**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**
**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

Выполнил студент группы:
БПИ2401
Мещеряков Кирилл Владимрович
Руководитель:
Харрасов Камиль Раисович

Москва, 2025

**Цель работы:**

Изучить механизм обработки исключений в Java, научиться различать проверяемые и непроверяемые исключения, освоить практические навыки перехвата и обработки исключительных ситуаций с использованием конструкций try-catch-finally, а также приобрести опыт создания и декларирования собственных исключений для повышения надежности и устойчивости программного кода к ошибкам времени выполнения.

**Индивидуальное задание:**

Задание 1.

Написать программу, которая будет находить среднее ариф- метическое элементов массива. При этом программа должна об- рабатывать ошибки, связанные с выходом за границы массива и неверными данными (например, если элемент массива не явля- ется числом). Пример реализации метода:

1. 2. 3. 4. 5. 6. 7. 8. 9.

10. 11.

public class ArrayAverage {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5}; int sum = 0;

try {

// Обработка массива }

// Вывод массива

} catch (*исключение*) {

// Вывод исключения } catch (*исключение*) {

12. 13. 14. 15. }

// Вывод исключения }

}

Задание 2.

Написать программу, которая будет копировать содержимое одного файла в другой. При этом программа должна обрабаты- вать возможные ошибки, связанные:

• с открытием и закрытием файлов (вариант 1); • чтением и записью файлов (вариант 2).

Задание 3.

Создайте Java-проект для работы с исключениями. Для каждой из восьми указанных ниже задач напишите собственный класс для обработки исключений. Создайте обработчик ис- ключений, который логирует информацию о каждом вы- брошенном исключении в текстовый файл.

Вариант 6

Создайте класс CustomInputMismatchException, который будет использоваться для обработки исключения InputMismatch- Exception. Напишите программу, которая считывает целое число с клавиатуры, и, если ввод пользователя не является числом, выбрасывайте исключение CustomInputMismatchException.

**Выполнение:**

```
public class Task1 {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int sum = 0;

        try {
            // Обработка массива
            for (int i = 0; i <= arr.length; i++) { // Специально <= для демонстрации выхода за границы
                sum += arr[i];
            }

            double average = (double) sum / arr.length;
            System.out.println("Среднее арифметическое: " + average);

        } catch (ArrayIndexOutOfBoundsException e) {
            // Обработка выхода за границы массива
            System.out.println("Ошибка: выход за границы массива!");
            System.out.println("Сообщение исключения: " + e.getMessage());

            // Вычисляем корректное среднее
```

```
            sum = 0;
            for (int i = 0; i < arr.length; i++) {
                sum += arr[i];
            }
            double average = (double) sum / arr.length;
            System.out.println("Корректное среднее арифметическое: " +
average);

        } catch (NumberFormatException e) {
            // Обработка неверных данных
            System.out.println("Ошибка: элемент массива не является числом!");
            System.out.println("Сообщение исключения: " + e.getMessage());
        }

        System.out.println("\nСодержимое массива:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
import java.io.*;
```

```
public class Task2 {
    public static void main(String[] args) {
        String sourceFile = "source.txt";
        String destFile = "destination.txt";

        // Создаем исходный файл для демонстрации
        createSourceFile(sourceFile);

        // Вариант 1: обработка ошибок открытия и закрытия файлов
        copyFileVariant1(sourceFile, destFile);

        // Вариант 2: обработка ошибок чтения и записи файлов
        copyFileVariant2(sourceFile, destFile + ".v2");
    }

    // Вариант 1: Обработка ошибок открытия и закрытия файлов
    public static void copyFileVariant1(String source, String dest) {
        FileInputStream fis = null;
        FileOutputStream fos = null;
```

```
        try {
            System.out.println("\n=== Вариант 1: Обработка открытия/закрытия файлов ===");

            // Попытка открыть файлы
            fis = new FileInputStream(source);
            fos = new FileOutputStream(dest);

            int content;
            while ((content = fis.read()) != -1) {
                fos.write(content);
            }

            System.out.println("Файл успешно скопирован из " + source + " в " + dest);

        } catch (FileNotFoundException e) {
            System.out.println("Ошибка: файл не найден!");
            System.out.println("Детали: " + e.getMessage());
        } catch (IOException e) {
```

```
            System.out.println("Ошибка ввода-вывода при работе с файлом!");
            System.out.println("Детали: " + e.getMessage());
        } finally {
            // Закрытие файлов в блоке finally
            try {
                if (fis != null) {
                    fis.close();
                    System.out.println("Входной поток закрыт");
                }
                if (fos != null) {
                    fos.close();
                    System.out.println("Выходной поток закрыт");
                }
            } catch (IOException e) {
                System.out.println("Ошибка при закрытии файлов!");
                System.out.println("Детали: " + e.getMessage());
            }
        }
    }
```

```
    // Вариант 2: Обработка ошибок чтения и записи файлов
    public static void copyFileVariant2(String source, String dest) {
        System.out.println("\n=== Вариант 2: Обработка чтения/записи файлов ===");

        // Используем try-with-resources для автоматического закрытия ресурсов
        try (FileInputStream fis = new FileInputStream(source);
             FileOutputStream fos = new FileOutputStream(dest)) {

            byte[] buffer = new byte[1024];
            int length;

            try {
                while ((length = fis.read(buffer)) > 0) {
                    try {
                        fos.write(buffer, 0, length);
                    } catch (IOException e) {
                        System.out.println("Ошибка при записи в файл!");
                        System.out.println("Детали: " + e.getMessage());
                        throw e; // Пробрасываем исключение дальше
```

```
                }
            }
            System.out.println("Файл успешно скопирован из " + source + " в "
+ dest);

        } catch (IOException e) {
            System.out.println("Ошибка при чтении из файла!");
            System.out.println("Детали: " + e.getMessage());
        }

    } catch (FileNotFoundException e) {
        System.out.println("Ошибка: файл не найден!");
        System.out.println("Детали: " + e.getMessage());
    } catch (IOException e) {
        System.out.println("Общая ошибка ввода-вывода!");
        System.out.println("Детали: " + e.getMessage());
    }
}

// Вспомогательный метод для создания исходного файла
private static void createSourceFile(String filename) {
```

```
        try (FileWriter writer = new FileWriter(filename)) {
            writer.write("Это тестовое содержимое файла.\n");
            writer.write("Строка 2: Java Exception Handling.\n");
            writer.write("Строка 3: Лабораторная работа №4.\n");
            System.out.println("Исходный файл " + filename + " создан.");
        } catch (IOException e) {
            System.out.println("Ошибка при создании исходного файла: " +
e.getMessage());
        }
    }
}
public class CustomInputMismatchException extends Exception {
    private String inputValue;

    public CustomInputMismatchException(String message) {
        super(message);
    }

    public CustomInputMismatchException(String message, String inputValue) {
        super(message);
        this.inputValue = inputValue;
```

```
    }

    public String getInputValue() {
        return inputValue;
    }

    @Override
    public String toString() {
        return "CustomInputMismatchException: " + getMessage() +
                (inputValue != null ? " (введено: '" + inputValue + "')" : "");
    }
}
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class ExceptionLogger {
    private static final String LOG_FILE = "exceptions.log";
```

```
    private static final DateTimeFormatter formatter =
        DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    public static void logException(Exception e) {
        try (FileWriter fw = new FileWriter(LOG_FILE, true);
             PrintWriter pw = new PrintWriter(fw)) {

            pw.println("=".repeat(60));
            pw.println("Время: " + LocalDateTime.now().format(formatter));
            pw.println("Тип исключения: " + e.getClass().getName());
            pw.println("Сообщение: " + e.getMessage());
            pw.println("Stack trace:");
            e.printStackTrace(pw);
            pw.println();

            System.out.println("Исключение записано в лог-файл: " +
LOG_FILE);

        } catch (IOException ioException) {
            System.err.println("Ошибка при записи в лог-файл: " +
                               ioException.getMessage());
```

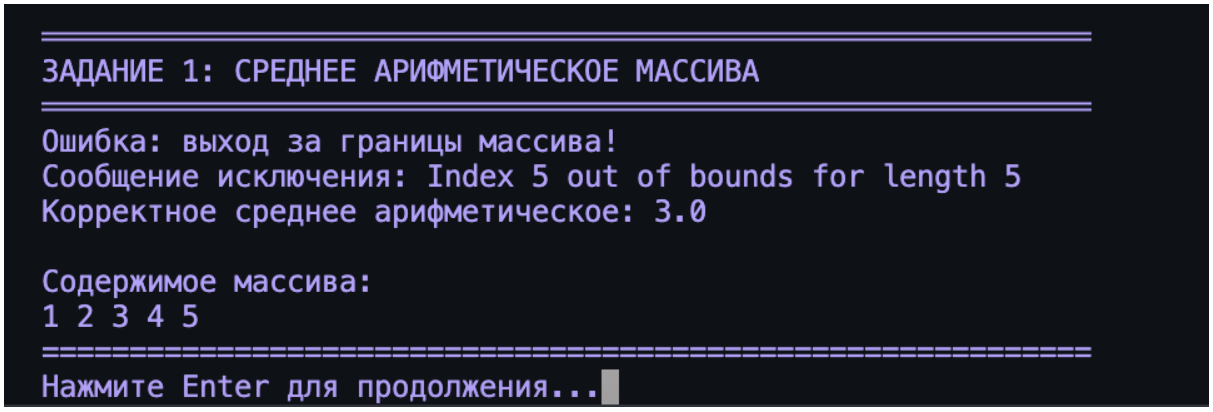
```
        }
    }

    public static void logException(Exception e, String additionalInfo) {
        try (FileWriter fw = new FileWriter(LOG_FILE, true);
             PrintWriter pw = new PrintWriter(fw)) {

            pw.println("=".repeat(60));
            pw.println("Время: " + LocalDateTime.now().format(formatter));
            pw.println("Тип исключения: " + e.getClass().getName());
            pw.println("Сообщение: " + e.getMessage());
            pw.println("Дополнительная информация: " + additionalInfo);
            pw.println("Stack trace:");
            e.printStackTrace(pw);
            pw.println();

            System.out.println("Исключение записано в лог-файл: " +
LOG_FILE);

        } catch (IOException ioException) {
            System.err.println("Ошибка при записи в лог-файл: " +
```

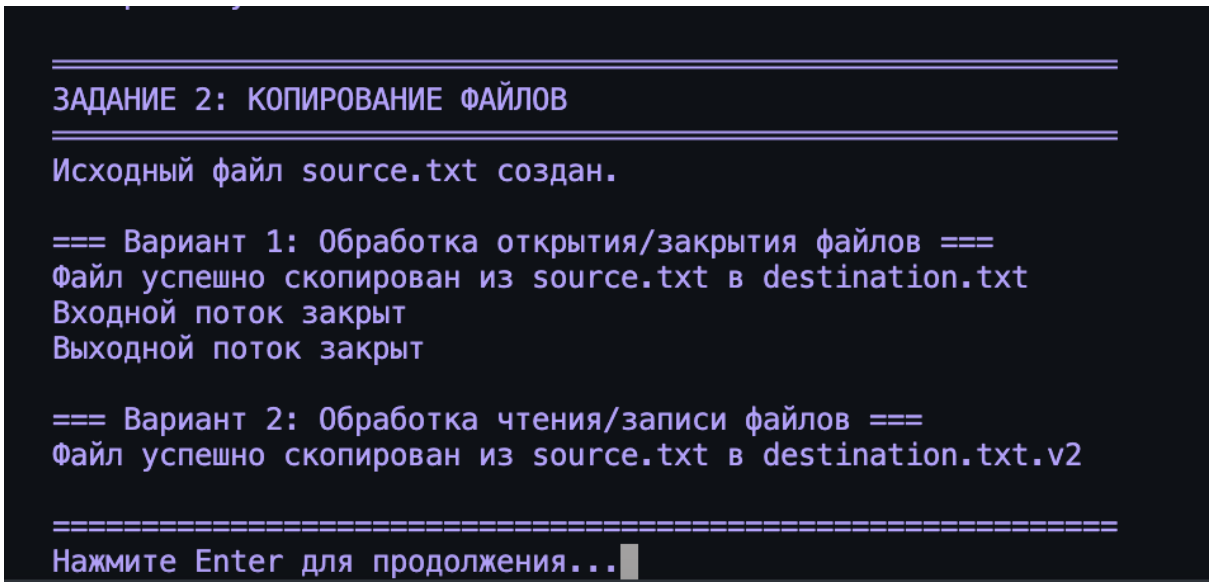
```
                    ioException.getMessage());
        }
    }
}
import java.util.Scanner;

public class Task3_Variant6 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("=== Программа проверки ввода целого числа ===");
        System.out.println("Введите целое число:");

        try {
            String input = scanner.nextLine();
            int number = readInteger(input);
            System.out.println("Успешно! Вы ввели число: " + number);
            System.out.println("Квадрат числа: " + (number * number));

        } catch (CustomInputMismatchException e) {
```

```
            System.err.println("\n✗ " + e);
            ExceptionLogger.logException(e, "Пользователь ввел некорректные данные");
            System.out.println("\nПопробуйте еще раз с корректным целым числом.");
        }

        // Демонстрация различных случаев
        System.out.println("\n" + "=".repeat(50));
        System.out.println("Демонстрация различных случаев:");
        System.out.println("=".repeat(50));

        testInput("123");      // Корректное число
        testInput("abc");      // Буквы
        testInput("12.5");     // Дробное число
        testInput("");         // Пустая строка
        testInput("  456  ");  // Число с пробелами
        testInput("99999999999999999"); // Слишком большое число

        scanner.close();
    }
```

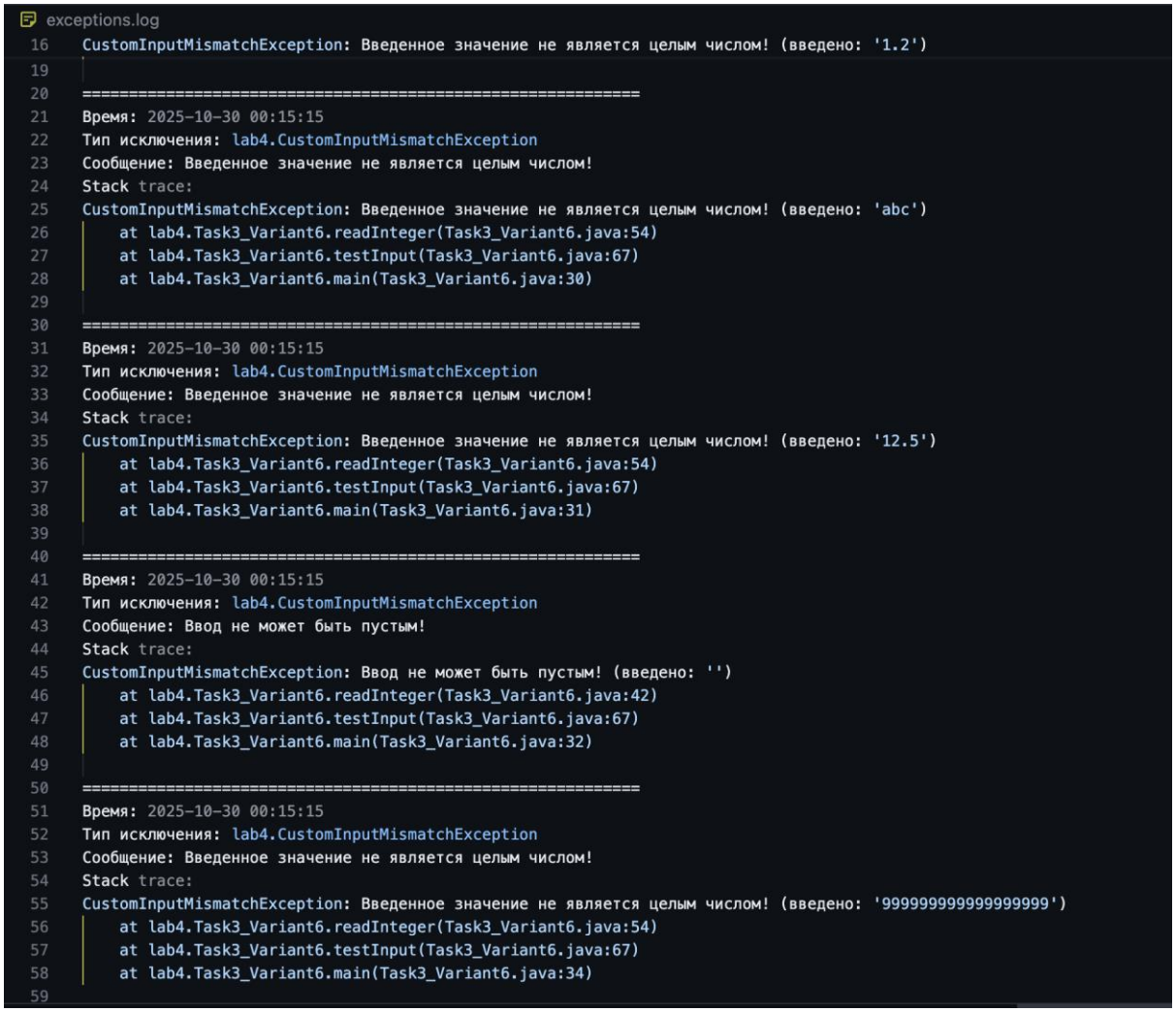
```
    public static int readInteger(String input) throws
CustomInputMismatchException {
        if (input == null || input.trim().isEmpty()) {
            CustomInputMismatchException exception =
                new CustomInputMismatchException(
                    "Ввод не может быть пустым!",
                    input
                );
            ExceptionLogger.logException(exception);
            throw exception;
        }

        try {
            return Integer.parseInt(input.trim());
        } catch (NumberFormatException e) {
            CustomInputMismatchException exception =
                new CustomInputMismatchException(
                    "Введенное значение не является целым числом!",
                    input
                );
```

```
            ExceptionLogger.logException(exception);
            throw exception;
        }
    }

    // Вспомогательный метод для тестирования
    private static void testInput(String input) {
        System.out.println("\nТест ввода: \"" + input + "\"");
        try {
            int number = readInteger(input);
            System.out.println("✓ Результат: " + number);
        } catch (CustomInputMismatchException e) {
            System.out.println("✗ " + e.getMessage());
        }
    }
}
```


ЗАДАНИЕ 1: СРЕДНЕЕ АРИФМЕТИЧЕСКОЕ МАССИВА
Ошибка: выход за границы массива!
Сообщение исключения: Index 5 out of bounds for length 5
Корректное среднее арифметическое: 3.0
Содержимое массива:
1 2 3 4 5
Нажмите Enter для продолжения...


ЗАДАНИЕ 2: КОПИРОВАНИЕ ФАЙЛОВ
Исходный файл source.txt создан.
=== Вариант 1: Обработка открытия/закрытия файлов ===
Файл успешно скопирован из source.txt в destination.txt
Входной поток закрыт
Выходной поток закрыт
=== Вариант 2: Обработка чтения/записи файлов ===
Файл успешно скопирован из source.txt в destination.txt.v2
Нажмите Enter для продолжения...

```
=== Программа проверки ввода целого числа ===
Введите целое число:
1.2
Исключение записано в лог-файл: exceptions.log

❌ CustomInputMismatchException: Введенное значение не является целым числом! (введено: '1.2')
Исключение записано в лог-файл: exceptions.log

Попробуйте еще раз с корректным целым числом.

==================================================
Демонстрация различных случаев:
==================================================

Тест ввода: "123"
✓ Результат: 123

Тест ввода: "abc"
Исключение записано в лог-файл: exceptions.log
✗ Введенное значение не является целым числом!

Тест ввода: "12.5"
Исключение записано в лог-файл: exceptions.log
✗ Введенное значение не является целым числом!

Тест ввода: ""
Исключение записано в лог-файл: exceptions.log
✗ Ввод не может быть пустым!

Тест ввода: "  456  "
✓ Результат: 456

Тест ввода: "9999999999999999999"
Исключение записано в лог-файл: exceptions.log
✗ Введенное значение не является целым числом!
```




**Контрольные вопросы:**

1. Исключение в Java — это событие, возникающее во время выполнения программы и нарушающее её нормальный ход; оно представляет собой объект, описывающий ошибку или необычную ситуацию.

2. Основные классы исключений — это Throwable, от которого наследуются Exception и Error. Ключевые подклассы включают IOException, RuntimeException, NullPointerException, ArithmeticException и др.

3. Проверяемые (checked) исключения — это те, которые необходимо явно обрабатывать (IOException, SQLException), а непроверяемые (unchecked) — потомки RuntimeException, их можно не обрабатывать (NullPointerException, IllegalArgumentException).

4. Проверяемые исключения нужно либо обрабатывать в try-catch, либо объявлять через throws, а непроверяемые — обрабатывать по необходимости, чтобы предотвратить падение программы.

5. Исключения класса Error (например, OutOfMemoryError, StackOverflowError) связаны с критическими ошибками JVM, обычно не подлежат обработке, так как свидетельствуют о проблемах на уровне системы.

6. К RuntimeException относятся ошибки программирования — NullPointerException, IndexOutOfBoundsException, IllegalStateException; их можно обрабатывать, но чаще исправляют саму причину.

7. Собственный класс исключения создаётся путём наследования от Exception или RuntimeException и добавления конструктора с сообщением об ошибке.

8. Исключения в Java обрабатываются с помощью конструкций try, catch, finally и throw; в try помещается рискованный код, catch перехватывает ошибку, finally выполняется всегда.

9. Да, можно использовать try без catch, но обязательно с finally или в виде try-with-resources.

10. Если в блоке finally произойдёт исключение, оно перекроет все предыдущие исключения, и именно оно будет выброшено наружу.

11. Чтобы пробросить исключение выше, используется оператор throw вместе с объявлением throws в сигнатуре метода.
12. finally просто выполняет код по завершении try, а try-with-resources автоматически закрывает ресурсы, реализующие AutoCloseable.
13. В try-with-resources можно использовать классы, реализующие интерфейс AutoCloseable (например, FileInputStream, Scanner, BufferedReader), который определяет метод close().
14. Да, можно использовать несколько catch; их размещают от более конкретных к более общим, иначе компилятор выдаст ошибку.
15. throw используется для генерации исключения, а throws — для объявления, что метод может его выбросить.
16. StackOverflowError возникает при переполнении стека (например, из-за бесконечной рекурсии), а OutOfMemoryError — при нехватке памяти; теоретически их можно поймать, но обычно программа не может корректно восстановиться.

**Заключение**

В ходе выполнения лабораторной работы были изучены и применены на практике основные механизмы обработки исключений в Java. Реализованы программы с использованием конструкций try-catch-finally для обработки различных типов ошибок времени выполнения.

В первом задании была разработана программа вычисления среднего арифметического элементов массива с обработкой исключений ArrayIndexOutOfBoundsException и NumberFormatException, что позволило предотвратить аварийное завершение программы при выходе за границы массива.

Во втором задании реализованы два варианта копирования файлов с различными подходами к обработке исключений: обработка ошибок открытия/закрытия файлов с использованием блока finally и обработка ошибок чтения/записи с применением конструкции try-with-resources, что продемонстрировало различные стратегии управления ресурсами.

В третьем задании (вариант 6) создан собственный класс исключения CustomInputMismatchException для валидации пользовательского ввода целых чисел, а также реализован универсальный логгер исключений ExceptionLogger, записывающий информацию о всех возникших исключениях в текстовый файл с временными метками и полным stack trace.

Полученные навыки работы с исключениями позволяют создавать более надежные и отказоустойчивые приложения, корректно обрабатывающие ошибочные ситуации и предоставляющие пользователю понятные сообщения об ошибках.

Ссылка на GitHub репозиторий: https://github.com/M1ke0-0/ITiP