DD2385

Programutvecklingsteknik

Bilder om BNF-syntax och XML

Innehåll

- Syntax och BNF
- Syntaxkontroll med recursive descent
- XML, DTD

Syntax enligt Backus-Naur-Form (BNF)

- Beskriver grammatik för hur språk byggs upp
- Slutsymboler, <Icke-slutsymboler> och regler

Exempel: årstid

```
<Årstid> ::= Vinter|Vår|Sommar|Höst
```

Exempel: Binära tal utan onödiga inledande nollor

```
<Binsiffra> ::= 0 | 1
<Tal> ::= |<Binsiffra><Tal>
<Bintal> ::= 0 | 1 <Tal>
```

Naturligt språk, litet exempel

Ett språk som består av satserna

JAG VET	JAG TROR
DU VET	DU TROR

definieras av syntaxen

```
<Sats> ::= <Subjekt><Predikat>
<Subjekt> ::= JAG|DU
<Predikat> ::= VET|TROR
```

Meningar som

```
JAG VET ATT DU TROR ATT JAG VET OCH
JAG TROR ATT DU VET ATT JAG TROR
```

kan nu definieras

```
<Mening> ::=
<Sats>|<Sats><Konjunktion><Mening>
<Konjunktion> ::= ATT|OCH
```

- Ett program som kollar syntax enligt BNF kan kallas automat
- Ett sådant program är relativt enkelt att skriva utifrån BNF-reglerna
- Tekniken kallas *recursive descent*

- Skriv en metod per grammatiksymbol:
  `lasMening()` `lasSats()`
  `lasSubj()` `lasPred()`
  `lasKonj()`
- Programmet använder klassen Scanner för läsning från terminalfönster/fil och en egen felklass SyntaxException.

Syntaxkontroll - programstruktur

```
class Syntaxkoll {
   static Scanner filscan;

   class SyntaxException
      extends Exception {...}

   boolean finnsMer()
      throws SyntaxException {...}

   void lasMening() throws SyntaxException{...}
   void lasSats() throws SyntException{...}
   void lasSubj() throws SyntException{...}
   void lasPred() throws SyntException{...}
   void lasKonj() throws SyntException{...}

   public static void main (..) {...}
}
```

```
import java.util.*;
class Syntaxkoll {
   static Scanner filscan;

   class SyntaxException extends Exception {
      SyntaxException (String s) {
         super(s);
      }
   }

   boolean finnsMer() throws SyntaxException {
      if (!filscan.hasNext())
         throw new SyntaxException
                    ("Ovantat slut: ");
      return true;
   }
```

fortsättning följer

```
   void lasSubj() throws SyntaxException {
      finnsMer();
      String ord = filscan.next();
      if (ord.equals("JAG")) return;
      if (ord.equals("DU")) return;
      throw new SyntaxException
          ("Fel subjekt: " + ord);
   }

   void lasPred() ... //kollar VET och TROR
   void lasKonj() ... //kollar OCH och ATT
```

Metoderna `lasSubj`, `lasPred` och `lasKonj`
är uppbyggda på samma sätt.

fortsättning följer

```
   void lasMening() throws SyntaxException {
      lasSats();
      if (filscan.hasNext()) {
         lasKonj();
         lasMening();
      }
   }

   void lasSats() throws SyntaxException {
      lasSubj();
      lasPred();
   }
```

fortsättning följer

```
   public static void main (String[] u) {

      filscan = new Scanner(System.in);
      Syntaxkoll syntkoll = new Syntaxkoll();

      try {
         syntkoll.lasMening();
         System.out.println("OK");
      }
      catch (SyntaxException e) {
         if (filscan.hasNextLine())
            System.out.println(e + "fore " +
                               filscan.next());
         else
            System.out.println(e);
      }
   }
```

XML

- eXtensible Markup Language
- standard för datautbyte
- Vilket datum är 12/05/07 ?
  - 5 december 2007
  - 12 maj 2007
  - 7 maj 2012
- Med XML behöver man inte tveka

```
<Date>
  <Year>12</Year><Month>05</Month><Day>07</Day>
</Date>
```

XML

```
<Date>
  <Year>12</Year>
  <Month>05</Month>
  <Day>07</Day>
</Date>
```

eller

```
<Date year="2012" month="05" day="07">
</Date>
```

XML - forts.

- XML används för att
  - strukturera
  - lagra
  - transportera

  data oberoende av programspråk
- Egna ord i taggarna
- Beskriver vad data betyder

HTML beskriver hur data ska presenteras

XML - forts.

Små data kan vara *attribut* inne i starttaggen

```
<?xml version="1.0"?>
<Date year="2009" month="05" day="04">
    <Name>Monika</Name>
    <Name>Mona</Name>
    <Moon phase="between"></Moon>
</Date>
```

men attribut bör användas försiktigt,
följande är bättre

XML - forts.

```
<Date>
    <Year>2009</Year>
    <Month>05</Month>
    <Day>04</Day>
    <Name>Monika</Name>
    <Name>Mona</Name>
    <Moon phase="between"></Moon>
</Date>
```

XML - forts.

Attribut används främst för

- identifikation eller
- tolkningsinformation

```
<Item id="A2711"> ...  </Item>
```

```
<Pris curr="EU">19.99</Pris>
```

```
<Pris curr="SEK">189.50</Pris>
```

XML - forts.

Kommentarer:

```
<!-- Det här är en kommentar -->
```

Specialtecken:

<	<
>	>
&	&
'	’
"	"

Välformad XML

- Har rot-element
- Har sluttagg
- Har korrekt nästling
- Har attribut inom ”...”
- Skiljer på gemener och versaler

DTD – beskrivning av XML-dokument

```
<!DOCTYPE Date [
  <!ELEMENT Date (Name*, Moon?)>
  <!ATTLIST Date year CDATA #REQUIRED>
  <!ATTLIST Date month CDATA #REQUIRED>
  <!ATTLIST Date day CDATA #REQUIRED>
  <!ATTLIST Date weekday CDATA #IMPLIED>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Moon EMPTY>
  <!ATTLIST Moon phase (new|between|full)
                              "between">
]>
```

Regler i DTD

E*	0 eller fler
E+	1 eller fler
E?	0 eller 1
(E1|E2|...|eN)	E1 eller E2 eller ...
(E1,E2,...,E3)	E1 och E2 och ...
#PCDATA	Parsad text
CDATA	Icke-parsad text
EMPTY	Tom, barn tillåts ej

Att tolka XML-filer

- BNF-liknande syntax kan användas för XML
- därmed också tekniken *recursive descent*

Syntaxen är ungefär så här

```
Xml ::= Starttagg Innehåll Sluttagg|
      Starttagg Text Innehåll Sluttagg
Innehåll  ::= |Xml Innehåll
Starttagg ::= <Namn Attribut>
Sluttagg  ::= </Namn>
Attribut  ::= |Namn=Värde Attribut
```

Samma Namn i Starttagg och Sluttagg krävs

JTree

- Grafisk komponent som visar trädstruktur
- Behöver en `TreeModel`
- `TreeModel` behöver en trädstruktur av `MutableTreeNode` (båda interface)
- Implementerande klasser är t.ex. `DefaultTreeModel` och `DefaultMutableTreeNode`

Exempel

```
DefaultMutableTreeNode root =
    new DefaultMutableTreeNode("World");
DefaultMutableTreeNode eur =
    new DefaultMutableTreeNode("Europe");
DefaultMutableTreeNode africa =
    new DefaultMutableTreeNode("Africa");

root.add(eur);
root.add(africa);
eur.add
  (new DefaultMutableTreeNode("Sweden"));
africa.add
  (new DefaultMutableTreeNode("Kilimanjaro"));
africa.add
  (new DefaultMutableTreeNode("Tanzania"));
```

Exempel, forts

```
DefaultTreeModel model =
    new DefaultTreeModel(root);

JTree tree = new JTree(model);
add(tree); // Om vi ar i en Container
```

Labb 5 - Läs XML, bygg träd, visa i JTree

```
<?xml version="1.0" encoding="UTF-8"?>
<Biosfar namn="Liv"> ar allt som fortplantar sej
  <Rike namn="Vaxter"> kan inte forflytta sej
    <Division namn="Kryptogamer"> har inte synliga konsdelar
      <Klass namn="Ormbunkar"> har blad och karl
      </Klass>
      <Klass namn="Alger"> ar roda, grona eller bruna
      </Klass>
    </Division>
      ...
  </Rike>
  <Rike namn="Djur"> kan forflytta sej
      ...
  </Rike>
  <Rike namn="Svampar"> ar varken djur eller vaxter
      ...
  </Rike>
</Biosfar>
```

Labb 5 - Läs XML, bygg träd

- EJ krav på att använda tekniken recursive descent
- EJ krav på att alla syntaxfel upptäcks och att Java-Exceptions kastas
- OK att utnyttja filens radindelning och förutsätta att filen är korrekt
- KRAV:
  Kontrollera att starttagg och sluttagg är lika