Composite Design Pattern

Composite pattern is a partitioning design pattern and describes a group of objects that is treated the same way as a single instance of the same type of object.

The intent of a composite is to “compose” objects into tree structures to represent part-whole hierarchies. It allows you to have a tree structure and ask each node in the tree structure to perform a task.

* As described by Gof, “Compose objects into tree structure to represent **part-whole hierarchies**. Composite lets client treat individual objects and compositions of objects uniformly”.
* When dealing with Tree-structured data, programmers often have to discriminate between a leaf-node and a branch. This makes code more complex, and therefore, error prone. The solution is an **interface** that allows treating complex and primitive objects uniformly.
* In object-oriented programming, a composite is an object designed as a composition of one-or-more similar objects, all exhibiting similar functionality. This is known as a **“has-a”** relationship between objects.

**The Composite Pattern has four participants:**

* 1. **Component –**Component declares the interface for objects in the composition and for accessing and managing its child components. It also implements default behaviour for the interface common to all classes as appropriate.
  2. **Leaf –**Leaf defines behaviour for primitive objects in the composition. It represents leaf objects in the composition.
  3. **Composite –**Composite stores child components and implements child related operations in the component interface.
  4. **Client –** Client manipulates the objects in the composition through the component interface.

Client use the component class interface to interact with objects in the composition structure. If recipient is a leaf then request is handled directly. If recipient is a composite, then it usually forwards request to its child components, possibly performing additional operations before and after forwarding.