

SOFTWARE TESTING(CS731)

PROJECT REPORT

Submitted by:

- 1) Arbaz Malik (MT2022019)**
- 2) Ashutosh Gajare(MT2022025)**

Repository Link:

<https://github.com/Arbazmalik/testing-project.git>

Testing Strategy:

We have used **mutation testing** in order to test our code base.

Mutation testing is a software testing technique designed to assess the effectiveness of a test suite by introducing deliberate, small changes (mutations) to the source code and evaluating whether the existing tests can detect these alterations. The primary goal of mutation testing is to identify weaknesses in the test suite, ensuring that it is capable of catching real faults in the code. In this process, mutations simulate potential bugs, such as introducing logical errors, changing operators, or modifying conditional statements. If the test suite successfully identifies and flags these mutations, it indicates robust test coverage. However, if mutations go undetected, it suggests areas where the test suite may be insufficient. Mutation testing serves as a valuable quality assurance measure, helping developers enhance the reliability and effectiveness of their testing strategies, ultimately leading to more robust and fault-tolerant software.

The steps that we undertook to perform mutation testing were:

1. By producing several mutant versions of the source code, we added a few errors. Then, using one of the compatible mutation operators in the source code, each mutant is created.
2. Since we have now introduced mutants, we need to provide some test cases to destroy them. Additionally, the original program's test cases should succeed, and the mutation should be effectively eliminated (i.e. propagating the error which results in incorrect output).

3. If the output of the original programme and the altered programme vary in a specific test scenario, the mutant is said to have been forcefully killed. Otherwise, it supposedly made it through.

Codebase:

We have developed a Java code which constitutes a comprehensive encryption and decryption application under the Spring Boot framework, encapsulated within the `MutationProjectApplication` class. Our program offers a selection of classic encryption and decryption algorithms, including the Caesar Cipher, Vigenère Cipher, Vernam Cipher, Rail Fence Cipher, and Playfair Cipher. Each encryption method is meticulously implemented as an individual function within the main class, ensuring a modular and well-organized codebase.

User interaction is facilitated through a command-line interface, allowing users to choose a specific cipher method and subsequently input the required parameters for encryption or decryption. For instance, the Caesar Cipher method (`Caesar()`) prompts users to input a string and a key, enabling them to choose between encryption and decryption operations. A similar structure is maintained for the other cipher methods.

The main method serves as the orchestrator of the program, guiding the execution based on the user's cipher selection. It prompts the user to choose from the available cipher methods, runs the selected method, and presents the encrypted or decrypted output.

Testing Tool - PIT

PIT, or Pitest, is a widely used mutation testing tool in the realm of Java development. Short for "Pit Mutation Testing," this open-source tool is designed to assess the effectiveness of a test suite by introducing artificial faults (mutations) into the source code and determining whether the existing tests can detect these mutations. PIT operates at the bytecode level, making it compatible with various testing frameworks, including JUnit and TestNG. Its mutation engine is highly configurable, allowing users to define specific mutation operators and customize the mutation testing process according to their project's requirements. PIT provides detailed mutation reports, highlighting which mutations were killed by the test suite and which remain undetected.

About our test cases

Our test code contains test cases cover various classic ciphers, including Caesar Cipher, Vernam Cipher, Vigenère Cipher, Playfair Cipher, and Rail Fence Cipher.

For the Caesar Cipher, test cases evaluate both encryption and decryption operations with different input strings and keys. Similarly, the Vernam Cipher test cases assess the correctness of encryption and decryption using predetermined input strings and keys. Vigenère Cipher test cases cover scenarios where the algorithm encrypts and decrypts input strings with specific keys. Playfair Cipher test cases examine the encryption and decryption functionality with given keywords and input strings. Lastly, Rail Fence Cipher test cases assess the accuracy of both encryption and decryption operations with different key values.

The test suite also includes additional utility methods such as LowerToUpper, which converts lowercase characters to uppercase, and KeyGen, which generates keys based on given keywords. These utility methods are validated through test cases to ensure their correctness.

The comprehensive test suite provides a robust validation of the implemented encryption and utility methods, helping to ensure the reliability and accuracy of the cryptographic functionalities in the utils class.

Mutation Operators

We have configured our pom.xml to ensure that PIT applies following mutation operators on our codebase.

1)NEGATE_CONDITIONALS mutator:

The negate conditionals mutator will mutate all conditionals found i.e (== to != and vice versa) , (>= to < and vice versa) and (<= to > and vice versa).

2)MATH mutator:

The math mutator replaces binary arithmetic operations for either integer or floating-point arithmetic with another operation.

3)CONDITIONALS_BOUNDARY mutator:

The conditionals boundary mutator replaces the relational operators <, <=, >, >= with their boundary counterpart.

4)NON_VOID_METHOD_CALLS mutator

The non void method call mutator removes method calls to non void methods. Their return value is replaced by the Java Default Value for that specific type.

5)EXPERIMENTAL_ARGUMENT_PROPAGATION mutator

Experimental mutator that replaces method call with one of its parameters of matching type.

6)EMPTY_RETURNS mutator

Replaces return values with an 'empty' value for that type as follows

```
java.lang.String -> ""  
java.util.Optional -> Optional.empty()  
java.util.List -> Collections.emptyList()  
java.util.Collection -> Collections.emptyList()  
java.util.Set -> Collections.emptySet()  
java.lang.Integer -> 0  
java.lang.Short -> 0  
java.lang.Long -> 0  
java.lang.Character -> 0  
java.lang.Float -> 0  
java.lang.Double -> 0
```

Pitest will filter out equivalent mutations to methods that are already hard coded to return the empty value.

Note that the first three mutators have been applied for **unit testing** while the bottom three mutators have been applied for **integration testing**.

Testing results

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	95% <div><div></div></div> 241/254	90% <div><div></div></div> 269/298	92% <div><div></div></div> 269/293

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
com.example.mutation.mutation.project 1	1	95% <div><div></div></div> 241/254	90% <div><div></div></div> 269/298	92% <div><div></div></div> 269/293

Report generated by [PIT](#) 1.15.3

Enhanced functionality available at [arcmutate.com](#)

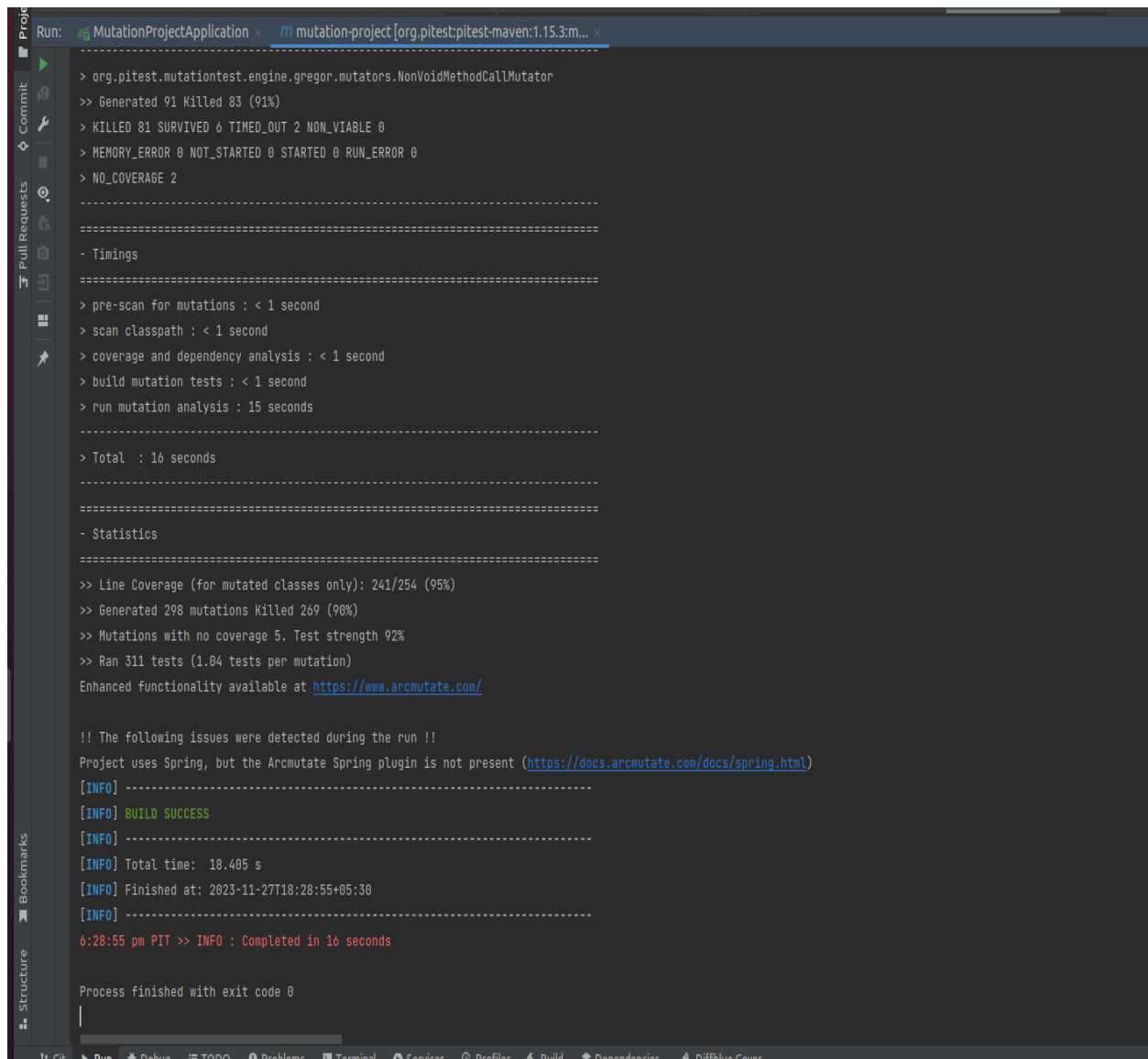
Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- EXPERIMENTAL_ARGUMENT_PROPAGATION
- INCREMENTS
- MATH
- NEGATE_CONDITIONALS
- NON_VOID_METHOD_CALLS
- VOID_METHOD_CALLS

Tests examined

- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:playfairDecryptData()] (10 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:playfairEncryptData()] (4 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:vinereEncryptData()] (0 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:caesarEncryptData()] (0 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:railfenceEncryptData()] (0 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:vinereDecryptData()] (0 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:railfenceDecryptData()] (1 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:caesarDecryptData()] (30 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:vernamEncryptData()] (0 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:vernamDecryptData()] (0 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:lowerToUpper()] (0 ms)
- com.example.mutation.mutation.project.utilsTest.[engine:junit-jupiter]/[class:com.example.mutation.mutation.project.utilsTest]/[method:keyGen()] (4 ms)

Report generated by [PIT](#) 1.15.3



```
Run: MutationProjectApplication x 1/1 mutation-project [org.pitest:pitest-maven:1.15.3.m... x]
> org.pitest.mutationtest.engine.gregor.mutators.NonVoidMethodCallMutator
>> Generated 91 Killed 83 (91%)
> KILLED 81 SURVIVED 6 TIMED_OUT 2 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 2

-----
- Timings
-----
> pre-scan for mutations : < 1 second
> scan classpath : < 1 second
> coverage and dependency analysis : < 1 second
> build mutation tests : < 1 second
> run mutation analysis : 15 seconds
-----
> Total : 16 seconds
-----
- Statistics
-----
>> Line Coverage (for mutated classes only): 241/254 (95%)
>> Generated 298 mutations Killed 269 (90%)
>> Mutations with no coverage 5. Test strength 92%
>> Ran 311 tests (1.04 tests per mutation)
Enhanced functionality available at https://www.arcmutate.com/

!! The following issues were detected during the run !!
Project uses Spring, but the Arcmutate Spring plugin is not present (https://docs.arcmutate.com/docs/spring.html)
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 18.405 s
[INFO] Finished at: 2023-11-27T18:28:55+05:30
[INFO] -----
6:28:55 pm PIT >> INFO : Completed in 16 seconds

Process finished with exit code 0
```

Contributions

For the purpose of this project, we developed a codebase from scratch, studied PIT documentation and wrote test cases accordingly to test various nuances of this codebase. The contribution of both of us was equal in all the aspects mentioned above.