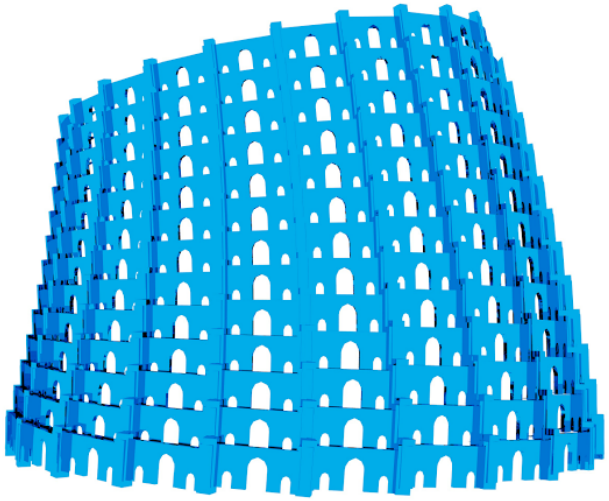
Technological spaces

Prof. Dr. Ralf Lämmel
University of Koblenz-Landau
Faculty of Computer Science
Software Languages Team



SOFTLANG

Creative Commons License: softlang logos by Wojciech Kwasnik, Archina Void, Ralf Lämmel, Software Languages Team, Faculty of Computer Science, University of Koblenz-Landau is licensed under a Creative Commons Attribution 4.0 International License

© 2016 Ralf Lämmel

What’s a technological space?

© 2016 Ralf Lämmel

Technological space = Technology and community context in software engineering

- UMLware — Mainstream software modeling
- MDEware — Model Driven Engineering
- Javaware — Mainstream OO programming
- XMLware — Interoperability in data exchange
- SQLware — Mainstream database management
- Pythonware — Scripting and frameworks
- RDFware — Semantic Web and Linked Data
- JSONware — Modern interoperability
- COBOLware — Legacy programming technology

© 2016 Ralf Lämmel

Technological Spaces: an Initial Appraisal•

Ivan Kurtev [1], Jean Bézivin [2], Mehmet Aksit [1]

[1] Software Engineering Group (TRESE), University of Twente, The Netherlands
{kurtev, aksit}@cs.utwente.nl

[2] Faculty of Sciences, University of Nantes, France
bezivin@sciences.univ-nantes.fr

Abstract. In this paper, we propose a high level view of technological spaces (TS) and relations among these spaces. A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. It is often associated to a given user community with shared know-how, educational support, common literature and even workshop and conference regular meetings. Although it is difficult to give a precise definition, some TSs can be easily identified, e.g. the XML TS, the DBMS TS, the abstract syntax TS, the meta-model (OMG/MDA) TS, etc. The purpose of our work is not to define an abstract theory of technological spaces, but to figure out how to work more efficiently by using the best possibilities of each technology. To do so, we need a basic understanding of the similarities and differences between various TSs, and also of the possible operational bridges that will allow transferring the results obtained in one TS to other TS. We hope that the presented industrial vision may help us putting forward the idea that there could be more cooperation than competition among alternative technologies. Furthermore, as the spectrum of such available technologies is rapidly broadening, the necessity to offer clear guidelines when choosing practical solutions to engineering problems is becoming a must, not only for teachers but for project leaders as well.

© 2016 Ralf Lämmel

Technological Spaces: an Initial Appraisal*

Ivan Kurtev [1], Jean Bézivin [2], Mehmet Aksit [1]

[1] Software Engineering Group (TRESE), University of Twente, The Netherlands
{kurtev, aksit}@cs.utwente.nl

[2] Faculty of Sciences, University of Nantes, France
bezivin@sciences.univ-nantes.fr

A technological space is **a working context with a set of associated concepts, body of knowledge,tools, required skills, and possibilities**. It is often associated to a given user community with shared know-how, educational support, common literature and even workshop and conference meetings. It is at the same time a zone of established expertise and ongoing research and a repository for abstract and concrete resources.

© 2016 Ralf Lämmel

More technological spaces ...

© 2016 Ralf Lämmel

Constituents of a technological space

- Application scenarios
- Programming languages
- Data representation formats
- Development tools such as IDEs and compilers
- Runtime tools such as libraries and frameworks
- Query and transformation languages
- Textbooks and other knowledge resources
- Conferences and other community resources
- …

© 2016 Ralf Lämmel

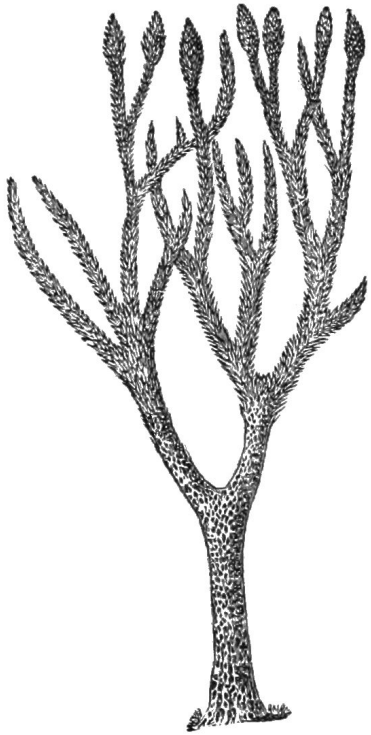
Different data paradigms

- Table
  - Relational tables in SQL
- Tree
  - XML trees
  - JSON trees
  - Algebraic trees
- Graph
  - OO graphs
  - Model graphs

An aspect of space diversity

© 2016 Ralf Lämmel

Major data paradigms:
trees versus tables versus graphs



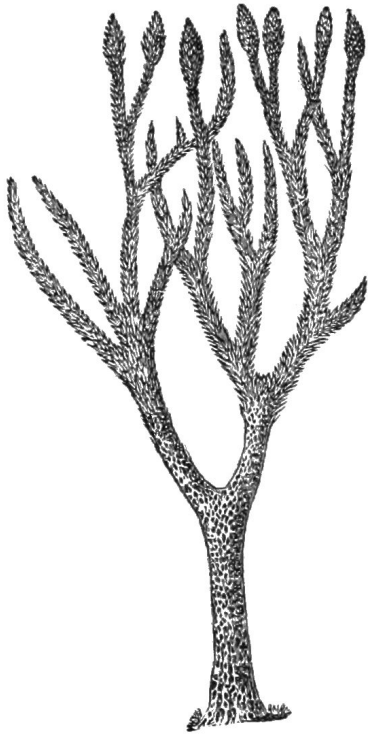
http://en.wikipedia.org/wiki/File:PSM_V18_D630_Restoration_of_a_lepidodendron.jpg

XML for a company

```
<department>
        <name>Research</name>
        <manager>
                <name>Craig</name>
                <address>Redmond</address>
                <salary>123456</salary>
        </manager>
        <employee>
                <name>Erik</name>
                <address>Utrecht</address>
                <salary>12345</salary>
        </employee>
        <employee>
                <name>Ralf</name>
                <address>Koblenz</address>
                <salary>1234</salary>
        </employee>
</department>
```

© 2016 Ralf Lämmel

Major data paradigms:
trees versus tables versus graphs



http://en.wikipedia.org/wiki/File:PSM_V18_D630_Restoration_of_a_lepidodendron.jpg

```
{
   "name" : "Research",
   "manager" : {
     "name" : "Craig",
     "address" : "Redmond",
     "salary" : 123456 },
   "employees" : [
     {
        "name" : "Erik",
        "address" : "Utrecht",
        "salary" : 12345 },
     {
        "name" : "Ralf",
        "address" : "Koblenz",
        "salary" : 1234 } ] }
```

JSON for a company

© 2016 Ralf Lämmel

Major data paradigms:
trees versus **tables** versus graphs

Relational tables for a company

Employee

Id	Name	Address	Salary	Manager	Department
1	Craig	Redmond	123456	TRUE	42
2	Erik	Utrecht	12345	FALSE	42
3	Ralf	Koblenz	1234	FALSE	42
…	…	…	…	…	…

Department

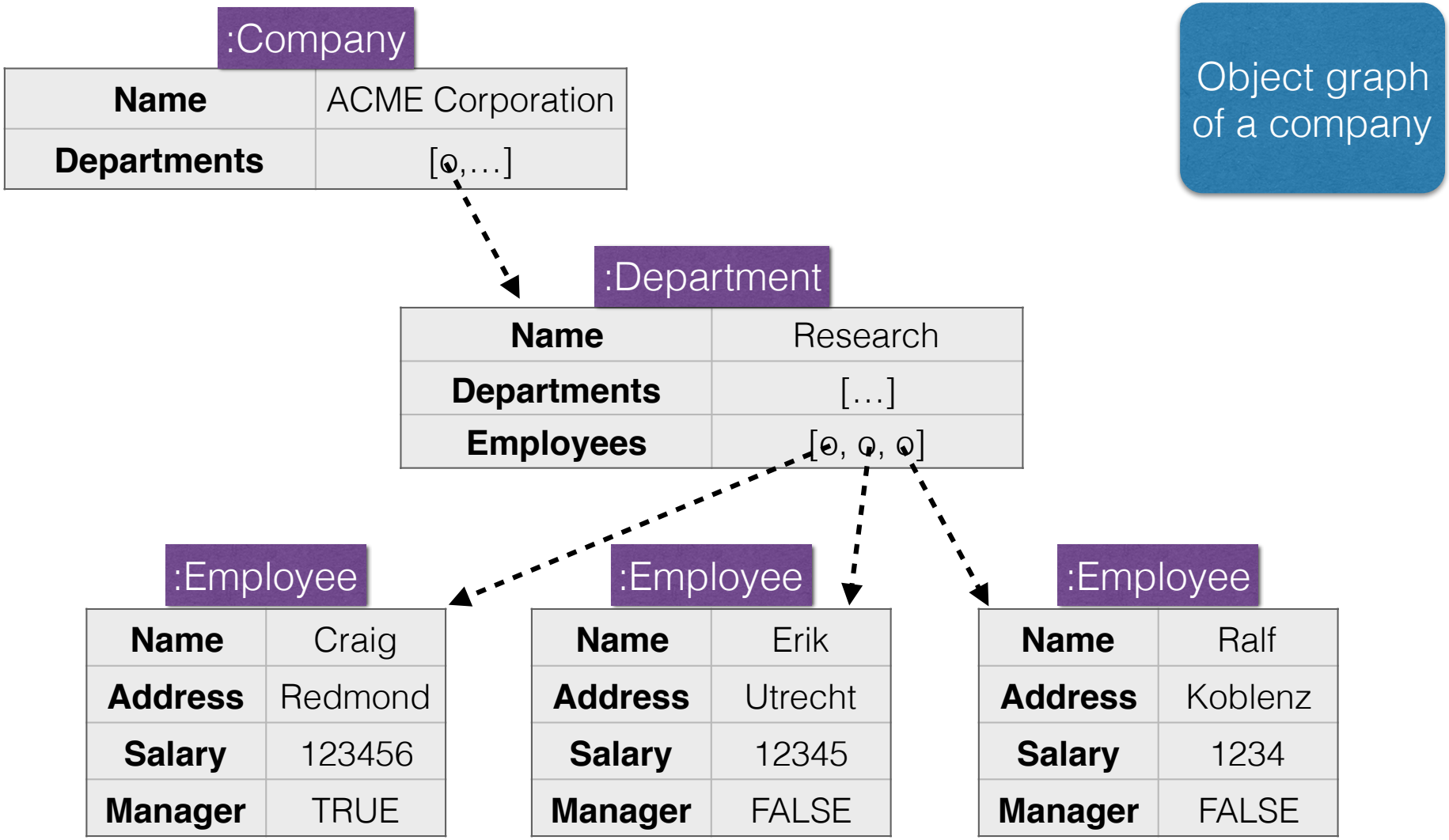
Id	Name	Department	Company
42	Research	NULL	88
43	Development	NULL	88
44	Dev1	43	88
…	…	…	…

Company

Id	Name
88	ACME Corporation
…	…

© 2016 Ralf Lämmel

Major data paradigms:
trees versus tables versus **graphs**



© 2016 Ralf Lämmel

Programming paradigms

- **Object-oriented**
- Procedural
- Functional
- Logical
- Relational (algebra-like)
- Model-based
- Multi-paradigm

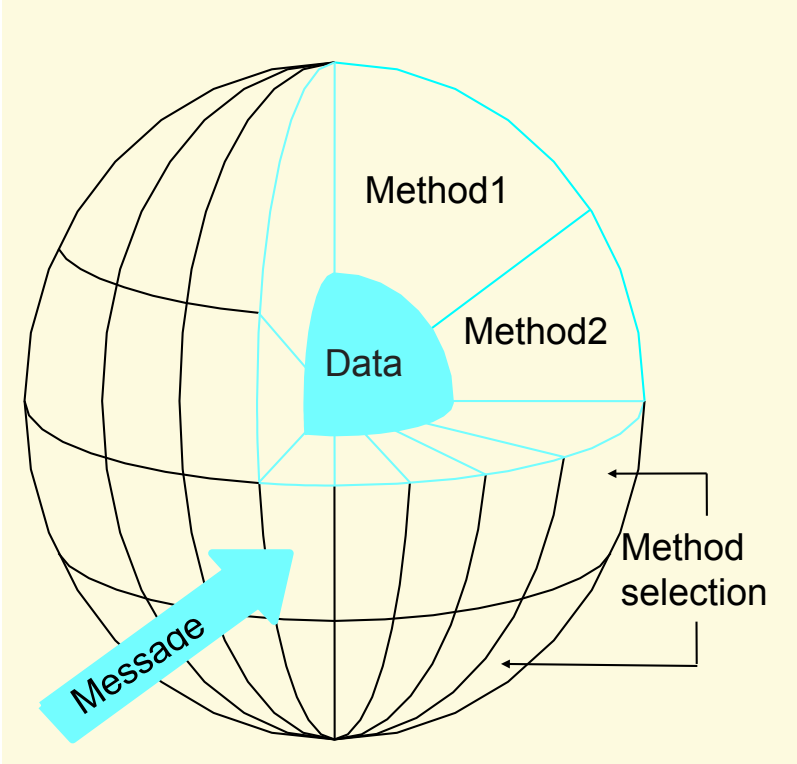
http://en.wikipedia.org/wiki/Programming_paradigm

A programming paradigm is a fundamental style of computer programming, a way of building the structure and elements of computer programs. Capablities and styles of various programming languages are defined by their supported programming paradigms; some programming languages are designed to follow only one paradigm, while others support multiple paradigms.

Another aspect of space diversity

© 2016 Ralf Lämmel

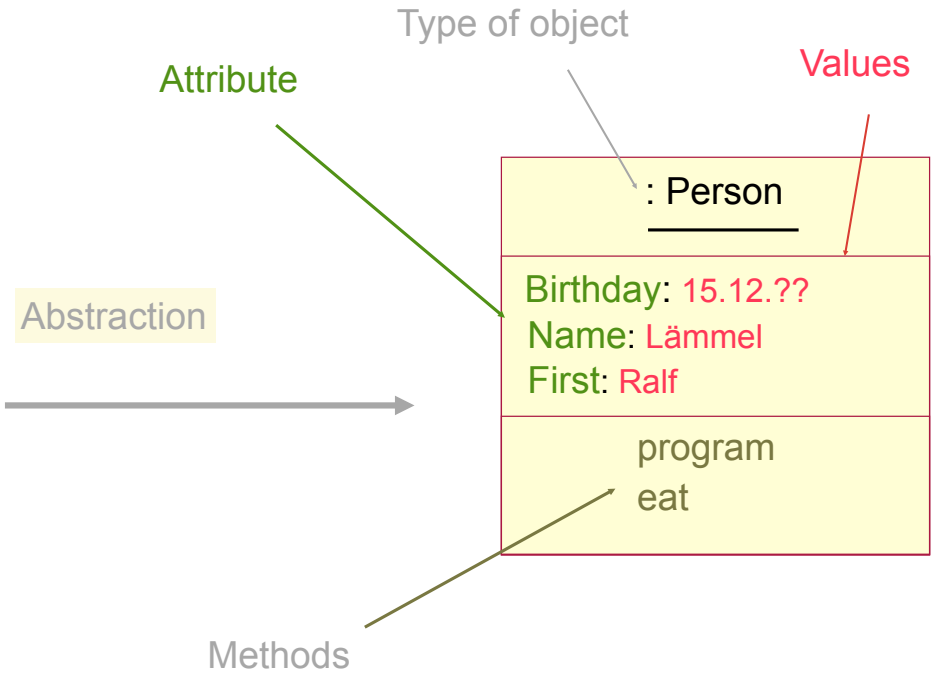
The **object-oriented** paradigm



- Objects encapsulate state.
- Objects provide an interface.
- Methods operate on state.
- Objects may be stamped out by classes.

© 2016 Ralf Lämmel

Objects as abstractions



© 2016 Ralf Lämmel

15

Everything is an object

- Physical entities (e.g., a rock or a house)
- Observable entities (e.g., a weather situation)
- Virtual entities (e.g., an account in a bank)
- Computational entities (e.g., a print task)

© 2016 Ralf Lämmel

The notion of *object*

Capsules of state and behavior in OO programming

```
>>> counter1 = Counter()
>>> counter1
0
>>> counter1.inc()
>>> counter1.inc()
>>> counter1
2
>>> counter1.reset()
>>> counter1
0
```

Create a counter

Show representation

Increment twice and show again

Reset counter and show again

© 2016 Ralf Lämmel

The notion of *object*

Capsules of state and behavior in OO programming

```
>>> cnum1 = Complex(1,2)
>>> cnum2 = Complex(3,4)
>>> cnum1.add(cnum2)
Complex(4, 6)
```

Create two complex numbers

Add and show result

© 2016 Ralf Lämmel

The notion of *object*

Capsules of state and behavior in OO programming

```
>>> l = [2,1,3]
>>> l.pop()
3
>>> l
[2, 1]
>>> l.append(4)
>>> l
[2, 1, 4]
>>> l.sort()
>>> l
[1, 2, 4]
```

Create a list

Remove the ‚last' element of the list

Add another element to the list

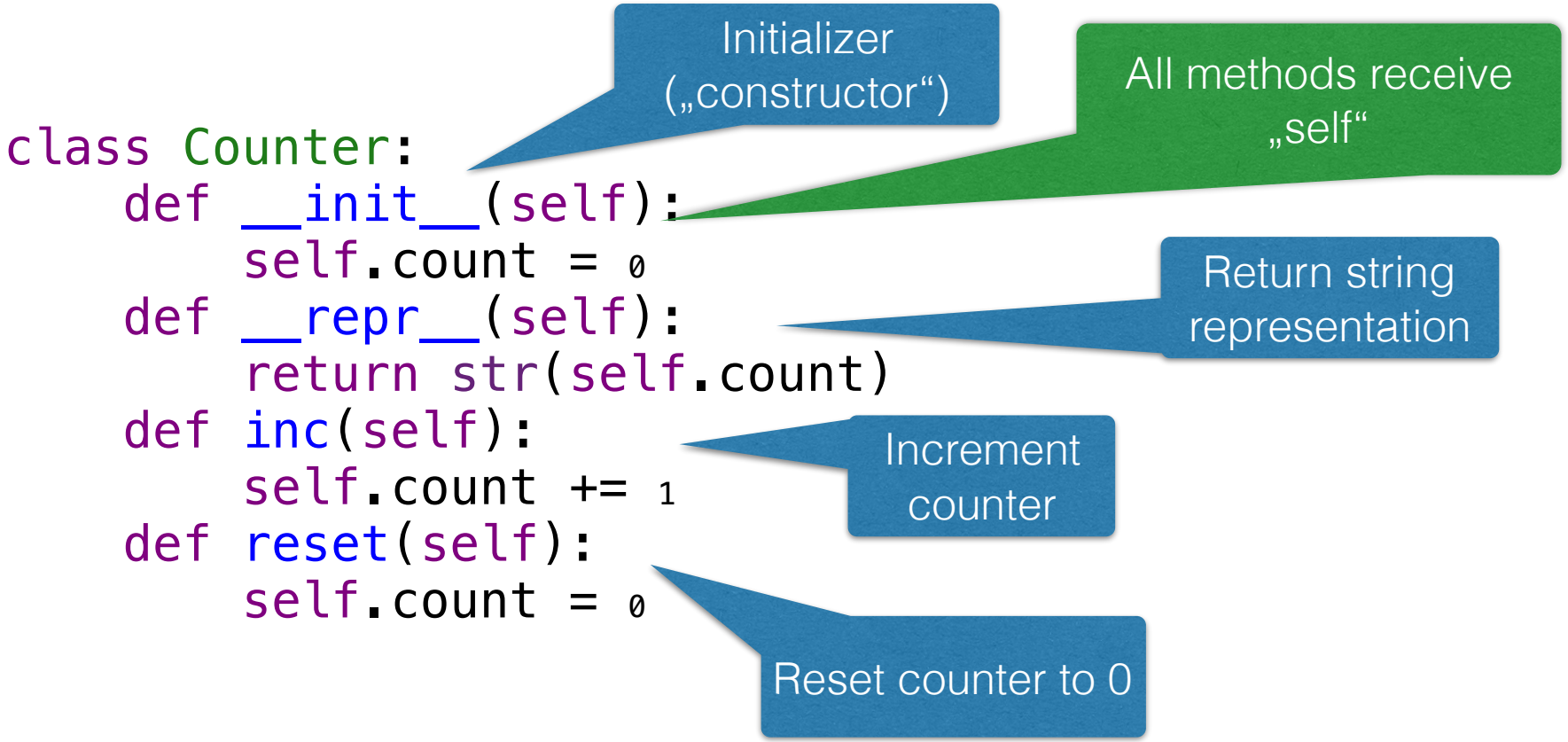
Sort the list

We don't care much how exactly those lists are internally represented. We basically care about the methods to **observe** and **manipulate** such **state**.

© 2016 Ralf Lämmel

The notion of *class*

An abstraction mechanism for objects




```
class Counter:
    def __init__(self):
        self.count = 0
    def __repr__(self):
        return str(self.count)
    def inc(self):
        self.count += 1
    def reset(self):
        self.count = 0
```

© 2016 Ralf Lämmel

The notion of *class*

An abstraction mechanism for objects

```
class Complex:
    def __init__(self, re, im):
        self.re = re
        self.im = im
    def __repr__(self):
        return "Complex(" \
               + str(self.re) \
               + ", " \
               + str(self.im) + ")"
    def add(self, cnum):
        return Complex(\
                   self.re+cnum.re, \
                   self.im+cnum.im)
```

Initializer („constructor“)

Return string representation

Addition of two complex numbers returns a new number as result.

© 2016 Ralf Lämmel

A Java class I/II

```
public class Department {

    private String name;
    private Employee manager;
    private List<Department> subdepts = new LinkedList<Department>();
    private List<Employee> employees = new LinkedList<Employee>();

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public Employee getManager() { return manager; }
    public void setManager(Employee manager) { this.manager = manager; }
    public List<Department> getSubdepts() { return subdepts; }
    public List<Employee> getEmployees() { return employees; }

    // Methods omitted
}
```

Private state

Getters und setters for public data access

© 2016 Ralf Lämmel

A Java class II/II

```
public class Department {

    // State omitted

    public double total() {
        double total = 0;
        total += getManager().getSalary();
        for (Department s : getSubdepts())
            total += s.total();
        for (Employee e : getEmployees())
            total += e.getSalary();
        return total;
    }

    public void cut() {
        getManager().cut();
        for (Department s : getSubdepts())
            s.cut();
        for (Employee e : getEmployees())
            e.cut();
    }
}
```

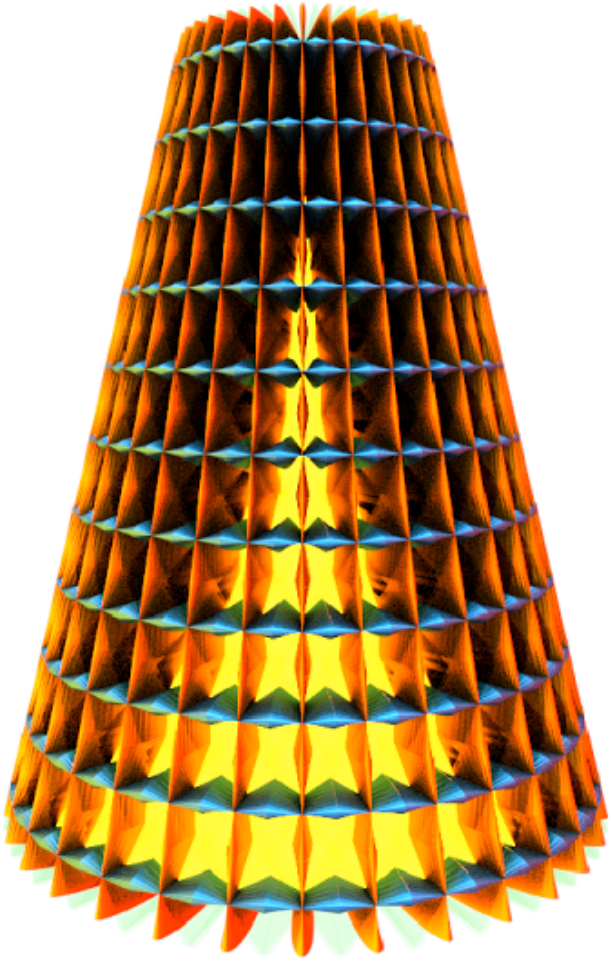
Recurse into object and aggregate salaries

Recurse into object and modify salaries

© 2016 Ralf Lämmel

Space exploration with 101companies

http://101companies.org/

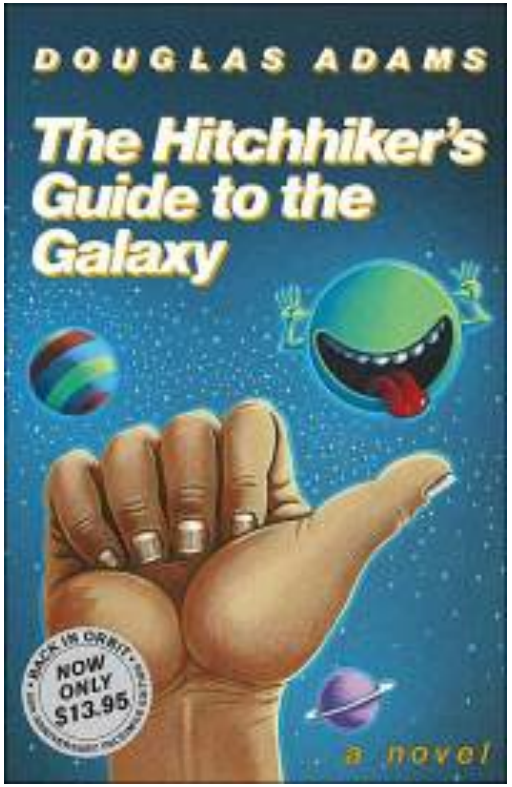


Creative Commons License: softlang logos by Wojciech Kwasnik, Archina Void, Ralf Lämmel, Software Languages Team, Faculty of Computer Science, University of Koblenz-Landau is licensed under a Creative Commons Attribution 4.0 International License

What's 101companies?

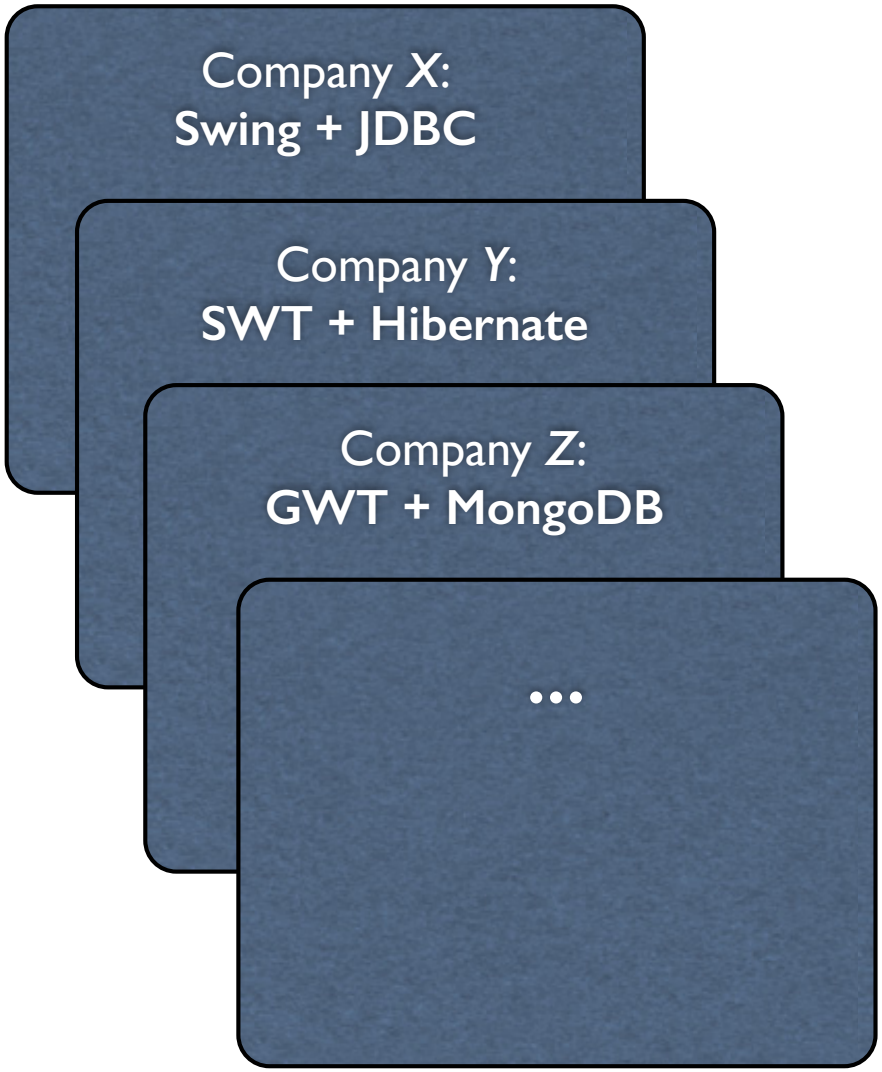
It is ...

The Hitchhiker's Guide to the *Software* Galaxy



© 2016 Ralf Lämmel

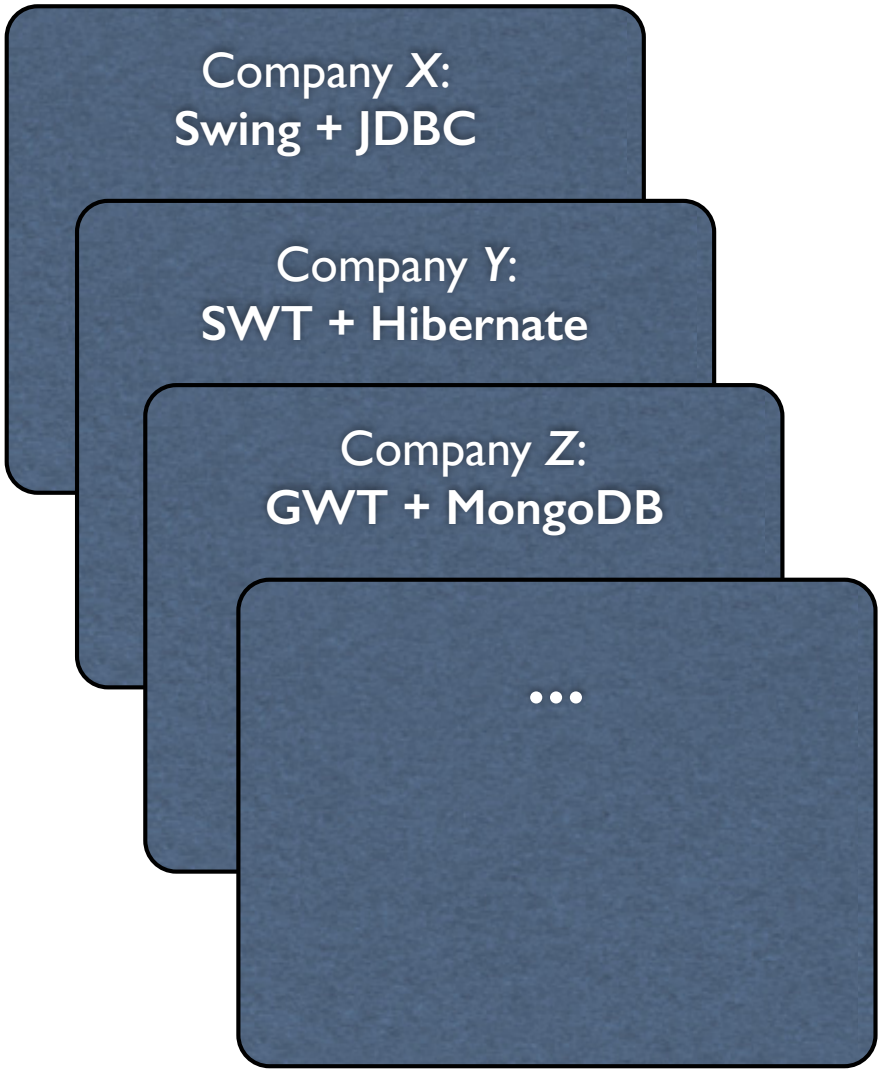
What's 101companies?



A **community project** aiming at a **knowledge base** about software **technologies and languages** based on implementations of a human-resources management system.

© 2016 Ralf Lämmel

Why is it called “101companies”?

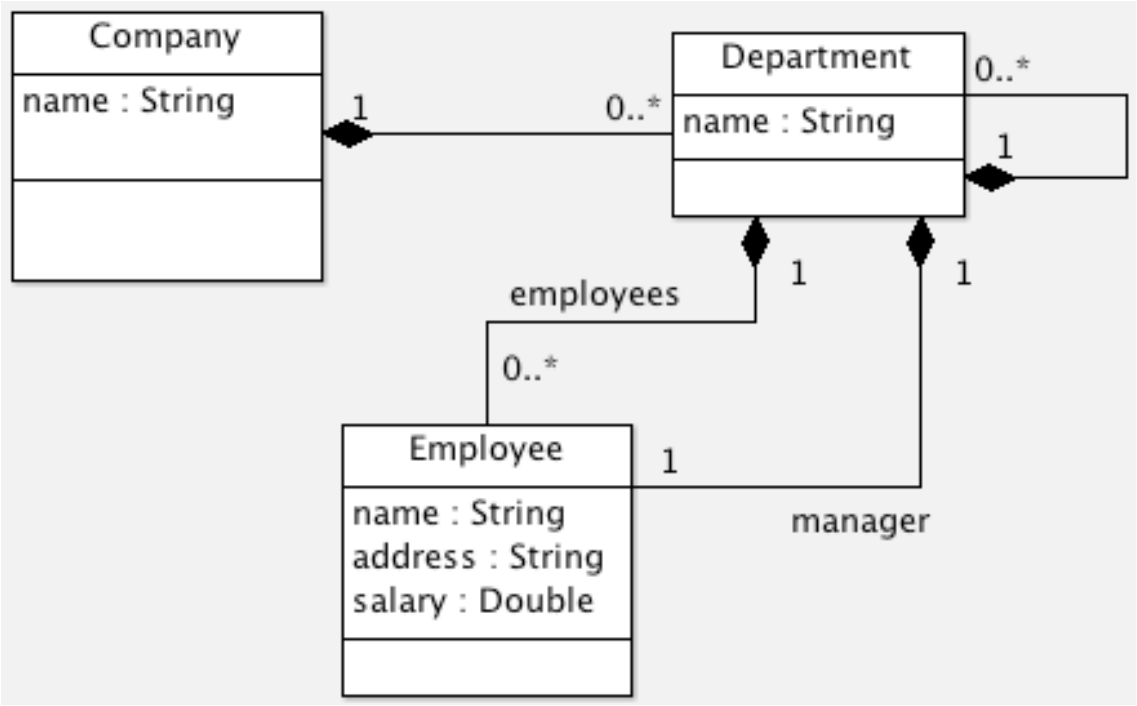


101 ways of building a HRMS.

Building a HRMS for 101 companies.

© 2016 Ralf Lämmel

A Human Resources Management System



- Total salaries
- ~~Increase salaries~~
- Cut salaries
- Edit employee data
- Import / export company data

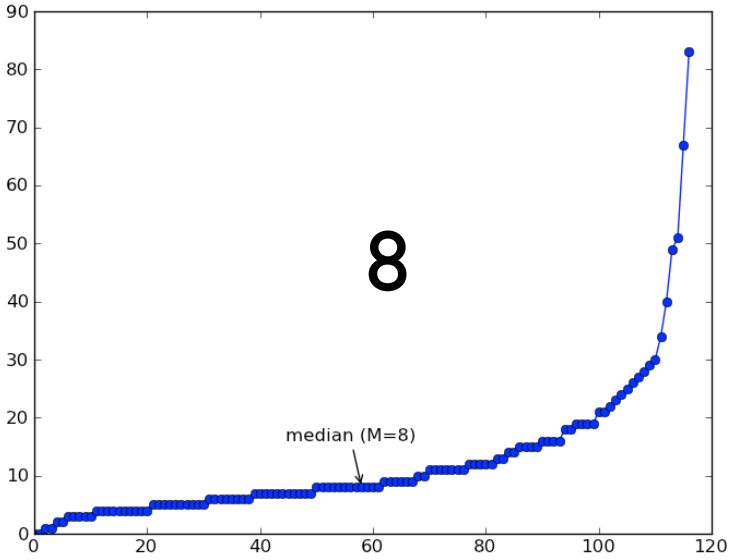
© 2016 Ralf Lämmel

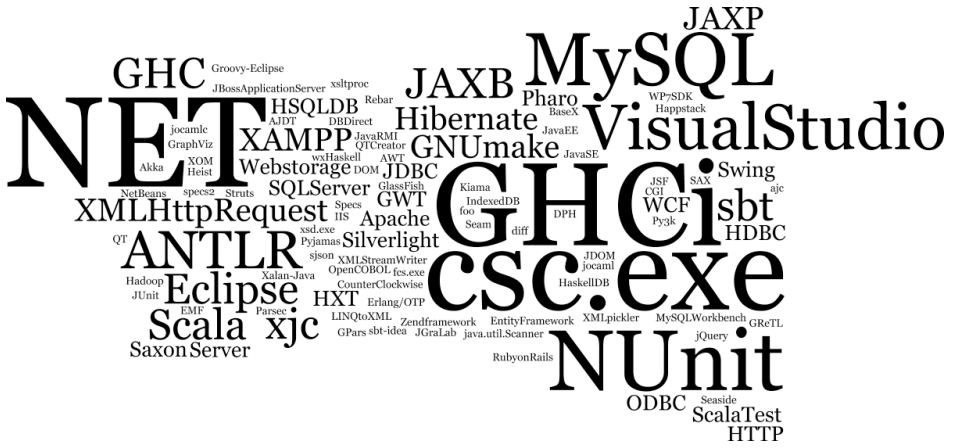
Demo

http://101companies.org/wiki/Contribution:pyjson

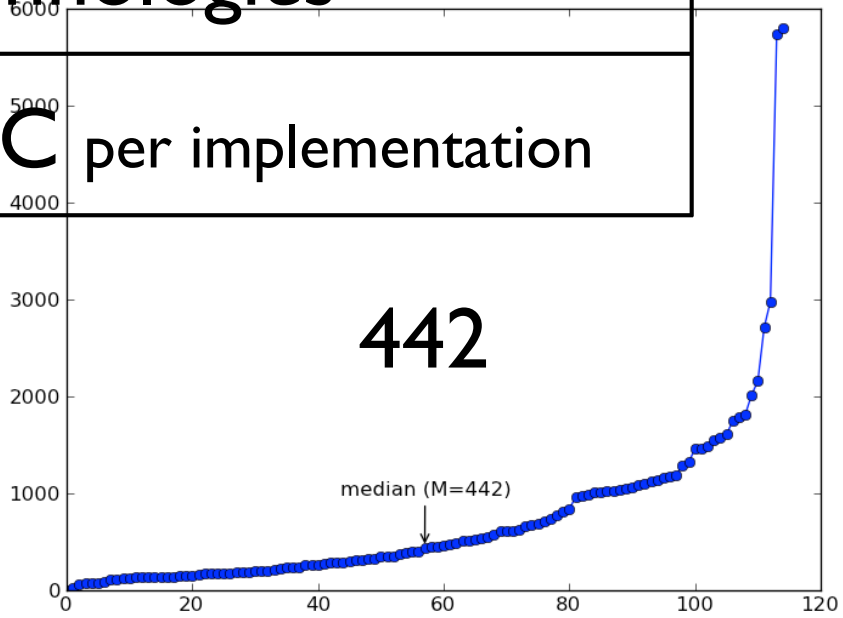
© 2016 Ralf Lämmel

The 101companies Repository





#Files per implementation	Technologies
Languages	LOC per implementation



© 2016 Ralf Lämmel

The *101companies* Wiki

- Classification (e.g., languages, technologies)
- Feature modeling for the *101system*
- Documentation of each contribution
- Collections of contributions in themes
- Tracking of semantic metadata
  - *Languages, technologies, developers, …*

© 2016 Ralf Lämmel

The 101companies Ontology

- 101companies contribution *X* ...
  - ... uses (‚is written in') language *L*
  - ... uses technology *T*
  - ... implements feature *F*
  - ... uses (‚exercises') concept *C*
- 101companies developer *D* ...
  - ... has skills regarding language *L*
  - ... has skills regarding technology *T*
- ...

© 2016 Ralf Lämmel

Acknowledgement: *Jean-Marie Favre* designed this (such a) slide.

We have a problem!

Too many technologies.

Too little time.

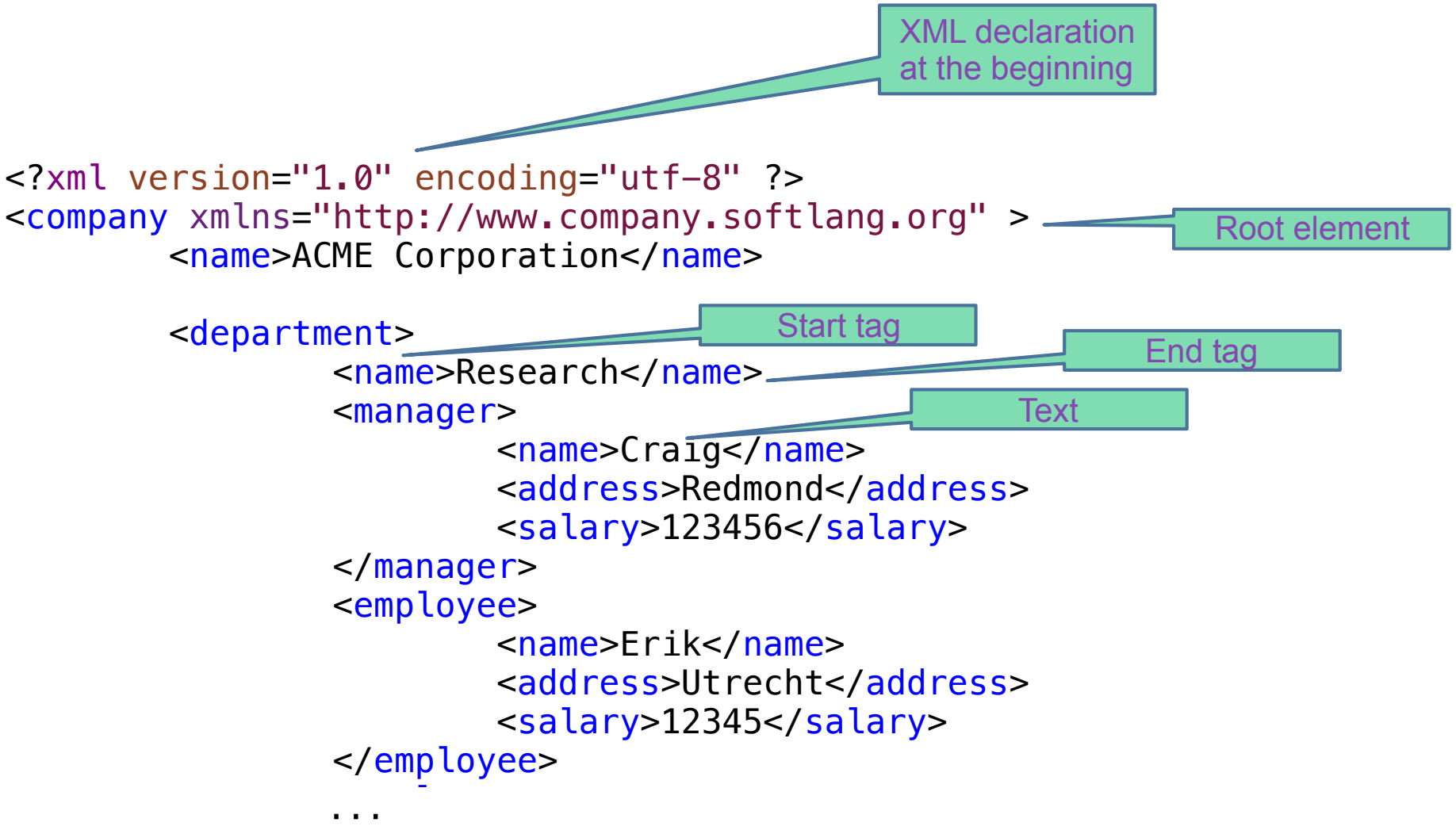
Selected technological spaces

© 2016 Ralf Lämmel

XMLware

© 2016 Ralf Lämmel

A sample XML document



© 2016 Ralf Lämmel

Classic motivation of XML

Consider the following realities:

- Applications are built in Cobol, C#, Java, Python, etc.
- "Messages" have to be send over the "wire".
- Configuration data has to be represented somehow.
- Objects have to be persisted in an human-readable way.

Conclusions:

- We need to model data in a language-independent manner.
- We need a comprehensive method for exchange/storage formats.

We need (something like) XML.

XML stands for eXtensible Markup Language.

Perhaps, JSON would also do. Let's discuss this later.

© 2016 Ralf Lämmel

XML-based languages
- Some examples -

- FpML – Financial products Markup Language
- HL7 – Standards in Healthcare
- MathML – Mathematical Markup Language
- XHTML – HTML defined in XML
- … and countless others …

© 2016 Ralf Lämmel

XML-related technologies and standards

© 2016 Ralf Lämmel

XML queries with XPath

- XPath is an XML ***query*** language.
- Queries are composed from query axes.
  - Children, Ancestors, Descendants, ...
- XPath is embedded into Java et al. by APIs.
  - XPath queries are encoded as strings.
- Examples:

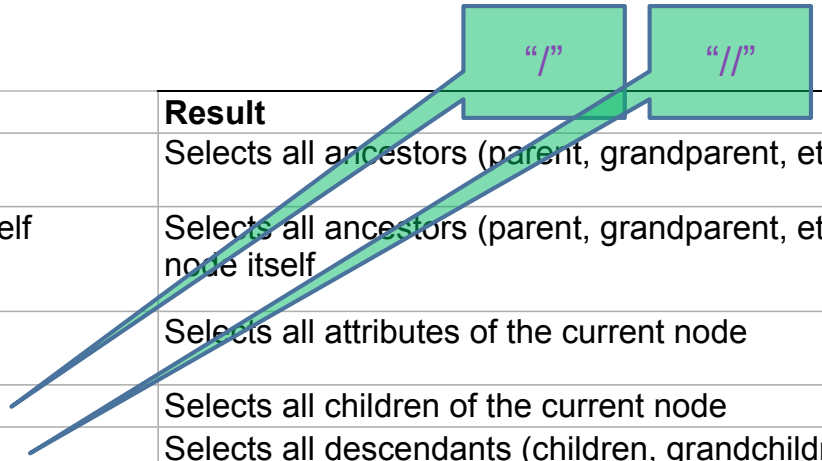
  “//salary”

  “//manager/salary”

All salary nodes below all manager nodes anywhere

© 2016 Ralf Lämmel

XPath axes



AxisName	Result
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects everything in the document that is before the start tag of the current node
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

© 2016 Ralf Lämmel

XML transformations with XSLT

- XSLT is an XML ***transformation*** language.
- XSLT leverages XPath for node selection.
- XSLT is an XML-based language.
- XSLT is a functional programming language.

© 2016 Ralf Lämmel

Cutting salaries with XSLT

```
<xsl:stylesheet>
  <xsl:template match="salary">
    <xsl:copy>
      <xsl:value-of select=". div 2"/>
    </xsl:copy>
  </xsl:template>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Type-specific template

Generic default

Recursion into kids

© 2016 Ralf Lämmel

XML languages
(defined by XML schemas)

© 2016 Ralf Lämmel

Well-formedness of XML

- A document has exactly one root element.
- There is an end tag for each start tag: <name>….</name>
- Empty elements can be abbreviated: <noname/>
- Elements must be properly nested.
- Attribute values must be enclosed into “…”.
- …

© 2016 Ralf Lämmel

Beyond well-formedness

OK

```
<employee>
    <name>Ralf</name>
    <address>Koblenz</address>
    <salary>1234</salary>
</employee>
```

Not OK!

```
<employee>
    <name>Ralf</name>
    <salary>1234 Euro</salary>
</employee>
```

Address missing

Not a number

© 2016 Ralf Lämmel

XML Schema (XSD)

http://www.w3.org/XML/Schema:

> "XML Schemas express shared vocabularies and allow machines to carry out rules made by people. ***They provide a means for defining the structure, content*** *and semantics* ***of XML documents****. […]* XML Schema was approved as a W3C Recommendation on 2 May 2001."

© 2016 Ralf Lämmel

XSD for 101

```
<xs:element name="company">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="name"/>
         <xs:element  maxOccurs="unbounded"
                      minOccurs="0"
            ref="department"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:element name="department"> ... </xs:element>

<xs:complexType name="employee"> ... </xs:complexType>
```

© 2016 Ralf Lämmel

XML validation

- Well-formed XML as a prerequisite
- Input
  - XML document
  - XML schema
- Output
  - “valid”: document valid w.r.t. schema
  - “invalid” + violations

© 2016 Ralf Lämmel

XSD – schema components

- **Element declarations**
- **Complex type definitions**
- Model-group definitions
- Simple type definitions
- Attribute declarations
- Attribute-group definitions
- Redefinitions
- Annotations

Sets of XML trees rooted by a certain element name

Recursive macros with subtyping

Nonrecursive macros without subtyping

Types of leafs in XML trees (both elements and attributes).

Deprecated

Comments and hints for schema processors

© 2016 Ralf Lämmel

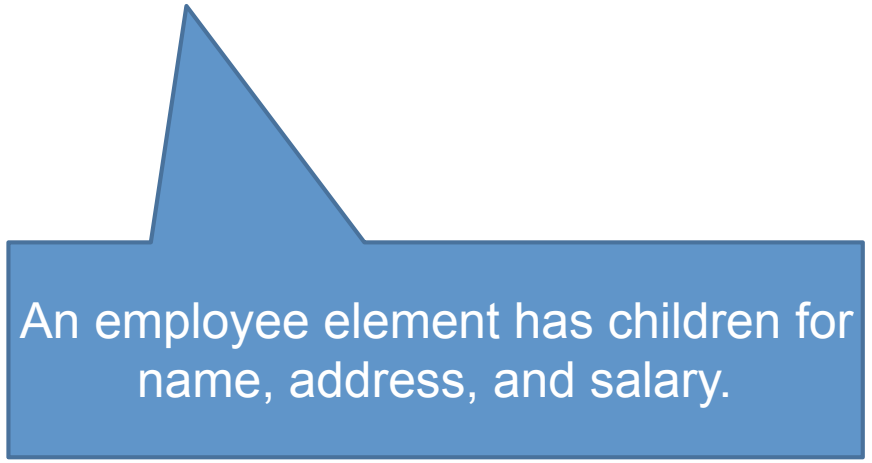
Sketch of the company schema

```
<xs:schema …>
    <xs:element name="company“> … </xs:element>
    <xs:element name="department“> … </xs:element>
    <xs:complexType name="employee"> … </xs:complexType>
    <xs:element name="name"> … </xs:element>
    <xs:element name="address"> … </xs:element>
    <xs:element name="salary"> … </xs:element>
</xs:schema>
```

© 2016 Ralf Lämmel

<sequence>

```
<xs:complexType name="employee">
   <xs:sequence>
        <xs:element ref="name" />
        <xs:element ref="address" />
        <xs:element ref="salary" />
   </xs:sequence>
</xs:complexType>
```



© 2016 Ralf Lämmel

<choice>

```
<xs:element name="subunit">
    <xs:complexType>
        <xs:choice>
            <xs:element name="employee"
    type="employee" />
            <xs:element ref="department" />
        </xs:choice>
    </xs:complexType>
</xs:element>
```

In a variation of our preferred schema, a **subunit** (of a department) is either an employee or a department.

© 2016 Ralf Lämmel

“*” (i.e., many)

```
<xs:element name="company">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="name"/>
         <xs:element     maxOccurs="unbounded"
                         minOccurs="0"
                      ref="department"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

A company element has any number of department elements as its children.

© 2016 Ralf Lämmel

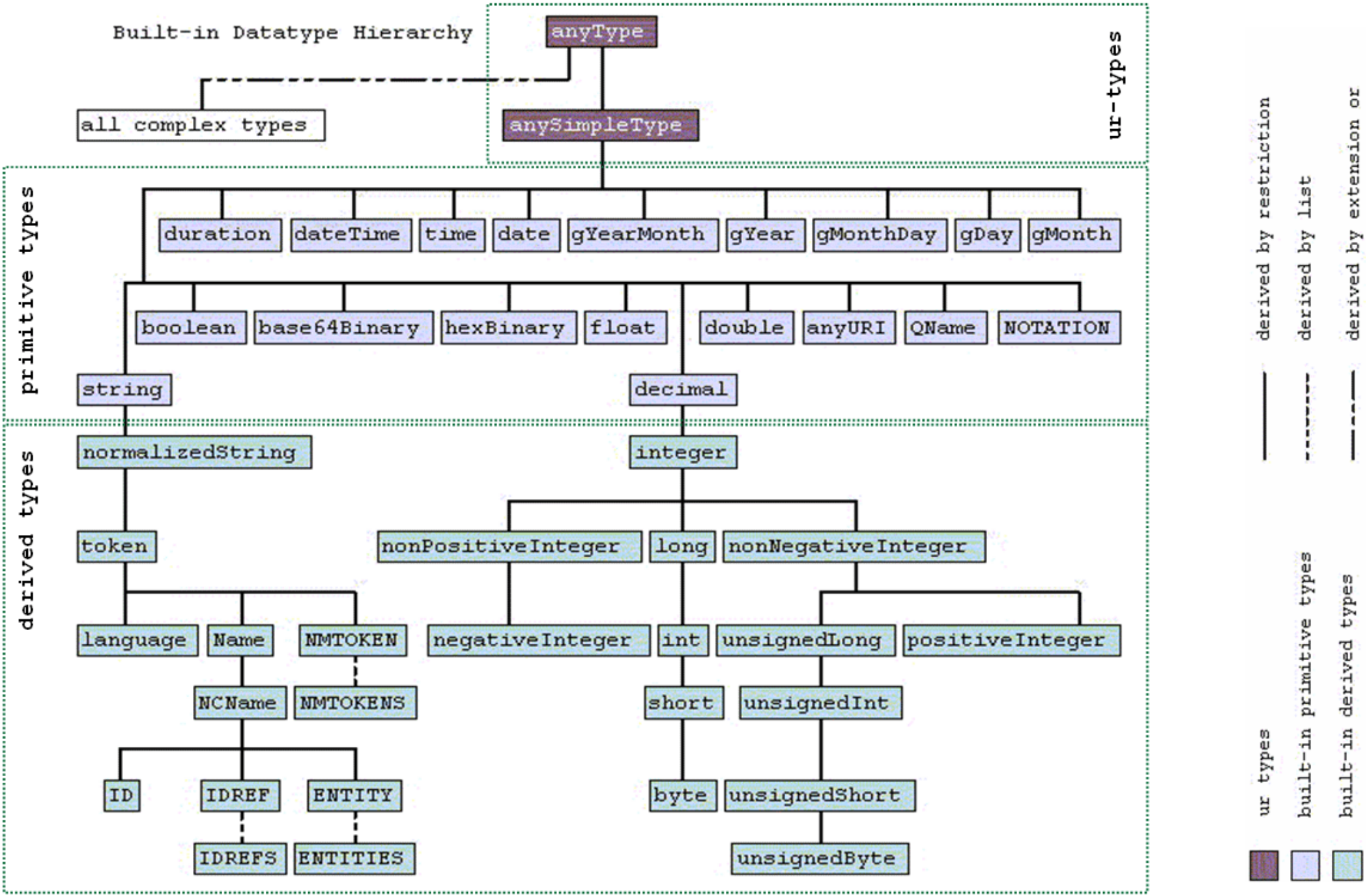
XSD simple types

- Comparable to primitive types in a programming.
- Example:

  <xs:element name="salary" **type="xs:double"**/>

- All attributes are of simple types.
- New simple types can be defined by:
  - Restriction
  - Union
  - List

© 2016 Ralf Lämmel



© 2016 Ralf Lämmel

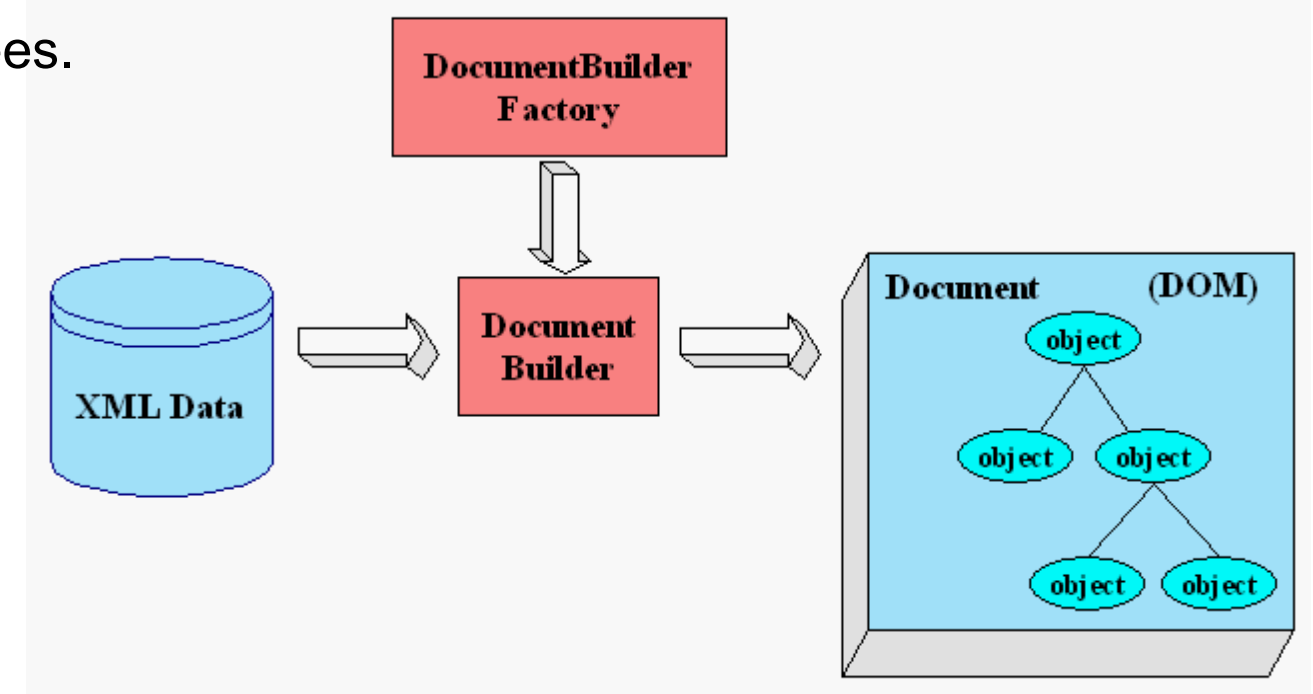
Options for XML processing in an OO programming language

- APIs for push-based parsers
  - Java's **SAX**, ...
- APIs for pull-based parsers
  - .NET's XmlReader, Java's StAX, ...
- APIs for in-memory XML trees
  - W3C's **DOM**, Java's JDOM, .NET's LinqToXml
- Programming languages with XML support
  - VB.NET, ...

© 2016 Ralf Lämmel

The DOM option
(DOM=Document Object Model)

- An object model for XML trees.
- Central types:
  - Document
  - Attribute
  - Node
    - Element
    - Text
    - Comment
    - CData
- API segments:
  - Construction
  - Navigation
  - Modification



Source: Armstrong: “Working with XML”

© 2016 Ralf Lämmel

Feature „Total“ with the JDOM API

http://101companies.org/wiki/Contribution:jdom

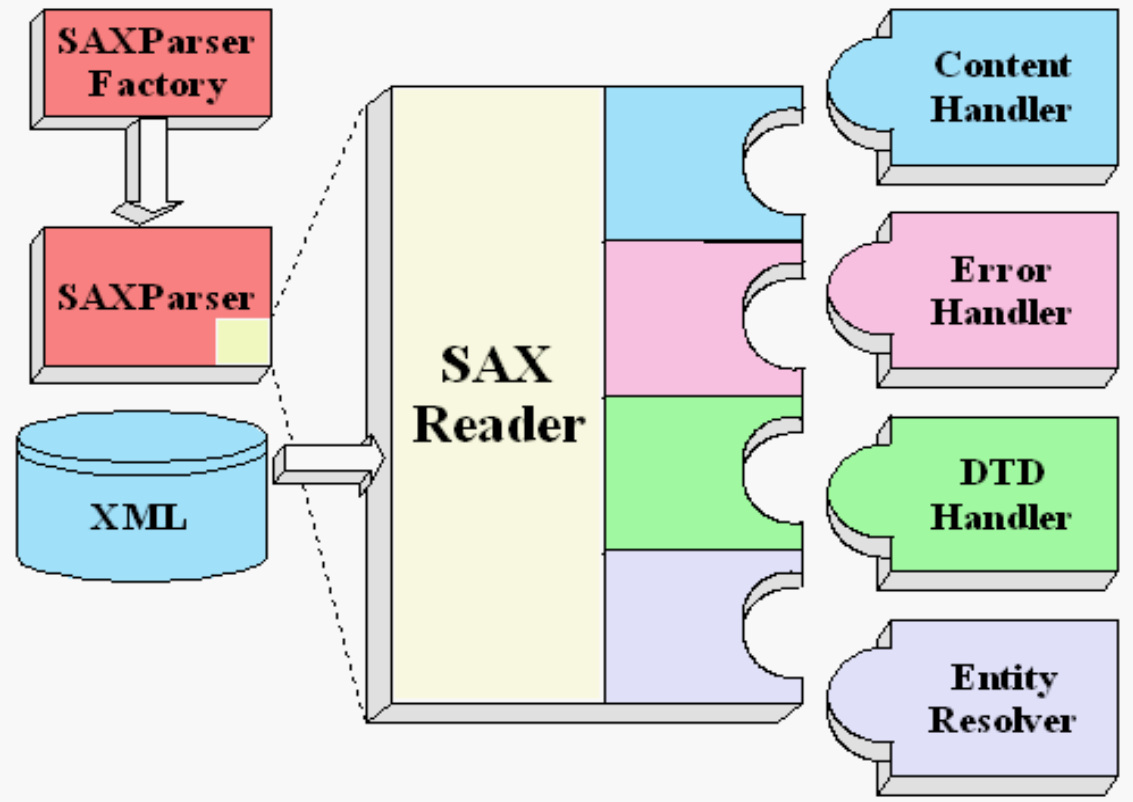
```
public class Total {

    public static double total(Document doc) {

        // Aggregate salaries
        double total = 0;

        // Iterate over all salary elements
        Iterator<?> iterator = doc.getDescendants(new ElementFilter("salary"));
        while (iterator.hasNext()) {
            Element elem = (Element)iterator.next();
            Double salary = Double.valueOf(elem.getText());
            total += salary;
        }

        return total;
    }

}
```

XML query axis

Parse XML text

© 2016 Ralf Lämmel

The SAX option
(SAX=Simple API for XML)



- A **framework** for event handling-based XML parsers.
- **Typical events**
  - Open element
  - Close element
  - Find text
  - ...

Source: Armstrong: “Working with XML”

© 2016 Ralf Lämmel

Feature „Total“ with the SAX API

http://lolcompanies.org/wiki/Contribution:sax

```
/**
 * Handle "start element"
 */
public void startElement(
        String uri,
        String name,
        String qName,
        Attributes atts) {
    isSalary = (uri.equals(ns) && name.equals("salary"));
}
```

Event handler for „start“ element: determine whether „salary“ element begins.

See the complete code online.

© 2016 Ralf Lämmel

JSONware

© 2016 Ralf Lämmel

Store „company“ data in JSON for platform independent data exchange and human consumption

```
{
  "name" : "Acme Corporation",
  "departments" : [
    {
      "name" : "Research",
      "manager" : {
        "name" : "Fred",
        "salary" : 88888
      }
    },
    {
      "name" : "Development",
       ...
```

So we use JSON rather than XML this time around.

© 2016 Ralf Lämmel

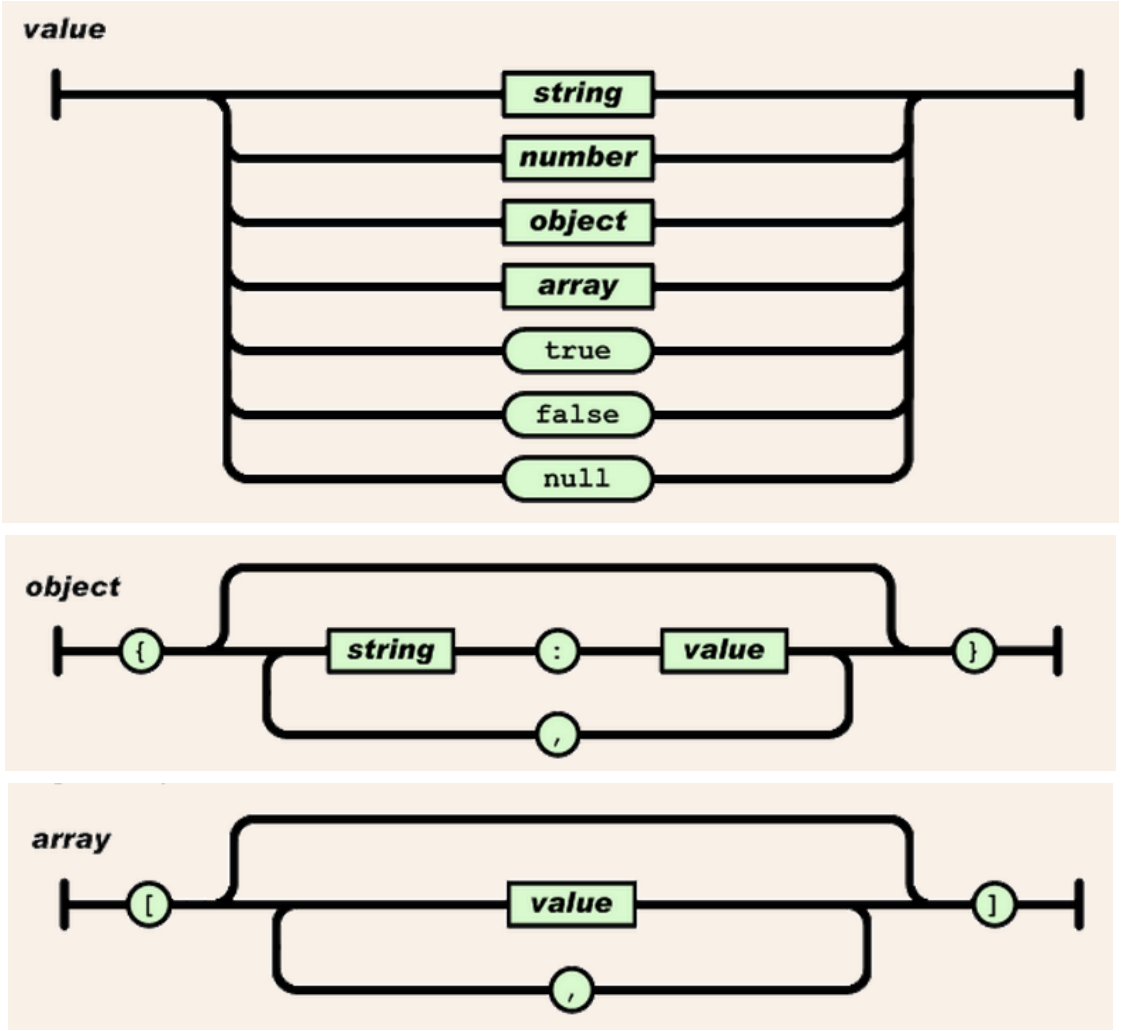
JSON — The JavaScript Object Notation
http://101companies.org/wiki/Language:JSON

- Inspired by JavaScript syntax and type system
  - Objects are essentially dictionaries.
    - **Keys** are strings. **Values** are JavaScript values.
- Thought to be much simpler than XML
- Very well supported across many platforms
- Increasingly dominant in the Web context

© 2016 Ralf Lämmel

JSON syntax

Source: http://www.json.org/



© 2016 Ralf Lämmel

Feature „Total“ with the javax.json API

http://101companies.org/wiki/Contribution:javaJson

```
package org.softlang.company.features;

import javax.json.JsonArray;
import javax.json.JsonNumber;
import javax.json.JsonObject;
import javax.json.JsonValue;

public class Total {

        /**
         * @param obj
         *            JSON object to traverse
         * @return salary total
         */
        public static double total(JsonObject obj) {
                return total(null, obj);
        }

        /**
         * @param key1
         *            key for the value at hand
         * @param val1
         *            JSON value to traverse
         * @return salary total
         */
        private static double total(String key1, JsonValue val1) {
```

Conceptually, the code for JSON processing is similar to the one for XML processing.

There is several JSON APIs for Java. We use the platform standard here.

See the complete code online.

© 2016 Ralf Lämmel

Feature „Total“ with the javax.json API

http://101companies.org/wiki/Contribution:javaJson

```
private static void cut(JsonObjectBuilder builder, String key, JsonValue val) {
        switch (val.getValueType()) {
        case NUMBER:
                Double num = ((JsonNumber) val).doubleValue();
                if (key.equals("salary"))
                        // Halve salary
                        num /= 2;
                builder.add(key, num);
                break;
        case OBJECT:
                builder.add(key, cut((JsonObject) val));
                break;
        case ARRAY:
                builder.add(key, cut((JsonArray) val));
                break;
        default:
                builder.add(key, val);
                break;
        }
}
```

The javax.json API assumes **immutable** JSON values. Thus, transformations need to **copy**.

See the complete code online.

© 2016 Ralf Lämmel

JSON schema

```
{
        "title": "Example Schema",
        "type": "object",
        "properties": {
                "firstName": {
                        "type": "string"
                },
                "lastName": {
                        "type": "string"
                },
                "age": {
                        "description": "Age in years",
                        "type": "integer",
                        "minimum": 0
                }
        },
        "required": ["firstName", "lastName"]
}
```

http://json-schema.org/examples.html

© 2016 Ralf Lämmel

SQLware

© 2016 Ralf Lämmel

Elevator speech

Think of information systems and data processing!

1. How to *persist* data?
2. How to *separate data and functionality*?
3. How to deal with a lot of data *efficiently*?
4. How to implement entity *relationships*?

XML and JSON may serve 1.-2.

Relational databases serve 1.-4.

Exercise: what's XML specifically good for?

Also: how to remain an OO programmer?

© 2016 Ralf Lämmel

Database programming (ignoring OO specifics)

1. Define tables according to relational model
2. Implement relational model via **SQL**
3. Implement **CRUD** functionality

Tables = rows / columns of cells

Create, Read, Update, Delete

© 2016 Ralf Lämmel

The relational model

© 2016 Ralf Lämmel

Relations (tables)

	id	name	address	salary	manager	cid	did
▶	1	Craig	Redmond	30864	1	1	1
	2	Ray	Redmond	58641.75	1	1	2
	3	Klaus	Boston	5864	1	1	3
	4	Karl	Riga	586.25	1	1	4
	5	Erik	Utrecht	3086.25	0	1	1
	6	Ralf	Koblenz	308.5	0	1	1
	7	Joe	Wifi City	586	0	1	4

- Relation
  - Vertically: set of tuples (“rows”)
  - Horizontally: set of columns
- Each cell is of some type
  - Strings
  - Numbers
  - Row IDs (numbers again)

© 2016 Ralf Lämmel

Relational algebra: compute relations

	id	name	address	salary	manager	cid	did
▶	1	Craig	Redmond	30864	1	1	1
	2	Ray	Redmond	58641.75	1	1	2
	3	Klaus	Boston	5864	1	1	3
	4	Karl	Riga	586.25	1	1	4
	5	Erik	Utrecht	3086.25	0	1	1
	6	Ralf	Koblenz	308.5	0	1	1
	7	Joe	Wifi City	586	0	1	4

- **Projection** (select columns)
- **Selection** (select rows)
- **Join** (compose two tables by condition)

© 2016 Ralf Lämmel

Projection
(select columns)

id	name	address	salary	manager	cid	did
1	Craig	Redmond	30864	1	1	1
2	Ray	Redmond	58641.75	1	1	2
3	Klaus	Boston	5864	1	1	3
4	Karl	Riga	586.25	1	1	4
5	Erik	Utrecht	3086.25	0	1	1
6	Ralf	Koblenz	308.5	0	1	1
7	Joe	Wifi City	586	0	1	4

- **Projection** (select columns)
- **Selection** (select rows)
- **Join** (compose two tables by condition)

© 2016 Ralf Lämmel

Selection
(select rows)

	id	name	address	salary	manager	cid	did
▶	1	Craig	Redmond	30864	1	1	1
	2	Ray	Redmond	58641.75	1	1	2
	3	Klaus	Boston	5864	1	1	3
	4	Karl	Riga	586.25	1	1	4
	5	Erik	Utrecht	3086.25	0	1	1
	6	Ralf	Koblenz	308.5	0	1	1
	7	Joe	Wifi City	586	0	1	4

- **Projection** (select columns)
- **Selection** (select rows)
- **Join** (compose two tables by condition)

© 2016 Ralf Lämmel

Join
(compose two tables by condition)

id	name	address	salary	manager	cid	did
1	Craig	Redmond	30864	1	1	1
2	Ray	Redmond	58641.75	1	1	2
3	Klaus	Boston	5864	1	1	3
4	Karl	Riga	586.25	1	1	4
5	Erik	Utrecht	3086.25	0	1	1
6	Ralf	Koblenz	308.5	0	1	1
7	Joe	Wifi City	586	0	1	4

Company Name
ACME Corporation
ACME Corporation
ACME Corporation
ACME Corporation
ACME Corporation
ACME Corporation
ACME Corporation

- **Projection** (select columns)
- **Selection** (select rows)
- **Join** (compose two tables by condition)

© 2016 Ralf Lämmel

Relational schemas

Key terms

- Attributes (names)
- Attribute domains (types)
- **Relational schema (attribute-domain pairs)**
- Instance of relational schema (sets of tuples)

© 2016 Ralf Lämmel

The relational schema for 101companies

- Relational schemas (names only)
  - company (<u>id</u>, name)
  - department (<u>id</u>, name, *cid*, *did*)
  - employee (<u>id</u>, name, address, salary, manager, *cid*, *did*)

Key constraints:
Primary key (underlined) for identification of tuple
Foreign key (italics) for reference to another table

© 2016 Ralf Lämmel

Variation

- Relational schemas (names only)
  - ✦ company (<u>id</u>, name)
  - ✦ department (<u>id</u>, name, *cid*, *did*)
  - ✦ employee (<u>id</u>, name, address, salary, manager, *cid*, *did*)

Key constraints:
Primary key (underlined) for identification of tuple
Foreign key (italics) for reference to another table

© 2016 Ralf Lämmel

Variation

- Relational schemas (names only)
  - ✦ company (<u>id</u>, name)
  - ✦ department (<u>id</u>, name, *cid*, *did*, ***mid***)
  - ✦ employee (<u>id</u>, name, address, salary, manager, *cid*, *did*)

Key constraints:
Primary key (underlined) for identification of tuple
Foreign key (italics) for reference to another table

© 2016 Ralf Lämmel

Database programming with SQL (Structured Query Language)

- Represent schema in DDL subset of SQL
  - DDL - Data Definition Language
    - Part of SQL for data definition
- Represent population in DML subset of SQL
  - DML - Data Manipulation Language
    - Part of SQL for CRUD (Create, Read, Update, Delete)

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

© 2016 Ralf Lämmel

SQL DDL

© 2016 Ralf Lämmel

DDL statement for company

```
CREATE TABLE company (
    id INTEGER,
    name VARCHAR(100)
)
```

© 2016 Ralf Lämmel

More details

```
CREATE TABLE company (
    id INTEGER PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
)
```

© 2016 Ralf Lämmel

More details

```
CREATE TABLE company (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
)
```

© 2016 Ralf Lämmel

```
CREATE TABLE department (
    id INTEGER,
    name VARCHAR(100),
    cid INTEGER,
    did INTEGER,
)
```

© 2016 Ralf Lämmel

More details

```
CREATE TABLE department (
    id INTEGER PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    cid INTEGER NOT NULL,
    did INTEGER,
    FOREIGN KEY (cid) REFERENCES company(id),
    FOREIGN KEY (did) REFERENCES department(id)
)
```

© 2016 Ralf Lämmel

More details

```
CREATE TABLE department (
    id INTEGER PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL,
    cid INTEGER NOT NULL,
    did INTEGER,
    FOREIGN KEY (cid) REFERENCES company(id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (did) REFERENCES department(id)
      ON DELETE CASCADE ON UPDATE CASCADE
)
```

© 2016 Ralf Lämmel

```
CREATE TABLE employee (
    id INTEGER PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    address VARCHAR(50) NOT NULL,
    salary DOUBLE NOT NULL,
    manager BOOL NOT NULL,
    cid INTEGER NOT NULL,
    did INTEGER NOT NULL,
    FOREIGN KEY (cid) REFERENCES company(id),
    FOREIGN KEY (did) REFERENCES department(id)
)
```

© 2016 Ralf Lämmel

DDL language summary

- CREATE TABLE
  - INTEGER, VARCHAR(…), DOUBLE
  - NOT NULL
  - UNIQUE
  - PRIMARY / FOREIGN KEY
  - ON DELETE / UPDATE CASCADE

© 2016 Ralf Lämmel

SQL DML

© 2016 Ralf Lämmel

CRUD

- C: Create (SQL: Insert)
- R: Read (SQL: Select)
- U: Update
- D: Delete

© 2016 Ralf Lämmel

```
INSERT INTO company (name)
 VALUES ("Acme Corporation")
```

Insert a new company into the corresponding table.

© 2016 Ralf Lämmel

```
INSERT INTO department (name,cid)
 VALUES ("Research",1)
INSERT INTO department (name,cid)
 VALUES ("Development",1)
...
```

Insert several departments into the corresponding table.

© 2016 Ralf Lämmel

```
SELECT * FROM DEPARTMENT
```

id,name,cid,did
1,Research,1,NULL
2,Development,1,NULL
3,Dev1,1,2
4,Dev1.1,1,3

List of tuples of the department table.

© 2016 Ralf Lämmel

SELECT SUM(salary) FROM employee

Select all employees, project to their salaries, and sum them up.

© 2016 Ralf Lämmel

```
SELECT SUM(salary) FROM employee
 WHERE cid = 1
```

Retrieve only salaries of a specific company.

© 2016 Ralf Lämmel

```
SELECT SUM(salary) FROM employee
 WHERE cid =
  (SELECT id FROM company
   WHERE name = "Acme Corporation")
```

Use a nested query to determine the company id.

© 2016 Ralf Lämmel

```
UPDATE employee
 SET salary = salary / 2
```

Cut all salaries in half.

© 2016 Ralf Lämmel

```
UPDATE employee
 SET salary = salary / 2
 WHERE cid = 1
```

Limit update to employees with company id = 1.

© 2016 Ralf Lämmel

```
UPDATE employee
 SET salary = salary / 2
 WHERE cid =
  (SELECT id FROM company
   WHERE name = "Acme Corporation")
```

Use a nested query to determine the company id.

© 2016 Ralf Lämmel

Embedding SQL into Java with JDBC

http://lol companies.org/wiki/Contribution:simplejdbc

Total all salaries

```
public static double total(MyConnection myConnection, String companyName){
   double total = 0;
   try {
   String query = "SELECT salary FROM employee "
         + "WHERE cid = (SELECT id FROM company WHERE name = ?);";
   PreparedStatement pstmtEmployees = myConnection.getConn()
         .prepareStatement(query);
   pstmtEmployees.setString(1, companyName);
   ResultSet salaries = pstmtEmployees.executeQuery();
   while (salaries.next())
      total += salaries.getDouble("salary");
   } catch (SQLException e){
      e.printStackTrace();
   }
   return total;
}
```

We do not use SQL's SUM here so that we can demonstrate JDBC's ResultSets with the example.

© 2016 Ralf Lämmel

Library support for database programming

- JDBC (part of Java Core API)
  - Connections to databases
  - Submit SQL statements
  - Retrieve results
- MySQL connector (part of demo project)
  - JDBC-based driver for MySQL

© 2016 Ralf Lämmel

Embedded SQL

- Important JDBC types
  - Connection
  - Statement (different forms thereof)
  - ResultSet (for queries in particular)
  - SQL Exception

© 2016 Ralf Lämmel

Cut all salaries in half

```
public static void cut(MyConnection myConnection, String companyName) {
   try {
      // cut salaries in all employee columns
      String sqlCut = "UPDATE employee SET salary = salary / 2 "
            + "WHERE cid = (SELECT id FROM company WHERE name = ?);";
      PreparedStatement pstmtEmployees = myConnection.getConn()
            .prepareStatement(sqlCut);
      pstmtEmployees.setString(1, companyName);
      pstmtEmployees.executeUpdate();
   } catch (SQLException e) {
      e.printStackTrace();
   }
}
```

Exercise: understand the notion of injection attacks and argue how "prepared statements" help avoiding the problem.

© 2016 Ralf Lämmel

Technological space travel

© 2016 Ralf Lämmel

Software developers need to be space travelers!

http://www.nasa.gov/images/content/63114main_highway_med.jpg

© 2016 Ralf Lämmel

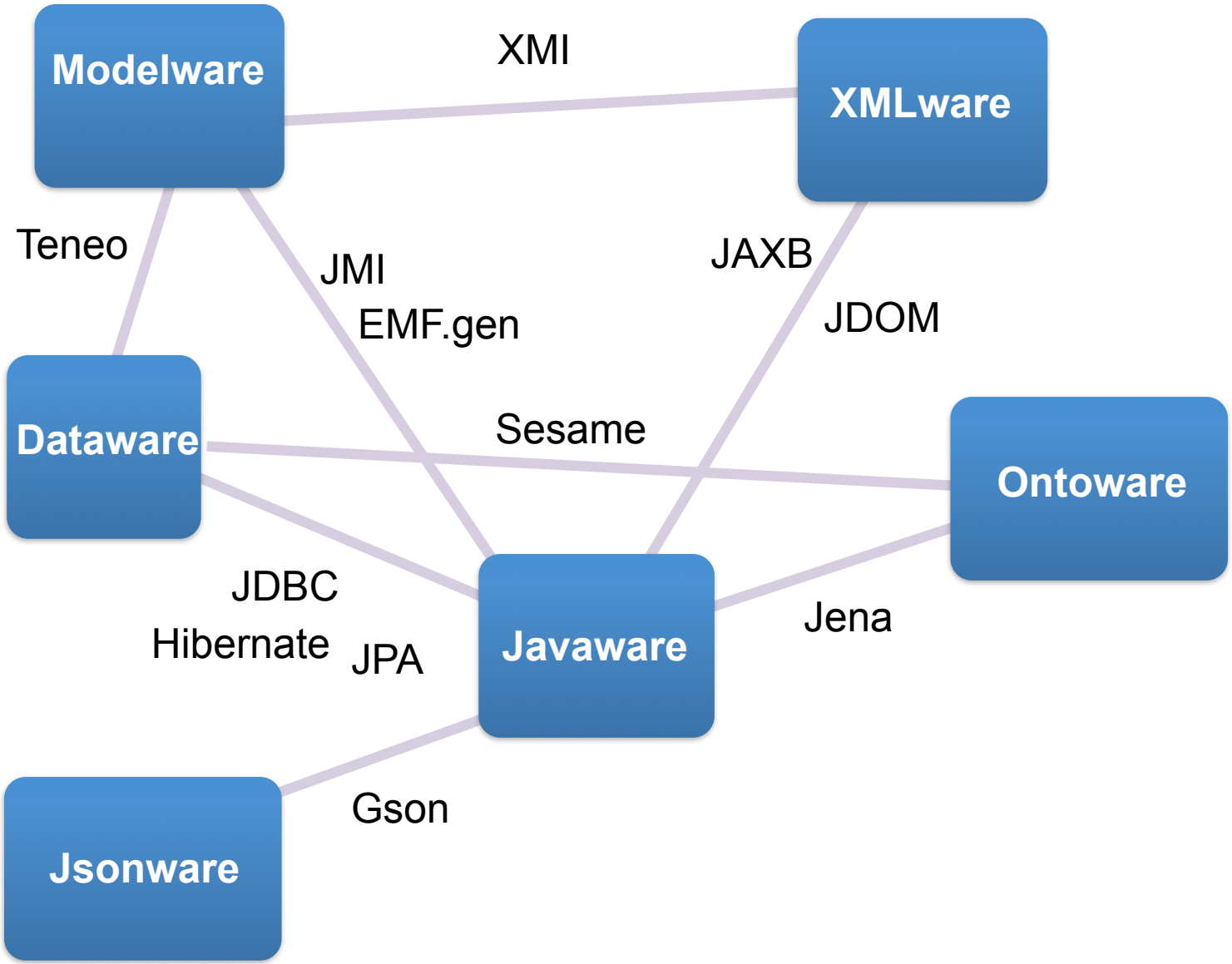
Why is space travel a burden?

- Terminology, notation, conventions, culture
- Computational models, data models, type systems
- Concrete technologies and languages
- Accidental complexity
- …

This also goes by the name O/R/X/… impedance mismatch

© 2016 Ralf Lämmel

Examples bridges for space travel



© 2016 Ralf Lämmel

Use 101companies as a knowledge resource for technological space travel

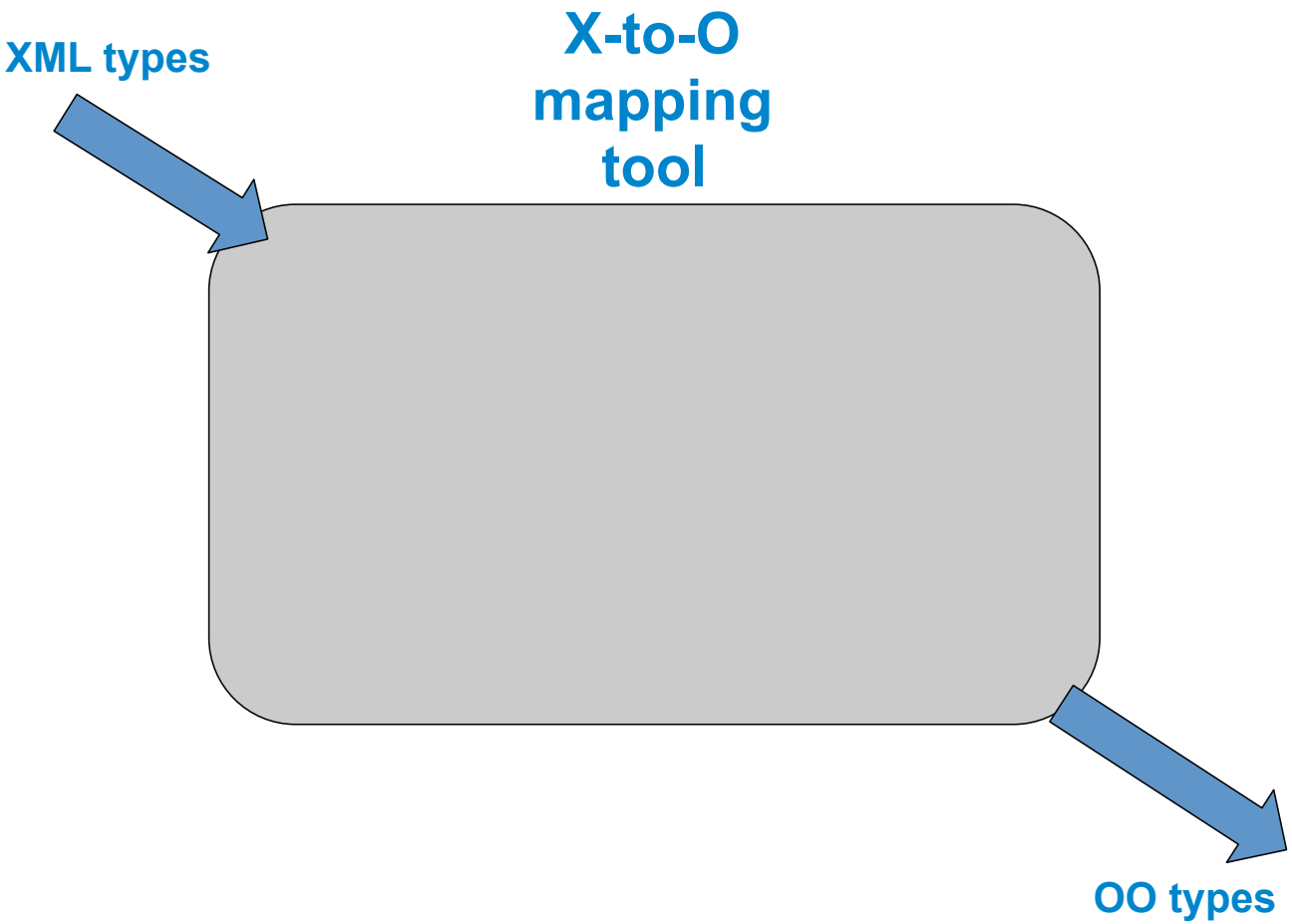
http://upload.wikimedia.org/wikipedia/en/thumb/6/66/E_t_the_extra_terrestrial_ver3.jpg/220px-E_t_the_extra_terrestrial_ver3.jpg

© 2016 Ralf Lämmel

XML data binding
(aka O/X mapping)

© 2016 Ralf Lämmel

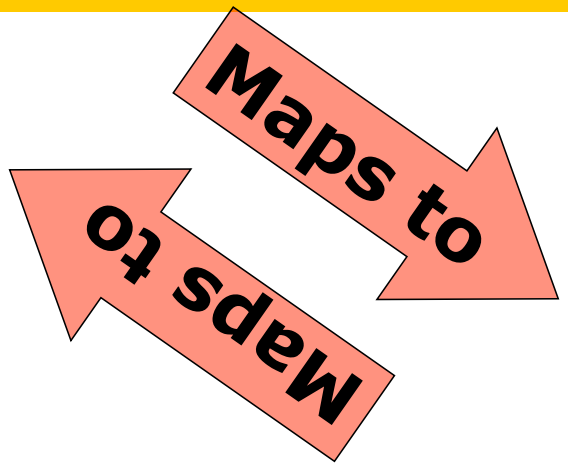
X-to-O mapping



© 2016 Ralf Lämmel

O/X type mapping

```
<element name=“point">
  <complexType>
    <sequence>
      <element name="x" type=“xs:int"/>
      <element name="y" type="xs:int"/>
    </sequence>
  </complexType>
</element>
```




```
public class Point
{
   public int x;
   public int y;
}
```

© 2016 Ralf Lämmel

```
public class Company {                                                    O
   private String name;
   private List<Department> depts = new LinkedList<Department>();
   public String getName() { return name; }
   public void setName(String name) { this.name = name; }
   public List<Department> getDepts() { return depts; }
}
public class Department { ... }
public class Employee { ... }
```

```
<xs:element name="company">                                               X
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="name"/>
         <xs:element maxOccurs="unbounded" minOccurs="0" ref="department"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="department"> ... </xs:element>
<xs:complexType name="employee"> ... </xs:complexType>
```

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

© 2016 Ralf Lämmel

XML data binding

- Directions for XML data binding
  - Generate classes from XML schemas.
  - Generate XML schemas from classes.
  - Describe mapping only without generation.
- Motivations for XML data binding
  - Support valid XML output.
  - Hide XML in OO programming.
  - Use XML-based object serialization.

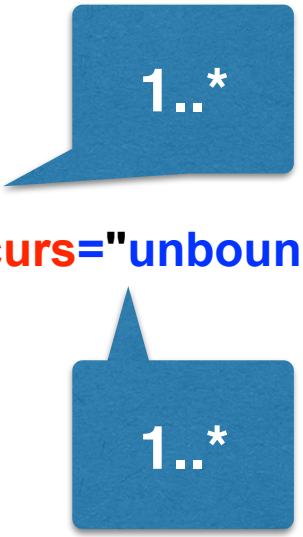
© 2016 Ralf Lämmel

The O/X impedance mismatch

© 2016 Ralf Lämmel

How to map this element declaration?

```
<xs:element name="s">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="a" type="x"/>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="b" type="y" maxOccurs="unbounded"/>
        <xs:element name="c" type="z"/>
      </xs:sequence>
    </xs:sequence>
  </xs:complexType>
</xs:element>`
```



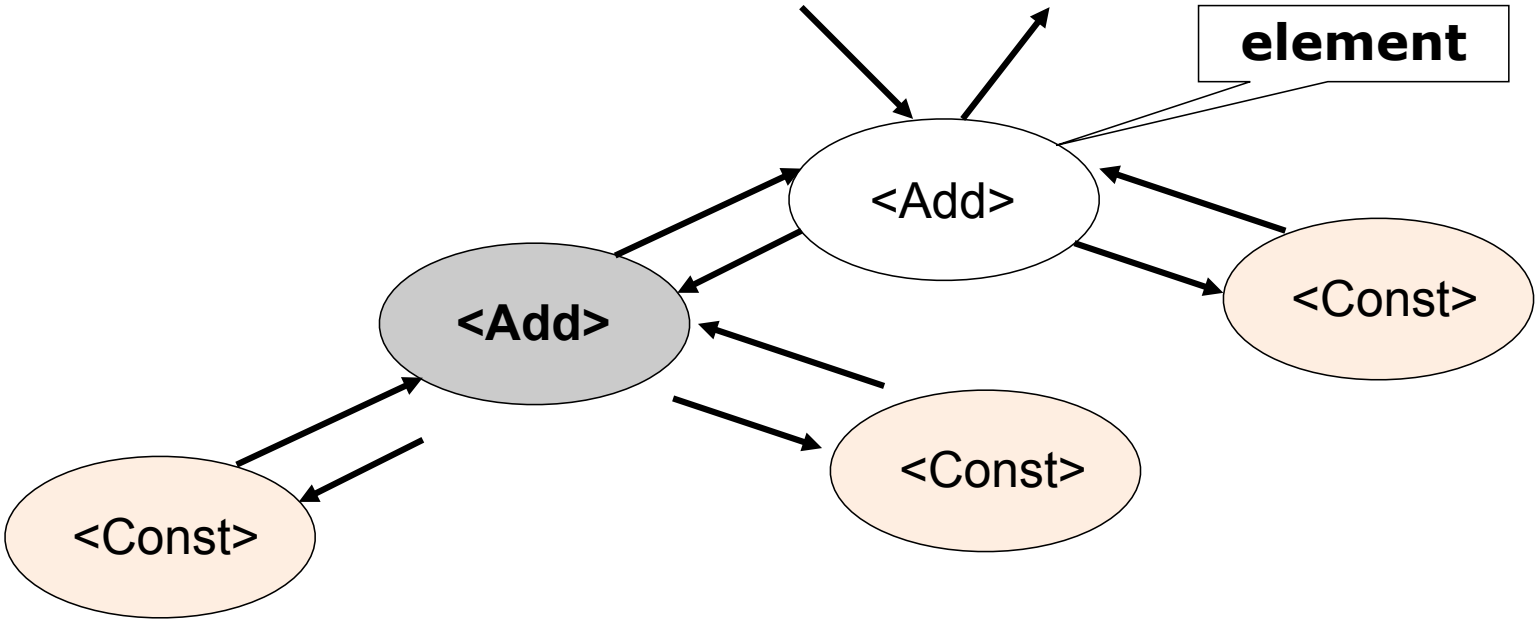
© 2016 Ralf Lämmel

How to map “s = a:x (b:y$^{1..*}$ c:z)$^{1..*}$ ”?

```
class s {
    x a;
    y[] b;
    z[] c;
}
```

© 2016 Ralf Lämmel

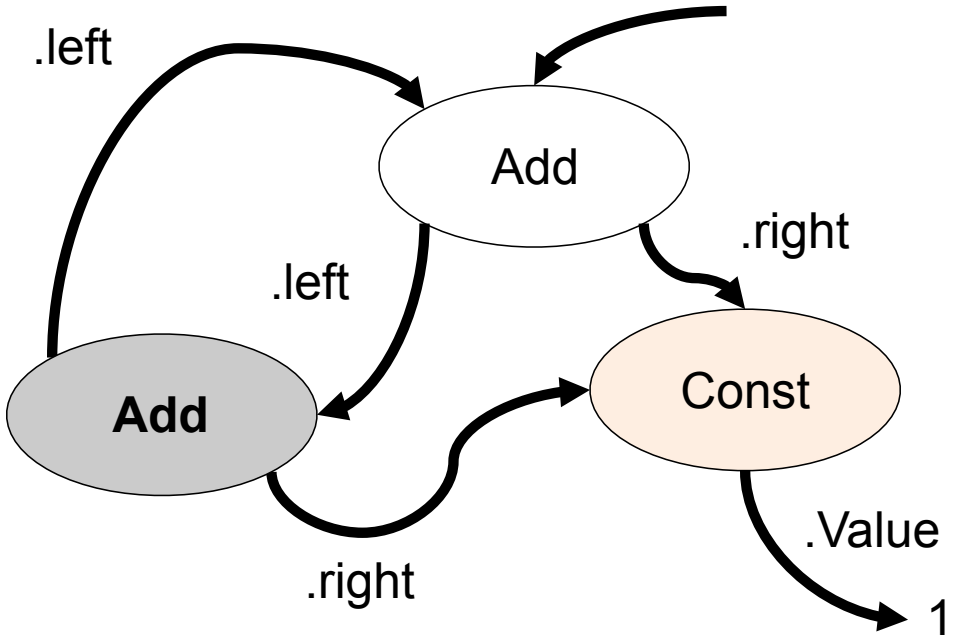
XML data binding is difficult because XML are parented trees whereas …



Thanks to Erik Meijer for contributing this slide or part thereof.

© 2016 Ralf Lämmel

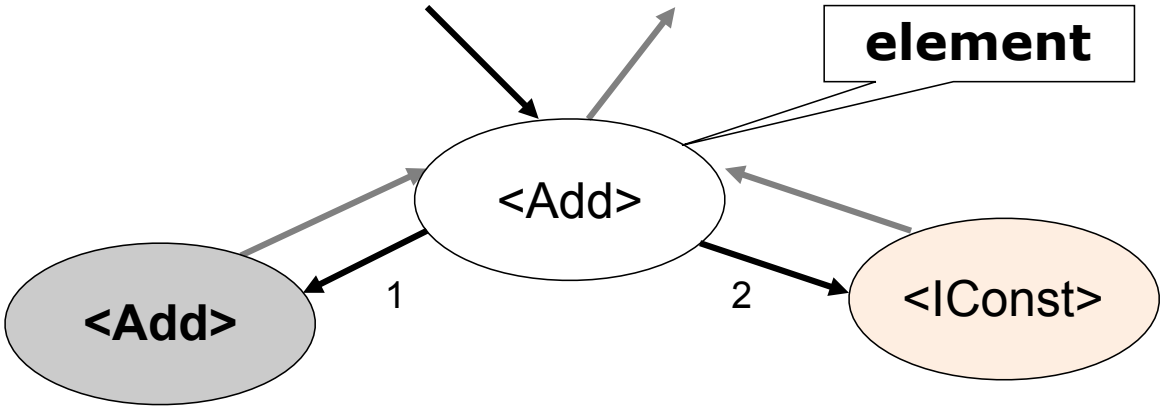
Objects are freewheeling graphs.



Thanks to Erik Meijer for contributing this slide or part thereof.

© 2016 Ralf Lämmel

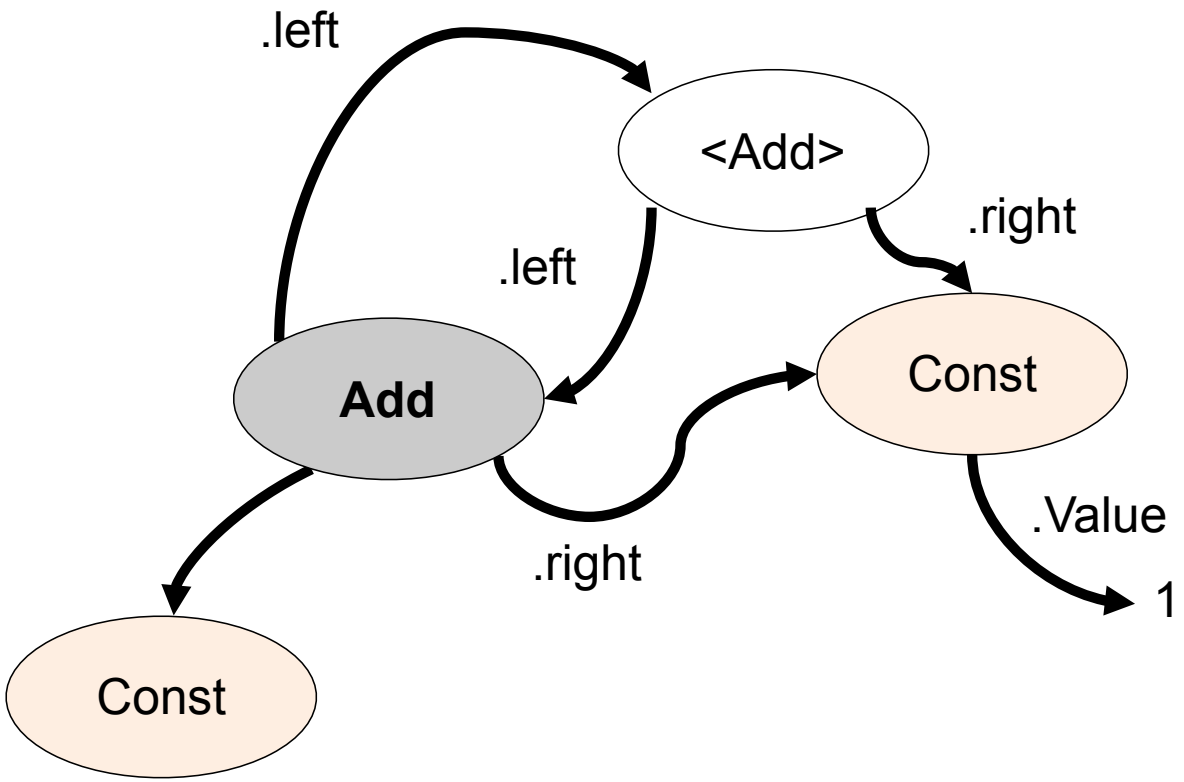
Its also difficult because XML trees are node-labeled whereas …



Thanks to Erik Meijer for contributing this slide or part thereof.

© 2016 Ralf Lämmel

Object graphs are edge-labeled.



Thanks to Erik Meijer for contributing this slide or part thereof.

© 2016 Ralf Lämmel

O/R mapping (related to persistence)

Object-relational mapping (OR mapping) products integrate object programming language capabilities with relational databases managed by Oracle, DB2, Sybase, and other RDBMSs. Database objects appear as programming language objects in one or more existing object programming languages.
[Source: http://www.service-architecture.com/articles/object-relational-mapping]

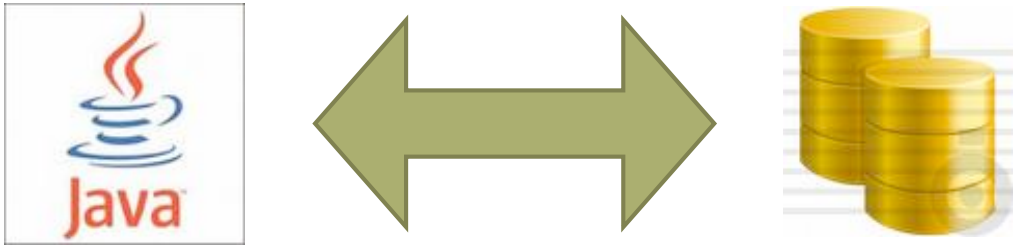
© 2016 Ralf Lämmel

Elevator pitch

How to persist objects in database tables?

How to map object models to relational schemas?

... or the other way around?



© 2016 Ralf Lämmel

Exercises

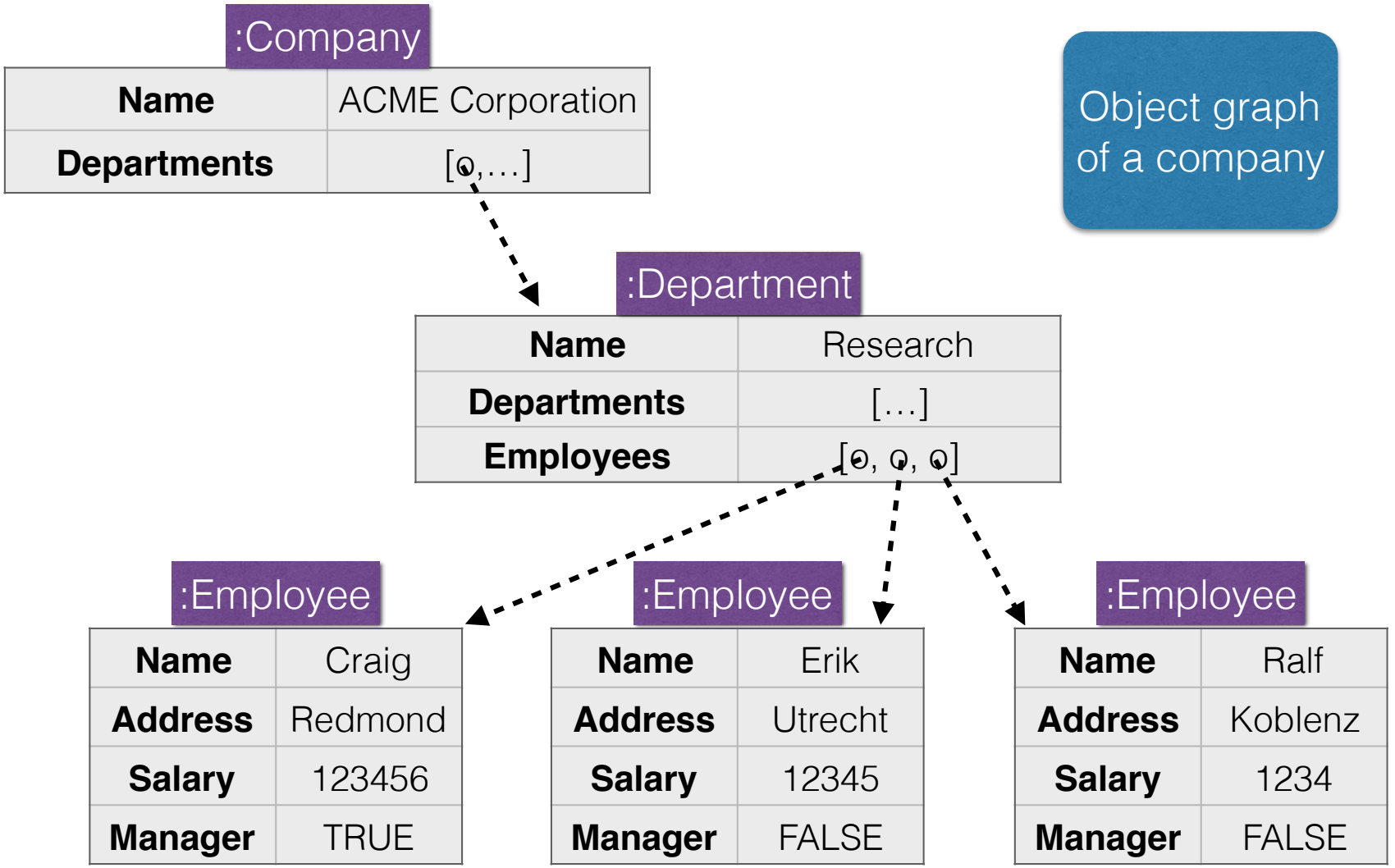
Bing for this statement and read about it:

"**O/R Mapping … is the Vietnam of Computer Science**"!

Read about the "O/R impedance mismatch"!

© 2016 Ralf Lämmel

Why is O/R mapping challenging?
Objects *vs.* tables



© 2016 Ralf Lämmel

Why is O/R mapping challenging?
Objects *vs.* **tables**

Company

Id	Name
88	ACME Corporation
…	…

Department

Id	Name	Department	Company
42	Research	NULL	88
43	Development	NULL	88
44	Dev1	43	88
…	…	…	…

Employee

Id	Name	Address	Salary	Manager	Department
1	Craig	Redmond	123456	TRUE	42
2	Erik	Utrecht	12345	FALSE	42
3	Ralf	Koblenz	1234	FALSE	42
…	…	…	…	…	…

Arrows are inverse when compared to objects

Relational tables for a company

© 2015 Ralf Lämmel

Why is O/R mapping challenging?

O	R
Collections refer to members	Rows use foreign key
no NOT NULL	NOT NULL
Non-primitive attributes	Normal form
In memory with GC	Huge data volume
…	…

© 2016 Ralf Lämmel

Poor men's O/R mapping

101implementation:jdbc2

© 2016 Ralf Lämmel

Employee POJOs

```
public class Employee {

   private int id;
   private String name;
   private String address;
   private double salary;

   public int getId() { return id; }
   public void setId(int id) { this.id = id; }
   public String getName() { return name; }
   public void setName(String name) { this.name = name; }
   public String getAddress() { return address; }
   public void setAddress(String address) { this.address = address; }
   public double getSalary() { return salary; }
   public void setSalary(double salary) {  this.salary = salary; }
}
```

© 2016 Ralf Lämmel

Load employee

```
int id = employee.getId();
String sql = "SELECT * FROM employee WHERE id = ?";
PreparedStatement stm =
 myConnection.getConn().prepareStatement(sql);
stm.setInt(1, id);
ResultSet result = stm.executeQuery();
result.next();
employee.setSalary(result.getDouble("salary"));
employee.setName(result.getString("name"));
employee.setAddress(result.getString("address"));
```

© 2016 Ralf Lämmel

Change tracking

```
public class Employee {

   // Some additional / revised members are shown.

   private boolean changed;

   public void setName(String name) {
      this.name = name;
      changed = true;
   }

   public void setChanged(boolean changed) { this.changed = true; }
   public boolean isChanged() { return changed; }
}
```

© 2016 Ralf Lämmel

Save employee

```
if (employee.isChanged()) {
  if (employee.getId() == 0) {
    String sqlInsert = "INSERT employee ...";
    ...
    stmInsert.execute();
       String sqlSelectId = "SELECT max(id) AS maxid FROM employee";
       ...
       ResultSet maxid = pstmtSelectId.executeQuery();
       maxid.next();
       employee.setId(maxid.getInt("maxid"));
    } else {
       String sqlUpdate = "UPDATE employee ...";
       ...
       stmUpdate.executeUpdate();
    }
    employee.setChanged(false);
}
```

Is that a robust way of retrieving the id?

This update could fail. What to do?

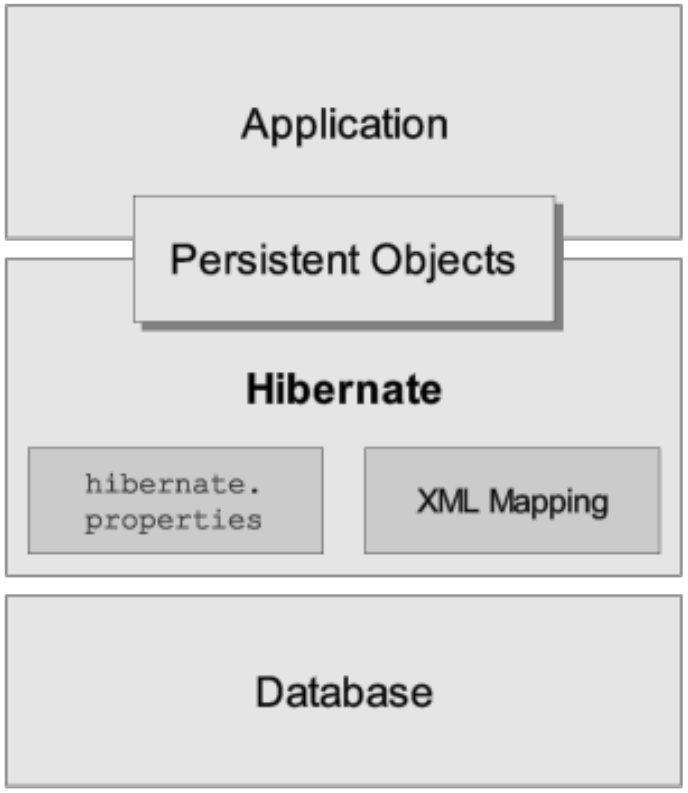
© 2016 Ralf Lämmel

Hibernate

© 2016 Ralf Lämmel

Hibernate: Simplifying persistence in Java

Source:
Hibernate Reference Documentation

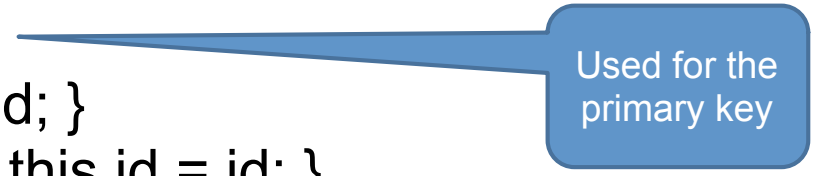


- Capabilities
  - Store objects in database tables, one object per row.
  - Restore objects in different programs / runs.

© 2016 Ralf Lämmel

A persistent class

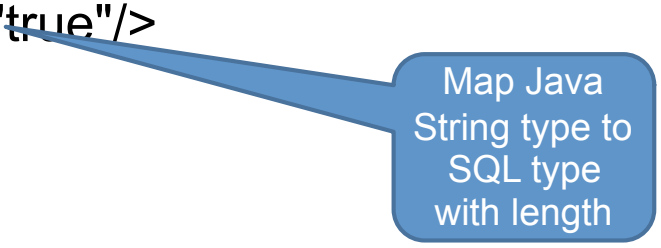
```
public class Cat {
   private String id;
   private String name;
   private char sex;
   private float weight;
   public String getId() { return id; }
   private void setId(String id) { this.id = id; }
   // … other getters and setters …
}
```



© 2016 Ralf Lämmel

Metadata for O/R mapping

```
<hibernate-mapping>
   <class name="Cat" table="CAT">
      <id name="id" type="string" unsaved-value="null" >
         <column name="CAT_ID" sql-type="char(32)" not-null="true"/>
         <generator class="uuid.hex"/>
      </id>
      <property name="name">
         <column name="NAME" length="16" not-null="true"/>
      </property>
      <property name="sex"/>
      <property name="weight"/>
   </class>
</hibernate-mapping>
```



© 2016 Ralf Lämmel

A database table

The CAT table in the database

```
 Column | Type                        | Modifiers
--------+-----------------------------+-----------
 cat_id | character(32)               | not null
 name   | character varying(16)       | not null
 sex    | character(1)                |
 weight | real                        |
```

Indexes: cat_pkey primary key btree (cat_id)

© 2016 Ralf Lämmel

A Hibernate session (in Java code)

```
Session session = HibernateUtil.currentSession();
Transaction tx= session.beginTransaction();

Cat princess = new Cat();
princess.setName("Princess");
princess.setSex('F');
princess.setWeight(7.4f);

session.save(princess);
tx.commit();
HibernateUtil.closeSession();
```

Set up session and begin transaction

Regular OO code

Make a n object persistent and commit changes.

© 2016 Ralf Lämmel

How to retrieve persistent objects?
Use HQL (Hibernate query language).

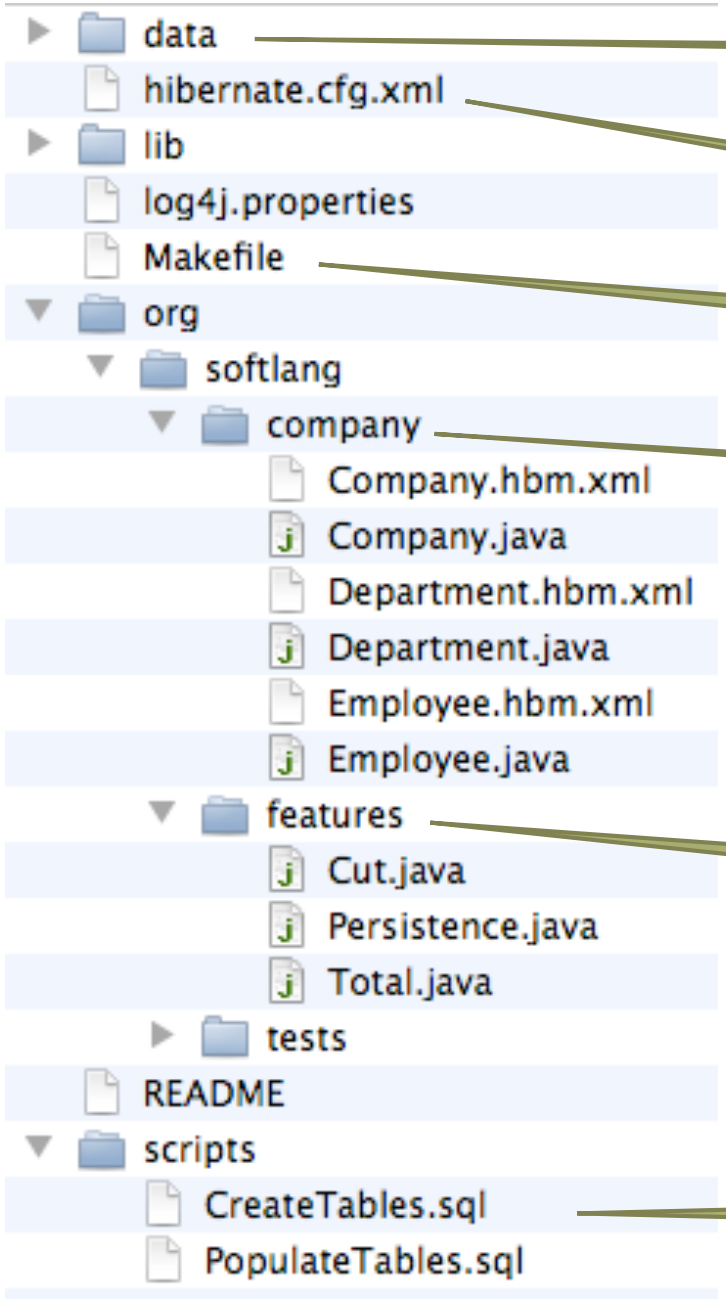
```
Query query = session.createQuery(
   "select c from Cat as c where c.sex = :sex");
query.setCharacter("sex", 'F');
for (Iterator it = query.iterate(); it.hasNext();) {
   Cat cat = (Cat) it.next();
   out.println("Female Cat: " + cat.getName() );
}
```

© 2016 Ralf Lämmel

http://101companies.org/wiki/

Contribution:hibernate

© 2016 Ralf Lämmel



Data dir to be used by HSQLDB

Hibernate **configuration**

See **Makefile** for usage scenario

Hibernate-enabled object model with mapping files

The usual features.

SQL scripts with relational schema and instance data.

© 2016 Ralf Lämmel

Employee POJOs

```
public class Employee {

   private long id;
   private String name;
   private String address;
   private double salary;
   private boolean manager;

   public long getId() {
      return id;
   }
   @SuppressWarnings("unused")
   private void setId(long id) {
      this.id = id;
   }
  ...
}
```

© 2016 Ralf Lämmel

Mapping for employees

```
<hibernate-mapping>

  <class name="org.softlang.company.Employee" table="EMPLOYEE">

    <id name="id" column="ID">
      <generator class="native" />
    </id>

    <property name="name" />
    <property name="address" />
    <property name="salary" />
    <property name="manager" />

  </class>

</hibernate-mapping>
```

© 2016 Ralf Lämmel

Mapping for departments

```
<hibernate-mapping>

  <class name="org.softlang.company.Department" table="DEPARTMENT">

    <id name="id" column="ID">
      <generator class="native" />
    </id>

    <property name="name" />

    <set name="employees" cascade="all">
      <key column="DEPT_ID" />
      <one-to-many class="org.softlang.company.Employee" />
    </set>

    <set name="subdepts" cascade="all">
      <key column="DEPT_ID" />
      <one-to-many class="org.softlang.company.Department" />
    </set>

  </class>

</hibernate-mapping>
```

© 2016 Ralf Lämmel

Hibernate configuration

```
<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings. -->
    <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
    <property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>

    <!-- SQL dialect -->
    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

    <!-- Create the database schema, if needed; update otherwise -->
    <property name="hbm2ddl.auto">update</property>

    <!-- Mapping files in the project -->
    <mapping resource="org/softlang/company/Company.hbm.xml" />
    <mapping resource="org/softlang/company/Department.hbm.xml" />
    <mapping resource="org/softlang/company/Employee.hbm.xml" />

    ...
  </session-factory>
</hibernate-configuration>
```

© 2016 Ralf Lämmel

So what's O/R mapping?

Wikipedia's definition

Ralf's definition (relatively naïve version):

- Category 1:
  - Start from (idiomatic) classes.
  - Map object model to relational schema.
  - Deploy relational schema in database.
  - Encode CRUD in OO code.
  - Add transactions to OO code.
- Category 2:
  - Start from database (schema, instance, SPROC).
  - Derive object model to encapsulate data access.
  - Continue as above ...
- Category 1 + 2: classes and tables given, mapping wanted.
- Category 2':
  - Like Category 2 but ...
  - ER/relational model-level mapping.
  - Coverage of distributed database and data integration.

© 2016 Ralf Lämmel

Perhaps, don’t use Hibernate or any „explicit“ O/R mapping. Rather, leverage „implicit“ O/R mapping as integrated, for example, with popular web-application frameworks such as Django and Rails (but this also comes with unique challenges).

© 2016 Ralf Lämmel

Other mapping options:

Object/JSON mapping

„Implicit“ Object/Relational mapping

Acknowledgement: Jürgen Starek has significantly contributed to this content.

© 2016 Ralf Lämmel

End of lecture

© 2016 Ralf Lämmel