**الگوی آذینگر یا Decorator**

**الگوی weapper**

**نمونه ها در dotnet**

StreamReader

StreamWriter

StringReader

StringWriter

برای مثال دکوراسیون منزل

ما یک خانه را خریداری میکنیم و با استفاده از دکوراسیون یک سری ویژگی جدیدی به آن اضافه میکنیم تا زیباتر شود.

در برنامه نویسی ما یک کلاس داریم و میخواهیم به یکسری Object های آن کلاس بعضی از رفتار ها یا رفتار خاصی را اضافه کنیم.

مثال یک کلاس لاگین توسعه یافته که در حالت عادی یک username و password میگیرد و عمل لاگین انجام میشود.

اما در برخی شرایط می توان sms و یا ایمیل برای کاربر ارسال شود و کاربر مطلع شود که ورود اتفاق افتاده است.

حال به جای استفاده از این الگو چرا خود کلاس لاگین توسعه یافته را تغییر ندهیم؟

1. ممکن است به کد دسترسی نداشته باشیم و صرفا فقط یک dll در اختیار ما باشد.

2. یا یک سیستم قدیمی است و ما به عنوان پشتیبان یا توسعه دهنده سیستم با الزام اینکه تغییری در روند ایجاد نشود.

3. یا این اتفاق اطلاع رسانی را میخواهیم وقتی استفاده شود که اگر عملی اتفاق افتاد یا در شرایط خاص

نکات:

1. به جای Abstract class می توان از interface استفاده کرد.
2. بین concreteComponent و concreteDecorator تفاوتی وجود ندارد.
3. کلاس concreteDecorator نمیداند قرار است چه چیزی را Decorate کند.تنها دانش این کلاس این است که قرار است Feature خودش را بعد از operation اجرا کند.

1. Object ساخته concreteComponent خودش اطلاع ندارد قرار است Feature جدید بر روی عمل خودش انجام شود.
2. اگر قرار است فقط یک ویژگی به برنامه اضافه شود بهتر است خود کلاس Decorator از حالت abstract خارج شده و ویژگی در همان کلا ایجاد شود و به ConcreateDecorator نیازی نمیباشد.

مزیت های الگوی Decorator نسبت به Inhertance

حجم کد ها خیلی کمتر می شود ( مثال پروژه Ex.Decorator.DecoratorVsInheritance)

در وراثت ما ویژگی را به کلاس اضافه میکنیم اما در decorator به object اضافه میشود.

5 مزیت این Design Pattern

1. رعایت اصل SRP (Single Responsibility Principle)
2. انعطاف پذیری بالا ( هر وقت نیاز داشتیم میتونیم قابلیت جدیدی به کلاس اضافه کنیم)
3. پیاده سازی با رایج ترین قابلیت های زبان برنامه نویسی شی گرا
4. توسعه مستقل از دیگر Decorator ها
5. میتوان زنجیره ای از Decorator ها ایجاد کرد.

نقات ضعف این Design Pattern

1. امکان اعمال Decorator های تکراری
2. امکان عمل نکردن به اصل LSP (Liskov Substitution Principle)
3. اجبار به Override کردن متدهای Component در Decorator
4. نمیتوان قابلیت های چند Decorator دیگر را به ارث برد. (یک کلاس فقط میتواند از یک کلاس دیگر ارث بری کند.)
5. ارث بری تمام ویژگی های کلاس Component در Decorator
6. قابل استفاده در کلاس های Protected نمی باشد.

الگوهای مرتبط با Decorator

1. Strategy ( اگر در طراحی اولیه پروژه هستیم بهتر است از این الگو استفاده شود.)
2. Composite

ولی اگر پروژه به این گونه است که از قبل طراحی شده یا به کد دسترسی نداریم استفاده از Decorator بهترین روش است.