Group-DB-09b-Assignment-3

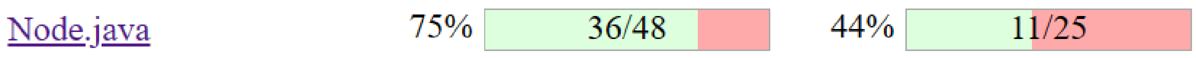
Task 1

Class: clusters-microservice/Node

https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM09b/-/merge_requests/66/diffs?commit_id=027f7e1ff5ff5b672112c170aacfaadb032c6fd8

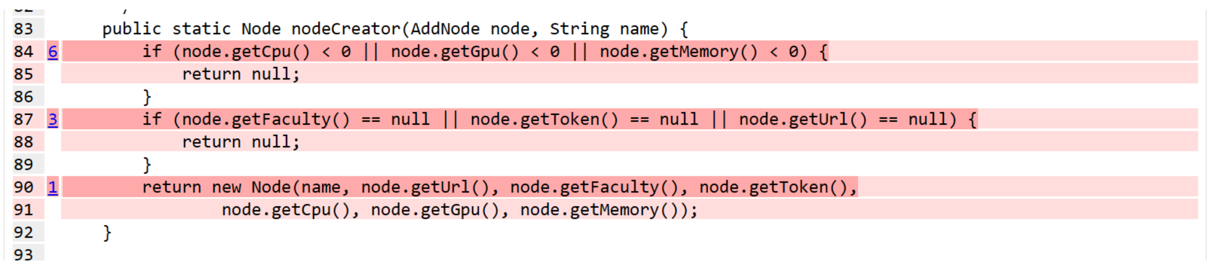
Introduction

Before the refactoring, the mutation score of the Node class was 44%.

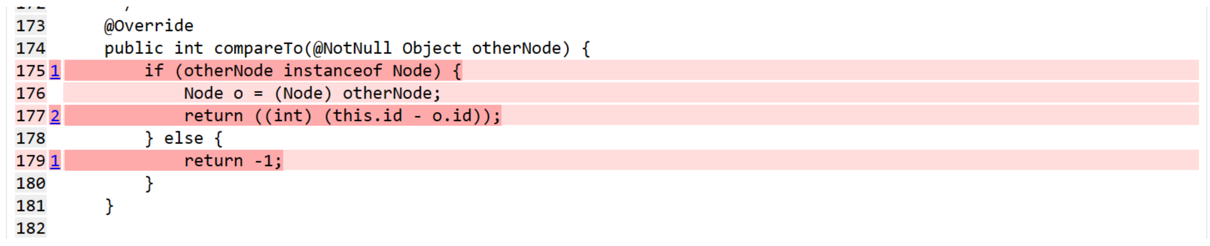


Node.java	75%	36/48	44%	11/25

Here are the mutants in the code:



```
    public static Node nodeCreator(AddNode node, String name) {
        if (node.getCpu() < 0 || node.getGpu() < 0 || node.getMemory() < 0) {
            return null;
        }
        if (node.getFaculty() == null || node.getToken() == null || node.getUrl() == null) {
            return null;
        }
        return new Node(name, node.getUrl(), node.getFaculty(), node.getToken(),
                node.getCpu(), node.getGpu(), node.getMemory());
    }
```



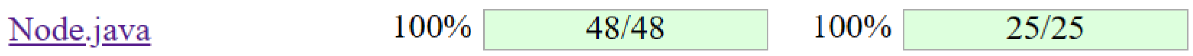
```
    @Override
    public int compareTo(@NotNull Object otherNode) {
        if (otherNode instanceof Node) {
            Node o = (Node) otherNode;
            return ((int) (this.id - o.id));
        } else {
            return -1;
        }
    }
```

Refactoring

In the tests we made sure that we checked the boundaries and the corner cases, where the mutants lived.

Conclusion

After the refactoring, the mutation score of the Node class is 100%.



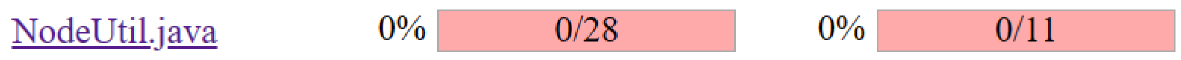
Node.java	100%	48/48	100%	25/25

Class: clusters-microservice/NodeUtil

https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM09b/-/merge_requests/67/diffs?commit_id=7da73540de61e9fbabb525f0826d728657396685

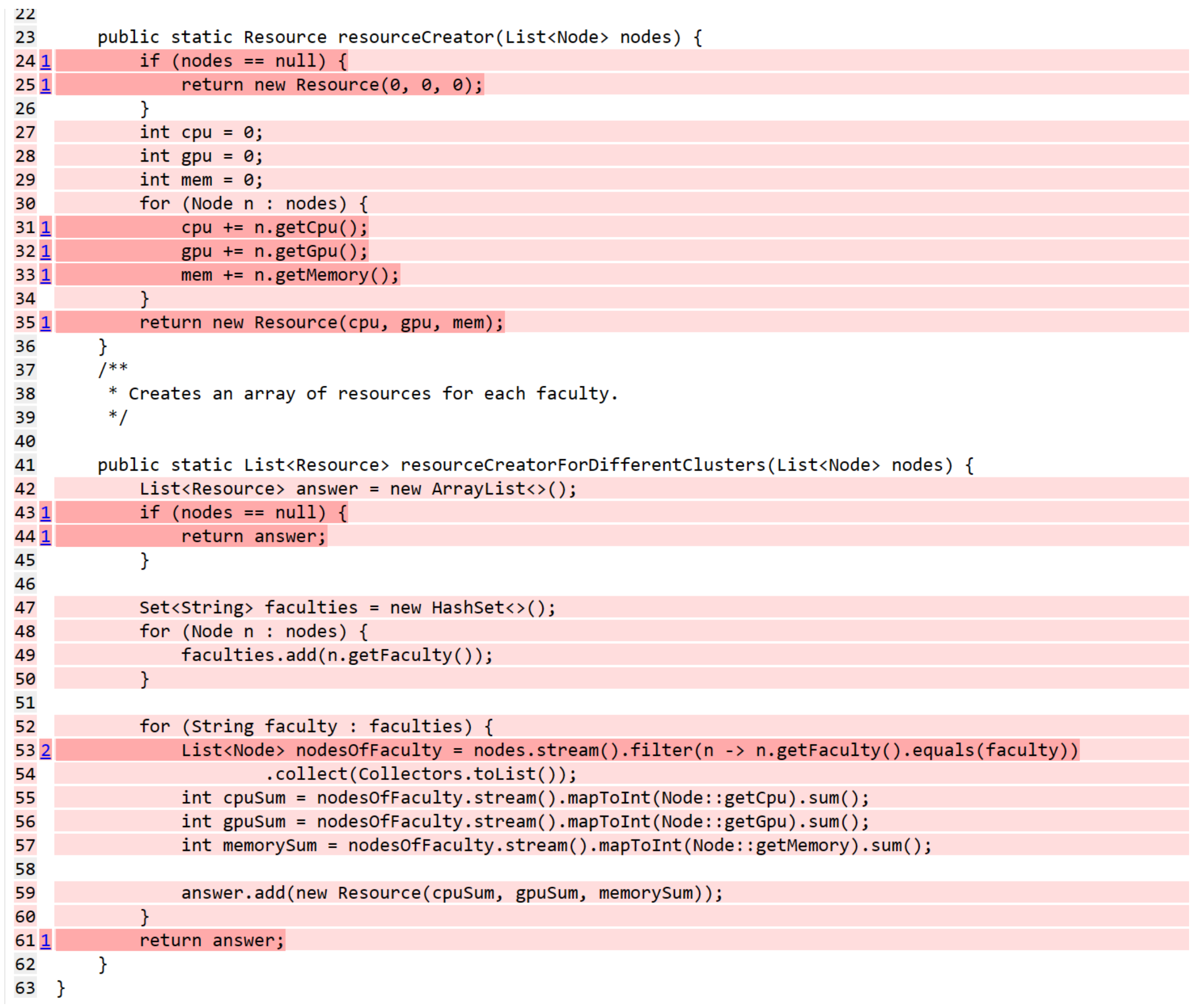
Introduction

Before refactoring, the mutation score of the class was 0%.



NodeUtil.java	0%	0/28	0%	0/11

Here are the mutants in the code:



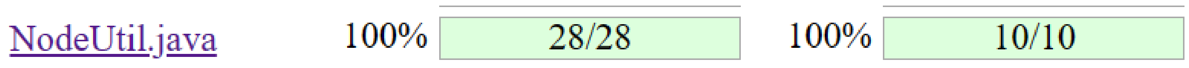
```
    public static Resource resourceCreator(List<Node> nodes) {
        if (nodes == null) {
            return new Resource(0, 0, 0);
        }
        int cpu = 0;
        int gpu = 0;
        int mem = 0;
        for (Node n : nodes) {
            cpu += n.getCpu();
            gpu += n.getGpu();
            mem += n.getMemory();
        }
        return new Resource(cpu, gpu, mem);
    }
    /**
     * Creates an array of resources for each faculty.
     */

    public static List<Resource> resourceCreatorForDifferentClusters(List<Node> nodes) {
        List<Resource> answer = new ArrayList<>();
        if (nodes == null) {
            return answer;
        }

        Set<String> faculties = new HashSet<>();
        for (Node n : nodes) {
            faculties.add(n.getFaculty());
        }

        for (String faculty : faculties) {
            List<Node> nodesOfFaculty = nodes.stream().filter(n -> n.getFaculty().equals(faculty))
                    .collect(Collectors.toList());
            int cpuSum = nodesOfFaculty.stream().mapToInt(Node::getCpu).sum();
            int gpuSum = nodesOfFaculty.stream().mapToInt(Node::getGpu).sum();
            int memorySum = nodesOfFaculty.stream().mapToInt(Node::getMemory).sum();

            answer.add(new Resource(cpuSum, gpuSum, memorySum));
        }
        return answer;
    }
}
```

Refactoring

In the tests we made sure that we checked the boundaries and the corner cases, where the mutants lived. Moreover, we had to change the production code on one occasion, so that we could modify the tests and kill the mutants.

Conclusion

After the refactoring, the mutation score of the Node class is 100%.



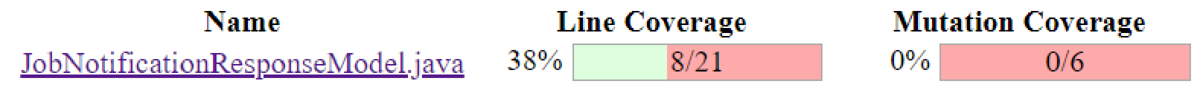
NodeUtil.java	100%	28/28	100%	10/10

Class: JobNotificationResponseModel

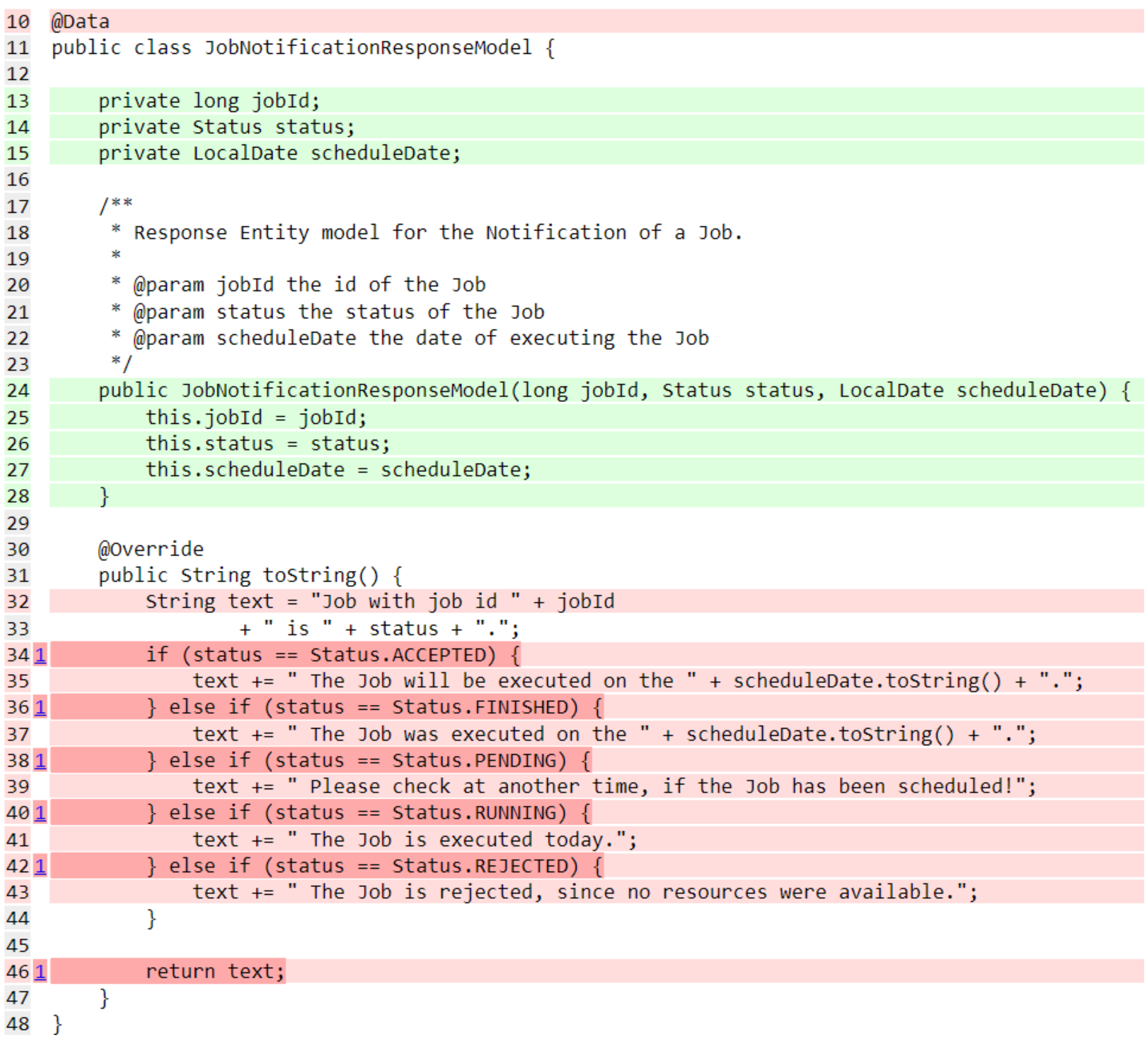
Commit:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM09b/-/merge_requests/71/diffs?commit_id=81764bc243d1ba1a811d6baf84b1d6bc3ed57ac2

Introduction

Initially, the JobNotificationResponseModel class had a line coverage of 38% and a mutation coverage of 0%.



Name	Line Coverage		Mutation Coverage	
JobNotificationResponseModel.java	38%	8/21	0%	0/6



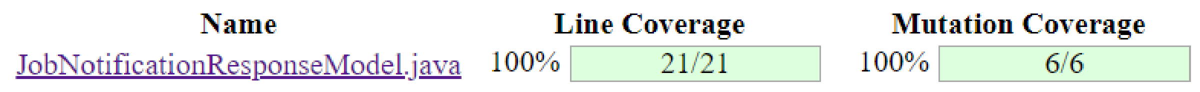
```
@Data
public class JobNotificationResponseModel {

    private long jobId;
    private Status status;
    private LocalDate scheduleDate;

    /**
     * Response Entity model for the Notification of a Job.
     *
     * @param jobId the id of the Job
     * @param status the status of the Job
     * @param scheduleDate the date of executing the Job
     */
    public JobNotificationResponseModel(long jobId, Status status, LocalDate scheduleDate) {
        this.jobId = jobId;
        this.status = status;
        this.scheduleDate = scheduleDate;
    }

    @Override
    public String toString() {
        String text = "Job with job id " + jobId
                + " is " + status + ".";
        if (status == Status.ACCEPTED) {
            text += " The Job will be executed on the " + scheduleDate.toString() + ".";
        } else if (status == Status.FINISHED) {
            text += " The Job was executed on the " + scheduleDate.toString() + ".";
        } else if (status == Status.PENDING) {
            text += " Please check at another time, if the Job has been scheduled!";
        } else if (status == Status.RUNNING) {
            text += " The Job is executed today.";
        } else if (status == Status.REJECTED) {
            text += " The Job is rejected, since no resources were available.";
        }

        return text;
    }
}
```

Refactoring

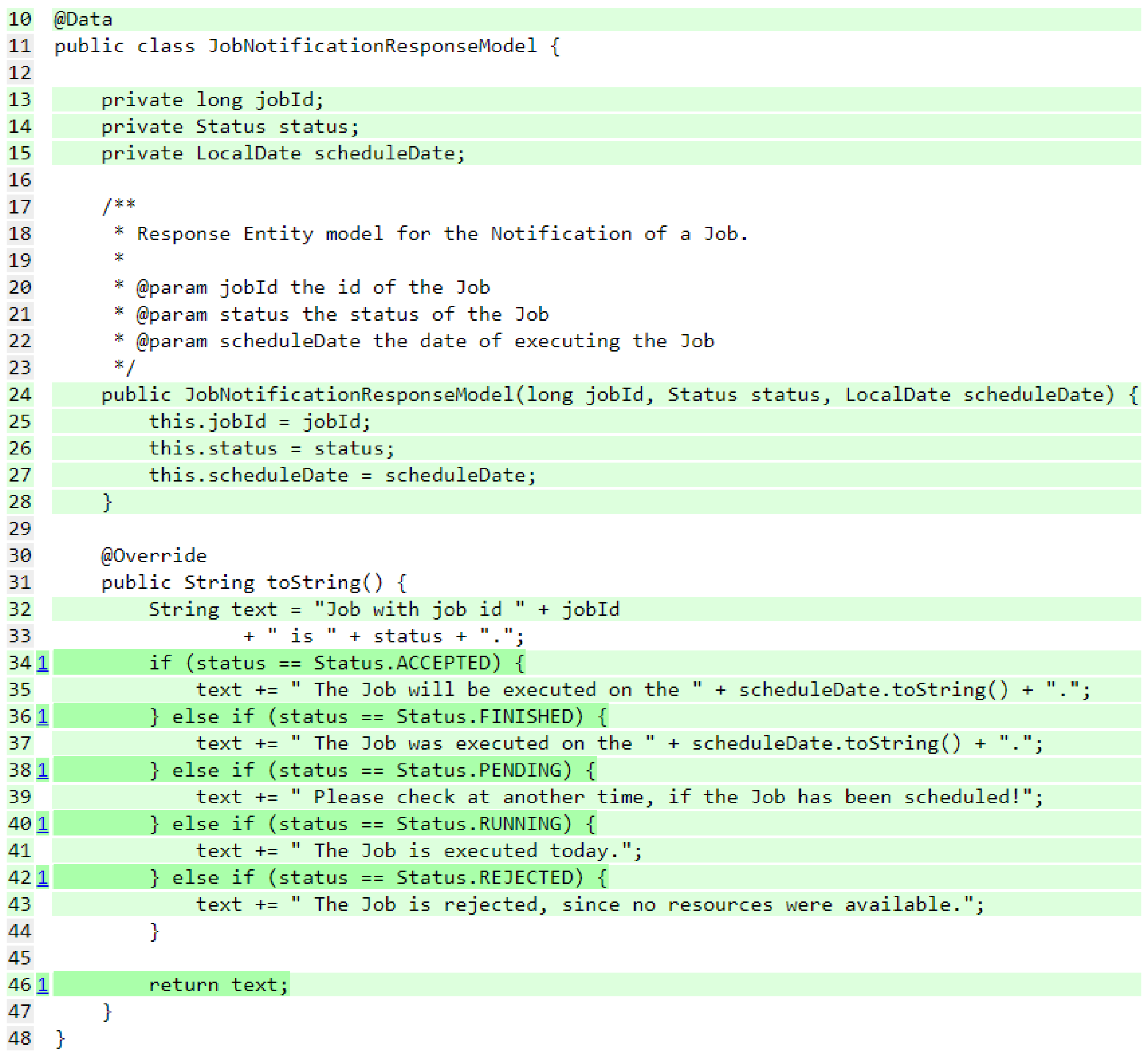
The low mutation coverage is caused by the toString() method, which can return five different types of texts, depending on the status of the JobNotificationResponseModel. Thus, five test methods have been added to increase the mutation coverage from 0% to 100%. Additionally, three test methods for the setters have been added, so that the line coverage also reaches 100%.

Conclusion

After adding eight new test methods, the line and mutation coverage increased to 100%.



Name	Line Coverage		Mutation Coverage	
JobNotificationResponseModel.java	100%	21/21	100%	6/6



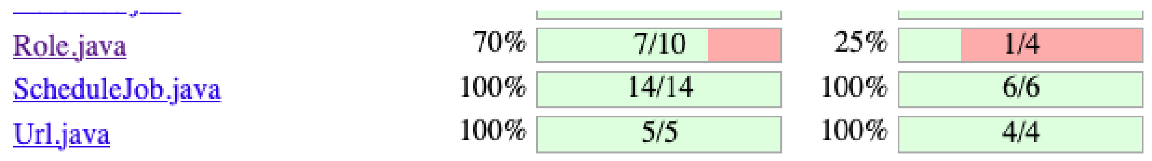
```
@Data
public class JobNotificationResponseModel {

    private long jobId;
    private Status status;
    private LocalDate scheduleDate;

    /**
     * Response Entity model for the Notification of a Job.
     *
     * @param jobId the id of the Job
     * @param status the status of the Job
     * @param scheduleDate the date of executing the Job
     */
    public JobNotificationResponseModel(long jobId, Status status, LocalDate scheduleDate) {
        this.jobId = jobId;
        this.status = status;
        this.scheduleDate = scheduleDate;
    }

    @Override
    public String toString() {
        String text = "Job with job id " + jobId
                + " is " + status + ".";
        if (status == Status.ACCEPTED) {
            text += " The Job will be executed on the " + scheduleDate.toString() + ".";
        } else if (status == Status.FINISHED) {
            text += " The Job was executed on the " + scheduleDate.toString() + ".";
        } else if (status == Status.PENDING) {
            text += " Please check at another time, if the Job has been scheduled!";
        } else if (status == Status.RUNNING) {
            text += " The Job is executed today.";
        } else if (status == Status.REJECTED) {
            text += " The Job is rejected, since no resources were available.";
        }

        return text;
    }
}
```

Class: Role

Commit:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM09b/-/commit/29fc436bf20f231f70104543b717135fc2755169

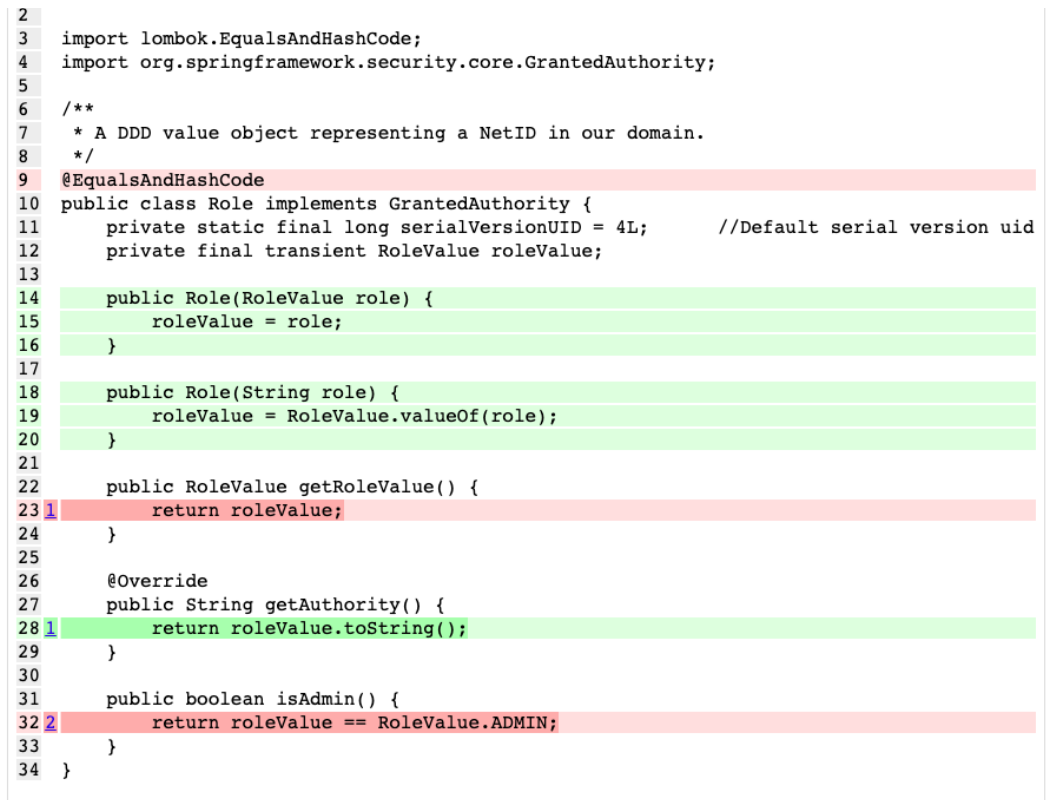
Introduction

After running Pitest in the commons package we can see that the Role class has a mutation score of 25%. This can be improved further.



Role.java	70%	7/10	25%	1/4
ScheduleJob.java	100%	14/14	100%	6/6
Url.java	100%	5/5	100%	4/4

This is the code in the Role.java file.



```java

import lombok.EqualsAndHashCode;
import org.springframework.security.core.GrantedAuthority;

/**
 * A DDD value object representing a NetID in our domain.
 */
@EqualsAndHashCode
public class Role implements GrantedAuthority {
    private static final long serialVersionUID = 4L;      //Default serial version uid
    private final transient RoleValue roleValue;

    public Role(RoleValue role) {
        roleValue = role;
    }

    public Role(String role) {
        roleValue = RoleValue.valueOf(role);
    }

    public RoleValue getRoleValue() {
        return roleValue;
    }

    @Override
    public String getAuthority() {
        return roleValue.toString();
    }

    public boolean isAdmin() {
        return roleValue == RoleValue.ADMIN;
    }
}
```

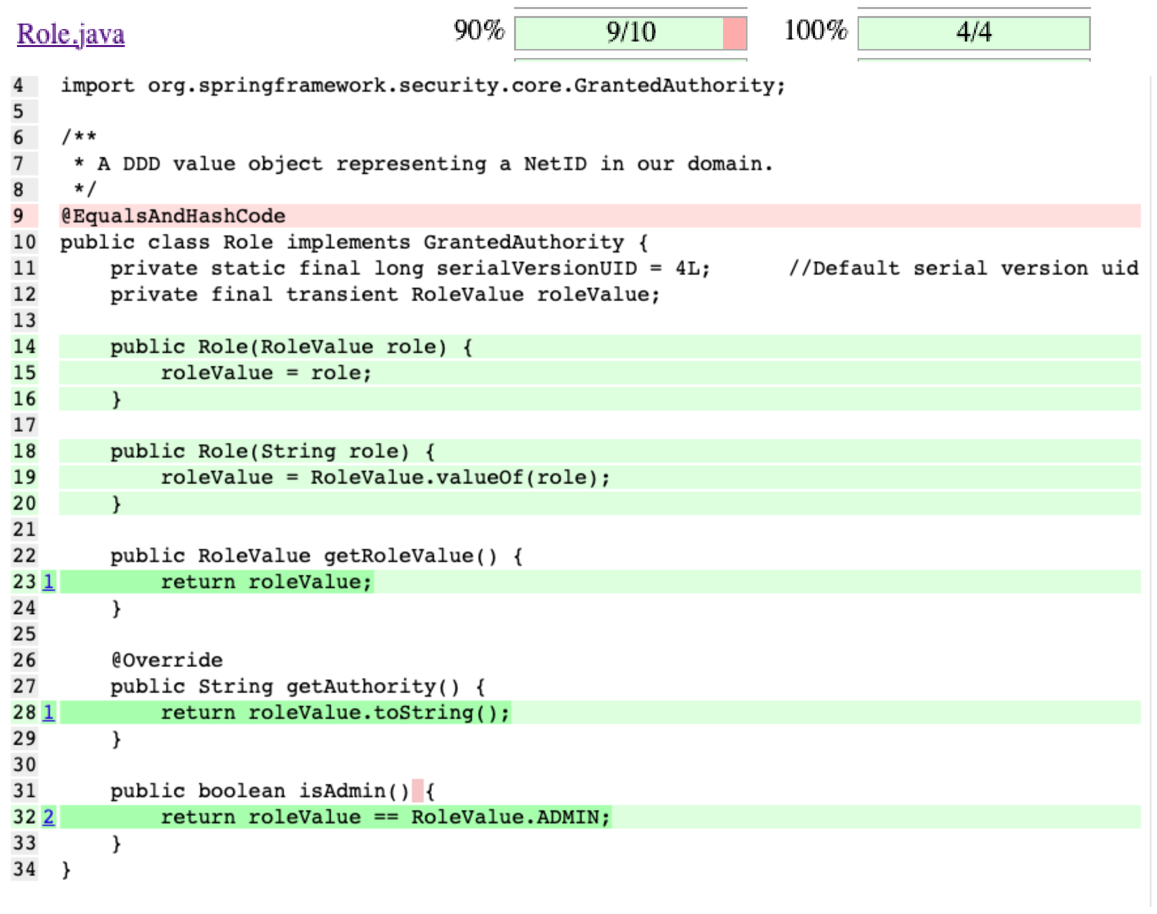
Refactoring

The first mutant we kill is in the function getRoleValue, where we create a check that makes sure the getter does not return null.

In order to achieve 100% mutation coverage in the Role class we have to make additional tests for the isAdmin function. As the function contains an equality condition, we have to perform 2 assertions, one when the Role is an Admin and when it is not.

Conclusion

After performing all the changes proposed above we achieve full mutation coverage as you can see below:



```
Role.java                90%   9/10      100%   4/4
```

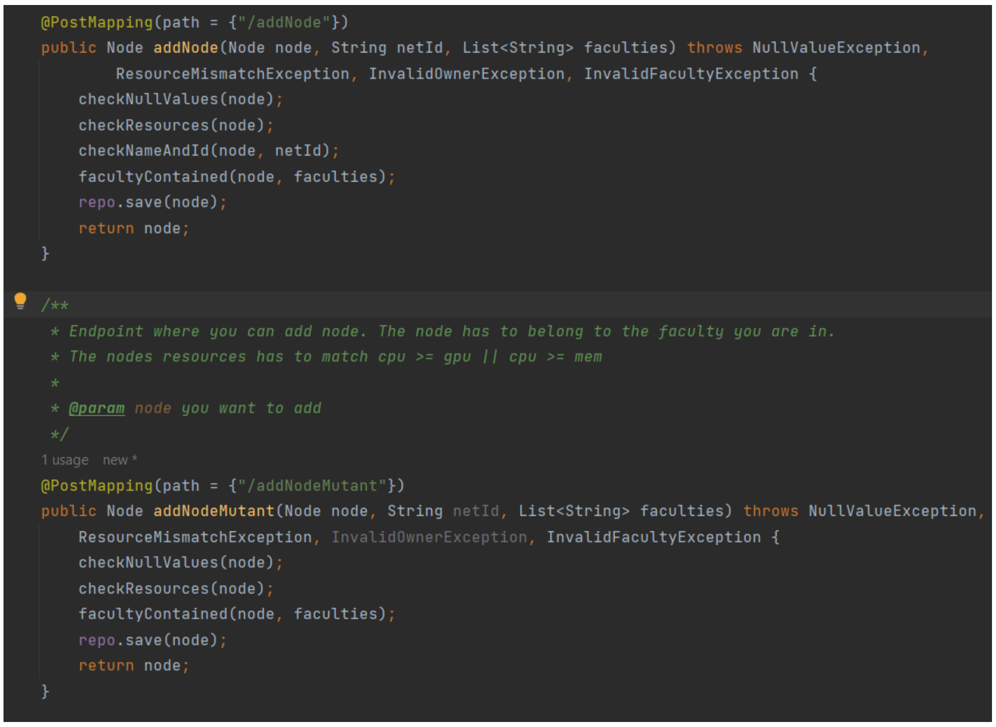
```java
import org.springframework.security.core.GrantedAuthority;

/**
 * A DDD value object representing a NetID in our domain.
 */
@EqualsAndHashCode
public class Role implements GrantedAuthority {
    private static final long serialVersionUID = 4L;      //Default serial version uid
    private final transient RoleValue roleValue;

    public Role(RoleValue role) {
        roleValue = role;
    }

    public Role(String role) {
        roleValue = RoleValue.valueOf(role);
    }

    public RoleValue getRoleValue() {
        return roleValue;
    }

    @Override
    public String getAuthority() {
        return roleValue.toString();
    }

    public boolean isAdmin() {
        return roleValue == RoleValue.ADMIN;
    }
}
```

The additional tests are in the RoleTest.java file.

Task 2

Class: ModifyRepoService

AddNode method seemed critical to me. There were many checks if the node can be added to the database. I found out that the tests if the user who is creating node has the same id as the owner of the node wasn't written. Thus I have created the mutant method which doesn't perform this check. I have created a new api as well to test it more easily called addNodeMutant.

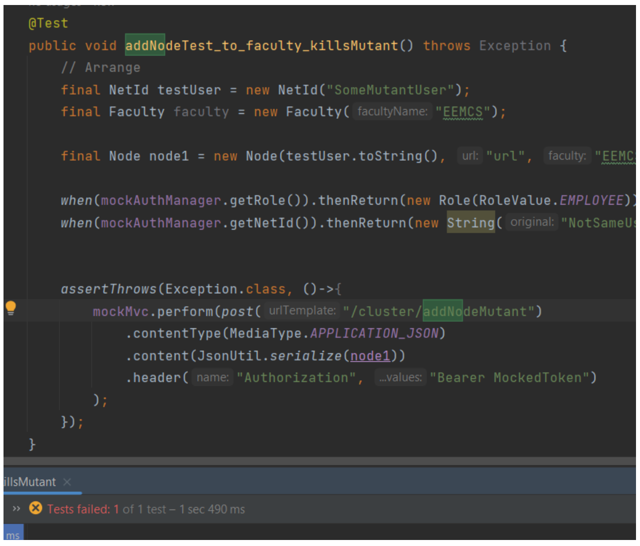


```java
@PostMapping(path = {"/addNode"})
public Node addNode(Node node, String netId, List<String> faculties) throws NullValueException,
        ResourceMismatchException, InvalidOwnerException, InvalidFacultyException {
    checkNullValues(node);
    checkResources(node);
    checkNameAndId(node, netId);
    facultyContained(node, faculties);
    repo.save(node);
    return node;
}

/**
 * Endpoint where you can add node. The node has to belong to the faculty you are in.
 * The nodes resources has to match cpu >= gpu || cpu >= mem
 *
 * @param node you want to add
 */
1 usage   new *
@PostMapping(path = {"/addNodeMutant"})
public Node addNodeMutant(Node node, String netId, List<String> faculties) throws NullValueException,
    ResourceMismatchException, InvalidOwnerException, InvalidFacultyException {
    checkNullValues(node);
    checkResources(node);
    facultyContained(node, faculties);
    repo.save(node);
    return node;
}
```

All the tests passed. Then I created a test case which fails for the mutant. Test case on the left fails because I call the mutant api while the test case passes as I call the regular correct api.

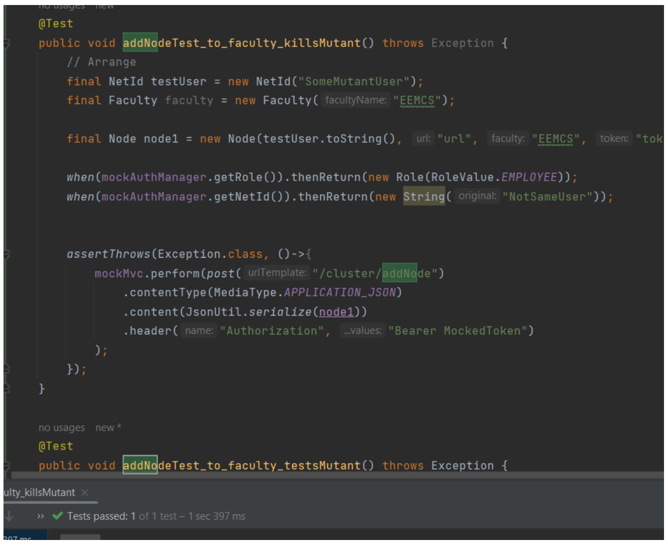
Commit:

https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM09b/-/commit/61ea4f93bcdccb921357133c2bd134c267c1b6ee



```
@Test
public void addNodeTest_to_faculty_killsMutant() throws Exception {
    // Arrange
    final NetId testUser = new NetId("SomeMutantUser");
    final Faculty faculty = new Faculty( facultyName: "EEMCS");

    final Node node1 = new Node(testUser.toString(), url: "url", faculty: "EEMC

    when(mockAuthManager.getRole()).thenReturn(new Role(RoleValue.EMPLOYEE))
    when(mockAuthManager.getNetId()).thenReturn(new String( original: "NotSameU


    assertThrows(Exception.class, ()->{
        mockMvc.perform(post( urlTemplate: "/cluster/addNodeMutant")
            .contentType(MediaType.APPLICATION_JSON)
            .content(JsonUtil.serialize(node1))
            .header( name: "Authorization", ...values: "Bearer MockedToken")
        );
    });
}
```

```
illsMutant
Tests failed: 1 of 1 test – 1 sec 490 ms
```



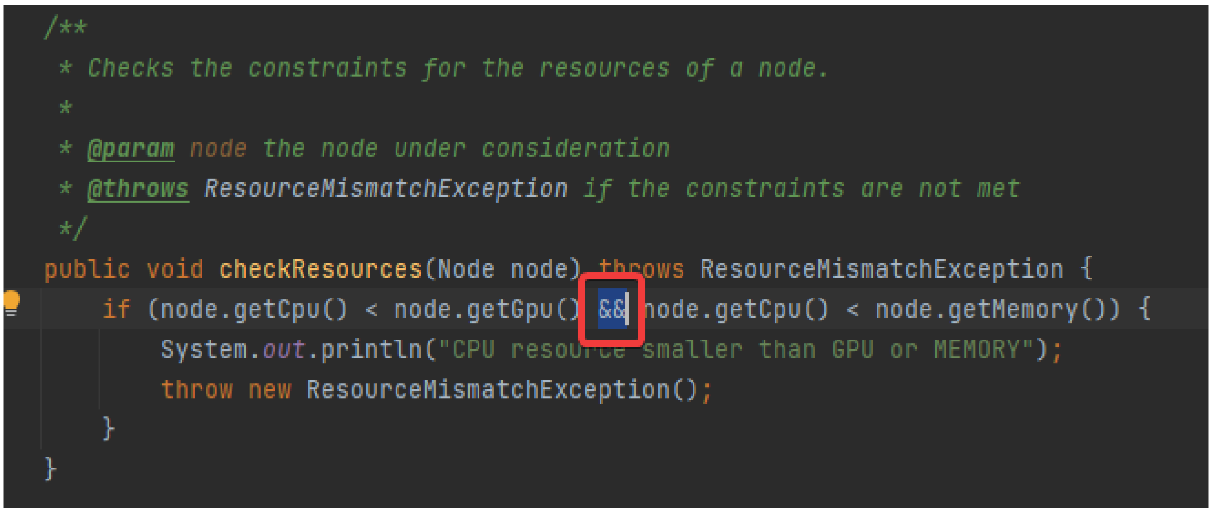
```
@Test
public void addNodeTest_to_faculty_killsMutant() throws Exception {
    // Arrange
    final NetId testUser = new NetId("SomeMutantUser");
    final Faculty faculty = new Faculty( facultyName: "EEMCS");

    final Node node1 = new Node(testUser.toString(), url: "url", faculty: "EEMCS", token: "tok

    when(mockAuthManager.getRole()).thenReturn(new Role(RoleValue.EMPLOYEE));
    when(mockAuthManager.getNetId()).thenReturn(new String( original: "NotSameUser"));


    assertThrows(Exception.class, ()->{
        mockMvc.perform(post( urlTemplate: "/cluster/addNode")
            .contentType(MediaType.APPLICATION_JSON)
            .content(JsonUtil.serialize(node1))
            .header( name: "Authorization", ...values: "Bearer MockedToken")
        );
    });
}

no usages   new *
@Test
public void addNodeTest_to_faculty_testsMutant() throws Exception {
```

```
ulty_killsMutant
Tests passed: 1 of 1 test – 1 sec 397 ms
```

Class: ModifyRepoHelper

This class is critical in our application as it provides methods that verify correctness of Node objects. With any, even subtle mistakes in this class, it is possible that incorrect nodes will be introduced to the system's clusters which could destabilize it and result in wrong schedules being made.

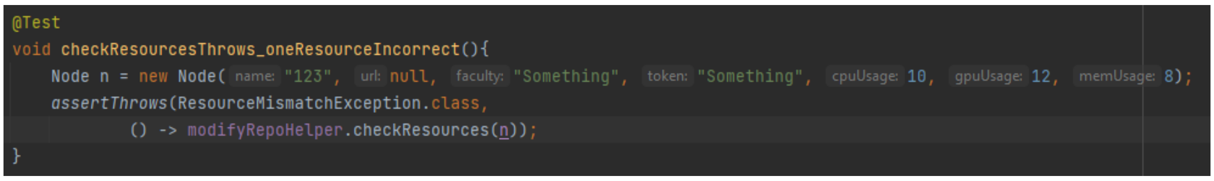
The chosen method from this class is **checkResources(Node node)** which checks if the *CPU >= MAX(GPU, Memory)* requirement is met. I injected a bug which changes the condition in the if statement from || (OR) to && (AND):



```
/**
 * Checks the constraints for the resources of a node.
 *
 * @param node the node under consideration
 * @throws ResourceMismatchException if the constraints are not met
 */
public void checkResources(Node node) throws ResourceMismatchException {
    if (node.getCpu() < node.getGpu() && node.getCpu() < node.getMemory()) {
        System.out.println("CPU resource smaller than GPU or MEMORY");
        throw new ResourceMismatchException();
    }
}
```

This change makes the function throw an exception only if **both** GPU and memory are greater than the CPU resource. This is clearly breaking the requirement, but it was not caught by the existing tests.

To catch this error, the following test was created:

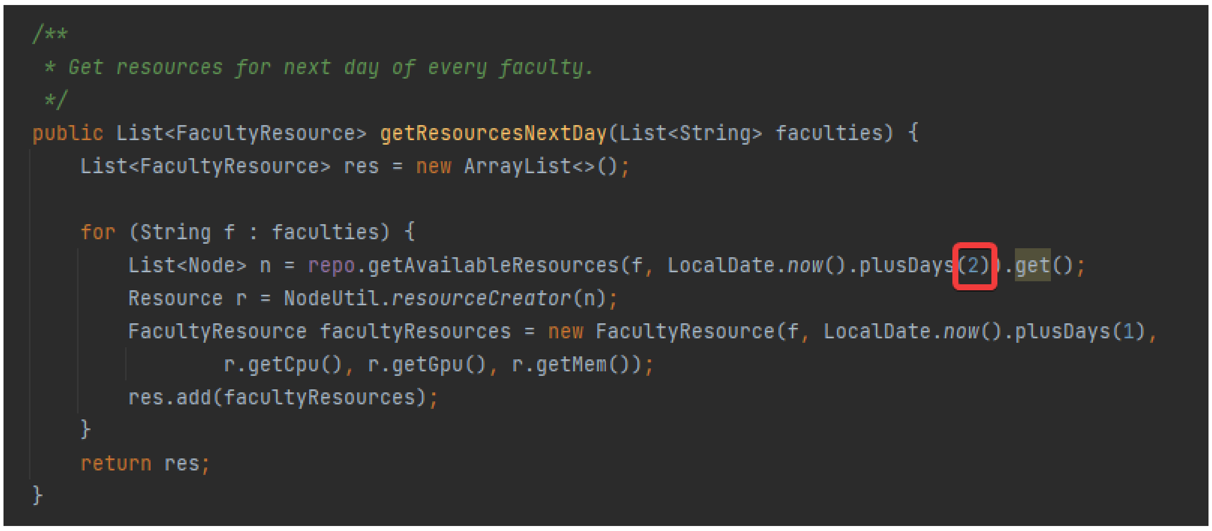


```
@Test
void checkResourcesThrows_oneResourceIncorrect(){
    Node n = new Node( name: "123",  url: null,  faculty: "Something",  token: "Something",  cpuUsage: 10,  gpuUsage: 12,  memUsage: 8);
    assertThrows(ResourceMismatchException.class,
            () -> modifyRepoHelper.checkResources(n));
}
```

It checks if the method throws an exception if only one resource (here GPU) is greater than CPU. It catches the manually created mutant.
Commit:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM09b/-/commit/87d02808cb6aa62838f0897ee3a33ee71509c105

Class: GetResourceService

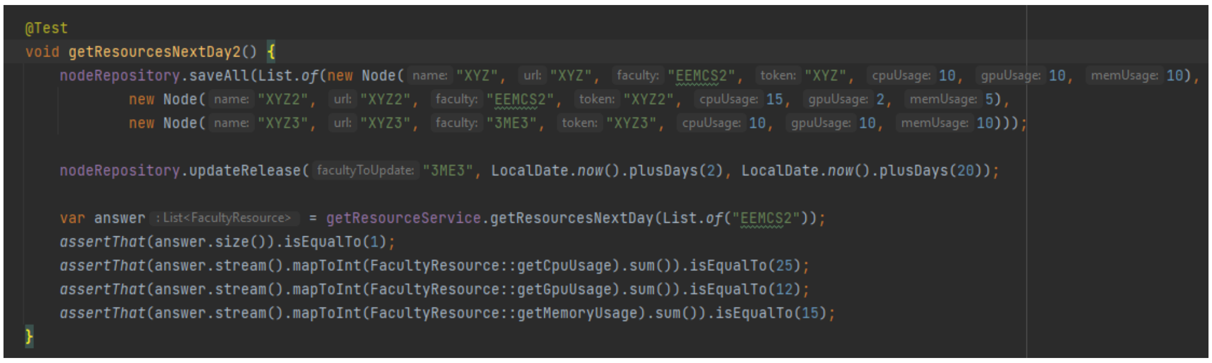
This class was chosen because it provides two functions that give an overview of available resources for the specified day (for an admin) or the next day (for the user). It is important that the system outputs correct information and not confuse users and admins.
The chosen method from this class is **getResourcesNextDay(List<String> faculties)** which returns the list of resources available for the next day. Since this function also considers freed clusters, it is important that it includes only clusters that are free in the next day.
The injected bug makes this function return resources from clusters that are free in 2 days, and not necessarily tomorrow:



```
/**
 * Get resources for next day of every faculty.
 */
public List<FacultyResource> getResourcesNextDay(List<String> faculties) {
    List<FacultyResource> res = new ArrayList<>();

    for (String f : faculties) {
        List<Node> n = repo.getAvailableResources(f, LocalDate.now().plusDays(2)).get();
        Resource r = NodeUtil.resourceCreator(n);
        FacultyResource facultyResources = new FacultyResource(f, LocalDate.now().plusDays(1),
                r.getCpu(), r.getGpu(), r.getMem());
        res.add(facultyResources);
    }
    return res;
}
```

Instead of *.plusDays(1)*, this function calls getAvailableResources with *.plusDays(2)*. It will now return available resources from clusters that were freed before that day and that are freed until that day. This bug was not caught with current tests.

The following tests catches the bug:



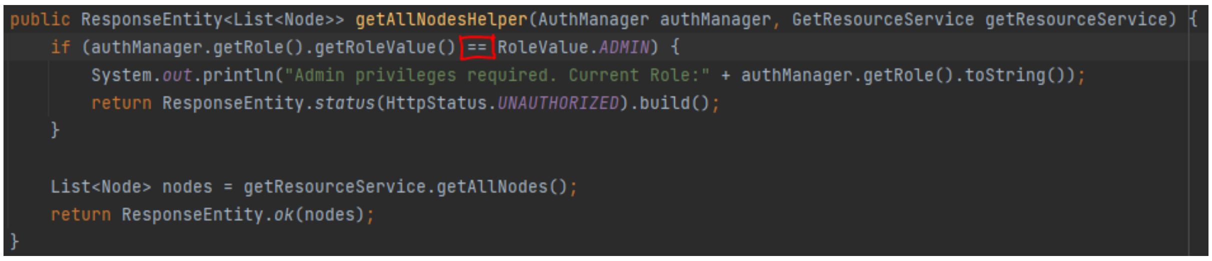
```
@Test
void getResourcesNextDay2() {
    nodeRepository.saveAll(List.of(new Node( name: "XYZ",  url: "XYZ",  faculty: "EEMCS2",  token: "XYZ",  cpuUsage: 10,  gpuUsage: 10,  memUsage: 10),
            new Node( name: "XYZ2",  url: "XYZ2",  faculty: "EEMCS2",  token: "XYZ2",  cpuUsage: 15,  gpuUsage: 2,  memUsage: 5),
            new Node( name: "XYZ3",  url: "XYZ3",  faculty: "3ME3",  token: "XYZ3",  cpuUsage: 10,  gpuUsage: 10,  memUsage: 10)));

    nodeRepository.updateRelease( facultyToUpdate: "3ME3", LocalDate.now().plusDays(2), LocalDate.now().plusDays(20));

    var answer : List<FacultyResource> = getResourceService.getResourcesNextDay(List.of("EEMCS2"));
    assertThat(answer.size()).isEqualTo(1);
    assertThat(answer.stream().mapToInt(FacultyResource::getCpuUsage).sum()).isEqualTo(25);
    assertThat(answer.stream().mapToInt(FacultyResource::getGpuUsage).sum()).isEqualTo(12);
    assertThat(answer.stream().mapToInt(FacultyResource::getMemoryUsage).sum()).isEqualTo(15);
}
```

It frees a node, but only from 2 days from now - not from tomorrow. It should not be included in the list of resources for tomorrow. The test failed and thus killed the manually injected mutant.
Commit:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM09b/-/commit/595d2524a03938d42ed7dbe54d8b075fdf7de019

Class: CheckHelper

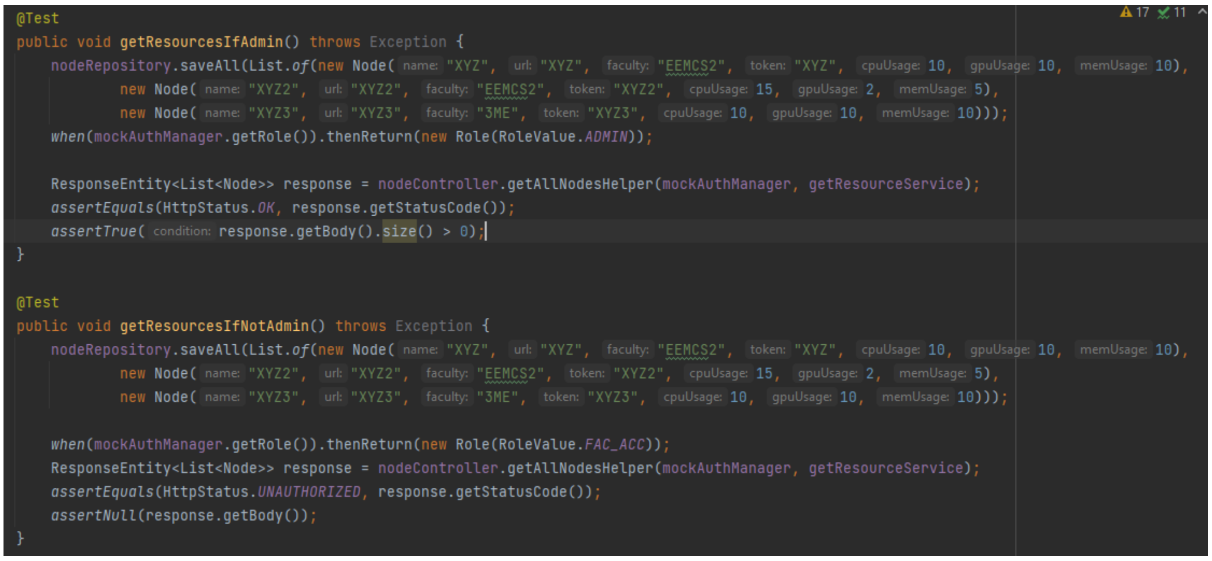
This class was chosen because it contains helper methods to verify the data or change the data accordingly. Errors in this class will result in incorrect Nodes assignment or the ability to access Nodes with the wrong Role value (eg. not Admin).
The chosen method is **getAllNodesHelper(AuthManager authManager, GetResourceService getResourceService)** because it is critical. The requirements have set out that only the admin should be able to see all nodes. If this method is wrong, all nodes could potentially be accessed by anyone which goes against the requirements set. It is important that only the Admin can see all Nodes at once.

The injected bug makes it so all Nodes can be seen if your role is not admin, and admin now has no access to see all nodes in the network at once.



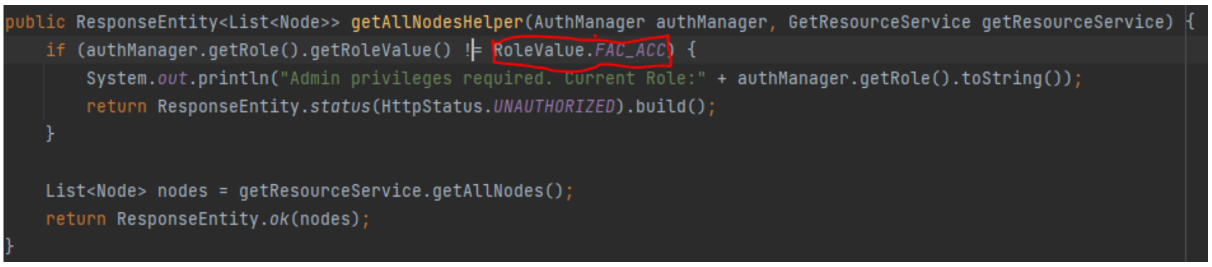
```
public ResponseEntity<List<Node>> getAllNodesHelper(AuthManager authManager, GetResourceService getResourceService) {
    if (authManager.getRole().getRoleValue() == RoleValue.ADMIN) {
        System.out.println("Admin privileges required. Current Role:" + authManager.getRole().toString());
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
    }

    List<Node> nodes = getResourceService.getAllNodes();
    return ResponseEntity.ok(nodes);
}
```

Instead of ".getRoleValue() != RoleValue.ADMIN" the '!=" becomes '==' which is the opposite of the correct functionality. It will now not allow Admin users to call getAllNodes() which they should be able to. This mutant was not originally caught. The following tests catch the bug:



```
@Test
public void getResourcesIfAdmin() throws Exception {
    nodeRepository.saveAll(List.of(new Node( name: "XYZ",  url: "XYZ",  faculty: "EEMCS2",  token: "XYZ",  cpuUsage: 10,  gpuUsage: 10,  memUsage: 10),
            new Node( name: "XYZ2",  url: "XYZ2",  faculty: "EEMCS2",  token: "XYZ2",  cpuUsage: 15,  gpuUsage: 2,  memUsage: 5),
            new Node( name: "XYZ3",  url: "XYZ3",  faculty: "3ME",  token: "XYZ3",  cpuUsage: 10,  gpuUsage: 10,  memUsage: 10)));
    when(mockAuthManager.getRole()).thenReturn(new Role(RoleValue.ADMIN));

    ResponseEntity<List<Node>> response = nodeController.getAllNodesHelper(mockAuthManager, getResourceService);
    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertTrue( condition: response.getBody().size() > 0);
}

@Test
public void getResourcesIfNotAdmin() throws Exception {
    nodeRepository.saveAll(List.of(new Node( name: "XYZ",  url: "XYZ",  faculty: "EEMCS2",  token: "XYZ",  cpuUsage: 10,  gpuUsage: 10,  memUsage: 10),
            new Node( name: "XYZ2",  url: "XYZ2",  faculty: "EEMCS2",  token: "XYZ2",  cpuUsage: 15,  gpuUsage: 2,  memUsage: 5),
            new Node( name: "XYZ3",  url: "XYZ3",  faculty: "3ME",  token: "XYZ3",  cpuUsage: 10,  gpuUsage: 10,  memUsage: 10)));

    when(mockAuthManager.getRole()).thenReturn(new Role(RoleValue.FAC_ACC));
    ResponseEntity<List<Node>> response = nodeController.getAllNodesHelper(mockAuthManager, getResourceService);
    assertEquals(HttpStatus.UNAUTHORIZED, response.getStatusCode());
    assertNull(response.getBody());
}
```

The first tests if the role is equal to Admin, only then it returns the list of all nodes. The second test pretends to be a FAC_ACC (Not Admin), therefore it should get the unauthorized response. The test failed and thus killed the manually injected mutant.
These tests also kill the following mutation where RoleValue.ADMIN changed to RoleValue.FAC_ACC.



```
public ResponseEntity<List<Node>> getAllNodesHelper(AuthManager authManager, GetResourceService getResourceService) {
    if (authManager.getRole().getRoleValue() != RoleValue.FAC_ACC) {
        System.out.println("Admin privileges required. Current Role:" + authManager.getRole().toString());
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
    }

    List<Node> nodes = getResourceService.getAllNodes();
    return ResponseEntity.ok(nodes);
}
```

Commit: 4cba19e3812a16df8474e3307aaa0108d2bfc608
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM09b/-/commit/4cba19e3812a16df8474e3307aaa0108d2bfc608