Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Факультет комп'ютерних наук

ЛАБОРАТОРНА РОБОТА № 4

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

Тема: «Вивчення модульного тестування (unit testing) на прикладі фреймворку JUnit.»

Виконав студент 2 курсу
групи КС-21
Безрук Юрій Русланович
Перевірив:
Доцент Нарєжній О. П.

Харків – 2020

Целью данной работы является изучение фреймворка для модульного тестировани JUnit, сборщиков проектов, написание автоматизированных тестов для программного обеспечения.

ХОД РАБОТЫ

После установки IntelliJ IDEA и скачивания фреймворка JUnit, создаём проект Java и тестируем подключение библиотеки. Для этого, например, можно подключить ее к сборщику проектов Maven.

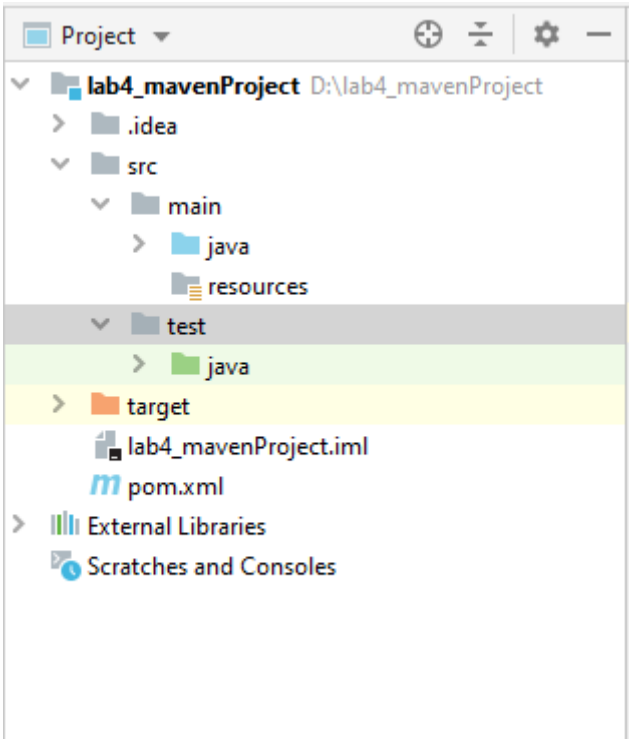

Рисунок 1 - дерево проекта Maven

Что бы подключить JUnit, необходимо подтянуть его в зависимости (те библиотеки, которые непосредственно используются в проекте для компиляции кода или его тестирования). В файле pom.xml прописываем зависимости:

```xml
<?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>org.example</groupId>
    <artifactId>lab4_mavenProject</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build...>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
    </dependencies>
</project>
```

Рисунок 2 - файл pom.xml

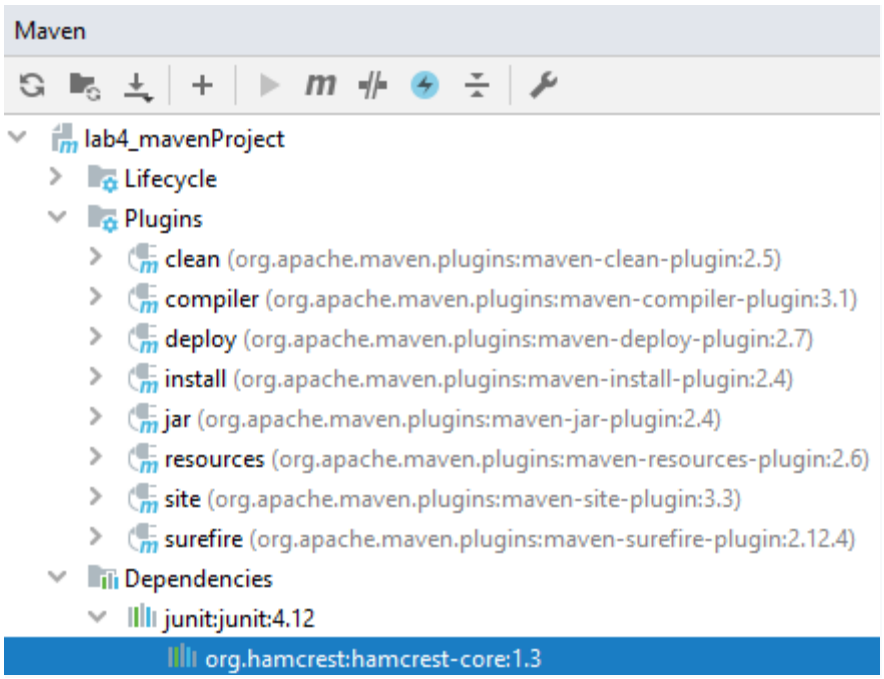

Рисунок 3 - зависимость JUnit

Для проверки возможностей фреймворка запустим пример калькулятора из задания.

```java
port java.util.Scanner;

blic class Calculator {
  public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      Calculator calculator = new Calculator();
      System.out.println("Input number A");
      int A = scanner.nextInt();
      System.out.println("Input 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.Subtraction(A,B));
              break;
          case "*":
              System.out.println(calculator.Multiplication(A,B));
              break;

          case "/":
              System.out.println(calculator.Division(A,B));
              break;
      }
  }
  public int Sum(int A, int B) {
      System.out.println("Result add " + (A + B));
      return (A + B);
  }
  public int Subtraction(int A, int B) {
          System.out.println("Result subtraction " + (A - B));
          return (A - B);
  }
  public int Multiplication(int A, int B) {
          System.out.println("Result Multiplication " + (A * B));
          return (A * B);
  }
  public double Division(int A, int B) {
          System.out.println("Result Division " + (A / B));
          return (A / B);
  }
```

Рисунок 4 - код калькулятора

Для тестирования работы программы напишем тестовый класс.

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

    @Test
    public void testSum() {
        double result = calculator.Sum( A: 3,  B: 2);
        double expectedResult = 5;
        Assert.assertEquals( message: "Fail!", expectedResult, result,  delta: 0);
    }
    @Test
    public void testSub() {
        double result = calculator.Subtraction( A: 3,  B: 2);
        double expectedResult = 1;
        Assert.assertEquals( message: "Fail!", expectedResult, result,  delta: 0);
    }
    @Test
    public void testMul() {
        double result = calculator.Multiplication( A: 3,  B: 2);
        double expectedResult = 6;
        Assert.assertEquals( message: "Fail!", expectedResult, result,  delta: 0);
    }
    @Test
    public void testDiv() {
        double result = calculator.Division( A: 3,  B: 2);
        double expectedResult = 1.5;
        Assert.assertEquals( message: "Fail!", expectedResult, result,  delta: 0);
    }
```

Рисунок 5 - тестовый класс

После запуска тестов обнаруживаем, что все методы прошли тест, кроме деления, т.к. в делении участвовали только целые числа, было выполнено целочисленное деление, поэтому получаем 3/2=1, вместо ожидаемого результата 1,5.

```
"C:\Program Files\Java\jdk1.8.0_201\bin\java.exe" ...
Result Division 1

java.lang.AssertionError: Fail!
Expected :1.5
Actual   :1.0
<Click to see difference>

<1 internal call>
    at org.junit.Assert.failNotEquals(Assert.java:834) <1 internal call>
    at CalculatorTest.testDiv(CalculatorTest.java:31) <19 internal calls>
    at com.intellij.rt.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRunner.java:33)
    at com.intellij.rt.junit.JUnitStarter.prepareStreamsAndStart(JUnitStarter.java:230)
    at com.intellij.rt.junit.JUnitStarter.main(JUnitStarter.java:58)


Process finished with exit code -1
```

Рисунок 6 - результат теста калькулятора

Для выполнения некоторых действий перед вызовом метода-теста или даже класса-теста, в JUnit существуют соответственные аннотации: @Before, @After, @BeforeClass, @AfterClass.

```
@Before
public void before(){
    System.out.println("before");
}
@BeforeClass
static void beforeClass(){
    System.out.println("beforeClass");
}
@After
public void after(){
    System.out.println("after");
}
@AfterClass
static void afterClass(){
    System.out.println("afterClass");
}
```

Рисунок 7 - методы с аннотациями

Результат выполнения теста приведен ниже.

```
beforeClass
before
Result Division 1
after
before
Result Multiplication 6
after
before
Result subtraction 1
after
before
Result add 5
after
afterClass

Process finished with exit code 0
```

Рисунок 8 - результат работы методов с аннотациями

Далее приступаем к выполнению индивидуального задания, чётко сформулированные требования:

Программа должна уметь сортировать массив целых чисел типа int любого размера по возрастанию, в пределах размера целочисленного типа данных. Размер массива и сам массив вводятся с консоли. Тип возвращаемого значения - int[], не должен равняться null. В случае, если массив не отсортирован, после выполнения метода он точно должен измениться.

Программа была реализована через класс со статическими методами.

```java
import java.util.Scanner;

public class ArraySort {
    public static int[] enterArray(){
        java.util.Scanner s = new Scanner(System.in);
        System.out.println("Введите размер массива: ");
        int n = s.nextInt();
        int[] arr = new int[n];
        System.out.println("Введите массив: ");
        for (int i = 0; i<n; i++)
            arr[i] = s.nextInt();
        s.close();
        return arr;
    }
    public static int[] sort(int[] arr) {
        quick(arr, start: 0, end: arr.length - 1);
        return arr;
    }
    public static void main(String[] args) {
        for (int a: sort(enterArray())) {
            System.out.print(a+" ");
        }
    }
```

Рисунок 9 - методы ввода и сортировки

Сама сортировка производилась при помощи алгоритма QuickSort.

```java
private static void quick(int arr[], int start, int end){
    int mid;
    int f=start, l=end;
    mid=arr[(f+l)/2];              //вычисление опорного элемента
    do{
        while(arr[f]<mid)f++;
        while(arr[l]>mid)l--;
        if(f<=l){
            int tmp = arr[f];
            arr[f] = arr[l];//перестановка элементов
            arr[l] = tmp;
            f++;
            l--;
        }
    }while(f<l);
    if(start<l)quick(arr, start, l);
    if(f<end)quick(arr, f, end);
}
```

Рисунок 10 – «быстрая сортировка»

Помимо различных исключений, обрабатываемых Java, у программы может быть несколько семантических проблем. Для тестирования некоторых ситуаций создадим класс ArraySortTest.

Среди его методов-проверок:

Проверка на правильность сортировки. Выполняется при помощи сравнения с встроенным алгоритмом сортировки.

Проверка что возвращаемое значение не null.

Проверка на тип возвращаемого значения при помощи механизмов рефлексии Java.

```
public class ArraySortTest {
    int[] arr = {5, 45, 2, -9, 3};

    @Test
    public void sortTest() {
        int[] result = ArraySort.sort(arr);
        int[] expectedResult = arr;
        Arrays.sort(expectedResult);
        Assert.assertArrayEquals( message: "Problem!", expectedResult, result);
    }
    @Test
    public void nullTest() {
        int[] result = ArraySort.sort(arr);
        Assert.assertNotNull(result);
    }
    @Test
    public void returnTypeTest(){
        Class<?> expected = arr.getClass();
        Class<?> result = ArraySort.sort(arr).getClass();
        Assert.assertSame(expected, result);

    }
```

Рисунок 11 - класс-тест

```
    @Test
    public void changeTest() {
        int[] unexpected = arr.clone();
        int[] result = ArraySort.sort(arr);
        Assert.assertNotSame(unexpected, result);
    }
}
```

Рисунок 12 - класс-тест (продолжение)

В методах в качестве тестируемого массива используют заранее оглашенный, чтобы исключить зависимость от консоли.

В ходе тестирования не было обнаружено проблем с несоответствием требованиям.

ВЫВОДЫ

Итак, модульное тестирование позволяет проводить тесты всей программы или ее отдельных компонентов (units). Такой уровень тестирования предполагает написание автоматизированных тестовых сценариев, которые выявляют дефекты на стадии разработки программного приложения и позволяют разработчику исправить ошибки сразу же без их занесения в базу дефектов. Библиотека JUnit используется для написания модульных тестов, которые повторяются. Она обеспечивает перегрузки методов для всех примитивных типов, объектов и массивов. Используемые параметры: ожидаемое значение (expectedResult) и актуальное значение (result). Возможно также вывод сообщения в случае неудачи теста.
Таким образом, в ходе выполнения лабораторной работы было изучено фрейморк для модульного тестирования JUnit, сборников проектов, написание автоматизированных тестов для программного обеспечения, а также написана программы «Калькулятор» и «Сортировщик»