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»

Лабораторная работа № 4

Исключения в Java

Выполнил: Студент группы

БВТ2303

Ситникова Дарья

Вариант 2

Москва

2024

Цель работы: Изучить исключения в Java и выполнить задания по теме.

Задачи:

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).
2. Написать программу, которая будет копировать содержимое одного файла в другой. При этом программа должна обрабатывать возможные ошибки, связанные с чтением и записью файлов.
3. Создать Java-проект для работы с исключениями.

Краткая теория:

Исключения в Java — это способ обработки ошибок и неожиданных ситуаций, которые могут возникнуть во время работы программы.

Когда программа выполняется, могут произойти различные ошибки. Например, программа может попытаться открыть файл, который не существует, или обратиться к элементу массива, который выходит за его пределы. Чтобы справиться с такими ситуациями, Java использует механизм исключений.

Исключения помогают:

1. **Изолировать ошибки**: Вместо того чтобы останавливать всю программу, ошибки можно обработать в специальном блоке кода.
2. **Улучшить надежность**: Программа становится более устойчивой к ошибкам, так как можно заранее предусмотреть, что может пойти не так.

В Java есть много различных типов исключений. Например:

- IOException: используется для обработки ошибок, связанных с вводом-выводом (например, при работе с файлами).
- FileNotFoundException: возникает, когда программа пытается открыть файл, который не существует.
- NullPointerException: возникает, когда программа пытается использовать объект, который равен null.
- ArrayIndexOutOfBoundsException: возникает, когда программа пытается получить доступ к элементу массива по индексу, который выходит за пределы массива.

Можно создать собственные исключения, если стандартные не подходят для текущей ситуации. Для этого нужно создать класс, который наследуется от класса Exception или его подклассов.

Для обработки исключений в Java используется конструкция try-catch. Вот как это работает:

1. **Блок try**: В этом блоке вы пишете код, который может вызвать исключение.
2. **Блок catch**: Если в блоке try возникает исключение, управление передается в блок catch, где вы можете обработать это исключение.

Пример:

```
try {
    // Код, который может вызвать исключение
} catch (ExceptionType1 e) {
    // Обработка исключения типа ExceptionType1
} catch (ExceptionType2 e) {
    // Обработка исключения типа ExceptionType2
} finally {
    // Код, который выполняется всегда, независимо от того, возникло ли
исключение или нет
}
```

Блок finally является необязательным, но если он присутствует, код внутри него будет выполнен всегда, независимо от того, возникло ли исключение или нет. Это полезно для выполнения завершающих действий, таких как закрытие файлов или освобождение ресурсов.

Ход работы:

Задание 1. Необходимо 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).

Создается класс ArrayAverage.

В методе main создается и инициализируется массив целых чисел пятью элементами.

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

Код, который может вызвать исключения, помещается в блок try. Здесь вызывается метод calculateAverage. Ему передается массив и выводится результат.

```
try {
      double average = calculateAverage(arr);
      System.out.println("Среднее арифметическое: " + average);
```

Далее обрабатываются различные типы исключений, которые могут возникнуть в блоке try. Обработка исключений позволяет программе продолжать выполнение, даже если произошла ошибка. Будет выведена понятное сообщение об ошибке для пользователя.

```
} catch (ArrayIndexOutOfBoundsException e) {
      // Обрабатываем исключение, если произошло обращение к несуществующему
индексу массива
      System.out.println("Ошибка: Выход за границы массива.");
} catch (NumberFormatException e) {
      // Обрабатываем исключение, если в массиве есть неверные данные
(нечисловые)
      System.out.println("Оштбка: Неверные данные в массиве.");
} catch (Exception e) {
      // Обрабатываем любые другие исключения, которые могут возникнуть
```

```
        System.out.println("Ошибка: " + e.getMessage());
}
```

Метод calculateAverage принимает массив целых чисел и возвращает среднее арифметическое. Если массив пустой, то выбрасывается исключение ArithmeticException.

```
public static double calculateAverage(int[] arr) {
        if (arr.length == 0) {
                throw new ArithmeticException("Массив пустой, невозможно вычислить среднее.");
        }
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
                sum += arr[i];
        }
        return (double) sum / arr.length;
```

Вывод в терминале:

```
PS C:\Users\manul\Desktop\ITiP\Lab 4> java ArrayAverage.java
Среднее арифметическое: 3.0
```

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

Для начала импортируем необходимые классы:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
```

- import java.io.BufferedReader; : импортирует класс BufferedReader, который позволяет читать текстовые данные из входного потока (например, из файла) с использованием буфкризации. Это делает чтение более эффективным, так как данные считываются большими блоками.
- import java.io.FileReader; : импортирует класс FileReader, который предназначен для чтения символов из файла. Он позволяет открывать файл и считывать его содержимое.

- import java.io.FileWriter; : импортирует класс FileWriter, который используется для записи симфолов в файл. Он позволяет создавать новый файл или перезаписывать уже существующий.
- import java.io.IOException; : импортирует класс IOException, которые представляет собой исключение, возникающее при ошибках ввода-вывода. Это исключение будет использоваться для обработки ошибок, связанных с чтением и записью файлов.

Создается класс FileCopy и основной метод main.

Определяем строковую переменную sourceFile, которая содержит имя исходного файла из которого будут копироваться данные. И строковую переменную destinationFile, которая содержит имя целевого файла, в котором будут записываться данные.

```
String sourceFile = "manul.txt";
String destinationFile = "PalasCat.txt";
```

Начинаем блок try, в котором будет выполняться код, который может вызывать исключение. Если в этом блоке произошла ошибка, управление перейдет в сообветствующй блок catch.

```
try {
      copyFile(sourceFile, destinationFile);
      System.out.println("Копирование завершено успешно.");
} catch (IOException e) { // Обработка ошибок ввода-вывода
      System.out.println("Ошибка при копировании файла: " + e.getMessage());
} catch (Exception e) { // Обработка любых других исключений
      System.out.println("Произошла ошибка: " + e.getMessage());
}
```

Определение метода copyFile:

```
public static void copyFile(String source, String destination) throws IOException
{
```

Определяем метод copyFile, который принимает два параметра: source (путь к исходному файлу) и destination (путь к целевому файлу). Метод объявлен с throws IOException, что означает, что он может выбрасывать исключение IOException, которое должно быть обработано вызывающим кодом.

```
try (BufferedReader reader = new BufferedReader(new FileReader(source));
```

```
FileWriter writer = new FileWriter(destination)) {
```

Создаем объект BufferedReader, который будет использоваться для чтения данных из исходного файла. Оборачиваем FileReader в BufferedReader для более эффективного чтения.

Создаем объект FileWriter, который будет использоваться для записи данных в целевой файл. Если файл не существует, он будет создан. Если файл уже существует, его содержимое будет перезаписано.

Использование try-with-resources гарантирует, что ресурсы (в данном случае reader и writer) будут автоматически закрыты после завершения работы блока try, даже если возникнет исключение.

Объявляем переменную line, которая будет использоваться для хранения каждой строки, считанной из исходного файла.

```
String line;

while ((line = reader.readLine()) != null)
```

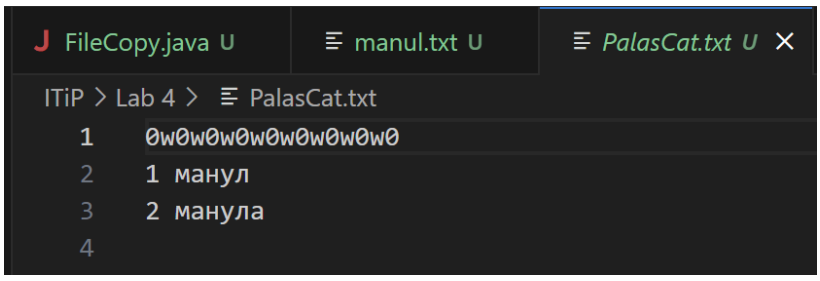
Цикл while используется для чтения файла построчно. Метод readLine() возвращает следующую строку из файла или null, если достигнут конец файла. Считанная строка присваивается переменной line.

```
writer.write(line);
writer.write(System.lineSeparator());
```

Записываем перевод строки в целевой файл. Метод System.lineSeparator() возвращает строку, представляющую символ новой строки, в зависимости от операционной системы (например, \n для Unix/Linux и \r\n для Windows).

Вывод в терминале:

```
PS C:\Users\manul\Desktop\ITiP\Lab 4> java FileCopy.java
Копирование завершено успешно.
```



Задание 3. Создайте Java-проект для работы с исключениями. Напишите свой собственный класс для обработки исключений. Создайте обработчик исключений, который логирует информацию о каждом выброшенном исключении в текстовый файл

Вариант 2: Создайте класс CustomAgeException, который будет использоваться для обработки недопустимых возрастов. Реализуйте программу, которая проверяет возраст пользователя с использованием этого класса, и, если возраст меньше 0 или больше 120, выбрасывайте исключение CustomAgeException.

Создаем новый класс CustomAgeException, который наследуется от стандартного класса Exception. Это означает, что CustomAgeException будет явдяться пользовательстким исключением, которое можно выбрасывать и обрабатывать.

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

public CustomAgeException(String message) { : конствуктор класса, который принимает строку message. Это чообщение будет описывать причину возникновения исключения.

super(message);: вызов конструктора родительсткого класса Exception с передачей сообщения. Это позволяет сохранить сообщение об ошибке, которое можно будет получить позже с помощью вывода getMessage().

Класс ExceptionLogger:

Импортируем необходимые классы:

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
```

- import java.io.BufferedWriter;: Импортируем класс BufferedWriter, который позволяет записывать текстовые данные в файл с буферизацией, что делает запись более эффективной.
- import java.io.FileWriter;: Импортируем класс FileWriter, который используется для записи символов в файл.
- import java.io.IOException;: Импортируем класс IOException для обработки ошибок ввода-вывода.
- import java.time.LocalDateTime;: Импортируем класс LocalDateTime, который позволяет работать с датой и временем.
- import java.time.format.DateTimeFormatter;: Импортируем класс DateTimeFormatter, который используется для форматирования даты и времени.

public class ExceptionLogger {: Определяем класс ExceptionLogger, который будет отвечать за логирование исключений.

```
public class ExceptionLogger {
    private static final String LOG_FILE = "C:\\Users\\manul\\Desktop\\ITiP\\Lab
4\\n_3\\log\\exceptions.txt";
```

private static final String LOG_FILE = "C:\\…\\exceptions.txt";: Определяем константу LOG_FILE, которая содержит имя файла, в который будут записываться логи исключений. static означает, что это поле принадлежит классу, а не экземпляру, а final означает, что значение не может быть изменено.

```
public static void logException(Exception e) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(LOG_FILE,
true))) {
            String timestamp =
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
            writer.write(timestamp + " - " + e.getMessage());
            writer.newLine();
        } catch (IOException ioException) {
            System.out.println("Ошибка при записи в лог-файл: " +
ioException.getMessage());
        }
    }
```

public static void logException(Exception e) {: Определяем статический метод logException, который принимает объект исключения e в качестве параметра. Этот метод будет записывать информацию об исключении в лог-файл.

Конструкция try-with-resources используется, чтобы автоматически закрыть BufferedWriter после завершения работы. Создаем объект BufferedWriter, который оборачивает FileWriter. Параметр true в FileWriter указывает, что мы хотим добавлять данные в конец файла, а не перезаписывать его.

Получаем текущее время и форматируем его в строку с помощью DateTimeFormatter. Формат даты и времени будет "год-месяц-день часы:минуты:секунды".

Записываем в лог-файл строку, содержащую временную метку и сообщение об исключении, полученное с помощью метода getMessage().

writer.newLine(); : Записываем перевод строки, чтобы следующая запись начиналась с новой строки.

Если возникает ошибка при записи в лог-файл, выводим сообщение об ошибке на консоль.

Класс AgeValidator:

```
import java.util.Scanner;
```

Импортируем класс Scanner, который используется для считывания пользовательского ввода.

Определяем класс App, который будет содержать основной код программы.

```
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Введите ваш возраст: ");
```

Создаем объект Scanner, который будет считывать данные с консоли.

```
try {
      int age = scanner.nextInt();
      validateAge(age);
      System.out.println("Ваш возраст: " + age);
```

Считываем введенное пользователем значение и сохраняем его в переменной age. Если пользователь введет нечисловое значение, это вызовет исключение.

Если возраст допустим, выводим его на консоль.

```
} catch (CustomAgeException e) {
      ExceptionLogger.logException(e);
      System.out.println("Ошибка: " + e.getMessage());
```

Начинаем блок catch, который будет обрабатывать исключение CustomAgeException, если оно было выброшено в блоке try.

Логируем информацию об исключении, используя метод logException из класса ExceptionLogger.

```
} catch (Exception e) {
      ExceptionLogger.logException(e);
      System.out.println("Произошла ошибка: " + e.getMessage());
```

Начинаем блок catch, который будет обрабатывать любые другие исключения, которые могут возникнуть в блоке try.

```
} finally {
      scanner.close();
}
```

Блок finally выполняется всегда, независимо от того, произошло ли исключение или нет.

Закрываем объект Scanner, чтобы освободить ресурсы.

```
public static void validateAge(int age) throws CustomAgeException {
      if (age < 0 || age > 120) {
            throw new CustomAgeException("Недопустимый возраст: " + age);
      }
}
```

Определяем метод validateAge, который принимает возраст в качестве параметра и может выбрасывать CustomAgeException.

Если возраст недопустим, выбрасываем исключение CustomAgeException с сообщением о недопустимом возрасте.

Вывод в терминале:

```
PS C:\Users\manul\Desktop\ITiP\Lab 4\n3\src> java App.java
Введите ваш возраст: 27
Ваш возраст: 27
```

```
PS C:\Users\manul\Desktop\ITiP\Lab 4\n3\src> java App.java
Введите ваш возраст: fff
Произошла ошибка: null
```

```
PS C:\Users\manul\Desktop\ITiP\Lab 4\n3\src> java App.java
Введите ваш возраст: 666
Ошибка: Недопустимый возраст: 666
```

ArrayAverage.java U | FileCopy.java U | *exceptions.txt U* | manul.txt U

ITiP > Lab 4 > n3 > log > exceptions.txt

```
1
2  2024-10-27T20:38:06.964665400 - Недопустимый возраст: 333
3  2024-10-27T20:38:25.395942 - Недопустимый возраст: 222
4  2024-10-27T20:38:59.018223 - Недопустимый возраст: 666
5  2024-10-27 21:07:38 - null
6  2024-10-27 21:13:30 - Недопустимый возраст: 666
```

Вывод: В ходе лабораторной работы юыло знакомство с исключениями в Java на примере нескольких типов исключений. Выполнены задания по теме.

Приложение

Полный листинг кода:

Задача 1. ArrayAverage.java

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

        try {
            double average = calculateAverage(arr);
            System.out.println("Среднее арифметическое: " + average);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Ошибка: Выход за границы массива.");
        } catch (NumberFormatException e) {
            System.out.println("Оштбка: Неверные данные в массиве.");
        } catch (Exception e) {
            System.out.println("Ошибка: " + e.getMessage());
        }
    }

    public static double calculateAverage(int[] arr) {
        if (arr.length == 0) {
            throw new ArithmeticException("Массив пустой, невозможно вычислить
среднее.");
        }
```

```
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
        }
        return (double) sum / arr.length;
    }
}
```

Задача 2. FileCopy.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class FileCopy {
    public static void main(String[] args) {
        String sourceFile = "manul.txt";
        String destinationFile = "PalasCat.txt";

        try {
            copyFile(sourceFile, destinationFile);
            System.out.println("Копирование завершено успешно.");
        } catch (IOException e) {
            System.out.println("Ошибка при копировании файла: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Произошла ошибка: " + e.getMessage());
        }
    }

    public static void copyFile(String source, String destination) throws
IOException {
        try (BufferedReader reader = new BufferedReader(new FileReader(source));
        FileWriter writer = new FileWriter(destination)) {

            String line;

            while ((line = reader.readLine()) != null) {
                writer.write(line);
                writer.write(System.lineSeparator());
            }
        }

    }
}
```

Задача 3. CustomAgeException.java

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

ExceptionLogger.java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class ExceptionLogger {
    private static final String LOG_FILE = "C:\\Users\\manul\\Desktop\\ITiP\\Lab
4\\n" + //
                "3\\log\\exceptions.txt";

    public static void logException(Exception e) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(LOG_FILE,
true))) {
            String timestamp =
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
            writer.write(timestamp + " - " + e.getMessage());
            writer.newLine();
        } catch (IOException ioException) {
            System.out.println("Ошибка при записи в лог-файл: " +
ioException.getMessage());
        }
    }
}
```

App.py

```
import java.util.Scanner;

public class App {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Введите ваш возраст: ");

        try {
            int age = scanner.nextInt();
            validateAge(age);
            System.out.println("Ваш возраст: " + age);
        } catch (CustomAgeException e) {
```

```
            ExceptionLogger.logException(e);
            System.out.println("Ошибка: " + e.getMessage());
        } catch (Exception e) {
            ExceptionLogger.logException(e);
            System.out.println("Произошла ошибка: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    public static void validateAge(int age) throws CustomAgeException {
        if (age < 0 || age > 120) {
            throw new CustomAgeException("Недопустимый возраст: " + age);
        }
    }
}
```