SDET Course

Design Patterns - Composite

3 Types of Design Patterns

- Creational
  - Singleton
  - Builder
  - Prototype
  - Factory Method
  - Abstract Factory

- Structural
  - Adapter
  - **Composite**
  - Proxy
  - Flyweight
  - Bridge
  - Facade
  - Decorator

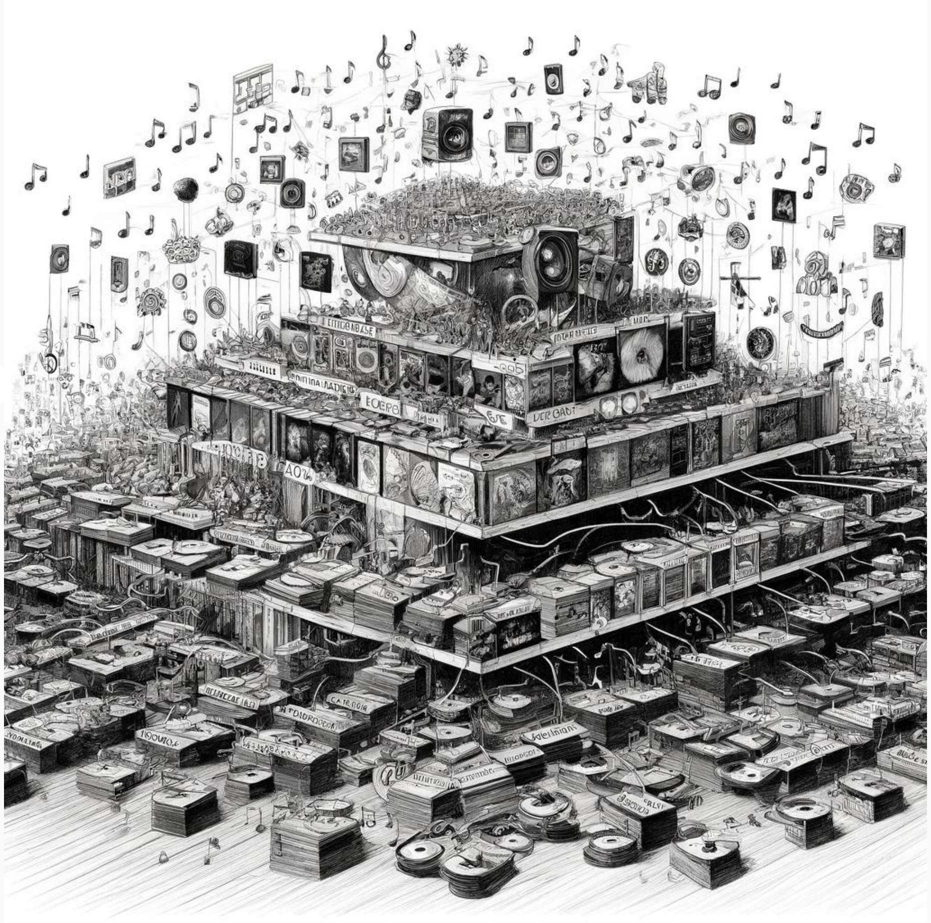
- Behavioral
  - Strategy
  - Observer
  - Command
  - Memento
  - State
  - Template Method
  - Mediator
  - Chain of Responsibility
  - Interpreter
  - Visitor
  - Iterator

Agenda

- Description
- Diagram
- Code sample (Java)
- Use cases

Descrip

on



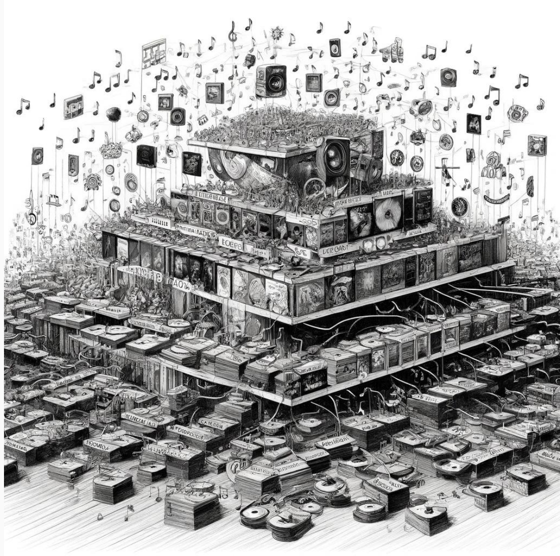
The Problem

- 2 types of elements - Leaf and container
- Container contains containers / leafs
- **Challenge: Generically build / read XML schema**

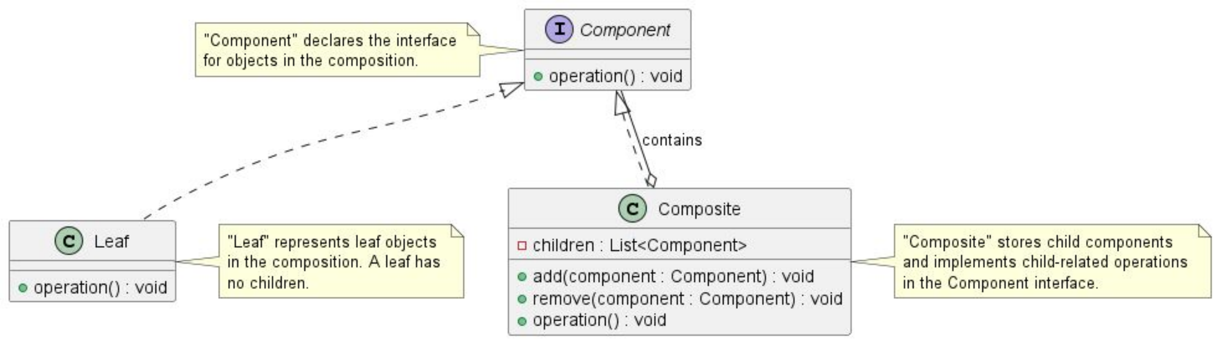
```
<book>
    <title/>
    <author>
        <firstName/>
        <lastName/>
    </author>
</book>
```

Description

The composite pattern describes a group of objects that are treated the same way as a single instance of the same type of object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly.



Class Diagram

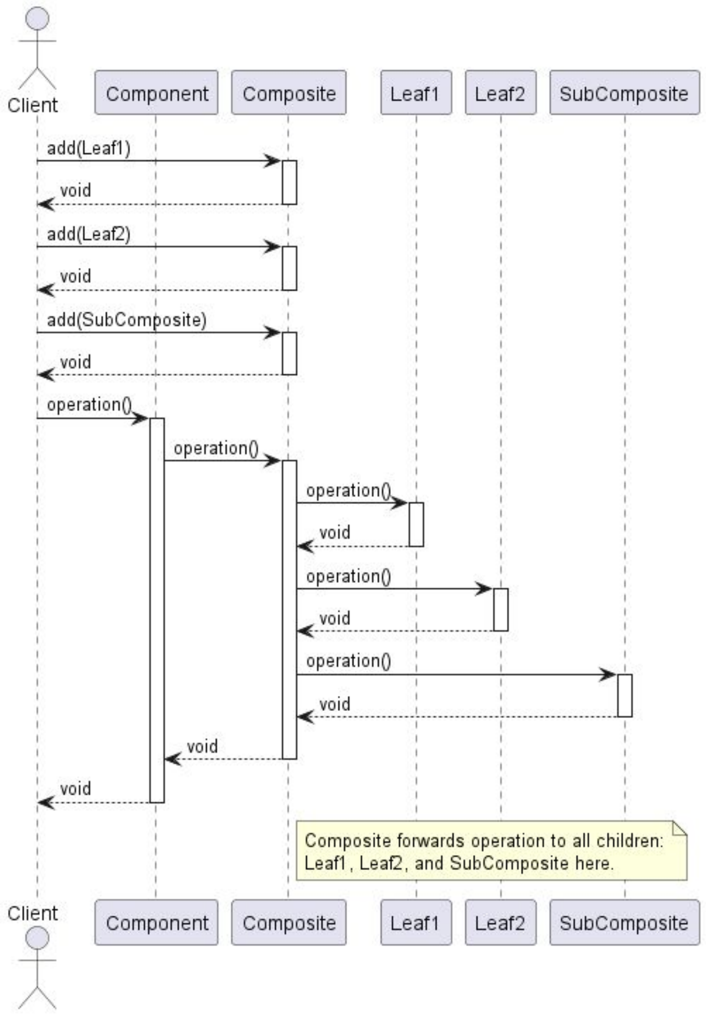

Component
operation() : void
contains
Leaf
operation() : void
Composite
children : List<Component>
add(component : Component) : void
remove(component : Component) : void
operation() : void

"Component" declares the interface for objects in the composition.

"Leaf" represents leaf objects in the composition. A leaf has no children.

"Composite" stores child components and implements child-related operations in the Component interface.

Class Diagram



Composite forwards operation to all children:
Leaf1, Leaf2, and SubComposite here.

Code Sample

Use cases

- General
  - GUI Elements
  - Schema handling (JSON, XML)
  - File Systems

- In Test Automation
  - Page Objects
  - Complicated objects (Salesforce, Dynamics CRM)

Verisoft

Happy Coding