PIT Report

Initial coverage

The initial coverage of the codebase before applying mutation testing practises as shown in the figure below.

It indicated, that there is a significant lack of killed mutations; but a relatively high test strength taking our novice level testing knowledge into account.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
13	55%	105/192	58%	64/110	86%	64/74

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.softwaretesting.testing.config	1	0%	0/16	0%	0/4	0%	0/0
com.softwaretesting.testing.customerManagement.controller	1	0%	0/12	0%	0/6	0%	0/0
com.softwaretesting.testing.customerManagement.service	1	100%	23/23	67%	8/12	67%	8/12
com.softwaretesting.testing.customerRegistration.controller	1	0%	0/6	0%	0/1	0%	0/0
com.softwaretesting.testing.customerRegistration.service	1	100%	11/11	100%	3/3	100%	3/3
com.softwaretesting.testing.dto.inbound	1	0%	0/15	0%	0/7	0%	0/0
com.softwaretesting.testing.dto.outbound	1	0%	0/23	0%	0/4	0%	0/0
com.softwaretesting.testing.model	1	84%	21/25	31%	5/16	71%	5/7
com.softwaretesting.testing.util	2	72%	23/32	83%	34/41	94%	34/36
com.softwaretesting.testing.validator	3	93%	27/29	88%	14/16	88%	14/16

Report generated by PIT 1.8.0

Line coverage and Mutation coverage seem to be less than sufficient. We aim to fix this in this weeks exercise.

However, when taking a look at the packages, which we did write tests for, we can observe, that the stats look slightly better. As stated in the practical seminar, we will not test DTO's because they are already indirectly testet with the repository instances.

The aim is to improve our tests and reach a better coverage.

Fixing the issues

`CustomerRegistrationServiceImp`

Running PIT in the project, we can immediately observe some uncovered lines regarding the mutation tests.

```
37          Optional<Customer> customer = customerRepository.findByUserName(userName);
38
39 1        customerValidator.validate404(customer, "User-Name", userName);
40
41 1        return customer.get();
42      }
43
44      @Transactional(readOnly = true)
45      @Override
46      public Customer findById(Long id) {
47          Optional<Customer> customer = customerRepository.findById(id);
48
49 1        customerValidator.validate404(customer, "id", String.valueOf(id));
50
51 1        return customer.get();
52      }
53
54      @Transactional(readOnly = true)
55      @Override
56      public Customer selectCustomerByPhoneNumber(String phoneNumber) {
57          Optional<Customer> customer = customerRepository.selectCustomerByPhoneNumber(phoneNu
58
59 1        customerValidator.validate404(customer, "phone number", phoneNumber);
60
61 1        return customer.get();
```

```
59 1        customerValidator.validate404(customer, "phone number", phoneNumber);
60
61 1        return customer.get();
62      }
63
64      @Transactional(rollbackFor = Exception.class)
65      @Override
66      public void delete(Long customerId) {
67
68
69 1        if (!customerRepository.existsById(customerId)) {
70              throw new CustomerNotFoundException(
71                      "Customer with id " + customerId + " does not exists");
72          }
73
74 1        customerRepository.deleteById(customerId);
75      }
76
77      @Transactional(rollbackFor = Exception.class)
78      @Override
79      public Customer addCustomer(Customer customer) {
80          Optional<Customer> existsPhoneNumber = customerRepository.selectCustomerByPhoneNumber(customer.ge
81
82 1        if (existsPhoneNumber.isPresent()) {
83              throw new BadRequestException(
84                      "Phone Number " + customer.getPhoneNumber() + " taken");
85          }
86 1        return customerRepository.save(customer);
87      }
88
89      @Transactional(rollbackFor = Exception.class)
90      @Override
91      public Collection<Customer> saveAll(List<Customer> customers) {
92 1        return StreamSupport.stream(customerRepository.saveAll(customers).spliterator(), false)
93                  .collect(Collectors.toSet());
94      }
95  }
```

Mutations

30 1. replaced return value with Collections.emptyList for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::list → KILLED
39 1. removed call to com/softwaretesting/testing/validator/CustomerValidator::validate404 → SURVIVED
41 1. replaced return value with null for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::findByUserName → KILLED
49 1. removed call to com/softwaretesting/testing/validator/CustomerValidator::validate404 → SURVIVED
51 1. replaced return value with null for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::findById → KILLED
59 1. removed call to com/softwaretesting/testing/validator/CustomerValidator::validate404 → SURVIVED
61 1. replaced return value with null for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::selectCustomerByPhoneNumber → KILLED
69 1. negated conditional → KILLED
74 1. removed call to com/softwaretesting/testing/dao/CustomerRepository::deleteById → SURVIVED
82 1. negated conditional → KILLED
86 1. replaced return value with null for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::addCustomer → KILLED
92 1. replaced return value with Collections.emptyList for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::saveAll → KILLED

```
1   @Test
```

```
2   void findByUserName_Validation404() {
3
    when(customerRepository.findByUserName(USERNAME)).thenReturn(Optional.of
4       doNothing().when(customerValidator).validate404(Optional.of(customer
    Name", USERNAME);
5       Customer result = customerManagementService.findByUserName(USERNAME)
6
7       verify(customerRepository, times(1)).findByUserName(USERNAME);
8       verify(customerValidator, times(1)).validate404(Optional.of(customer
    Name", USERNAME);
9       assert result.equals(customer);
10  }
```

To resolve the removal issues in Lines 39, 49, and 59, we count the invocations of the `customerValidator.validate404` method. We ensure that the method gets called once per invocation of the parent method.

`removed call to <method> → SURVIVED` is solvable by just counting the number of invocations.

Mutations

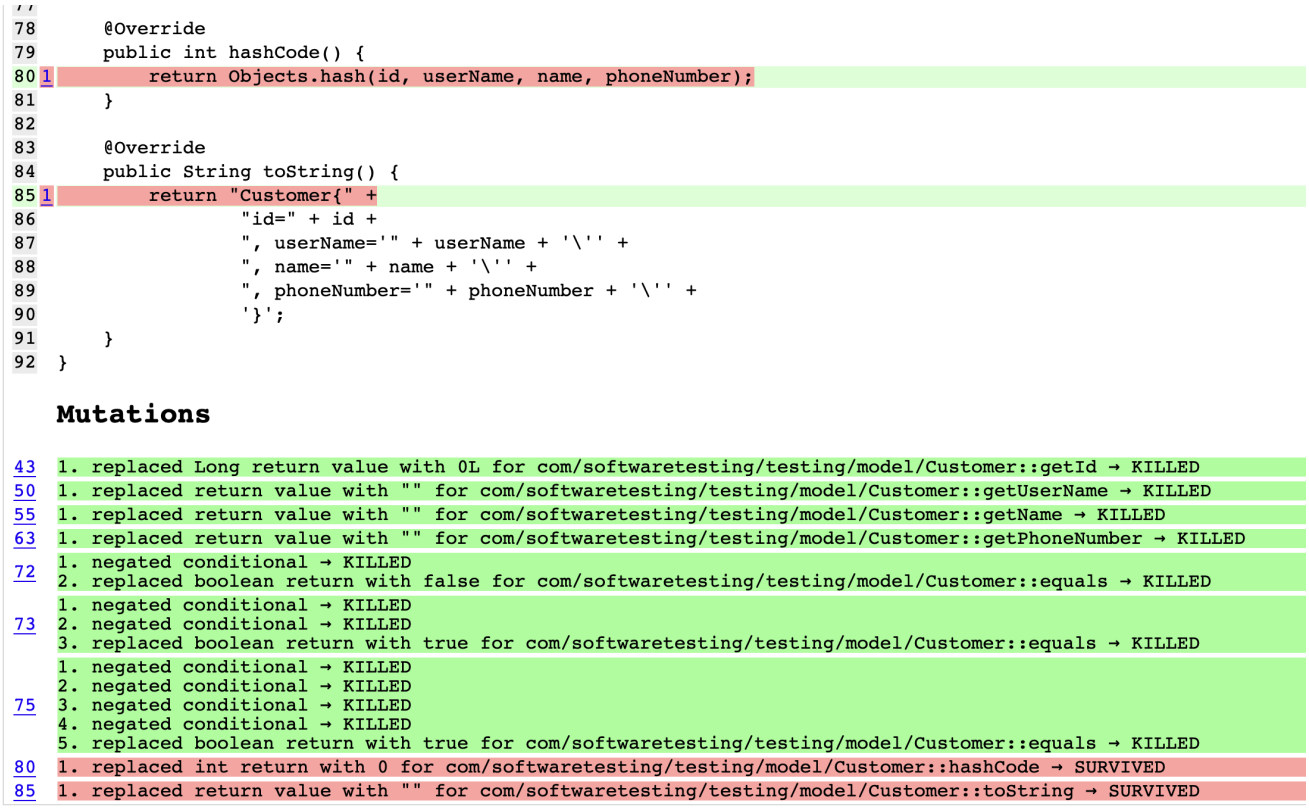
30	1. replaced return value with Collections.emptyList for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::list → KILLED
39	1. removed call to com/softwaretesting/testing/validator/CustomerValidator::validate404 → KILLED
41	1. replaced return value with null for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::findByUserName → KILLED
49	1. removed call to com/softwaretesting/testing/validator/CustomerValidator::validate404 → KILLED
51	1. replaced return value with null for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::findById → KILLED
59	1. removed call to com/softwaretesting/testing/validator/CustomerValidator::validate404 → KILLED
61	1. replaced return value with null for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::selectCustomerByPhoneNumber → KILLED
69	1. negated conditional → KILLED
74	1. removed call to com/softwaretesting/testing/dao/CustomerRepository::deleteById → KILLED
82	1. negated conditional → KILLED
86	1. replaced return value with null for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::addCustomer → KILLED
92	1. replaced return value with Collections.emptyList for com/softwaretesting/testing/customerManagement/service/CustomerManagementServiceImp::saveAll → KILLED

`Customer.java`

This class is not tested completely, and provides a number of different messages from PIT, see figure below.

This includes the warning, that some lines are not covered by tests at all.

After implementing 100% line coverage, we get the following result:



Final test cases for 100% mutation coverage are listed below:

```
@Test
void testHashCode() {
    assertEquals(customerA.hashCode(), customerA.hashCode());
}
```

```
5
6   @Test
7   void testHashCodeNotZero() {
8       assertNotEquals(0, customerA.hashCode());
9   }
10
11  @Test
12  void testToString() {
13      assertEquals(customerA.toString(), customerA.toString());
14  }
15
16  @Test
17  void testToStringNotEmpty() {
18      assertNotEquals("", customerA.toString());
```

`DebugFactorial.java`

Moving on to the next file `DebugFactorial.java`, we observe that it is completely untested at the beginning.

DebugFactorial.java

```
1   package com.softwaretesting.testing.util;
2
3   public class DebugFactorial {
4
5       // Java program to find factorial of given number
6
7       // Method to find factorial of given number
8       static int factorial(int n)
9       {
10          int res = 1;
11          int i;
12 2        for (i=2; i< n; i++)
13 1            res *= i;
14 1        return res;
15      }
16
17          // Driver method
18      public static void main(String[] args)
19      {
20          int num = 5;
21 1        System.out.println("Factorial of "+ num + " is " + factorial(5));
22      }
23  }
```

Mutations

```
12  1. changed conditional boundary → NO_COVERAGE
    2. negated conditional → NO_COVERAGE
13  1. Replaced integer multiplication with division → NO_COVERAGE
14  1. replaced int return with 0 for com/softwaretesting/testing/util/DebugFactorial::factorial → NO_COVERAGE
21  1. removed call to java/io/PrintStream::println → NO_COVERAGE
```

After creating some initial tests, we can see an issue with the parameter value begin `n = 2`, failing to return `2` as the correct factorial of `n =2`.

Debugging the `DebugFactorial.factorial` method using the debugger, we can come to the conclusion, that the loop condition is the issue:

```
7       // Method to find factorial of given number
8       static int factorial(int n)  1 usage  emmanuelcharleson.dapaah        n: 2
9       {
10          int res = 1;   res: 1
11          int i;   i: 2
12          for (i=2; i< n; i++)   n: 2
13              res *= i;   i: 2
14          return res;   res: 1
15      }
```

The test assertion is `assertEquals(2, DebugFactorial.factorial(2));`.

When `i` is allowed to run up to `n`, with the condition `i ⩽ 2` we come to the correct answer.

Testing the `main` method firstly seemed to be trickier, since it required the redirection of the produced output.
Utilities like `PrintStream` and `ByteArrayOutputStream` helped to capture the output of `DebugFactorial.main`.

This resulted in a nearly perfect coverage, with the exception of the class header missing from the line coverage:

DebugFactorial.java

```
1   package com.softwaretesting.testing.util;
2
3   public class DebugFactorial {
4
5       // Java program to find factorial of given number
6
7       // Method to find factorial of given number
8       static int factorial(int n)
9       {
10          int res = 1;
11          int i;
12  2       for (i=2; i<= n; i++)
13  1           res *= i;
14  1       return res;
15      }
16
17          // Driver method
18      public static void main(String[] args)
19      {
20          int num = 5;
21  1       System.out.println("Factorial of "+ num + " is " + factorial(5));
22      }
23  }
```

Mutations

```
12  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
13  1. Replaced integer multiplication with division → KILLED
14  1. replaced int return with 0 for com/softwaretesting/testing/util/DebugFactorial::factorial → KILLED
21  1. removed call to java/io/PrintStream::println → KILLED
```

Creating a placeholder test resolved this issue:

```
1   @Test
2   void testConstructor() {
```

```
3         DebugFactorial debugFactorial = new DebugFactorial();
4         assertNotNull(debugFactorial);
5     }
```

`Misc.java`

Misc has the same issue with the class declaration as `DebugFactorial`. However, since `Misc` has a private constructor. It is being kept that way, because it remains a utility class.

Furthermore, some relevant information is yielded by PIT:

```
51      public static boolean isPrime(int n, int i) {
52 2        if (n <= 0) {
53 1            return false;
54          }
55 2        if (n <= 2) {
56 2            return n == 2;
57          }
58 2        if (n % i == 0) {
59 1            return false;
60          }
61 3        if (i * i > n) {
62 1            return true;
63          }
```

with the footnotes:

```
    3. replaced long return with 0 for com/softwaretesting/testing/util/Misc:
52  1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
53  1. replaced boolean return with true for com/softwaretesting/testing/util
55  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
56  1. negated conditional → KILLED
    2. replaced boolean return with true for com/softwaretesting/testing/util
58  1. Replaced integer modulus with multiplication → KILLED
    2. negated conditional → KILLED
59  1. replaced boolean return with true for com/softwaretesting/testing/util
    1. changed conditional boundary → SURVIVED
61  2. Replaced integer multiplication with division → KILLED
```

Reverting to the Forks source original implementation, we managed to eliminate the first mutation issue. The latter remains, since it's difficult to kill due to algorithmic reasons.

`CustomerValidator` as well as `PhoneNumberValidator` were completely covered from the beginning. However, the helper class, `PhoneNumberValidatorUtils` had two missing mutation kills.

```
8   public class PhoneNumberValidatorUtils {
9       private PhoneNumberValidatorUtils() {
10      }
11
12      private static final String[] codes = {"1", "20", "212", "213", "216", "218", "220", "221", "222", "223", "224", "225", "226", "227",
13
14      /**
15       * All Country Calling Codes
16       *
17       * @see <a href="https://gist.github.com/tahirSmartboy/14cd096f25c24b30c203#file-gistfile1-txt-L223">Borrowed from here.</a>
18       */
19 1    protected static final List<String> COUNTRY_CALLING_CODES = Arrays.stream(codes).map(x -> "+" + x).collect(Collectors.toList());
20
21      /**
22       * Removes all characters, except a leading "+", if there is one.
23       *
```

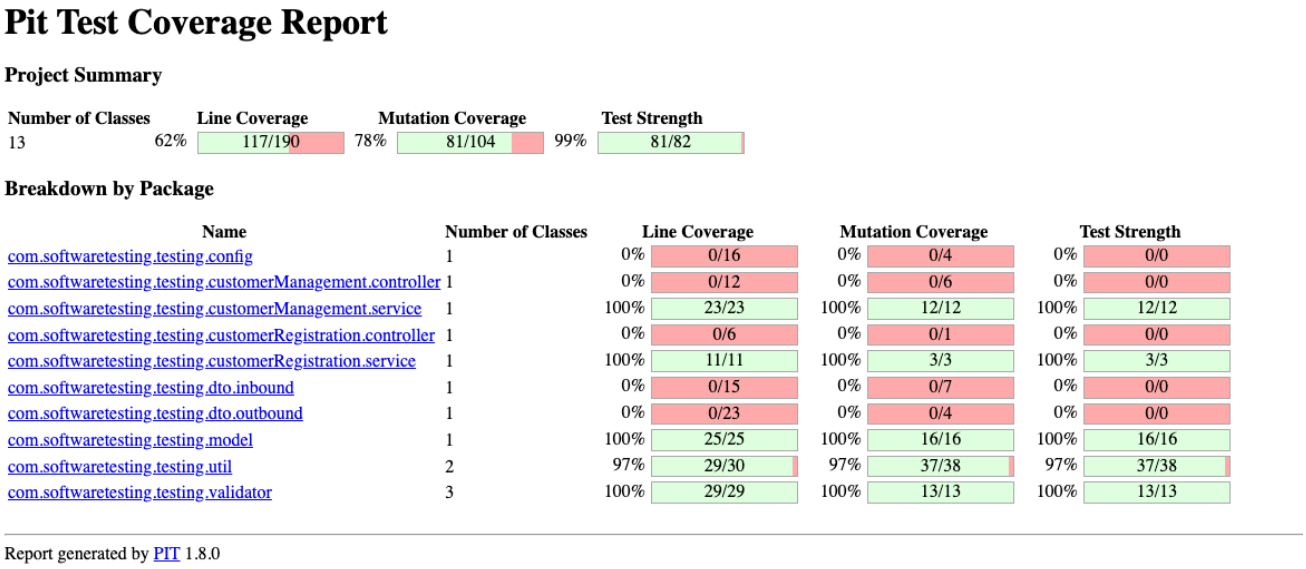
Removing the wrapper / mapping and using literals instead solved the issue. Refactoring the class like debug factorial helped to gain 100% coverage.

No SonarLint issues are in the test folder and older tests are being refactored to hold at most

Conclusion

With the help of the PIT Testing Tool we were able to extend the quality of our test cases. With a nearly complete coverage of the classes from the assignment, we ensured that bugs are harder to introduce by making our test suite more robust.

This is due to the reason, that a 100% coverage cannot always be implemented, because of the nature of some specifications and methods. Therefore, PIT introduces strong indications which test cases need to be reworked.



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
13	62%	117/190	78%	81/104	99%	81/82

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.softwaretesting.testing.config	1	0%	0/16	0%	0/4	0%	0/0
com.softwaretesting.testing.customerManagement.controller	1	0%	0/12	0%	0/6	0%	0/0
com.softwaretesting.testing.customerManagement.service	1	100%	23/23	100%	12/12	100%	12/12
com.softwaretesting.testing.customerRegistration.controller	1	0%	0/6	0%	0/1	0%	0/0
com.softwaretesting.testing.customerRegistration.service	1	100%	11/11	100%	3/3	100%	3/3
com.softwaretesting.testing.dto.inbound	1	0%	0/15	0%	0/7	0%	0/0
com.softwaretesting.testing.dto.outbound	1	0%	0/23	0%	0/4	0%	0/0
com.softwaretesting.testing.model	1	100%	25/25	100%	16/16	100%	16/16
com.softwaretesting.testing.util	2	97%	29/30	97%	37/38	97%	37/38
com.softwaretesting.testing.validator	3	100%	29/29	100%	13/13	100%	13/13

Report generated by PIT 1.8.0