Design Patterns

Dirk Riehle, FAU Erlangen

ADAP D01

Agenda

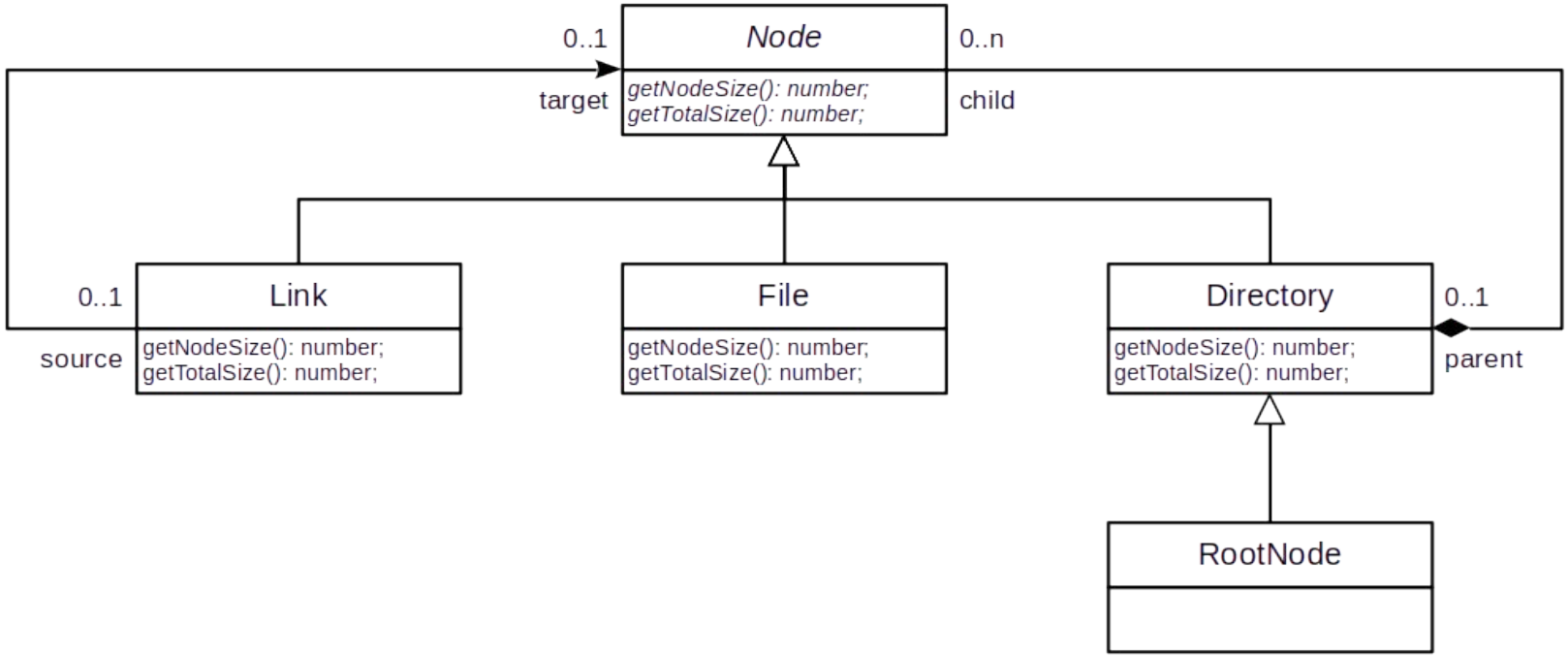
1. Three design examples
2. The Composite pattern
3. Software design patterns
4. Other types of patterns
5. Describing design patterns
6. Versus language features

1. Three Design Examples

Three Design Examples

1. File system
2. Financial portfolios
3. Test suites

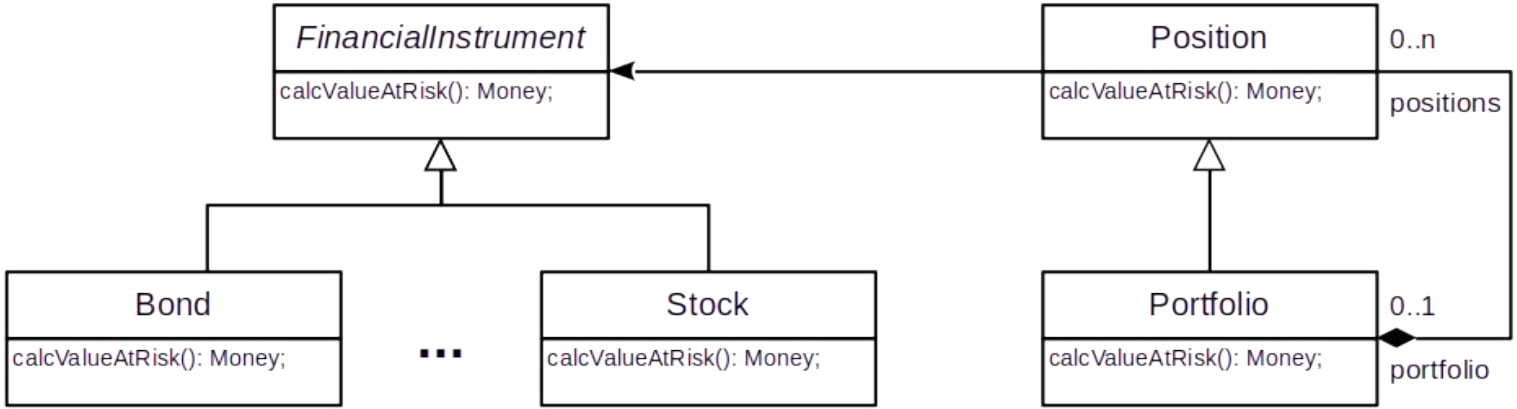
The File System Design Example



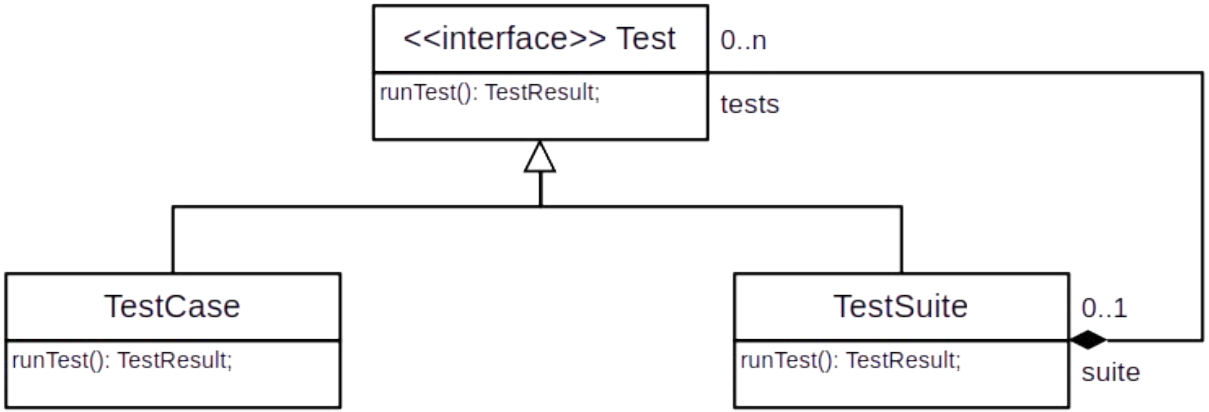
Node.getTotalSize() Exercise

#	Size	Type	Path	Who?	Result
1	1	RootNode	/		
2	1	Directory	/usr		
3	1	Directory	/usr/bin		
4	12	File	/usr/bin/ls		
5	579	File	/usr/bin/code		
6	1	Directory	/media		
7	1	Directory	/home		
8	1	Directory	/home/riehle		
9	10	File	/home/riehle/.bashrc		
10	344	File	/home/riehle/wallpaper.jpg		
11	1	Directory	/home/riehle/projects		

Example Design of a Financial Trading Application



The Core JUnit Design (Java)



What is Common to the Three Examples?

- Classes?
- Names?
- Structure?
- Dynamics?

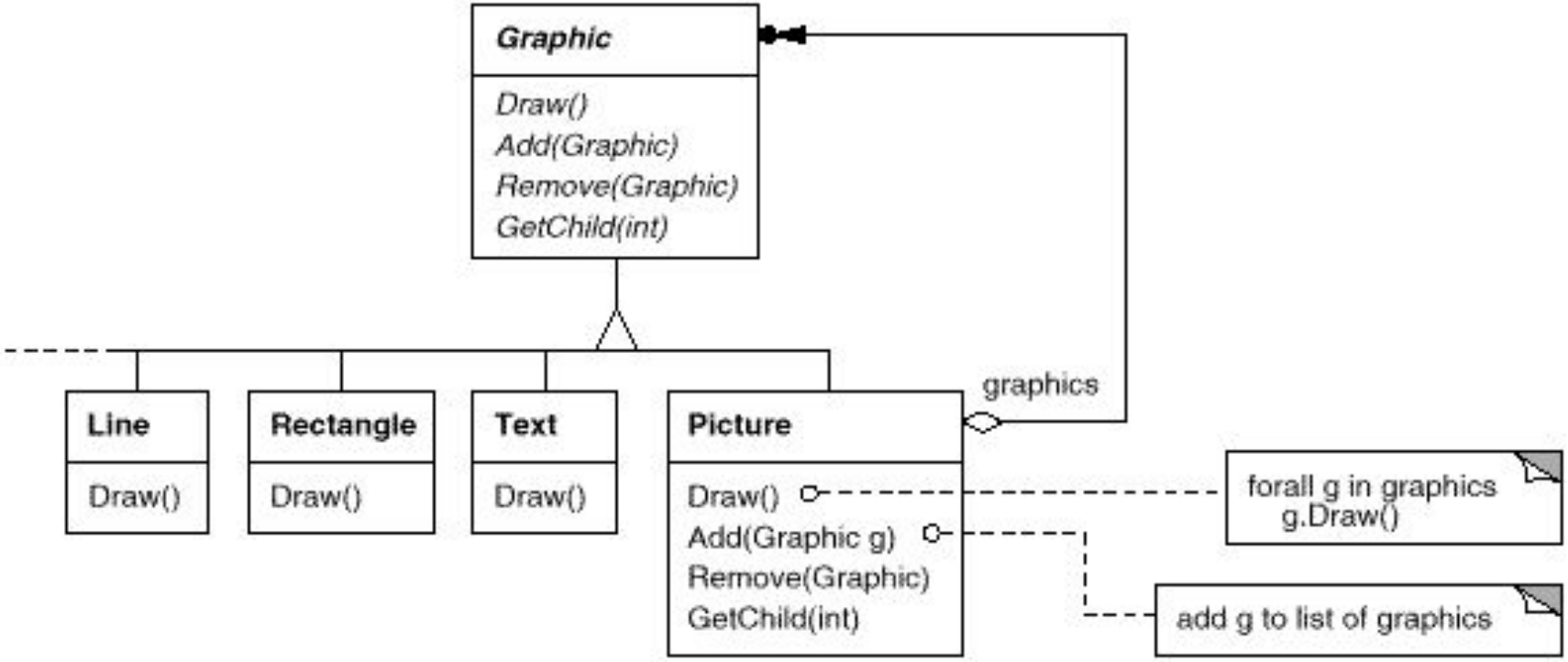
2. The Composite Pattern

Intent of the Composite Design Pattern [1]

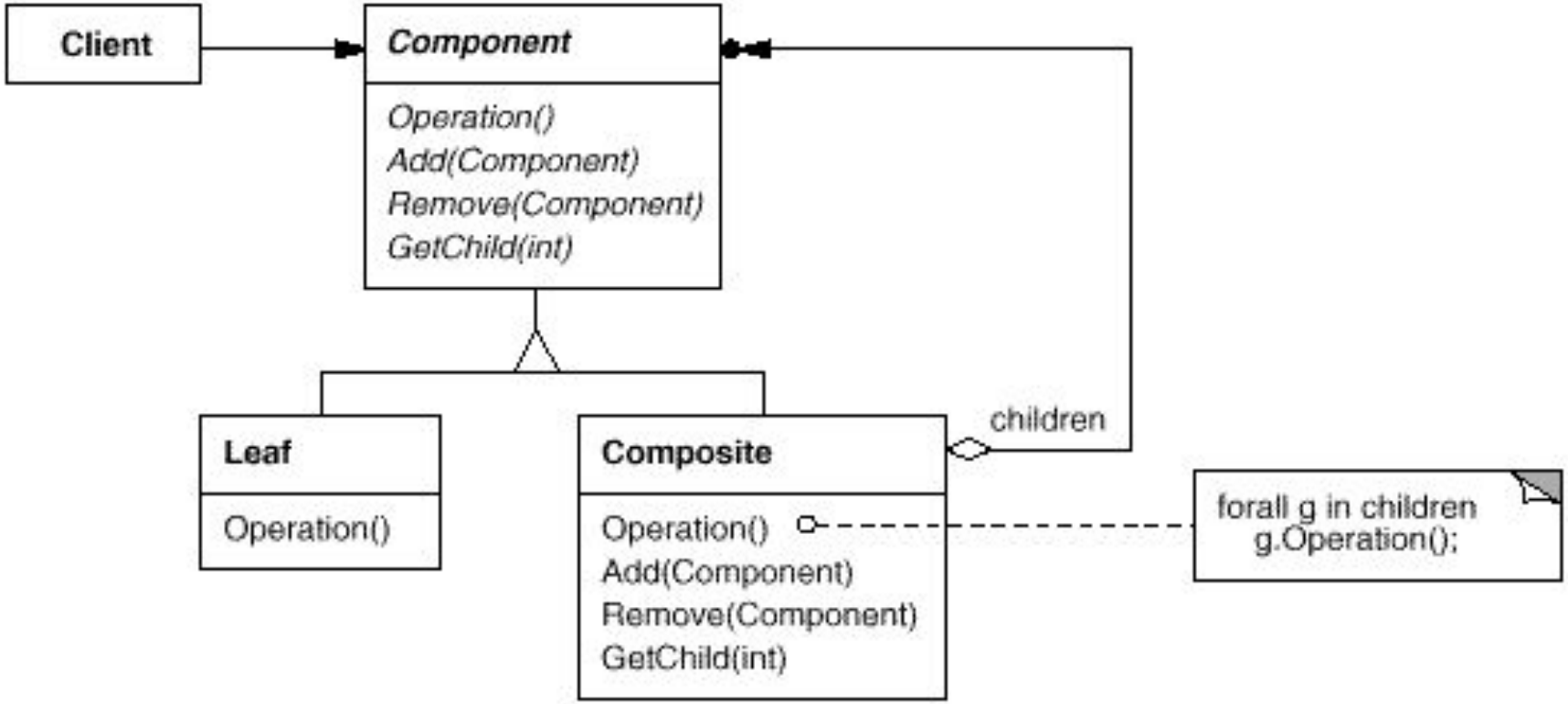
Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

[1] Gamma, E. et al. (1995). Design patterns. Addison Wesley.

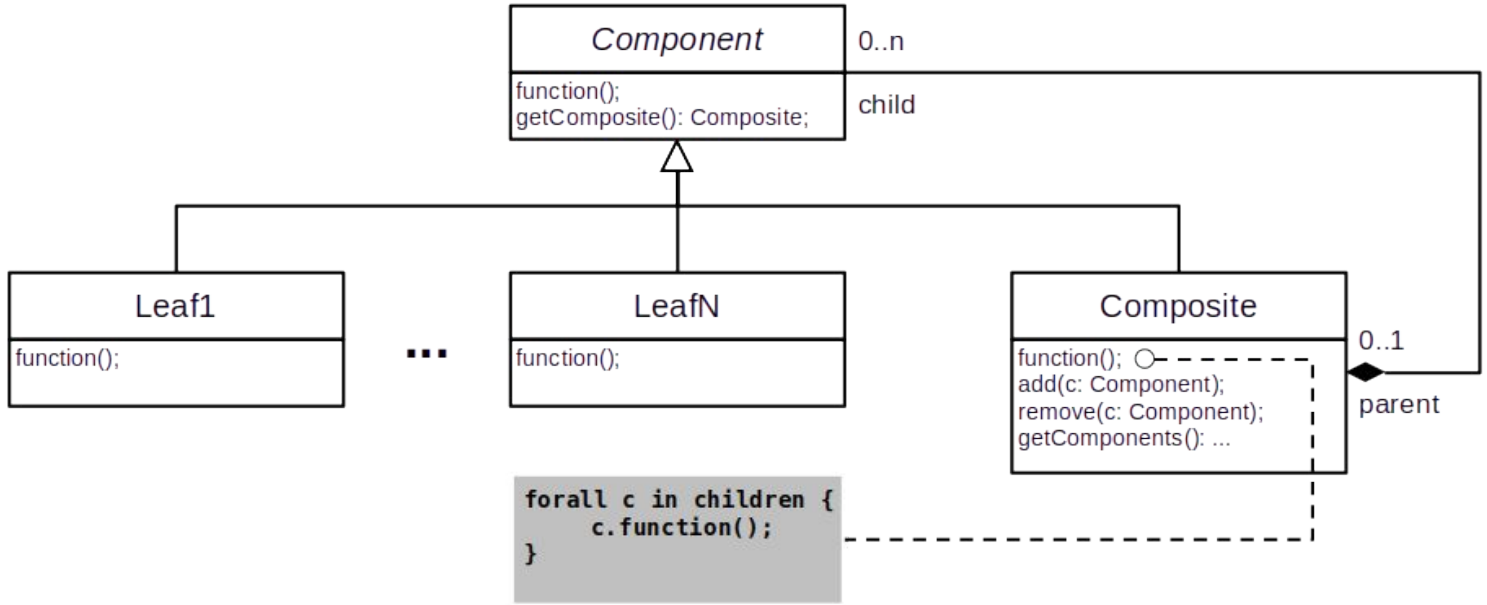
Motivation of the Composite Design Pattern



Structure Diagram of the Composite Design Pattern



Modernized Structure Diagram of the Composite Pattern



How Would You Model a Computer Configuration?

Hint: Use the Composite design pattern

3. Software Design Patterns

Design Pattern

A design pattern is

- The abstraction from
  - **A recurring solution**
  - **To a recurring problem**
  - **In a defined context**

Benefits of Using Design Patterns

Better, faster, cheaper …

- Designing of software
- Documenting software
- Communicating designs

The Design Patterns Book 1 / 2

The design patterns book [1]

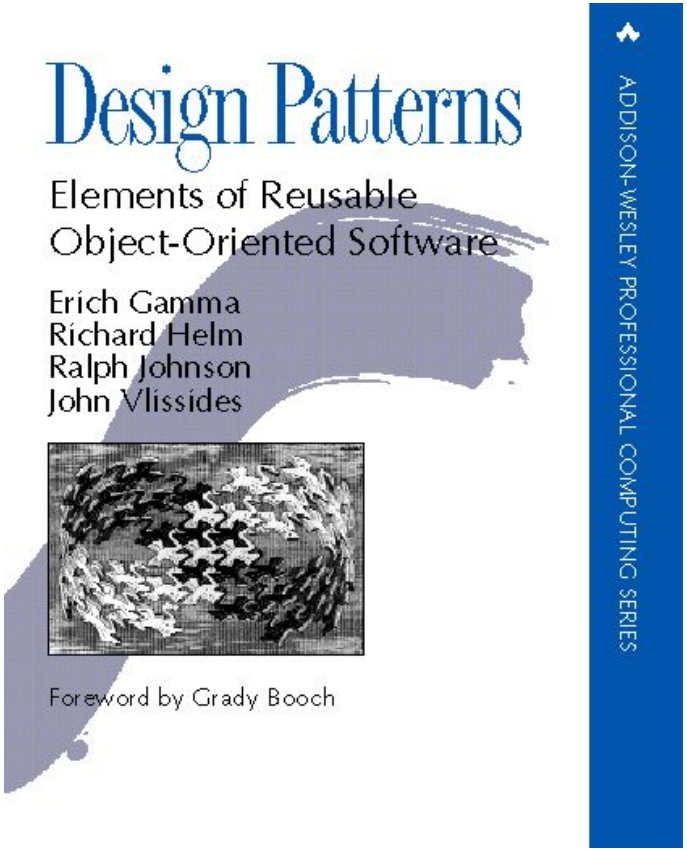
- A.k.a. the patterns catalog
- A.k.a. the Gang-of-four book

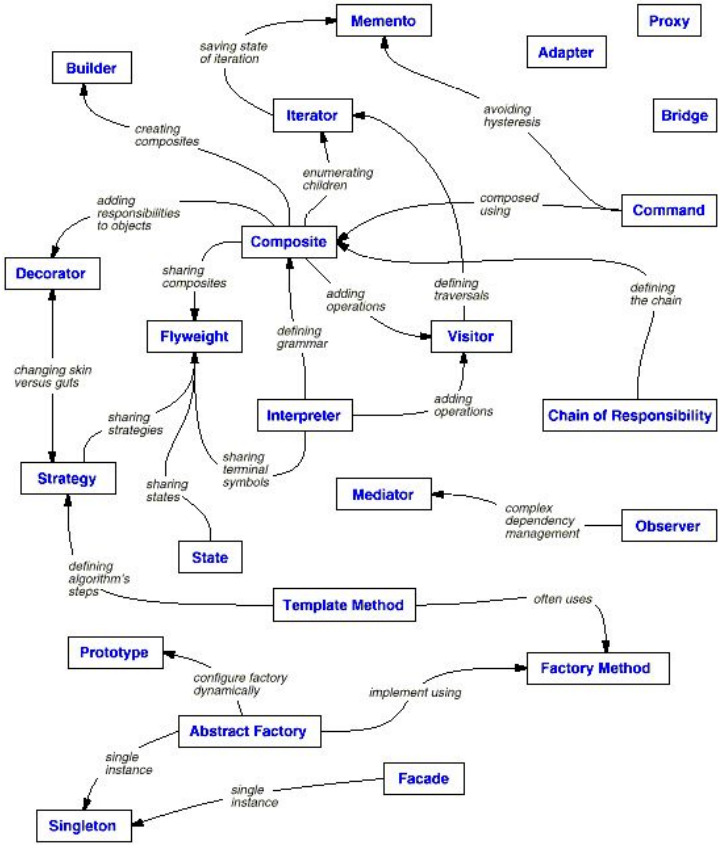
is a seminal book that started the design patterns movement.

(Original German translation by yours truly [2])

[1] Gamma, E. et al. (1995). Design Patterns. Addison Wesley.
[2] Gamma, E. et al. (1996). Entwurfsmuster. Addison Wesley, Germany.

The Design Patterns Book 2 / 2





Beyond The Design Patterns Book

- Null Object
- Value Object
- Type Object
- Role Object
- …

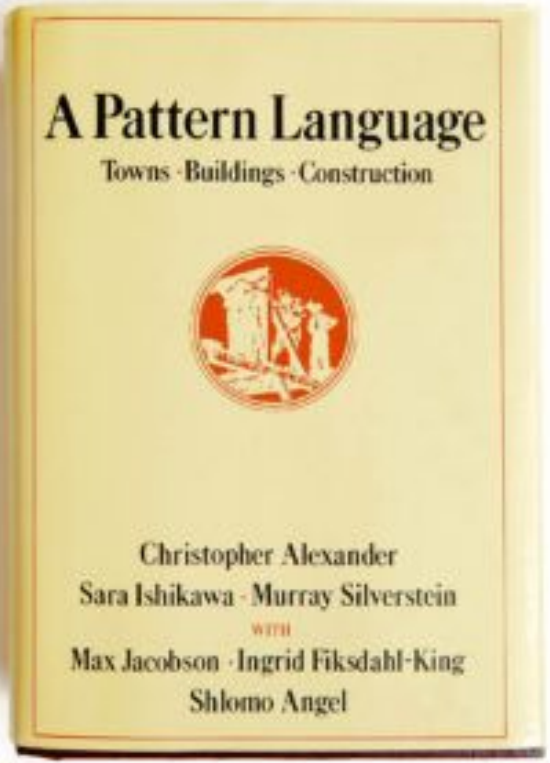
Sections of an Original Design Pattern Description [1]

1. Intent	2. Also known as	3. Motivation	4. Applicability
5. Structure	6. Participants	7. Interactions	8. Consequences
9. Implementation	10. Example code	11. Known uses	12. Related patterns

[1] According to Gamma et al. (1995)

A Simpler Pattern Description [1]

1. Context (where are we?)
2. Problem (what's the problem?)
3. Solution (how to solve it)



[1] Alexander, C. et al. (1977). A Pattern Language. Oxford University Press.

The Composite Pattern as Context / Problem / Solution

Context

- We need an easy way to work with a tree on every level

Problem

- Each tree node must be able to represent the whole subtree

Solution

- Move child object handling from node class into new subclass

A Highly Abridged History of Patterns in Software

1. A pattern language (1977)
2. “No object is an island” (1992)
3. ET++ and Interviews (1988-1995)
4. The design pattern catalog (1994)
5. A system of patterns (1996)
6. The PLoPD book series (1998-2006)
7. The TPLoP journal (2000-today)

4. Other Types of Patterns

Levels of Granularity

1. Architectural patterns
2. Design patterns
3. Programming idioms

Architectural Pattern Example

Event bus (a.k.a. publish/subscribe architecture)

- Context
  - A software system of communicating components that come and go unbeknownst to each other
- Problem
  - How to ensure components can send and receive all information they need?
- Solution
  - Mediate communication between components through a bus that
    - Allows for the definition of event types
    - Can receive events for defined event types
    - Can distribute events to those registered for the event type

Architectural Style Example

Pipes and filters architecture (a.k.a. dataflow architecture)

- Context
  - A system to transform data received from data sources for consumption by data sinks
- Problem
  - How to flexibly create those systems from reusable components
- Solution
  - Allow only for two types of components: Pipes and filters
    - Pipes receive data from filters or sources and distribute them to other filters or sinks
    - Filters receive data from pipes, process it, and feed it into further pipes
    - Build a pipes and filters system as a DAG from existing pipes and filters

The 23 Design Patterns from the Patterns Catalog

Creational Patterns

- Abstract Factory (87)
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- Builder (97)
  - Separate the construction of a complex object from its representation so that the same construction process can create different representations.

Structural Patterns

- Adapter (139)
  - Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
- Bridge (151)
  - Decouple an abstraction from its implementation so that the two can vary independently.

Programming Idiom Example [1]

```
class NumberArrayWithFixedIterator {
   protected data: number[] = [];
   protected count: number = 0;

   constructor(data: number[]) {
       this.data = [...data];
   }

   public getNextAndInc(): number {
       const result = this.data[this.count++];
       return result;
   }
}
```

[1] PS: Anything wrong with this example?

5. Describing Design Patterns

What Notation is This? [1]

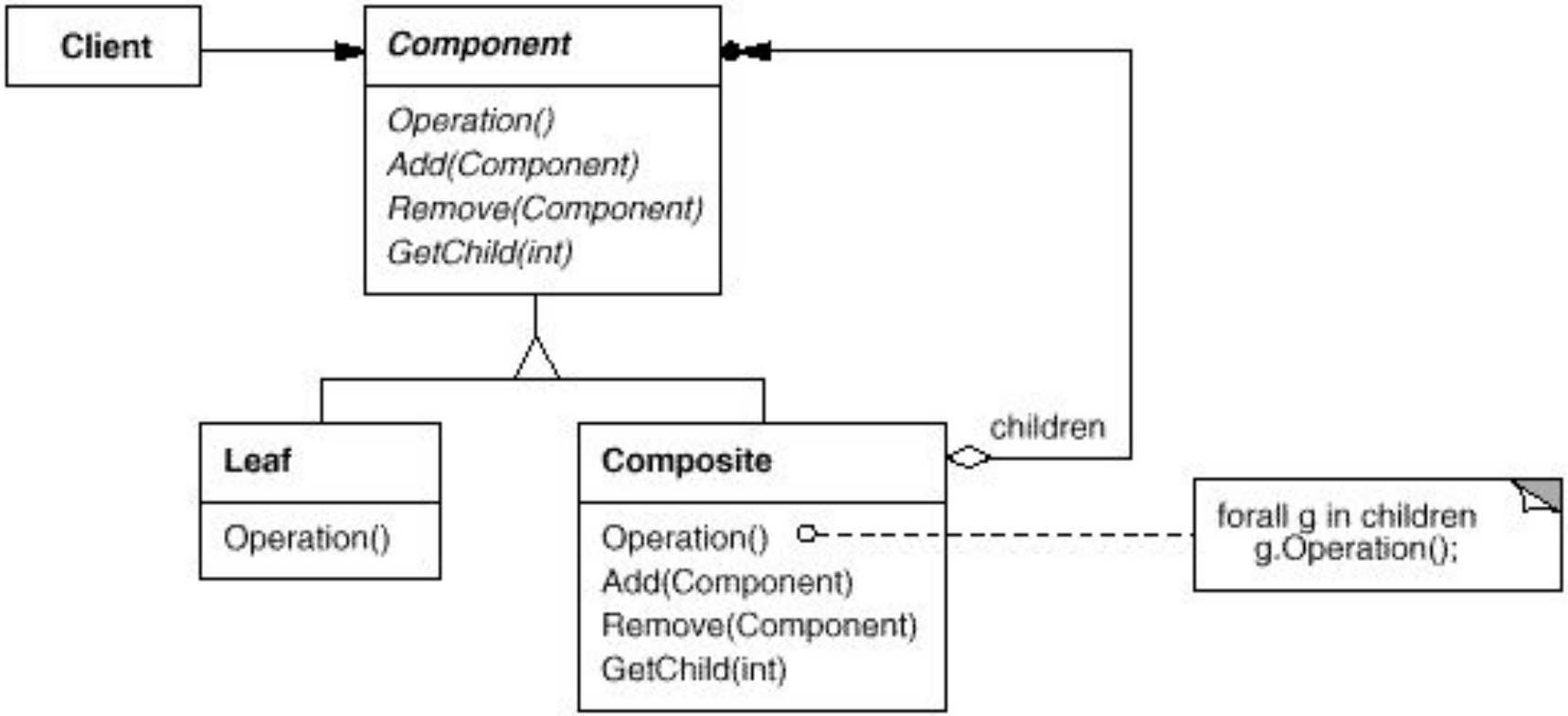


Illustration vs. Template

An **illustration** is

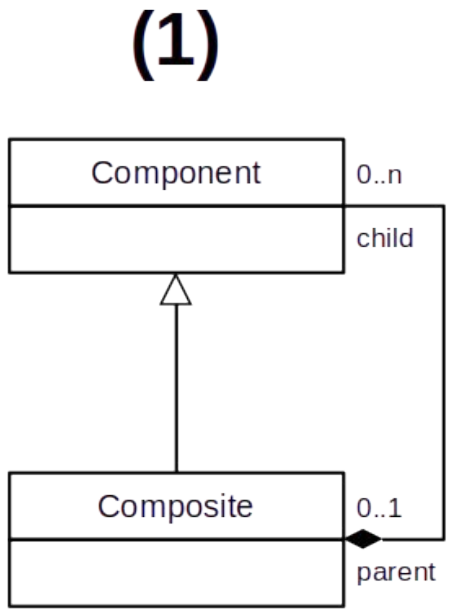
- A made-for-humans informal description (of a design pattern)

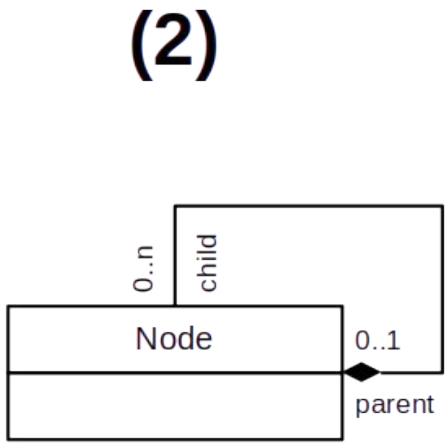
A **template** is a model that is

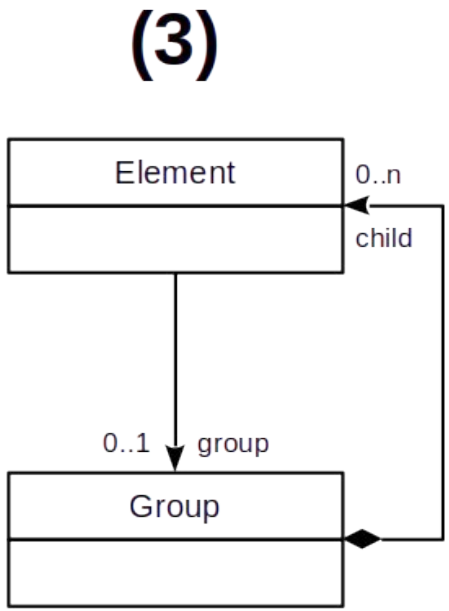
- A sufficiently formal description that can be applied to deliver an instance

The structure diagram is an illustration of the most common form

Variations of the Composite Pattern's Structure







Pattern Collections

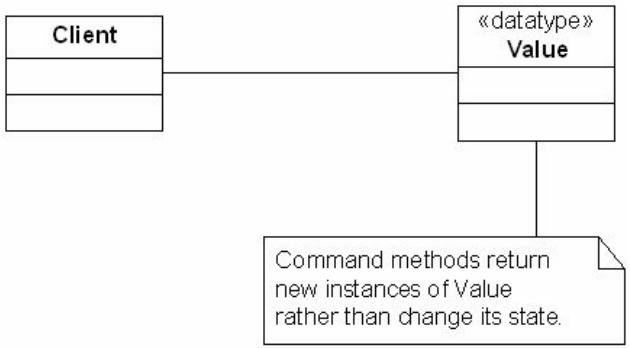
1. Pattern catalogs
2. Pattern handbooks
3. Pattern languages

6. Versus Language Features

The Value Object Pattern

Design pattern book style [1]

- Intent
  - Implement data types as immutable classes so that their instances can be handled like built-in values.
- Structure



Alexanderian form

- Context
  - Any object-oriented system
- Problem
  - You can't natively express values
- Solution
  - Implement value types as immutable classes

[1] Riehle, D. (2006). Value Object. In Proceedings of the 2006 Conference on Pattern Languages of Programming (PLoP '06). ACM: Article no. 30, pp. 1-6.

Some Design Patterns Could be Language Features

Visitor → Double-dispatch (or multiple dispatch) function calling

Value Object → Domain-specific values as first-class citizens

Type Object → Types as first class citizens (a.k.a. Meta Object)

Role Object → Traits and collaborations as first class citizens

[1] Gamma, E. et al. (1995). Design Patterns. Addison Wesley.
[2] Riehle, D. (2006). Value Object. In Proceedings of the 2006 Conference on Pattern Languages of Programming (PLoP '06). ACM: Article no. 30, pp. 1-6.
[3] Johnson, R., & Woolf, B. (1997). Type object. In Pattern Languages of Program Design 3, pp. 47-65. Addison Wesley.
[4] Bäumer, D., Riehle, D., Siberski, W. & Wulf, M. (2000). Role Object. In Pattern Languages of Program Design 4 (PLoPD 4), pp. 15-32. Addison-Wesley.

Summary

1. Three design examples
2. The Composite pattern
3. Software design patterns
4. Other types of patterns
5. Describing design patterns
6. Versus language features

Thank you! Any questions?

dirk.riehle@fau.de – https://oss.cs.fau.de

dirk@riehle.org – https://dirkriehle.com – @dirkriehle

Legal Notices

License

- Licensed under the CC BY 4.0 International license

Copyright

- © 2012, 2018, 2024 Dirk Riehle, some rights reserved