**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 از 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 عملکرد 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 عبارت 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 کلاینت با چندین system قرار است تعامل داشته باشد، در این صورت برای ساده سازی فرایند یک واسطه به میان می آید که همه این system ها را در خود جمع می کند تا به جای اینکه کلاینت چندین system را صدا بزند فقط با یک کلاس Facade سر و کار داشته باشد.
6. : Flyweight
7. : Proxy این الگو به شما امکان ایجاد جایگزین یا نگهدانده برای یک شی اصلی را فراهم کرده و دسترسی به شی اصلی را به وسیله ی آن کنترل میکن، تا بتوان قبل یا بعد از درخواست به شی اصلی، کاری را انجام دهد. وازه پروکسی به معنی نماینده یا واسط در اینجا شی پروکسی واسطی بین ما و شی اصلی یا نماینده ای برای دسترسی به شی اصلی می باشد.این الگو استقاده از یک واسطه به منطور جلوگیری از دسترسی مستقیم به شی اصلی و متمرکز کردن کنترل دسترسی می باشد. به صورتی که به وسیله ی آن میخواهیم چالش های دسترسی به اشیا برنامه را حل کنیم و در عین حال Interface کلاس اصلی را تغییر ندهیم، تا بتوانیم، با آن کلاس مانند قبل رفتار کنیم.

الگو های طراحی رفتاری:

1. : Chain of Responsibility
2. : Command
3. : Iterator
4. : Mediator برای تعامل بین کلاس ها،سرویس ها و یا ماژول های خود استفاده می شود به طوری که وابستگی بین ماژول ها را به حداقل میرساند.در این الگو فراخواننده ماژول یک پیام تحت عنوان Event توسط این الگو ارسال می کند و این الگو وظیفه جستجوی ماژول فراخوانده شده را بر عهده داد.در این صورت ماژول گیرنده و ماژول فرستنده Event هیچ وابستگی فیزیکی به یکدیگر ندارند.
5. : Memento برای ذخیره وضعیت اشیا و بازگشت به وضعیت قبلی یا بعدی به کار برده می شود.بنابراین میتوان برای عملیات Undo این الگو را پیاده نمود.نام دیگر این الگو Token می باشد. ذخیره اطلاعات توسط این الگو بسته به نیاز می تواند در حافظه اصلی باشد و یا در حافظه های جانبی مثل فایل و ......
6. : Observer الگوی ناظر یا همان Observer در یک شی به نام Subject فهرست وابستگی هایش را با نام Observers نگه می دارد و هرگونه تغییر در وضعیت خود را به طور خودکار و معمولا با صدا کردن یک تابع مشترک در ناظران به آن ها اطلاع رسانی می کند.برای بالابردن خوانایی برنامه و توسعه بهتر پروژه، ما از الگوی Observer استفاده می کنیم. این الگو به ما می گوید شی اصلی ما که قرار است کار اصلی را انجام بدهد به عنوان Subject و بقیه ی اشیا را به عنوان Observer در نظر بگیریم و وقتی شی اصلی ما کار خود را انجام داد به بقیه اشیا اعلام کند که آن ها نیز کار خود را انجام دهند.
7. : State رفتار این الگو به این شکل است که وقتی حالت داخلی یک شی تغییر می کند، با توجه به آن تغییر، رفتار برنامه خود را تغییر می دهد و این طور به نظر میرسد که شی، کلاس خود را تغییر داده است. این تغییر با فرخوانی حالت های از پیش تعریف شده درون الگو اتفاق می افتد.شاید درباره state Machine شنیده باشید.ماشین های State معمولا با عمگر های شرطی زیادی اجرا می شوند که با توجه به وضعیت فعلی شی رفتار مناسب را انتخاب می کنند.روش الگوی طراحی State روشی تمیز تر برای ماشین های State می باشد که می تواند رفتار خود را در زمان اجرا تغییر دهد، بدون این که تبدیل به عبارت بزرگ شرطی شود.
8. : Strategy این الگو شامل مجموعه ای از الگوریتم ها می باشد که برای یک عمل خاص مورد استفاده قرار می گیرد و هر کدام از آن ها نتایج یکسانی خواهد داشت. هر الگوریتم به صورت جداگانه در کلاسی پیاده سازی شده و کپسوله می شود که این امر امکان تعویض الگوریتم های مختلف را به ما خواهد داد. با استفاده از این الگو امکان تغییر دادن الگوریتم هنگام استفاده از آن ها و در زمان اجرا وجود خواهد داشت. از کاربرد های این الگوریتم میتوان به مواردی اشاره کرد که چند روش برای یک عمل خاص وجود دارد و همینطور مواردی که امکان انتخاب یک الگوریتم از بین چندین الگوریتم موجود توسط کاربر وجود دارد.
9. : Template Method Feature های مشترک بین کلاس در قالب یک ابر کلاس کنار هم جمع آوری می کند.ساختار کلی الگوریتم مد نظر و همچنین متد های مشترک مابین کلاس های مذکور را تنها یک کلاس به اصطلاح Template پیاده سازی کرد و هر کلاس فرزندی که از آن ارث بری کند باید متد ها و فیچر های مد نظر متناسب با نیاز خود را بیفزاید.به عبارتی ، یک اسکلت یا قالب اصلی وجود دارد که کلیه ساب کلاس ها از آن قالب تبعیت کرده اما در عین حال هر کدام کد های اختصاصی خود را دارند.
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.

**PrimaryKey چیست؟**

ترکیبی از فیلد ها است که به صورت منحصر به فرد یک ردیف را مشخص می کند.

**Unique Key چیست؟**

هر رکورد از پایگاه داده را از بقیه متمایز می کند.این ویزگی Unique بودن میتواند روی یک ستون یا مجموعه ای از ستون ها باشد.Primary Key به طور خودکار Unique key است. در هر جدول می توان چند Unique key تعریف کرد اما فقط یک Primary key می توان تعریف کرد.

**Foreign Key چیست؟**

کلید خارجی یک جدول دیگر است که میتوان کلید اصلی جداول دیگر باشد. که برای ارتباط بین جداول به کار می رود.

**Join چیست؟**

یک کلمه کلیدی است که برای پرس و جو از داده های بیشتر از یک جدول استفاده می شود که در این دستورات کلید اصلی و خارجی نقش مهمی را ایفا می کند.

**انواع Join**

: Inner join ردیف های مشترک جداول بر اساس فیلد های مشترک را بازیابی می کند.

: Full outer join همه ردیف های تمامی جداول را چه مطابقت داشته باشند چه نداشته باشند را بر میگرداند

: Right join مشابه inner join است با این تفاوت که رکورد های جدول سمت راست را بر میگرداند حتی اگر با جدول سمت چپ اشتراک نداشته باشد.

: Right outer join تمام سطرهای موجود در جدول سمت راست را که با جدول سمت چپ مطابقت داشته باشد یا حتی نداشته باشد را برمی‌گرداند.

: Left join مشابه inner join است با این تفاوت که رکورد های جدول سمت چپ را بر میگرداند حتی اگر با جدول سمت راست اشتراک نداشته باشد.

: Left outer join تمام سطرهای موجود در جدول سمت چپ را که با جدول سمت راست مطابقت داشته باشد یا حتی نداشته باشد را برمی‌گرداند.

: Full join همه سطر ها را از جدول سمت راست و چپ بر می گرداند.

: Self join برای جوین کردن یک جدول با خودش مورد استفاده قرار میگیرد.این کار از طریق Alias ها انجام میگیرد.در واقع این امکان را به ما می دهد تا داده های موجود در یک جدول را با داده های دیگر در همان جدول ترکیب و مقایسه کنیم.

**View چیست؟**

**Index چیست؟**

یک روش بهبود عملکرد است که امکان بازیابی سریع جداول را فراهم می کند.

**انواع Index ها:**

: Unique Index زمانی که این نوع ایندکس تعریف می شود به مقادیر تکراری اجازه ذخیره شدن را نمی دهد.هنگامی که کلید اصلی تعریف می شود، Unique Index به طور خودکار اعمال می شود.

: Clustered Index ترتیب فیزیکی رکورد ها را بر اساس مقادیر اصلی تغییر می دهد تا بازیابی مقادیر بر این اساس سریع تر گردد.هر جدول می تواند فقط یک Clustered Index داشته باشد.

: Non-Clustered Index ترتیب فیزیکی را تغییر نمی دهد و ترتیب منطقی داده ها را حفظ می کند.در هر جدول میتوان هر تعداد از این نوع ایندکس تعریف کرد.

**Stored Procedure چیست؟**

یک تابع است که شامل دستوراتی است که با هم ادغام می کنیم تا زمان و هرجا که نیاز شد اجرا کنیم.

**Trigger چیست؟**

کوئری هایی هستند که به صورت خودکار در پاسخ به یک رویداد در یک Table یا یک View در دیتابیس اجرا می شوند.

**انواع Trigger**

: DDL Trigger Data Definition Language هنگام ساخت و تغییر و حذف جدول و یا View فعال می شود.

DML Trigger : Data Manipulation Language هنگامی که عملیاتی نظیر Insert,update,delete انجام شود این تریگر فعال می شود.

Logon Trigger : هنگامی فعال می شود که یک سشن از کاربر ساخته شود یا به عبارت دیگر کاربر احراز هویت شود.

**Constraint چیست؟**

می توان برای اعمال محدودیت روی جدول هنگام ایجاد و یا ویرایش جدول استفاده کرد.Constraint ها در دیتابیس به صورت زیر هستند:

Not NULL, CHECK, DEFAULT, UNIQUE, PRIMARY KEY, FOREIGN KEY

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.

**Compiler and Interpreter**

کامپایلر و مفسر چیستند؟

مترجمی هستند برای ترجمه زبان های برنامه نویسی سطح بالا به زبان ماشین

ویژگی های کامپایلر:

1. کد ها قبل اجرا یک بار به زبان ماشین ترجمه می شوند ولی در اجرای این اتفاق نمی افتد چون یک بار ترجمه شده اند، اما اگر تغییری در کد ایجاد شود، کد دوباره از اول باید ترجمه شود.
2. یک فایل خروجی برای اجرا ساخته می شود.
3. سرعت اجرا به دلیل عدم نیاز به ترجمه دوباره بالا تر از مفسر است.
4. وابسته به سیستم عامل هستند و در همان سیستم عاملی که ترجمه شده اند اجرا می شوند.

زبان های کامپایلری: C#, C, C++, JAVA, GO

ویژگی های مفسر:

1. در هنگام اجرا کد خط به خط ترجمه و اجرا می شود.
2. فایل خروجی برای اجرا نمی دهد.
3. سرعت اجرا به دلیل نیاز به ترجمه دوباره پایین تر از کامپایلر است.هر بار باید خط به خط ترجمه شود.
4. وابسته به سیستم عامل نیستند.

زبان های مفسری: JavaScript, Python, PHP, Ruby

**Data Structure**

**ساختمان داده چیست؟**

تکنیکی برای ذخیره سازی داده ها برای استفاده کارآمد از آن هاست که به گونه ای طراحی می شوند که قابلیت اجرا با الگوریتم های مختلف را داشته باشند.ساختمان داده دو نوع خطی و غیر خطی دارد.

**ساختمان داده خطی**

ساختاری است که در آن هر عنصر به عنصر قبل و بعد خودش متصل است و به صورت خطی و متوالی در حافظه ذخیره می شوند. مثل Queue, Heap, Linked List, Array

**ساختمان داده غیر خطی**

عناصر به صورت متوالی به یکدیگر متصل نمیشوند و از یک سلسله مراتبی به نام پدر و فرزند استفاده می کنند. به عبارت دیگر داده ها توسط ایجاد رابطه با یکدیگر به هم متصل می شوند.در ساختمان داده خطی امکان اتصال هر داده با دوعنصر دیگر را داشت ولی در ساختمان داده غیر خطی امکان اتصال هر عنصر به بیش از دو عنصر دیگر امکان پذیر است.مثال Graph, Tree

**تفاوت های ساختمان داده های خطی و غیر خطی**

خطی با افزایش داده پیچیدگی زمانی افزایش می یابد ولی در غیر خطی ثابت می ماند. در خطی استفاده از حافظه غیر بهینه است ولی در غیر خطی بهینه است.در ساختار غیر خطی باچندین اجرا میتوان به تمام عناصر دسترسی داشت ولی در ساختار خطی تنها با یک اجرا میتوان به تمام عناصر دسترسی داشت.

**Array**

عناصری از یک نوع را در خود ذخیره می کند و به هر عنصر در هر موقعیت از آرایه یک مقدار مثبت به نام Index اختصاص می دهد که با استفاده از آن در آرایه قابل جستجو است.

**Stack**

این نوع ساختمان داده از قانون LIFO (Last In-First Out) برای ذخیره سازی پیروی میکند،یعنی آخرین عنصری که وارد پشته شود از همه زودتر حذف میگردد.عمل اضافه کردن(Push) یا حذف کردن(Pop) فقط از یک سمت آن صورت می گیرد.

**Queue**

این ساختار تقریبا شبیه به پشته است با این تفاوت که از قانون FIFO (First In – First Out) برای ذخیره سازی اطلاعات استفاده می کند.یعنی اولین عنصر ورودی اولین عنصر خروجی خواهد بود. کلمه Front به یک عنصر قبل از عنصر اول اشاره دارد و کلمه Rear به آخرین عنصر در این ساختار اشاره دارد.عملیات درج عنصر را Enqueue و عملیات حذف آن را Dequeue نام دارد.

**Linked List**

شامل دنباله ای از عناصر است که در این ساختار به نام Node شناخته می شود. هر Node شامل اشاره گر و آدرس Node کناری است، ممکن است Node ها و عناصر در این ساختار منحصر به فرد و یا تکراری باشند.

**Tree**

ساختاری است که شامل Node های به هم متصل از طریق یال ها هستند و یک ساختار سلسله مراتبی است که به صورت رابطه پدر فرزندی تعریف می شودوانواع درخت ها بر اساس ساختارشان عبارت اند از :

, Binary Tree, AVL درخت قرمز-سیاه

**Graph**

از تعدادی Node تشکیل شده است و عنصر یا داده را در خود ذخیره می کند و ارتباط میان آن ها با یال مشخص می گردد.تفاوت درخت با گراف اینست که گراف قانون مشخصی برای ارتباط میان Node ها ندارد.

**Hashmap**

از جفت مقادیر کلید و مقدار برای ذخیره سازی استفاده می کند. این ساختار با استفاده از یک تابع درهمسازی، یک داده را دریافت و آن را به یک کلید مناسب تبدیل می کند.پس میتوان با استفاده از یک جدول هش به هر یک از داده ها یک کلید متناظر یا آن را داد و به صورت کاراکتری داده ها را جستجو کرد.

**Algorithm**

**انواع مختلف الگوریتم:**

جستجو،مرتب سازی،رمزگذاری،حریصانه،Brute-Force،تصادفی،بازگشتی،هشینگ،تقسیم و غلبه،برنامه نویسی پویا

**رمزگذاری**

ویرایش اطلاعات به طوری که فقط اشخاص مجاز قادر به درک آن باشند.به عبارت دیگر تبدیل متن ساده به متن رمزی

**حریصانه**

معمولا در مسائلی استفاده می شود که بتوان داده ها را در قالب یک گراف نشان داد.هر قسمت بر اساس ارزش که دارد انتخاب می شود.راه حلی که بیشترین فایده را دارد به عنوان مرحله بعدی انتخاب می شود.

[**Brute-Force**](https://www.kaspersky.com/resource-center/definitions/brute-force-attack)

**تمام راه حل های ممکن برای یک مسئله را کورکورانه تکرار میکند یا به یک یا چند جواب برای تابع خود پیدا کند.**

**بازگشتی**

**تابعی به صورت مستقیم و یا غیر مستقیم خودش را تا زمانی شرط توقف اعمال نشود فراخوانی می کند.**

**هشینگ**

**داده ها را گرفته و با درهم کردن آن ها را به یک پیام یکنواخت تبدیل می کند.**

**تقسیم و غلبه**

**دارای سه مرحله زیر می باشد:**

**تقسیم: مسئله را به مسائل فرعی به روش بازگشتی تقسیم کنید**

**غلبه: مسائل فرعی را به صورت بازگشتی حل کنید اگر مسئله به اندازه کافی کوچک است آن را مستقیما حل کنید**

**ترکیب: راه حل های مسائل فرعی را که بخشی از فرآیند بازگشتی هستند برای حل مسئله واقعی ترکیب کنید.**

**برنامه نویسی پویا**

**از راه حل از قبل یافت شده برای جلوگیری از محاسبه تکراری همان قسمت از مسئله استفاده می کند.مشکل را به زیر مشکلات کوچکتر تقسیم می کند و آن ها را حل می کند.مثلا اعداد فیبوناچی(عدد جاری = مجموع دو عدد قبل)**

**FAQ**

**تفاوت‌های دو کلمه کلیدی ref و out در سی شارپ چیست؟**

ارسال پارامتر ها به متد در سی شارپ به سه صورت انجام میگیرد.

1. ارسال پارامتر عادی: در ورودی متد متغیر همراه با نوع خود تعریف می شود و تغییرات بدنه متد تغییری در آن ندارد.
2. ارسال پارامتر به صورت Ref: در این حالت آدرس متغیر ارسال می شود و هر تغییری در بدنه متد انجام شود، روی متغییر اصلی نیز انجام می شود، به این نوع پارا.
3. ارسال پارامتر به صورت Out: این پارامتر زمانی استفاده می شود که قصد ارسال اطلاعاتی به متد نداشته باشیم.بلکه میخواهیم اطلاعات از متد برگردد.

نکات تکمیلی:

آرگومان های out نیازی به مقداردهی اولیه ندارند در حالی که داخل متد آن را مقداردهی و برای خروجی از آن استفاده می کنیم.

آرگومان های ref  حتما باید مقدار دهی شوند و هنگامی که بخواهیم اطلاعاتی را به متد ارسال کنیم، پس از دستکاری شدن، اطلاعات برگشت داده شود.

**تفاوت بین Class و Struct در برنامه نویسی سی شارپ چیست؟**

کلاس از نوع Reference Type و Struct از نوع Value Type است. در Struct نمیتوان از Inheritance استفاده کرد.در نهایت Struct جایگزین سبک حجم برای کلاس است.

**تفاوت کلاس‌های Sealed و Partial در #C چیست؟**

کلاس های Sealed :جلو ارث بری از آن کلاس را میگیرد.

کلاس های Partial : با استفاده از این نوع کلاس میتوان چند کلاس هم نام هم ساخت و هنگام نمونه سازی از این نوع میتوان از تمام متد هایی که در همه نمونه کلاس هم نام ساخته شده استفاده کرد. استفاده از Partial چند فانون دارد:

1. همه Access Modifier هم باید یکسان باشند.
2. اگر یک کلاس از کلاس دیگری ارث بری کند دیگر Partial ها دیگر نمیتوانند از کلاس دیگری ارث بری کنند.
3. اگر یک Partial به صورت Sealed باشد، بقیه هم Sealed هستند.

**معماری چند لایه چیست؟**

این معماری از سه بخش منطقی تشکیل شده است که هر لایه فقط با لایه پایین تر از خودش کار میکند و هر لایه وظیفه خاص خود را دارد که در اکثر مواقع به صورت زیر نام گذاری می شوند:

لایه نمایش یا Presentation Layer

لایه تجاری یا Bussiness Layer

لایه دسترسی به منابع داده یا DataAccess Layer

**تفاوت معماری تمیز با پیازی چیست؟**

در معماری تمیز وابستگی بین کامپوننت ها یکنواخت است اما در پیازی به صورت سلسله مراتبی.

معماری تمیز از روش MVC و یا MVP برای اپلیکیشن استفاده می کند، در حالی که پیازی از ابزاری مانند IOCوDI برای پیاده سازی استفاده می کند.

**Dependency Injection چیست؟**

یک الگوی طراحی است که به جای اینکه کلاس به صورت مستقیم از وابستگی ها استفاده کند، وابستگی های موجود در یک کلاس را به صورت خارجی تزریق کند.این کار باعث جدا شدن مفهوم های مختلف و کاهش وابستگی میان کد ها می شودکه کمک میکند مدیریت کد و ایجاد تست های واحد (Unit tests )قوی تر می کند.

**Dependency Inversion Principle چیست؟**

ماژول ها و کلاس های بالا دستی نباید به کلاس های پایین دستی وابسته باشند. بلکه هر دو باید به Abstraction وابسته باشند و همچنین جزئیات باید به Abstraction وابسته باشند نه Abstraction به جزئیات.

**Inversion Of Control چیست؟**

بعضی از کار هایی که به صورت روتین در حال انجام می باشند را به یک ماژول جدا می سپاریم(ناقص)

**IOC Container چیست؟**

فریم ورکی برای ایجاد وابستگی و تزریق آن زمانی که نیاز دارید می باشد.به این معنی که شما نیازی نیست به صورت دستی این وابستگی را ایجاد و آن را تزریق کنید،بلکه این فرسم ورک این کار را برای شما انجام می دهد.

**Delegate در سی شارپ به چه معناست؟**

Data Type هایی هستند که می توانند یک متد را داخل خود ذخیره کنند. یک نوع داده Referance Type است که Signature یک متد را تعریف می کند.

**Expression Tree در سی شارپ به چه معناست؟**

یک ساختار داده ای درختی است به طوری که هر گره یک expression است.یک expression میتواند یک فراخوانی متد یا Method Call یا یک عملیات باینری مانند x>y باشد که با همکاری عبارت های لامبدا صورت می گیرد.

**منظور از توابع Boxing و Unboxing در سی شارپ چیست؟**

تبدیل یک مقدار(عددی،رشته،کارکتر،....) به یک شی را Boxing و تبدیل یک شی به یک مقدار را Unboxing گویند.

**تفاوت Array با ArrayList در سی شارپ چیست؟**

Array مجموعه ای از متغیر های مشابه به حساب می آید که تحت عنوان یک ویژگی مشترک کنار یکدیگر جمع شده اند.ArrayList مجموعه ای از اشیا است که امکان شاخص گذاری و ایندکس گذاری آن به صورت تک تک وجود دارد.علاوه بر آن ویژگی های مانند جستجو و مرتب سازی آیتم ها و افزودن و تخصیص حافظه پویا را داراست. اما از مهم ترین تفاوت آن ها می توان به موارد زیر اشاره کرد:

هنگام تعریف Array چون اندازه آن ثابت است پس تخصیص حافظه به صورت ثابت خواهد بود در صورتی که در arraylist میتوان تخصیص حافظه را به صورت پویا کاهش و یا افزایش داد.

تمام آیتم های Array یکسان هستند در صورتی که در Array List میتوان از هر نوع داده افزود.

Array در سی شارپ امکان پذیرش Null را ندارند ولی Array List مقدار Null را می پذیرد.

**تفاوت string builder و concat و $ و + چیست؟**

**تفاوت Abstract و Interface چیست؟**

Abstract Class زمانی استفاده می‌شود که بخواهیم دولوپرها را موظف سازیم تا در حین ارث‌بری **حتماً** متد خاصی را پیاده‌سازی کنند و این در حالی است که صرفاً در مورد نام متد اطمینان داریم و نمی‌دانیم الگوریتم داخلش به چه شکل باید باشد. زمانی که کلاسی از جنس اَبسترکت می‌نویسیم، موظف هستیم تا حداقل یک متد از جنس اَبسترکت نیز داخل آن کلاس تعریف نماییم و لازم به توضیح است که متدهای اَبسترکت فقط حاوی نام و احتمالاً آرگومان‌های ورودی هستند که با این تفاسیر هرگز قادر به ساخت آبجکت از روی کلاس‌های اَبسترکت نخواهیم بود بلکه برای ساخت یک آبجکت، ابتدا باید یک کلاس فرزند بسازیم که از کلاس اَبسترکت والد ارث‌بری کرده باشد سپس الگوریتم مد نظر خود را داخل کلاس فرزند پیاده‌سازی نماییم.

به نوعی می‌توان کلاس‌های اَبسترکت را به اینترفیس‌ها تشبیه کرد با این تفاوت که اینترفیس یک کلاس اَبسترکت است که کلیهٔ متدهایش اَبسترکت می‌باشند اما این در حالی است که در کلاس‌های اَبسترکت علاوه بر حداقل یک متد اَبسترکت، می‌توان متدهای معمولی نیز داشت.

**Thread چیست؟**