Tutorial 5 – Parte A

Expressões regulares

Isabel Nunes e André Souto

Cadeira de
Laboratório de Programação

2020/2021

Expressões Regulares

As *expressões regulares*, ou *padrões*, são expressões que permitem representar sequências de símbolos com dadas características. Diz-se que um padrão *mapeia* (*match*) ou *aceita* um dado conjunto de cadeias de carateres. Diz-se que uma cadeia de carateres é *mapeada* ou *aceite* por um padrão.

No contexto das linguagens de programação, as expressões regulares são usadas para representar conjuntos de cadeias de caracteres.

As expressões regulares são uma ferramenta muito útil para extração de dados com determinadas características a partir de fontes não estruturadas de carateres.

Expressões regulares e o seu significado:

- Um qualquer caráter `c` é uma expressão regular que representa o conjunto com uma única cadeia {`c`};
- Um ou mais carateres entre parêntesis retos é uma expressão regular que representa o conjunto de cadeias de comprimento 1 mapeados pela expressão dentro dos parêntesis. Exemplos:
  - A expressão regular `[aeiou]` representa o conjunto {`a,e,i,o,u`};
  - A expressão regular `[a-r]` representa o conjunto das letras minúsculas do alfabeto entre `a` e `r` (ignorando os acentos);
  - A expressão regular `[0-4]` representa o conjunto {`0,1,2,3,4`};
- Uma expressão dentro de parêntesis retos composta pelo caráter `^` seguido de uma expressão regular `R` representa o conjunto de cadeias de comprimento 1 que <u>não</u> são mapeados pela expressão `R`. Exemplos:
  - A expressão regular `[^aeiou]` representa qualquer caráter que não seja uma vogal;
  - A expressão regular `[^0-9]` representa qualquer caráter que não seja um algarismo;
- Sejam `R` e `S` expressões regulares. As seguintes operações produzem expressões regulares:
  - *Alternativa*: A expressão regular `R|S` representa a união dos conjuntos de cadeias aceites por `R` e as aceites por `S`. Exemplos:
    - a expressão `d = 0|1|2|3|4|5|6|7|8|9` representa qualquer algarismo, isto é, corresponde ao conjunto de cadeias `{0,1,2,3,4,5,6,7,8,9}`;
    - Se `R` for `[a-d]` e `S` for `8`, então `R|S` é a expressão regular à qual corresponde o conjunto de cadeias `{a,b,c,d,8}`;

- *Concatenação*: A expressão regular `RS` representa o conjunto de cadeias que podem ser obtidas concatenando uma cadeia aceite por `R` com uma cadeia aceite por `S`. Exemplos:
  - A expressão `X[1-4]` representa o conjunto de cadeias `{X1,X2,X3,X4}`;
  - A expressão `a[1-9]b` representa o conjunto de cadeias `{a1b, a2b, a3b, a4b, a5b, a6b, a7b, a8b, a9b}`;
  - Se `R` for `a|b` e `S` for `c`, então `RS` é a expressão regular à qual corresponde o conjunto de cadeias `{ac, bc}`;
  - Se `R = 0|1|2|3|4|5|6|7|8|9` então `RRR` representa os números inteiros inferiores a 1000;
- *Agrupamento*: usam-se parêntesis para definir o âmbito e a precedência de operadores (ver mais adiante). Exemplos:
  - A expressão `estran(g|j)eiro` representa o conjunto de cadeias `{estrangeiro, estranjeiro}`;
- *Opção*: A expressão regular `R?` representa zero ou uma ocorrência de `R`, ou seja, `R` é opcional. Exemplos:
  - `ac?ção` representa o conjunto de cadeias `{acção, ação}`; é equivalente a `acção|ação`
  - `X(1|2|3)?` representa o conjunto de cadeias `{X,X1,X2,X3};`
  - `H(ä|ae?)ndel` representa o conjunto de cadeias `{Händel, Handel, Haendel};`
- *Repetição*: Seja `R` uma expressão regular. A aplicação dos seguintes operadores de *repetição* sobre `R` produzem expressões regulares:
  - ***** : a expressão regular `R*` representa zero ou mais ocorrências de `R`. A repetição **"**zero vezes**"** de uma cadeia de caracteres é a cadeia vazia. Exemplos:
    - `ab*c` representa o conjunto infinito de cadeias `{ac, abc, abbc, abbbc, abbbbc, ...}`;
  - **+** : a expressão regular `R+` representa uma ou mais ocorrências de `R`. Exemplos:
    - `ab+c` representa o conjunto infinito de cadeias `{abc, abbc, abbbc, abbbbc, ...}`;
    - `X[0-9]+` representa o conjunto de cadeias compostas pela letra `X` seguida de um ou mais algarismos;
    - `[0-9]+([.][0-9]*)?` representa um número possivelmente com parte decimal;
  - *{n} e {n1,n2}* : esta notação nem sempre está acessível; a expressão regular `R{n}` representa `n` ou mais ocorrências de `R`; a expressão regular `R{n1,n2}` representa entre `n1` e `n2` ocorrências de `R` Exemplos:

- `A{2}` representa o conjunto infinito de cadeias `{AA, AAA, AAAA, AAAAA, ...}`;
- `o{2,4}[1-3]` representa o conjunto de cadeias `{oo1, oo2, oo3, ooo1, ooo2, ooo3, oooo1, oooo2, oooo3}`;

Tal como nas expressões aritméticas, existem regras de precedência entre operações que compõem as expressões regulares (nomeadamente a concatenação, a alternativa e a repetição). O uso de parênteses nas expressões regulares permite contornar o problema das precedências:

- A repetição tem precedência em relação à concatenação. Por exemplo, `ab*` é equivalente à expressão: `a(b*)`
- A concatenação tem precedência em relação à alternativa. Por exemplo `ab|c` é equivalente à expressão: `(ab)|c`

A introdução de operadores levanta um problema: *"Como representar uma expressão regular que contém um dos caracteres* **+** *ou* ***"**? Consegue-se isso recorrendo a parêntesis retos. Por exemplo:

- a expressão regular `a[+]+b` corresponde às cadeias `a+b` `a++b` `a+++b` `a++++b` etc, ou seja, um `a`, seguido de pelo menos um `+` e por fim um `b`.

As expressões regulares constituem uma ferramenta muito útil e existem bibliotecas para usá-las na maioria das linguagens de programação. Todavia, podem existir variantes na notação usada.

De seguida vamos ver como usar expressões regulares em Java.

Expressões regulares em Java

Duas classes da API do java que são úteis para usar expressões regulares: `java.util.regex.Pattern` e `java.util.regex.Matcher`. Veja a documentação em http://docs.oracle.com/javase/10/docs/api/java/util/regex/Pattern.html e http://docs.oracle.com/javase/10/docs/api/java/util/regex/Matcher.html

A primeira serve para criar uma expressão regular:

```
Pattern pNumber = Pattern.compile("[0-9]+([.][0-9]*)?");
```

A partir deste momento pode-se criar um "matcher" que será capaz de reconhecer cadeias de caracteres desta expressão regular numa sequência. Por exemplo, podemos imaginar que um programa lê linha a linha um ficheiro e imprime no ecrã as linhas que contêm um número:

```
Matcher mNumber = pNumber.matcher(line);
if (mNumber.find()){
    System.out.println(line);
}
```

Caso o objetivo seja de copiar apenas os números para o ecrã, poderemos usar o método `group`:

```
if (mNumber.find()){
    System.out.println(mNumber.group());
}
```

Chamadas sucessivas do método `find` procuram as próximas ocorrências do padrão. O método `group` retorna a cadeia de caracteres que corresponde ao último padrão encontrado.

Após execução das seguintes linhas de código, as *strings* `X1` `X3` e `X2` seriam escritas no *standard output*:

```
Pattern myPat = Pattern.compile("X[1-4]");
Matcher myMatch = myPat.matcher("De X1 e X3 daria tX5 e tX2");
List<String> found = new LinkedList<>();
while (myMatch.find()) {
      found.add(myMatch.group());
}
for(String s : found) {
      System.out.println(s);
}
```

Para facilitar a escrita de expressões regulares existem expressões pré-definidas:

- **\t** o caráter "tab",
- **\n** o caráter "newline",
- **\r** o caráter "return",
- **.** um caráter qualquer,
- **\d** um dígito (equivalente a `[0-9]`),
- **\D** um caráter qualquer diferente de um dígito (equivalente a `[^0-9]`),

- **\s** um caráter branco (equivalente a `[ \t\n\x0B\f\r]`),
- **\S** um qualquer caráter visível (equivalente a `[^\s]`),
- **\w** uma letra, um dígito ou um "underscore": `[a-zA-Z_0-9]`,
- **\W** um caráter diferente de letra, dígito ou um "underscore" `[^\w]`,
- **^** o início de linha,
- **$** o fim de linha.

As classes `String`, `StringBuilder` e `Scanner` também oferecem métodos que trabalham com expressões regulares.

Por exemplo, o método `split`, que permite criar um *array* de *strings* resultante de dividir uma dada *string*, pode receber uma expressão regular para definir o ou os carateres que devem servir de delimitadores ou separadores.

Veja vários exemplos nos textos sobre `String` e `Stringuilder` e sobre `Scanner` acessíveis na página de LabP, na secção "Guiões e Apontamentos" em https://moodle.ciencias.ulisboa.pt/mod/page/view.php?id=138072

O material aqui apresentado é uma adaptação de um documento já existente de anos anteriores e todos os créditos são devidos aos seus autores.

Este documento apenas serve para suporte às aulas de LabP.