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 «Математическая кибернетика и информационные технологии»

Отчет по лабораторной работе №4

по дисциплине «Информационные технологии и программирование»

Выполнил: студент группы БПИ2401

Юлдашев Всеволод

Проверил: Харрасов Камиль Раисович

Москва

2025

1. **Цель работы:** познакомиться с механизмом обработки исключений в Java, научиться различать виды исключений (checked/unchecked, Exception/Error), создавать собственные классы исключений и корректно использовать конструкции **try**, **catch**, **finally** и **try-with-resources** при работе с массивами, файлами и собственными структурами данных (стек).

2. **Ход работы:**

```
class ArrayAverage {
public static void main(String[] args) {
    Object[] arr = {1, 2, 3,"das", 4, 5};
    double sum = 0;
    int count = 0;

    try {
        if (arr == null || arr.length == 0) {
            throw new IllegalArgumentException("Массив пуст");
        }

        System.out.println("Обработка массива:");
        for (int i = 0; i < arr.length; i++) {
            try {
                if (arr[i] instanceof Number) {
                    sum += ((Number) arr[i]).doubleValue();
                    count++;
                    System.out.println("Элемент " + i + ": "
+ arr[i]);
                } else {
                    throw new ClassCastException("Элемент " +
i +
                            " не является числом: " + arr[i]);
                }
            } catch (ClassCastException e) {
```

```
                    System.err.println("Предупреждение: " +
e.getMessage());
                }
            }

            if (count == 0) {
                throw new ArithmeticException("Нет числовых
элементов для вычисления");
            }

            double average = sum / count;
            System.out.println("\nСумма: " + sum);
            System.out.println("Среднее арифметическое: " +
average);
            System.out.println("Обработано элементов: " + count +
" из " + arr.length);

        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Ошибка: Выход за границы
массива!");
        } catch (IllegalArgumentException e) {
            System.err.println("Ошибка: " + e.getMessage());
        } catch (ArithmeticException e) {
            System.err.println("Ошибка: " + e.getMessage());
        }
    }
}
```

```
❯ java ArrayAverage
Обработка массива:
Элемент 0: 1
Элемент 1: 2
Элемент 2: 3
Предупреждение: Элемент 3 не является числом: das
Элемент 4: 4
Элемент 5: 5

Сумма: 15.0
Среднее арифметическое: 3.0
Обработано элементов: 5 из 6
```

```java
import java.io.*;


public class CopyFileV1 {
    public static void main(String[] args) {


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


            int b;
            while ((b = fis.read()) != -1) {


                //if (b == 'X') { // при встрече буквы X — кидаем
ошибку
                //    throw new IOException("Искусственная ошибка
записи");
                //}


                fos.write(b);
```

```
            }

            fis.close();
            fos.close();

            System.out.println("Файл успешно скопирован!");

        } catch (FileNotFoundException e) {
            System.out.println("Ошибка открытия файла: " +
e.getMessage());
        } catch (IOException e) {
            System.out.println("Ошибка закрытия файла: " +
e.getMessage());
        }
    }
}
```

```
import java.io.*;

public class CopyFileV2 {
    public static void main(String[] args) {

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

            int b;
            while ((b = fis.read()) != -1) {
                fos.write(b);
```

```
                // if (b > 0) fos.close();
            }

            fis.close();
            fos.close();

            System.out.println("Файл успешно скопирован!");

        } catch (IOException e) {
            System.out.println("Ошибка чтения/записи: " +
e.getMessage());
        }
    }
}
```

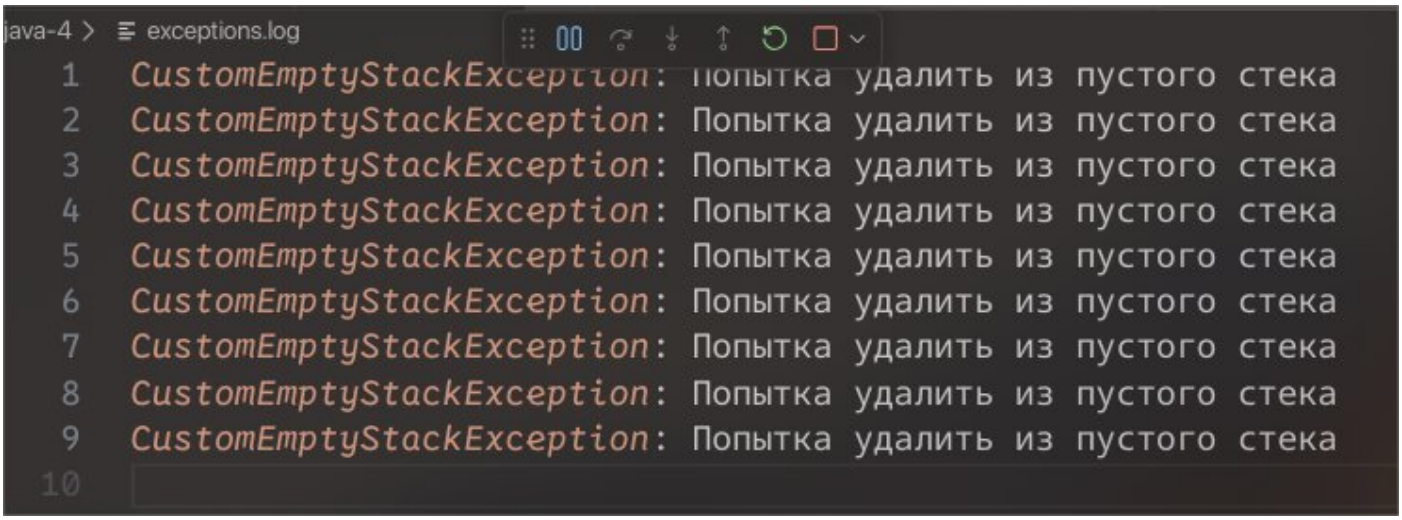
```
❯ java CopyFileV1
Ошибка открытия файла: input.txt (No such file or directory)
```

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

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

class ExceptionLogger {
```

```
    public static void log(Exception e) {
        try (FileWriter log = new FileWriter("exceptions.log",
true)) {
            log.write(e.toString() + "\n");
        } catch (IOException io) {
            System.out.println("Не удалось записать лог: " +
io.getMessage());
        }
    }
}

class CustomStack<T> {
    private ArrayList<T> stack = new ArrayList<>();

    public void push(T item) {
        stack.add(item);
    }

    public T pop() throws CustomEmptyStackException {
        if (stack.isEmpty()) {
            CustomEmptyStackException e = new
CustomEmptyStackException("Попытка удалить из пустого стека");
            ExceptionLogger.log(e);
            throw e;
        }
        return stack.remove(stack.size() - 1);
    }
}

public class StackApp {
    public static void main(String[] args) {
```

```
        CustomStack<Integer> stack = new CustomStack<>();
        stack.push(10);
        stack.push(20);
        try {
            stack.pop();
            stack.pop();
            stack.pop();
        } catch (CustomEmptyStackException e) {
            System.out.println("Ошибка: " + e.getMessage());
        }
    }
}
```

```
) java StackApp
Ошибка: Попытка удалить из пустого стека
```



Контрольные вопросы:

1. Исключение в Java – это объект, описывающий ошибочную или необычную ситуацию во время выполнения программы (например, деление на ноль, выход за границы массива, ошибка ввода/вывода).
2. Ключевые классы: **Throwable** (корень), его подклассы **Exception** и **Error**, далее **RuntimeException**, **IOException**, **NullPointerException**, **IndexOutOfBoundsException**, **ArithmeticException** и другие специализированные подклассы.
3. Проверяемые (checked) исключения – это подклассы **Exception**, не являющиеся наследниками **RuntimeException**; компилятор требует их обработать или

объявить через **throws**. Непроверяемые (unchecked) – это **RuntimeException** и его наследники, а также **Error**; компилятор не заставляет их явно обрабатывать.

4. Проверяемые исключения (например, **IOException**, **SQLException**) нужно либо перехватывать в **try-catch**, либо объявлять в сигнатуре метода через **throws**, чтобы вызывающий код их обработал. Непроверяемые обычно не объявляют в **throws**, а либо исправляют причину в коде, либо при необходимости перехватывают на верхнем уровне (например, логирование).

5. К классу **Error** относятся, например, **OutOfMemoryError**, **StackOverflowError**, **InternalError**, **VirtualMachineError**; их обычно не обрабатывают в обычном приложенческом коде, так как они сигнализируют о критических проблемах JVM или ресурсов. Теоретически их можно перехватить **catch (Error e)**, но чаще всего программа уже не может корректно продолжать работу.

6. К **RuntimeException** относятся **NullPointerException**, **IndexOutOfBoundsException**, **IllegalArgumentException**, **ArithmeticException**, **ClassCastException** и др.; это ошибки программирования, поэтому основной способ «обработки» – исправить логику кода, а не заворачивать всё в **try-catch**. При необходимости их можно перехватывать на границах слоя (например, в веб-контроллере) для логирования и вывода понятного сообщения пользователю.

7. Собственный класс исключения создаётся путём наследования от **Exception** (для checked) или от **RuntimeException** (для unchecked) и определения конструкторов, чаще всего с сообщением и/или причиной. Пример: **class MyException extends Exception { public MyException(String message) { super(message); } }**.

8. Исключения обрабатываются конструкциями **try-catch** и **try-catch-finally**, а также **try-with-resources**, где в **try** выполняется потенциально опасный код, в **catch** – реакция на конкретный тип исключения, а в **finally** – освобождение ресурсов, которое выполняется почти всегда.

9. Использовать **try** без **catch** или **finally** нельзя, но можно использовать **try** только с **finally**, либо использовать форму **try-with-resources** без явного **catch**, если исключение пробрасывается дальше.

10. Если исключение возникнет в блоке **finally**, оно заменит собой исходное исключение, возникшее ранее в **try** или **catch**, и именно оно «уйдёт» вверх по стеку. Это может скрыть первоначальную причину ошибки, поэтому в **finally** нужно писать минимальный и безопасный код.

11. Пробросить исключение выше по стеку можно, объявив его в сигнатуре метода через **throws** и не перехватывая внутри метода, либо явно перекинув

из **catch** через **throw**. Например: **void m() throws IOException { ... }** или **catch (IOException e) { throw e; }**.

12. **finally** – это блок, который выполняется после **try**/**catch** и в котором вручную закрывают ресурсы (файлы, потоки и т.п.). **try-with-resources** – это специальный синтаксис **try(...) {}**, который автоматически закрывает ресурсы, реализующие **AutoCloseable**, без явного **finally**, делая код короче и безопаснее.
13. В **try-with-resources** можно использовать классы, которые реализуют интерфейс **AutoCloseable** (или **Closeable**); это, например, потоки ввода/вывода, соединения с БД и т.п. **AutoCloseable** – это интерфейс с методом **close()**, который вызывается автоматически при выходе из блока **try-with-resources**.
14. В одном **try** можно использовать несколько **catch**-блоков, перехватывая разные типы исключений. Располагать их нужно от более конкретных (подклассы) к более общим (родительские классы), иначе общий **catch** перехватит всё раньше, и до частных блоков программа не дойдёт.
15. **throw** используется для фактического выбрасывания конкретного объекта исключения в коде (**throw new MyException("msg")**). **throws** используется в объявлении метода и показывает, какие типы исключений этот метод может выбросить и не обрабатывать сам.
16. **StackOverflowError** возникает при переполнении стека вызовов (например, бесконечной рекурсии), а **OutOfMemoryError** – при нехватке памяти в куче. Формально их можно перехватить как **Error**, но обычно после этого приложение не может продолжать работу, поэтому основной подход – предотвращать такие ситуации (исправлять рекурсию, настраивать память, оптимизировать использование объектов).

Вывод: в ходе лабораторной работы были закреплены практические навыки работы с механизмом исключений в Java: обработка ошибок при работе с массивами и файлами, различение проверяемых и непроверяемых исключений, а также создание собственных классов исключений и их логирование. Реализация задач с массивом, файловыми операциями и пользовательским стеком показала, как с помощью конструкций **try-catch-finally** и использования собственных исключений можно повысить надёжность программ и отделить основную логику от кода обработки ошибок.

https://github.com/zevy3/java-labs/tree/main/java-4