Regular Expressions

Thinking In Java Pages 523-546

Break;

Regex basics

Basic symbols

- `a` , `b` , `c` - match "a", "b", "c" respectively (all numbers & letters)
- `.` - matches any one character
- `*` - match 0 or more occurrences of the last symbol
- `+` - match 1 or more occurrences of the last symbol
- `?` - match 0 or 1 occurrences of the last symbol
- `{n}` , `{n,m}` - Repeat previous match *n* times or *n* to *m* times
- `|` - alternation, matches either the pattern on the left or the right
- `()` - group characters together (eg: `(abc){3}` matches `abcabcabc`)

Character classes

- Match any single character in the character class
- groups of characters within [] brackets (eg `[abc]` matches "a", "b" or "c")
- hyphen denotes a range (eg: `[a-c]` == `[abc]` . Imagine `[a-zA-Z]` without `-`)
- Negate a character class with `^` (eg: `[^13579]` matches any character other than odd numbers)
- Note: Many special characters behave differently inside of character classes than they do outside of them.

Predefined character classes

Shortcuts for commonly used character classes

- `\s` - any whitespace character (`\\s` in Java Strings)
- `\S` - non-whitespace characters aka `[^\s]` (`\\S` in Java Strings)
- `\d` - any numeric digit aka `[0-9]` (`\\d` in Java)
- `\D` - any non-digit aka `[^0-9]` (`\\D` in Java)
- `\w` - a word character (`a-zA-Z_0-9`)
- `\W` - a non-word character aka `[^\w]`

Boundaries

- `^` - Beginning of line
- `$` - End of line
- `\b` - Word boundary

Using Regex in Java

Quirks

- Java Strings are converted to regex patterns -- this means backslashes and escape sequences are parsed and interpreted as what they represent in a String
- `\n` and `\t` (and some others) become the character they represent; `\\` becomes `\` and `\\\\` becomes `\\`
- As a result, Java regex escape sequences (such as `\d` , `\.` , `\\`) must be double-escaped (eg: `\\d` , `\\.` , `\\\\`)

Methods and Classes

- Used in String methods like `replaceAll` , `replaceFirst` , `split` , and `matches`
- `Pattern` objects represent compiled regular expressions (using `Pattern thePattern = Pattern.compile(regexStr)`
- `Matcher` objects provide information about matching a regex on to an input, created with

`Matcher m = thePattern.matcher(inputStr);`

Examples

```
public boolean batman(String str){
  return str.matches("(na){16}");
}
```

```
public boolean manOfSteel(String str){
  return str.matches("(Clark Kent)|(Superman)|(Kal-El)");
}
```

```
public boolean time(String str){
  return str.matches("w[io]bbly");
}
```

Backtracking

- Aforementioned Regex quantifiers are "greedy" - They will match the most characters possible to still allow the pattern to match
- Greedy quantifiers start at their longest and backtrack one match at a time until the whole pattern matches.
- Beware of catastrophic backtracking
- `.*` is very, **very** suspicious.

Lazy quantifiers

AKA Reluctant quantifiers

- consume smallest possible sequence to achieve a match.
- Same syntax as greedy quantifiers, plus a `?` at the end
- eg: `abc*?`, `[0-9]+?3`

Additional Resources

- Regexr

- Regular-expressions.info
- Regex Crossword - Regex-based challenges
- Duck Duck Go Regex Cheat Sheet
- Pattern class documentation - includes extensive regex explanations.