عنوان: **الگوی Composite**
نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۳۹۲/۰۹/۲۴ ۱۱:۲۰
آدرس: www.dotnettips.info
گروه‌ها: Design patterns

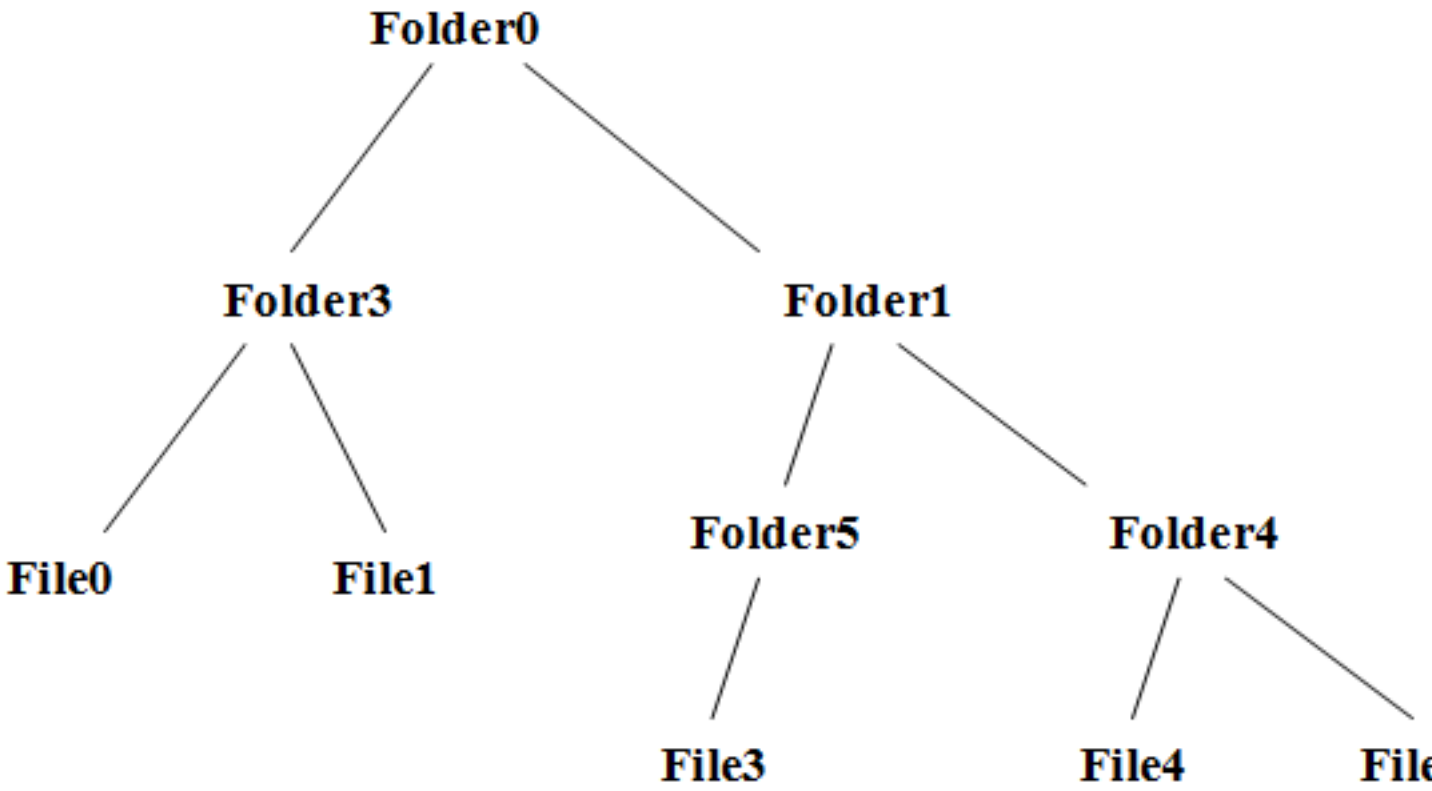
الگوی Composite یکی دیگر از الگوهای ساختاری می‌باشد که قصد داریم در این مقاله آن را بررسی نماییم.
الگوی Composite در عمل یک Collection Pattern (الگوی مجموعه ای) است. که می‌توان در درون آن ترکیبی از زیر مجموعه‌های مختلف را قرار داد و سپس هر زیر مجموعه را به نوبه خود فراخوانی نمود.به بیان دیگر الگوی Composite به ما کمک می‌کند که در یک ساختار درختی بتوانیم مجموعه ای (Collection ی)،از بخشی از آبجکتهای سلسله مراتبی را نمایش دهیم. این الگو به Client اجازه می‌دهد، که رفتار یکسانی نسبت به یک Collection ی از آبجکتها یا یک آبجکت تنها داشته باشد.

مثالهای متعددی می‌توان از الگوی Composite زد، که در ذیل به چند نمونه از آنها می‌پردازیم:

نمونه اول: همانطور که می‌دانیم یک سازمان از بخشهای مختلفی تشکیل شده است، که بصورت سلسله مراتبی با یکدیگر در ارتباط می‌باشند، چنانچه بخواهیم بخشها و زیر مجموعه‌های تابعه آنها را بصورت آبجکت نگهداری نماییم، یکی از بهترین الگوهای پیشنهاد شده الگوی Composite می‌باشد.

نمونه دوم: در بحث حسابداری،یک حساب کل از چندین حساب معین تشکیل شده است و هر حساب معین نیز از چندین سرفصل حسابداری تشکیل می‌شود. بنابراین برای نگهداری آبجکتهای معین مرتبط به حساب کل، می‌توان آنها را در یک Collection قرار داد. و هر حساب معین را می‌توان،در صورت داشتن چندین سرفصل در مجموعه خود به عنوان یک Collection در نظر گرفت. برای دسترسی به هر حساب معین و سرفصل‌های زیر مجموعه آن نیز می‌توان از الگوی Composite استفاده نمود.

نمونه سوم: یک File System را در نظر بگیرید،که ساختارش از File و Folder تشکیل شده است. و می‌تواند یک ساختار سلسله مراتبی داشته باشد.بطوریکه درون هر Folder می‌تواند یک یا چند File یا Folder قرار گیرد. و در درون Folderهای زیر مجموعه می‌توان چندین File یا Folder دیگر قرار داد.اگر بخواهیم به عنوان نمونه شکل ساختار درختی File و فولدر را نمایش دهیم بصورت زیر خواهد بود:

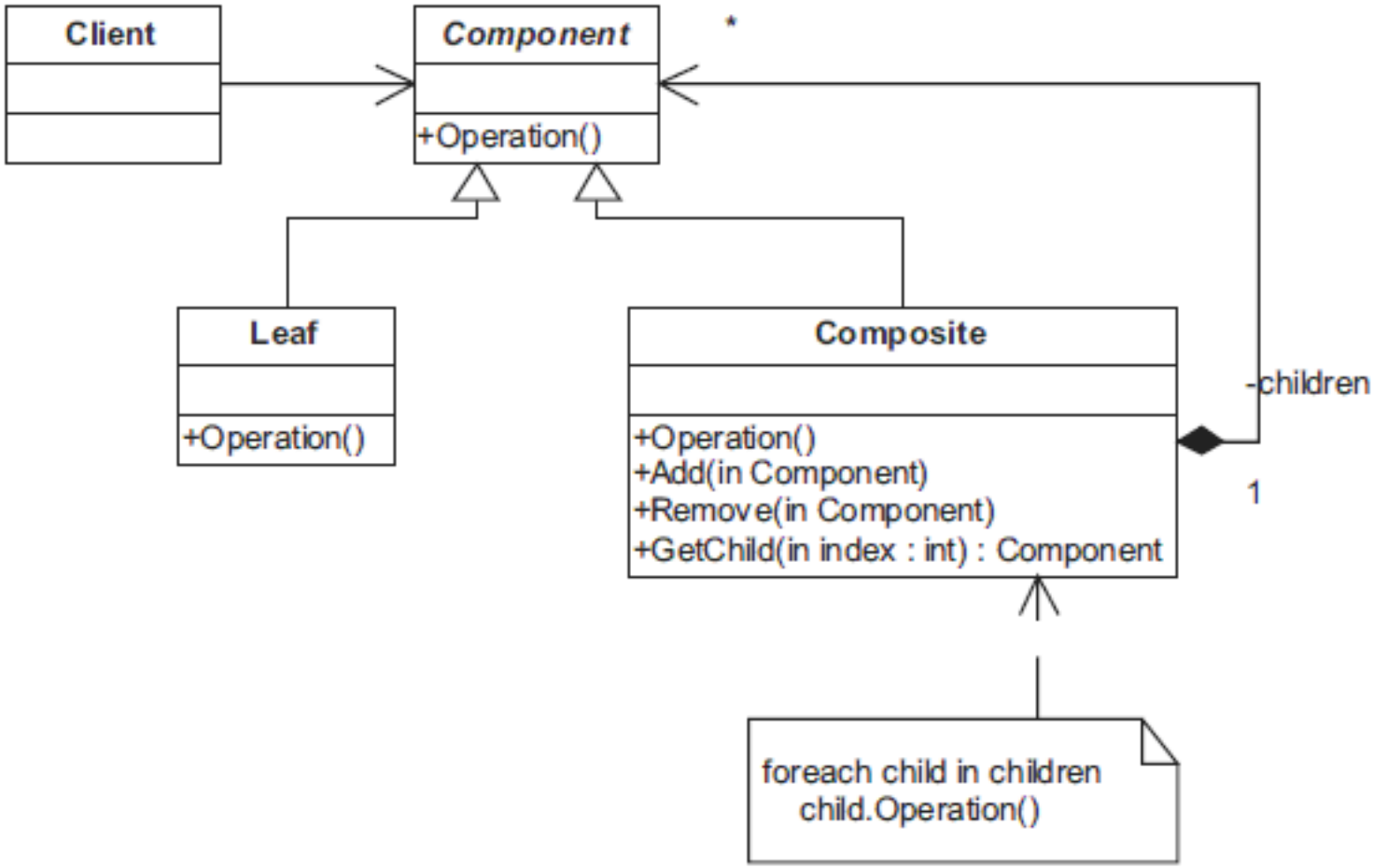


در ساختار درختی به Folder شاخه یا Branch گویند، چون می‌تواند زیر شاخه‌های دیگری نیز در خود داشته باشد. و به File برگ یا Leaf گویند.برگ نمی‌تواند زیر مجموعه ای داشته باشد. در واقع برگ (Leaf) بیانگر انتهای یک شاخه می‌باشد.

نمونه آخر:می توان به ساختار منوها در برنامه‌ها اشاره نمود.هر منو می‌تواند شامل چندین زیر منو باشد. و همان زیر منوها می‌توانند از چندین زیر منوی دیگر تشکیل شوند. این ساختار نیز یک ساختار سلسله مراتبی می‌باشد، و برای نگهداری آبجکتهای یک مجموعه می‌توان از الگوی Composite استفاده نمود.

الگوی Composite از سه Component اصلی تشکیل شده است،که یکایک آنها را بررسی می‌کنیم:
Component: کلاس پایه ای است که در آن متدها یا Functionalityهای مشترک تعریف می‌گردد. Component می‌تواند یک Abstract Class یا Interface باشد.
Leaf : به آبجکتهای گفته می‌شود که هیچ Child ی ندارند. و فقط یک آبجکت مستقل تنها می‌باشد. کلاس Leaf متدهای مشترک تعریف شده در Component را پیاده سازی می‌کند.اگر مثال File و Folder را بخاطر آورید،File یک آبجکت از نوع Leaf است چون نمی‌تواند هیچ فرزندی داشته باشد و یک آبجکت تنها می‌باشد.
Composite: کلاس فوق Collection ی از آبجکتها را در خود نگهداری می‌کند، به عبارتی در Composite می‌توان بخشی از ساختار درختی را قرار داد، که این ساختار می‌تواند ترکیبی از آبجکتهای Leaf و Composite باشد. در مثال File و Folder، یک Folder را می‌توان به عنوان Composite در نظر گرفت،زیرا که یک Folder می‌تواند چندین File یا Folder را در خود جای دهد. در کلاس Composite معمولا متدهایی همچون Add (افزودن Child)،Remove (حذف یک Child) و غیرو... وجود دارد.

کلاس Leaf و کلاس Composite از کلاس Component ارث بری (Inherit) می‌شوند.
شکل زیر بیانگر الگوی Composite می‌باشد:



توصیف شکل: طبق تعاریف گفته شده، دو کلاس Leaf و Composite از Component، Inherit شده اند. و Client نیز فقط متدهای مشترک تعریف شده در Component را مشاهده می‌کند، به عبارتی Client رفتار یکسانی نسبت به Leaf و Composite خواهد داشت.

برای درک بیشتر الگوی Composite مثالی را بررسی می‌کنیم، فرض کنید در کلاس Component متدی به نام Display را تعریف می‌کنیم،بطوریکه نام آبجکت را نمایش دهد.بنابراین خواهیم داشت:

اینترفیسی را برای Component در نظر می‌گیریم، و متد Display را در آن تعریف می‌کنیم:

```
public interface Icomponent
    {
        void Display(int depth);
    }
```

در کلاس Leaf، اینترفیس IComponent را پیاده سازی می‌نماییم:

```
public class Leaf:Icomponent
    {
        private String name = string.Empty;
        public Leaf(string name)
        {
            this.name = name;
        }

        public void Display(int depth)
        {

            Console.WriteLine(new String('-', depth) + ' ' + name);
```

```
        }
    }
```

در کلاس Composite نیز اینترفیس IComponent را پیاده سازی می‌نماییم، با این تفاوت که متدهای Add و Remove را نیز در کلاس Composite اضافه می‌کنیم، چون قبلا هم گفته بودیم، Composite در حکم یک Collection می‌باشد، بنابراین می‌بایست قابلیت حذف و اضافه نمود آبجکت در خود را داشته باشد. پیاده سازی متد Display در آن بصورت Recursive (بازگشتی) میباشد. و علتش این است که بتوانیم ساختار سلسله مراتبی را بازیابی نماییم.

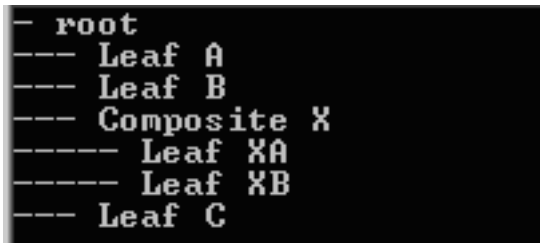
```
public class Composite:Icomponent
    {
        private List<Icomponent> _children = new List<Icomponent>();
        private String name = String.Empty;

        public Composite(String sname)
        {
            this.name = sname;
        }

        public void Add(Icomponent component)
        {

            _children.Add(component);

        }


        public void Remove(Icomponent component)
        {

            _children.Remove(component);

        }


        public void Display(int depth)
        {

            Console.WriteLine(new String('-', depth) + ' ' + name);


            // Recursively display child nodes

            foreach (Icomponent component in _children)
            {

                component.Display(depth + 2);

            }

        }
    }
```

در ادامه بوسیله چندین آبجکت Leaf و Composite یک ساختار درختی را ایجاد می‌کنیم.

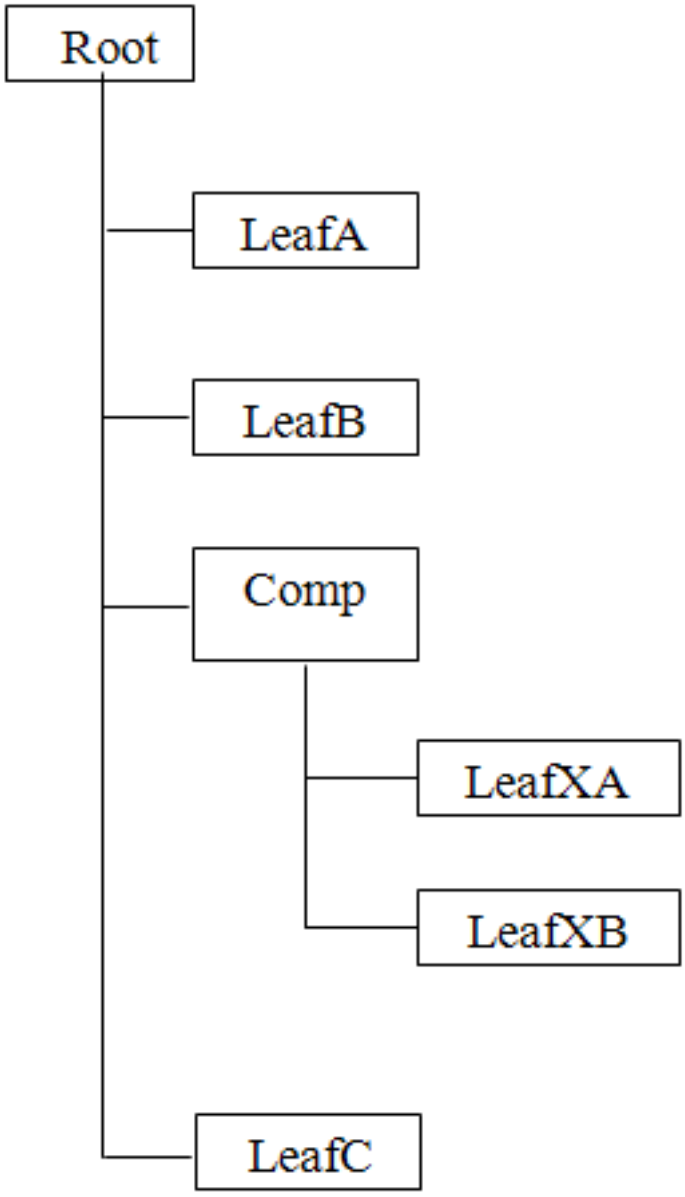
```
class Program
    {
        static void Main(string[] args)
        {
            // Create a tree structure

            Composite root = new Composite("root");

            root.Add(new Leaf("Leaf A"));

            root.Add(new Leaf("Leaf B"));


            Composite comp = new Composite("Composite X");

            comp.Add(new Leaf("Leaf XA"));
```

```
            comp.Add(new Leaf("Leaf XB"));

            root.Add(comp);
            root.Add(new Leaf("Leaf C"));

            // Add and remove a leaf
            Leaf leaf = new Leaf("Leaf D");
            root.Add(leaf);
            root.Remove(leaf);

            // Recursively display tree
            root.Display(1);
            Console.ReadKey();
        }
    }
```

در ابتدا یک آبجکت Composite ایجاد می‌کنیم و آن را به عنوان ریشه در نظر گرفته و نام آن را Root قرار می‌دهیم. سپس دو آبجکت LeafA و LeafB را به آن می‌افزاییم، در ادامه آبجکت Composite دیگری به نام Comp ایجاد می‌کنیم، که خود دارای دو فرزند به نامهای LeafXA و LeafXB می باشد. و سر آخر هم یک آبجکت LeafC ایجاد می‌کنیم.
آبجکت LeafD صرفا جهت نمایش افزودن و حذف کردن آن در یک آبجکت Composite نوشته شده است. برای این که بتوانیم ساختار سلسله مراتبی کد بالا را مشاهده نماییم، متد Root.Display را اجرا می‌کنیم و خروجی آن بصورت زیر خواهد بود:



اگر بخواهیم،شکل درختی آن را تصور کنیم بصورت زیر خواهد بود:



درپایان باید بگویم،که نمونه کد بالا را می‌توان به ساختار File و Folder نیز تعمیم داد، بطوریکه متدهای مشترک بین File و Folder را در اینترفیس IComponent تعریف می‌کنیم و بطور جداگانه در کلاسهای Composite و Leaf پیاده سازی می‌کنیم.
امیدوارم توضیحات داده شده در مورد الگوی Composite مفید واقع شود.