C# Interview Questions & Answers E-book

www.questpond.com

Contents

Question 1 :- Explain difference between .NET and C# ?

.NET is a framework and C# is a programming language.

Question 2 :- .NET Framework vs .NET Core vs .NET 5.0

Question 3 :- What is IL (Intermediate Language) Code ?

IL Code is a partially compiled code.

Question 4 :- What is the use of JIT (Just in time compiler) ?

JIT compiled IL code to Machine Language.

Question 5 :- Is it possible to view IL code ?

Yes by using assemblers like IL DASM.

Question 6 :- What is the benefit of compiling in to IL code ?

The runtime environment and development environment can be very different. So depending upon the runtime environment JIT compiles the best optimized code as per that environment.

Question 7 :- Does .NET support multiple programming languages ?

Yes .NET support multiple programming languages like C#, F#, C++, Visual Basic etc.

Question 8 :- What is CLR (Common Language Runtime) ?

- CLR invokes JIT to compile to IL code.
- CLR cleans any unused objects by using Garbage Collector(GC).

Question 9 :- What is managed and unmanaged code ?

- Code that executes under CLR execution environment is called managed code.
- Unmanaged Code executes outside the CLR boundary. Unmanaged code is nothing but written in C++, VB6, VC++ etc.
  Unmanaged code have their own environment in which the code runs and its completely outside the control of CLR.

Question 10 :- Explain the importance of Garbage collector ?

- Garbage Collector is a back ground process which cleans unused managed resources.

Question 11 :- Can garbage collector claim unmanaged objects ?

- No

Question 12 :- What is the importance of CTS(Common Types System) ?

- CTS ensures that data types defined in two different languages get compiled to a common data type.

Question 13 :- Explain CLS ?

- CLS is a specification or set of rules or guidelines. When any .NET programming language adheres to these rules it can be consumed by any language following .NET specifications.

Question 14 :- Difference between Stack vs Heap ?

- Stack and heap are memory types in an application. Stack memory stores data types like int, double, Boolean etc. While heap stores data types like string and objects.

Question 15 :- What are Value types & Reference types?

- Value types contain actual data while reference types are contain pointers and pointers point to the actual data.
- Value types are stored on stack while reference types are stored on heap. Value types are your normal data types like int, bool, double and reference types are all objects.

Question 16 :- Explain boxing and unboxing ?

- When value type is moved to a reference type its called as boxing.
- When reference type is moved to a value type its called as unboxing.

Question 17 :- What is consequence of boxing and unboxing ?

- Boxing and unboxing decrease the performance of the program.

Question 18 :- Explain casting, implicit casting and explicit casting ?

- Type casting is a mechanism where we convert one type of data to other type.
- Implicit casting is when you move from lower to higher data type.
- Explicit casting is when you move from higher to lower data type.

Question 19 :- What can happen during explicit casting ?

- In explicit casting you can have data loss.

Question 20 :- Differentiate between Array and ArrayList ?

-

Parameters	**Array**	**ArrayList**
Fixed Length	Yes	No(flexible)
Strongly typed	Yes	No
Performance	Better than Array List	Slower because of boxing/unboxing

Question 21 :- Whose performance is better array or arraylist ?

- Array is better because it is strongly typed. ArrayList is slower than array because of boxing and unboxing.

Question 22 :- What are generic collections ?

Generic collection is strongly typed and flexible. It has better performance as compared to ArrayList.

Strongly typed in the sense we have to insert the list of data of same datatype and flexible in the sense we have the flebility of array size as of ArrayList.

Question 23 :- What are threads (Multithreading)?

If you want to run code parallely then we use threads.

Question 24 :- How are threads different from TPL(Task Parallel Library) ?

Question 25 :- How do we handle exceptions in C#(try/catch)?

- The code where it has chances of occuring exception is kept inside try block and if the exception occur then that is thrown to the catch block where we can decide what to do with that exception.

Question 26 :- What is the need of finally?

- Finally block runs even if you have an exception or if you do not have an exception.

Question 27 :- Why do we need the out keyword ?

- If you want to return multiple outputs from a function you will use OUT keyword.

Question 28 :- What is the need of Delegates ?

- Delegates is a pointer to a function and very useful as callbacks to communicate between threads.

Question 29 :- What are events ?

- Events are encapsulation over delegates.

Question 30 :- Whats the difference between Abstract class and interface ?

- Abstract class is a half defined parent class while interface is a contract. Abstract class is inherited while interface is implemented.

Question 31 - What is a Delegate and how to create a Delegate?

- Delegate is a pointer to a function.
- We can create delegate in two step:

  Step 1:- Declare the delegate.
  [Access modifier] delegate [return type] [delegate name] ([parameters])

  For example,
  Public delegate void **methodName**(string message);

  Step 2:- Create instance of the delegate.
  Public **methodName** publisher=null;

Question 32 - Where have you used Delegates?

- Wherever we want non-blocking call and want to communicate back we need delegates.
  For example: HTTP calls, FileSearch, TaskScedulers and so on.

Question 33 - What is a Multicast Delegates?

- Multicast delegates means attaching multiple functions to a delegate.

Question 34 - What is an Event?

- Events use delegates internally. They encapsulate delegates and make them safe.

Question 35 - How to Create an Event?

- By using the event keyword.
  Syntax:
  [Access modifier] event [delegate name] Variable_Name;

For example:

public delegate void **delegateName**(string search); // delegate

public event **delegateName** Variable_Name=null; // event

Question 36 - Delegate VS Events.

- Its unfair to compare delegates and events as events use delegates internally.
- Events encapsulate delegate and create a publisher subscriber model.
- In real time delegates are rarely used directly. It is mostly used in form of events.

Question 37 :- Why do we need Object Oriented Programming (OOP) ?

- OOP helps us to think about real world objects and is needed to solve real world problems.

Question 38 :- What are the important pillars of OOPs ?

There are four pillars of OOPs. They are:

- Abstraction
- Polymorphism
- Inheritance
- Encapsulation

Question 39 :- What is a class and object ?

- Class is a type, blueprint.
- Object is a instance of the class.

Question 40 :- Abstraction vs Encapsulation?

- Abstraction:- Show only what is necessary.
- Encapsulation:- Hide complexity.

Question 41 :- Explain Inheritance ?

- Inheritance defines the parent child relationship.

Question 42 :- Explain virtual keyword ?

- Virtual Keyword helps us to define some logic in the parent class which can be overridden in the child class.

Question 43 :- What is overriding ?

- Methods having same name and same signature but are in different class and these classes must have the parent-child relationship.

Question 44 :- Explain overloading ?

Method overloading means same method names with different signature in the same class.

Question 45 :- Overloading vs Overriding ?

Overloading:- Method with same names with different signatures in the same class.

Overriding:- Methods having same name and same signature but are in different class and these classes must have the parent-child relationship.

Question 44 :- What is polymorphism ?

- Polymorphism means the ability of an object to act differenly under different condition.

Question 45 :- Can polymorphism work with out inheritance ?

- No

Question 46 :- Explain static vs dynamic polymorphism ?

- Static polymorphism is implemented by Method overloading.
- Dynamic polymorphism is implemented by Method overriding.

Question 47 :- Explain operator overloading ?

- Operator overloading helps to redefine additional functionalities for plus, minus, multiplication and division.

Question 48 :- Why do we need Abstract classes ?

- Abstract class is a partially defined parent class.

Question 49 :- Are Abstract methods virtual ?

- Yes abstract method of the abstract class are bydefault virtual.

Question 50 :- Can we create a instance of Abstract classes ?

- No we can't.

Question 51 :- Is it compulsory to implement Abstract methods ?

- Yes.

Question 52 :- Why can't simple base class replace Abstract class ?

- A simple base class can not be defined in a pure half way so simple base class never replace abstract class.

Question 53 :- Explain interfaces and why do we need it ?

- Interface is a contract. It is a legal binding between the developer who is creating the class and the consumer who is using the class.

Question 54 :- Can we write logic in interface ?

- No. Interface can only have pure signatures.

Question 55 :- Can we define methods as private in interface ?

- No. All the methods or properties are public in interface.

Question 56 :- If I want to change interface what's the best practice ?

We will create new interface. The class that want to implement both the interface will do multiple inheritance because interface support multiple inheritance.

Question 57 :- Explain Multiple inheritance in Interface ?

- Multiple inheritance in interface means a class is implementing more than one interface.

Question 58 :- Explain Interface Segregation principle ?

- Interface Segregation principle defines that "Clients should not be forced to depend upon interfaces that they do not use".

Question 59 :- Can we create instance of interface ?

- No we can't create the instance of the interface.

Question 60 :- Can we do Multiple inheritance with Abstract classes ?

- No we can't do multiple inheritance with the abstract classes.

Question 61:- Abstract Class vs Interface

- In abstract class there are some methods which are defined and some are not defined. But in the case of interface there are only empty methods.

Interface	**Abstract Class**
It is a contract.	It is a half-defined parent class.
Interface is for Planning abstraction	Abstract class is for Sharing common logic in child classes
Interface is implemented.	Abstract class is inherited.
Multiple inheritance is possible for interface.	Multiple inheritance is not possible for abstract class.

Question 62 :- Why do we need constructors ?

- Constructors are special methods which is invoked automatically when the instance of that class is created and is used to initialize the variables.

Question 63 :- In parent child which constructor fires first ?

- First parent is fired then child.

Question 64 :- How are initializers executed ?

- First child then parent.

Question 65 :- How are static constructors executed in Parent child ?

First child then parent.

Question 66 :- When does static constructor fires ?

When first time the class is accessed.

Association indicates there is dependency between objects. Aggregation and Composition are subset of Association

	Aggregation	**Composition**
Part whole	Yes	Yes
Life time	Independent	Dependent
OwnerShip	No ownership	Parent object has ownership Death relationship

Question 127: - Explain Garbage collector (GC)?

It's a background process which runs undetermestically and cleans unreferenced managed objects from the memory.

Question 128:- How does Garbage collector know when to clean the objects ?

When the objects goes out of scope GC reclaims the memory and gives it to operating system.

Question 129 :- Is there a way we can see this Heap memory ?

Yes, we can analyze GC using performance counters. Performance counters are counters or they are measures of events in a software which allows us to do analysis. These counters are installed when software is installed.
So we can use counters like GC Heap size , GC0 , GC1 , GC2 and working set to make more sense of how GC is working.

Question 130 :- Does Garbage collector clean primitive types ?

No , Garbage collector does not clean primitive types. They are allocated on stack and stack removes them as soon as the variables goes out of scope.

Question 131: - Managed vs UnManaged code/objects/resources?

Managed resources are those which are pure .NET objects and these objects are controller by .NET CLR. Unmanaged resources are those which are not controlled by .NET CLR runtime like File handle , COM objects , Connection objects and so on.

Question 132:- Can garbage collector clean unmanaged code ?

No, GC only cleans managed objects.

Question 133:- Explain Generations ?

Generations are logical buckets which have objects and every bucket defines how much old the objects are.

Question 134:- What is GC0,GC1, and GC2 ?

GC0:- Short lived objects. Ex. Local Objects.
GC1:- Intermediate lived objects.(Buffer).
GC2:- Long live objects. Ex. Static objects.

Question 135:- Why do we need Generations ?

The whole goal of generations is performance. GC makes a assumption that if objects are needed longer in memory then it should be visited less as compared to objects which are freshly created and which have high probability of going out of scope.

Question 136:- Which is the best place to clean unmanaged objects ?

Destructor is the best place to clean unmanaged objects.

Question 137:- How does GC behave when we have a destructor ?

When a class has a destructor GC takes more trips to clean them and due that the objects are promoted to upper generation and thus putting more pressure on memory.

Question 138:- What do you think about empty destructor ?

Having Empty destructor will cause lot of harm as objects gets promoted to higher generations thus putting pressure on the memory.

Question 139:- Explain the Dispose Pattern?

In Dispose pattern we implement “IDisposable” interface and call “GC.SuppressFinalize()”

Question 140 :- Finalize vs Destructor ?

Finalize and Destructor are one the same. Destructor calls the Finalize method.

Question 141:- What is the use of using keyword ?

Using statement defines a scope at the end of the scope “Dispose()” is called automatically.

Question 142:- Can you force Garbage collector ?

Yes,you can by calling “GC.Collect()”.

Question 143:- Is it a good practice to force GC ?

“GC” runs depending on various criteria’s like is memory running low , is processor getting overloaded and it does its work wonderfully. Fiddling with GC is not recommended at all.

Question 144:- How can we detect a memory issues ?

Memory issues can be detected by running tools like Visual studio profiler. And we can check for two things: -

- If the memory is increasing linearly it’s a indication of memory issues. If the memory is moving in a range it’s a healthy sign.
- Also the memory allocation and deallocation should be balanced. If you just see memory allocation and no deallocation is other sign that there is serious memory issues.

Question 145:- How can we know the exact source of memory issues ?

In profiler we should check for the top most memory allocated to objects. Once we know the top most memory allocated to objects we can then focus on code around those objects.

Question 146 :- What is a memory leak ?

Memory leak is a situation where the memory consumed by the application is not returned back to the operating system when the application exits.

Question 147:- Can .NET Application have memory leak as we have GC ?

Yes its still possible to have memory leaks because GC only takes care of managed memory. If unmanaged memory is not claimed properly we can have memory leaks.

Question 148:- How to detect memory leaks in .NET applications ?

Total memory of .NET app = Unmanaged + Managed. So, if you see just see total memory is increasing and managed is in a range then it means there is Unmanaged leak.

Question 149:- Explain weak and strong references ?

Weak reference: - It permits the GC to collect the object but still allows to access the object until GC collects the object. We need to use the “WeakReference” object to create weak reference.

Strong reference: - This is a normal referenced objects and once object is marked for GC it can never be referenced.

Question 150 :- When will you use weak references ?

Caching , Object pooling . Wherever object creation process is resource intensive caching and pooling can improve performance.

Question 151:- What are design patterns?

Design patterns are time tested solution for recurring architecture problems.

Also, with the above definition try to give one or two design patterns as examples. While giving examples remember three things: -

1. Do not give example of Singleton pattern so that you can stand out in the crowd.
2. Give one example of non-GOF pattern like repository, CQRS and so on.
3. And whatever patterns you say in example make sure you are fully aware of the same.

Examples of pattern please tailor as per your knowledge

"Prototype" helps to fully initialize instance to be copied or cloned

"Adapter pattern" helps to make incompatible interfaces compatible.

"Iterator pattern" helps iterate over elements of an aggregate object sequentially without exposing the underlying representation of the object.

"Template pattern" define a skeleton of the algorithm in parent class and let subclass override specific steps.

"Repository pattern" helps to abstract and centralize the data access logic in an application. It provides a layer of separation between the application's business logic and the data access code, making the code more modular, maintainable, and testable.

Question 152 :- Which are the different types of design patterns?

Question 153 :- Explain structural , Behavioral and Creational design pattern ?

There are 3 categories of design pattern: -

- Creational: - Problems and solutions around object creation issues.
    - Prototype - A fully initialized instance to be copied or cloned
    - Singleton - A class of which only a single instance can exist

- Behavioral: - Problems and solutions around Communication between objects.
    - Chain of responsibility - A way of passing a request between a chain of objects
    - Command - Encapsulate a command request as an object
    - Mediator - Defines simplified communication between classes
    - Memento - Capture and restore an object's internal state

    - Template method - Defer the exact steps of an algorithm to a subclass

- Structural: - Solving concerns around class structure and object composition.
    - Adapter - Match interfaces of different classes
    - Composite - A tree structure of simple and composite objects
    - Decorator - Add responsibilities to objects dynamically

Question 154 :- Which design pattern have you used in your project?

Note: - first and foremost do not answer SINGLETON pattern as it will not help you stand in the crowd. and remember: -

- My choice of patterns is not yours so pick what you are comfortable.
- Only talk about patterns you are confident of. Many patterns you have already used in your projects. If you can have closer look on them you can speak not only confidently but also naturally. When answers are natural interviewers love it.
- Do not just talk GOF but also talk about Non-GOF patterns.
- Yes, avoid singleton stand in the crowd. Its over used and abused in interviews.

Some of the most used patterns are: -

Most of the applications are CRUD so repository pattern comes at the top.

Repository pattern: - Acts like an abstract layer between Models and Data access technologies like EF, ADO.NET and so on. Data access logic is centralized making code maintainable, testable and modular.

With repository pattern, UOW goes like hand in gloves.

UOW (Unit of work): - This pattern helps to manage transactions and changes made to objects. It goes with repository pattern well.

We all know we use FOR EACH so much so the next pattern is also very must used. One important point to note here talk about IEnumerator and IEnumerable as they implement iterator by default.

Iterator pattern: - helps iterate over elements of an aggregate object sequentially without exposing the underlying representation of the object.

Factory pattern:- From creational pattern this is the most used one. Creates an instance of several derived classes.

Used when third party components are used.

Adapter pattern :- Makes incompatible interfaces compatible.

Decorator pattern :- Add behavior dynamically.

Command pattern:- Treat command as objects. Heavy use in CQRS

Façade :- Represent subsystem in a simplified way.

Composite - A tree structure of simple and composite objects

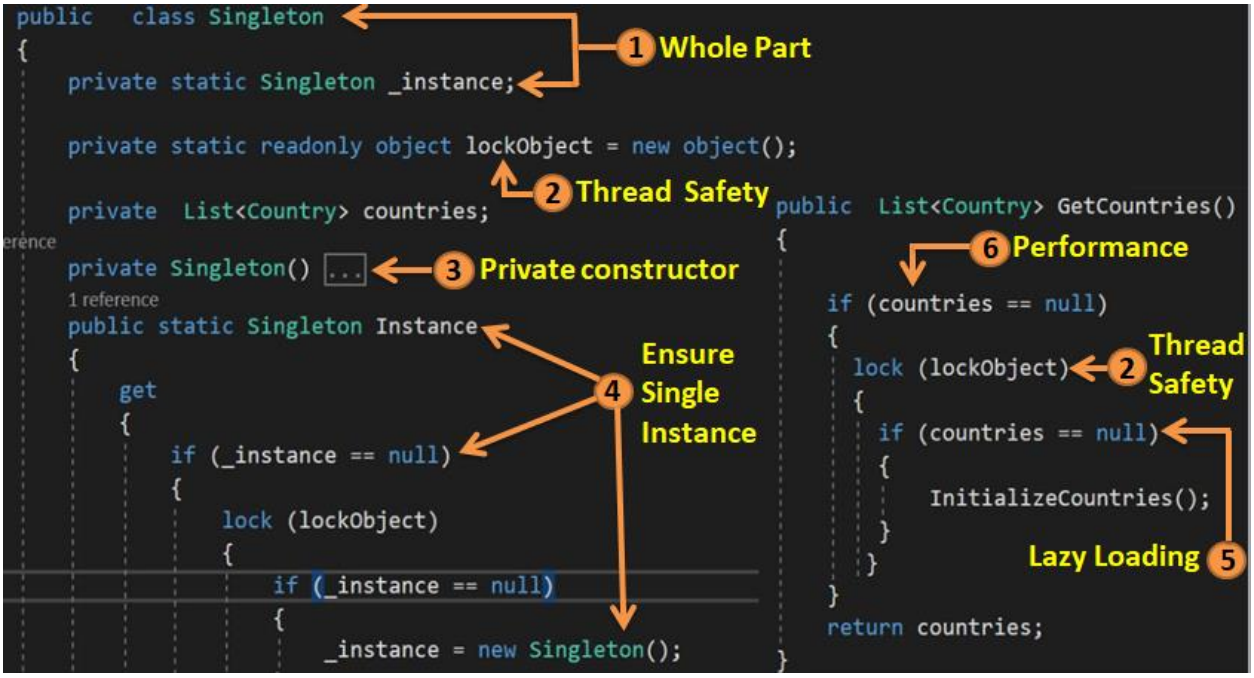
Template method - Defer the exact steps of an algorithm to a subclass

Question 154 :- Explain Singleton Pattern and the use of the same?

Singleton pattern helps to create a single instance of an object. Some of the uses of Singleton patterns are: -

- Caching of data like Countries, States, Currencies and so on.
- Global sharing of data like common themes , hit counters and so on.

Question 155 :- How did you implement singleton pattern?



To implement singleton pattern (Check the number with the above figure): -

1. Whole part relationship: - The first thing needed in singleton is a root class through which all shared objects should be exposed. This root class instance should be created inside the class and it should be static.
2. Thread safety: - Whenever we are loading the singleton object use the “lock” keyword to make sure only one thread manipulates at a time.
3. Private constructor: - Make sure the root class can not be instantiated from outside.
4. Ensure root object is single instance: - use the double null check and make sure that only one instance is created and also lock is not executed unnecessarily.
5. Lazy loading implementation: - We would like to load the objects when demanded then loading it unnecessarily.
6. Performance for threading: - NULL check before the locking to ensure that we do not execute locks unnecessarily.

Question 156 :- Can we use Static class rather than using a private constructor?

Question 157:- Static vs Singleton pattern?

Let’s answer both the questions in One Go.

- Keyword VS Design Pattern: - Static is a language keyword while Singleton is a Design pattern.
- Loose many OOP features: - If you make a class static you cannot implement interfaces, cannot inherit and so on.
- Lazy loading / Thread Safety: - When you make a class static, object is created in the first call itself without giving you option of when to load and when not. Second you should also make sure its thread safe as it’s a global object.

Question 158:- How did you implement thread safety in Singleton?

C# “lock” keyword helps to implement thread safety.

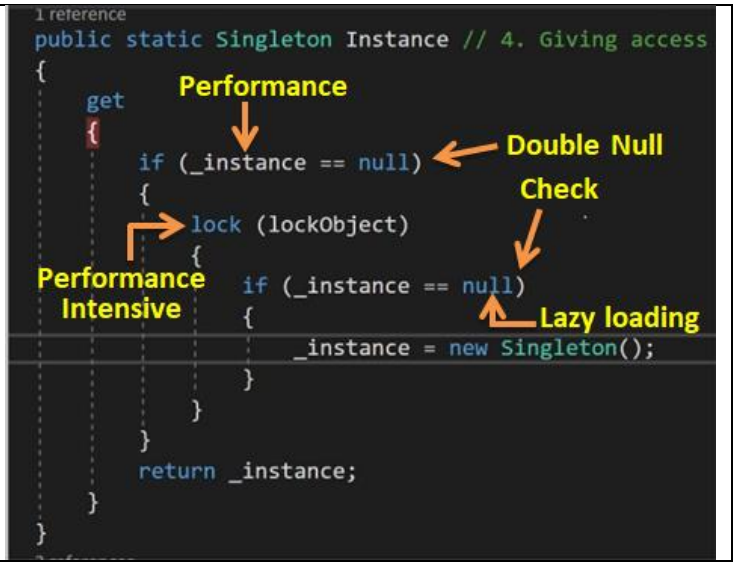
Whatever code is in the scope of “lock” keyword will get executed by only ONE thread at time avoiding any thread unsafe situation.

Question 159:- What is double null check in Singleton?

Double null check is done for two purposes.

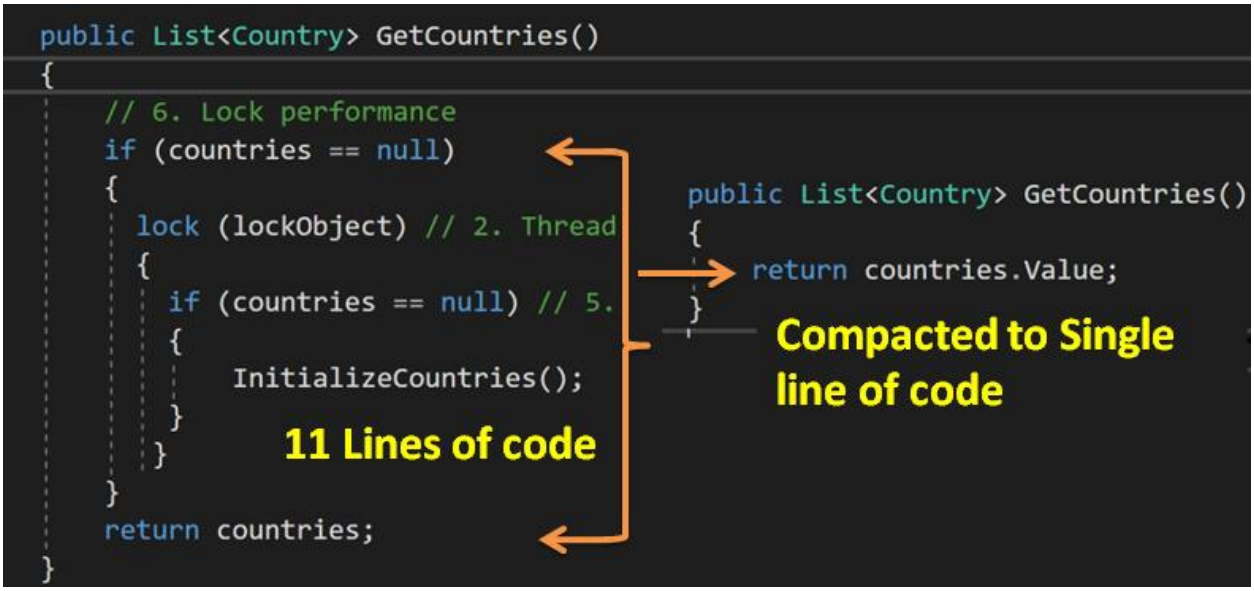
Internal null check: - This is done to make sure the instance load only once and its loaded OnDemand.

External null check :- This is done so that “lock” is not acquired unnecessarily. “lock” is an intensive process and should be executed only when the singleton is null.



Question 160:-Can Singleton pattern code be made easy with Lazy keyword?

Yes, by using LAZY keyword we can make the code size smaller. LAZY makes code thread safe and also does late initialization.

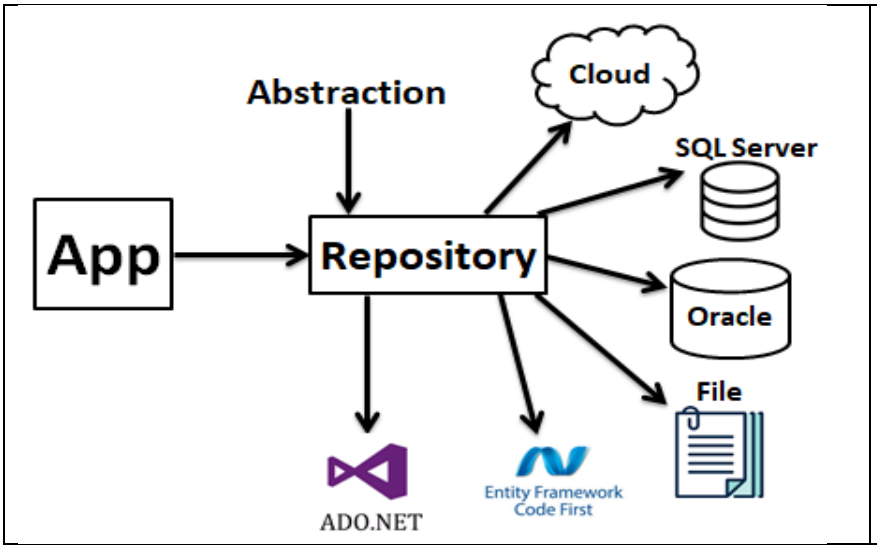


Question 161:-Can we rid of this double null check code?

Yes, by using LAZY keyword you can get rid of double null check.

Question 162:-What is the use of repository pattern?

Note: - While answering this question the word abstraction should come out prominently.



Repository pattern helps to **abstract** and **centralize** the data access logic thus providing a clean and consistent interface for CRUD operation on models.

Question 163:-Is Dal (Data access Layer) and Repository same?

Note: - In interviews for repositories do not use vocabulary Dal interchangeably, it can send a different impression to the interviewer.

Dal focuses on the technical details of data access while repository pattern provides a higher level of abstraction over Dal. Dal provides technical centric approach of how to access the data source, while Repositories provide Domain centric approach.

Question 164:-What is Generic repository pattern ?

Generic repository pattern is an extension to repository pattern. Rather than creating separate interfaces / classes for each entity, you can just create a generic interface / class for all entities.

Question 165:-Is abstraction the only benefit of Repository?

Note: - I framed this question on purpose as this can be a part of discussion more than a question. During this discussion the interviewer is expecting that **reusability** is also another big benefit of Repository other than loose coupling.

The other benefit of repository pattern is reusability across data access layers. There can be many logic which is common across data layers and they can be put in a common class and all repositories would inherit from that common class.

Question 166:-How to implement transaction in repository?

Note: - In this question the interviewer is expecting you should know Unit of work design pattern which is answered in the next question.

Question 167:-What is Unit of work design pattern?

Unit of Work (UOW) helps to abstract / implement transactions in repository.

Note :- Many developers do not know how to answer transaction definition so here is the answer :- Transaction is a group of task / operation where in either the whole group is committed or the whole group is rolled back.

Question 168:-Do we need repository pattern as EF does almost the same work?

Note: - This question cannot be answered in black and white, it's a subjective question and interviewers can have their own perspective. So, the challenge is if you do not answer from interviewer's perspective, he will feel uncomfortable and on the other hand there can be interviewers who would like to get in to intellectual discussion and understand that do you know both the perspectives.

Perspective 1 :- Repository pattern is a must

The first school of thought believes in best practices like loose coupling and unit testing. They understand the importance of repository pattern that it helps to abstract the Data access layer (Heterogenous data sources) so that we can change / use different DAL technologies (EF , Dapper , ADONET etc.) with ease.

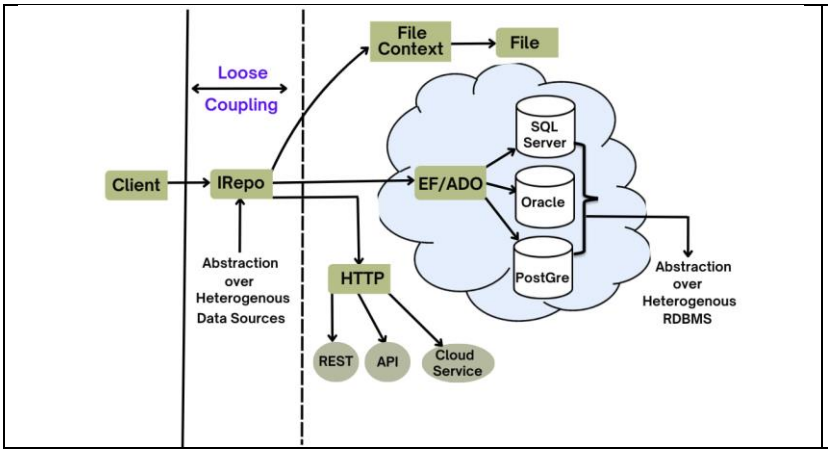
If you are using multiple DAL technologies (heterogenous data sources) and you want plug and play of DAL in future, repository is a MUST.

Second reason this school also thinks that unit testing is important. Which again is a very compelling reason you need to have a repository so that you can mock test it easily.

Perspective 2 :- EF implements Repository not needed

Myth: - Many developers from this school think that EF implements repository pattern. But that's not true. Repository helps you to only abstract RDBMS data source and not other data sources (File , API and so on) . Repository pattern abstracts across Heterogenous data access technologies.

But if you are only dealing with heterogenous RDBMS then I **personally** feel repository pattern is overkill.



Also let's try to visualize the same in a picture format.

Repository's main goal is to provide abstraction over heterogenous data sources while EF is ORM which provides abstraction over heterogenous RDBMS sources.

Now summing the answer from interview perspective.

Comparison of Repository pattern with EF is like comparing apples with oranges because one is Data access technology while other is a design pattern which focuses on abstraction over heterogenous data access technologies.

As a good practice it's always good to use Repository pattern as it creates a higher level of abstraction over any kind of data source and also makes MOCK UNIT TESTING easy to implement.

After this watch the interviewers reaction if you think he belongs to the second school and if he is having an ANTI- REPOSITORY mind set then you can just say *"Yes , I do agree if we do not have heterogenous data sources repository pattern can be overkill"*.

*Note :- One of the key to success during interview is "**Satisfying the interviewer that he is RIGHT (The EGO)** ". That releases a chemical called as dopamine which make him feel good and that can be affect your interview results in a very positive way.*

Question 169:-Did you do unit testing with Repository ?

Question 170:-How does repository pattern make unit testing easy?

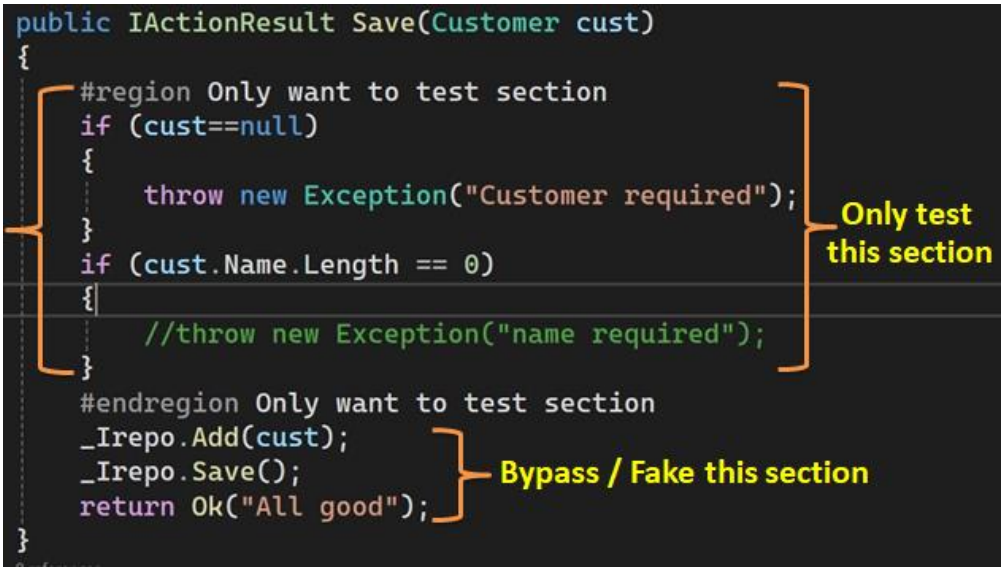
Question 171:-How can we do mock testing with Repository?

Note: - All the above flavor of questions stresses the importance of unit testing and mock testing in repository. Interviewer would like to know that do you understand the importance of unit testing in repository.

Repository pattern is a higher level of abstraction over Data access logic. And this Abstraction is represented by EMPTY GENERIC INTERFACE. Because of this generic Interfaces unit testing is like a cake walk.

The ONE big benefit of doing unit testing on Empty interface is ISOLATED MOCK UNIT Testing. Mock Unit testing has two primary uses: -

- Unit testing by passing Data access Logic: - Many times you want to only test business logic and not the data access part. With repository this is like a cake walk.
- Parallel Development and Testing: - Many times data access is developed by some other team , so either they have not completed the code or are in between and you want to still test your part of the code , repository is the answer for it.




```
private bool FakeSave(){
   // Do nothing just by pass
  return true;
}
```

```
var mockRepo = new Mock<IRepository<Customer>>();

// create the fake object with fake method

mockRepo.Setup(x => x.Save()).Returns(FakeSave());
```

Question 172 :- What is Factory pattern and how does it benefit?

Factory pattern belongs to creational pattern category. It centralizes objection creation and the biggest benefit of centralizing object creation is "*application becomes LOOSELY COUPLED*".

Question 173 :- How does centralizing object creation helps in loose coupling ?

Object creation is normally done by the NEW keyword. For new keyword you need reference of the concrete class , once you are referencing the concrete class you have TIGHT coupling scenario.

Now if these “NEW” keyword is scattered all across the places think about the amount of changes you need to do when you want to switch to Entity Data layer or a different type of Data layer.

So if we can centralize this at one place we do not have to make changes all over the places.

Question 174 :- What is IOC and DI ?

Question 175 :- DI vs IOC ?

IOC (Inversion of Control) is a principle while DI (Dependency Injection) is an implementation of IOC.

So, IOC says that delegate / invert the object creation to external framework. While DI is an implementation for IOC which says how the dependent objects will be injected (from constructor , method etc).

Note :- In interviews you will find use of DI containers which means a central collection of dependencies.

Question 176 :- What is a service locator ?

Service locator is like a centralized registry for retrieving and managing dependencies. Now a days DI is used more as it’s a neat and does proper Decoupling.

Question 177:- Service Locator vs DI ?

Service locator is more of a pull while DI is more of a push. SL can make searches, calls to many registries , DI containers or any other service / third parties to create instances.

Question 178 :- Which is good to use Service Locator or DI ?

Service locator makes system tightly coupled while DI makes it more loose coupled. Currently developers mostly use DI.

Question 179 :- Can not we use a simple class rather than interface for DI ?

You can do it technically, but if you want proper loose coupling , you need proper abstraction and that abstractness only Interfaces can provide.

Question 180 :- Is DI a Factory Pattern?

No, DI is not factory pattern. DI just says how the objects are injected. While Factory pattern centralizes COMPLEX object creation.

Question 181 :- So If you just centralize object creation is it Factory pattern?

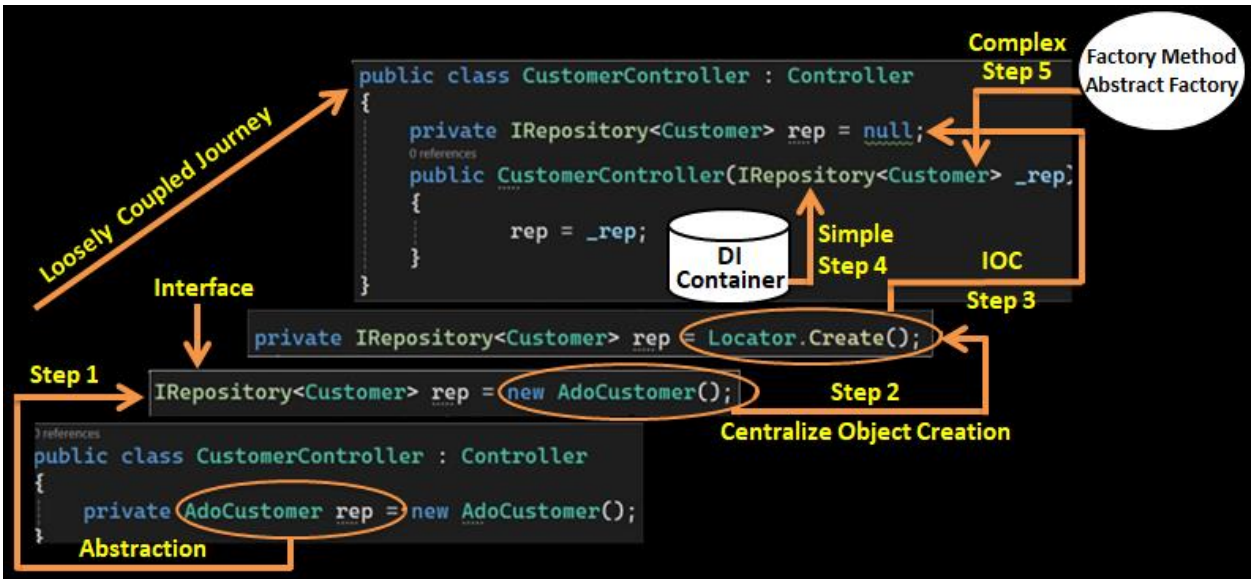
No. In case of Factory pattern the object creation has to be complex. There has to be some complex logic.

Question 182 :- Static DI and Dynamic DI ?

Static DI means application needs to restart for taking up the changes. Dynamic DI means during the runtime which object will be injected will be decided.

Question 183 :- In which scenarios to use Static DI vs Dynamic DI ?

Static DI is mostly used for services while Dynamic DI is used for complex model creation during runtime.



Question 184 :- The real Factory pattern ?

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Question 185 :- Factory Method VS Factory pattern ?

Personally, for me both of them are same. You will need to cross question the interviewer what exactly he is looking for. There are some vocabulary confusion, just to try to align with the interviewer to avoid frictions.

Question 186 :- How are new behaviors created in Factory pattern ?

By inheritance.

Question 187 :- What is Abstract Factory Pattern ?

Provides an interface for creating families of related or dependent objects without specifying their concrete classes.

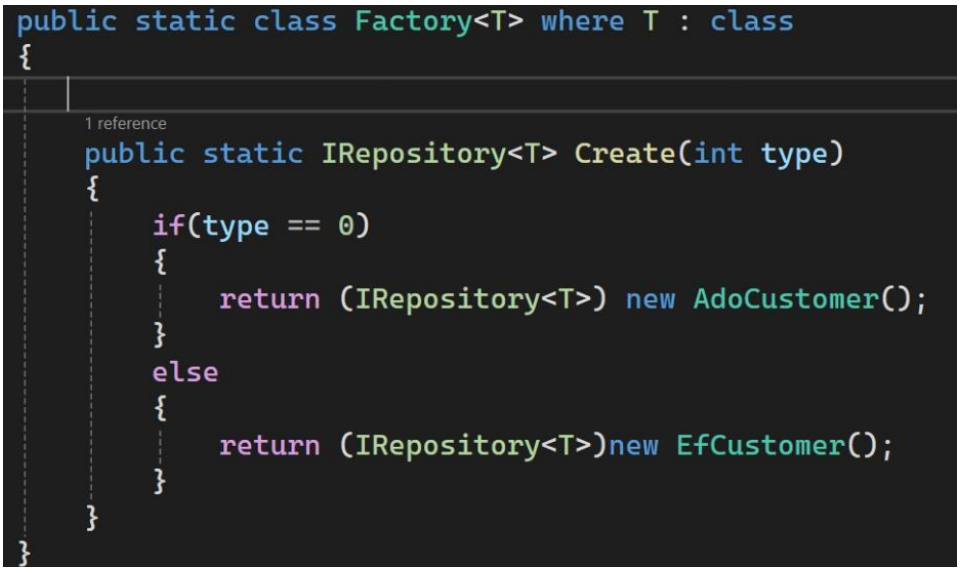
Question 188 :- Does Abstract Factory Pattern use FP inside ?

Note :- This is test question to see that if you understand relationship between Factory and Abstract Factory.

Yes , Abstract factory uses Factory internally. So, they kind of Aggregate similar families of Factory.

Question 189 :- Explain Simple Factory Pattern ?

There is nothing officially termed as Simple Factory. Below is a sample code what developers term as simple factory. But Simple factory is not an official terminology coming from GOF. The below code is nothing but a great use of two concepts OOP polymorphism and centralizing the NEW keyword.



Question 190 :- Simple Factory vs Factory (Factory Method) vs Abstract Factory ?

Note :- There is no Pattern as such called as Simple Factory. This name has come up from the industry to distinguish between Real Factory pattern and Simple IF condition polymorphism.

	Simple Factory	**Factory Pattern/Method**	**Abstract Factory**
Complexity	Less complex. Simple Single Object creation.	Very Complex. Composition of object giving out Different behaviors.	Very Complex. Family of related Factories.
Technique	IF / Select / Lookup/ DI container	Defer to Subclasses (Inheritance)	Uses Aggregation to logically group Families.
Interface for Creation	NA	Needed	Needed

Question 191 :- How to remove IF conditions from Simple Factory?

We can create a simple collection or use some DI container frameworks.

```
public static class Factory1<T> where T : class
{
     static Dictionary<int, IRepository<T>> dicts = new
                    Dictionary<int, IRepository<T>> ();
    0 references
    static Factory1()
    {
        dicts.Add(1, (IRepository<T>) new EfCustomer());
        dicts.Add(1, (IRepository<T>)new AdoCustomer());
    }

    0 references
    public static IRepository<T> Create(int type)
    {
        return dicts[type];
    }
}
```

Can you provide the MS Architecture for library management and requested to explain my thought process in the white board" and will continue with following design pattern questions

Which approach we follow to upgrade monolithic applications to micro service

is it mandatory to use unit of work while implementing Repository pattern?