Національний технічний університет України
«Київський політехнічний університет імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра Системного проектування

Лабораторна робота № 5

з предмету «Проектування інформаційних систем»

Виконав:
Студент групи ДА-72
ННК «ІПСА»
Кандель К.В.

Варіант № 8

Київ – 2020

Мета роботи: оволодіти навичками створення програмного забезпечення за метолологією TDD та ознайомитися з процедурами рефакторинга.

Хід роботи:

1. Створимо функції для тестування.

В даній лабораторній оберемо функції для обрахунку площі квадрату, кола, та трикутника

```
public double getTriangleSquare(int a, int b, int c) {
    return 0;
}

public double getSquareSquare(int a) {
    return 0;
}

public double getCircleSquare(int a) {
    return 0;
}
```

2. Напишемо тести

Оскільки використовуємо методологію TDD, то спочатку потрібно написати тести. Для написання тестів використаємо java бібліотеку Jupiter, яка включає в себе необхідні анотації та методи.

```
@Test
public void whenPassWrongArgumentToTriangleSquare_thenExceptionThrown() {
    assertThrows(IllegalArgumentException.class, () -> FigureUtils.getTriangleSquare( a: -1,  b: 5,  c: 5));
    assertThrows(IllegalArgumentException.class, () -> FigureUtils.getTriangleSquare( a: 5,  b: -5,  c: 5));
    assertThrows(IllegalArgumentException.class, () -> FigureUtils.getTriangleSquare( a: 5,  b: 5,  c: -5));
}

@Test
public void whenPassWrongArgumentToSquareSquare_thenExceptionThrown() {
    assertThrows(IllegalArgumentException.class, () -> FigureUtils.getSquareSquare( a: -1));
}

@Test
public void whenPassWrongArgumentToCircleSquare_thenExceptionThrown() {
    assertThrows(IllegalArgumentException.class, () -> FigureUtils.getCircleSquare( a: -1));
}

@Test
public void whenPassCorrectArgumentToTriangleSquare_thenGetCorrectResult() {
    assertEquals( expected: 10.8,  actual: Math.floor(FigureUtils.getTriangleSquare( a: 5,  b: 5,  c: 5) * 10) / 10);
    assertEquals( expected: 0,  actual: Math.floor(FigureUtils.getTriangleSquare( a: 0,  b: 5,  c: 5) * 10) / 10);
}
```

```java
@Test
public void whenPassCorrectArgumentToSquareSquare_thenGetCorrectResult() {
    assertEquals( expected: 25, FigureUtils.getSquareSquare( a: 5));
    assertEquals( expected: 0, FigureUtils.getSquareSquare( a: 0));
}

@Test
public void whenPassCorrectArgumentToCircleSquare_thenGetCorrectResult() {
    assertEquals( expected: 78.5,  actual: Math.floor(FigureUtils.getCircleSquare( a: 5) * 10) / 10);
    assertEquals( expected: 0,  actual: Math.floor(FigureUtils.getCircleSquare( a: 0) * 10) / 10);
}
```

3. Запустимо тести

Як можна бачити, жоден з тестів не пройшов.

4. Використовуючи baby steps додамо логіку до методів

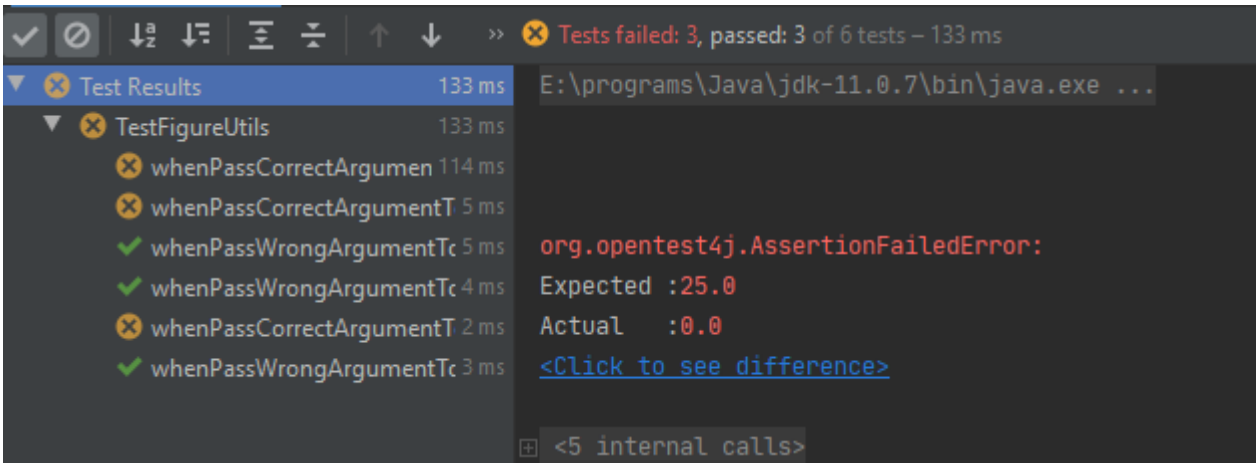
Додамо спочатку перевірку на неправильні значення

```java
public static double getTriangleSquare(int a, int b, int c) {
    if (a < 0 || b < 0 || c < 0) {
        throw new IllegalArgumentException();
    }
    return 0;
}

public static double getSquareSquare(int a) {
    if (a < 0) {
        throw new IllegalArgumentException();
    }
    return 0;
}

public static double getCircleSquare(int a) {
    if (a < 0) {
        throw new IllegalArgumentException();
    }
    return 0;
}
```

Перевіримо тести



Як можна бачити, деякі з тестів пройшли. Додамо логіку розрахунку значень.

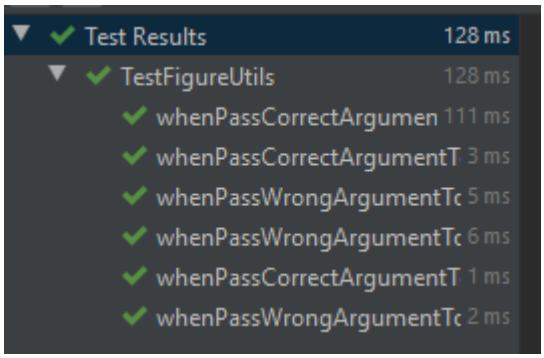
```java
public static double getTriangleSquare(int a, int b, int c) {
    if (a < 0 || b < 0 || c < 0) {
        throw new IllegalArgumentException();
    }
    double p = 1.0 * (a + b + c) / 2;

    return Math.sqrt(p * (p - a) * (p - b) * (p - c));
}

public static double getSquareSquare(int a) {
    if (a < 0) {
        throw new IllegalArgumentException();
    }
    return Math.pow(a, 2);
}

public static double getCircleSquare(int a) {
    if (a < 0) {
        throw new IllegalArgumentException();
    }
    return Math.PI * Math.pow(a, 2);
}
```

Перевіримо тести



Як можна бачити, всі тести пройшли – це означає, що методи написані правильно

5. Проведемо ре факторинг

Оскільки всі методи мають однакову предметну область та виконують однакову функцію, то краще розділити їх на окремі класи.

```java
public interface Figure {
    double getSquare();
}
```

```java
public class Triangle implements Figure{

    private int edgeA;

    private int edgeB;

    private int edgeC;

    public Triangle() {
    }

    @Override
    public double getSquare() {
        if (edgeA < 0 || edgeB < 0 || edgeC < 0) {
            throw new IllegalArgumentException();
        }
        double p = 1.0 * (edgeA + edgeB + edgeC) / 2;

        return Math.sqrt(p * (p - edgeA) * (p - edgeB) * (p - edgeC));
    }
```

```java
public class Square implements Figure{

    private int edge;

    public Square() {
    }

    @Override
    public double getSquare() {
        if (edge < 0) {
            throw new IllegalArgumentException();
        }
        return Math.pow(edge, 2);
    }
```

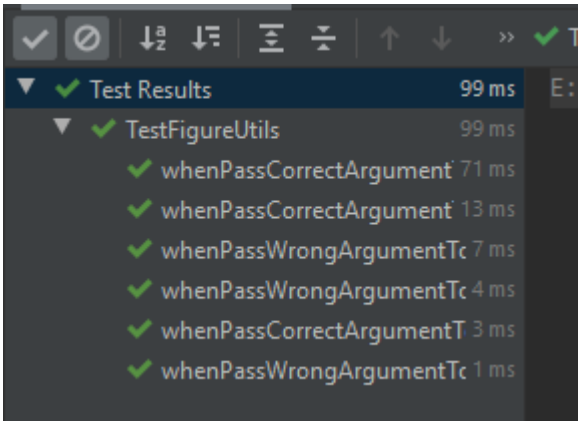
```java
public class Circle implements Figure{

    private int radius;

    public Circle() {
    }

    @Override
    public double getSquare() {
        if (radius < 0) {
            throw new IllegalArgumentException();
        }
        return Math.PI * Math.pow(radius, 2);
    }
```

Перевіримо тести



Як можна бачити, після ре факторингу всі тести проходять, отже він зроблений правильно.

Висновок:

В ході виконання лабораторної роботи було детально досліджено методологію TDD. Написані модульні тести та проведений рефакторинг створеної програми. Ці складові є невід’ємною частиною надійного програмного продукту. Також вони пришвидшують створення ПО та його підтримку в майбутньому.