|  |  |
| --- | --- |
| 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, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ 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**

**Название:** Исключения. Файлы

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

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

Москва, 2023

**Задания**

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.

package lab5

import kotlin.math.sqrt

// 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.

fun main() {

try {

val vectors = arrayOf(

Vector(1.0, 1.0, 1.0),

Vector(1.0, 2.0, 0.0),

Vector(0.0, -1.0, 1.0),

Vector(3.0, 3.0, 3.0)

)

val coplanarSet = Vector.findCoplanar(vectors)

coplanarSet.forEach { set -> set.forEach { println(it) } }

val matrixArray = arrayOf(

Matrix(

3,

arrayOf(doubleArrayOf(1.0, 2.0, 0.0), doubleArrayOf(4.0, 5.0, 6.0), doubleArrayOf(7.0, 8.0, 9.0))

),

Matrix(3, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0),

Matrix(3, 1.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0),

)

println(Matrix.withSmallestFirstNorm(matrixArray))

println(Matrix.withSmallestSecondNorm(matrixArray))

} catch (err: IllegalArgumentException) {

println("bad arguments: $err")

}

}

class Vector(private val \_x: Double = 0.0, private val \_y: Double = 0.0, private val \_z: Double = 0.0) {

/\*\* Возвращает скалярное произведение этого вектора на другой \*/

fun dot(other: Vector): Double {

return \_x \* other.\_x + \_y \* other.\_y + \_z \* other.\_z

}

/\*\* Возвращает векторное произведение этого вектора на другой \*/

fun cross(other: Vector): Vector {

val cx = \_y \* other.\_z - \_z \* other.\_y

val cy = \_z \* other.\_x - \_x \* other.\_z

val cz = \_x \* other.\_y - \_y \* other.\_x

return Vector(cx, cy, cz)

}

/\*\* Возвращает, является ли этот вектор ортогональным с другим \*/

fun isOrthogonal(other: Vector): Boolean {

return dot(other) == 0.0

}

/\*\* Возвращает, пересекаются ли этот вектор и другой \*/

fun intersects(other: Vector): Boolean {

return !cross(other).isEqual(Vector(0.0, 0.0, 0.0))

}

/\*\* Возвращает, равны ли этот вектор и другой \*/

fun isEqual(other: Vector): Boolean {

return \_x == other.\_x && \_y == other.\_y && \_z == other.\_z

}

/\*\* Возвращает, компланарны ли 3 вектора \*/

fun isCoplanarWith(v1: Vector, v2: Vector): Boolean {

val n = v1.cross(v2)

return n.dot(this) == 0.0

}

override fun toString(): String {

return "(${\_x}, ${\_y}, ${\_z})"

}

companion object {

/\*\* Возвращает, компланарные вектора \*/

fun findCoplanar(vectors: Array<Vector>): ArrayList<Set<Vector>> {

val arr = ArrayList<Set<Vector>>()

for (i in 0 until vectors.size - 2) {

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

val coplanarSet = mutableSetOf<Vector>(vectors[i], vectors[j])

for (k in j + 1 until vectors.size) {

if (vectors[i].isCoplanarWith(vectors[j], vectors[k]))

coplanarSet.add(vectors[k])

}

arr.add(coplanarSet)

}

}

return removeRepeated(arr)

}

private fun removeRepeated(arr: ArrayList<Set<Vector>>): ArrayList<Set<Vector>> {

for (i in 0 until arr.size - 1) {

for (j in i + 1 until arr.size) {

while (j < arr.size && arr[i].containsAll(arr[j])) {

arr.removeAt(j)

}

}

}

return arr

}

}

}

class Matrix {

private val n: Int

constructor(n: Int) {

// Выбрасываем исключение, если размерность матрицы меньше 1

require(n >= 0) { "Matrix size must be greater than 0" }

this.n = n

this.matrix = Array(n) { DoubleArray(n) }

}

private val matrix: Array<DoubleArray>

override fun toString(): String {

var s = ""

for (i in 0 until n) {

for (j in 0 until n) {

s += "${matrix[i][j].toString()} "

}

s += "\n"

}

return s

}

constructor(n: Int, values: Array<DoubleArray>) : this(n) {

// Выбрасываем исключение, если размерность матрицы меньше 1

require(n >= 0) { "Matrix size must be greater than 0" }

for (i in 0 until n) {

for (j in 0 until n) {

matrix[i][j] = values[i][j]

}

}

}

constructor(n: Int, vararg values: Double) : this(n) {

// Выбрасываем исключение, если размерность матрицы меньше 1

require(n >= 0) { "Matrix size must be greater than 0" }

for (i in 0 until n) {

for (j in 0 until n) {

matrix[i][j] = values[i \* n + j]

}

}

}

fun add(other: Matrix): Matrix {

// Выбрасываем исключение, если размерности матриц не равны

require(other.n == this.n) { "Matrix size must be equal" }

val result = Matrix(n)

for (i in 0 until n) {

for (j in 0 until n) {

result.matrix[i][j] = this.matrix[i][j] + other.matrix[i][j]

}

}

return result

}

fun subtract(other: Matrix): Matrix {

// Выбрасываем исключение, если размерности матриц не равны

require(other.n == this.n) { "Matrix size must be equal" }

val result = Matrix(n)

for (i in 0 until n) {

for (j in 0 until n) {

result.matrix[i][j] = this.matrix[i][j] - other.matrix[i][j]

}

}

return result

}

fun multiply(other: Matrix): Matrix {

// Выбрасываем исключение, если размерности матриц не равны

require(other.n == this.n) { "Matrix size must be equal" }

val result = Matrix(n)

for (i in 0 until n) {

for (j in 0 until n) {

for (k in 0 until n) {

result.matrix[i][j] += this.matrix[i][k] \* other.matrix[k][j]

}

}

}

return result

}

fun firstNorm(): Double {

var norm = 0.0

for (j in 0 until n) {

var sum = 0.0

for (i in 0 until n) {

sum += Math.abs(matrix[i][j])

}

norm = Math.max(norm, sum)

}

return norm

}

fun secondNorm(): Double {

var sum = 0.0

for (i in 0 until n) {

for (j in 0 until n) {

sum += matrix[i][j] \* matrix[i][j]

}

}

return sqrt(sum)

}

companion object {

fun withSmallestFirstNorm(matrixArray: Array<Matrix>): Matrix {

var smallestNorm = Double.MAX\_VALUE

var matrixWithSmallestNorm = Matrix(0)

for (matrix in matrixArray) {

val norm = matrix.firstNorm()

if (norm < smallestNorm) {

smallestNorm = norm

matrixWithSmallestNorm = matrix

}

}

return matrixWithSmallestNorm

}

fun withSmallestSecondNorm(matrixArray: Array<Matrix>): Matrix {

var smallestNorm = Double.MAX\_VALUE

var matrixWithSmallestNorm = Matrix(0)

for (matrix in matrixArray) {

val norm = matrix.secondNorm()

if (norm < smallestNorm) {

smallestNorm = norm

matrixWithSmallestNorm = matrix

}

}

return matrixWithSmallestNorm

}

}

}

Выполнить задания из варианта 2 лабораторной работы 3, реализуя собственные обработчики исключений и исключения ввода/вывода.

package lab5

import lab3.Abiturient

import lab3.Patient

// Выполнить задания из варианта 2 лабораторной работы 3,

// реализуя собственные обработчики исключений и исключения ввода/вывода.

class Patient(

private var id: Int,

private var firstName: String,

private var lastName: String,

private var patronymic: String,

private var address: String,

private var phoneNumber: String,

private var medicalCardNumber: Int,

private var diagnosis: String

) {

override fun toString(): String {

return "Patient(id=$id, lastName='$lastName', firstName='$firstName', patronymic='$patronymic', address='$address', phoneNumber='$phoneNumber', medicalCardNumber=$medicalCardNumber, diagnosis='$diagnosis')"

}

fun getId(): Int {

return id

}

fun setId(v: Int) {

require(v > 0) { "ID must be greater than 0" }

this.id = v

}

fun getFirstName(): String {

return firstName

}

fun setFirstName(v: String) {

require(v != "") { "Name must be not empty" }

this.firstName = v

}

fun getLastName(): String {

return lastName

}

fun setLastName(v: String) {

require(v != "") { "Name must be not empty" }

this.lastName = v

}

fun getPatronymic(): String {

return patronymic

}

fun setPatronymic(v: String) {

require(v != "") { "Patronymic must be not empty" }

this.patronymic = v

}

fun getAddress(): String {

return address

}

fun setAddress(v: String) {

this.address = v

}

fun getPhoneNumber(): String {

return phoneNumber

}

fun setPhoneNumber(v: String) {

this.phoneNumber = v

}

fun getMedicalCardNumber(): Int {

return medicalCardNumber

}

fun setMedicalCardNumber(v: Int) {

this.medicalCardNumber = v

}

fun getDiagnosis(): String {

return diagnosis

}

fun setDiagnosis(v: String) {

this.diagnosis = v

}

}

fun Array<Patient>.filterByDiagnosis(diagnosis: String): Array<Patient> {

return this.filter { it.getDiagnosis() == diagnosis }.toTypedArray()

}

fun Array<Patient>.filterByMedicalCardNumber(start: Int, end: Int): Array<Patient> {

return this.filter { it.getMedicalCardNumber() in start..end }.toTypedArray()

}

class Abiturient(

private var id: Int,

private var surname: String,

private var firstName: String,

private var patronymic: String,

private var address: String,

private var phone: String,

private var grades: List<Int>

) {

fun setId(id: Int) {

this.id = id

}

fun setSurname(surname: String) {

this.surname = surname

}

fun setFirstName(firstName: String) {

this.firstName = firstName

}

fun setPatronymic(patronymic: String) {

this.patronymic = patronymic

}

fun setAddress(address: String) {

this.address = address

}

fun setPhone(phone: String) {

this.phone = phone

}

fun setGrades(grades: List<Int>) {

this.grades = grades

}

fun getId(): Int {

return id

}

fun getSurname(): String {

return surname

}

fun getFirstName(): String {

return firstName

}

fun getPatronymic(): String {

return patronymic

}

fun getAddress(): String {

return address

}

fun getPhone(): String {

return phone

}

fun getGrades(): List<Int> {

return grades

}

fun getAverageGrade(): Double {

return grades.average()

}

override fun toString(): String {

return "Abiturient(id=$id, surname='$surname', firstName='$firstName', patronymic='$patronymic', address='$address', phone='$phone', grades=$grades)"

}

companion object {

}

}

class AbiturArray(var abiturients: Array<Abiturient>) {

fun filterByUnsatisfactoryGrades(): List<Abiturient> {

return abiturients.filter { it.getGrades().any { grade -> grade < 3 } }

}

fun filterByAverageGradeGreaterThan(average: Double): List<Abiturient> {

return abiturients.filter { it.getAverageGrade() > average }

}

fun getTopNAbiturients(n: Int): List<Abiturient> {

val semiPassingScore = 3.5

val sortedAbiturients = abiturients.sortedByDescending { it.getAverageGrade() }

val topNAbiturients = sortedAbiturients.take(n)

val semiPassingAbiturients =

sortedAbiturients.filter { it.getAverageGrade() >= semiPassingScore && it !in topNAbiturients }

println("Semi-passing abiturients: $semiPassingAbiturients")

return topNAbiturients

}

}

Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести:

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.

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

package lab5

import java.io.BufferedReader

import java.io.FileInputStream

import java.io.IOException

import java.io.InputStreamReader

/\*\*

\* Вариант 3. Задача 2.

\* 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 Александра Блока найти и заменить заданную подстроку на подстроку иной длины.

\*/

fun main(args: Array<String>) {

val substr1 = "улица, фонарь"

val substr2 = "проспект, неон"

val inputFileName = "lab5/v3-1.txt"

try {

val reader = BufferedReader(InputStreamReader(FileInputStream(inputFileName), "UTF-8"))

for (line in reader.lines()) {

val parts = line.split(substr1)

if (parts.size == 1) {

println(line)

continue

}

var newLine = ""

newLine += parts[0]

for (part in parts.drop(1)) {

newLine += substr2 + part

}

println(newLine)

}

reader.close()

} catch (e: IOException) {

println("Error while reading file: " + e.message)

return

}

}

3. В каждой строке найти слова, начинающиеся с гласной буквы.

package lab5

import java.util.\*

/\*\*

\* Вариант 3. Задача 3.

\* 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 найти слова, начинающиеся с гласной буквы.

\*/

val vowels: CharArray = charArrayOf('a', 'o', 'u', 'e', 'i', 'A', 'O', 'U', 'E', 'I')

fun main(args: Array<String>) {

val scanner = Scanner(System.`in`)

var foundWords = arrayOf<String>()

var line: String

while (scanner.nextLine().also { line = it }.isNotEmpty()) {

val words = line.split("\\s+".toRegex()).dropLastWhile { it.isEmpty() }.toTypedArray()

for (word in words) {

if (word.first() in vowels) {

foundWords += word

}

}

println("Words with first vowel letter: ${foundWords.contentToString()}")

foundWords = arrayOf<String>()

}

}

При выполнении следующих заданий для вывода результатов создавать новую директорию и файл средствами класса File

2. Прочитать текст Java-программы и записать в другой файл в обратном порядке символы каждой строки.

package lab5

import java.io.\*

/\*\*

\* Вариант 4. Задача 2.

\* При выполнении для вывода результатов создавать

\* новую директорию и файл средствами класса File.

\*

\* 2. Прочитать текст Java-программы и записать в другой файл в обратном порядке символы каждой строки.

\*/

fun main(args: Array<String>) {

// Входной и выходной файлы

val inputFileName = "lab5/v4-1\_in.txt"

val outputFileName = "lab5/v4-1\_out.txt"

try {

val inputFile = File(inputFileName)

val outputFile = File(outputFileName)

val outputDir = outputFile.parentFile

if (!outputDir.exists()) {

outputDir.mkdirs()

}

val reader = BufferedReader(FileReader(inputFile))

val writer = BufferedWriter(FileWriter(outputFile))

for (line in reader.lines()) {

val outLine = line.reversed()

writer.write(outLine)

writer.write("\n")

}

// Закрываем потоки

reader.close()

writer.close()

} catch (e: IOException) {

e.printStackTrace()

}

}

3. Прочитать текст Java-программы и в каждом слове длиннее двух символов все строчные символы заменить прописными.

package lab5

import java.io.\*

object e8 {

/\*\*

\* Вариант 4. Задача 6.

\* При выполнении для вывода результатов создавать

\* новую директорию и файл средствами класса File.

\*

\* 3. Прочитать текст Java-программы и в каждом слове длиннее двух символов все строчные символы заменить прописными.

\*

\* 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.

\*/

@JvmStatic

fun main(args: Array<String>) {

// Входной и выходной файлы

val inputFileName = "lab5/v4-2\_in.txt"

val outputFileName = "lab5/v4-2\_out.txt"

try {

val inputFile = File(inputFileName)

val outputFile = File(outputFileName)

val outputDir = outputFile.parentFile

if (!outputDir.exists()) {

outputDir.mkdirs()

}

val reader = BufferedReader(FileReader(inputFile))

val writer = BufferedWriter(FileWriter(outputFile))

for (line in reader.lines()) {

val words: Array<String?> = line.split("\\s+".toRegex()).dropLastWhile { it.isEmpty() }.toTypedArray()

var outLine = ""

for (word in words) {

if (word != null) {

outLine += if (word.length > 2) {

word.uppercase()

} else {

word

}

outLine += ' '

}

}

writer.write(outLine)

writer.write("\n")

}

// Закрываем потоки

reader.close()

writer.close()

} catch (e: IOException) {

e.printStackTrace()

}

}

}

**Вывод:** В ходе лабораторной работы мы изучили и реализовали задания по работе с исключениями, файлами и классами в Kotlin. Также мы практиковались в контроле состояния потоков ввода/вывода и создании новых директорий и файлов. В целом, выполнение лабораторной работы помогло нам укрепить свои навыки программирования на Kotlin и применение ее возможностей в различных задачах.