Лабораторная работа №5

ОБРАБОТКА СТРОК. ИСПОЛЬЗОВАНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ В JAVA-ПРИЛОЖЕНИЯХ

Цель работы:

Во второй лабораторной работе необходимо реализовать консольное приложение, позволяющее манипулировать строкой, разбив ее на элементы путем использования регулярных выражений.

Указания к работе:

Регулярные выражения – эта система обработки текста, основанная на специальной системе записи образцов для поиска. Образец (*pattern*), задающий правило поиска, по-русски также иногда называют «шаблоном», «маской». Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил. Язык программирования Java также поддерживает регулярные выражения для работы со строками.

Основными классами для работы с регулярные выражения являются класс **java.util.regex.Pattern** и класс **java.util.regex.Matcher.**

Класс **java.util.regex.Pattern** применяется для определения регулярных выражений, для которого ищется соответствие в строке, файле или другом объекте представляющем собой некоторую последовательность символов. Для определения шаблона применяются специальные синтаксические конструкции. О каждом соответствии можно получить больше информации с помощью класса **java.util.regex.Matcher.** Далее приведены основные логические конструкции для задания шаблона. Если в строке, проверяемой на соответствие, необходимо, чтобы в какой-либо позиции находился один из символов некоторого символьного набора, то такой набор (класс символов) можно объявить, используя одну из конструкций, представленных в табл.1.

Таблица 1 – Способы определения классов символов

[abc]	a, b или c
[^abc]	символ, исключая a, b и c
[a-z]	символ между a и z
[a-d[m-p]]	либо между a и d, либо между m и p
[e-z&&[dem]]	e либо m (конъюнкция)

Кроме стандартных классов символов существуют предопределенные классы символов (табл. 2)

Таблица 2 – Дополнительные способы определения классов символов

.	любой символ
\d	[0-9]
\D	[^0-9]
\s	[\t\n\x0B\f\r]
\S	[^ \s]
\w	[a-zA-Z_0-9]
\W	[^\w]
\p{javaLowerCase}	тоже , что и Character.isLowerCase()
\p{javaUpperCase}	тоже, что и Character.isUpperCase()

При создании регулярного выражения могут использоваться логические операции (табл.3).

Таблица 3 – Способы задания логических операций

XY	После X следует Y
X|Y	X либо Y
(X)	X

Скобки, кроме их логического назначения, также используются для выделения групп. Для определения регулярных выражений недостаточно одних классов символов, т. к. в шаблоне часто нужно указать количество повторений. Для этого существуют квантификаторы (табл. 4).

Таблица 4 – Квантификаторы

X?	X один раз или ни разу
X*	X ноль или более раз
X+	X один или более раз
X{n}	X n раз
X{n,}	X n или более раз
X{n,m}	X от n до m

Существует еще два типа квантификаторов, которые образованы прибавлением суффикса **?** (слабое или неполное совпадение) или **+** («жадное» или собственное

совпадение) к вышеперечисленным квантификаторам. Неполное совпадение соответствует выбору с наименее возможным количеством символов, а собственное – с максимально возможным.

Примеры использования регулярных выражений для поиска и замены текста можно посмотреть здесь и здесь.

Класс **Pattern**

Класс Pattern используется для простой обработки строк. Для более сложной обработки строк используется класс Matcher, рассматриваемый ниже.

В классе Pattern объявлены следующие методы:

- compile(String regex) – возвращает Pattern, который соответствует regex;
- matcher(CharSequence input) – возвращает Matcher, с помощью которого можно находить соответствия в строке input;
- matches(String regex, CharSequence input) – проверяет на соответствие строки input шаблону regex;
- pattern() – возвращает строку, соответствующую шаблону;
- split(CharSequence input) – разбивает строку input, учитывая, что разделителем является шаблон;
- split(CharSequence input, int limit) – разбивает строку input на не более чем limit частей.

С помощью метода **matches**() класса Pattern можно проверять на соответствие шаблону целой строки, но если необходимо найти соответствия внутри строки, например, *определять участки*, которые соответствуют шаблону, то класс Pattern **не может** быть использован. Для таких операций необходимо использовать класс Matcher.

Класс **Matcher**

Начальное состояние объекта типа Matcher не определено. Попытка вызвать какой-либо метод класса для извлечения информации о найденном соответствии приведет к возникновению ошибки IllegalStateException. Для того чтобы начать работу с объектом Matcher нужно вызвать один из его методов:

- matches() – проверяет, соответствует ли вся строка шаблону;
- lookingAt() – пытается найти последовательность символов, начинающуюся с начала строки и соответствующую шаблону;

- find() или find(int start) – пытается найти последовательность символов, соответствующих шаблону, в любом месте строки. Параметр start указывает на начальную позицию поиска.

Иногда необходимо сбросить состояние объекта класса Matcher в исходное, для этого применяется метод **reset()** или **reset(CharSequence input)**, который также устанавливает новую последовательность символов для поиска.

Для замены всех подпоследовательностей символов, удовлетворяющих шаблону, на заданную строку можно применить метод **replaceAll(String replacement).**

Для того чтобы ограничить поиск границами входной последовательности применяется метод region(int start, int end), а для получения значения этих границ – regionEnd() и regionStart(). С регионами связано несколько методов:

- useAnchoringBounds(boolean b) – если установлен в true, то начало и конец региона соответствуют символам **^ и $** соответственно;
- hasAnchoringBounds() – проверяет закрепленность границ.

В регулярном выражении для более удобной обработки входной последовательности применяются группы, которые помогают выделить части найденной подпоследовательности. В шаблоне они обозначаются скобками «(» и «)». Номера групп начинаются с единицы. Нулевая группа совпадает со всей найденной подпоследовательностью. Далее приведены методы для извлечения информации о группах:

- end() – возвращает индекс последнего символа подпоследовательности, удовлетворяющей шаблону;
- end(int group) – возвращает индекс последнего символа указанной группы;
- group() – возвращает всю подпоследовательность, удовлетворяющую шаблону;
- group(int group) – возвращает конкретную группу;
- groupCount() – возвращает количество групп;
- start() – возвращает индекс первого символа подпоследовательности, удовлетворяющей шаблону;
- start(int group) – возвращает индекс первого символа указанной группы;
- hitEnd() – возвращает истину, если был достигнут конец входной последовательности.

Следующий **пример** показывает использование возможностей классов Pattern и Matcher, для поиска, разбора и разбиения строк.

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

public class DemoRegular {

    public static void main(String[] args) {
    // проверка на соответствие строки шаблону
    Pattern p1 = Pattern.compile("a*y");
    Matcher m1 = p1.matcher( "aaay" );
    boolean b = m1.matches();
    System.out.println(b);
    // поиск и выбор подстроки, заданной шаблоном
    String regex = "(\\w+)@(\\w+\\.)(\\w+)(\\.\\w+)*" ;
    String s = "адреса эл.почты: mymail@tut.by и rom@bsu.by";
    Pattern p2 = Pattern. compile (regex);
    Matcher m2 = p2.matcher(s);
    while (m2.find()) {
    System.out.println("e-mail: " + m2.group());
    }
    // разбиение строки на подстроки с применением шаблона в качестве
    // разделителя
    Pattern p3 = Pattern.compile("\\d+\\s?");
    String[] words = p3.split("java5tiger 77 java6mustang");
    for (String word : words)
    System.out.println(word);
    }
}
```

В результате будет выведено:

```
true
e-mail: mymail@tut.by
e-mail: rom@bsu.by
java
tiger
java
mustang
```

Следующий **пример** демонстрирует возможности использования групп, а также собственных и неполных квантификаторов.

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

public class Groups {
    public static void main(String[] args) {
    String input = "abdcxyz";
    myMatches("([a-z]*)([a-z]+)", input);
    myMatches("([a-z]?)([a-z]+)", input);
    myMatches("([a-z]+)([a-z]*)", input);
    myMatches("([a-z]?)([a-z]?)", input);
    }

    public static void myMatches(String regex,
    String input) {
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(input);

    if(matcher.matches()) {
    System.out.println("First group: "
    + matcher.group(1));
    System.out.println("Second group: "
    + matcher.group(2));
    } else
        System.out.println("nothing");
        System.out.println();
```

```
        }
    }
```

Результат работы программы:

```
First group: abdcxy
Second group: z

First group: a
Second group: bdcxyz

First group: abdcxyz
Second group:

nothing
```

В первом случае к первой группе (First group) относятся все возможные символы, но при этом остается минимальное количество символов для второй группы (Second group). Во втором случае для первой группы выбирается наименьшее количество символов, т. к. используется *слабое* совпадение. В третьем случае первой группе будет соответствовать вся строка, а для второй не остается ни одного символа, так как вторая группа использует *слабое* совпадение. В четвертом случае строка не соответствует регулярному выражению, т. к. для двух групп выбирается наименьшее количество символов.

В классе Matcher объявлены два полезных метода для замены найденных подпоследовательностей во входной строке.

Matcher appendReplacement(StringBuffer sb, String replacement) – метод читает символы из входной стоки и добавляет их в sb. Чтение останавливается на start() – 1 позиции предыдущего совпадения, после чего происходит добавление в sb строки replacement. При следующем вызове этого метода, производится добавление символов, начиная с символа с индексом end() предыдущего совпадения.

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Написать регулярное выражение, определяющее является ли данная строка строкой "abcdefghijklmnopqrstuv18340" или нет.

– пример правильных выражений: abcdefghijklmnopqrstuv18340.
– пример неправильных выражений: abcdefghijklmnoasdfasdpqrstuv18340.

2. Написать регулярное выражение, определяющее является ли данная строка GUID с или без скобок. Где GUID это строчка, состоящая из 8, 4, 4, 4, 12 шестнадцатеричных цифр разделенных тире.

– пример правильных выражений: e02fd0e4-00fd-090A-ca30-0d00a0038ba0.
– пример неправильных выражений: e02fd0e400fd090Aca300d00a0038ba0.

3. Написать регулярное выражение, определяющее является ли заданная строка правильным MAC-адресом.

– пример правильных выражений: aE:dC:cA:56:76:54.
– пример неправильных выражений: 01:23:45:67:89:Az.

4. Написать регулярное выражение, определяющее является ли данная строчка валидным URL адресом. В данной задаче правильным URL считаются адреса http и https, явное указание протокола также может отсутствовать. Учитываются только адреса, состоящие из символов, т.е. IP адреса в качестве URL не присутствуют при проверке. Допускаются поддомены, указание порта доступа через двоеточие, GET запросы с передачей параметров, доступ к подпапкам на домене, допускается наличие якоря через решетку. Однобуквенные домены считаются запрещенными. Запрещены спецсимволы, например «–» в начале и конце имени домена. Запрещен символ «_» и пробел в имени домена. При составлении регулярного выражения ориентируйтесь на список правильных и неправильных выражений заданных ниже.

– пример правильных выражений: http://www.example.com, http://example.com.
– пример неправильных выражений: Just Text, http://a.com.

5. Написать регулярное выражение, определяющее является ли данная строчка шестнадцатиричным идентификатором цвета в HTML. Где #FFFFFF для белого, #000000 для черного, #FF0000 для красного и т.д.

– пример правильных выражений: #FFFFFF, #FF3421, #00ff00.
– пример неправильных выражений: 232323, f#fddee, #fd2.

6. Написать регулярное выражение, определяющее является ли данная строчка датой в формате dd/mm/yyyy. Начиная с 1600 года до 9999 года.

– пример правильных выражений: 29/02/2000, 30/04/2003, 01/01/2003.
– пример неправильных выражений: 29/02/2001, 30-04-2003, 1/1/1899.

7. Написать регулярное выражение, определяющее является ли данная строчка валидным E-mail адресом согласно RFC под номером 2822.

– пример правильных выражений: user@example.com, root@localhost
– пример неправильных выражений: bug@@@com.ru, @val.ru, Just Text2.

8. Составить регулярное выражение, определяющее является ли заданная строка IP адресом, записанным в десятичном виде.

– пример правильных выражений: 127.0.0.1, 255.255.255.0.
– пример неправильных выражений: 1300.6.7.8, abc.def.gha.bcd.

9. Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов. Где символом может быть английская буква, цифра и знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру.

– пример правильных выражений: C00l_Pass, SupperPas1.
– пример неправильных выражений: Cool_pass, C00l.

10. Проверить является ли заданная строка шестизначным числом, записанным в десятичной системе счисления без нулей в старших разрядах.

– пример правильных выражений: 123456, 234567.
– пример неправильных выражений: 1234567, 12345.

11. Есть текст со списками цен. Извлечь из него цены в USD, RUR, EU.

– пример правильных выражений: 23.78 USD.
– пример неправильных выражений: 22 UDD, 0.002 USD.

12. Проверить существуют ли в тексте цифры, за которыми не стоит «+».

– пример правильных выражений: (3 + 5) – 9 × 4.
– пример неправильных выражений: 2 * 9 – 6 × 5.

13. Создать запрос для вывода только правильно написанных выражений со скобками (количество открытых и закрытых скобок должно быть одинаково).

– пример правильных выражений: (3 + 5) – 9 × 4.
– пример неправильных выражений: ((3 + 5) – 9 × 4.

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

public class NameChecker {

    private final static Pattern pattern = Pattern.compile("(^\\S+\\b)\\s(\\b\\S+$)");

    public static void main(String[] args) {

        String str1 = "Hello Helen"; // true
        String str2 = "Good day"; // false
        String str3 = "Good good!"; // false

        System.out.println(isValid(str1));
        System.out.println(isValid(str2));
        System.out.println(isValid(str3));

    }

    private static boolean isValid(String str) {
        final Matcher m = pattern.matcher(str);
        return m.matches() && m.groupCount() == 2 && m.group(1).length() == m.group(2).length();
    }
}
```