The Java™ Tutorials

Trail: Essential Classes
Lesson: Regular Expressions

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

Capturing Groups

In the previous section, we saw how quantifiers attach to one character, character class, or capturing group at a time. But until now, we have not discussed the notion of capturing groups in any detail.

Capturing groups are a way to treat multiple characters as a single unit. They are created by placing the characters to be grouped inside a set of parentheses. For example, the regular expression `(dog)` creates a single group containing the letters `"d" "o"` and `"g"`. The portion of the input string that matches the capturing group will be saved in memory for later recall via backreferences (as discussed below in the section, Backreferences).

Numbering

As described in the `Pattern` API, capturing groups are numbered by counting their opening parentheses from left to right. In the expression `((A)(B(C)))`, for example, there are four such groups:

1. `((A)(B(C)))`
2. `(A)`
3. `(B(C))`
4. `(C)`

To find out how many groups are present in the expression, call the `groupCount` method on a matcher object. The `groupCount` method returns an `int` showing the number of capturing groups present in the matcher's pattern. In this example, `groupCount` would return the number `4`, showing that the pattern contains 4 capturing groups.

There is also a special group, group 0, which always represents the entire expression. This group is not included in the total reported by `groupCount`. Groups beginning with `(?` are pure, *non-capturing groups* that do not capture text and do not count towards the group total. (You'll see examples of non-capturing groups later in the section Methods of the Pattern Class.)

It's important to understand how groups are numbered because some `Matcher` methods accept an `int` specifying a particular group number as a parameter:

- `public int start(int group)`: Returns the start index of the subsequence captured by the given group during the previous match operation.
- `public int end (int group)`: Returns the index of the last character, plus one, of the subsequence captured by the given group during the previous match operation.
- `public String group (int group)`: Returns the input subsequence captured by the given group during the previous match operation.

Backreferences

The section of the input string matching the capturing group(s) is saved in memory for later recall via *backreference*. A backreference is specified in the regular expression as a backslash (`\`) followed by a digit indicating the number of the group to be recalled. For example, the expression `(\d\d)` defines one capturing group matching two digits in a row, which can be recalled later in the expression via the backreference `\1`.

To match any 2 digits, followed by the exact same two digits, you would use `(\d\d)\1` as the regular expression:

```
Enter your regex: (\d\d)\1
Enter input string to search: 1212
I found the text "1212" starting at index 0 and ending at index 4.
```

If you change the last two digits the match will fail:

```
Enter your regex: (\d\d)\1
Enter input string to search: 1234
No match found.
```

For nested capturing groups, backreferencing works in exactly the same way: Specify a backslash followed by the number of the group to be recalled.

Previous page: Quantifiers
Next page: Boundary Matchers