fusberti@ic.unicamp.br

Expressões Regulares em Java

prof. Fábio Luiz Usberti

MC322 - Programação Orientada a Objetos

Instituto de Computação - UNICAMP - 2014

Sumário

Introdução

O que são expressões regulares?

- Uma expressão regular consiste em uma maneira de descrever um conjunto de strings com base em características comuns (padrões) a cada string do conjunto.
- As expressões regulares são utilizadas para procurar, editar e manipular textos e dados.
- Há uma sintaxe específica para criar expressões regulares e, apesar delas poderem variar muito em complexidade, os elementos que a compõem são simples.
- Java dispõe de APIs (pacote `java.util.regex`) para a construção e utilização de expressões regulares.

Introdução

Classes no pacote `java.util.regex`

- O pacote `java.util.regex` contém três classes: `Pattern`, `Matcher` e `PatternSyntaxException`.
  1. Classe `Pattern`: Um objeto `Pattern` representa uma versão compilada de uma expressão regular. Essa classe não possui construtores e para criar um objeto, é necessário chamar o método estático `compile`, que recebe uma expressão regular e retorna um objeto `Pattern`.
  2. Classe `Matcher`: Um objeto `Matcher` é um objeto que interpreta um padrão e compara com uma string. Assim como a classe `Pattern`, o construtor de `Matcher` é privado. Para obter um objeto `Matcher` é necessário chamar o método de instância `matcher` de um objeto `Pattern`.
  3. Classe `PatternSyntaxException`: trata-se de uma exceção não verificada que indica um erro de sintaxe em uma expressão regular.

Código para teste de expressões regulares

```
import java.util.Scanner;
import java.util.regex.*;
// RegexTestClass.java
// A classe RegexTestClass testa expressões regulares em strings.
public class RegexTestClass {
    static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {

        while (true) {
            try {
                // recebe uma expressão regular
                System.out.print("\nEntre com a expressão regular: ");
                Pattern pattern = Pattern.compile(input.nextLine());

                // recebe uma string onde será realizada a busca
                System.out.print("Entre com a string onde será feita a busca: ");
                Matcher matcher = pattern.matcher(input.nextLine());

                boolean found = false;
                // busca enquanto a string da entrada não for esgotada
                while (matcher.find()) {

                    // encontrou uma substring correspondente
                    found = true;
                    System.out.printf("Encontrei o texto"
                            + " \"%s\" começando no índice "
                            + "%d e terminando no índice %d.%n",
                            matcher.group(), matcher.start(), matcher.end());

    /* continua na próxima página */
```

Código para teste de expressões regulares

```
    /* continua da página anterior */

                    // para cada grupo de captura encontrado
                    for (int i = 1; i <= matcher.groupCount(); i++) {
                        // se o grupo não estiver vazio
                        if (matcher.group(i) != null) {
                            System.out.printf("Grupo de captura " + i + ": "
                                    + "Encontrei o texto"
                                    + " \"%s\" começando no índice "
                                    + "%d e terminando no índice %d.%n",
                                    matcher.group(i), matcher.start(i),
                                    matcher.end(i));
                        } // fim if
                    } // fim for
                } // fim while
                // se não encontrou nenhuma substring correspondente
                if (!found) {
                    System.out.printf("Nenhuma correspondência encontrada.%n");
                }
                // expressão regular inválida
            } catch (PatternSyntaxException patternSyntaxException) {
                System.out
                        .println("Entrada inválida, por favor tente novamente.");
            } // fim try
        } // fim while
    } // fim main
} // fim classe RegexTestClass
```

Introdução

Exemplo de execução

- Considere o exemplo onde a expressão regular é `foo` e a string é `foo`.

```
Entre com a expressão regular: foo
Entre com a string onde será feita a busca: foo
Encontrei o texto "foo" começando no índice 0 e terminando no índice 3.
```

- Nesse caso, a busca será bem sucedida pois as duas strings são idênticas.
- Por convenção, o intervalo de uma string é inclusivo no índice inicial e exclusivo no índice final. Por esse motivo, é possível que ocorram sobreposição de índices em buscas consecutivas, como no exemplo a seguir:

```
Entre com a expressão regular: foo
Entre com a string onde será feita a busca: foofoofoo
Encontrei o texto "foo" começando no índice 0 e terminando no índice 3.
Encontrei o texto "foo" começando no índice 3 e terminando no índice 6.
Encontrei o texto "foo" começando no índice 6 e terminando no índice 9.
```

Metacaracteres

Caracteres especiais

- A API Java considera um conjunto de caracteres especiais, denominados metacaracteres, que afetam a maneira como é feita a busca de um padrão.
- Exemplo: Supondo como expressão regular `cat.` (com ponto final) e a string de entrada é `cats`.

```
Entre com a expressão regular: cat.
Entre com a string onde será feita a busca: cats
Encontrei o texto "cats" começando no índice 0 e terminando no índice 4.
```

- Nesse caso, a busca é bem sucedida porque o metacaractere `.` representa qualquer caractere.

Metacaracteres

Caracteres especiais

- Os metacaracteres da API Java são: `<([{\^-=$!|]})?*+.>`
- É possível forçar um metacaractere ser tratado como um caractere normal delimitando-o pelo par `\Q` e `\E`.

```
Entre com a expressão regular: cat\Q.\E
Entre com a string onde será feita a busca: cats
Nenhuma correspondência encontrada.

Entre com a expressão regular: cat\Q.\E
Entre com a string onde será feita a busca: cat.
Encontrei o texto "cat." começando no índice 0 e terminando no índice 4.
```

Classes de caracteres

Especificando um conjunto de caracteres

- Diferente do conceito de classe em orientação a objetos, uma classe de caracteres consiste em um conjunto de caracteres delimitado por colchetes.
- Uma classe de caracteres especifica quais caracteres fazem correspondência com um único caractere de uma string.

Construção	Descrição
`[abc]`	a, b ou c (forma simples)
`[^abc]`	qualquer caractere exceto a, b, ou c (negação)
`[a-zA-Z]`	a até z, ou A até Z, inclusivo (intervalo)
`[a-d[m-p]]`	a até d, ou m até p: [a-dm-p] (união)
`[a-z&&[def]]`	d, e, ou f (interseção)
`[a-z&&[^bc]]`	a até z, exceto b e c: [ad-z] (subtração)
`[a-z&&[^m-p]]`	a até z, e não m até p: [a-lq-z] (subtração)

Classes de caracteres

Formas simples

- Uma forma simples de classe de caracteres consiste em posicionar um conjunto de caracteres lado a lado delimitados por colchetes.
- Por exemplo, a construção `[bcr]at` tem correspondência com as palavras `bat`, `cat` e `rat`, porque a classe de caracteres `[bcr]` aceita qualquer das letras (`b`, `c` ou `r`) como o primeiro caractere.

```
Entre com a expressão regular: [bcr]at
Entre com a string onde será feita a busca: bat
Encontrei o texto "bat" começando no índice 0 e terminando no índice 3.

Entre com a expressão regular: [bcr]at
Entre com a string onde será feita a busca: cat
Encontrei o texto "cat" começando no índice 0 e terminando no índice 3.

Entre com a expressão regular: [bcr]at
Entre com a string onde será feita a busca: rat
Encontrei o texto "rat" começando no índice 0 e terminando no índice 3.

Entre com a expressão regular: [bcr]at
Entre com a string onde será feita a busca: hat
Nenhuma correspondência encontrada.
```

Classes de caracteres

Negações

- Para corresponder com todos os caracteres com exceção de um conjunto de caracteres especificados, basta utilizar o metacaractere circunflexo ^ no início da classe de caracteres.
- Essa técnica é denominada negação.

```
Entre com a expressão regular: [^bcr]at
Entre com a string onde será feita a busca: bat
Nenhuma correspondência encontrada.

Entre com a expressão regular: [^bcr]at
Entre com a string onde será feita a busca: cat
Nenhuma correspondência encontrada.

Entre com a expressão regular: [^bcr]at
Entre com a string onde será feita a busca: rat
Nenhuma correspondência encontrada.

Entre com a expressão regular: [^bcr]at
Entre com a string onde será feita a busca: hat
Encontrei o texto "hat" começando no índice 0 e terminando no índice 3.
```

Classes de caracteres

Intervalos

- Para definir uma classe que inclui um intervalo de caracteres, como as letras de a até h ou os números de 1 a 5, basta inserir o metacaractere hífen `-` entre o primeiro e o último caractere do intervalo.
- Também é possível adicionar múltiplos intervalos na mesma classe de caracteres, um ao lado do outro, expandindo as correspondências possíveis.
- Exemplo: a construção `[a-zA-Z]` irá corresponder a qualquer letra do alfabeto, seja ela maiúscula ou minúscula.

Classes de caracteres

Intervalos

```
Entre com a expressão regular: [a—c]
Entre com a string onde será feita a busca: a
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [a—c]
Entre com a string onde será feita a busca: b
Encontrei o texto "b" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [a—c]
Entre com a string onde será feita a busca: c
Encontrei o texto "c" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [a—c]
Entre com a string onde será feita a busca: d
Nenhuma correspondência encontrada.

Entre com a expressão regular: [^a—c]
Entre com a string onde será feita a busca: a
Nenhuma correspondência encontrada.

Entre com a expressão regular: [^a—c]
Entre com a string onde será feita a busca: d
Encontrei o texto "d" começando no índice 0 e terminando no índice 1.
```

Classes de caracteres

Intervalos

```
Entre com a expressão regular: foo[1-5]
Entre com a string onde será feita a busca: foo1
Encontrei o texto "foo1" começando no índice 0 e terminando no índice 4.

Entre com a expressão regular: foo[1-5]
Entre com a string onde será feita a busca: foo5
Encontrei o texto "foo5" começando no índice 0 e terminando no índice 4.

Entre com a expressão regular: foo[1-5]
Entre com a string onde será feita a busca: foo6
Nenhuma correspondência encontrada.

Entre com a expressão regular: foo[^1-5]
Entre com a string onde será feita a busca: foo1
Nenhuma correspondência encontrada.

Entre com a expressão regular: foo[^1-5]
Entre com a string onde será feita a busca: foo6
Encontrei o texto "foo6" começando no índice 0 e terminando no índice 4.
```

Classes de caracteres

Uniões

- É possível definir uma classe de caracteres como a união de duas ou mais classes de caracteres.
- Para criar uma união, basta aninhar uma classe de caracteres dentro da outra.
- Por exemplo, a construção `[0-4[6-8]]` faz correspondência com os números `0, 1, 2, 3, 4, 6, 7, 8`. Uma construção equivalente a essa é dada por `[0-46-8]`.

```
Entre com a expressão regular: [0—4[6—8]]
Entre com a string onde será feita a busca: 2
Encontrei o texto "2" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [0—4[6—8]]
Entre com a string onde será feita a busca: 5
Nenhuma correspondência encontrada.

Entre com a expressão regular: [0—4[6—8]]
Entre com a string onde será feita a busca: 8
Encontrei o texto "8" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [0—4[6—8]]
Entre com a string onde será feita a busca: 9
Nenhuma correspondência encontrada.
```

Classes de caracteres

Interseções

- Para criar uma classe de caracteres que faz correspondência somente com os caracteres comuns a todas as classes aninhadas, basta utilizar o metacaractere `&&`.
- Por exemplo, a construção `[0-5&&[357]]` faz correspondência com os números `3` e `5`.

```
Entre com a expressão regular: [0-5&&[357]]
Entre com a string onde será feita a busca: 2
Nenhuma correspondência encontrada.

Entre com a expressão regular: [0-5&&[357]]
Entre com a string onde será feita a busca: 3
Encontrei o texto "3" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [0-5&&[357]]
Entre com a string onde será feita a busca: 4
Nenhuma correspondência encontrada.

Entre com a expressão regular: [0-5&&[357]]
Entre com a string onde será feita a busca: 5
Encontrei o texto "5" começando no índice 0 e terminando no índice 1.
```

Classes de caracteres

Interseções

- Outro exemplo consiste na interseção de intervalos, onde a construção `[3-6&&[1-4]]` faz correspondência com os números `3` e `4`.

```
Entre com a expressão regular: [3-6&&[1-4]]
Entre com a string onde será feita a busca: 2
Nenhuma correspondência encontrada.

Entre com a expressão regular: [3-6&&[1-4]]
Entre com a string onde será feita a busca: 3
Encontrei o texto "3" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [3-6&&[1-4]]
Entre com a string onde será feita a busca: 4
Encontrei o texto "4" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [3-6&&[1-4]]
Entre com a string onde será feita a busca: 5
Nenhuma correspondência encontrada.
```

Classes de caracteres

Subtração

- É possível utilizar a técnica de subtração para construir classes de caracteres das quais são excluídas (negadas) uma ou mais classes aninhadas.
- Uma subtração é feita utilizando os metacaracteres de interseção `&&` e negação `^`.
- Por exemplo, a construção `[0-9&&[^357]]` faz correspondência com os números `0, 1, 2, 4, 6, 8, 9`.

```
Entre com a expressão regular: [0-9&&[^357]]
Entre com a string onde será feita a busca: 2
Encontrei o texto "2" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [0-9&&[^357]]
Entre com a string onde será feita a busca: 3
Nenhuma correspondência encontrada.

Entre com a expressão regular: [0-9&&[^357]]
Entre com a string onde será feita a busca: 4
Encontrei o texto "4" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: [0-9&&[^357]]
Entre com a string onde será feita a busca: 5
Nenhuma correspondência encontrada.
```

Classes de caracteres pré-definidas

Simplificando as construções

- A classe `Pattern` contém um conjunto de construções pré-definidas que representam classes de caracteres mais usuais, no intuito de simplificar algumas expressões regulares.

Construção	Descrição
`.`	Qualquer caractere
`\d`	Um dígito: `[0-9]`
`\D`	Qualquer caractere exceto dígitos: `[^0-9]`
`\s`	Qualquer caractere branco (inclui espaço): `[ \t\n\x0B\f\r]`
`\S`	Qualquer caractere exceto caracteres brancos: `[^\s]`
`\w`	Qualquer caractere de palavra: `[a-zA-Z_0-9]`
`\W`	Qualquer caractere exceto caracteres de palavra: `[^\w]`

Classes de caracteres pré-definidas

Simplificando as construções

- As construções pré-definidas tornam mais fácil a leitura de um padrão e reduzem erros por construções mal-formadas de classes de caracteres.
- Em uma string Java, quando se utiliza a barra invertida \ na construção de uma expressão regular, torna-se necessário incluir uma barra invertida adicional para que o padrão compile. Exemplo:

```
String  myRegEx = "\\d"; //  um dígito
```

Classes de caracteres pré-definidas

Simplificando as construções

```
Entre com a expressão regular: .
Entre com a string onde será feita a busca: @
Encontrei o texto "@" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: .
Entre com a string onde será feita a busca: .
Encontrei o texto "." começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: .
Entre com a string onde será feita a busca: 1
Encontrei o texto "1" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: .
Entre com a string onde será feita a busca: a
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: \d
Entre com a string onde será feita a busca: 1
Encontrei o texto "1" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: \d
Entre com a string onde será feita a busca: a
Nenhuma correspondência encontrada.
```

Classes de caracteres pré-definidas

Simplificando as construções

```
Entre com a expressão regular: \D
Entre com a string onde será feita a busca: 1
Nenhuma correspondência encontrada.

Entre com a expressão regular: \D
Entre com a string onde será feita a busca: a
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: \s
Entre com a string onde será feita a busca:
Encontrei o texto " " começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: \s
Entre com a string onde será feita a busca: a
Nenhuma correspondência encontrada.

Entre com a expressão regular: \S
Entre com a string onde será feita a busca:
Nenhuma correspondência encontrada.

Entre com a expressão regular: \S
Entre com a string onde será feita a busca: a
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.
```

Classes de caracteres pré-definidas

Simplificando as construções

```
Entre com a expressão regular: \w
Entre com a string onde será feita a busca: 1
Encontrei o texto "1" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: \w
Entre com a string onde será feita a busca: a
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: \w
Entre com a string onde será feita a busca: @
Nenhuma correspondência encontrada.

Entre com a expressão regular: \W
Entre com a string onde será feita a busca: 1
Nenhuma correspondência encontrada.

Entre com a expressão regular: \W
Entre com a string onde será feita a busca: a
Nenhuma correspondência encontrada.

Entre com a expressão regular: \W
Entre com a string onde será feita a busca: @
Encontrei o texto "@" começando no índice 0 e terminando no índice 1.
```

Grupos de captura

Tratando múltiplos caracteres

- Um grupo de captura (*capturing group*) consiste em uma maneira de tratar múltiplos caracteres como uma única unidade.
- Um grupo de caracteres é formado por caracteres delimitados por parênteses.
- Exemplo: a expressão regular `(dog)` consiste em um grupo de captura formado pelas letras `d`, `o` e `g`.
- Para realizar correspondência com um grupo de captura, uma string precisa conter todos os caracteres do grupo e na mesma ordem.

Grupos de captura

Exemplos

```
Entre com a expressão regular: (abc)
Entre com a string onde será feita a busca: @abc1
Encontrei o texto "abc" começando no índice 1 e terminando no índice 4.
Grupo de captura 1: Encontrei o texto "abc" começando no índice 1 e terminando no índice 4.

Entre com a expressão regular: (abc)
Entre com a string onde será feita a busca: cba
Nenhuma correspondência encontrada.

Entre com a expressão regular: (\d\w\W)
Entre com a string onde será feita a busca: 1a@
Encontrei o texto "1a@" começando no índice 0 e terminando no índice 3.
Grupo de captura 1: Encontrei o texto "1a@" começando no índice 0 e terminando no índice 3.

Entre com a expressão regular: (\d\w\W)
Entre com a string onde será feita a busca: @1a
Nenhuma correspondência encontrada.
```

Grupos de captura

Tratando múltiplos caracteres

- Uma expressão regular pode conter múltiplos grupos de captura (aninhados ou lado a lado), por exemplo, `my((d)(o(g)))`.
- Os grupos de captura de uma expressão regular são numerados na ordem em que parênteses esquerdos aparecem.
- Exemplo: a expressão regular `my((d)(o(g)))` contém quatro grupos de captura, cuja numeração é dada por:
  1. `((d)(o(g)))`
  2. `(d)`
  3. `(o(g))`
  4. `(g)`
- Há também um grupo especial com numeração 0 que representa a expressão regular completa `my((d)(o(g)))`.
- A ordem em que os grupos são numerados é relevante pois um grupo em particular pode ser recuperado conhecendo seu número.

Grupos de captura

Exemplos

```
Entre com a expressão regular: my((d)(o(g)))
Entre com a string onde será feita a busca: mydog
Encontrei o texto "mydog" começando no índice 0 e terminando no índice 5.
Grupo de captura 1: Encontrei o texto "dog" começando no índice 2 e terminando no índice 5.
Grupo de captura 2: Encontrei o texto "d" começando no índice 2 e terminando no índice 3.
Grupo de captura 3: Encontrei o texto "og" começando no índice 3 e terminando no índice 5.
Grupo de captura 4: Encontrei o texto "g" começando no índice 4 e terminando no índice 5.

Entre com a expressão regular: my((d)(o(g)))
Entre com a string onde será feita a busca: dog
Nenhuma correspondência encontrada.
```

Grupos de captura

Retroreferências

- Suponha que uma string faça correspondência com uma expressão regular que contém um grupo de captura. O literal que faz correspondência com o grupo de captura é salvo em memória para uso futuro em um processo denominado retroreferência (*backreference*).
- Uma retroreferência pode fazer parte de uma expressão regular através do metacaractere barra invertida `\` seguido do dígito correspondente ao número do grupo que se deseja recuperar.
- Exemplo: `(abc)\1` faz correspondência com o literal `abc` seguido pelo mesmo literal `abc`, ou seja, `abcabc`.

Grupos de captura

Exemplos

```
Entre com a expressão regular: (abc)\1
Entre com a string onde será feita a busca: abcabc
Encontrei o texto "abcabc" começando no índice 0 e terminando no índice 6.
Grupo de captura 1: Encontrei o texto "abc" começando no índice 0 e terminando no índice 3.

Entre com a expressão regular: (abc)\1
Entre com a string onde será feita a busca: abc
Nenhuma correspondência encontrada.

Entre com a expressão regular: (ab)\1(cd)\2
Entre com a string onde será feita a busca: ababcdcd
Encontrei o texto "ababcdcd" começando no índice 0 e terminando no índice 8.
Grupo de captura 1: Encontrei o texto "ab" começando no índice 0 e terminando no índice 2.
Grupo de captura 2: Encontrei o texto "cd" começando no índice 4 e terminando no índice 6.

Entre com a expressão regular: (ab)(cd)\2\1
Entre com a string onde será feita a busca: abcdcdab
Encontrei o texto "abcdcdab" começando no índice 0 e terminando no índice 8.
Grupo de captura 1: Encontrei o texto "ab" começando no índice 0 e terminando no índice 2.
Grupo de captura 2: Encontrei o texto "cd" começando no índice 2 e terminando no índice 4.
```

Grupos de captura

Exemplos

Entre com a expressão regular: (\d\w)\1
Entre com a string onde será feita a busca: 1a1a
Encontrei o texto "1a1a" começando no índice 0 e terminando no índice 4.
Grupo de captura 1: Encontrei o texto "1a" começando no índice 0 e terminando no índice 2.

Entre com a expressão regular: (\d\w)\1
Entre com a string onde será feita a busca: 1a2b
Nenhuma correspondência encontrada.

Entre com a expressão regular: (\d\w)(\d\w)\2\1
Entre com a string onde será feita a busca: 1a2b2b1a
Encontrei o texto "1a2b2b1a" começando no índice 0 e terminando no índice 8.
Grupo de captura 1: Encontrei o texto "1a" começando no índice 0 e terminando no índice 2.
Grupo de captura 2: Encontrei o texto "2b" começando no índice 2 e terminando no índice 4.

Entre com a expressão regular: (\d\w)(\d\w)\2\1
Entre com a string onde será feita a busca: 1a2b1a2b
Nenhuma correspondência encontrada.

Quantificadores

Número de ocorrências de um padrão

- Quantificadores especificam o número de ocorrências de um certo padrão ao realizar uma busca em uma string.
- Os metacaracteres utilizados para expressar quantificadores são `?`, `*`, `+` e `{}` e seus efeitos encontram-se descritos na tabela a seguir.

Construção	Descrição
`X?`	`X`, zero ou uma vez
`X*`	`X`, zero ou mais vezes
`X+`	`X`, uma ou mais vezes
`X{n}`	`X`, exatas n vezes
`X{n,}`	`X`, pelo menos n vezes
`X{n,m}`	`X`, pelo menos n e no máximo m vezes

Quantificadores

Exemplos

```
Entre com a expressão regular: dogs?cats?
Entre com a string onde será feita a busca: dogscats
Encontrei o texto "dogscats" começando no índice 0 e terminando no índice 8.

Entre com a expressão regular: dogs?cats?
Entre com a string onde será feita a busca: dogcat
Encontrei o texto "dogcat" começando no índice 0 e terminando no índice 6.

Entre com a expressão regular: dogs?cats?
Entre com a string onde será feita a busca: dogssscatsss
Nenhuma correspondência encontrada.

Entre com a expressão regular: dogs*cats*
Entre com a string onde será feita a busca: dogcat
Encontrei o texto "dogcat" começando no índice 0 e terminando no índice 6.

Entre com a expressão regular: dogs*cats*
Entre com a string onde será feita a busca: dogssscatsss
Encontrei o texto "dogssscatsss" começando no índice 0 e terminando no índice 12.

Entre com a expressão regular: dogs+cats+
Entre com a string onde será feita a busca: dogcat
Nenhuma correspondência encontrada.

Entre com a expressão regular: dogs+cats+
Entre com a string onde será feita a busca: dogssscatsss
Encontrei o texto "dogssscatsss" começando no índice 0 e terminando no índice 12.
```

Quantificadores

Correspondências de comprimento zero

- Observe os seguintes exemplos do uso de quantificadores.

```
Entre com a expressão regular: a?
Entre com a string onde será feita a busca:
Encontrei o texto "" começando no índice 0 e terminando no índice 0.

Entre com a expressão regular: a*
Entre com a string onde será feita a busca:
Encontrei o texto "" começando no índice 0 e terminando no índice 0.

Entre com a expressão regular: a+
Entre com a string onde será feita a busca:
Nenhuma correspondência encontrada.
```

- As buscas nos primeiros dois casos acima tiveram sucesso porque as construções `a?` e `a*` permitem zero ocorrências da letra `a`.
- É possível observar que nos dois primeiros casos a busca retornou uma string vazia, o que é denominado como correspondência de comprimento zero (*zero-length match*).

Quantificadores

Correspondências de comprimento zero

```
Entre com a expressão regular: a?
Entre com a string onde será feita a busca: a
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.
Encontrei o texto "" começando no índice 1 e terminando no índice 1.

Entre com a expressão regular: a*
Entre com a string onde será feita a busca: a
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.
Encontrei o texto "" começando no índice 1 e terminando no índice 1.

Entre com a expressão regular: a+
Entre com a string onde será feita a busca: a
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.

Entre com a expressão regular: a?
Entre com a string onde será feita a busca: aaa
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.
Encontrei o texto "a" começando no índice 1 e terminando no índice 2.
Encontrei o texto "a" começando no índice 2 e terminando no índice 3.
Encontrei o texto "" começando no índice 3 e terminando no índice 3.

Entre com a expressão regular: a*
Entre com a string onde será feita a busca: aaa
Encontrei o texto "aaa" começando no índice 0 e terminando no índice 3.
Encontrei o texto "" começando no índice 3 e terminando no índice 3.

Entre com a expressão regular: a+
Entre com a string onde será feita a busca: aaa
Encontrei o texto "aaa" começando no índice 0 e terminando no índice 3.
```

Quantificadores

Correspondências de comprimento zero

Nos exemplos a seguir, pelo fato das expressões regulares `a?` e `a*` fazerem correspondência com zero vezes a ocorrência da letra `a`, qualquer caractere diferente de `a` vai retornar uma correspondência de comprimento zero.

```
Entre com a expressão regular: a?
Entre com a string onde será feita a busca: abaabba
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.
Encontrei o texto "" começando no índice 1 e terminando no índice 1.
Encontrei o texto "a" começando no índice 2 e terminando no índice 3.
Encontrei o texto "a" começando no índice 3 e terminando no índice 4.
Encontrei o texto "" começando no índice 4 e terminando no índice 4.
Encontrei o texto "" começando no índice 5 e terminando no índice 5.
Encontrei o texto "a" começando no índice 6 e terminando no índice 7.
Encontrei o texto "" começando no índice 7 e terminando no índice 7.

Entre com a expressão regular: a*
Entre com a string onde será feita a busca: abaabba
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.
Encontrei o texto "" começando no índice 1 e terminando no índice 1.
Encontrei o texto "aa" começando no índice 2 e terminando no índice 4.
Encontrei o texto "" começando no índice 4 e terminando no índice 4.
Encontrei o texto "" começando no índice 5 e terminando no índice 5.
Encontrei o texto "a" começando no índice 6 e terminando no índice 7.
Encontrei o texto "" começando no índice 7 e terminando no índice 7.

Entre com a expressão regular: a+
Entre com a string onde será feita a busca: abaabba
Encontrei o texto "a" começando no índice 0 e terminando no índice 1.
Encontrei o texto "aa" começando no índice 2 e terminando no índice 4.
Encontrei o texto "a" começando no índice 6 e terminando no índice 7.
```

Quantificadores

Padrões com um número especificado de ocorrências

- Para definir um número fixo *n* de ocorrências de um padrão `X` qualquer, basta delimitar *n* por chaves logo após o padrão desejado `X{n}`.
- O construtor `X{n,}` define um intervalo aberto de *n* ou mais ocorrências de um padrão `X` qualquer.
- O construtor `X{n,m}` define um intervalo fechado de *n* até *m* ocorrências de um padrão `X` qualquer.

Quantificadores

Padrões com um número especificado de ocorrências

```
Entre com a expressão regular: a{3}
Entre com a string onde será feita a busca: aa
Nenhuma correspondência encontrada.

Entre com a expressão regular: a{3}
Entre com a string onde será feita a busca: aaaa
Encontrei o texto "aaa" começando no índice 0 e terminando no índice 3.

Entre com a expressão regular: a{3}
Entre com a string onde será feita a busca: aaaaaaaa
Encontrei o texto "aaa" começando no índice 0 e terminando no índice 3.
Encontrei o texto "aaa" começando no índice 3 e terminando no índice 6.

Entre com a expressão regular: a{3,}
Entre com a string onde será feita a busca: aaaaaaaa
Encontrei o texto "aaaaaaaa" começando no índice 0 e terminando no índice 8.

Entre com a expressão regular: a{3,5}
Entre com a string onde será feita a busca: aaaaaaaa
Encontrei o texto "aaaaa" começando no índice 0 e terminando no índice 5.
Encontrei o texto "aaa" começando no índice 5 e terminando no índice 8.
```

Quantificadores

Quantificadores com classes de caracteres e grupos de captura

- Um quantificador, quando anexado a um caractere, tem seu efeito restrito somente a esse caractere.
- Exemplo: a expressão regular `abc+` faz correspondência com as strings `abc`, `abcc`, `abccc`, e assim sucessivamente. Essa expressão não faz correspondência com `abcabc`, como poderia-se imaginar.
- Quando se quer que o efeito do quantificador se estenda para um conjunto de caracteres, é possível anexá-lo a uma classe de caractere ou grupo de captura.
- Exemplo: a expressão regular `[abc]+` faz correspondência com strings de qualquer tamanho formadas com as letras `a`, `b` e `c`.
- Exemplo: a expressão regular `(abc)+` faz correspondência com as strings `abc`, `abcabc`, `abcabcabc`, e assim sucessivamente.

Quantificadores

Quantificadores com classes de caracteres e grupos de captura

```
Entre com a expressão regular: [ab]{3}
Entre com a string onde será feita a busca: abcaabbccaaabbbccc
Encontrei o texto "aab" começando no índice 3 e terminando no índice 6.
Encontrei o texto "aaa" começando no índice 9 e terminando no índice 12.
Encontrei o texto "bbb" começando no índice 12 e terminando no índice 15.

Entre com a expressão regular: [ab]{3,}
Entre com a string onde será feita a busca: abcaabbccaaabbbccc
Encontrei o texto "aabb" começando no índice 3 e terminando no índice 7.
Encontrei o texto "aaabbb" começando no índice 9 e terminando no índice 15.

Entre com a expressão regular: [ab]{3,5}
Entre com a string onde será feita a busca: abcaabbccaaabbbccc
Encontrei o texto "aabb" começando no índice 3 e terminando no índice 7.
Encontrei o texto "aaabb" começando no índice 9 e terminando no índice 14.
```

Quantificadores

Quantificadores com classes de caracteres e grupos de captura

```
Entre com a expressão regular: (dog){3}
Entre com a string onde será feita a busca: dogdogdogdog
Encontrei o texto "dogdogdog" começando no índice 0 e terminando no índice 9.
Grupo de captura 1: Encontrei o texto "dog" começando no índice 6 e terminando no índice 9.

Entre com a expressão regular: dog{3}
Entre com a string onde será feita a busca: dogdogdogdog
Nenhuma correspondência encontrada.

Entre com a expressão regular: [abc]{3}
Entre com a string onde será feita a busca: abccabaaaccbbbc
Encontrei o texto "abc" começando no índice 0 e terminando no índice 3.
Encontrei o texto "cab" começando no índice 3 e terminando no índice 6.
Encontrei o texto "aaa" começando no índice 6 e terminando no índice 9.
Encontrei o texto "ccb" começando no índice 9 e terminando no índice 12.
Encontrei o texto "bbc" começando no índice 12 e terminando no índice 15.

Entre com a expressão regular: abc{3}
Entre com a string onde será feita a busca: abccabaaaccbbbc
Nenhuma correspondência encontrada.
```

Demarcadores de fronteiras

Especificando locais em uma string

- Uma expressão regular busca por uma correspondência em qualquer posição de uma string, exceto quando uma localização é especificada.
- Para atribuir uma localização específica, existem os demarcadores de fronteiras (*boundary matchers*), descritos na tabela a seguir.

Construção	Descrição
`^`	Início de linha.
`$`	Fim de linha.
`\b`	Fronteira de uma palavra.
`\B`	Não-fronteira de uma palavra.
`\A`	Início da string de entrada.
`\Z`	Final da string de entrada.

Demarcadores de fronteiras

Especificando locais em uma string

```
Entre com a expressão regular: ^dog$
Entre com a string onde será feita a busca: dog
Encontrei o texto "dog" começando no índice 0 e terminando no índice 3.

Entre com a expressão regular: ^dog$
Entre com a string onde será feita a busca:          dog
Nenhuma correspondência encontrada.

Entre com a expressão regular: \s*dog$
Entre com a string onde será feita a busca:              dog
Encontrei o texto "            dog" começando no índice 0 e terminando no índice 12.

Entre com a expressão regular: ^dog\w*
Entre com a string onde será feita a busca: dogblahblah
Encontrei o texto "dogblahblah" começando no índice 0 e terminando no índice 11.
```

Demarcadores de fronteiras

Especificando locais em uma string

- O demarcador `\b` faz uma correspondência de comprimento zero com uma posição que se encontra entre um caractere de palavra `\w` e um não caractere de palavra `\W`.
- Esse demarcador é útil quando se deseja verificar se uma string é uma substring, ou seja, faz parte de uma string maior.

```
Entre com a expressão regular: \bdog\b
Entre com a string onde será feita a busca: The dog plays in the yard.
Encontrei o texto "dog" começando no índice 4 e terminando no índice 7.

Entre com a expressão regular: \bdog\b
Entre com a string onde será feita a busca: The doggie plays in the yard.
Nenhuma correspondência encontrada.

Entre com a expressão regular: \bdog\B
Entre com a string onde será feita a busca: The dog plays in the yard.
Nenhuma correspondência encontrada.

Entre com a expressão regular: \bdog\B
Entre com a string onde será feita a busca: The doggie plays in the yard.
Encontrei o texto "dog" começando no índice 4 e terminando no índice 7.
```

Operador alternativo

Metacaractere `|`

- O metacaractere barra vertical `|` representa um operador alternativo (“ou lógico”), aplicado quando se deseja fazer a correspondência de uma expressão regular dentre um conjunto de expressões regulares.
- Exemplo: Se desejamos encontrar um literal `red` ou `blue` em uma string, a expressão regular `red|blue` seria suficiente. Se mais opções são desejadas, basta expandir a lista: `red|blue|orange|yellow`.
- Dentre os operadores de expressão regular, o operador alternativo é o de mais baixa precedência. Ou seja, se nenhum delimitador (parênteses ou colchetes) for utilizado, a barra vertical implica em fazer uma correspondência com tudo que está a sua esquerda ou com tudo que está a sua direita.

Operador alternativo

Metacaractere |

```
Entre com a expressão regular: \b(cat|dog)\b
Entre com a string onde será feita a busca: The cat and the dog are playing.
Encontrei o texto "cat" começando no índice 4 e terminando no índice 7.
Grupo de captura 1: Encontrei o texto "cat" começando no índice 4 e terminando no índice 7.
Encontrei o texto "dog" começando no índice 16 e terminando no índice 19.
Grupo de captura 1: Encontrei o texto "dog" começando no índice 16 e terminando no índice 19.

Entre com a expressão regular: \b(cat|dog)\b
Entre com a string onde será feita a busca: The catand thedog are playing.
Nenhuma correspondência encontrada.

Entre com a expressão regular: \bcat|dog\b
Entre com a string onde será feita a busca: The catand thedog are playing.
Encontrei o texto "cat" começando no índice 4 e terminando no índice 7.
Encontrei o texto "dog" começando no índice 14 e terminando no índice 17.
```

Métodos da API `java.util.regex`

Classe `Pattern`

Alguns métodos da classe `Pattern`:

- `public static Pattern compile(String regex)`: Compila uma expressão regular em um autômato finito. Esse método é vantajoso quando o padrão é utilizado muitas vezes.
- `static boolean matches(String regex, CharSequence input)`: Verifica diretamente (sem criar um objeto `Pattern`) se há correspondência entre uma expressão regular com uma cadeira de caracteres. Esse método é vantajoso quando a expressão regular vai ser utilizada uma única vez.
- `String[] split(CharSequence input)`: Divide uma string utilizando como critério de separação uma expressão regular.

Métodos da API `java.util.regex`

Classe `Pattern`

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

// SplitClass.java
// Classe exemplifica o uso do método split da classe Pattern.
public class SplitTestClass {

    private static final String REGEX = "\\d";
    private static final String INPUT = "one9two4three7four1five";

    public static void main(String[] args) {
        // compila a expressão regular
        Pattern p = Pattern.compile(REGEX);
        // divide a expressão regular utilizando como critério de separação caracteres numéricos
        String [] items = p. split (INPUT);
        // imprime as substrings resultantes
        for (String s : items) {
            System.out.println(s);
        } // fim for
    } // fim main
} // fim classe SplitClass
```

Métodos da API `java.util.regex`

Classe `Pattern`

```
one
two
three
four
five
```

Métodos da API `java.util.regex`

Classe `Matcher`

Métodos da classe `Matcher` que retornam índices diversos da string onde foi realizada a busca:

- `public int start()`: Retorna o índice inicial da última correspondência na string.
- `public int start(int i)`: Retorna o índice inicial da string capturada pelo *i*-ésimo grupo de captura, referente à última correspondência na string.
- `public int end()`: Retorna o índice final $+1$ da última correspondência na string.
- `public int end(int i)`: Retorna o índice final $+1$ da substring capturada pelo *i*-ésimo grupo de captura, referente à última correspondência na string.

Métodos da API `java.util.regex`

Classe `Matcher`

Métodos de busca da classe `Matcher`:

- `public boolean find()`: Procura pela próxima correspondência da expressão regular com a string.
- `public boolean find(int start)`: Procura pela próxima correspondência da expressão regular com a string, partindo da *i*-ésima posição da string.
- `public boolean lookingAt()`: Tenta realizar uma correspondência com uma substring contida no início da string original.
- `public boolean matches()`: Verifica se a expressão regular faz correspondência com a string completa (e não somente uma substring).

Métodos da API `java.util.regex`

Classe `Matcher`

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

// MatchesLooking.java
// A classe MatcherTestClass testa os métodos lookingAt() e matches() da classe Matcher
public class MatcherTestClass {

    private static final String REGEX = "foo";
    private static final String INPUT = "fooooooooo";

    private static Pattern pattern;
    private static Matcher matcher;

    public static void main(String[] args) {

        //  Initialize
        pattern = Pattern.compile(REGEX);
        matcher = pattern.matcher(INPUT);

        System.out.println("Current REGEX is: " + REGEX);
        System.out.println("Current INPUT is: " + INPUT);

        System.out.println("lookingAt(): " + matcher.lookingAt());
        System.out.println("matches(): " + matcher.matches());
    } // fim main
} // fim classe MatcherTestClass
```

Métodos da API `java.util.regex`

Classe `Matcher`

```
Current REGEX is: foo
Current INPUT is: fooooooooo
lookingAt() : true
matches(): false
```

Métodos da API `java.util.regex`

Classe `Matcher`

Métodos de reposição da classe `Matcher`:

- `public String replaceAll(String replacement)`: Toda correspondência da expressão regular com a string original é trocada pela string fornecida como argumento.
- `public String replaceFirst(String replacement)`: Somente a primeira correspondência (se houver) da expressão regular com a string original é trocada pela string fornecida como argumento.

Métodos da API `java.util.regex`

Classe `Pattern`

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

// ReplaceTestClass.java
// Classe ReplaceTestClass testa o método de reposição replaceAll() da classe Matcher
public class ReplaceTestClass {

    private static String REGEX = "a*b"; // expressão regular
    private static String INPUT = "aabfooaabfooabfoob"; // string de entrada
    private static String REPLACE = "-"; // string de reposição

    public static void main(String[] args) {

        Pattern p = Pattern.compile(REGEX);
        Matcher m = p.matcher(INPUT);
        INPUT = m.replaceAll(REPLACE);
        System.out.println(INPUT);
    } // fim main
} // fim classe ReplaceTestClass
```

Métodos da API `java.util.regex`

Classe `Pattern`

Referências

1. Java: Como Programar, Paul Deitel & Heivey Deitel; Pearson; 7a. Ed. (no. chamada IMECC – 05.133 D368j)
2. Data Structures and Algorithms with Object Oriented Design Patterns in Java, Bruno Preiss; (http://www.brpreiss.com/books/opus6/)
3. The Java Tutorials (Oracle) (http://docs.oracle.com/javase/tutorial/)
4. Guia do Usuário UML, Grady Booch et. al.; Campus(1999)
5. Java Pocket Guide - Robert Liguori & Patricia Liguori; O'Reilley, 2008.