**SOLID**

**Solid** مجموعه ای اصول و قوانین است که با رعایت آن ها برنامه قابل نگه داری و قابل توسعه و قابل فهم می شود.

اصل اول:

: Single Responsibility Principle

هر بخش از فقط باید مسئولیت یک وظیفه را بر عهده دارد.یک کد ار بخش های مختلفی تشکیل شده است، هر بخش یک و فقط یک مسئولیت دارد و با کنار هم قرار کرفتن این بخش ها، کد نهایی ساده تر و مرتب تر خواهد بود.

اصل دوم:

: Open-Closed Principle

برنامه باید برای توسعه باز باشد.یعنی بشود عملکرد آن را تغییر داد و ویژگی های جدیدی به آن اضافه کرد. اما برای تغییرات خارجی بسته بماند. تغییرات خارجی تغییراتی هستند که در برنامه اعمال می شوند و منجر به تغییر رفتار برنامه در سمت کاربر با سایر سیستم ها می شوند. با این کار برنامه در عین حال پایداری و عملکرد صحیح خود را حفظ خواهد کرد.

اصل سوم:

: Liskov Substitution Principle

اگر کلاسی از یک کلاس پدر ارث بری کند باید با آبجکت های کلاس والد قابل جایگزینی باشد.یعنی بتوان آن را به جای کلاس والد استفاده کرد بدوناینکه خطا یا تغییری در عملکذد برنامه ایجاد شود.

اصل چهارم:

: Interface Segregation Principle

باید رابط ها را جداگانه و بر اساس نیازمندی ها طراحی کنیم. این به بدان معناست که هر کلاس فقط به رابط های که برای انجام وظایف خود نیاز دارد وابسته است و بقیه بخش ها از این رابط ها استفاده نمی کنند.

این باعث افزایش سازگاری، کاهش وابستگی غیر ضروری و افزایش انعطاف پذیری سیستم می شود.

اصل پنجم :

: Dependency Inversion Principle

به طور ساده برنامه نویس و توسعه دهنده باید تمام تلاش خود را برای به حداقل رساندن وابستگی ها میان کلاس ها، ماژول ها و آبجکت های سطح بالا با ماژول های سطح پایین انجام دهد.

این کار باعث می شود که کلاس ها به صورت مستقل از یکدیگر قابل استفاده و توسعه باشند. به عبرات دیگر، هدف اصلی وارونگی وابستگی این است که برنامه نویس به گونه ای کد زنی کند که تغییرات در یک بخش از کد تاثیر کمتری روی بخش های دیگر داشته باشد.

**Object Oriented Programming**

**شی گرایی** یک مدل برنامه نویسی است که به جای تعریف توابع و منطق مختلف برای موجودیت های مختلف از اشیا و داده ها استفاده می کند.در این روش برنامه نویسی میتوان هر شی را یک مدل داده ای در نظر گرفت که دارای خصوصیت ها و ویژگی های مختلف است.

شی گرایی دارای چهار اصل می باشد.

اصل اول:

: Abstraction

پیچیدگی توجه به جزئیات را کم میکنیم و بیشتر به عملکرد توجه میکنیم. که این کار کد را ساده تر و خواناتر میکند.برای مثال خودرو و حیوان دو مفهوم انتزاعی هستند چون جزئیات آن ها مشخص نیست و یک موضوع کلی هستند، اما اگر به جای خودرو از پراید یا بنز و به جای حیوان از سگ یا گربه استفاده کنیم وارد جزئیات شده ایم.

**اما چرا از انتزاع (Abstraction) در شیءگرایی استفاده می‌کنیم؟** **اصل انتزاع** برای پنهان‌سازی جزئیاتی است که به آن نیاز نداریم.

اصل دوم:

: Encapsulation

کلیه ویژگی ها و رفتار های مربوط به یک موجودیت را در شی ای محصور میکند تا ویژگی (Property) هایی که مورد نیاز کاربر نیست و فقط برای اجرای فرایند داخلی است را پنهان کند.

**اما چرا از کپسوله سازی (Encapsulation) در شیءگرایی استفاده می‌کنیم؟** حفاظت از داده ها در برابر خرابی تصادفی،مشخص کردن قابلیت دسترسی هر یک از اعضای یک کلاس به کد خارج از کلاس(این کار با استفاده از ویژگی Access Modifier در سی شارپ قابل انجام است.)،انعطاف پذیری و گسترش و کاهش پیچیدگی کد،متصل بودن کمتر اشیا و در نتیجه بهبود قابلیت نگهداری کد

اصل سوم:

: Polymorphism

به بیان ساده، **Polymorphism در شیءگرایی** این قابلیت را برای یک کلاس فراهم می‌کند که چندین پیاده‌سازی با یک نام داشته باشد. به این ترتیب، می‌توان یک عملکرد را با استفاده از یک نام یکسان در چندین کلاس مختلف تعریف کرد و بسته به نوع شیءی که در زمان اجرا استفاده می‌شود، عملکرد متفاوتی را اجرا کرد. اصلی‌ترین مزیت polymorphism این است که کد قابل استفاده مجدد، قابلیت تعمیم‌پذیری و انعطاف‌پذیری را به ارث‌بری‌ها و ارتباطات بین کلاس‌ها می‌دهد.

**اما چرا از چندریختی (Polymorphism) در شیءگرایی استفاده می‌کنیم؟** با استفاده از Abstraction و Polymorphism رفتار های مختلف کلاس های مختلف را پیاده سازی کنیم و بدین ترتیب کد قابل استفاده مجدد بنویسیم

اصل چهارم:

: Inheritance

**شیءگرایی**، امکان ارث‌ بردن ویژگی‌ها و رفتارها از یک کلاس به کلاس دیگر را فراهم می‌کند. با استفاده از **ارث ‌بری**، کلاس فرزند می‌تواند خصوصیات کلاس والد را در خود داشته باشد.

**اما چرا از وراثت (Inheritance) در شیءگرایی استفاده می‌کنیم؟** از **ارث‌بری** می‌توان برای ساختاردهی منطقی‌تر و سازماندهی بهتر کلاس‌ها استفاده و کد قابل استفاده مجدد و کارایی بیشتری را فراهم کرد. همچنین، با استفاده از **ارث‌ بری** می‌توان روابط و ارتباطات بین کلاس‌ها را بهبود بخشید.

Design Patterns (GOF)

الگو های طراحی راه حل هایی برای مشکلات رایجی است که اغلب در طراحی نرم افزار اتفاق می افتد که شامل 3 نوع اصلی و 14 الگو رایج و اصلی است.

انواع الگو های طراحی:

1. سازنده: مکانیزم ایجاد شی را ایجاد میکند به نحوی که کد موجود قابل استفاده و انعطاف پذیر باشد.
2. ساختاری: مکانیزم ترکیب اشیا برای ایجاد یک ساختار منسجم را ایجاد میکند.
3. رفتاری :با الگوریتم ها و تقسیم وظایف بین اشیا سر و کار دارد.

الگو های طراحی سازنده:

1. : Factory Method یک رابط برای ساخت اشیا در یک ابر کلاس می کند در حالی که هر یک از زیر کلاس ها امکان تغییر خود طبق نیاز دارد. به عنوان مثال می توان به اپلیکیشن شرکت لجستیک اشاره کرد که در سال اول تاسیس خود دارای حمل و نقل از راه زمینی است در حالی که در سال دوم قصد افزودن راه دریایی را نیز دارد.در این مثال ابر کلاس حمل و نقل است و زیر کلاس ها راه دریایی و زمینی هستند.
2. : Abstract Factory میتوان یک گروه از اشیا که با هم رابطه دارند را بدون مشخص کردن پیاده سازی آن ها ساخت. در پیاده سازی یک فروشگاه اینترنتی محصولات x میتواند از چند دسته تشکیل شده باشد.به عنوان مثال محصول x میتواند از دسته x1 و همچنین دسته x1 خود میتواند از دسته x2 باشد همین طور الا آخر. در این صورت در این الگوی سازنده برای هر دسته رابطی(Interface) تعریف می شود و هر دسته والد وظیفه ساخت هر نوع از زیر دسته و محصول را دارد.
3. Builder: یک ساختار پیچیده میتواند به بخش های کوچکتر تقسیم شده و مرحله به مرحله ساخته شود. میتوان انواع و نمایش های مختلف یک شی را با کد های ساخت یکسان تولید کرد.
4. Prototype: بدون تغییری در ساختار کد، از شی مورد نظر کپی گرفته می شود.
5. Singleton: از هر کلاس فقط و فقط یک نمونه در سراسر پروژه تحویل می دهد.

الگو های طراحی ساختاری:

1. Adapter : به اشیا اجازه همکاری با رابط های ناسازگار را می دهد.فرض کنید سرویسی را فراخوانی میکنید که خروجی آن XML است. در صورتی که سرویس های شما ورودی JSON می پذیرد. در این صورت اشیا Adapter مانند یک تبدیل کننده خروجی و ورودی عمل کرده و سرویس ها را با هم سازگار می کند.
2. Bridge:
3. Composite:
4. : Decorator
5. : Facade
6. : Flyweight
7. : Proxy

الگو های طراحی رفتاری:

1. : Chain of Responsibility
2. : Command
3. : Iterator
4. : Mediator
5. : Memento
6. : Observer
7. : State
8. : Strategy
9. : Template Method
10. : Visitor

**Database**

**پایگاه داده های رابطه ای SQL (Structured Query Language) چیست؟**

Sql به خودی خود پایگاه داده نیست، بلکه زبانی است برای برقراری با پایگاه داده های رابطه ای. با استفاده از این زبان با سیستم مدیریت پایگاه داده های رابطه ای) (Relational Database management system با (RDBMS)ارتباط برقرار کرد. در این نوع پایگاه داده، داده ها به صورت ساختار یافته و سازماندهی شده در سطر و ستون مشخص قرار میگیرند و با جداول دیگر ارتباط دارد.

**پایگاه داده های رابطه ای NOSQL (Not Only Structured Query Language) چیست؟**

داده ها ساختار مشخصی ندارند و رابطه ای بین آن ها وجود ندارد. داده ها در این نوع پایگاه داده به صورت Collection که معادل جدول در SQL است ذخیره می شود و ردیف ها با نام Document که معادل Record در SQL است شناخته می شوند.

**مزایای SQL:**

1. داده ها ساختار مشخصی دارند که صحت داده ها را تامین می کند.
2. داشتن رابطه بین جداول مانع ایجاد داده های تکراری می شود.

**معایب SQL:**

1. انعطاف پذیری به دلیل ساختار از پیش تعریف شده در کم است.
2. پیچیدگی دستورات JOIN
3. مقیاس پذیری افقی(اضافه کردن سرور و تقسیم کار بین آن ها) در آن سخت است و اکثر مواقع مقیاس دهی به صورت عمودی(بالا بردن منابع سخت افزاری مانند CPU و.....) است و اگر داده ها زیاد شود به حد نصاب خواهد رسید.

**مزایای پایگاه داده های NOSQL**

1. انعطاف پذیری به دلیل عدم وجود ساختار از پیش تعیین شده
2. داده ها در قالبی ذخیره می شود که برنامه در ان نیاز دارد. بنابراین سرعت خواندن داده ها در این پایگاه داده بیشتر است.
3. مقیاس‌پذیری افقی (اضافه کردن سرورهای جدید و تقسیم کار بین آن‌ها) و عمودی (بالا بردن منابع سخت‌افزاری مثل CPU و...) به راحتی قابل انجام است بنابراین حد نصابی برای درخواست‌های پایگاه داده وجود ندارد.

**معایب پایگاه داده های NOSQL**

1. وجود داده های تکراری به دلیل نبودن رابطه بین داده ها
2. وجود داده‌های تکراری باعث می‌شوند که برای انجام عملیاتی مانند به‌روزرسانی مجبور شویم چندین دستور مختلف، برای چندین مقدار مختلف اجرا کنیم.

Bugs

1. باگ سینتکسی: اگر نوشتاری یک زبان برنامه نویسی را اشتباه بنویسیم، با باگ سینتکسی مواجه می‌شویم که معمولا با پیغام Syntax Error نمایان می‌شود.
2. باگ منطقی : مثال افتادن در حلقه بی نهایت
3. باگ محاسباتی: برای مثال یک عدد را تقسیم بر 0 کنیم (جواب بی‌نهایت) یا نتیجه عملیاتی مورد انتطار دارای خطا نباشد.
4. باگ منابع
5. باگ تیمی: این باگ معمولا در کد نویسی بین اعضای تیم برنامه نویسی اتفاق می‌افتد. برای مثال تداخل در نام‌گذاری متغیرها و... که باعث به وجود آمدن خطا یا باگ نرم‌افزاری می‌شود.
6. باگ تعامل :  این باگ زمانی اتفاق می‌افتد که یک نرم‌افزار در ارتباط با نرم‌افزار یا سخت‌افزار دیگر دچار خطا شود. برای مثال از یک API نادرست استفاده شود.
7. باگ چند رشته ای :  این باگ زمانی اتفاق می‌افتد که اجرای threadها دچار مشکل شوند. مثلا جایی که وظیفه A تا زمانی که وظیفه B تمام نشده باشد، ادامه نخواهد یافت، اما در عین حال، وظیفه B نمی‌تواند ادامه یابد تا زمانی که کار A تمام شود.