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 № 5

Регулярные выражения

Выполнил: Студент группы

БВТ2303

Ситникова Дарья

Москва

2024

Цель работы: Изучить и освоить основы работы с регулярными выражениями в языке программирования Java, используя класс java.util.regex.Pattern для создания, компиляции и применения регулярных выражений к текстовым данным.

Задачи:

1. Изучение синтаксиса регулярных выражений;
2. Написать программы с использованием регулярных выражений.

Краткая теория:

Регулярные выражения (или regex) — это мощный инструмент для поиска и обработки текстовой информации. Они представляют собой последовательности символов, которые формируют шаблоны для поиска строк, соответствующих определённым критериям. Регулярные выражения широко используются в программировании, в том числе в языке Java, для валидации данных, поиска и замены текста, а также для разбора строк.

Основные элементы синтаксиса регулярных выражений:

1. **Специальные символы:**
    - . — соответствует любому одиночному символу.
    - * — соответствует нулю или более повторениям предыдущего символа или группы.
    - + — соответствует одному или более повторениям предыдущего символа или группы.
    - ? — соответствует нулю или одному повторению предыдущего символа или группы.
    - ^ — указывает на начало строки.
    - $ — указывает на конец строки.

- [] — определяет набор символов, из которых может быть выбран один (например, [abc] соответствует a, b или c).
- [^] — определяет набор символов, которые не могут быть использованы (например, [^abc] соответствует любому символу, кроме a, b или c).

2. **Квантификаторы:**
   - {n} — соответствует точно n повторениям (например, \d{3} соответствует трем цифрам подряд).
   - {n,} — соответствует n или более повторениям.
   - {n,m} — соответствует от n до m повторениям.
3. **Группировка и альтернативы:**
   - () — используется для группировки символов и определения порядка выполнения операций.
   - | — обозначает логическое "ИЛИ" (например, (abc|def) соответствует либо abc, либо def).
4. **Специальные символы для классов символов:**
   - \d — соответствует любой цифре (0-9).
   - \D — соответствует любому нецифровому символу.
   - \w — соответствует любой букве или цифре (a-z, A-Z, 0-9, _).
   - \W — соответствует любому неалфавитно-цифровому символу.
   - \s — соответствует любому пробельному символу (пробел, табуляция, новая строка).
   - \S — соответствует любому непробельному символу.

Ход работы:

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.

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

Импортируются классы Pattern и Matcher для работы с регулярными выражениями.

```
public static void main(String[] args) {
        String text = "The price of the product is $19.99";

        try {
            findNumbers(text);
        } catch (Exception e) {
            System.out.println("Произошла ошибка: " + e.getMessage());
        }
    }
```

Блок try-catch для обработки возможных исключений. Вызывается функция, findNumbers которая проверяет предложение на наличие чисел и выводит их.

Метод findNumbers:

```
    public static void findNumbers(String text) {
        Pattern pattern = Pattern.compile("\\d+\\.\\d+");
        Matcher matcher = pattern.matcher(text);
        boolean found = false;
        while (matcher.find()) {
            System.out.println(matcher.group());
            found = true;
        }

         if (!found) {
            System.out.println("Чисел нет");
        }
    }
}
```

Pattern pattern = Pattern.compile("\\d+\\.\\d+");

Создается объект Pattern, который содержит регулярное выражение для поиска чисел с плавающей запятой. Регулярное выражение выглядит следующим образом:

\\d+\\.\\d+

1. **\\d+** — соответствует одной или более цифрам (0-9). Это соответствует целой части числа.
2. **\\.** — соответствует символу точки.
3. **\\d+** — снова соответствует одной или более цифрам (0-9). Это соответствует дробной части числа.

Таким образом, данное регулярное выражение будет находить числа, которые имеют формат, например, 123.456 или 0.789.

Метод find() ищет совпадения в строке. Если найдено хотя бы одно совпадение, оно выводится на экран, и переменная found устанавливается в true.

Если переменная found осталась false, это означает, что ни одно совпадение не было найдено, и выводится сообщение о том, что чисел не найдено.

Вывод результата в терминале:

```
PS C:\Users\manul\Desktop\ITiP\Lab 5> java NumberFinder.java
19.99
```

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.

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

```
import java.util.regex.Pattern;
```
Импортируются классы Scanner для считывыания пользовательского ввода и классы Pattern и Matcher для работы с регулярными выражениями.

```
try {
	String password = scanner.nextLine();
	passwordVer(password);
} catch (Exception e) {
	System.out.println("Произошла ошибка: " + e.getMessage());
} finally {
	scanner.close();
}
```
Блок try-catch для обработки возможных исключений при вводе данных. После ввода пароля вызывается функция passwordVer, которая проверяет его корректность.

Функция проверки пароля:

```
public static void passwordVer(String password) {
	Pattern pattern = Pattern.compile("^(?=.*[A-Z])(?=.*\\d)[A-Za-
z\\d]{8,16}");
	Matcher matcher = pattern.matcher(password);
	boolean found = false;
	while (matcher.find()) {
		System.out.println("Корректный пароль");
		found = true;
	}

	if (!found) {
		System.out.println("Некорректный пароль");
	}
}
```
В этой функции используется регулярное выражение для проверки пароля.

```
"^(?=.*[A-Z])(?=.*\\d)[A-Za-z\\d]{8,16}"
```
1. ^ — указывает на начало строки. Это означает, что проверка будет начинаться с первого символа введенного пароля.

2. **(?=.*[A-Z])** — это позитивный просмотр вперед (positive lookahead). Он проверяет, что в строке есть хотя бы одна заглавная буква (от A до Z). Этот фрагмент не захватывает символы, а просто проверяет их наличие.

3. **(?=.*\\d)** — еще один позитивный просмотр вперед, который проверяет, что в строке есть хотя бы одна цифра (от 0 до 9). Символ \\d соответствует любой цифре.

4. **[A-Za-z\\d]{8,16}** — это основная часть выражения, которая определяет, что пароль должен состоять только из букв (как заглавных, так и строчных) и цифр. При этом длина пароля должна быть от 8 до 16 символов.

    - [A-Za-z\\d] — класс символов, который включает все буквы латинского алфавита (как заглавные, так и строчные) и цифры.
    - {8,16} — квантификатор, который указывает, что данный класс символов должен повторяться от 8 до 16 раз.

Вывод в терминале:

```
PS C:\Users\manul\Desktop\ITiP\Lab 5> java PasswordVer.java
Введите пароль:
Dasha20051606
Корректный пароль
PS C:\Users\manul\Desktop\ITiP\Lab 5> java PasswordVer.java
Введите пароль:
manullll
Некорректный пароль
```

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.

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

Импортируются классы Pattern и Matcher для работы с регулярными выражениями.

```
public static void main(String[] args) {
        String text1 = "The quick brown fox jumps over the lazy dog";
        String text2 = "FloppaIsABigRussianCat";
        String text3 = "aaaAAaaaAaAaaaAAAAaaA";
```

Данные, которые будут проверяться.

```
try {
	String result1 = searchLowUpper(text1);
	String result2 = searchLowUpper(text2);
	String result3 = searchLowUpper(text3);
	System.out.println(result1);
	System.out.println(result2);
	System.out.println(result3);
} catch (Exception e) {
	System.out.println("Найдена ошибка: " + e.getMessage());
}
```

Блок try-catch для обработки возможных исключений при вводе данных. Вызов функции searchLowUpper и вывод результата.

Функция searchLowUpper:

```
public static String searchLowUpper(String text) {
	Pattern pattern = Pattern.compile("([a-z])([A-Z])");
	Matcher matcher = pattern.matcher(text);
	String result = matcher.replaceAll("!$1$2!");

	if (result.equals(text)) {
		return "Таких мест нет";
	} else {
		return result;
	}
}
```

Здесь создается объект Pattern, который содержит регулярное выражение ([a-z])([A-Z]). Это выражение ищет последовательности, где строчная буква (от a до z) предшествует заглавной букве (от A до Z).

- **([a-z])** — первая группа, которая соответствует любой строчной букве.
- **([A-Z])** — вторая группа, которая соответствует любой заглавной букве.

Создается объект Matcher, который будет использоваться для поиска совпадений в строке text.

Метод replaceAll заменяет все найденные совпадения на строку !$1$2!, где $1 и $2 представляют собой содержимое первой и второй групп соответственно. Таким образом, каждая пара строчной и заглавной буквы будет обрамлена символами !.

Вывод результата в терминале:

```
PS C:\Users\manul\Desktop\ITiP\Lab 5> java AaAaA.java
Таких мест нет
Flopp!aI!!sA!Bi!gR!ussia!nC!at
aa!aA!Aaa!aA!!aA!aa!aA!AAAa!aA!
```

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.

```
public static void main(String[] args) {
	String IP1 = "192.168.0.1";
	String IP2 = "500.168.0.1";
	String IP3 = "0w0.0w0.0.0";

	try {
		ipVer(IP1);
		ipVer(IP2);
		ipVer(IP3);
	} catch (Exception e) {
		System.out.println("Произошла ошибка: " + e.getMessage());
	}
 }
```

Также импортируются классы Pattern и Matcher для работы с регулярными выражениями и проводится проверка исключений.

Метод ipVer для проверки корректности IP-адреса:

```
public static void ipVer(String IP) {
	Pattern pattern = Pattern.compile("^(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]?)$");
	Matcher matcher = pattern.matcher(IP);
	boolean found = false;
	while (matcher.find()) {
		System.out.println("Корректный адрес");
		found = true;
	}
	if (!found) {
		System.out.println("Некорректный адрес");
	}
}
```

Pattern pattern = Pattern.compile("^(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]?)$");

Здесь создается объект Pattern, который содержит регулярное выражение для проверки формата IP-адреса. Регулярное выражение выглядит следующим образом:

^(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]?)$

1. ^ — указывает на начало строки.
2. **(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)** — соответствует первой октете IP-адреса:
    - 25[0-5] — соответствует числам от 250 до 255.
    - 2[0-4][0-9] — соответствует числам от 200 до 249.
    - [01]?[0-9][0-9]? — соответствует числам от 0 до 199 (может быть 1 или 2 цифры).
3. **\\.** — соответствует символу точки.
4. Повторяются предыдущие шаги для каждой из трех оставшихся октетов.
5. **$** — указывает на конец строки.

Вывод в терминале:

```
PS C:\Users\manul\Desktop\ITiP\Lab 5> java IPadress.java
Корректный адрес
Некорректный адрес
Некорректный адрес
```

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.

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

public class isA {
    public static void main(String[] args) {
        String text1 = "The quick brown fox jumps over the lazy dog";
        String text2 = "Apple and orange";
        String let = "a";
        char letter = let.charAt(0);
        try {
            findWordsStartingWith(text1, letter);
            findWordsStartingWith(text2, letter);
        } catch (Exception e) {
            System.out.println("Произошла ошибка: " + e.getMessage());
        }
    }
```

Также импортируются классы Pattern и Matcher для работы с регулярными выражениями и проводится проверка исключений. Выводится список подходящих слов.

Метод findWordsStartingWith:

```
    public static void findWordsStartingWith(String text, char letter) {

        Pattern pattern = Pattern.compile("\\b" + letter + "\\w*",
Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(text);
        boolean found = false;
        System.out.println("Найденные слова:");
        while (matcher.find()) {
            System.out.println(matcher.group());
            found = true;
        }
        if (!found) {
            System.out.println("Слов начинающихся с буквы '" + letter + "' не
найдено");
        }
    }
}
```

Pattern pattern = Pattern.compile("\\b" + letter + "\\w*", Pattern.CASE_INSENSITIVE);

Создается объект Pattern, который содержит регулярное выражение для поиска слов, начинающихся с заданной буквы. Регулярное выражение выглядит следующим образом:

\\b + letter + \\w*

Подробный разбор регулярного выражения:

1. **\\b** — обозначает границу слова. Это означает, что поиск будет начинаться с начала слова.
2. **letter** — это символ, с которого должно начинаться слово. Он добавляется к регулярному выражению.
3. **\\w*** — соответствует нулю или более буквенно-цифровым символам (буквы, цифры и символ подчеркивания). Это позволяет захватывать все символы, следующие за начальной буквой.

Также используется флаг Pattern.CASE_INSENSITIVE, чтобы сделать поиск нечувствительным к регистру.

Вывод результата в терминале:

```
PS C:\Users\manul\Desktop\ITiP\Lab 5> java isA.java
Найденные слова:
Слов начинающихся с буквы 'a' не найдено
Найденные слова:
Apple
and
```

Вывод: В ходе лабораторной работы были изучены и применины регулярные выражения для решения различных задач, связвнных с обработкой текстовой информации.

Приложение

Полный листинг кода:

Задача 1. NumberFinder.java

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

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

public class NumberFinder {
    public static void main(String[] args) {
        String text = "The price of the product is $19.99";

        try {
            findNumbers(text);
        } catch (Exception e) {
            System.out.println("Произошла ошибка: " + e.getMessage());
        }
    }

    public static void findNumbers(String text) {
        Pattern pattern = Pattern.compile("\\d+\\.\\d+");
        Matcher matcher = pattern.matcher(text);
        boolean found = false;
        while (matcher.find()) {
            System.out.println(matcher.group());
            found = true;
        }

         if (!found) {
            System.out.println("Чисел нет");
        }
    }
}
```

Задача 2. PasswordVer.java

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

public class PasswordVer {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите пароль: ");

        try {
            String password = scanner.nextLine();
            passwordVer(password);
        } catch (Exception e) {
            System.out.println("Произошла ошибка: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
```

```
    public static void passwordVer(String password) {
        Pattern pattern = Pattern.compile("^(?=.*[A-Z])(?=.*\\d)[A-Za-
z\\d]{8,16}");
        Matcher matcher = pattern.matcher(password);
        boolean found = false;
        while (matcher.find()) {
            System.out.println("Корректный пароль");
            found = true;
        }

        if (!found) {
            System.out.println("Некорректный пароль");
        }
    }
}
```

Задача 3. AaAaA.java

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

public class AaAaA {
    public static void main(String[] args) {
        String text1 = "The quick brown fox jumps over the lazy dog";
        String text2 = "FloppaIsABigRussianCat";
        String text3 = "aaaAAaaaAaAaaaAAAAaaA";
        try {
            String result1 = searchLowUpper(text1);
            String result2 = searchLowUpper(text2);
            String result3 = searchLowUpper(text3);
            System.out.println(result1);
            System.out.println(result2);
            System.out.println(result3);
        } catch (Exception e) {
            System.out.println("Найдена ошибка: " + e.getMessage());
        }
    }

    public static String searchLowUpper(String text) {
        Pattern pattern = Pattern.compile("([a-z])([A-Z])");
        Matcher matcher = pattern.matcher(text);
        String result = matcher.replaceAll("!$1$2!");

        if (result.equals(text)) {
            return "Таких мест нет";
        } else {
            return result;
        }
```

```
    }
```

Задача 4. IPadress.java

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

public class IPadress {
    public static void main(String[] args) {
        String IP1 = "192.168.0.1";
        String IP2 = "500.168.0.1";
        String IP3 = "0w0.0w0.0.0";

        try {
            ipVer(IP1);
            ipVer(IP2);
            ipVer(IP3);
        } catch (Exception e) {
            System.out.println("Произошла ошибка: " + e.getMessage());
        }
    }

    public static void ipVer(String IP) {
        Pattern pattern = Pattern.compile("^(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]?)$");
        Matcher matcher = pattern.matcher(IP);
        boolean found = false;
        while (matcher.find()) {
            System.out.println("Корректный адрес");
            found = true;
        }
         if (!found) {
            System.out.println("Некорректный адрес");
        }
    }
}
```

Задача 5. isA.java

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

public class isA {
    public static void main(String[] args) {
        String text1 = "The quick brown fox jumps over the lazy dog";
        String text2 = "Apple and orange";
        String let = "a";
        char letter = let.charAt(0);
        try {
            findWordsStartingWith(text1, letter);
            findWordsStartingWith(text2, letter);
```

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

    public static void findWordsStartingWith(String text, char letter) {
        Pattern pattern = Pattern.compile("\\b" + letter + "\\w*",
Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(text);
        boolean found = false;
        System.out.println("Найденные слова:");
        while (matcher.find()) {
            System.out.println(matcher.group());
            found = true;
        }
        if (!found) {
            System.out.println("Слов начинающихся с буквы '" + letter + "' не
найдено");
        }
    }
}
```