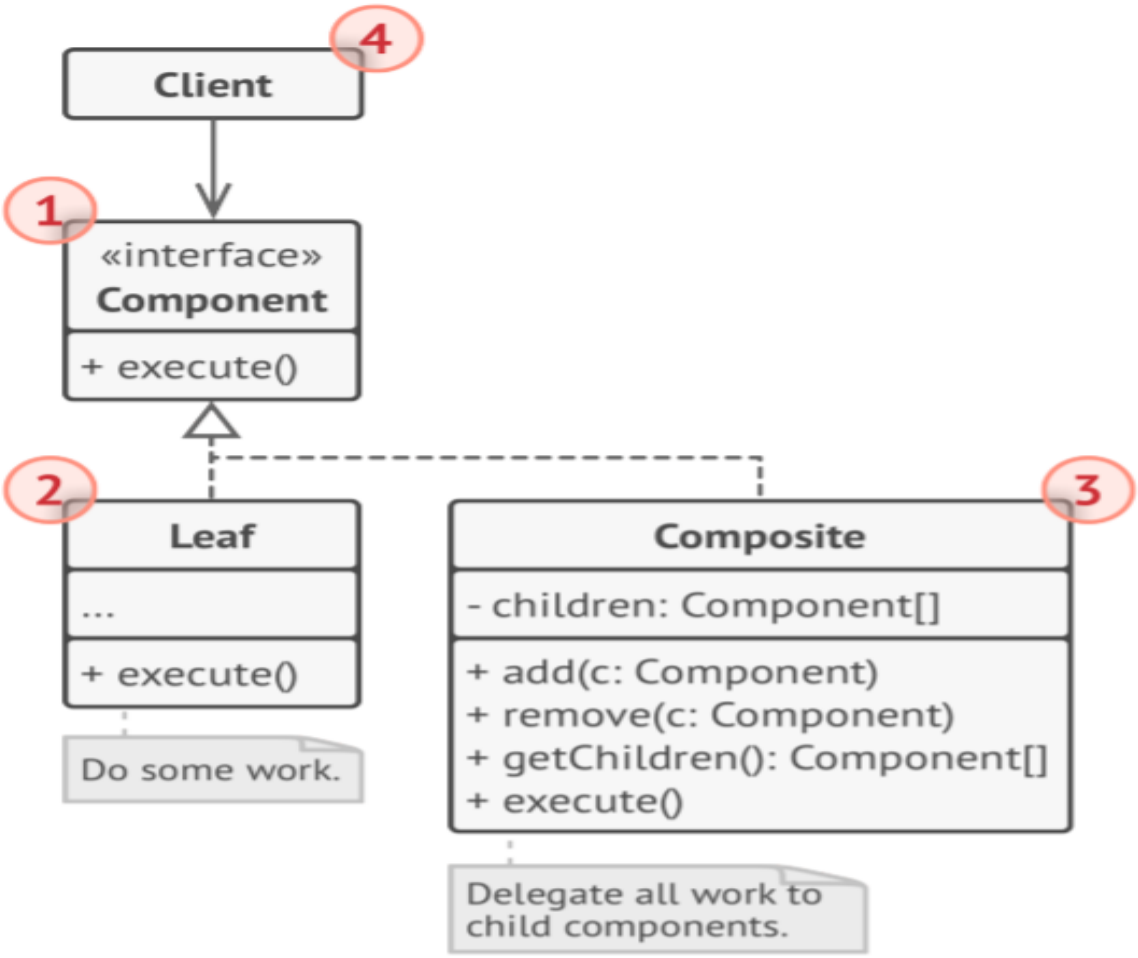
Composite Pattern Implementation Lab Task 1

Introduction & Concept

In this lab task we will learn how to implement the composite pattern using C#.Net

- Compose objects into tree structures to represent whole-part hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
- Composite is an entity made up of several elements.
- The composite pattern is a partitioning design pattern, it describes that the group of objects are treated same way as a single instance of the same type of object.
- The following is the structure of composite pattern.




Aamir Shabbir Pare

Implementation

1. Create a console application in visual studio and name it CompositeApp.
2. Create a folder and name it BasicCompositeImplementation
3. Create ***IComponent.CS*** file in this folder and write the following code in it:

```
/// <summary>
/// Composite Interface
/// </summary>
public interface IComponent
{
    void ShowInformation(int indent=0);
}
```

The above created interface is an abstraction that declares a single method ShowInformation(). It will be used by the client.

In the next step we are going to create a leaf class and implement the IComponent interface. The leaf class in composite structure is a participant that can be a single instance of object.

4. Create **Leaf.cs** file in this folder and write the following code in it:

```
/// <summary>
/// Leaf Class (A single instance in Composite)
/// </summary>
public class Leaf : IComponent
{
    public string Name { get; set; }
    public void ShowInformation(int indent)
    {
     Console.WriteLine($"{ "-".PadLeft(indent)} Leaf : {Name}");
    }
}
```

Next, we are going to create the composite class, it also is known as container that contains a collection of object instances.

5. Create **Composite.cs** file in this folder and write the following code in it:

```
/// <summary>
/// Composite implementing the IComponent
/// </summary>
public class Composite : IComponent
{
    public List<IComponent> Leaves { get; set; }
    public string Name { get; set; }
    public void ShowInformation(int indent)
    {
        Console.WriteLine($"{ "+".PadLeft(indent)} Composite :
                {Name}");
        foreach (var leaf in Leaves)
        {
            leaf.ShowInformation(indent+3);
        }
    }
}
```

The composite class is similar as leaf but in addition, it contains a collection to hold the list of object instances. The leaf and the composite implement the same interface which is ***IComponent*** that declares only a single method ***ShowInformation(int indent).*** Notice the logic of this method in both leaf and composite.

In the leaf class the ***ShowInformation(int indent)*** there is no foreach loop because the leaf doesn't contain the collection but it's a single instance object. But in the composite class ***ShowInformation(int indent)*** there is a foreach loop because it contains the collection.

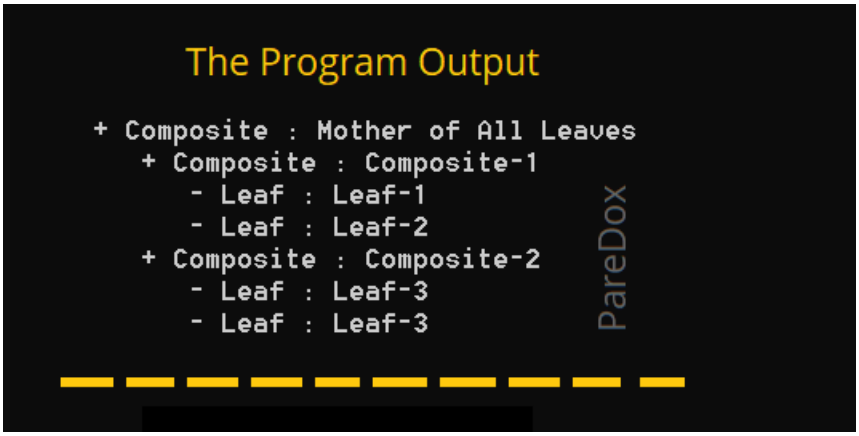
Here we have accomplished the composite design pattern implementation task.

Next, we are going to create a client that will demonstrate how to use the composite pattern implementation.

6. Create ***Client.cs*** file in this folder and write the following code in it:

```
/// <summary>
/// Using Composite in a client
/// </summary>
public static class Client
{
    public static void Main(string[] args)
    {
        IComponent leaf1 = new Leaf { Name = "Leaf-1" };
        IComponent leaf2 = new Leaf { Name = "Leaf-2" };
        IComponent leaf3 = new Leaf { Name = "Leaf-3" };
        IComponent leaf4 = new Leaf { Name = "Leaf-3" };

        IComponent composite1 = new Composite
        {
            Name = "Composite-1",
            Leaves = new List<IComponent> { leaf1, leaf2 }
        };
        IComponent composite2 = new Composite
        {
            Name = "Composite-2",
            Leaves = new List<IComponent> { leaf3, leaf4 }
        };

        IComponent whole = new Composite
        {
            Name = "Mother of All Leaves",
            Leaves = new List<IComponent> { composite1, composite2}
        };

        whole.ShowInformation();

        Console.ReadKey();
    }

}
```

Program Output



Summary

In this lab task we have learned how to implement the composite design pattern in a basic scenario. The purpose of the Composite Pattern is to create tree structures to represent whole-part hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly as is show above in the demo.

Composite Pattern Implementation Lab Task 2

In this lab task we are going to solve a problem similar in context as demonstrated in Lab Task 1 but a bit complex. This example is comprehensive to solve the real problem.

The Problem

Let's imagine that we have functionality in which we represent the organizational employee hierarchal behavior. The employee in the organization structure works in a section or department under some officer. Similarly, the officer replies to and works under upper level manager and the upper level management replies to CEO or big boss.

As we know that composite design pattern is a best candidate in hierarchal representation problems. The employee hierarchal management representation problem can be resolved by implanting the composite design pattern.

Application Code

Follow with me to create this application to gain more knowledge about the composite design pattern.
Create a new console project in Visual Studio or use the solution we created in above lab task. Then create a new folder on the root, name it CompositePattern.
Create the following class files in this folder.

1. Component – *IEmployee.cs*

```
//Component
public interface IEmployee
{
    void DisplayEmployeeDetails(int indent=1);
}
```

2. Leaf : *Employee.cs*

```
//Leaf
public class Employee : IEmployee
{
    public string FullName { get; set; }

    public string Department { get; set; }

    public void DisplayEmployeeDetails(int indent)
    {
        Console.WriteLine($"{"-".PadLeft(indent)}
          Name : {FullName}, Department : {Department}");
    }
}
```

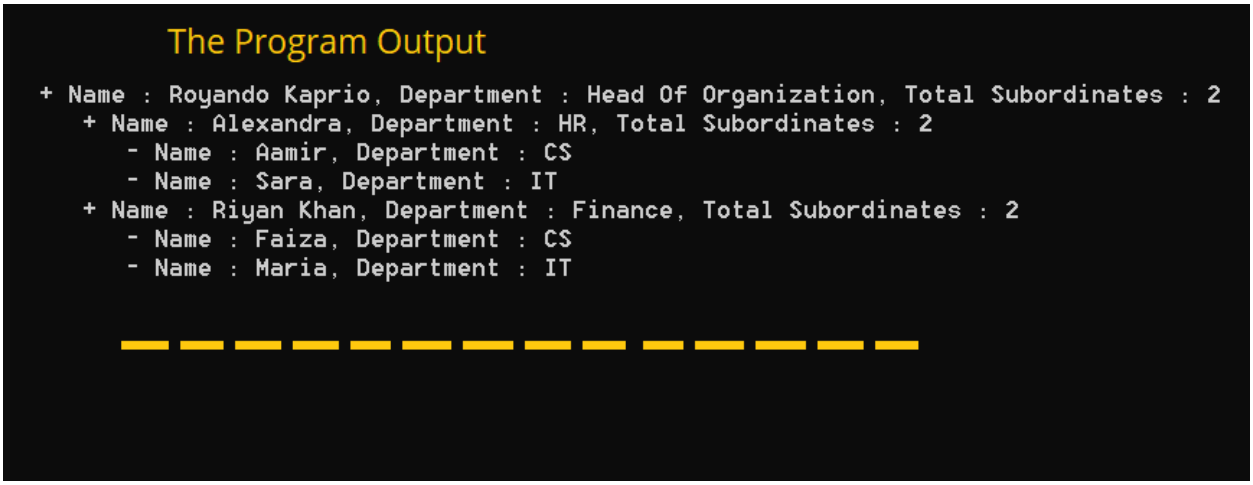
3. Composite or Conatiner : *Manager.cs*

```
//Coposite or Container
public class Manager : IEmployee
{
    public List<IEmployee> Subordinates { get; set; }

    public string FullName { get; set; }

    public string Department { get; set; }

    public void DisplayEmployeeDetails(int indent)
    {
        Console.WriteLine($"{"+".PadLeft(indent)}
          Name : {FullName}, Department : {Department},
         Total Subordinates : {Subordinates.Count}");

        foreach (var so in Subordinates)
        {
            so.DisplayEmployeeDetails(indent + 3);
        }
    }
}
```

4. Client : *Client.cs*

```
//Client
public static class Client
{
    public static void Main(string[] args)
    {
        //Leaves
        IEmployee emp1 = new Employee
        {
            FullName = "Aamir", Department = "CS"
        };
        IEmployee emp2 = new Employee
        {
            FullName = "Faiza", Department = "CS"
        };
        IEmployee emp3 = new Employee
        {
            FullName = "Sara", Department = "IT"
        };
        IEmployee emp4 = new Employee
        {
            FullName = "Maria", Department = "IT"
        };

        //Composties
        IEmployee emp5 = new Manager
        {
            FullName = "Alexandra", Department = "HR",
            Subordinates = new List<IEmployee> { emp1, emp3}
        };
        IEmployee emp6 = new Manager
        {
            FullName = "Riyan Khan", Department = "Finance",
            Subordinates = new List<IEmployee> { emp2, emp4 }
        };

        IEmployee bigBoss = new Manager
        {
            FullName = "Royando Kaprio",
            Department = "Head Of Organization",
            Subordinates = new List<IEmployee> { emp5, emp6 }
        };
        bigBoss.DisplayEmployeeDetails();
        Console.ReadKey();
```

Program Output



Summary

In this lab task we have learned how to implement the composite design pattern in a more realistic problem scenario that is organizational employee hierarchal representation. The purpose of the Composite Pattern is to create tree structures to represent whole-part hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly as is show above in the more realistic employee hierarchal representation demo.

Goodbye, wish you all the best and see you in next lab task!