Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Кубанский государственный технологический университет»

(ФГБОУ ВО «КубГТУ»)

Факультет информационных технологий и кибербезопасности

Кафедра информационных систем и программирования

**ОТЧЕТ**

по лабораторной работе №5.

Тема работы: «Тестирование, управляемое данными и анализ покрытия кода».

Наименование дисциплины: «Тестирование и отладка программного обеспечения».

Подготовил:

Студент группы 23-КБ-ПР1

Юданов М. Д.

Проверил:

Волик А. Г.

Краснодар

**2025**

**Цель работы**: изучить подход к автоматизации процесса тестирования с помощью средств среды разработки Microsoft Visual Studio, а также понятие покрытия кода тестами и научиться использовать на практике средства автоматизации определения покрытия.

**Ход работы**:

1. Копирую код класса из третьей лабораторной работы и класс с наборами тестирований метода проверки принадлежности точки к определенной площади.

**Код класса:**

import scala.math.\_

class CartesianPlane13(val R1: Double, val R2: Double) {

def isBelongToZone(x: Double, y: Double): Boolean = {

val distance = sqrt(x \* x + y \* y)

(x <= 0 && y >= 0 && distance <= R1) ||

(x >= 0 && y <= 0 && distance >= R1 && distance <= R2)

}

}

**Код тестирований:**

import org.scalatest.funsuite.AnyFunSuite

class CartesianPlane13Tests extends AnyFunSuite {

val cartesianPlane = new CartesianPlane13(5, 10)

val testData: Seq[(Double, Double, Boolean)] = Seq(

// Классы эквивалентности

(2, 1, false),

(-3, 3, true),

(-4, 4, false),

(-2, -2, false),

(2, -2, false),

(7, -3.5, true),

(9.5, -5, false),

// Граничные значения

(0, 0, true),

(0, 4.999, true), (0, 5, true), (0, 5.001, false),

(-3, 4, true),

(-4.999, 0, true), (-5, 0, true), (-5.001, 0, false),

(0, -4.999, false), (0, -5, true), (0, -7.5, true), (0, -10, true), (0, -10.001, false),

(3, -4, true), (8, -6, true),

(4.999, 0, false), (5, 0, true), (7.5, 0, true), (10, 0, true), (10.001, 0, false)

)

for (((x, y, expected), index) <- testData.zipWithIndex) {

test(f"Test #${index + 1}: isBelongToZone($x%.3f, $y%.3f) should be $expected") {

val result = cartesianPlane.isBelongToZone(x, y)

assert(result == expected, s"Expected $expected but got $result for (x=$x, y=$y)")

}

}

}2. Запускаю проект тестирования и слежу за результатами. Все тесты пройдены успешно.

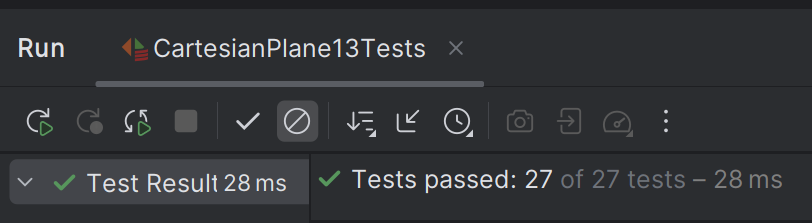


Рисунок 1 – Результаты тестирований.

3. Провожу анализ покрытия кода тестами. Для этого устанавливаю соответствующие расширения, указанные в теоретической части лабораторной работы.

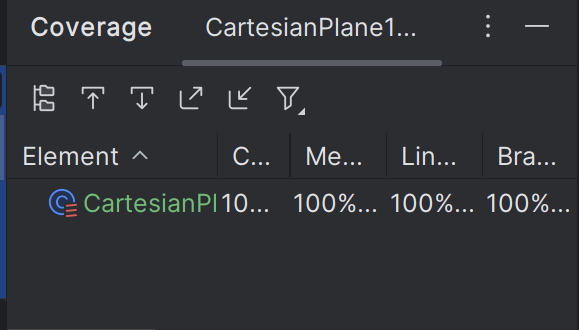


Рисунок 2 – Результаты анализа покрытия кода тестами

4. Провожу точно такой же анализ с классом и тестами из прошлой лабораторной работы.

**Код класса:**

// Вариант 13

class StringReplacer {

def replace(originalString: String, searchString: String, replacementString: String): String = {

// 1: Проверяем, что строка поиска не пуста

if (searchString.isEmpty) return originalString // 2

val result = new StringBuilder // 3

var currentIndex = 0 // 4

// 5: Пока не дошли до конца строки

while (currentIndex < originalString.length) {

val remainingLength = originalString.length - currentIndex // 9

var matchFound = false // 6

if (remainingLength >= searchString.length) { // 10

// 12: Посимвольное сравнение без break

var tempSearchIndex = 0

var allMatch = true

while (tempSearchIndex < searchString.length && allMatch) {

if (originalString.charAt(currentIndex + tempSearchIndex) != searchString.charAt(tempSearchIndex)) {

allMatch = false

}

tempSearchIndex += 1

}

matchFound = allMatch

}

if (matchFound) { // 18

result.append(replacementString) // 19

currentIndex += searchString.length // 20

} else {

result.append(originalString.charAt(currentIndex)) // 21

currentIndex += 1 // 22

}

}

result.toString // 23

}

}

**Код тестирований:**

import org.scalatest.funsuite.AnyFunSuite

class StringReplacerTests extends AnyFunSuite {

var replacer: StringReplacer = \_

def beforeEach(): Unit = {

replacer = new StringReplacer()

}

def afterEach(): Unit = {

replacer = null

}

// MARK: - Граничные случаи и обработка ошибок

test("replace — with empty search string") {

beforeEach()

val original = "Hello"

val search = ""

val replacement = "World"

val result = replacer.replace(original, search, replacement)

assert(result == original, "Должно возвращать оригинальную строку, если строка поиска пуста")

afterEach()

}

test("replace — empty original string") {

beforeEach()

val original = ""

val search = "abc"

val replacement = "xyz"

val result = replacer.replace(original, search, replacement)

assert(result == "", "Должно возвращать пустую строку, если исходная строка пуста")

afterEach()

}

// MARK: - Основные пути выполнения (покрытие веток)

test("replace — no match") {

beforeEach()

val original = "abcdef"

val search = "xyz"

val replacement = "123"

val result = replacer.replace(original, search, replacement)

assert(result == original, "Должно возвращать оригинальную строку, если совпадений нет")

afterEach()

}

test("replace — single match at beginning") {

beforeEach()

val original = "abcde"

val search = "ab"

val replacement = "XX"

val result = replacer.replace(original, search, replacement)

assert(result == "XXcde", "Должно заменить подстроку в начале")

afterEach()

}

test("replace — single match in middle") {

beforeEach()

val original = "axbyc"

val search = "xb"

val replacement = "YY"

val result = replacer.replace(original, search, replacement)

assert(result == "aYYyc", "Должно заменить подстроку в середине")

afterEach()

}

test("replace — single match at end") {

beforeEach()

val original = "abcd"

val search = "cd"

val replacement = "XY"

val result = replacer.replace(original, search, replacement)

assert(result == "abXY", "Должно заменить подстроку в конце")

afterEach()

}

test("replace — multiple non-overlapping matches") {

beforeEach()

val original = "banana"

val search = "an"

val replacement = "!!"

val result = replacer.replace(original, search, replacement)

assert(result == "b!!!!a", "Должно заменить все непересекающиеся вхождения")

afterEach()

}

test("replace — with empty replacement string (deletion)") {

beforeEach()

val original = "applepineapple"

val search = "apple"

val replacement = ""

val result = replacer.replace(original, search, replacement)

assert(result == "pine", "Должно удалять подстроку")

afterEach()

}

}

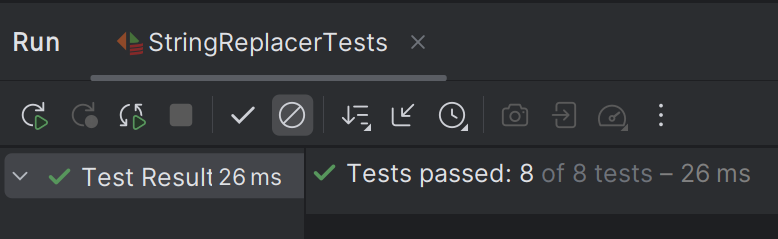


Рисунок 3 – Результаты тестирований

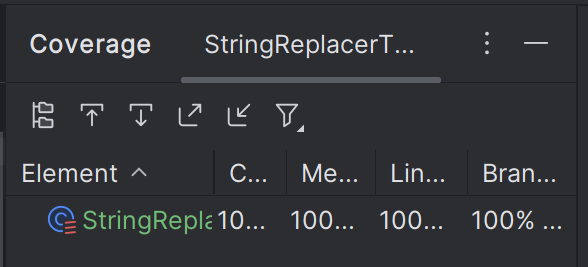


Рисунок 4 – Результаты анализа покрытия кода тестами

5. Делаю вывод касательно проделанной работы.

**Вывод:** в ходе выполнения лабораторной работы я изучил подход к автоматизации процесса тестирования, а также понятие покрытия кода тестами и научиться использовать на практике средства автоматизации определения покрытия.