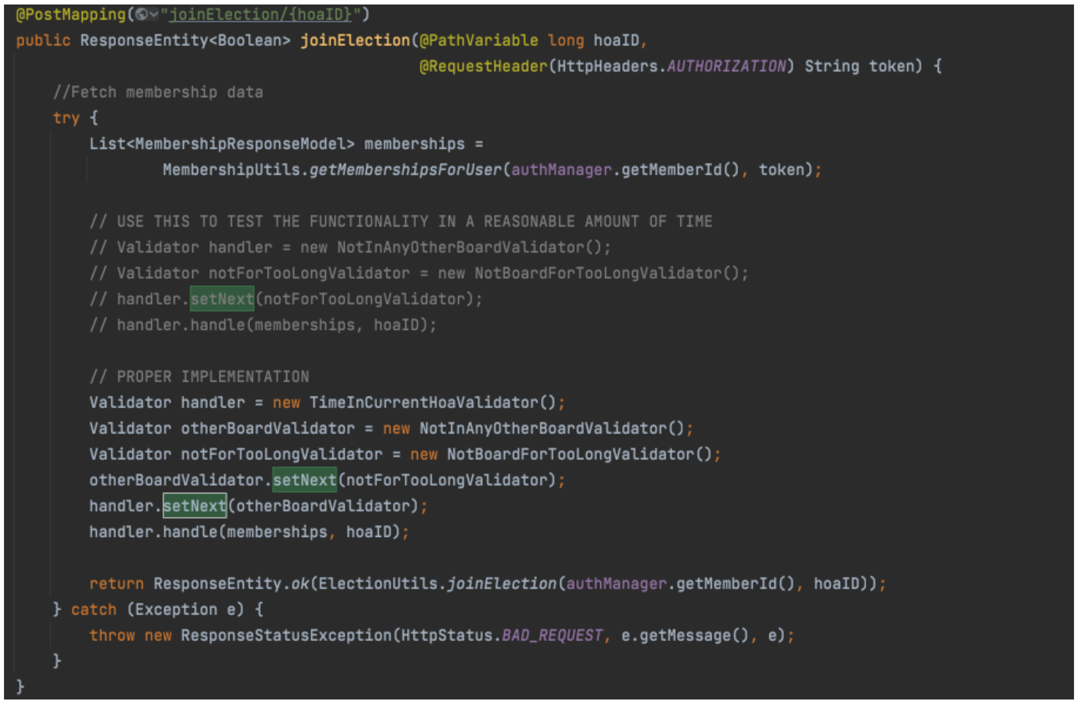
SHA1 of the commit where we introduced the test to kill the bug (proving it actually killed the bug): 8e3f9f31a779f527701d15b5e8987662267daee9

2. ***ElectionController***

The ElectionController is a crucial part of our code as it handles all the logic for managing Elections. This class is responsible for processing the incoming API calls and therefore needs to be thoroughly tested for mutants as users might try to input unexpected values into our application. As the methods in this class are a bit more logically heavy there were plenty of places where to inject mutants.

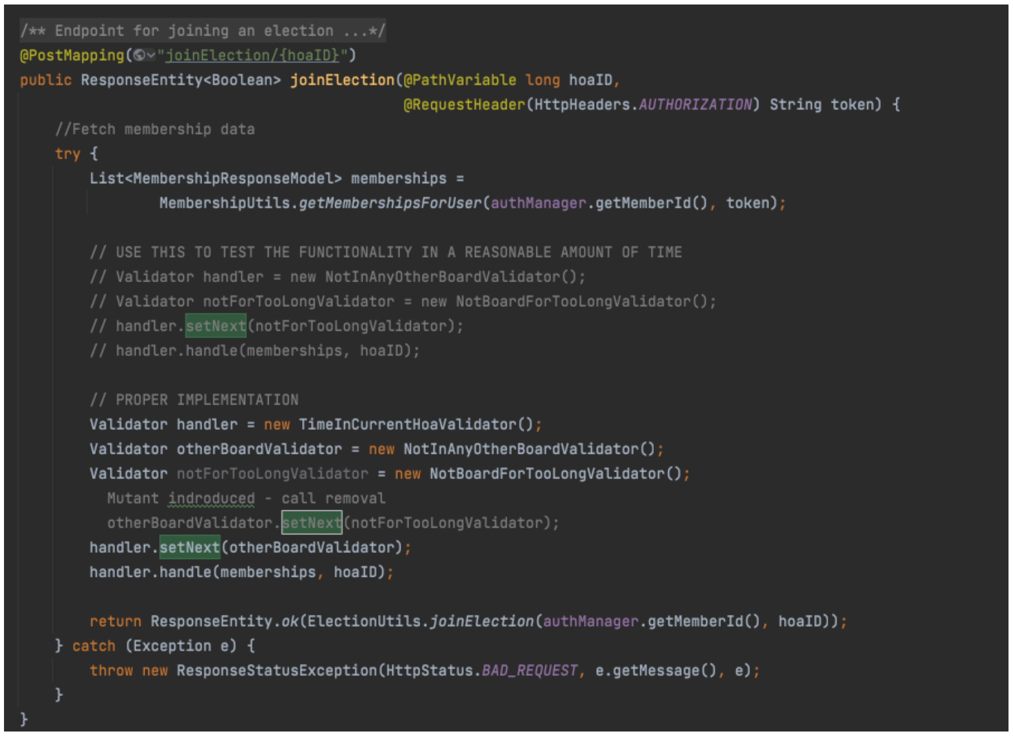
Method, where we will introduce a mutant and the mutant, introduced:

Before bug:



```
@PostMapping("joinElection/{hoaID}")
public ResponseEntity<Boolean> joinElection(@PathVariable long hoaID,
                                            @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    //Fetch membership data
    try {
        List<MembershipResponseModel> memberships =
                MembershipUtils.getMembershipsForUser(authManager.getMemberId(), token);

        // USE THIS TO TEST THE FUNCTIONALITY IN A REASONABLE AMOUNT OF TIME
        // Validator handler = new NotInAnyOtherBoardValidator();
        // Validator notForTooLongValidator = new NotBoardForTooLongValidator();
        // handler.setNext(notForTooLongValidator);
        // handler.handle(memberships, hoaID);

        // PROPER IMPLEMENTATION
        Validator handler = new TimeInCurrentHoaValidator();
        Validator otherBoardValidator = new NotInAnyOtherBoardValidator();
        Validator notForTooLongValidator = new NotBoardForTooLongValidator();
        otherBoardValidator.setNext(notForTooLongValidator);
        handler.setNext(otherBoardValidator);
        handler.handle(memberships, hoaID);

        return ResponseEntity.ok(ElectionUtils.joinElection(authManager.getMemberId(), hoaID));
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage(), e);
    }
}
```

After bug:

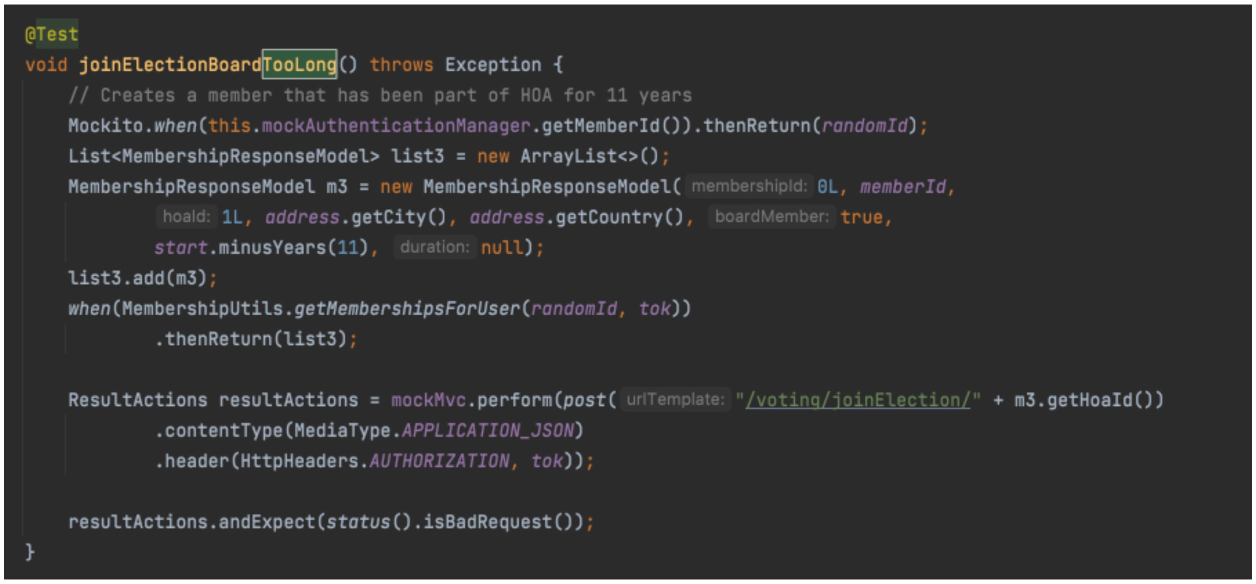


```
/** Endpoint for joining an election ...*/
@PostMapping("joinElection/{hoaID}")
public ResponseEntity<Boolean> joinElection(@PathVariable long hoaID,
                                            @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    //Fetch membership data
    try {
        List<MembershipResponseModel> memberships =
                MembershipUtils.getMembershipsForUser(authManager.getMemberId(), token);

        // USE THIS TO TEST THE FUNCTIONALITY IN A REASONABLE AMOUNT OF TIME
        // Validator handler = new NotInAnyOtherBoardValidator();
        // Validator notForTooLongValidator = new NotBoardForTooLongValidator();
        // handler.setNext(notForTooLongValidator);
        // handler.handle(memberships, hoaID);

        // PROPER IMPLEMENTATION
        Validator handler = new TimeInCurrentHoaValidator();
        Validator otherBoardValidator = new NotInAnyOtherBoardValidator();
        Validator notForTooLongValidator = new NotBoardForTooLongValidator();
          Mutant indroduced - call removal
          otherBoardValidator.setNext(notForTooLongValidator);
        handler.setNext(otherBoardValidator);
        handler.handle(memberships, hoaID);

        return ResponseEntity.ok(ElectionUtils.joinElection(authManager.getMemberId(), hoaID));
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage(), e);
    }
}
```

The joinElection method was chosen as it is crucial in the whole Election logic, while it is the method that allows members of an HOA to join elections and validates them. A mutant of type REMOVED_CALL was introduced as our test suite was not verifying for this.

SHA1 of the commit where we introduced the bug (proving that it survives our test suite): 87541f915fbbeccdfaf71f04ce0cb8e9db2310f1

The test we made to kill this bug:



```
@Test
void joinElectionBoardTooLong() throws Exception {
    // Creates a member that has been part of HOA for 11 years
    Mockito.when(this.mockAuthenticationManager.getMemberId()).thenReturn(randomId);
    List<MembershipResponseModel> list3 = new ArrayList<>();
    MembershipResponseModel m3 = new MembershipResponseModel( membershipId: 0L, memberId,
            hoaId: 1L, address.getCity(), address.getCountry(), boardMember: true,
            start.minusYears(11), duration: null);
    list3.add(m3);
    when(MembershipUtils.getMembershipsForUser(randomId, tok))
            .thenReturn(list3);

    ResultActions resultActions = mockMvc.perform(post( urlTemplate: "/voting/joinElection/" + m3.getHoaId())
            .contentType(MediaType.APPLICATION_JSON)
            .header(HttpHeaders.AUTHORIZATION, tok));

    resultActions.andExpect(status().isBadRequest());
}
```

The test kills the mutant as the method “joinElection” gives a BAD_REQUEST response, as the member is always part of the HOA for more than the maximum number of years.

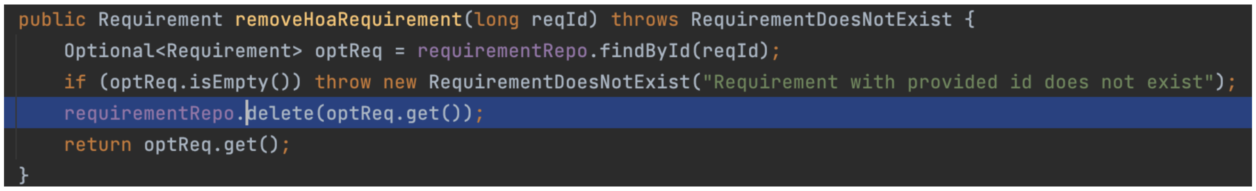
SHA1 of the commit where we introduced the test to kill the bug (proving it actually killed the bug): 931d2015818247fcfbac63b16950410cde1ce2d3

3. ***RequirementService***

This class queries a repository for requirements and adds and deletes requirements to specific associations. This ensures that each association can have its own set of requirements, and it handles the logic of storing, retrieving, and removing them, therefore being an essential class in our application.

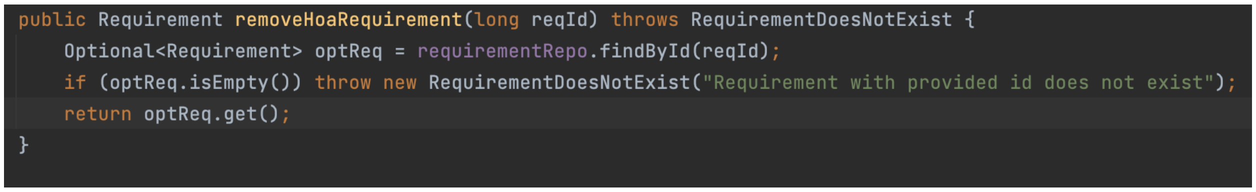
Method, where we will introduce a mutant and the mutant, introduced:

Before bug:



```java
public Requirement removeHoaRequirement(long reqId) throws RequirementDoesNotExist {
    Optional<Requirement> optReq = requirementRepo.findById(reqId);
    if (optReq.isEmpty()) throw new RequirementDoesNotExist("Requirement with provided id does not exist");
    requirementRepo.delete(optReq.get());
    return optReq.get();
}
```

After bug:



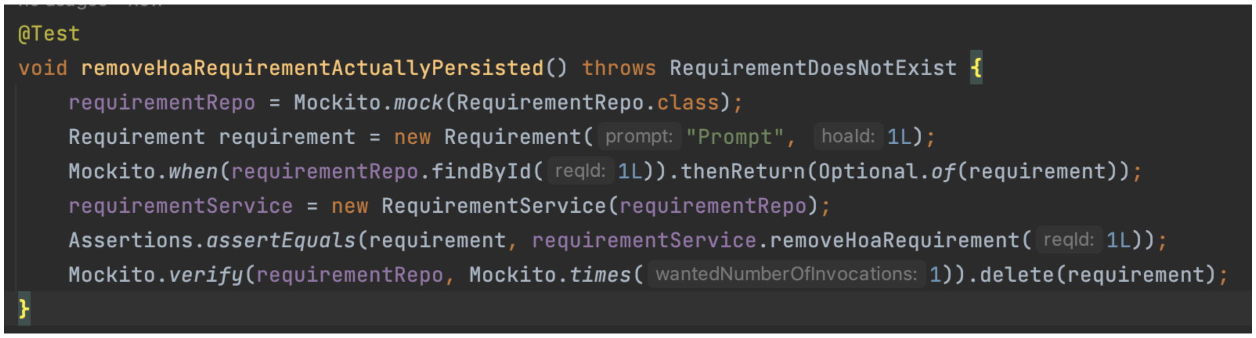
```java
public Requirement removeHoaRequirement(long reqId) throws RequirementDoesNotExist {
    Optional<Requirement> optReq = requirementRepo.findById(reqId);
    if (optReq.isEmpty()) throw new RequirementDoesNotExist("Requirement with provided id does not exist");
    return optReq.get();
}
```

We chose to introduce a mutant in this method because it is supposed to delete a valid entry in the repository, thus being important in the bookkeeping of requirements in a certain association.

The mutant we introduced is of type VOID_METHOD_CALL_MUTATOR, as we deleted the void call “requirementRepo.delete(optReq.get())”. This means that whenever we want to delete an existing requirement, the deletion won’t actually be persisted. This is something that sometimes happens in different applications that we use day to day, where some deletions are not ever persisted, and this is not desirable, thus came our choice for this mutant.

SHA1 of the commit where we introduced the bug (proving that it survives our test suite): 1afdddadaf7edcf4bc257fdbcc4342c66197836d

The test we made to kill this bug:



```java
@Test
void removeHoaRequirementActuallyPersisted() throws RequirementDoesNotExist {
    requirementRepo = Mockito.mock(RequirementRepo.class);
    Requirement requirement = new Requirement( prompt: "Prompt",  hoaId: 1L);
    Mockito.when(requirementRepo.findById( reqId: 1L)).thenReturn(Optional.of(requirement));
    requirementService = new RequirementService(requirementRepo);
    Assertions.assertEquals(requirement, requirementService.removeHoaRequirement( reqId: 1L));
    Mockito.verify(requirementRepo, Mockito.times( wantedNumberOfInvocations: 1)).delete(requirement);
}
```

This test kills the mutant, as the last line ensures that the delete call has been made to the repository exactly once.

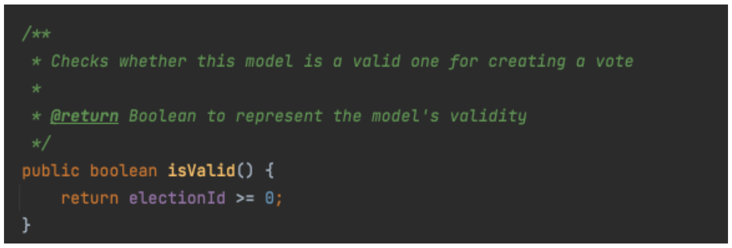
SHA1 of the commit where we introduced the test to kill the bug (proving it actually killed the bug): cb3caa6bb1370ecddc54cc2bf07c75359d6d9fb9

4. ***VotingModel***

This class is extremely important as it is responsible for holding information about the Voting process. It carries 2 IDs (electionID and memberId) and a String representation of the choice made in the election.

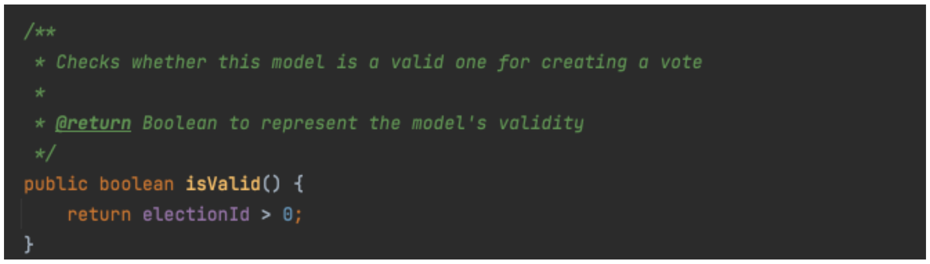
Method, where we will introduce a mutant and the mutant, introduced:

Before bug:



```
/**
 * Checks whether this model is a valid one for creating a vote
 *
 * @return Boolean to represent the model's validity
 */
public boolean isValid() {
    return electionId >= 0;
}
```

Before bug:



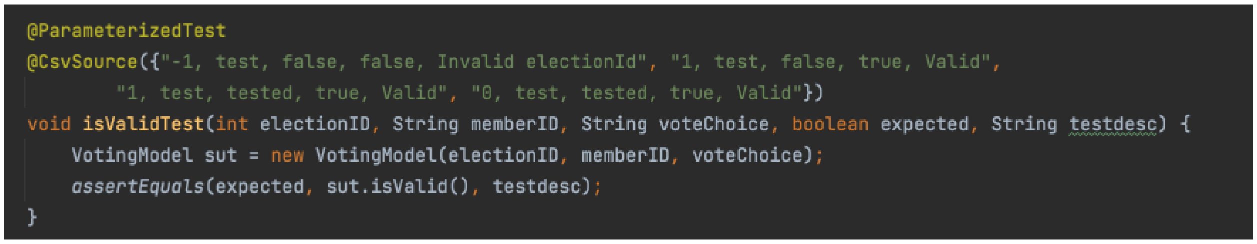
```
/**
 * Checks whether this model is a valid one for creating a vote
 *
 * @return Boolean to represent the model's validity
 */
public boolean isValid() {
    return electionId > 0;
}
```

We chose to manually introduce a mutant in this method because it's crucial in how IDs are assigned to the Elections. Each ID can have any non-negative integer value. Although this method is quite simple, we had no other classes where mutants could be introduced in HOA microservice, so we decided to test it at last.

The mutant we introduced is of type CHANGED_CONDITIONAL_BOUNDRY, as we replaced the original check >= with >. This means that if a model with ID 0 is created it will not be considered valid.

SHA1 of the commit where we introduced the bug (proving that it survives our test suite): 47c363fb5f0c8eeb96fbc5c244d0104badd204f6

The test we made to kill this bug:



```
@ParameterizedTest
@CsvSource({"-1, test, false, false, Invalid electionId", "1, test, false, true, Valid",
        "1, test, tested, true, Valid", "0, test, tested, true, Valid"})
void isValidTest(int electionID, String memberID, String voteChoice, boolean expected, String testdesc) {
    VotingModel sut = new VotingModel(electionID, memberID, voteChoice);
    assertEquals(expected, sut.isValid(), testdesc);
}
```

The test kills the mutant as it now contains a case in which ID with value 0 is present.

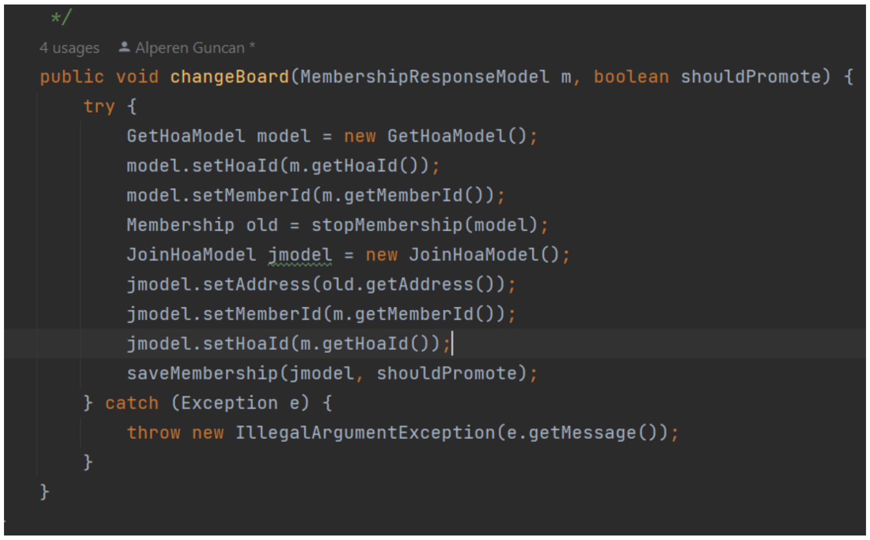
SHA1 of the commit where we introduced the test to kill the bug (proving it actually killed the bug): d610888dfc7b0c943021f93933ce719efd5e5f32

5. ***MembershipService***

MembershipService is responsible for fetching and managing membership objects, which relate users to their HOAs, which makes it a critical part of the repository. It is thus thoroughly tested, however, it was still possible to introduce a mutant:

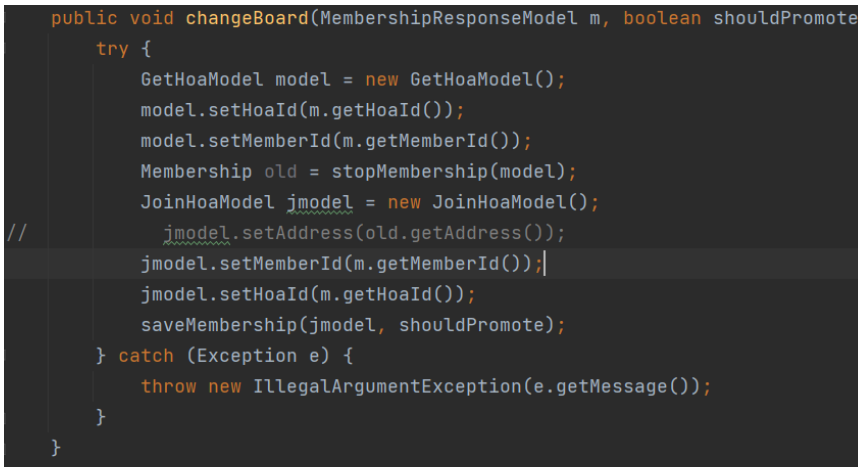
Method, where we will introduce a mutant and the mutant, introduced:

Before bug:



```
*/
4 usages   Alperen Guncan *
public void changeBoard(MembershipResponseModel m, boolean shouldPromote) {
    try {
        GetHoaModel model = new GetHoaModel();
        model.setHoaId(m.getHoaId());
        model.setMemberId(m.getMemberId());
        Membership old = stopMembership(model);
        JoinHoaModel jmodel = new JoinHoaModel();
        jmodel.setAddress(old.getAddress());
        jmodel.setMemberId(m.getMemberId());
        jmodel.setHoaId(m.getHoaId());
        saveMembership(jmodel, shouldPromote);
    } catch (Exception e) {
        throw new IllegalArgumentException(e.getMessage());
    }
}
```

After bug:



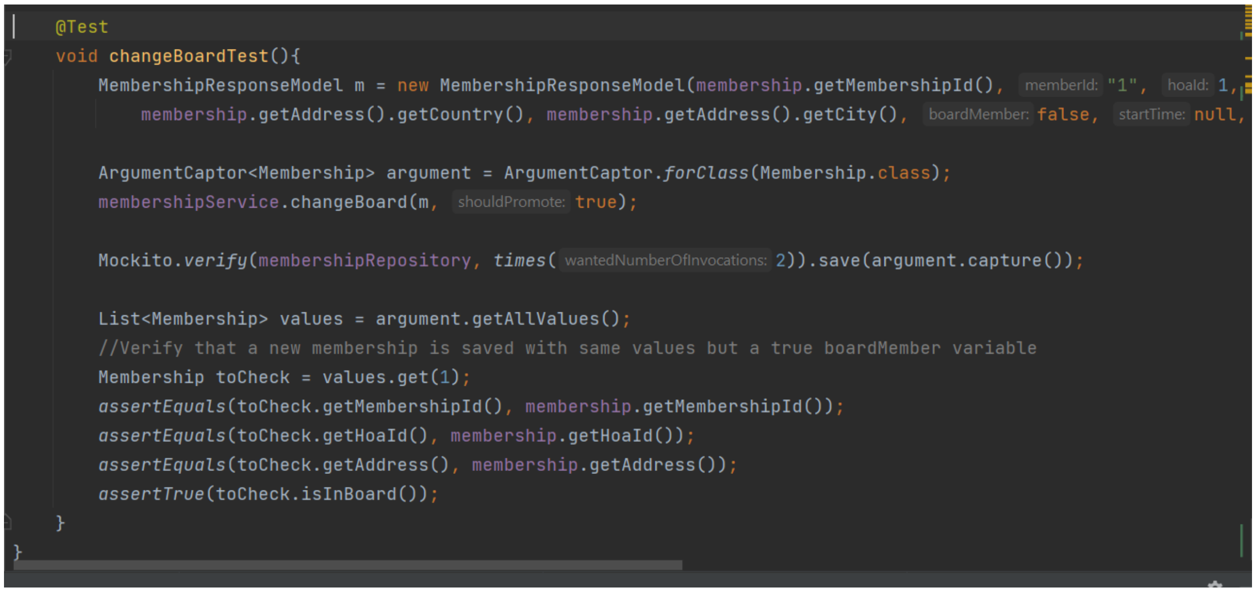
```
public void changeBoard(MembershipResponseModel m, boolean shouldPromote
    try {
        GetHoaModel model = new GetHoaModel();
        model.setHoaId(m.getHoaId());
        model.setMemberId(m.getMemberId());
        Membership old = stopMembership(model);
        JoinHoaModel jmodel = new JoinHoaModel();
//          jmodel.setAddress(old.getAddress());
        jmodel.setMemberId(m.getMemberId());
        jmodel.setHoaId(m.getHoaId());
        saveMembership(jmodel, shouldPromote);
    } catch (Exception e) {
        throw new IllegalArgumentException(e.getMessage());
    }
}
```

Changing a member to board is an important part of our infrastructure and no data should be lost in the process.

The mutant we introduced is of type VOID_METHOD_CALLS.

SHA1 of the commit where we introduced the bug (proving that it survives our test suite): 26268c8dcf4c45d3508ec5ed5077f41fb7506cff

The test we made to kill this bug:



```
    @Test
    void changeBoardTest(){
        MembershipResponseModel m = new MembershipResponseModel(membership.getMembershipId(), memberId: "1", hoaId: 1,
            membership.getAddress().getCountry(), membership.getAddress().getCity(), boardMember: false, startTime: null,

        ArgumentCaptor<Membership> argument = ArgumentCaptor.forClass(Membership.class);
        membershipService.changeBoard(m, shouldPromote: true);

        Mockito.verify(membershipRepository, times( wantedNumberOfInvocations: 2)).save(argument.capture());

        List<Membership> values = argument.getAllValues();
        //Verify that a new membership is saved with same values but a true boardMember variable
        Membership toCheck = values.get(1);
        assertEquals(toCheck.getMembershipId(), membership.getMembershipId());
        assertEquals(toCheck.getHoaId(), membership.getHoaId());
        assertEquals(toCheck.getAddress(), membership.getAddress());
        assertTrue(toCheck.isInBoard());
    }
}
```

The test kills the mutant by checking whether the saved membership is the same as the previous one, but with a new inBoard variable.

SHA1 of the commit where we introduced the test to kill the bug: 26268c8dcf4c45d3508ec5ed5077f41fb7506cff