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.

Character Classes

If you browse through the `Pattern` class specification, you'll see tables summarizing the supported regular expression constructs. In the "Character Classes" section you'll find the following:

Construct	Description
`[abc]`	a, b, or c (simple class)
`[^abc]`	Any character except a, b, or c (negation)
`[a-zA-Z]`	a through z, or A through Z, inclusive (range)
`[a-d[m-p]]`	a through d, or m through p: [a-dm-p] (union)
`[a-z&&[def]]`	d, e, or f (intersection)
`[a-z&&[^bc]]`	a through z, except for b and c: [ad-z] (subtraction)
`[a-z&&[^m-p]]`	a through z, and not m through p: [a-lq-z] (subtraction)

The left-hand column specifies the regular expression constructs, while the right-hand column describes the conditions under which each construct will match.

Note: The word "class" in the phrase "character class" does not refer to a `.class` file. In the context of regular expressions, a *character class* is a set of characters enclosed within square brackets. It specifies the characters that will successfully match a single character from a given input string.

Simple Classes

The most basic form of a character class is to simply place a set of characters side-by-side within square brackets. For example, the regular expression `[bcr]at` will match the words "bat", "cat", or "rat" because it defines a character class (accepting either "b", "c", or "r") as its first character.

```
Enter your regex: [bcr]at
Enter input string to search: bat
I found the text "bat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: cat
I found the text "cat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: rat
I found the text "rat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: hat
No match found.
```

In the above examples, the overall match succeeds only when the first letter matches one of the characters defined by the character class.

Negation

To match all characters *except* those listed, insert the "`^`" metacharacter at the beginning of the character class. This technique is known as *negation*.

```
Enter your regex: [^bcr]at
Enter input string to search: bat
No match found.
```

```
Enter your regex: [^bcr]at
Enter input string to search: cat
No match found.

Enter your regex: [^bcr]at
Enter input string to search: rat
No match found.

Enter your regex: [^bcr]at
Enter input string to search: hat
I found the text "hat" starting at index 0 and ending at index 3.
```

The match is successful only if the first character of the input string does *not* contain any of the characters defined by the character class.

Ranges

Sometimes you'll want to define a character class that includes a range of values, such as the letters "a through h" or the numbers "1 through 5". To specify a range, simply insert the "-" metacharacter between the first and last character to be matched, such as `[1-5]` or `[a-h]`. You can also place different ranges beside each other within the class to further expand the match possibilities. For example, `[a-zA-Z]` will match any letter of the alphabet: a to z (lowercase) or A to Z (uppercase).

Here are some examples of ranges and negation:

```
Enter your regex: [a-c]
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: [a-c]
Enter input string to search: b
I found the text "b" starting at index 0 and ending at index 1.

Enter your regex: [a-c]
Enter input string to search: c
I found the text "c" starting at index 0 and ending at index 1.

Enter your regex: [a-c]
Enter input string to search: d
No match found.

Enter your regex: foo[1-5]
Enter input string to search: foo1
I found the text "foo1" starting at index 0 and ending at index 4.

Enter your regex: foo[1-5]
Enter input string to search: foo5
I found the text "foo5" starting at index 0 and ending at index 4.

Enter your regex: foo[1-5]
Enter input string to search: foo6
No match found.

Enter your regex: foo[^1-5]
Enter input string to search: foo1
No match found.

Enter your regex: foo[^1-5]
Enter input string to search: foo6
I found the text "foo6" starting at index 0 and ending at index 4.
```

Unions

You can also use *unions* to create a single character class comprised of two or more separate character classes. To create a union, simply nest one class inside the other, such as `[0-4[6-8]]`. This particular union creates a single character class that matches the numbers 0, 1, 2, 3, 4, 6, 7, and 8.

```
Enter your regex: [0-4[6-8]]
Enter input string to search: 0
I found the text "0" starting at index 0 and ending at index 1.

Enter your regex: [0-4[6-8]]
Enter input string to search: 5
No match found.

Enter your regex: [0-4[6-8]]
Enter input string to search: 6
I found the text "6" starting at index 0 and ending at index 1.

Enter your regex: [0-4[6-8]]
Enter input string to search: 8
I found the text "8" starting at index 0 and ending at index 1.

Enter your regex: [0-4[6-8]]
Enter input string to search: 9
No match found.
```

Intersections

To create a single character class matching only the characters common to all of its nested classes, use `&&`, as in `[0-9&&[345]]`. This particular intersection creates a single character class matching only the numbers common to both character classes: 3, 4, and 5.

```
Enter your regex: [0-9&&[345]]
Enter input string to search: 3
I found the text "3" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[345]]
Enter input string to search: 4
I found the text "4" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[345]]
Enter input string to search: 5
I found the text "5" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[345]]
Enter input string to search: 2
No match found.

Enter your regex: [0-9&&[345]]
Enter input string to search: 6
No match found.
```

And here's an example that shows the intersection of two ranges:

```
Enter your regex: [2-8&&[4-6]]
Enter input string to search: 3
No match found.

Enter your regex: [2-8&&[4-6]]
Enter input string to search: 4
I found the text "4" starting at index 0 and ending at index 1.

Enter your regex: [2-8&&[4-6]]
Enter input string to search: 5
I found the text "5" starting at index 0 and ending at index 1.

Enter your regex: [2-8&&[4-6]]
Enter input string to search: 6
I found the text "6" starting at index 0 and ending at index 1.

Enter your regex: [2-8&&[4-6]]
Enter input string to search: 7
No match found.
```

Subtraction

Finally, you can use *subtraction* to negate one or more nested character classes, such as `[0-9&&[^345]]`. This example creates a single character class that matches everything from 0 to 9, *except* the numbers 3, 4, and 5.

```
Enter your regex: [0-9&&[^345]]
Enter input string to search: 2
I found the text "2" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 3
No match found.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 4
No match found.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 5
No match found.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 6
I found the text "6" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[^345]]
Enter input string to search: 9
I found the text "9" starting at index 0 and ending at index 1.
```

Now that we've covered how character classes are created, You may want to review the Character Classes table before continuing with the next section.

Previous page: String Literals
Next page: Predefined Character Classes