ارشیا عطایی نایینی ۸۱۰۱۰۰۲۵۲
امیرپارسا موبد ۸۱۰۱۰۰۲۷۱

آزمون نرم افزار
تمرین پنجم

Github Reposiroty: https://github.com/ParsaMOBB/SWT-Course-Projects

Last Commit: 2cff70f5b1b36f4a582aae1d000f30c3487c1341

سوال اول:

در این بخش ابتدا از تست‌های نوشته شده در فاز 3 استفاده شده و مراحل گفته شده را انجام میدهیم.

```
- Mutators
================================================================================
> org.pitest.mutationtest.engine.gregor.mutators.returns.PrimitiveReturnsMutator
>> 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.ConditionalsBoundaryMutator
>> 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.BooleanTrueReturnValsMutator
 >> 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
```

```
> pre-scan for mutations : < 1 second
> scan classpath : < 1 second
> coverage and dependency analysis : 3 seconds
> build mutation tests : < 1 second
> run mutation analysis : 6 seconds
----------------------------------------------------------------------------
> Total  : 10 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 59 tests (2.03 tests per mutation)
```

همان طور که قابل مشاهده است تعداد mutant ها 29 تاست که 4 تای آن برای Transaction.java و 25 تای دیگر برای کلس TransactionEngine.java است.

تعداد mutant های کشته شده: 28

تعداد mutant های زنده مانده: 1

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

```
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          }
```

Mutations

18	1. negated conditional → KILLED
19	1. Replaced integer addition with subtraction → KILLED
20	1. Changed increment from 1 to -1 → KILLED
24	1. negated conditional → KILLED
28	1. Replaced integer division with multiplication → KILLED 2. replaced int return with 0 for domain/TransactionEngine::getAverageTransactionAmountByAccount → KILLED
32	1. negated conditional → KILLED
40	1. negated conditional → KILLED
44	1. negated conditional → KILLED 2. changed conditional boundary → KILLED
48	1. negated conditional → KILLED
49	1. Replaced integer subtraction with addition → KILLED
51	1. negated conditional → KILLED 2. Replaced integer subtraction with addition → KILLED
56	1. replaced int return with 0 for domain/TransactionEngine::getTransactionPatternAboveThreshold → KILLED
62	1. Replaced integer multiplication with division → KILLED 2. negated conditional → KILLED 3. negated conditional → KILLED 4. changed conditional boundary → SURVIVED
63	1. Replaced integer subtraction with addition → KILLED 2. replaced int return with 0 for domain/TransactionEngine::detectFraudulentTransaction → KILLED 3. Replaced integer multiplication with division → KILLED
70	1. negated conditional → KILLED
75	1. negated conditional → KILLED
80	1. replaced int return with 0 for domain/TransactionEngine::addTransactionAndDetectFraud → KILLED

چون با تغییر < و => تغییری در نتیجه بدست آمده حاصل نمیشود و این دو یک mutation هستند، این mutation زنده میماند اما سایرین کشته میشوند.

تاثیر Mutation Coverage بالا در میزان خطر Refactoring

Mutation Coverage یا پوشش تغییرات، ابزاری در تست نرم‌افزار است که به بررسی دقت و پوشش تست‌ها پرداخته و با اعمال تغییرات کوچک و کنترل شده (موتیشن‌ها) به کد، میزان واکنش تست‌ها به این تغییرات را می‌سنجد. اگر تست‌ها به درستی نوشته شده باشند، باید قادر به شناسایی این تغییرات و ناکام گذاشتن آن‌ها باشند. این فرآیند به بهبود کیفیت تست‌ها کمک شایانی می‌کند و می‌تواند به شناسایی نقاط ضعف تست‌ها کمک نماید.

تاثیر بر Refactoring:

1. **کاهش ریسک:** داشتن پوشش تغییرات بالا قبل از انجام refactoring می‌تواند ریسک‌های مرتبط با تغییر کد را کاهش دهد. زیرا وقتی تست‌ها توانایی شناسایی خطاهای کوچک در کد را دارند، می‌توان با اطمینان بیشتری کد را تغییر داد و اطمینان حاصل کرد که تغییرات جدید

مشکلی ایجاد نخواهند کرد.

2. **تشخیص نقاط قوت و ضعف**: قبل از شروع refactoring، mutation coverage به شما نشان می‌دهد که کدام بخش‌های کد به خوبی توسط تست‌ها پوشش داده نشده‌اند. این اطلاعات می‌تواند در تصمیم‌گیری برای اولویت‌بندی بخش‌هایی که نیاز به تغییر دارند، مفید باشد.

3. **بهبود کیفیت کد**: Refactoring معمولا با هدف بهبود خوانایی و کارآیی کد انجام می‌شود و وقتی این تغییرات در کنار تست‌های قوی و جامع قرار می‌گیرد، امکان پذیرش تغییرات با خیال راحت بیشتر می‌شود. Mutation coverage می‌تواند به اطمینان از اینکه تست‌ها پس از refactoring همچنان کارآمد هستند، کمک کند.

4. **فراهم آوردن اطمینان**: وجود پوشش تغییرات بالا قبل از refactoring به توسعه‌دهندگان اطمینان می‌دهد که هرگونه خطا یا ناسازگاری ناشی از تغییرات، به سرعت توسط سیستم تست شناسایی و رفع خواهد شد.

بنابراین، **Mutation Coverage** بالا نه تنها نشان‌دهنده‌ی قدرت تست‌های شما است، بلکه می‌تواند پشتوانه‌ای برای انجام دادن refactoringهای امن و کارآمد باشد. این اطمینان از کیفیت، امکان تغییر و بهبود مستمر بدون ترس از ایجاد خطاهای ناخواسته را فراهم می‌آورد.