|  |  |
| --- | --- |
| 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 Интеллектуальный анализ больших**

**данных в системах поддержки принятия решений.**

**Отчет**

**по лабораторной работе № 5**

**Вариант № 12**

**Название:** Исключения и файлы

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

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

Москва, 2024

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

**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 программы:

import kotlin.math.\*

import java.lang.Exception

class Vector(private val elements: MutableList<Double>) {

// Размерность вектора

private val size: Int

get() = elements.size

// Индексация

operator fun get(index: Int): Double {

require(index in 0..<size) { "Индекс вне диапазона: $index" }

return elements[index]

}

// Индексация для изменения

operator fun set(index: Int, value: Double) {

require(index in 0..<size) { "Индекс вне диапазона: $index" }

elements[index] = value

}

// Сложение векторов

operator fun plus(other: Vector): Vector {

require(size == other.size) { "Размер векторов должен быть одинаковый" }

return Vector((elements.indices).map { i -> elements[i] + other[i] }.toMutableList())

}

// Вычитание векторов

operator fun minus(other: Vector): Vector {

require(size == other.size) { "Размер векторов должен быть одинаковый" }

return Vector((elements.indices).map { i -> elements[i] - other[i] }.toMutableList())

}

// Умножение вектора на скаляр

operator fun times(scalar: Double): Vector {

return Vector(elements.map { it \* scalar }.toMutableList())

}

// Инкремент

operator fun inc(): Vector {

return this + Vector(MutableList(size) { 1.0 })

}

// Декремент

operator fun dec(): Vector {

return this - Vector(MutableList(size) { 1.0 })

}

// Скалярное произведение

fun dot(other: Vector): Double {

require(size == other.size) { "Размер векторов должен быть одинаковый" }

return elements.indices.sumOf { i -> elements[i] \* other[i] }

}

// Длина вектора

private fun magnitude(): Double {

return sqrt(elements.sumOf { it \* it })

}

// Угол между векторами в градусах

fun angleWith(other: Vector): Double {

val dotProduct = this.dot(other)

val magnitudes = this.magnitude() \* other.magnitude()

require(magnitudes != 0.0) { "Длина векторов не должна быть нулевой" }

return acos(dotProduct / magnitudes) \* (180 / PI)

}

}

fun main() {

try {

val vectors = arrayOf(

Vector(mutableListOf(1.0, 2.0, 3.0)),

Vector(mutableListOf(4.0, 5.0, 6.0)),

Vector(mutableListOf(0.0, 0.0, 0.0))

)

for (i in vectors.indices) {

for (j in i + 1..<vectors.size) {

val v1 = vectors[i]

val v2 = vectors[j]

println("Угол между вектором ${i+1} и вектором ${j+1} - ${v1.angleWith(v2)} градусов")

}

}

} catch (e: IllegalArgumentException) {

println("Ошибка: ${e.message}")

} catch (e: ArithmeticException) {

println("Ошибка в математических вычислениях: ${e.message}")

} catch (e: OutOfMemoryError) {

println("Ошибка: Недостаточно памяти")

} catch (e: Exception) {

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

}

}

Результат работы программы представлен на рисунке 1.

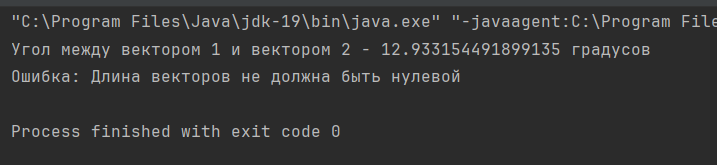


Рисунок 1 – Результат работы программы

**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 программы:

import kotlin.math.\*

class VectorSizeMismatchException(message: String) : Exception(message)

class ZeroMagnitudeException(message: String) : Exception(message)

class InvalidIndexException(message: String) : Exception(message)

class Vector2(private val elements: MutableList<Double>) {

// Размерность вектора

private val size: Int

get() = elements.size

// Индексация

operator fun get(index: Int): Double {

if (index !in 0..<size) throw InvalidIndexException("Индекс вне диапазона: $index")

return elements[index]

}

// Индексация для изменения

operator fun set(index: Int, value: Double) {

if (index !in 0..<size) throw InvalidIndexException("Индекс вне диапазона: $index")

elements[index] = value

}

// Конструктор, принимающий размерность вектора и инициализирующий нулями

constructor(size: Int) : this(MutableList(size) { 0.0 })

// Конструктор, принимающий массив значений

constructor(elements: Array<Double>) : this(elements.toMutableList())

// Сложение векторов

operator fun plus(other: Vector2): Vector2 {

if (size != other.size) throw VectorSizeMismatchException("Размер векторов должен быть одинаковый")

return Vector2((elements.indices).map { i -> elements[i] + other[i] }.toMutableList())

}

// Вычитание векторов

operator fun minus(other: Vector2): Vector2 {

if (size != other.size) throw VectorSizeMismatchException("Размер векторов должен быть одинаковый")

return Vector2((elements.indices).map { i -> elements[i] - other[i] }.toMutableList())

}

// Умножение вектора на скаляр

operator fun times(scalar: Double): Vector2 {

return Vector2(elements.map { it \* scalar }.toMutableList())

}

// Скалярное произведение

fun dot(other: Vector2): Double {

if (size != other.size) throw VectorSizeMismatchException("Размер векторов должен быть одинаковый")

return elements.indices.sumOf { i -> elements[i] \* other[i] }

}

// Модуль вектора (длина)

fun magnitude(): Double {

return sqrt(elements.sumOf { it \* it })

}

// Проверка отношений между векторами

fun checkRelationship(other: Vector2): String {

if (this.dot(other) == 0.0) {

return "Ортогональные"

} else {

var ratio: Double? = null

for (i in elements.indices) {

if (elements[i] != 0.0) {

if (other.elements[i] == 0.0) {

return "Не ортогональны и не коллинеарны"

}

val currentRatio = other.elements[i] / elements[i]

if (ratio == null) {

ratio = currentRatio

} else if (ratio != currentRatio) {

return "Не ортогональны и не коллинеарны"

}

} else if (other.elements[i] != 0.0) {

return "Не ортогональны и не коллинеарны"

}

}

return "Коллинеарны"

}

}

}

fun main() {

try {

val vectors = arrayOf(

Vector2(arrayOf(1.0, 2.0, 3.0)),

Vector2(arrayOf(2.0, 4.0, 6.0, 3.2)), // Коллинеарный с Vector 1

Vector2(arrayOf(0.0, 3.0, -1.0)),

Vector2(arrayOf(1.0, 1.0, -1.0)) // Ортогонален к v1

)

// Проверка пар векторов

for (i in vectors.indices) {

for (j in i + 1..<vectors.size) {

println("Отношение между вектором ${i + 1} и вектором ${j + 1}: ${vectors[i].checkRelationship(vectors[j])}")

}

}

} catch (e: VectorSizeMismatchException) {

println("Ошибка: ${e.message}")

} catch (e: ZeroMagnitudeException) {

println("Ошибка: ${e.message}")

} catch (e: InvalidIndexException) {

println("Ошибка: ${e.message}")

} catch (e: Exception) {

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

}

}

Результат работы программы показан на рисунке 2.

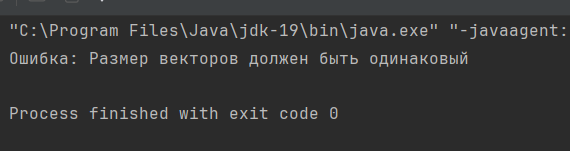


Рисунок 2 – Результат работы программы

**Задание 3:** 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.

Имена входного и выходного файлов, а также абсолютный путь к ним могут быть введены как параметры командной строки или храниться в файле.

1. В каждой строке найти и удалить заданную подстроку.

2. В каждой строке стихотворения Александра Блока найти и заменить заданную подстроку на подстроку иной длины.

Код программы:

import java.io.File

import java.io.IOException

// Пользовательские исключения для обработки ошибок ввода-вывода

class FileReadException(message: String) : Exception(message)

class FileWriteException(message: String) : Exception(message)

// Функция для чтения файла

fun readFile(filePath: String): List<String> {

return try {

File(filePath).readLines()

} catch (e: IOException) {

throw FileReadException("Ошибка чтения файла: $filePath")

}

}

// Функция для записи в файл

fun writeFile(filePath: String, lines: List<String>) {

try {

File(filePath).bufferedWriter().use { out ->

lines.forEach { out.write(it + "\n") }

}

} catch (e: IOException) {

throw FileWriteException("Ошибка записи в файл: $filePath")

}

}

// Функция для удаления подстроки из каждой строки

fun removeSubstring(lines: List<String>, substring: String): List<String> {

return lines.map { it.replace(substring, "") }

}

// Функция для замены подстроки в каждой строке

fun replaceSubstring(lines: List<String>, oldSubstring: String, newSubstring: String): List<String> {

return lines.map { it.replace(oldSubstring, newSubstring) }

}

fun main(args: Array<String>) {

if (args.size < 4) {

println("Использование: <режим> <входной файл> <выходной файл> <подстрока1> [подстрока2]")

println("Режимы: remove - удалить подстроку, replace - заменить подстроку")

return

}

val mode = args[0]

val inputFilePath = args[1]

val outputFilePath = args[2]

val substring1 = args[3]

val substring2 = if (args.size > 4) args[4] else ""

try {

val lines = readFile(inputFilePath)

val processedLines = when (mode) {

"remove" -> removeSubstring(lines, substring1)

"replace" -> replaceSubstring(lines, substring1, substring2)

else -> {

println("Неизвестный режим: $mode")

return

}

}

writeFile(outputFilePath, processedLines)

println("Обработка завершена успешно.")

} catch (e: FileReadException) {

println("Ошибка при чтении файла: ${e.message}")

} catch (e: FileWriteException) {

println("Ошибка при записи файла: ${e.message}")

} catch (e: Exception) {

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

}

}

Результат работы программы представлен на рисунке 3.

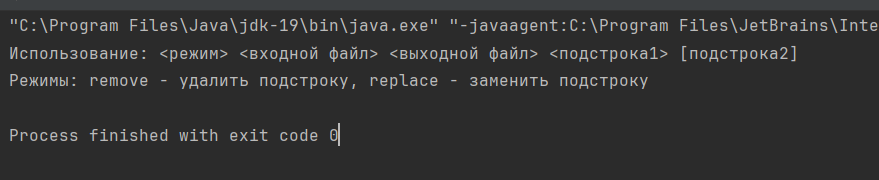
****

Рисунок 3 – Результат работы программы

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

Код программы:

package Part4

import java.io.File

import java.io.IOException

// Пользовательские исключения для обработки ошибок ввода-вывода

class FileReadException(message: String) : Exception(message)

class FileWriteException(message: String) : Exception(message)

// Функция для чтения файла

fun readFile(filePath: String): List<String> {

return try {

File(filePath).readLines()

} catch (e: IOException) {

throw FileReadException("Ошибка чтения файла: $filePath")

}

}

// Функция для записи в файл

fun writeFile(filePath: String, lines: List<String>) {

try {

File(filePath).bufferedWriter().use { out ->

lines.forEach { out.write(it + "\n") }

}

} catch (e: IOException) {

throw FileWriteException("Ошибка записи в файл: $filePath")

}

}

// Функция для замены "public" на "private"

fun replacePublicWithPrivate(lines: List<String>): List<String> {

return lines.map { it.replace("public", "private") }

}

// Функция для обращения символов в каждой строке

fun reverseLines(lines: List<String>): List<String> {

return lines.map { it.reversed() }

}

fun main(args: Array<String>) {

if (args.size < 2) {

println("Использование: <режим> <входной файл> <директория для вывода>")

println("Режимы: replace - заменить public на private, reverse - обратить строки")

return

}

val mode = args[0]

val inputFilePath = args[1]

val outputDirPath = args[2]

try {

val lines = readFile(inputFilePath)

// Создание директории для вывода, если она не существует

val outputDir = File(outputDirPath)

if (!outputDir.exists()) {

outputDir.mkdirs()

}

when (mode) {

"replace" -> {

val processedLines = replacePublicWithPrivate(lines)

val outputFilePath = "$outputDirPath/replaced\_output.kt"

writeFile(outputFilePath, processedLines)

println("Результаты сохранены в файл: $outputFilePath")

}

"reverse" -> {

val processedLines = reverseLines(lines)

val outputFilePath = "$outputDirPath/reversed\_output.kt"

writeFile(outputFilePath, processedLines)

println("Результаты сохранены в файл: $outputFilePath")

}

else -> {

println("Неизвестный режим: $mode")

}

}

} catch (e: FileReadException) {

println("Ошибка при чтении файла: ${e.message}")

} catch (e: FileWriteException) {

println("Ошибка при записи файла: ${e.message}")

} catch (e: Exception) {

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

}

}

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