Compiler Construction

Winter semester 2016/2017

UNI FREIBURG

University of Freiburg

Matthias Keil

University of Freiburg

14. Oktober 2016

Outline

Time and Place

UNI FREIBURG

Lecture

- **Monday, 2pm - 4pm**
  Room SR 01-016, Building 101
- **Thursday, 2pm - 3pm**
  Room SR 01-016, Building 101

Exercise

- **Thursday, 3pm - 4pm**
  Room SR 01-016, Building 101

UNI FREIBURG

Assistant

Matthias Keil

Address Room 00-013, Building 079

Email keilr@informatik.uni-freiburg.de

Phone +49 (0)761 203 8060

Final Grade

- No oral or written exam.
- Final grade is calculated from the exercise program.

Compiler Project

- Compiler from MiniJava to MIPS.
- Six exercise sheets.
- 2-3 weeks per exercise sheet, depending on the difficulty.
- Each sheet focuses on one part of a compiler.
- *Java knowledge is required.*

Written Homework

- 2-4 pages.
- Possible topics are given by us.

Registration for Participation

UNI FREIBURG

Fill out the form until Thursday, 20.10.2016 12:00 (midnight)

Form `https://goo.gl/forms/8Q61RZSQMv51Wi7y2`

- *name, prename*
- *email address*
- *tf username (pool account)*

UNI FREIBURG

Repository

Link `https://proglang.informatik.uni-freiburg.de/svn/cc` + SVNNUMBER

SVNNUMBER $\in \{00 \ldots 25\}$

WWW Password

Link `https://support.informatik.uni-freiburg.de/cgi/support/fawmgr.cgi?wpassword:en`

Submission Guidelines

Submission

- Submit your solution to the repository.
- Create one folder (e.g. *exercise1*) for each exercise sheet.
- Submissions consist of an **executable Jar file with source code** inside and a **report**.
- *Submission instructions are given on each exercise sheet.*

Deadline

- **Thursday, 12:00 (noon)**.
- *Late submissions will not be accepted.*

Submission Guidelines (cont'd)

UNI FREIBURG

Implementation

- Efficient and clear code is required.
- You are strongly encouraged to test your solution with the provided test cases.
- Provide your source code with comments.

Submission Guidelines (cont'd)

Comments

- Clear and understandable style is required.
- **Comments do not explain it the syntax**.
- Comments clarify its intension.

Bad style

```
1    // declare name
2    // concatenates uid and nr
3    String name = uid + nr;
```

Submission Guidelines (cont'd)

Report

- Pdf file (e.g. report_exercise1.pdf) with a formal description.
  - Conception
  - Structure
  - ...
- Limited to one (two) page(s) per exercise.
- Either in German or in English.
- Clear and understandable style is required.

Submission Guidelines (cont'd)

UNI FREIBURG

The exercises have to be done by yourself!

- Team work is not allowed.
- You are not allowed to copy source code.

UNI FREIBURG

Tools you need to know/learn

- Java $\geq$ 1.6
- Eclipse $\geq$ 4.2 (other IDEs: you're on your own)
- SableCC 3.6
- LaTeX (or anything else)
- **See exercise page for detailed installation instructions.**

Tools you will use without knowing

- Ant
- Checkstyle

Summing the Elements of a List

```
1 interface List {}
2
3 class Nil implements List {}
4 class Cons implements List {
5   int head;
6   List tail;
7 }
```

1. Approach: InstanceOf and Type Casts


```
1 List l;
2 int sum = 0;
3 boolean proceed = true;
4 while(proceed){
5   if (l instanceof Nil)
6     proceed = false;
7   else if (l instanceof Cons){
8     sum += ((Cons) l).head;
9     l = ((Cons) l).tail;
10  }
11 }
```

- Classes are not touched.
- Frequent type casts and instanceof checks.

2. Approach: Dedicated Methods

UNI FREIBURG

```
1 interface List {
2   public int sum();
3 }
4 class Nil implements List {
5   public int sum() { return 0; }
6 }
7 class Cons implements List {
8   int head;
9   List tail;
10  public int sum() {
11    return head + tail.sum();
12  }
13 }
```

- No type casts, systematic and object-oriented.
- Need to change classes.

Intension

Represent an operation to be performed on the elements of an object structure. The Visitor pattern lets you define a new operation *without changing the classes* of the elements on which it operates.

Idea

- Distinguish between object structure and the visitor.
- Insert an `accept` method in each class of the object structure.
- For each of these classes, a visitor contains a `visit` method.

Visitor Pattern

```
1  interface List {
2    void accept(Visitor v);
3  }
4
5  class Nil implements List {
6    public void accept(Visitor v) {
7      v.visitNil(this);
8    }
9  }
10 class Cons implements List {
11   int head;
12   List tail;
13   public void accept(Visitor v) {
14     v.visitCons(this);
15   }
16 }
```

Visitor Pattern (cont'd)

UNI FREIBURG

```
1 interface Visitor {
2   void visitNil(Nil l);
3   void visitCons(Cons l);
4 }
5 class SumVisitor implements Visitor {
6   int sum;
7   public void visitNil(Nil l) {}
8   public void visitCons(Cons l) {
9     sum += l.head;
10    l.tail.accept(this);
11  }
12 }
13 ...
14 SumVisitor sv = new SumVisitor();
15 l.accept(sv);
16 System.out.println(sv.sum);
```

Visitor Pattern (cont'd)

The visitor pattern gives you:

- New functionality without recompiling the object structure!
- Related operations are structured together.
- Visitors can accumulate (and also encapsulate) state.

But:

- All classes must have an accept method.
- Adding new classes to the object structure is nasty.

Be Careful!

The visit methods describe actions **and** access to subobjects.

What is SableCC?

- Open-source parser generator for Java.
- `http://sablecc.org`.
- Generates LALR(1) parsers.
- Featuring: lexer, parser, nodes/ast, analysis/visitors

A specification for SableCC

Grammar

- Package *package-name*;
- Helpers *id* = *regexp*;
- Tokens *id* = *regexp*;
- Ignored Tokens *token1*,...,*tokenN*;
- Productions (simplified)
  id = {*altname*} *elem** —— ... ;
  with *elem* = *[id]: id* (+—*—?)

A specification for SableCC (cont'd)

Example

```
1  Package simpleAdder;
2
3  Tokens
4  l_par  = '(';
5  r_par  = ')';
6  plus   = '+';
7  number = ['0'..'9'];
8
9  Productions
10
11 exp = {constant} number
12 | {add} addition;
13
14 addition = l_par [left]:exp plus [right]:exp
       r_par;
```

A specification for SableCC (cont'd)

Generated files

```
1 abstract class Node {}
2 abstract class PExp extends Node{}
3 class AConstantExp extends PExp {
4   TNumber getNumber(){...}  ...
5 }
6 class AAddExp extends PExp {
7   PAddition getAddition(){...}  ...
8 }
9 class AAddition extends PAddition {
10  TLPar getLPar() {...}
11  PExp getLeft() {...}
12  ...
13 }
```

Visitor Pattern in SableCC (cont'd)

Generated Files

```
1 class DepthFirstAdapter extends AnalysisAdapter
      {
2   void caseXxx(Xxx node) {
3     inXxx(node);
4     node.getYyy.apply(this); // first child of
          Xxx
5     node.getZzz.apply(this); // second child of
           Xxx
6     outXxx(node);
7   }
8   ...
9 }
```

Important

UNI FREIBURG

Do **not**

- Modify any generated files!
- Submit any homework late!
- Copy anyone's homework!
- Panic! Ask for help!

Do

- Comment your submissions!
- Start early on the assignments!
- Consult manuals, tutorials, our forum and the homepage!
- Have fun!

Questions ?