Mauda
IT, Java and Music

Expressões Regulares – Números e Letras

August 21, 2017 by Mauda (http://www.mauda.com.br/?author=1)

Conteúdo do Post:

Olá Pessoal, tudo bom?

No artigo de hoje iremos começar a explicar algumas expressões regulares úteis para o dia a dia, no caso a trabalhar com números e letras. Veja na continuação desse artigo.

Expressões Regulares

Uma das coisas que, independente da linguagem de programação, poucos desenvolvedores dominam são as expressões regulares. O formato dessas expressões indica uma forma de encontrar um determinado padrão de caracteres (**pattern**) dentro de uma String. Esse formato pode ser muito simples ou extremamente complexo, dependendo do que está sendo usado para compor essa expressão.

A composição desse formato é estabelecida por três categorias:

1. Meta-Caracteres
2. Intervalos e Simbolos reservados
3. Quantificadores

Ainda assim, é possível usar apenas uma dessas categorias e uma expressão regular estará construída. Para maiores detalhes de todas as expressões que existem isoladamente no Java veja a API da *classe* Pattern [1)] API da *classe* Pattern – link (https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html)

Nesse artigo o foco da expressão regular será sobre uma String de tamanho 1, indicando se a String possui letras ou números. Veja na tabela abaixo:

Expressão Regular	**O que significa?**
d	Um Número
D	Qualquer coisa MENOS Número
w	Uma Letra ou Número
W	Qualquer coisa MENOS letra ou número

Agora vamos exemplificar alguns testes somente com Meta-Caracteres. Suponha a *classe* de teste JUnit abaixo:

```
1  package br.com.mauda.expressoes.regulares;
2
3  import org.junit.Assert;
4  import org.junit.Test;
5
6  public class PatternsTeste {
7
```

```
 8      @Test
 9      public void somenteNumeroStringTamanho1() {
10          String stringASerValidada = "A";
11          String pattern = "\\d";
12          Assert.assertFalse(stringASerValidada.matches(pattern));
13
14          stringASerValidada = "1";
15          Assert.assertTrue(stringASerValidada.matches(pattern));
16
17          stringASerValidada = ",";
18          Assert.assertFalse(stringASerValidada.matches(pattern));
19
20          stringASerValidada = "12";
21          Assert.assertFalse(stringASerValidada.matches(pattern));
22
23          stringASerValidada = "1A";
24          Assert.assertFalse(stringASerValidada.matches(pattern));
25
26          stringASerValidada = "AB";
27          Assert.assertFalse(stringASerValidada.matches(pattern));
28
29          stringASerValidada = "A2";
30          Assert.assertFalse(stringASerValidada.matches(pattern));
31      }
32
33      @Test
34      public void semNumeroStringTamanho1() {
35          String stringASerValidada = "A";
36          String pattern = "\\D";
37          Assert.assertTrue(stringASerValidada.matches(pattern));
38
39          stringASerValidada = "1";
40          Assert.assertFalse(stringASerValidada.matches(pattern));
41
42          stringASerValidada = ",";
43          Assert.assertTrue(stringASerValidada.matches(pattern));
44
45          stringASerValidada = "12";
46          Assert.assertFalse(stringASerValidada.matches(pattern));
47
48          stringASerValidada = "1A";
49          Assert.assertFalse(stringASerValidada.matches(pattern));
50
51          stringASerValidada = "AB";
52          Assert.assertFalse(stringASerValidada.matches(pattern));
53
54          stringASerValidada = "A2";
55          Assert.assertFalse(stringASerValidada.matches(pattern));
56      }
57
58      @Test
59      public void somenteLetraOuNumeroStringTamanho1() {
60          String stringASerValidada = "A";
61          String pattern = "\\w";
62          Assert.assertTrue(stringASerValidada.matches(pattern));
63
64          stringASerValidada = "1";

65          Assert.assertTrue(stringASerValidada.matches(pattern));
66
67          stringASerValidada = ",";
68          Assert.assertFalse(stringASerValidada.matches(pattern));
69
70          stringASerValidada = "12";
71          Assert.assertFalse(stringASerValidada.matches(pattern));
```

```
72
73          stringASerValidada = "1A";
74          Assert.assertFalse(stringASerValidada.matches(pattern));
75
76          stringASerValidada = "AB";
77          Assert.assertFalse(stringASerValidada.matches(pattern));
78
79          stringASerValidada = "A2";
80          Assert.assertFalse(stringASerValidada.matches(pattern));
81      }
82
83      @Test
84      public void semLetraOuNumeroStringTamanho1() {
85          String stringASerValidada = "A";
86          String pattern = "\\W";
87          Assert.assertFalse(stringASerValidada.matches(pattern));
88
89          stringASerValidada = "1";
90          Assert.assertFalse(stringASerValidada.matches(pattern));
91
92          stringASerValidada = ",";
93          Assert.assertTrue(stringASerValidada.matches(pattern));
94
95          stringASerValidada = "12";
96          Assert.assertFalse(stringASerValidada.matches(pattern));
97
98          stringASerValidada = "1A";
99          Assert.assertFalse(stringASerValidada.matches(pattern));
100
101         stringASerValidada = "AB";
102         Assert.assertFalse(stringASerValidada.matches(pattern));
103
104         stringASerValidada = "A2";
105         Assert.assertFalse(stringASerValidada.matches(pattern));
106     }
107 }
```

Vamos agora explicar cada método de teste e seus resultados:

método de teste somenteNumeroStringTamanho1()

Esse método possui a premissa de testar uma String, cujo tamanho é 1 e contenha somente números. Assim um exemplo prático é uma String que represente um número de 0 a 9. Assim o único resultado verdadeiro será um número. Strings com tamanho maior que 1 terão resultado falso assim como qualquer outro elemento não sendo um número.

método de teste semNumeroStringTamanho1()

Esse método possui a premissa de testar uma String, cujo tamanho é 1 e não contenha números. Assim um exemplo prático é uma String que represente uma letra de a-z ou A-Z ou mesmo um caractere especial, como uma “,”. Assim no nosso método de teste existem dois resultados verdadeiros, a letra “A” e a virgula “,”. Strings com tamanho maior

que 1 terão resultado falso assim como números.

método de teste somenteLetraOuNumeroStringTamanho1()

Esse método possui a premissa de testar uma String, cujo tamanho é 1 e contenha somente números ou letras. Assim um exemplo prático é uma String que represente uma letra de a-z ou A-Z ou um número de 0 a 9. Assim no nosso método de teste existem dois resultados verdadeiros, a letra “A” e o número “1”. Strings com tamanho maior que 1 terão resultado falso assim como caracteres especiais.

método de teste semLetraOuNumeroStringTamanho1()

Esse método possui a premissa de testar uma String, cujo tamanho é 1 e não contenha números ou letras. Assim um exemplo prático é uma String que represente um caractere especial, como uma “,”. Assim o único resultado verdadeiro será a virgula “,”. Strings com tamanho maior que 1 terão resultado falso assim como letras e números.

Adicionando o quantificador *

Se adicionar o quantificador * as expressões como ficaria nossos resultados? Vamos primeiro modificar nossa tabela:

Expressão Regular	**O que significa?**
*\\d**	Uma String formada apenas por números

*\\D**	Uma String formada por qualquer coisa MENOS Número
*\\w**	Uma String formada apenas por letras ou Números
*\\W**	Uma String formada por qualquer coisa MENOS letras ou números

Entendeu para que serve o quantificador *. Ele indica zero ou mais repetições de um determinado padrão na String a ser avaliada. Agora vamos exemplificar alguns testes somente com Meta-Caracteres. Vamos modificar a *classe* de teste JUnit abaixo com as novas expressões:

```
1  package br.com.mauda.expressoes.regulares;
2
3  import org.junit.Assert;
4  import org.junit.Test;
5
6  public class PatternsTeste {
7
```

```
 8      @Test
 9      public void somenteNumeroStringTamanho1() {
10          String stringASerValidada = "A";
11          String pattern = "\\d*";
12          Assert.assertFalse(stringASerValidada.matches(pattern));
13
14          stringASerValidada = "1";
15          Assert.assertTrue(stringASerValidada.matches(pattern));
16
17          stringASerValidada = ",";
18          Assert.assertFalse(stringASerValidada.matches(pattern));
19
20          stringASerValidada = "12";
21          Assert.assertTrue(stringASerValidada.matches(pattern));
22
23          stringASerValidada = "1A";
24          Assert.assertFalse(stringASerValidada.matches(pattern));
25
26          stringASerValidada = "AB";
27          Assert.assertFalse(stringASerValidada.matches(pattern));
28
29          stringASerValidada = "A2";
30          Assert.assertFalse(stringASerValidada.matches(pattern));
31      }
32
33      @Test
34      public void semNumeroStringTamanho1() {
35          String stringASerValidada = "A";
36          String pattern = "\\D*";
37          Assert.assertTrue(stringASerValidada.matches(pattern));
38
39          stringASerValidada = "1";
40          Assert.assertFalse(stringASerValidada.matches(pattern));
41
42          stringASerValidada = ",";
43          Assert.assertTrue(stringASerValidada.matches(pattern));
44
45          stringASerValidada = "12";
46          Assert.assertFalse(stringASerValidada.matches(pattern));
47
48          stringASerValidada = "1A";
49          Assert.assertFalse(stringASerValidada.matches(pattern));
50
51          stringASerValidada = "AB";
52          Assert.assertTrue(stringASerValidada.matches(pattern));
53
54          stringASerValidada = "A2";
55          Assert.assertFalse(stringASerValidada.matches(pattern));
56      }
57
58      @Test
59      public void somenteLetraOuNumeroStringTamanho1() {
60          String stringASerValidada = "A";
61          String pattern = "\\w*";
62          Assert.assertTrue(stringASerValidada.matches(pattern));
63
64          stringASerValidada = "1";

65          Assert.assertTrue(stringASerValidada.matches(pattern));
66
67          stringASerValidada = ",";
68          Assert.assertFalse(stringASerValidada.matches(pattern));
69
70          stringASerValidada = "12";
71          Assert.assertTrue(stringASerValidada.matches(pattern));
```

```
72
73          stringASerValidada = "1A";
74          Assert.assertTrue(stringASerValidada.matches(pattern));
75
76          stringASerValidada = "AB";
77          Assert.assertTrue(stringASerValidada.matches(pattern));
78
79          stringASerValidada = "A2";
80          Assert.assertTrue(stringASerValidada.matches(pattern));
81      }
82
83      @Test
84      public void semLetraOuNumeroStringTamanho1() {
85          String stringASerValidada = "A";
86          String pattern = "\\W*";
87          Assert.assertFalse(stringASerValidada.matches(pattern));
88
89          stringASerValidada = "1";
90          Assert.assertFalse(stringASerValidada.matches(pattern));
91
92          stringASerValidada = ",";
93          Assert.assertTrue(stringASerValidada.matches(pattern));
94
95          stringASerValidada = "12";
96          Assert.assertFalse(stringASerValidada.matches(pattern));
97
98          stringASerValidada = "1A";
99          Assert.assertFalse(stringASerValidada.matches(pattern));
100
101         stringASerValidada = "AB";
102         Assert.assertFalse(stringASerValidada.matches(pattern));
103
104         stringASerValidada = "A2";
105         Assert.assertFalse(stringASerValidada.matches(pattern));
106     }
107 }
```

Perceba que a maior modificação nos resultados foram os testes com tamanho maior que 1. Agora a *classe* de teste passa a validar também essas Strings que antes sempre retornavam false para qualquer tamanho maior que 1.

finally{

É possível utilizar essas expressões regulares para substituir caracteres em uma String, de uma olhada no método replaceAll [2] API da *classe* String – método replaceAll – link (https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#replaceAll-java.lang.String-java.lang.String-) da *classe* String!

No futuro iremos descrever situações do dia a dia sobre como aplicar expressões regulares, fique ligado!

Duvidas ou sugestões? Deixe seu feedback! Isso ajuda a saber a sua opinião sobre os

artigos e melhorá-los para o futuro! Isso é muito importante!

Até um próximo post!

References

1. ↑ API da *classe* Pattern – link (https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html)
2. ↑ API da *classe* String – método replaceAll – link (https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#replaceAll-java.lang.String-java.lang.String-)

About Mauda

Mestre em Informática, Analista de Sistemas, Professor, SCJP e Baterista. Desde 2002 trabalhando no mundo Java e ensinando pessoas sobre desenvolvimento de sistemas. Mais informações (http://mauda.com.br/?author=1)