Short report – Assignment 1

1. Found Bugs

1.1. Bug: Throwing storage overflow even if there was enough space

Testmethod in code: saveMessage_shouldSaveSingleShortMessage

Behaviour: When I was testing the saveMessage functionality, it did not save a message even if there was enough space:

Line 43; MobileStorageTest.java:

```
MobileStorage storage = new MobileStorage(1);
String message = "One short Message";
storage.saveMessage(message);
Assertions.assertEquals(message, storage.listMessages());
```

The problem was that this method was throwing a StorageOverflow Exception even if there was enough space, therefore I had to fix the functionality in MobileStorage.java:

```
//BUGFIX
//if(requiredStorage > inbox.length || (inbox.length - occupied) <=
requiredStorage) {--> original line of code
if(requiredStorage > inbox.length || (inbox.length - occupied) <
requiredStorage) {
    throw new NotEnoughSpaceException("Storage Overflow");
}
```

Changing the comparison from “<=” to “<” solved the problem smoothly.

1.2. Bug in search

Testmethod in code: search_shouldFindSearchedTexts

Behaviour: when I was first running this test I got a NPE: 'java.lang.NullPointerException: Cannot invoke "Object.equals(Object)" because the return value of "at.ac.tuwien.inso.peso.MobileMessage.getPredecessor()" is null' at 'at at.ac.tuwien.inso.peso.MobileStorage.lambda$getLastMessage$3(MobileStorage.java:157)'

In order to find the bug I started trying out some approaches and after each approach I was running the still failing unit test to easily find out wether or not my bugfix approach worked.

The .map Method led to the NPE:

```
List<MobileMessage> messages = Arrays.stream(inbox)
    .filter(msg -> msg != null && msg.getText().contains(searchCriteria))
    .map(this::getLastMessage)//calling this method led to NPE --> had to
change code in getLastMessage Method
    .distinct()
    .toList();
```

Therefore I added a condition to check whether or not an object was null:

```
List<MobileMessage> successors = Arrays.stream(inbox)
        .filter(msg -> msg != null && msg.getPredecessor() != null &&
msg.getPredecessor().equals(message))//BUGFIX added: &&
msg.getPredecessor() != null to avoid NPE
        .toList();
```

Adding this condition solved the bug.

1.3. Bug when deleting a message

When trying to run deleteMessage_shouldDeleteOneShortAndOneLongMessage I realized that the behaviour of the

```
storage.deleteMessage();
```

was not what I expected.

Firstly, there was a bug when I tried to delete a message: I ran into an Index out of bound exception and had to add -1 in order to work:

```
inbox[occupied-1] = null;//NOTE: I added '-1' to fix bug
```

Secondly, when I was comparing the result with the expected result in the Assertions.equals() function I expected that the method would delete a multi-part message entirely, but it only had to delete one as it was listed in the assignment as the expected behaviour. Therefore, I adjusted my expected result by prepending

```
"1\n"
```

To my expected result (Line 155).

2. Discussing Testcases

2.1. Testcase

```
public void saveMessage_shouldSaveSingleShortMessage(){//NOTE: this test
would initially fail, this shows me that there is a bug.
```

See Bug 1.1.

2.2. Testcase

```
public void deleteMessage_shouldDeleteOneShortAndOneLongMessage(){//NOTE:
When first running this test, I discovered a
'java.lang.ArrayIndexOutOfBoundsException: Index 6 out of bounds for length
6' at 'at
at.ac.tuwien.inso.peso.MobileStorage.deleteMessage(MobileStorage.java:93)'
```

and

```
public void search_shouldFindSearchedTexts(){//NOTE: when first running
this test I got a NPE: 'java.lang.NullPointerException: Cannot invoke
"Object.equals(Object)" because the return value of
"at.ac.tuwien.inso.peso.MobileMessage.getPredecessor()" is null' at 'at
at.ac.tuwien.inso.peso.MobileStorage.lambda$getLastMessage$3(MobileStorage.
java:157)'
```

See Bug 1.2.

2.3. Testcase

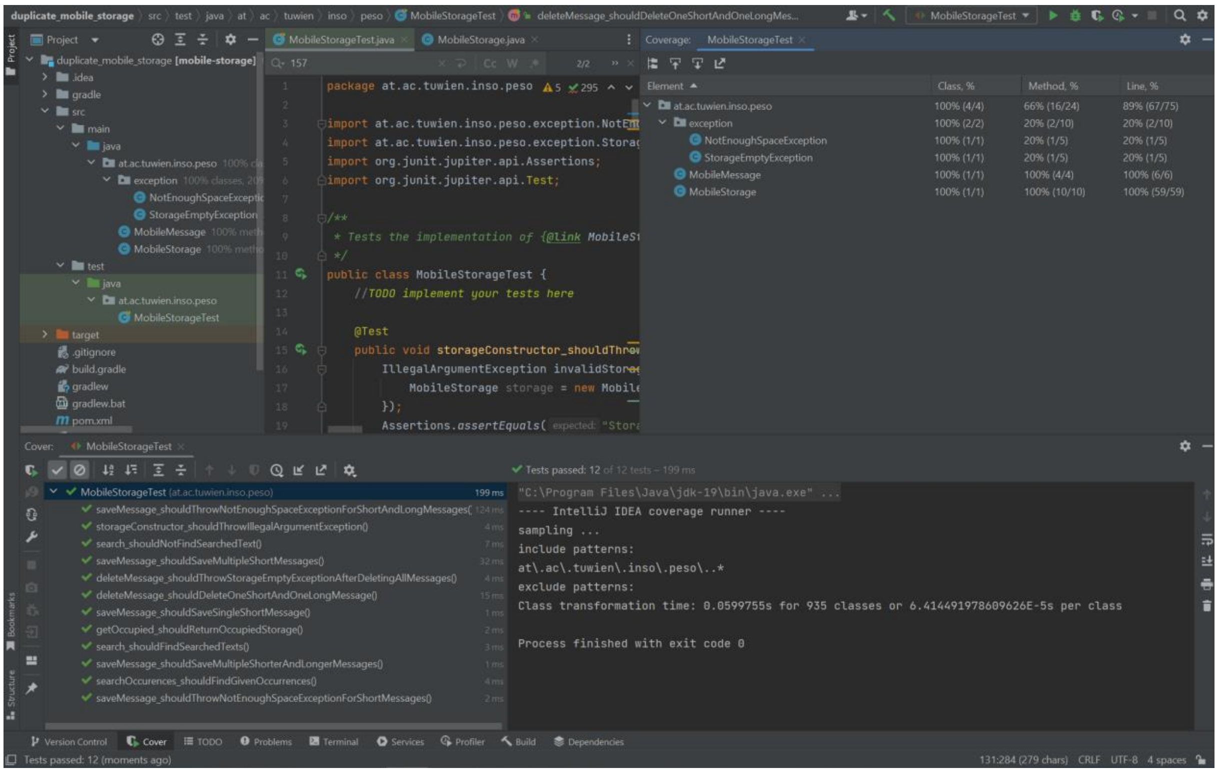
```
public void deleteMessage_shouldDeleteOneShortAndOneLongMessage(){//NOTE: When first running this test, I discovered a 'java.lang.ArrayIndexOutOfBoundsException: Index 6 out of bounds for length 6' at 'at at.ac.tuwien.inso.peso.MobileStorage.deleteMessage(MobileStorage.java:93)'
```

See Bug 1.3.

3. Coverage Report

The coverage report can be found in the folder "coverage-report".

Here are two images of running the coverage tests:



```
package at.ac.tuwien.inso.peso

import at.ac.tuwien.inso.peso.exception.NotEn
import at.ac.tuwien.inso.peso.exception.Storag
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

/**
 * Tests the implementation of {@link MobileSt
 */
public class MobileStorageTest {
    //TODO implement your tests here

    @Test
    public void storageConstructor_shouldThrow
        IllegalArgumentException invalidStor
            MobileStorage storage = new Mobile
        });
        Assertions.assertEquals( expected: "Stor
```

Coverage: MobileStorageTest

Element	Class, %	Method, %	Line, %
at.ac.tuwien.inso.peso	100% (4/4)	66% (16/24)	89% (67/75)
exception	100% (2/2)	20% (2/10)	20% (2/10)
NotEnoughSpaceException	100% (1/1)	20% (1/5)	20% (1/5)
StorageEmptyException	100% (1/1)	20% (1/5)	20% (1/5)
MobileMessage	100% (1/1)	100% (4/4)	100% (6/6)
MobileStorage	100% (1/1)	100% (10/10)	100% (59/59)

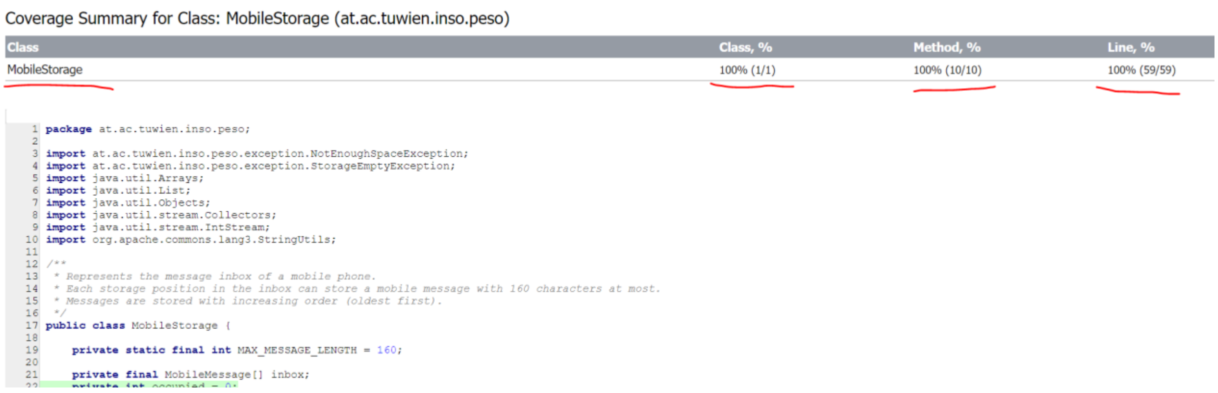
```
MobileStorageTest (at.ac.tuwien.inso.peso)                                           199 ms
  saveMessage_shouldThrowNotEnoughSpaceExceptionForShortAndLongMessages(           124 ms
  storageConstructor_shouldThrowIllegalArgumentException()                           4 ms
  search_shouldNotFindSearchedText()                                                 7 ms
  saveMessage_shouldSaveMultipleShortMessages()                                     32 ms
  deleteMessage_shouldThrowStorageEmptyExceptionAfterDeletingAllMessages()           4 ms
  deleteMessage_shouldDeleteOneShortAndOneLongMessage()                             15 ms
  saveMessage_shouldSaveSingleShortMessage()                                         1 ms
  getOccupied_shouldReturnOccupiedStorage()                                          2 ms
  search_shouldFindSearchedTexts()                                                   3 ms
  saveMessage_shouldSaveMultipleShorterAndLongerMessages()                           1 ms
  searchOccurences_shouldFindGivenOccurrences()                                      4 ms
  saveMessage_shouldThrowNotEnoughSpaceExceptionForShortMessages()                   2 ms
```

```
Tests passed: 12 of 12 tests – 199 ms
"C:\Program Files\Java\jdk-19\bin\java.exe" ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
at\.ac\.tuwien\.inso\.peso\..*
exclude patterns:
Class transformation time: 0.0599755s for 935 classes or 6.414491978609626E-5s per class

Process finished with exit code 0
```

Tests passed: 12 (moments ago)

duplicate_mobile_storage › src › test › java › at › ac › tuwien › inso › peso › MobileStorageTest › deleteMessage_shouldDeleteOneShortAndOneLongMes...
MobileStorageTest
Project
MobileStorageTest.java
MobileStorage.java
Version Control
Cover
TODO
Problems
Terminal
Services
Profiler
Build
Dependencies
131:284 (279 chars) CRLF UTF-8 4 spaces



Coverage Summary for Class: MobileStorage (at.ac.tuwien.inso.peso)

Class	Class, %	Method, %	Line, %
MobileStorage	100% (1/1)	100% (10/10)	100% (59/59)

```
package at.ac.tuwien.inso.peso;

import at.ac.tuwien.inso.peso.exception.NotEnoughSpaceException;
import at.ac.tuwien.inso.peso.exception.StorageEmptyException;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import org.apache.commons.lang3.StringUtils;

/**
 * Represents the message inbox of a mobile phone.
 * Each storage position in the inbox can store a mobile message with 160 characters at most.
 * Messages are stored with increasing order (oldest first).
 */
public class MobileStorage {

    private static final int MAX_MESSAGE_LENGTH = 160;

    private final MobileMessage[] inbox;
    private int occupied = 0;
```

3.1. Is the coverage sufficient? Why (not)?

Since the goal was to test MobileStorage and running the Unit Tests led to a coverage of 100% I assume the coverage is sufficient. However focussing on the coverage % can be misleading. It's important to test the right scenarios (especially corner cases) instead of just trying to run each line of code.

When looking at the Unit tests I always had the intention of testing edge cases and never to blindly achieve a high coverage. However since it is a very small application achieving 100% coverage seemed possible.

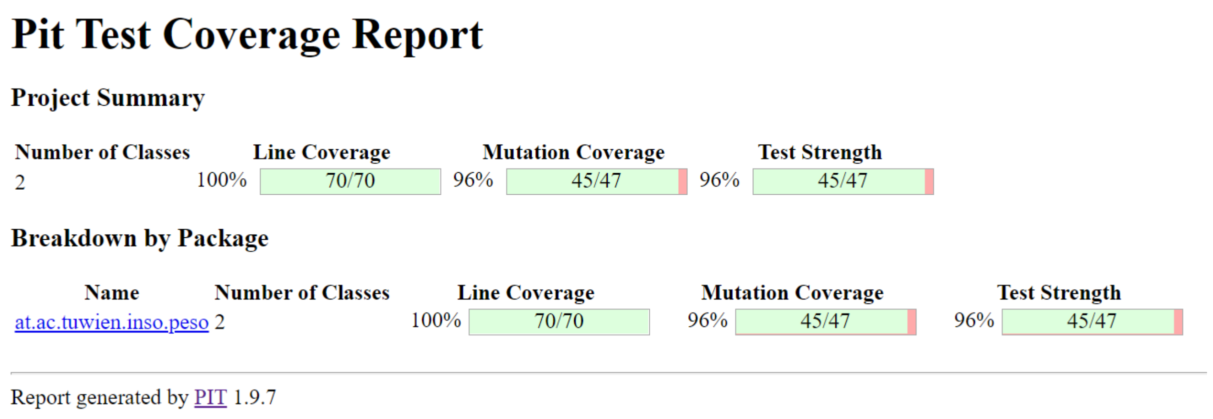
3.2. Which coverage value should be reached and why?

This honestly is more a philosophical question in my opinion. There is no percentage of coverage which should be reached. Most of the time a coverage of 100% would be either almost impossible (think of DTOs, getters & setters; why should you test those?), is just useless or might give a false feeling of 'everything works because we achieved 100% coverage'. Never assume that the software works perfectly just because of a high coverage! Having this in mind, test cases should reflect real scenarios but also test edge cases. Whatever coverage can be achieved in such a way will be sufficient. Edge cases will also be important for mutation testing.

4. Mutation Testing

4.1. Mutation Test Report

The PIT Mutation Test Report can be found in /pit-reports



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
2	100% 70/70	96% 45/47	96% 45/47

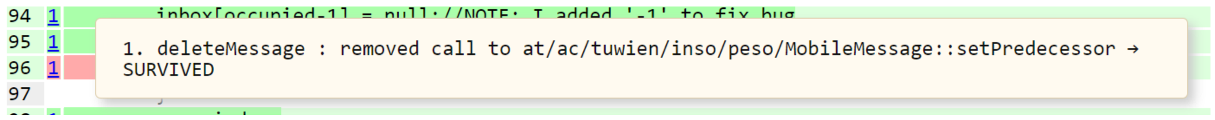
Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
at.ac.tuwien.inso.peso	2	100% 70/70	96% 45/47	96% 45/47

Report generated by PIT 1.9.7

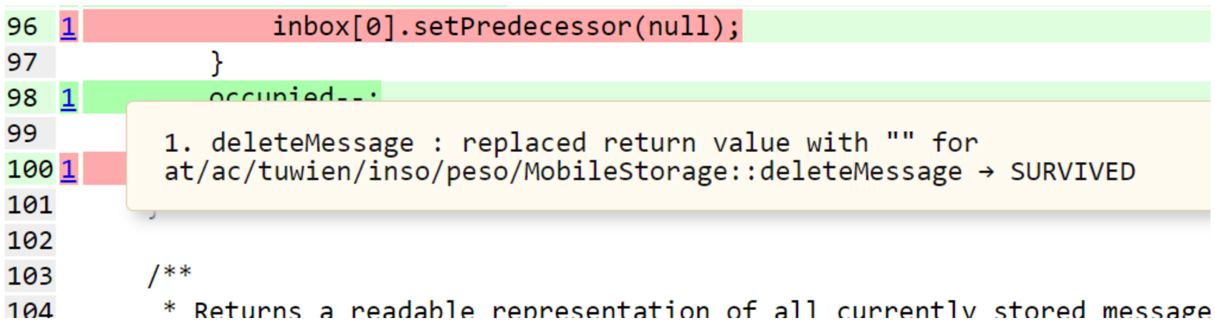
4.2. Reason why mutants survived

2 Mutants were able to survive:



```
        inbox[occupied-1] = null;//NOTE: I added '-1' to fix bug
1. deleteMessage : removed call to at/ac/tuwien/inso/peso/MobileMessage::setPredecessor →
SURVIVED
```

I assume that the setPredecessor Line is not always needed (only for longer Messages) and therefore the mutant has a higher chance to be undetected. I was not able to kill that mutant.



```
            inbox[0].setPredecessor(null);
        }
        occupied--;
1. deleteMessage : replaced return value with "" for
at/ac/tuwien/inso/peso/MobileStorage::deleteMessage → SURVIVED

    /**
     * Returns a readable representation of all currently stored message
```

Since the printMessages method will obviously not print deleted Messages, the deleted Messages do not occur in the String and therefore have the value “” (empty String). This might be the reason why this mutant was able to survive.

Since my Mutant Coverage is >95% and I only have assumptions for the reasons why 2 mutants survived I did not change any code after running the mutation tests for the first time.

Own opinion on mutation testing

Even if mutation testing does not mean your tests are perfect, I consider it a useful and feasible extension to unit testing. The time overhead is very low and it is a good indication of the quality of the unit tests. However I don’t think it is necessary to focus on a mutation coverage of 100% since this would be inefficient.