Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Программирование мобильных информационных систем

ОТЧЁТ по лабораторной работе №3 на тему:

**«Функциональное программирование и лямбда-выражения»**

Выполнил: Проверил:

ст. гр. 214301 Усенко Ф.В.

Чехомов Е.Г.

Минск 2024

**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.

1. **Вариант 10. Функция для работы с динамическими структурами данных:** Реализуйте программу, которая использует функции высшего порядка для работы с динамическими структурами данных, такими как очереди и стеки. Программа должна поддерживать разные режимы обработки данных.

# КОД ПРОГРАММЫ

import java.util.LinkedList  
import java.util.Scanner  
  
// Тип для функций обработки очереди  
typealias QueueOperation = (LinkedList<String>) -> Unit  
  
// Тип для функций обработки стека  
typealias StackOperation = (MutableList<String>) -> Unit  
  
fun main() {  
 val scanner = Scanner(System.*`in`*)  
  
 val queue = LinkedList<String>()  
 val stack = mutableListOf<String>()  
  
 while (true) {  
 println("\nВыберите режим работы:")  
 println("1 - Работа с очередью")  
 println("2 - Работа со стеком")  
 println("3 - Выход")  
 when (scanner.nextLine()) {  
 "1" -> {  
 println("\nРежим работы с очередью.")  
 handleQueue(queue, scanner)  
 }  
 "2" -> {  
 println("\nРежим работы со стеком.")  
 handleStack(stack, scanner)  
 }  
 "3" -> {  
 println("Завершение работы.")  
 return  
 }  
 else -> {  
 println("Некорректный выбор, попробуйте снова.")  
 }  
 }  
 }  
}  
  
// Функция для обработки операций над очередью  
fun handleQueue(queue: LinkedList<String>, scanner: Scanner) {  
 while (true) {  
 println("\nТекущая очередь: $queue")  
 println("1 - Добавить элемент")  
 println("2 - Удалить элемент")  
 println("3 - Назад")  
 when (scanner.nextLine()) {  
 "1" -> {  
 println("Введите элемент для добавления в очередь:")  
 val element = scanner.nextLine()  
 modifyQueue(queue) **{** q **->** q.add(element) **}** println("Элемент добавлен в очередь.")  
 }  
 "2" -> {  
 if (queue.isEmpty()) {  
 println("Очередь пуста, нечего удалять.")  
 } else {  
 modifyQueue(queue) **{** q **->** q.poll() **}** println("Первый элемент удален из очереди.")  
 }  
 }  
 "3" -> return  
 else -> println("Некорректный выбор.")  
 }  
 }  
}  
  
// Функция для обработки операций над стеком  
fun handleStack(stack: MutableList<String>, scanner: Scanner) {  
 while (true) {  
 println("\nТекущий стек: $stack")  
 println("1 - Добавить элемент")  
 println("2 - Удалить элемент")  
 println("3 - Назад")  
 when (scanner.nextLine()) {  
 "1" -> {  
 println("Введите элемент для добавления в стек:")  
 val element = scanner.nextLine()  
 modifyStack(stack) **{** s **->** s.add(element) **}** println("Элемент добавлен в стек.")  
 }  
 "2" -> {  
 if (stack.isEmpty()) {  
 println("Стек пуст, нечего удалять.")  
 } else {  
 modifyStack(stack) **{** s **->** s.removeLast() **}** println("Последний элемент удален из стека.")  
 }  
 }  
 "3" -> return  
 else -> println("Некорректный выбор.")  
 }  
 }  
}  
  
// Функция высшего порядка для работы с очередью  
fun modifyQueue(queue: LinkedList<String>, operation: QueueOperation) {  
 operation(queue)  
}  
  
// Функция высшего порядка для работы со стеком  
fun modifyStack(stack: MutableList<String>, operation: StackOperation) {  
 operation(stack)  
}

# ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как объявить функцию в Kotlin? В чем разница между обычной функцией и однострочной функцией?

fun greet(name: String) = "Hello, $name!"

Разница:

Однострочная функция сокращает синтаксис за счёт автоматического определения возвращаемого типа (если он может быть выведен).

В обычной функции требуется явно использовать return, если есть тело блока { }.

1. Что такое функция высшего порядка, и как её использовать? Приведите пример.

Функция высшего порядка принимает другую функцию в качестве аргумента или возвращает функцию как результат.

**Пример:**

fun operateOnNumbers(a: Int, b: Int, operation: (Int, Int) -> Int): Int {

return operation(a, b)

}

fun add(x: Int, y: Int) = x + y

fun main() {

val result = operateOnNumbers(5, 3, ::add)

println(result) // 8

}

1. Как передать функцию в качестве параметра другой функции? Приведите пример.

Можно передать функцию в качестве параметра, используя ссылку на неё (::) или лямбду.

**Пример:**

fun multiply(a: Int, b: Int) = a \* b

fun calculate(a: Int, b: Int, operation: (Int, Int) -> Int): Int {

return operation(a, b)

}

fun main() {

val result = calculate(4, 6, ::multiply) // Передаём функцию multiply

println(result) // 24

}

1. Чем отличается лямбда-выражение от обычной функции?

**Различия:**

Лямбда-выражение — это анонимная функция, которая не имеет имени.

Лямбда-выражения обычно используются для краткости при передаче функций как параметров.

Лямбда-выражение имеет более компактный синтаксис: параметры указываются перед ->.

1. Как создать лямбда-выражение с двумя параметрами? Приведите пример.

val calculate = { x: Int, y: Int -> x \* y }

fun main() {

val result = calculate(7, 8)

println(result) // 56

}

# ВЫВОД

В ходы выполнения лабораторной работы было проведено ознакомление с основами функционального программирования в Kotlin, изучены лямбда-выражения, анонимные функции и замыкания.