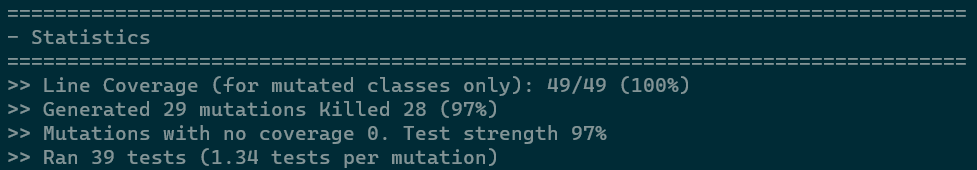
***Github Repository:*** <https://github.com/PashaBarahimi/Software-Testing-Course-Projects>

***Last Commit:*** b08d828c4b96d013ca6c3dcabfd099d380198527

# بخش اول

## نتیجه اجرای تست Mutation



### تعداد Mutant-های ساخته شده

همانطور که در تصویر مشاهده می‌شود، 29 عدد Mutant طی پردازش Pitest ساخته شده‌اند.

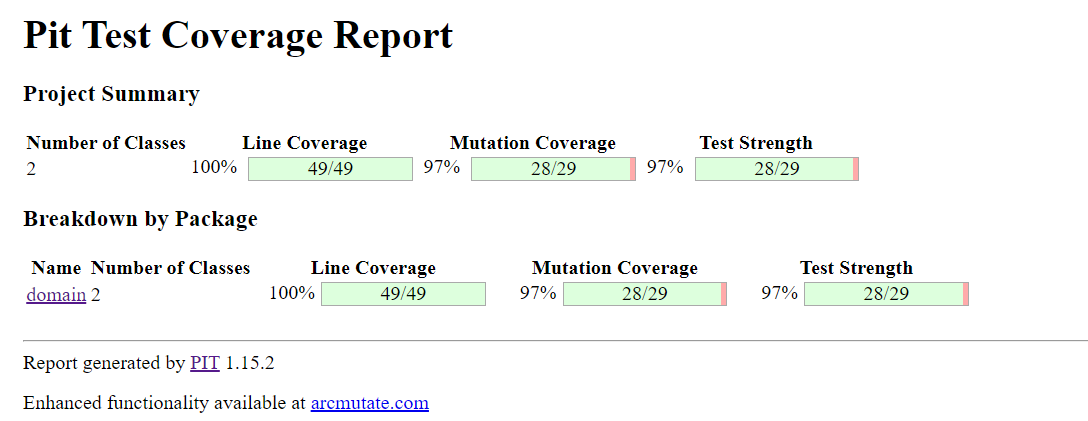
### تعداد Mutant-های کشته شده

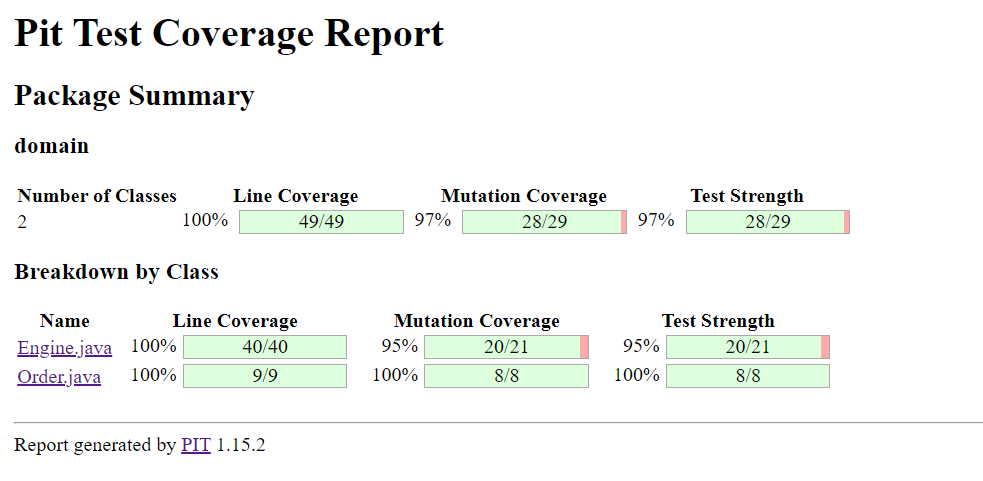
28 عدد از 29 تا Mutant ساخته شده توسط تست‌ها kill شده‌اند که باعث می‌شود به Mutation Coverage برابر با 97 درصد برسیم.

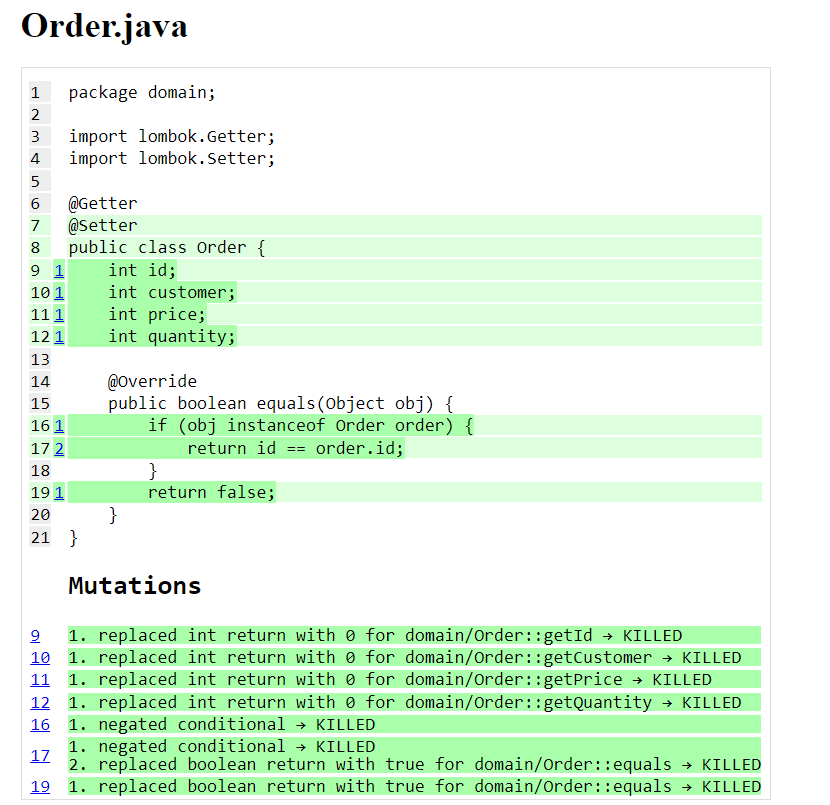
### تعداد Mutant-های زنده مانده

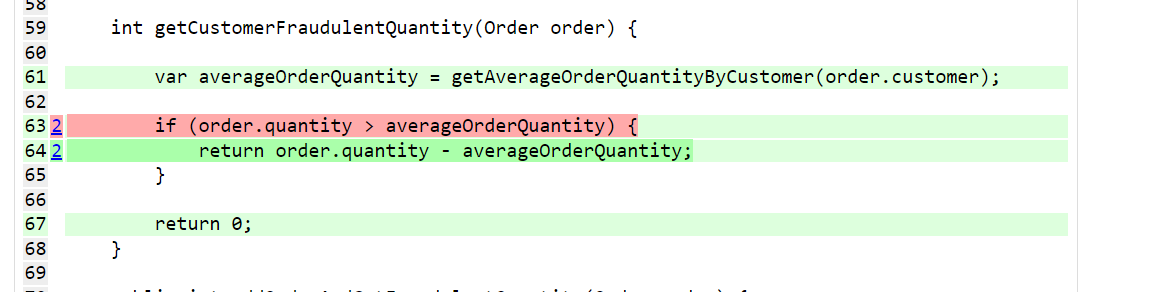
تنها 1 عدد از Mutant-ها زنده مانده‌اند که در ادامه (در بخش تحلیل) به توضیح چرایی آن می‌پردازیم.

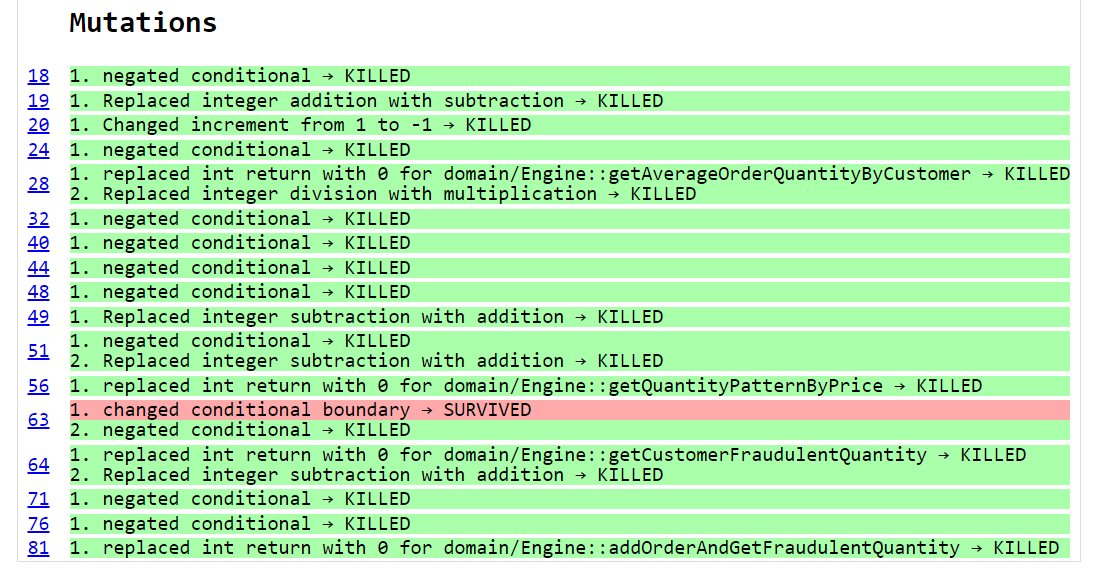
### گزارش Pitest











### تحلیل گزارش Pitest

همانطوز که مشاهده می‌شود، تنها Mutant-ای که kill نشده است، اپراتور Conditionals Boundary است که روی شرط if در تابع زیر اعمال شده است:

int getCustomerFraudulentQuantity(Order order) {  
 var averageOrderQuantity = getAverageOrderQuantityByCustomer(order.customer);  
  
 if (order.quantity > averageOrderQuantity) {  
 return order.quantity - averageOrderQuantity;  
 }  
  
 return 0;  
}

اتفاقی که توسط انجام این Mutation رخ داده، این است که اپراتور در شرط if به اپراتور تبدیل شده است. دلیل kill نشدن این Mutant این است که اگر تستی را در نظر بگیریم که در آن، مقدار order.quantity با مقدار averageOrderQuantity برابر باشد، مقدار return شده در داخل if با مقدار return شده در بیرون if که مقدار 0 است، برابر خواهد بود. در واقع در این حالت خروجی در هر دو حالت یکسان بوده و به اصطلاح، Equivalent Mutation داریم. به همین دلیل، امکان kill شدن این Mutant با هیچ تستی وجود ندارد.

## رابطه Mutation Coverage بالا در میان خطر Refactoring

هدف از Refactoring، بهبود ساختار کد و طراحی، خوانایی بالاتر، دیباگ سریع‌تر، و توسعه سریع‌تر است. اما نکته مهم در خصوص Refactoring این است که رفتار خارجی کد نباید در حین آن تغییر کند و باید عینا مانند رفتار قبل از Refactoring باشد.

از طرفی، Refactoring هم بر روی کد اصلی قابل انجام است و هم بر روی تست‌ها. زمانی که این کار را بر روی کد اصلی اعمال می‌کنیم، در صورتی که تست‌های خوبی داشته باشیم، می‌توان تقریبا مطمئن بود که رفتار کد در حین انجام این کار، تغییر نمی‌کند زیرا در صورت تغییر، تست‌ها fail می‌شوند و متوجه تغییر رفتار می‌شویم.

اما زمانی که می‌خواهیم عمل Refactoring را بر روی خود تست‌ها اعمال کنیم، هیچ Safety Net-ای وجود ندارد که مطمئن باشیم رفتار تست‌ها عینا مانند پیش از Refactor باشد. به همین دلیل، می‌توانیم از Mutation Testing برای اطمینان حاصل کردن از ثبات رفتار تست‌ها استفاده کنیم. این مورد به این صورت این که Mutation Coverage پس از انجام Refactoring باید دقیقا مانند پیش از Refactoring باشد. در این حالت، تا حد خوبی می‌توانیم مطمئن باشیم که Refactoring به درستی انجام شده است.

[منبع](https://arxiv.org/pdf/1506.07330)

# بخش دوم

برای ایجاد pipeline، فایل زیر را در آدرس .github/workflow/maven.yml قرار دادیم که هر دو پروژه را در زمان پوش شدن کد، تست کند. در صورتی که تست‌ها پاس نشوند، pipeline هم fail خواهد شد.

name: Java CI with Maven  
  
on:  
 push:  
 branches:  
 - master  
 pull\_request:  
 branches:  
 - master  
  
jobs:  
 test:  
 runs-on: ubuntu-latest  
  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v4  
  
 - name: Set up JDK 19  
 uses: actions/setup-java@v3  
 with:  
 java-version: '19'  
 distribution: 'oracle'  
 cache: 'maven'  
  
 - name: Run Baloot 1 tests  
 run: cd Baloot1 && mvn test -B -f pom.xml  
  
 - name: Run Baloot 2 tests  
 run: cd Baloot2 && mvn test -B -f pom.xml

نتیجه اجرای pipeline در تصویر زیر قابل مشاهده است:

