QL-6) Mutation Testing

> We will walk through a simple problem to see how mutation testing can be used to verify and improve the quality of tests

Git repo: stream2stream/palindrome

Understanding the codebase

1. Open the project called palindrome
   a. It's a Maven project and has the canonical structure
2. Open PalindromeTest.java, examine contents - there are only two tests

```java
package com.s2s.palindrome;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class PalindromeTest
{
    @Test
    public void when_imputLength_is_zero_then_accept()
    {
        Palindrome palindromeTester = new Palindrome();
        assertTrue(palindromeTester.isPalindrome(""));
    }

    @Test
    public void when_palindrome_then_accept() {
        Palindrome palindromeTester = new Palindrome();
        assertTrue(palindromeTester.isPalindrome("noon"));
    }
}
```

- Here is the CUT

```java
package com.s2s.palindrome;

public class Palindrome
{
    public boolean isPalindrome(String inputString)
    {
        boolean result = false;

        if (inputString.length() == 0)
        {
            result = true;
        }
        else
        {
            char firstChar = inputString.charAt(0);
            char lastChar = inputString.charAt(inputString.length() - 1);
            String mid = inputString.substring(1, inputString.length() - 1);
            result = (firstChar == lastChar) && isPalindrome(mid);
        }
```

```
        return result;
    }
}
```

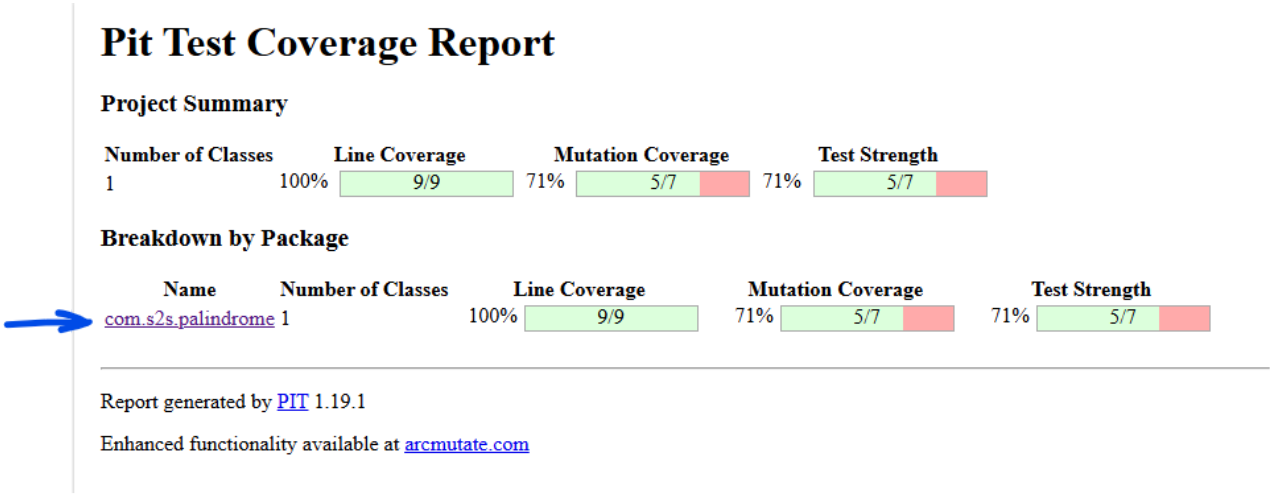
3. Run the tests (in debug mode if it helps) to get a better understanding of what the code actually does
4. Can you think of any other tests?

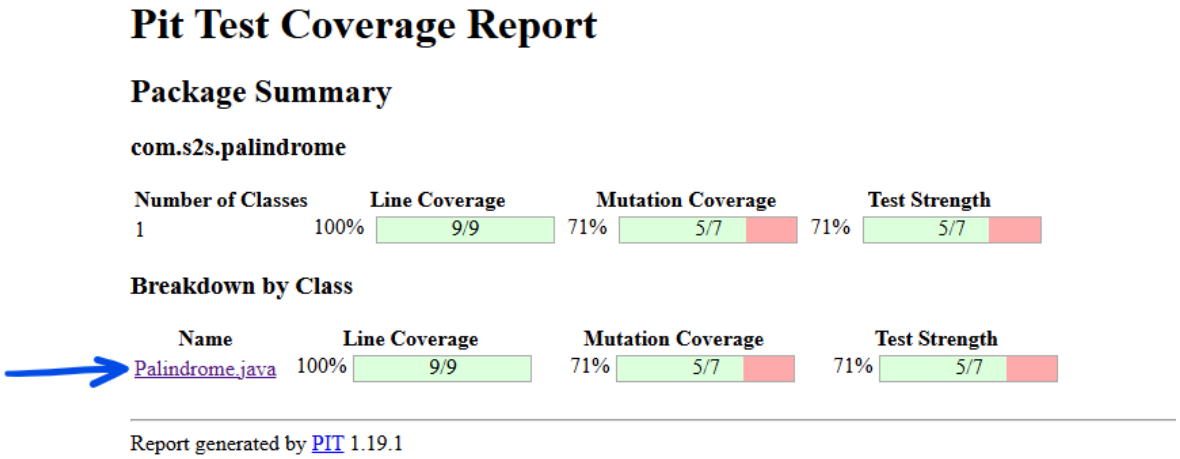
Using PITest

> Let's use the PITest to mutate the production and see whether our test coverage is good enough

1. If you haven't already done so, open a command shell
2. Navigate to the project folder
3. Run `git tag` to see all the tags
   - ```
     v1.0-first_tdd_test
     v1.1-palindrome_word_added
     v2.0-pitest_enabled
     ```
4. Run `git checkout v2.0-pitest_enabled`
5. Open pom.xml, and examine the contents. Notice some additional attributes, the PITest dependency and the build section
   - ```
     <build>
       <plugins>
         <plugin>
           <groupId>org.pitest</groupId>
           <artifactId>pitest-maven</artifactId>
           <version>1.19.1</version>
           <dependencies>
             <dependency>
               <groupId>org.pitest</groupId>
               <artifactId>pitest-junit5-plugin</artifactId>
               <version>1.1.1</version>
             </dependency>
           </dependencies>
           <configuration>
             <targetClasses>
               <param>com.s2s.palindrome.Palindrome</param>
             </targetClasses>
             <targetTests>
               <param>com.s2s.palindrome.PalindromeTest</param>
             </targetTests>
           </configuration>
         </plugin>
       </plugins>
     </build>
     ```
   - Lines 3-22: we use the pitest plugin for building the reports
   - Lines 6 and 11, it's important to tie up your version numbers with the version of JUnit. We are using jupiter 5.9.1. You may have to adjust your version numbers to suite your needs. More info can be found at GitHub - pitest/pitest-junit5-plugin: JUnit 5 test framework support for Pitest
   - Line 16: this should point to the package or class (CUT) that you want to analyse
   - Line 19: this should point to the package or test class holding the suite of tests. PITest will compare its analysis against your suite of tests to produce a report of where weaknesses can be found in your tests
6. From the command line run the commands
   - ```
     mvn clean package
     ```
     ```

```
mvn org.pitest:pitest-maven:mutationCoverage
```

7. If the build is successful, type `explorer .`
   a. Notice we have included the full stop in the command
8. When Explorer opens, navigate into the `target/pit-reports` folder
9. A mutation test report should open up
   - 

   - Select the package where the class whose mutation report lives, that you want to examine `com.s2s.palindrome`
10. Select the class
    - 

11. Here is the report

- **Palindrome.java**

```
1  package com.s2s.palindrome;
2
3  public class Palindrome
4  {
5      public boolean isPalindrome(String inputString)
6      {
7          boolean result = false;
8
9  1       if (inputString.length() == 0)
10         {
11             result = true;
12         }
13         else
14         {
15             char firstChar = inputString.charAt(0);
16 1           char lastChar = inputString.charAt(inputString.length() - 1);
17 1           String mid = inputString.substring(1, inputString.length() - 1);
18 2           result = (firstChar == lastChar) && isPalindrome(mid);
19         }
20 2       return result;
21     }
22 }
```

Mutations

```
9   1. negated conditional → KILLED
16  1. Replaced integer subtraction with addition → KILLED
17  1. Replaced integer subtraction with addition → KILLED
18  1. negated conditional → KILLED
    2. negated conditional → SURVIVED Covering tests
20  1. replaced boolean return with false for com/s2s/palindrome/Palindrome::isPalindrome → KILLED
    2. replaced boolean return with true for com/s2s/palindrome/Palindrome::isPalindrome → SURVIVED Covering tests
```

Understanding the results

The production code is shown with a number after the line number. This number is the mutation test number. If you hover your mouse over the mutation test number, you will see more info about the mutution (it's the same information that is listed under the **Mutations** section)

In the mutations section, clicking the link **"Covering tests"** will list the tests that were run to cover that line of code.

Any mutation marked as KILLED indicates that after the mutation, the test was rerun, and the test failed. This is good. It means your tests are working as expected.

Any mutation marked as SURVIVED indicates that after the mutation, the test was rerun, and the test passed. This is not 100% bad but an indication that you either don't have enough test coverage, the value returned is not a good indicator of success (as is the case here, returning a true does indicate that a Palindrome exists, but it can be easily fudged by always returning a true), or it's a genuine issue of brittle code that you need to look at.

What to do about the results

Poor test coverage - consider writing more tests so that the number of surviving mutations is reduced.

A poor return value that can not really be used as an indicator of success - consider rethinking what you return as success, and how you return a success indicator (an out parameter or a function return). See below for a new solution.

A genuine issue of brittle code - fix the code

Dealing with a return value that is a poor indicator of success

Returning a boolean is a clean and simple solution, but a single value such as this makes it harder to verify if there are any outlier issues. It would be better if `isPalindrome()` returned the number of palindromes it actually found. Test results can be more objective. As we can see from the mutation test results, true or false are very subjective here.

New tests

```java
package com.s2s.palindrome;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class PalindromeTest
{
    @Test
    public void when_imputLength_is_zero_then_accept()
    {
        Palindrome palindromeTester = new Palindrome();
        //assertTrue(palindromeTester.isPalindrome(""));
        assertEquals( 1, palindromeTester.isPalindrome(""));
    }

    @Test
    public void when_palindrome_of_two_letters_then_accept() {
        Palindrome palindromeTester = new Palindrome();
//        assertTrue(palindromeTester.isPalindrome("noon"));
        assertEquals( 1, palindromeTester.isPalindrome("oo"));
    }

    @Test
    public void when_palindrome_of_four_letters_then_accept() {
        Palindrome palindromeTester = new Palindrome();
//        assertTrue(palindromeTester.isPalindrome("noon"));
        assertEquals( 2, palindromeTester.isPalindrome("noon"));
    }

    @Test
    public void when_palindrome_of_six_letters_then_accept() {
        Palindrome palindromeTester = new Palindrome();
//        assertTrue(palindromeTester.isPalindrome("noon"));
        assertEquals( 3, palindromeTester.isPalindrome("snoons"));
    }
}
```

New CUT

```java
package com.s2s.palindrome;

public class Palindrome
{
    private class Result
    {
        String inputString;
        int count = 0;

        public Result(String inputString, int initCount)
        {
            this.inputString = inputString;
            count = initCount;
        }

        public Result(Result inputValues)
```

```
17          {
18              this.count = inputValues.count;
19              this.inputString = inputValues.inputString;
20          }
21      }
22
23      public int isPalindrome(String inputString)
24      {
25          int result = 0;
26
27          if (inputString.length() == 0)
28          {
29              result = 1;
30          }
31          else
32          {
33              char firstChar = inputString.charAt(0);
34              char lastChar = inputString.charAt(inputString.length() - 1);
35              String mid = inputString.substring(1, inputString.length() - 1);
36              if( firstChar == lastChar)
37              {
38                  Result tempResult = new Result(mid, 1);
39                  // We are passing an object in because it data members can be modified by all called methods
    and the results passed back up the stack
40                  isPalindrome(tempResult);
41                  result = tempResult.count;
42              }
43          }
44          return result;
45      }
46
47      private void isPalindrome(Result inputValues)
48      {
49          if( inputValues.inputString.length() > 0)
50          {
51              char firstChar = inputValues.inputString.charAt(0);
52              char lastChar = inputValues.inputString.charAt(inputValues.inputString.length() - 1);
53              String mid = inputValues.inputString.substring(1, inputValues.inputString.length() - 1);
54              if (firstChar == lastChar)
55              {
56                  inputValues.count++;
57                  inputValues.inputString = mid;
58                  isPalindrome(inputValues);
59              }
60          }
61      }
62  }
63
```

Lines 33-25 and 51-53 could be optimised into a private function

Rerun mutation tests. The results speak for themselves.

Palindrome.java

```
1  package com.s2s.palindrome;
2
3  public class Palindrome {
4      private class Result {
5          String inputString;
6          int count = 0;
7
8          public Result(String inputString, int initCount) {
9              this.inputString = inputString;
10             count = initCount;
11         }
12
13         public Result(Result inputValues) {
14             this.count = inputValues.count;
15             this.inputString = inputValues.inputString;
16         }
17     }
18
19     public int isPalindrome(String inputString) {
20         int result = 0;
21
22         if (inputString.length() == 0){
23             result = 1;
24         }
25         else{
26             char firstChar = inputString.charAt(0);
27 1           char lastChar = inputString.charAt(inputString.length() - 1);
28             String mid = inputString.substring(1, inputString.length() - 1);
29             if( firstChar == lastChar){
30                 Result tempResult = new Result(mid, 1);
31                 isPalindrome(tempResult);
32                 result = tempResult.count;
33             }
34 1       }
35 1       return result;
36 1   }
37
38     private void isPalindrome(Result inputValues){
39 1       if( inputValues.inputString.length() > 0){
40             char firstChar = inputValues.inputString.charAt(0);
41             char lastChar = inputValues.inputString.charAt(inputValues.inputString.length() - 1);
42             String mid = inputValues.inputString.substring(1, inputValues.inputString.length() - 1);
43 1           if (firstChar == lastChar){
44                 inputValues.count++;
45                 inputValues.inputString = mid;
46                 isPalindrome(inputValues);
47             }
48 2       }
49     }
50 }
```

Mutations

```
27 1. negated conditional → KILLED
34 1. Replaced integer subtraction with addition → KILLED
35 1. Replaced integer subtraction with addition → KILLED
36 1. negated conditional → KILLED
39 1. removed call to com/s2s/palindrome/Palindrome::isPalindrome → KILLED
43 1. replaced int return with 0 for com/s2s/palindrome/Palindrome::isPalindrome → KILLED
48 1. changed conditional boundary → KILLED
   2. negated conditional → KILLED
51 1. Replaced integer subtraction with addition → KILLED
52 1. Replaced integer subtraction with addition → KILLED
53 1. negated conditional → KILLED
55 1. Replaced integer addition with subtraction → KILLED
57 1. removed call to com/s2s/palindrome/Palindrome::isPalindrome → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- com.s2s.palindrome.PalindromeTest.[engine:junit-jupiter]/[class:com.s2s.palindrome.PalindromeTest]/[method:when_palindrome_of_four_letters_then_accept()] (0 ms)
- com.s2s.palindrome.PalindromeTest.[engine:junit-jupiter]/[class:com.s2s.palindrome.PalindromeTest]/[method:when_palindrome_of_six_letters_then_accept()] (73 ms)
- com.s2s.palindrome.PalindromeTest.[engine:junit-jupiter]/[class:com.s2s.palindrome.PalindromeTest]/[method:when_palindrome_of_two_letters_then_accept()] (1 ms)
- com.s2s.palindrome.PalindromeTest.[engine:junit-jupiter]/[class:com.s2s.palindrome.PalindromeTest]/[method:when_imputLength_is_zero_then_accept()] (0 ms)

Git codebase

```
1 git checkout v2.1-return_number_of_palindromes
```

Great result, all mutations are killed and we have better test coverage - 100%