|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа,**

**обработки и интерпретации больших данных.**

**Отчет**

**по лабораторной работе № 5**

**Вариант № 6**

**Название:** Исключения и файлы

**Дисциплина:** Языки программирования для работы с большими данными

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | Д.А. Залимханов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В. Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2023

**Цель:** изучить и освоить работу с исключениями и файлами в Kotlin.

**Задание 1:** выполнить задания на основе варианта 1 лабораторной работы 3, контролируя состояние потоков ввода/вывода. При возникновении ошибок, связанных с корректностью выполнения математических операций, генерировать и обрабатывать исключительные ситуации. Предусмотреть обработку исключений, возникающих при нехватке памяти, отсутствии требуемой записи (объекта) в файле, недопустимом значении поля и т.д.

Код класса ContinuedFraction:

class ContinuedFraction(private val coefficients: List<Double>, private var x: Double) {

fun evaluate(n: Int): Double {

if (n < 0 || n >= coefficients.size)

throw IndexOutOfBoundsException("Index $n is out of bounds for list of size ${coefficients.size}")

var value = coefficients.getOrElse(n) { throw IllegalArgumentException("Coefficient for index $n is not provided") }

for (i in n - 1 downTo 0) {

if (value == 0.0)

throw ArithmeticException("Division by zero encountered during evaluation")

value = coefficients.getOrElse(i) { throw IllegalArgumentException("Coefficient for index $i is not provided") } + x / value

}

return value

}

operator fun plus(other: ContinuedFraction): Double {

return this.evaluate(coefficients.size - 1) + other.evaluate(other.coefficients.size - 1)

}

operator fun minus(other: ContinuedFraction): Double {

return this.evaluate(coefficients.size - 1) - other.evaluate(other.coefficients.size - 1)

}

operator fun times(other: ContinuedFraction): Double {

return this.evaluate(coefficients.size - 1) \* other.evaluate(other.coefficients.size - 1)

}

operator fun div(other: ContinuedFraction): Double {

val otherValue = other.evaluate(other.coefficients.size - 1)

if (otherValue == 0.0)

throw ArithmeticException("Attempt to divide by zero")

return this.evaluate(coefficients.size - 1) / otherValue

}

}

Код функции Main:

fun main() {

val scanner = Scanner(System.`in`)

try {

println("Введите количество констант первой дроби")

val n1 = scanner.nextInt()

println("Введите константы первой дроби")

val coefficientsFirst = List(n1) { scanner.nextDouble() }

println("Введите количество констант второй дроби")

val n2 = scanner.nextInt()

println("Введите константы второй дроби")

val coefficientsSecond = List(n2) { scanner.nextDouble() }

val fractionFirst = ContinuedFraction(coefficientsFirst, 5.0)

val fractionSecond = ContinuedFraction(coefficientsSecond, 5.0)

println("Введите n")

val n = scanner.nextInt()

println("Введите x")

val x = scanner.nextDouble()

println("Значение первой дроби при x = $x и n = $n: ${fractionFirst.evaluate(n)}")

println("Значение второй дроби при x = $x и n = $n: ${fractionSecond.evaluate(n)}")

println("Сложение дробей: ${fractionFirst + fractionSecond}")

println("Вычитание дробей: ${fractionFirst - fractionSecond}")

println("Умножение дробей: ${fractionFirst \* fractionSecond}")

println("Деление дробей: ${fractionFirst / fractionSecond}")

} catch (e: InputMismatchException) {

println("Ошибка ввода. Пожалуйста, введите корректные данные.")

} catch (e: IndexOutOfBoundsException) {

println("Ошибка: ${e.message}")

} catch (e: ArithmeticException) {

println("Математическая ошибка: ${e.message}")

} catch (e: Exception) {

println("Произошла ошибка: ${e.localizedMessage}")

} finally {

scanner.close()

}

}

Результат работы программы представлен на рисунке 1.

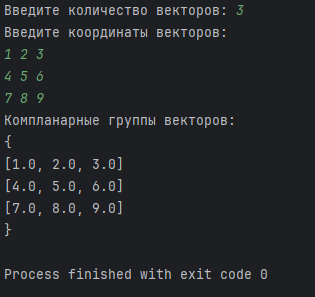


Рисунок 1 – Результат работы программы

**Задание 2:** выполнить задания на основе варианта 1 лабораторной работы 3, контролируя состояние потоков ввода/вывода. При возникновении ошибок, связанных с корректностью выполнения математических операций, генерировать и обрабатывать исключительные ситуации. Предусмотреть обработку исключений, возникающих при нехватке памяти, отсутствии требуемой записи (объекта) в файле, недопустимом значении поля и т.д.

Код класса Fraction:

data class Fraction(val numerator: Int, val denominator: Int) {

init {

require(denominator != 0) { "Знаменатель не может быть равен 0" }

}

operator fun plus(other: Fraction): Fraction {

val commonDenominator = this.denominator \* other.denominator

val newNumerator = this.numerator \* other.denominator + other.numerator \* this.denominator

return Fraction(newNumerator, commonDenominator).simplify()

}

operator fun minus(other: Fraction): Fraction {

val commonDenominator = this.denominator \* other.denominator

val newNumerator = this.numerator \* other.denominator - other.numerator \* this.denominator

return Fraction(newNumerator, commonDenominator).simplify()

}

operator fun times(other: Fraction): Fraction {

return Fraction(this.numerator \* other.numerator, this.denominator \* other.denominator).simplify()

}

operator fun div(other: Fraction): Fraction {

if (other.numerator == 0) {

throw ArithmeticException("Деление на дробь с нулевым числителем невозможно")

}

return Fraction(this.numerator \* other.denominator, this.denominator \* other.numerator).simplify()

}

private fun simplify(): Fraction {

val gcd = gcd(this.numerator, this.denominator)

return Fraction(this.numerator / gcd, this.denominator / gcd)

}

private fun gcd(a: Int, b: Int): Int {

return if (b == 0) a else gcd(b, a % b)

}

override fun toString(): String = "$numerator/$denominator"

}

fun modifyEvenIndexedFractions(fractions: Array<Fraction>) {

for (i in fractions.indices step 2) {

if (i + 1 < fractions.size) {

fractions[i] = fractions[i] + fractions[i + 1]

}

}

}

Код функции Main:

fun main() {

val scanner = java.util.Scanner(System.`in`)

try {

println("Введите количество дробей в массиве:")

val k = scanner.nextInt()

val fractions = Array(k) {

println("Введите числитель и знаменатель для дроби ${it + 1}:")

val numerator = scanner.nextInt()

val denominator = scanner.nextInt()

Fraction(numerator, denominator)

}

println("Исходный массив дробей: ${fractions.joinToString()}")

modifyEvenIndexedFractions(fractions)

println("Модифицированный массив дробей: ${fractions.joinToString()}")

if (fractions.size > 1) {

println("Сложение первых двух дробей модифицированного массива: ${fractions[0] + fractions[1]}")

println("Вычитание первых двух дробей модифицированного массива: ${fractions[0] - fractions[1]}")

println("Умножение первых двух дробей модифицированного массива: ${fractions[0] \* fractions[1]}")

println("Деление первых двух дробей модифицированного массива: ${fractions[0] / fractions[1]}")

}

} catch (e: ArithmeticException) {

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

} catch (e: InputMismatchException) {

println("Некорректный ввод. Пожалуйста, введите целые числа.")

} catch (e: Exception) {

println("Произошла ошибка: ${e.localizedMessage}")

} finally {

scanner.close()

}

}

Результат работы программы показан на рисунке 2.

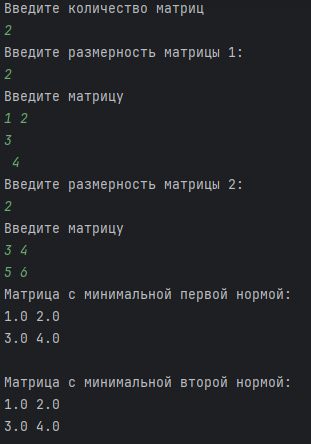


Рисунок 2 – Результат работы программы

**Задание 3:** выполнить задания из варианта 2 лабораторной работы 3, реализуя собственные обработчики исключений и исключения ввода/вывода.

Код классов House и Houses:

data class House(

var id: Int,

var apartmentNumber: Int,

var area: Double,

var floor: Int,

var numberOfRooms: Int,

var street: String,

var buildingType: String,

var operationTerm: Int

) {

override fun toString(): String {

return "House(id=$id, apartmentNumber=$apartmentNumber, area=$area, floor=$floor, " +

"numberOfRooms=$numberOfRooms, street='$street', buildingType='$buildingType', " +

"operationTerm=$operationTerm)"

}

}

class Houses(val houses: Array<House>) {

fun listApartmentsWithRooms(rooms: Int) {

println("Список квартир с заданным числом комнат:")

houses.filter { it.numberOfRooms == rooms }

.forEach { println(it) }

}

fun listApartmentsWithRoomsAndFloor(rooms: Int, floorRange: IntRange) {

println("Список квартир с заданным числом комнат и этажом в промежутке $floorRange:")

houses.filter { it.numberOfRooms == rooms && it.floor in floorRange }

.forEach { println(it) }

}

fun listApartmentsWithAreaGreaterThan(area: Double) {

println("Список квартир с площадью превышающей $area:")

houses.filter { it.area > area }

.forEach { println(it) }

}

}

fun readIntSafe(scanner: Scanner): Int {

if (!scanner.hasNextInt()) throw InputMismatchException("Ожидалось целое число")

return scanner.nextInt()

}

fun readDoubleSafe(scanner: Scanner): Double {

if (!scanner.hasNextDouble()) throw InputMismatchException("Ожидалось число с плавающей точкой")

return scanner.nextDouble()

}

Код функции Main:

fun main() {

val scanner = Scanner(System.`in`)

try {

val houses = Houses(arrayOf(

House(1, 12, 55.0, 3, 2, "Green Street", "Apartment", 5),

House(2, 45, 68.0, 6, 3, "Sunset Boulevard", "Townhouse", 10),

House(3, 78, 42.0, 1, 1, "Oak Street", "Apartment", 2),

House(4, 10, 120.0, 9, 4, "Elm Street", "Penthouse", 15),

House(5, 2, 85.0, 5, 3, "Maple Avenue", "Condominium", 8)

))

println("Введите число комнат:")

val roomsNum = readIntSafe(scanner)

println("Введите нижнюю границу диапазона этажей:")

val floorRangeFrom = readIntSafe(scanner)

println("Введите верхнюю границу диапазона этажей:")

val floorRangeTo = readIntSafe(scanner)

val floorRange = floorRangeFrom..floorRangeTo

println("Введите минимальную площадь:")

val area = readDoubleSafe(scanner)

houses.listApartmentsWithRooms(roomsNum)

houses.listApartmentsWithRoomsAndFloor(roomsNum, floorRange)

houses.listApartmentsWithAreaGreaterThan(area)

} catch (e: InputMismatchException) {

println("Ошибка ввода. Пожалуйста, введите корректные данные.")

} catch (e: Exception) {

println("Произошла ошибка: ${e.message}")

} finally {

scanner.close()

}

}

Результат работы программы представлен на рисунке 3.

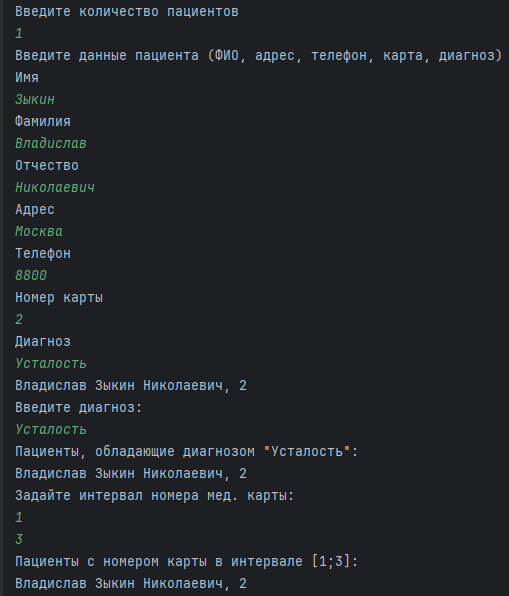
****

Рисунок 3 – Результат работы программы

**Задание 4:** выполнить задания из варианта 2 лабораторной работы 3, реализуя собственные обработчики исключений и исключения ввода/вывода.

Код интерфейса Abiturient:

data class Phone(

var id: Int,

var lastName: String,

var firstName: String,

var middleName: String,

var address: String,

var creditCardNumber: String,

var debit: Double,

var credit: Double,

var cityCallTime: Int,

var intercityCallTime: Int

) {

override fun toString(): String = """

Phone(id=$id, LastName='$lastName', FirstName='$firstName', MiddleName='$middleName',

Address='$address', CreditCardNumber='$creditCardNumber', Debit=$debit,

Credit=$credit, CityCallTime=$cityCallTime min, IntercityCallTime=$intercityCallTime min)

""".trimIndent()

}

class Phones(private val phones: Array<Phone>) {

fun listSubscribersWithHighCityCallTime(minCityCallTime: Int) {

println("\nСведения об абонентах с временем внутригородских разговоров выше $minCityCallTime минут:")

phones.filter { it.cityCallTime > minCityCallTime }

.forEach { println(it) }

}

fun listSubscribersWithIntercityCalls() {

println("\nСведения об абонентах, которые пользовались междугородной связью:")

phones.filter { it.intercityCallTime > 0 }

.forEach { println(it) }

}

fun listSubscribersAlphabetically() {

println("\nСведения об абонентах в алфавитном порядке:")

phones.sortedWith(compareBy { it.lastName + it.firstName + it.middleName })

.forEach { println(it) }

}

}

Код функции Main:

fun main() {

val scanner = Scanner(System.`in`)

val phonesArray = arrayOf(

Phone(1, "Ivanov", "Ivan", "Ivanovich", "Some Address 1", "1111222233334444", 100.0, 50.0, 60, 30),

Phone(2, "Petrov", "Petr", "Petrovich", "Some Address 2", "5555666677778888", 200.0, 40.0, 40, 50),

Phone(3, "Sidorov", "Sidr", "Sidrovich", "Some Address 3", "2222333344445555", 300.0, 60.0, 30, 10),

Phone(4, "Smirnov", "Smir", "Smirnovich", "Some Address 4", "6666777788889999", 400.0, 80.0, 20, 20),

Phone(5, "Kuznetsov", "Kuzma", "Kuzmich", "Some Address 5", "7777888899990000", 500.0, 100.0, 70, 40)

)

val phones = Phones(phonesArray)

try {

println("Введите минимальное время внутригородских разговоров для фильтрации:")

val minCityCallTime = readIntSafe(scanner)

phones.listSubscribersWithHighCityCallTime(minCityCallTime)

println("Абоненты, использовавшие междугородную связь:")

phones.listSubscribersWithIntercityCalls()

println("Абоненты в алфавитном порядке:")

phones.listSubscribersAlphabetically()

} catch (e: InputMismatchException) {

println("Ошибка ввода. Убедитесь, что вводите корректные числовые значения.")

} catch (e: Exception) {

println("Произошла ошибка: ${e.localizedMessage}")

} finally {

scanner.close()

}

}

Результат работы программы показан на рисунке 4.

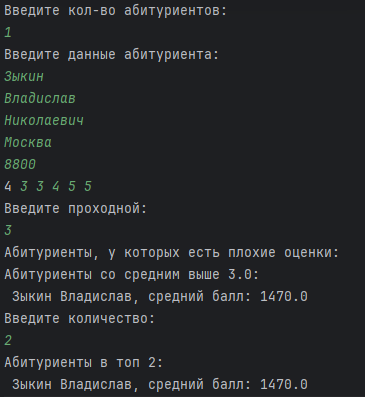


Рисунок 4 – Результат работы программы

**Задание 5:** В следующих заданиях требуется ввести последовательность строк из текстового потока и выполнить указанные действия. При этом могут рассматриваться два варианта:

• каждая строка состоит из одного слова;

• каждая строка состоит из нескольких слов.

В каждой строке стихотворения Анны Ахматовой подсчитать частоту повторяемости каждого слова из заданного списка и вывести эти слова в порядке возрастания частоты повторяемости.

Код функции Main:

fun main(args: Array<String>) {

if (args.size < 3) {

println("Usage: <input file> <output file> <words to count separated by commas>")

return

}

val inputFile = args[0]

val outputFile = args[1]

val wordsToCount = args[2].split(",").map { it.trim().lowercase() }.toSet()

println(wordsToCount)

val wordCounts = mutableMapOf<String, Int>()

try {

File(inputFile).useLines { lines ->

lines.forEach { line ->

line.lowercase().split("[^\\p{L}]+".toRegex()).forEach { word ->

if (word in wordsToCount) {

wordCounts[word] = wordCounts.getOrDefault(word, 0) + 1

}

}

}

}

println(wordCounts)

val sortedWords = wordCounts.toList().sortedBy { (\_, value) -> value }.toMap()

File(outputFile).printWriter().use { out ->

sortedWords.forEach { (word, count) ->

out.println("$word: $count")

}

}

println("Word counts written to $outputFile")

} catch (e: IOException) {

println("Error reading from $inputFile or writing to $outputFile: ${e.message}")

} catch (e: Exception) {

println("An error occurred: ${e.message}")

}

}

Результат работы программы показан на рисунке 5.

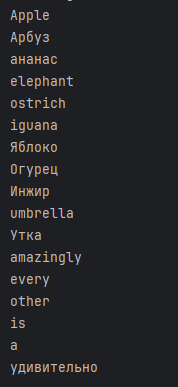


Рисунок 5 – Результаты работы программы

**Задание 6:** в следующих заданиях требуется ввести последовательность строк из текстового потока и выполнить указанные действия. При этом могут рассматриваться два варианта:

• каждая строка состоит из одного слова;

• каждая строка состоит из нескольких слов.

В каждом слове стихотворения Николая Заболоцкого заменить первую букву слова на прописную.

Код класса Main:

fun main(args: Array<String>) {

if (args.size < 2) {

println("Usage: <input file> <output file>")

return

}

val inputFile = args[0]

val outputFile = args[1]

try {

File(outputFile).printWriter().use { out ->

File(inputFile).useLines { lines ->

lines.forEach { line ->

out.println(line.split("\\s+".toRegex()).joinToString(" ") { word ->

word.replaceFirstChar { if (it.isLowerCase()) it.titlecase() else it.toString() }

})

}

}

}

println("Processed lines written to $outputFile")

} catch (e: IOException) {

println("Error reading from $inputFile or writing to $outputFile: ${e.message}")

} catch (e: Exception) {

println("An error occurred: ${e.message}")

}

}

Результат работы программы представлен на рисунке 6.

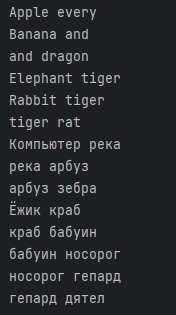


Рисунок 6 – Результат работы программы

**Задание 7**: при выполнении следующих заданий для вывода результатов создавать новую директорию и файл средствами класса File.

Из файла удалить все слова, содержащие от трех до пяти символов, но при этом из каждой строки должно быть удалено только максимальное четное количество таких слов.

Код решения задачи:

fun main(args: Array<String>) {

if (args.size < 2) {

println("Usage: <input file> <output directory>")

return

}

val inputFile = File(args[0])

val outputDir = File(args[1])

if (!outputDir.exists()) {

outputDir.mkdirs() // Создаем директорию, если она не существует

}

val outputFile = File(outputDir, "filtered\_text.txt")

try {

inputFile.bufferedReader().use { reader ->

outputFile.printWriter().use { writer ->

reader.forEachLine { line ->

writer.println(removeWords(line))

}

}

}

println("Processed text written to ${outputFile.absolutePath}")

} catch (e: IOException) {

println("Error handling file: ${e.message}")

} catch (e: Exception) {

println("An error occurred: ${e.message}")

}

}

fun removeWords(line: String): String {

val words = line.split("\\s+".toRegex())

val toRemove = words.filter { it.length in 3..5 }

val maxEvenCount = toRemove.size - toRemove.size % 2 // Находим максимальное четное количество

var newWords = mutableListOf<String>()

var removedCount = 0

for (word in words) {

if (word.length in 3..5 && removedCount < maxEvenCount) {

removedCount++

} else {

newWords.add(word)

}

}

return newWords.joinToString(" ")

}

Результаты работы программы представлены на рисунке 7.

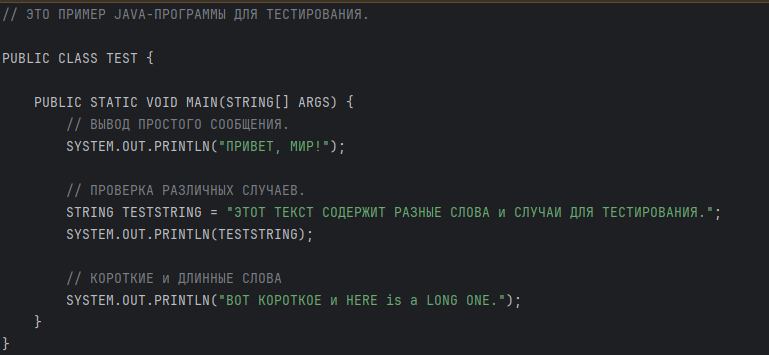


Рисунок 7 – Результаты работы программы

**Задание 8**: при выполнении следующих заданий для вывода результатов создавать новую директорию и файл средствами класса File.

Прочитать текст Java-программы и удалить из него все “лишние” пробелы и табуляции, оставив только необходимые для разделения операторов.

Код решения задачи:

fun main(args: Array<String>) {

if (args.size < 2) {

println("Usage: <input file> <output directory>")

return

}

val inputFile = File(args[0])

val outputDir = File(args[1])

if (!outputDir.exists()) {

outputDir.mkdirs()

}

val outputFile = File(outputDir, "formatted\_code.txt")

try {

inputFile.bufferedReader().use { reader ->

outputFile.printWriter().use { writer ->

reader.forEachLine { line ->

writer.println(formatJavaCode(line))

}

}

}

println("Formatted code written to ${outputFile.absolutePath}")

} catch (e: IOException) {

println("Error reading from ${inputFile.path} or writing to ${outputFile.path}: ${e.message}")

} catch (e: Exception) {

println("An error occurred: ${e.message}")

}

}

fun formatJavaCode(line: String): String {

return line.replace("\\s+".toRegex(), " ")

.replace("\\s\*\\{\\s\*".toRegex(), " { ")

.replace("\\s\*;\\s\*".toRegex(), "; ")

.trim()

}

Результат работы программы показан на рисунке 8.

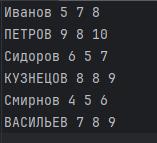


Рисунок 8 – Результат работы программы

**Вывод:** была освоена работа с исключениями и файлами в Kotlin.