### Приветствие

Привет! В этом материале вы познакомитесь с мощным инструментом работы со строками — регулярными выражениями. Узнаете, как их писать и как с их помощью решать различные задачи.

### Понятие регулярного выражения

****Регулярные выражения — это специальные выражения, позволяющие описывать множества строк.****

Давайте сразу обратимся к практической задаче. Пример: вы пишете поисковую систему и хотите реализовать метод разбиения текста на отдельные слова. Как это можно сделать? Самый простой способ, который приходит в голову — разделить строку на слова по пробелам. Теперь представьте, что текст, который вам нужно разделить, выглядит так:

****Василий вышел как-то из подъезда, увидел стаю чаек****

****и посчитал, что их около тысячи, а точнее — 700–800.****

Если разделить этот текст на фрагменты просто по пробелам, не все получившиеся фрагменты будут являться словами, а некоторые фрагменты будут содержать не только слова, но и знаки препинания. В решении этой задачи нам как раз могут быть очень полезны регулярные выражения.

В частности, с помощью регулярного выражения можно описать множество символов, по которым надо разделить слова, и включить в это множество не только пробелы, но и все знаки препинания, а также учесть ряд других нюансов, которые могут возникнуть при решении такой задачи. Давайте  теперь разберёмся по порядку, что можно делать при помощи регулярных выражений и как их использовать.

Регулярные выражения позволяют решать такие задачи со строками, как:

* замена в строках одних фрагментов на другие;
* разбивка строки на фрагменты определённым образом;
* проверка соответствия строк определённому шаблону;
* поиск в строках фрагментов, соответствующих шаблону;
* выделение в найденных фрагментах отдельных компонентов.

Ниже вы увидите, как с помощью регулярных выражений можно решать каждую из этих задач.

Мы будем показывать вам процесс составления регулярных выражений по частям — как они пишутся в коде. Будем объяснять каждую часть по мере их составления. В итоге будем формировать финальное выражение, которое будет решать ту или иную задачу.

### Замена одних фрагментов строк на другие

Первая задача, которую вы научитесь решать при помощи регулярных выражений — замена в строках одних фрагментов на другие. Например, у вас есть номера телефонов в разных форматах:

String phone1 = "+7 903 712-37-54";  
String phone2 = "7 (903) 968-60-45";  
String phone3 = "7999-666-66-66";  
String phone4 = "7(903)9616245";

Вам нужно написать метод, который будет приводить номера телефонов к единому формату:

****79037123754****

Чтобы преобразовать номера в такой формат, нужно удалить из строк символы, которые не являются цифрами. Это можно сделать с помощью регулярного выражения:

String regex = "[^0-9]";

В этом регулярном выражении указан диапазон цифр от нуля до девяти, он помещён в квадратные скобки, обозначающие множество символов, а внутри них в начале стоит «крышечка» (^) — символ, обозначающий отрицание множества. То есть сначала обозначено множество цифр от 0 до 9, а затем с помощью «^» указано, что наоборот имеется в виду множество всех символов, кроме цифр.

Применим это регулярное выражение. Создадим специальный метод и напишем в нём вызов метода replaceAll. Этот метод заменяет все фрагменты, соответствующие регулярному выражению, указанному в этом методе в качестве первого параметра, на строку, указанную вторым параметром:

public static String formatPhoneNumber(String phone) {  
    return phone.replaceAll(regex, "");          
}

То есть в строке phone все фрагменты, соответствующие регулярному выражению regex, будут заменены на пустую строку, и результат будет возвращён из этого метода.

В квадратных скобках в регулярных выражениях при этом можно указывать подряд несколько диапазонов, например, так:

String regex = "[0-9a-f]";

Такое регулярное выражение будет соответствовать любому символу, являющемуся цифрой или буквой от a до f.

### Разбиение строк на фрагменты

Следующая задача, которую можно решать с помощью регулярных выражений — это задача разбиения строк на фрагменты. С помощью регулярного выражения мы можем описать множество строк, по которым хотим разделять исходную строку на фрагменты.

Например, вы хотите разбить текст на слова. Возьмём такой текст:

String text = "I know something about it";

Разбить его на слова можно таким образом:

String[] words = text.split("\\s");

Обратный слеш и буква s — вы уже помните — это любой пробельный символ. Метод строки split разделит наш текст по пробелам на пять строк, каждая из которых будет отдельным словом из данного текста.

Обратите внимание, какая переменная получится в итоге:

****String[] words****

Эта переменная является массивом строк. Квадратными скобками обозначаются массивы. Массив — это множество элементов определённого типа (или класса) фиксированной длины. В случае разбиения нашего текста этот массив будет иметь длину 5.

Давайте немного отойдём от темы регулярных выражений и кратко познакомимся с массивами. Есть переменная words, являющаяся массивом и содержащая пять элементов. Как обратиться к каждому такому элементу? Это можно сделать так:

System.out.println(words[0]);

В результате выполнения этого кода в консоль будет выведен нулевой элемент массива, то есть слово I. Можно использовать цикл for и перебрать все элементы массива, начиная с нулевого, следующим образом:

for(int i = 0; i < words.length; i++) {  
    System.out.println(words[i]);  
    }

В результате выполнения этого кода все элементы данного массива, то есть все слова из текста выше, будут выведены в консоль.

Вы познакомитесь с массивами детальнее в одном из следующих модулей. Пока разберёмся подробнее с регулярными выражениями. Представим, что в этом предложении встречаются по два-три пробела, и могут быть пробельные символы — переносы строк и символы табуляции:

String text = "I  know\tsomething   about\n  it";

Регулярное выражение рассчитано на разбиение строк по одному пробелу, а не по нескольким. Чтобы решить эту проблему, добавим символ плюса, означающий, что пробельный символ может встретиться как один раз, так и несколько:

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

Такое выражение также будет работать корректно и разбивать текст на слова по одному или нескольким стоящим подряд пробельным символам.

### Проверка соответствия строк шаблону

Ещё один тип задач, решаемых при помощи регулярных выражений — это задачи определения соответствия строки заданному шаблону.

Представьте, что вы разрабатываете веб-приложение, в котором есть форма с полем ввода номера автомобиля, и значение этого поля попадает в переменную number:

String number = "А674МР197";

Вам нужно проверить корректность введённого значения на стороне сервера. Это можно сделать с помощью регулярного выражения, которое описывает, как могут выглядеть номера автомобилей.

Давайте его напишем. Регулярное выражение — это тоже строка, поэтому сначала создадим соответствующую переменную, в которой мы его будем постепенно составлять:

String regex = "";

В номерах могут быть только те буквы кириллицы, которые имеют латинские аналоги. Напишем эти буквы:

String regex = "АВЕКМНОРСТУХ";

Напомним, что номера автомобилей в России выглядят следующим образом:

****А674МР197****

Начнём с первой буквы номера:

****А674МР197****

Первая буква может быть любой из перечисленных выше. Чтобы это обозначить, используем квадратные скобки, обозначающие набор значений:

String regex = "[АВЕКМНОРСТУХ]";

Это регулярное выражение описывает любую строку, состоящую из буквы из указанного набора.

Далее в номерах автомобилей следуют цифры:

****А674МР197****

Их может быть три. Каждая из них может быть от 0 до 9, то есть у нас будет набор:

String regex = "[АВЕКМНОРСТУХ][0-9]";

Обратите внимание на символ дефиса между цифрами 0 и 9: он обозначает диапазон — сразу все символы от нуля до девяти.

Цифр в номере всегда три. Их количество обозначается в регулярном выражении с помощью фигурных скобок и числа 3 внутри них:

String regex = "[АВЕКМНОРСТУХ][0-9]{3}";

Данное выражение уже будет описывать все строки, состоящие из одной буквы из набора и трёх цифр от 0 до 9.

Далее в номерах снова идут буквы:

****А674МР197****

Фрагмент номера от первой буквы до последних двух можно описать таким выражением:

String regex = "[АВЕКМНОРСТУХ][0-9]{3}[АВЕКМНОРСТУХ]{2}";

В конце — номер региона, который состоит из двух или трёх цифр:

****А674МР197****

Его мы обозначим диапазоном цифр от 0 до 9, каждая из которых встречается от двух до трёх раз — напишем числа 2 и 3 через запятую в фигурных скобках:

String regex ="[АВЕКМНОРСТУХ][0-9]{3}[АВЕКМНОРСТУХ]{2}[0-9]{2,3}";

Существуют строго определённые номера регионов, но их сложно проверить через регулярные выражения, поэтому для простоты оставим цифры от 0 до 9, встречающиеся от двух до трёх раз.

Обратите внимание, что числа 2 и 3 через запятую в фигурных скобках — это не перечисление, а значения «от и до». Например, {2,10} означало бы, что тот или иной символ может встречаться в проверяемой строке от двух до десяти раз.

В составленном регулярном выражении есть повторяющаяся подстрока — буквы. Чтобы упростить выражение, эти буквы лучше вынести в отдельную переменную:

String lettersRange = "[АВЕКМНОРСТУХ]";

Тогда выражение будет выглядеть проще:

String regex = lettersRange + "[0-9]{3}" +lettersRange + "{2}[0-9]{2,3}";

Теперь нужно применить это выражение. У строк в Java есть удобный метод matches, который в качестве параметра принимает регулярное выражение и возвращает true или false:

System.out.println(number.matches(regex));

В итоге получается следующий код:

String number = "А674МР197";  
String lettersRange = "[АВЕКМНОРСТУХ]";  
String regex = lettersRange + "[0-9]{3}" +  
    lettersRange + "{2}[0-9]{2,3}";  
System.out.println(number.matches(regex));

Попробуйте создать отдельный проект в среде разработки, скопировать в него этот код, запустить и посмотреть, как он работает. Он должен вывести в консоль true. Также попробуйте заменить в номере русскую букву «А» на латинскую, запустите код и убедитесь, что теперь выводится false, поскольку русская и латинская буквы «А» являются разными символами.

### Поиск фрагментов строк, соответствующих шаблону

Следующая задача, решаемая при помощи регулярных выражений — это поиск фрагментов в строке, соответствующих определённому фрагменту. Обычно эта задача возникает, когда нужно найти в тексте тот или иной фрагмент. Например, извлечь из текста ссылки на веб-страницы.

Посмотрим, как можно решить такую задачу при помощи регулярных выражений. Представьте, что у вас есть строка, содержащая текст письма, в котором есть ссылки:

String text = "Алексей, добрый день!\nМой гитхаб — https://github.com/, а также ссылка на мой персональный сайт — https://www.skillbox.ru/\nЕсли возникнут вопросы, пишите мне напрямую. Я всегда доступен";

Вы делаете веб-приложение — почтовый клиент. Если один человек отправляет другому письмо со ссылками, то у получателя они должны отобразиться как ссылки — стать синими, подчёркнутыми и вести на нужные страницы.

Для того, чтобы это сделать, нужно извлечь их из текста и найти, где они находятся в тексте. Конечно, мы можем воспользоваться уже известным методом indexOf для поиска фрагментов, начинающихся со строки http, например:

int start = text.indexOf("http");

Но это не очень удобно, так как нужно искать конец ссылки, затем переходить к следующей ссылке и так далее. Есть способ проще — использовать регулярное выражение, описывающее ссылки. Давайте напишем такое регулярное выражение. Начнём с первой части ссылки, которая называется протоколом:

String regex = "https://";

Так выглядит начало любой ссылки, представленной в письме выше. Далее могут быть разные символы, и после ссылки — пробел или запятая. Можно написать, что должны идти любые символы, кроме запятой и пробела:

String regex = "https://[^,\\s]+";

Квадратные скобки, как вы уже знаете, обозначают набор символов, и в нём как раз два символа: запятая и пробельный символ \\s — специальный символ, обозначающий в регулярных выражениях пробел, перенос строки, символ табуляции и ещё некоторые варианты пробельных символов.

Перед пробельным символом стоит ещё один обратный слеш, поскольку обратный слеш — это специальный символ для строк, который тоже необходимо заэкранировать обратным слешем.

«Крышечка» — ^ — в начале набора символов (внутри квадратных скобок) означает отрицание этого набора. То есть это обратный набор — все символы, кроме входящих в этот набор — все символы, кроме пробела и запятой.

Символ плюса после символа, в том числе после набора символов, означает, что эти символы, которые «не запятая» и «не пробельный символ», могут встречаться один или более раз.

Ещё раз разберём, как будет работать такое выражение. Фрагмент https:// совпадает, далее совпадает ещё некоторое количество символов, а потом в тексте встречается запятая или пробельный символ, и на этом поиск первого фрагмента останавливается. Затем идёт поиск следующего фрагмента.

Давайте с помощью этого выражения найдём все ссылки в нашем тексте. Для этого воспользуемся специальным классом Pattern:

Pattern pattern = Pattern.compile(regex);

С помощью специального класса Matcher, использующего паттерн, переберём фрагменты, которые этому паттерну, а точнее переданному в паттерн регулярному выражению, соответствуют:

Matcher matcher = pattern.matcher(text);  
while (matcher.find()) {  
  int start = matcher.start();  
    int end = matcher.end();  
    System.out.println(text.substring(start, end));  
}

 Фрагменты можно получать и более простым способом, используя метод group класса Matcher:

Matcher matcher = pattern.matcher(text);  
while (matcher.find()) {  
  System.out.println(matcher.group());  
}

Скопируйте к себе в проект код из этого блока и запустите его. Не забудьте скопировать переменные text, regex и pattern. После запуска кода убедитесь, что он действительно выводит в консоль отдельные ссылки из текста, содержащегося в переменной text.

### Маски в регулярных выражениях

При использовании регулярных выражений иногда нужно извлечь фрагменты из части строки, которая соответствует заданному в регулярном выражении шаблону.

Представьте, что вам необходимо извлечь из текста ниже все цитаты — фрагменты в кавычках:

****Дмитрий сообщил следующее: «Я вернусь в 12:40 и, будьте добры, подготовьте к этому времени все документы!» На что Анна ему ответила: «А документы-то так и не привезли». Дмитрий удивлённо посмотрел на неё и сказал: «Ну и ладно», — вздохнул, махнул рукой и удалился.****

Для того чтобы извлечь из текста все цитаты, можно написать такое регулярное выражение:

String regex = "«[^»]+»";

Оно будет соответствовать всем цитатам в данном тексте: в начале стоит открывающая кавычка, потом множество символов, кроме закрывающей кавычки, а потом — закрывающая кавычка. Как и в примере выше, можем воспользоваться классами Pattern и Matcher, но что делать, если хочется сразу получить цитаты без кавычек? Для этого можно использовать так называемые маски — фрагменты выражений, выделяемые круглыми скобками:

String regex = "«([^»]+)»";

Обратите внимание, что в круглые скобки помещено содержимое каждой цитаты, а кавычки находятся за скобками. Для работы с масками у класса Matcher есть метод group, в качестве параметра в который можно передавать порядковый номер маски, начиная с единицы. То есть:     matcher.group(1); вернёт содержимое первой маски, matcher.group(2); — второй, а  matcher.group(0); будет возвращать целиком весь соответствующий регулярному выражению текст. Теперь можем написать код:

Pattern pattern = Pattern.compile(regex);  
Matcher matcher = pattern.matcher(text);  
while (matcher.find()) {  
  String citation = matcher.group(1);  
    System.out.println(citation);  
}

Скопируйте к себе в проект код из этого блока и запустите его. Не забудьте создать переменную text и скопировать в неё представленный выше текст.  После запуска кода убедитесь, что он действительно выводит в консоль цитаты без кавычек. Кстати, если в коде будет нарушено количество и очерёдность кавычек, то код будет работать некорректно.

### Элементы синтаксиса регулярных выражений

Регулярные выражения можно использовать для широкого спектра задач обработки строк. Рассмотрим некоторые ключевые элементы регулярных выражений, которые часто используются.

|  |  |
| --- | --- |
| ****Выражение**** | ****Описание**** |
| [abc] | Набор символов. Данное выражение соответствует любому символу из перечисленных в квадратных скобках. Например, регулярное выражение [13579] будет соответствовать символу, являющемуся нечётным числом. |
| [^abc] | Отрицание набора символов. Данное выражение соответствует любому символу, кроме перечисленных в квадратных скобках. Символ-«крышечка» — ^ — не входит в этот набор и обозначает отрицание набора.  Например, регулярное выражение [^xyz] будет соответствовать любому символу, кроме букв x, y и z. |
| [0-9] | Диапазон символов. Дефис означает, что в набор символов входят все цифры от 0 до 9. Например, регулярное выражение [a-z] будет соответствовать любой строчной латинской букве. |
| \\s | Пробельный символ — пробел, символ переноса строки, символ табуляции или иной пробельный символ. |
| n+ | Один или несколько символов n. Например, регулярное выражение [0-9]+ будет соответствовать всем числам, независимо от того, состоят ли они из одной или нескольких цифр. При этом, если в числе есть точка, запятая или хотя бы один пробел, оно не будет соответствовать такому выражению. |
| n\* | Ноль, один или несколько символов n. Например, регулярное выражение [A-Z][a-zA-Z]\* будет соответствовать как просто отдельным прописным буквам, так и всем прописным буквам, после которых идут прописные или строчные буквы в любом количестве, например:  A  An  Article  ANTICS |
| n? | Ноль или один символ n. Например, регулярное выражение https?:// будет соответствовать как строкам http://, так и строкам https://, поскольку после буквы s стоит вопросительный знак, означающий, что буква s может присутствовать или отсутствовать. |
| n{5} | Символ n, встречающийся пять раз. Например, выражение [1-9][0-9]{3} будет соответствовать всем числам, состоящим из четырёх цифр, и начинающихся с цифр от 1 до 9. |
| n{5,} | Символ n, встречающийся пять и более раз. Например, регулярное выражение [A-Z]{2,} будет соответствовать всем строкам, состоящим из двух и более прописных букв. |
| n{5,8} | Символ n, встречающийся пять-восемь раз. Например, регулярное выражение:  [0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}  будет соответствовать всем IP-адресам стандарта IPv4, которые имеют такой вид:  192.168.0.1 0.0.0.0 255.255.255.0  При этом важно, что каждое число в таких адресах может быть в диапазоне от 0 до 255, и проверить это при помощи регулярных выражений сложнее (см. ссылку в допматериалах). |

Это далеко не все элементы. Если вы хотите узнать о регулярных выражениях больше, ознакомьтесь со следующими материалами:

* [Регулярные выражения. Всё проще, чем кажется](https://habr.com/ru/post/567106/" \t "/home/parviz/Documents\\x/_blank)
* [Java — Regular Expressions](https://www.tutorialspoint.com/java/java_regular_expressions.htm" \t "/home/parviz/Documents\\x/_blank)
* [Class Pattern](https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html" \t "/home/parviz/Documents\\x/_blank)
* [Сервис проверки регулярных выражений](https://regex101.com/" \t "/home/parviz/Documents\\x/_blank)
* [Regular Expressions Cookbook: Matching IPv4 Addresses](https://www.oreilly.com/library/view/regular-expressions-cookbook/9780596802837/ch07s16.html" \t "/home/parviz/Documents\\x/_blank)

### Заключение

Итак, в этом материале вы увидели, какими бывают регулярные выражения, как они пишутся, выглядят и работают, а также познакомились с четырьмя примерами задач, в которых их удобно использовать: с заменой одних фрагментов строк на другие, разбиением строк на фрагменты, а также проверкой соответствия строк шаблону и поиском в строках фрагментов, соответствующих шаблону.