به نام خدا

عرفان مالکی

IT کارشناسی

استاد میثاق یاریان

تمرین 5 سمت سرور

موضوع : **(deconstructor) (Solid) (Dry) (Kiss) (Yagni)**

(gc.collect) (constructor)

KISS ”Keep It Simple, Stupid“

این اصطلاح به مبحث تمرکز بر سادگی در طراحی نرم‌افزار اشاره دارد. معنی آن است که برنامه‌نویسان باید تا حد ممکن از پیچیدگی‌های غیرضروری خودداری کنند و سعی کنند کد خود را به سادگی و قابل فهم نگه دارند.

عناصر اصلی KISS عبارتند از:

1. **سادگی :(Simplicity)** برنامه‌نویسان باید از پیچیدگی‌های غیرضروری و تعقیدات اضافی خودداری کنند. طراحی‌ها و پیاده‌سازی‌ها باید به حداقل تعداد متغیرها، توابع و کلاس‌ها محدود شوند.
2. **استفاده از راه‌حل‌های ساده :(Simple Solutions)** برنامه‌نویسان باید تا حد امکان از راه‌حل‌های ساده و مستقیم استفاده کنند. در بسیاری از موارد، راه‌حل‌های ساده‌تر نه‌تنها بهتر قابل فهم هستند، بلکه باعث افزایش قابلیت اطمینان کد نیز می‌شوند.
3. **خودداری از اضافه‌کردن پیچیدگی بی‌فایده (Avoiding Unnecessary Complexity):** اگر یک ویژگی یا عملکرد نرم‌افزار بدون افزودن پیچیدگی به کد قابل پیاده‌سازی باشد، برنامه‌نویسان باید از اضافه کردن آن به کد خودداری کنند.
4. **قابلیت فهم :(Ease of Understanding)** کد باید به سادگی قابل فهم باشد. برنامه‌نویسان باید فکر کنند که کدی که می‌نویسند توسط دیگران به راحتی قابل فهم باشد، زیرا فهمیدن سریع کد به اشتباهات کمتر و اداره بهتر پروژه کمک می‌کند.

YAGNI "You Ain't Gonna Need It"

اصل "YAGNI" مخفف "You Ain't Gonna Need It" است و یک اصل مهم در مهندسی نرم‌افزار و توسعه نرم‌افزارهاست. این اصل به توسعه‌دهندگان توصیه می‌کند که فقط آن‌چه در حال حاضر نیاز دارند را پیاده‌سازی کنند و از اضافه کردن ویژگی‌ها یا کدهایی که در حال حاضر نیاز ندارند، خودداری کنند. به عبارت دیگر، به جای پیاده‌سازی قابلیت‌ها یا ویژگی‌هایی که ممکن است در آینده مفید باشند، تمرکز بر روی نیازها و اولویت‌های فعلی قرار گیرد.

برخی اصول اساسی مرتبط با YAGNI عبارتند از:

1. **پیاده‌سازی بر اساس نیازهای فعلی:**
   - برنامه‌نویسان باید فقط ویژگی‌ها و کدهایی را پیاده‌سازی کنند که در حال حاضر نیاز دارند، نه آنچه که ممکن است در آینده مورد نیاز باشد.
2. **اجتناب از اضافه‌کردن پیچیدگی غیرضروری:**
   - خودداری از اضافه کردن ویژگی‌ها یا کدهایی که برنامه در حال حاضر به آن‌ها احتیاج ندارد، باعث کاهش پیچیدگی کد و افزایش قابلیت فهم آن می‌شود.
3. **تاکید بر سادگی:**
   - مانند اصلKISS ، اصل YAGNI هم تاکید بر سادگی دارد. برنامه‌نویسان باید از اضافه کردن اجزاء یا ویژگی‌ها که در حال حاضر مورد نیاز نیستند، پرهیز کنند.
4. **استفاده از اطلاعات فعلی:**
   - تصمیم‌گیری بر اساس اطلاعات فعلی و وضعیت فعلی پروژه به جای پیش‌بینی‌های زیاد در آینده.

DRY "Don't Repeat Yourself"

اصل DRY مخفف عبارت "Don't Repeat Yourself" است و یک اصل اساسی در مهندسی نرم‌افزار است. این اصل به توسعه‌دهندگان توصیه می‌کند که تا حد امکان از تکرار کد خودداری کنند و قطعات کد مشترک را با استفاده از روش‌های تجزیه و تحلیل مناسب یا ایجاد توابع و کلاس‌های قابل استفاده مجدد پیاده‌سازی کنند.

مثال :فرض کنید یک پروژه دارید و در چندین قسمت از کد شما یک الگوی خاص را استفاده می‌کنید. به جای تکرار کردن کد مربوط به این الگو در هر قسمت، بهتر است یک تابع یا یک کلاس بنویسید که این الگو را پیاده‌سازی کند و در صورت نیاز، از آن در اکثر قسمت‌ها استفاده کنید. این اقدام باعث اجتناب از تکرار کد غیرضروری، کاهش خطاها و افزایش قابلیت نگهداری می‌شود.

SOLID

اصل SOLID یک مجموعه از پنج اصل مهم در زمینه طراحی شیءگرا در برنامه‌نویسی است. این اصول توسط رابرت مارتین (Robert C. Martin) ارائه شده‌اند و به عنوان یک راهنمایی برای طراحی ساختار کد به منظور افزایش انعطاف‌پذیری، قابلیت نگهداری، و توسعه‌پذیری برنامه مورد استفاده قرار می‌گیرند. این پنج اصل با حروف اول کلماتی که آنها را تشکیل می‌دهند، سازگار شده‌اند و به نام SOLID مشهور هستند.

1. **S - Single Responsibility Principle (SRP):**

- این اصل می‌گوید که یک کلاس باید فقط یک مسئولیت داشته باشد. به عبارت دیگر، یک کلاس باید تنها یک دلیل برای تغییر داشته باشد.

2. **O - Open/Closed Principle (OCP):**

- این اصل به این معناست که یک کلاس باید برای افزایش ویژگی‌ها قابل توسعه باشد، اما باید برای تغییرات در خود بسته باشد. به عبارت دیگر، می‌توانید به کد خود ویژگی‌های جدید اضافه کنید بدون اینکه کد قبلی را تغییر دهید.

3. **L - Liskov Substitution Principle (LSP):**

- این اصل تاکید دارد که یک شیء نبایدی که از یک کلاس به دیگری تبدیل می‌شود، رفتار کلاس اصلی را تغییر دهد. یعنی باید بتوانید هر جایی که یک شیء از یک نوع مشخص به کار می‌رود، از شیء نوع پایه آن نیز استفاده کرد.

4. **I - Interface Segregation Principle (ISP):**

- این اصل به این ایده اشاره دارد که یک کلاس نباید از ویژگی‌هایی که نیاز ندارد استفاده کند. در واقع، کلاس‌ها نباید به اجبار از ویژگی‌هایی که به آن‌ها نیاز ندارند، وابسته باشند.

5. **D - Dependency Inversion Principle (DIP):**

- این اصل می‌گوید که ما باید وابستگی‌ها را به یک کلاس انعطاف‌پذیر وابسته کنیم، نه به یک کلاس خاص. با استفاده از این اصل، ما می‌توانیم به سادگی تغییرات در کدها را اعمال کنیم و از تغییرات کمترین تأثیر ممکن برند.

GC Collection

GC (Garbage Collection) به مجموعه‌ای از فرآیندها در محیط‌های اجرایی زبان‌های برنامه‌نویسی اشاره دارد که به تشخیص و حذف اشیاء غیرقابل دسترس (garbage) در حافظه برنامه می‌پردازند. مهمترین هدف این فرآیند، آزادسازی منابع حافظه‌ای است که توسط اشیاء غیرقابل دسترس اشغال شده‌اند تا فضای حافظه خالی شود و برنامه به صورت بهینه‌تر اجرا شود.

برخی از مزایای GC عبارتند از:

1. **آسانی مدیریت حافظه :**برنامه‌نویسان نیازی به دسترسی دقیق و مدیریت دستی حافظه ندارند GC .به صورت خودکار اشیاء غیرقابل دسترس را شناسایی کرده و حذف می‌کند.

2. **کاهش خطاهای حافظه :**از آنجا که GC به صورت خودکار اجرا می‌شود، احتمال خطاهای حافظه مانند دسترسی به اشیاء حذف شده یا حافظه تخصیص داده نشده کاهش می‌یابد.
3. **افزایش بهینگی :**حذف دستی اشیاء غیرقابل دسترس و تخصیص مجدد فضای حافظه به صورت خودکار باعث بهبود کارایی برنامه می‌شود.

Constructor

Constructor یک متد یا تابع ویژه است که در زمان ایجاد یک شیء از یک کلاس صدا زده می‌شود. عملکرد اصلی Constructor این است که مقادیر اولیه به ویژگی‌ها (متغیرها) را اختصاص دهد و هرگونه تنظیمات اولیه یا فعالیت‌های لازم را انجام دهد. اغلب Constructorها همان نامی دارند که نام کلاس است. در بسیاری از زبان‌های برنامه‌نویسی مانند Java، C++, C#, و Python، Constructor به عنوان متدی با نام کلاس و بدون نوع دیگر اعلام می‌شود.

Destructor

Destructorنقش معکوس Constructor را دارد و در زمان حذف یک شیء از حافظه (به عنوان مثال، زمانی که یک شیء از دامنه‌ی قابل دسترس خارج می‌شود) اجرا می‌شود Destructor .اغلب برای آزادسازی منابعی که در طول عمر شیء اختصاص داده شده‌اند (مانند حافظه‌های دینامیک، اتصالات پایگاه داده، و ...) استفاده می‌شود. در بسیاری از زبان‌های برنامه‌نویسی، نام Destructor با تعریف یک تابعی به نام ~ClassName اعلام می‌شود.