DePaul University

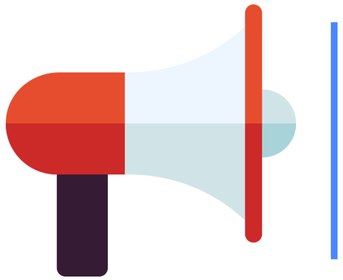
Design Patterns:

Composite, Adapter

Object-oriented Software Development
SE 350– Spring 2021

Vahid Alizadeh

Week 10.1
June 1, 2021



Announcements

Future Schedule

Assignment 3 is graded.

Assignment 4 Solutions > GitHub

- ~~**Assignment 1**~~
- ~~**Assignment 2**~~
- ~~**Mid Term Exam**~~
- ~~**Assignment 3:**~~
  - ~~Release: Week 7~~
  - ~~Due: Week 8~~
- **Assignment 4:**
  - ~~Release: Week 8~~
  - ~~Due: Week 9~~
- **Bonus Research Project:**
  - Presentation Due: Week 10.2 >> **7 Mins**
  - Report Due: Week 11 >> **June 8**
- **Final Exam:**
  - Week 11 - June 9-11 (Wed-Fri)

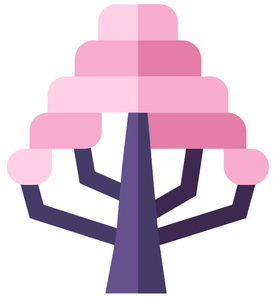
SE 350: OO Software Development

Final Exam

Instructor: Vahid Alizadeh
Email: v.alizadeh@depaul.edu
Quarter: Spring 2021
Date: June 9-11, 2021

DEPAUL UNIVERSITY
COLLEGE OF COMPUTING AND DIGITAL MEDIA

Last update: May 27, 2021

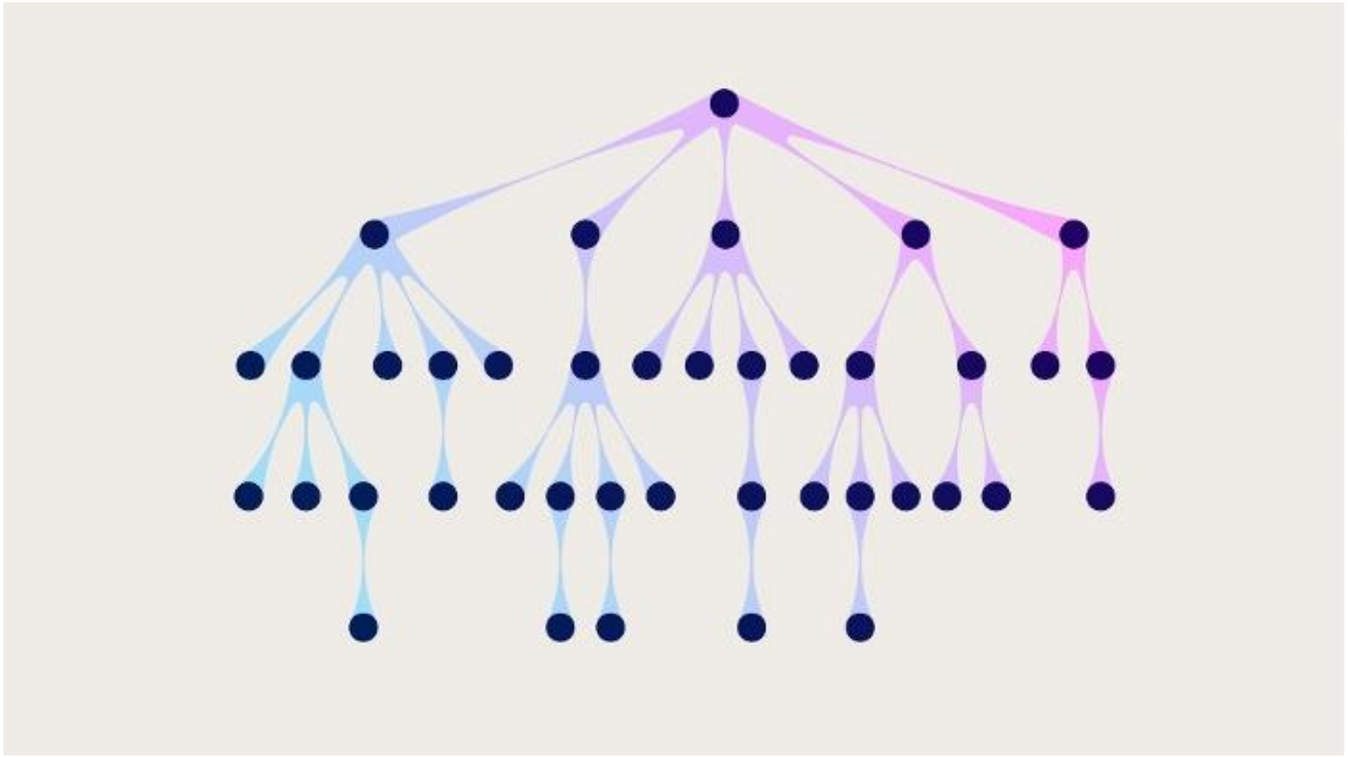


Composite Pattern

STRUCTURAL

Composite Pattern Introduction

Composite is a structural pattern that let us Compose objects into tree structures to represent part-whole hierarchies. Composite allows clients treat individual objects and compositions of objects uniformly.



Composite Design Pattern

INTENT

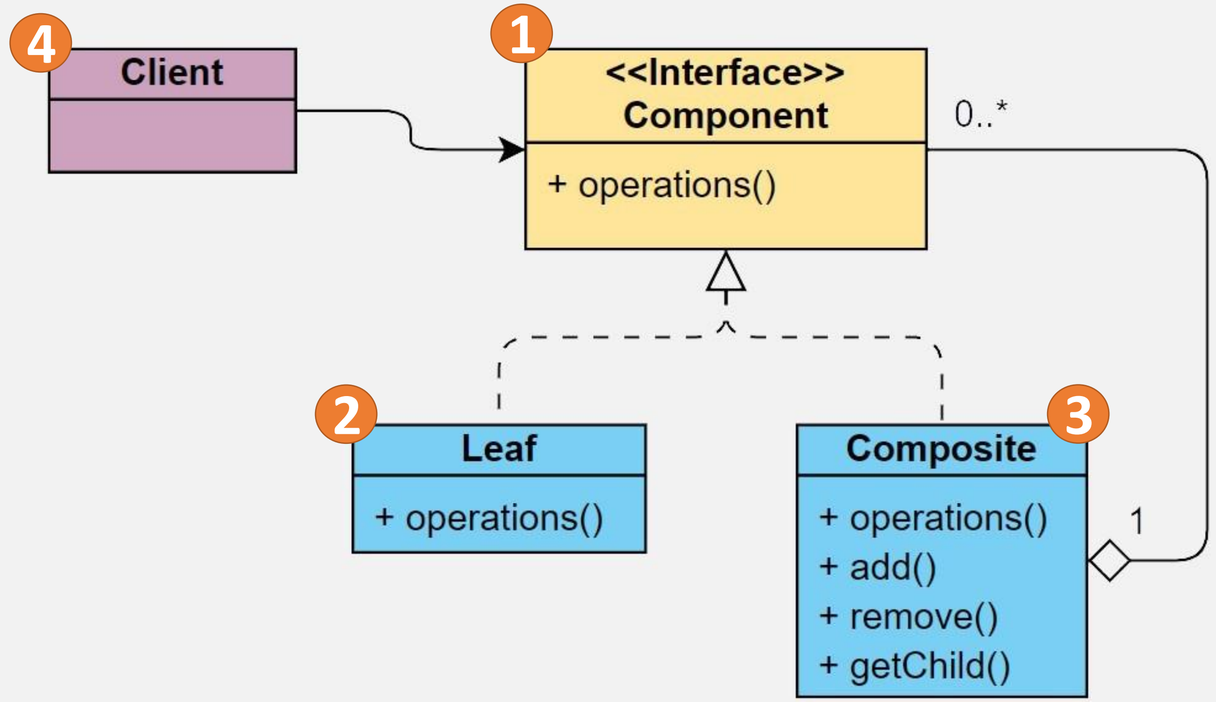
- Modeling of objects into a tree structure and treat them in the same manner.
- The client doesn't know a node is an individual or a composition of objects.

PROBLEM

- Need to manipulate a hierarchy of "primitive" and "composite" objects.

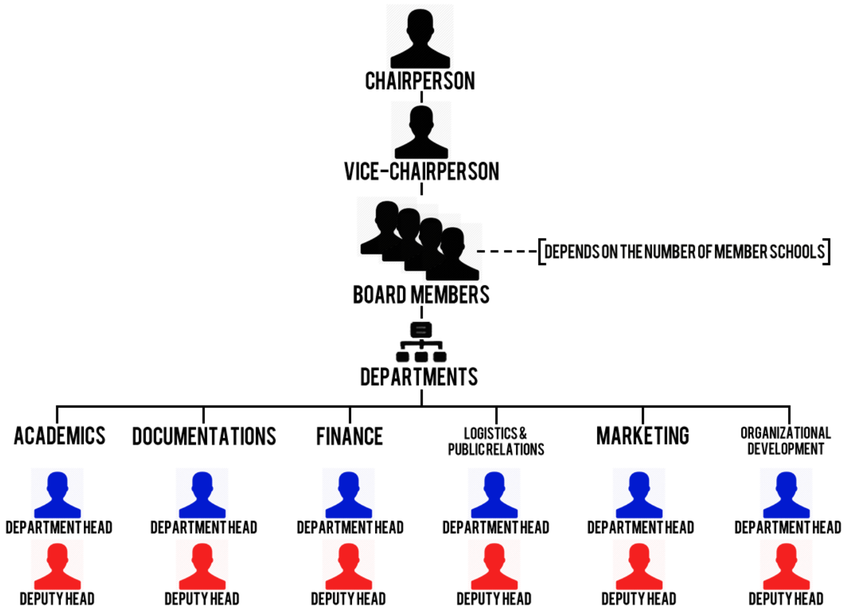
STRUCTURE

- 1- Component Interface
- 2- Leaf Nodes
- 3- Composite
- 4- Client

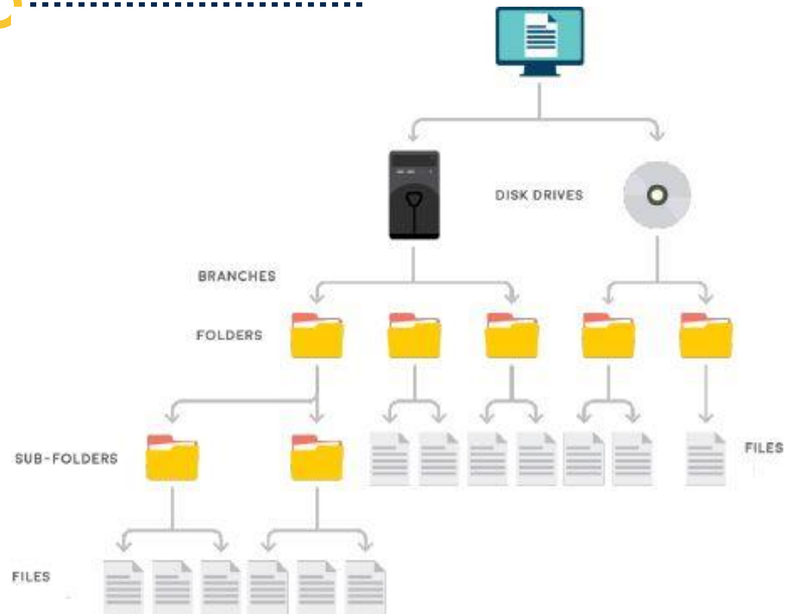


Composite Pattern: Real-world Example

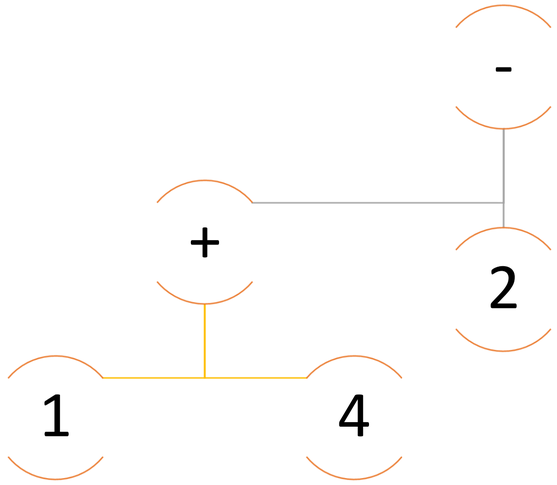
1



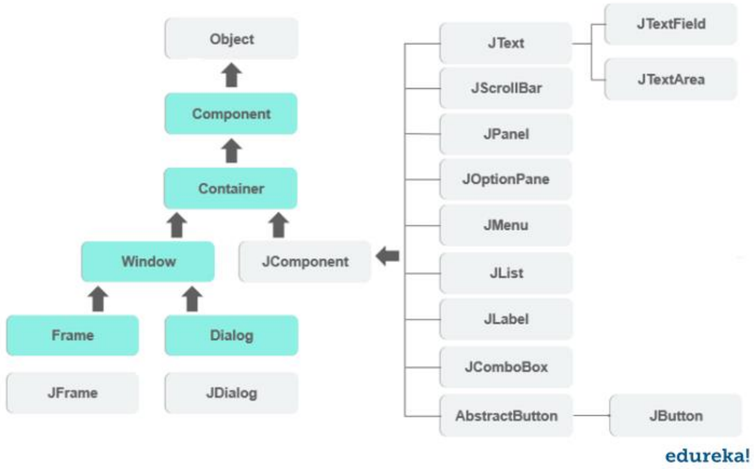
2



3

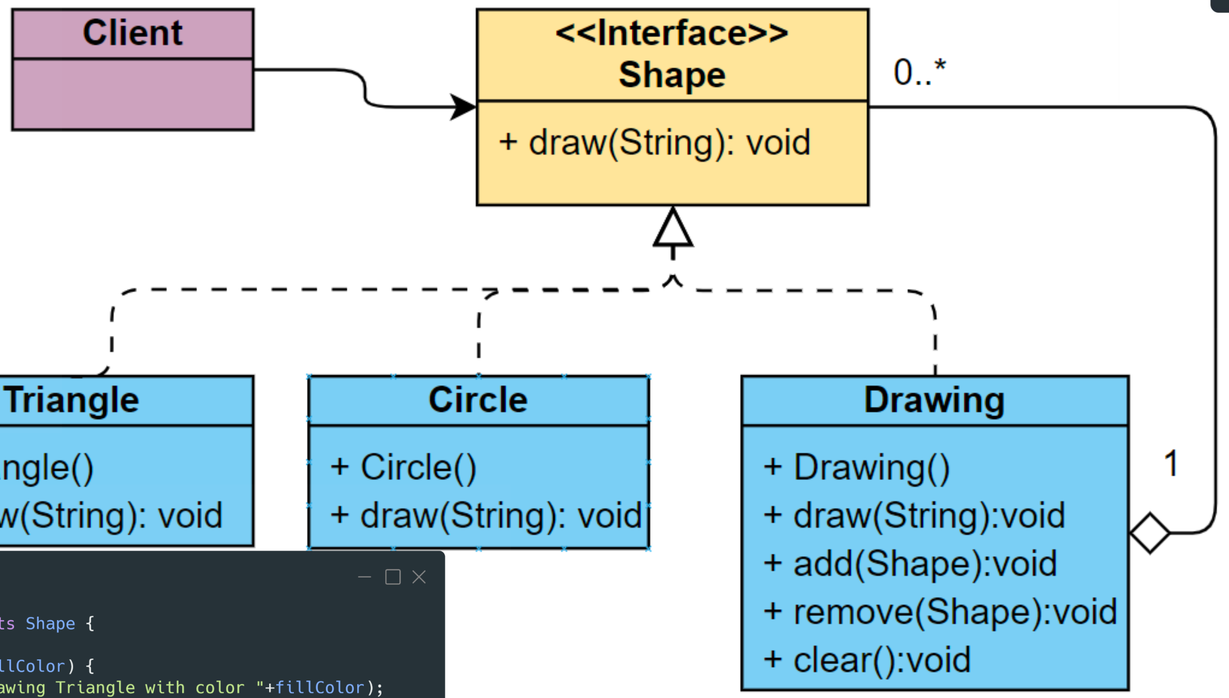


4



Composite Use Case Example: Drawing App

```java
public class CompisiteClient {

    public static void main(String[] args) {
        Shape tri = new Triangle();
        Shape tri1 = new Triangle();
        Shape cir = new Circle();

        Drawing drawing = new Drawing();
        drawing.add(tri1);
        drawing.add(tri1);
        drawing.add(cir);

        drawing.draw("Red");

        drawing.clear();

        drawing.add(tri);
        drawing.add(cir);
        drawing.draw("Green");
    }
}
```




```java
public interface Shape {
    public void draw(String fillColor);
}
```

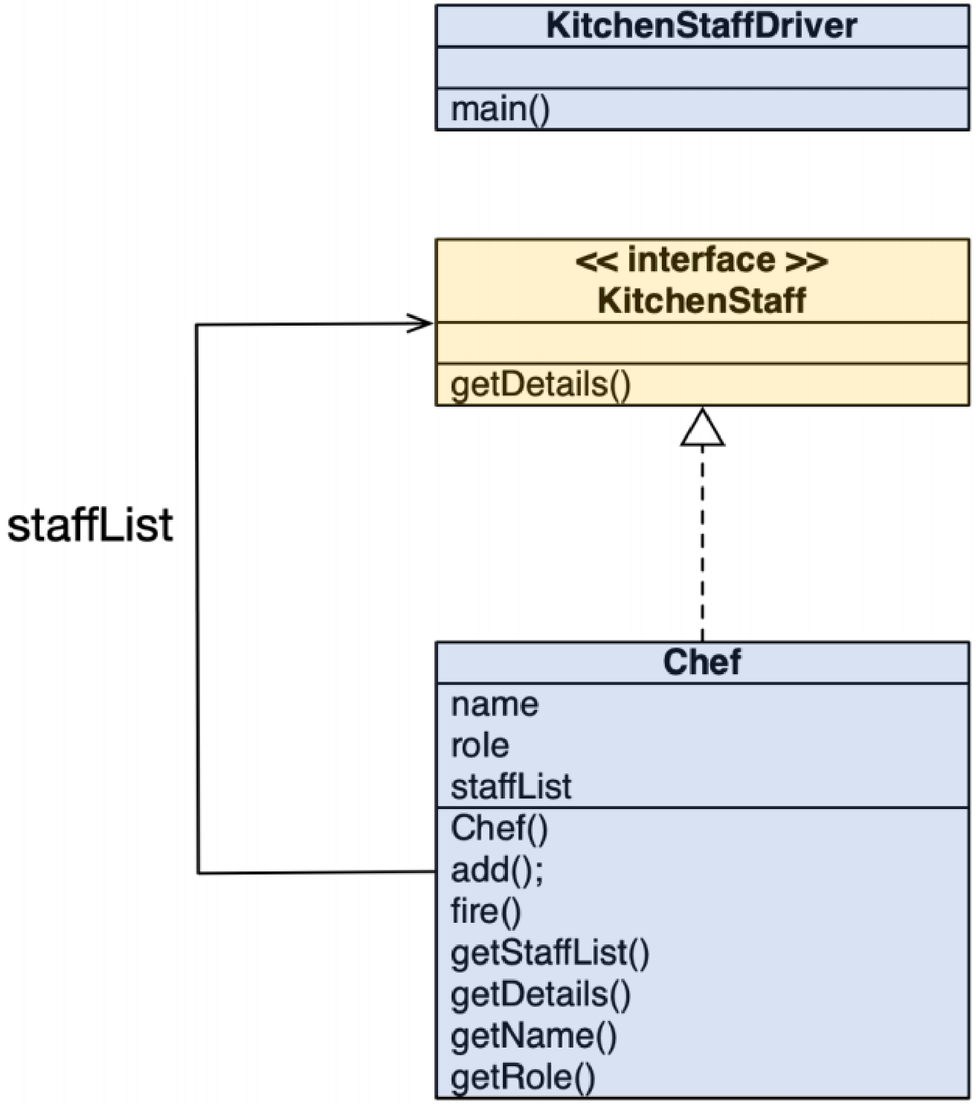
```java
public class Triangle implements Shape {
    @Override
    public void draw(String fillColor) {
        System.out.println("Drawing Triangle with color "+fillColor);
    }
}
```

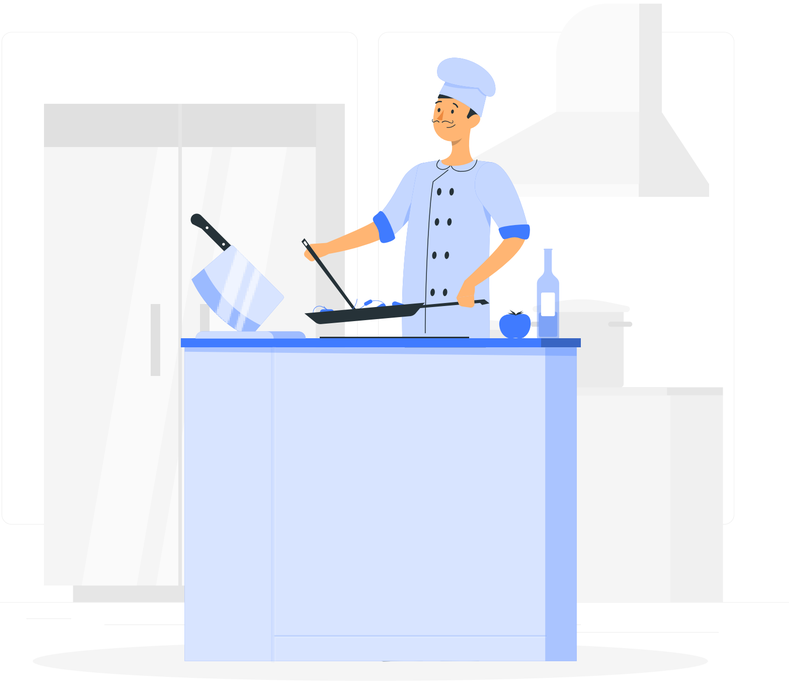
```java
public class Circle implements Shape {
    @Override
    public void draw(String fillColor) {
        System.out.println("Drawing Circle with color "+fillColor);
    }
}
```

```java
public class Drawing implements Shape{
    //collection of Shapes
    private List<Shape> shapes = new ArrayList<Shape>();
    @Override
    public void draw(String fillColor) {
        for(Shape sh : shapes)
        {
            sh.draw(fillColor);
        }
    }
    //adding shape to drawing
    public void add(Shape s){
        this.shapes.add(s);
    }
    //removing shape from drawing
    public void remove(Shape s){
        shapes.remove(s);
    }
    //removing all the shapes
    public void clear(){
        System.out.println("Clearing all the shapes from drawing");
        this.shapes.clear();
    }
}
```


Find the source codes in the course GitHub repository.

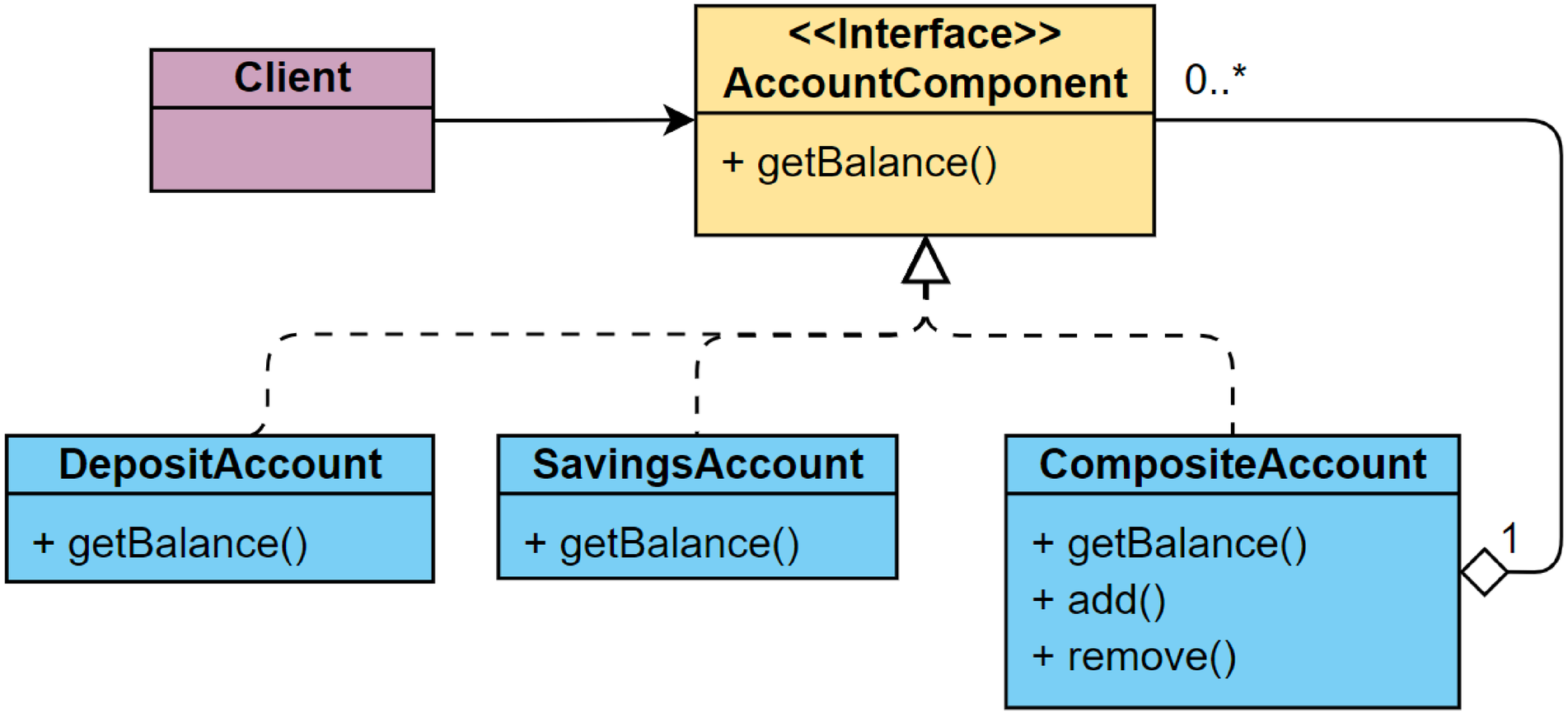
Composite Use Case Example: Kitchen staff management system





Find the source codes in the course GitHub repository.

Composite Use Case Example: Financial App



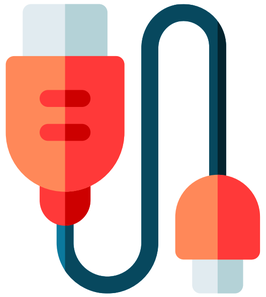
Composite Pattern Pros & Cons

Pros

- Open/Closed Principle: Easy to add new components to our structure.
- Makes the client code simpler.
- Easier to work with complex structures.
- Open/Closed Principle

Cons

- Difficult to define a common interface for every scenario.
- Difficult to restrict the components of the tree to only particular types.

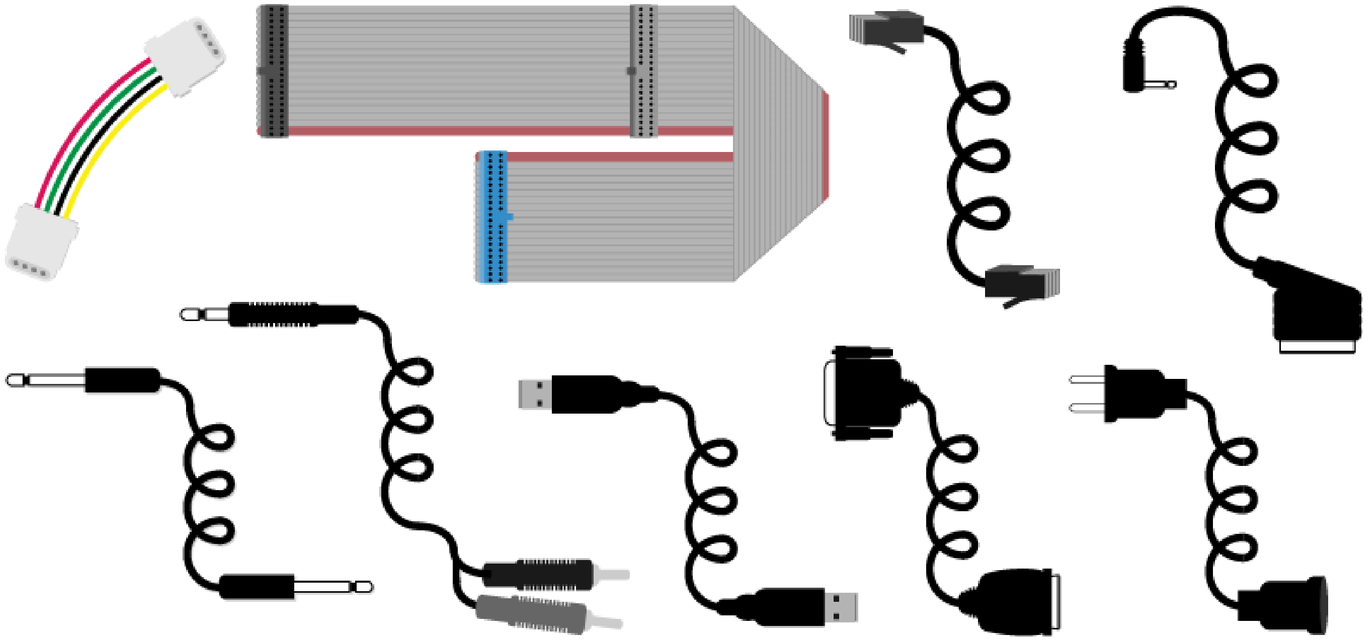


Adapter Pattern

STRUCTURAL

Adapter Pattern Introduction

Adapter is a structural pattern that allows two incompatible interfaces work together. The object that joins these unrelated interface is called an Adapter.



Adapter Design Pattern

INTENT

- Adopt an interface to a new client interface.
- Wrap an old existing class with a new interface.

PROBLEM

- Reusing an already existed component but its representation is not compatible with your architecture.

STRUCTURE

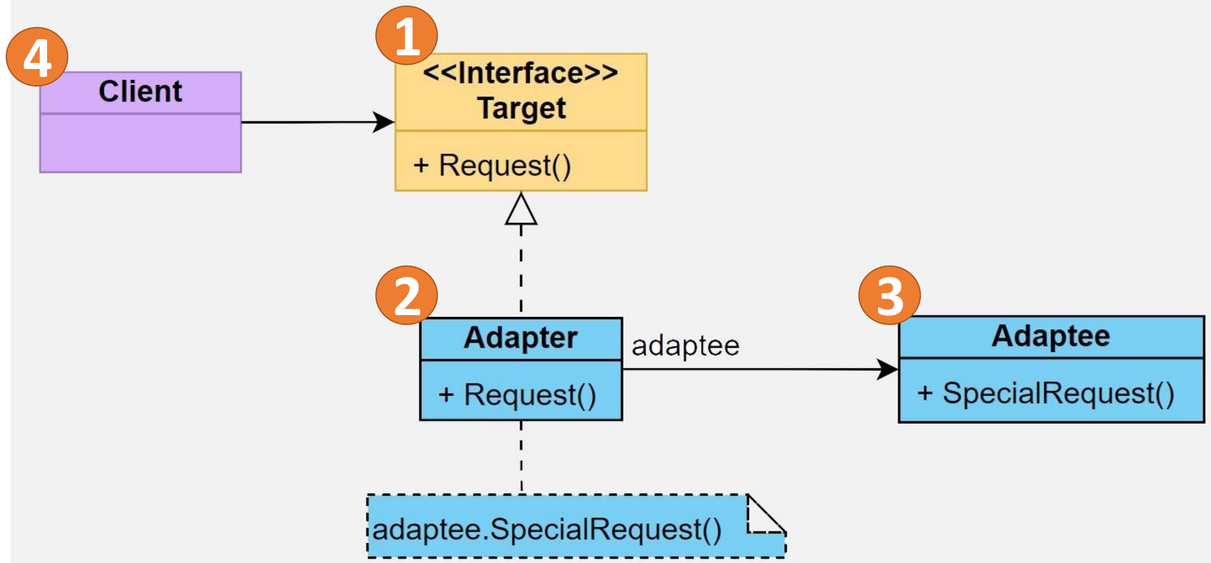
Class Adapter
(via Java Inheritance)

Object Adapter
(via Java Composition)

Adapter Design Pattern: Structures

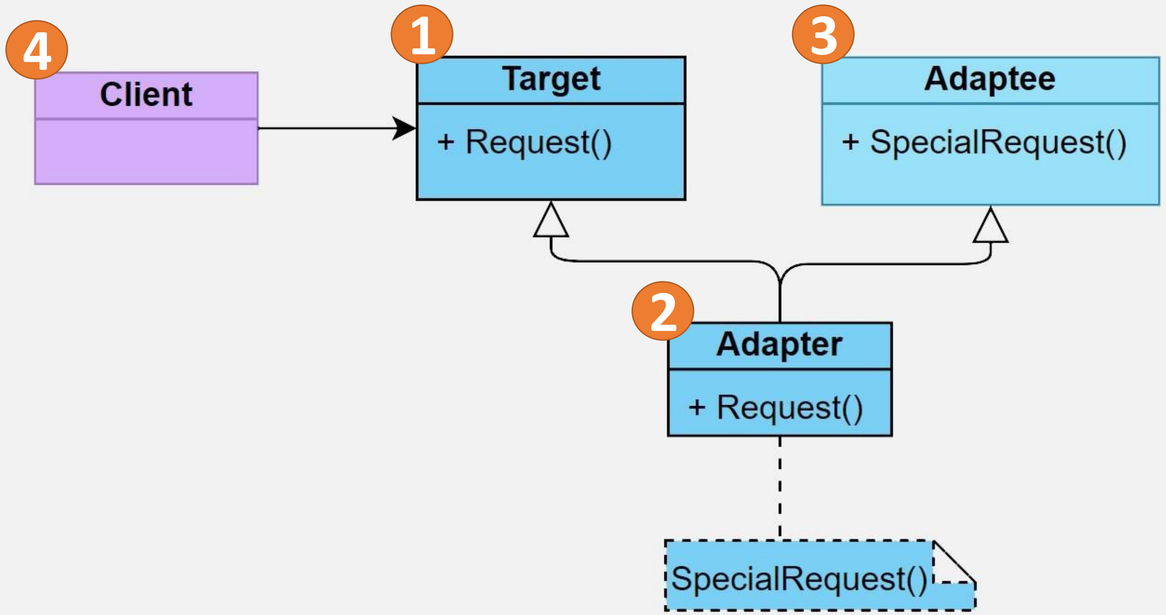
STRUCTURE: Object Adapter

- 1- Target
- 2- Adapter (Uses **composition**)
- 3- Adaptee
- 4- Client



STRUCTURE: Class Adapter

- 1- Target
- 2- Adapter (Uses **inheritance**)
- 3- Adaptee
- 4- Client



Any Question

??????????????

How do you feel about the course?

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Please Send Your Question or Feedback...

Top **New**

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**