



## Programmering för Språkteknologer II

Markus Saers  
`markus.saers@lingfil.uu.se`  
Rum 9-2040  
`stp.lingfil.uu.se/~markuss/ht09/pst2`



## Innehåll

- Reguljära uttryck
- Gott och blandat
  - `break / continue`
  - `i++ / ++i`
- Lokalisering
  - `java.util.Locale`
  - `java.text.Collator`



## Reguljära uttryck

- Ett kompakt sätt att beskriva en ändlig automat
- Byggs upp med tecken (övergångar) som kombineras på olika sätt
- Ett reguljärt uttryck kan omvandlas till en automat (eller tolkas som en automat)
  - Kan avgör ifall en sträng tillhör språket som beskrivs
  - Kallas matching



## Reguljära uttryck

- Varje tecken motsvarar en automat
- Automater kan kopplas ihop på olika sätt
- Serielt (konkatenering)
  - "a" och "b" → "ab"
- Parallellt
  - "a" och "b" → "[ab]"



## Teckenklasser i Java

- Tecken inom hakparenteser (unionen)
  - `[ab12]`
- Teckenområden inom hakparenteser
  - Definieras med start- och slutpunkter i teckentabellen
  - `[a-zAÖA-ZÄÖ]` [0-9]
- Negationer av ovan
  - `[^aeiouääö]` (alla tecken som inte är vokaler)
- Alla tecken inom två klasser (snittet)
  - `[a-zA-Z][aeiouääö]` (alla vokaler)
  - `[a-zA-Z][^aeiouääö]` (alla bokstäver utom vokalerna)



## Teckenklasser

- Fördefinierade teckenklasser
  - `.` ("punkt", vilket tecken som helst)
  - `\d` ("digits", motsvarar [0-9])
  - `\w` ("word" i Java, motsvarar [a-zA-Z\_0-9])
  - `\s` ("space")
  - `\D \W \S` (negationer av ovanstående)



## Kvantifierare

- Hur många gånger ska tecknet/teckenklassen X förekomma?
  - X? En eller noll
  - X\* Noll eller fler
  - X+ En eller fler (XX<sup>+</sup>)
  - X{n} Exakt n
  - X{n,} Minst n
  - X{n,m} Minst n och högst m
- Matchingsstrategier
  - Girig (greedy, som ovan)
  - Försiktig (reluctant, modifieras med ?)
  - Possessiv (possessive, modifieras med +)



## Matchingsstrategier

- Girig
  - Försöker matcha så långa sjok som möjligt
- Försiktig
  - Försöker matcha så korta sjok som möjligt
- Possessiv
  - Lägger beslag på allt den matchar
  - Kan göra det svårt att hitta matchningar



## Exempel

- Vi försöker utföra matchning på "apa"
- "`\w*a`" lyckas
  - "`\w*`" matchar "ap"
  - "a" matchar "a"
- "`\w*a?`" lyckas
  - "`\w*a?`" matchar ""
  - "a" matchar "a"
- "`\w*+a`" misslyckas
  - "`\w*+`" matchar och lägger beslag på "apa"
  - "a" matchar inte ""



## Intressanta metoder med reguljära uttryck som argument

- `String.replaceAll`
  - Ger en `String`
  - Tar ett reguljärt uttryck och en ersättningssträng
  - Ersätter alla matchningar av det reguljära uttrycket med ersättningssträngen
  - "`aaa.replaceAll("a", "b")`" = "bbb"
  - "`aaa.replaceAll("a+", "b")`" = "b"
- `String.split`
  - Ger en `String[]`
  - Tar ett reguljärt uttryck
  - Ger en vektor med strängarna som avdelas av det reguljära uttrycket
  - "`en fras.split("\\s")`" = ["en", "fras"]



## Skillnaden mellan olika matchningsstrategier

```
String sucWord =
"<w n=71>gäller<ana><ps>VB<m>PRS AKT<b>gälla</w>";
sucWord.replaceAll("<.*?>", "\t");
sucWord.replaceAll("<.*?>", "\t");
sucWord.replaceAll("<.*+>", "\t");
```

- Girig:
 

```
"\t";
```
- Försiktig
 

```
"\tgäller\t\tVB\tPRS AKT\tgälla\t";
```
- Possessiv
 

```
"<w n=71>gäller<ana><ps>VB<m>PRS AKT<b>gälla</w>";
```



## Skillnaden mellan olika matchningsstrategier

```
String sucWord =
"<w n=71>gäller<ana><ps>VB<m>PRS AKT<b>gälla</w>";
sucWord.replaceAll("<.*>", "\t");
sucWord.replaceAll("<.*?>", "\t");
sucWord.replaceAll("<.*+>", "\t");
```

- Girig:
 

```
"\t";
```
- Försiktig
 

```
"\tgäller\t\tVB\tPRS AKT\tgälla\t";
```
- Possessiv
 

```
"<w n=71>gäller<ana><ps>VB<m>PRS AKT<b>gälla</w>";
```



## Frågor?



## Gott och blandat

- Två satser jag tror vi missat...
  - `break`
  - `continue`
  - Kan användas i loopar
    - `break` avbryter loopen
    - `continue` avbryter den aktuella iterationen av loopen
- Prefix/postfix inkrementering
  - Vad är skillnaden mellan `i++` och `++i`?



## Exempel

```
for (int i = 0; i < 100; i++) {
    if (i % 3 == 0) continue;
    System.out.println(i);
}

while (true) {
    try {
        // Do something risky
    } catch (Exception e) {
        break;
    }
}
```



## Exempel

```
int i = 0;
while (i < 100) {
    System.out.println(i++);
}

int i = 0;
while (i < 100) {
    System.out.println(++i);
}
```



## Exempel

```
int i = 0;
while (i++ < 100) {
    System.out.println(i);
}

int i = 0;
while (++i < 100) {
    System.out.println(i);
}
```



## Frågor?



UPPSALA  
UNIVERSITET

## Lokalisering

- **java.util.Locale**
  - Beskriver en språkmiljö
  - Skapas med
    - En språkkod
    - En ladskod
    - En variantsträng
  - Ett antal fördefinierade finns
- **java.text.Collator**
  - Beskriver strängsorteringsordningen för ett **Locale**-objekt
  - Implementerar **Comparator<Object>**



UPPSALA  
UNIVERSITET

## Enkelt exempel

```
public static void main(String[] args) {
    Locale loc = Locale.GERMANY;
    // Locale loc = new Locale("sv");

    TreeSet<String> ts =
        new TreeSet<String>(
            Collator.getInstance(loc)
        );

    Scanner scan = new Scanner(System.in);
    while (scan.hasNext()) {
        ts.add(scan.next());
    }

    for (String s : ts) {
        System.out.println(s);
    }
}
```