Reguljära uttryck, automater, hashtabeller, mm

Programmering för språkteknologer 2

Sara Stymne

2013-09-09

Idag

- Reguljära uttryck (huvudfokus)
- Ändliga automater
- Läsning/skrivning (delvis repetition)
- Hashtabeller

Reguljära uttryck – relation till tidigare kurser

- Matematik för språkteknologer
  - Teori kring reguljära uttryck och finita automater
- Introduktion till datateknik för språkvetare
  - Praktisk användning av reguljära uttryck
  - grep och sed

Reguljära uttryck – relation till tidigare kurser

- Matematik för språkteknologer
  - Teori kring reguljära uttryck och finita automater
- Introduktion till datateknik för språkvetare
  - Praktisk användning av reguljära uttryck
  - grep och sed
- Den här kursen
  - Fokus på hur reguljära uttryck används i Java
  - Ska kunna representera enkla uttryck som automater

Reguljära uttryck

- Sätt att karaktärisera teckensekvenser
- En teckensekvens matchar eller matchar inte ett reguljärt uttryck
- Reguljära uttryck är nära besläktade med ändliga automater (de kan karaktärisera samma klasser av teckensekvenser)
- Mycket viktiga både i språkteknologin och datatekniken allmänt sett

Reguljära uttryck – exempel

- Alla strängar av ett a följt av ett b.

Reguljära uttryck – exempel

- Alla strängar av ett *a* följt av ett *b*.
  `ab`

Reguljära uttryck – exempel

- Alla strängar av ett *a* följt av ett *b*.
  `ab`
- Alla strängar av noll eller fler *a* följt av noll eller fler *b*.

Reguljära uttryck – exempel

- Alla strängar av ett *a* följt av ett *b*.
  `ab`
- Alla strängar av noll eller fler *a* följt av noll eller fler *b*.
  `a*b*`

Reguljära uttryck – exempel

- Alla strängar av ett *a* följt av ett *b*.
  `ab`
- Alla strängar av noll eller fler *a* följt av noll eller fler *b*.
  `a*b*`
- Alla strängar av sekvenser av *ab*

Reguljära uttryck – exempel

- Alla strängar av ett *a* följt av ett *b*.
  `ab`
- Alla strängar av noll eller fler *a* följt av noll eller fler *b*.
  `a*b*`
- Alla strängar av sekvenser av *ab*
  `(ab)*`

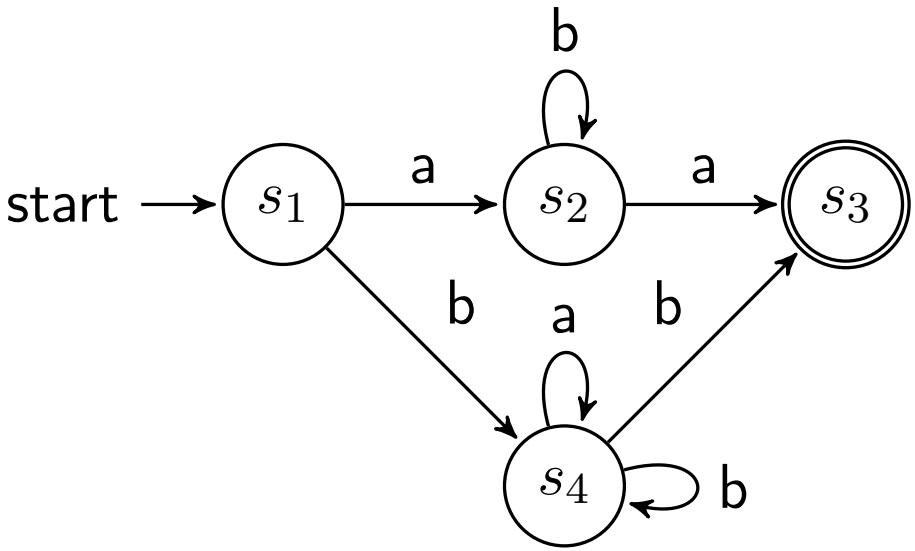
Reguljära uttryck – exempel

- Alla strängar av ett *a* följt av ett *b*.
  `ab`
- Alla strängar av noll eller fler *a* följt av noll eller fler *b*.
  `a*b*`
- Alla strängar av sekvenser av *ab*
  `(ab)*`
- Alla strängar av ett eller fler *a* följt av ett eller fler *b*.

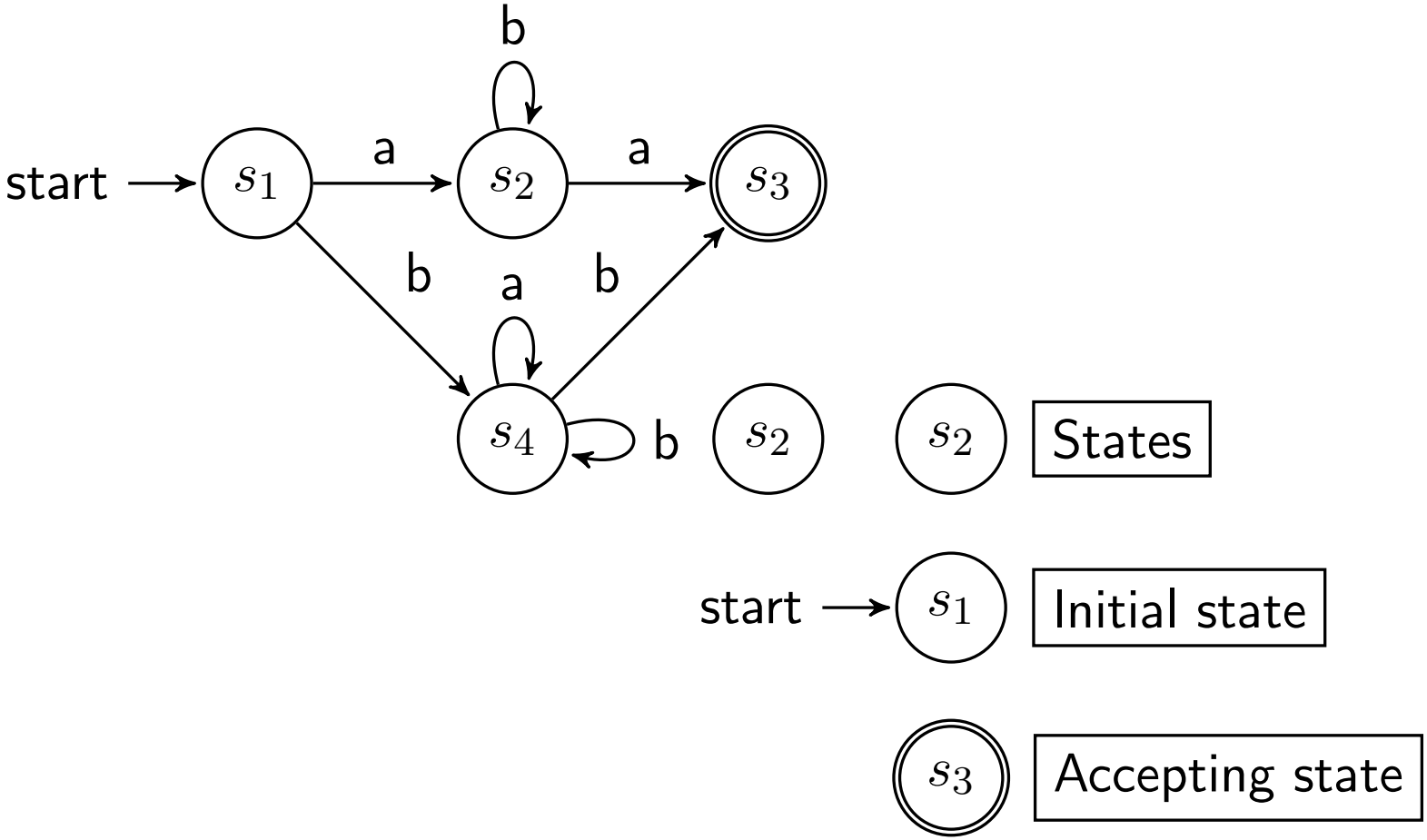
Reguljära uttryck – exempel

- Alla strängar av ett *a* följt av ett *b*.
  `ab`
- Alla strängar av noll eller fler *a* följt av noll eller fler *b*.
  `a*b*`
- Alla strängar av sekvenser av *ab*
  `(ab)*`
- Alla strängar av ett eller fler *a* följt av ett eller fler *b*.
  `a+b+`

Deterministiska ändliga automater



Deterministiska ändliga automater



Ändliga automater – input

- **Accepterad input**: En maskin accepterar input om maskinen hamnar i ett sluttillstånd efter att ha behandlat hela inputsekvensen
- **Icke-accepterad input**: En maskin accepterar **inte** input om maskinen hamnar i ett sluttillstånd efter att ha behandlat hela inputsekvensen

Ändliga automater – exempel

- Rita automater för:
- Alla strängar av ett *a* följt av ett *b*.
  `ab`
- Alla strängar av noll eller fler *a* följt av noll eller fler *b*.
  `a*b*`
- Alla strängar av sekvenser av *ab*
  `(ab)*`
- Alla strängar av ett eller fler *a* följt av ett eller fler *b*.
  `a+b+`

Reguljära uttryck i Java

- Reguljära uttryck representeras som strängar – som alltså tolkas på ett speciellt sätt
- Syntaxen kan skilja sig lite från vad ni är vana vid
- Relevanta klasser:
  - java.util.regex
    - Pattern
    - Matcher
  - String-klassen har en del funktioner med reguljära uttryck
  - Reguljära uttryck förekommer även som argument för metoder i andra klasser, t.ex. Scanner

Reguljära uttryck som strängar

- I Java skrivs reguljära uttryck som strängar
- Specialtecken
  - . – vilket tecken som helst
  - () – grupperar tecken
  - *+? – kvantifierare
  - Alla specialtecken: `([{\^-=$!|]})?*+.`
- Backslash måste dubbleras
  - För att använda specialtecken som vanliga tecken (”\\.” för tecknet punkt)
  - För att använda fördefinierade kalsser (”\\s” för vita tecken)

Reguljära uttryck – kvantifierare

- * : matcha noll eller flera
  "a*" matchar "", "a", "aa", "aaa", ...
- + : matcha ett eller flera
  "a+" matchar "a", "aa", "aaa", ...
- ? : matcha noll eller ett
  "a?" matchar "" eller"a"

Reguljära uttryck – teckenklasser

- Teckenklasser [] – ett tecken av de som står innanför hakparanteserna matchas
  `"[abc]"` matchar vilken som av `"a"`, `"b"` och `"c"`.
  `"[a-z]"` matchar vilken som av `"a"`, `"b"`, ..., `"z"`.
  `"[a-zA-Z]"` matchar vilken som av `"a"`, `"b"`, ..., `"z"` och `"A"`, `"B"`, ..., `"Z"`.

Reguljära uttryck – negerade teckenklasser

- Negerade teckenklasser: matchar ett tecken förutom de som anges
- `"[^abc]"` matchar alla (enstaka) tecken UTOM `"a"`, `"b"` och `"c"`.
- `"[^a-ms]"` matchar alla (enstaka) tecken UTOM `"a"`, `"b"`, ..., `"m"` och `"s"`.
- `"[^a-zåäöA-ZÅÄÖ]"` matchar alla (enstaka) tecken som inte är svenska bokstäver.

Reguljära uttryck – fördefinierade grundklasser

- Symboler för **fördefinierade klasser**:
- `"\d"` matchar en siffra (synonym `"[0-9]"`).
  Obs! `"\d"` måste stå som `"\\d"` i koden. (Man måste backslasha backslashen för att få in den i strängen.)
- `"\D"` matchar en icke-siffra (synonym `"[^0-9]"`).
  Motsvarande finns för w och s
- `"\w"` matchar ett (engelskt) ordtecken (synonym `"[a-zA-Z0-9]"`).
- `"\s"` matchar "vita tecken" (blanktecken, tab, nyrad, . . .)
- `"\b"` ordgräns (matchar inget tecken)

Reguljära uttryck – teckenklasser i Unicode

- Unicode: standard för de teckenklasser som behövs när vi vill ha alla skriftsystem för naturliga språk.
- Rekommenderas för svenska eftersom "åäö" hanteras korrekt!
- Exempel på Unicode-teckenklasser:
  - `\p{L}`: bokstav
  - `\p{Ll}`: gemen
  - `\p{Lu}`: versal
  - `\p{P}`: interpunktionstecken
- Det finns många fler klasser. Slå upp!
- Glöm inte att skriva med dubbla backslash: "\\p{L}+" för att matcha ett ord, till exempel!

Regexp i Java – String

- boolean matches(String regex)
- String replaceAll(String regex, String replacement)
- String replaceFirst(String regex, String replacement)
- String[] split(String regex)

Regexp i Java – String

- boolean matches(String regex)
- String replaceAll(String regex, String replacement)
- String replaceFirst(String regex, String replacement)
- String[] split(String regex)

VISA KODEXEMPEL!

Reguljära uttryck – alternativ

- | kan användas för att uttrycka alternativ
  - `"a|b"` matchar `"a"` och `"b"` (motsvarar `"[ab]"`)
  - `"apa|ko"` matchar `"apa"` och `"ko"`

Reguljära uttryck – gruppering

- Delar av reguljära uttryck kan grupperas med paranteser
- Kan användas för att begränsa alternativ
  - "god(are|ast)" matchar "godare", "godast"
  - "godare|ast" matchar "godare", "ast"
- Kan användas för kvantifierare
  - "(hem)+" matchar "hem", "hemhem", ... – EJ "hemm"
  - "god(are|ast)?" matchar "god", "godare", "godast"
- Kan även användas för att hänvisa tillbaka i uttrycket
  - "([1-4])\\1" matchar "11", "22", "33", "44"

Reguljära uttryck – gruppering

- Man kan ha flera grupper i ett uttryck
- Grupperna räknas från vänster baserat på vänsterparanteser
  - "((an(teckna))(r))"
    - grupp 1: antecknar
    - grupp 2: anteckna
    - grupp 3: teckna
    - grupp 4: r
- Grupperna kan användas för att välja ut eller byta ut delar av ett uttryck
  - I samma uttryck – \x
    "([rst])\\1" "rr", "ss", "tt"
  - I utbytesuttryck – $x

Regexp i Java – java.util.regex

- Mer avancerade funktioner än i String
- Två huvudklasser:
  - Pattern – representerar ett reguljärt uttryck
  - Matcher – matchar ett Pattern, och har funktioner för att jobba med en matching
- Typisk användning:

```
String sentence = ...
Pattern p = Pattern.compile("\\b[Jj]ag\\b");
Matcher m = p.matcher(sentence);
\\ Gör något med matchning m
```

java.util.regex – exempel på vad man kan göra

- Matcha (jfr grep)
  - Hitta strängar där ett uttryck förekommer (som en del av strängen)
  - Hitta alla matchningar för ett uttryck i en sträng
- Ersätt (jfr sed)
  - Byt ut ett uttryck mot ett annat

java.util.regex – exempel på vad man kan göra

- Matcha (jfr grep)
  - Hitta strängar där ett uttryck förekommer (som en del av strängen)
  - Hitta alla matchningar för ett uttryck i en sträng
- Ersätt (jfr sed)
  - Byt ut ett uttryck mot ett annat

VISA KODEXEMPEL!

Matcher – metoder

- find – hitta nästa förekomst av uttrycket
- group(int) – ta fram grupper ur det matchade uttrycket
  - group() eller group(0) – hela matchningen
  - group(n) – gruppering n från matchningen
- groupCount – hur många grupper finns det?
- start/end – första/sista index för matchningen
- replaceAll
- replaceFirst
- matches

Reguljära uttryck – mer om kvantifiering

- **Greedy**: Matchar så lång delsträng som möjligt först; försöker med kortare därefter. (Default.)
- **Reluctant**: Matchar så kort delsträng som möjligt först; försöker med längre därefter.

Reguljära uttryck – mer om kvantifiering

- **Greedy**: Matchar så lång delsträng som möjligt först; försöker med kortare därefter. (Default.)
- **Reluctant**: Matchar så kort delsträng som möjligt först; försöker med längre därefter.
- **Possessive**: Matchar som greedy, men eliminerar alternativa tidigare matchningsalternativ. Den lägger så att säga beslag på den delsträng som den matchat.

Reguljära uttryck – mer om kvantifiering

- **Greedy**: Matchar så lång delsträng som möjligt först; försöker med kortare därefter. (Default.)
- **Reluctant**: Matchar så kort delsträng som möjligt först; försöker med längre därefter.
- **Possessive**: Matchar som greedy, men eliminerar alternativa tidigare matchningsalternativ. Den lägger så att säga beslag på den delsträng som den matchat.

Greedy	Reluctant	Possessive
X?	X??	X?+
X*	X*?	X*+
X+	X+?	X++

Exempel – kvantifiering

- Greedy:
  - `"en <i>liten</i> hund".replaceAll("</?.*>","")` returnerar "en hund"
- Reluctant
  - `"en <i>liten</i> hund".replaceAll("</?.*?>","")` returnerar "en liten hund"

Reguljära uttryck – Java vs grep/sed

- Varför ska man använda reguljära uttryck i Java?
- Fungerar inte grep och sed lika bra?

Reguljära uttryck – Java vs grep/sed

- Varför ska man använda reguljära uttryck i Java?
- Fungerar inte grep och sed lika bra?
- Beror på vad man ska göra!
- Om man ska använda datan ytterligare, till exempel göra en frekvenstabell som i labben, kan det vara bra att använda sig av ett fullt programmeringsspråk!

Läsning/skrivning

- Läsning
  - Scanner
- Skrivning
  - format

Scanner

- Klass som används för att läsa data
- Kan läsa från olika källor
  - Standard in

    ```
    Scanner s = new Scanner(System.in);
    ```
  - Från en annan sträng

    ```
    String row = ...
    Scanner s = new Scanner(row);
    ```
  - Standard in

    ```
    File f = new File(filename);
    Scanner s = new Scanner(f);
    ```

Scanner läsning

- Läsmetoder
  - next() – läs nästa token
  - nextLine() – läs nästa rad
  - nextInt() – läs nästa heltal
  - ...
  - next(String/Pattern regex) – läs med avgränsaren regex
- Motsvarande metoder för kontroll
  - hasNext();
  - hasNextLine()
  - hasNextInt()
  - ...

Scanner – exempel

Läs data från en fil: (ordrad.txt)

```
bok 23 34 78
kål 2
```

KOD:

```
try {
   Scanner lineReader = new Scanner(new File("ordrad.txt"));
   while (lineReader.hasNextLine()) {
      Scanner rowReader = new Scanner(lineReader.nextLine());
      String word = rowReader.next();
      WordInfo wi = new WordInfo(word);
      while (rowReader.hasNextInt()) {
         wi.addRow(rowReader.nextInt());
      }
      wi.print();
   }
}
catch (FileNotFoundException e) {
   System.out.println("Problem reading " + e);
}
```

Format

- Formatter är en klass som kan användas för att formattera output
- Kan skriva ut formatterat
  - Tal
  - Datum
  - Marginaler
  - ...
- Kan hantera olika locales – anpassa till konventioner i olika språk

Format – användning

- Kan användas för att skriva till olika typer av mål
  - Formatter(Appendable a) – till exempel för StringBuilder
  - Formatter(File file)
  - Formatter(OutputStream os)
- Kan även ange en locale,
  ex: Formatter(Appendable a, Locale l)
- Används ofta direkt för att skriva till standard out:
  - System.out.format

Formatter – användning

- System.out.format(String format, Object... args)
  - format: formatsträng
  - args: 0–n antal argument
  - formatsträngen kan innehålla text och ”platshållare” som fylls i från argumenten
  - Exempel:
    - System.out.format(”Ordet %s finns %d gånger, word, freq)”
    - Utskrift: Ordet hund finns 4 gånger

Formatspecificerare

- %s – sträng
- %d – heltal, decimalformat
- %o – heltal, oktalformat
- %f – decimaltal
- %e – decimaltal, vetenskaplig notation
- %t – datum/tid (kräver ytterligare specificering)
- %n – nyrad
- ...

Formatspecificerare – detaljer

- %[flags][width][.precision]conversion
  - conversion – vilken huvudtyp det är, se förra sliden
  - flags – specifikationsflaggor, vilka som är möjliga beror på conversion
  - width – minimumvidd för utskriften
  - .precision – antal decimaler för decimaltal (används enbart för tal)
- Exempel
  - ”%.2f” – decimaltal avrundat till två decimaler, ex. 23.57
  - ”%10s” – sträng utskriven högerjusterad, med minst 10 tecken, utfyllt med mellanslag, ex. ” hej”
  - ”%-10s” – sträng utskriven vänsterjusterad, med minst 10 tecken, utfyllt med mellanslag, ex. ”hej ”
  - ”%0+8.3f” – decimaltal avrundat till två decimaler, med tecken angivet, minst 8 positioner långt, utfyllt med nollor, ex. ”+023.568”

Hashtabeller

- Idag
  - Vad är en hashtabell (översiktligt)
  - Vilka klasser finns i Javas bibliotek
  - Hur använder man den
- Senare i kursen
  - Hur fungerar en hashtabell teoretiskt
  - Hur är den implementerad

Mappningar

- Arrayer
  - Mappning från heltal (0–n) till värden
  - ArrayList<String> – mappning från heltal till strängar
- Hashtabeller
  - Mappning från en typ till en annan
  - HashMap<String,String> – mappning från strängar till strängar
  - HashMap<Person,Integer> – mappning från klassen person till heltal

Hashtabeller – terminologi

- Mappning från **nyckel** till **värde**
- Exempel:
  - Frekvensordlista
    - Nyckel: ord (String)
    - Värde: frekvens (Integer)
    - HashMap<String,Integer>
  - Mappa från ordform till lemma
    - Nyckel: ordform (String)
    - Värde: lemma (String)
    - HashMap<String,String>

Javaklasser

- Det finns flera klasser för hashtabeller i Java
- Osorterade hashtabeller
  - HashMap
  - Hashtable
- Specialiserade hashtabeller
  - TreeMap (sorterad på nycklarna)
  - ...

Javaklasser

- Det finns flera klasser för hashtabeller i Java
- Interface: Map
- Osorterade hashtabeller
  - HashMap
  - Hashtable
- Specialiserade hashtabeller
  - TreeMap (sorterad på nycklarna)
  - ...

Parentes – gränssnitt (Interface)

- Klass
  - Kan ha instanser
  - Alla metoder är implementerade
  - Kan vara superklass

Parentes – gränssnitt (Interface)

- Klass
  - Kan ha instanser
  - Alla metoder är implementerade
  - Kan vara superklass
- Abstrakt klass
  - Kan inte ha instanser
  - Kan ha oimplementerade och implementerade metoder
  - Kan vara superklass

Parentes – gränssnitt (Interface)

- Klass
  - Kan ha instanser
  - Alla metoder är implementerade
  - Kan vara superklass
- Abstrakt klass
  - Kan inte ha instanser
  - Kan ha oimplementerade och implementerade metoder
  - Kan vara superklass
- Interface (gränssnitt)
  - Kan inte ha instanser
  - Har enbart oimplementerade metoder
  - Andra klasser kan **implementera** godtyckligt många gränssnitt

Parentes – gränssnitt (Interface)

- Klass
  - Kan ha instanser
  - Alla metoder är implementerade
  - Kan vara superklass
- Abstrakt klass
  - Kan inte ha instanser
  - Kan ha oimplementerade och implementerade metoder
  - Kan vara superklass
- Interface (gränssnitt)
  - Kan inte ha instanser
  - Har enbart oimplementerade metoder
  - Andra klasser kan **implementera** godtyckligt många gränssnitt
  - Ett alternativ till multipelt arv, som inte finns i Java

Använd HashMap

- Skapa hashtabell

```
HashMap<K,V> name =
                new HashMap<K, V>(int capacity);
```

- Lägg till nytt nyckel-värde par

```
name.put(K key, V value);
```

- Hämta värde

```
V value = name.get(K key);
returnerar noll om värdet ej finns
```

- Finns en nyckel?

```
name.containsKey(K key);
```

Exempel: öka värdet i en frekvenslista

```
HashMap<String,Integer> freqList =
          new HashMap<String,Integer>();
\\lägg till värden i tabellen
```

Lägg till 1 för värdet för nyckeln s

Alternativ 1:

```
if (freqList.containsKey(s)) {
    freqList.put(s, freqList.get(s)+1);
} else {
    freqList.put(s,1);
}
```

Exempel: öka värdet i en frekvenslista

```
HashMap<String,Integer> freqList =
          new HashMap<String,Integer>();
\\lägg till värden i tabellen
```

Lägg till 1 för värdet för nyckeln s

Alternativ 2:

```
Integer count = freqList.get(s);
if (count == null) {
    freqList.put(s,1);
} else {
    freqList.put(s,count+1);
}
```

Lab 1

- Huvudsakligen om reguljära uttryck
- Även övning på automater, läsning/skrivning, hashtabeller
- Språkteknologrelaterade uppgifter, ta ut information och ta fram statistik från texter
- Tre delar
  1. Matcha och bearbeta rå text
  2. Ta fram frekvenstabeller för olika aspekter ur taggad text
  3. Ta fram statistik från text: meningslängd, ordlängd, vanliga ord, etc
- Givet testprogram för del 1. I övrigt får ni skriva och strukturera er egen kod
- Deadline: tisdag 17/9

Labbrapporter

- Vad ni ska lämna in framgår i respektive labinstruktion
- Allt mailas till Sara, zippat (se labhemsida)
- Alltid med: lämna in all kod
  - Använd bra namn på klasser, metoder, variabler
  - Indentera koden korrekt
  - Följ kodstandarden
- Övriga delar i lab 1
  - Testkörning för era program
  - Finita automater för övning 1.1 och 1.2
- I framtida labbar
  - Skriftlig diskussion – inga formella rapporter, men skriv tydligt och korrekt. Lämna in som pdf. Skriv namn!
  - UML-diagram
  - Skärmdump

Nästa vecka

- Tema: sökning och sortering
- Två föreläsningar
  - Fö 1: Sökning och sortering
  - Fö 2: Generics, interface, stackar, köer mm (även inför lab 3)
- Lab 2 om sökning och sortering

Jobba själv

- Labbar
  - Lab 1!
  - Lab 0 om den ej är klar
- Gör progammeringsövningar
  - Från boken
  - Förra årets lab 2 (mer om finita automater)
- Läs om veckans område
  - Material om reguljära uttryck och finita automater från kurshemsidan
  - Läsning/skrivning
  - Hashtabeller
  - Javas API för klasser vi pratat om idag!
- Läs inför nästa vecka
  - Sökning
  - Sortering