Der Scannergenerator *JLex*

- JLex ist ein Tool welches aus einer Lex–Spezifikation ein Java–Programm generiert, das den DEA–Simulator implementiert.
- Die Actions der Lex–Spezifikation werden in Java ausprogrammiert.

Die JLex - Spezifikation

```
User Code (Java)
%%
JLex Direktive
%%
Lex--Spezifikation
```

JLex–Direktiven (Anweisungen)

`%class` *name* : definiert den Klassennamen *name* der erzeugten Klasse (default: `Yylex`)

`%function` *name* : definiert den Methodennamen *name*, die das nächste Lexem liest (default: `yylex()`)

`%type` *name* : definiert den Rückgabetyp von der Methode, die das nächste Lexem liest (default: `Yytoken`)

`%{`
Java − Code
`%}` : *Java − Code* wird an den Anfang der erzeugten Klasse kopiert

`%init{`
Java − Code
`%init}` : *Java − Code* wird in den Konstruktor der erzeugten Klasse kopiert.

```
%eof{
Java − Code
%eof}
```
: $Java - Code$ wird ausgeführt bei Erreichen des Dateiendes.

Java–Klasse erzeugen

Das Packet `JLex2.jar` muss in den `CLASSPATH` der Java–Umgebung aufgenommen werden

Aufruf der Scannergenerators:

`java -cp JLex2.jar JLex2.Main Datei` (JLex-Spezifikation)

Dadurch wird die Klasse `Datei.java` erzeugt, die den DEA–Simulator für die JLex-Spezifikation `Datei` enthält

JLex–Spezifikation

Beispiel: `html`–Lexeme

Als Beispiel geben wir eine JLex–Spezifikation für einen kleiner Ausschnitt der Sprache html an.

```
%%

%public
%class browserlexer
%type int
%eofval{
  System.out.println("EOF reached");
  return -1;
%eofval}

a = (a|A)
b = (b|B)
```

```
ws = [ \t\r\n\b\015]+

%%
"<"(h|H)(t|T)(m|M)(l|L)">"  { System.out.println(yytext()); }
"</"(h|H)(t|T)(m|M)(l|L)">" { System.out.println(yytext()); }

"<"{b}">"           { System.out.println(yytext()); }
"</"{b}">"          { System.out.println(yytext()); }

[^\<]+              { System.out.println(yytext()); }
{ws}                { System.out.println(yytext()); }
.                   { System.out.println("FEHLER: "+yytext()); }
```

Java–Klasse für den Aufruf

```
class Main {

  public static void main(String args[])
                    throws java.io.IOException {

    browserlexer b =
        new browserlexer (
                    new java.io.InputStreamReader(System.in));
    while (b.yylex() != -1) {}
    }
}
```

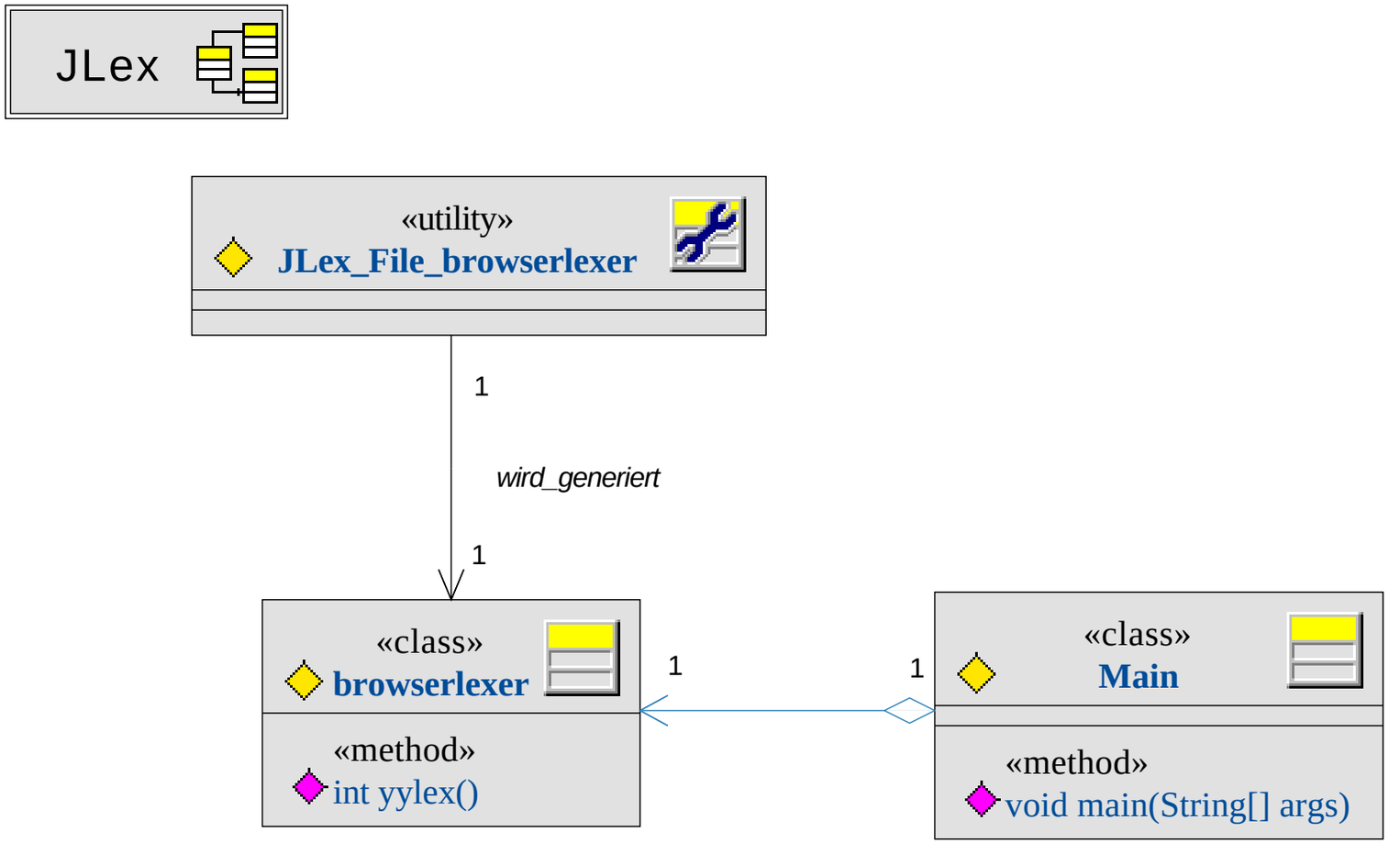
Makefile

```
Main.class: browserlexer.class Main.java
<TAB>   javac Main.java

browserlexer.class: browserlexer.java
<TAB>   javac browserlexer.java

browserlexer.java: browserlexer
<TAB>   java -cp JLex2.jar JLex2.Main browserlexer


clean:
<TAB>   rm *.class browserlexer.java
```


JLex
«utility»
JLex_File_browserlexer
1
wird_generiert
1
«class»
browserlexer
«method»
int yylex()
1
1
«class»
Main
«method»
void main(String[] args)

Tokens

- Lexer/Scanner fasst die Strings zusammen, die an einer Stelle als Terminalsymbol in einer Grammatik stehen dürfen:

  Lexeme mit gleicher Bedeutung: z.B.: [`<html>`, `<HTML>`]

  Lexeme unterschiedlicher Bedeutung: z.B.: Text zwischen Tags

Ziel: Definition einer Datenstruktur, die diesem Rechnung trägt.

Klasse yyTokenclass

```
class yyTokenclass {
   public int tokennr;
   public Object value;

   yyTokenclass () {
      this.tokennr=-1;
   }
   yyTokenclass (Object o) {
      this.value = o;
   }
}
```

Alle Tokens erben von `yyTokenclass`.

JLex–Spezifikation mit Tokens

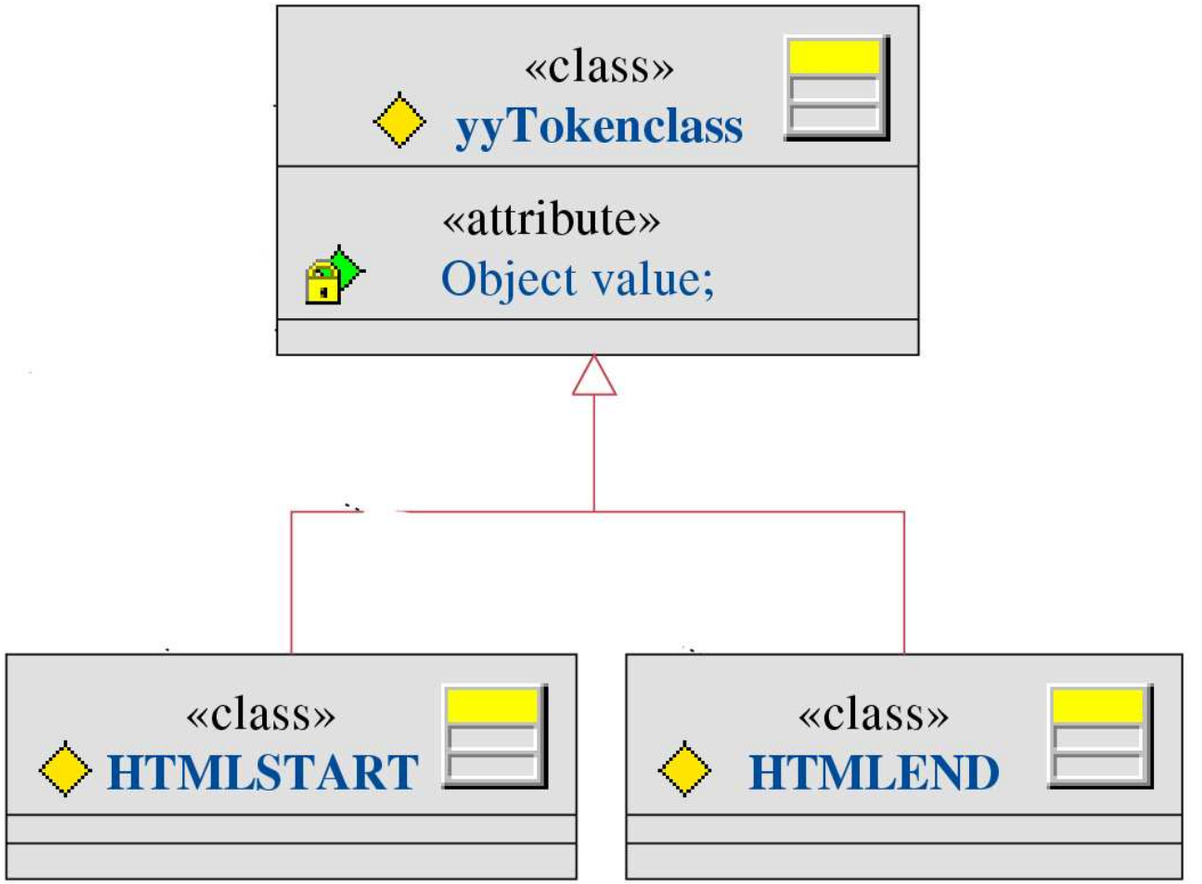
```
%%
%class browserlexer
%eofval{
  return new EOF();
%eofval}

ws = [ \t\r\n]+

%%
"<"(h|H)(t|T)(m|M)(l|L)">" { return new HTMLSTART(yytext());}
"</"(h|H)(t|T)(m|M)(l|L)">" { return new HTMLEND(yytext());}
[^\<]+          { System.out.println(yytext());
                          return new WORT(yytext());}
```

Wo kommen die Tokens (Java–Klassen) her?

Token–Klasse



jay–Tool (jaooy)

Eingabe (Grammatik)

- Namen der Tokenklassen (Terminale)
- Produktionen

Ausgabe

- Kellerautomat, der die Sprache der eingegebenen Grammatik erkennt als Java–Programm

jay–Spezifikation

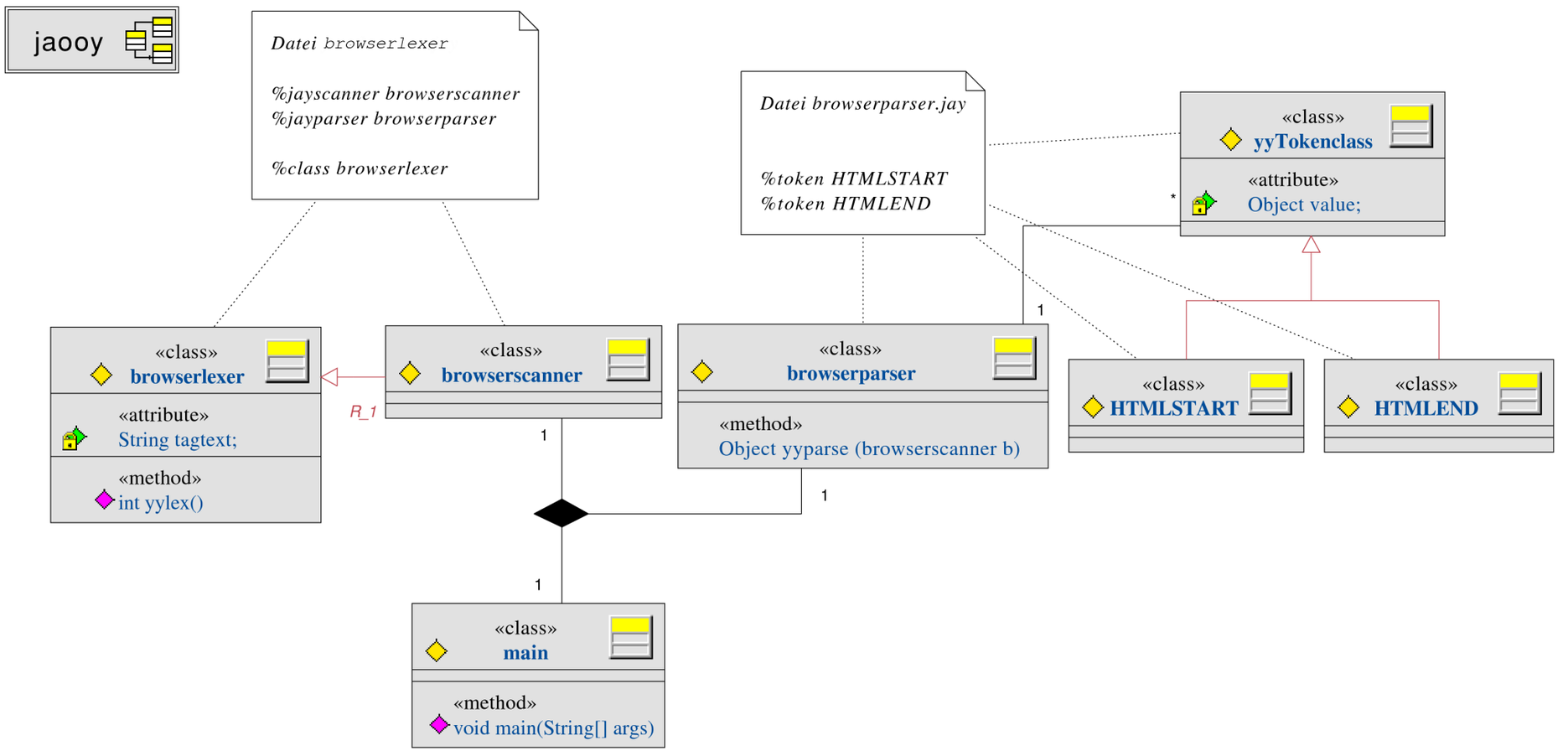
```
%{
class browserparser {
%}

%token HTMLSTART
%token HTMLEND
...

%%
S : HTMLSTART Head Body HTMLEND { }
Head : HEADSTART  Title   HEADEND { }
...

%%
}
```

Klassendiagramm JLex und jay



JLex–Spezifikation mit Klassendeklarationen

```
%%
%jayscanner browserscanner
%jayparser browserparser
%class browserlexer
%eofval{
  return new EOF();
%eofval}

ws = [ \t\r\n]+

%%
"<"(h|H)(t|T)(m|M)(l|L)">" { return new HTMLSTART(yytext());}
"</"(h|H)(t|T)(m|M)(l|L)">" { return new HTMLEND(yytext());}
[^\<]+          { System.out.println(yytext());
                          return new WORT(yytext());}
```

Java–Klasse für den Aufruf

```
public class main {
    public static void main (String [] args) {
        browserscanner scanner =
            new browserscanner
                    (new java.io.InputStreamReader (System.in));
        browserparser parser = new browserparser() ;
        try {
            parser.yyparse(scanner);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Makefile

```
Main.class: Main.java yyTokenclass.class browserscanner.class
        javac Main.java

yyTokenclass.class: browserparser.java
        javac browserparser.java

browserscanner.class: browserlexer.java
        javac browserlexer.java

browserlexer.java: browserlexer
        java -cp JLex2.jar JLex2.Main browserlexer

browserparser.java: browserparser.jay skeleton.jaooy
        jaooy -v browserparser.jay < skeleton.jaooy > browserparser.java
clean:
        rm -f *.class browserlexer.java browserparser.java
```

Aufgabe

Entwickeln Sie eine Mini–html–Parser als Grundlage für einen Browser.

- Definieren Sie eine Grammatik für Mini-html.
- Ergänzen Sie das JLex– und das Jay–File dementsprechend.