Tori Lentz

Dave Retterer

Design Patterns

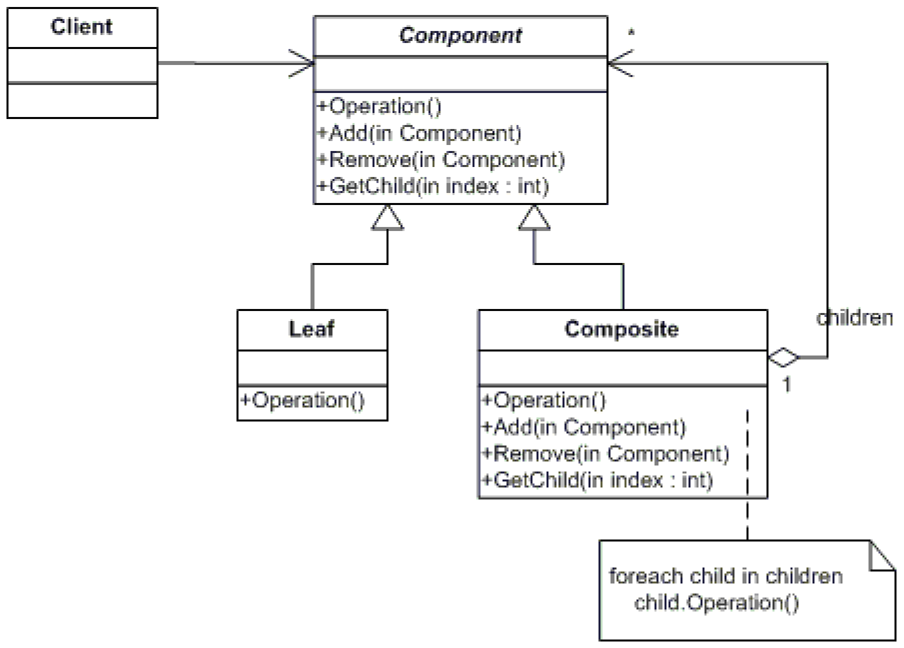
October 19, 2016

Composite Pattern

Introduction

The goal of this assignment was to design and implement the Composite Pattern. This pattern uses a tree structure which it can then manipulate. A tree structure of date contains both branches and leaves which the pattern then treats as uniform parts. Branches are containers that can have multiple leaves inside of them. Since both branches and leaves can contain the same functions, one usually creates an interface that contains the methods for both parts.

The UML Diagram



Component	The abstraction for leaves and composites.
Composite	Stores child components in addition to implementing methods defined by the component interface.
Leaf	Objects that have no children.
Form	The client manipulates objects in the hierarchy using the component interface.
http://www.dofactory.com/net/composite-design-pattern	

The UML diagram above shows the needed components for this pattern. The interface or Component contains the methods for the program and is an abstract class. These methods include a do something operation, an add operation, a remove operation and a get children operation. Both the leaf and Composite classes derive from the component class. Within the composite class, all of the methods are defined and within the leaf class, the operations are defined.

Code and Description

Component Class

```
namespace Composite_Pattern
{
    public abstract class Component
    {
        protected string _name;
        // Constructor
        public Component(string name)
        {
            this._name = name;
        }
        public abstract void Add(Component d);

        public abstract void Remove(Component d);

        public abstract void Display(int indent);

    }
}
```

Displayed to the left is my component class. It is an abstract class because it is only meant to be used as a base class for other classes, therefor it is left as an incomplete implementation. Within this class I defined my Add, Remove and Display methods, as well as the constructor for component.

Composite Class

```
namespace Composite_Pattern
{
    public class Composite : Component
    {
        private List<Component> elements = new List<Component>();

        // Constructor

        public Composite(string name)
            : base(name)
        {

        }

        public override void Add(Component d)
        {
            elements.Add(d);
        }

        public override void Remove(Component d)
        {
            elements.Remove(d);
        }

        public override void Display(int indent)
        {
            Console.WriteLine(new String('-', indent) + "+ " + _name);
```

Here is my composite class which derives from my component class. Within this class, I define my list of variables and my override methods for Add, Remove and Display. The constructor is also here in this class as well and the keyword base is used to access the members of the base class, Component, from within this derived class, Composite. The Add method will add the designated branches, the Remove method will remove the branched and the display method will display the branch names.

```
            // Display each child element on this node
            foreach (Component d in elements)
            {
                d.Display(indent + 2);
            }
        }
    }
}
```

Within this foreach loop, each element within my branch is added to the list.

Leaf Class

```
namespace Composite_Pattern
{
    public class Leaf : Component
    {

        // Constructor
        public Leaf(string name)
          : base(name)
        {

        }

        public override void Add(Component c)
        {
            Console.WriteLine("Cannot add to a Leaf");
        }

        public override void Remove(Component c)
        {
            Console.WriteLine("Cannot remove from a
Leaf");
        }

        public override void Display(int indent)
        {
            string itemName = "";
            string itemPriceStr = "";
            decimal itemPrice = 0;
            char[] delimiterChars = {'|'};
            string[] words = _name.Split(delimiterChars);
            itemName = words[0].ToString();
            itemPriceStr = words[2].ToString();
            itemPrice = Convert.ToDecimal(itemPriceStr);
            Console.WriteLine(new String('-', indent) + " " + _name);
            Global.gv_TotalPrice = Global.gv_TotalPrice + itemPrice;
            Global.gv_DisplayData = Global.gv_DisplayData + "   - " + itemName + " -
Price: $" + itemPriceStr + "\r\n";
        }
    }
}
```

This is my leaf class which commands all of the parts under my branch components. Again,all of the methods are overridden here so that this class can perform the necessary operations with the defined methods.

It is within this class specifically, the Display method, that I am able to concatenate the parts together as well as get the item prices and total. I used two global variables with one defined in my global class in order to carry my values from class to

Global Class

```
namespace Composite_Pattern
```

```
{
    public static class Global
    {
        public static string gv_DisplayData = "";
        public static decimal gv_TotalPrice = 0;
    }
}
```

This is my global class which contains my global variables.

Form

```
namespace Composite_Pattern
{

    public partial class Form1 : Form
    {

        Composite ShoppingCart = new Composite("Shopping Cart");
        Composite WatercolorKit = new Composite("Watercolor Kit");
        Composite AcrylicKit = new Composite("Acrylic Kit");
        Composite MixedMediaKit = new Composite("Mixed Media Kit");

        public Form1()
        {
            InitializeComponent();

            WatercolorKit.Add(new Leaf("Watercolor Paint Set||14.53"));
            WatercolorKit.Add(new Leaf("Pentel Aquash Set||13.58"));
            WatercolorKit.Add(new Leaf("Canson Watercolor Paper Pad||9.49"));

            AcrylicKit.Add(new Leaf("Liquitex Acrylic Paint||35.84"));
            AcrylicKit.Add(new Leaf("Paint Brush Set||24.17"));
            AcrylicKit.Add(new Leaf("Canvas Pack||17.05"));
            AcrylicKit.Add(new Leaf("Paint Palette||9.65"));

            MixedMediaKit.Add(new Leaf("Watercolor Paint Set||14.53"));
            MixedMediaKit.Add(new Leaf("Pentel Aquash Set||13.58"));
            MixedMediaKit.Add(new Leaf("Canson Watercolor Paper Pad||9.49"));
            MixedMediaKit.Add(new Leaf("Liquitex Acrylic Paint||35.84"));
            MixedMediaKit.Add(new Leaf("Paint Brush Set||24.17"));
            MixedMediaKit.Add(new Leaf("Canvas Pack||17.05"));
            MixedMediaKit.Add(new Leaf("Paint Palette||9.65"));

        }

        private void updateShoppingCartDisplay()
        {
            tb_DisplayData.Text = "";
            ShoppingCart.Display(0);
            tb_DisplayData.Text = Global.gv_DisplayData +
"\r\n" + " Total Price: " + Global.gv_TotalPrice.ToString();
        }

        private void btn_Watercolor_Click(object sender, EventArgs e)
        {
            Global.gv_DisplayData = Global.gv_DisplayData + "Watercolor Kit \r\n";
            ShoppingCart.Add(WatercolorKit);
```

My client for this pattern is my form. Within it, I defined my branches as WatercolorKit, AcrylicKit, and MixedMediaKit. The overall tree is my ShoppingCart.

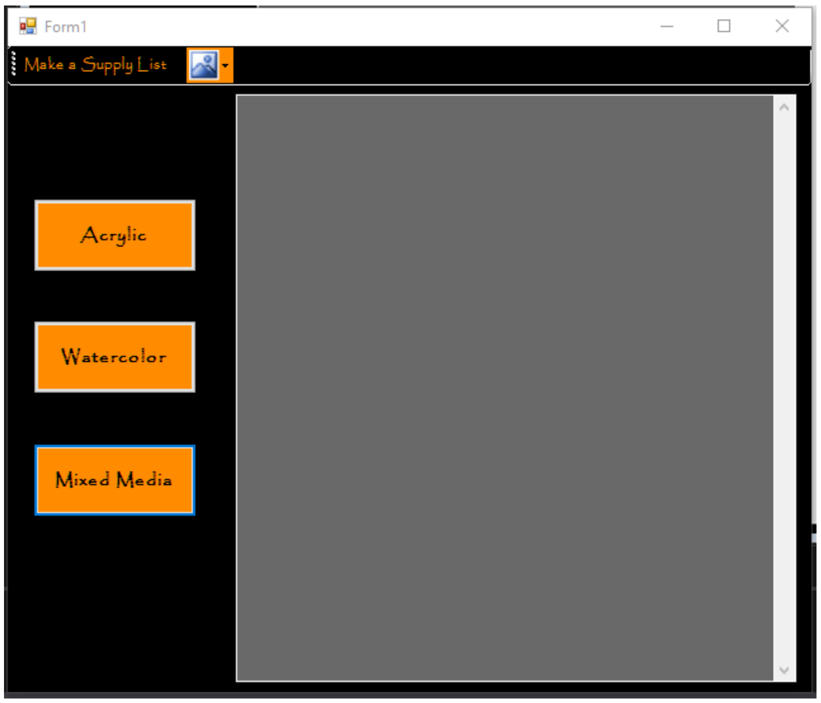
Above, all of my leaves are defined.

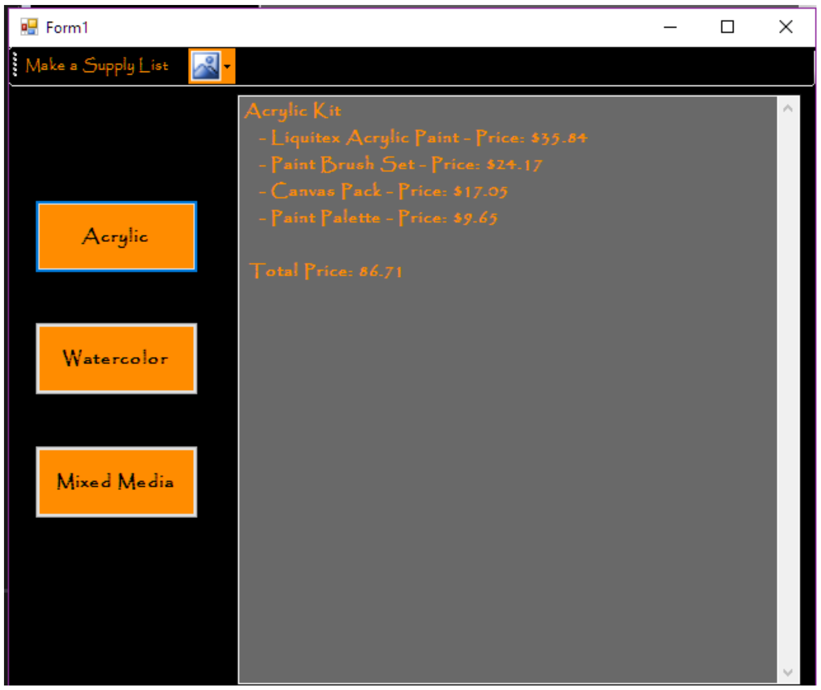
This is my update method that will add the clicked kits to the cart and add their prices to the total.

```
            updateShoppingCartDisplay();
        }

        private void btn_Acrylic_Click(object sender, EventArgs e)
        {
            Global.gv_DisplayData = Global.gv_DisplayData + "Acrylic Kit \r\n";
            ShoppingCart.Add(AcrylicKit);
            updateShoppingCartDisplay();
        }

        private void btn_MixedMedia_Click(object sender, EventArgs e)
        {
            Global.gv_DisplayData = Global.gv_DisplayData + "Mixed Media Kit \r\n";
            ShoppingCart.Add(MixedMediaKit);
            updateShoppingCartDisplay();
        }


        private void acrylicToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Global.gv_DisplayData = Global.gv_DisplayData + "Acrylic Kit \r\n";
            ShoppingCart.Add(AcrylicKit);
            updateShoppingCartDisplay();
        }

        private void waterColorToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Global.gv_DisplayData = Global.gv_DisplayData + "Watercolor Kit \r\n";
            ShoppingCart.Add(WatercolorKit);
            updateShoppingCartDisplay();
        }

        private void mixedmediaToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Global.gv_DisplayData = Global.gv_DisplayData + "Mixed Media Kit \r\n";
            ShoppingCart.Add(MixedMediaKit);
            updateShoppingCartDisplay();
        }

    }
}
```

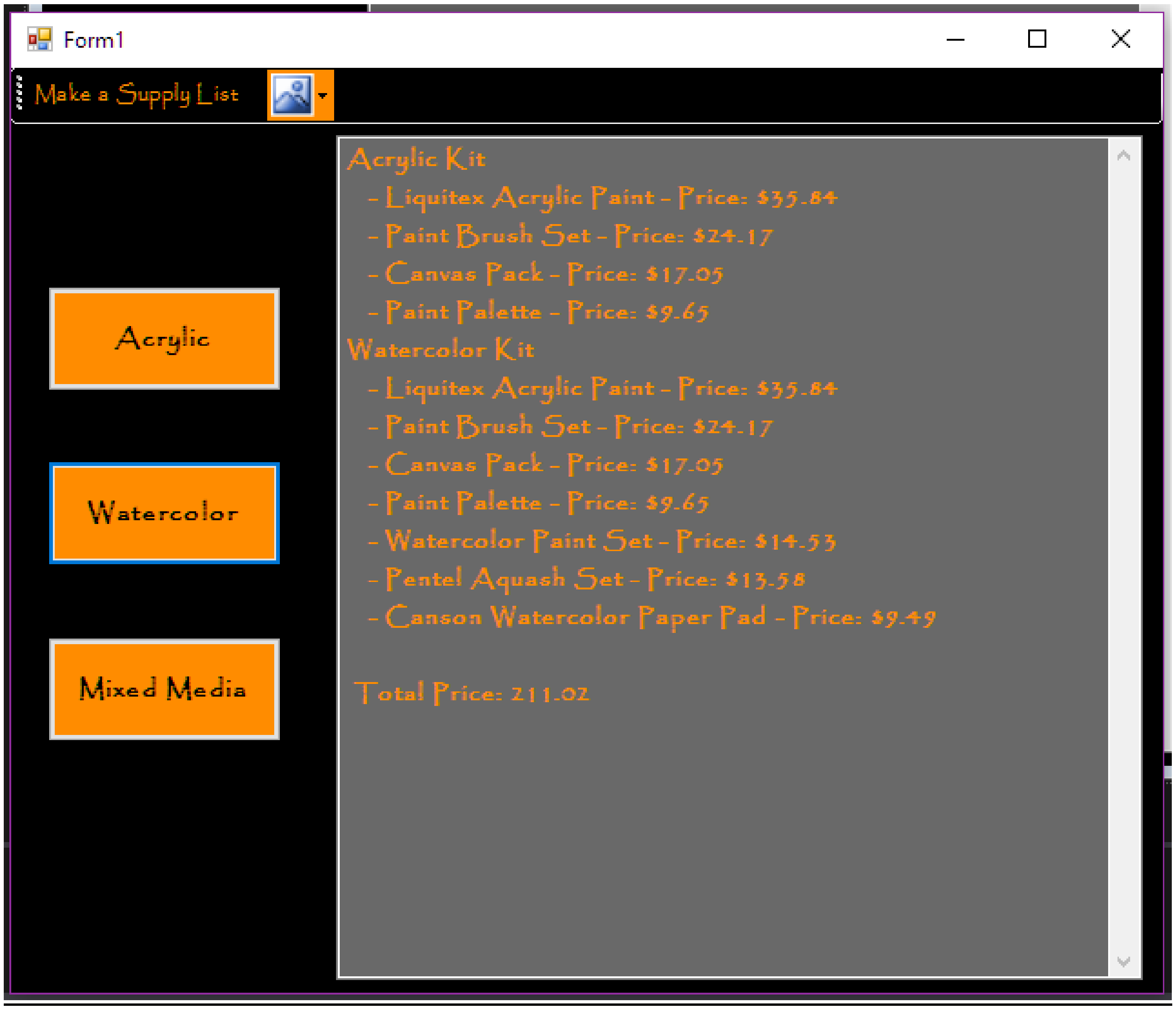
Within each of my buttons and my tool strip menu items I have the names of the specific branches, an add call and an update call.

Screen Shots





```
Acrylic Kit
  - Liquitex Acrylic Paint - Price: $35.84
  - Paint Brush Set - Price: $24.17
  - Canvas Pack - Price: $17.05
  - Paint Palette - Price: $9.65

 Total Price: 86.71
```



```
Acrylic Kit
  - Liquitex Acrylic Paint - Price: $35.84
  - Paint Brush Set - Price: $24.17
  - Canvas Pack - Price: $17.05
  - Paint Palette - Price: $9.65
Watercolor Kit
  - Liquitex Acrylic Paint - Price: $35.84
  - Paint Brush Set - Price: $24.17
  - Canvas Pack - Price: $17.05
  - Paint Palette - Price: $9.65
  - Watercolor Paint Set - Price: $14.53
  - Pentel Aquash Set - Price: $13.58
  - Canson Watercolor Paper Pad - Price: $9.49


 Total Price: 211.02
```

<u>Observations</u>

Overall, I thought this program went very well. It has taken me a while but the more I work with C#, the more things make sense. Up until now I have been really struggling with trying to understand how the classes go together, how they are implemented and how they call upon each other. This program in particular really helped me to understand why things are the way they are and how to fix things. Once I was able to find some sample code that I could reference on the dofactory website, the writing of this code came a lot easier and I was able to expand upon what I learned from the sample. I really enjoyed writing this program and I am very happy with the final result. The only thing I wish I could add to this would be a clear button that will remove the variables from the list. I know my program has the ability to do this but, I cannot figure out how to get my form to call the method.