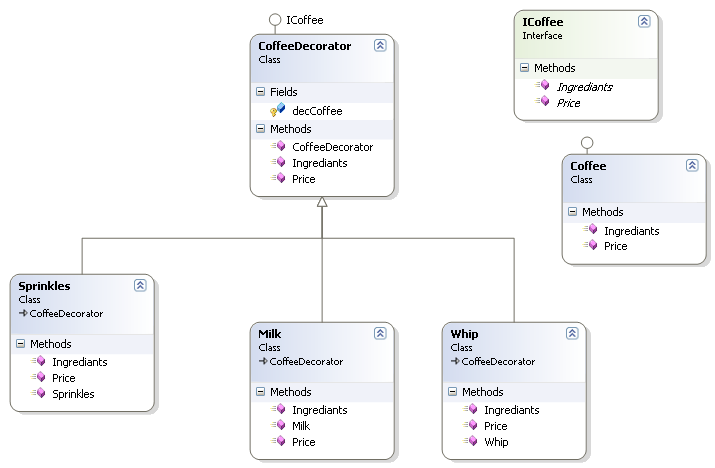
Top of Form

we saw that how Microsoft has used abstract Factory (GOF design pattern) pattern in .Net framework library. In this article, we will see how Decorator Pattern has used in .Net Framework Library.

Introduction

Decorator pattern is a software design pattern where we can use to extend the functionality of the object run time. Decorator pattern attaches additional responsibility by decorating the other object by assigning an instance to similar base interface instance.  
  
Here is the example of Coffe making system.

Class Diagram



First there is an interface Called ICoffee which every subclasses will implement this interface.

public interface ICoffee

{

double Price();

string Ingrediants();

}

Next we will declare the basic implementation of the coffe class by implementing ICoffee Interface.

public class Coffee : ICoffee

{

public double Price()

{

return 3.5;

}

public string Ingrediants()

{

return "Simple Coffee";

}

}

Next we will create a decorator base implementation by implementing the ICoffee interface. This class declare the ICoffee interface variable and accepts the instance ICoffee concrate instance object.

public class CoffeeDecorator : ICoffee

{

protected ICoffee decCoffee;

public CoffeeDecorator(ICoffee coffee)

{

decCoffee = coffee;

}

public virtual double Price()

{

return decCoffee.Price();

}

public virtual string Ingrediants()

{

return decCoffee.Ingrediants();

}}

We will implement the generalize version of the cofee classes.

public class Milk : CoffeeDecorator

{

public Milk(ICoffee coffee)

: base(coffee) { }

public override double Price()

{

return base.Price() + .50d;

}

public override string Ingrediants()

{

return base.Ingrediants() + " + Milk Addedd";

}

}

public class Whip : CoffeeDecorator

{

public Whip(ICoffee coffee)

: base(coffee) { }

public override double Price()

{

return base.Price() + .50d;

}

public override string Ingrediants()

{

return base.Ingrediants() + " + Whip Addedd";

}

}

public class Sprinkles : CoffeeDecorator

{

public Sprinkles(ICoffee coffee)

: base(coffee) { }

public override double Price()

{

return base.Price() + .50d;

}

public override string Ingrediants()

{

return base.Ingrediants() + " + Sprinkles Addedd";

}

}

Finally we will see how we can attach the additional functionality to the classes using decorator pattern.

class Program

{

static void Main(string[] args)

{

ICoffee coffee = new Coffee();

Console.WriteLine("Price: " + coffee.Price() + " Ingrediants: " + coffee.Ingrediants());

coffee = new Milk(coffee);

Console.WriteLine("Price: " + coffee.Price() + " Ingrediants: " + coffee.Ingrediants());

coffee = new Whip(coffee);

Console.WriteLine("Price: " + coffee.Price() + " Ingrediants: " + coffee.Ingrediants());

coffee = new Sprinkles(coffee);

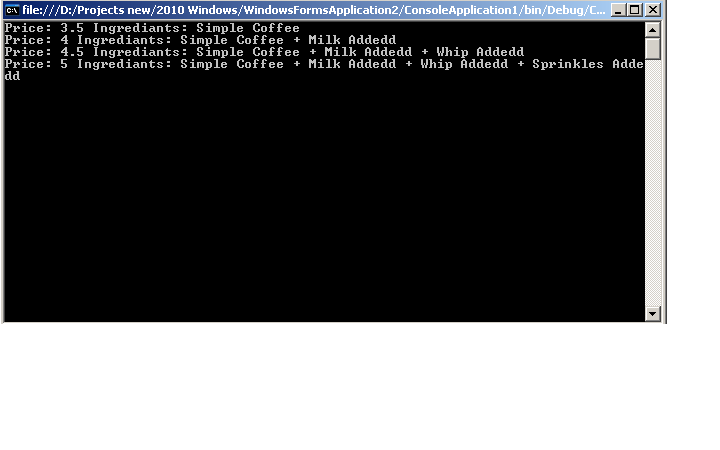
Console.WriteLine("Price: " + coffee.Price() + " Ingrediants: " + coffee.Ingrediants());

Console.ReadLine();

}

}

Output



.Net Framework Library Example

Similarly, in .net library, the following is one of the example where the decorator pattern is being used.  
There is an abstract stream class is in .net library and there are set of classes designed aound the stream class. The BufferedStream & CryptoStream classes are the example of decorator pattern.

if you try to create an instance of the BufferedStream class then you see that it accepts the stream class instance in the contructor which means that bufferstream class add additional functionality to the stream class as BufferedStream and Stream class has the similar behavior.

Bottom of Form