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 №5

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

Москва, 2025

Цель работы:

Изучить основные принципы работы со строками в Java, освоить механизмы создания, обработки и сравнения строковых объектов. Получить практические навыки использования регулярных выражений для поиска, валидации и преобразования текстовых данных с помощью классов Pattern и Matcher.

Индивидуальное задание:

Задание 1. Поиск всех чисел в тексте

Написать программу, которая будет искать все числа в за- данном тексте и выводить их на экран. При этом программа должна использовать регулярные выражения для поиска чисел и обрабатывать возможные ошибки. Пример реализации такого функционала представлен на листинге 5.1.

Листинг 5.1. Класс NumberFinder

1. 2. 3. 4.

5. 6. 7. 8. 9.

10. 11. 12.

```
import java.util.regex.*;

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

String text = "The price of the product is $19.99"; Pattern pattern =
Pattern.compile("\\d+ \\.\\d+ "); Matcher matcher = pattern.matcher(text);

while (matcher.find()) { System.out.println(matcher.group());

} }
```

}

Задание 2. Проверка корректности ввода пароля

Написать программу, которая будет проверять корректность ввода пароля. Пароль должен состоять из латинских букв и цифр, быть длиной от 8 до 16 символов и содержать хотя бы одну заглавную букву и одну цифру. При этом программа долж- на использовать регулярные выражения для проверки пароля и обрабатывать возможные ошибки.

Задание 3. Поиск заглавной буквы после строчной

Написать программу, которая будет находить все случаи в тексте, когда сразу после строчной буквы идет заглавная без какого-либо символа между ними и выделять их знаками «!» с двух сторон.

Задание 4. Проверка корректности ввода IP-адреса

Написать программу, которая будет проверять корректность ввода IP-адреса. IP-адрес должен состоять из 4 чисел, разде- ленных точками, и каждое число должно быть в диапазоне от 0 до 255. При этом программа должна использовать регулярные выражения для проверки IP-адреса и обрабатывать возможные ошибки.

Задание 5. Поиск всех слов, начинающихся с задан- ной буквы

Написать программу, которая будет искать все слова в за- данном тексте, начинающиеся с заданной буквы, и выводить их на экран. При этом программа должна использовать регулярные выражения для поиска слов и обрабатывать возможные ошибки.

Выполнение:

```
package lab5;


import java.util.regex.*;
import java.util.Scanner;


/**
 * Задание 1. Поиск всех чисел в тексте
 * Программа ищет все числа в заданном тексте и выводит их на экран.
 */
public class Task1 {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.println("=== Задание 1: Поиск всех чисел в тексте ===\n");

            // Демонстрация с примером из задания
            String exampleText = "The price of the product is $19.99";
            System.out.println("Пример текста: " + exampleText);
            findNumbers(exampleText);

            System.out.println("\n--- Дополнительные примеры ---");
```

```
// Дополнительные примеры для демонстрации
String[] testTexts = {
    "В корзине 5 яблок, 3.5 кг апельсинов и 12 бананов",
    "Температура составляет -15.7 градусов, а влажность 80%",
    "Цены: 100, 250.50, 1000.99 рублей",
    "Нет чисел в этой строке!"
};

for (String text : testTexts) {
    System.out.println("\nТекст: " + text);
    findNumbers(text);
}

// Интерактивный режим
System.out.println("\n--- Интерактивный режим ---");
System.out.println("Введите ваш текст (или 'exit' для выхода):");

while (true) {
    System.out.print("> ");
```

```
            String userInput = scanner.nextLine();


            if (userInput.equalsIgnoreCase("exit")) {
                break;
            }


            findNumbers(userInput);
        }


    } catch (Exception e) {
        System.err.println("Ошибка: " + e.getMessage());
        e.printStackTrace();
    }
}


/**
 * Находит и выводит все числа в заданном тексте
 * @param text текст для поиска
 */
private static void findNumbers(String text) {
```

```
        try {
            // Регулярное выражение для поиска целых и дробных чисел (включая отрицательные)
            Pattern pattern = Pattern.compile("-?\\d+(\\.\\d+)?");
            Matcher matcher = pattern.matcher(text);

            boolean found = false;
            System.out.print("Найденные числа: ");

            while (matcher.find()) {
                System.out.print(matcher.group() + " ");
                found = true;
            }

            if (!found) {
                System.out.print("числа не найдены");
            }
            System.out.println();

        } catch (PatternSyntaxException e) {
```

```
            System.err.println("Ошибка в регулярном выражении: " +
e.getMessage());
        } catch (Exception e) {
            System.err.println("Ошибка при обработке текста: " +
e.getMessage());
        }
    }
}
package lab5;


import java.util.Scanner;
import java.util.regex.*;


/**
 * Задание 2. Проверка корректности ввода пароля
 * Пароль должен:
 * - Состоять из латинских букв и цифр
 * - Быть длиной от 8 до 16 символов
 * - Содержать хотя бы одну заглавную букву
 * - Содержать хотя бы одну цифру
 */
```

```
public class Task2 {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.println("=== Задание 2: Проверка корректности пароля ===\n");

            System.out.println("Требования к паролю:");
            System.out.println("- Длина от 8 до 16 символов");
            System.out.println("- Только латинские буквы и цифры");
            System.out.println("- Минимум одна заглавная буква");
            System.out.println("- Минимум одна цифра\n");

            // Демонстрация с примерами
            System.out.println("--- Примеры проверки паролей ---");
            String[] testPasswords = {
                "Password1",      // Корректный
                "MyPass123",      // Корректный
                "weakpass",       // Нет заглавной буквы и цифры
                "NODIGITS",       // Нет цифры
                "NoDigit",        // Нет цифры
                "short1A",        // Слишком короткий (7 символов)
```

```
            "TooLongPassword12345", // Слишком длинный (21 символ)
            "Pass123!",       // Содержит спецсимвол
            "Пароль123",      // Содержит кириллицу
            "ValidPass99"     // Корректный
        };

        for (String password : testPasswords) {
            validatePassword(password);
        }

        // Интерактивный режим
        System.out.println("\n--- Интерактивный режим ---");
        System.out.println("Введите пароль для проверки (или 'exit' для выхода):");

        while (true) {
            System.out.print("> ");
            String userInput = scanner.nextLine();

            if (userInput.equalsIgnoreCase("exit")) {
                break;
```

```
            }

            validatePassword(userInput);
        }

    } catch (Exception e) {
        System.err.println("Ошибка: " + e.getMessage());
        e.printStackTrace();
    }
}

/**
 * Проверяет корректность пароля согласно требованиям
 * @param password пароль для проверки
 */
private static void validatePassword(String password) {
    try {
        System.out.println("\nПроверка пароля: \"" + password + "\"");

        // Проверка длины
```

```
        if (password.length() < 8 || password.length() > 16) {
            System.out.println("❌НЕВЕРНО: длина должна быть от 8 до 16 символов (текущая: " + password.length() + ")");
            return;
        }

        // Проверка на только латинские буквы и цифры
        Pattern latinAndDigitsOnly = Pattern.compile("^[a-zA-Z0-9]+$");
        if (!latinAndDigitsOnly.matcher(password).matches()) {
            System.out.println("❌НЕВЕРНО: пароль должен содержать только латинские буквы и цифры");
            return;
        }

        // Проверка наличия хотя бы одной заглавной буквы
        Pattern hasUpperCase = Pattern.compile("[A-Z]");
        if (!hasUpperCase.matcher(password).find()) {
            System.out.println("❌НЕВЕРНО: пароль должен содержать хотя бы одну заглавную букву");
            return;
        }
```

```
            // Проверка наличия хотя бы одной цифры
            Pattern hasDigit = Pattern.compile("\\d");
            if (!hasDigit.matcher(password).find()) {
                System.out.println("❌НЕВЕРНО: пароль должен содержать хотя бы одну цифру");
                return;
            }

            // Все проверки пройдены
            System.out.println("✅ВЕРНО: пароль соответствует всем требованиям");

        } catch (PatternSyntaxException e) {
            System.err.println("Ошибка в регулярном выражении: " + e.getMessage());
        } catch (Exception e) {
            System.err.println("Ошибка при проверке пароля: " + e.getMessage());
        }
    }
```

```
    /**
     * Альтернативная версия с одним регулярным выражением
     * @param password пароль для проверки
     * @return true если пароль валиден
     */
    @SuppressWarnings("unused")
    private static boolean validatePasswordOneRegex(String password) {
        try {
            // Комплексное регулярное выражение для всех требований
            // (?=.*[A-Z]) - хотя бы одна заглавная буква
            // (?=.*\d) - хотя бы одна цифра
            // [a-zA-Z0-9]{8,16} - только латинские буквы и цифры, длина 8-16
            Pattern pattern = Pattern.compile("^(?=.*[A-Z])(?=.*\\d)[a-zA-Z0-
9]{8,16}$");
            return pattern.matcher(password).matches();
        } catch (Exception e) {
            System.err.println("Ошибка при проверке: " + e.getMessage());
            return false;
        }
    }
}
```

```
package lab5;


import java.util.Scanner;
import java.util.regex.*;


/**
 * Задание 3. Поиск заглавной буквы после строчной
 * Программа находит все случаи, когда сразу после строчной буквы идет заглавная
 * без какого-либо символа между ними и выделяет их знаками «!» с двух сторон.
 */
public class Task3 {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.println("=== Задание 3: Поиск заглавной буквы после строчной ===\n");


            System.out.println("Программа находит случаи, когда строчная буква сразу");
            System.out.println("переходит в заглавную и выделяет их знаками '!'.\n");
```

```
// Демонстрация с примерами
System.out.println("--- Примеры обработки текста ---");
String[] testTexts = {
    "helloWorld",
    "iPhone и iPad от Apple",
    "thisIsACamelCaseExample",
    "HTML и CSS это основа веб-разработки",
    "JavaScript",
    "Москва Moscow",
    "testStringWithManyCamelCaseWords",
    "нет переходов в этом тексте",
    "ABCDEF",
    "abcdef"
};

for (String text : testTexts) {
    highlightLowerToUpper(text);
}
```

```
        // Интерактивный режим
        System.out.println("\n--- Интерактивный режим ---");
        System.out.println("Введите текст для обработки (или 'exit' для выхода):");

        while (true) {
            System.out.print("> ");
            String userInput = scanner.nextLine();

            if (userInput.equalsIgnoreCase("exit")) {
                break;
            }

            highlightLowerToUpper(userInput);
        }

    } catch (Exception e) {
        System.err.println("Ошибка: " + e.getMessage());
        e.printStackTrace();
    }
}
```

```
    /**
     * Находит и выделяет все случаи перехода строчной буквы в заглавную
     * @param text исходный текст
     */
    private static void highlightLowerToUpper(String text) {
        try {
            System.out.println("\nИсходный текст: " + text);

            // Регулярное выражение для поиска строчной буквы, за которой следует заглавная
            // ([a-za-я]) - строчная буква (латиница или кириллица)
            // ([A-ZA-Я]) - заглавная буква (латиница или кириллица)
            Pattern pattern = Pattern.compile("([a-za-яё])([A-ZA-ЯЁ])");
            Matcher matcher = pattern.matcher(text);

            // Подсчет количества найденных случаев
            int count = 0;
            StringBuffer result = new StringBuffer();

            while (matcher.find()) {
```

```
            count++;
            // Заменяем найденную пару на пару, обрамленную восклицательными знаками
            String replacement = "!" + matcher.group(1) + matcher.group(2) + "!";
            matcher.appendReplacement(result, replacement);
        }
        matcher.appendTail(result);

        if (count > 0) {
            System.out.println("Результат:      " + result.toString());
            System.out.println("Найдено переходов: " + count);
        } else {
            System.out.println("Результат:      " + text);
            System.out.println("Переходы не найдены");
        }

    } catch (PatternSyntaxException e) {
        System.err.println("Ошибка в регулярном выражении: " + e.getMessage());
    } catch (Exception e) {
```

```
            System.err.println("Ошибка при обработке текста: " +
e.getMessage());
        }
    }

    /**
     * Альтернативная версия с детальным выводом найденных пар
     * @param text исходный текст
     */
    @SuppressWarnings("unused")
    private static void findAndDisplayPairs(String text) {
        try {
            System.out.println("\nИсходный текст: " + text);

            Pattern pattern = Pattern.compile("([a-za-яё])([A-ZA-ЯЁ])");
            Matcher matcher = pattern.matcher(text);

            boolean found = false;
            System.out.println("Найденные пары:");

            while (matcher.find()) {
```

```
                found = true;
                System.out.println("  Позиция " + matcher.start() + ": '" +
                                   matcher.group(1) + matcher.group(2) +
                                   "' (" + matcher.group(1) + " → " + matcher.group(2) + ")");
            }

            if (!found) {
                System.out.println("  Переходы не найдены");
            }

        } catch (Exception e) {
            System.err.println("Ошибка: " + e.getMessage());
        }
    }
}
```

```
package lab5;

import java.util.Scanner;
import java.util.regex.*;
```

```
/**
 * Задание 4. Проверка корректности ввода IP-адреса
 * IP-адрес должен:
 * - Состоять из 4 чисел, разделенных точками
 * - Каждое число должно быть в диапазоне от 0 до 255
 */
public class Task4 {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.println("=== Задание 4: Проверка корректности IP-адреса ===\n");

            System.out.println("Требования к IP-адресу:");
            System.out.println("- Формат: XXX.XXX.XXX.XXX");
            System.out.println("- Каждое число в диапазоне от 0 до 255\n");

            // Демонстрация с примерами
            System.out.println("--- Примеры проверки IP-адресов ---");
            String[] testIPs = {
                "192.168.1.1",       // Корректный
                "255.255.255.255",   // Корректный (максимальный)
```

```
    "0.0.0.0",           // Корректный (минимальный)
    "127.0.0.1",         // Корректный (localhost)
    "256.1.1.1",         // Неверно: число > 255
    "192.168.1.256",     // Неверно: число > 255
    "192.168.1",         // Неверно: только 3 октета
    "192.168.1.1.1",     // Неверно: 5 октетов
    "192.168.-1.1",      // Неверно: отрицательное число
    "192.168.1.abc",     // Неверно: не число
    "192.168.01.1",      // Неверно: ведущий ноль
    "192.168.1.1 ",      // Неверно: пробел в конце
    " 192.168.1.1",      // Неверно: пробел в начале
    "192 168 1 1",       // Неверно: пробелы вместо точек
    "8.8.8.8",           // Корректный (Google DNS)
    "10.0.0.1",          // Корректный
    "172.16.0.1",        // Корректный
    "300.300.300.300"    // Неверно: все числа > 255
};

for (String ip : testIPs) {
    validateIP(ip);
```

```
        }

        // Интерактивный режим
        System.out.println("\n--- Интерактивный режим ---");
        System.out.println("Введите IP-адрес для проверки (или 'exit' для выхода):");

        while (true) {
            System.out.print("> ");
            String userInput = scanner.nextLine();

            if (userInput.equalsIgnoreCase("exit")) {
                break;
            }

            validateIP(userInput);
        }

    } catch (Exception e) {
        System.err.println("Ошибка: " + e.getMessage());
        e.printStackTrace();
```

```
        }
    }

    /**
     * Проверяет корректность IP-адреса
     * @param ip IP-адрес для проверки
     */
    private static void validateIP(String ip) {
        try {
            System.out.println("\nПроверка IP: \"" + ip + "\"");

            // Базовая проверка формата (4 группы чисел, разделенных точками)
            // Регулярное выражение для IP-адреса:
            // ^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})$
            // ^ - начало строки
            // (\d{1,3}) - группа из 1-3 цифр
            // \. - точка (экранирована)
            // $ - конец строки
            Pattern basicPattern = Pattern.compile("^(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})$");
```

```
        Matcher matcher = basicPattern.matcher(ip);


        if (!matcher.matches()) {
            System.out.println("✗НЕВЕРНО: некорректный формат IP-адреса");
            System.out.println("   Ожидается формат: XXX.XXX.XXX.XXX (где X - цифра)");
            return;
        }


        // Проверка диапазона каждого октета (0-255)
        for (int i = 1; i <= 4; i++) {
            String octet = matcher.group(i);


            // Проверка на ведущие нули (кроме "0")
            if (octet.length() > 1 && octet.startsWith("0")) {
                System.out.println("✗НЕВЕРНО: октет \"" + octet + "\" содержит ведущий ноль");
                System.out.println("   IP-адреса не должны содержать ведущие нули");
                return;
            }
```

```
            int value = Integer.parseInt(octet);
            if (value < 0 || value > 255) {
                System.out.println("✗НЕВЕРНО: октет \"" + octet + "\" вне диапазона [0-255]");
                System.out.println("   Значение: " + value);
                return;
            }
        }

        // Все проверки пройдены
        System.out.println("✓ВЕРНО: IP-адрес корректен");
        System.out.println("   Октеты: " + matcher.group(1) + ", " +
                           matcher.group(2) + ", " +
                           matcher.group(3) + ", " +
                           matcher.group(4));

    } catch (PatternSyntaxException e) {
        System.err.println("Ошибка в регулярном выражении: " + e.getMessage());
    } catch (NumberFormatException e) {
```

```
            System.err.println("Ошибка преобразования числа: " +
e.getMessage());
        } catch (Exception e) {
            System.err.println("Ошибка при проверке IP-адреса: " +
e.getMessage());
        }
    }

    /**
     * Альтернативная версия с более сложным регулярным выражением
     * Проверяет IP-адрес одним регулярным выражением
     * @param ip IP-адрес для проверки
     * @return true если IP корректен
     */
    @SuppressWarnings("unused")
    private static boolean validateIPAdvanced(String ip) {
        try {
            // Сложное регулярное выражение, проверяющее диапазон 0-255
для каждого октета:
            // (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?) - один октет:
            //   25[0-5] - 250-255
```

```
            //   2[0-4][0-9] - 200-249
            //   [01]?[0-9][0-9]? - 0-199
            String ipPattern =
                "^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\." +
                "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\." +
                "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\." +
                "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$";

            Pattern pattern = Pattern.compile(ipPattern);
            return pattern.matcher(ip).matches();

        } catch (Exception e) {
            System.err.println("Ошибка при проверке: " + e.getMessage());
            return false;
        }
    }
}
package lab5;

import java.util.Scanner;
```

```
import java.util.regex.*;
import java.util.ArrayList;
import java.util.List;

/**
 * Задание 5. Поиск всех слов, начинающихся с заданной буквы
 * Программа ищет все слова в заданном тексте, начинающиеся с заданной буквы,
 * и выводит их на экран.
 */
public class Task5 {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.println("=== Задание 5: Поиск слов по начальной букве ===\n");

            // Демонстрация с примерами
            System.out.println("--- Примеры поиска слов ---");

            String sampleText = "Java is a popular programming language. " +
                    "JavaScript and JSON are also important. " +
```

```
        "Many Java developers love Java's features. " +
        "Программирование на Java требует знания основ.";

System.out.println("Пример текста:");
System.out.println(sampleText + "\n");

// Поиск слов на разные буквы
char[] testLetters = { 'J', 'j', 'p', 'П', 'a', 'М' };

for (char letter : testLetters) {
    findWordsByLetter(sampleText, letter);
}

// Дополнительные примеры
System.out.println("\n--- Дополнительные примеры ---");

String[] testCases = {
        "Apple, apricot and avocado are fruits starting with A",
        "The cat climbed the tree carefully",
        "Быстрая бурая лиса прыгает через ленивую собаку",
```

```
            "One two three four five six seven eight nine ten"
        };

        for (String text : testCases) {
            System.out.println("\nТекст: " + text);
            Scanner testScanner = new Scanner(System.in);
            System.out.print("Введите букву для поиска (или Enter для пропуска): ");
            // Для демонстрации будем искать первую букву каждого текста
            if (text.length() > 0) {
                char firstChar = text.charAt(0);
                findWordsByLetter(text, firstChar);
            }
        }

        // Интерактивный режим
        System.out.println("\n--- Интерактивный режим ---");
        System.out.println("Введите текст для анализа (или 'exit' для выхода):");

        while (true) {
```

```
System.out.print("Текст> ");
String userText = scanner.nextLine();

if (userText.equalsIgnoreCase("exit")) {
    break;
}

if (userText.trim().isEmpty()) {
    System.out.println("Текст не может быть пустым!");
    continue;
}

System.out.print("Введите букву для поиска> ");
String letterInput = scanner.nextLine();

if (letterInput.isEmpty()) {
    System.out.println("Буква не может быть пустой!");
    continue;
}
```

```
            char letter = letterInput.charAt(0);
            findWordsByLetter(userText, letter);
        }

    } catch (Exception e) {
        System.err.println("Ошибка: " + e.getMessage());
        e.printStackTrace();
    }
}

/**
 * Находит и выводит все слова, начинающиеся с заданной буквы
 *
 * @param text   текст для поиска
 * @param letter начальная буква
 */
private static void findWordsByLetter(String text, char letter) {
    try {
        System.out.println("\nПоиск слов, начинающихся с буквы '" + letter +
"':");
```

```
        // Создаем регулярное выражение для поиска слов
        // \\b - граница слова
        // [letter] - указанная буква (с учетом регистра)
        // \\w* - остальная часть слова (буквы, цифры, подчеркивание)

        // Экранируем специальные символы регулярных выражений
        String escapedLetter = Pattern.quote(String.valueOf(letter));

        // Поиск с учетом регистра
        Pattern patternCaseSensitive = Pattern.compile(
                "\\b" + escapedLetter + "\\w*",
                Pattern.UNICODE_CHARACTER_CLASS);

        // Поиск без учета регистра
        Pattern patternCaseInsensitive = Pattern.compile(
                "\\b" + escapedLetter + "\\w*",
                Pattern.CASE_INSENSITIVE |
Pattern.UNICODE_CHARACTER_CLASS);

        // Используем поиск без учета регистра
        Matcher matcher = patternCaseInsensitive.matcher(text);
```

```
    List<String> foundWords = new ArrayList<>();
    while (matcher.find()) {
        String word = matcher.group();
        if (!foundWords.contains(word)) {
            foundWords.add(word);
        }
    }

    if (foundWords.isEmpty()) {
        System.out.println("  Слова не найдены");
    } else {
        System.out.println("  Найдено слов: " + foundWords.size());
        System.out.println("  Список слов:");
        for (int i = 0; i < foundWords.size(); i++) {
            System.out.println("    " + (i + 1) + ". " + foundWords.get(i));
        }
    }

} catch (PatternSyntaxException e) {
```

```
            System.err.println("Ошибка в регулярном выражении: " +
e.getMessage());
        } catch (Exception e) {
            System.err.println("Ошибка при поиске слов: " + e.getMessage());
        }
    }

    /**
     * Альтернативная версия с подсчетом вхождений каждого слова
     *
     * @param text   текст для поиска
     * @param letter начальная буква
     */
    @SuppressWarnings("unused")
    private static void findWordsWithCount(String text, char letter) {
        try {
            System.out.println("\nПоиск слов, начинающихся с буквы '" + letter +
"' (с подсчетом):");

            String escapedLetter = Pattern.quote(String.valueOf(letter));
            Pattern pattern = Pattern.compile(
```

```
            "\\b" + escapedLetter + "\\w*",
            Pattern.CASE_INSENSITIVE |
Pattern.UNICODE_CHARACTER_CLASS);

        Matcher matcher = pattern.matcher(text);

        List<String> allMatches = new ArrayList<>();
        while (matcher.find()) {
            allMatches.add(matcher.group());
        }

        if (allMatches.isEmpty()) {
            System.out.println("  Слова не найдены");
            return;
        }

        System.out.println("  Всего найдено: " + allMatches.size() + "
вхождений");
        System.out.println("  Слова с количеством повторений:");

        // Подсчет уникальных слов
```

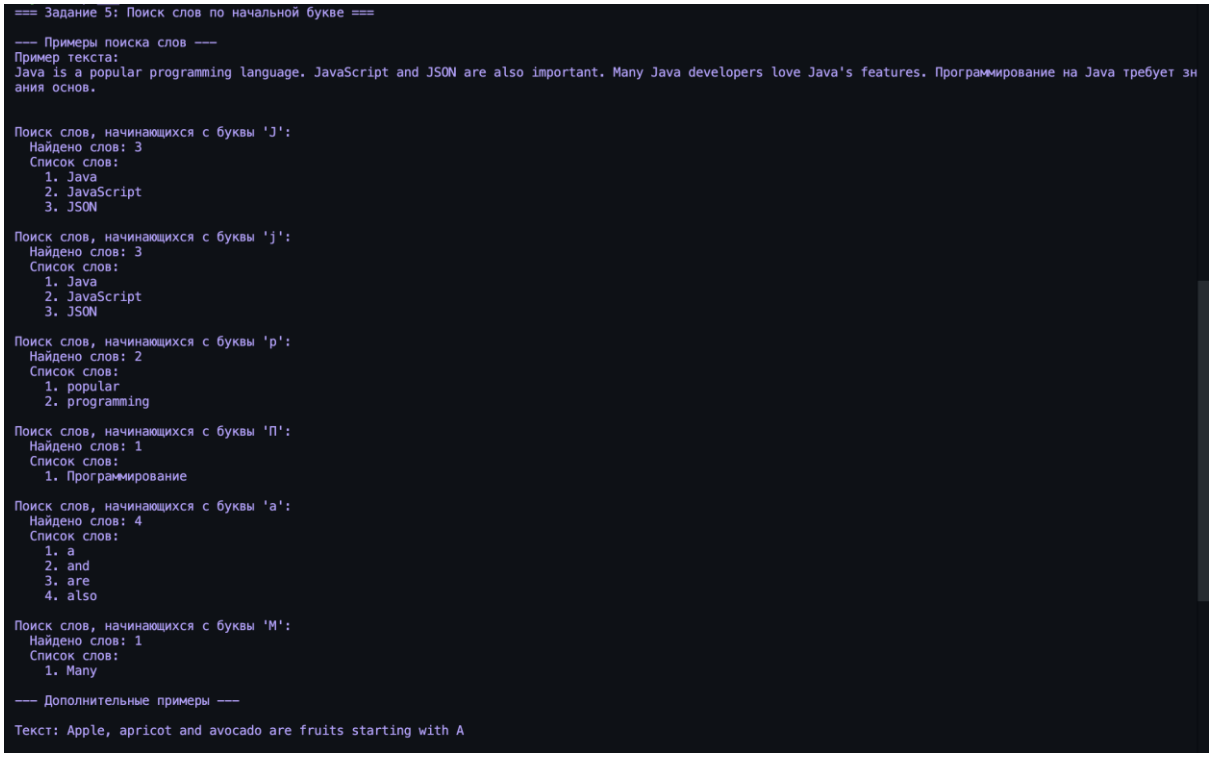
```
            java.util.Map<String, Integer> wordCount = new
java.util.HashMap<>();
            for (String word : allMatches) {
                wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
            }

            for (java.util.Map.Entry<String, Integer> entry : wordCount.entrySet())
{
                System.out.println("    " + entry.getKey() + " - " + entry.getValue() +
" раз(а)");
            }

        } catch (Exception e) {
            System.err.println("Ошибка: " + e.getMessage());
        }
    }
}
```

```
=== Задание 1: Поиск всех чисел в тексте ===

Пример текста: The price of the product is $19.99
Найденные числа: 19.99

--- Дополнительные примеры ---

Текст: В корзине 5 яблок, 3.5 кг апельсинов и 12 бананов
Найденные числа: 5 3.5 12

Текст: Температура составляет -15.7 градусов, а влажность 80%
Найденные числа: -15.7 80

Текст: Цены: 100, 250.50, 1000.99 рублей
Найденные числа: 100 250.50 1000.99

Текст: Нет чисел в этой строке!
Найденные числа: числа не найдены
```

```
=== Задание 2: Проверка корректности пароля ===

Требования к паролю:
- Длина от 8 до 16 символов
- Только латинские буквы и цифры
- Минимум одна заглавная буква
- Минимум одна цифра

--- Примеры проверки паролей ---

Проверка пароля: "Password1"
✅ ВЕРНО: пароль соответствует всем требованиям

Проверка пароля: "MyPass123"
✅ ВЕРНО: пароль соответствует всем требованиям

Проверка пароля: "weakpass"
❌ НЕВЕРНО: пароль должен содержать хотя бы одну заглавную букву

Проверка пароля: "NODIGITS"
❌ НЕВЕРНО: пароль должен содержать хотя бы одну цифру

Проверка пароля: "NoDigit"
❌ НЕВЕРНО: длина должна быть от 8 до 16 символов (текущая: 7)

Проверка пароля: "short1A"
❌ НЕВЕРНО: длина должна быть от 8 до 16 символов (текущая: 7)

Проверка пароля: "TooLongPassword12345"
❌ НЕВЕРНО: длина должна быть от 8 до 16 символов (текущая: 20)

Проверка пароля: "Pass123!"
❌ НЕВЕРНО: пароль должен содержать только латинские буквы и цифры

Проверка пароля: "Пароль123"
❌ НЕВЕРНО: пароль должен содержать только латинские буквы и цифры

Проверка пароля: "ValidPass99"
✅ ВЕРНО: пароль соответствует всем требованиям
```

```
=== Задание 3: Поиск заглавной буквы после строчной ===

Программа находит случаи, когда строчная буква сразу
переходит в заглавную и выделяет их знаками '!'.

--- Примеры обработки текста ---

Исходный текст: helloWorld
Результат:      hell!oW!orld
Найдено переходов: 1

Исходный текст: iPhone и iPad от Apple
Результат:      !iP!hone и !iP!ad от Apple
Найдено переходов: 2

Исходный текст: thisIsACamelCaseExample
Результат:      thi!sI!!sA!Came!lC!as!eE!xample
Найдено переходов: 4

Исходный текст: HTML и CSS это основа веб-разработки
Результат:      HTML и CSS это основа веб-разработки
Переходы не найдены

Исходный текст: JavaScript
Результат:      Jav!aS!cript
Найдено переходов: 1

Исходный текст: Москва Moscow
Результат:      Москва Moscow
Переходы не найдены

Исходный текст: testStringWithManyCamelCaseWords
Результат:      tes!tS!trin!gW!it!hM!an!yC!ame!lC!as!eW!ords
Найдено переходов: 6

Исходный текст: нет переходов в этом тексте
Результат:      нет переходов в этом тексте
Переходы не найдены

Исходный текст: ABCDEF
Результат:      ABCDEF
Переходы не найдены

Исходный текст: abcdef
Результат:      abcdef
Переходы не найдены
```

```
=== Задание 4: Проверка корректности IP-адреса ===

Требования к IP-адресу:
- Формат: XXX.XXX.XXX.XXX
- Каждое число в диапазоне от 0 до 255

--- Примеры проверки IP-адресов ---

Проверка IP: "192.168.1.1"
✅ ВЕРНО: IP-адрес корректен
   Октеты: 192, 168, 1, 1

Проверка IP: "255.255.255.255"
✅ ВЕРНО: IP-адрес корректен
   Октеты: 255, 255, 255, 255

Проверка IP: "0.0.0.0"
✅ ВЕРНО: IP-адрес корректен
   Октеты: 0, 0, 0, 0

Проверка IP: "127.0.0.1"
✅ ВЕРНО: IP-адрес корректен
   Октеты: 127, 0, 0, 1

Проверка IP: "256.1.1.1"
❌ НЕВЕРНО: октет "256" вне диапазона [0-255]
   Значение: 256

Проверка IP: "192.168.1.256"
❌ НЕВЕРНО: октет "256" вне диапазона [0-255]
   Значение: 256

Проверка IP: "192.168.1"
❌ НЕВЕРНО: некорректный формат IP-адреса
   Ожидается формат: XXX.XXX.XXX.XXX (где X - цифра)

Проверка IP: "192.168.1.1.1"
❌ НЕВЕРНО: некорректный формат IP-адреса
   Ожидается формат: XXX.XXX.XXX.XXX (где X - цифра)

Проверка IP: "192.168.-1.1"
❌ НЕВЕРНО: некорректный формат IP-адреса
   Ожидается формат: XXX.XXX.XXX.XXX (где X - цифра)

Проверка IP: "192.168.1.abc"
❌ НЕВЕРНО: некорректный формат IP-адреса
   Ожидается формат: XXX.XXX.XXX.XXX (где X - цифра)

Проверка IP: "192.168.01.1"
❌ НЕВЕРНО: октет "01" содержит ведущий ноль
   IP-адреса не должны содержать ведущие нули
```



Контрольные вопросы:

1. Что такое класс String?

String — это класс в Java, представляющий неизменяемую последовательность символов. Он находится в пакете java.lang и автоматически импортируется в каждую программу. Объекты String используются для хранения и манипулирования текстовыми данными.

2. Почему объект класса String является иммутабельным?

Иммутабельность String обеспечивается несколькими механизмами: класс объявлен как final (нельзя наследовать), внутренний массив символов также final и private, а все методы, которые "изменяют" строку, на самом

деле возвращают новый объект. Это сделано для безопасности (строки можно безопасно использовать как ключи в HashMap), потокобезопасности и оптимизации через пул строк.

3. Что такое интернирование строк?

Интернирование — это процесс сохранения строковых литералов в специальной области памяти (пуле строк). Когда создаётся строковый литерал, JVM проверяет, существует ли уже такая строка в пуле. Если да, возвращается ссылка на существующий объект. Метод intern() позволяет явно добавить строку в пул.

4. В чем разница между String, StringBuilder и StringBuffer?

String — неизменяемый класс, каждая модификация создаёт новый объект. StringBuilder — изменяемый класс для построения строк, не потокобезопасный, но быстрый. StringBuffer — аналогичен StringBuilder, но потокобезопасный (методы синхронизированы), поэтому работает медленнее. Для частых модификаций строк в однопоточной среде предпочтителен StringBuilder.

5. Как сравнить две строки? В чем разница между ==, equals() и equalsIgnoreCase()?

Оператор == сравнивает ссылки на объекты (идентичность в памяти). Метод equals() сравнивает содержимое строк посимвольно с учётом регистра. Метод equalsIgnoreCase() сравнивает содержимое без учёта регистра. Для сравнения текстового содержания всегда следует использовать equals() или equalsIgnoreCase().

6. Как хранятся строки в памяти? Что такое пул строк?

Строки хранятся в куче (heap). Пул строк (String Pool) — это специальная область в heap, где JVM хранит уникальные строковые литералы. Строки, созданные через литералы, попадают в пул автоматически. Строки, созданные через new String(), размещаются вне пула, но могут быть добавлены туда методом intern().

7. Что такое code unit и code point?

Code point — это числовое значение символа в стандарте Unicode (например, U+0041 для буквы 'A'). Code unit — это единица кодирования, в Java это 16-битное значение (тип char). Для символов из Basic Multilingual Plane один code point равен одному code unit, но для символов за пределами BMP требуется два code unit (суррогатная пара).

8. Что необходимо для использования регулярных выражений в Java?

Необходимо импортировать классы из пакета java.util.regex. Основные классы: Pattern — скомпилированное представление регулярного выражения, Matcher — механизм поиска соответствий в тексте. Также можно использовать методы класса String: matches(), split(), replaceAll(), replaceFirst().

9. Какие есть режимы работы квантификатора?

Жадный (greedy) — захватывает максимально возможное количество символов, обозначается как *, +, ?, {n,m}. Ленивый (reluctant) — захватывает минимально возможное количество, добавляется ? после квантификатора (*?, +?, ??). Сверхжадный (possessive) — как жадный, но без возврата, добавляется + (*+, ++, ?+).

10. Как проверить, соответствует ли строка регулярному выражению?

Можно использовать метод matches() класса String: string.matches("regex"). Или через Pattern и Matcher: Pattern.matches("regex", string). Также можно создать Pattern и использовать matcher: pattern.matcher(string).matches(). Метод matches() проверяет соответствие всей строки шаблону.

11. Как найти все вхождения подстроки по шаблону?

Нужно создать объект Matcher и использовать цикл с методом find():

java

```
Pattern p = Pattern.compile("шаблон");

Matcher m = p.matcher(text);

while (m.find()) {

    System.out.println(m.group());

}
```

Метод group() возвращает найденное соответствие, start() и end() — позиции.

12. Как разбить строку по регулярному выражению?

Используется метод split() класса String или Pattern. Например: string.split("\\s+") разобьёт строку по пробелам. Можно указать лимит: split("regex", limit). Через Pattern: pattern.split(text). Результатом является массив строк.

13. Как заменить подстроку по шаблону?

Методы класса String: replaceAll("regex", "replacement") заменяет все вхождения, replaceFirst("regex", "replacement") — только первое. Через Matcher: matcher.replaceAll("replacement") или

matcher.replaceFirst("replacement"). В строке замены можно использовать $1, $2 для групп захвата.

14. Как экранировать спецсимволы в регулярных выражениях?

Спецсимволы экранируются обратным слэшем: \. для точки, \\ для слэша. В Java строках нужно двойное экранирование: "\\." для точки в regex. Метод Pattern.quote(string) экранирует всю строку целиком, делая её литеральной. Метод Matcher.quoteReplacement() экранирует строку замены.

Заключение

В ходе лабораторной работы были изучены основные аспекты работы со строками в Java: иммутабельность класса String, механизм интернирования и пул строк, различия между String, StringBuilder и StringBuffer. Освоены способы корректного сравнения строк и понимание их внутреннего представления в памяти.

Практическая часть работы позволила получить навыки использования регулярных выражений для решения типичных задач: поиска числовых значений в тексте, валидации пользовательского ввода (пароли, IP-адреса), обнаружения текстовых паттернов и фильтрации данных. Изучены классы Pattern и Matcher, различные режимы квантификаторов и методы замены текста по шаблонам.

Полученные знания являются фундаментальными для разработки приложений, работающих с текстовыми данными, валидацией форм, парсингом файлов и обработкой пользовательского ввода.

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