

Test, Verification and Validation of Software

Ana Cristina Ramada Paiva

Mutation Testing Exercises

Hugo Almeida - up202103394

Orlando Macedo - up202103400



FEUP
Universidade do Porto

12-11-2021

Contents

1 Tools 2

2 Context 2

3 Exercises 4

3.1 Exercise 1 4

3.2 Exercise 2 4

3.3 Exercise 3 5

3.4 Exercise 4 5

3.5 Conclusion 5

1 Tools

For this practical class it will be used the **Java** programming language with several tools. It is recommended to use the following:

- **IntelliJ** as IDE
- **Maven** as Compilation Automation Tool
- **JUnit** as Unit Testing Framework
- **PITest** as tool for Mutation Testing in **Java**

2 Context

In this class, we are going to test a simple implementation of a calendar. The main function of it, is to calculate the day of the week that a specific date occurred on. The calculation was based on the [Disparate variation of the Gauss's algorithm](#) represented in the program with the method **getDayOfWeek** that has the following calculation:

$$w = \left(d + \lfloor 2.6m - 0.2 \rfloor + y + \left\lfloor \frac{y}{4} \right\rfloor + \left\lfloor \frac{c}{4} \right\rfloor - 2c \right) \bmod 7,$$

Figure 1: Disparate variation of the *Gauss's algorithm*

Where:

- **Y** is the year minus 1 for January or February, and the year for any other month
- **y** is the last 2 digits of Y
- **c** is the first 2 digits of Y
- **d** is the day of the month (1 to 31)
- **m** is the shifted month (March=3,...January = 13, February=14)
- **w** is the day of week (1=Sunday,...0=Saturday)

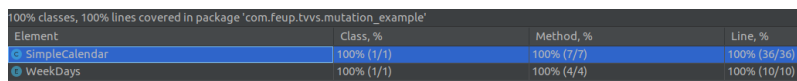
After the 4th of October of 1582 the human race has been using the *Gregorian calendar* however, before this calendar, the *Julian calendar* was the default. In this implementation of the calendar, it is assumed that every date on the 4th October of 1582 or before is using the *Julian calendar*, causing the program to apply a conversion to the *Gregorian calendar*. The procedure is represented in the program in the method **parseJulianToGregorian** using the following procedure:

1. Add 10 days to the Julian date.
2. Subtract one day for each century not divisible by 400 between the Julian date and October 15, 1582. If the year in the date in question is a century, it should only count to this rule if the month is March or greater.

We are going to test the class **SimpleCalendar** that has the following available public methods:

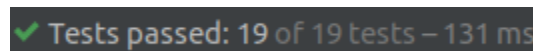
- **getDayOfWeek(LocalDateTime date)** - Does the main calculations using the complementary methods when needed
- **getGregorianDate(LocalDateTime date)** - Returns a Gregorian date, applying the conversion to Julian if the date is before the date of change from Julian to Gregorian Calendar
- **dateIsGregorian(LocalDateTime date)** - Checks if the date is Gregorian (after the date of change from Julian to Gregorian Calendar) or not
- **parseJulianToGregorian(LocalDateTime date)** - Converts a Julian date to Gregorian

There are tests already provided that can be found in the class **SimpleCalendarTest**. You can see that they are all passing by right clicking on the test class in the IntelliJ, going to "More Run/Debug" and "Run 'SimpleCalendarTest' with Coverage":



100% classes, 100% lines covered in package 'com.feup.tvvs.mutation_example'			
Element	Class, %	Method, %	Line, %
SimpleCalendar	100% (1/1)	100% (7/7)	100% (36/36)
WeekDays	100% (1/1)	100% (4/4)	100% (10/10)

Figure 2: Test Coverage



✓ Tests passed: 19 of 19 tests – 131 ms

Figure 3: Tests ran

Although they are all passing, giving **100% test coverage**, this coverage doesn't prove the quality of our test suite. But we can add another layer of tests, a layer that tests the tests themselves. This is done by mutating certain statements in the source code and checking if the tests are able to find errors, in sum, doing **Mutation Testing**.

3 Exercises

To solve the proposed exercises it is needed to run **PITest**. This project is using a version of this tool made for **Maven** that runs with the 'test' phase, which is part of Maven default lifecycle. To run it, just execute ***mvn test*** on the root of the source code. After this, a new report should be generated in the directory */target/pit-reports/YYYYMMDDHHMMI*. To see the report, just open the *index.html* inside it.

In order to help you implementing tests in the following exercises, this [Day of Week Calculator](#) can be helpful.

3.1 Exercise 1

To start, you need to kill the two mutants on the methods **getDayOfWeek** and **parseMonth** that are still alive. These were created using [Conditionals Boundary Mutators](#) that can be easily killed by looking at the conditions on the methods. **Be aware that it is possible that your resolution of a line can kill two mutants at once.**

1. a) Kill the first mutant on the method **getDayOfWeek** by adding code to the test **testToKillMutantOne** on the test class **SimpleCalendarTest**.

- Pay attention to the changes made by the **Conditionals Boundary Mutator**. On the image 4 are the multiple mutations possible by this mutator.

Original conditional	Mutated conditional
<	<=
<=	<
>	>=
>=	>

Figure 4: Conditionals Boundary Mutator

1. b) The mutant on the method **parseMonth** is similar to the first one, so killing him should be easy.

1. c) The mutant on the method **parseJulianToGregorian** is similar to the previous two, so killing him should also be easy.

- When finished, pay special attention to the **Mutation Coverage** generated when running **PITest**, in the HTML report. At this point, it should be 100%, which means this test suite is good, or is it not...?

3.2 Exercise 2

The **PITest** tool has several Mutation Operators [available](#) to use beyond the default ones we have been using. In this exercise we are going to use new operators to create Mutants. These new operators belong to the "STRONGER" group of **PITest** Mutation Operators and to use them, you need to go to your *pom.xml* and unncomment the line 74 to make them available, specifying this group of operators on the **PITest** configuration.

2. a) In this exercise, you will kill the two mutants that were created using the Mutation Operator *REMOVE_CONDITIONALS_EQ_IF* and survived. These mutants are found on the method **datelsGregorian** and should be killed by writing code on the tests **testToKillMutantFour** and **testToKillMutantFive** on the test class **SimpleCalendarTest**.

3.3 Exercise 3

To continue this set of exercises, we are going to edit the **pom.xml** file again, in order to change the Mutation Operator group to the one that uses all the operators, called "ALL".

Since **PITest** is going to use more Mutators, the execution of the analysis is going to take a little bit more. To make things faster, we suggest that you uncomment the line 77 of the **pom.xml** file, making **PITest** run with history. This is a convenient way of using temporary history files to speed up local analysis down the line.

3. a) A lot of new mutants were created with this group of operators. This time, you should kill a Mutant at line 127 of the **SimpleCalendar** class. The Mutant was created using a Mutator of type *Constant Replacement Mutator (CRCR3)*.

3.4 Exercise 4

a) Can you find one Equivalent Mutant generated by these new operators? Why is it Equivalent?

- **Hint:** In java, $x \% y$ is always equals to $x \% -y$

b) If you want to make the build fail when the tests doesn't meet your expectations, **PITest** has got your back. You can set the **Mutation Threshold** on the **pom.xml** file in order to configure the tool to fail when the threshold is not met. Check your **pom.xml** and activate the threshold to test the results.

- **Note:** You can also set a **Coverage Threshold** in order meet your expectations in regards of line coverage of the overall tests.

Optional: We know that you probably won't do the following, but we invite you to kill the rest of the mutants and save humanity.

3.5 Conclusion

Some questions for you to think about...

- Do you think Mutation Testing brings benefits to the software development process?
- Which group of Mutation Operators do you think is the best compromise between performance and discovering the greatest number of vulnerabilities in testing?