МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Лабораторная работа № 5

«Регулярные выражения»

Выполнила: Студентка группы БВТ2306

Максимова Мария

Москва 2024

GitHub: https://github.com/Mashaaaaa7/Lab5

Задание 1: Поиск всех чисел в тексте

Необходимо написать программу, которая будет искать все числа в заданном тексте и выводить их на экран. При этом программа должна использовать регулярные выражения для поиска чисел и обрабатывать возможные ошибки.

Задание 2: Проверка корректности ввода пароля

Необходимо 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 должна использовать регулярные выражения для проверки пароля и обрабатывать возможные ошибки.

Задание 3: Поиск заглавной буквы после строчной

Необходимо 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

Необходимо написать программу, которая будет проверять корректность ввода IP-адреса. IP-адрес должен состоять из 4 чисел, разделенных 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 заданной буквы

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

Задание 1

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class NumberFinder {

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

```
        String text = "В этом тексте есть числа: 123, 456 и 789, а также
1.234, 56.789 и 0.987 - дробные." +
                "Есть еще отрицательные числа: -10, -2.5 и -345.678." +
                "И даже большие числа, например 1000000 и 1234567890.";

        // Регулярное выражение для поиска чисел, включающее целые и дробные
        String regex = "[-+]?\\d+(\\.\\d+)?";

        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(text);

        System.out.println("Найденные числа:");
        while (matcher.find()) {
            System.out.println("Match found!"+matcher.group()); // Выводим
найденное число
        }
    }
}
```

Этот код ищет все числа в заданном тексте и выводит их на консоль.

1. **Импортируем необходимые классы:**
    - import java.util.regex.Pattern;: Импортируем класс Pattern, который используется для создания шаблонов регулярных выражений.
    - import java.util.regex.Matcher;: Импортируем класс Matcher, который используется для поиска соответствий в тексте с помощью регулярного выражения.
2. **Создаем главный класс:**
    - public class NumberFinder { ... }: Создаем класс с именем NumberFinder, в котором будет находиться основной код.
3. **Создаем метод main:**
    - public static void main(String[] args) { ... }: Создаем метод main, который является точкой входа для программы.
    - String text = "...";: Создаем строку text, в которой содержится текст для поиска чисел.
4. **Создаем регулярное выражение:**
    - String regex = "[-+]?\\d+(\\.\\d+)?";: Создаем строку regex, в которой хранится регулярное выражение для поиска чисел.
    - [-+]?: Соответствует необязательному знаку минус или плюс.
    - \\d+: Соответствует одной или более цифрам.
    - (\.\\d+)?: Соответствует необязательному десятичному разделителю (точка) с одной или более цифрами после запятой.

5. **Компилируем регулярное выражение:**
    - Pattern pattern = Pattern.compile(regex);: Создаем объект Pattern с помощью метода compile, который компилирует регулярное выражение в шаблон для быстрого поиска.
6. **Создаем объект поиска:**
    - Matcher matcher = pattern.matcher(text);: Создаем объект Matcher, который будет искать соответствия в тексте text с помощью шаблона pattern.
7. **Ищем соответствия:**
    - System.out.println("Найденные числа:");: Выводим сообщение на консоль.
    - while (matcher.find()) { ... }: Используем цикл while, чтобы найти все соответствия в тексте.
    - System.out.println("Match found!"+matcher.group());: Если найдено соответствие, выводим на консоль сообщение “Match found!” и найденное число с помощью метода matcher.group().

Что делает код:

- Код использует регулярное выражение для поиска чисел в тексте.
- Регулярное выражение позволяет находить как целые, так и дробные числа, а также числа с знаком минус или плюс.
- Код выводит на консоль все найденные числа.

Задание 2

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class checkWhether {

    public static void main(String[] args) {

        String password = "Aa13245768910";
        // Регулярное выражение, проверяющее, что пароль содержит
        // хотя бы одну заглавную букву, одну цифру и не менее 8 символов
        Pattern pattern = Pattern.compile("^(?=.*[A-Z])(?=.*\\d).{8,}$");
        Matcher matcher = pattern.matcher(password);
        if (matcher.matches()) {
            System.out.println("Valid password!");
        } else {
            System.out.println("Invalid password. Password must contain at
least one uppercase letter, one digit and be at least 8 characters long.");

        }
```

```
    }
}
```

Этот код :

1. **Содержит хотя бы одну заглавную букву:** (?=.*[A-Z])
2. **Содержит хотя бы одну цифру:** (?=.*\\d)
3. **Не менее 8 символов:** . {8,}

- **^(?=.*[A-Z])(?=.*\\d).{8,}$**: Это регулярное выражение с помощью **положительных просмотров вперед ((?=...))** проверяет наличие заглавных букв и цифр, а затем с помощью **квантификатора .{8,}** проверяет минимальную длину пароля.

Пример:

- **Валидный пароль:** “Aa13245768910”
- **Невалидный пароль:** “Aa123” (недостаточно символов)
- **Невалидный пароль:** “aaa12345” (отсутствует заглавная буква)
- **Невалидный пароль:** “12345678” (отсутствует заглавная буква)

Задание 3

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class FindCapitalAfterLowercase{

    public static void main(String[] args) {

        String text = "CaPS letTerS";
        Pattern pattern = Pattern.compile("[a-z][A-Z]");
        Matcher matcher = pattern.matcher(text);

        System.out.println("Найденные совпадения:");
        while (matcher.find()) {
            System.out.println("Совпадение найдено: " + matcher.group());
        }
    }

}
```

Находит случаи, когда в строке встречается строчная буква, за которой следует заглавная.

Пояснение кода:

1. **import java.util.regex.Pattern; и import java.util.regex.Matcher;**: Импортируют необходимые классы для работы с регулярными выражениями.
2. **String text = "CaPS letTerS";**: Задается текст, в котором нужно искать совпадения.
3. **Pattern pattern = Pattern.compile("[a-z][A-Z]");**: Создается шаблон регулярного выражения [a-z][A-Z].
    - **[a-z]**: Соответствует любой строчной букве.
    - **[A-Z]**: Соответствует любой заглавной букве.
    - **[a-z][A-Z]**: Означает, что нужно найти последовательность, состоящую из строчной буквы, за которой следует заглавная.
4. **Matcher matcher = pattern.matcher(text);**: Создается объект Matcher, который связывает регулярное выражение с текстом для поиска совпадений.
5. **System.out.println("Найденные совпадения:");**: Выводится заголовок.
6. **while (matcher.find()) { ... }**: Цикл, который ищет все совпадения в тексте.
7. **System.out.println("Совпадение найдено: " + matcher.group());**: Если совпадение найдено, выводится информация о найденном совпадении.

Задание 4

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

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

        String text = "192.168.1.1 300.168.1.1 255.255.255.0";

        // Разделяем текст на отдельные IP-адреса по пробелам
        String[] ipAddresses = text.split("\\s+");

        String regex = "^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(25[0-
5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$";

        Pattern pattern = Pattern.compile(regex);

        System.out.println("Найденные IP-адреса:");
        for (String ipAddress : ipAddresses) { // Проверяем каждый IP-адрес
            Matcher matcher = pattern.matcher(ipAddress);
            if (matcher.matches()) {
                System.out.println("Найденные IP-адреса: " + ipAddress);
            }
        }
    }
}
```

Этот код нужен для проверки валидности IP-адресов.

1. Импорт необходимых классов:

```
import java.util.regex.Pattern;

import java.util.regex.Matcher;
```

Эти строки импортируют классы Pattern и Matcher, которые используются для работы с регулярными выражениями в Java.

2. Задание текста с IP-адресами:

```
String text = "192.168.1.1 300.168.1.1 255.255.255.0";
```

Здесь задаём строку, содержащую несколько IP-адресов, разделенных пробелами.

3. Разделение текста на отдельные IP-адреса:

```
String[] ipAddresses = text.split("\\s+");
```

Метод split("\\s+") разбивает строку text на массив строк ipAddresses, используя пробелы (\\s+) в качестве разделителя. Теперь есть отдельные IP-адреса в массиве.

4. Создание регулярного выражения для проверки IP-адреса:

```
String regex = "^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$";
```

Регулярное выражение regex проверяет, соответствует ли строка формату IP-адреса.

- ^: Начало строки.
- ((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.): Проверяет, что первые три октета IP-адреса соответствуют правильному формату:
  - 25[0-5]: От 250 до 255.
  - 2[0-4][0-9]: От 200 до 249.
  - [01]?[0-9][0-9]?: От 0 до 199.
  - \\. : Точка (разделитель октетов).
- {3}: Повторяет предыдущую группу три раза.
- (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?): Проверяет последний октет IP-адреса на соответствие тому же формату.
- $: Конец строки.

5. Компиляция регулярного выражения:

Pattern pattern = Pattern.compile(regex);

Создается объект Pattern с помощью компиляции регулярного выражения regex.

6. Проверка каждого IP-адреса:

System.out.println("Найденные IP-адреса:");

for (String ipAddress : ipAddresses) { // Проверяем каждый IP-адрес

  Matcher matcher = pattern.matcher(ipAddress);

  if (matcher.matches()) {

    System.out.println("Найденные IP-адреса: " + ipAddress);

  }

}

- В цикле for проверяется каждый IP-адрес из массива ipAddresses.
- Для каждого IP-адреса создается объект Matcher, связывающий его с шаблоном pattern.
- Метод matcher.matches() проверяет, соответствует ли IP-адрес заданному регулярному выражению.
- Если IP-адрес валидный, он выводится на экран.

Задание 5

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class WordFinder {

    public static void main(String[] args) {

        String text = "The world is full of beauty, waiting to be
discovered.";
        String letter = "w"; // Буква для поиска

        String[] words = text.split("\\s+"); // Разбиваем текст на слова

        System.out.println("Words starting with '" + letter + "':");

        for (String word : words) {
            String regex = "\\b" + letter + "\\w+\\b"; // Регулярное
выражение
```

```
            Pattern pattern = Pattern.compile(regex);
            Matcher matcher = pattern.matcher(word);

            if (matcher.find()) {
                System.out.println(word);
            }
        }
    }
}
```

Ваш код выглядит отлично и работает как надо! Он правильно находит все слова в тексте, которые начинаются с заданной буквы. Давайте разберем его по шагам:

1. Импорт необходимых классов:

`import java.util.regex.Pattern;`

`import java.util.regex.Matcher;`

Эти строки импортируют классы Pattern и Matcher, которые используются для работы с регулярными выражениями в Java.

2. Задание текста и буквы для поиска:

`String text = "The world is full of beauty, waiting to be discovered.";`

`String letter = "w"; // Буква для поиска`

Здесь вы задаете текст, в котором нужно искать слова, и букву, с которой они должны начинаться.

3. Разбиение текста на слова:

`String[] words = text.split("\\s+"); // Разбиваем текст на слова`

Метод split("\\s+") разделяет строку text на массив строк words, используя пробелы (\\s+) в качестве разделителя. Теперь у вас есть отдельные слова в массиве.

4. Создание регулярного выражения для поиска слов:

`String regex = "\\b" + letter + "\\w+\\b"; // Регулярное выражение`

- **\\b**: Символ границы слова. Это означает, что регулярное выражение будет искать слова, начинающиеся с буквы letter и ограниченные пробелами или началом/концом текста.

- **letter**: Переменная, в которую записана буква для поиска. В этом случае это “w”.
- **w+**: Любые буквенно-цифровые символы (a-z, A-Z, 0-9, _) один или более раз.

5. Компиляция регулярного выражения:

```
Pattern pattern = Pattern.compile(regex);
```

Создается объект Pattern с помощью компиляции регулярного выражения regex.

6. Проверка каждого слова:

```
System.out.println("Words starting with '" + letter + "':");

for (String word : words) {

   Matcher matcher = pattern.matcher(word);



   if (matcher.find()) {

       System.out.println(word);

   }

}
```

- В цикле for проверяется каждое слово из массива words.
- Для каждого слова создается объект Matcher, связывающий его с шаблоном pattern.
- Метод matcher.find() проверяет, соответствует ли слово заданному регулярному выражению.
- Если слово соответствует, то выводится на экран.