**The Scenario**:

Imagine you have an organizational structure with employees and departments. You want to display this structure in a way that employees and departments are treated the same way when it comes to viewing the organization's hierarchy.

**Composite Design Pattern Explained**:

* **Part-Whole Hierarchy**: The Composite design pattern is all about creating a structure where individual objects (leaves) and compositions of objects (composites) are treated uniformly.
* **Common Interface**: You create a common interface, IOrganizationComponent, which both employees and departments implement. This interface ensures that all components (employees and departments) have consistent methods like Display().
* **Leaf Component**: You implement the Employee class, which represents individual employees. This class implements the IOrganizationComponent interface. Each employee is a "leaf" component because they don't contain any sub-components.
* **Composite Component**: You implement the Department class, which represents departments within the organization. This class also implements the IOrganizationComponent interface. Each department is a "composite" component because it can contain other components (employees or sub-departments).
* **Constructing the Organization**: In the Main method, you construct the organizational structure. You create instances of employees and departments and assemble them into a hierarchical structure.
* **Uniform Display**: When you call the Display() method on any component (employee or department), the Composite pattern ensures that the correct implementation is called. This makes it possible to treat individual components and composite structures the same way when displaying the organization's hierarchy.

**In Simple Words**: Think of the Composite pattern like building with LEGO bricks. Each brick (employee) can stand alone and be viewed individually. But you can also build more complex structures (departments) by stacking bricks together. Whether it's a single brick or a tower of bricks, they all have the same basic properties and can be handled in a consistent manner.

In this code, the Composite pattern lets you create an organized representation of an organization's hierarchy, where employees and departments are all considered components. This helps in displaying the structure and performing operations on the components in a unified way.

**Step 1: Define Interface**

In this step, you define the IOrganizationComponent interface. This interface provides a common structure for both individual employees and departments within the organization.

internal class Program

{

**// Interface for both employees and departments**

interface IOrganizationComponent

{

string Name { get; }

void Display();

}

**// ...**

}

**Step 2: Implement Leaf Components**

In this step, you create a Employee class that implements the IOrganizationComponent interface. Each employee is a "leaf" component.

class Employee : IOrganizationComponent

{

**// ...**

}

**Step 3: Implement Composite Components**

In this step, you create a Department class that implements the IOrganizationComponent interface. A department is a "composite" component that can contain other components, which can be either employees or sub-departments.

class Department : IOrganizationComponent

{

**// ...**

}

**Step 4: Construct the Organization Structure**

In this step, you create instances of departments and employees and construct an organizational structure by adding employees and sub-departments to the CEO's office.

static void Main(string[] args)

{

var ceo = new Department("CEO Office");

var hrDepartment = new Department("HR Department");

var techDepartment = new Department("Tech Department");

**// Adding employees and departments to the CEO's office**

ceo.Add(new Employee("John CEO"));

ceo.Add(hrDepartment);

ceo.Add(techDepartment);

**// Adding employees to the HR department**

hrDepartment.Add(new Employee("Alice HR"));

hrDepartment.Add(new Employee("Bob HR"));

**// Adding employees to the Tech department**

techDepartment.Add(new Employee("Eve Developer"));

techDepartment.Add(new Employee("Charlie Developer"));

ceo.Display();

}

**Explanation of Composite Pattern**:

The Composite design pattern is used to compose objects into tree structures to represent part-whole hierarchies. It lets clients treat individual objects and compositions of objects uniformly.

In this example,

* IOrganizationComponent interface serves as the basis for both individual employees and departments. This allows you to treat them uniformly when displaying the organizational structure.
* The Employee class represents individual employees and implements the IOrganizationComponent interface. It's a "leaf" component because it doesn't contain any other components.
* The Department class represents departments within the organization and implements the IOrganizationComponent interface. It's a "composite" component because it can contain other components, either employees or sub-departments.
* In the Main method, you create instances of departments and employees, and you construct the organization's structure by adding employees and sub-departments to the CEO's office.
* When you call the Display method on the CEO's office (ceo.Display()), it displays the hierarchical structure of the organization, including employees and sub-departments.

In this code example, the Composite pattern helps you represent the organization's hierarchy in a structured manner, allowing you to treat individual components and composite structures uniformly.