بسم الله الرحمن الرحیم

نام و نام خانوادگی : امیرحسین تقی زاده

رشته : فناوری اطلاعات و ارتباطات

موضوع : در مورد اصل KISS , YAGNI , DRY,Solid, Constructor,Destructor,GC.Collec

استاد : میثاق یاریان

DRY

در حوزه ی توسعه ی نرم افزار، اصول و قواعد بسیاری وجود دارد که گاها یکی از دیگری مهم‌تر جلوه می‌کند اما یکی از اساسی‌ترین قواعد برنامه نویسی، قانون DRY است که مخفف واژگان Don`t Repeat Yourself به معنی»دوباره کاری نکن «است
این قانون توسط دو توسعه‌دهنده به نام های Andy Hunt و Dave Thomas ابداع شد که بسیاری از دیزاین پترن های معروف برنامه نویسی، ریشه در این قانون دارند در واقع هر چقدر کد کمتری تولید کنیم در زمان و انرژی صرفه جویی کردیم. میزان زمان و هزینه‌های پشتیبانی خیلی کمتر میشه و مشکلات کمتری هم توی پروژه ایجاد میشه.

KISS

KISSمخفف شده کلمات Keep It Simple & Stupidاست اما در عین حال برخی بر این باورند که این اصطلاح مخفف واژگان Keep It Super Simpleاست. این سرواژه مخفف هرچه که باشد یک معنی بیشتر نمی‌دهد و آن هم اینکه سعی کنید تا حد ممکن چیزهایی که در اختیار سایرین قرار می‌دهید را ساده طراحی کنید تا ایشان برای استفاده از آن مجبور به فکر کردن نشوند .

توی این قاعده ساده گرایی و پرهیز از پیچیدگی حرف اول رو میزنه. هرچقدر که کارها به واحدهای کوچک با فرآیندهای ساده تبدیل بشه بازدهی افراد بیشتر میشه، میزان خطا کمتر میشه و دستیابی به موفقیت راحتتر انجام میشه.

methodهای کوچیک بنویسیم، هر متد فقط یه کار مشخص انجام بده و یه مشکل کوچیک رو حل کنه. برای فرآیندهای پیچیده یه متد دیگه بنویسیم که متدهای ساده (که حاوی یک usecase بودند) رو به ترتیب کنار هم قرار بده و نهایتا یه usecase پیچیده حل بشه.

YAGNI

“You Aren’t Gonna Need It”

اصل YAGNI ("شما به آن نیاز ندارید") یک تمرین در توسعه نرم افزار است که بیان می کند که ویژگی ها، فقط باید در صورت نیاز اضافه شوند .

این اصل به توسعه‌دهندگان کمک می‌کند تا از تلاش‌های بیهوده روی ویژگی‌هایی که فرض می‌شود در برخی مواقع مورد نیاز هستند اجتناب کنند. ایده این است که این فرض اغلب نادرست است. حتی اگر یک ویژگی در نهایت مطلوب باشد، باز هم ممکن است مشخص شود که پیاده سازی آن ضروری نیست. بحث این است که توسعه دهندگان زمان خود را برای ایجاد عناصر اضافی که ممکن است ضروری نباشند و مانع یا کند کردن روند توسعه هستند، تلف نکنند.

SOLID

SOLID مخفف پنج اصل بسیار مهم در مدیریت وابستگی Dependency)

یکی از مشکلاتی که طراحی نامناسب برنامه های شی گرا برای برنامه نویسان ایجاد می کند موضوع مدیریت وابستگی در اجزای برنامه می باشد. اگر این وابستگی به درستی مدیریت نشود مشکلاتی شبیه موارد زیر در برنامه ایجاد می شوند:

برنامه ی نوشته شده را نمی توان تغییر داد و یا قابلیت جدید اضافه کرد. دلیل آن هم این است که با ایجاد تغییر در قسمتی از برنامه، این تغییر به صورت آبشاری در بقیه ی قسمت ها منتشر می شود و مجبور خواهیم بود که قسمت های زیادی از برنامه را تغییر دهیم. یعنی برنامه به یک برنامه ی ثابت و غیر پیشرفت تبدیل می شود. (این مشکل را Rigidity می نامیم.)

تغییر دادن برنامه مشکل است و آن هم به این دلیل که با ایجاد تغییر در یک قسمت از برنامه، قسمت های دیگر برنامه از کار می افتند و دچار مشکل می شوند. این مشکل را Fragility می نامیم.

قابلیت استفاده مجدد از اجزای برنامه وجود ندارد. در واقع، قسمت های مجدد برنامه ی شی گرای شما آنچنان به هم وابستگی تو در تو دارند که به هیچ وجه نمی توانید یک قسمت را جدا کرده و در برنامه ی دیگری استفاده کنید. این مشکل را **Immobility** می نامیم. اصول **SOLID** که قصد رفع کردن این مشکلات و بسیاری مسائل گوناگون را دارد عبارت اند از:

- **Single Responsibility Principle**
- **Open-Closed Principle**
- **Liskov Substitution Principle**
- **Interface Segregation Principle**
- **Dependency Inversion Principle**

با کنار هم گذاشتن حرف اول هر کدام از این اصول کلمه ی **SOLID** ایجاد می شود. با در نظر گرفتن این پنج اصل و پیاده سازی آنها در برنامه های خود می توانید به یک طراحی شی گرا پاک و درست دست پیدا کنید.

S مخفف **Single responsibility principle** یا **SRP** به معنی اینکه هر کلاس بایستی فقط یک کار انجام دهد نه بیشتر, که در ادامه توضیح خواهم داد.

O مخفف **Open/closed principle** یا **OCP** به معنی اینکه کلاس ها جوری نوشته بشن که قابل گسترش باشند اما نیاز به تغییر نداشته باشند. در مطالب بعد بیشتر توضیح خواهم داد.

L مخفف **Liskov Substitution Principle** یا **LSP** به مفهوم اینکه هر کلاسی که از کلاس دیگر ارث بری میکند هرگز نباید رفتار کلاس والد را تغییر دهد.

I مخفف **Interface Segregation Principle** با **ISP** به مفهوم اینکه چند اینترفیس کوچک و خورد شده همیشه بهتر از یک اینترفیس کلی و بزرگ است.

D مخفف Dependency inversion principle یا DIP به معنی اینکه از اینترفیس ها به خوبی استفاده کن!

Constructor

در برنامه‌نویسی شی‌گرا(OOP) ، اصطلاح Constructor به متدی اشاره دارد که به محض ساخت یک آبجکت از روی کلاسی، به صورت خودکار فراخوانی می‌شود. در حقیقت یکی از وظایف اصلی کانستراکتور این است که آبجکت ساخته‌شده را برای استفاده آماده سازد.

در واقع، در برنامه‌نویسی شیی‌گرا هر زمانی که بخواهیم به محض ساخت یک آبجکت (شیی) از روی یک کلاس، تَسکی اول از همه و آن هم به صورت خودکار انجام شود، می‌بایست از مفهومی تحت عنوان کانستراکتور استفاده نماییم (همچنین یکی دیگر از کاربردهای کانستراکتور، مقداردهی کردن پراپرتی‌ها است.)

Destructor

تابع تخریب کننده یا Destructor در زبان #C، متدی هستش که در هنگام از بین رفتن یک شی از کلاس، اجرا میشه. زبان #C، یک زبان پاک کننده خودکار سیستم یا garbage collector است. به این معنی که اشیایی که دیگر در برنامه نیاز ندارید را جهت خالی کردن حافظه و آزاد نمودن سیستم، پاک می کند.

GC.Collec

جمع آوری زباله Garbage Collection)یا به‌طور مختصر GCیک ویژگی بازیابی حافظه است که در زبان‌های برنامه‌نویسی مانند C#و جاوا **تعبیه شده است GC به‌طور خودکار فضایی که دیگر مورد نیاز برنامه نیست را حذف می‌کند .جمع آوری زباله تضمین می‌کند که یک برنامه از میزان حافظه خود بیشتر استفاده نمی‌کند یا به نقطه‌ای نمی‌رسد که دیگر نتواند کار کند.**