عنوان : اصول برنامه نویسی

استاد: جناب آقای دکتر میثاق یاریان

دانشجو: سیدمحمد موسوی مطلق

درس: برنامه نویسی سمت سرور

پاییز ۱۴۰۲

برنامه چیست؟

فرض کنید من ۱۰ عدد به شما می دهم و به شما می گویم میانگین ۱۰ عدد داده شده را پیدا کنید. خب، شما میانگین را چطور پیدا می کنید؟ شما تمام آن اعداد را جمع و سپس مجموع اعداد را بر تعداد اعداد داده شده تقسیم می کنید.

کار آسانی است اما اگر ۱۰ مجموعه داده شود که هر مجموعه شامل ۱۰ عدد باشد، چه کار می کنید؟

برای این مسئله دو راه حل وجود دارد:

۱-شما هر مجموعه را جدا جدا بگیرید، هر عدد را با مجموع قبلی جمع و سپس آن را بر تعداد کل تقسیم کنید. اگر بخواهید می توانید این مسئله را با این روش حل کنید، اما اگر قرار است یک مهندس نرم افزار باشید، حل مسئله از طریق این روش جالب نیست!

۲- یا برای حل مسئله یک برنامه بنویسید. فقط باید برنامه ای بنویسید که در آن کامپیوتر ورودی را از کاربر بگیرد و سپس از یک رویه برای یافتن میانگین استفاده کند.

اما معنای رویه چیست؟

ما یک رویه را با دادن مراحل یک به یک به کامپیوتر تعریف می کنیم و به آن برنامه می گوییم. به طور مثال گرفتن اعداد از یک کاربر، جمع کردن آن ها و تقسیم مجموع بر تعداد کل، یک رویه است.

به طور خلاصه، یک برنامه مجموعه ای از رویه یا دستورالعمل ها است.

روش های برنامه نویسی

تاکنون سه روش برنامه نویسی بیشتر از بقیه مورد استفاده قرار گرفته اند که عبارتند از:

۱-برنامه نویسی یک پارچه

۲- برنامه نویسی ماژولار / رویه‌ای

۳- برنامه نویسی شی‌گرا

برنامه نویسی یکپارچه :(Monolithic) این متد زمانی که برنامه نویسی به تازگی معرفی شده بود، به کار برده می‌شد. در برنامه نویسی یکپارچه، همه چیز از کد، داده و دستورالعمل در یک فایل واحد هستند که بررسی کد را دشوار می‌کند. از کد هم نمی توان دوباره استفاده کرد.

برنامه نویسی ماژولار یا رویه‌ای :(Procedural) برنامه نویسی رویه ای به تقسیم کار بین اعضای یک تیم کمک می کند و امکان استفاده از از تابع را هم می دهد. در این صورت امکان استفاده مجدد از کد فراهم خواهد شد. داده ها و تابع به طور جداگانه استفاده می شوند.

برنامه نویسی شی‌گرا :(OOP) این متد به طور گسترده در صنایع مختلف استفاده می شوند، ما داده ها و توابع را با هم می گیریم و آن ها را به صورت یک کلاس درمی آوریم.

در همه این روش های برنامه نویسی، کد ها و منطق ثابت می مانند، اما چیزی که تغییر می کند این است که نحوه سازماندهی کد توسط شما است.

۶ اصل مهم برنامه نویسی

۱- اصل KISS:

هیچ کس در برنامه نویسی دوست ندارد مرتبا درگیر رفع باگ و یا ایجاد تغییر در کد های پیچیده باشد. پس توصیه می شود که اصل KISS یا keep It Simple, Stupid را در ذهن داشته باشید. بیشتر سیستم ها در صورتی بهترین کارکرد را دارند که کد ساده باشد تا پیچیده. بنابراین وقتی در حال نوشتن کد هستید الگوریتم تان نباید طوری باشد که درک آن زمان و انرژی زیادی از کامپیوتر بگیرد. اگر کد شما ساده باشد، سایر توسعه دهندگان با مشکلی در درک منطق کد مواجه نخواهند شد و به راحتی می توانند روی آن کار کنند. پس همیشه سعی کنید کدتان را با استفاده از رویکرد های مختلف مثل تقسیم یک مسئله پیچیده به تکه های کوچک تر یا حذف کد های غیر ضروری که نوشته اید، ساده کنید.

۲- اصل DRY:

تکرار داده یا منطق در یک کد نه تنها باعث طولانی شدن برنامه شما می شود، بلکه زمان زیادی را برای کنترل، رفع باگ یا اصلاح کد هدر می دهد. هدف اصلی اصل DRY یا Don’t Repeat Yourselfکاهش تکرار کد است. بر اساس این قاعده یک قطعه کد باید فقط در یک مکان از کد منبع پیاده سازی شود. نقطه مقابل این اصل اینست WET (Waste Everyone’s Time)که یعنی همه چیز را دوبار بنویس! اگر منطق یکسانی را در چندین مکان پیاده سازی کنید، اصل عدم تکرار کد را زیر پا می گذارید. می توانید یک تابع مشترک ایجاد یا کدتان را انتزاعی کنید تا از تکرار جلوگیری شود.

۳- اصل YAGNI :

اگر در حال نوشتن کدی باشید که ممکن است در آینده به آن نیاز داشته باشید و نه الان، نرم افزار یا برنامه تان می تواند بزرگ تر و پیچیده تر شود. اصل YAGNI (You Aren't Gonna Need IT) که یعنی "قرار نیست به آن نیاز پیدا کنید " را همیشه در ذهن تان داشته باشید. در واقع این قاعده می گوید تا زمانی که چیزی لازم نیست، آن را پیاده سازی نکنید زیرا در بیشتر موارد (در آینده) از آن قطعه کد استفاده نمی کنید. اکثر برنامه نویسان در حین پیاده سازی نرم افزار به آینده فکر می کنند و برای برخی از ویژگی هایی که شاید در آینده لازم شوند، چند خط کد یا منطق به برنامه اضافه می کنند. آن ها کلی کلاس و عملکرد غیر ضروری اضافه می کنند که ممکن است در آینده هیچ وقت از آن ها استفاده نکنند. انجام این کار کاملا اشتباه است و در نهایت به حجیم شدن کد ختم می شوید، همچنین پروژه تان را پیچیده و کنترلش را دشوارتر می کند. به همه برنامه نویسان توصیه می کنیم که از این اشتباه اجتناب کنند تا زمان و انرژی شان هدر نرود.

٤- اصل SOLID:

این مورد مخفف پنج اصل است : مسئولیت واحد(Single responsibility) ، باز- بسته (Open-closed)، جایگزینی لیسکو(Liskov substitution) ، جدا سازی رابط (Interface Segregation) و معکوس سازی وابستگی (Dependency inversion) این اصول توسط رابرت سی مارتین ارائه شده است

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.

سالید بر پایه پنج اصل زیر هست.

۱ . اصل تک مسئولیتی(Single Responsibility Principle)

هر کلاسی که توی برنامه‌ی ما وجود داره، باید یک مسئولیت خاص و مشخص داشته. در واقع این کلاس باید فقط و فقط مسئول یک عملکرد توی برنامه باشه.

۲ . اصل باز - بسته(Open/Closed Principle)

موجودیت‌های یک نرم‌افزار (کلاس‌ها، ماژول‌ها، توابع و ...) باید برای توسعه داده شدن، باز و برای تغییر دادن، بسته باشن

۳ . اصل جایگزینی لیسکوف(Liskov Substitution Principle)

اگر S یک زیر کلاس T باشه، آبجکت‌های نوع T باید بتونن بدون تغییر دادن کد برنامه، با آبجکت‌های نوع S جایگزین بشن.

به بیان ساده‌تر کلاس‌های فرزند نباید رفتار و ویژگی‌های کلاس والد رو تغییر بدن

۴ .اصل جداسازی اینترفیس‌ها(Interface Segregation Principle)

کلاس‌ها نباید مجبور باشن متدهایی که به اونها احتیاجی ندارن رو پیاده‌سازی کنن.

در واقع این اصل میگه که ما باید اینترفیس (Interface) ها رو جوری بنویسیم که وقتی یک کلاس از اون استفاده میکنه، مجبور نباشه متدهایی که لازم نداره رو پیاده‌سازی کنه.

۵ . اصل وارونگی وابستگی(Dependency Inversion Principle)

کلاس‌های سطح بالا نباید به کلاس‌های سطح پایین وابسته باشن؛ هر دو باید وابسته به انتزاع (Abstractions)باشن. موارد انتزاعی نباید وابسته به جزییات باشن. جزییات باید وابسته به انتزاع باشن

باید بدونیم که:

اکثر الگوهای طراحی (Design Patterns) که وجود دارن، تلاش میکنن اصول سالید رو پیاده‌سازی کنن. مخصوصا اصل اول و دوم.

برنامه‌های خیلی کمی وجود دارن که همه‌ی این ۵ اصل رو همزمان پیاده‌سازی کرده باشن.

مثل دنیای واقعی، رعایت کردن همه اصول غیر ممکن هست.

اعمال کردن هر اصل باید با چشم باز انجام بگیره. وگرنه باعث میشه مشکل پیچیده‌تر بشه.

۵- تفکیک نگرانی ها (Separation of Concerns):

اصل تفکیک نگرانی ها یا به اختصارSoC ، یک برنامه پیچیده را به بخش ها یا دامنه های مختلف تقسیم می کند. هر بخش یا دامنه به مسئله جداگانه ای می پردازد یا وظیفه خاصی دارد. هر بخش مستقل از دیگری است و به همین دلیل است که هر بخش را می توان به طور مستقل بررسی کرد. بعلاوه کنترل، به روزرسانی و استفاده مجدد از کد آسان تر می شود.

به طور مثال منطق تجاری (محتوای صفحه وب) در یک اپلیکیشن یک نگرانی متفاوت است و رابط کاربری یک نگرانی متفاوت در یک وب اپلیکیشن است. یکی از نمونه های خوب SoC الگویMVC است که در آن داده ها (“مدل”)، منطق (“کنترل کننده”) و آنچه کاربر نهایی می بیند (“نما”) به سه بخش مختلف تقسیم و هر قسمت به طور مستقل مدیریت می شود . ذخیره داده ها در پایگاه داده هیچ ارتباطی با ارائه داده ها در وب ندارد.

۶- اصل اجتناب از بهینه سازی زود هنگام کد(Avoid Premature Optimization) :

بهینه سازی در واقع به سرعت بخشیدن به برنامه یا الگوریتم کمک می کند ، اما طبق این اصل، شما نیازی به بهینه سازی الگوریتم تان در مراحل اولیه توسعه برنامه ندارید. اگر زودتر از موعد به سراغ بهینه سازی بروید، نمی توانید تشخیص بدهید که نقطه ضعف های یک برنامه کجا هستند و کنترل برنامه برای تان سخت تر می شود. اگر کدتان را از همان اول بهینه سازی کنید، در صورت تغییر شرایط، زحمات تان هدر می رود و کد شما به هیچ دردی نخواهد خورد. پس بهتر است الگوریتم را زمانی بهینه کنید که به دردتان بخورد.