The Java™ Tutorials

Trail: Essential Classes
Lesson: Regular Expressions

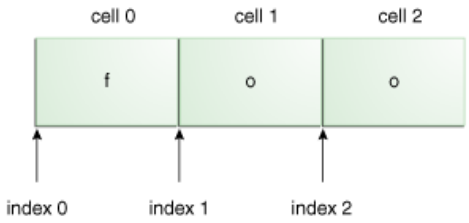
The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

String Literals

The most basic form of pattern matching supported by this API is the match of a string literal. For example, if the regular expression is `foo` and the input string is `foo`, the match will succeed because the strings are identical. Try this out with the test harness:

```
Enter your regex: foo
Enter input string to search: foo
I found the text foo starting at index 0 and ending at index 3.
```

This match was a success. Note that while the input string is 3 characters long, the start index is 0 and the end index is 3. By convention, ranges are inclusive of the beginning index and exclusive of the end index, as shown in the following figure:



The string literal foo, with numbered cells and index values.

Each character in the string resides in its own *cell*, with the index positions pointing between each cell. The string "foo" starts at index 0 and ends at index 3, even though the characters themselves only occupy cells 0, 1, and 2.

With subsequent matches, you'll notice some overlap; the start index for the next match is the same as the end index of the previous match:

```
Enter your regex: foo
Enter input string to search: foofoofoo
I found the text foo starting at index 0 and ending at index 3.
I found the text foo starting at index 3 and ending at index 6.
I found the text foo starting at index 6 and ending at index 9.
```

Metacharacters

This API also supports a number of special characters that affect the way a pattern is matched. Change the regular expression to `cat.` and the input string to `cats`. The output will appear as follows:

```
Enter your regex: cat.
Enter input string to search: cats
I found the text cats starting at index 0 and ending at index 4.
```

The match still succeeds, even though the dot "`.`" is not present in the input string. It succeeds because the dot is a *metacharacter* — a character with special meaning interpreted by the matcher. The metacharacter "." means "any character" which is why the match succeeds in this example.

The metacharacters supported by this API are: `<([{\^-=$!|]})?*+.>`

Note: In certain situations the special characters listed above will *not* be treated as metacharacters. You'll encounter this as you learn more about how regular expressions are constructed. You can, however, use this list to check whether or not a specific character will ever be considered a metacharacter. For example, the characters `@` and `#` never carry a special meaning.

There are two ways to force a metacharacter to be treated as an ordinary character:

- precede the metacharacter with a backslash, or

- enclose it within `\Q` (which starts the quote) and `\E` (which ends it).

When using this technique, the `\Q` and `\E` can be placed at any location within the expression, provided that the `\Q` comes first.

Previous page: Test Harness
Next page: Character Classes