
DHBW
Stuttgart

Organisatorisches

- Ablauf der Volesung gliedert sich in 3 Teile:
  1. Vorlesung
  2. Übungsblatt (in 2er Gruppen bearbeiten)
  3. Vorstellen/Besprechung der Ergebnisse

- Bearbeiten der Aufgaben mit eigenem Laptop
  - Jede Zweiergruppe sollte mindestens einen Rechner besitzen

Literatur

- Offizielle ANTLR-Website unter: `https://www.antlr.org/`
- *The Definitive ANTLR 4 Reference* von Terence Parr, ISBN: 978-1-93435-699-9
- Readme auf Github:
  `https://github.com/antlr/antlr4/blob/master/doc/index.md`

Entwicklungsumgebung

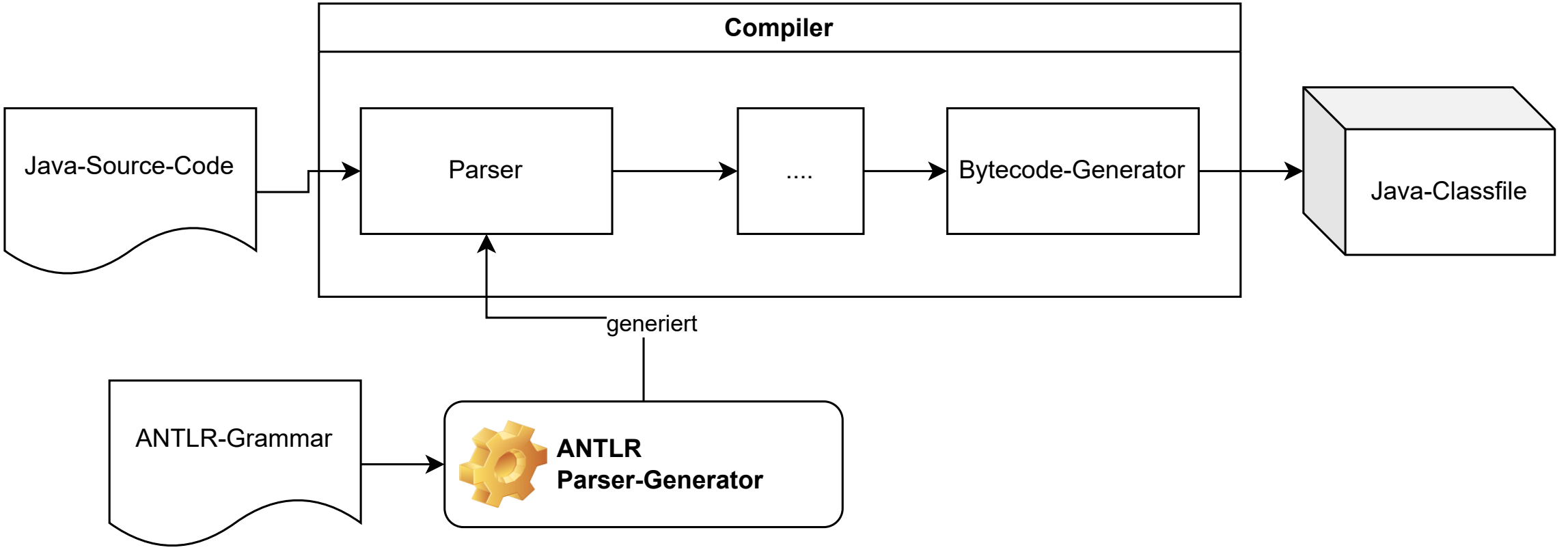
ANTLR-Jar Datei
`https://www.antlr.org/download/antlr-4.13.2-complete.jar` lässt sich zusammen mit jeder Entwicklungsumgebung auf jedem gängigen Betriebssystem nutzen.

Plugin ANTLR lässt sich per Plugin in gängige Java IDEs integrieren:

- **Intellij** `http://plugins.jetbrains.com/plugin/7358?pr=idea`
- **Eclipse** `https://github.com/jknack/antlr4ide`

ANTLR Lab `http://lab.antlr.org` ist eine Webanwendung zum Testen von ANTLR Grammatiken

Parser Generator



Vorzüge von ANTLR

- Unterstützt jede kontextfreie Grammatik
  - Ausnahme: indirekte linksrekursive Regeln
- Trennung von Grammatik und Verarbeitungslogik
  - Programm verarbeitet den geparsten Syntaxbaum. Kein Java-Source Code innerhalb der Grammatik
  - Bessere Integration des SourceCodes in die IDE
  - Leichter anpassbar/erweiterbar
  - Arbeiten mit bekannter Java Syntax
- Gutes Tooling: Plugins für gängige Java IDEs, wie Eclipse und Intellij IDEA
  - Syntax Highlighting
  - Gute Fehlerausgabe

G4-Grammatik: Syntax

- Comments sind gleich wie in Java (`/** //`)
- Tokens/Terminale beginnen mit Großbuchstaben, Non-Terminale mit Kleinbuchstaben
- Literale in Single-Quotes (`'literal'`)
- Regeln bestehen aus Namen für Token, Literale und den Zeichen (`|` `.` `*` `+` `?`)
  - `|` entspricht *Oder*
  - `.` `*` `+` `?` verhalten sich wie bei Regulären Ausdrücken
  - `~[A]` entspricht allen 16-bit wertigen Zeichen außer *A*

Syntax Veranschaulichung

```
//Datei GrammatikBeispiel.g4
//Name der Grammatik (gleich wie Dateiname):
grammar GrammatikBeispiel;
regel1 : Token1 Token2;
Token1 : 'Hallo';
Token2 : [a-z]+;
/*
Grammatik parst Texte der Form:

Hallo nameinkleinbuchstaben
*/
Linefeed: '\r'? '\n';
WS : [ \t]+ -> skip; //Leerzeichen ignorieren
```

G4-Grammatik Beispiel

```
grammar Beispiel2;
s : s '(' s ')' s
  | TEXT?;
TEXT : ~[()]+;
```

- (Gibt es ein äquivalent zu dieser Grammatik als regulären Ausdruck?)

Hinweise

- Keine separate Datei für Lexeme notwendig. Alle Regeln und Lexeme werden in der G4-Grammatik beschrieben.
- Der Lexer nimmt immer das erste in der Grammatik auftauchende Lexem. Allgemeinere Terminale also ans Ende der Grammatik verlegen.

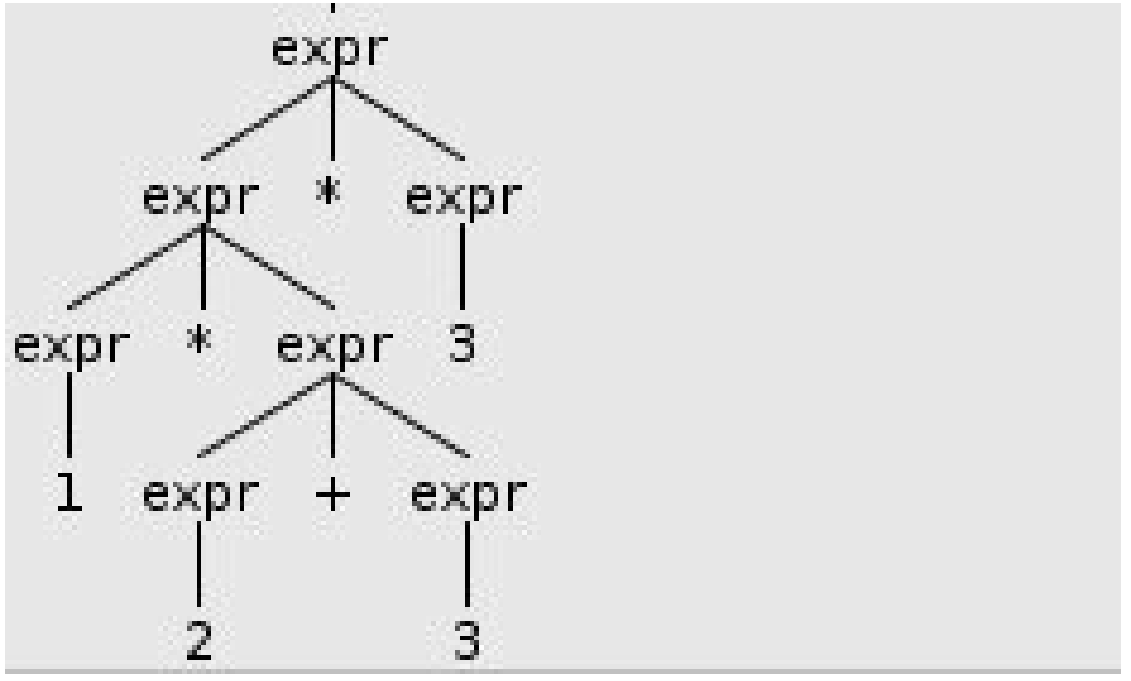
```
//Warum kann diese Grammatik "int 1" nicht parsen?
grammar Number;
number : integer | float;
integer : 'int␣' Integer;
float : 'float␣' Float;

Float : [0-9\.];
Integer : [0-9];
```

- Bei nicht eindeutigen Regeln wird die zuerst spezifizierte Regel bevorzugt
- ANTLR versucht immer das längste Lexem auszuwählen

Falsch:

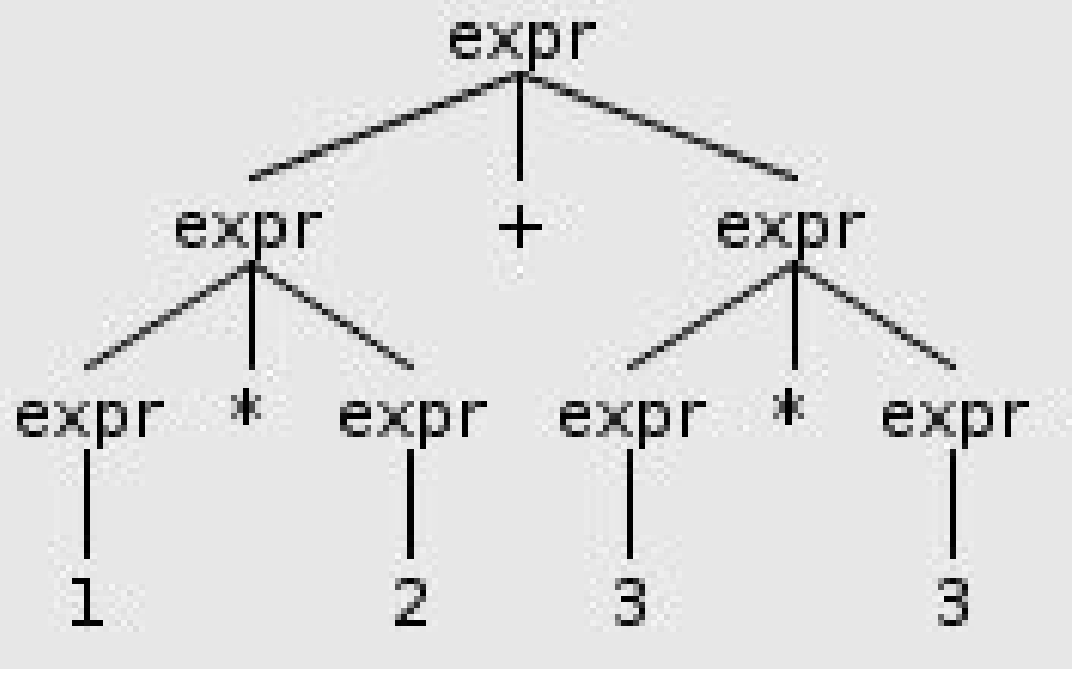
```
expr : expr '+' expr | expr '*' expr | Number;
```



Baum beim Parsen von `1 * 2 + 2 * 3`

Korrekt:

```
expr : expr '*' expr | expr '+' expr | Number;
```



Baum beim Parsen von `1 * 2 + 2 * 3`

Übungsblock 1

Übungsblatt 1: Aufgabe 1 und 2

- Link zum Übungsblatt: `http://www2.ba-horb.de/~stan/%C3%BCbung1.pdf`
- Intellij-Download: `https://www.jetbrains.com/idea/download`
- Download der ANTLR-Java-Library
  `https://www.antlr.org/download/antlr-4.13.2-complete.jar`

Tipp: Installation des ANTLR Plugins in Intellij

- File → Settings → Suche nach *Antlr v4 Grammar Plugin*
  Oder: Download ANTLR-Plugin: `https://plugins.jetbrains.com/plugin/download?pr=idea&updateId=26416`

ANTLR-Library dem Projekt hinzufügen

- antlr-complete.jar ins Projektverzeichnis kopieren
- File → Project Structure → Libraries
- anschließend '+' → Java → antlr-complete.jar auswählen

Beispielimplementierung für Aufgabe 3.b

Grammatik für Expressions (Fehlerhaft)

```
 grammar IntExpression;
s : expr;

expr : expr ADD expr | expr MUL expr
       | expr SUB expr | '(' expr ')'
       | Number;

MUL : '*' ;
ADD : '+' ;
SUB : '-' ;

Number : [0-9]+;
WS : [ \t\r\n] -> skip;
```

Aufbau des ParseTrees

- ANTLR generiert zu jeder Regel in der Grammatik eine eigene Klasse
- Der Aufbau der Klasse wird direkt von dieser Regel abgeleitet
- Der ParseTree baut sich aus diesen Klasse auf

Beispiel 1:

```
regel : unterRegel1 | unterRegel2
```

wird durch die Klasse `RegelContext` repräsentiert, welche folgende Methoden enthält:

```
UnterRegel1Context unterRegel1();
UnterRegel2Context unterRegel1();
```

Aufbau des ParseTrees

Beispiel 2:

```
regel : unterRegel+
```

wird durch die Klasse `RegelContext` repräsentiert, welche folgende Methoden enthält:

```
List<UnterRegelContext> unterRegel();
```

Methode zum Einlesen und Parsen eines Strings aus System.in

```
public static void main(String[] args) throws Exception {
    CharStream input = CharStreams.fromString("100+2*3");
    IntExpressionLexer lexer = new IntExpressionLexer(input);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    IntExpressionParser parser = new IntExpressionParser(tokens);
    IntExpressionParser.StartContext tree = parser.start(); //Parsen

    ExpressionCalculator calc = new ExpressionCalculator();
    int ergebnis = calc.calculate(tree.expr()); // initiate walk of tree with listener
    System.out.println(ergebnis);
}
```

ExprAdapter:

```
class ExpressionCalculator{
        int calculate(IntExpressionParser.ExprContext ctx){
            if(ctx.MUL()!=null){
                return this.calculate(ctx.expr(0)) * this.calculate(ctx.expr(1));
            }else
            if(ctx.ADD()!=null){
                return this.calculate(ctx.expr(0)) + this.calculate(ctx.expr(1));
            }else
            if(ctx.Number()!=null){
                return Integer.parseInt(ctx.Number().toString());
            }
            throw new RuntimeException("Unknown Expression");
        }
    }
```

Einlesen von Texten

- **CharStreams** Doku: `http://www.antlr.org/api/Java/org/antlr/v4/runtime/CharStreams.html`
- Factory zur Generierung von `CharStream`'s
- Kann String, InputStream, File und viele weitere Eingabeformate in das von ANTLR benutzte CharStream konvertieren

```
//Beispiel:
CharStream input = CharStreams.fromFileName("/pfad/zu/Datei");
```

Übungsblock 2

Übungsblatt 1: Aufgabe 3