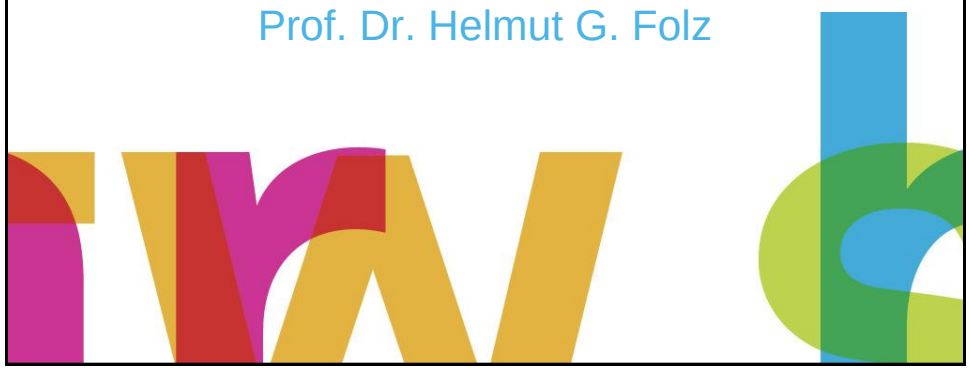
ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Einführung reguläre Ausdrücke

Prof. Dr. Helmut G. Folz



Regulärer Ausdruck

- Ein **Regulärer Ausdruck** (engl. *regular expression*) ist eine Zeichenfolge zur konstruktiven Beschreibung von Mengen beziehungsweise Untermengen von Zeichenketten mit Hilfe bestimmter syntaktischer Regeln.
- Anwendungen:
  - ⇨ Filterung von Texten: der reguläre Ausdruck wird mit dem Text abgeglichen
  - ⇨ Suchen und Ersetzen: zu ersetzende Zeichenketten können durch reguläre Ausdrücke beschrieben werden.

Anwendungen

- In Editoren wie z. B. ed, sed, vi, emacs, …
- In Skriptsprachen wie Perl, Python, Ruby, …
- In Programmiersprachenbibliotheken wie C, C++, Java, .Net, ..

Übersicht

Symbol	***Bedeutung***
X	Das Zeichen X
.	Beliebiges Zeichen
[abc] [a-z]	Ein Zeichen aus der Menge
[^abc] [^a-z]	Ein Zeichen nicht aus der Menge
^ $	Zeilenbeginn Zeilenende

Übersicht

Symbol	*Bedeutung*
`*`	Wiederholung, 0 oder beliebig viel
`+`	Wiederholung, 1 oder beliebig viel
`?`	Wiederholung, 0 oder 1-mal
`{n}`	Wiederholung, genau n-mal
`{n,}`	Wiederholung, mindestens n-mal
`{n,m}`	Wiederholung, mindestens n-mal, höchstens m-mal

Übersicht

Symbol	*Bedeutung*
`XY`	Ausdruck X gefolgt von Ausdruck Y
`X|Y`	Ausdruck X oder Ausdruck Y
`(X)`	Ausdruck geklammert

Übersicht

Vordefinierte Zeichenklassen

`.`	Any character (may or may not match line terminators)
`\d`	A digit: `[0-9]`
`\D`	A non-digit: `[^0-9]`
`\s`	A whitespace character: `[ \t\n\x0B\f\r]`
`\S`	A non-whitespace character: `[^\s]`
`\w`	A word character: `[a-zA-Z_0-9]`
`\W`	A non-word character: `[^\w]`

Beispiele

Regulärer Ausdruck	*Bedeutung*

`[abc]`
`[^abc]`
`[a-zA-Z]`
	Das Zeichen a, b oder c
	Beliebiges Zeichen außer a, b oder c
	Ein Zeichen aus den angegebenen Intervallen

`\d+`
`\d{1,4}`
`[-+]?\d{1,4}`
	Eine mindestens einstellige Zahl
	Eine ein- bis vierstellige Zahl
	Zusätzlich optionales Vorzeichen

`(http|ftp)`	Das Wort "http" oder das Wort "ftp"

Reguläre Ausdrücke in der Klasse String

`boolean matches(String regex)` ermittelt, ob dieser String mit dem angegebenen regulären Ausdruck übereinstimmt.
`String replaceAll(String regex, String replacement)` ersetzt alle Fundstellen, die dem Muster regex entsprechen, durch replacement.
`String replaceFirst(String regex, String replacement)` ersetzt die erste Fundstelle.
`String[] split(String regex)` spaltet diesen String auf in ein Feld von Teilstrings. Trennstellen sind die Fundstellen von regex. Diese werden nicht mit zurückgegeben.
`String[] split(String regex, int limit)` wie oben, jedoch mit begrenzter Fundstellenzahl.

Beispiel MatchesTest

```
public class MatchesTest {
    private Scanner input = new Scanner(System.in);
    public void start() {
        String zeile;
        String regex;
        System.out.print("Regulärer Ausdruck: ");
        regex = input.nextLine();
        System.out.println("Zeilen eingeben bis ende eingegeben");
        do {
            System.out.print("Zeile: ");
            zeile = input.nextLine();
            if (zeile.matches(regex))
               System.out.println(zeile + " wird durch „
                                  + regex + " beschrieben!");
            else
               System.out.println(zeile + " wird NICHT durch "
                                  + regex + " beschrieben!");

        } while (!zeile.equals("ende"));
}
```

Beispiel ReplaceTest

```
public class ReplaceTest {
    private Scanner input = new Scanner(System.in);
    public void start(){
        String zeile;
        String regex;
        String ersatz;
        String neueZeile;
        System.out.println("Zeilen eingeben bis ende eingegeben");
        do {
            System.out.print("Regulärer Ausdruck: ");
            regex  = input.nextLine();
            System.out.print("ersetzen durch: ");
            ersatz = input.nextLine();
            System.out.print("Zeile: ");
            zeile  = input.nextLine();
            neueZeile = zeile.replaceAll(regex, ersatz);
            System.out.println("Alle Auftreten von " + regex
                 + " durch " + ersatz + " ersetzt: \n" + neueZeile);
        } while (!zeile.equals("ende"));
}
```


Pattern und Matcher

- **java.util.regex**
  - ⇨ Package für die Behandlung von regulären Ausdrücken
- **Pattern**
  - ⇨ Objekte dieser Klasse kapseln reguläre Ausdrücke, die für eine effiziente Verarbeitung intern kompiliert werden.
  - ⇨ Erzeugung von Pattern-Objekten mittels der compile-Methode

```
Pattern p = Pattern.compile("[a-z]*");
```

- **Matcher**
  - ⇨ Matcher-Objekte sind die „Engine“ zum eigentlichen Suchen von auf einen regulären Ausdruck passenden Zeichenketten.
  - ⇨ Sie werden mit Hilfe der matcher-Methode von Pattern erzeugt

```
Matcher m = p.matcher("ein kleiner Test");
```

Wesentliche Methoden

Pattern	
`static Pattern compile(String regex)`	übersetzt den regulären Ausdruck in ein Pattern-Objekt
`Matcher matcher (CharSequence input)`	erzeugt ein Matcher-Objekt zum Überprüfen des übergebenen Strings

Matcher	
`boolean matches(String input)`	versucht, den ganzen Input mit dem Muster zu matchen
`boolean find(String input)`	versucht, das nächste Vorkommen des Musters im input zu finden
`String group()`	gibt das zuletzt gefundene String zurück
`int start()`	Startindex des letzten Treffers
`int end()`	ist der Index des letzten Zeichens des letzten Treffers + 1.

Beispiel RegexTestHarness (1)

```
import java.util.Scanner;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class RegexTestHarness {
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);

        while (true) {
            System.out.print("\nEnter your regex: ");
            Pattern pattern = Pattern.compile(input.nextLine());

            System.out.print("Enter input string to search: ");
            Matcher matcher = pattern.matcher(input.nextLine());

            boolean found = false;
```

Beispiel RegexTestHarness (2)

```
            while (matcher.find()) {
                System.out.printf(
                    "I found the text \"%s\" starting at " +
                    "index %d and ending at index %d.%n",
                    matcher.group(), matcher.start(),
                    matcher.end());
                found = true;
            }
            if(!found){
                System.out.printf("No match found.%n");
            }
        }
    }
}
```


Beispiel EGrep (1)

```
/**
 * Eine einfache Klasse, um über reguläre Ausdrücke Dateien
 * auszuwerten
 *
 * @author Folz
 * @version 2015
 */
public class EGrep {
    private File[] fileTab;
    private Scanner input = null;
    private Pattern pattern;
    private int totalNumberOfMatches = 0;

    public EGrep(String[] args) {
        System.out.printf("Searching %s ...%n", args[0]);
        pattern = Pattern.compile(args[0]);
        fileTab = new File[args.length - 1];
        for (int i = 0; i < fileTab.length; ++i) {
            fileTab[i] = new File(args[i+1]);
        }
}
```

Beispiel EGrep (2)

```
public void start() {
    for (File file : fileTab) {
        scanFile(file);
    }
    System.out.printf("Total number of matches: %d%n",
                      totalNumberOfMatches);
}

/**
 * Eine konkrete Datei durchsuchen
 * (vorerst über konkrete Dateiprüfungen)
 * @param file
 */
public void scanFile(File file) {
    String line;
    int numberOfLines = 0;
    int numberOfMatches = 0;
```


Beispiel EGrep (3)

```
    try {
        input = new Scanner(file);
        while (input.hasNextLine()) {
            line = input.nextLine();
            numberOfLines++;
            Matcher m = pattern.matcher(line);
            if (m.find()) {
                numberOfMatches++;
                System.out.printf("%3d: %s%n", numberOfLines,
                                                line);
            }
        }
    } catch (FileNotFoundException e) {
        System.out.println(e);
    } catch (Exception e) {
        System.out.println(e);
```

Beispiel EGrep (4)

```
        } catch (Exception e) {
            System.out.println(e);
        } finally {
            if (input != null)
                input.close();
            if (numberOfMatches > 0) {
                totalNumberOfMatches += numberOfMatches;
                System.out.printf("%s: Number of matches: %d%n%n",
                            file.getName(), numberOfMatches);
            }
        }
    }

    public static void main(String[] args) {
        if (args.length > 1)
            new EGrep(args).start();
        else
            System.out.println(
                        "Gebrauch: java EGrep regex datei1 ...");
    }
}
```