Unidade Curricular

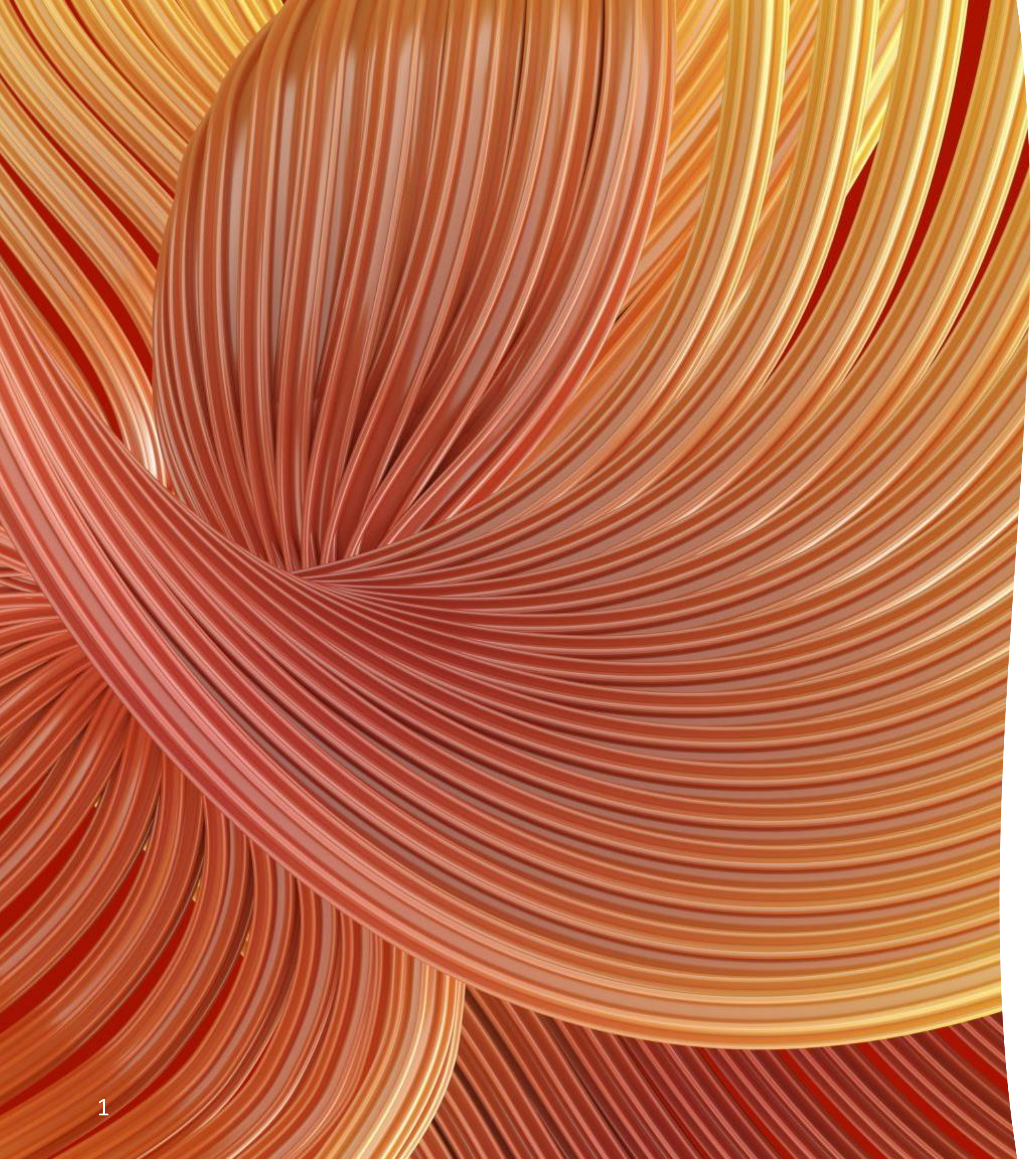
“Padrões e Desenho de Software”

#08 – Structural Patterns (2)

António José Ribeiro Neves

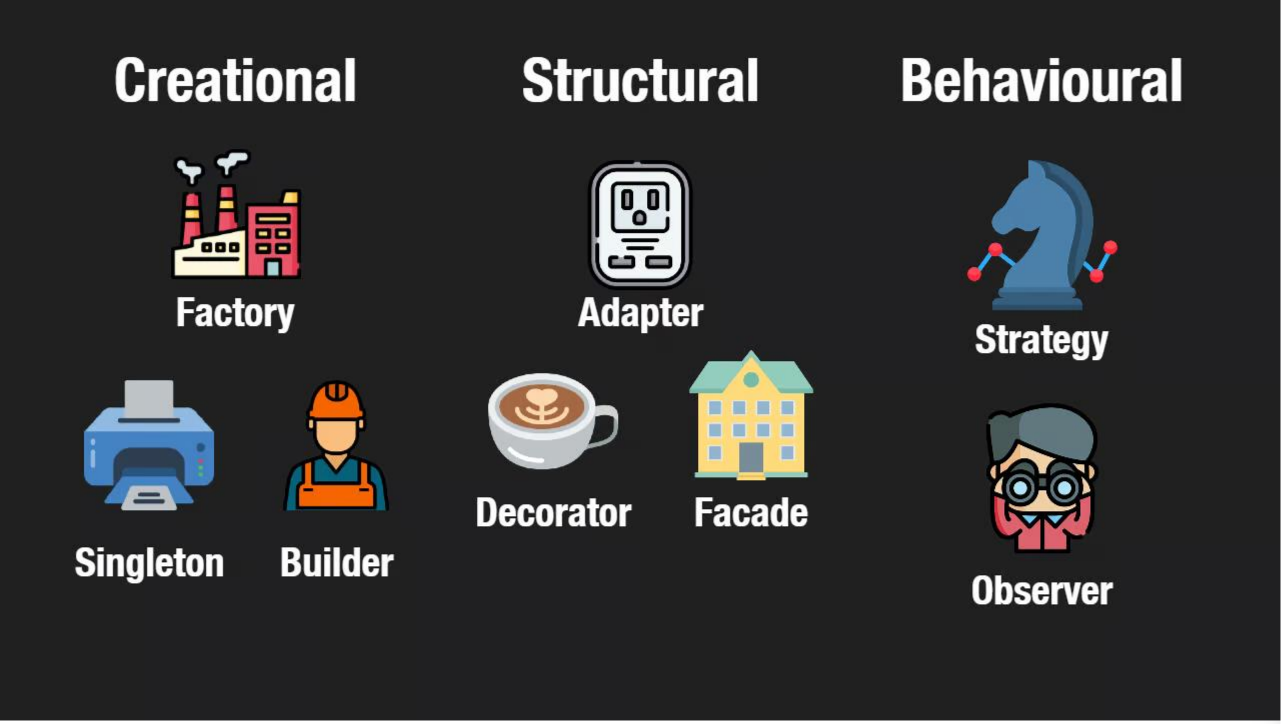
an@ua.pt

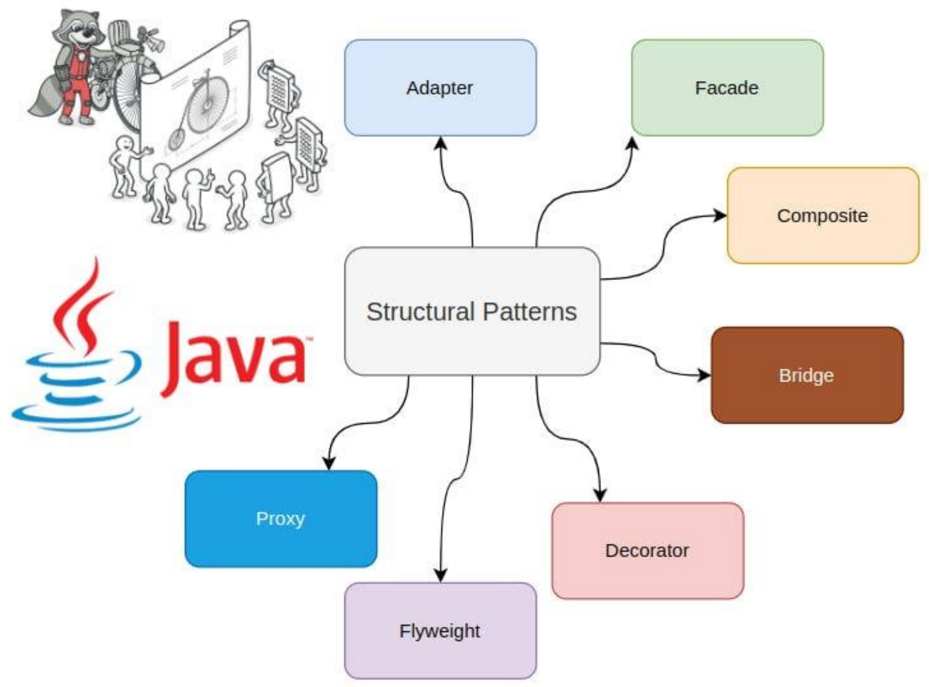
https://www.ua.pt/pt/uc/12275



Outline

- Composite Pattern
- Decorator Pattern
- Presentations

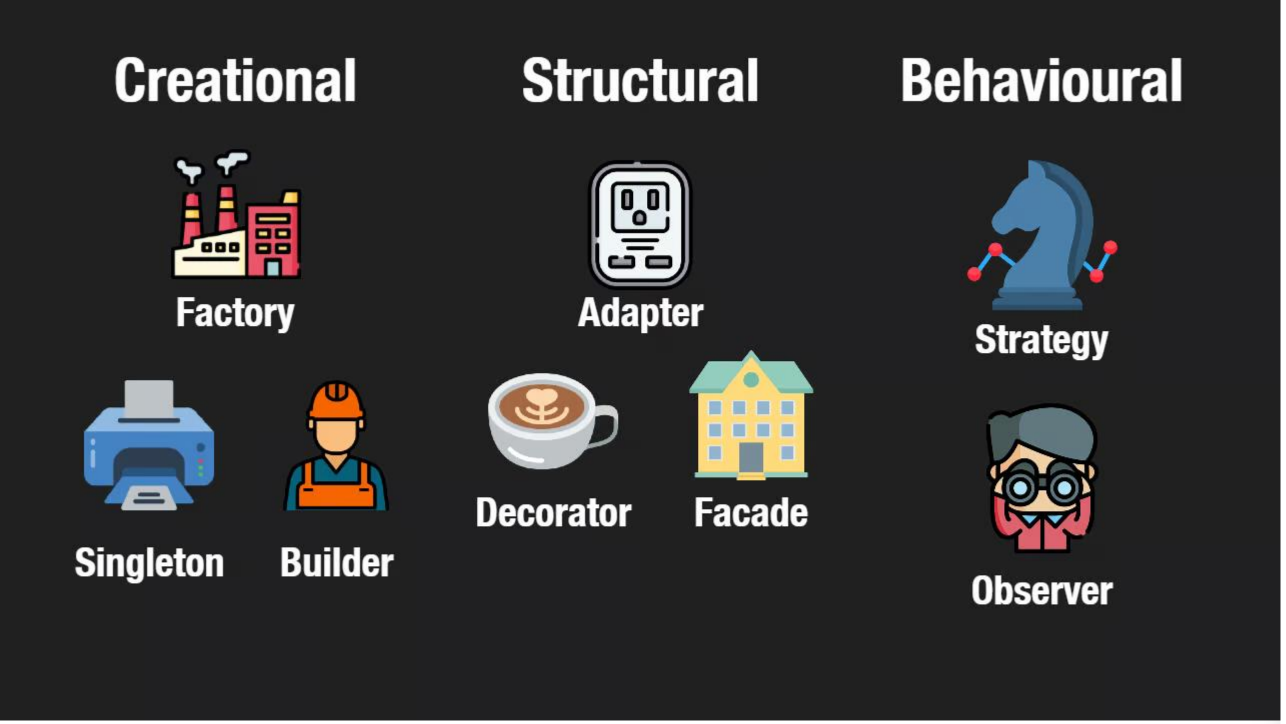


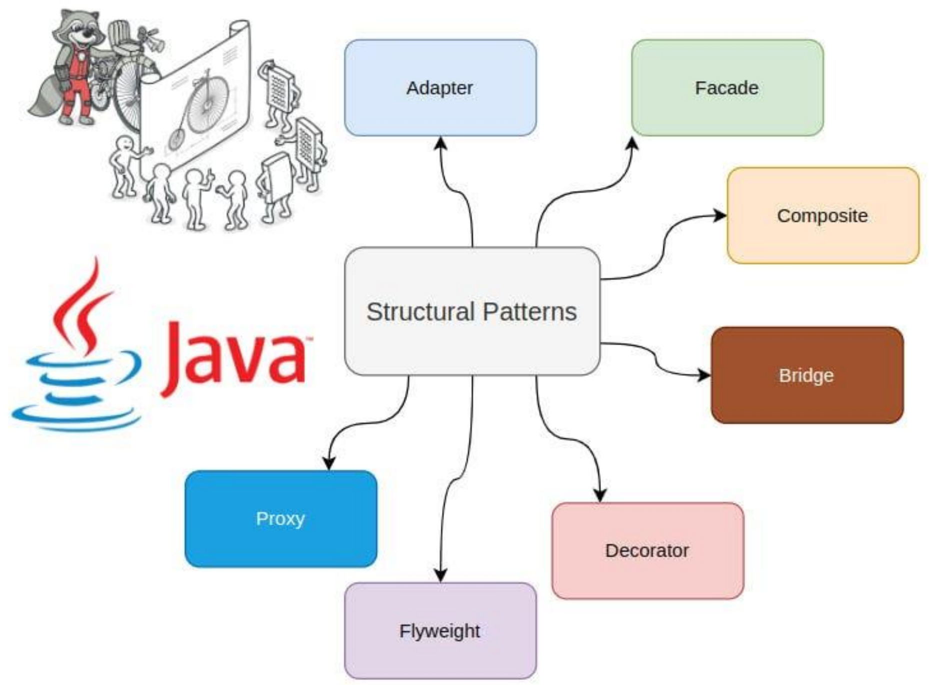


Composite Design Pattern

- **30 minutes** to explore the following problem:

Imagine you are designing a system to represent a company's organizational structure. The structure consists of employees and departments. Each department can contain either individual employees or sub-departments.





Decorator Design Pattern

- **30 minutes** to explore the following problem:

Imagine you are designing a system to represent different types of coffee beverages with various decorations (e.g., milk, sugar, caramel). Each coffee beverage can have multiple optional decorations that enhance its flavor.

Let’s take a short break

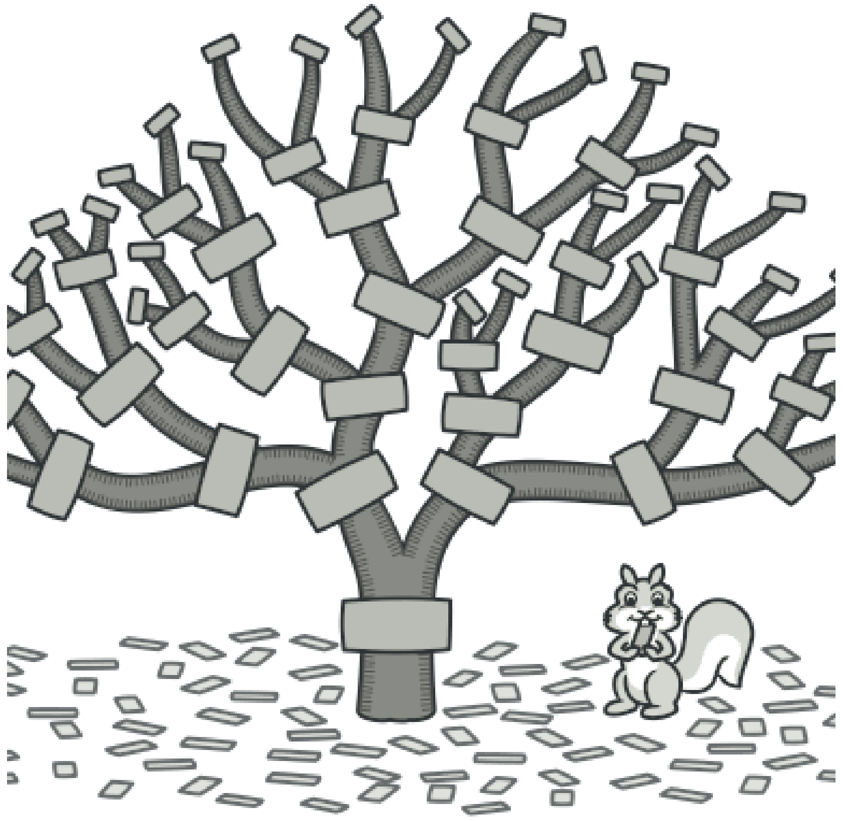
10 Minutes

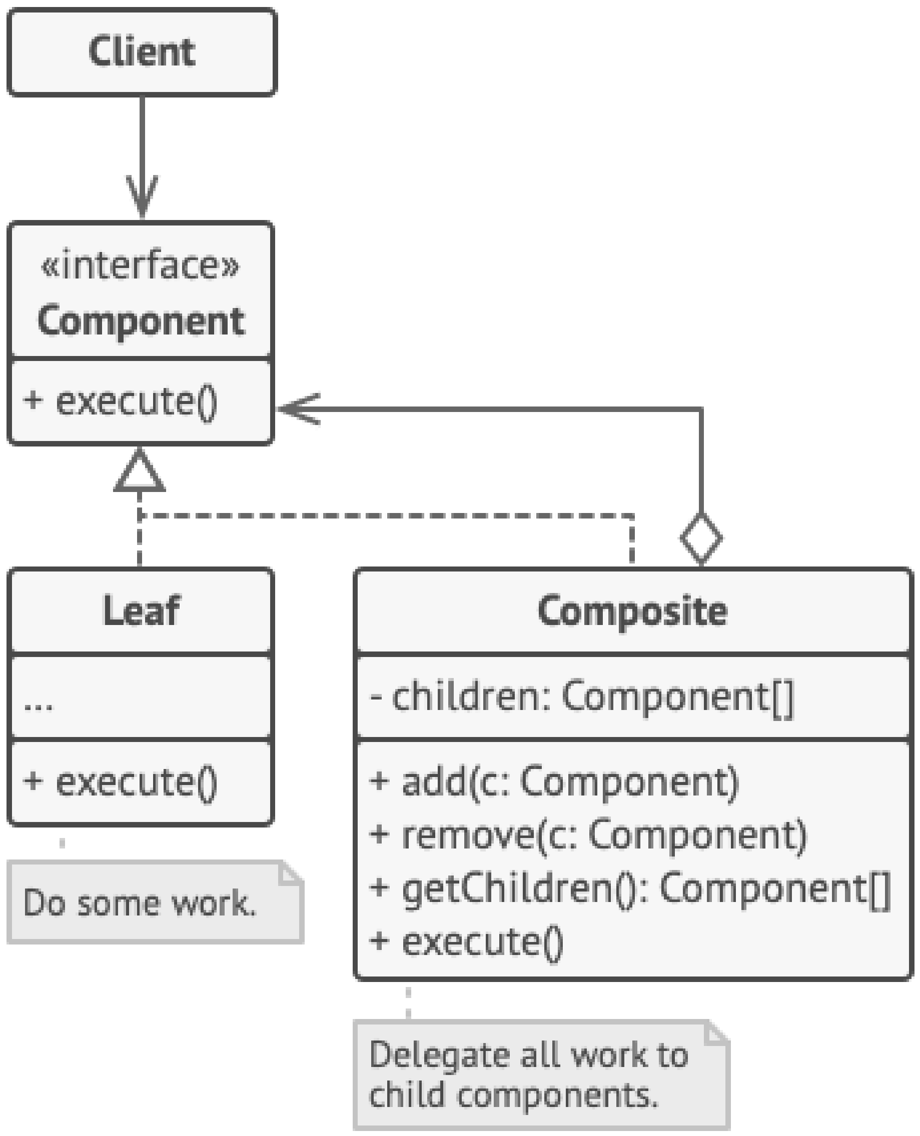
You are free to go grab
a coffee, water, etc.

But… 10 minutes **is 10 minutes** (600 seconds, **not 601 seconds!)**

Composite Pattern

- The Composite pattern is a structural design pattern that allows objects to be composed into tree-like structures to represent part-whole hierarchies.
- **Component**: Defines the interface for objects in the composition and provides default behaviors.
- **Leaf**: Represents individual objects that do not have any children.
- **Composite**: Represents objects that can have child components





Objectives of the Composite Pattern

- Treat individual objects and compositions of objects uniformly.
- Allow clients to work with complex structures of objects without worrying about the details of the hierarchy.
- **Recursive Composition**: Components can be composed of other components recursively, forming a tree structure.
- **Uniform Interface**: Both individual objects (Leaf) and compositions (Composite) share a common interface (Component).
- Simplifies client code by treating objects uniformly.
- Allows for the creation of complex structures using simple objects.

Example

Use case: Representing a company's organizational structure with employees and departments.

Employee interface represents the base component.

IndividualEmployee is a leaf node representing an individual employee.

Department is a composite node representing a department containing employees or sub-departments.

Decorator Pattern

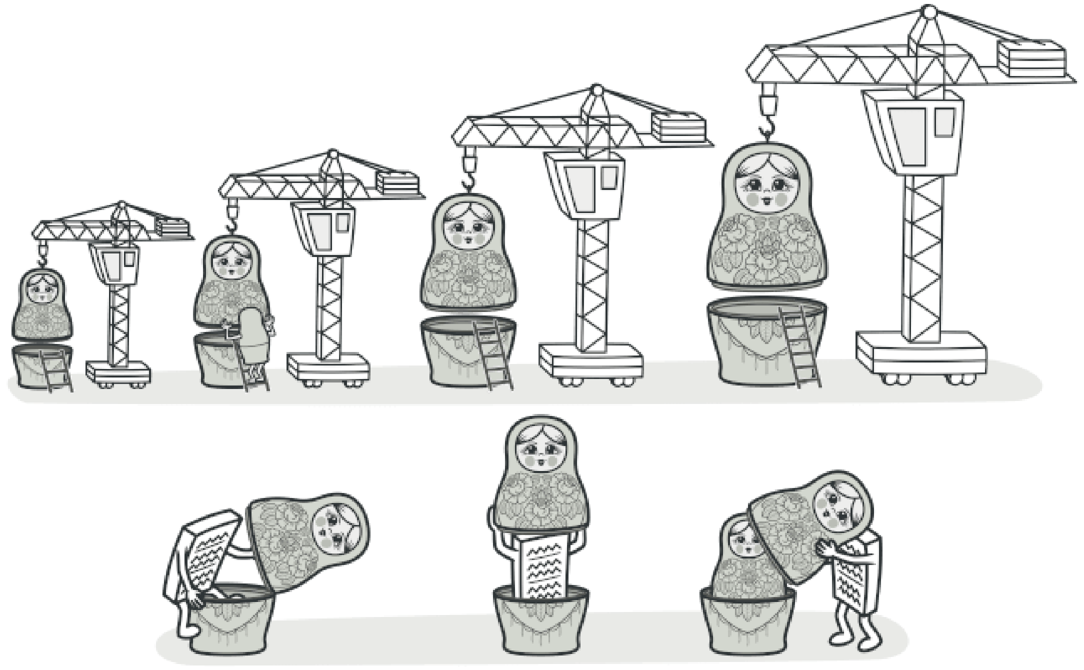
The Decorator pattern is a structural design pattern that allows behaviour to be added to individual objects, either statically or dynamically, without affecting the behaviour of other objects of the same class.

Component: Defines the interface for objects that can have responsibilities added to them dynamically.

ConcreteComponent: Represents the base object to which additional responsibilities can be attached.

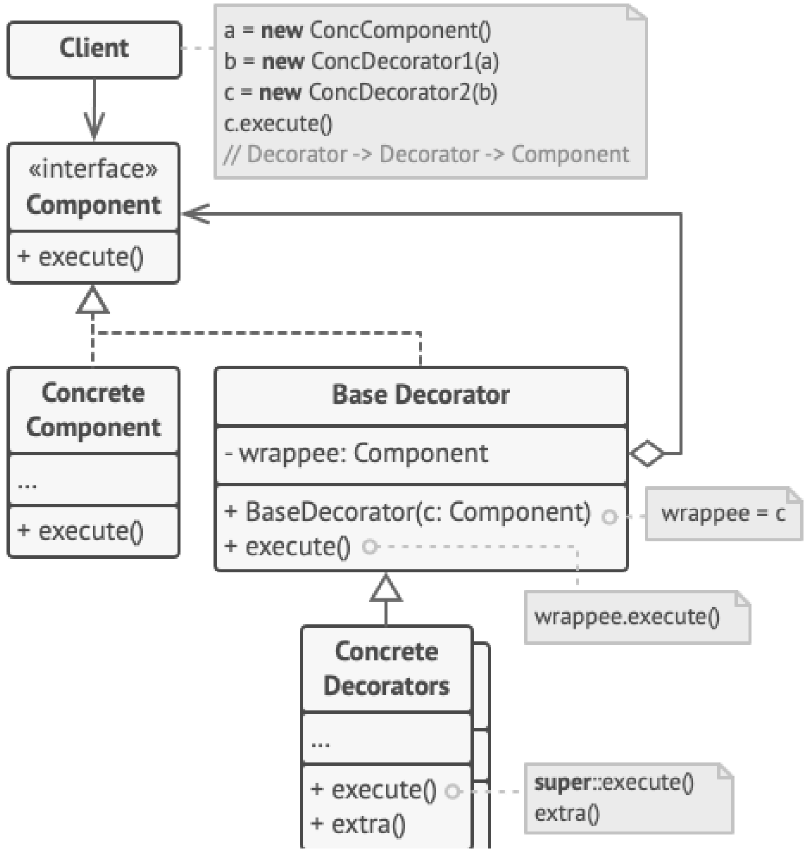
Decorator: Abstract class that implements the Component interface and maintains a reference to a Component object.

ConcreteDecorator: Adds responsibilities to the Component dynamically.



Objectives of the Decorator Pattern

- Enhance the behavior of individual objects without altering their structure.
- Allow for flexible, reusable object composition.
- **Dynamic Behavior Extension**: Decorators can dynamically add new behavior or modify existing behavior of an object at runtime.
- **Composition**: Decorators use composition to add functionality by wrapping objects recursively.
- Promotes open-closed principle: Allows new functionality to be added to existing objects without altering their structure.
- Supports single responsibility principle: Each decorator focuses on a specific responsibility.



Example

- Use case: Decorating coffee beverages with additional ingredients (milk, caramel).
- Coffee interface represents the base component.
- BasicCoffee is a concrete implementation of Coffee.
- CoffeeDecorator is the abstract decorator class.
- MilkDecorator and CaramelDecorator are concrete decorators that add milk and caramel to coffee, respectively.