МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Программная инженерия»

Отчёт по Лабораторной работе №4

По дисциплине: «Информационные технологии и программирование»

«Обработка исключений в Java»

Выполнила: студентка группы БПИ2401

Алексеева Татьяна Игоревна

Проверил: Харрасов Камиль Раисович

Москва

2025

Цель работы

Изучить механизм обработки исключений в языке Java, научиться правильно использовать конструкции try-catch-finally, оператор throw и ключевое слово throws, а также освоить создание собственных классов исключений.

Выполнение работы

Задание 1: написать программу, которая будет находить среднее арифметическое элементов массива. При этом программа должна обрабатывать ошибки, связанные с выходом за границы массива и неверными данными (например, если элемент массива не является числом).

```
public class ArrayAverage {
    public static void main(String[] args) {
        String[] arr = {"1", "2", "3", "4", "5"};
        //String[] arr = {"1", "2", "abc", "4"};
        //String[] arr = {};

        int sum = 0;
        int counter = 0;

        try {
            for (int i = 0; i <= arr.length; i++) {
                int summa = Integer.parseInt(arr[i]);
                sum += summa;
                counter++;
            }

            if (counter == 0) {
                throw new ArithmeticException("Массив пуст или не содержит чисел");
            }

            double average = (double) sum / counter;
            System.out.println("Среднее арифметическое: " + average);
        } catch (NumberFormatException e) {
            System.out.println("Ошибка: элемент массива не является числом");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Ошибка: выход за границы массива");
        } catch (ArithmeticException e) {
            System.out.println("Ошибка: " + e.getMessage());
        }

    }
}
```

Создаём несколько тестовых массивов (полностью подходящий, с неверными данными (должен вызвать NumberFormatException) и пустой массив (должен вызвать ArithmeticException)). Вводим переменные для вычислений: sum – сумма всех элементов, counter – подсчёт количества элементов в массиве.

В блоке try{…} прописываем подсчёт суммы элементов массива и их количество с помощью цикла for. В цикле намеренно используется <=, чтобы вызвать для проверки ошибку выхода за границы массива (ArrayIndexOutOfBoundsException).

Метод Integer.parseInt() принимает строку и пытается преобразовать её в целое число. Если строка будет содержать нецифровые символа появится ошибка NumberFormatException.

После цикла for идёт проверка массива на пустоту, можно сказать защита от деления 0 на 0. Здесь вручную создано исключение (throw new ArithmeticException).

Каждый блок catch ловит конкретный тип исключения:

- catch (NumberFormatException e) срабатывает когда parseInt() не может преобразовать строку
- catch (ArrayIndexOutOfBoundsException e) срабатывает из-за ошибки в цикле <=
- catch (ArithmeticException e) ловит ранее прописанное исключение, которое срабатывает в случае пустого массива. Метод getMessage() выводит сообщение, указанное в прописанном исключении.

```
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour>  c:; cd 'c:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour'; & 'C:\Program Files\Java\jdk-25\bin\java.exe'
s' '-cp' 'C:\Users\taale\AppData\Roaming\Code\User\workspaceStorage\9fbbda8bacc39efcba4912c89818c108\redhat.java\jdt_ws\LabWorkFour_f5ed24b6\bin' 'ArrayAverage'
Среднее арифметическое: 3.0
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour>  c:; cd 'c:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour'; & 'C:\Program Files\Java\jdk-25\bin\java.exe'
s' '-cp' 'C:\Users\taale\AppData\Roaming\Code\User\workspaceStorage\9fbbda8bacc39efcba4912c89818c108\redhat.java\jdt_ws\LabWorkFour_f5ed24b6\bin' 'ArrayAverage'
Ошибка: выход за границы массива
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour>  c:; cd 'c:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour'; & 'C:\Program Files\Java\jdk-25\bin\java.exe'
s' '-cp' 'C:\Users\taale\AppData\Roaming\Code\User\workspaceStorage\9fbbda8bacc39efcba4912c89818c108\redhat.java\jdt_ws\LabWorkFour_f5ed24b6\bin' 'ArrayAverage'
Ошибка: элемент массива не является числом
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour>  c:; cd 'c:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour'; & 'C:\Program Files\Java\jdk-25\bin\java.exe'
s' '-cp' 'C:\Users\taale\AppData\Roaming\Code\User\workspaceStorage\9fbbda8bacc39efcba4912c89818c108\redhat.java\jdt_ws\LabWorkFour_f5ed24b6\bin' 'ArrayAverage'
Ошибка: Массив пуст или не содержит чисел
```

Рис. 1. Работа программы ArrayAverage

Задание 2: написать программу, которая будет копировать содержимое одного файла в другой. При этом программа должна обрабатывать возможные ошибки, связанные: с открытием и закрытием файлов, чтением и записью файлов.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class OpenAndCloseErrors {
    public static void main(String[] args) {
        FileInputStream input = null;
        FileOutputStream output = null;
```

```
        try {
            input = new FileInputStream("input.txt");
            output = new FileOutputStream("output.txt");

            int data;
            try {
                while ((data = input.read()) != -1) {
                    output.write(data);
                }
                System.out.println("Копирование завершено успешно");
            } catch (IOException e) {
                System.out.println("Ошибка при чтении или записи файла");
                System.out.println(e.getMessage());
            }


        } catch (FileNotFoundException e) {
            System.out.println("Ошибка: не удалось открыть один из файлов");
            System.out.println(e.getMessage());
        } finally {
            try {
                if (input != null) input.close();
            } catch (IOException e) {
                System.out.println("Ошибка при закрытии входного файла");
            }

            try {
                if (output != null) output.close();
            } catch (IOException e) {
                System.out.println("Ошибка при закрытии выходного файла");
            }
        }
    }
}
```

Подключаем необходимые классы: FileInputStream (позволяет читать данные из файла по байтам), FileOutputStream (записывает данные в файл), FileNotFoundException (исключение при невозможности открыть файл), IOException (общие ошибки ввода-вывода).

Объявляем переменные для файловых потоков. В блоке try{…} могут возникнуть ошибки открытия файлов, например input.txt не существует или output.txt нельзя создать, тогда выбросится ошибка FileNotFoundException и программа перейдёт в соответствующий блок catch.

int data – это переменная для чтения байтов, каждый байт, считанный из файла, будет попадать в эту переменную перед записью в другой файл.

Метод read() возвращает байт из файла, если достигнут конец файла, то возвращается -1. Цикл while продолжается, пока не кончится файл. write() записывает каждый считанный байт в новый файл.

Здесь могут возникнуть ошибки IOException: файл недоступен в процессе работы, проблемы при чтении, диск переполнен, проблемы при записи. Если одна из ошибок возникает, то мы попадает в этот блок catch.

Блок catch (FileNotFoundException e) отлавливает исключение, которое может произойти при создании потоков. Если файла нет или он недоступен, появится это сообщение.

В блоке finally мы закрываем входной и выходной файлы. Проверяем, что input.txt != null, потому что файл мог не открыться, если попытаться закрыть null, возникнет ошибка. Аналогично работает закрытие output.

```
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour>  c:; cd 'c:\Users\taale\OneD
s' '-cp' 'C:\Users\taale\AppData\Roaming\Code\User\workspaceStorage\9fbbda8bacc39ef
Копирование завершено успешно
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour>  c:; cd 'c:\Users\taale\OneD
s' '-cp' 'C:\Users\taale\AppData\Roaming\Code\User\workspaceStorage\9fbbda8bacc39ef
Ошибка: не удалось открыть один из файлов
input.txt (Не удается найти указанный файл)
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour>  c:; cd 'c:\Users\taale\OneD
s' '-cp' 'C:\Users\taale\AppData\Roaming\Code\User\workspaceStorage\9fbbda8bacc39ef
Ошибка: не удалось открыть один из файлов
output.txt (Отказано в доступе)
```

Рис. 2. Работа программы OpenCloseReadWriteErrors

Задание 3: создать Java-проект для работы с исключениями. Для первой задачи написать собственный класс для обработки исключений. Создайть обработчик исключений, который логирует информацию о каждом выброшенном исключении в текстовый файл.

Создать класс CustomDivisionException, который будет использоваться для обработки исключений при делении на ноль. Написать программу, которая делит два числа, и, если происходит деление на ноль, выбрасывает исключение CustomDivisionException.

```java
package JavaProject;

public class CustomDivisionException extends Exception{
    public CustomDivisionException(String message) {
        super(message);
    }
}
```

Это класс CustomDivisionException. Происходит наследование от Exception. CustomDivisionException – это собственный тип исключения, наследование делает его проверяемым исключением (checked exception), которое Java требует либо обрабатывать, либо объявлять в методе через throws.

Конструктор принимает строку message, которая описывает причину ошибки. super(message) передаёт это сообщение в базовый класс Exception, чтобы оно стало доступным через методы getMessage() или toString().

То есть этот класс просто создаёт удобный способ «именовать» ошибку деления на ноль и передавать пояснения.

```
package JavaProject;


import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;

public class ExceptionLogger {

    private static final String LOG_FILE = "JavaProject/log.txt";

    public static void log(Exception e) {
        try (FileWriter writer = new FileWriter(LOG_FILE, true)) {
            writer.write(LocalDateTime.now() + " - " + e.getClass().getName() +
": " + e.getMessage() + "\n");
        } catch (IOException io) {
            System.out.println("Ошибка записи в файл лога: " + io.getMessage());
        }
    }
}
```

LOG_FILE – это путь к файлу, куда будут записываться ошибки. Используется абсолютный путь, но можно использовать и относительный “log.txt”.

Метод log(Exception e) – статичный метод, чтобы его можно было вызвать без создания объекта класса. Он получает объект Exception и записывает в файл дату/время, имя класса исключения и сообщение.

В блоке try записываем сообщение в конец файла, а не полностью перезаписываем его (параметр true).

Если что-то пошло не так во время записи в файл (IOException), выводим сообщение в консоль.

```
package JavaProject;

import java.util.Scanner;
```

```java
public class DivisionProgram {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        try {
            System.out.println("Введите первое число: ");
            double a = s.nextDouble();

            System.out.print("Введите второе число: ");
            double b = s.nextDouble();

            double result = divide(a, b);
            System.out.println("Результат деления: " + result);

        } catch (CustomDivisionException e) {
            System.out.println("Ошибка: " + e.getMessage());
            ExceptionLogger.log(e);
        }

        s.close();
    }


    public static double divide(double a, double b) throws
CustomDivisionException {
        if (b == 0) {
            throw new CustomDivisionException("Попытка деления на ноль");
        }
        return a/ b;
    }
}
```

Здесь пользователь вводит два числа с плавающей, которые хочет использовать. В блоке try вызывается метод divide(a, b), если b = 0, то метод выбросит CustomDivisionException, и управление перейдёт в блок catch.

Обработка в блоке catch: выводим сообщение об ошибке пользователю, также записываем информацию об ошибке в файл.

Таким образом, программа безопасно делит два числа, обрабатывает деление на 0 собственным исключением и логирует его в файл.

```
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkFour> & 'C:\Program Files\Java\jdk-25\bin\java.exe' '--enable-pr
torage\9fbbda8bacc39efcba4912c89818c108\redhat.java\jdt_ws\LabWorkFour_f5ed24b6\bin' 'JavaProject.DivisionProgram'
Введите первое число:
12
Введите второе число: 6
Результат деления: 2.0
```

Рис. 3. Успешная работа DivisionProgram

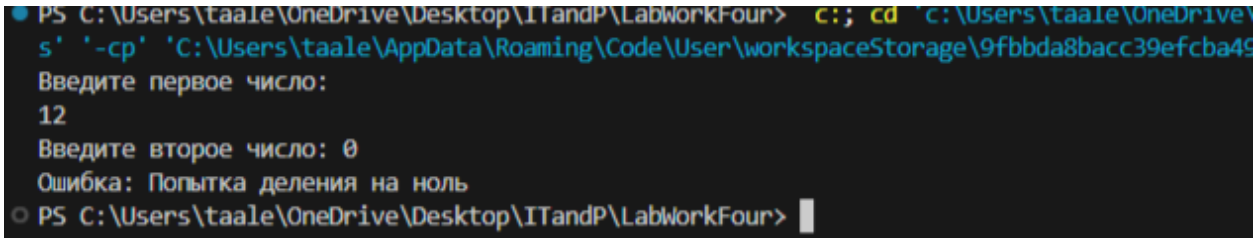

Рис. 4. Деление на ноль

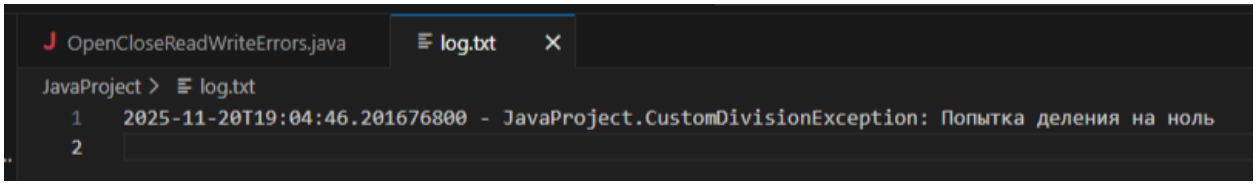

Рис. 5. Запись в файле log.txt

Вывод

В ходе выполнения лабораторной работы были изучены и применены на практике основные механизмы обработки исключений в языке Java. Была реализована программа для вычисления среднего арифметического элементов массива с обработкой стандартных исключений, написана программа для копирования файлов с использованием потоков ввода-вывода, обеспечивающая корректную обработку ошибок открытия, чтения, записи и закрытия файлов, разработан собственный класс исключения для обработки деления на ноль. В результате работы были закреплены навыки использования конструкций try-catch-finally, ключевого слова throws, оператора throw.