Theory: Sets, ranges, alternations

18 minutes 0 / 5 problems solved

Skip this topic Start practicing

3323 users solved this topic. Latest completion was about 2 hours ago.

The dot character allows us to write common patterns for matching. The dot, though, matches almost every possible character, and sometimes we want to be more specific in our regex. Then, the sets come to our rescue.

§1. The set of characters

Each set corresponds to a single character in the string, but what character exactly it can be is defined by the content of the set. The set is written in square brackets, `[]`. For example, the set `"[abc]"` means that a single character `"a"`, `"b"` or `"c"` can match it. Take a look at the example below:

```
String pattern = "
[bcr]at"; // it matches strings "bat", "cat", "rat", but not "fat"

"rat".matches(pattern); // true
"fat".matches(pattern); // false
```

You can use as many sets as you want and combine them with usual characters in a line. There are two sets in the following example:

```
String pattern = "
[ab]x[12]"; // can match a or b followed by x followed by either 1 or 2
```

This pattern can be successfully matched by the following strings:

```
"ax1", "ax2", "bx1", "bx2"
```

But the following strings **do not match** this pattern:

```
"xa1", "aax1", "bx"
```

As you can see, the order of sets in regular expressions is important. On the other hand, the order you put the characters inside the set does not matter.

§2. The range of characters

Sometimes we want to make our character sets quite large. In this case, we don't have to write them all down: we can specify the **range** designated by the dash symbol `-` instead. The character that precedes the dash denotes the starting point of the range, the character that follows it is the last character that falls into the range. If the needed characters immediately follow each other in the ASCII/Unicode encoding table we can put them to the set by means of a range of characters. This includes alphabetically ordered letters and numeric values. For example, we can write a set that matches every digit:

```
String anyDigitPattern = "[0-9]"; // matches any digit from 0 to 9
```

The same works for letter ranges such as `"[a-z]"` or `"[A-Z]"`. These ranges match all Latin lowercase and uppercase letters respectively. Since patterns are case sensitive, and if we want to match any letter ignoring the case, we can write the following regex:

```
String anyLetterPattern = "[a-zA-
Z]"; // matches any letter "a", "b", ..., "A", "B", ...
```

Note, that although the range `[A-z]` is technically valid, it includes additional symbols that are placed between uppercase and lowercase letters in ASCII table.

Current topic:

Sets, ranges, alternations

Topic depends on:

✕ Regexps in Java

Topic is required for:

Shorthands

Quantifiers

Table of contents:

As you can see, you can easily put several ranges in one set and mix them with separate characters in any order:

```
String anyLetterPattern = "[a-z!?.A-Z]"; // matches any letter and "!", "?", "."
```

§3. Excepting characters

In some cases, it is easy to define which characters are not wanted. Then, we can write a set that will match everything *except* for the characters mentioned in it. To do that, we write the hat character `^` as the first character of the set.

```
String regex = "[^abc]"; // matches everything except for "a", "b", "c"

"a".matches(regex); // false
"b".matches(regex); // false
"c".matches(regex); // false
"d".matches(regex); // true
```

The same works for ranges:

```
String regex = "[^1-6]";

"1".matches(regex); // false
"2".matches(regex); // false
"0".matches(regex); // true
"9".matches(regex); // true
```

§4. Escaping characters in sets

The general rule is that you do not need to escape special characters within sets since they are used in their literal meaning. For example, the set `[.?!]` will match a single dot, question, exclamation mark, and nothing else. However, the characters that form a set or a range should be escaped or put in a neutral position if we look for these literal symbols:

- to match the dash character itself, we should put it in the first or in the last position in the set: `"[-a-z]"` matches lowercase letters and the dash, and `"[A-Z-]"` matches uppercase letters and the dash;
- hat `^` need not escaping if placed anywhere but beginning. This way, the set `"[^a-z^]"` matches everything except for lowercase letters and the hat character;
- the square brackets should always be escaped:

```
String pattern = "[\\[\\]]"; // matches "[" and "]"
```

§5. Alternations

To this point, we were talking about single characters. There's another way to match one of the listed options, which includes longer sequences to choose from. The vertical bar `|` is used to match either the character sequences before or after the symbol.

```
String pattern = "yes|no|maybe"; // matches "yes", "no", or "maybe", but not "y" or "e"

"no".matches(pattern); // true
```

This is useful in situations when we want to look for one of the particular words, for example, "bear", "bat" or "bird" to complete the sentence `The giant ___ scared me`, when it's easier to indicate whole words. The vertical bar can be used together with parentheses that designate the boundaries of alternating substrings: everything that's in the parentheses is an optional substrings that can match the alternation block:

```
String pattern = "(b|r|go)at"; // matches "bat", "rat" or "goat"
String answer = "The answer is definitely (yes|no|maybe)";
```

In general, alternations serve for a purpose similar to that one of the sets: they describe multiple alternatives that a particular part of the pattern can match. While the sets, though, can match only a single character in the string, the alternations are used to define multi-character alternatives.

§6. Conclusions

Here's a recap:

- square brackets designate a set and are used to describe the set of characters that can match the pattern, like this `[123]`
- inside sets, ranges of characters designated by the dash can be used: `[1-3]`
- there are sets that "ban" specific characters. These are the sets with the hat character in the first position: `[^123]`
- the vertical bar can be used to describe multi-character alternating substring: `1|2|3`

Report a typo

324 users liked this theory. 2 didn't like it. **What about you?**

Start practicing

Comments (3) Hints (0) Useful links (0) Show discussion