## **الگوی طراحی یک راه حل کلی قابل تکرار برای یک مشکل معمول در طراحی نرم افزار است. الگوی طراحی یک طرح تمام شده نیست که بتواند مستقیماً به کد تبدیل شود بلکه توضیحات یا الگویی برای نحوه حل مسئله است که می تواند در شرایط مختلف مورد استفاده قرار گیرد. الگوهای طراحی می توانند با ارائه پارادایم های آزمایش شده و اثبات شده توسعه ، روند توسعه را تسریع کنند. طراحی موثر نرم افزار مستلزم در نظر گرفتن مواردی است که ممکن است در مراحل ابتدایی قابل مشاهده نباشد. استفاده مجدد از الگوهای طراحی کمک می کند تا از ایجاد شدن مسائل ظریفی که می توانند مشکلات اساسی ایجاد کنند جلوگیری شود. غالباً ، مردم فقط می دانند که چگونه برخی از تکنیک های طراحی نرم افزار را برای مشکلات خاص به کار بگیرند. به کارگیری این تکنیک ها برای طیف وسیعی از مشکلات دشوار است. الگوهای متداول طراحی می توانند با گذشت زمان بهبود یابند و آنها را نسبت به طرح های موقت مقاوم تر سازد**

**به طور خلاصه الگوهای طراحی پاسخی به مشکلات متداول در تولید نرم‌افزار است**

الگوهای طراحی به ۳ بخش تقسیم می‌شوند که عبارت اند از :creational و structural و behavioral

نکته ی بسیار مهم این است که الگوهای طراحی قطعه کد نیستند که ما آن‌ها را حفظ کنیم . الگوهای طراحی مفهوم هستند که باید آن‌ها را درک کنیم

creational : الگوهای طراحی هستند که روی نحوه ی ایجاد شدن یک object از یک class تمرکز میکنند

========================================================================

در پایتون متد call به ما اجازه میده که نمونه هایی که از کلاس ها ایجاد میکنیم رو بتونیم مثل یک فانکشن صدا بزنیم به زبان دیگر هنگامیکه نمونه‌ای که از کلاس ساخته شده است به صورت فانکشن صدا زده شود این متد فراخوانی می‌شود ((قطعه کد call.py))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

در پایتون متد new  مسئول ساخت یک کلاس جدید هست. متد  new تنها متدیست که قبل از متد init  اجرا شده و میتواند مانع فعال شدن متد init شود به زبان دیگر اگر بخواهید نحوه ی ایجاد شدن کلاس را تغییر دهید از \_\_new\_\_ استفاده میکنید

خیلی جاها گفته شده که \_\_init\_\_ سازنده ی کلاس هست اما این جمله اشتباه میباشد چرا که سازنده ی کلاس \_\_new\_\_ میباشد .

\_\_init\_\_ فقط initializer (آغازگر) میباشد و کارهای اولیه را انجام میدهد

توجه شود که \_\_new\_\_ اولین مقداری که دریافت میکند cls میباشد یعنی خود کلاس بر خلاف \_\_init\_\_ که اولین مقداری که میگیرد self میباشد

دقت کنید که بعد از اینکه تغییراتی در \_\_init\_\_ به وجود آورید حتماً باید فانکشن new کلاس پدر را return کنید هماننده کد زیر :

return super().\_\_new\_\_(cls, \*args, \*\*kwargs)

به طور کلی نحوه ی کارکردن \_\_new\_\_ به این شکل میباشد:زمانی که نمونه‌ای از کلاس در حال ایجاد شدن میباشد \_\_new\_\_ سریع فعال می‌شود و کلاس خودش را به عنوان object به cls ارسال میکند ( به خودش ارسال میکند ) و در ادامه میتوانیم استفاده هایی را که مد نظر داشتیم را بکنیم

(( قطعه کد new.py ))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

متا کلاس ها در پایتون برای تغییر در نحوه رفتار دیگر کلاس ها به کار میرن

در بالا اشاره شد که کلاسی که هنوز ساخته نشده است خودش را به عنوان object به خودش ارسال میکند این موارد چگونه ممکن است به کمک meta class ها

هنگامیکه خروجی قطعه کد زیر را اجرا کنیم خروجی به این مفهوم میباشد که این کلاس از نوع type میباشد . این type همان typeی میباشد که مشخص میکند هر چیزی از چه نوعی میباشد :

a1.\_\_class\_\_.\_\_class\_\_

<class ‘type’>

print(type(213))

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

type به 2 شکل زیر استفاده می‌شود که با استفاده از دستور help(type) قابل مشاهده میباشد:

type(object) -> the object’s type

type(name, base, dict) -> a new type

حالت دوم به این معنی میباشد که با استفاده از type میتوانیم کلاس جدید ساخته می‌شود به این صورت که name اسم کلاس جدیدمان میباشد , base یک تاپل میباشد که قرار است این کلاس جدید از آن‌ها ارث بری میکند , dict فانکشن هایی است که قرار است این کلاس جدیدمان داشته باشد

متد type بالاترین و قویترین meta class ی است که در پایتون وجود دارد

تمامی کلاس‌هایی رو که ایجاد میکنیم به صورت پیش‌فرض type را به عنوان meta class خودشون دارند و برای ساخته شدن از این meta class استفاده میکنند و برای ساخته شدنشون از type استفاده میکنند برای همین است که در \_\_new\_\_ میتوانیم خوده کلاس را به clsی که ما فکر میکنیم هنوز ساخته نشده است ارسال کنیم که در‌واقع در پس زمینه type کلاس را میسازد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ما میتوانیم meta classهای خود را ایجاد کنیم کافیست meta class ما از type ارث بری کند و برای استفاده به کلاس مورد نظر اینگونه استفاده می‌شود :

class MyMetaClass(type): ......

class one (metaclass = MyMetaClass): .....

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

در قطعه کد metaclass.py کلاسی به عنوان metaclass میسازیم که باعث می‌شود که از کلاس db فقط و فقط یک نمونه ساخته شود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

دیزاین پترن singleton به ما این امکان رو میده که بتونیم کلاس ها رو جوری بسازیم که از اون کلاس فقط بشه یک نمونه ساخت.و اگر نمونه ی دیگری ساخته شود همان نمونه ی اول را برمیگرداند

خوده پایتون هم از singleton استفاده میکند . زمانی که دوباره یک import خاصی انجام می‌شود

(( قطعه singleton.py ))

========================================================================