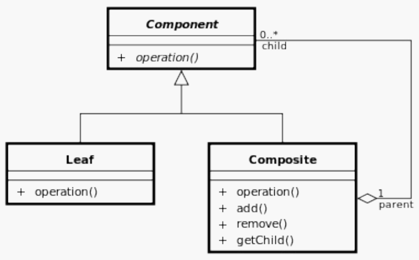
Composite design pattern used in ASP.NET Framework

Published 6/13/2013 by Raghav Khunger

In this article I will explain one reference of Composite pattern used in ASP.NET framework. We all are familiar with Render method used in ASP.NET. It is the last hook to modify the content which is to be render. ASP.NET uses *Composite Design Pattern* to work with this render behavior. This pattern is used when there is manipulation of a hierarchical collection of “primitive” and “composite” objects. By primitive and composite objects here we can think of a simple control like *Literal* and composite control like *Repeater* which itself can contains other controls like Literal, TextBox or Repeater itself. The key factor in this pattern is that you can manipulate a single instance of the object just as you would manipulate a group of them.

Structure:



Component:
It is the abstraction for all components. It can be an interface or an abstract class.
Leaf:
Represents a single and separate unit of composition.
Composite:
It contains methods to manipulate children in recursive way if needed.

This pattern is mostly used in tree hierarchy like structures. Let's understand how this pattern is used with Render method. Below is a simple representation how controls hierarchy looks in ASP.NET.

```
Control
--HtmlControl
--WebControl
----Literal
```

```
----Repeater
------Contents inside repeater(Literal, TextBox or Repeater itself)
```

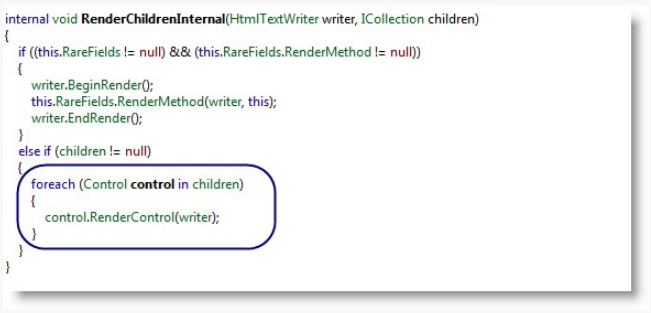
System.Web.UI.Control (**Composite**):

You can think of System.Web.UI.Control as *Composite.* It has exposed its *Render* and *RenderChildren* methods as virtual so that the child classes (Literal, TextBox etc.) can override it and can set their custom behavior on these methods. Here, Literal and TextBox class represents separate components.

```
protected internal virtual void Render(HtmlTextWriter writer)
{
    this.RenderChildren(writer);
}
```

```
protected internal virtual void RenderChildren(HtmlTextWriter writer)
{
   ICollection children = this._controls;
   this.RenderChildrenInternal(writer, children);
}
```

```
internal void RenderChildrenInternal(HtmlTextWriter writer, ICollection children)
{
    if ((this.RareFields != null) && (this.RareFields.RenderMethod != null))
    {
        writer.BeginRender();
        this.RareFields.RenderMethod(writer, this);
        writer.EndRender();
    }
    else if (children != null)
    {
        foreach (Control control in children)
        {
            control.RenderControl(writer);
        }
    }
}
```



```
internal void RenderChildrenInternal(HtmlTextWriter writer, ICollection children)
{
  if ((this.RareFields != null) && (this.RareFields.RenderMethod != null))
  {
    writer.BeginRender();
    this.RareFields.RenderMethod(writer, this);
    writer.EndRender();
  }
  else if (children != null)
  {
    foreach (Control control in children)
    {
      control.RenderControl(writer);
    }
  }
}
```

As you can see it contains the behavior call the Render behavior of each child controls. In this manner we need not to call Render method of each child controls in ASP.NET. For example if we have to call Render method of a repeater we will call Render of repeater only and it will internally call the Render of each control inside it via composite pattern.

Literal, TextBox will act as leaf nodes.

```
public class Literal: Control
{
   ...
}
```

```
public class TextBox: Control
{
   ...
}
```

There are other places too where ASP.NET implements this pattern like ASP.NET Tree control. I hope from above explanation which we discussed in context of ASP.NET framework you got the idea how composite pattern works.

tags : .net, asp.net, composite-design-pattern, design-patterns