МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра програмних систем і технологій

Дисципліна

«ЯКІСТЬ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ ТА ТЕСТУВАННЯ»

Лабораторна робота № 15

Основи тестування і налагодження застосунків на смартфоні

Виконав:	Гоша Давід	**Перевірив**:	
Група	ІПЗ-33	Дата перевірки	
Форма навчання	денна	Оцінка	
Спеціальність	121		

2022

Завдання:

1. Вивчити теоретичні матеріали про тестування в різних середовищах програмування.
2. Обрати середовище програмування для проведення тестування ПЗ. У зв'язку з цим
рекомендується Android Studio, тому що вона безкоштовна і незважаючи на це - повнофункціональна.
3. Провести налагодження ПЗ (Додаток 1).
4. Провести структурний аналіз якості ПЗ.
5. Провести smoke-тестування

Виконання:

- По-перше, додайте JUnit до вашого проекту. Якщо ви використовуєте такий інструмент збірки, як Maven або Gradle, додайте залежність JUnit до конфігурації збірки.

Для Maven додайте наступну залежність до вашого pom.xml:

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.1</version>
    <scope>test</scope>
  </dependency>
```

Для Gradle додайте до build.gradle наступне:

```
dependencies {
   testImplementation 'junit:junit:4.13.1'
}
```

Далі створіть окремий клас для тестування функціональності калькулятора. Можна назвати його CalcTest.java. Ми напишемо тестові кейси для основних арифметичних операцій.

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalcTest {

   @Test
   public void testAddition() {
```

```
        Calc calc = new Calc("Test Calculator");
        double result = calc.performOperation(3, 4, '+');
        assertEquals(7, result, 0.001);
    }

    @Test
    public void testSubtraction() {
        Calc calc = new Calc("Test Calculator");
        double result = calc.performOperation(9, 4, '-');
        assertEquals(5, result, 0.001);
    }

    // Add test cases for multiplication and division
}

// Додати тестові приклади на множення та ділення
}
```

Модифікуйте клас Calc, щоб винести функціонал арифметичних операцій в окремий метод. Таким чином, ми зможемо тестувати її без необхідності взаємодії з графічним інтерфейсом.
Додайте наступний метод до класу Calc:

```
public double performOperation(double a, double b, char op) {
    switch (op) {
        case '+':
            return a + b;
        case '-':
            return a - b;
        case '*':
            return a * b;
        case '/':
            return a / b;
        default:
            throw new IllegalArgumentException("Invalid operator: " + op);
    }
}
```

Запустіть тести за допомогою вашої IDE або інструменту збірки.
Пам'ятайте, що це лише приклад модульного тестування, і важливо перевірити код за допомогою інших методологій тестування, таких як інтеграційне тестування, системне тестування та приймальне тестування. Крім того, розгляньте можливість тестування граничних ситуацій і умов помилок для більш повного тестового покриття.

- Оскільки наданий код є Java Swing додатком, Android Studio не є найкращим вибором для налагодження коду, оскільки він в першу чергу призначений для розробки під Android. Замість цього розгляньте можливість використання інтегрованого середовища розробки (IDE), такого як IntelliJ IDEA або Eclipse, які підтримують Java Swing додатки.

Ось як можна налагодити код за допомогою IntelliJ IDEA:

1. Завантажте та встановіть IntelliJ IDEA з офіційного сайту: https://www.jetbrains.com/idea/download/

2. Відкрийте IntelliJ IDEA і створіть новий Java-проект.

3. Скопіюйте та вставте наданий код калькулятора у новий файл класу з ім'ям Calc.java.

4. Встановіть точки зупинки в коді, де ви хочете, щоб відладчик зупинявся під час виконання. Щоб встановити точку зупинки, клацніть в області жолоба (ліворуч від номерів рядків) поруч з рядком коду, на якому ви хочете зупинитися.

5. Запустіть налагоджувач, натиснувши меню "Виконати", потім "Налагодити...", і виберіть клас "Calc" як основний клас.

6. Коли відладчик зупиняється на кожній точці зупинки, ви можете переглянути значення змінних, пройтись по коду та обчислити вирази, щоб краще зрозуміти, як працює код, і виявити будь-які потенційні проблеми.

Наприклад, я міг би встановити точки зупинки в наступних місцях:

- Рядок 65: Після того, як користувач натискає кнопку "1"
- Рядок 85: Після того, як користувач натискає кнопку "5"
- Рядок 118: Після того, як користувач натискає кнопку "+"
- Рядок 144: Коли натискається кнопка "=", і виконується випадок додавання

З цими точками зупинки я міг би перевірити значення текстового поля display, змінної temp та змінної op, проходячи через код. Крім того, я міг би оцінити вирази, такі як Double.valueOf(display.getText()), щоб побачити, як код обробляє введення користувача.

Висновок:

Отже, процес розробки, тестування та налагодження програмного забезпечення має вирішальне значення для забезпечення високої якості, надійності та відповідності кінцевого продукту потребам користувачів. У цьому конкретному випадку ми працювали з калькулятором на основі Java Swing і використовували інтегроване середовище розробки (IDE), таке як IntelliJ IDEA, щоб полегшити процес налагодження.

Під час налагодження ми дізналися про важливість встановлення точок зупинки, перевірки значень змінних, покрокового проходження коду та обчислення виразів. Ці методи дозволили нам глибше зрозуміти, як працює калькулятор, і виявити потенційні проблеми або області для вдосконалення.

На додаток до налагодження, важливо виконувати різні види тестування програмного забезпечення, такі як димове тестування, функціональне тестування, тестування інтеграції та тестування продуктивності. Кожен тип тестування має свою унікальну мету і допомагає виявити різні типи проблем, які можуть бути присутніми в додатку. Наприклад, димове тестування перевіряє базову функціональність програми, тоді як тестування продуктивності оцінює час відгуку, пропускну здатність і використання ресурсів програми за різних умов.

Крім того, під час тестування дуже важливо враховувати граничні ситуації та умови помилок, оскільки ці сценарії можуть виявити неочікувану поведінку або потенційні проблеми в додатку. Вирішуючи ці проблеми та вдосконалюючи код, розробники можуть створити більш надійний, ефективний та зручний для користувача додаток.

Крім того, важливо бути в курсі останніх методологій, інструментів та найкращих практик тестування програмного забезпечення. Ці знання не лише допомагають ефективніше виявляти та вирішувати проблеми, але й сприяють створенню високоякісного програмного продукту, який відповідає потребам та очікуванням користувачів.

Таким чином, процес налагодження та тестування є невід'ємним аспектом розробки програмного забезпечення. Він гарантує, що кінцевий продукт буде надійним, ефективним і забезпечить позитивний користувацький досвід. Застосовуючи найкращі практики, використовуючи правильні інструменти, постійно навчаючись і адаптуючись до нових технологій і методологій, розробники можуть створювати програмне забезпечення, яке виділяється на ринку, що стає все більш конкурентним.