

# A DSL for Traversing Object Graphs

Felix Leipold

30/11/07

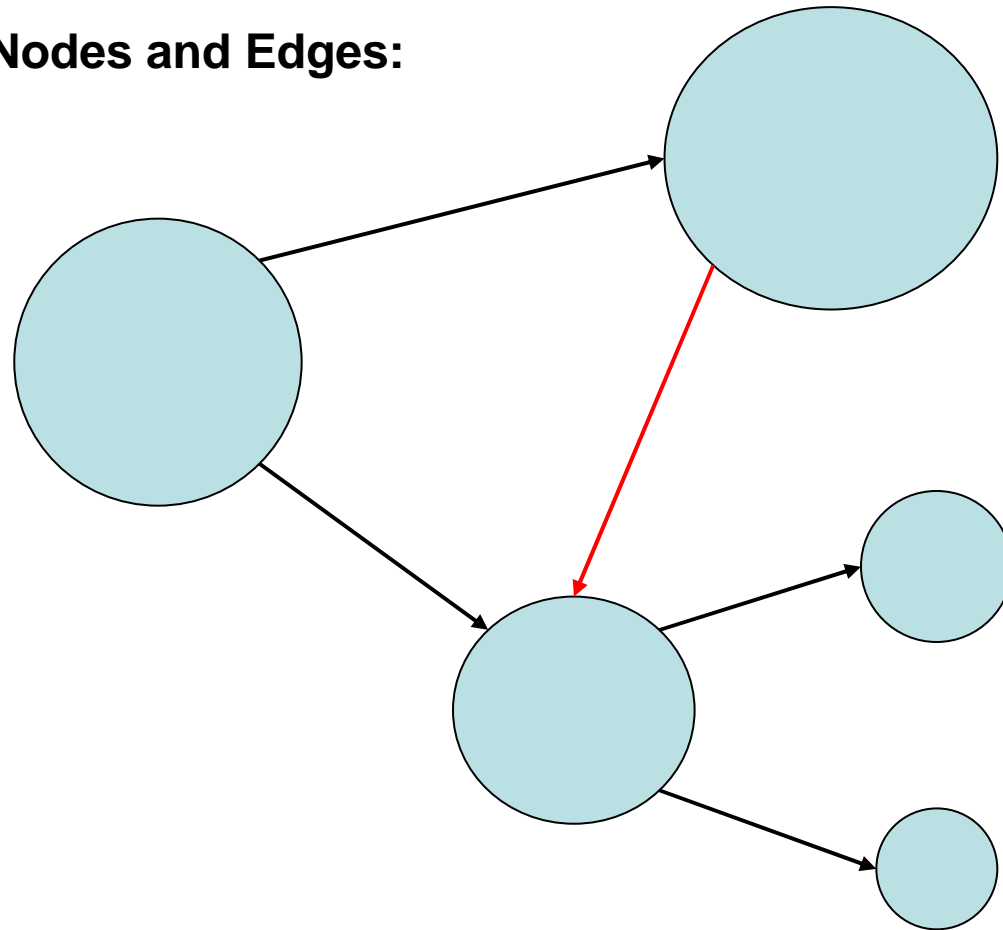
## What's that?

---

- Extracted from an existing system
- Some ideas I think might be worth considering...
- ... and beware: generalising

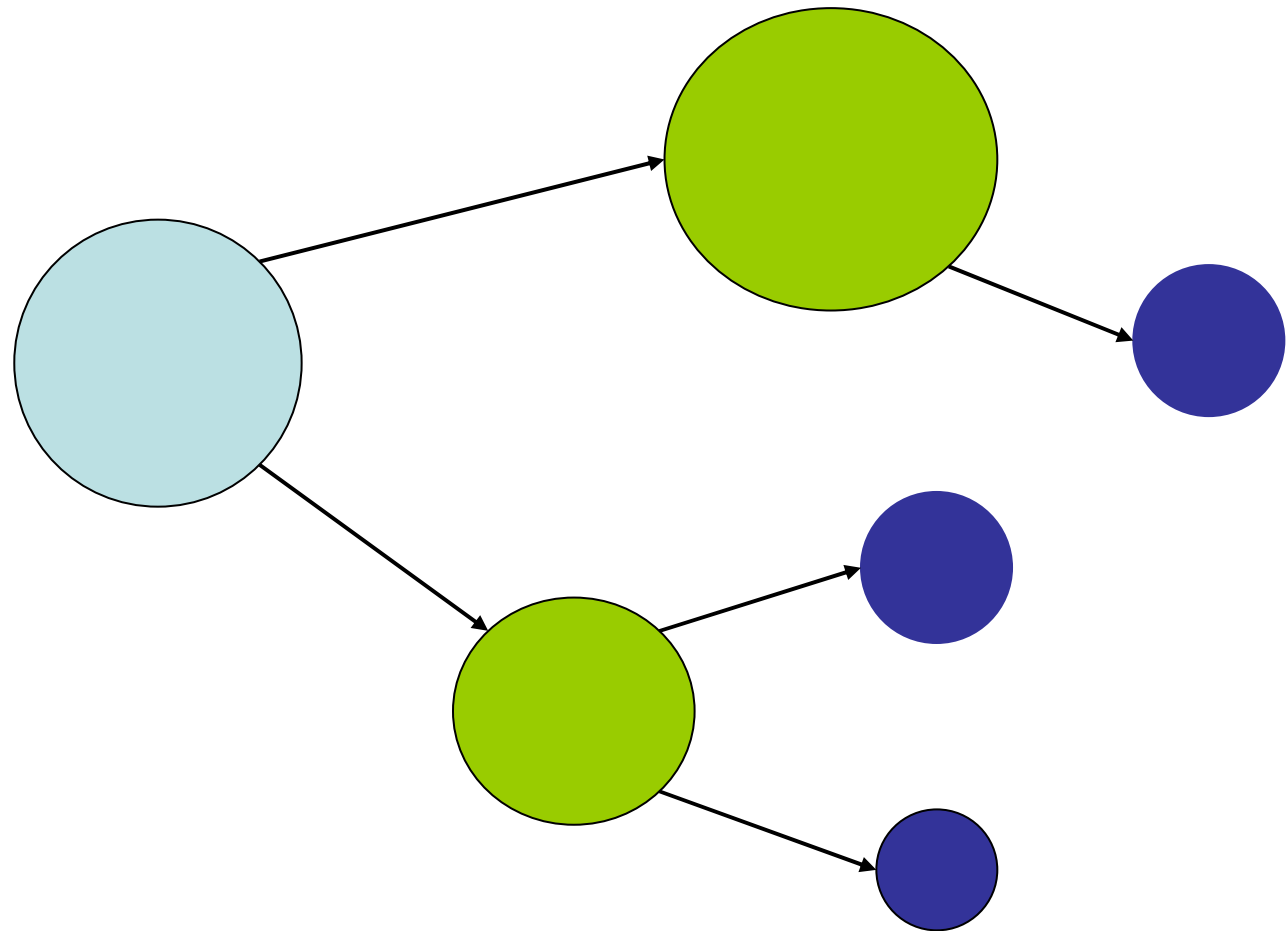
## What is an Object Graph?

It is a Graph, Nodes and Edges:



## And in Java we've got Classes..

---



## Example: How to Model a Book

---

### **Domain Driven Design**

#### **Putting the Domain Model to Work**

Crunching Knowledge

Ingredients of Effective Modeling

Knowledge Crunching

Communication and the Use of Language

Binding Model and Implementation

#### **The Building Blocks of a Model-Driven Design**

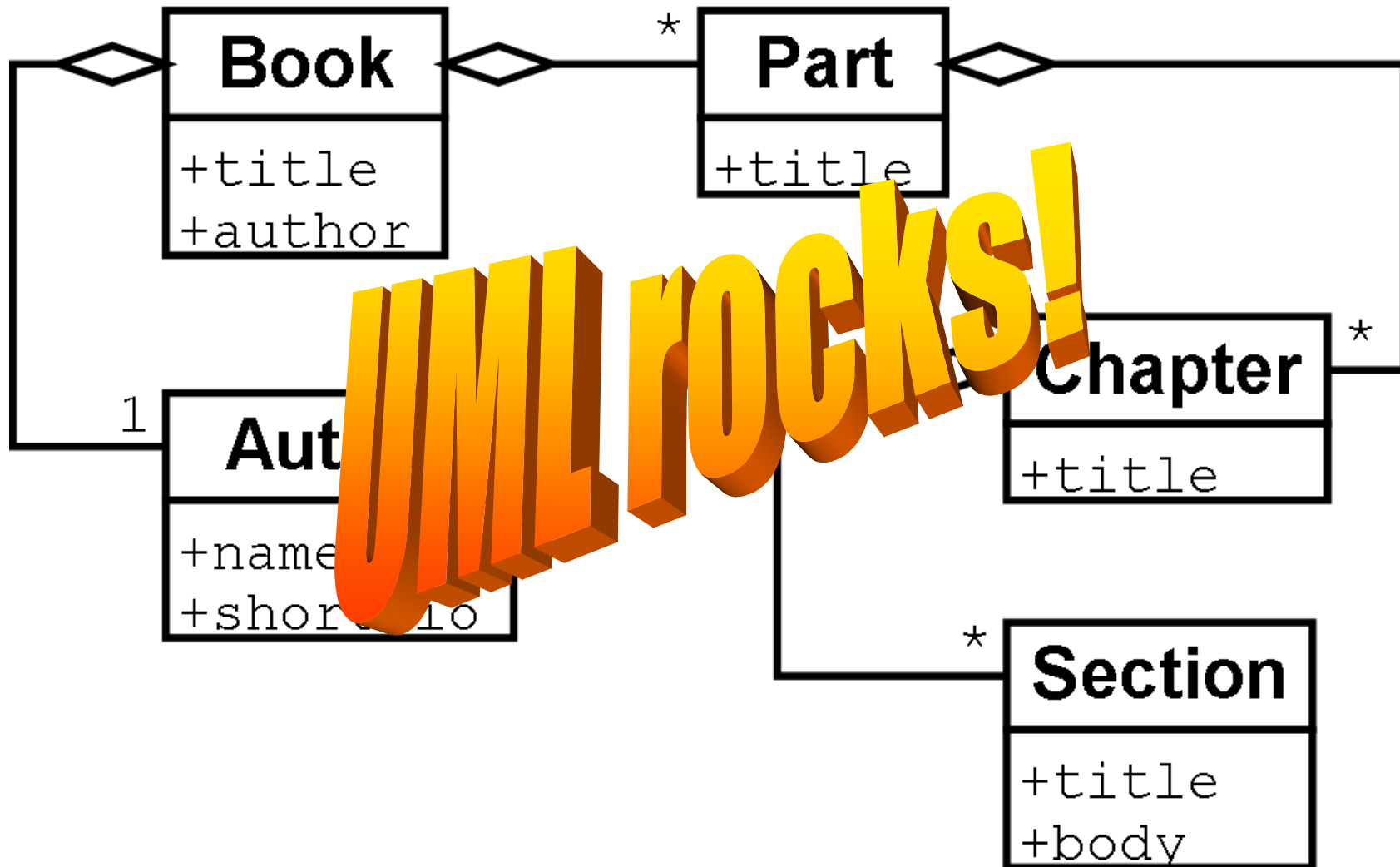
Isolating the Domain

#### **Refactoring Toward Deeper Insight**

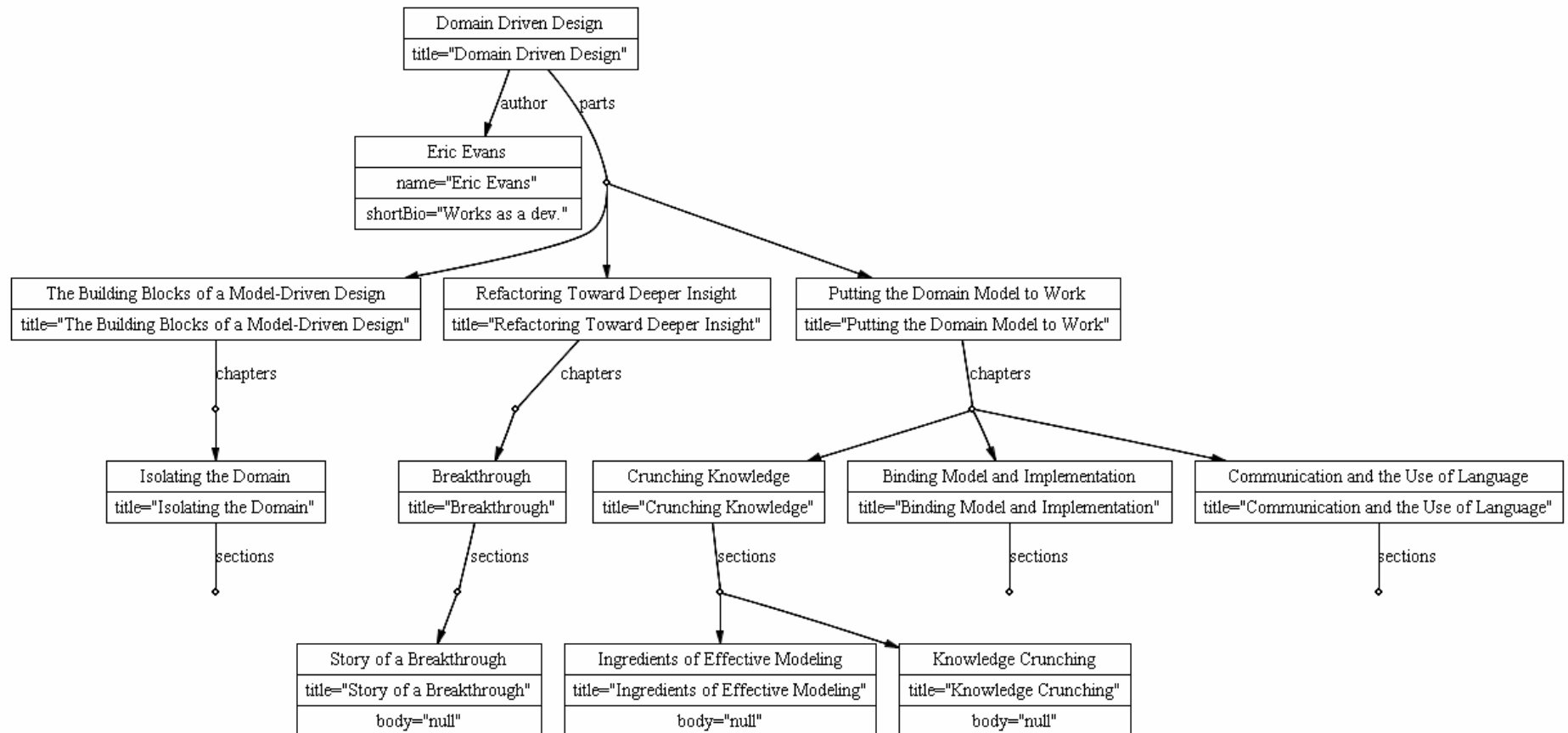
Breakthrough

Story of a Breakthrough

## Class Diagram



# An Example



## But why would I like to traverse such a Graph?

---

- Rendering
  - As Graph (as in graphics)
  - As Widget (tree)
  - As String
- Cloning (Deep copy or not so deep copy)
- Comparing
- Serialising
- Deserialising (also: gives you literals)

## How could I do that?

---

- a) Explicit (imperative) code
- b) Reflection

### **But**

- a) Not very nice to read
- b) Inflexible (Where does that knowledge belong?)

## Let's have a DSL!

```
Book(  
  title,  
  author: Author(  
    name,  
    shortBio  
  ),  
  parts : collection(Part(  
    title,  
    chapters : collection(Chapter(  
      title,  
      sections : collection (Section(  
        title,  
        body  
      ))  
    ))  
  ))  
))  
)
```

## Yielding a serialized Graph like this:

---

```
("Domain Driven Design", ("Eric Evans", "Works as a dev."), [ ("Putting the Domain Model to Work", [ ("Crunching Knowledge", [ ("Ingredients of Effective Modeling", null), ("Knowledge Crunching", null)]) ], ("Communication and the Use of Language", []), ("Binding Model and Implementation", [])]), ("The Building Blocks of a Model-Driven Design", [ ("Isolating the Domain", [])]), ("Refactoring Toward Deeper Insight", [ ("Breakthrough", [ ("Story of a Breakthrough", null)]) ]) ])
```

## Or Alternatively?

---

```
Book(  
  title,  
  author: Author(  
    name,  
    shortBio  
  )  
)
```

**Yielding:**

```
("Domain Driven Design", ("Eric Evans", "Works as a dev."))
```

## What has been implemented?

---

- **Parser and Metamodel**
  - **Serialisation**
  - **Cloning**
  - Comparing
  - Dot based Visualizer
  - Swing tree model
- 
- Used in “production”
- Toys

## How does it work?

---

```
traversalModel = TraversalUtils.parse("full.trv");  
Book book = Book.buildExample();
```

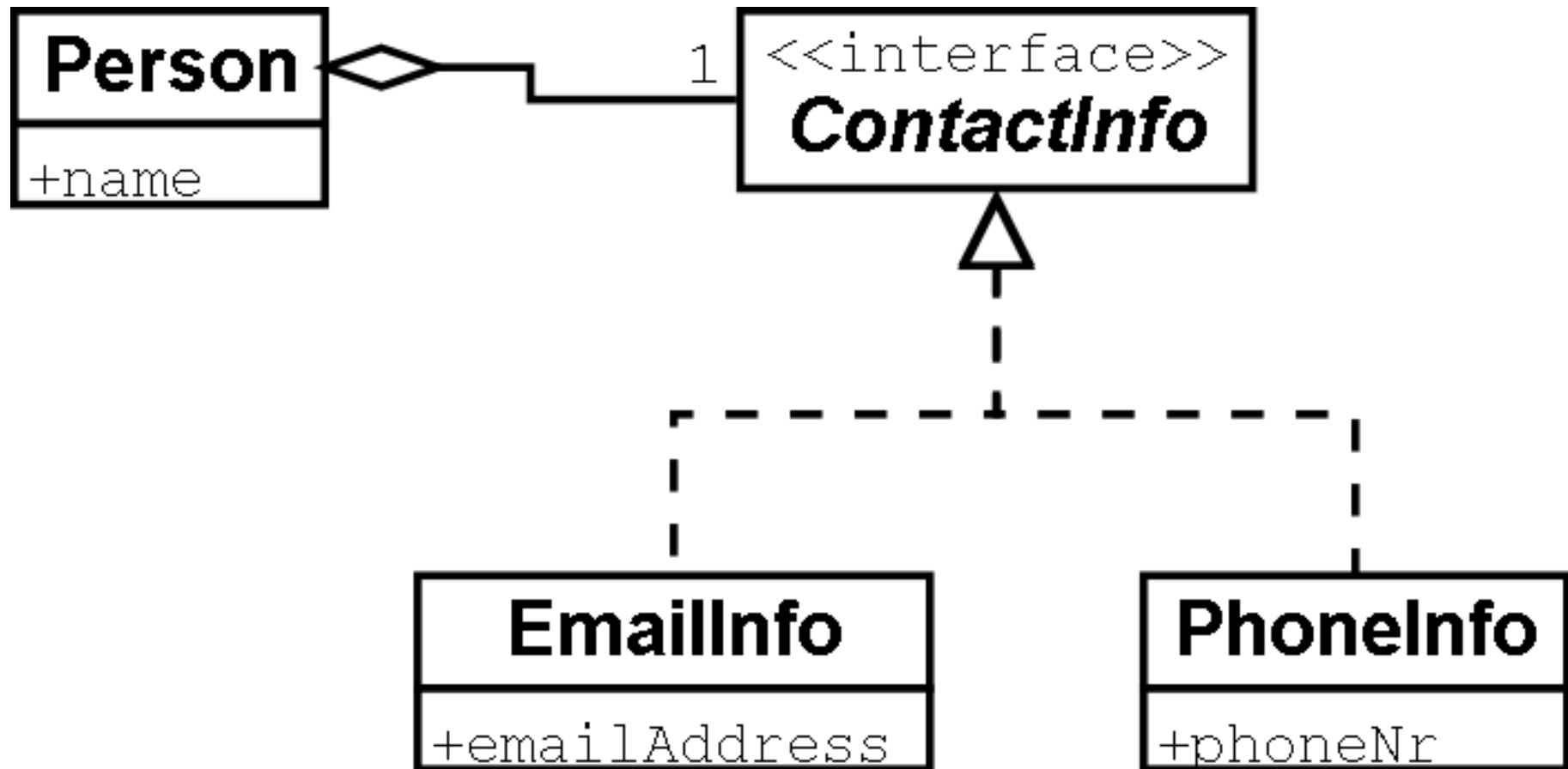
```
System.out.println(  
    TraversalUtils.serialize(traversalModel, book)  
);
```

```
GraphViz.graph(traversalModel, book, true, "Graph  
Test");
```

```
TraversalUtils.compare(traversalModel, book,  
    anotherBook)
```

```
TraversalUtils.clone(traversalModel, book)
```

## What about Polymorphism?



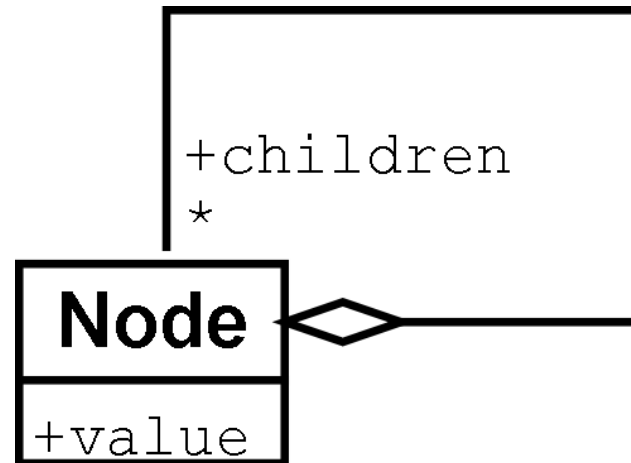
## How about that?

---

```
Person (  
    name ,  
    contactInfo :  
    switch (ContactInfo) (  
        EmailInfo (emailAddress) ,  
        PhoneInfo (phoneNr )  
    )  
)
```

## And Recursive Data Structures?

---



```
Node as node (  
    value,  
    children: reference (node)  
)
```

## Possible Extensions

---

- Have Predicates on nodes?

```
Node as node (  
    value,  
    children: reference (node where value>2)  
)
```

- Collapse parts of the graph?

```
Book(  
    parts : collection( @collapse Part(  
        title,  
        chapters : collection(Chapter(  
            title  
        ))  
    ))  
)
```

# Questions

**Thanks for Enduring**