

**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ
КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**
**Ордена труда 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
Трухина Анастасия Александровна
Проверил: Харрасов Камиль Раисович

Москва,
2025

[Оглавление](#)

Цель работы:	3
Задание:	3
Основная часть	3
Задание 1:	3
Задание 2	4
Ответы на контрольные вопросы:	6
Заключение	Error! Bookmark not defined.

Цель работы:

Изучение механизма обработки исключений в Java, включая создание пользовательских классов исключений, обработку стандартных исключений (проверяемых и непроверяемых), а также применение конструкций `try-catch`, `finally` и `throws`.

Закрепление навыков обработки ошибок ввода-вывода, работы с массивами, файлами и пользовательскими данными.

Задание:

1. Реализовать программу для вычисления среднего арифметического элементов массива с обработкой исключений (выход за границы массива, нечисловые данные).
2. Написать программу копирования файла с обработкой ошибок:
 - Вариант 1: Ошибки открытия/закрытия файлов.
 - Вариант 2: Ошибки чтения/записи.
3. Создать Java-проект с восемью пользовательскими классами исключений для обработки специфических ошибок (деление на ноль, неверный возраст, некорректный email и др.) и логированием информации об исключениях в файл.

Основная часть

Задание 1:

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows a Java project named "javaLaba1" with several subfolders like ".idea", "out", "src", and "Laba1-Laba4". A file named "ArrayAverage.java" is selected and highlighted in blue.
- Code Editor:** Displays the content of "ArrayAverage.java". The code calculates the average of an array of integers, handling potential exceptions for array bounds and non-integer values.
- Run Tab:** Shows the command used to run the program: "C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=5000,127.0.0.1:5000" "javaLaba1".
- Output Tab:** Displays the run results:
 - An error message: "Ошибка: выход за границы массива - Index 5 out of bounds for length 5".
 - The calculated average: "Среднее арифметическое без ошибочных значений: 3".
 - A final message: "Нет значений, которые не вызывали бы ошибки".

Задание 2

The screenshot shows the IntelliJ IDEA interface with the code editor open. The project navigation bar at the top shows the current file as `Task2.java`. The code implements two methods: `openCloseException()` and `main()`. The `openCloseException()` method demonstrates how to handle a `FileNotFoundException` when trying to close a file. The `main()` method shows how to write to a file named `fileForText.txt` located at the path `C:\Users\Владимир\IdeaProjects\javaLaba1\src\Lab4\fileForText.txt`. A tooltip in the bottom right corner says "Everything is up to date Create pull request".

```
package Laba4;

import java.io.*;

public class Task2 { new {
    public static void openCloseException(){ no usages new }

}

public static void main(String[] Args){ new *
    String path = "C:\\Users\\Владимир\\IdeaProjects\\javaLaba1\\src\\Lab4\\fileForText.txt";
    InputStream in = null;

    // Обработка открытия\\закрытия файла - Вариант 1

    try {
        in = new FileInputStream(path);
        in.close();
    } catch (FileNotFoundException e){
        System.err.println("Файл по пути " + path + " не найден. \\n Ошибка: " + e.getMessage());
    } catch (IOException e) {
        System.err.println("Ошибка (в данном случае) при закрытии файла: " + e.getMessage());
    }

    // Обработка чтения и записи - Вариант 2

    String writePath = "C:\\Users\\Владимир\\IdeaProjects\\javaLaba1\\src\\Lab4\\fileForWrite.txt";
    String text = "Hello world!";

    try{FileOutputStream output = new FileOutputStream(writePath){
        byte[] buffer = text.getBytes();

        output.write(buffer, off: 0, buffer.length);
    } catch (FileNotFoundException e) {
```

The screenshot shows the IntelliJ IDEA interface with the code editor open. The project navigation bar at the top shows the current file as `Task2.java`. The code is identical to the one in the previous screenshot. Below the code editor, the run output window shows the command used to run the program: `"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=61893:C:\Program Files\`. The output window also displays the results of the program execution, which include some internal logs and the message "Process finished with exit code 0".

```
public class Task2 { new *
    public static void main(String[] Args){ new *
        byte[] buffer = text.getBytes();

        output.write(buffer, off: 0, buffer.length);
    } catch (FileNotFoundException e) {
        System.err.println("Файл по пути " + writePath + " не найден. \\n Ошибка: " + e.getMessage());
    } catch (IOException e) {
        System.err.println("Произошла ошибка во время чтения или записи: " + e.getMessage());
    }

    String readPath = "C:\\Users\\Владимир\\IdeaProjects\\javaLaba1\\src\\Lab4\\fileForText.txt";
    try( FileInputStream input = new FileInputStream(readPath)){
        int i;
        while((i = input.read()) != -1){
            System.out.print((char) i);
        }
    } catch(FileNotFoundException e){
        System.err.println("Файл по пути " + readPath + " не найден. \\n Ошибка: " + e.getMessage());
    } catch(IOException e){
        System.err.println("Произошла ошибка во время чтения или записи: " + e.getMessage());
    }
}
```

Run Task2 x

"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=61893:C:\Program Files\

1. -----
2. Some Info...
3. Empty.
Process finished with exit code 0

Задание 3

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project:** javaLaba1
- Current File:** CustomNumberFormatException.java
- Code:** The code defines a custom exception class `CustomNumberFormatException` that extends `Exception`. It includes a constructor that takes an `String invalidInput` and a `Throwable cause`. A static method `parseStringToInt` takes a `String str` and throws a `CustomNumberFormatException` if the string cannot be converted to an integer. The `main` method demonstrates this by trying to parse the string "123456" and then "bacdef".
- Run:** The run configuration is set to "CustomNumberFormatException".
- Output:** The output window shows the command "C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=61899:C:\Program Files\Java\jdk-22\bin\java.exe" and the result "Итог: 123456". An error message "Ошибка: Невозможно преобразовать строку 'bacdef' в целое число." is displayed.
- Status Bar:** Shows the time as 13:11, encoding as CRLF, character set as UTF-8, and 4 spaces.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project:** javaLaba1
- Current File:** CustomNumberFormatException.java
- Code:** The code is identical to the first screenshot, defining the `CustomNumberFormatException` class and its methods.
- Run:** The run configuration is set to "CustomNumberFormatException".
- Output:** The output window shows the command "C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=61899:C:\Program Files\Java\jdk-22\bin\java.exe" and the result "Итог: 123456". An error message "Ошибка: Невозможно преобразовать строку 'bacdef' в целое число." is displayed.
- Status Bar:** Shows the time as 17:32, encoding as CRLF, character set as UTF-8, and 4 spaces.

Заключение

Вывод:

В ходе лабораторной работы были освоены принципы обработки исключений в Java. На практике реализованы механизмы обработки стандартных исключений и созданы

пользовательские классы исключений для конкретных сценариев. Использованы блоки try-catch-finally для безопасного выполнения кода, а также ключевые слова throw\throws для генерации и передачи исключений. Логирование ошибок в файл продемонстрировало важность отслеживания исключений в реальных приложениях. Работа подтвердила, что корректная обработка исключений повышает надежность и устойчивость программ к ошибкам.

Ответы на контрольные вопросы:

1. Исключение в Java - это событие, возникающее во время выполнения программы, которое нарушает нормальный поток инструкций. Оно представляет собой объект, описывающий ошибку или нестандартную ситуацию. Обработка исключений позволяет изолировать код, способный вызвать ошибку, и корректно обработать её, предотвращая аварийное завершение программы.
2. Ключевые классы исключений:
 - o Throwable - базовый класс для всех исключений и ошибок.
 - o Error (непроверяемые): OutOfMemoryError, StackOverflowError (критические ошибки JVM).
 - o Exception (проверяемые и непроверяемые):
 - Проверяемые: IOException, SQLException.
 - Непроверяемые (RuntimeException): NullPointerException, ArrayIndexOutOfBoundsException, IllegalArgumentException.
3. Проверяемые исключения (checked) должны быть объявлены в сигнатуре метода через throws или обработаны в try-catch. Компилятор проверяет их наличие. Пример: IOException. Непроверяемые исключения (unchecked) - это RuntimeException и Error. Их обработка не обязательна, компилятор не требует явного указания.
4. Обработка исключений осуществляется через блок try-catch. Код, который может вызвать исключение, помещается в try. Если исключение возникает, управление передается в соответствующий catch. Обязательно обрабатывать проверяемые исключения. Непроверяемые обычно связаны с логическими ошибками и обрабатываются по необходимости.
5. Класс Error включает критические ошибки JVM (например, OutOfMemoryError). Их обрабатывать не рекомендуется, так как они указывают на проблемы, которые программа не может исправить. Попытка обработки может быть бесполезной.
6. RuntimeException (непроверяемые) возникают из-за ошибок в логике программы (например, деление на ноль, обращение к null). Обрабатываются через try-catch, но лучше предотвращать их проверкой условий.
7. Создание собственного класса исключения: нужно унаследоваться от Exception (для проверяемых) или RuntimeException (для непроверяемых). Пример:

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

```
}
```

```
}
```

8. Обработка исключений в Java использует конструкции:

- try - блок с кодом, который может вызвать исключение.
- catch - блок для перехвата и обработки исключений.
- finally - блок, выполняющийся всегда (даже при исключении).
- throw - генерация исключения.
- throws - объявление исключений в сигнатуре метода.

9. Try без catch или finally можно использовать только с блоками ресурсов (try-with-resources). В противном случае компилятор выдаст ошибку. Без catch или finally блок try теряет смысл.

10. Исключение в блоке finally: если исключение возникает в finally, оно "замещает" исходное исключение из try или catch. Важно аккуратно обрабатывать код в finally, чтобы не потерять информацию о первоначальной ошибке.

11. Проброс исключения выше осуществляется через ключевое слово throws в объявлении метода. Например:

```
public void readFile() throws IOException {  
    // код, который может вызвать IOException  
}
```

12. Разница между finally и try-with-resources:

- finally - блок для кода, который должен выполниться независимо от исключений.
- try-with-resources (доступен с Java 7) автоматически закрывает ресурсы, реализующие интерфейс AutoCloseable. Избавляет от необходимости писать finally для закрытия ресурсов.

13. Классы для try-with-resources должны реализовывать интерфейс AutoCloseable (или его подинтерфейс Closeable). Пример: FileInputStream, Scanner. AutoCloseable требует реализации метода close(), который вызывается автоматически.

14. Несколько catch блоков в одном try допустимы. Их нужно располагать от более конкретных исключений к более общим. Например, сначала FileNotFoundException, затем IOException. Если сначала указать суперкласс, компилятор выдаст ошибку.

15. Разница между throw и throws:

- throw - оператор для генерации исключения внутри метода.
- throws - ключевое слово в сигнатуре метода для объявления возможных исключений.

16. StackOverflowError возникает при бесконечной рекурсии (переполнение стека вызовов). OutOfMemoryError - при нехватке памяти в heap. Оба относятся к Error и обрабатывать их не рекомендуется, так как это обычно свидетельствует о критических проблемах в коде или окружении.

ГитХаб с файлами кода: [NT-005-TN/ITiP_LabWorks_Trukhina](#)