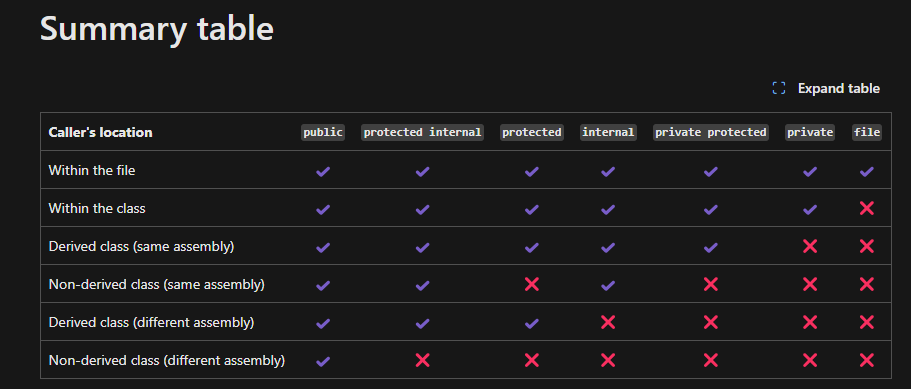
1. C# چیست و چگونه استفاده می‌شود؟  
   # یک زبان برنامه‌نویسی مدرن و شیء‌گرا است که توسط مایکروسافت به عنوان بخشی از ابتکار .NET توسعه یافته است. این زبان برای توسعه طیف گسترده‌ای از برنامه‌ها از جمله وب، موبایل، دسکتاپ و بازی‌ها طراحی شده است. C# قدرت C++ را با سادگی Visual Basic ترکیب کرده و شامل تایپ قوی، اصول دستوری، اعلانی، تابعی، عمومی، شیء‌گرا (بر پایه کلاس) و برنامه‌نویسی مولفه‌گرا می‌شود. C# به طور معمول برای موارد زیر استفاده می‌شود:
   1. ساخت برنامه‌های ویندوز با فریم‌ورک‌هایی مثل Windows Forms و WPF.
   2. توسعه برنامه‌ها و خدمات وب با استفاده از ASP.NET.
   3. ایجاد برنامه‌های موبایل با Xamarin.
   4. توسعه بازی با استفاده از Unity.
   5. برنامه‌های ابری با استفاده از Azure.
2. ویژگی‌های اصلی C# چیست؟  
   # دارای چندین ویژگی کلیدی است که آن را به یک زبان برنامه‌نویسی قدرتمند و منعطف تبدیل می‌کند:
   1. شیء‌گرا: پشتیبانی از اصولی مانند ارث‌بری، محصورسازی و چندریختی.
   2. تایپ قوی: اعمال ایمنی نوع (type-safety)، کاهش خطاهای زمان اجرا.
   3. مدیریت خودکار حافظه: استفاده از جمع‌آوری زباله (garbage collection) برای مدیریت تخصیص و آزادسازی حافظه.
   4. کتابخانه استاندارد غنی: ارائه مجموعه کاملی از کتابخانه‌ها برای وظایف رایج.
   5. نمایندگان و رویدادها: پشتیبانی از توابع کلاس اول و برنامه‌نویسی مبتنی بر رویداد (event).
   6. جنریک‌ها: امکان تعریف ساختارهای داده‌ای با ایمنی نوع.
   7. LINQ (زبان یکپارچه پرس و جو): امکان پرس و جوی داده با استفاده از نحو مشابه SQL.
   8. برنامه‌نویسی ناهمگام: پشتیبانی از کلمات کلیدی async و await برای برنامه‌نویسی ناهمگام.
   9. مدیریت استثناها: ارائه مدیریت خطاهای ساختاریافته با استفاده از بلوک‌های try، catch و finally.
3. تفاوت dotnet framework با dotnet core چیست؟
   1. اولی نسخه ابتدایی dotnet بود که عرضه شد و فقط قابلیت publish روی ویندوز را داشت اما دومی نسخه cross platform می باشد که خروجی برنامه های آنرا می توان روی هر سیستم عاملی اجرا کرد و محدود به فقط ویندوز دیگر نیست
   2. نسخه دوم open source است.
   3. نسخه دوم performance بالاتری به نسبت اولی دارد ( به این دلیل که از JIT استفاده می کند و JIT بر اساس پلتفرمی که روی آن در حال اجرا است عمل می کند همچنین استفاده از ahead of time compiler و استفاده بهتر از مموری)
   4. نسخه دوم ماژولار است یعنی میتواند روی هر سیستم عاملی نصب شود
   5. نسخه دوم از سیستم های cloud based، میکروسرویس ها و container ها پشتیبانی می کند.
   6. قابلیت side by side versioning در نسخه دوم که اجازه می دهد ورژن های مختلفی از دات نت روی یک اپ استفاده شود. (همزمان بتوان از نسخه های مختلف دات نت روی یک اپ استفاده و روی یک سیستم اجرا کرد.)
4. تفاوت class و object در چیست؟  
   object ها یک نوع در C# به شمار می روند که در واقع نوع پایه محسوب می شود. یعنی تمام کلاس ها و type های مختلف در C# قابلیت cast شدن به نوع object را دارا هستند. در نتیجه، هر کلاسی قابل تبدیل شدن به object را دارد.  
   class ها از سوی دیگر مانند یک قالب یا شکل هستند که میتوان با آنها type های مختلف را در برنامه ایجاد کرد. با class ها می توان type های جدید تعریف کرده و خصوصیت ها و کاربرد های مختلف به آنها نسبت داد..
5. نحوه تبدیل کد Csharp به کد ماشین چگونه است؟
   1. اول از همه کد برای compile error بررسی میشه که آیا error داریم یا نه.
   2. بعدش کد Csharp توسط کامپایلر C# به یه intermediate language (IL) تبدیل میشه که روی هر سیستمی قابل اجراس (دلیل اینکه Csharp، cross platform هست). در این مرحله metadata ها هم (مثل type ها، method ها و...) تولید میشن (این metadata س که امکان reflection رو فراهم میکنه).
   3. بعدش Common language runtime (CLR) میاد که اون IL رو لود کنه توی مموری اما اون IL به تنهایی قابل اجرا نیست.
   4. پس CLR میاد اون IL رو به کد ماشین تبدیل (کامپایل) کنه. این کار رو میگن Just-In-time compilation که توسط Just-in-time compiler انجام میشه. نکته ای که داره اینه که JIT داره در زمان execution اون کد، کار کامپایل رو هم انجام میده.
   5. بعد از JIT، یه تعداد assembly تولید میشه که فایل هایی هستن که شامل تعدادی namespace و class میشن. بسته به نوع پروژه این فایل ها یا exe یا dll هستن.
   6. در حین اجرا، CLR داره کارایی مثل garbage collection, memory management, exception handling,… رو هم انجام میده.
   7. غیر از JIT، یه نوع compilation دیگه هم داریم که بهش Ahead-of-time (AOT) میگن. این یکی میاد قبل از اینکه کد بخواد اجرا بشه کامپایل اش رو انجام میده. یعنی یه کامپایل کامل داریم و بعدش کد تولید شده میره واسه اجرا.
6. نحوه کار garbage collector؟  
   به طور کلی وظیفه GC اینه که آبجکت های بدون استفاده رو از مموری خارج کنه تا فضای مموری برای استفاده باز بشه. خوشبختانه چون GC اینکارو توی یه thread جدا گونه انجام میده، عملا تاثیر خاصی روی performance برنامه نمیذاره مگه شرایط خاصی مثل اینکه یهو مقدار زیادی از حافظه توسط GC، collect بشه. GC کارش رو توی سه مرحله انجام میده (که این مراحل به صورت اتوماتیک توسط خود GC انجام میشه مگه اینکه ما مستقیما و دستی با GC.Collect دستور بدیم)
   1. Marking: توی این مرحله رفرنس های موجود به آبجکت های توی heap بررسی میشه تا لیستی از آبجکت های زنده (live objects) بدست بیاد. هر آبجکتی که جزو این لیست نباشه میره برای پاکسازی و از heap پاک میشه.
   2. Relocating: توی این مرحله آبجکت های زنده به مکان های جدیدی relocate میشن و اشاره گر به اونها هم به محل جدید آپدیت میشه.
   3. Compacting: از اونجایی که قسمت هایی از heap توی مرحله اول پاکسازی میشه، بخش هایی از حافظه خالی میشن. توی این مرحله آبجکت های ما به سمت انتهای heap جابجا میشن و پشت سر هم قرار میگیرن تا دوباره heap از بالا بتونه پر بشه. (مثلا اگه heap 5 تا خونه حافظه باشه و ما توی مرحله اول خونه های 1 و 4 رو پاک کرده باشیم، خونه 2 میاد اول heap، خونه 3 میاد دوم و خونه 5 میاد سوم قرار میگیره و اگه آبجکت جدیدی بخواد به heap اضافه بشه، میاد توی خونه چهارم و همینطور تا سیکل بعدی GC پر میشه)  
      این وسط، GC میاد از یه مکانیزمی استفاده میکنه به اسم heap generation. این مکانیزم میاد heap رو به 3 تا نسل (generation) مختلف تقسیم میکنه که این نسل ها به نوعی نماینده lifetime های مختلف در برنامه ان؛ اینکه هر آبجکت چه مدتی زنده اس و میتونه در برنامه بمونه و استفاده بشه. میزان حافظه برای هر نسل توسط CLR در runtime تعیین میشه (گرچه حتما هر نسلی که آبجکت های با طول عمر بیشتر داشته باشه، مموری بیشتری نسبت به نسل با آبجکت های با طول عمر کمتر، داره). هر نسل به این صورته:
      1. Generation 0: تمام متغیر های موقتی، محلی و تازه ساخته شده توی این نسل قرار میگیرن. در واقع هر آبجکتی حداقل یه بار به این نسل سر میزنه. (newly allocated objects)
      2. Generation 1: اگه توی یه سیکل از GC، آبجکت هایی توی نسل صفر باشن که حذف نشدن و هنوز زنده ان، منتقل میشن به این نسل. این نسل یه جورایی مثل یه میانه رو بین آبجکت های با عمر کم (نسل صفر) و آبجکت های با عمر طولانی (نسل دوم) عمل میکنه.
      3. Generation 2: اینم مثل نسل قبل، زمانی که یه سیکل از GC تموم بشه، آبجکت هایی از نسل یک که هنوز زنده ان رو منتقل میکنه به این نسل. این نسل واسه آبجکت های با عمر طولانی هست مثل static object ها چون در کل طول عمر برنامه حضور دارن و باید در دسترس باشن.
7. تفاوت file.WriteAllText و StreamWriter؟
8. تفاوت interface با abstract؟
   1. کلاسی که abstract باشه نمیتونه شئ ازش ساخته بشه؛ فقط کلاسی که ازش ارث بری میکنه میتونه شئ بسازه. البته abstract class میتونه Ctor داشته باشه اما مستقیما نمیشه با این Ctor از abstract class شئ ساخت؛ این ctor با constructor chaining قابل استفاده اس یعنی کلاسی که از abstract class ارث بری میکنه میتونه داخل ctor اش از ctor مربوط به abstract class هم استفاده کنه. (در زمان ساخت شئ، ctor کلاس abstract زودتر از ctor کلاس فرزند اجرا میشه)
   2. اگه متد یا پراپرتی ای رو پشتش abstract بزاریم، دیگه نمیتونه توی اون کلاس، بدنه یا مقدار بگیره. این کارا باید توی کلاس فرزند اتفاق بیفته (و باید اتفاق بیفته).
   3. Abstract از multiple inheritance پشتیبانی نمیکنه، interface اما میتونه
   4. یه interface رو هم نمیشه ازش شئ ساخت.
   5. کلاسی که داره یه interface رو implement میکنه، باید تمامی متد های اون اینترفیس رو پیاده سازی کنه.
   6. قبلا داخل interface نمیتونستیم پیاده سازی داشته باشیم اما در نسخه های جدیدتر csharp این قابلیت هم اضافه شد. دلیلش این بود که توی پروژه هایی که بزرگ هستن، اگه متدی توی interface باشه که بعدا بفهمیم یه مفهوم و پیاده سازی مشترک بین همه کلاس هایی که از این اینترفیس استفاده کردن رو داره، بتونیم با یه تغییر کوچیک این متد رو برای همه اون کلاس ها به صورت یکسان پیاده کنیم. اگر به جای اینکار تصمیم بگیریم که interface رو به abstract تغییر بدیم هزینه تغییر به مراتب بیشتر میشه.
   7. Interface ها در runtime متد ها رو map میکنن اما abstract class ها در compile time. در نتیجه performance توی abstract class ها به نسبت interface ها (گرچه خیلی کم) بالاتره.
9. چرا default method interface ها را داریم؟
   1. یه مزیتشون اینه که وقتی توی خود اینترفیس داریم implementation انجام میدیم، دیگه در زمان استفاده نباید متغیر ما از نوع کلاسی باشه که داره این اینترفیس رو استفاده میکنه بلکه باید از نوع همون اینترفیس باشه.
   2. یه اینترفیس که داره از یه اینترفیس دیگه که default method داره، ارث بری میکنه، میتونه اون متد رو بازتعریف کنه.
10. روش های polymorphism در C#؟
    1. Override: اینو بهش dynamic / runtime-polymorphism هم میگن. توی این مدل، signature متد ها یکیه اما پیاده سازی متفاوته. همون داستان late binding هست.
    2. Overload: اینو بهش static/compile-time polymorphism هم میگن. این مدل، اسم متدها یکیه اما ورودی هاشون متفاوته (پیاده سازی هم میتونه متفاوت باشه که اکثرا هست). همون داستان early binding هست.
11. تفاوت بین early binding و late binding چیست؟  
    در early binding، کامپایلر در زمان کامپایل (compile time) میتواند تشخیص دهد که object مورد استفاده از چه نوعی بوده و چه method ها و property هایی دارد. مزیت این مدل binding در سرعت بالاتر برنامه و کاهش ارور ها در runtime است.  
    در late binding اما نوع object در زمان اجرا (runtime) مشخص می شود و تا زمان اجرا، نوع آن object مشخص نیست. این روش گرچه انعطاف پذیری برنامه را بالاتر می برد اما در عین حال سرعت اجرا را کمتر و امکان بروز ارور در کد آن هم در زمان اجرا را بالا می برد.
12. Covariance و contravariance؟
13. تفاوت == با Equals چیست؟
    1. در Equals فقط محتوای داخل متغیر و پارامتر ورودی آن، بررسی می شود اما == علاوه بر محتوا، reference را نیز چک می کند. این مورد در حالتی که هر دو یا یکی از دو متغیر از نوع object باشند به چشم می آید.
14. تفاوت override و overload چیست؟  
    در overload، امضا یا signature یک متد عوض می شود مانند input parameter ها. اما در override، نحوه پیاده سازی متد تغییر می کند. مثلا نحوه انجام Read() تغییر می کند (خواندن از console، فایل و...)
15. تفاوت destructor و deconstructor چیست؟
    1. اولی برای آزاد سازی منابع استفاده میشه. وقتی که شئ ای از یه کلاس داریم که دیگه باید پاک بشه و یا منابعی که دستش بوده (حافظه) آزاد بشن، از این استفاده میکنیم.
    2. از اون طرف، دومی برای استخراج منابع از شئ مورد نظر استفاده میشه و اصلا کاری به از بین بردن شئ مورد نظر نداره. دیتایی که داخل شئ هست رو از طریق این متد میتونیم به بیرون انتقال بدیم. هدف بعدی استفاده از deconstructor، ساده سازی و راحتی در کد نویسی در زمان استخراج دیتا از شئ (یا tuple) هست.
16. چرا object initializer ها از optional parameter ها بهترن؟
    1. گرچه دومی باعث میشد که فیلد های ما به صورت readonly در بیان (که در کل خوبه که اینطور باشه)، اما اولی هم اینکار رو میتونه بکنه (با کمک init).
    2. مشکل بعدی توی library هاس که استفاده از دومی باعث میشه library ما backward compatibility نداشته باشه چون که اگه نیاز به تغییرات توی مقداردهی اولیه به فیلد هامون داشته باشیم باید توی signature های constructor هامون هم دست ببریم که خب این باعث میشه نسخه های قدیم و جدید با هم سازگاری کمتری داشته باشن اما اولی اجازه میده که بتونیم بدون از بین بردن سازگاری، تغییرات رو روی نسخه جدید اعمال کنیم.
17. انواع Access modifier ها و میزان دسترسی هر کدام؟
    1. 
18. Primary constructor چیه؟
    1. این نوع ctor در صورتی که نوشته بشه برای یه کلاس، باعث میشه که یه ctor ای داشته باشیم که باید باقی ctor ها اونو call کنن.
19. Static constructor چیه؟
    1. این ctor، هر زمان که یه شئ میخواد ساخته بشه به صورت اتوماتیک اجرا میشه و میاد قبل از اون ctor دیگه ای که کال شده اجرا میشه.   
       اما اگه فیلد یا پراپرتی static داریم، اول مقداردهی اونا انجام میشه و بعد Static ctor اجرا میشه.
20. کار checked و unchecked چیست؟  
    برخی کد ها ممکن است در زمان کامپایل، ارور ندهند. به عنوان مثال overflow که در متغیر های عددی بیشتر اتفاق می افتد، در زمان کامپایل اروری به آن داده نمی شود و نیز در زمان اجرا نه تنها به excpetion نمی خورد بلکه یک مقدار متفاوت و نابرابر با مقدار واقعی را نمایش می دهد. با استفاده از اپراتور checked، اگر مواردی مثل overflow اتفاق بیفتد، آنگاه در runtime یک exception اتفاق افتاده و از ادامه برنامه به آن صورت جلوگیری می کند. اپراتور unchecked برعکس آن عمل کرده و کد در runtime دیگر exception بر نمی گرداند. تنظیماتی وجود دارد که میتوان پیش فرض بودن هر کدام از این حالات در کد را در آن تعیین کرد و سپس بر طبق آن از این اپراتور ها در کد استفاده نمود.
21. مزیت BitArray بر آرایه معمولی چیست؟  
    متغیر های از نوع bool گرچه باید فقط یک بیت حافظه اشغال کنند اما در واقعیت در runtime یک بایت را اشغال می کنند. اگه لیستی از چند متغیر یا مقدار bool بخواهیم داشته باشیم، استفاده از BitArray به جای آرایه عادی بهتر است چرا که در آن دقیقا به ازای هر خانه BitArray یک بیت از حافظه به جای یک بایت اشغال می شود.
22. تفاوت Jagged array با Rectangular array در چیست؟  
    در Rectangular، طول آرایه های داخلی یکسان است اما در Jagged طول آرایه های داخلی دلخواه است و در زمان declaration آن می توان این مقدار را به آن داد.
23. تفاوت readonly و const در چیست؟  
    فیلد های readonly در زمان ساخت یک instance از کلاس، در constructor قابل مقدار دهی هستند در نتیجه در runtime قابلیت پر شدن دارند. فیلد های const اما در زمان declaration باید مقداردهی شوند در نتیجه در زمان کامپایل باید پر بشوند. همچنین تنها type های bool, string, char, enum و تمام نوع های عددی در خود C# می توانند const باشند.
24. تفاوت init, readonly, required چیست؟
    1. با readonly، اون فیلد فقط میتونه یه بار مقدار بگیره اما اینکه مقدارشو به صورت default میگیره، از ctor میگیره یا از طریق object initializer، فرقی نمیکنه
    2. Init میاد میگه همون کار readonly رو انجام میدم اما باید از طریق object initializer باشه.
    3. Required میگه این فیلد یا پراپرتی باید مقدار بگیره و از طریق object initializer اما میتونه در آینده تغییر کنه.
    4. Init و required رو میشه با هم ترکیب کرد توی یه پراپرتی و نتیجه اینه که باید اون پراپرتی از طریق object initializer مقداردهی بشه و دیگه بعد از اون قابل تغییر نیست. (با این مشکل دارم چون تفاوتی با init خالی نداره)
25. تفاوت boxing و unboxing چیست؟  
    در boxing، یک value-type به reference-type تبدیل می شود و در unboxing، یک reference-type به value-type تبدیل می شود. همچنین در boxing، صرفا ریختن یک value-type در یک reference-type کافی است اما در unboxing باید حتما explicit cast انجام شود و در صورت ناموفق بودن، برنامه exception می دهد.
26. تفاوت typeof و GetType در چیست؟  
    هر دو type یک متغیر یا instance را به ما می دهند اما typeof در زمان کامپایل بررسی و انجام می شود و GetType در runtime.
27. تفاوت class و struct در چیست؟
    1. Class یک reference-type است و struct یک value-type
    2. Struct قابلیت ارث بری ندارد و abstract نیز نمی شود.
    3. Member ها در class به صورت پیش فرض private و در struct به صورت پیش فرض public اند.
28. Callback method چیست؟  
    در chatgpt هست
29. تفاوت object با Generic ها چیست؟  
    generic ها object های type-safe هستند چرا که نوع در زمان کامپایل مشخص شده و دیگر نگران ارور در runtime نخواهیم بود.
30. تفاوت Task با thread؟
    1. Task ها جزو TPL هستن اما thread مفهوم پایه ای تری هست
    2. با Task میشه خروجی برگردوند
    3. Task میتونه از Cancellation Token ها استفاده کنه
    4. Task میتونه چندتا process رو همزمان اجرا کنه
    5. Task ها شکل ساده تر و پیشرفته تر thread ها هستن.
31. تفاوت Task و ValueTask؟
    1. یه task آبجکت ها رو داخل heap میذاره اما ValueTask داخل stack
    2. در نتیجه نکته بالا، ValueTask پرفورمنس بهتری داره
    3. Task برای جاهایی بهتره که async داریم کار می کنیم و همون لحظه خروجی رو نمی خوایم اما ValueTask برای جایی که ممکنه خروجی به صورت sync هم در دسترس باشه بهتره
32. تفاوت Concurrency و parallelism؟
    1. در concurrency ما چندین تسک رو به صورت همزمان داریم اما لزوما همزمان اجرا نمیشن؛ صرفا توهم این همزمانی رو ایجاد می کنه به صورتی که هی داره بین این تسک ها جابجا میشه و به همه شون سرویس میده اما انقدر این جابجایی سریع انجام میشه که فکر میکنیم همزمان داره به همه سرویس میده  
       در parallelism اما دقیقا اینطوریه که همزمان داره به همه اون تسک ها سرویس میده. این کار یعنی داریم از چندتا cpu یا core استفاده می کنیم که باعث میشه این اتفاق بیفته
33. Cancellation token چیه؟
    1. توی async programming، گاهی پیش میاد که به دلایل مختلف (مثلا طولانی شدن اجرای یه Task)، بخوایم اجرای Task ای رو متوقف کنیم. یکی از راه های انجام اینکار استفاده از cancellation token هست که اکثر متد های async توی دات نت به صورت پیش فرض اون رو توی پیاده سازی هاشون دارن.
    2. برای استفاده ازش، اول باید یه CancellationTokenSource داشته باشیم که روی اون میایم کانفیگ های Cancellation رو ست میکنیم (مثلا اینکه اگه بعد از n ثانیه، تسک ما تموم نشده بود، cancellation token بیاد و جلوی ادامه دادن تسک رو بگیره). توی CancellationTokenSource یه پراپرتی به اسم Token هست که CancellationToken رو برمیگردونه. حالا ما میتونیم با پاس دادن این token به متد هامون، از قابلیت های CancellationToken استفاده کنیم. (میتونیم هم خود CancellationTokenSource رو پاس بدیم به متد ها اما CancellationToken رایج تر و بهتره مخصوصا که encapsulation رو هم رعایت میکنیم)
34. مزایای استفاده از collection ها؟
    1. بر خلاف data structure هایی مثل array که طول ثابت دارن، collection ها میتونن طول متغیر داشته باشن.
    2. داشتن طیف وسیعی از متد ها و پراپرتی های built-in که قابلیت های بیشتری رو در اختیار ما میذارن.
    3. Generic collection هایی مثل List باعث type-safety میشن.
    4. هر کدوم از collection ها به صورت بهینه پیاده سازی شدن مثلا List برای indexing بهینه عمل میکنه.
    5. یه راه حل استاندارد و همگانی برای کد نویسی در اختیار همه قرار میده به جای اینکه هر کس بیاد مدل خودشو پیاده سازی کنه.
35. تفاوت collection های generic با non-generic؟
    1. مدل های generic به صورت collection<T> تعریف میشن این یعنی باید حتما از یه نوعی باشن (boxing و unboxing نداریم) اما non-generic ها قسمت <T> رو ندارن چون که داخلشون به صورت object دیتا ذخیره میشه در نتیجه در زمان compile اروری نمی خوریم در ضمن در زمان استخراج داده (خوندن داده) باید یه conversion روی دیتا انجام بشه.
    2. Performance توی generic ها بالاتره چون type-safe هستن و نیاز به boxing و unboxing نداریم.
36. تفاوت array با arrayList؟
    1. تقریبا عین همدیگه ان اما arrayList میتونه سایز متغیر داشته باشه و چون non-generic ئه، میتونه تقریبا از هر نوعی، دیتا داخل خودش داشته باشه.
37. یه سری از خونده های قبلیت توی Onenote هست