Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

Информатики и радиоэлектроники»

Учебная дисциплина: «Программирование мобильных информационных систем»

Отчет

по выполнению лабораторной работы «Функциональное программирование и лямбда-выражения»

Выполнил студент: гр. 210101 Войтович Е.А.

Проверил: Усенко Ф.В.

Минск 2024

**1. Цель работы:** ознакомиться с основами функционального программирования в Kotlin, изучить лямбда-выражения, анонимные функции и замыкания. Научиться использовать эти концепции для написания более гибкого и читаемого кода.

**2. Задание:** создание калькулятора с динамическими операциями: Напишите программу, которая позволяет пользователю добавлять новые арифметические операции в виде лямбда-выражений и применять их к выражениям. Программа должна поддерживать сохранение и загрузку этих операций.

**3. Оснащение работы:** техническое задание, технический проект, ЭВМ, import java.io.File

import javax.script.ScriptEngineManager

import javax.script.ScriptException

fun main() {

val operations = mutableMapOf<String, String>()

val fileName = "operations.txt"

loadOperations(fileName, operations)

addDefaultOperations(operations)

while (true) {

println("\n=== Меню ===")

println("1. Добавить новую операцию (add)")

println("2. Выполнить операцию (calc)")

println("3. Сохранить операции (save)")

println("4. Загрузить операции (load)")

println("5. Выход (exit)")

print("Введите номер команды: ")

when (readLine()) {

"1", "add" -> {

println("Введите название новой операции:")

val operationName = readLine() ?: continue

println("Введите лямбда-выражение для новой операции (например, {a: Double, b: Double -> a + 2 \* b}):")

val lambdaExpression = readLine() ?: continue

operations[operationName] = lambdaExpression

println("Операция $operationName добавлена.")

}

"2", "calc" -> {

if (operations.isEmpty()) {

println("Доступные операции отсутствуют.")

} else {

println("Доступные операции:")

for (operation in operations.keys) {

println("- $operation")

}

}

println("Введите название операции:")

val operationName = readLine() ?: continue

println("Введите два числа:")

val a = readLine()?.toDoubleOrNull() ?: continue

val b = readLine()?.toDoubleOrNull() ?: continue

if (operationName == "divide" && b == 0.0) {

println("Деление на 0!")

} else {

val result = evaluateExpression(operations[operationName], a, b)

println("Результат: $result")

}

}

"3", "save" -> {

saveOperations(fileName, operations)

println("Операции сохранены в файл $fileName.")

}

"4", "load" -> {

operations.clear()

loadOperations(fileName, operations)

println("Операции загружены из файла $fileName.")

println("Текущие операции после загрузки: $operations")

}

"5", "exit" -> return

else -> println("Неизвестная команда")

}

}

}

fun evaluateExpression(expression: String?, a: Double, b: Double): Any? {

if (expression == null) {

println("Операция не найдена.")

return null

}

val engine = ScriptEngineManager().getEngineByExtension("kts")

return try {

val script = "$expression(a = $a, b = $b)"

engine.eval("val func = $expression; func($a, $b)")

} catch (e: ScriptException) {

println("Ошибка при выполнении выражения: ${e.message}")

null

}

}

fun saveOperations(fileName: String, operations: Map<String, String>) {

File(fileName).bufferedWriter().use { writer ->

for ((name, expression) in operations) {

writer.write("$name:$expression\n")

}

}

}

fun loadOperations(fileName: String, operations: MutableMap<String, String>) {

val file = File(fileName)

if (!file.exists()) {

println("Файл $fileName не найден.")

return

}

file.forEachLine { line ->

val regex = """(\w+):\s\*(\{.\*\})""".toRegex()

val matchResult = regex.find(line)

if (matchResult != null) {

val operationName = matchResult.groups[1]?.value?.trim()

val lambdaExpression = matchResult.groups[2]?.value?.trim()

if (operationName != null && lambdaExpression != null) {

operations[operationName] = lambdaExpression

println("Загружена операция: $operationName")

}

} else {

println("Некорректная строка в файле: $line")

}

}

}

fun addDefaultOperations(operations: MutableMap<String, String>) {

operations["add"] = "{a: Double, b: Double -> a + b}"

operations["subtract"] = "{a: Double, b: Double -> a - b}"

operations["multiply"] = "{a: Double, b: Double -> a \* b}"

operations["divide"] = "{a: Double, b: Double -> if (b != 0.0) a / b else Double.NaN}"

}

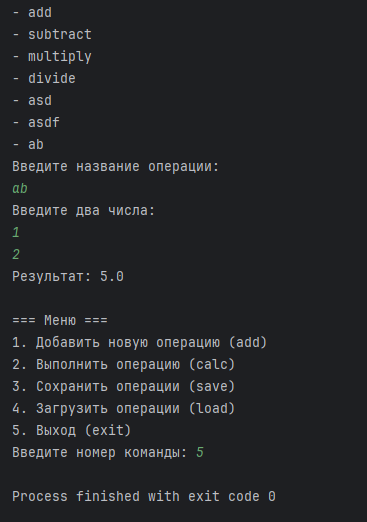


Рисунок 1 – Результат исполнения кода программы

**Вывод:** в ходе выполнения лабораторной работы была изучена концепция функционального программирования на языке Kotlin, включая такие элементы, как лямбда-выражения, анонимные функции и замыкания.