

Assignment 3 Part 2

CSE2115: Software Engineering Methods

Group 06b

Assignment 3 Part 2

by

Group 06b

Instructor:	Dr. A. Panichella & F. Mulder
Teaching Assistant:	M. Segers
Institution:	Delft University of Technology
Place:	Faculty of Electrical Engineering, Mathematics and Computer Science
Date:	20-01-2023

Contents

1	Manual Mutation Testing	1
	List of Figures	7

1

Manual Mutation Testing

Introduction

We selected order microservice as core domain, because it is central to the project and communicates most frequently with other microservices.

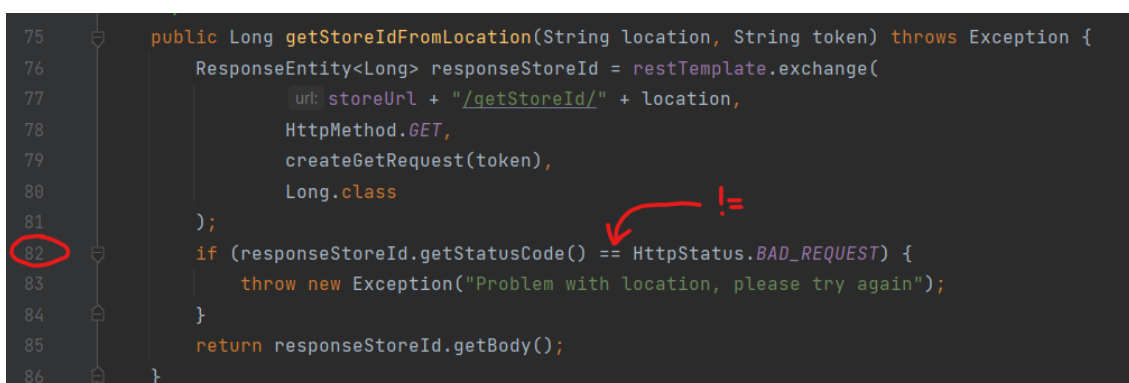
1. StoreCommunication.getStoreIdFromLocation()

1. Class StoreCommunication is one of the most critical classes because it is responsible for communication with store microservice. Communication between these two microservices is crucial for the food ordering system.

2. A manual bug was injected in `getStoreIdFromLocation()` method on line 82. The injected bug is a relational operator replacement, in particular changing an equality to an inequality. The method was chosen based on the fact that the communication establishes the validity and identification of the store and is very important to the food ordering application. The injected bug catches if the response is not valid, but still goes though as valid.

3. The mutant was caught after introducing the `getStoreIdFromLocationTest()` in `StoreCommunicationTest`.

Commit link: [e98bce049ed97e7d5437c3ff761f2f7b67f043e0](https://github.com/e98bce049ed97e7d5437c3ff761f2f7b67f043e0)

A screenshot of a code editor showing the `getStoreIdFromLocation` method in a Java class. The code is as follows:

```
75 public Long getStoreIdFromLocation(String location, String token) throws Exception {  
76     ResponseEntity<Long> responseStoreId = restTemplate.exchange(  
77         url: storeUrl + "/getStoreId/" + location,  
78         HttpMethod.GET,  
79         createGetRequest(token),  
80         Long.class  
81     );  
82     if (responseStoreId.getStatusCode() == HttpStatus.BAD_REQUEST) {  
83         throw new Exception("Problem with location, please try again");  
84     }  
85     return responseStoreId.getBody();  
86 }
```

The line number 82 is circled in red. A red arrow points from the text "!=" to the equals sign in the `getStatusCode() == HttpStatus.BAD_REQUEST` condition on line 82, indicating the location of the injected bug.

Figure 1.1: `getStoreIdFromLocation` and place of the injected bug

```

27     @Test
28     public void getStoreIdFromLocationTest() {
29         String location = "location";
30         String token = "token";
31
32         HttpHeaders headerForValidation = new HttpHeaders();
33         headerForValidation.set("Authorization", String.format("Bearer %s", token));
34         HttpEntity he = new HttpEntity(headerForValidation);
35
36         StoreCommunication sc = new StoreCommunication(restTemplateMock);
37
38         ResponseEntity<Long> response = new ResponseEntity(HttpStatus.BAD_REQUEST);
39
40         when(restTemplateMock.exchange(
41             url: storeUrl + "/getStoreId/" + location,
42             HttpMethod.GET,
43             he,
44             Long.class
45         )).thenReturn(response);
46
47         Exception e = assertThrows(
48             Exception.class,
49             () -> sc.getStoreIdFromLocation(location, token),
50             message: "Problem with location, please try again");
51         assert e.getMessage().equals("Problem with location, please try again");
52     }
53 }

```

Figure 1.2: getStoreIdFromLocationTest

2. OrderProcessorImpl.fetchAllStoreOrders()

1. Class `OrderProcessorImpl` is another critical class because it is responsible for processing the orders. It contains several important methods such as starting the order and cancelling the order.

2. The manually generated mutant was injected in `fetchAllStoreOrders()` method on line 278. The mutant is generated using the return statement replacement operator, namely replacing the returned boolean value of `Objects.equals()` method by **true** on line 282. This method is chosen because it is a core method for querying order data of a specific store from the order database. Before introducing new test, the mutant survived as the result list contains all orders from a specific store, nevertheless, the result list also contains orders of other stores and this is undesirable.

3. The mutant was killed by adding the `testFetchAllStoreOrdersKillingMutant()` in `OrderProcessorTest` class.

Commit link: [c15a472dbb54d0b58344ca11f3a0d1eaf5b760f3](https://github.com/1024j1024/OrderProcessor/commit/c15a472dbb54d0b58344ca11f3a0d1eaf5b760f3)

```

277      @Override
278      public Collection<Order> fetchAllStoreOrders(String token, String memberId,
279                                                  String roleName, Long storeId) throws Exception {
280
281          List<Order> orders = orderRepository.findAll().stream()
282              .filter(x -> Objects.equals(x.getStoreId(), storeId)).collect(Collectors.toList());
283

```

Figure 1.3: fetchAllStoreOrders before mutant injection

```

277      @Override
278      public Collection<Order> fetchAllStoreOrders(String token, String memberId,
279                                                  String roleName, Long storeId) throws Exception {
280
281          List<Order> orders = orderRepository.findAll().stream()
282              .filter(x -> true).collect(Collectors.toList());
283

```

Figure 1.4: fetchAllStoreOrders after mutant injection

```

429      @Test
430      public void testFetchAllStoreOrdersKillingMutant() throws Exception {
431          Order order1 = new Order();
432          order1.setStoreId(1L);
433          order1.setId(1L);
434          Order order2 = new Order();
435          order2.setStoreId(2L);
436          order2.setId(2L);
437          Order order3 = new Order();
438          order3.setStoreId(1L);
439          order3.setId(3L);
440          List<Order> orders = List.of(order1, order2, order3);
441
442          when(mockOrderRepository.findAll()).thenReturn(orders);
443          when(mockStoreCommunication.validateManager(manager: "man", token: "token")).thenReturn(value: true);
444          when(mockStoreCommunication.getStoreIdFromManager(manager: "man", token: "token")).thenReturn(value: 1L);
445
446          Collection<Order> res = orderProcessor.fetchAllStoreOrders(token: "token", memberId: "man",
447                                                                    roleName: "customer", storeId: 1L);
448          assertThat(res).containsExactlyInAnyOrder(order1, order3);
449      }

```

Figure 1.5: testFetchAllStoreOrdersKillingMutant

3. OrderEditorImpl.addPizza()

1. Class `OrderEditorImpl` is an extremely important class as it is directly responsible for directly modifying the orders upon user and admin requests. It can add new pizzas and toppings to the order, as well as remove them. Having this class be prone to mutations would be extremely detrimental to the system, potentially allowing for fraud, glitches, etc. by users adding non-existent toppings, or attempting to remove data that is not even there.

2. A mutation was added in the `addPizza()` method on line 61. The bug injected is a removed method call, to the `validatePizza()` method in the other service. This method was chosen as it is one of the most crucial methods in this class since it is directly responsible for all the logic for adding new pizzas to the order. If a pizza is not properly verified to exist in the system, it could lead to fraud from the user's part. This bug allows the entire method to forgo the verification of a pizza and for an invalid pizza to be added to the order.

3. After adding the `testAddValidPizza()` in `OrderServiceEditingTests` class, the mutation was killed.

Commit link: [261fdc97286b7659306ac1a8f95b1d4f64f215b0](https://github.com/261fdc97286b7659306ac1a8f95b1d4f64f215b0)

```
@Override
public Allergens addPizza(String token, String memberId, Long orderId, Pizza pizza) throws Exception {
    if (token == null) {
        throw new Exception(InvalidToken);
    } else if (orderId == null) {
        throw new Exception(InvalidOrderId);
    } else if (orderRepository.findOne(orderId) == null
        || orderRepository.findOne(orderId).getStatus() != Status.ORDER_ONGOING) {
        throw new Exception(noActiveOrderMessage);
    } else if (pizza == null) {
        throw new Exception(InvalidPizza);
    } else if (memberId == null) {
        throw new Exception(InvalidMemberId);
    }

    Allergens allergens = new Allergens( allergensContent: "No allergens");

    // Query the Menu to see if pizza is valid
    menuCommunication.validatePizza(pizza, token);

    // Query the Menu to see if pizza contains allergens and store the response to inform the user
    String responseMessage = menuCommunication.containsAllergen(pizza, memberId, token);
    if (responseMessage != null && !responseMessage.equals("")) {...}

    Order order = orderRepository.findOne(orderId);
    OrderBuilder orderBuilder = Builder.toBuilder(order);
    orderBuilder.addPizza(pizza);

    Order newOrder = orderBuilder.build();
    orderRepository.save(newOrder);

    return allergens;
}
```

Figure 1.6: `addPizza()` before mutant injection

```

@Override
public Allergens addPizza(String token, String memberId, Long orderId, Pizza pizza) throws Exception {
    if (token == null) {
        throw new Exception(InvalidToken);
    } else if (orderId == null) {
        throw new Exception(InvalidOrderId);
    } else if (orderRepository.getOne(orderId) == null
        || orderRepository.getOne(orderId).getStatus() != Status.ORDER_ONGOING) {
        throw new Exception(noActiveOrderMessage);
    } else if (pizza == null) {
        throw new Exception(InvalidPizza);
    } else if (memberId == null) {
        throw new Exception(InvalidMemberId);
    }

    Allergens allergens = new Allergens(allergensContent: "No allergens");

    // Query the Menu to see if pizza contains allergens and store the response to inform the user
    String responseMessage = menuCommunication.containsAllergen(pizza, memberId, token);
    if (responseMessage != null && !responseMessage.equals("")) {...}

    Order order = orderRepository.getOne(orderId);
    OrderBuilder orderBuilder = Builder.toBuilder(order);
    orderBuilder.addPizza(pizza);

    Order newOrder = orderBuilder.build();
    orderRepository.save(newOrder);

    return allergens;
}

```

Figure 1.7: addPizza() after mutant injection

```

@Test
public void testAddValidPizza() throws Exception {
    Pizza pizza = new Pizza(pizzaId: 666L, List.of(10L, 12L), new BigDecimal(val: "11.29"));
    when(menuCommunication.containsAllergen(pizza, memberId: "human", token: "userToken")).thenReturn(value: "");
    ordEdit.addPizza(token: "userToken", memberId: "human", orderId: 7L, pizza);
    verify(menuCommunication).validatePizza(pizza, token: "userToken");
}

```

Figure 1.8: addPizzaTest

3. MenuCommunication.validatePizza()

1. The MenuCommunication class is critical for our application as the user sends a list of pizza ids and topping ids to the Order Microservice. The user is allowed to input any id that they wish, so they could input an id that isn't recognized. The MenuCommunication class checks with the menu microservice if these ids are established so that the orders are only created with known id's.

2. We manually injected a Relational Operator Replacement mutant in the validatePizza() method in the MenuCommunication class. The validatePizza() method sends a request to the menu microservice to make sure that the pizza id is valid and throws an exception if the HTTP status is not an OK response or if the id is not known to the menu microservice. In order to test the mutant, we negated the equality sign when checking if the id was known to the menu microservice.

3. After adding the validatePizzaTest() in the MenuCommunication class, the mutation was killed.

Commit link: 95122b1575e5fa93c3279b68f02f33b47ecdcd5


```

public void validatePizza(Pizza pizza, String token) throws Exception {
    HttpHeaders headers = makeHeader(token);
    headers.setContentType(MediaType.APPLICATION_JSON);

    Map<String, Object> map = new HashMap<>();
    map.put("id", pizza.getPizzaId());
    map.put("toppingIds", pizza.getToppings());
    HttpEntity<Map<String, Object>> entity = new HttpEntity<>(map, headers);

    ResponseEntity<Boolean> response = restTemplate.postForEntity(url: menuUrl + "/isValid", entity, Boolean.class);

    if (response.getStatusCode() != HttpStatus.OK || response.getBody() == false) {
        throw new Exception("Pizza " + pizza.getPizzaId() + " is not valid");
    }
}

```

Figure 1.9: validatePizza() before mutant injection

```

public void validatePizza(Pizza pizza, String token) throws Exception {
    HttpHeaders headers = makeHeader(token);
    headers.setContentType(MediaType.APPLICATION_JSON);

    Map<String, Object> map = new HashMap<>();
    map.put("id", pizza.getPizzaId());
    map.put("toppingIds", pizza.getToppings());
    HttpEntity<Map<String, Object>> entity = new HttpEntity<>(map, headers);

    ResponseEntity<Boolean> response = restTemplate.postForEntity(url: menuUrl + "/isValid", entity, Boolean.class);

    if (response.getStatusCode() != HttpStatus.OK || response.getBody() != false) {
        throw new Exception("Pizza " + pizza.getPizzaId() + " is not valid");
    }
}

```

Figure 1.10: validatePizza() after mutant injection

```

@Test
public void validatePizzaTest() {

    doReturn(ResponseEntity.ok(body: true)).when(restTemplate).postForEntity(
        Mockito.anyString(), Mockito.<HttpEntity<?>>any(), Mockito.<Class<Map>>any());

    MenuCommunication mc = new MenuCommunication(restTemplate);

    Assertions.assertThatNoException().isThrownBy(() -> mc.validatePizza(pizza, token: "token"));
}

```

Figure 1.11: validatePizzaTest

List of Figures

1.1	getStoreIdFromLocation and place of the injected bug	1
1.2	getStoreIdFromLocationTest	2
1.3	fetchAllStoreOrders before mutant injection	3
1.4	fetchAllStoreOrders after mutant injection	3
1.5	testFetchAllStoreOrdersKillingMutant	3
1.6	addPizza() before mutant injection	4
1.7	addPizza() after mutant injection	5
1.8	addPizzaTest	5
1.9	validatePizza() before mutant injection	6
1.10	validatePizza() after mutant injection	6
1.11	validatePizzaTest	6