در ادامه مقالات سری الگوهای طراحی نرم افزار به الگوی Command می رسیم.

**مقدمه**

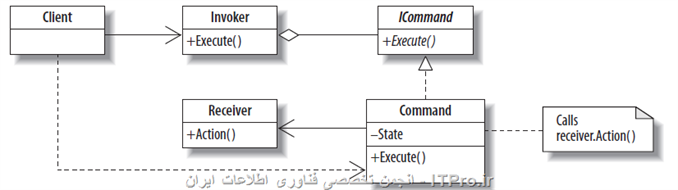
این الگو به این شکل است که بین کاربری که یک درخواست را ارسال کرده و شیئی که می تواند آن دستور را اجرا کند یک فاصله می اندازد. این الگو یک الگوی چندکاره است که امکانات زیر را پشتیبانی می کند. 

* ارسال در خواست ها برای گیرنده های مختلف
* توانایی صف بندی و لاگ گرفتن از درخواست های دریافت شده و یا رد کردن در خواست ها
* ایجاد تراکنش هاس سطح بالا از اعمال سطح پایین
* انجام اعمال Redo و Undo

**شرح**

لگوی Command در سیستم منو های بسیاری از برنامه های معروف استفاده می شود. در بسیاری از نرم افزار هایی که امروزه با آنها سرو کار داریم منویی به نام Edit دارند که اعمال کپی و کات کردن را داخل منوی Edit دارند و از طرفی هم همان اعمال کپی و کات کردن در داخل نوار ابزار بالای صفحه نیز موجود می باشد. این به این معنی است که هر دوی آنها یک عمل را انجام می دهند. و این اعمال دارای قابلیت Undo نیز می باشند. همه ی این اعمال کاربرد های الگوی Command می باشد.

**طراحی**



طراحی الگوی command به شکل زیر است. کلاس Client روش معینی دارد که درخواست خود را اعلام کند. Receiver ها که ممکن است بیش از یکی باشند می دانند که چگونه باید درخواست کاربر را اجرا نمایند با توجه به مثال منو ها که قبلا گفتیم یک فرمان Cut برای یک متن در قسمتی از برنامه اجرا می شود و فرمان Cut مربوط به یک عکس در قسمت دیگری از برنامه هندل می شود.   
کلاس Command به عنوان یک واسط (interface)بین Client و Receiver عمل می نماید. در یک شی Command درخواست های Client مشخص شده است و همچنین اعمال مربوط به این درخواست ها و گیرنده آن نیز در کلاس Command معین گردیده است. ممکن است که Client بخواهد وضعیت خود را نیز برای دریافت کننده ارسال کند که کلاس Command این اجازه را می دهد. کلاس Invoker برای جدا کردن کلاس client از کلاس Receiver می باشد.   
به بیان ساده تصور کنید که Client تابع Execute مربوط به Invoker را فراخوانی می کند. با این کار درخواست کاربر به Command ارسال می شود و تابع Action مربوط به Receiver مورد نظر اجرا می شود. یک برنامه می تواند دارای تعداد زیادی درخواست با انواع متفاوت باشد که هر کدام به گیرنده های مربوط به خود ارسال می شوند. اینترفیس ICommand اطمینان حاصل می کند که هرکدام از درخواست ها به فرم استاندارد برای گیرنده مورد نظر باشد.   
کلاس هایی که در این الگو نقش دارند عبارتند از: 

* Client: کلاس های Command را می سازد و اجرا می کند.
* ICommand: اینترفیسی است که عملیات قابل اجرا را مشخص می نماید.
* Invoker: درخواست اجرای Action را به کلاس Command می دهد.
* Command: کلاسی که عملیات اجرایی را پیاده سازی می کند که در داخل این عملیات دستورات فراخوانی دستوراالعمل های روی گیرنده وجود دارد.

از ویژگی های الگوی Command می توان به موارد زیر اشاره کرد: 

* دستورات در کلاس Command می توانند با هم ترکیب شده و دستورات بزرگتری به وجود آورند.
* دستورات جدید می توانند به مجموعه ما اضافه شوند بدون این که خللی در کار دستورات قبلی به وجود آورند.

**پیاده سازی**

یک مثال ساده از پیاده سازی این الگو در ادامه آمده است. 

using System;

internal class Program

{

private delegate void Invoker();

private static Invoker Execute, Undo, Redo;

private class Command

{

public Command(Receiver receiver)

{

Execute = receiver.Action;

Redo = receiver.Action;

Undo = receiver.Reverse;

}

}

public class Receiver

{

private string build, oldBuild;

private readonly string s = "Itpro string ";

public void Action()

{

oldBuild = build;

build += s;

Console.WriteLine("Receiver is adding " + build);

}

public void Reverse()

{

build = oldBuild;

Console.WriteLine("Receiver is reversing to " + build);

}

}

private static void Main(string[] args)

{

new Command(new Receiver());

Execute();

Redo();

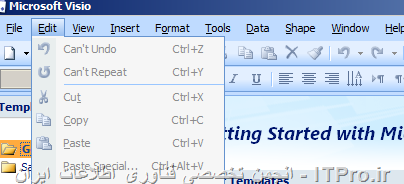
Undo();

Execute();

Console.ReadKey();

}

}



کلاس Invoker وICommand به عنوان یک نوع delegate معرفی شده اند و در خط بعد سه شی از آن ساخته شده است. نام هرکدام از اشیا با توجه به کاری که کاربر از آنها می خواهد نام گذاری شده اند. دستور مربوط به کلاس Command در داخل Receiver با استفاده از دو تابع Action و Reverse پیاده سازی شده است. دستورات Execute و Redo هر دو تابع Action را اجرا می نمایند و دستور Undo تابع Reverse را اجرا می کند. در این کد Receiver مسئول نگهداری حالت شی است و همانگونه که در کد می بینید هر بار که اجرا می شود مقدار s تغییر می کند. همچنین در این کد عمل undo کردن فقط برای یک بار ممکن است ولی اگر بخواهیم این کار را در واقعیت انجام دهیم باید برای نگهداری حالت های قبلی باید بیشتر از یک عمل را در نظر بگیریم که مکانیزم های متفاوتی برای پیاده سازی آن وجود دارد.

**Command های دارای چند گیرنده (Multireceiver)**

حال اگر در یک برنامه چندین نوع delegate داشته باشیم و بیشتر از یک گیرنده داشته باشیم چه اتفاقی خواهد افتاد؟ اول از همه باید نام delegate ها (Invoker) باید با هم متفاوت باشد. اشیاء Command باید به گونه ای باشند که بین اشیا delegate و گیرنده ها ارتباط ایجاد کند. در برنامه ای که در ادامه آورده شده است، دو کلاس Command و دو گیرنده وجود دارند. چون هر دوی command ها از delegateها (invoker) یکسان استفاده می کنند، Client دستورات را اول در یکی از گیرنده ها و سپس در دیگری اجرا می کند ولی ممکن است این کار لازم نباشد. برای این که این مشکل را حل کنیم می توانیم اشیا delegaet بیشتری معرفی کنیم یا آنها را در داخل Command ها گروه بندی نماییم که در مثال زیر نشان داده شده است. 

public class MultiReceiver

{

private delegate void Invoker();

private delegate void InvokerSet(string s);

private static Invoker Execute, Redo, Undo;

private static InvokerSet Set;

class Command

{

public Command(Receiver receiver)

{

Set = delegate

{

Console.WriteLine("Not implemented- default of Itpro used");

receiver.S = "Itpro";

};

Execute = receiver.Action;

Redo = receiver.Action;

Undo = receiver.Reverse;

}

}

class Command2

{

public Command2(Receiver2 receiver2)

{

Set = receiver2.SetData;

Execute = receiver2.DoIt;

Redo = receiver2.DoIt;

Undo = () => Console.WriteLine("Not Implemented");

}

}

class Receiver

{

private string build, oldBuild;

private string s = "Itpro String ";

public string S { get { return s; } set { s = value; } }

public void Action()

{

oldBuild = build;

build += s;

Console.WriteLine("Receiver is adding "+build);

}

public void Reverse()

{

build = oldBuild;

Console.WriteLine("Receiver is reverting to "+build);

}

}

class Receiver2

{

private string build, oldBuild;

private string s = "Itpro String2 ";

public void SetData(string s)

{

this.s = s;

}

public void DoIt()

{

oldBuild = build;

build += s;

Console.WriteLine("Receiver is building "+build);

}

}

class Client

{

public void ClientMain()

{

new Command(new Receiver());

Execute();

Redo();

Undo();

Set("III");

Execute();

Console.WriteLine();

new Command2(new Receiver2());

Set("Hoses ");

Execute();

Set("Castles ");

Undo();

Execute();

Redo();

}

}

private static void Main(string[] args)

{

new Client().ClientMain();

}

}

از کاربرد های دیگر این الگو ثبت لاگ برنامه است که یک تاریخچه از کارهایی که انجام شده است نگهداری می کند مثلا ساده ترین نوع لاگ گیری این است که تعداد انجام یک کار را بشماریم. برای این کار عملیات مورد نظر را در قسمت Invoker پیاده سازی می کنیم.**Itpro باشید**