Reguläre Ausdrücke in Java

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Mit einem einzigen **regulären Ausdruck** kann man eine Menge unterschiedlicher Texte beschreiben, die **ähnlich aussehen**, z.B.

- Telefonnummern
- Mail-Adressen
- Datumsangaben
- Uhrzeiten

Reguläre Ausdrücke dienen dazu, solche Texte

- zu untersuchen,
- in ihre Einzelteile zu zerlegen oder
- zu verändern (suchen und ersetzen)

Es gibt verschiedene **Dialekte** für reguläre Ausdrücke, die in Einzelheiten voneinander abweichen, wie etwa

- Portable Operating System Interface-kompatibel (POSIX-kompatibel)
- Perl Compatible Regular Expressions (PCRE)

Der Begriff „Regulärer Ausdruck" wird auch zu **Regex** oder **Regexp** abgekürzt (englisch: regular expression).

Bestandteile

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Zeichen

Zeichen(folge)	passt zu
Zeichenliterale (Auswahl)	
`x`	dem selben Zeichen `'x'`
Den folgenden Metazeichen: `[ ] ( ) { } | ? + - * ^ $ \ .` muss man jedoch das Zeichen `'\'` voranstellen, um sie als Literale zu verwenden.	
`\n`	dem Zeilenvorschubzeichen `'\n'`
`\t`	dem Tabulatorzeichen `'\t'`
Zeichenklassen (Auswahl)	
`.` (ein Punkt)	einem beliebigen Zeichen (meist außer `'\n'`)
`[abc]`	einem der Zeichen `'a'`, `'b'` oder `'c'`
`[^abc]`	jedem Zeichen außer `'a'`, `'b'` oder `'c'`
`[a-zA-Z]`	`'a'` bis `'z'`, `'A'` bis `'Z'` (meist ASCII-Zeichensatz)
`\d`	einer Ziffer: `[0-9]`
`\D`	allem, was keine Ziffer ist: `[^0-9]`
`\s`	Leerraum: `[ \t\n\x0B\f\r]`
`\S`	allem, was kein Leerraum ist: `[^\s]`

Quantoren (Quantifizierer)

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Quantoren geben an, wie oft das davor stehende Zeichen/die davor stehende Gruppe vorkommen kann/muss (Default: einmal).

Quantor	Anzahl der Wiederholungen
Gierig (wiederholt möglichst oft, steckt zurück wenn nötig)	
`?`	einmal oder überhaupt nicht („optional“)
`*`	beliebig oft oder auch überhaupt nicht
`+`	beliebig oft, aber mindestens einmal
`{`n`}`	genau n Mal
`{`n`,}`	mindestens n Mal
`{`n`,`m`}`	mindestens n bis höchstens m Mal
Zurückhaltend (wiederholt möglichst selten, weitet aus wenn nötig)	
`??`	wie `?`
`*?`	wie `*`
usw.	
Besitzergreifend (wiederholt möglichst oft und steckt nicht mehr zurück)	
`?+`	wie `?`
`*+`	wie `*`
usw.	

Grenzen

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Diese prüfen, ob eine bestimmte **Position** (kein Zeichen!) innerhalb des Textes erreicht ist.

Grenze	Position
`^`	Text-Anfang (unterscheide von: `[^...]`)
`$`	Text-Ende bzw. Zeilenende (vor `'\n'`)
`\b`	Wortgrenze (Anfang oder Ende eines Wortes)
`\B`	jede Position außer einer Wortgrenze

Alternativen

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Um zu überprüfen ob **einer von mehreren unterschiedlichen Ausdrücken** im Text vorliegt, verbindet man diese Ausdrücke mit dem Zeichen `'|'`.

Beispiel:

Regulärer Ausdruck	passt zu
`Montag|Dienstag|Mittwoch`	`"Montag"`, `"Dienstag"` oder `"Mittwoch"`

Gruppen

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Gruppen werden mit runden Klammern **( )** notiert und dienen dazu,

1. mehrere **Teile eines regulären Ausdrucks** zu einer Einheit **zusammenzufassen**, auf die man dann z.B. einen Quantor anwenden kann, oder/und
2. die **zur Gruppe passenden Teile des Textes zu speichern**, damit man sie an anderer Stelle wieder verwenden kann.

Gruppen werden – beginnend bei **1** – in der Reihenfolge der öffnenden Klammern **nummeriert**. Unter dieser Nummer findet man später den gespeicherten Inhalt.

Die Gruppe **0** entspricht demjenigen Teil des Textes, der zum gesamten regulären Ausdruck passt.

Beispiele:

Regulärer Ausdruck	Text	Gruppe		
		0	1	2
`((ab)*)c+`	`abc`	`abc`	`ab`	`ab`
	`ababccc`	`ababccc`	`abab`	`ab`
	`c`	`c`	--leer--	--leer--
	`xyz`	--leer--	--leer--	--leer--

Ungespeicherte Gruppen

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Zweck: mehrere Teile eines regulären Ausdrucks zu einer Einheit zusammenfassen, **ohne dass** die dazupassenden Teile des Textes **gespeichert werden** (wie es bei gewöhnlichen Gruppen der Fall ist).

Ungespeicherte Gruppen notiert man in der Form `(?:  )`

Beispiele:

Regulärer Ausdruck	Text	Gruppe 0	Gruppe 1
`((?:ab)*)c+`	`abc`	`abc`	`ab`
	`ababccc`	`ababccc`	`abab`
	`c`	`c`	--leer--
	`xyz`	--leer--	--leer--

Rückwärtsreferenzen

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Diese erlauben es, den zuvor ermittelten **Inhalt einer Gruppe**

- später im regulären Ausdruck **wieder zu verwenden**
- in einem **Ersetzungsausdruck** zu verwenden

Rückwärtsreferenzen werden als $\backslash n$ notiert, wobei n die Nummer der Gruppe ist.

Beispiele:

Regulärer Ausdruck	passt zu
`('|").+\1`	(fast) jedem Stringliteral in einfachen oder doppelten Hochkommas, z.B. `"abc"` oder `'abc'`, aber nicht zu `'abc"`
`<(\w+)>(.*)</\1>`	einem zusammengehörenden Paar von öffnendem und schließendem XML-Tag, z.B. `<h1>Titel</h1>` oder `<b>fett</b>`. Der Tag-Inhalt wird in Gruppe 2 gespeichert.

Pattern und Matcher

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Um in Java mit einem regulären Ausdruck zu arbeiten, geht man meist in **drei Schritten** vor:

1. Man läßt den regulären Ausdruck (der ja eine Art von „Programm" darstellt) in ein **`Pattern`**-Objekt **kompilieren**. Dies ist zeitaufwändig, daher sollte man jeden regulären Ausdruck **nur einmal kompilieren**.

   Dazu dient die statische Methode **`Pattern.compile()`**, der man den regulären Ausdruck als **`String`**-Argument übergibt. Ungültige reguläre Ausdrücke bewirken eine **`PatternSytaxEception`**.

   ACHTUNG: wie in jedem Java-**`String`** muss auch in regulären Ausdrücken das Zeichen `'\'` verdoppelt werden!

2. Für den zu untersuchenden Text erzeugt man aus diesem **`Pattern`** einen **`Matcher`**. Dazu dient die **`Pattern`**-Methode **`matcher()`**.

3. Man verwendet geeignete **`Matcher`**-Methoden, um den Text zu untersuchen oder zu bearbeiten.

Matcher-Suchmethoden

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

1. `matches()`: ganzen Text vergleichen

Vergleicht den Text mit dem **`Pattern`** und liefert **`true`** bei **vollständiger** Übereinstimmung.

2. `lookingAt()`: Beginn des Textes vergleichen

Vergleicht den Text mit dem **`Pattern`** und liefert **`true`**, wenn der Text **vom Anfang an** (aber nicht notwendigerweise bis zum Ende) mit dem **`Pattern`** übereinstimmt.

3. `find()`: nächsten passenden Teil des Textes suchen

Sucht den **nächsten Teil** des Textes, der mit dem Pattern übereinstimmt und liefert **`true`**, wenn gefunden. Mit Hilfe der Methoden **`start()`** und **`end()`** kann man die folgenden passenden Textteile finden.

4. `group()`: gespeicherte Gruppeninhalte

Liefert den Inhalt einer gespeicherten Gruppe, nachdem **`matches()`**, **`lookingAt()`** oder **`find()`** erfolgreich ausgeführt wurde.

Matcher-Ersetzungsmethoden

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

1. `replaceAll()`

Liefert einen neuen **`String`**, in dem alle Teile des Textes, die mit dem **`Pattern`** übereinstimmen, durch den angegebenen **Ersetzungsausdruck** ersetzt sind. Im Ersetzungsausdruck dürfen auch Rückwärtsreferenzen vorkommen.

2. `replaceFirst()`

Wie **`replaceAll()`**, jedoch wird nur der allererste zum **`Pattern`** passende Teil ersetzt.

`String`-Methoden und reguläre Ausdrücke

Reguläre Ausdrücke in Java
Bestandteile
- Zeichen
- Quantoren (Quantifizierer)
- Grenzen
- Alternativen
- Gruppen
- Ungespeicherte Gruppen
- Rückwärtsreferenzen

Pattern und Matcher
- Matcher-Suchmethoden
- Matcher-Ersetzungsmethoden

String-Methoden und reguläre Ausdrücke

Reguläre Ausdrücke können auch in **`String`**-Methoden verwendet werden. Der reguläre Ausdruck muss aber bei jedem Aufruf neu kompiliert werden (⇒ langsamer).

`String`-Methode	**entspricht**
`matches``(`*regex*`)`	**`Pattern.compile(`***regex***`).matcher(`***string***`).matches()`**
`replaceFirst``(`*regex*`, `*ersetzung*`)`	**`Pattern.compile(`***regex***`).matcher(`***string***`).replaceFirst(`***ersetzung***`)`**
`replaceAll``(`*regex*`, `*ersetzung*`)`	**`Pattern.compile(`***regex***`).matcher(`***string***`).replaceAll(`***ersetzung***`)`**
`split``(`*regex*`, `*limit*`)`	**`Pattern.compile(`***regex***`).split(`***string***`, `***limit***`)`**