# Composite Design Pattern

Moldovan Balázs

## Why?

- Hierarchical collection of “primitive” and “composite” objects
- Require different handling modes
- Constant querying for object type

## Idea

- Compose objects intro tree structures in order to represent WHOLE-PART hierarchies
- Recursive composition
- 1 to many “has-a” & “is-a” hierarchy

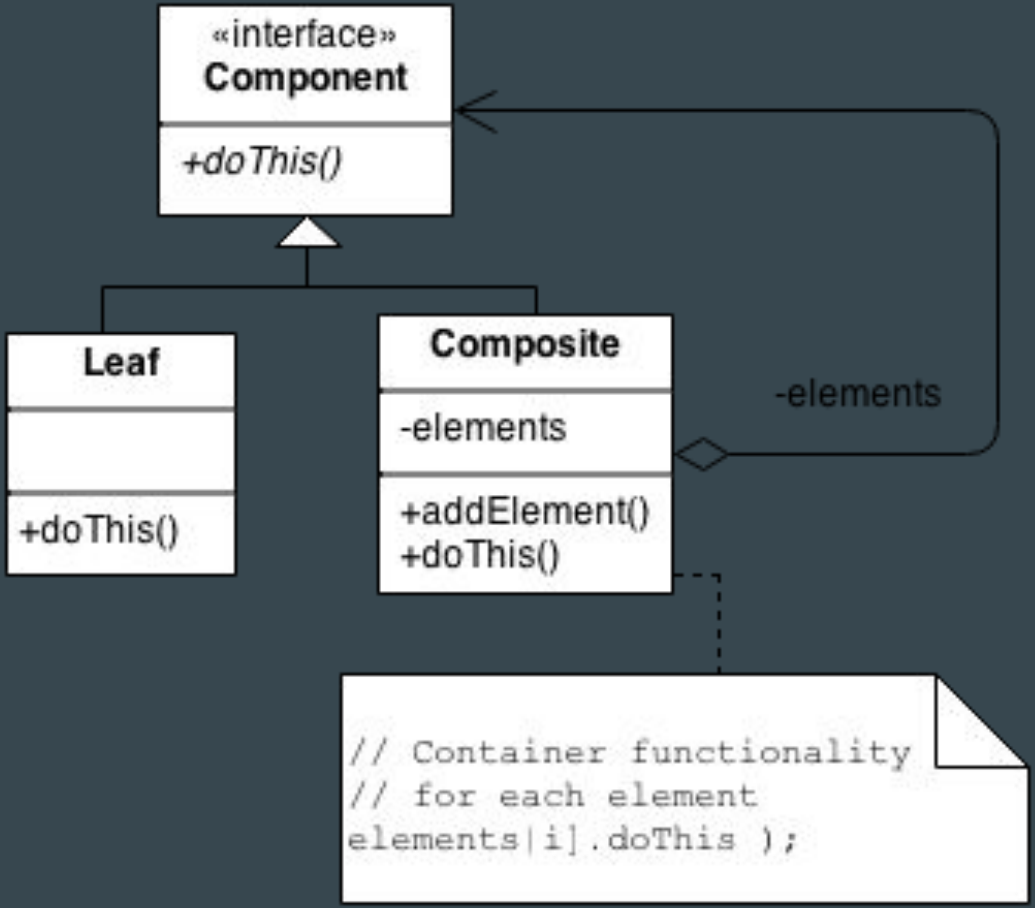
## How?

- Define abstract base class to specify behaviour: Component class
- Subclasses Primitive and Component exercise uniformly the super behaviour

“Composites that contain components,

each of which could be a composite”

- Child management methods defined in Composite class: abstract methods



```
// Container functionality
// for each element
elements[i].doThis );
```

https://sourcemaking.com/design_patterns/composite

## (Dis)/Advantages

### Advantages:

- Clients use the **Component** class interface to interact with objects in the composite structure.
- If call is made to a **Leaf**, the request is handled directly.
- If call is to a **Composite**, it forwards the request to its *child components*.

### Disadvantages:

- Once tree structure is defined, the composite design makes the tree overly general.
- In specific cases, it is difficult to restrict the components of the tree to only *particular types*.
- Therefore, to enforce such constraint, the program must rely on run-time checks, since it cannot use the *type system* the of programming language.

## Examples

- File and Directory Example: Code