Principles of Software Construction: Objects, Design and Concurrency

Introduction to Design Patterns

15-214

toad

Christian Kästner Charlie Garrod

School of Computer Science

institute for SOFTWARE RESEARCH

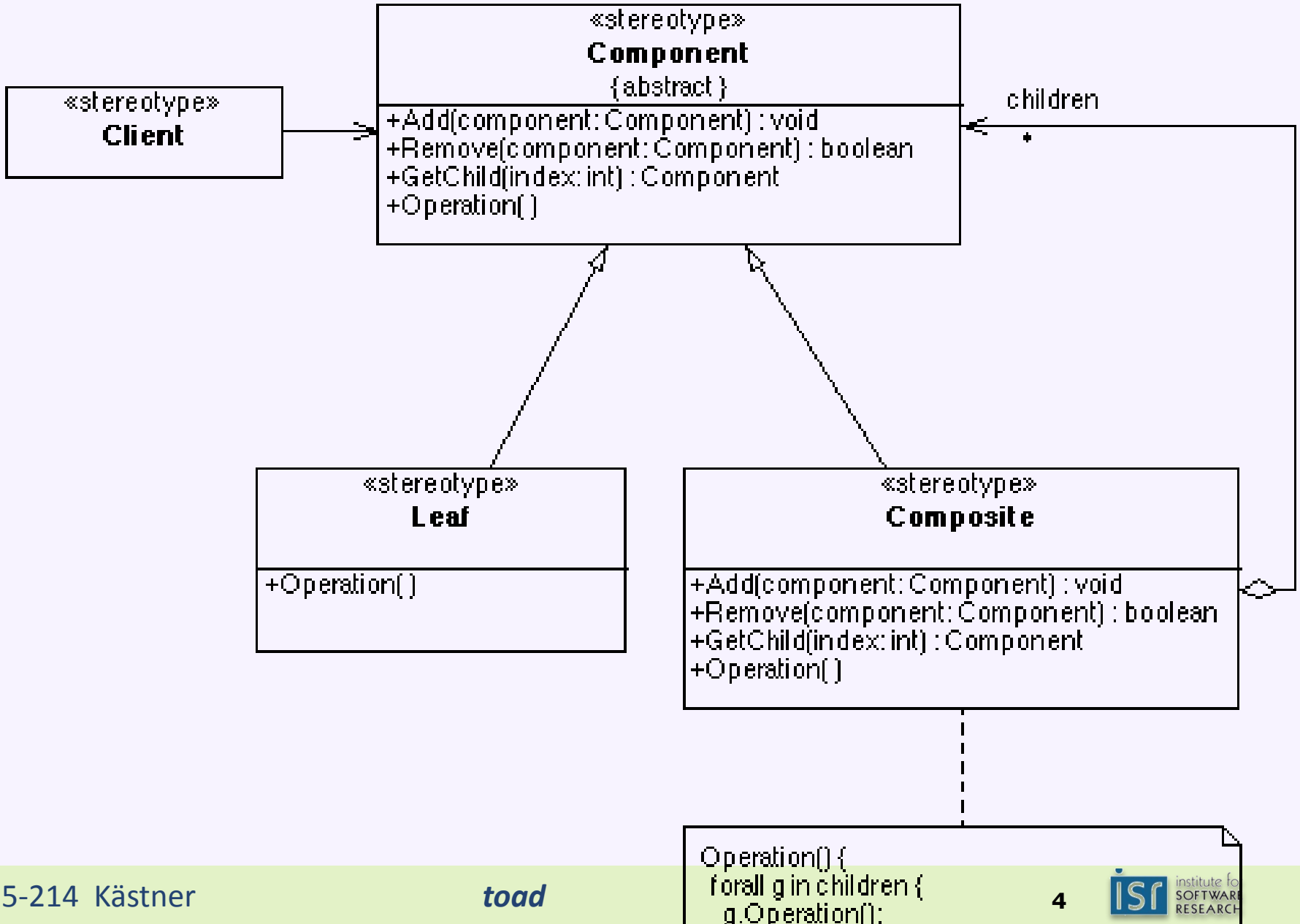
Learning Goals

- Understand the nature of design patterns
  - Parts of a design pattern
  - Applicability and benefits of design patterns
  - Limitations and pitfalls of design patterns
- Apply the composite design pattern

Design Exercise (on paper!)

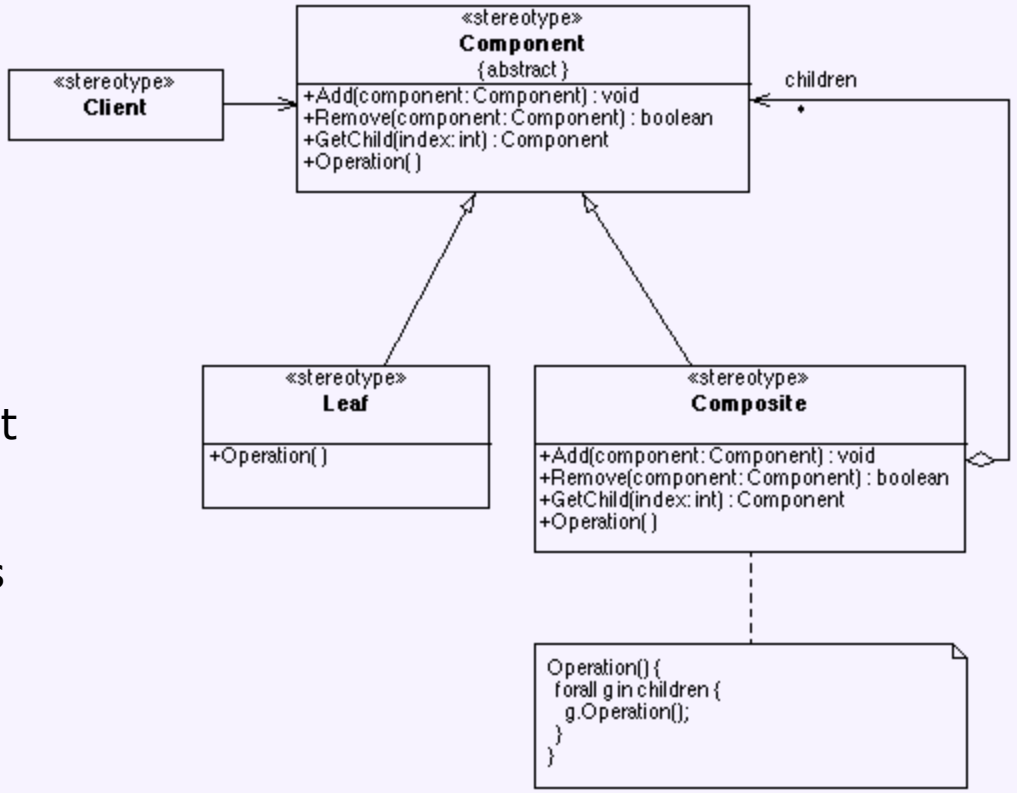
- You are designing software for a shipping company.
- There are several different kinds of items that can be shipped: letters, packages, fragile items, etc.
- Two important considerations are the **weight** of an item and its **insurance cost**.
  - Fragile items cost more to insure.
  - All letters are assumed to weigh an ounce
  - We must keep track of the weight of other packages.
- The company sells **boxes** and customers can put several items into them.
  - The software needs to track the contents of a box (e.g. to add up its weight, or compute the total insurance value).
  - However, most of the software should treat a box holding several items just like a single item.
- Show a **class diagram** for representing packages, complete with **inheritance relations** and **method signatures**.

A First Pattern: Composite (Structural)



A First Pattern: Composite (Structural)

- Applicability
  - You want to represent part-whole hierarchies of objects
  - You want to be able to ignore the difference between compositions of objects and individual objects
- Consequences
  - Makes the client simple, since it can treat objects and composites uniformly
  - Makes it easy to add new kinds of components
  - Can make the design overly general
    - Operations may not make sense on every class
    - Composites may contain only certain components



We have seen this before! (kindof)

```
interface Point {
   int getX();
   int getY();
}
class MiddlePoint implements Point {
   Point a, b;
   MiddlePoint(Point a, Point b) {this.a = a; this.b = b; }
   int getX() { return (this.a.getX() + this.b.getX()) / 2;}
   int getY() { return (this.a.getY() + this.b.getY()) / 2; }
}
```

- **Our designs for composite figures, grouped packages, and union sets solve similar problems in similar ways**
- **We call this problem-solution pair a design pattern**

Design Patterns

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"
  – Christopher Alexander

- Every Composite has its own domain-specific interface
  - But they share a common problem and solution

History

- Christopher Alexander, The Timeless Way of Building (and other books)
  - Proposes patterns as a way of capturing design knowledge in architecture
  - Each pattern represents a tried-and-true solution to a design problem
  - Typically an engineering compromise that resolves conflicting forces in an advantageous way
    - Composite: you have a part-whole relationship, but want to treat individual objects and object compositions uniformly

Patterns in Physical Architecture

- When a room has a window with a view, the window becomes a focal point: people are attracted to the window and want to look through it. The furniture in the room creates a second focal point: everyone is attracted toward whatever point the furniture aims them at (usually the center of the room or a TV). This makes people feel uncomfortable. They want to look out the window, and toward the other focus at the same time. If you rearrange the furniture, so that its focal point becomes the window, then everyone will suddenly notice that the room is much more "comfortable".
  – Leonard Budney, Amazon.com review of *The Timeless Way of Building*

Benefits of Patterns

- Shared language of design
  - Increases communication bandwidth
  - Decreases misunderstandings
- Learn from experience
  - Becoming a good designer is hard
    - Understanding good designs is a first step
  - Tested solutions to common problems
    - Where is the solution applicable?
    - What are the tradeoffs?

Illustration [Shalloway and Trott]

- Carpenter 1: How do you think we should build these drawers?

- Carpenter 2: Well, I think we should make the joint by cutting straight down into the wood, and then cut back up 45 degrees, and then going straight back down, and then back up the other way 45 degrees, and then going straight down, and repeating…

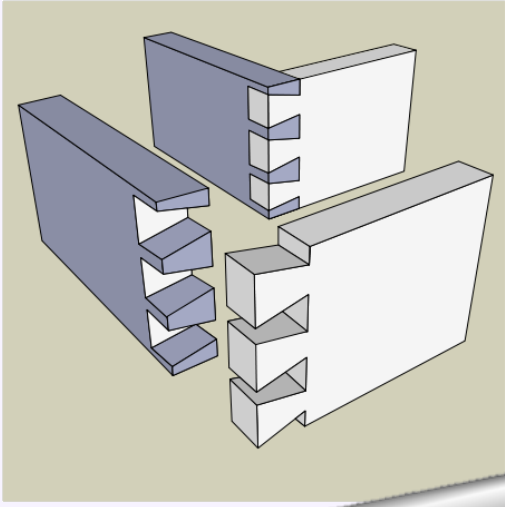
Illustration [Shalloway and Trott]

- Carpenter 1: How do you think we should build these drawers?

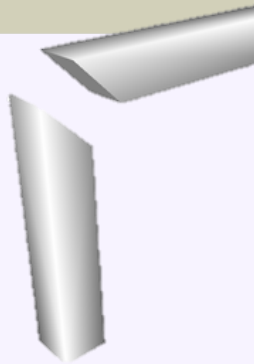
- Carpenter 2: Well, I think we should make the joint by cutting straight down into the wood, and then cut back up 45 degrees, and then going straight back down, and then back up the other way 45 degrees, and then going straight down, and repeating…

- SE example: "I wrote this if statement to handle … followed by a while loop … with a break statement so that…"

A Better Way



- Carpenter 1: Should we use a dovetail joint or a miter joint?



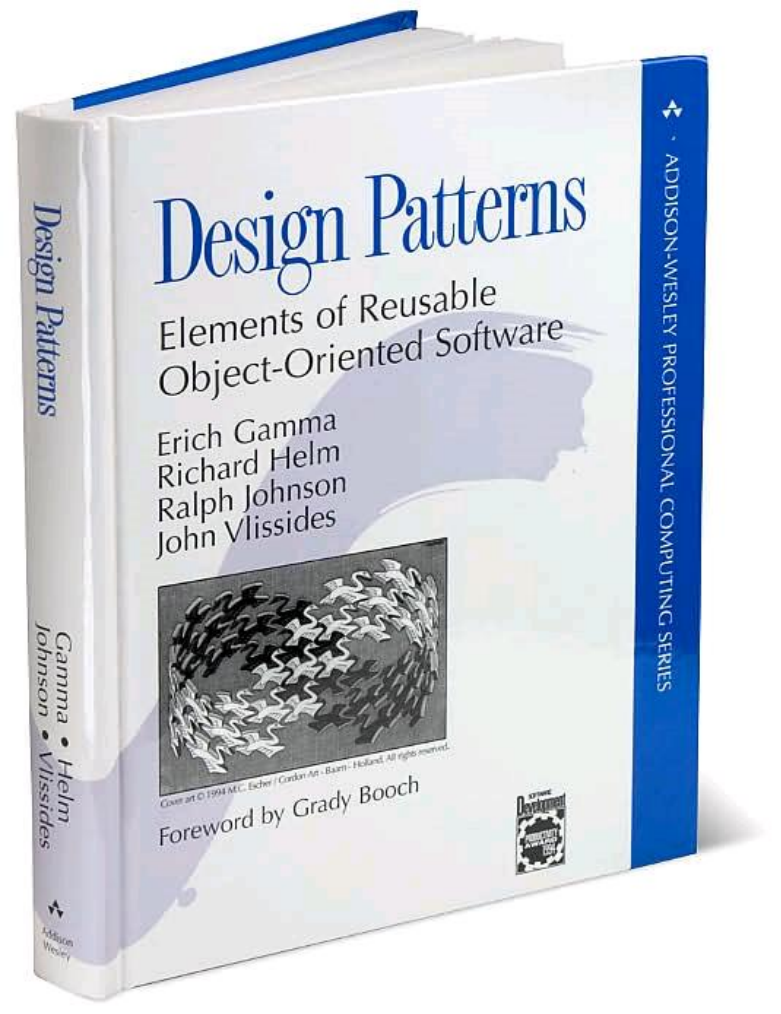
- Subtext:
  - miter joint: cheap, invisible, breaks easily
  - dovetail joint: expensive, beautiful, durable

- Shared terminology and knowledge of consequences raises level of abstraction
  - CS: Should we use a Composite?
  - Subtext
    - Is there a part-whole relationship here?
    - Might there be advantages to treating compositions and individuals uniformly?

Elements of a Pattern

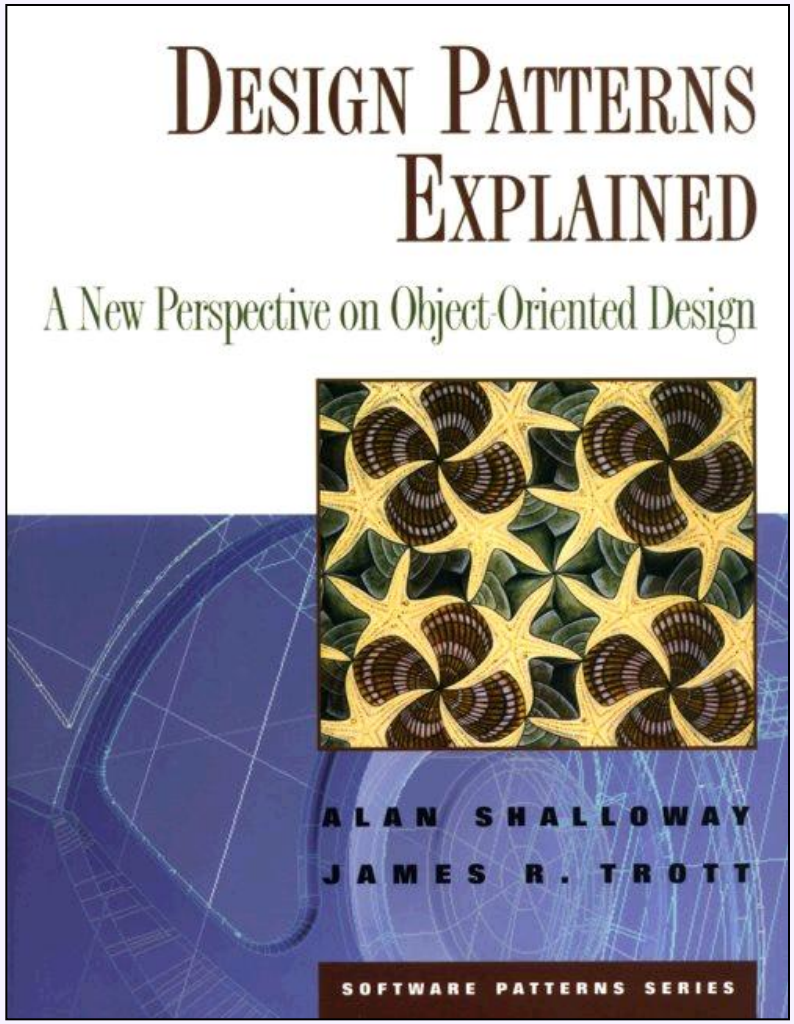
- Name
  - Important because it becomes part of a design vocabulary
  - Raises level of communication
- Problem
  - When the pattern is applicable
- Solution
  - Design elements and their relationships
  - Abstract: must be specialized
- Consequences
  - Tradeoffs of applying the pattern
    - Each pattern has costs as well as benefits
    - Issues include flexibility, extensibility, etc.
    - There may be variations in the pattern with different consequences

History: Design Patterns Book



- Brought Design Patterns into the mainstream
- Authors known as the Gang of Four (GoF)
- Focuses on *descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context*
- Great as a reference text
- Uses C++, Smalltalk

A More Recent Patterns Text



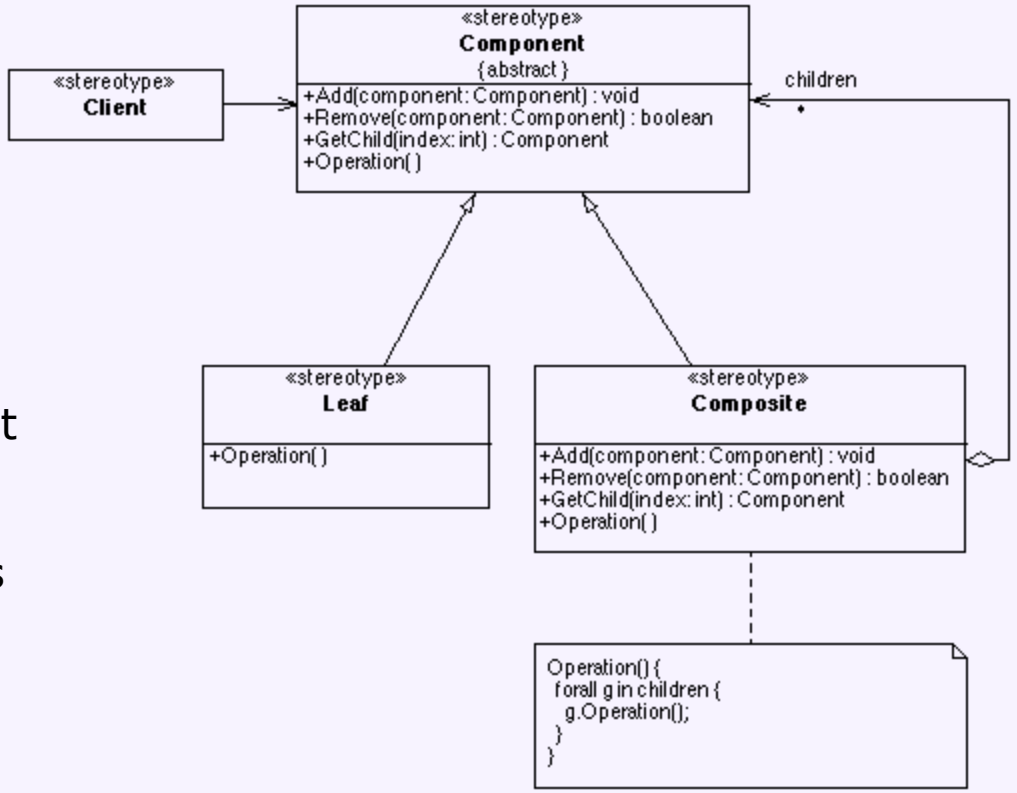
- Uses Java
  - The GoF text was written before Java went mainstream
- Good pedagogically
  - General design information
  - Lots of examples and explanation
  - GoF is really more a reference text
- Mandatory reading
- Helpful for HW4 and 5

Fundamental OO Design Goals and Principles

- Patterns emerge from fundamental design goals and principles applied to recurring problems
  - Reduce *coupling*, increase *cohesion*
  - Design to *interfaces*
  - Favor *composition* over inheritance
  - Design for change (find what varies and encapsulate it)
- Patterns are discovered, not invented
  - Best practice by experienced developers
  - Shared experience

Composite Pattern

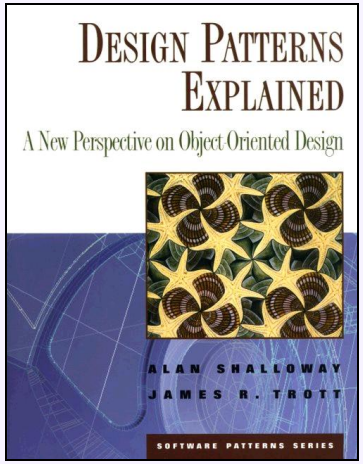
- Applicability
  - You want to represent part-whole hierarchies of objects
  - You want to be able to ignore the difference between compositions of objects and individual objects
- Consequences
  - Makes the client simple, since it can treat objects and composites uniformly
  - Makes it easy to add new kinds of components
  - Can make the design overly general
    - Operations may not make sense on every class
    - Composites may contain only certain components



Patterns to Know

- Façade*, Adapter*, Composite, Strategy*, Bridge*, Abstract Factory*, Factory Method*, Decorator*, Observer*, Template Method*, Singleton*, Command, and Model-View-Controller

- Know pattern name, problem, solution, and consequences
- Know when to use them and when not



* explained in: