Регулярные выражения

Регулярные выражения используются в случае, если необходимо выполнить проверку текста в строке на соответствие определенному шаблону.

Пример: в тексте необходимо найти числа, в том числе отрицательные, то есть последовательности цифр, которые могут начинаться со знака минус.

Регулярное выражение, описывающее такое сочетание знаков, можно записать несколькими способами:

-?[0123456789]+

-?[0-9]+

-?\\d+

Все три варианта начинаются с сочетания -?

Знак вопроса означает, что стоящий перед ним символ может встретиться один раз или ни разу. Последовательность знаков в квадратных скобках означает, что это может быть любой из перечисленных знаков. Знак плюс после скобок означает, что такой знак должен встречаться один или более раз.

Если знаки идут в кодовой таблице подряд, можно указать первый и второй через тире, что и было использовано во втором примере. В третьем примере используется спецсимвол **\d**, указывающий, что это должна быть цифра.

Для поиска чисел можно использовать любой из приведенных шаблонов.

Методы класса String для работы с регулярными выражениями

В простейшем случае, можно воспользоваться встроенным в класс **String** методом matches.

```
System.out.println("20934".matches("-?[0-9]+"));
```

Результатом выполнения этого выражения будет вывод **true**.

Метод **matches** получает на вход регулярное выражение и проверяет соответствие всей строки целиком этому выражению. Результатом будет логическое значение **true** или **false**.

Еще один полезный метод, использующий регулярные выражения, – метод **split**. Данный метод разрезает строку на части, и возвращает массив. На вход метод получает регулярное выражение, строка разрезается там, где ее фрагменты соответствуют выражению.

```
String s = "Test string for split";
```

System.out.println(Arrays.toString(s.split(" +"))); Результатом будет вывод массива:

[Test, string, for, split]

В данном случае строка была разрезана там, где встречаются пробелы. Регулярное выражение позволило исключить влияние нескольких пробелов. Также следует обратить

внимание на то, что фрагмент строки, являющийся разделителем (пробелы в нашем случае) вырезается из конечного результата.

Еще два метода, использующие регулярные выражения, – это методы **replaceFirst** и **replaceAll**. Эти методы позволяют выполнить поиск и замену в строке.

System.out.println("Test multiple spaces". **replaceAll**(" +", " "));

Результатом такого выражения будет вывод:

Test multiple spaces

Таким образом, метод **replaceAll** находит фрагмент строки, соответствующий регулярному выражению, полученному первым аргументом, и заменяет его строкой, полученной как второй аргумент. В нашем случае произвольное количество пробелов заменяется единичным пробелом. Метод **replaceFirst** работает аналогично, но заменяет только первое найденное совпадение.

Классы Pattern и Matcher

Более мощные средства для работы с регулярными выражениями предлагают классы **Pattern** и **Matcher**. Класс **Pattern** служит для хранения регулярного выражения, а **Matcher** служит для выполнения операций поиска и сравнения.

Для их использования следует сначала задать регулярное выражение и создать объект **Pattern**. Например, необходимо найти в тексте все тире, окруженные пробелами:

```
Pattern p = Pattern.compile(" +- +");
```

Объект создается статическим методом compile, получающим на вход регулярное выражение. Затем следует создать объект класса **Matcher**, указав, строку, с которой будут выполняться все последующие операции поиска:

```
Matcher m = p.matcher("Test – string – test");
```

Объект **Matcher** создается методом matcher, получающим на вход строку. После этого можно выполнять различные операции с данной строкой и регулярным выражением.

Одна из наиболее часто используемых операций – поиск. Поиск выполняется с помощью метода find:

```
m.find()
```

Метод ищет соответствие заданному регулярному выражению в строке и возвращает истину, если соответствие найдено, и ложь – если нет. При повторном вызове данного метода он продолжает поиск и возвращает истину, если найдено еще одно соответствия.

Местоположение найденного совпадения можно найти с помощью методов start и end, которые возвращают позиции начала соответствия и конца. Например:

```
while(m.find()) {
        System.out.println(m.start()+ " "+m.end());
}
```

В данном примере находятся соответствия и выводятся через пробел позиции их начала и конца. Для нашей строки результат будет таким:

4 7

13 16

Регулярным выражениям, как правило, могут соответствовать разные строки. Для того, чтобы узнать, как выглядит очередная найденная методом **find** строка, можно использовать метод m.group().

Для объекта класса **Matcher** имеются также:

- метод **matches**, проверяющий соответствие строки регулярному выражению;
- метод **lookingAt**, ищущий соответствие только в самом начале строки, с ее первого знака.

Если все операции с первоначальной строкой закончены и надо работать с новой, то не обязательно создавать новый объект класса **Matcher**, можно воспользоваться методом reset:

m.reset("New string for search");

Синтаксис регулярных выражений

Специальные символы:

- \xhh – символ с шестнадцатеричным кодом 0xhh \uhhhh Символ Unicode с шестнадцатеричным кодом 0xhhhh;
- \t – табуляция;
- \n – новая строка;
- \r – возврат каретки. Классы символов:
- . – любой символ;
- [abc] – любой из указанных в скобках знаков a, b или c;
- [^abc] – любой символ, кроме a, b или c;
- [a-zA-Z] – любой символ начиная от a заканчивая z, а также от A до Z;
- [abc[hij]] – любой из символов a,b,c,h,j;
- \s – пробельный символ (пробел, перевод строки, табуляция и т.д.);
- \S – любой не пробельный символ;
- \d – цифра, то же самое, что [0-9];
- \D – не цифра, то же самое, что [^0-9];
- \w – буква, то же самое, что [a-zA-Z_0-9] ;
- \W – не буква [^\w]. Обозначения границ:

- ^ – начало строки;
- $ – конец строки;
- \b – граница слова;
- \B – не граница слова;
- \G – конец предыдущего соответствия. Группировка:
- X? – ни одного или один раз;
- X* – любое количество;
- X+ – раз один или больше;
- X{n} – точно n раз (n – число);
- X{n,} – n и больше раз;
- X{n,m} – n или больше, но не больше m.