Nathan Bemus

Design Patterns

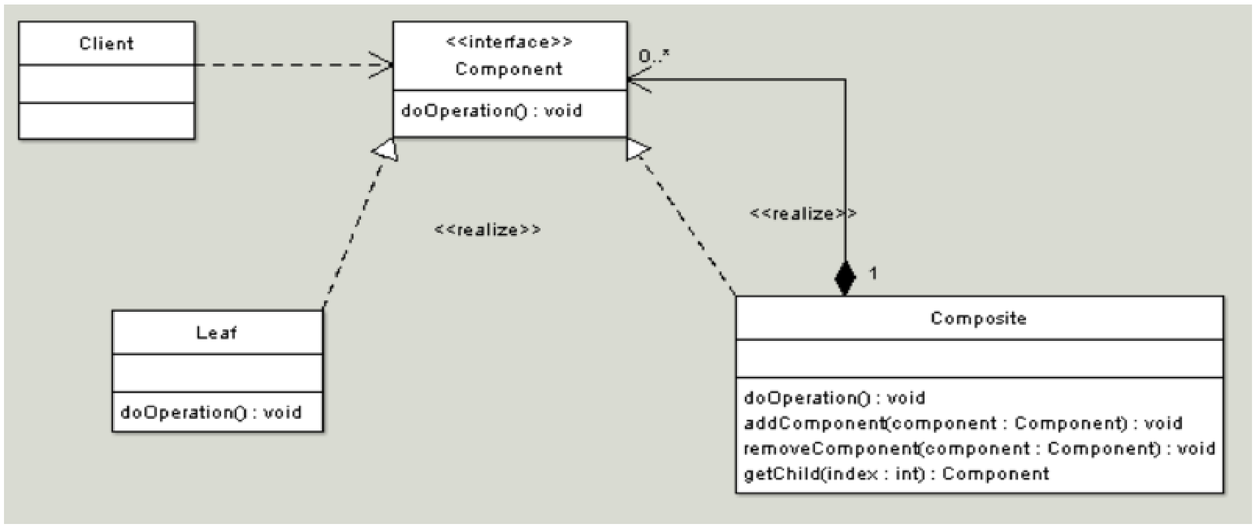
10/17/16

Composite Pattern Paper

Introduction:

The Composite Pattern is used to treat individual objects of a part-whole hierarchy and the compositions of objects in the same manner. My particular program uses this pattern by creating a hierarchy of imaginary computer files and sums up the size of the files.

UML Diagram Analysis:



Component (Memory)	Component creates the interface for the leaves and the composites.
Leaf (Individual)	These are the objects that have no children. They can either stand alone or be a part of a composite branch.
Composite (File)	Stores the child components and defines the operations declared in the component class.
Client (Form1)	Manipulates the objects in in the composite hierarchy.

Program Walkthrough:

Memory Class:

```
public abstract class Memory
    {
        string fileType;
        public abstract string Display(int depth);
        public abstract void remove(Memory file);
        public abstract void add(Memory file);
    }
```

This is the component class for my program. The methods being declared are Display, Add, and Remove.

Individual Class:

```
public class Individual : Memory
    {
        string theType;
        int theFileSize;

        public Individual(string type, int fileSize)
        {
            theType = type;
            theFileSize = fileSize;
        }

        public override void add(Memory file)
        {
            throw new NotImplementedException();
        }

        public override void remove(Memory file)
        {
            throw new NotImplementedException();
        }

        public override string Display(int depth)
        {
            String word = new String('-', depth) + theType + " " + theFileSize + " Kb 
file size " + System.Environment.NewLine;
            return word;
        }
    }
```

This is the Leaf Class. The individual class inherits methods from the Memory class. The individual class overrides the declarations stated in the Memory class.

File Class:

```
public class File : Memory
    {
        string fileType;
        string display;
        string current;

        private List<Memory> memoryUsed = new List<Memory>();

        public override void add(Memory file)
        {
            memoryUsed.Add(file);
        }

        public override void remove(Memory file)
        {
            memoryUsed.Remove(file);
        }

        public override string Display(int depth)
        {
            foreach (Memory memory in memoryUsed)
            {
                if (fileType == "Gaming")
                {
                    display = "Gaming :" + System.Environment.NewLine;
                }
                display += memory.Display(depth + 2);
            }
            current = display;
            display = "";
            return current;
        }

        public File(string type)
        {
            fileType = type;
        }
    }
```

The File class is the composite class. This class inherits methods from the Memory class and overrides them to have functionality. This class also creates a list object that the methods add and remove from.

Form1 Class:

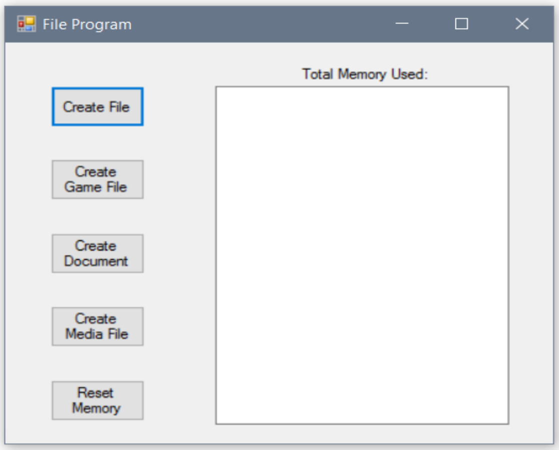
```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        int memoryAllocated = 0;
        File file = new File("File");
        File gaming = new File("Gaming");
        File media = new File("Media");
```

This is the client class. This class is used to manipulate and send information to the other classes. Based off of the button chosen by the user the program creates a new object adds it to the list.

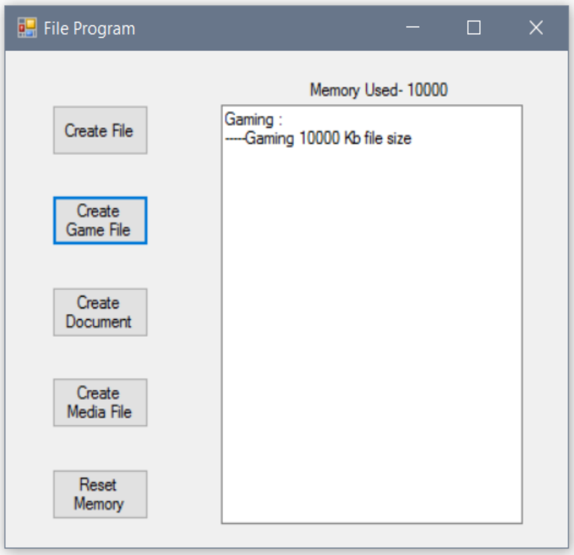
```
        private void btnCreateFile_Click(object sender, EventArgs e)
        {
            file.add(new Individual("File", 1));
            memoryAllocated += 1;
            tbxMemoryStatus.Text = file.Display(1);
            lblMemoryStatus.Text = "Memory Used- " + memoryAllocated;
        }

        private void btnCreateGame_Click(object sender, EventArgs e)
        {
            gaming.add(new Individual("Gaming", 10000));
            memoryAllocated += 10000;

            file.add(gaming);
            tbxMemoryStatus.Text = file.Display(1);
            lblMemoryStatus.Text = "Memory Used- " + memoryAllocated;
        }

        private void btnCreateDoc_Click(object sender, EventArgs e)
        {
            file.add(new Individual("Document", 10));
            memoryAllocated += 10;
            tbxMemoryStatus.Text = file.Display(1);
            lblMemoryStatus.Text = "Memory Used- " + memoryAllocated;
        }

        private void btnCreateMediaFile_Click(object sender, EventArgs e)
        {
            file.add(new Individual("Media", 1000));
            memoryAllocated += 1000;
            tbxMemoryStatus.Text = file.Display(1);
            lblMemoryStatus.Text = "Memory Used- " + memoryAllocated;
        }

        private void btnClear_Click(object sender, EventArgs e)
        {
            memoryAllocated = 0;
            tbxMemoryStatus.Text = "";
            lblMemoryStatus.Text = "Memory Used- " + memoryAllocated;
        }
    }
```

Screenshots:

This is what it looks like first opened up.



This is when a composite is added.



```
Gaming :
----Gaming 10000 Kb file size
```

Conclusion:

This program was very difficult to write properly. I still am not sure if I did it correctly but it does follow the UML Diagram. I think this pattern could be very useful I however was unable to find a proper use for it.