IT-Coaching RegEx/SEd

Stefan Wagner

Zusammenfassung

Reguläre Ausdrücke in Java mit string.replaceAll (“A”, “B”), bzw. Pattern und Matcher aus java.util.regex. reguläre Ausdrücke, Quantoren, Back-Referenzen, Syntax.

IT - Coaching java-figd-2425

XKCD zu regular expressions (3), (c) CC-by 2.5

“Immer, wenn ich etwas Neues lerne, hecke ich mir ausgefeilte Fantasiesituationen aus, in denen ich mit ihnen zum Retter in der Not werde.” - “Oh, nein, der Mörder muss ihr in den Urlaub gefolgt sein!” - “Aber um sie zu finden müssen wir uns durch 200 MB an Emails arbeiten, auf der Suche nach etwas, das wie eine Adresse gestaltet ist.” - “Es ist hoffnungslos.” - “Treten Sie zurück!” - “Ich kann reguläre Ausdrücke!” - (Perl) ‘tap, tap’.

Kursziel: Der Kurs verfolgt 3 Ziele: Das Programm **sed** und reguläre Ausdrücke sollen soweit gelernt werden, dass damit alltägliche Problemstellungen selbständig gelöst werden können. Außerdem soll ein umfassender Überblick gegeben werden, was mit *regulären Ausdrücken* alles möglich ist, ohne dass erwartet wird, dies auch gleich ohne nochmaliges Nachlesen in der Dokumentation, selbst umsetzen zu können. Aber es soll gezeigt werden, welcher Typ von Problem damit lösbar ist, so dass die Entscheidung, sich tiefer in die Materie einzuarbeiten, befördert wird.

Praxisrelevanz: Fast alle **Programmiersprachen**, die über BF hinausgehen, haben heute Bibliotheken zur Verarbeitung von regulären Ausdrücken, v.a. zum Suchen und Ersetzen, an Bord, und die Syntax ist dankenswerterweise überall sehr ähnlich.

Leistungsstarke **Editoren** haben häufig eine eingebaute Funktion des Suchens und Ersetzens mit regulären Ausdrücken. Auch Endanwendersoftware mit solchem Feature wurde schon gesehen. (Der Dateimanager Thunar erlaubt es beispielsweise, 89 Bilddateien in einem Verzeichnis zu markieren und alle Namen wie ‘CIMG-223456.JPG’ auf einen Rutsch in ‘strandparty-223456.jpg’ etc. umzubenennen).

Für ein einzelnes Dokument ist die Suchen(/Ersetzen)-Funktion oft einfacher, für 2-3 Dateien auch noch eine Option, aber wenn man hunderte oder **tausende Dokumente** durchsuchen will macht sich der routinierte Umgang mit Regexes bezahlt. Da es kostenlos ist, sprechen wir von der Zeit, die man spart, weil man’s kann, im Vergleich zur Zeit, die man ins Lernen steckt.

Häufiges Einsatzgebiet ist die Umwandlung der Ausgabe eines Programms, das die richtige Information im falschen Format liefert, zur Weiterverarbeitung in einem anderen Programm.

Links

Die Doku auf den Seiten Oracles ist umfangreich und soll hier auf die typischsten Anwendungsfälle gekürzt werden. Beispielsweise kommt die Unterscheidung von Whitespace in horizontalen und vertikalen Space in der Praxis selten vor.

```
digraph RegexStatemachine {
    # titel "Statemachine for Regex"
    # Quelle:
    # https://www.youtube.com/watch?v=528Jc3q86F8
    # Prof. Brailsford on Computerphile
    # s.a. https://www.youtube.com/watch?v=NTfOnGZUZDk
    # Brian Kernighan über G/RE/P

    subgraph ab {
        node [shape=rect] ac
        node [shape=circle]

        start -> 1 [label="a"]
        1 -> End  [label="c"]
```

```
    }

    subgraph "ab*c" {
        node [shape=rect] "ab*c"
        node [shape=circle]

        "start " -> "1 " [label=a]
        "1 " -> "1 " [label=b]
        "1 " -> "End " [label=c]
    }
}
```

Statemachine

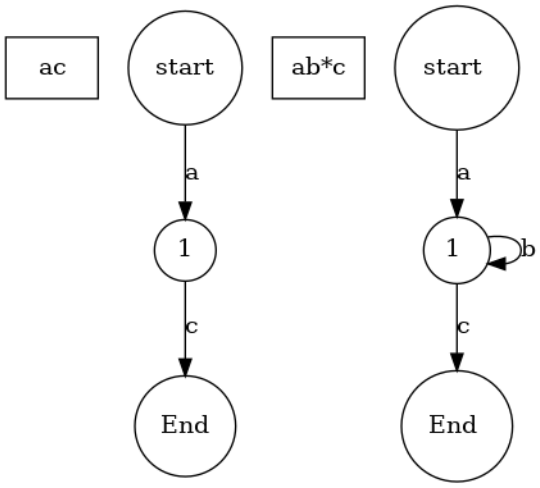

Abbildung 1: TITEL

Statemachine for Regex, Prof. Brailsford on Computerphile

Brian Kernighan über G/RE/P, ebd.

Beispiele

Die häufigsten Einsatzgebiete für reguläre Ausdrücke mit Java sind die Inputvalidierung (`String.matches(Pattern)`) und die Ersetzung (`String.replaceAll (Pattern, Replacement)`).

```
println ("192.168.32.53".matches ("([0-9]{1,3}.){3}[0-9]{1,3}"));
println ("Müller,Heinrich,12345 Berlin".replaceAll (",", "\t"));
```

XKCD zu regulare expressions (2), (c) CC-by 2.5

"Wenn Du Perlprobleme hast, tust Du mir leid. / Ich hatte 99 Probleme, / also benutzte ich reguläre Ausdrücke. / Jetzt habe ich 100 Probleme"

Zusammenfassung einfacher und spezieller Muster

```
x   Das Zeichen x
\\  Der Backslash
\xhh    Das Zeichen mit Hexwert 0xhh
\uhhhh  Das Zeichen mit Hexwert 0xhhhh
\N{name}    Das Zeichen mit dem Unicodezeichennamen 'name'
\t  The tab character ('\u0009')
\n  The newline (line feed) character ('\u000A')
\r  The carriage-return character ('\u000D')
\cx     Das Kontrollzeichen, welches mit x correspondiert (?)

Zeichenklassen

[abc]   a, b, oder c (einfache Klasse)
```

```
[^abc]  Alle Zeichen außer a, b, oder c (Negation)
[a-zA-Z]    a bis z oder A bis Z, inklusiv (Bereich)

Vordefinierte Zeichenklassen

.   Beliebiges Zeichen (kann Zeilenenden umfassen oder nicht)
\d  digit: [0-9]
\D  non-digit: [^0-9]
\s  whitespace character: [ \t\n\x0B\f\r] if UNICODE_CHARACTER_CLASS is not set. See Unicode Support.
\S  non-whitespace character: [^\s]
\w  word character: [a-zA-Z_0-9]
\W  non-word character: [^\w]

POSIX character classes (nur US-ASCII) (siehe auch Grafik ...)

\p{Lower}   A lower-case alphabetic character: [a-z]
\p{Upper}   An upper-case alphabetic character:[A-Z]
\p{ASCII}   All ASCII:[\x00-\x7F]
\p{Alpha}   An alphabetic character:[\p{Lower}\p{Upper}]
\p{Digit}   A decimal digit: [0-9]
\p{Alnum}   An alphanumeric character:[\p{Alpha}\p{Digit}]
\p{Punct}   Punctuation: One of !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
\p{Graph}   A visible character: [\p{Alnum}\p{Punct}]
\p{Print}   A printable character: [\p{Graph}\x20]
\p{Blank}   A space or a tab: [ \t]
\p{Cntrl}   A control character: [\x00-\x1F\x7F]
\p{XDigit}  A hexadecimal digit: [0-9a-fA-F]
\p{Space}   A whitespace character: [ \t\n\x0B\f\r]

java.lang.Character classes (simple java character type)

(ausgelassen)

Classes for Unicode scripts, blocks, categories and binary properties

\p{Sc}  A currency symbol

Begrenzer, Anker

^   Zeilenanfang
$   Zeilenende
\b  Wortgrenze: (?:(?<=\w)(?=\W)|(?<=\W)(?=\w)) (the location where a non-word character abuts a word char
\B  A non-word boundary: [^\b]

Zeilenumbruch

\R  Any Unicode linebreak sequence, is equivalent to \u000D\u000A|[\u000A\u000B\u000C\u000D\u0085\u2028\u2

Gierige Quantoren

X?  X, optional, einmal oder gar nicht
X*  X, beliebig oft, inkl. 0 mal
X+  X, einmal oder mehrmals
X{n}    X, genau n mal
X{n,}   X, mindestens n mal
X{n,m}  X, von n bis m mal
¬ X{,m} // bis zu n-mal, prüfen - nicht in Java, aber X{0,m} ist

Zögerliche Quantoren

(ausgelassen)

Besitzergreifende Quantoren
```

```
(ausgelassen)

Logische Operatoren

XY  X gefolgt von Y
X|Y     entweder X oder Y
(X)     X, als Gruppe für Rückbezüge

Rückbezüge

\n  Worauf imner die n-te Gruppe gepasst hat.
\k<name>    Worauf immer die benannte Gruppe "name" gepasst hat.

Quotierung

\   Nichts, aber maskiert das nächste Zeichen
\Q  Nichts, aber maskiert alle Zeichen bis \E
\E  Nichts, aber beendet ein zuvor mit \Q gestartetes Quoting

Spezialkonstrukte (benanntes Passen und Nichtpassen)

(ausgelassen)

## Reguläre Ausdrücke ##

### Testcode für JShell:

```java
void check (String muster, String ... worte) {
    println ("Muster: " + muster);
    for (String s : worte) println (s + "\t" + s.matches (muster));
}

String pattern = "fo{2,3}.*";
check (pattern, "foo", "Foo", "fool", "folsäure", "foooo", "foo ");

void filter (String pattern) throws IOException {
     Scanner sc = new Scanner (new File ("./palmstroem.txt"));
     while (sc.hasNext ())
     {
         String line = sc.nextLine ();
         if (line.matches (pattern))
             println (line);
     }
}
```

Muster und Quantoren

Muster

- `.` (Punkt, dot) Jokerzeichen

Der . Punkt passt auf jedes Zeichen. Ein literaler Punkt muss daher maskiert werden. - Der folgende Befehl findet alle 5 Zeichen langen Wörter, die mit "oo", gefolgt von einem belieibigen Zeichen und dann einem literalen Punkt enden.

```
pattern = ".oo.\\."
check (pattern, "Moos.", "Moor.", "poor.", "Moor ", "zooom", "Zoom.", "Zoo .", "Zoo.");
```

- Zeichenklassen

Eine Zeichenklasse wird durch eckige Klammern beschrieben:

```
 [a-z] # ein Kleinbuchstabe
 [abc] # eines der 3 Zeichen a-c
 [0-9a-fA-F] # eine Hexziffer
```
```

```
[0-9-] # 0 bis 9 und das Minuszeichen. Dieses muss 1. o. letztes Zeichen sein, um nicht `bis`
       # zu bedeuten)
[abc.] # a, b, c, oder Punkt. In der Klasse muss der Punkt nicht maskiert werden
[[:alnum:]] # ein alphanumerisches Zeichen, weitere Klassen siehe Manpage `man 7 regex` u. Graphik
[^aeiou] # Kein Vokal. Ist das erste Zeichen einer Klasse ein Caret, negiert es alle Zeichen der Klasse
```

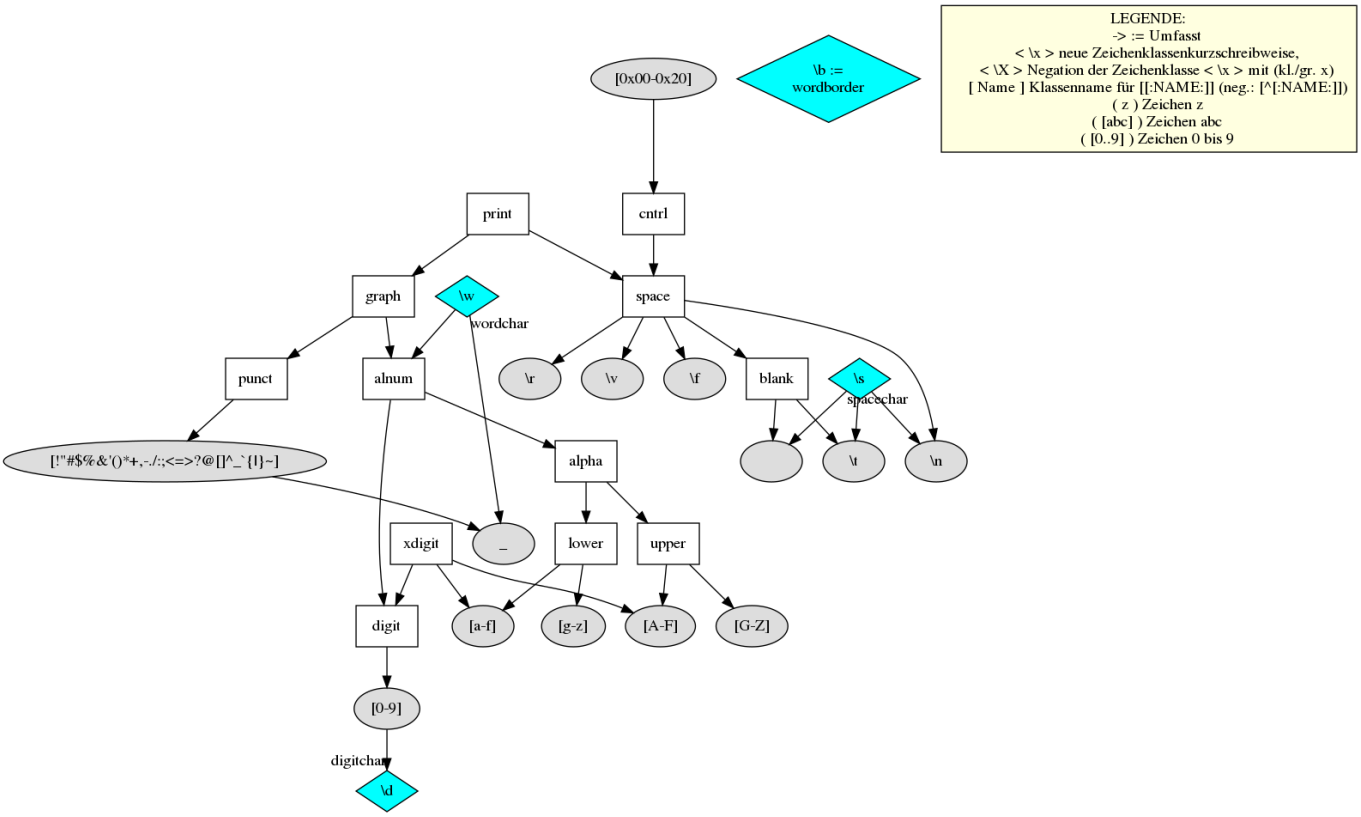

Abbildung 2: Regex Zeichenklassen, (c) CC-by-SA-NC 3.0, S.Wagner, Berlin

/pagebreak

Quellcode für die dot-Grafik, kein Lernstoff, aber für die interessierte Öffentlichkeit:

```
digraph Zeichenklassen {
    # https://spline.de/static/talks/graphviz.pdf
    node [shape=box] upper; lower; alpha; digit; alnum; punct; space; cntrl; blank; print; "graph"; xdigit
    node [shape=diamond; color=black; fillcolor="#00ffff"; style=filled;] "\\d"; "\\w"; "\\s"; "\\b" [labe
    node [shape=oval; color=black; fillcolor="#dddddd"; style=filled;];
    alpha -> upper
    alpha -> lower
    lower -> "[a-f]"
    lower -> "[g-z]"
    upper -> "[A-F]"
    upper -> "[G-Z]"
    alnum -> alpha
    alnum -> digit
    xdigit -> digit
    digit -> "[0-9]";
    "[0-9]" -> "\\d"  [headlabel="digitchar"];
    xdigit -> "[a-f]"
    xdigit -> "[A-F]"
    "graph" -> alnum
    "\\w" -> alnum
    "\\w" -> "_" [taillabel="wordchar"];
    "\\s" -> " "  [taillabel="spacechar"];
    "\\s" -> "\\t"
    "\\s" -> "\\n"
    "[!\"#$%&'()*+,-./:;<=>?@[\]^_`{|}~]" -> "_"
    "graph" -> punct
```

```
    print -> "graph"
    print -> space
    blank -> " "
    blank -> "\\t"
    space -> blank
    space -> "\\n"
    space -> "\\r"
    space -> "\\v"
    space -> "\\f"
    "[0x00-0x20]" -> cntrl
    cntrl -> space
    punct -> "[!\"#$%&'()*+,-./:;<=>?@[\]^_`{|}~]"
    node [shape="rect",color=black,fillcolor=lightblue] "LEGENDE:
    -> := Umfasst
    < \\x > neue Zeichenklassenkurzschreibweise,
    < \\X > Negation der Zeichenklasse < \\x > mit (kl./gr. x)
    [ Name ] Klassenname für [[:NAME:]] (neg.: [^[:NAME:]])
    ( z ) Zeichen z
    ( [abc] ) Zeichen abc
    ( [0..9] ) Zeichen 0 bis 9"
}
```

```
    println ("foo;Bbar?29ha,ha".replaceAll ("\\p{Alnum}", "-"));

    println ("foo;Bbar?29ha,ha".replaceAll ("[^\\p{Alnum}]", "-"));
```

- Spezielle Zeichen
  - Das ^ steht, wenn nicht als Klassenerstes, für den **Zeilenanfang** (oder Ausdrucksanfang).
  - Das $-Zeichen steht, wenn nicht als Zeilennummer für Dateiende, für das **Zeilenende** (s.o).
  - Das `|` Pipesymbol steht für ein **oder** `a|b|c` steht für a, b, oder c.

Oder u. Zeichenklassen

1. Findet einen Ausdruck und testet mit Beispielen (check), ob ein Wort mit **P** beginnt.
2. Findet … testet, ob ein Wort mit **p** oder **P** beginnt.
3. Findet … und testet, ob ein Wort mit **p, q, r oder s** beginnt.

XKCD zu regular expressions (1), (c) CC-by 2.5

Quantoren

```
    a+ # mindestens ein a (1 oder mehr). Das Plus muss nicht maskiert werden, d.h. "a\\+" passt auf "a+" a
    a? # vielleicht ein a (0 oder 1) # Das ? bedarf der Maskierung im Pattern, wenn es für
       # sich selbst steht (wie beim "+").
    a* # beliebig viele a (0 oder mehr)
    a{4}    # 4 a, aka aaaa #
    a{2,3}  # 2-3 a
    a{0,3}  # bis zu 3 a, evtl. keins, "a{,3}" ist in Java nicht zulässig.
    a{2,}   # 2 oder mehr a
```

`+ Beispiele:`

- Hier kommen Rohdaten zur Übung (Datei `teilnehmer`):

```
1;Vorn;Nachn;Fach;Stadt;Alter
2;Fritz;Maier;Bio;HH;24
3;Walter;Gropius;Arch;L;22
4;Frannie;Walter;Mat;Do;25
5;Katrin;Müller-Lüdenscheidt;Phy;HL;19
6;Franz;Maier;Bio;HH;21
7;Franz;Mayer;Bio;HH;27
8;Franz;Meyer;Bio,Inf;HH;22
9;Fritz;Mayer;Bio;HH;28
10;Stefan;Blue;Inf;B;22
11;Franz;Meier;Bio;HH;23
```

```
12;Franzie;Walter;Mat,Inf;Do;20
13;Fritz;Meier;Bio;HH;27
14;Karin;Müller-Lüdenscheid;Phy;HL;22
15;Stefanie;Blue;Inf;B;28
16;Fritz;Meyer;Bio;HH;24
17;Erna;Holtz;Arch;HR;26
18;Frank-Walter;Steinmeier;El;HR;23
19;Archibald;Rosenthal;El;Es;23
20;Anna-Sofia;Ups;Mat;F;22
```

Hilfsmethode filterTn

```
void filterTn (String pattern) throws IOException {
     Scanner sc = new Scanner (new File ("./teilnehmer.txt"));
     while (sc.hasNext ())
     {
         String line = sc.nextLine ();
         if (line.matches (pattern))
             println (line);
     }
}
```

- Beispiel mit Fragezeichen:
  - Lüdenscheid und Lüdenscheidt finden.

  `filterTn (".*Lüdenscheidt?;.*")`

- Beispiel mit mehreren Mengen: Mayer, Maier, Meyer und Meier finden:

  `filterTn (".*;M[ae][iy]er;.*")`

Große Aufgabe:

- a Alle Franz finden.
- b Alle Hamburger finden.
- c Alle Hansestädter finden.
- d Alle Architekten finden.
- e Alle Informatiker finden.
- f Alle Dortmunder finden.
- g Alle mit Bindestrichnamen finden.
- h Alle mit Bindestrichvornamen finden.
- i Alle mit Bindestrichnachnamen finden.

Wiederholungsaufgabe 9:

- Suchen Sie Franz oder Fritz (aber nicht Frannie oder Franzie)
- Suchen Sie Franz, Fritz, Frannie oder Franzie
- Regexes haben keine Vergleichsoperatoren für Zahlen. Suchen Sie dennoch alle Personen von 24-26 Jahren.
- … von 19 bis 21 Jahren

Aufgabe 10:

Bei dem Blödellied singt man erst die 4 Zeilen des Originalen Texts:

```
echo 'Zwei Chinesen mit dem Kontrabass,
saßen auf der Straße und erzählten sich was,
da kam die Polizei: "Na was ist denn das?"
"Zwei Chinesen mit dem Kontrabass!"' > 2chinesen.txt
```

Dann werden reihum all Vokal/Umlautkombinationen durch einen einheitlichen Vokal ersetzt, also erst durch a, dann e, i, o, u - Fortgeschrittene setzen mit y, ü, ö und ä fort.

Für das A soll die erste Zeile so aussehen:

> Zwa Chanasan mat dam Kantrabass,

Erzeugen Sie ein Textbuch mit 5 (8) Sedbefehlen.

Back-Referenzen

- Mit runden Klammern können literal oder durch Muster dargestellte Zeichen(ketten) zusammengefasst, und es kann später, bei der Ersetzung, darauf Bezug genommen werden.
- Die runden Klammern sind mit Backslash zu maskieren oder der Schalter -r (-E) muss verwendet werden.

```
head passwd | sed -r 's/.*:([a-z][a-z]+):.*/\1/p' # Gruppe aus passwd extrahieren
head passwd | sed 's/.*:\([a-z][a-z]\+\):.*/\1/p'
```

Mehrere Backreferenzen, umsortiert:

Aufgabe 11:

Gegen sei ein mathematischer Ausdruck mit 3 positiven Ganzzahlen a, b, c beliebiger Länge wie 8, 17, 234579901, in Klammern eine Addition oder Subtraktion, gefolgt von einem Produkt oder einer Division.

Also * (a+b)*c* (a+b)/c * (a-b)*c* (a-b)/c

Formen Sie diesen mit Sed nach dem Distributivgesetz um, also

- a*c+b*c
- a/c+b/c
- a*c-b*c
- a/c-b/c

Wenn Sie vor Ihren Kollegen fertig sind, erlauben Sie auch negative Ganzzahlen. Sind Sie auch damit vorzeitig fertig: Dezimalzahlen erlauben.

```
cat teilnehmer | sed -r 's/([^;]*);([A-Z][^;]*;)(.*)/\2 \3 id:=\1 \3/;  s/;HH;/;Hamburg;/;  s/;L;/;Lei

Vorn; Nachn;Fach;Stadt;Alter id:=1 Nachn;Fach;Stadt;Alter
Fritz; Maier;Bio;Hamburg;24 id:=2 Maier;Bio;HH;24
Walter; Gropius;Arch;Leipzig;22 id:=3 Gropius;Arch;L;22
Frannie; Walter;Mat;Dortmund;25 id:=4 Walter;Mat;Do;25
Katrin; Müller-Lüdenscheidt;Phy;Lübeck;19 id:=5 Müller-Lüdenscheidt;Phy;HL;19
Franz; Maier;Bio;Hamburg;21 id:=6 Maier;Bio;HH;21
Franz; Mayer;Bio;Hamburg;27 id:=7 Mayer;Bio;HH;27
Franz; Meyer;Bio;Hamburg;22 id:=8 Meyer;Bio;HH;22
Fritz; Mayer;Bio;Hamburg;28 id:=9 Mayer;Bio;HH;28
Stefan; Blue;Inf;Berlin;22 id:=10 Blue;Inf;B;22
Franz; Meier;Bio;Hamburg;23 id:=11 Meier;Bio;HH;23
Franzie; Walter;Mat;Dortmund;20 id:=12 Walter;Mat;Do;20
Fritz; Meier;Bio;Hamburg;27 id:=13 Meier;Bio;HH;27
Karin; Müller-Lüdenscheid;Phy;Lübeck;22 id:=14 Müller-Lüdenscheid;Phy;HL;22
Stefanie; Blue;Inf;Berlin;28 id:=15 Blue;Inf;B;28
Fritz; Meyer;Bio;Hamburg;24 id:=16 Meyer;Bio;HH;24
Erna; Holtz;Arch;Rostock;26 id:=17 Holtz;Arch;HR;26
Frank-Walter; Steinmeier;El;Rostock;23 id:=18 Steinmeier;El;HR;23
Archibald; Rosenthal;El;Essen;23 id:=19 Rosenthal;El;Es;23
```

Demo komplexe Aufgabe:

```
* Aus den Daten einen Serienbrieftext generieren:
```

"Liebe(r) $Vorn $Nachn in $(expanded (ORT)),"

```
+ Schrittweise Annäherung an das Ziel:
    - Die Zeilennummer brauchen wir nicht - sie kann weg:

sed 's/^[^;]*;//;' teilnehmer
    Vorn;Nachn;Fach;Stadt;Alter
    Fritz;Maier;Bio;HH;24
    Walter;Gropius;Arch;L;22
    ...

Das Muster `/^[^;]*;/` bedeutet: Zeilenanfang, gefolgt von Nichtsemikolons, beliebig viele (*),
dann aber gefolgt von Semikolon. Das ersetzen wir durch ein schnödes Nichts.

    - Was vor dem ersten Semikolon steht ist der Vorname:
```

```
sed -r 's/^[^;]*;//;  s/^([^;]*);/Liebe(r) \1;/' teilnehmer
    Lieber Vorn;Nachn;Fach;Stadt;Alter
    Lieber Fritz;Maier;Bio;HH;24
    Lieber Walter;Gropius;Arch;L;22
    Lieber Frannie;Walter;Mat;Do;25
    ...
```

- "Lieber Frannie" geht aber nicht - wir fragen die GF ob wir "Liebe(r) benutzen dürfen.
- Unseren Trick, mit beliebig vielen Nicht-Semikolons bis zum Semikolon zu arbeiten können wir wiederholen:

```
sed -r 's/^[^;]*;//;  s/^([^;]*);/Liebe(r) \1/;  s/^([^;]*);([^;]*);/ \1 \2 ;/' \
   teilnehmer
...
```

```
sed -r 's/^[0-9]*;//;  s/^([^;]*);/Liebe(r) \1 #/; s/(.*)#([^;]*);/\1\2 #/; s/^(.*)#[^;]*;/\1#/' teilnehme
    Lieber Vorn Nachn #Stadt;Alter
    Lieber Fritz Maier #HH;24
    Lieber Walter Gropius #L;22
    Lieber Frannie Walter #Do;25
    Lieber Katrin Müller-Lüdenscheidt #HL;19
```

- Der Übersichtlichkeit halber in eine Datei speichern in serienbrief.sed:

```
s/^[0-9]*;//
s/^([^;]*);/Liebe(r) \1 #/
s/(.*)#([^;]*);/\1\2 #/
s/^(.*)#[^;]*;/\1#/
```

weitere Anmerkungen

Die Gier regulärer Ausdrücke

Betrachtet werde der Ausdruck `"a.*bc"` Die zu durchsuchende Zeichenkette sei:

```
aaababaababaabaaadaadaababaaacaabaafaaaaaabaaabbaabababababX
```

, X sei entweder 'c' oder nicht 'c'. Der passende Ausdruck kann - wenn X c ist - an verschiedenen Stellen gefunden werden, die sich aber alle überlappen, weil sie alle bis zum c reichen. Java versucht im Normalfall den längstmöglichen Ausdruck zu bilden, aber wenn dieser nicht passt wird an der nächsten Stelle versucht.

Beim String

```
aaababaababaabaaadaadaababaaacaabaafaaaaaabaaabbaabababababcbcbcbc
```

passt der Ausruck an mehreren Stellen. Es wird nicht nur nach Möglichkeit der String gesucht, der möglichst weit links beginnt, sondern auch möglichst weit nach rechts reicht. Dies kann aber auch mit Syntaxmitteln modifiziert werden. Der Fachausdruck dazu lautet "greedy" (gierig) bzw. "non greedy" (bzw. pathologischer Input für reguläre Ausdrücke).

weitere Quellen:

- Interactive Visualisierung von Pattternmatching
- Linux `man 7 regex`
- Hackerrank-Aufgaben, Anmeldung nötig
- How to validate an email address using a regEx?