Средства Java для работы с регулярными выражениями

В стандартную библиотеку Java входит пакет, специально предназначенный для работы с регулярными выражениями – java.util.regex.  
Эта библиотека может быть использована для выполнения таких задач:

* поиск данных;
* проверка данных;
* выборочное изменение данных;
* выделение фрагментов данных;
* и др..

Получить подробную информацию о библиотеке можно из официальной справки к JavaSDK, размещённой на сайте [java.sun.com](http://java.sun.com/). Здесь же можно почитать The Java Tutorial (учебное пособие по Java), в котором регулярным выражениям посвящён отдельный урок.

В этой статье я хочу показать примеры использования регулярных выражений для проверки правильности ввода данных, и извлечения фрагментов этих данных.  
Основную работу выполняют два класса: Pattern и Matcher.  
Pattern – представляет собой сам шаблон, т.е. наше регулярное выражение.  
Matcher – это объект, который выполняет сравнение заданной строки с шаблоном.  
Для того чтобы создать объект типа Pattern нужно воспользоваться его статическим методом  
compile(< регулярное выражение >) или  
compile(< регулярное выражение >, <параметры>).  
Создать Matcher можно с помощью объекта Pattern.

Например, так:

1. Pattern p = Pattern.compile("^**\\**s+",
2. Pattern.UNICODE\_CASE | Pattern.CASE\_INSENSITIVE);
3. Matcher m = p.matcher(inputString);

После этого можно сравнивать шаблон со строкой, выполнять поиск нужных фрагментов и т.д.  
Но, прежде всего, давайте разберёмся, как составлять регулярные выражения.

В простейшем случае регулярное выражение представляет собой просто текст, например, "стол". Такое выражение можно использовать для поиска слова "стол" в строке, для замены этого слова на какое-нибудь другое и т.д..

Для создания более сложных выражений используются специальные символы (метасимволы). К ним относятся: ([{\^$|)?\*+.. Эти символы имеют специальное значение, которое зависит от их положения в выражении.

Но, что будет, если мы захотим использовать эти символы в выражении в их обычном значении? Для этого нужно перед этим символом поставить \.

Внимание: если регулярное выражение записано в программе, а не читается из файла или формы, то необходимо использовать два символа \ подряд (\\). Вы, конечно, помните, что при обработке строк Java считает символ \ служебным (начало эскейп-последовательности), а для того, чтобы получить просто \ нужно ввести \\.

Теперь посмотрим, как можно использовать специальные символы при составлении выражений. Наверное, наиболее распространённый символ – это ".". Он означает – любой символ. Если поставить справа от символа знак "+", то это будет означать, что предыдущий символ может встречаться несколько раз. Поясню сказанное на простых примерах:  
выражение ".+" будет соответствовать любому тексту (один или более любых символов);  
"A.+" – любое выражение, которое начинается на букву "А".  
Если мы хотим найти текст с известным расположением в строке, то можно воспользоваться символами:  
^ - начало строки  
$ - конец строки

С помощью квадратных скобок ([]) можно задать класс символов. Класс символов задаёт некоторый набор символов, который можно использовать в выражениях. Например, выражение "[кп]от" соответствует словам "кот" и "пот".  
Вот ещё несколько примеров использования классов символов.

|  |  |
| --- | --- |
| [^abc] | любые символы кроме abc (в данном случае символ ^ используется как отрицание) |
| [a-d[m-p]] | любые символы от a до d и от m до p (по алфавиту) |
| [a-z&&[^bc]] | любые символы от a до z кроме b и c |

Кроме этого существуют, так называемые предопределённые классы. Например,

|  |  |
| --- | --- |
| \d | означает любую цифру |
| \D | любые символы кроме цифр |
| \s | пробельные символы ([ \t\n\x0B\f\r]) |
| \S | любые символы кроме пробельных |
| \w | любые текстовые символы [a-zA-Z\_0-9] (как вы заметили сюда символы русского алфавита не входят, как их включить, я покажу позже) |
| \W | любые не текстовые символы |

**Удаление лишних пробелов**  
При вводе данных в форму пользователь может случайно поставить несколько пробелов (или знаков табуляции) до начала текста или в его конце. Эти пробелы обычно не несут никакой информации и их нужно удалить.

В первую очередь создадим необходимые нам шаблоны. Для выделения пробелов в начале строки нам подойдёт "^\\s+". Символ ^ означает начало строки, а \\s+ – один или более пробелов. Для того чтобы удалить пробелы можно воспользоваться методом replaceFirst("") класса Matcher с пустой строкой в качестве параметра. Этот метод заменяет первый найденный фрагмент строки, который соответствует регулярному выражению, на заданную строку.  
Например:

1. Pattern p = Pattern.compile("^**\\**s+");
2. Matcher m = p.matcher(input);
3. String output = m.replaceFirst("");

Для выделения пробелов в конце строки немного изменим шаблон "\\s+$". Практически тоже самое, только в конце выражения стоит символ $, который означает конец строки.

Код для использования шаблона будет выглядеть примерно так:

1. p = Pattern.compile("**\\**s+$");
2. m = p.matcher(output);
3. output = m.replaceAll("");

Это продолжение предыдущего фрагмента кода, поэтому строка output используется как исходная строка. Т.е. сначала мы удаляем пробелы в начале строки, а потом – в её конце. Метод replaceAll, в отличие от replaceFirst заменяет на заданную строку (в данном случае "") все фрагменты строки, которые соответствуют шаблону. Для этой программы не играет роли, каким методом мы воспользуемся, т.к. может быть только одна группа пробелов в начале строки, и одна – в конце.

Вы можете [скачать программу](http://www.simplecoding.org/wp-content/uploads/2007/08/removespaces.zip) для удаления пробелов, которая работает по описанному здесь методу. После запуска введите какую-нибудь строку, например: " привет ". В результате вы получите: "привет".

**Проверка формата телефонного номера**  
Теперь посмотрим более сложный пример. Допустим, у нас есть форма, в которую пользователь должен ввести номер телефона. Телефонные номера обычно записываются в виде групп цифр, разделенных дефисами, но это не обязательно. Между цифрами могут быть пробелы, а могут и не быть. Кроме того, номер может содержать код города, который обычно записывается в круглых скобках.  
Т.е. возможны такие варианты записи:

555-33-44  
555 – 43 – 78  
(333) 44 – 55- 88  
8-(022)-4-333-111  
456789

Наша задача: проверить, насколько это возможно, правильность ввода номера.  
Для этого можно воспользоваться таким регулярным выражением: "[\\d\\s()\\-]+ ". Здесь в квадратных скобках мы определили класс символов, в который входят все цифры \\d, все пробельные символы \\s, круглые скобки и дефис. Знак + в конце выражения означает, что любой из этих символов, может встречаться один или более раз.

Для того чтобы проверить соответствие введенного номера, нашему выражению, воспользуемся методом matches() класса Matcher. Он возвращает true, если заданная строка соответствует регулярному выражению.  
Например:

1. Pattern p = Pattern.compile("[**\\**d**\\**s**\\**(**\\**)**\\**-]+");
2. Matcher m = p.matcher(input);
3. if(m.matches()) {
4. System.out.println("**\"**" + input + "**\"** – OK");
5. }
6. else {
7. System.out.println("**\"**" + input + "**\"** – ERR");
8. }

Теперь попробуем выделить код города (если он есть) из номера. Для этого составим ещё одно регулярное выражение – "\\([\\d]+\\)". Здесь мы задаем последовательность символов. Первой должна идти открывающая круглая скобка, за ней – одна или более цифр, после них – закрывающая круглая скобка.

Для того чтобы найти часть строки, которая соответствует этому выражению, можно воспользоваться методом find() класса Matcher. Этот метод возвращает true, если в исходной строке найдена заданная подстрока. Для того чтобы получить саму подстроку, нужно использовать метод group().

1. Pattern codeP = Pattern.compile("**\\**([**\\**d]+**\\**)");
2. m = codeP.matcher(input);
3. if(m.find()) {
4. System.out.println("City code: " + m.group());
5. }

Вы можете [скачать демонстрационную программу](http://www.simplecoding.org/wp-content/uploads/2007/08/checkphonenumber.zip) для проверки телефонных номеров.

**Разбивка строки на лексемы**  
В первую очередь разберемся, что такое лексемы. Это фрагменты строки, которые находятся между символами разделителями. Например, в обычном предложении, лексемы – это слова, а символы разделители – это пробелы между ними.

Вообще-то, для того, чтобы разбить предложение на слова использовать регулярные выражения не обязательно. Можно воспользоваться классом StringTokenizer из пакета java.util. В его конструкторе можно задать строку с разделителями, а затем в цикле с помощью метода nextToken() получить все лексемы.

Но, что делать, если разделителем служит сложный фрагмент, состоящий из смешанных данных? Рассмотрим конкретный пример. Допустим, у нас есть строка (или строки), которая содержит имена и даты рождения. Мы точно знаем, что после имени идет символ двоеточия, а дата состоит из какого-то набора цифр и дефисов. Имя может состоять из одного или нескольких слов.  
Например:  
Вова:2000-11-3Александр Петрович:1970-02-14Петя:1980-07-1Костя:1990-08-23

Нам нужно выделить все имена из строки. Для этого составим регулярное выражение, которое соответствует разделителю, т.е. символу двоеточия, за которым следует дата. У меня получилось такое выражение ":[\\d\\-]+". Первым идет двоеточие, за ним мы объявляем символьный класс, в который входят все цифры и дефис. Символы из этого класса могут встречаться один или более раз (знак +). После этого мы можем воспользоваться методом split класса Pattern для получения массива строк с именами.  
Код, выполняющий эти операции, будет выглядеть примерно так:  
Например:

1. Pattern p = Pattern.compile(":[**\\**d**\\**-]+");
2. String[] tokens = p.split(input);

Как видите, получить лексемы с помощью класса Pattern значительно проще, чем, используя StringTokenizer.