26. Data-Oriented Design Methods

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de/teaching/swt2
Version 15-0.7, 1/12/16

1) Jackson Structured Programming (JSP) and Jackson Structured Diagrams (JSD)
2) Grammar-Driven Programming
3) Extensibility of JSP and Grammar-Based Applications

Obligatory Reading

- Ghezzi Ch. 3.3, 4.1-4, 5.5
- Pfleeger Ch. 4.1-4.4, 5
- M. Jackson. The Jackson Development Methods. Wiley Encyclopedia of Software Engineering. J. Marciniak (ed.), 1992
  - http://www.jacksonworkbench.co.uk/stevefergspages/jackson_methods/index.html
  - http://www.ferg.org%2Fpapers%2Fjackson--the_jackson_development_methods.pdf

23.1 Jackson Structured Programming as Data-Oriented Development with Regular Data

Data-Oriented Development

- **Data-oriented development** focuses first on the development of a data structure
  - Path specifications with automata
  - Tree specification withstring grammars or tree grammars
  - Attributed tree specification with attributed grammars
  - Graph specifications with graph grammars and graph transformation systems (e.g., reducible graphs)
- Second step: Derive a visiting algorithm that works on all elements of the data structure in a pre-defined, specified way (similar to design pattern Visitor)
- Surprising: Grammars cannot only be used to parse strings, but to specify the **walk order** of a visiting algorithm!

How is the data structured?
so that the algorithm can homomorphically be derived from its structure

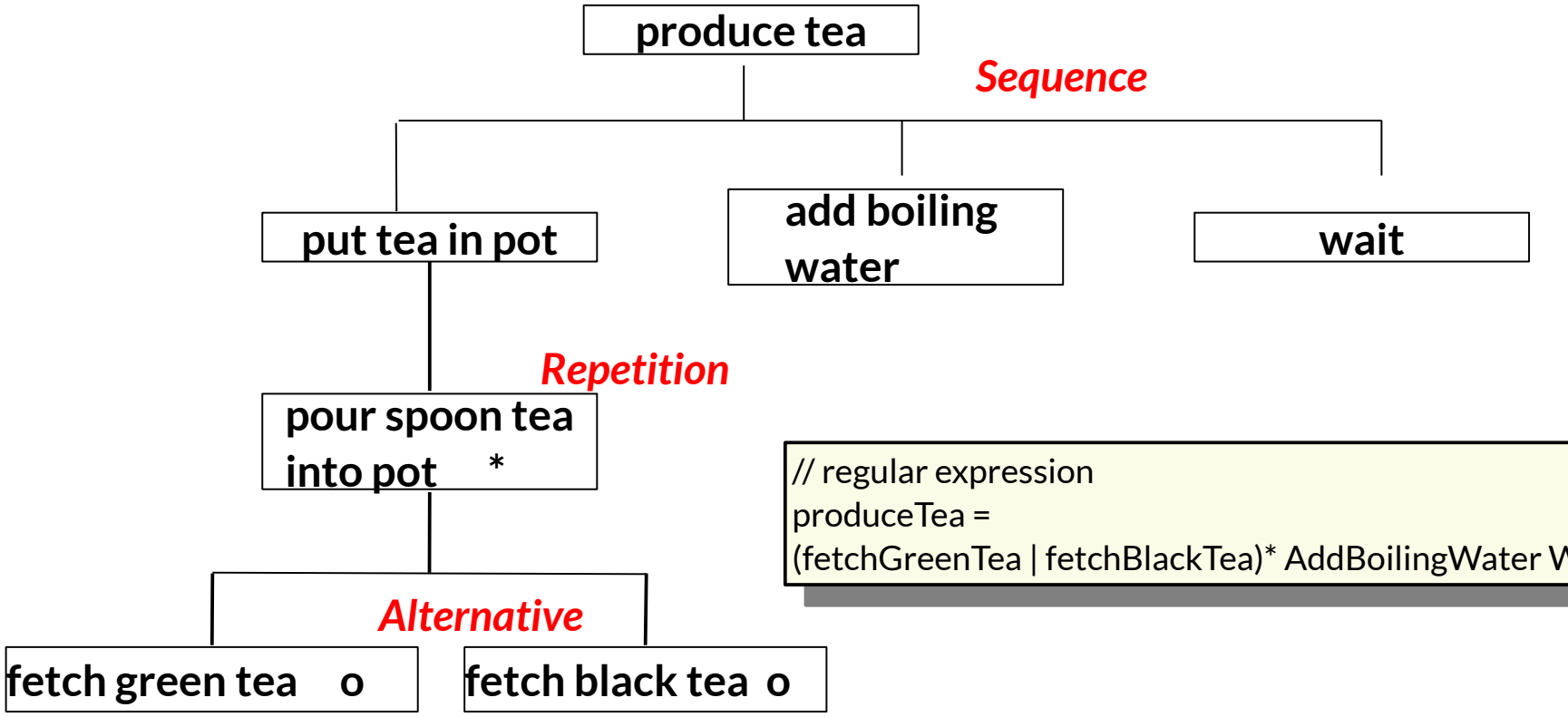
Example for Data-Oriented Design: Jackson Structured Programming JSP

- Data-oriented developing with hierarchical tree diagrams, a variant of a function/action tree
- The tree defines a ***walk order*** over a sequence of data elements or an event stream from which code is generated
  - JSP was one of the earliest model-driven development methods (from specifications, code is generated)

How is the data structured?
so that the algorithm can homomorphically be derived from its structure

Jackson Structured Diagrams (Jackson Process Trees)

- A **Jackson Structured Diagram (JSD Jackson Process Tree)** is a function free with iteration and alternatives. Its tree constructors stem from regular expressions:
  - **Sequence** transforms to sequenced statements
  - **Repetition** * transforms to loops or recursion (Kleene star)
  - **Alternative** o transforms to if- and case-instructions




```
// regular expression
produceTea =
(fetchGreenTea | fetchBlackTea)* AddBoilingWater Wait
```

Example for Data-Oriented Design: Jackson Structured Programming JSP

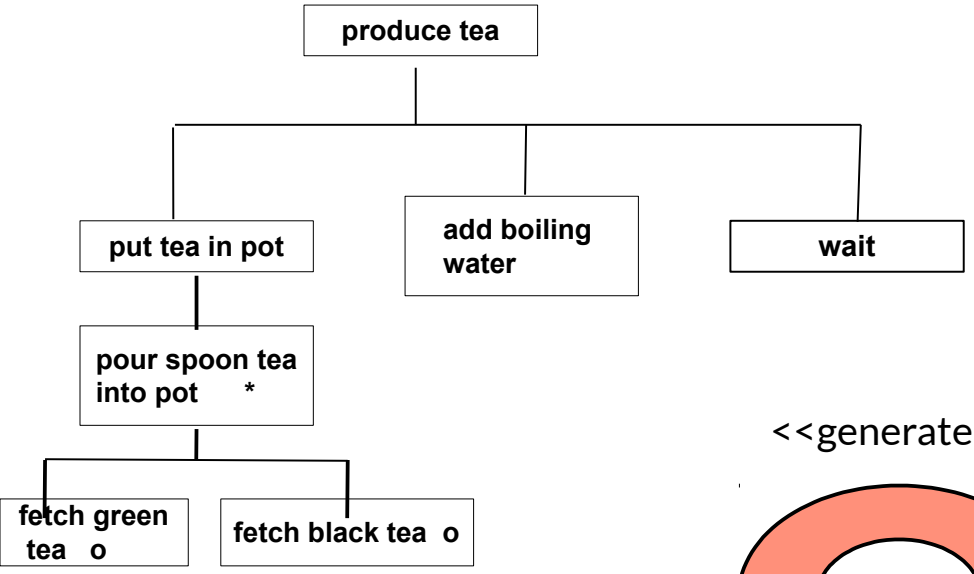
- **Notation:**
  - **Jackson Structured Diagrams JSD (regular actions),** equivalent to regular expressions on actions and finite state machines
- **Development Process:**
  - **Elaboration**: Draw JST trees for inputs and outputs
  - **Transformation**: Merge them
  - Elaboration: List the operations and allocate to program parts
  - **Elaboration**: Convert program to code (generate code)
  - **Elaboration**: Add conditions
- **Heuristics**:
  - Readahead
  - Backtracking
  - Program inversion if structure of input does not match output

When Should JSP Be Applied?

- JSP is good for problems that are “governed” by a data structure that corresponds to a regular expression:
  - if data has the structure of a regular expression
  - and input is homomorphic to output
  - -> Algorithm becomes homomorphic to data structure
- JST can describe the activity in a DFD (instead of minispecs in pseudocode)
  - Then, input is read from the input channels until end-of-stream
  - Output is produced by the JST
- Table processing in information systems is a perfect application area
  - COBOL is still being used in these information systems
  - DFD form the data flow
  - JSP is the specification of the elementary activities
  - The generated implementation is in COBOL or another imperative language

Deriving a Regular Grammar from a JSD Tree

- The generated grammar can be fed into a *parser generator* to produce a *parser* recognizing the order of events, e.g., www.antlr.org



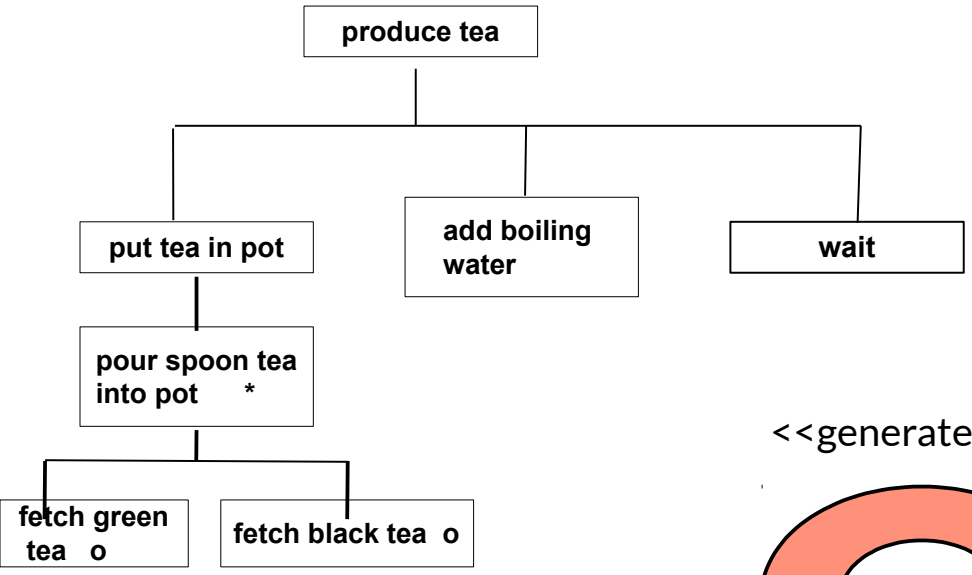
<<generate>>

```
Grammar TeaPot {
RULES
ProduceTea ::= PutTeaInPot
      AddBoilingWater  Wait .

PutTeaInPot ::=
    PourSpoonTeaIntoPot* .

PourSpoonTeaIntoPot ::=
       FetchGreenTea
    |  FetchBlackTea
    .
}
```

Deriving a System of Procedures from the JSD Tree



<<generate>>

```
procedure ProduceTea() {
   PutTeaInPot();
   AddBoilingWater();
   Wait();
}
procedure PutTeaInPot() {
    while (condition) {
            PourSpoonTeaIntoPot();
}
procedure PourSpoonTeaIntoPot() {
    if (condition)
        FetchGreenTea();
    else
        FetchBlackTea(),
}
```

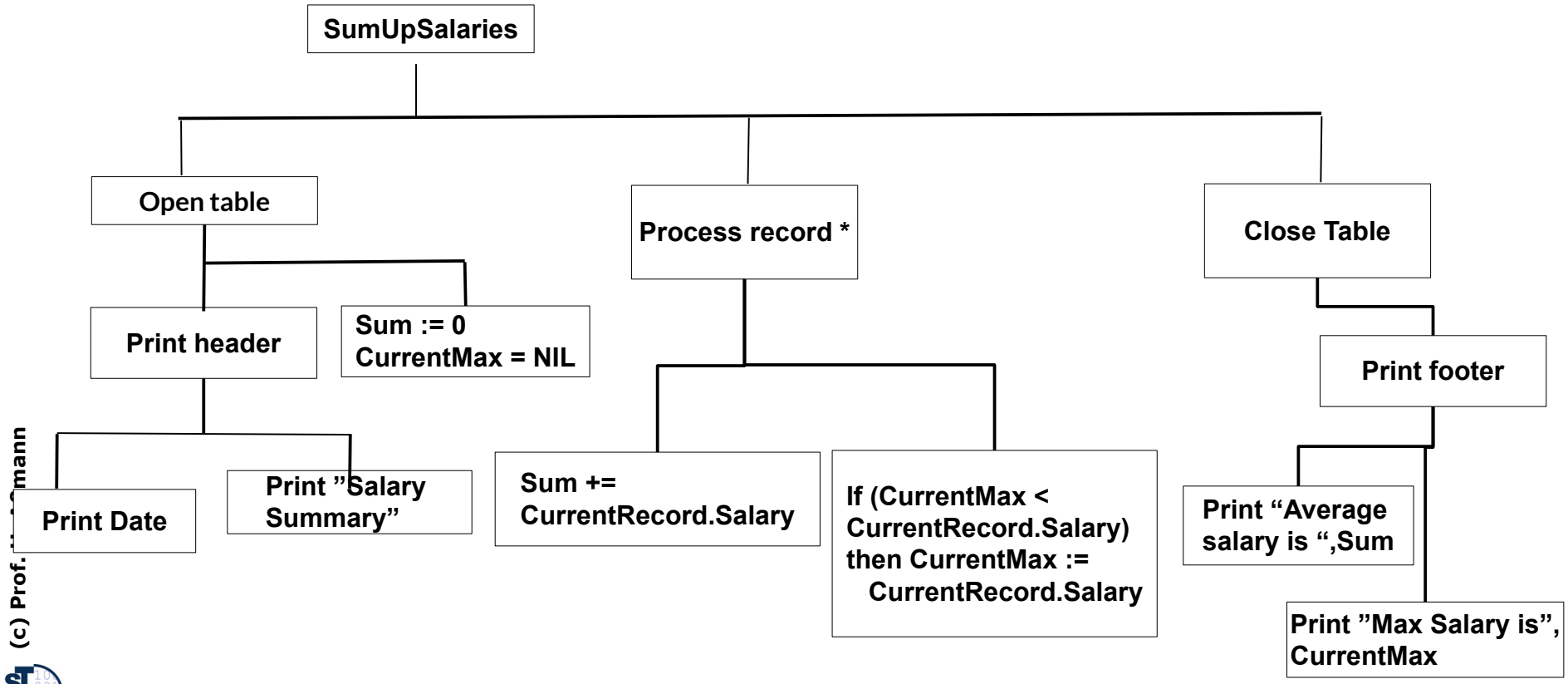
Table- and Record-Manipulation Programs in Information Systems

- Many information systems rely on *relational data processing* with tables containing records (tuples) with information about employees, insured persons, members of networks, unemployed people, customers, etc
- Algorithms on these tables with recorrds can easily be expressed by JSP process trees
  - „which persons earn more than 1500€ in our company? (threshold query)
  - „who earns most of our Austrian employees“ (max query)
  - „compute the average salary of our employees“ (avg operator)
  - „how much would a 5% increase of salary cost our company?“ (map-reduce operator)

#id	Name	FirstName	Street	Town	Salary
12	John	Silver	Obergasse 2a	Wien	1200€
13	Bobby	Brown	Traubenweg 12	Bad Tölz	600€
14	Frank	Foster	Blumenweg 6	München	2000€
20	Sue	Smith	Tulpengasse 3	Füssen	2300€
25	Mary	Miller	Heurigenweg 2	Linz	1500€

A Table-Processing Program (Sum and Max)

- Operators Sum, Max, Min, Avg, Map, Reduce, Map-Reduce, Group-By are simple to use
- JSP was used to generate COBOL applications in banks and insurances
- → JSP was also the first Big-Data approach

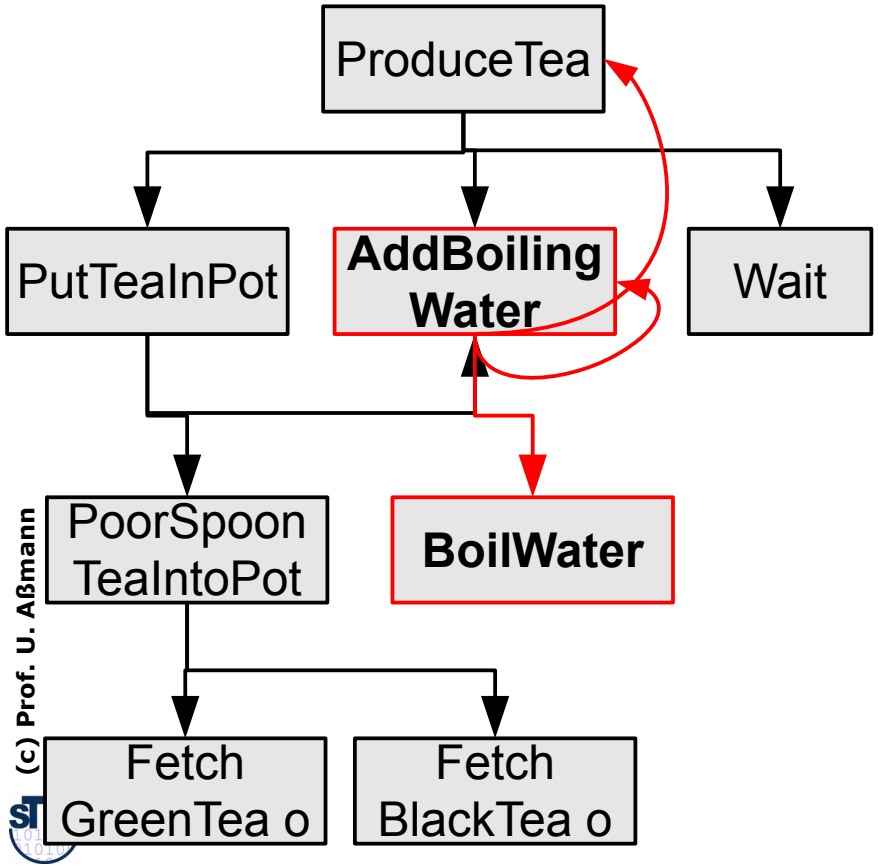


26.2 Programming with Data Structure Grammars

Grammars can indirectly specify a Visitor for a data structure

Grammar-Driven Programming

- A **context-free grammar** extends a regular grammar with free recursion: left, right, intertwined
- Like in the regular grammar case, from the grammar similar code can be derived
  - Visitors, parsers, generators




```
Grammar TeaPot {
RULES
ProduceTea ::= PutTeaInPot
      AddBoilingWater  Wait .

PutTeaInPot ::= PourSpoonTeaIntoPot*
     AddBoilingWater .

PourSpoonTeaIntoPot ::=
       FetchGreenTea | FetchBlackTea.

AddBoilingWater ::=
     BoilWater AddBoilingWater
     ProduceTea.
}
```

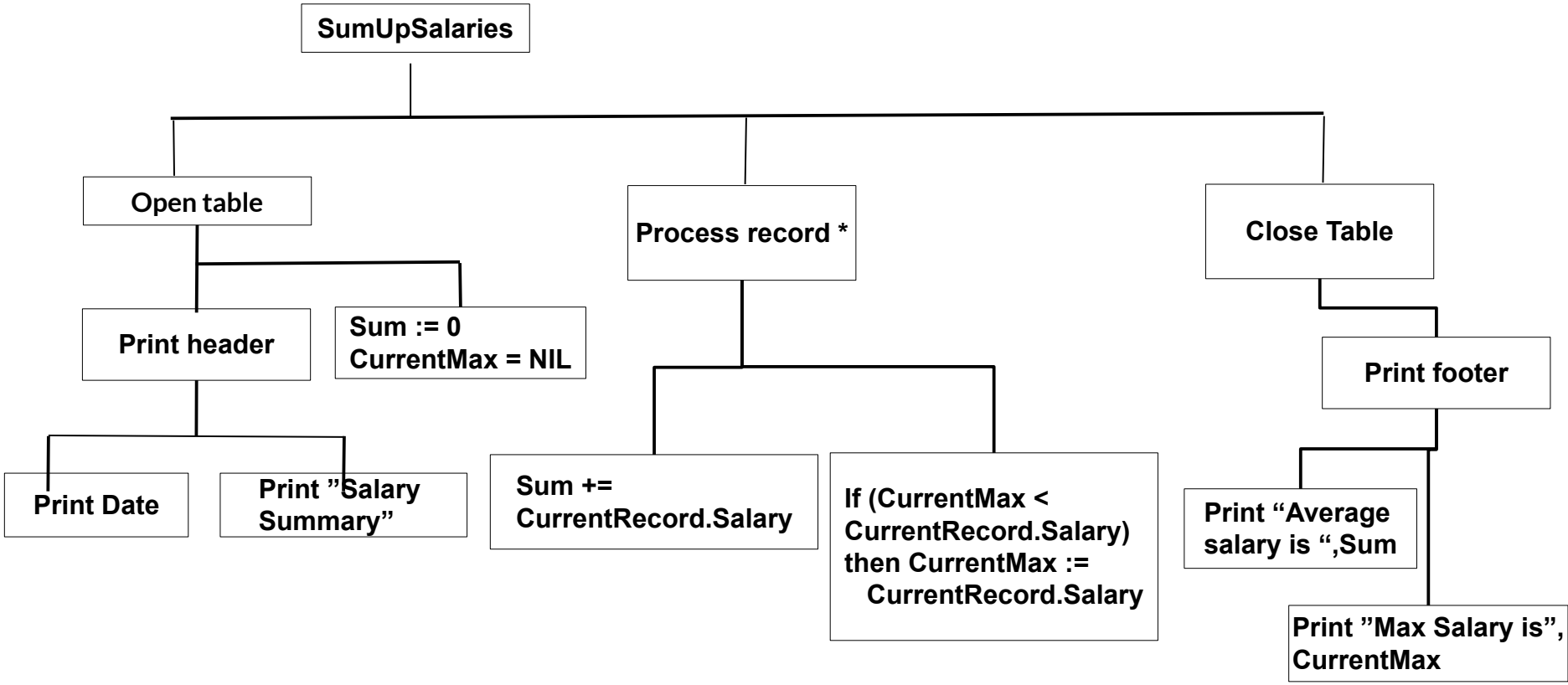
Applications of Grammar-Driven Programming

- EBNF is the standardized grammar language for all kind of actions based on context-free languages
  - Generation of code: parsers, generators, analyzers visitors
- **Parsing** character streams in compilers and software tools
  - Many *parser generators* exist
  - But parsing of lists of objects is also possible
- **Generators** of data
  - Test data generators for databases, compilers, software tools, metric tools, BI tools,...
- **Visitors** and **Analyzers** for complex data structures
  - Complex Big Data applications, which are non-regular
  - Complex Event Recognition in event streams in cyber-physical and embedded systems
    - „If several cars enter a parking house simultaneously through different gates, who gets the last free parking lot?“

26.3 Extensibility of JSD- and Grammar-Based Applications

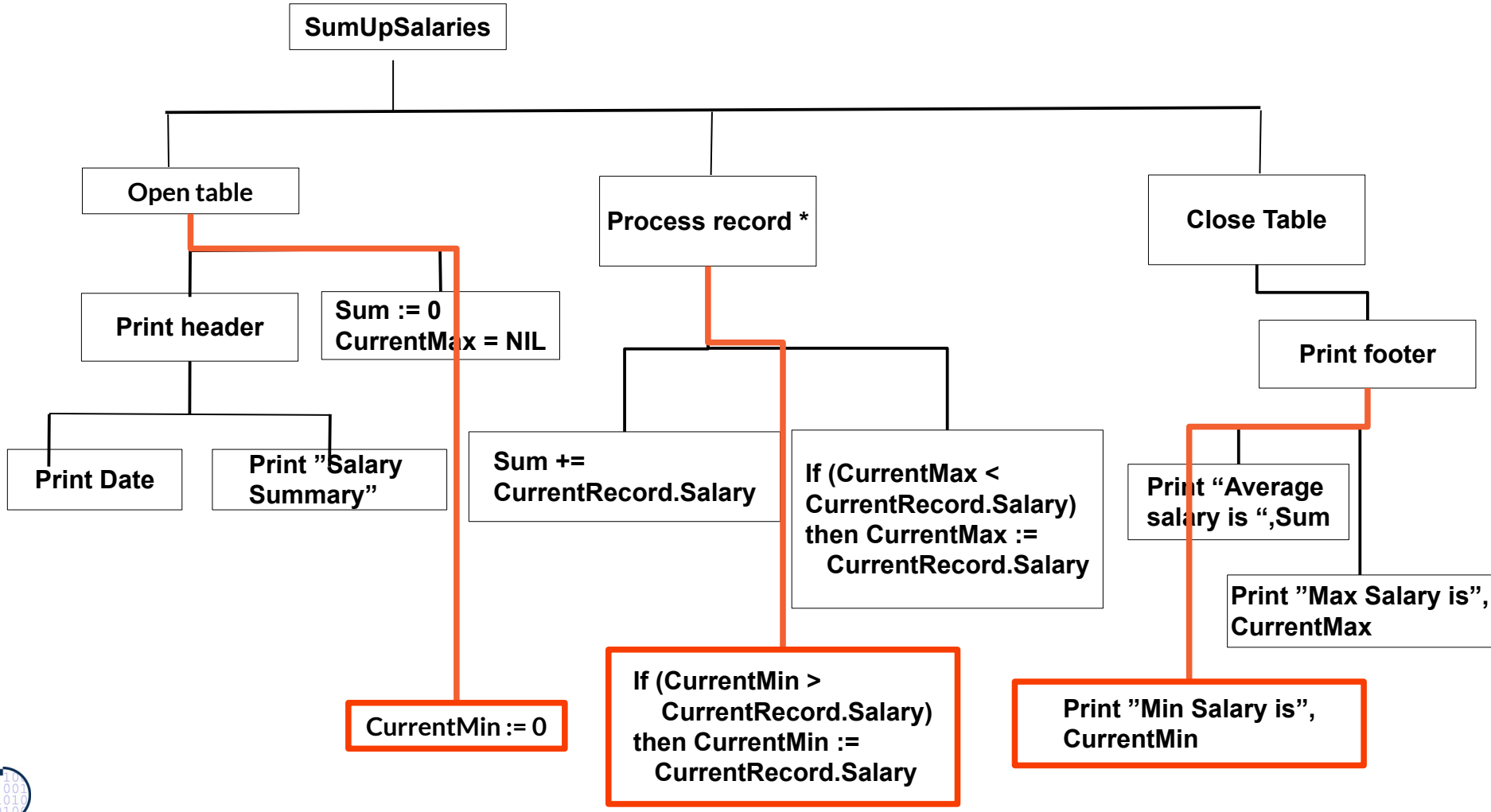
Tree Constructors in a JSD are Open Constructs

- A new slice (view) can be added easily to the core algorithm (aspect-based extension, see chapter „Aspect-oriented development“)



Tree Constructors in a JSD are Open Constructs

- A new slice (view) can be added easily to the core algorithm (aspect-based extension, see chapter „Aspect-oriented development“)



Further Data-Driven Design Methods

- Grammars:
  - **Attribute grammars** define more complex languages (→ course MOST)
    - Structure function spaces according to a hierarchic data structure
  - **Graph grammars** describe the structure of graphs
    - Room generation in MOOD games
    - Test data generation for graphs
- Map-Reduce based „Big Data“ Processing
  - Modern Map-Reduce frameworks such as Hadoop (Apache) offer distributed processing of data with many operators

The End

- Why is table and record processing important? Describe how the operators max, min, avg, sum are used on the records of a table.
- Give an example for a DFD in which the activities are specified by JSD.
- Why will COBOL never die? (unfortunately)
- Compare the structure of a JST with its generated implementation in an imperative language.
- Do the same for a generated grammar.