**الگوی Proxy یا واسطه**

**یک کلاس اصلی (شامل یکسری عملیات )داریم و کاربر میخواهد از این کلاس اصلی تحت یکسری شرایط استفاده کند. ویا قبل یا بعد از اجرا شدن کلاس اصلی منطقی را اجرا کنیم.اما نمیخواهیم این منطق در کلاس اصلی اجرا شود.**

**مثال در دنیای واقعی :**

**کارت های بانکی و پول را می توان نامبرد. در این جا پول نقد کلاس اصلی ما می باشد ، در گذشته برای خرید شما باید از پول نقد استفاده میکردید اما بعد ها توسط بانک کارت های اعتباری را ارائه شد و میتوان در زمان خرید به جای پرداخت وجه نقد از کارت اعتباری استفاده کرد.**

روش های پیاده سازی Proxy ها

1. Virtual Proxy : زمانیکه ما در برنامه یک کلاس سنگین داریم و نمونه سازی از این کلاس دردساز می شود مثل: اشغال بیش از حد سیستم. ویژگی این روش در این است که نمونه سازی از این کلاس سنگین را زمانی که به آن نیاز است انجام میدهیم.
2. Protection Proxy : زمانی که ما کلاسی داریم و میخواهیم از نمونه های آن کلاس تحت شرایطی را بررسی کنیم که مجوز استفاده از این نمونه ها داده شده یا خیر.
3. Remote Proxy : مثلا یک شی که در درسترس مستقیم نیست و نمی توان آن را مستقیم نمونه سازی کرد،برای استفاده باید از آدرس Ip و Port یا اتصالات شبکه استفاده کرد ما از این نوع استفاده میکنیم.
4. Loggin Proxy : لاگ کردن اینکه چه کسی یا چه Client هایی مورد استفاده قرار گرفته.
5. Caching Proxy : کش کردن اطلاعاتی که توسط شی نمونه سازی شده تولید کرده.

نکات این الگو

1. از کلاس abstract هم یتوان به جای interface هم استفاده کرد.
2. کلاس RealSubject را به سطح دسترسی internal تعریف کنیم که client نتواند از آن نمونه سازی کند.
3. اگر Subject و RealSubject ما ساده باشد می توان interface را حذف کرد کلاسProxy مستقیما RealSubject استفاده کرد.
4. در نگاه اول این الگو شباهتی به کلاس الگوی Decorator دارد.الگوی decorator ویژگی هایی را به یک کلاس اضافه میکرد ولی در این الگو منطق دسترسی به یک کلاس را انجام می دهیم.
5. بین کلاس Proxy و RealSubject اتصال محکمی برقرار شده استاما اگر میخواهیم از اتصال سست استفاده کنیم. از تکنیک تزریق وابستگی در Proxy( به جای RealSubject از ISubject )استفاده میکنیم. اگر به ازای Isubject چندین پیاده سازی داشتیم حتما و حتما از اتصال سست استفاده میکنیم.
6. ما میتوانیم نمونه ای که از کلاس RealSubject می سازیم را به صورت متد Protected ایجاد کنیم و در کلاس Proxy هرجا نیاز بود از این متد استفاده کنیم.

مزایا:

1. منطق دسترسی به نمونه های کلاس را در یک کلاسی بنام Proxy متمرکز میکنیم
2. می توانیم اصل SRP جدا کردن منطق دسترسی به نمونه های کلاس از خود کلاس و پیاده سازی در کلاسی بنام SRP
3. اصل Open/Close : ما براساس Subject ، proxy های مختلفی را توسعه میدهیم. اگر بخواهیم یک proxyرا توسعه بدهیم می توانیم بدون تغییر در Proxyهای دیگر را توسعه دهیم.
4. اگر شی اصلی در دسترس نباشد ،کلاینت می تواند با استفاده از Proxy کار خود را انجام دهد.

معایب:

1. ممکن است کد های ما را پیچیده تر کند.
2. با توجه به شباهت به الگوی decorator همه نقاط ضعف این الگو را دارا می باشد.
3. در صورت تغییر Subject همه Proxy های ارث بری کرده از subject این تغییرات را اعمال کند.

چه زمانی از این الگو استفاده می شود:

1. اشیایی داریم که ایجادشان هزینه بر است.
2. اشیایی که نیاز به کنترل دسترسی دارند.
3. اشیایی داریم که می خواهیم به اشیای راه دور دسترسی داشته باشند (Remote Proxy)
4. اشیایی داریم که هنگام دسترسی به آن ها می خواهیم همان لحظه کاری انجام دهیم (مثل لاگ زدن )