1) Difference between dotnet & c#

dotnet is framework & C# is programming language (it has syntax)

dotnet has collection of library & runtime / run the application

---

2) Differentiate between .net framework vs .Net core vs .Net 5.0

- .Net framework runs only on windows. Only
- .Net Core core is cross platform [linux, macos, windows]
- .net 5.0 provides a unified experience.

one single collection has been divided into many

ex!- system. collection ⇒ System. collections
System. collections. Concurrent
system. collections. Immutable
system. collections. NonGeneric
system. collections specialized.

Difference between .Net framework vs .NET Core vs NET 5.0

	.NET framework 4.x	.NET Core 3.x	.NET 5.0
Cross platform	Only for windows	Work cross platform	.NET 5.0 is a united platform which unifies all .NET framework, net core, mono and so on.
Performance	slow comparison .NET Core	Better	
Types of Application supported	Winform, WPF, Asp.NET webforms, Asp.NET MVC5	NWC Core 3.x	Developer no longer need to choose between .Net core, .Net Framework and more based on which platform they're developing this applications.
WCF WPF WWF	Yes	No	
Packaging	Packaged as one big framework.	Delivered via modularly using Nuget	This provides a common experience to developers. Irrespective of which .NET version they are working
Micro service support	No	Yes	
CLI tools	More IDE Based	full CLI command supported	
Mobility compatibility	No support directly	with .NET standard compatible with Xamarin	
Cloud	Work but only windows	yes	

Q3) What is IL (Intermediate language) code? (.dll) (drag to notepad)

Q4) What is the use of JIT (Just in Time compiler)?

Console.writeline(" ") → IL (Intermediate language) → JIT (Just in time compiler)

Q6) What is the benefit of compiling in to IL code?

What does the interviewer want to hear?

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

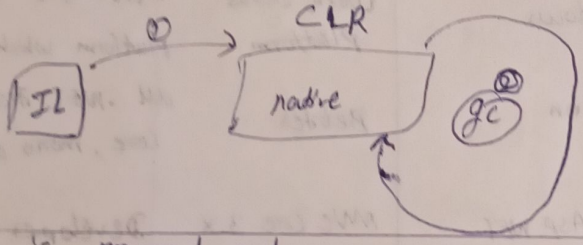
Q7) Does .net support multiple Programming languages?

VB.net, C#, F#, C++, Javascript

8) What is CLR (Common language Runtime)?

CLR invoks JIT to compiler to IL code.

Cleans any unused objects by using GC



9) What is managed and unmanaged code?

Code that executes under CLR execution environment is called as managed code. Unmanaged code executes outside CLR boundary. Unmanaged code is nothing but code written in C++, VB6, VC++ etc.

Unmanaged codes have their own environment in which the code runs and its completely outside the control of CLR.

```
class program
{
    [DLLImport("user32.dll")]
    static extern int MessageBoxA(int hwnd, string strMsg, string
                                  strCaption, int iType);
    static void main(string[] args)
    { MessageBox(0, "Yes", "No")
```

Q10) Explain the importance of Garbage collector?

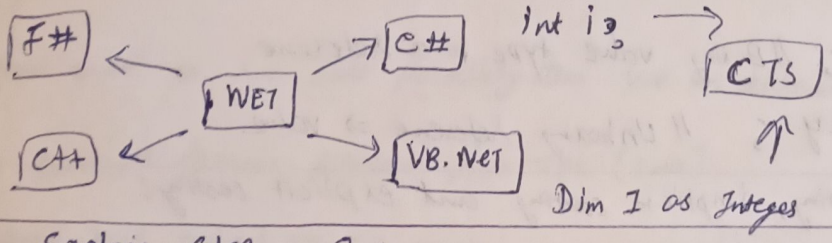
Garbage collector is a background process which cleans unused managed resources.

Q11) Can garbage collector claim unmanaged objects?

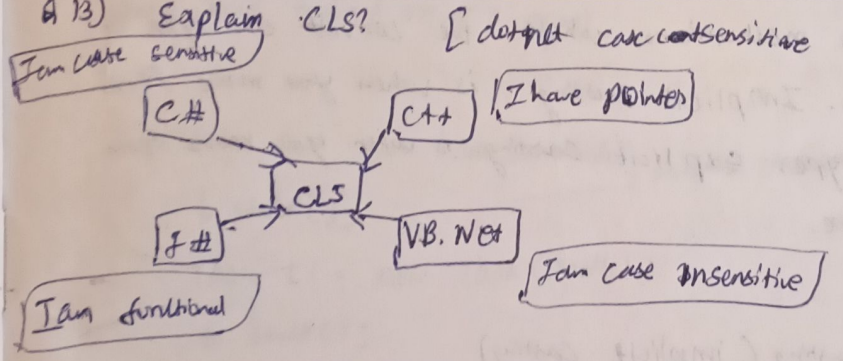
no, garbage collector doesn't ~~create~~ collect object out side CLR. compiler.

Q12) what is the important of CTS?

CTS (Common types system) ensures that data types defined in two different language gets compiled to a common data types.



Q13) Explain CLS? [dotnet case sensitive int i=3 int I → this will work



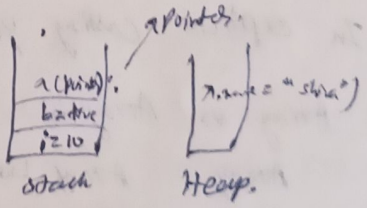
CLS is a specification or set of rules or guidelines. when any .net programming language adheres to these set of rules it can be consumed by any language following .NET specifications.

Q14) Difference between stack vs Heap?

Primitive datatype → Stack → int, double, etc.

Customer x = new customer();

x.name = "shiva";



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.

Q.15) what are values Types & Reference types?

Value types contain actual data while reference types contain pointers and the 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.

Q17) What is a concept of boxing and unboxing?

When value type is moved to a reference type its called as boxing. The vice-versa is termed as unboxing.

```
int i=10;          [~~post~~ performes implicaton will be this]
object y=i; // Boxing value type to a reference.
int z=(int)y; // Unboxing reference => value.
```

Q18) Explain casting, implicit casting and explicit casting?

Type casting is a mechanisms 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.

```
int i=10;
double d=i; // casting (implicit casting)
double d1=100.23;
int y=(int)d1; // explicit casting
```

Q19) what can happen during explicit casting?

In explicit casting you can have data loss

Q20) Array vs Array List.

	Array	Array List
Fixed type	yes	No(flexible)
Strongly typed	yes	no
performance	Better than Array List	slower because of boxing/unboxing

Q12) What is generic collections?

generic collection is strongly typed and flexible. It has better performance as compared to Array list.

Q13) what are threads (multithreading)

```
static void main (string[] args)
{
    Thread t = new Thread(method1);
    t.start();
    Thread t1 = new Thread(method2);
    t1.Start();
}
```

parallel

method1 method2

using System.Threading;

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

Q14) How are threads different from TPL? [Task]

```
Static void main (string[] args)
{
    Task t = new Task(method1)  → parallel processing.
    t.start();
    Task t1 = new Task(method2)
    t1.start();
}
```

"Do not hit around the bush. Interviewer does not have time."

Thread → context switching, Time slicing.

Remember 3 p's

P → Pooling Threads

P → parallel Processing.

P → plus (Return, result, Cancel, Chain, Async Await)

legacy code → Threads

new dotnet 3 → TPL

Q25) How do we handle exceptions in C# (try/catch)?

where ever exception occure add try block and

Q26) what is the use of finally block?

Finally Block runs you have a exception or you do not have an exception.

Q27) Why do we need the out keyword?

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

Q28) what is the need of Delegates?

Delegate is a pointer to a function and very useful as call backs to communicate between threads.

Q29) what are events?

Events are encapsulation over delegates.

Q30) whats the different 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

Tips for the interview

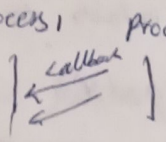
1. Do not hit around the bush.
2. Say the important points first.
3. use polished right technical vocabulary.

Q31) what is a delegate and How to create a delegates.

```
Class Program
{
  public delegate void some method(string search)
  public static Some method somepointer = null;
  static void main(string[] args)
  { myfile search x = new myfile search();
    some pointer = x.search;
    some pointer ("E:\\Shivaprasad data");
```

Delegates are callback by which helps to communicate between asyn and parallel execution/process. Creation of delegates is a two step process.

callbacks



Q) where have you used delegates?

Q) what is a multicast delegates?

Q) What is the event?

Event use delegates internally. event are encapsulation over delegates Events use delegates. Delegates are of callbacks, not encapsulated Events publisher subscriber model, encapsulated.

step 1:- Declare the delegate

public delegate void searchmethod(string search)

Step 2:- Create instance of the delegate

public searchmethod publisher = null;

Q) where have you used delegates?

Whereever we want non-blocking call and want to communicate back we need delegates. for example HTTP calls, file search, ToshSchedulers and so on.

Q) what is a multicast delegates?

multicast delegates are delegates which broadcast to multiple function.

Q) what is a Events?

Event are encapsulation over delegates they help to implement publisher subscribes model.

Q) How to create a event?

By using the event keyword

public delegate void searchmethod(string search)

public event searchmethod publisher = null

Delegate vs Events

It unfair to compare delegate and events as event use delegates internally.

Events encapsulate delegate and create a publisher subscriber model.

In real time Delegate are rarely used directly. Its mostly used in form of Events.

OOP

Q37) why do we need OOPs?

It helps developers think in terms of real world objects.

overloading - method with same names with different signatures.

overriding - using virtual keyword and overriding in child class

Q38) what are the important pillars of OOPs?

APIE :- Abstraction, polymorphism, inheritance and Encapsulation.

ABSTRACTION
POLYMORPHISM
INHERITANCE
ENCAPSULATION

Abstraction - show only what necessary

Encapsulation - Hide complexity

abstraction happen in design phase uses Encapsulation to implement of abstraction.

Q39) what is a class and object?

class is a type, blue print and object is a instance of the class.

Q40) Abstraction vs Encapsulation?

Abstraction :- show only whats necessary

Encapsulation :- Hide complexity.

Q41) Explain Inheritance?

Inheritance defines a parent child relationship.

Q42) Explain virtual keyword?

Q43) What is overriding?

virtual keyword helps us to define some logic in the parent class which can be overridden in the child class.

Q44) Explain overloading!

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

---

Q45) overloading vs overriding

overloading :- method with same names with different signatures

overriding :- using virtual keyword and overriding in child class.

Q44) what is polymorphism?

poly means many, morph means change as per situation.

It an ability of object to act differently under different condition.

Employee e = new manager();

e = new supervisor(); {Both are inheritance one class}

Q45) can polymorphism work with out inheritance?

No, with out inheritance not able to implement polymorphism.

Q46) Static vs Dynamic polymorphism?

↳ compile time ↳ Runtime

Static poly is implemented by method overloading

Dynamic poly is implemented by method overriding

Q47) Explain operator overloading?

operator overloading helps to redine redefine additional functionalities for plus, minus, multi and addition

---

Q48) How to do custom operator overloading?

need to use operator keyword

```
var o1 = new someClass(10);
var o2 = new someClass(20);
var o3 = o1 + o2
```

```
public class someClass
{
    public static someClass operator+(
        someClass arg1, someClass arg2)
    {
        return new someClass(arg1.some, arg2.some);
    }
}
```

Q48) Why do we need Abstract classes?

Abstract class is a half defined parent class Impress the interviewer by saying partially defined?

Q49) Are Abstract methods virtual?

Yes. it is virtual

Q50) Can we create a instance of Abstract classes?

Q51) Is it compulsory to implement Abstract methods

No, it will throw error. Yes, it is compulsory to need to implement

Q52) Why simple base class replace Abstract class?

A simple base class can not be defined in a pure halt way.

Q53) Explain interfaces and why do we need it?

interface is a contract

Q54) Can we write logic in interface?

Q55) Can we define methods as private in interface?

No, we cannot have logic [only signature]. No. all the method and property shall be public.

Q56) If I want to change interface whats the best practise?

Interface is a contract, by having contract we have better control on impact analysis, change management and breaking changes?

Q57) Explain multiple inheritance in interface?

multiple inheritance helps to add new methods with out affecting the old interfaces.

Q58) Explain Interface seggregation principle?

Q59) Can we create instance of interface?

No, we cannot create instance of interface.

Q.60) can we do multiple inheritance with Abstract classes?

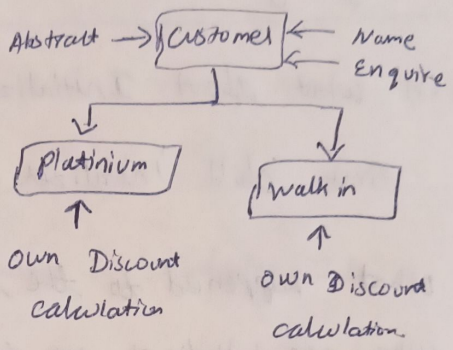
No, we cannot do multiple inheritance with Abstract classes, it is possible with interface.

Difference between Abstract class and interface.

Abstract class is a Half Defined/partial base parent class.

interface are implemented

Abstract classes are inherited.



Interface are contract which leads to other benefits like unwanted impact analysis.

Interface	Abstract Class
It is a contract	It's a half-defined parent class
planning abstraction	Share common logic in child classes
Interface is implemented	Abstract class is inherited.

All the Abstract ~~cell~~ class are interface.

No multiple inheritance in Abstract Class

Technically yes Abstract Class can become a interface but in complicated Scenarios it can be a design mistake.

Abstract class with out anything half defined can be a architecture mistake.

---

point to be remembered in interview

1. Interface is a contract
2. Interface helps to identify Abstraction
3. Abstraction class are partially defined parent classes
4. Interface are Implemented, Abstract classes are Inherited.
5. yes Abstract class can be pulled to make it look like a interface. But somewhere down the line you can have design issue.

Q62) why do we need constructors?

A constructor is special method of class, which get automically invoked when ever instance of a ~~created~~ class is created.

Q63) In parent child which constructor fires first?

logically parent constructor fire first.

Q64) What about Initializers?

first child initalizer Run, then second child initalizer Run.

what happened to the logical thinking?
why are initalizer not following logic? } → Home work.

Q65) How are static constructors executed in parent child?

First static constructor of child first, then parent static constructor

---

Q66) when does static ~~constructor~~ constructor fires?

any thing of the class is accessed, then static constructor called first.

---

why do need constructors?

constructor are special method which fire when first time instance is created. and can be used write ~~initia~~ initialization logic

---

In parent child which constructor fires first?

→ First parent fires then child

How are initalizers executed?

→ First child then parent

How are static constructors executed in parent child?

→ First child then parent

When does static constructor fires?

When first time the class is accessed.

Q70) what is shadowing?

Shadowing is a process where child methods/ properties/ function etc can be hidden from parent during polymorphism.

```
class S
{
  public virtual decimal calculateDiscount()
  {
    return 0;
  }
}
class y : S
{
  public new decimal calculateDiscount()
  { throw new exception ("no discount"); }
}
```

Q71) Explain method hiding?

method Hiding is a just a different name to shadowing. Its a synonym for shadowing.

Q72) shadowing vs overriding?

In overriding polymorphism is followed parent object will invoke child methods which are overridden. while incase of shadowing the parent object will not see/invoke the shadowed methods during polymorphism.

Q73) when do we need shadowing?

Shadowing is a reactive measure which developers implement when the child classes do not implement all the methods of the parent and we can have abnormal behaviour. This is also termed as Liskov problem. and it happens due to wrong abstraction.

SOLID represent 5 design principles.

S:- Single Responsibility principle (SRP)

O:- open close principle (OCP)

L:- Liskov Substitution principle (LSP)

I:- Interface seggregation principle (ISP)

D:- Dependency Inversion (DI)

Q74) Explain sealed classes?

A sealed class is a class which can not be inherited any further.

Q75) can we create instance of sealed classes?

yes.

Q76) what are nested classes and when to we them?

nested class is a class inside a class. It used in two scenarios

- for logically grouping the classes
- when a class you are using is only useful for the nested class.

---

solid.

Solid represent 5 design principles.

S:- Single Responsibility principle (SRP)

O:- open close principle(ocp)

L:- Liskov Substitution Principle(LSP)

I:- Interface ~~Segtal~~ Segg Segregation principle(ISP)

D:- Dependency Inversion (DI)

Q83) what is the goal of solid?

The main goal of solid is to minimize dependancies thus making code understand, maintain, and extend.

Q84) Explain SRP with a example?

A class should have one and only purpose.

Q85) what is the benefit of SRP?

Creates modular and focussed classes and thus improving quality of code.

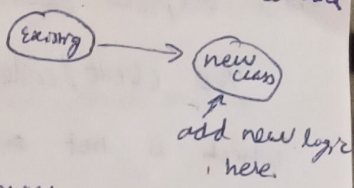
class should have one and only intent and should focus on his intent.

Q86) Explain OCP with a examples

open closed principle says that class should be opened for extension and close for modification

make it virtual and create new class & add logic.

modification ✗ not allowed



Q87) what is the benefit of OCP?

Reduces impact changes as you are now making changes in the inherited classes.

Q88) Can you explain LISKOV principle and its violation?

"Try to create a simplified definition in your own way. Do not try to imitate others definition."

Liskov principle says that child class should be cable to substitute the parent class seamlessly during object polymorphism.

Parent object should replace child object seamLessly without side effects.

```
{
  Customer c = new Customer()
  C.Amount = 1000;
     C = new premiumCustomer();
     var dis = c.calculateDiscount();
     c = new EnquiryCustomer();
     dis = c.calculateDiscount();
}
```

If all the child class has implemented all the properties of parent then it will be no liskov problem.

Q89) How can we fix Liskov problem?

Liskov problem stems when requirement is not clear, or understanding is not clear. there is a confusion of family Abstraction.

Liskov problem can be avoided by understanding requirement clearly.

If Liskov issue has already happened then you will need to refactor the class.

Q90) Explain Interface segregation principle?

No client/code should be forced to depend on method/properties which is not concerned with them.

Abstraction :. show only what is necessary.

Split the Interfaces.

Q91) Is there a connection between Liskov and Isp?

Liskov is more related to inheritance where we have grouped class in a wrong family.

Due to which the child class is forced to implement methods which it should not.

Isp is more broad and deals with interfaces.

1. clients which consumes the classes
2. when classes are forced to implement interface methods.
3. when classes are put in wrong family.

LSP focusses on wrong inheritance.

While Isp focuses on client been forced to use Interface methods even when not needed.

Q92) Define dependency inversion?

Higher level modules should depend on lower level modules via abstraction.

module which calls the other module is termed as higher level module.

The module which gets consumed is termed as lower level module.

Any change in lower level module impacts the higher modules. (Interface)

Q93) will only dependency inversion solve decoupling problem?

we will need to move the object creation process outside the higher level module.

Q96) Why do developers move object creation outside high level mod

when a higher level module creates a instance of lower level module it leads to tight coupling.

So when you move the object creation process outside the higher level module full decoupling is achieved.

Q97) explain Ioc (Inversion of control)?

we invert the object creation control outside the higher module.

This is a synonym of SRP as well.

Q98) Explain Dependency Injection with an example?

Dependency injection is a process where we inject dependent object from outside.

Decoupled = Dependency Injection + Dependency Inversion.

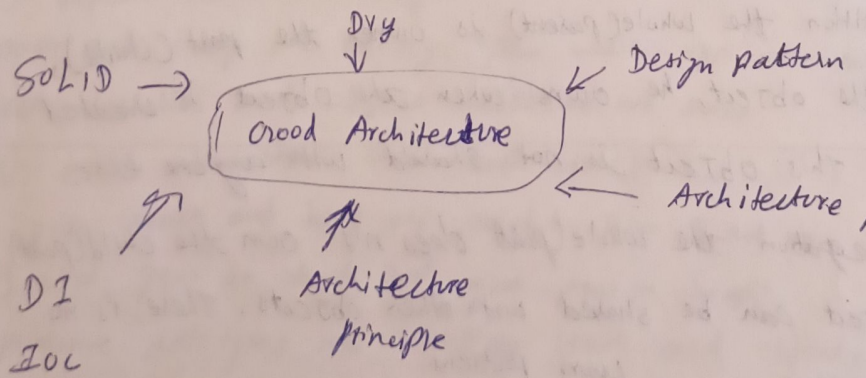
Q99) Is SOLID, Ioc and DI design pattern (or) principle?

SOLID is a principle

Ioc is a principle

DI is a technique.

Q100) Is only SOLID Enough for good code/architecture?



C# interview Question. part 10

Aggregation, Association, and Composition.

Q101) what are the different types of "USING/HAS A" Relationship?

Inheritance :- IS A
using :- HAS A.

1. Composition
2. Aggregation
3. Association.

Q102) what is a Composition relationship?

composition and Aggregation have PART-WHOLE Relationship.

Composition is a part-whole Relationship where both part and whole object have same life time.

one can not stay without each other. It like a death Relationship.

It can be rude. but many also say its a DEATH Relationship. so this vocab with a mild tone can also impress the interviewer.

Q103) Explain Aggregation?

Aggregation is a part-whole Relationship where both part and whole object can have different life time. They can exist independently.

In composition the whole(parent) is owner the part (child), he owns the object, he owns when the object is created a destroyed. this object is not shared with anyone else.

In aggregation the whole(part) does n't own the child(part). The child object can be shared with others objects. There is no excusive ownership.

```
class patient
{
    public List<Problem> problems {get; set;}
    public Doctor Doctor {get; set;}
    public patient(Doctor _doc)
    {
        CheckBed _bed = new CheckBed();
    }
}
```

Q 104) Explain Association?

Composition and aggregation are subset of Association. [bed, bednumber]

Try to keep ~~part~~ practial, convincing examples. Do not use examples like tree, cats, houses, etc.

Try to use examples like inventory, billing, accounting etc.

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.

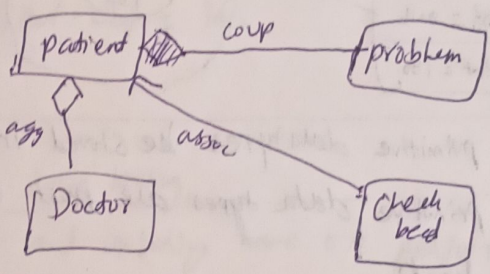
Q 105) Composition Vs Aggregation Vs Association?

Q 106) UML Symbols for composition, Aggregation and Association?

Association :- Simple arrow

Aggregation :- Empty Diamond

composition :- Filled Diamond



C# memory managment Question Garbage collections GC0, GC1, GC2 Dispose, Stack, Heap, structure and so on

(Stack, Heap, Boxing, UnBoxing, value types, reference types)

Q 107) Explain stack and heap?

Stack and Heap are memory types where application running variables, object variables and object reference are stored.

Q 108) where are stack and heap stored?

Both stack and heap are stored in RAM & not on Hard disk.

Q 109) what goes on stack and what goes on heap?

Primitive datatypes are stored on stack and objects are stored on Heap. → Partially correct.

primitive datatypes and object reference are stored on stack. Actual object data and primitive datatypes (Depending on implementation) on Heap.

Q110) How is the stack memory address arranged?

Stack memory address is contigious and memory address do not have gaps.

Q111) How is stack memory deallocated LIFO or FIFO?

Stack memory is dellocation in LIFO manner means Last in First out.

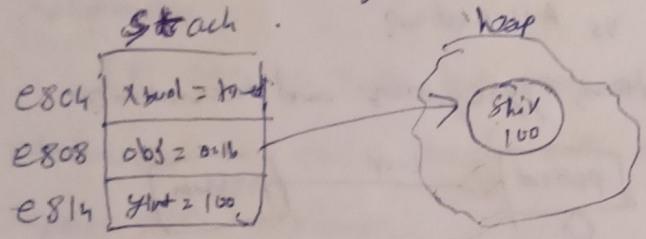
Q112) How are primitive and objective stored in memory?

~~Pri~~ Primitive → V

Object → memory. → V

In a stack primitive data types memory address points to the actual value.

In object its memory address points to an other memory address points and that memory address towards the actual object data.



Q113) Can primitive datatypes be stored in heap?

Yes if primitive data types are part of an object then it will go on heap

Stack = primitive values & objects ref pointers

Heap = Actual object data + primitive (attribution)

Q114) Explain value types and reference types.?

value types are primitive datatypes while reference types are objects.

Q115) Explain by val and by ref?

Q116) Differentiate between copy by value and copy by ref?

In by val/copy by value values are copied and fresh memory address is allocated.

while in case of by ref/copy by ref they point to same memory address and so the same reference.

Q117) what is Boxing and UnBoxing

Boxing :- when variables/objects moves from stack to Heap

UnBoxing :- when variables/objects moves from Heap to Stack

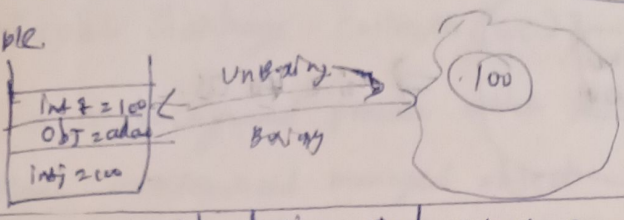
Q118) Is boxing unboxing good or bad?

No, its a bad practise as it decrease performance. So should be avoided as far as possible.

{ int inti = 100;

Boxing → object obj = inti;

Unboxing → int intz = Convert.ToInt32(obj);



Q119) can we avoid boxing and unboxing?

Int age = convert.ToInt32(Console.ReadLine());

Boxing and unBoxing can not be avoided completely. Situations like taking data from UI or binding to Grid will have to go through boxing and unBoxing. But uncessary doing boxing and unboxing is a very bad practise

```
for (int i=0; i<10000; i++)
{
    int inti = 100;
    int z = inti;  →  object obj = inti
}
```

4000 - 7000 → Boxing

1000 - 2000 → do not have boxing.

Q120) what effect does boxing and unboxing have on performance?

Boxing and unBoxing decrease performance as data has to be moved from stack to heap and heap to stack

Can we avoid boxing and unboxing?

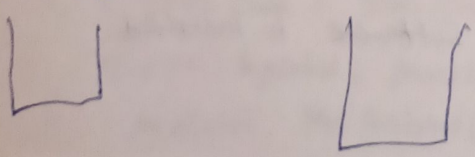
Boxing and unBoxing can not be avoided completely. Situations like taking data from UI or binding to Grid will have to go through boxing and unboxing. But unnecessarily doing boxing and unboxing is very bad practice

Q121) Are string allocated on stack or heap?

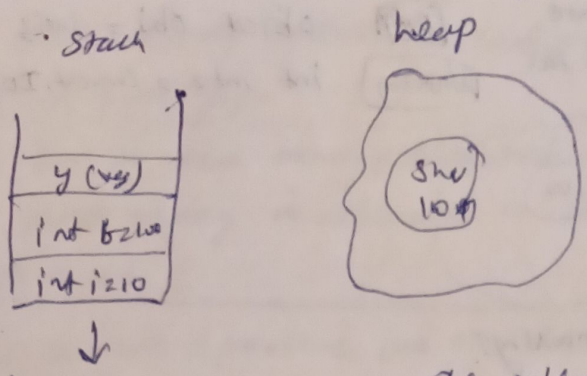
String are objects and they have reference on stack and actual data is allocated on heap.

Q122) How many stack and heaps are created for an application.

One stack — Thread — One heap and per thread one stack is created.



Q123) How are stack and heap memory deallocated?



Q125) where is structure allocated Stack or heap?

Q126) are structures copy by val or copy by ref?

Q127) Can structures get created on Heap?

GC, Generations, GC 0, 1, 2, Dispose pattern, memory leaks, weak reference

Q128) Explain Garbage Collector (GC)?

It's a background process which Run indeterministically and clears unreferenced managed objects from memory

---

Q129) 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.

Q130) Is there a way we can see this Heap memory?

Yes we can analyse the 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

performance monitor to check gc other from stats.

C# Dev do not worry about GC.

Q128) Explain Garbage collector (GC)?

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

Q129) 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.

Q130) 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,

Window search for → "performance monitor"

Visual studio → debug → performance Profile

Q131) 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 variable goes out of scope.

Q132) managed vs unmanaged code/objects/resources?

managed resources are those which are pure .NET objects and these objects are controlled 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.

```
{ objCls = "Hello";
SomeClass x = new SomeClass();
x.SomeData = DateTime.NOW.ToString();
IntPtr hglobal = Marshal.AllocHGlobal(100);
Marshal.FreeHGlobal(hglobal);
```

Q133) can garbage collector clean unmanaged objects?

No, GC only cleans managed objects.

Total memory = managed + unmanaged

Q134) explain Generations?

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

Q135) what is GC0, GC1, and GC2?

GC0:- Short lived objects ex:- local objects.

GC1:- Intermediate lived objects. (Buffers)

GC2:- long lived objects: Ex:- static objects.

Q136) 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.

Q137) which is the best place to clean unmanaged objects?

Destructor is the best place to clean unmanaged objects.

Q138) How does GC behave when we have a destructor? [· uses pes counter]

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.

Q138) what do you think about empty destructor?

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

Q139) Explain Dispose pattern?

IDispose in terface need to add.

GC.SuppressFinalized(this);

Q140) Finalized Vs Destructor?

finalize and Destructor are one the same Destructor calls the finalized method.

Summary Answer of Dispose pattern

In Dispose pattern we implement "IDisposable" interface and call "GC.SuppressFinalize()" to tell the GC that the unmanaged object has been cleaned up and claim the object do not wait.

Q141) What is the use of using keyword?

Using statement defines a scope at the end of the scope "Dispose()" is called automatically

---

Q142) can you force Garbage collector?

yes, you can by calling "GC.Collect()".

Q143) Is it a good practise to force GC?

"GC" run depending on various Criteria's like is memory running low, is processor getting overloaded and its does its work wonderfully

Fiddling with GC is not recommended at all

Q144) How can we detect a bad memory? {.net object allocation task}

Q145) How can we know the exact source of memory issues?

memory issue can be ~~detected~~ by running tools like visual studio profiles. 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 its a healthy sign. Also the memory allocation and deallocation should be balance. ? allocation and no deallocation is other sign that there is serious

Q145) 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

Q146) 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 exists.

Q147) 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.

Q148) 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 its means there is unmanaged

Q149) explain weak and strong references?

weak references:- It permits the GC to collect the object but still allows to access the object until GC collects. we need to use the "weak reference" object to create weak reference

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

```
Static WeakReference weakRef = new WeakReference(null);
Static void Fun()
{
SomeClass t = new SomeClass();
weakRef.Target = t;
}
```

Console.WriteLine(weakRef.IsAlive);

Q150) When will you use weak references?

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

Some topics in my mind

1. Access modifiers
2. Design patterns
3. String vs string Buffer, yield, Var Vs Dynamic, collections and Threading questions.