

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ В.Н. КАРАЗІНА**  
**ФАКУЛЬТЕТ «КОМП'ЮТЕРНИХ НАУК»**

**КУРСОВА РОБОТА**

з дисципліни «Математичні методи та технології тестування і верифікації  
програмного забезпечення»

Тема «Тестування мутацій (Mutation testing)»

Виконав студент 2 курсу

групи КС-21

Зоренко Ярослав Сергійович

Керівник:

ст. викладач Мелкозьорова О. М.

## РЕФЕРАТ

Звіт про виконання КР: 23 ст., 20 рис., 8 джерел.

Ключові слова: МУТАЦІЯ, ТЕСТУВАННЯ, КАЛЬКУЛЯТОР, ДІЛЕННЯ, МНОЖЕННЯ, ДОДАВАННЯ, ВІДНІМАННЯ, INFINITY.

Курсова робота передбачає тестування звичайного калькулятора за допомогою мутаційного методу тестування програмного забезпечення.

Калькулятор являє собою примітивний консольний застосунок, реалізований мовою програмування Java і виконаний в інтегрованому середовищі розробки програмного забезпечення IntelliJ IDEA. Він складається з чотирьох основних арифметичних дій: додавання, віднімання, множення та ділення. Під час тестування будуть перевірені ці дії і виявлено всі можливі помилки в програмному коді. Також в ході роботи будуть продемонстровані всі переваги та недоліки цього методу.

У поточній роботі розглядається мутаційний метод, який був використаний під час тестування програмної моделі, і за допомогою якого було виявлено ряд недоліків програмного продукту.

## ЗМІСТ

Перелік позначень та скорочень .....	3
Вступ.....	4
1 Теоретичні відомості.....	5
1.1 Тестування програмного забезпечення.....	5
1.2 Особливості методу тестування мутаціями .....	6
2 Практичні відомості .....	9
2.1 Інструментальні засоби реалізації мутаційного методу .....	9
2.2 Модель програмного продукту.....	11
2.3 Реалізація тестування мутаціями .....	15
Висновок.....	21
Список джерел інформації .....	22

## **ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ**

КР – курсова робота;

ТЗ – технічне завдання;

ОС – операційна система;

ПЗ – програмне забезпечення;

IDE (англ. Integrated development environment) – інтегроване середовище розробки.

## ВСТУП

В роботі розглядається програмна модель калькулятора, за допомогою якого буде відтворено мутаційне тестування, а саме проведено декілька тестів з використанням різних логічних та арифметичних операторів для отримання бажаного результату. Для найкращого відображення суті цього тесту спочатку буде взята дія ділення, над якою і відбудуватиметься мутація. Зрештою, інші оператори теж будуть протестовані.

У першому розділі розглядаються теоретичні відомості щодо тестування програмного забезпечення і необхідності його проведення. Для чого проводять тестування та чому тестування мутаціями актуальне на сьогоднішній день.

У другому розділі розглядаються практичні відомості, а саме методи реалізації мутаційного тестування, шляхом зміни логічних та арифметичних операторів, значення змінної або навіть її видалення. Також будуть розглянуті всі види мутацій вихідного коду, пов'язаних з перевіркою працездатності програмної моделі калькулятора.

Таким чином, в роботі буде порушено питання актуальності тестування програмного забезпечення цим методом та питання його надійності при використанні на практиці.

## 1 ТЕОРЕТИЧНІ ВІДОМОСТІ

### 1.1 Тестування програмного забезпечення

Тестування програмного забезпечення – це дуже важлива частина життєвого циклу будь-якого проекту. У зв'язку з шаленим розвитком ІТ-індустрії та інформаційним прогресом, виробники ПЗ вимушені з кожним роком більш серйозно відноситися до тестування своїх продуктів, а насамперед до продуктів, які мають пряме відношення до людського життя.

Перш за все, тестування – це процес технічного дослідження, призначений для виявлення дефектів. В більш глибокому сенсі, тестування ПЗ – це процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, який здійснюється шляхом спостереження за його роботою в штучно створених ситуаціях. Тестування має досить складну структуру і обов'язково включає в себе наступні етапи роботи:

- Планування робіт (Test Management);
- Проектування тестів (Test Design);
- Виконання тестування (Test Execution);
- Аналіз отриманих результатів (Test Analysis).

Ці чотири етапи є основою для всього процесу тестування і працюють незалежно від обраного життєвого цикла. Щодо життєвих циклів, можна зазначити, що їх всього три:

- Каскадний;
- V-образний;
- Спіральний.

В залежності від вибраної моделі життєвого циклу можна відповісти на питання як довго обслуговуватиметься поточний проект.

Найоптимальнішим серед них є спіральний життєвий цикл. Він поєднує в собі чотири основні фази тестування: аналіз, проектування, розробку та верифікацію ПЗ, які повторюються по черзі один за одним. Це дає змогу постійно

підтримувати гідний стан розробленого проекту та покращувати його в разі необхідності.

Хоча V-образний життєвий цикл не є на стільки універсальним, як спіральний, він користується досить великою популярністю. Завдячує він цим своїй досить цікавій структурі, яка поділяється на дві гілки. Перша – складається з поступової реалізації поставлених завдань. Друга використовує розгалужений сценарій на етапі перевірки, а саме перевірка та виправлення помилок в разі необхідності на конкретному етапі.

Найпростішим серед життєвих циклів є каскадний життєвий цикл. Він знову ж складається з декількох етапів, серед яких є етап тестування. В цьому випадку всі узгоджені етапи виконуються послідовно, виключно один за одним. В разі необхідності повернутися до будь-якого з етапів, команда, що працює над проектом, повинна завершити всі етапи, які залишилися, і лише потім повернутися до початкового.

## **1.2 Особливості методу тестування мутаціями**

Мутаційне тестування (мутаційний аналіз або мутація програм) – це особливий метод тестування програмного забезпечення, який ґрунтується на внесенні змін до вихідного коду програми. Внесені помилки зазвичай настільки незначні, що на спільні цілі програми вони не впливають. До таких змін можуть належати:

- Заміна логічного оператора;
- Заміна арифметичного оператора;
- Заміна оператора відношення;
- Видалення змінної, або навіть виразу;
- Введення свідомо некоректного значення та інші.

Після внесення помилок-мутантів цільова програма і програми-мутанти тестуються одними тестами. Якщо набір тестів не в змозі виявити такі зміни, то він розглядається як недостатній. Ці зміни називаються мутаціями і ґрунтуються на мутаційних операторах, які або імітують типові помилки програмістів або

стимулюють до створення корисних тестів. Також є випадки, коли мутація операторів може призвести до еквівалентних програм, наприклад оператор мутації умов може замінити «= =» на «> =». В таких випадках не існує тесту, який міг би вбити цього мутанта, бо отримана програма еквівалентна вихідній програмі. Такі мутанти називаються еквівалентними мутантами. Розпізнавання еквівалентних мутантів є одним з найбільших перешкод для використання мутаційного тестування на практиці, бо зусилля (затрати) для перевірки того, чи є мутант еквівалентним, можуть бути дуже великими навіть для невеликих програм. Це є досить великим мінусом та основною перешкодою до ще більшої популяризації цього методу тестування.

Мета Mutation Testing полягає в тому, щоб допомагати тестеру розробляти ефективні тести, або знайти слабкі місця в тестових даних, що використовуються для програми. Результатами виконання мутацій можуть бути наступні повідомлення:

- KILLED – в результаті мутації впали всі тести, що перевіряють цей рядок. Це означає, що всі помилки знайдені;
- SURVIVED – мутація пройшла непоміченою. Це означає, що зміна у функціональності не вкрита тестами;
- TIMED\_OUT – тест працював занадто довго (наприклад, в результаті виникнення нескінченного циклу);
- NON\_VIABLE – отриманий в результаті мутації байт-код з якоїсь причини виявився не дійсним;
- MEMORY\_ERROR – в результаті мутації код почав споживати дуже багато пам'яті і впав;
- RUN\_ERROR – в результаті мутації вийшов код, що генерує виняток.

Виходячи з цього всього, можна виділити головні переваги та недоліки методу тестування мутаціями.



#### Переваги:

- Дозволяє охопити всю вихідну програму;
- Всебічно тестуються програми-мутанти;
- Тестування розкриває всі неясності у вихідному коді.

#### Недоліки:

- Мутаційне тестування – надзвичайно дорогий і трудомісткий процес;
- Не автоматизований вид тестування;
- Не може бути застосований для тестування методом чорного ящика, оскільки цей спосіб передбачає зміни у вихідному коді.

Вважається, що Mutation Testing є найбільш універсальним методом тестування програмного продукту для таких мов програмування, як Java та XML, але нажаль, не здобув велику популярність через велику потребу ресурсів для його виконання.

## 2 ПРАКТИЧНІ ВІДОМОСТІ

### 2.1 Інструментальні засоби реалізації мутаційного методу

Головним елементом для реалізації мутаційного методу та написання програмного продукту – калькулятора – є середа розробки. В цій курсовій роботі, для розробки програми та проведення тестів, було обрано інтегроване середовище розробки програмного забезпечення IntelliJ IDEA (Рис. 2.1).



Рисунок 2.1 – Середа розробки IntelliJ IDEA

Особливістю цього середовища є велика багатофункціональність і зручність у використанні, а також дружній інтерфейс (Рис. 2.2). Це інтегроване середовище розробки програмного забезпечення для багатьох мов програмування, зокрема Java, JavaScript, Python та інші, було розроблено компанією JetBrains і була випущена в грудні 2001 року. IntelliJ IDEA, як і Visual Studio, має багато вбудованих модулів для простішої реалізації програмного продукту, зокрема модулем для створення UML діаграм та будь-яких блок-схем. Це дуже зручно особливо у випадках коли

необхідно структурувати поточний проект. Але на жаль ця функція доступна тільки в Ultimate версії і не доступна в звичайній Community версії.

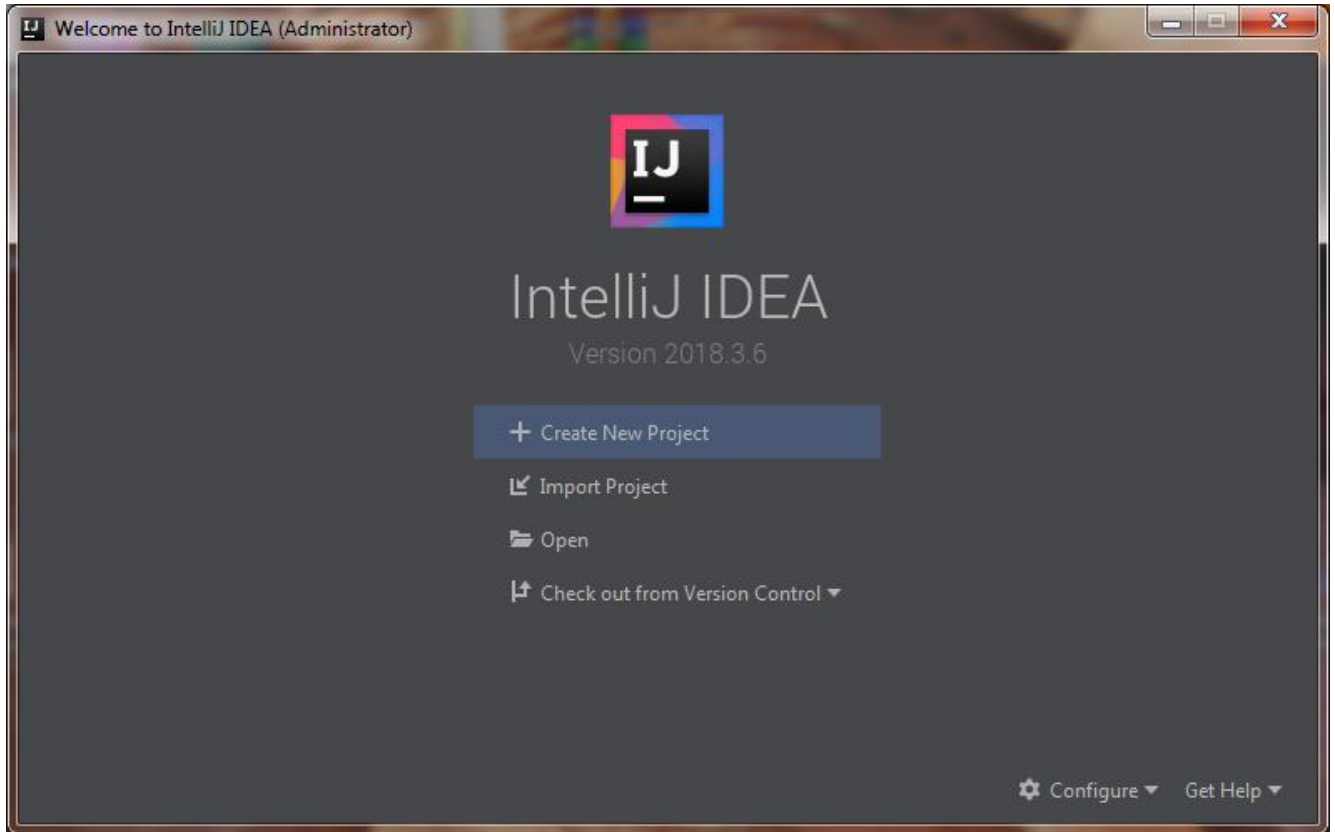


Рисунок 2.2 – Початок роботи з середою розробки ПЗ IntelliJ IDEA

Професійна версія IntelliJ має ще багато функцій для спрощування всіляких задач для швидкого та якісного відтворення моделі програмного продукту. А саме додаткові шаблони та розширену функціональність для більш простого користування тощо.

Автоматизовані функції дають змогу прискорювати процес створення кінцевого продукту та поліпшити умови праці. Це є дуже важливим аспектом при виборі середовища для розробки. Саме тому було обрано це ПЗ.

## 2.2 Модель програмного продукту

Моделлю поточного програмного продукту для тестування мутаціями було обрано калькулятор, створений за допомогою інтегрованого середовища розробки IntelliJ IDEA. Проект складається з підключених бібліотек, класу «Calculator», тест-класу «CalculateTest» та Maven-залежностей (Рис. 2.3)

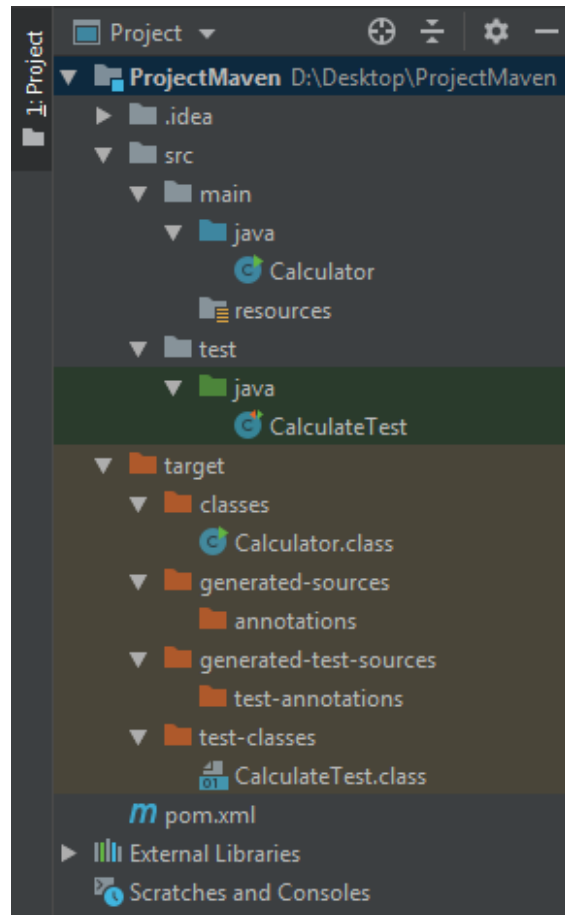


Рисунок 2.3 – Структура проекту

В свою чергу класу «Calculator» включає в себе п'ять методів: головний метод – «main» та чотири інші – «Sum», «Substraction», «Multiplication» і «Division».

Головний метод робить запит першого та другого числа, а також дії, яка буде між ними виконана. Залежно від дії, яку вибере користувач, буде викликано один з методів (Рис. 2.4).

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Calculator calculator = new Calculator();
    System.out.println("Number a:");
    int a = scanner.nextInt();
    System.out.println("Number b:");
    int b = scanner.nextInt();
    System.out.println("Choose operation +, -, *, /:");
    String operation = new String();
    operation = scanner.next();
    switch (operation){
        case "+":
            System.out.println(calculator.Sum(a,b));
            break;
        case "-":
            System.out.println(calculator.Substraction(a,b));
            break;
        case "*":
            calculator.Multiplication(a,b);
            break;
        case "/":
            calculator.Division(a,b);
            break;
    }
}

```

Рисунок 2.4 – Головний метод «main»

Кожен з інших чотирьох методів виконує одну з арифметичних дій та виводить відповідь на екран (Рис, 2.5).

```

public int Sum(int a,int b){
    System.out.println("Result  "+ (a+b)); //Заменяем знак "+" на любой другой "*", "+", "-",
    return (a+b); //получаем результат не "123" -> тест не пройден, мутант убит
}
public int Substraction(int a,int b){
    System.out.println("Result  "+ (a-b)); //Заменяем знак "-" на любой другой "*", "+", "-",
    return (a-b); //получаем результат не "123" -> тест не пройден, мутант убит
}
public int Multiplication(int a,int b){
    System.out.println("Result  "+ (a*b)); //Заменяем знак "*" на любой другой "*", "+", "-",
    return (a*b); //получаем результат не "0" -> тест не пройден, мутант убит
}
public double Division(double a,double b){
    System.out.println("Result  "+ (a/b)); //Заменяем знак "/" на любой другой "*", "+", "-",
    return (a/b); //получаем результат не "Infinity" -> тест не пройден, мутант убит
}

```

Рисунок 2.5 – Методи «Sum», «Substraction», «Multiplication» і «Division»

Тест-клас містить метод «TestDiv», який у свою чергу складається з чотирьох перевірок на наявність мутацій для кожного з методів головного класу. Проведення тесту передбачає отримання результату виконання конкретного методу – «result1» – із заданими змінними «a» та «b», встановлення очікуваного результату – «expectedresult1» та порівняння цих показників за допомогою функції «assertEquals» (Рис. 2.6).

```
public class CalculateTest {
    Calculator calculator = new Calculator();

    @Test
    public void TestDiv() {
        double result1 = calculator.Division( a: 123, b: 0);
        double expectedresult1 = Infinity;
        Assert.assertEquals(expectedresult1, result1, delta: 0);

        double result2 = calculator.Multiplication( a: 123, b: 0);
        double expectedresult2 = 0;
        Assert.assertEquals(expectedresult2, result2, delta: 0);

        double result3 = calculator.Substraction( a: 123, b: 0);
        double expectedresult3 = 123;
        Assert.assertEquals(expectedresult3, result3, delta: 0);

        double result4 = calculator.Sum( a: 123, b: 0);
        double expectedresult4 = 123;
        Assert.assertEquals(expectedresult4, result4, delta: 0);
    }
}
```

Рисунок 2.6 – Тест-клас «CalculateTest»

Також бачимо структуру Maven-залежності (Рис. 2.7)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>ProjectMaven</groupId>
  <artifactId>ProjectMaven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>7</source>
          <target>7</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>3.14.0</version>
    </dependency>
  </dependencies>
</project>
```

Рисунок 2.7 – Maven-залежності

## 2.3 Реалізація тестування мутаціями

Тестування програмного продукту перш за все починається з тестування початкового стану програмного продукту для подальшого порівняння результатів. Використовуємо оператор додавання (Рис. 2.8).



```
Run: Calculator x
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
Number a:
123
Number b:
0
Choose operation +, -, *, /:
+
Sum Result  123
123

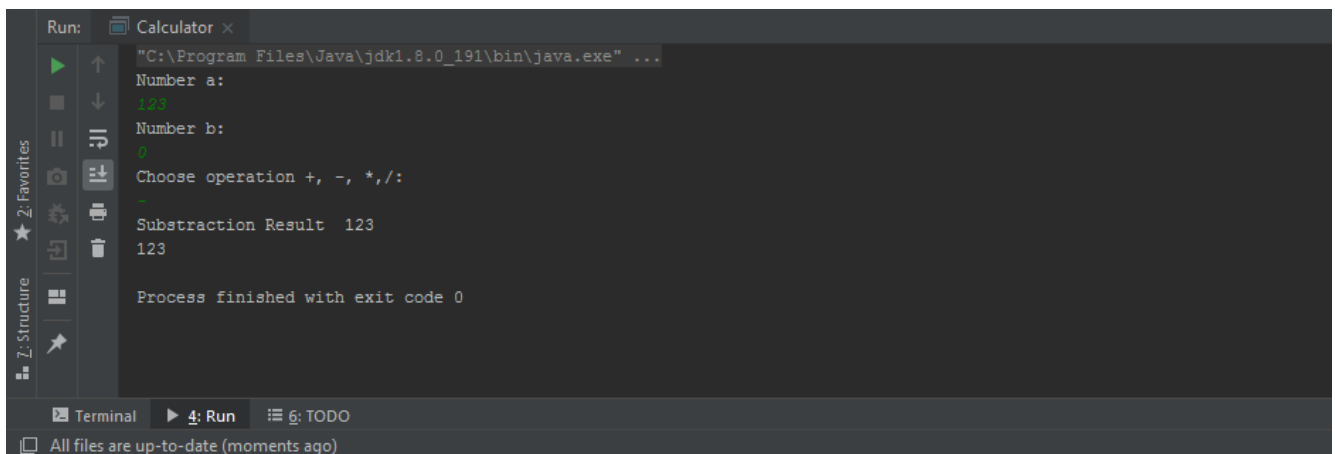
Process finished with exit code 0
```

Terminal | Run | TODO

All files are up-to-date (3 minutes ago)

Рисунок 2.8 – Огляд роботи програмного продукту №1

Використовуємо оператор віднімання (Рис. 2.9).



```
Run: Calculator x
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
Number a:
123
Number b:
0
Choose operation +, -, *, /:
-
Subtraction Result  123
123

Process finished with exit code 0
```

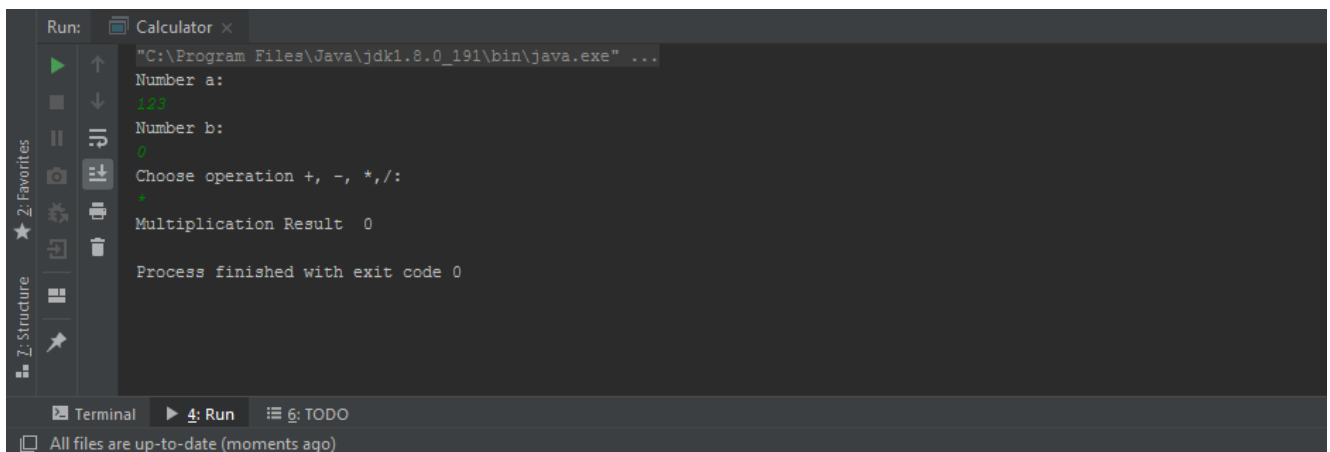
Terminal | Run | TODO

All files are up-to-date (moments ago)

Рисунок 2.9 – Огляд роботи програмного продукту №2



Використовуємо оператор множення (Рис. 2.10).



```
Run: Calculator x
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
Number a:
123
Number b:
5
Choose operation +, -, *, /:
*
Multiplication Result  0

Process finished with exit code 0
```

Terminal | Run | TODO

All files are up-to-date (moments ago)

Рисунок 2.10 – Огляд роботи програмного продукту №3

Використовуємо оператор ділення (Рис. 2.11).



```
Run: Calculator x
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
Number a:
123
Number b:
0
Choose operation +, -, *, /:
/
Division Result  Infinity

Process finished with exit code 0
```

Terminal | Run | TODO

All files are up-to-date (moments ago)

Рисунок 2.11 – Огляд роботи програмного продукту №4

Далі переходимо до створення невеликих мутацій в необхідних місцях програми, а саме мутацій арифметичних операторів в класі «Calculator». Спочатку мутуємо метод «Sum» (Рис. 2.12).

```
public int Sum(int a,int b){
    System.out.println("Sum Result  "+ (a/b)); //Заменяем знак "+" на любой другой "*", "+", "-",
    return (a+b);                               //получаем результат не "123" -> тест не пройден, мутант убит
}
```

Рисунок 2.12 – Мутація ключового оператора метода «Sum»

Бачимо негативний результат тестування програми. Тест не пройдено, мутацію вбито (Рис. 2.13).

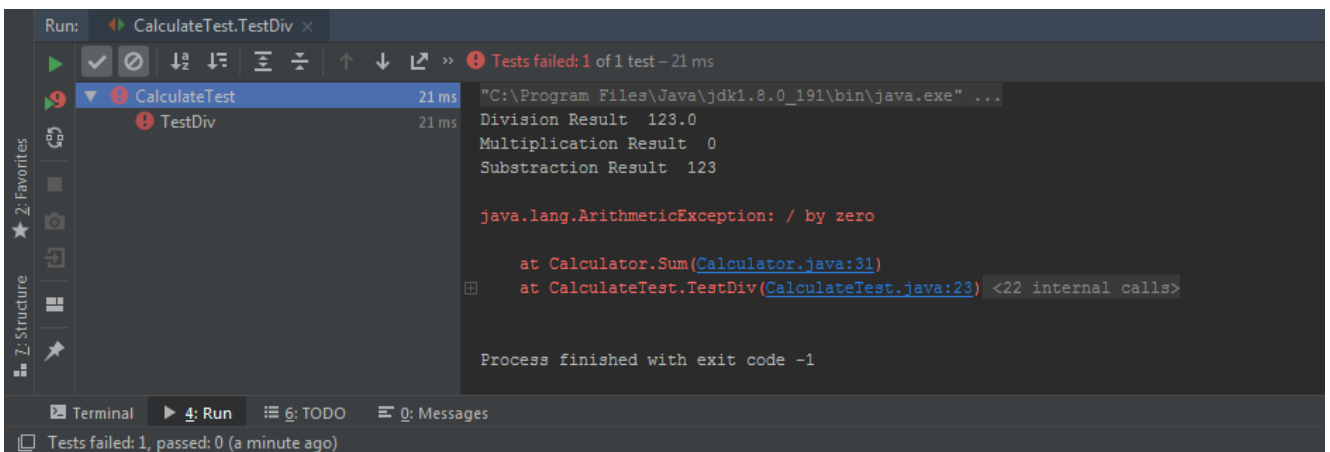


Рисунок 2.13 – Негативний результат тестування метода «Sum»

Далі мутуємо інший метод – «Substraction» (Рис. 2.14).

```
public int Substraction(int a,int b){
    System.out.println("Substraction Result  "+ (a/b)); //Заменяем знак "-" на любой другой "*", "+", "-",
    return (a-b);                               //получаем результат не "123" -> тест не пройден, мутант убит
}
```

Рисунок 2.14 – Мутація ключового оператора метода «Substraction»

Знову бачимо негативний результат тестування програми. Тест не пройдено, мутацію вбито (Рис. 2.15).

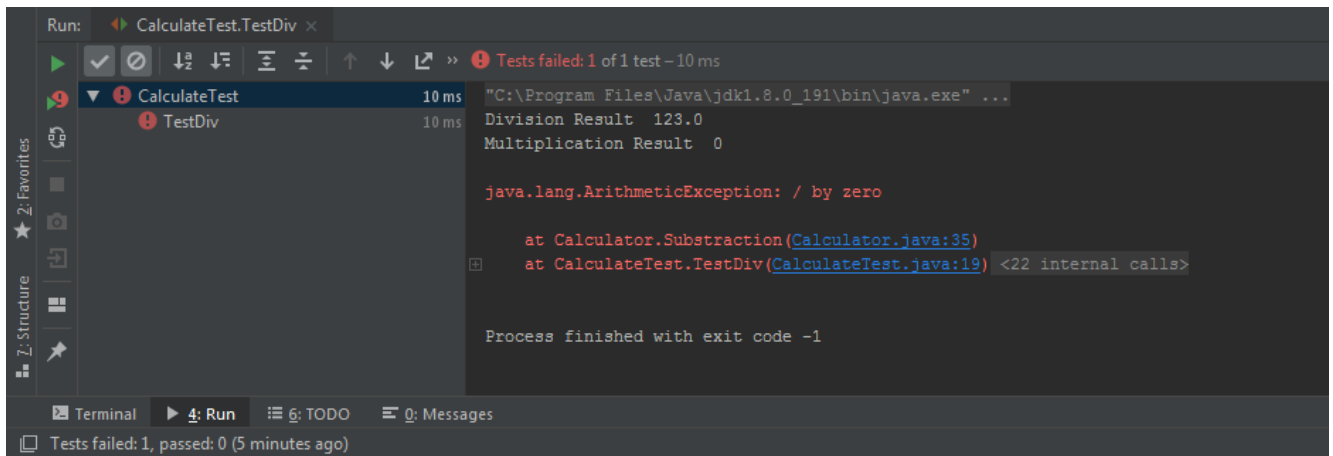


Рисунок 2.15 – Негативний результат тестування метода «Substraction»

Мутуємо наступний метод – «Multiplication» (Рис. 2.16).

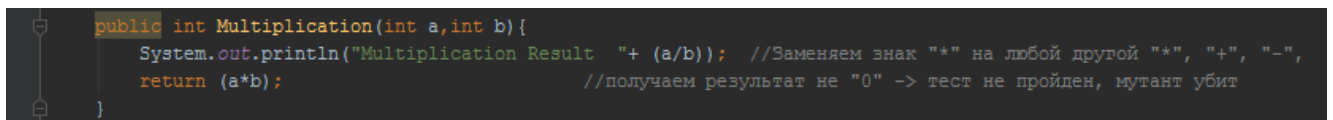


Рисунок 2.16 – Мутація ключового оператора метода «Multiplication»

І знову бачимо негативний результат тестування програми. Тест не пройдено, мутацію вбито (Рис. 2.17).

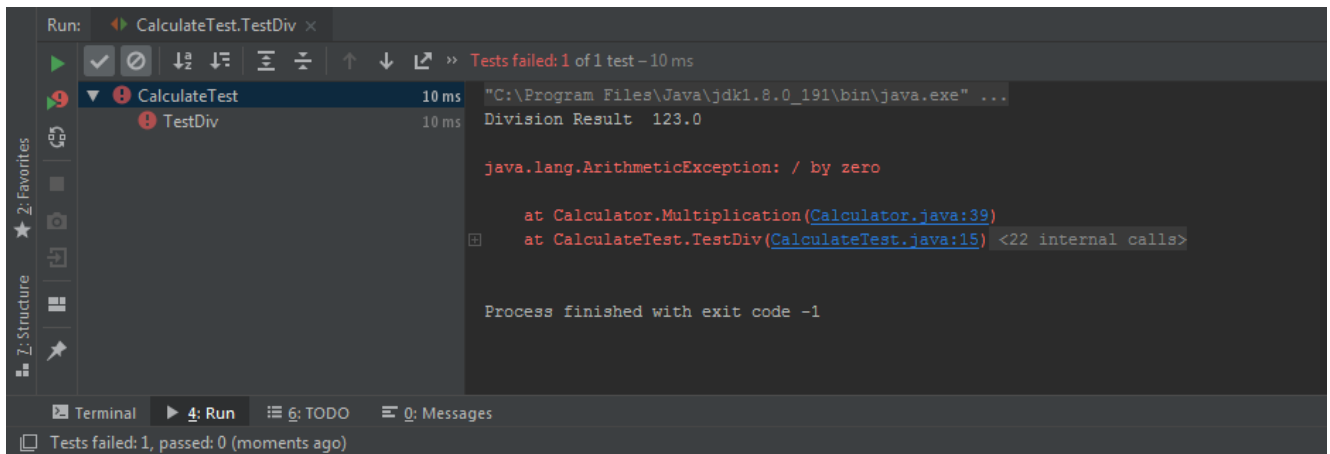


Рисунок 2.17 – Негативний результат тестування метода «Multiplication»

Нарешті мутуємо останній метод – «Division» (Рис. 2.18).

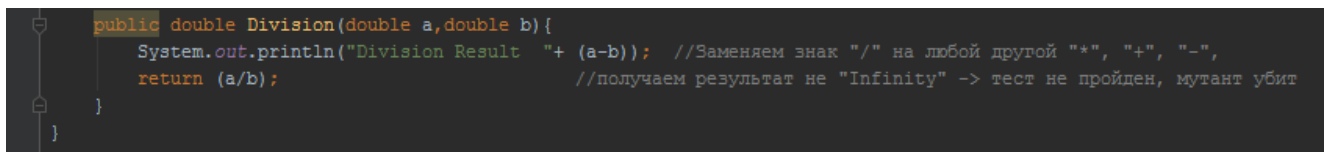


Рисунок 2.18 – Мутація ключового оператора метода «Division»

Спостерігаємо дивне явище: мутацію зроблено, але тест пройшов без жодних помилок. Отже тест пройдено і мутація не була вбита (Рис. 2.19).

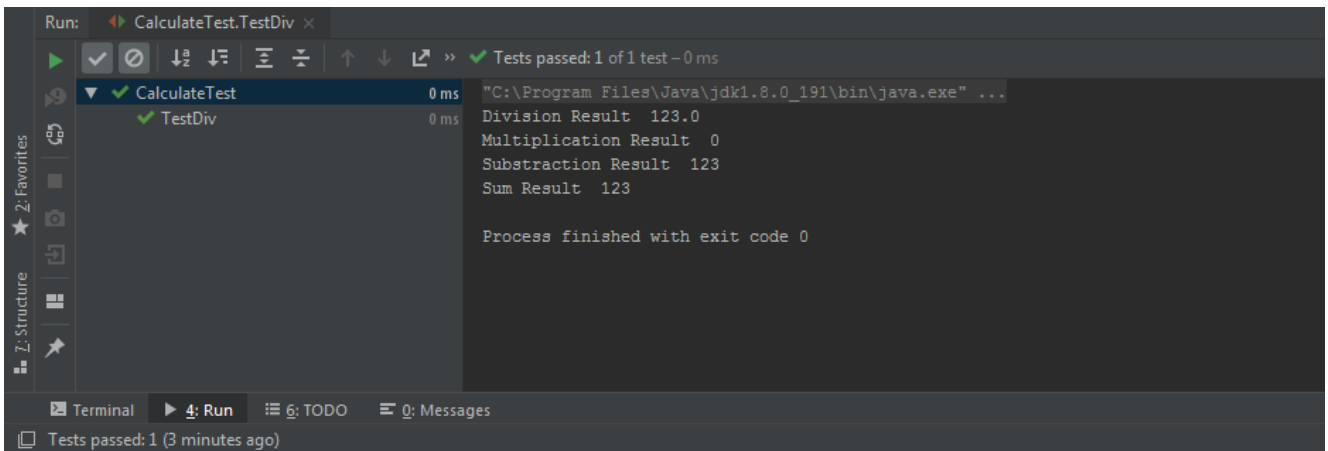


Рисунок 2.19 – Позитивний результат тестування метода «Division»

Звідси виходить, що при мутуванні ключового оператора метода «Division» було зроблено еквівалентне мутування, яке в подальшому тестуванні проекту не було знайдено. Це і призвело до ігнорування тестом зроблених змін.

Змінюємо значення ключового оператора метода «Division» на потрібне і бачимо, що тест знову ж таки проходить успішно, але вже з правильною відповіддю (Рис. 2.20).

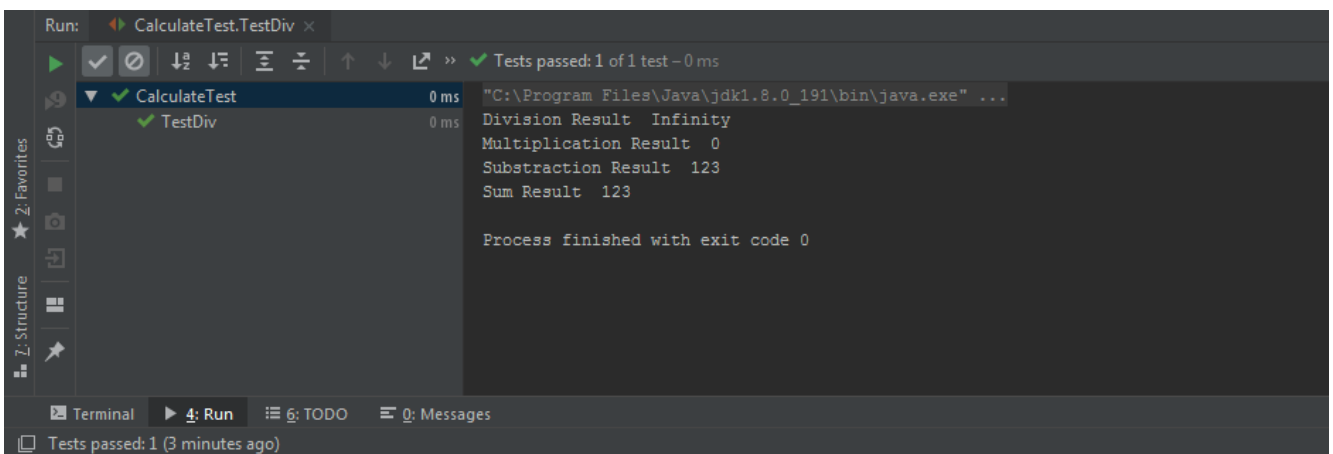


Рисунок 2.20 – Правильний результат тестування метода «Division»

## ВИСНОВОК

В роботі була розглянута програмна модель калькулятора, за допомогою якого було відтворено мутаційне тестування, а саме проведено декілька тестів з використанням різних логічних та арифметичних операторів для отримання бажаного результату. Для найкращого відображення суті цього тесту спочатку було взято дію ділення, над якою і відбувалася мутація. Зрештою, інші оператори теж були протестовані.

У першому розділі розглядалися теоретичні відомості щодо тестування програмного забезпечення і необхідності його проведення. Для чого проводять тестування та чому тестування мутаціями актуальне на сьогоднішній день.

У другому розділі розглядалися практичні відомості, а саме методи реалізації мутаційного тестування, шляхом зміни логічних та арифметичних операторів, значення змінної або навіть її видалення. Також були розглянуті всі види результатів виконання мутацій вихідного коду, пов'язаних з перевіркою працездатності програмної моделі калькулятора.

Таким чином, в роботі були порушені питання актуальності тестування програмного забезпечення цим методом та його надійності використання на практиці.

## СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

- 1 Jefferson O. A Practical System for Mutation Testing: Help for the Common Programmer. / 2016. – 437 с.
- 2 Weisfeld M. The Object-Oriented Thought Process / Matt Weisfeld. // Addison-Wesely. – 2013. – №4. – С. 306.
- 3 Available Mutators And Groups [Електронний ресурс] / Mutators – Режим доступу до ресурсу: <http://pitest.org/quickstart/mutators/>.
- 4 Frankl P./ All-uses versus mutation testing: An experimental comparison of effectiveness. Journal of Systems and Software, /. Weiss S., / Hu C. – 38:235-253, 1997.
- 5 Roger A., Mutation of Java Objects /. Bieman J./ Sudipto G/ Bixia J. – 2014 – №1 – С. 285.
- 6 Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. / Creig Larman., 2004. – 627 с. – (Addison-Wesely).
- 7 Jeremy S. Mutation Operators for Concurrent Java (J2SE 5.0) / James R., Juergen D./ – С.142.
- 8 Yu-Seung M. / MuJava: An Automated Class Mutation System by / – С. 43 –, Jeff O., Yong R.