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 **09.04.01/12 Интеллектуальный анализ больших данных в системах поддержки принятия решений**

О Т Ч Е Т
по лабораторной работе № 5

Название: Исключения, файлы

Дисциплина: Языки программирования для работы с большими данными

Студент ИУ6-23М _____ М.А. Гейне
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ П.В. Степанов
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

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

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) Найти и вывести слова текста, для которых последняя буква одного слова совпадает с первой буквой следующего слова.
2) Найти в строке наибольшее число цифр, идущих подряд.

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

1) В файле, содержащем фамилии студентов и их оценки, записать прописными буквами фамилии тех студентов, которые имеют средний балл более “7”.
2) Файл содержит символы, слова, целые числа и числа с плавающей запятой. Определить все данные, тип которых вводится из командной строки.

Задача 1

В разработанную ранее систему классов добавлены обработчики ошибок, в том числе в функциональном стиле. В классе и его методах уже имелись проверки с использованием require для предотвращения получения некорректных входных данных. Добавлены методы для записи объекта в файл и чтения и создания объектов из файла. В методах предусмотрена обработка ошибок: есть версии методов, использующих исключения, и версии с обработкой ошибок в функциональном стиле. Добавленный код приведён в листинге 1.

Листинг 1 -- Решение задачи 1

```
import java.io.{FileNotFoundException, IOException, PrintWriter}
import scala.util.{Try, Success, Failure}


class Matrix(private val data: Array[Array[Int]]):
  require(data.forall(_.length == data.length), "Matrix must be square!")
...

  def saveToFileExn(filename: String): Unit =
    Try(new PrintWriter(filename)) match
      case Success(writer) =>
        try
          data.foreach(row => writer.println(row.mkString(",")))
        finally
          writer.close()
      case Failure(ex) =>
        throw new IOException(s"Failed to save matrix to file '$filename':
${ex.getMessage}")

  def saveToFile(filename: String): Try[Unit] =
    Try(new PrintWriter(filename)) match
      case Success(writer) =>
        val res = Try(data.foreach(row => writer.println(row.mkString(","))))
        writer.close()
        res
      case Failure(ex) =>
        Failure(IOException(s"Failed to save matrix to file '$filename':
${ex.getMessage}"))
```

```
object Matrix:
  def loadFromFileExn(filename: String): Matrix =
    Try(io.Source.fromFile(filename).getLines().map(line =>
line.split(",").map(_.toInt).toArray).toArray) match
      case Success(rows) =>
        if (rows.forall(_.length == rows.length))
          new Matrix(rows)
        else
          throw new IllegalArgumentException(s"Invalid matrix size in file
'$filename'")
      case Failure(ex: FileNotFoundException) =>
        throw ex
      case Failure(ex) =>
        throw new IOException(s"Failed to load matrix from file '$filename':
${ex.getMessage}")

  def loadFromFile(filename: String): Try[Matrix] =
    Try(io.Source.fromFile(filename).getLines().map(line =>
line.split(",").map(_.toInt).toArray).toArray) match
      case Success(rows) =>
        if (rows.forall(_.length == rows.length))
          Success(new Matrix(rows))
        else
          Failure(new IllegalArgumentException(s"Invalid matrix size in file
'$filename'"))
      case Failure(ex: FileNotFoundException) =>
        Failure(ex)
      case Failure(ex) =>
        Failure(new IOException(s"Failed to load matrix from file '$filename':
${ex.getMessage}"))
```

Задача 2

Код задачи был модифицирован путём добавления проверок исключительных ситуаций. Такая ситуация возможна только в функции возведения матрицы в квадрат: функция накладывает ограничение на размер матрицы. Это условие описано с использованием метода require, как показано в листинге 2.

Листинг 2 -- Задача 2

```
def square(arr: List[Matrix], i: Int): List[Matrix] = {
  require(arr(i).data.forall(_.length == arr(i).data.length), "To square a
matrix it should be n*n dimensions!")
  val oldData = arr(i).data
  val size = oldData.length
  val newData = Vector.tabulate(size, size) { (i, j) =>
    (0 until size).map(k => oldData(i)(k) * oldData(k)(j)).sum
  }
  arr.updated(i, new Matrix(newData))
}
```

Задача 3

К коду системы классов абитуриентов были добавлены методы чтения и записи в файл с обработкой исключений в функциональном стиле. Также создан класс исключения для обработки ситуации, в которой были получены некорректные оценки. Добавленный код приведён в листинге 3.

Листинг 3 -- Задача 3

```
import scala.util.{Try, Success, Failure}
import java.io.{FileNotFoundException, IOException, PrintWriter}
class InvalidMarkException(msg: String) extends Exception(msg)
class Abiturient(...):
  require(!(_marks.exists(mark => (mark>100) || (mark<0))), "Marks should be
in range 0..100")
...
  def marks_=(marks:List[Int]) =
    if(marks.exists(mark => (mark>100) || (mark<0)))
      throw new InvalidMarkException("Marks should be in range 0..100")
    _marks = marks
...
class AbitList(private val _lst: List[Abiturient]):
...
  def avgOver(threshold: Int) =
    AbitList(_lst.filter(abt =>
      Try(abt.marks.sum / abt.marks.length.toDouble) match
        case Success(avg) => avg > threshold
        case Failure(exception) => throw exception))

  def top(n: Int = _lst.length) =
    AbitList(_lst.sortBy(abt =>
      Try(abt.marks.sum / abt.marks.length.toDouble) match
        case Success(avg) => avg
        case Failure(exception) => throw
exception)(Ordering[Double].reverse).take(n))
object AbitList:
  def readFromFile(filename: String): Try[AbitList] =
    Try {
      val source = scala.io.Source.fromFile(filename)
      val lines = source.getLines().toList
      source.close()
      val abts = lines.map(line =>
        val fields = line.split(";")
        val id = fields(0).toInt
        ...
        val marks = fields.drop(6).map(_.toInt).toList
        Abiturient(id, surname, name, patronymic, address, telephone, marks))
      new AbitList(abts)
    }.recoverWith {
      case e: FileNotFoundException => Failure(new IOException("File not
found", e))
      case e: IOException => Failure(new IOException("Error reading file", e))
      case e: Exception => Failure(e)
    }
```

```
def writeToFile(filename: String, abts: AbitList): Try[Unit] =
  Try {
    val writer = new PrintWriter(filename)
    for (abt <- abts.lst) {
      val line = List(
        abt.id.toString,
        ...
      ).mkString(";") + ";" + abt.marks.mkString(";")
      writer.println(line)
    }
    writer.close()
  }.recoverWith {
    case e: IOException => Failure(new IOException("Error writing file", e))
    case e: Exception => Failure(e)
  }
```

Задача 4

Аналогично предыдущему пункту были добавлены методы чтения и записи в файл с обработкой ошибок в функциональном стиле. Добавлено исключение на случай, если был предоставлен некорректный список авторов. Добавленный код приведён в листинге 4.

Листинг 4 -- Задача 4

```
import scala.util.{Try, Success, Failure}
import java.io.{FileNotFoundException, IOException, PrintWriter}

class InvalidAuthorsException(msg: String) extends Exception(msg)
class Book(...):
  require(_authors.length>=1, "There must be at least 1 author of a book!")
...
  def authors_=(authors:List[String]) =
    if(_authors.length>=1)
      throw new InvalidAuthorsException("There must be at least 1 author of a
book!")
    _authors = authors
...

object BookList:
  def readFromFile(filename: String): Try[BookList] =
    Try {
      val source = scala.io.Source.fromFile(filename)
      val lines = source.getLines().toList
      source.close()
      val books = lines.map(line =>
        val fields = line.split(";")
        val id = fields(0).toInt
        val title = fields(1)
        ...
        val authors = fields.drop(7).toList
        Book(id, title, authors, publisher, year, pages, price, cover))
      new BookList(books)
    }.recoverWith {
      case e: FileNotFoundException => Failure(new IOException("File not
found", e))
      case e: IOException => Failure(new IOException("Error reading file", e))
      case e: Exception => Failure(e)
    }

  def writeToFile(filename: String, books: BookList): Try[Unit] =
    Try {
      val writer = new PrintWriter(filename)
      for (book <- books.lst) {
        val line = List(
          book.id.toString,
          ...
          book.cover
        ).mkString(";") + ";" + book.authors.mkString(";")
        writer.println(line)
      }
      writer.close()
    }.recoverWith {
      case e: IOException => Failure(new IOException("Error writing file", e))
      case e: Exception => Failure(e)
    }
```

Задача 5

Для решения задачи написана функция чтения из файла и поиска в прочитанной информации в соответствии с условием. Чтение из файла производится с помощью объекта Using: он позволяет безопасно получить ресурс для ввода/вывода, утилизировать его и обернуть результат в класс Try (либо вернуть результат в случае успеха, либо же вернуть неуспех с исключением). Функция поиска рекурсивно обрабатывает список строк и ищет заданное условие с использованием паттерн-матчинга. Запись в файл производится также с помощью Using. Код задачи приведён в листинге 5.

Листинг 5 -- Задача 5

```
import scala.util.{Try, Success, Failure}
import java.io.{PrintWriter}
import scala.util.Using
import scala.annotation.tailrec

def readFromFile(filename: String) =
  Using(scala.io.Source.fromFile(filename))(_.getLines().toList.map(_.split("
")).flatten)

def search(lst: List[String]) =
  @tailrec def recsearch(lst: List[String], acc: List[String]): List[String] =
    lst match
      case fst :: sec :: tail if fst.charAt(fst.length-1) == sec.charAt(0)  =>
recsearch(sec :: tail, fst :: acc)
      case head :: next => recsearch(next, acc)
      case Nil => acc
  recsearch(lst, Nil)

def test(input: String, output: String) =
  readFromFile(filename = input) match
    case Failure(e) => println(s"Exeption occured when reading input file:
${e.toString()}")
    case Success(value) =>
      Using(new PrintWriter(output))(writer
=>search(value).reverse.foreach(writer.println(_))) match
        case Failure(e) => println(s"Exeption occured when writing to output
file: ${e.toString()}")
        case Success(u) => u
```

Результат работы программы приведён в листинге 6.

Листинг 6 -- Решение задачи 5

```
Input:
ab bc cd
de fe gx
de ef fe
Output:
ab
bc
cd
de
ef
```

Задача 6

В задаче был использован аналогичный задаче 5 подход. При чтении файла сразу происходит его разбиение на цифры и преобразование к Int. При поиске учитывается, что цифры могут идти подряд как в сторону их увеличения, так и уменьшения. Код задачи приведён в листинге 7.

Листинг 7 -- Задача 6

```
import scala.util.{Try, Success, Failure}
import java.io.{PrintWriter}
import scala.util.Using
import scala.annotation.tailrec

def readFromFile(filename: String) =

Using(scala.io.Source.fromFile(filename))(_.getLines().toList.map(_.split("").
toList.map(_.toInt)))

def search(lst: List[Int]) =
  @tailrec def recsearch(lst: List[Int], acc: Int, max: Int): Int =
    lst match
      case a :: b :: tail if a == b || a == b-1 || a == b+1 =>
        if acc == 0 then recsearch(b :: tail, 2, if 2 > max then 2 else max)
        else recsearch(b :: tail, acc+1, if acc+1 > max then acc+1 else max)
      case head :: next => recsearch(next, 0, max)
      case Nil => max
  recsearch(lst, 0, 0)

def test(input: String, output: String) =
  readFromFile(filename = input) match
    case Failure(e) => println(s"Exeption occured when reading input file:
${e.toString()}")
    case Success(value) =>
      val lengths = value.map(search(_))
      Using(new PrintWriter(output))(writer =>
lengths.foreach(writer.println(_))) match
        case Failure(e) => println(s"Exeption occured when writing to output
file: ${e.toString()}")
        case Success(u) => u
```

Результат выполнения программы приведён в листинге 8.

Листинг 8 -- Решение задачи 6

```
Input:
12345434568345 69
54323470987 83
87656923454390 98
Output:
10
6
6
```

Задача 7

Для решения задачи использовался Using.Manager для обработки нескольких ресурсов единовременно. В задаче предусмотрено создание директорий, указанных в пути к файлу вывода, если их ещё нет в системе. Обработка входных данных происходит в tail-рекурсивной функции. Код задачи приведён в листинге 9.

Листинг 9 -- Задача 7

```
import scala.util.{Try, Success, Failure}
import java.io.{PrintWriter}
import scala.util.Using
import scala.annotation.tailrec
import java.io.File


def test(input: String, output:String) =
  Using.Manager {use =>
    val source = use(scala.io.Source.fromFile(input))
    val outputFile = new File(output)
    outputFile.getParentFile().mkdirs()
    val writer = use(new PrintWriter(outputFile))
    val lines = source.getLines()

    @tailrec def transform(iter: Iterator[String]): Unit =
      iter.nextOption() match
        case None => ()
        case Some(line) =>
          val grades = line.split(" ").tail.map(_.toDouble)
          if grades.sum / grades.length > 7 then
            writer.println(s"${line.split(" ").head.toUpperCase()}
${grades.mkString(" ")}")
          else
            writer.println(line)
          transform(iter)

    transform(lines)
  } match
    case Failure(exception) => println(s"Exeption occured when working with
file: ${exception.toString()}")
    case Success(_) => ()
```

Результат работы программы приведён в листинге 10.

Листинг 10 -- Решение задачи 7

```
Input:
Ivanov 8 8 8.2
Petrov 3 2 1.5
Sidorov 7 7 7
Vasechkin 7.1 7.1 7.1
Output:
IVANOV 8.0 8.0 8.2
Petrov 3 2 1.5
Sidorov 7 7 7
VASECHKIN 7.1 7.1 7.1
```

Задача 8

В задаче для работы с ресурсами используется Using.Manager. Сохраняется возможность создания директорий для вывода. Искомый тип указывается при работе программы и сравнивается с имеющимися вариантами обработки в программе. Типы данных ищутся с помощью регулярных выражений. Код задачи приведён в листинге 11.

Листинг 11 -- Задача 8

```
import scala.util.{Try, Success, Failure}
import java.io.{PrintWriter}
import scala.util.Using
import scala.annotation.tailrec
import java.io.File
import scala.io.StdIn


def test(input: String, output:String) =

  Using.Manager {use =>
    val source = use(scala.io.Source.fromFile(input))
    val outputFile = new File(output)
    outputFile.getParentFile().mkdirs()
    val writer = use(new PrintWriter(outputFile))

    val body = source.mkString("").split("\\s+")

    println("What type of data are you looking for?")
    StdIn.readLine() match
      case "char" | "Char" =>
writer.println(body.filter(_.matches("\\D")).mkString(" "))
      case "word" | "Word" => writer.println(body.filter(_.matches("[a-zA-
Z]+")).mkString(" "))
      case "int" | "Int" => writer.println(body.filter(_.matches("-
?\\d+")).mkString(" "))
      case "float" | "Float" => writer.println(body.filter(_.matches("-
?\\d+\\.\\d+")).mkString(" "))
      case _: String => println("Incorrect type!")

  } match
    case Failure(exception) => println(s"Exeption occured when working with
file: ${exception.toString()}")
    case Success(_) => ()
```

Результат работы программы приведён в листинге 12.

Листинг 12 -- Решение задачи 8

```
Input:
c 12 string 2.5 13 5.6 a bib
$> float
Output:
2.5 5.6
```

ВЫВОДЫ

Изучены способы работы с исключениями и обработки ошибок в функциональном стиле на языке Scala.

Изучены способы работы с файлами на языке Scala и управления ресурсами. Изучены методы создания директорий.