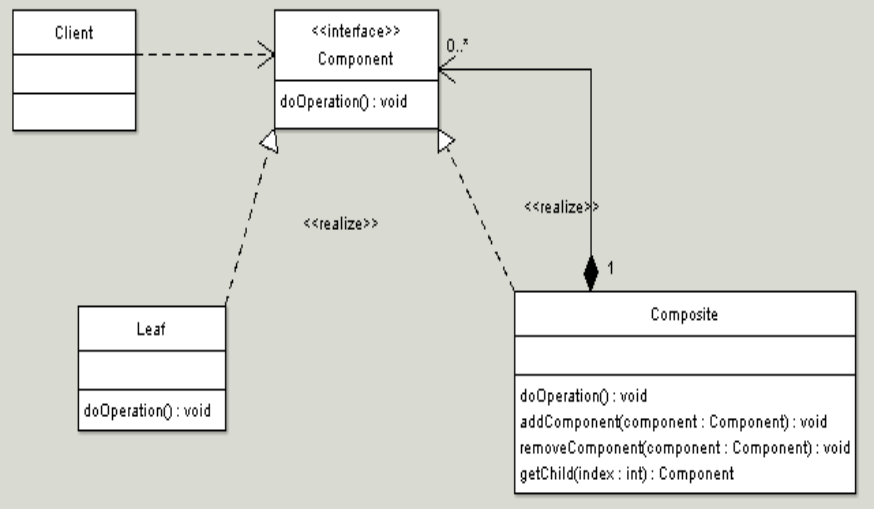
Introduction

In this assignment I had to create a program that composes objects into hierarchies. I did this by making a class-role guide for uses with a hypothetical game. It groups the classes together into different roles and gives the power of the class to help hypothetical players choose the proper character. The UML diagram to the right shows structure I followed. I kept all the class names the same so as to help myself keep everything organized.



The Code

```
public interface Component
    {
        int getStrength();
        string ToString();

    }

public class Leaf : Component
    {

        private string className;
        private int strength;


        public Leaf(string theClass, int theStrength)
        {
            strength = theStrength;
            className = theClass;

        }

        public int getStrength()
        {

            return strength;
        }


        public override string ToString()
        {
            return className;
        }

     }
public class Composite : Component
    {
        private string className;

        public int power;
        public List<Component> classes = new List<Component>();
```

The component class contains two methods that are used to hold information on the Character that the user chooses. This is the same as the doOperation from the UML.

The Leaf class is used to set up the individual pieces of the pattern, the parts that aren't part of a bigger group. It contains a constructor to use later.

```
        public Composite(string theClass, Component R1, Component R2)
        {

            className = theClass;
            addClass(R1);
            addClass(R2);

        }
        public Composite(string theClass, Component R1, Component R2, Component R3)
        {

            className = theClass;
            addClass(R1);
            addClass(R2);
            addClass(R3);

        }

        public void addClass(Component c)
        {
            classes.Add(c);
        }

        public void removeClass(Component c)
        {
            classes.Remove(c);
        }

        public Component getClasses(int index)
        {
            return classes[index];
        }

        public int getStrength()
        {

            return power;
        }

        public override string ToString()
        {
            return className;
        }


    }
```

The Composite class also has a constructor, two in fact. The two are needed to compensate for the different amount of info in the groups. I also have the addComponenet, removeComponent, and getChild functions. Though only add is used.

```
public partial class Form1 : Form
    {
        const string Indent = "    ";
         void PrintType(Composite c)
        {
            txtResponse.Text +=   c.ToString() + Environment.NewLine;
            foreach (Component co in c.classes)
            {
                txtResponse.Text += Indent;
                PrintClass((Leaf) co);

            }

        }
        void PrintClass(Leaf l)
        {
            txtResponse.Text += l.ToString() +" " + l.getStrength()+ Environment.NewLine;
        }
```

```
static List<Component> ClassList = new List<Component>()
{
    new Leaf("Fighter" , 20),
    new Leaf("Mage" , 6),
    new Leaf("Rogue" , 14),
    new Leaf("Cleric" , 16),
    new Leaf("Paladin" , 18),
    new Leaf("Ranger" , 16),
};

static List<Component> ClassType = new List<Component>()
{
    new Composite ("Tank" , ClassList[0], ClassList[4]),
    new Composite ("Magic" , ClassList[1], ClassList[3], ClassList[4]),
    new Composite ("Damage" , ClassList[0], ClassList[2], ClassList[5]),
    new Composite ("Healer" , ClassList[4], ClassList[3]),

};

public Form1()
{
    InitializeComponent();
}

private void btnTank_Click(object sender, EventArgs e)
{
    PrintType((Composite)ClassType[0]);

}

private void btnDamage_Click(object sender, EventArgs e)
{
    PrintType((Composite)ClassType[2]);
}

private void btnMagic_Click(object sender, EventArgs e)
{
    PrintType((Composite)ClassType[1]);
}

private void btnHealer_Click(object sender, EventArgs e)
{
    PrintType((Composite)ClassType[3]);
}

private void btnFighter_Click(object sender, EventArgs e)
{
    PrintClass((Leaf)ClassList[0]);
}

private void btnMage_Click(object sender, EventArgs e)
{
    PrintClass((Leaf)ClassList[1]);
}

private void btnRogue_Click(object sender, EventArgs e)
{
    PrintClass((Leaf)ClassList[2]);
}

private void btnCleric_Click(object sender, EventArgs e)
{
    PrintClass((Leaf)ClassList[3]);
}

private void btnPaladin_Click(object sender, EventArgs e)
{
    PrintClass((Leaf)ClassList[4]);
```

The form contains lists of the different classes and role types associated with this hypothetical game. It also has a PrintType function and PrintClass function to make setting up the buttons easier.

```
        }

        private void btnRanger_Click(object sender, EventArgs e)
        {
            PrintClass((Leaf)ClassList[5]);
        }
    }
```

Conclusion

I had a lot of trouble with this pattern. I feel like it is mostly due to lack of experience with anything similar. It also took a while for the concept to click with me. I actually didn't understand the pattern until I saw presentations of it. After that, I was able to piece together this program. I didn't have fun while coding it, but it felt great when it was finally working. In fact, I felt like doing this with my computer most of the time, https://www.youtube.com/watch?v=HX0DDigasd0 .