# Composite

A structural pattern

## Learning goals

1. Understand the idea of structural patterns.
2. Learn the idea, structure, and Java implementation of the Composite design pattern.
3. Learn to apply the Composite DP in your own programming.

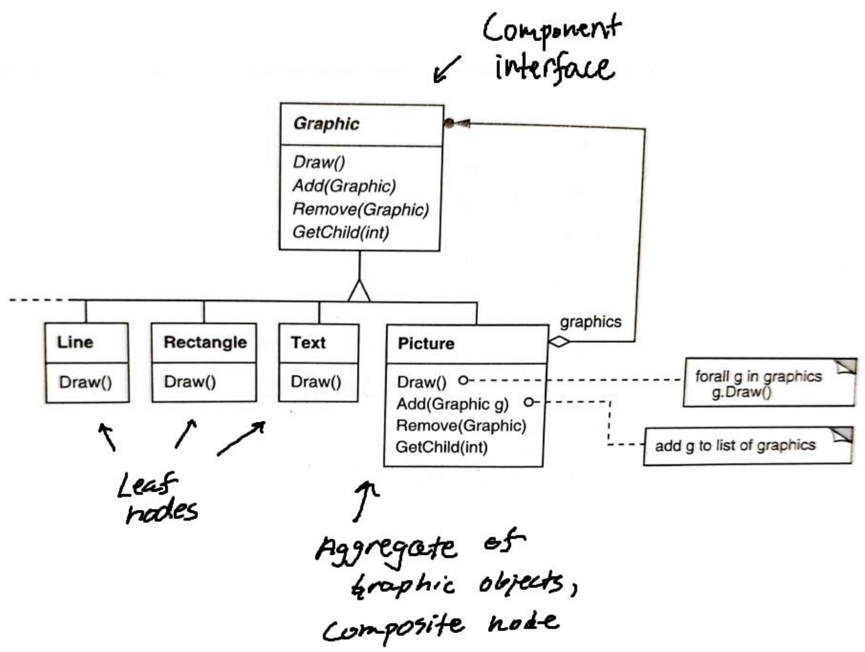
## Structural patterns

- **Structural patterns** provide guidelines for organizing classes and objects into larger structures.
- These patterns focus on simplifying the composition of classes and their relationships.
- Idea: provide ways to design simple components in such a way that they can be used to build larger, complex systems in a modular way.
- Structural DPs: Composite, Adapter, Proxy, Flyweight, Facade, Bridge, Decorator

## Idea of Composite

- Organize objects into a tree-like structure.
- Each node in the tree can be:
  - An Intermediate node, or
  - A leaf node
- Leaf nodes, or terminal nodes, must not have children.
- The Composite design pattern provides a uniform interface for intermediate and leaf nodes.
- The client can treat all components in a uniform way.

# Example explained



- Graphic objects can be Line, Rectangle, Text or Picture object.
- A Picture object contains Graphic objects.
- Note the recursion. The draw() method of a Picture can call the constituting components' draw() method recursively.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 163

# General structure

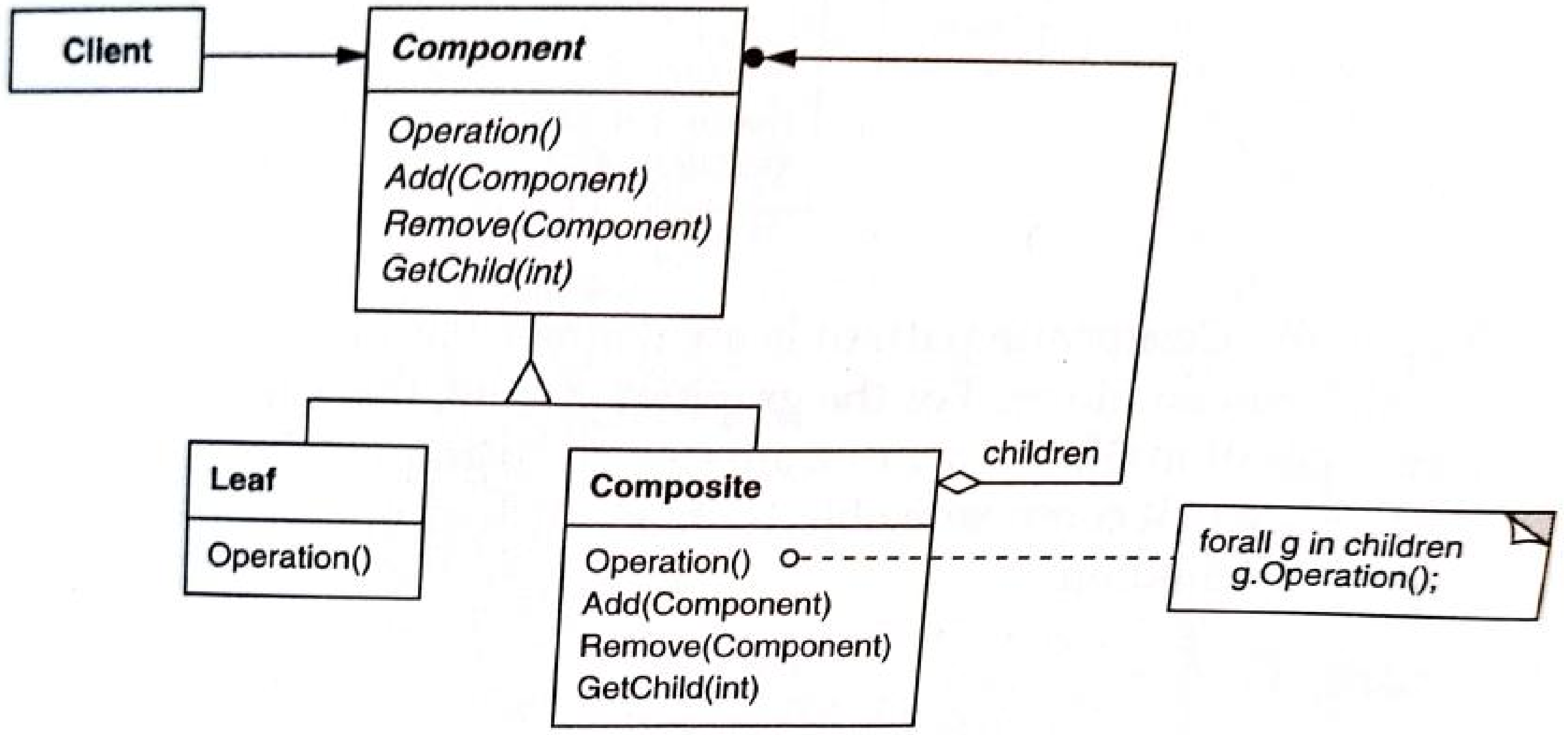


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 164

## Roles

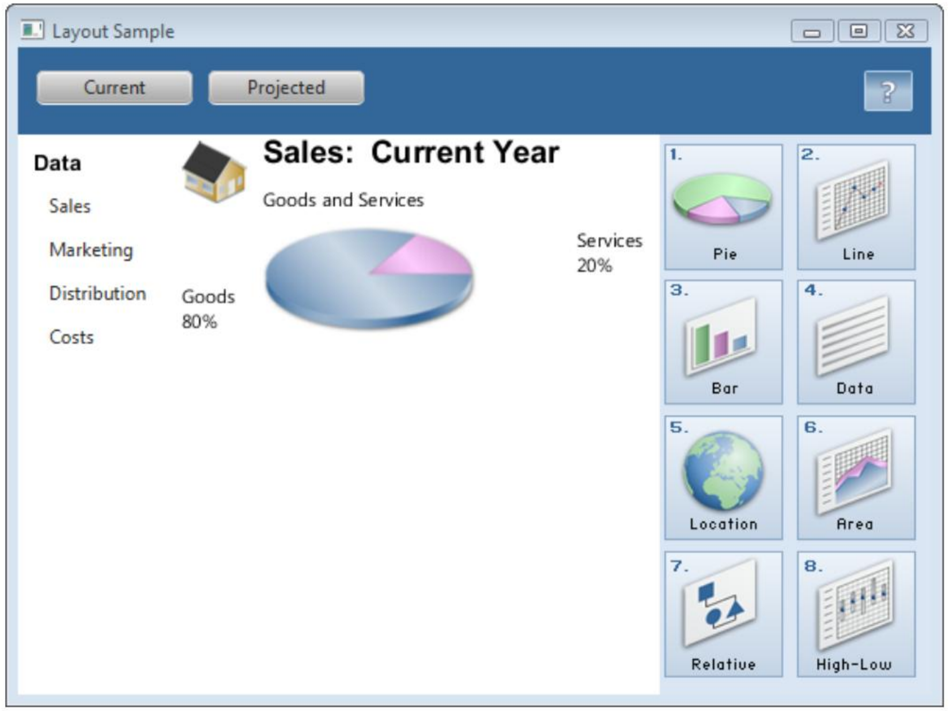
- **Client**: uses the components via the **Component** interface.
  - Can call **operation()** for any component.
- **Component**: implements the interface for all components, whether they are **Leaf** or **Composite** objects.
- **Leaf**: implements the Component interface.
  - For **add()**, **remove()**, **getChild()**, see next slide
- **Composite**: implements the **Component** interface.

## Child management options

- In the previous slide, **add()**, **remove()**, and **getChild()** are not shown for **Leaf** class in the image.
- Java implementation options:
  1. Implement them as dummy methods in the **Leaf** class.
     - Emphasizes transparency: all components can be treated the same way
  2. Move the methods from the **Component** interface into the **Leaf** class.
     - Emphasizes safety: not possible to call illegal operations for the **Leaf** nodes.

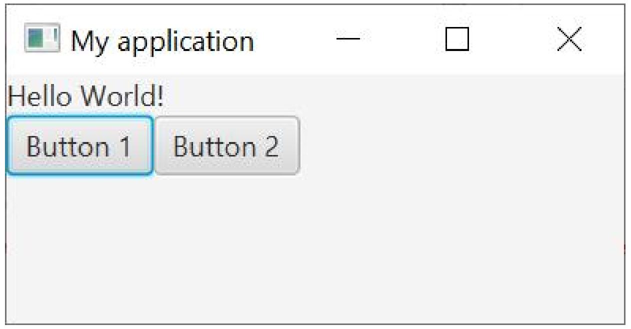
# Example: JavaFX layouts and components

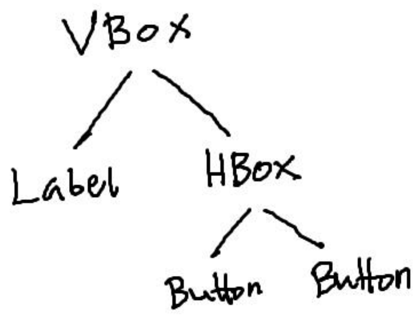
- JavaFX layout patterns can contain other layouts, components, or even nested layouts.
  - Examples: **VBox**, **HBox**, **BorderPane**, and **GridPane**.



# Example: JavaFX layouts and components

```
public class AltGUI extends Application {
    VBox vBox = new VBox();
    HBox hBox = new HBox();
    Label label = new Label("Hello World!");

    public void start(Stage stage){
        stage.setTitle("My application");
        stage.setWidth(300);
        stage.setHeight(300);

        vBox.getChildren().add(label);
        vBox.getChildren().add(hBox);

        hBox.getChildren().add(new Button("Button 1"));
        hBox.getChildren().add(new Button("Button 2"));

        Scene scene = new Scene(vBox);
        stage.setScene(scene);
        stage.show();
    }
}
```


My application
Hello World!
Button 1
Button 2


VBox
Label
HBox
Button
Button