جلسه امروز در مورد factoryو abstract factory هست

یکی از مزایای دسته creational این بود که پیچیدگی های ساخت یک آبجکت رو پنهان میکنه

توی فکتوری ما این طوری در نظر میگیریم که ساختار ساختن ابجکت ها به شرایط و کاندیشن های مختلفی بستگی داره و ما نمیخواهیم که داکیومنتش کنیم که برنامه نویسی که از برمنامه ما استفاده میکنه یا اصلا خودمون که دو سه ماه دیگه از کدمون ادامه بدیم همه این کاندیشن رو توی کدمون توی اون زمانی که کد میزنیم چک کنیم و بتونیم آبجکتی که مد نظرمونه رو بسازیم .

ما میخواهیم که نتنها کلاس ها مون رو که اینجا نوشتیم ونوعشون رو مشخص کردیم حتی استاندارد های ساخت اون آبجکت ها برای اون کلاس ها رو مشخص کنیم و بگیم که اگر ی نمونه از این کلاس بخوای بسازی و سه چهار تا شرط مختلف داشته باشی شروطت رو بده به این متود این متود خودش برا تو اون ابجکتی که مد نظرت رو باا اون شرط هایی که داری میسازه

این علاوه براین که باعث میشه اون منطق وکاندیشن های اضافی از وسط کدمون بیاد بیرون و اضافه بشه ب خود پترنمون ی حسن دیگه ای هم که داره ادیت یا اصلاح رو برامون ساده تر میکنه یعنی هر موقع که بخواهیم هر تغییر در کلاسمون بدیم کافیه که متود فکتوریش رو اصلاح کنیم.

فیلم جدید

دیزاین پترن prototype :

خیلی پر استفاده نیس بیشتر سلوشن هایی که با پروتوتایپ راه اندازی میکنن یه راه حل جایگزین هم دارن

یک سوال استخدامی ---------> کپی پیش فرض پایتون shadow copy هست که یک کپی سطحی میگیرد

ولی derp copy یک کپی کامل میگیره و عمیق هست

پروتوتایپ یعنی ی موقع هایی نیاز میشه از نمونه ای که از یک کلاس ساختیم کپی بگیریم

فرض کنیم که ی سیستم مثل اسنپ نوشتیم که بعد از این که هر سفر انجام میشه یک لینک نظر سنجی برای کاربر ارسال ممیشه که میگیم نظرتو در مورد این سفر بگو نظر سنجی سوالهاش و جواب هاش مشخصه چیزی که کاربر باید بزنه ی گزینه از بین چند تا گزینه هست در واقع اگر بخواهیم از پروتوتایپ استفاده کنیم باید از همه ی ابجکت ها یک کپی از نوع پروتوتایپ بگیریم و بفرستیم.

پروتوتایپ یعنی نمونه سازی

یعنی از نظر سنجی که مد نظرم هست یک نمونه دارم من میخوام بعد پایان هر سفر یک پروتوتایپی از نظر سنجی من ساخته بشه و به کاربر نشون بده

پترن های اسنراکچرال:

Adapter:

یعنی هماهنگی ورودی دوتا اینترفیسه یعنی انگار دو تا انیترفیس به هم متصل اند.

فرض کنیم خروجی یک دستگاهی میشه ورودی تک دستگاه دیگه اگر به هر دلیلی ورودی یا بدنه دیستگاه اولی آپدیت بشه دیگه این دوتا تابع با هم همانگ نیستند. برای این که همیشه این هماهنگی را بین توابع داشته باشیم بین دوتا تابع نیاز به آداپتر داریم که این دوتا رو با هم سازگار بکنه.

Decorator:

پترن دکورتور رو با اون دکورتوری که توی پایتون داریم اشتباه نگیرید.

پترن دکورتور :

الگو رو ما وقتی استفاده میکینیم که بخواهیم ی ویژگی رو به تابع یا به ابجکت اضافه کنیم بدون که به ماهیت اون تابع یا ابجکت آسیب بزنیم.

میدونیم که این دکورتوری ما ساختیم در قسمت های مختلف برنامه داره اجرا میشه حالا ما ی جا میخواهیم شکل این کاره عوض شه به خاطر اون ی جا ما نمیتونیم ماهیت ابجکتی که داریم یا تابعی که داریم روتغییر بدیم برای همین ما میاییم از پترن دکورتور استفاده میکنیم. یعنی میگیم اون تابع یا ابجکت کارش رو انجام بده هر موقع خواستم ماهیتش عوض بشه دکوورتش میکنم یعنی ی امکانی بهش اضافه میکنم.

\*\*تفاوت Adaptor با Decorator اینه که اولی بین دوتا اینترفیس هست دکورتور لزوما بین دوتا اینترفیس نبود یدونه ابجکت یا یدونه اینترفیس هم میشه دکوریت زد.

Proxy:

از پراکسی برای دسترسی استفاده میکنیم. تنها تفاوتش با دکورتور همینه دکورتور میاد ی قابلیتی رو اضافه میکنه ولی پراکسی میاد چک میکنه ما برای اضافه کردن ی چیزی به یک تابع به ان دسترسی داریم یا نه.

ی نمونه ترکیبی هم بگم

جنگو تمام عملیات های دسترسی مثل اجازه دادن به کاربر رو با پروکسی نوشته ولی با دکورتور صدا زده میشه.

دومدل پراستفاده رو میگیم.

\_\_getarrt\_\_ برای وقتیه که ما ی متودی رو صدا زدیم و هنوز مفسر پایتون نرفته سراغ اینکه ببینه این کلاسی که ما داریم اصلا این متود رو داره یا نه

\_\_getattribute\_\_ برای وقتیه که مفسر پایتون اون متودی که مادنبالش میکردیم رو پیدا نکرد