Universität Bremen

Praktische Informatik 1
Reguläre Ausdrücke

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

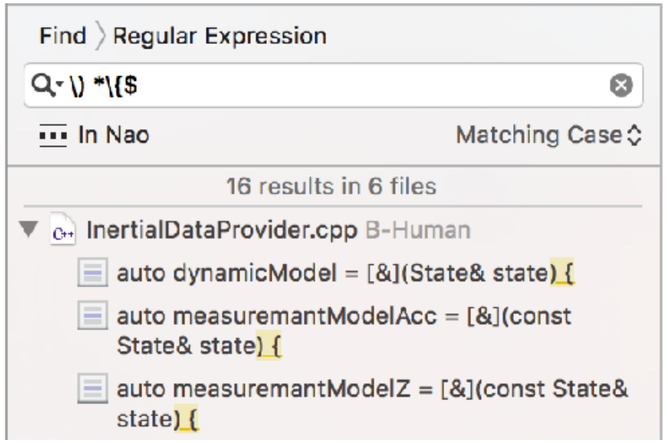
Bild: ©Michael Ihle

Universität Bremen

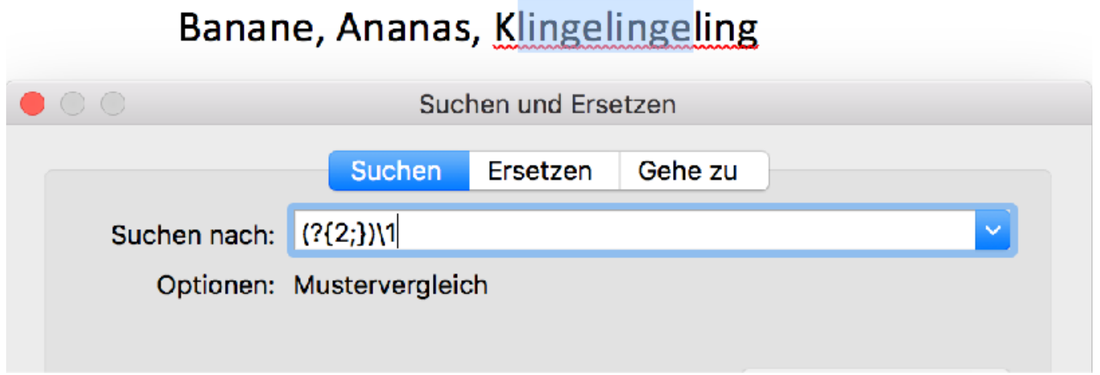
Motivation

```
final String[] words = (sentence + " ???").split(" ");
for (int i = 0; i < words.length; ++i) {
    final String[] responses = responseRules.get(words[
```

Eliza



Apple Xcode



Microsoft Word

```
 -> finger Schmidt | grep "Login:" | sed "s/Login: \(\w*\)[ \t]*Name: \(.*\)/\2: \1/"
Urs-Bjoern Schmidt: uschmidt
Sylvia Schmidt: sschmidt
Christian Schmidt: chrische
Swantje Schmidt: swsc01
...
```

Unix-Terminal

Universität Bremen

Reguläre Ausdrücke

- Reguläre Ausdrücke sind ein Werkzeug, um Zeichenketten auf eine bestimmte Syntax zu prüfen
  - „Ist die Zeichenkette ein valider Name?“
  - „Enthält Zeichenkette eine Zeichenfolge, die einem Muster folgt?“
- Regulärer Ausdruck ist Muster/„Pattern“
- Dieses Muster wird versucht, in der Zeichenkette wiederzufinden („Matching“)

Universität Bremen

Java-Methoden

- Klasse **String** verwendet reguläre Ausdrücke
- **matches(regex)**: Entspricht die Zeichenkette dem Muster?

```
"PI-1".matches("[A-Z][A-Z]-[0-9]") // true
```

- **replaceFirst(regex, replacement)**: Ersetze erstes Vorkommen des Musters

```
"PI-1".replaceFirst("[A-Z]","M") // "MI-1"
```

- **replaceAll(regex, replacement)**: Ersetze alle Vorkommen des Musters

```
"PI-1".replaceAll("[A-Z]","M") // "MM-1"
```

- **split(regex)**: Zerlege die Zeichenkette in Teile bei jedem Vorkommen des Musters

```
"A B  C   D".split(" +") // {"A", "B", "C", "D"}
```

Universität Bremen

Darstellung

- Darstellung in Java mittels Zeichenketten
  - Viele reguläre Ausdrücke enthalten den Backslash, um dem nachfolgenden Zeichen eine besondere Bedeutung zuzuweisen: **\d**
  - In Java-Zeichenketten muss dieser **escaped** werden: **"\\d"**
- Notation in diesem Foliensatz
  - Ohne Anführungszeichen: regulärer Ausdruck (jeweils nur ein ****)
  - Mit Anführungszeichen: Gültiger Java-String
- Diese Vorlesung stellt die Notation in Java dar
  - Details können in anderen Sprachen oder Anwendungen abweichen

Universität Bremen

Aufbau

- Elementarer regulärer Ausdruck
  - Zeichen
  - Zeichenklasse (z. B. „ein Großbuchstabe“)
  - Platzhalter (beliebiges Zeichen)
- Komplexer regulärer Ausdruck
  - Setzt sich aus anderen regulären Ausdrücken zusammen
  - Vielfachheit (z. B. „beliebig viele Ziffern“)
  - logische Verknüpfung (z. B. „Buchstabe oder Ziffer“)

Universität Bremen

Zeichen

- Buchstaben, Ziffern und Symbole: z.B. **d**, **7**, **:**
- Escape-Sequenzen für
  - bekannte Spezialzeichen: ****, **\f**, **\n**, **\r**, **\t**
    (es werden aber auch **"\f"**, **"\n"**, **"\r"**, **"\t"** akzeptiert)
  - Zeichen, die selbst eine funktionelle Bedeutung in regulären Ausdrücken haben: **\.**, *****, **\+**, **\?**, **\(**, **\)**, **\[**, **\]**, ****, **\^**, **\$**, **\|**
  - ASCII-Zeichen mittels Zahlenwert
    - **\0mnn**: Im Oktal-System (**m**, **n** Oktalziffern, **m < 4**)
    - **\xhh**: Im Hexadezimalsystem (**h** Hexadezimalziffern)

Universität Bremen

Zeichenklassen

- Menge der Zeichen, die akzeptiert werden als Liste in eckigen Klammern
  - Beispiel: **[aeiou]** akzeptiert „**a**“, „**e**“, „**i**“, „**o**“ oder „**u**“
- **^:** Komplement am Anfang der Klasse
  - Beispiel: **[^aeiou]** akzeptiert beliebige Zeichen außer den obigen
- **-**: Bereich zwischen Grenzen
  - Beispiel: **[A-Z]** akzeptiert beliebige Großbuchstaben
- Weiterhin Aneinanderreihung möglich: **[A-Ca-ck89]**
- **&&**: Und-Verknüpfung, d.h. beides muss erfüllt sein
  - Beispiel: **[a-f&&[^b-e]]** akzeptiert „**a**“ oder „**f**“

Universität Bremen

Vordefinierte Zeichenklassen

- **.**: Platzhalter (beliebige Zeichen)
- **\d**: Ziffern (Abkürzung für **[0-9]**)
- **\w**: Wort-Zeichen (Buchstaben, Ziffern, Unterstrich), Abkürzung für **[a-zA-Z_0-9]**
  - **(?U)** am Anfang des Ausdrucks aktiviert alle Unicode-Buchstaben und Ziffern
- **\s**: Zwischenraumzeichen (Leerzeichen, Zeilenschaltung usw.), Abkürzung für **[\t\n\x0B\f\r]**
- Wie normale Klassen verwendbar
  - Beispiel: **[^\d]** akzeptiert jedes Zeichen außer Ziffern

Universität Bremen

Text-Anker

- Text-Anker entsprechen keinem Zeichen, sondern Bedingungen, die an einer Stelle im Text erfüllt sein müssen
- **^**: Textanfang, z.B. passt **^Es war einmal** zu „**Es war einmal**“ am Anfang des Texts
- **$**: Textende, z.B. passt **sie noch heute\.$** zu „**sie noch heute.**“ am Ende des Texts
- **\b**: Wortrand, z.B. passt **\b[A-Z]** zu einem Großbuchstaben am Anfang eines Worts, **[a-z]\b** zu einem Kleinbuchstaben am Ende eines Worts
- **\B**: Nicht-Wortrand, z.B. passt **\B[A-Z]\B** zu einer Binnenmajuskel

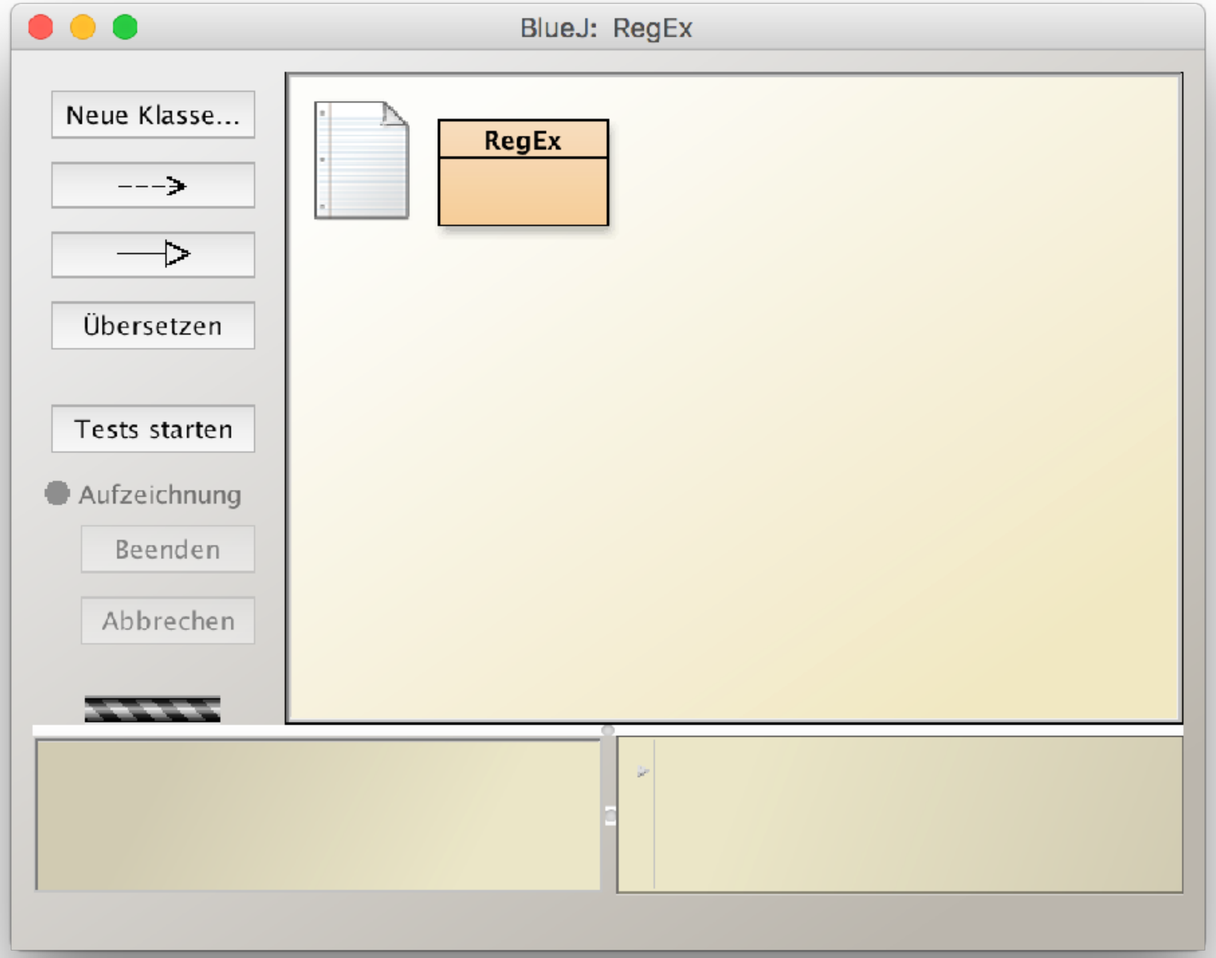
Universität Bremen

Quantifikatoren

- Legen akzeptierte Anzahl des Vorkommens eines regulären Ausdrucks fest
- **?**: keinmal oder einmal, z.B. passt **Schmidt?** zu „**Schmid**“ und „**Schmidt**“
- ***:** beliebig oft, auch keinmal, z.B. passt **so *genannt** zu „**sogenannt**“, „**so genannt**“, „**so genannt**“…
- **+**: beliebig oft, mindestens einmal, z.B. passt **Wi+lma** zu „**Wilma**“, „**Wiilma**“, „**Wiiilma**“…
- **{n}**: genau **n**-mal, z.B. passt **Schif{3}ahrt** zu „**Schifffahrt**“
- **{n,}**: mindestens **n**-mal, z.B. passt **PI-1!{3,}** zu „**PI-1!!!**“, „**PI-1!!!!**“, „**PI-1!!!!!**“…
- **{n,m}**: mindestens **n**-, höchstens **m**-mal, z.B. passt **Schmit{1,2}** zu „**Schmit**“ und „**Schmitt**“

Universität Bremen

Stud.IP-Downloads umbenennen: Demo



Universität Bremen

Quantifikatoren: Auswertung

- Quantifikatoren können auf drei Arten arbeiten
  - **greedy** (gierig)
  - **reluctant** (widerwillig)
  - **possessive** (besitzergreifend)
- Standardmäßig sind sie **greedy**
- Können umgestellt werden durch weiteres Zeichen
  - **?**: Macht den Quantifikator reluctant
  - **+**: macht den Quantifikator possessive

Universität Bremen

Quantifikatoren: greedy (gierig)

- Identifiziert größtmöglichen Ausdruck
- Nutzt **Backtracking**, wenn Gesamtausdruck sonst fehlschlägt
- Beispiel: In Zeichenkette „**Banane**“ passt **a.*n** auf Teil-String „**anan**“
  - Das „**a**“ an zweiter Position wird erkannt
  - Der greedy-Quantifikator nimmt (erstmal) gesamten Reststring für **.*** („**anane**“)
  - Dann bleibt aber kein abschließendes „**n**“ mehr
  - Gibt so viele Zeichen von rechts wieder frei (Backtracking), bis hinteres „**n**“ identifiziert werden kann

Universität Bremen

Quantifikatoren: reluctant (widerwillig)

- Identifiziert kleinstmöglichen Ausdruck
- Nutzt **Backtracking**, wenn Gesamtausdruck sonst fehlschlägt
- Beispiel 1: In Zeichenkette „**Banane**“ passt **a.*?n** auf Teil-String „**an**“
  - Das „**a**“ an zweiter Position wird erkannt
  - Der reluctant-Quantifikator nimmt nur so viel vom Rest-String, bis ein „**n**“ folgt („**an**“)
- Beispiel 2: In Zeichenkette „**Banane**“ passt **a.*?ne** auf Teil-String „**anane**“
  - Wie oben
  - Dann folgt aber kein „**e**“, also wird **.*** solange bis zum nächsten „**n**“ erweitert („**anan**“), bis auch der Rest passt (Backtracking)

Universität Bremen

Quantifikatoren: possessive (besitzergreifend)

- Identifiziert größtmöglichen Ausdruck
- Kein Backtracking (dadurch eventuell schneller, wenn es sowieso nicht passt)
- Beispiel 1: In Zeichenkette „**Banane**“ kann **a.*+n** nicht identifiziert werden
  - Das „**a**“ an zweiter Position wird erkannt
  - Der possessive-Quantifikator nimmt gesamten Rest-String für **.***
  - Dann bleibt aber kein abschließendes „**n**“ mehr
  - Der possessive-Quantifikator gibt aber im Gegensatz zum greedy-Quantifikator keine Zeichen wieder frei
    →Ausdruck nicht erkannt
- Beispiel 2: In Zeichenkette „**Banane**“ passt **a[^e]*+e** auf Teil-String „**anane**“

Universität Bremen

Verknüpfung regulärer Ausdrücke

- Aneinanderreihung, z.B. passt
  [A-Z]\d[A-Z]\w
  zu „**R2D2**“ oder „**C3PO**“

- **|** : Alternativen (Oder-Verknüpfung), z.B. passt
  [A-Z]\d[A-Z]\d|[A-Z]\d[A-Z][A-Z]
  zu „**R2D2**“ oder „**C3PO**“

- **(…)**: Klammern zur Gruppierung, z.B. passt
  [A-Z](\d[A-Z]|[A-Z]-)\d
  zu „**R2D2**“ oder „**BB-8**“

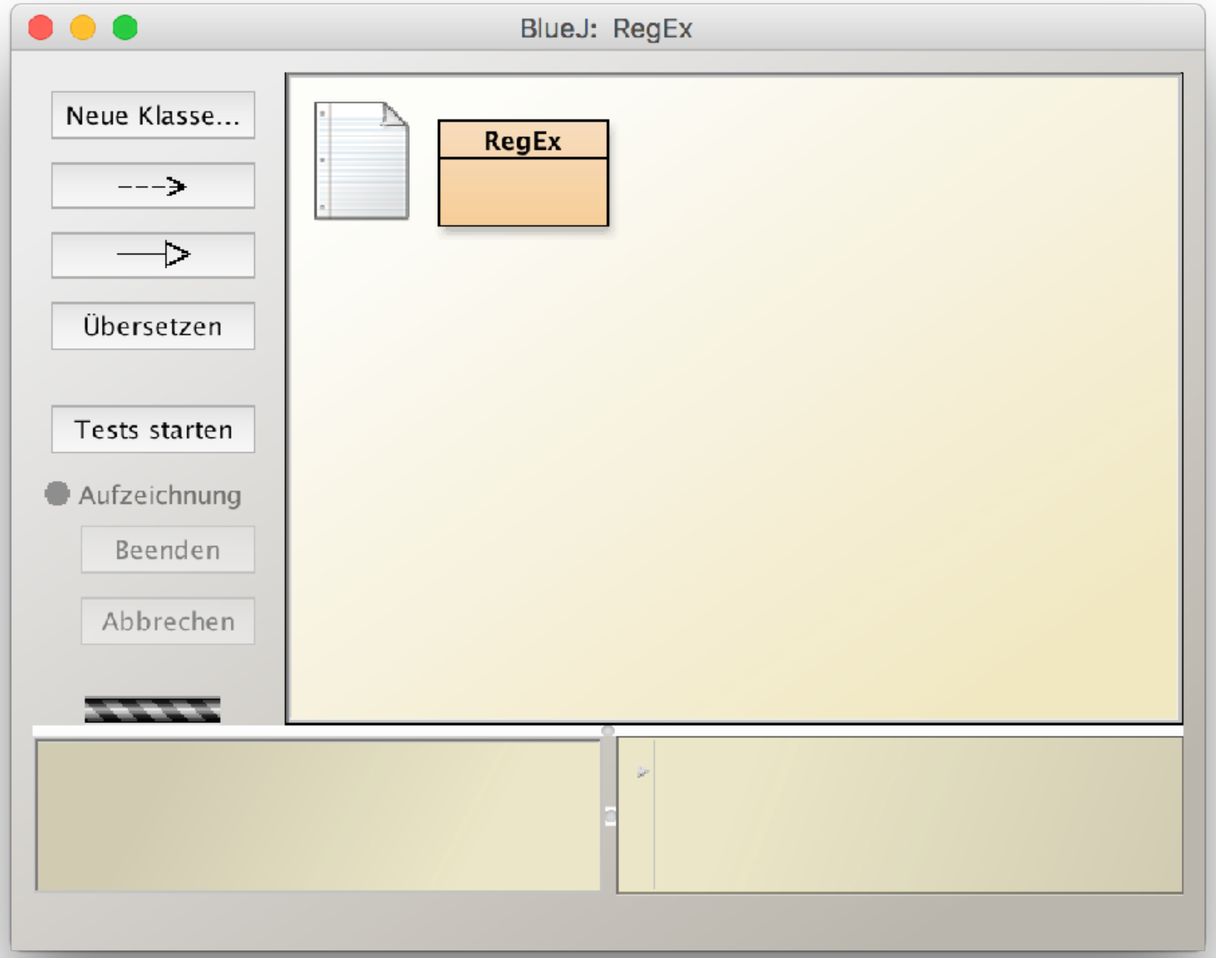
Universität Bremen

Rückbezüge

- Runde Klammern definieren auch **capturing groups**
- Beispiel: **\b(([A-Z])([a-z]*))\b**
  - Gruppe 1: **(([A-Z])([a-z]*))**
  - Gruppe 2: **([A-Z])**
  - Gruppe 3: **([a-z]*)**
- Auf **capturing groups** können Rückbezüge gemacht werden mit einem Backslash gefolgt von der Gruppennummer, z.B. **\1** (bei **replaceFirst**/**All** per Dollarzeichen-Nummer im Ersatztext, z.B. **$1**)
- Beispiel: **.*(.{2,})\1.*** akzeptiert alle Eingaben, die eine Wiederholung einer Gruppe aus mindestens zwei Zeichen haben, z.B. „**Banane**“ oder „**Klingelingeling**“

Universität Bremen

Pattern und **Matcher**: Demo



Universität Bremen

Java-Klassen

- Klasse **Pattern**: Realisiert die eigentliche Funktionalität regulärer Ausdrücke
  - **Pattern.compile(regex)** erzeugt Objekt für einen regulären Ausdruck
  - **split(input)** erlaubt Zerlegen anhand dieses Ausdrucks
  - **matcher(input)** erzeugt Objekt zum Durchsuchen der Eingabe anhand des regulären Ausdrucks
- Klasse **Matcher**: Realisiert die Suche mit Hilfe regulärer Ausdrücke
  - **find()** sucht nächstes Vorkommen des Suchmusters, **start()** und **end()** liefern seine Grenzen
  - **groupCount()** liefert Anzahl zugewiesener **capture groups**, **group(index)** liefert eine Belegung
  - **replaceFirst**/**replaceAll(replacement)** ersetzen Vorkommen des Musters durch Ersatz-String
- Einmalige Erzeugung von **Pattern** schneller als wiederholte Erzeugung, z.B. durch die Methoden von **String**

Universität Bremen

Zusammenfassung der Konzepte

- **Regulärer Ausdruck**
- **Zeichenklassen** und **Text-Anker**
- **Quantifikatoren** (**greedy**, **reluctant**, **possessive**)
- **Aneinanderreihung**, **Alternativen** und **Klammerung**
- **Capturing Groups** und **Rückbezüge**