Compiler Construction 2010/2011: Exercises

Konrad Anton

October 20, 2010

Outline

1. Organization
2. Tools
3. Visitor Pattern
4. SableCC

Organization

Organization

Who, where, when?

- Konrad Anton, anton@informatik.uni-freiburg.de
- Office hours: Thu 10-11 in Building 079, Room 013
- Lab session/Exercises: Wed 17-18

Exercises

The Project

A compiler from MiniJava to MIPS.

- Each sheet focuses on one part: Parser, Typechecker, . . .
- Approximately six sheets
- Points per sheet and due dates: varies by difficulty
- Sheet 1: warmup exercises, due 2010-10-27, 5% of total points

Exam and final grade

- You will need 50% of points on exercises to be admitted to final exam!
- Oral examination
- Alternative: Grade based on project (50 % corresponding to 4.0)

Tools

Tools

Tools you need to know/learn

- Java ($\geq$ 1.5)
- Eclipse (other IDEs: you're on your own)
- SableCC 3.2
- LaTeX (or anything else for high-quality type-system typesetting)

Tools you will use without knowing

- Ant
- Checkstyle
- See tools page for installation instructions.

Visitor Pattern

Motivation (Palsberg and Jay, The Essence of the Visitor Pattern, 1998)

Summing the elements of a list

```
interface List {}

class Nil implements List {}
class Cons implements List {
  int head;
  List tail;
}
```

1. Approach: InstanceOf and Type Casts

```java
List l;
int sum = 0;
boolean proceed = true;
while(proceed){
  if (l instanceof Nil)
    proceed = false;
  else if (l instanceof Cons){
    sum += ((Cons) l).head;
    l = ((Cons) l).tail;
  }
}
```

- Classes are not touched.
- ... but frequent type casts and instanceof! :(

2. Approach: Dedicated Methods

```
interface List {
  public int sum();
}
class Nil implements List {
  public int sum() { return 0; }
}
class Cons implements List {
  int head;
  List tail;
  public int sum() { return head + tail.sum(); }
}
```

- No type casts, systematic and object-oriented.
- ... but frequent re-compilation and changing of classes! :(

3. Approach: Visitor Pattern (Gamma et al., Design Patterns, 1995)

Intent

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 `visitXXX` method.

Visitor Pattern

```
interface List {
  void accept(Visitor v);
}

class Nil implements List {
  public void accept(Visitor v) {
    v.visitNil(this);
  }
}
class Cons implements List {
  int head;
  List tail;
  public void accept(Visitor v) {
    v.visitCons(this);
  }
}
```

Visitor Pattern

```
interface Visitor {
  void visitNil(Nil x);
  void visitCons(Cons x);
}
class SumVisitor implements Visitor {
  int sum;
  public void visitNil(Nil x) {}
  public void visitCons(Cons x) {
    sum += x.head;
    x.tail.accept(this);
  }
}
...
SumVisitor sv = new SumVisitor();
l.accept(sv);
System.out.println(sv.sum);
```

Visitor Pattern - Summary

The visitor pattern gives you..

- New methods/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.

Careful!

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

SableCC

SableCC

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

Parts

- 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

Example

```
Package simpleAdder;

Tokens
  l_par  = ’(’;
  r_par  = ’)’;
  plus   = ’+’;
  number = [’0’..’9’];

Productions
  exp = {constant} number
      | {add} addition;
  addition = l_par [left]:exp plus [right]:exp r_par;
```

A specification for SableCC

Generated files

```
/*     exp = {constant} number | {add} addition;
  addition = l_par [left]:exp plus [right]:exp r_par; */
abstract class Node {}
/** Superclass of all exp-> right-hand sides  */
abstract class PExp extends Node{}
/** One exp->number right-hand side */
class AConstantExp extends PExp {
  TNumber getNumber(){...}  ...
}
/** One exp->{add}addition right-hand side  */
class AAddExp extends PExp {
  PAddition getAddition(){...}  ...
}
/** one addition->l_par... subtree */
class AAddition extends PAddition {
  TLPar getLPar() {...} // corresponds to l_par
  PExp getLeft() {...} // corresponds to [left]:exp
  ...
}
```

Visitor Pattern in SableCC

Generated files

```java
class DepthFirstAdapter extends AnalysisAdapter {
  void caseXxx(Xxx node) {
    inXxx(node);
    node.getYyy.apply(this); // first child of Xxx
    node.getZzz.apply(this); // second child of Xxx
    outXxx(node);
  }
  ...
}
```

Important

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!