Design Patterns
Prof. Retterer
Composite Pattern

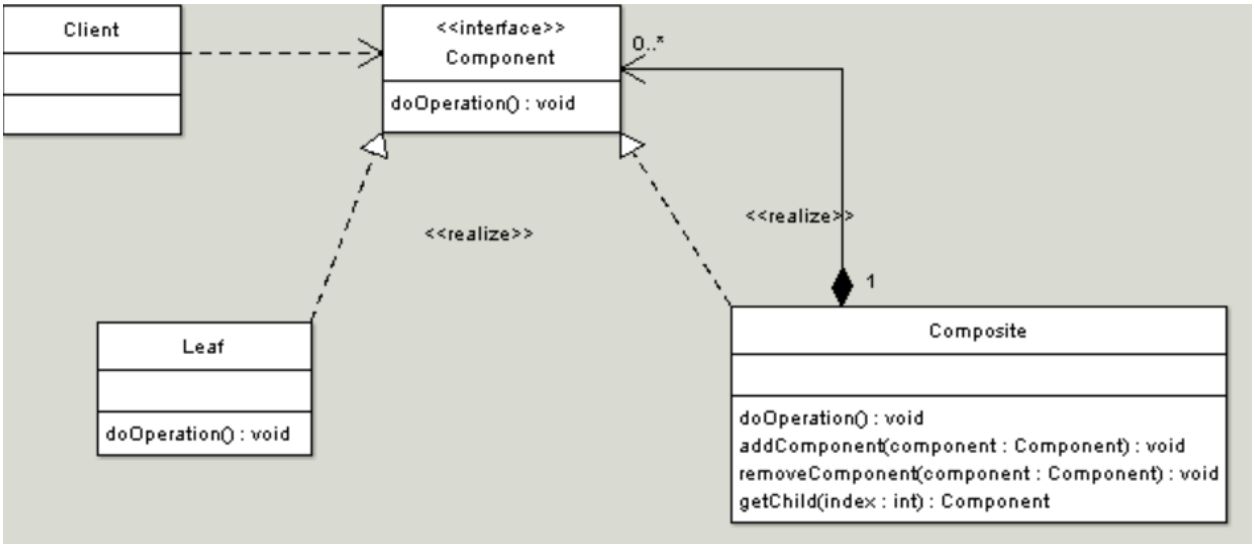
Jeremy Martin
10/17/16

Introduction

For this assignment we were tasked with making a program that displayed the Composite pattern. For my program, I decided to make a program that would describe the status effects of an ability or multiple statuses if it is a combo ability.

Composite UML

To the right you will find a picture of the UML diagram displaying the Composite pattern.



Below I will show the code that is used for each class and how they come together in the form.

The first piece of code shows the Component class.

```
public abstract class Component
    {
        public abstract string getCondition();
    }
```

The next piece deals with the Leaf class.

```
public class Leaf : Component
    {
        private string Ability;
        private string Condition;

        public Leaf(string ability,string condition)
        {
            Ability = ability;
            Condition = condition;
        }

        public Leaf()
        {

        }
```

```
        public override string getCondition()
        {
            return Condition;
        }

        public override string ToString()
        {
            return Ability;
        }
    }
```

The next piece deals with the Composite class.

```
public class Composite : Component
    {
        private string Ability;
        private List<Component> conditions = new List<Component>();


        public Composite(string ability,Component a1,Component a2)
        {
            Ability = ability;
            addComponent(a1);
            addComponent(a2);
        }

        public override string getCondition()
        {
            string totalConditions = "";

            foreach(Leaf l in conditions)
            {
                totalConditions += " "+l.getCondition();
            }

            return totalConditions;
        }

        public void addComponent(Component com)
        {
            conditions.Add(com);
        }

        public void removeComponent(Component com)
        {
            conditions.Remove(com);
        }

        public Component getChild(int index)
        {
            return conditions[index];
        }

        public override string ToString()
        {
            return Ability;
        }
    }
```

Jeremy Martin
10/17/16

The last piece shows all of it coming together in Form1.

```
public partial class Form1 : Form
    {
        static List<Component> abilityList = new List<Component>()
        {
            new Leaf("Stomp","Stun"),
            new Leaf("Charge","Daze"),
            new Leaf("Interrupt","Silence"),
            new Leaf("Blaze","Burn"),
            new Leaf("Freeze","Chill"),
            new Leaf("Shock","Paralyze")
        };

        static List<Component> comboList = new List<Component>()
        {
            new Composite("Leap",abilityList[1],abilityList[0]),
            new Composite("Tremor",abilityList[0],abilityList[0]),
            new Composite("Flash Freeze",abilityList[4],abilityList[4]),
            new Composite("Frostbite",abilityList[4],abilityList[3]),
            new Composite("Lightningbolt",abilityList[5],abilityList[2])
        };

        public Form1()
        {
            InitializeComponent();

            m_rbAbility.Checked = true;
            populate(m_cbList, abilityList);
        }

        private void populate(ComboBox cb, List<Component> l)
        {
            cb.Items.Clear();

            foreach(Component c in l)
            {
                cb.Items.Add(c);
            }

            cb.SelectedItem = l[0];
        }

        private void m_rbCombo_CheckedChanged(object sender, EventArgs e)
        {
            populate(m_cbList, comboList);
        }

        private void m_rbAbility_CheckedChanged(object sender, EventArgs e)
        {
            populate(m_cbList, abilityList);
        }

        private void m_btnAbility_Click(object sender, EventArgs e)
        {
            Component ability = (Component)m_cbList.SelectedItem;
            string condition = "";
```

Jeremy Martin

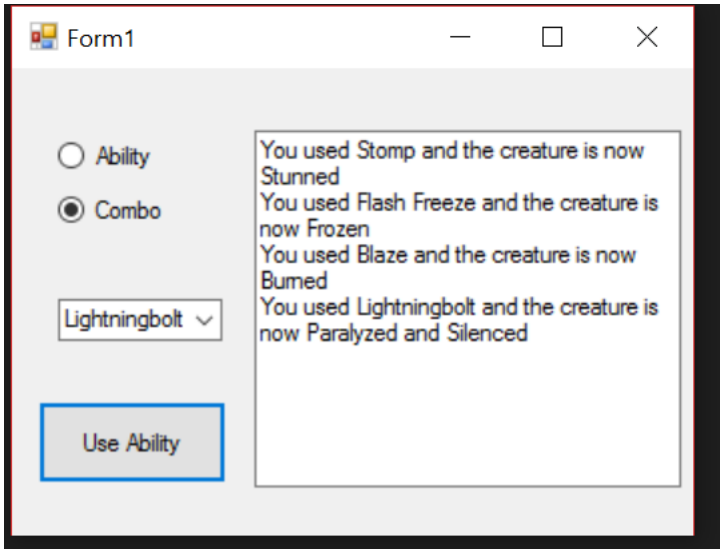

```
            if (ability.getCondition()=="Stun")
            {
                condition = "Stunned";
            }
            if (ability.getCondition() == "Daze")
            {
                condition = "Dazed";
            }
            if (ability.getCondition() == "Silence")
            {
                condition = "Silenced";
            }
            if (ability.getCondition() == "Burn")
            {
                condition = "Burned";
            }
            if (ability.getCondition() == "Chill")
            {
                condition = "Chilled";
            }
            if (ability.getCondition() == "Paralyze")
            {
                condition = "Paralyzed";
            }
            if(ability.getCondition() == " Daze Stun")
            {
                condition = "Dazed and Stunned";
            }
            if (ability.getCondition() == " Stun Stun")
            {
                condition = "Unconscious";
            }
            if (ability.getCondition() == " Chill Chill")
            {
                condition = "Frozen";
            }
            if (ability.getCondition() == " Chill Burn")
            {
                condition = "Chilled and Burned";
            }
            if (ability.getCondition() == " Paralyze Silence")
            {
                condition = "Paralyzed and Silenced";
            }

            m_tbLog.Text += "You used " + ability + " and the creature is now " +
condition + Environment.NewLine;
        }
    }
```

Design Patterns
Prof. Retterer
Composite Pattern

Jeremy Martin
10/17/16

And here is a picture of the code working.



Conclusion

Overall, this was an interesting pattern to work on. I couldn't get this to work without doing a bit of hard coding in order to get the condition to display right. Since all of the conditions were strings it became hard to make sure that things were equal without using the hard coding. This will be a helpful pattern if I ever need to make a program that deals with folders.