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»

(МТУСИ)

Кафедра «Математическая Кибернетика и Информационные технологии»

# ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

по дисциплине

«Информационные технологии и программирование»

на тему

“Работа с регулярными выражениями”

Выполнил:

студент группы БВТ2302

Миронов А. А.

Москва, 2024 г.

**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.Pattern;
import java.util.regex.PatternSyntaxException;
import java.util.regex.Matcher;
public class RegexNum{
    Run | Debug
    public static void main(String[] args) {
        try{
        //String text = "number 90 number $91 number $92.05 number 93.15";
        String text = "number";
        Pattern ptrn = Pattern.compile(regex:"(\\d+\\.\\d+)|\\d+");
        Matcher mtcr = ptrn.matcher(text);
        if (mtcr.find()==false){
            System.out.println(x:"Oops, Ze r no any of numbers in da str:/");
        }
        while (mtcr.find()){
            System.out.println(mtcr.group());
        }
        } catch (PatternSyntaxException p){
        System.out.println(x:"Your regex pattern composed with some mistakes:/");
        p.printStackTrace();
    }

}
}
```

Сначала нужно импортировать необходимые классы, такие как Pattern (который мы используем для компиляции строки в регекс) и Matcher (который мы используем, чтобы находить соответствие скомпилированного регекса с определённое строкой. Наш регекс тут непосредственно является шаблоном, соответствующим либо числу с неограниченным количеством цифр, либо числу с неограниченным количеством цифр до и после точки (то есть дробному числу)— заметим, что точка в данном случае выступает не как специальный символ, а как просто символ точки, потому что мы экранировали её с помощью обратного слэша, но так как обратный слэш в джава является тоже специальным символом в строке, то чтобы не получить ошибку, мы экранируем этот

обратный слэш с помощью ещё одного обратного слэша, чтобы в итоге получить «обычный» для джавы слэш, который уже будет «необычным» в контексте регекса и успешно сэкранирует нашу точку . Далее мы прописываем условие, которое будет выполняться при строке, не содержащей чисел — выведется специальное сообщение. Метод find класса matcher находит следующее соответствие шаблону регекс в строке и возвращает true, если эта операция успешна, также find() запоминает сам текст соответствия и его позицию в строке.  Метод group() класса matcher возвращает последнее соответствие, найденное find в строковом формате. С помощью связки find, match и цикла while мы выводим все найденные в тексте числа на экран. Также мы оборачиваем весь наш код внутри мэйна в блок try и с помощью catch ловим и обрабатываем исключение PatternSyntaxException, которое возникает при ошибках в компиляции строки в регекс, связанными с тем, что строка-регекс написана с неверными синтаксисом.

**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.

```java
import java.util.regex.*;
public class Password {
Run | Debug
public static void main(String[] args) {
    try{
    String text = "HIkidsss1205";
    Pattern pat = Pattern.compile(regex:"(?=.*[A-Z])(?=.*\\d)[A-Za-z0-9]{8,16}");
    Matcher mat = pat.matcher(text);

    if (mat.matches()){
        System.out.println(x:"Valid password");
    } else {
        System.out.println(x:"password pattern's not correct, try again");

    }
} catch (PatternSyntaxException p){
    System.out.println(x:"Your regex pattern composed with some mistakes:/");
    p.printStackTrace();
}

}💡
}
```

Здесь для задания шаблона пароля в регексе нам нужно включить в него весь допустимый алфавит, задав его в квадратных скобках и с помощью повторения этой части шаблона, соответствующего одной букве от 8 до 16ти раз задать допустимую длину пароля. Так же с помощью конструкций lookahead в самом начале регекса проверяем идёт ли в строке любое количество любых символов и хотя бы одна заглавная латинская буква и аналогичное условие прописываем для любой цифры. Таким образом если в пароле нет, к примеру, ни одной цифры, то он не будет соответствовать этому шаблону регекс. Далее с помощью метода matches класса Matcher проверяем соответствует ли наш пароль полностью нашему шаблону (matches возвращает boolean) и если да, ты выводим одобрительное сообщение, в обратном же случае выводим сообщение о том, что пароль не подходит.

**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.

```java
import java.util.regex.*;
import java.util.ArrayList;
public class ReplExclam {
    Run | Debug
    public static void main(String[] args) {
        try{
        ArrayList<String> elemlist = new ArrayList<>();
        String regex = "[a-z][A-Z]";
        String text = "nUmberA mNi aM";
        Pattern ptrn = Pattern.compile(regex);
        Matcher mtcr = ptrn.matcher(text);
        while (mtcr.find()) {
            elemlist.add(mtcr.group());
           text = text.replaceFirst(regex, replacement:"*");
        }
        for (String elem : elemlist){
            text = text.replaceFirst(regex:"\\*", "!"+elem+"!");
        }
        System.out.println(text);
    } catch (PatternSyntaxException p){
        System.out.println(x:"Your regex pattern composed with some mistakes:/");
        p.printStackTrace();
    }
    }

}
```

В данном случае шаблоном регексом будет являться просто связка любой строчной и заглавной буквы, идущих последовательно. Ищем все подобные связки и заменяем их специальным символом, по мере нахождения каждой связки складываем их в список, чтобы не получилось путаницы при итоговых заменах. Затем заменяем последовательно каждый специальный символ на соответствующую связку из списка (важно, чтобы при заполнении списка связками был сохранён порядок появления связок в строке), окружённую восклицательными знаками с помощью replaceFirst.

**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.

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

public class IP{
    Run | Debug
    public static void main(String[] args){
        try{
        String regex = "((25[0-5]|2[0-4][0-9]|[1]?[0-9]?[0-9])\\.){3}(25[0-5]|2[0-4][0-9]|[1]?[0-9]?[0-9])";
        String IP = "0.199.24.255";
        Pattern pat = Pattern.compile(regex);
        Matcher mat = pat.matcher(IP);
        if (mat.matches()){
            System.out.println(x:"valid IP");
        }else {
            System.out.println(x:"invalid IP");
        }
    } catch (PatternSyntaxException p){
        System.out.println(x:"Your regex pattern composed with some mistakes:/");
        p.printStackTrace();
    }
    }
}
```

Здесь также используем matches для проверки того, что именно вся строка целиком соответствует айпи адресу и не содержит каких либо лишних символов. Наше регулярное выражение в данной задаче состоит из 4ёх одинаковых групп, являющихся шаблоном для числа от 0 до 255 с тем лишь отличием, что после последнего шаблона не ставится точка. Сам же шаблон 0-255 включает в себя три других шаблона разделённые или, охватывающих числа 0-199, 200-249 и 250-255.

**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.

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

public class Beginlet{
    Run | Debug
    public static void main(String[] args){
        try{
        String keylet = "T";
        String text = "Table stand in the ceter of Tom's room for long time";
        String regex = "\\b"+keylet+"\\S*\\b";
        Pattern pat = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
        Matcher mat = pat.matcher(text);
        while (mat.find()){
            System.out.println(mat.group());
        }
    } catch (PatternSyntaxException p){
        System.out.println(x:"Your regex pattern composed with some mistakes:/");
        p.printStackTrace();
    }
    }
}
```

В данной задаче для нахождения слова, начинающегося с определённой буквы, обозначаем в выражении regex границы слова с помощью \\b...\\b, чтобы шаблону соответствовали только отдельные слова, начинающиеся с этой буквы и не включались случаи, когда у нас есть подходящее «слово» в другом слове. Внутри этих границ в шаблоне на первую позицию мы устанавливаем искомую букву, а после неё неопределённое количество непробельных символов, которые будут представлять остальную часть слова. При компиляции регекса в Pattern в методе compile указываем дополнительный параметр CASE_INSENSETIVE, который отвечает за игнорированием скомпилированным регексом регистра, мы сделали это чтобы не писать отдельные регексы для заглавных и строчных букв.

**Вывод:**

Мы научились работать с регулярными выражениями в общем и задавать шаблоны, соответствующие нужным нам видам последовательностей символов в строках. Также мы получили опыт использования Regular Expressions внутри джавы и познакомились со функционалом и применением специальных классов и методов в джава, разработанных для взаимодействия с regex.