Repository Link: https://github.com/Shahriar-0/Software-Testing-Course-Projects-S2025
Latest Commit Hash: 0ab94b465433eb34893f96976d6c1554a721e414

Mutation Testing

1. Report

```
================================================================================
- Mutators
================================================================================
>
org.pitest.mutationtest.engine.gregor.mutators.returns.PrimitiveReturnsM
utator
>> Generated 4 Killed 4 (100%)
> KILLED 4 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
>
org.pitest.mutationtest.engine.gregor.mutators.ConditionalsBoundaryMutat
or
>> Generated 2 Killed 1 (50%)
> KILLED 1 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.IncrementsMutator
>> Generated 1 Killed 1 (100%)
> KILLED 1 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
>
org.pitest.mutationtest.engine.gregor.mutators.returns.BooleanTrueReturn
ValsMutator
>> Generated 2 Killed 2 (100%)
> KILLED 2 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.MathMutator
>> Generated 7 Killed 7 (100%)
> KILLED 7 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
```

```
--------
>
org.pitest.mutationtest.engine.gregor.mutators.NegateConditionalsMutator
>> Generated 13 Killed 13 (100%)
> KILLED 13 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
------------------------------------------------------------------------
--------
========================================================================
- 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 : 2 seconds
------------------------------------------------------------------------
> Total  : 3 seconds
------------------------------------------------------------------------




========================================================================
- Statistics
========================================================================
>> Line Coverage (for mutated classes only): 46/50 (92%)
>> Generated 29 mutations Killed 28 (97%)
>> Mutations with no coverage 0. Test strength 97%
>> Ran 84 tests (2.9 tests per mutation)
Enhanced functionality available at https://www.arcmutate.com/
```

Transaction.java

```
1  package domain;
2
3  import lombok.Getter;
4  import lombok.Setter;
5
6  @Getter
7  @Setter
8  public class Transaction {
9      int transactionId;
10     int accountId;
11     int amount;
12     boolean isDebit;
13
14     @Override
15     public boolean equals(Object obj) {
16 1       if (obj instanceof Transaction transaction) {
17 2           return transactionId == transaction.transactionId;
18        }
19 1       return false;
20     }
21 }
```

Mutations

```
16 1. negated conditional → KILLED
17 1. replaced boolean return with true for domain/Transaction::equals → KILLED
   2. negated conditional → KILLED
19 1. replaced boolean return with true for domain/Transaction::equals → 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

- domain.TransactionTest.[engine:junit-jupiter]/[class:domain.TransactionTest]/[method:testEquals_DifferentId()] (0 ms)
- domain.TransactionTest.[engine:junit-jupiter]/[class:domain.TransactionTest]/[method:testEquals_SameId()] (1 ms)
- domain.TransactionTest.[engine:junit-jupiter]/[class:domain.TransactionTest]/[method:testEquals_DifferentClass()] (2 ms)
- domain.TransactionEngineTest.[engine:junit-jupiter]/[class:domain.TransactionEngineTest]/[method:testAddTransactionAndDetectFraud_DuplicateTransaction()] (0 ms)
- domain.TransactionEngineTest.[engine:junit-jupiter]/[class:domain.TransactionEngineTest]/[method:testGetTransactionPatternAboveThreshold_WithoutPattern()] (0 ms)
- domain.TransactionEngineTest.[engine:junit-jupiter]/[class:domain.TransactionEngineTest]/[method:testGetTransactionPatternAboveThreshold_WithPattern()] (0 ms)
- domain.TransactionEngineTest.[engine:junit-jupiter]/[class:domain.TransactionEngineTest]/[method:testGetAverageTransactionAmountByAccount()] (0 ms)
- domain.TransactionEngineTest.[engine:junit-jupiter]/[class:domain.TransactionEngineTest]/[method:testGetTransactionPatternAboveThreshold_PatternMultiple()] (0 ms)
- domain.TransactionEngineTest.[engine:junit-jupiter]/[class:domain.TransactionEngineTest]/[method:testDetectFraudulentTransaction_ExcessiveDebit()] (0 ms)
- domain.TransactionEngineTest.[engine:junit-jupiter]/[class:domain.TransactionEngineTest]/[method:testDetectFraudulentTransaction_WithoutExcessiveDebitButIsDebit()] (20 ms)

The four lines appear to be uncovered, but since Lombok is being used, they are indeed covered, even though this is not recognized.

Pit Test Coverage Report

Package Summary

domain

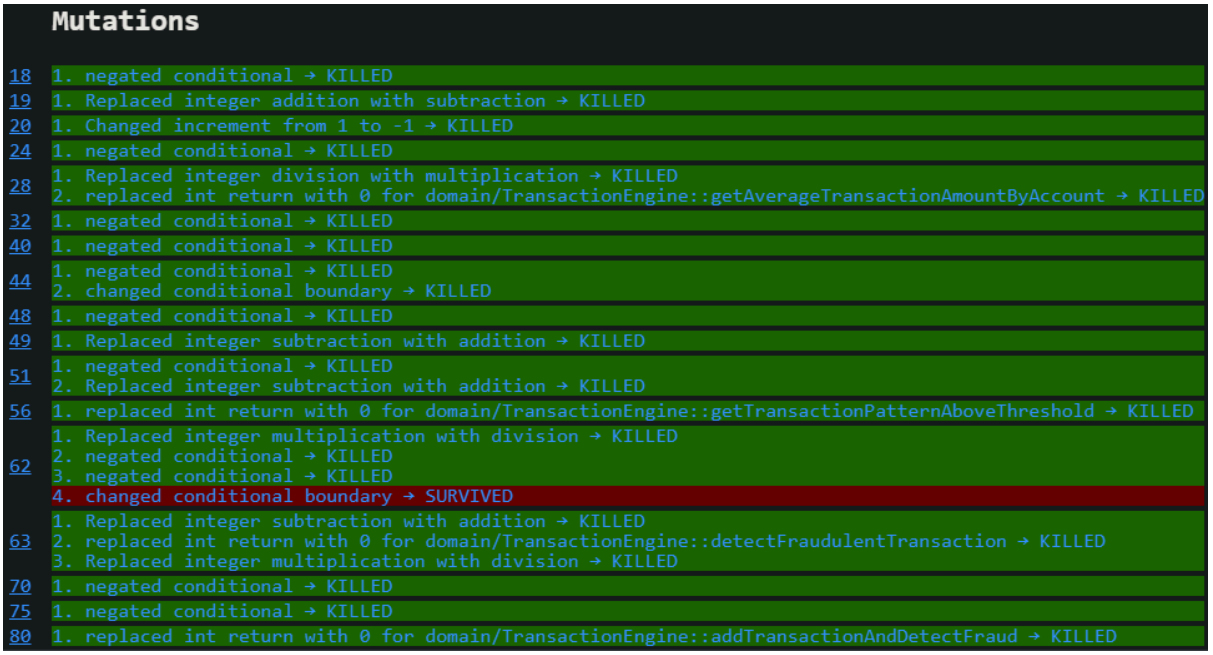
Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	92%	46/50	97%	28/29	97%	28/29

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Transaction.java	56%	5/9	100%	4/4	100%	4/4
TransactionEngine.java	100%	41/41	96%	24/25	96%	24/25

TransactionEngine.java

```
1  package domain;
2
3  import java.util.ArrayList;
4
5  public class TransactionEngine {
6      ArrayList<Transaction> transactionHistory;
7      int THRESHOLD = 1000;
8
9      public TransactionEngine() {
10         transactionHistory = new ArrayList<>();
11     }
12
13     int getAverageTransactionAmountByAccount(int accountId) {
14         var totalAmount = 0;
15         var count = 0;
16
17         for (Transaction txn : transactionHistory) {
18 1           if (txn.accountId == accountId) {
19 1               totalAmount += txn.amount;
20 1               count++;
21             }
22         }
23
24 1       if (count == 0) {
25             return 0;
26         }
27
28 2       return totalAmount / count;
29     }
30
31     int getTransactionPatternAboveThreshold(int threshold) {
32 1       if (transactionHistory.isEmpty()) {
33             return 0;
34         }
35
36         var diff = 0;
37         var previous = transactionHistory.getFirst();
38
39         for (Transaction txn : transactionHistory) {
40 1           if (txn.transactionId == previous.transactionId) {
41               continue;
42           }
43
44 2           if (txn.amount <= threshold) {
45               continue;
46           }
47
48 1           if (diff == 0) {
49 1               diff = txn.amount - previous.amount;
50               previous = txn;
51 2           } else if (diff != txn.amount - previous.amount) {
52               return 0;
53           }
54         }
55
56 1       return diff;
57     }
58
59     int detectFraudulentTransaction(Transaction txn) {
60         var averageAmount = getAverageTransactionAmountByAccount(txn.accountId);
61
62 4       if (txn.isDebit && txn.amount > 2 * averageAmount) {
63 3           return txn.amount - 2 * averageAmount;  // Excessive debit, marked as suspicious
64         }
65
66         return 0;
67     }
68
69     public int addTransactionAndDetectFraud(Transaction txn) {
70 1       if (transactionHistory.contains(txn)) {
71             return 0;
72         }
73
74         var fraudScore = detectFraudulentTransaction(txn);
75 1       if (fraudScore == 0) {
76             fraudScore = getTransactionPatternAboveThreshold(THRESHOLD);
77         }
78
79         transactionHistory.add(txn);
80 1       return fraudScore;
81     }
82 }
```



The only failed mutation is for the following line

```
        if (txn.isDebit && txn.amount > 2 * averageAmount) {
             return txn.amount - 2 * averageAmount;  // Excessive debit,
marked as suspicious
         }

       return 0;
```

Since changing the > to >= won't change the outcome, it's impossible to strongly kill this mutation (i.e. they are equivalent mutations).

2. Refactoring and Mutation Coverage Relationship

Significant challenges arise when refactoring test code, as there is no inherent safety net to ensure that the tests' behavior remains unchanged post-refactoring. When refactoring the main code, comprehensive unit tests can help confirm that the external behavior remains consistent; if any changes occur, the tests will likely fail. In contrast, when refactoring the tests themselves, developers lack this assurance. This is where mutation testing becomes invaluable.

After refactoring the test code, it is essential to evaluate mutation coverage to verify that the behavior of the tests has not changed. This process provides confidence that the refactoring was successful without introducing errors. By analyzing how well the tests perform against mutated code versions, developers can ascertain whether any critical behaviors have been inadvertently altered during refactoring.

Citations:

- https://www.mdpi.com/2073-431X/12/11/230
- https://www.sngular.com/insights/330/mutation-testing-testing-your-tests
- https://arxiv.org/abs/1506.07330
- https://nexocode.com/blog/posts/mutation-testing/