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:

ст. гр. 214302

Кранцевич В.С.

Проверил:

Усенко Ф. В.

Минск 2024

**Цель**: 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.

**Вариант 17**: Функция с динамическим выбором стратегии обработки: Напишите программу, которая позволяет пользователю выбирать стратегии обработки данных на лету (например, сортировка, фильтрация, агрегация) и применять их к спискам с использованием лямбда-выражений.

**КОД ПРОГРАММЫ**

fun main() {

val numbers = listOf(3, 7, 1, 9, 2, 6, 5)

println("""Выберите стратегию обработки:

|1. Сортировка

|2. Фильтрация > 5

|3. Агрегация (Сумма)

|4. Агрегация (Среднее)

""".trimMargin())

val choice = readLine()?.toIntOrNull()

val result = when (choice) {

1 -> applyStrategy(numbers) { it.sorted() }

2 -> applyStrategy(numbers) { it.filter { num -> num > 5 } }

3 -> applyStrategy(numbers) { listOf(it.sum()) } // обе агрегация

4 -> applyStrategy(numbers) { listOf(it.average().toInt()) }

else -> {

println("Неверный выбор")

return

}

}

println("Результат обработки: $result")

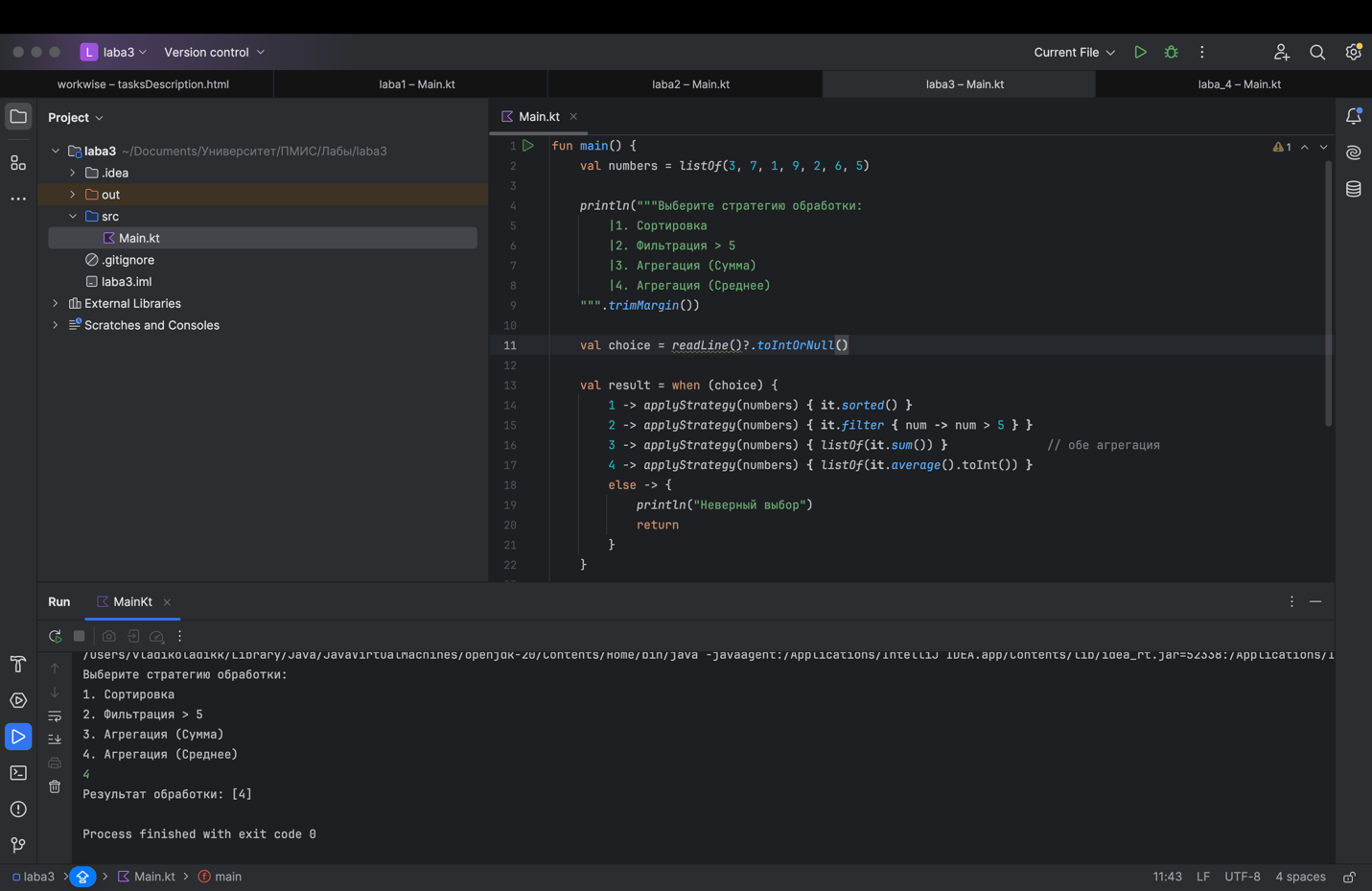
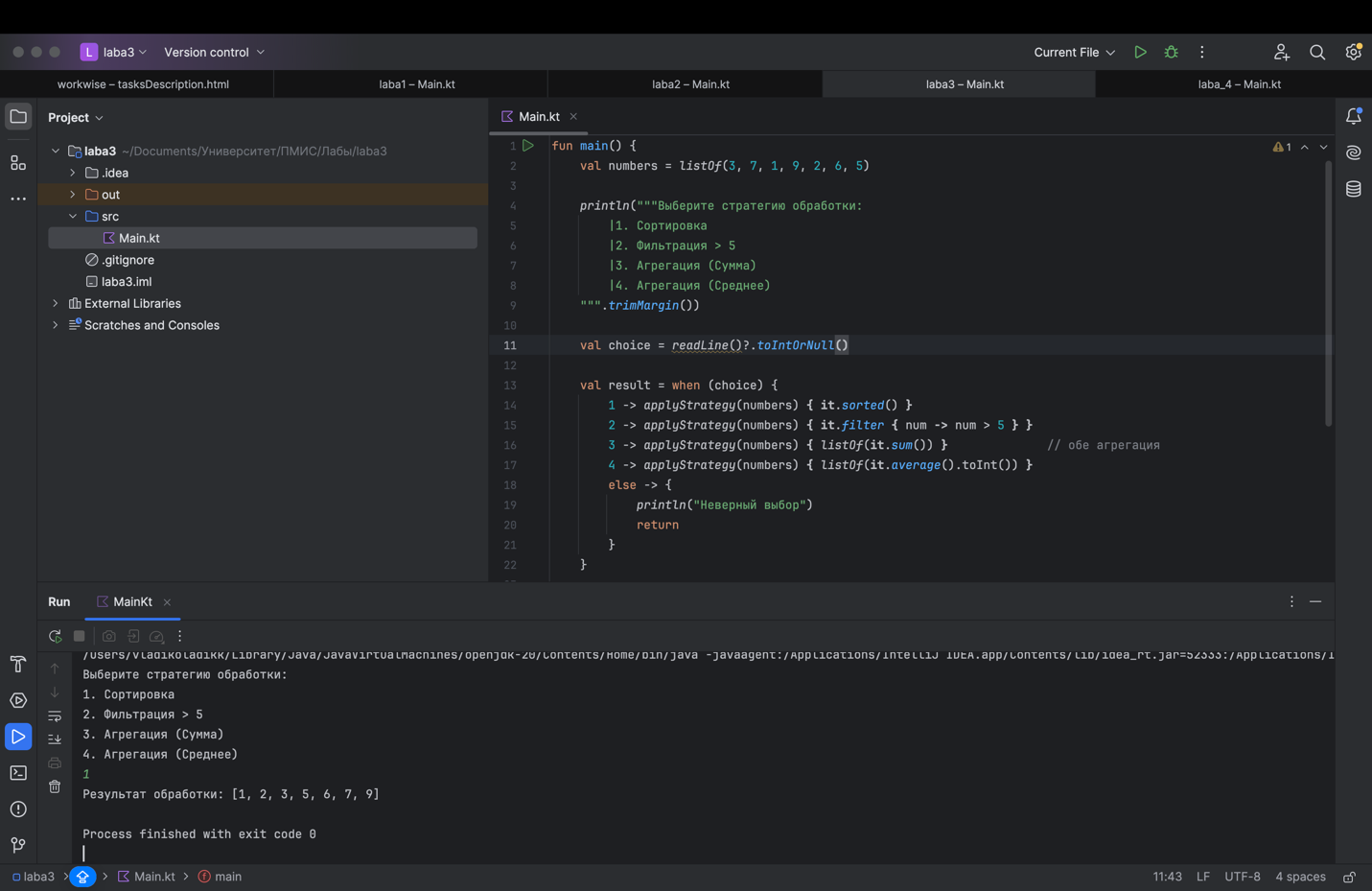
}

fun applyStrategy(data: List<Int>, strategy: (List<Int>) -> List<Int>): List<Int> {

return strategy(data)

}

**СКРИНШОТЫ РАБОЧИХ ОКОН**

****

**ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Как объявить функцию в Kotlin? В чем разница между обычной функцией и однострочной функцией?  
   Функция объявляется с использованием ключевого слова fun:

fun add(a: Int, b: Int): Int {

return a + b

}

Однострочная функция (single-expression) возвращает результат без явного return и фигурных скобок:

fun add(a: Int, b: Int): Int = a + b

1. Что такое функция высшего порядка, и как её использовать? Приведите пример.  
   Функция высшего порядка принимает другую функцию в качестве аргумента или возвращает её.

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

return operation(a, b)

}

val result = operate(4, 2) { x, y -> x + y }

println(result) // 6

1. Как передать функцию в качестве параметра другой функции? Приведите пример.  
   Функцию можно передать, используя ссылку на неё или лямбду:

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

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

return operation(a, b)

}

println(applyOperation(3, 4, ::multiply)) // 12

1. Чем отличается лямбда-выражение от обычной функции?  
   Лямбда-выражение — это анонимная функция, не требующая имени и обычно используется для коротких операций. Пример лямбды:

val sum = { a: Int, b: Int -> a + b }

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

val add = { a: Int, b: Int -> a + b }

println(add(3, 5)) // 8

1. Что такое анонимная функция, и как она отличается от лямбда-выражения?  
   Анонимная функция — это функция без имени, объявленная с использованием fun:

val multiply = fun(a: Int, b: Int): Int {

return a \* b

}

В отличие от лямбд, анонимные функции могут иметь явное возвращаемое значение.

1. Какие возможности предоставляют анонимные функции в Kotlin?

* Более точный контроль над возвращаемым значением.
* Возможность использовать вместо лямбды, если требуется сложная логика.

1. Что такое замыкание, и как оно работает? Приведите пример использования.  
   Замыкание — это функция, которая захватывает переменные из внешней области видимости:

fun makeCounter(): () -> Int {

var count = 0

return { ++count }

}

val counter = makeCounter()

println(counter()) // 1

println(counter()) // 2

1. Как лямбда-выражения могут захватывать переменные из внешней области видимости?  
   Лямбды автоматически захватывают переменные, к которым обращаются:

var total = 0

val add = { value: Int -> total += value }

add(5)

println(total) // 5

1. В чем преимущества использования функций высшего порядка в программировании?

* Повышение гибкости и повторного использования кода.
* Упрощение работы с коллекциями.
* Возможность параметризации поведения функций.

1. Как реализовать функцию map, используя лямбда-выражения? Приведите пример.

val numbers = listOf(1, 2, 3)

val squares = numbers.map { it \* it }

println(squares) // [1, 4, 9]

1. Как функция reduce работает с массивами? Приведите пример использования.  
   Функция reduce сворачивает массив в одно значение:

val numbers = listOf(1, 2, 3, 4)

val sum = numbers.reduce { acc, num -> acc + num }

println(sum) // 10

1. Как работает функция filter, и как её использовать для фильтрации списка строк?  
   filter возвращает элементы, соответствующие условию:

val words = listOf("Kotlin", "Java", "Python")

val filtered = words.filter { it.startsWith("K") }

println(filtered) // [Kotlin]

1. Как реализовать функцию forEach для выполнения действий над каждым элементом массива?

val numbers = listOf(1, 2, 3)

numbers.forEach { println(it) }

1. Что такое функция partition, и как она может быть полезна?  
   partition разделяет коллекцию на две, основываясь на предикате:

val numbers = listOf(1, 2, 3, 4, 5)

val (even, odd) = numbers.partition { it % 2 == 0 }

println(even) // [2, 4]

println(odd) // [1, 3, 5]

1. Как использовать лямбда-выражения для сортировки списка по произвольному критерию?

val people = listOf("John", "Alice", "Bob")

val sorted = people.sortedBy { it.length }

println(sorted) // [Bob, John, Alice]

1. Как создать замыкание, которое увеличивает значение переменной-счетчика при каждом вызове?

fun counter(): () -> Int {

var count = 0

return { ++count }

}

val myCounter = counter()

println(myCounter()) // 1

println(myCounter()) // 2

1. Как в Kotlin реализовать обработку ошибок при помощи анонимных функций?  
   Анонимные функции могут быть использованы внутри try...catch:

val safeDivide = fun(a: Int, b: Int): Int? {

return try {

a / b

} catch (e: ArithmeticException) {

null

}

}

println(safeDivide(10, 0)) // null

1. Чем полезны функции высшего порядка в создании гибких и расширяемых программ?

* Они позволяют изменять поведение функции без её переписывания.
* Упрощают создание универсальных алгоритмов.
* Снижают количество дублирующегося кода.

1. Как использовать лямбда-выражения для форматирования строк? Приведите пример.

val format: (String, Int) -> String = { name, age -> "Имя: $name, Возраст: $age" }

println(format("Алексей", 30)) // Имя: Алексей, Возраст: 30

**ВЫВОД**

В ходе лабораторной работы были изучены основы функционального программирования в *Kotlin*, лямбда-выражения, анонимные функции и замыкания, использованы эти концепции для написания более гибкого и читаемого кода.