به نام خدا

استاد : جناب آقای یاریان

دانشجو : امیر جلوداریان

01221033720015

اصول solid , dry , kiss , yangi

تفاوت deconstructor , constructor

Gc.collect

SOLID

(سالید) یک کلمه مخفف برای 5 اصل هست. هدف معرفی این اصول اینه که برنامه‌ها قابل درک‌تر، انعطاف‌پذیر تر و بیشتر قابل نگهداری باشن. به عنوان یک برنامه‌نویس، توسعه‌دهنده و مهندس نرم‌افزار، یادگیری این پنج اصل جزو "باید" ها هست. این اصول میتونن توی هر طراحی شی‌گرایی اعمال بشن.

KISS

“Keep It Simple, Stupid”

اصل KISS در برنامه نویسی بسیار اهمیت دارد.سعی کنید این اصل را سعی کنید به خاطر بسپارید و برای حفظ آن تلاش کنید.هرچقدر کد ساده تر باشد نگهداری آن در آینده ساده تر است.افراد دیگری که بخواهند کد شما را مورد ارزیابی قرار دهند در آینده با استقبال بیشتری این کار را انجام میدهند.اصل KISS توسط Kelly Johnson پایه گذاری شده است و سیستم های خوب به جای پیچیدگی، به سمت ساده سازی پیش میروند.از این رو سادگی، کلید طلایی طراحی است و باید از پیچیدگی های غیرضروری دوری کرد.

YAGNI

“You Aren’t Gonna Need It”

گاهی اوقات تیم های توسعه و برنامه نویسان در مسیر پروژه تمرکز خود را بر روی قابلیت های اضافه ی پروژه که "فقط الان به آن نیاز دارند" یا "در نهایت به آن نیاز پیدا میکنند" میگذارند.در یک کلام :اشتباه است!در اکثر مواقع شما به آن نیاز پیدا ندارید و نخواهید داشت. "شما به آن نیازی ندارید".

اصل YAGNI قبل از کدنویسی بی انتها و بر پایه ی مفهموم "آیا ساده ترین چیزی است که می تواند احتمالا کار کند" قرار دارد.حتی اگر YAGNI را جزوی از کدنویسی بی انتها بدانیم، بر روی تمام روش ها و فرآیند های توسعه قابل اجرا است.با پیاده سازی ایده‌ی "شما به آن نیازی ندارید" میتوان از هدر رفتن وقت جلوگیری کرد و تنها رو به جلو و در مسیر پروژه پیش رفت.

هر زمان اضطراب ناشناخته ای در کد حس کردید نشانه ی یک امکان اضافی بدون مصرف در این زمان است.احتمالا شما فکر میکنید یک زمانی این امکان اضافی را نیاز دارید.آرامش خود را حفظ کنید! و تنها به کارهای موردنیاز پروژه در این لحظه نگاه کنید. شما نمیتوانید زمان خود را صرف بررسی آن امکان اضافی کنید چون در نهایت مجبور به تغییر، حذف یا احتمالا پذیرفتن هستید ولی در نهایت جزو امکانات اصلی محصول شما نیست.

DRY

“Don’t Repeat Yourself”

اصل DRY توسط Andrew Hunt و David Thomas در کتاب The Pragmatic Programmer پایه گذاری ده است.خلاصه ی این کتاب به این موضوع اشاره میکنید که **"هر بخش از دانش شما در پروژه باید یک مرجع معتبر، یکپارچه و منحصربفرد داشته باشد"**. به عبارت دیگر شما باید سعی کنید رفتار سیستم را در یک بخش از کد مدیریت کنید.

از سوی دیگر زمانی که از اصل DRY پیروی نمیکنید، در حقیقت اصل WET که به معنای Write Everything Twice یا We Enjoy Typing دامن گیر شما شده است!(لذت بردن از وقت تلف کردن)

استفاده از اصل DRY در برنامه نویسی بسیار کارآمد است. مخصوصا در پروژه های بزرگ که کد دائما در حال نگهداری و توسعه است

ممکن است علاقه ای به رعایت اصل DRY نداشته باشید ولی اصول قبلی (KISS ,YAGNI) را حتما بخاطر بسپارید.

تفاوت constructor و deconstructor

منظور از Constructor نوع خاصی از متدها است که برای ارتباط با دیتابیس، تنظیم کوکی‌ها و یا اختصاص مقدار اولیه به متغیرها و کارهایی از این دست کاربرد دارد که در زبان پی‌اچ‌پی از سه خصیصه‌ای برخوردارند که متدهای معمولی عاری از آن‌ها هستند که عبارتند از:

- این متد به صورت خودکار و به محض اینکه یک شیئ از روی کلاسی که این متد داخلش تعریف شده بسازیم، به اصطلاح Call می‌شود.
- داخل این نوع متدها به هیچ وجه نمی‌توان از دستور return استفاده کرد.

در مقابل Constructor، مفهوم دیگری داریم تحت عنوان Destroctor. گفتیم کانستراکتور زمانی فراخوانی می‌شود که یک شیئ از روی کلاسی ساخته می‌شود؛ اما در مقابل، دیستراکتور زمانی فراخوانی می‌شود که یک آبجکت یا بهتر بگوییم یک شیئ از بین می‌رود و از جمله مواقعی که یک شیئ ساخته شده از روی کلاسی از بین می‌رود (یا اصطلاحاً Destroy می‌شود) می‌توان به زمانی اشاره کرد که اسکریپت ما به اِتمام رسیده که بالتبع بخشی از حافظهٔ اِشغال‌شده توسط آن اسکریپت نیز آزاد می‌شود.

garbage Collector چیست؟

در یک تعریف عامیانه میتوان گفت Garbage Collector که از این پس آنرا با GC میشناسیم ، راهیست برای پس گرفتن حافظه از اشیائی که در حال حاضر در برنامه مان بلا استفاده هستند، اما جدا از این تعریف اجازه دهید کمی وارد ساختار ذخیره داده ها در حافظه شویم و در مقاله ای دیگر تعریف GC را بازتر کنیم .

از لحاظ تقسیم بندی زبانهای برنامه نویسی در لول پایین دو نوع را میتوان نام برد:

-Unmanaged

-Managed

زبانهای مدیریت نشده ! زبانهایی هستند که برنامه نویس بصورت دیتیل جدا از بحث بیزینس بایستی درگیر مدیریت بزرگترین چالش پروژه های نرم افزاری یعنی حافظه باشد ، پس بیشتر این زبانها برای کارهایی همچون بازی سازی،توسعه سیستم عامل و ... مورد استفاده قرار میگیرند، مدیریت حافظه ها بصورت دستی توسط برنامه نویس انجام می شود(برنامه نویس موظف به کنترل نشت های حافظه ای ، مدیریت حافظه ها و... میباشد)

در سوی دیگر زبانهای مدیریت شده همچون جاوا و سی شارپ رو داریم که هر کدام به لطف عملکردهایی که داخل هسته این زبانها (در مورد سی شارپ CLR) وجود دارد تا حد زیادی مدیریت حافظه را بر عهده میگیرند و برنامه نویس را درگیر جزئیات نمیکنند.

Heap و Stack

در سی شارپ با توجه به وجود دو نوع داده Value Type و Reference Type ، هر کدام از آنها موقع استفاده در قسمتی از حافظه قرار میگیرند برای Value Type ها این قسمت از حافظه Stack است ، ورود و خروج از Stack بصورت LIFO می باشد .