گزارش‌کار تمرین کامپیوتری پنجم آزمون نرم افزار

اولدوز نیساری 810199505

مجید فریدفر 810199569

لینک ریپازیتوری: https://github.com/maj2idfar/Baloot

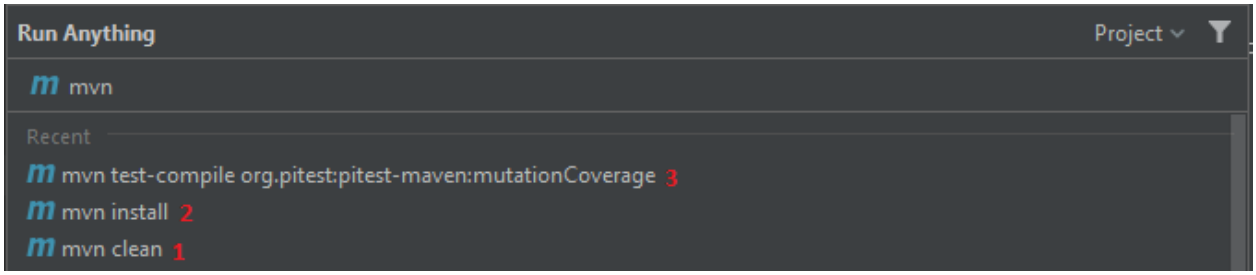
هش آخرین کامیت: 661dcaf

تغییرات داده شده

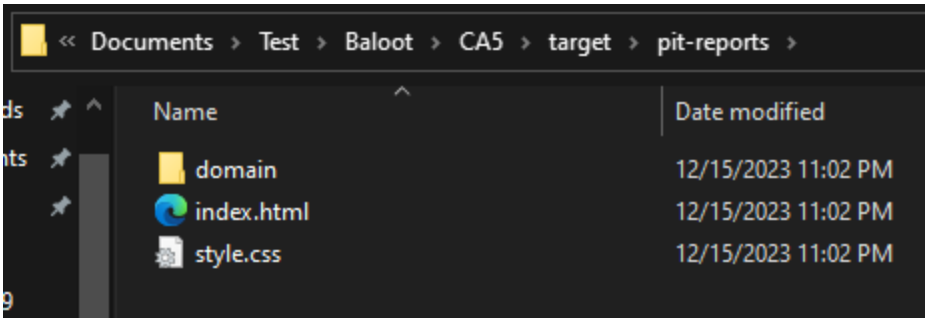
برای افزودن پلاگین Pitest، کد زیر را به فایل pom.xml (اینجا) اضافه می‌کنیم (در بخش plugins):

```
<plugin>
   <groupId>org.pitest</groupId>
   <artifactId>pitest-maven</artifactId>
   <version>1.15.2</version>
   <dependencies>
       <dependency>
           <groupId>org.pitest</groupId>
           <artifactId>pitest-junit5-plugin</artifactId>
           <version>1.2.1</version>
       </dependency>
   </dependencies>
</plugin>
```

سپس برای ران کردن تست‌ها، سه دستور زیر را به این ترتیب ران می‌کنیم:



گزارش‌های تولید شده را در این فولدر مشاهده می‌کنیم (با دیدن فایل index.html):



خوشبختانه ما خوش‌شانس بودیم و همان تست‌هایی که برای فاز قبلی نوشته بودیم، کاوریج حداکثری گرفتند. 🙂

بخش اول

```
> pre-scan for mutations : < 1 second
> scan classpath : < 1 second
> coverage and dependency analysis : 2 seconds
> build mutation tests : < 1 second
> run mutation analysis : 7 seconds
-----------------------------------------------------------------------------
> Total  : 10 seconds
-----------------------------------------------------------------------------
=============================================================================
- Statistics
=============================================================================
>> Line Coverage (for mutated classes only): 49/49 (100%)
>> Generated 29 mutations Killed 28 (97%)
>> Mutations with no coverage 0. Test strength 97%
>> Ran 39 tests (1.34 tests per mutation)
Enhanced functionality available at https://www.arcmutate.com/
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  22.450 s
[INFO] Finished at: 2023-12-15T17:42:25+03:30
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

- تعداد mutant های ساخته شده : 29

- تعداد mutant های کشته شده توسط آزمون های نوشته (killed ها) : 28
- تعداد mutant های زنده مانده پس از آزمون های شما (lived ها) : 29 - 28 = 1
- در گزارش کار خود در رابطه با تاثیر Mutation Coverage بالا در میزان خطر refactoring بحث کنید.

به طور کلی از آنجایی که mutation coverage بالا به این معنی است تست ها قابلیت تشخیص تغییرات کوچکی را که می توانند بر رفتار نرم افزار تاثیرگذار باشند را دارند .mutation coverage بالا می تواند اطمینان بیشتری ایجاد کند و در نتیجه تغییرات ناشی از refactoring را شناسایی کند و در نتیجه خطر آن را کاهش دهد . در واقع mutation coverage بالا با ایجاد خطا می توانند به اطمینان بیشتر کمک کند. البته قابل ذکر است که دستیابی به muatation coverage بالا ممکن است نیاز به تست هایی با جزئیات بسیار زیاد که این مسئله باعث ایجاد مشکل درجلوگیری از refactoring شوند ، چرا که در این شرایط خود تست ها به تغییرات کوچک کد حساس می شوند و با این تغییرات نیاز به تغییر تست ها هم هست که خود این مسئله از آنجایی که maintenance را دشوار تر می کند خطر refactoring را افزایش می دهد.

- - گزارشی که PIT برای کد می دهد را تحلیل و به گزارش اضافه کنید.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	100%	49/49	97%	28/29	97%	28/29

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
domain	2	100%	49/49	97%	28/29	97%	28/29

Report generated by PIT 1.15.2

Enhanced functionality available at arcmutate.com

/target/pit-reports/index.html

Pit Test Coverage Report

Package Summary

domain

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	100%	49/49	97%	28/29	97%	28/29

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Engine.java	100%	40/40	95%	20/21	95%	20/21
Order.java	100%	9/9	100%	8/8	100%	8/8

Report generated by PIT 1.15.2

/target/pit-reports/domain/index.html

```
57 2            } else if (diff != currentOrder.quantity - previous.quantity) {
58                  return 0;
59              }
60          }
61
62 1        return diff;
63      }
64
65      int getCustomerFraudulentQuantity(Order order) {
66
67          var averageOrderQuantity = getAverageOrderQuantityByCustomer(order.customer);
68
69 2        if (order.quantity > averageOrderQuantity) {
70 2            return order.quantity - averageOrderQuantity;
71          }
72
73          return 0;
74      }
75
76      public int addOrderAndGetFraudulentQuantity(Order order) {
77 1        if (orderHistory.contains(order)) {
78              return 0;
79          }
80
81          var quantity = getCustomerFraudulentQuantity(order);
82 1        if (quantity == 0) {
83              quantity = getQuantityPatternByPrice(order.price);
84          }
85
```

/target/pit-reports/domain/Engine.java.html

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
34	1. replaced int return with 0 for domain/Engine::getAverageOrderQuantityByCustomer → KILLED 2. Replaced integer division with multiplication → KILLED
38	1. negated conditional → KILLED
46	1. negated conditional → KILLED
50	1. negated conditional → KILLED
54	1. negated conditional → KILLED
55	1. Replaced integer subtraction with addition → KILLED
57	1. negated conditional → KILLED 2. Replaced integer subtraction with addition → KILLED
62	1. replaced int return with 0 for domain/Engine::getQuantityPatternByPrice → KILLED
69	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
70	1. replaced int return with 0 for domain/Engine::getCustomerFraudulentQuantity → KILLED 2. Replaced integer subtraction with addition → KILLED
77	1. negated conditional → KILLED
82	1. negated conditional → KILLED
87	1. replaced int return with 0 for domain/Engine::addOrderAndGetFraudulentQuantity → KILLED

/target/pit-reports/domain/Engine.java.html

Order.java

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

Mutations

```
9   1. replaced int return with 0 for domain/Order::getId → KILLED
10  1. replaced int return with 0 for domain/Order::getCustomer → KILLED
11  1. replaced int return with 0 for domain/Order::getPrice → KILLED
12  1. replaced int return with 0 for domain/Order::getQuantity → KILLED
16  1. negated conditional → KILLED
17  1. negated conditional → KILLED
    2. replaced boolean return with true for domain/Order::equals → KILLED
19  1. replaced boolean return with true for domain/Order::equals → KILLED
```

/target/pit-reports/domain/Order.java.html

همانطور که مشاهده می‌کنید،

در ابتدا pit test را می بینیم . این بخش مربوط به mutation coverage است که عملکرد را در پروسه وارد کردن تست های مصنوعی و تشخیص داده شدن آن ها را به ما نمایش می دهد .

این بخش به ما نشان می‌دهد که 29 تا mutant ساخته شده اند ، که 28 تا از آن ها کشته شده اند و یکی زنده مانده است . برای مشاهده جزئیات بیشتر وارد بخش domain می شویم .

در بخش domain مشاهده می کنیم که کلاس order.java و کلاس engine .java تمام لاین ها را پوشش داده اند ، در order.java تمامی mutant هایشان کشته شده اند ولی در engine.java یکی از mutant ها زنده مانده است.

همان طور که در عکس سوم مشاهده می کنید اگر کلاس ها را بررسی کنیم متوجه میشویم کشته نشده یک mutant به دلیل یک if رخ داده است و دلیل آن هم استفاده mutant از <= به جای < است که از ورود به if جلوگیری کرده است.

عکس آخر هم کشته شده همه mutant ها در تست های مربوط order.java را نشان می دهد.

در مجموع می توانیم بگوییم توانستیم mutation coverage خوبی کسب کنیم و تنها مشکلی که داشتیم زنده ماندن جهش changed conditional boundary بود که آن هم به دلیل مشکلات کشته شدن حالت نقیض می باشد و به نوعی به دلیل ذات تابع است.

به طور کلی در تحلیل گزارشی PIT میتوانیم موارد زیر را ذکر کنیم :

1) نشان دهنده mutation coverage که یک معیار مهم نشان دهنده درصد جهش muatation ها است.
2) دسته بندی mutant ها بر اساس کشته شده ، زنده مانده یا برخورد کرده با ارور را نشان می دهد .
3) نشان دهنده line coverage است که یک معیار کمی است.
4) اطلاعات دقیق ارائه شده می تواند به شناسایی ناحیه هایی که تست در آن ها موثر است و ناحیه هایی که موثر نیست در کد کمک کند.