Composite Pattern Assignment

Design Patterns

Jacob Hartman

16 October 2016

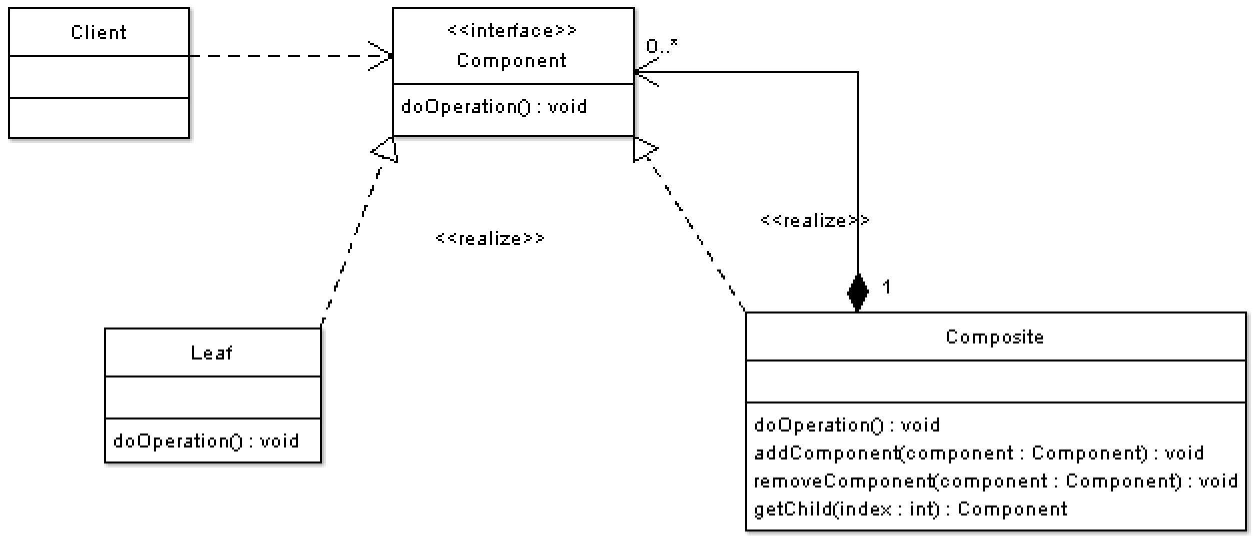
Introduction

This assignment required the creation of a program that demonstrates use of the composite pattern. The composite pattern is used to build a tree structure and handle access to various leaves and containers. The containers are known as composites.

The application I wrote for this assignment is a designer for organic compound structures. It allows the user to add various components to a compound and view which components it contains. It also generates a name for the compound based on its structure.

UML Diagram

image taken from http://www.oodesign.com/composite-pattern.html



Above is the UML diagram for the composite pattern. It shows the need for three classes apart from the client. First is the Component class, which is an interface with the doOperation method. The Leaf class implements the Component interface and its doOperation method. The Composite class also implements the interface and its method, also containing three new methods. addComponent adds a component to the composite. removeComponent removes a component from the interface. getChild returns the component at the specified index.

I wrote a Component class with getName and getSubName as doOperation methods. I wrote a Methane class and a Benzene class as Leaf classes. I also wrote a Composite class. It has the addComponent, removeComponent, and getChild methods.

<u>Narrative</u>

I began by writing the Component interface as an abstract class.

```
public abstract class Component // this is the component class
{
    public abstract string getName();
    public abstract string getSubName();
}
```

The getName and getSubName methods are both doOperation methods. Next, I wrote the Leaf classes.

```
public class Methane : Component // this is a leaf class
{
    public override string getName()
    {
        return "Methane";
    }

    public override string getSubName()
    {
        return "Methyl";
    }
}
```

The getName method returns the name of the independent component, and the getSubName method returns the name of the component as a subgroup of a larger compound.

```
public class Benzene : Component // this is a leaf class
{
    public override string getName()
    {
        return "Benzene";
    }

    public override string getSubName()
    {
        return "Phenyl";
    }
}
```

The Benzene class serves a similar purpose to the Methane class, but the names relate to benzene rather than methane. Next, I wrote the Composite class.

```
public class Composite : Component // this is the composite class
```

```
    {
        private List<Component> content = new List<Component>();
        private bool isCyclic = false;
```

Each composite contains a List called content, which contains all the child components of the composite. There is also a boolean value stating whether or not the compound is cyclically bonded.

```
        public override string getName()
        {
            int count = 0;
            string name = "";
            int i = 0;
            foreach (Component c in content)
            {
                i++;
                if (c.getName() == "Methane")
                {
                    count++;
                }
                else if (c.getName() == "Benzene")
                {
                    name += "" + i + "-" + c.getSubName() + "-";
                }
                else
                {
                    name += "" + i + "-" + c.getSubName() + "-";
                }
            }
            if (isCyclic == true)
            {
                name += "Cyclo";
            }
            switch (count)
            {
                case 0:
                    return "Benzene";
                case 1:
                    name += "Methane";
                    break;
                case 2:
                    name += "Ethane";
                    break;
                case 3:
                    name += "Propane";
                    break;
                case 4:
                    name += "Butane";
                    break;
                case 5:
                    name += "Pentane";
                    break;
                case 6:
                    name += "Hexane";
                    break;
                case 7:
                    name += "Heptane";
                    break;
                case 8:
                    name += "Octane";
                    break;
                case 9:
                    name += "Nonane";
                    break;
                default:
                    name += "error: unsupported structure";
                    break;
            }
            return name;
        }
```

The getName method returns the name of the compound based on its child components. It gives the compound a name based on the number of methyl groups, adding bits to account for any contained benzene rings as well.

```
        public override string getSubName()
        {
            int count = 0;
            string name = "";
            int i = -1;
            foreach (Component c in content)
            {
                i++;
                if (c.getName() == "Methane")
                {
                    count++;
                }
                else if (c.getName() == "Benzene")
                {
                    name += "" + i + "-" + c.getSubName() + "-";
                }
                else
                {
                    name += "" + i + "-" + c.getSubName() + "-";
                }
            }
            switch (count)
            {
                case 0:
                    return "Phenyl";
                case 1:
                    name += "Methyl";
                    break;
                case 2:
                    name += "Ethyl";
                    break;
                case 3:
                    name += "Propyl";
                    break;
                case 4:
                    name += "Butyl";
                    break;
                case 5:
                    name += "Pentyl";
                    break;
                case 6:
                    name += "Hexyl";
                    break;
                case 7:
                    name += "Heptyl";
                    break;
                case 8:
                    name += "Octyl";
                    break;
                case 9:
                    name += "Nonyl";
                    break;
                default:
                    name += "error: unsupported structure";
                    break;
            }
            return name;
        }
```

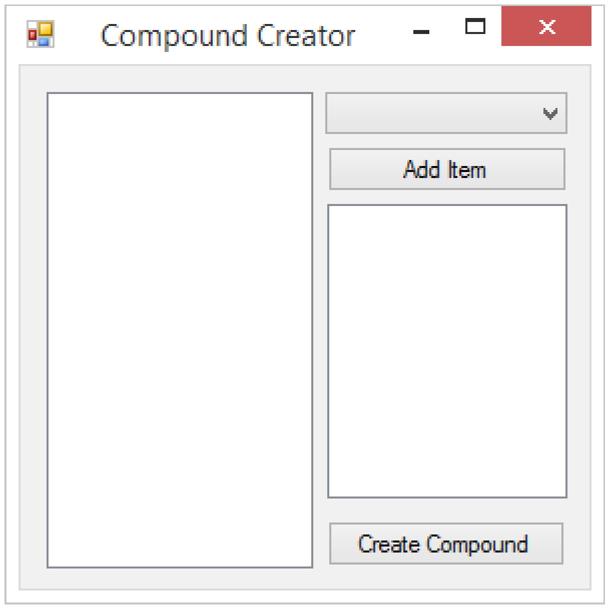
The getSubName method uses the same technique of iterating through the tree that the getName method does. However, it returns the name of the group as a part of a larger compound instead.

```
        public void setCyclic(bool a)
        {
            isCyclic = a;
        }

        public void addComponent(Component c)
        {
            content.Add(c);
        }

        public void removeComponent(Component c)
        {
            content.Remove(c);
        }

        public int getChildCount()
        {
            return content.Count;
        }

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

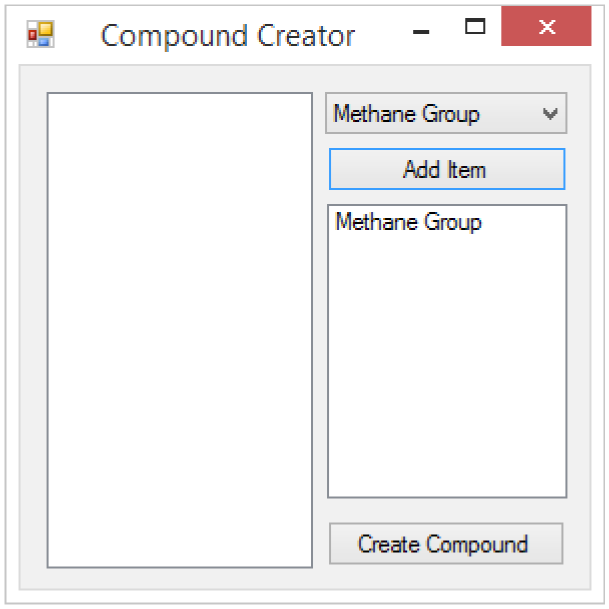
The setCyclic method allows the system to set whether the compound contains a cyclic bond. addComponent and removeComponent allow the adding and removing of leaves and other composites from the composite. getChildCount returns the number of children in the composite, used for iterating through the children. getChild allows access to the children of the composite.

```
public partial class Form1 : Form
    {
        public List<Component> compounds;
        public List<Component> formula;
        public bool isCyclic;
        public Form1()
        {
            InitializeComponent();
            compounds = new List<Component>();
            formula   = new List<Component>();
            isCyclic = false;
        }

        private void btn_addItem_Click(object sender, EventArgs e)
        {
            int a = cb_newItem.SelectedIndex;
            if (a != -1)
            {
                if (a == 0)
                {
                    formula.Add(new Methane());
                    lb_newCompound.Items.Add("Methane Group");
                }
                else if (a == 1)
                {
                    formula.Add(new Benzene());
                    lb_newCompound.Items.Add("Benzene Ring");
                }
                else if (a == 2)
                {
                    isCyclic = true;
                    lb_newCompound.Items.Add("Cyclic Bond");
                }
            }
        }

        private void btn_create_Click(object sender, EventArgs e)
        {
            int a = compounds.Count;
            Composite c = new Composite();
            foreach (Component b in formula)
            {
                c.addComponent(b);
            }
            c.setCyclic(isCyclic);
            compounds.Add(c);
            lb_newCompound.Items.Clear();
            Composite temp = (Composite)compounds[a];
            lb_compoundList.Items.Add(temp.getName());
            int d = temp.getChildCount();
            for (int i = 0; i < d; i++)
            {
                Component child = temp.getChild(i);
                lb_compoundList.Items.Add("    " + child.getName());
            }
            formula.Clear();
            isCyclic = false;
        }
    }
```

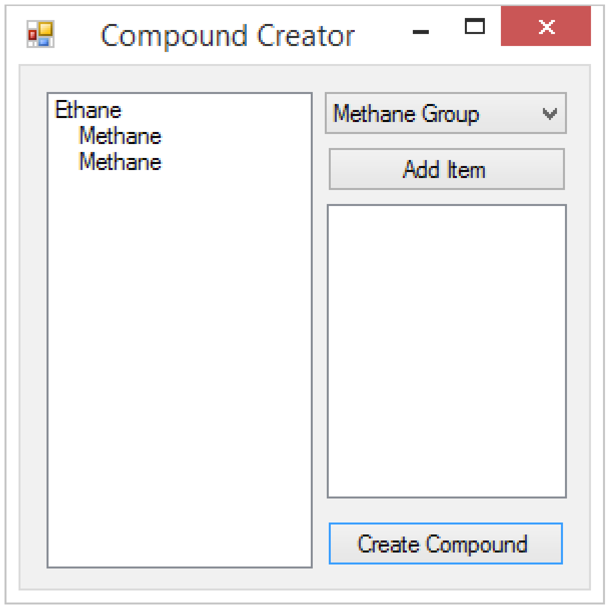
The Form1 class is the client. It contains two component lists. The compounds list contains all the final composites for each compound. The formula list contains all the components for the compound currently being generated. The class also contains a boolean tracking whether the compound being created has a cyclic bond. When the add item button is clicked, the item currently selected is added to the formula list, and its name is added to the ListBox displaying the current compound. When the create compound button is clicked, the formula list's components are transferred to a new composite, which is then added to the compounds list. The formula list and the ListBox showing the current compound are then cleared, and the cyclic boolean is reset to the default value. The new compound's name is then added to the ListBox displaying compounds, with all children being displayed. Below are some screenshots of the program in use.



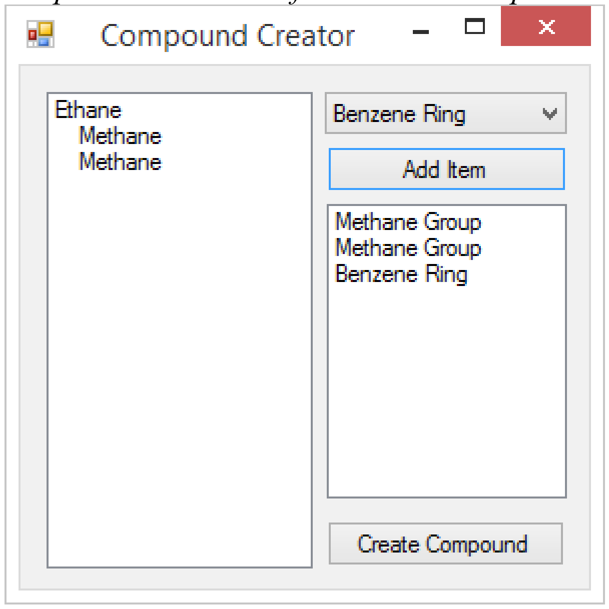
The initial view of the program.



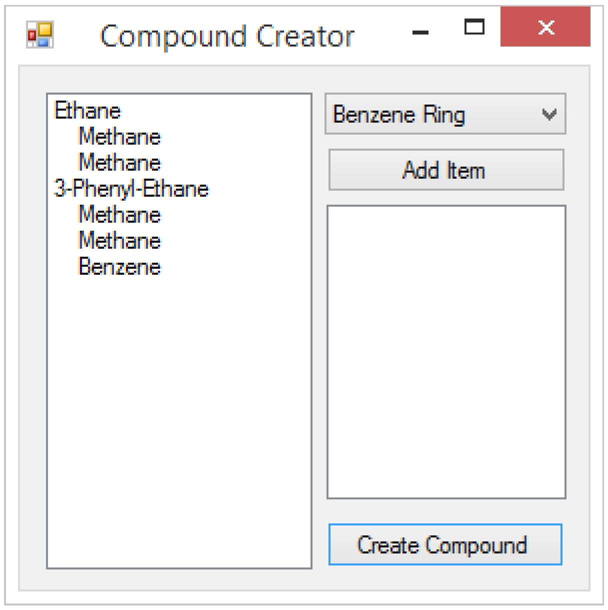
Selecting a component from the dropdown menu and hitting add item creates the component in the list of the current compound.



On hitting create compound, the compound is created and displayed with all its children. The menu is cleared for the new compound.

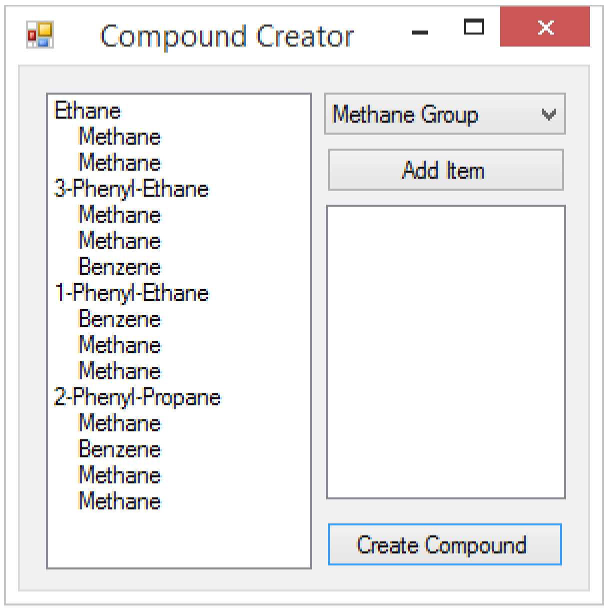


The benzene ring component can also be added.



```
Ethane
    Methane
    Methane
3-Phenyl-Ethane
    Methane
    Methane
    Benzene
```

Components containing benzene are named differently.



```
Ethane
    Methane
    Methane
3-Phenyl-Ethane
    Methane
    Methane
    Benzene
1-Phenyl-Ethane
    Benzene
    Methane
    Methane
2-Phenyl-Propane
    Methane
    Benzene
    Methane
    Methane
```

The placement of benzene affects the naming.

<u>Observations</u>

The composite pattern is useful and interesting. It creates a simple way of handling tree hierarchies. It could also be used in tandem with the iterator pattern, resulting in a tree structure that can easily be iterated through. I feel that my submission fits the assignment very well. I followed the UML Diagram closely in the creation of my program. Organic compounds also function well as the subject matter for such an application. This assignment helped me understand how tree structures are created in an internal system. I believe it will be very useful in the future.