Unparsing (pretty printing)

Prof. Dr. Ralf Lämmel
Universität Koblenz-Landau
Software Languages Team

Language processing *patterns*

1. The *Chopper* Pattern
2. The *Lexer* Pattern
3. **The *Copy/Replace* Pattern**
4. The *Acceptor* Pattern
5. The *Parser* Pattern
6. The *Lexer Generation* Pattern
7. The *Acceptor Generation* Pattern
8. The *Parser Generation* Pattern
9. The *Text-to-object* Pattern
10. The *Parser Generation*^{2} Pattern
11. (The *Text-to-tree* Pattern)
12. (The *Tree-walk* Pattern)
13. **The *Object-to-text* Pattern**

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Remember?

The *Copy/Replace* Pattern

Remember?
The *Copy/Replace Pattern*

- Intent:

  Transform text at the lexical level.

- Operational steps (run time):
  1. Recognize token/lexeme pairs in input.
  2. Process token/lexeme pairs in a stream.
     1. Copy some lexemes.
     2. Replace others.

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Precise copy for comparison

```
public Copy(String in, String out) throws ... {
  Recognizer recognizer = new Recognizer(in);
  Writer writer = new OutputStreamWriter(
        new FileOutputStream(out));
  String lexeme = null;
  Token current = null;
  while (recognizer.hasNext()) {
    current = recognizer.next();
    lexeme = recognizer.getLexeme();
    writer.write(lexeme);
  }
  writer.close();
}
```

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Copy/replace for cutting salaries in half

```
...
    lexeme = recognizer.getLexeme();

    // Cut salary in half
    if (current == FLOAT && previous == SALARY)
       lexeme = Double.toString(
              (Double.parseDouble(
         recognizer.getLexeme()) / 2.0d));

    writer.write(lexeme);
...
```

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Demo

http://101companies.org/wiki/Contribution:**javaLexer**

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

The counterpart for the Text-to-object pattern

The *Object-to-text* Pattern

The *Text-to-object* Pattern

- Intent:

  Map abstract syntax (in objects) to concrete syntax.

- Operational steps (run/compile time):

  This is ordinary OO programming.

  Walk the object model for the AST.

  Generate output via output stream.

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Methods for pretty printing

```
public void ppCompany(Company c, String s) throws IOException {
   writer = new OutputStreamWriter(new FileOutputStream(s));
   write("company");
   space();
   write(c.getName());
   space();
   write("{");
   right();
   nl();
   for (Department d : c.getDepts())
      ppDept(d);
   left();
   indent();
   write("}");
   writer.close();
}
```

Write into OutputStream

Indent

Pretty print substructures

Undo indentation

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Demo

http://101companies.org/wiki/Contribution:**antlrObjects**

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

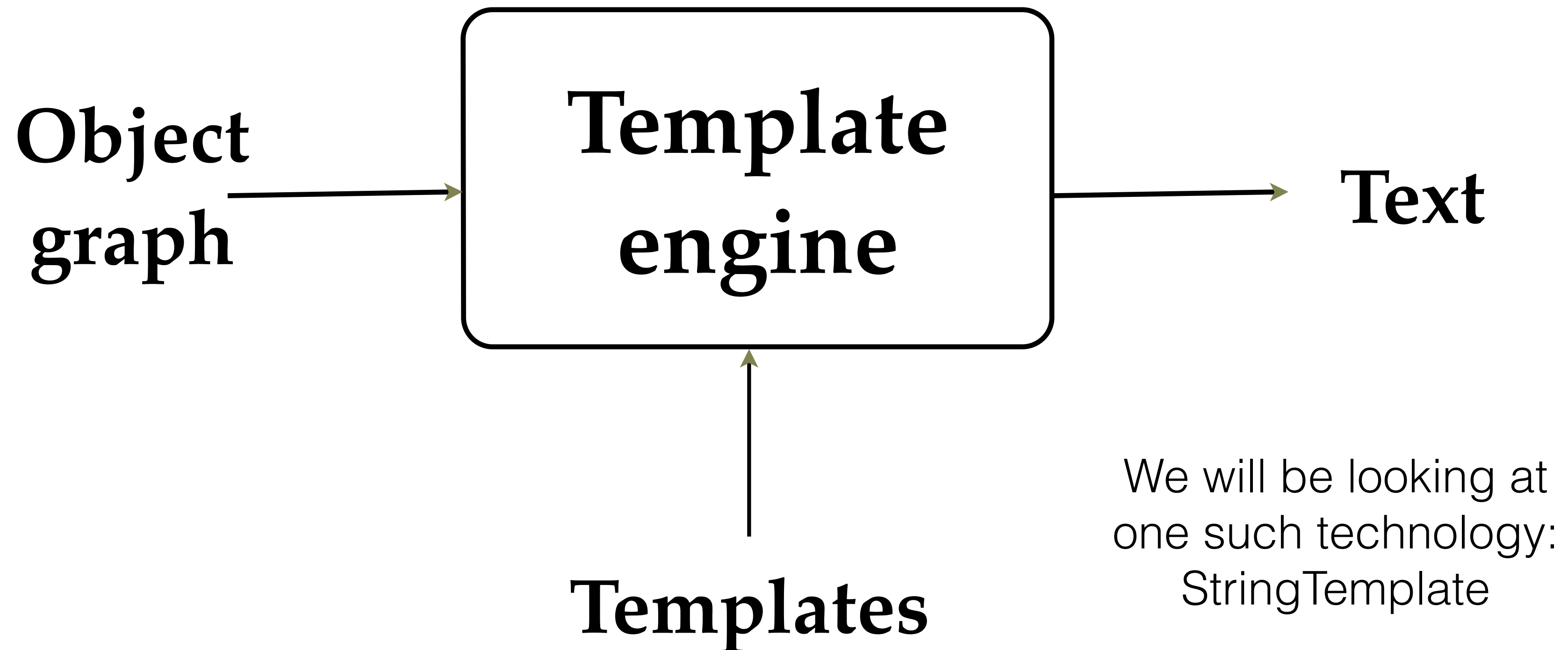
Discussion

- We should assume *Visitor* pattern on object model.
- Challenges:
  - How to ensure that correct syntax is generated?
  - How can we maintain layout of input?
- Alternative technology / technique options:
  - Template processors
  - Pretty printing libraries.

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Pretty printing with templates

Pretty printing with templates



We will be looking at one such technology: StringTemplate

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
company "ACME Corporation" {
  department "Research" {
    manager "Craig" {
      address "Redmond"
      salary 123456.0
    }
    employee "Erik" {
      address "Utrecht"
      salary 12345.0
    }
    employee "Ralf" {
      address "Koblenz"
      salary 1234.0
    }

  }
  department "Development" {
    manager "Ray" {
      address "Redmond"
      salary 234567.0
    }
```

Sample output

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Template

- **Definition**: the description of a mapping of structured data (such as an object graph) to text.

- **Trivia**: Templates may have names so that they can call each other. There is typically expressiveness to take apart input data. Templates can be compared to grammar productions.

- **Example:**

```
company(c) ::="company \"<c.name>\" {
      <c.departments:{x|<x:department()><\n>}>
}"
```

Access top-level departments

Invoke template for each department

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Templates for 101

```
delimiters "<", ">"
```

Delimiters used for the expressions in the templates

```
company(c) ::="company \"<c.name>\" {
    <c.departments:{x|<x:department()><\n>}>
}"

department(d) ::="department \"<d.name>\" {
    <{manager <d.manager:employee()>}>
    <d.employees:{x|employee <x:employee()><\n>}>
    <d.subDepartments:{x|<x:department()><\n>}>
}"

employee(e) ::="\"<e.name>\" {
    address \"<e.address>\"
    salary <e.salary>
}"
```

https://github.com/101companies/101simplejava/blob/master/contributions/javaStringTemplate/src/main/stringtemplate/companyUnparsing.stg

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Invoking the template engine

```
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;

STGroup group = new STGroupFile("company.stg");

ST st = group.getInstanceOf("company");

st.add("c", c);

String result = st.render();
```

Refer to template file

Look up template for type „company“

Add actual company to environment

Invoke engine

https://github.com/101companies/101simplejava/blob/master/contributions/javaStringTemplate/src/main/java/org/softlang/company/features/Unparsing.java

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Demo

101companies.org/wiki/
Contribution:javaStringTemplate

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Summary

- We speak of **serialization**, if we map program data to XML, JSON or alike.
- We speak of **unparsing**, if we map program data to text.
- We speak of **pretty printing**, if the unparser can withstand a beauty contest.
- Unparsing may be based on at least three different methods:
  - **Copying tokens** at a lexical level
  - Custom **methods** visiting objects and writing output
  - **Templates** to be executed by a template engine

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)