Composite Design Pattern:

**Other name (if any)**

placeholder

**What it does**

The main idea behind the **Composite Pattern** is to build a tree structure of objects, where individual objects and composite objects share a common interface. This common interface allows clients to interact with both types of objects without needing to distinguish between them. In composite object, it has "has a" relation with its parent. This is kind of recursive call. The **parts** are individual shapes. The **wholes**are the complex shapes.

**Where to use**

1. Used in f**ile Systems.**  
2. Used in o**rganization structures. It m**odels hierarchical organizational structures like departments, teams and employees.  
3. Can be used in calculator.

**Steps**

1. Component Interface: The Component is the common interface for all objects in the composition. It defines the methods that are common to both leaf and composite objects.  
2. Leaf Class: The Leaf is the individual object that does not have any children. It implements the component interface and provides the specific functionality for individual objects.  
3. Composite Class: The Composite is the container object that can hold Leaf objects as well as the other Composite objects. It implements the Component interface and provides methods for adding, removing and accessing children.  
4. Client Code: Here leaf and composite objects will be created using leaf class and composite class. Then by using add function of composite class we can add leaf object or composite object to the composite object.

**Special cases (if any)**

placeholder

**Advantages**

1. The Composite Pattern provides a way to represent complex hierarchical structures of objects. You can create tree-like structures where both individual objects and composite objects (objects that contain other objects) can be treated uniformly.  
2. Clients can work both with individual objects and composites without needing to know the difference.  
3. Clients (code that uses the composite structure) can work both with individual objects and composites without needing to know the difference.

**Disadvantages**

1. Implementing the Composite Pattern can be more complex compared to a non-composite approach.   
2. In some cases, traversing and performing operations on a composite structure can result in performance overhead, especially when dealing with deep and complex hierarchies. This is because you may need to navigate through multiple layers of objects.  
3. Since the Composite Pattern involves a common interface for both leaf and composite objects, it can lead to a lack of type safety. It’s possible to call methods on composite objects that are not applicable, leading to runtime errors.  
4. It might be less obvious when you’re working with a composite structure, and this can make the code harder to understand, especially for developers who are not familiar with the pattern.  
5. Composite structures can consume additional memory due to the need to store references to child objects within composite objects. This can be a concern when dealing with large hierarchies.

**Code**

Coding Concept

**Difference with similar pattern**

placeholder

**Diagram**

Coding Concept