1. What is serialization and deserialization? Where you used in your project?

Serialization : When developing smaller applications that do not have a database (or other formal storage mechanism) or data that doesn't need to be stored in a database (such as the state of a web application), you often still would like to save the data for later retrieval. There are many ways to do this, but many of them are subject to a lot of extra code (work) and extra time spent debugging. With .NET, there is now an easy way to add this functionality to your code with only a few lines of easily tested code. This easy way is called serialization.

1. What are design patterns? Explain any one.

**Creational Design Pattern**

* 1. Factory Method
  2. Abstract Factory
  3. Builder
  4. Prototype
  5. Singleton

1. **Structural Design Patterns**
   1. Adapter
   2. Bridge
   3. Composite
   4. Decorator
   5. Façade
   6. Flyweight
   7. Proxy
2. **Behavioral Design Patterns**
   1. Chain of Responsibility
   2. Command
   3. Interpreter
   4. Iterator
   5. Mediator
   6. Memento
   7. Observer
   8. State
   9. Strategy
   10. Visitor
   11. Template Method

Factory: As with other design patterns, there are countless variations of the Factory pattern, although most variants typically used the same set of primary actors, a client, a factory, and a product.

Client uses factory creates product.

1. What are the differences b/w Abstract and Interface?
2. What are the differences b/w Overload and Override?

Method Overloading means creating multiple methods in the class having same name but different signatures (Parameters). It permits a class, struct, or interface to declare multiple methods with the same name with unique signatures.

Method Overriding means having two methods with same name and same signature [parameters], one should be in base class and other method should be in derived class [child class]. You can override the functionality of base class to create a same name method with same signature in derived class. You can achieve method overriding using inheritance. Virtual and Override keywords are used to achieve method overriding.

1. What is Idispose method?

It can be used to free unmanaged resources (when you implement it) like files, database connections etc. held by an object before that object is destroyed. Explicitly, it is called by user code and the class which is implementing **dispose method**, must has to implement IDisposable interface

1. How foreach loop works? Explain its internal process.

. In .NET, a collection implementing IEnumerable must implement a method called GetEnumerator which returns an IEnumerator.  
  
IEnumerator actually represents the cursor which points to a certain logical index of the collection. Lets see the IEnumerator interface.

public interface IEnumerator<out T>{ T Current { get; } bool MoveNext(); void Reset();}

Hence the IEnumerator actually keeps track of the state of the collection, which allows you to use the method MoveNext to point to the next location, Current holds the current value of the collection and Reset allows you to re-initialize the cursor.   
  
IEnumerable on the other hand wraps the enumerator into it such that any collection which implements it, can use the foreach loop. I will discuss in-depth about how to generate Enumerable in my next post.   
  
Now let us take an example to demonstrate what is happening inside of a foreach loop :

IEnumerable<int> enumerable = Enumerable.Range(1, 100); foreach (int e in enumerable) { }

The above code actually generates a IEnumerable (collection) of 99 elements from 1 through 100. We loop through the enumerable to get each integer and perform the action on it. Now this is not that simple. Foreach loop is actually broken into the following code after it is been compiled to IL.

var enumerable = Enumerable.Range(1, 100);IEnumerator<int> enumerator = enumerable.GetEnumerator();try{ while (enumerator.MoveNext()) { int element = enumerator.Current; //here goes your action instructions }}finally{ IDisposable disposable = enumerator as System.IDisposable; if (disposable != null) disposable.Dispose();}

I I think this looks great to you. a foreach loop is actually an abstraction to what we see above. It first finds the enumerator using GetEnumerator from the enumerable, uses MoveNext to loop through the instructions and in the finally block, it tries to dispose the enumerator.

Similar to what I showed in the code above you can see the IL to correspond to the same rule. The try / finally ensures that enumerator is disposed after the foreach loop is complete or even an exception occurs.

* What is the use of Private Constructor? Where exactly we can use this?

For example if you have a class that should only be created through factory methods. Or if you have overloads of the constructor, and some of them should only be used by the other constructors. Probably other reasons as well

* Can a class be inherited or instantiated if we use private constructor ?( Ans : No)
* What is the use of Static access specifier?

A static member cannot be referenced through an instance. Instead, it is referenced through the type name.

* How many times a static constructor will be loaded when the class is initialized?
* While an instance of a class contains a separate copy of all instance fields of the class, there is only one copy of each static field.
* What are the differences between Finalize and Dispose?

Dispose() is called when we want for an object to release any unmanaged resources with them. On the other hand Finalize() is used for the same purpose but it doesn't assure the garbage collection of an object.

* What is the use of garbage collector?

When any process gets triggered, separate virtual space is assigned to that process, from a physical memory which is the same and used by every process of a system, any program deals with virtual space not with physical memory, GC also deals with the same virtual memory to allocate and de-allocate memory. Basically, there are free-blocks that exist in virtual memory (also known as holes), when any object request for memory allocation manager searches for free-block and assigns memory to the said object.

Virtual memory has three blocks:

* Free (empty space)
* Reserved (already allocated)
* Committed (This block is give-out to physical memory and not available for space allocation)
* What are the generations of Garbage collector?

Generations

A generational garbage collector collects the short-lived objects more frequently than the longer lived ones. Short-lived objects are stored in the first generation, generation 0. The longer-lived objects are pushed into the higher generations, 1 or 2. The garbage collector works more frequently in the lower generations than in the higher ones.

When an object is first created, it is put into generation 0. When the generation 0 is filled up, the garbage collector is invoked. The objects that survive the garbage collection in the first generation are promoted onto the next higher generation, generation 1. The objects that survive garbage collection in generation 1 are promoted onto the next and the highest generation, generation 2. This algorithm works efficiently for garbage collection of objects, as it is fast. Note that generation 2 is the highest generation that is supported by the garbage collector.

* Can Finalize () method called directly by the user code?(Ans : No)
* Which is better among Finalize () and Dispose () and why?
* What is inheritance and its types? Will Dotnet supports multiple inheritance, if so how?
* Difference between static polymorphism and dynamic polymorphism?
* Which method is to be called is decided at compile-time only. Method overloading is an example of this. Method overloading is a concept where we use the same method name many times in the same class, but different parameters. Depending on the parameters we pass, it is decided at compile-time only. The same method name with the same parameters is an error and it is a case of duplication of methods which C# does not permit. In Static Polymorphism the decision is made at compile time.
* Run time polymorphism is also known as method overriding. In this mechanism by which a call to an overridden function is resolved at a Run-Time (not at Compile-time) if a base Class contains a method that is overridden. Method overriding means having two or more methods with the same name, same signature but with different implementation. In this process, an overridden method is called through the reference variable of a superclass, the determination of the method to be called is based on the object being referred to by reference variable.
* Difference between Read-only, constant and static modifiers?
* **Constant** fields are defined at the time of declaration in the code snippet, because once they are defined they can't be modified. By default a constant is static, so you can't define them static from your side.  
    
  It is also mandatory to assign a value to them at the time of declaration otherwise it will give an error during compilation of the program snippet. That's why it is also called a compile-time constant.
* Readonly  
    
  The readonly keyword is different from the [const](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/const) keyword. A const field can only be initialized at the declaration of the field. A readonlyfield can be initialized either at the declaration or in a constructor. Therefore, readonly fields can have different values depending on the constructor used. Also, while a const field is a compile-time constant, the readonly field can be used for runtime constants
* Static   
    
  The static keyword is used to declare a static member. If we are declare a class as a static class then in this case all the class members must be static too. The static keyword can be used effectively with classes, fields, operators, events, methods and so on effectively.
* What is delegate and how many types of delegates does Dotnet support?

There are three types of delegates that can be used in C#.

* Single Delegate
* Multicast Delegate
* Generic Delegate
* func action predicate

### **Single Delegate**

Single delegate can be used to invoke a single method. In the given source code example, a delegate CalculateSimpleInterest invokes a method getTotalInterest().

### **Multicast Delegate**

Multicast delegate can be used to invoke the multiple methods. The delegate instance can do multicasting (adding new method on existing delegate instance) using the + operator and – operator can be used to remove a method from a delegate instance. All methods will invoke in sequence as they are assigned.

There are three types of generic delegates:

* Func
* Action
* Predicate
* Difference between abstract class and interface with example

| Feature | Interface | Abstract class |
| --- | --- | --- |
| Multiple inheritance | A class may inherit several interfaces. | A class may inherit only one abstract class. |
| Default implementation | An interface cannot provide any code, just the signature. | An abstract class can provide complete, default code and/or just the details that have to be overridden. |
| Access Modfiers | An interface cannot have access modifiers for the subs, functions, properties etc everything is assumed as public | An abstract class can contain access modifiers for the subs, functions, properties |
| Core VS Peripheral | Interfaces are used to define the peripheral abilities of a class. In other words both Human and Vehicle can inherit from a IMovable interface. | An abstract class defines the core identity of a class and there it is used for objects of the same type. |
| Homogeneity | If various implementations only share method signatures then it is better to use Interfaces. | If various implementations are of the same kind and use common behaviour or status then abstract class is better to use. |
| Speed | Requires more time to find the actual method in the corresponding classes. | Fast |
| Adding functionality (Versioning) | If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method. | If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly. |
| Fields and Constants | No fields can be defined in interfaces | An abstract class can have fields and constrants defined |

* What are anonymous methods?
* C# **Anonymous Method**: As the name suggests, an **anonymous method** is a**method** without a name. **Anonymous methods** in C# can be defined using the**delegate** keyword and can be assigned to a variable of **delegate** type.
* What is the use of yield specifier?

“Yield keyword helps us to do custom stateful iteration over .NET collections.”

There are two scenarios where “yield” keyword is useful:-

* Customized iteration through a collection without creating a temporary collection.
* Stateful iteration.
* What are the difference between ienumerable and ienumerator?

IEnumerable and IEnumerator are both interfaces. IEnumerable has just one method called GetEnumerator.

IEnumerator has 2 methods (MoveNext and Reset) and a property Current.

* What are the difference between iqueryable and ienumerable?

Ienumerable IEnumerable is best suitable for working with in-memory collection. IEnumerable doesn’t move between items, it is forward only collection.

IQueryable IQueryable best suits for remote data source, like a database or web service. IQueryable is a very powerful feature that enables a variety of interesting deferred execution scenarios (like paging and composition based queries).

So when you have to simply iterate through the in-memory collection, use IEnumerable, if you need to do any manipulation with the collection like Dataset and other data sources, use IQueryable

* What is the use of partial classes and partial methods?

# **Fundamentals of partial classes**

A partial class allows a single class to be divided into two separate physical files. During compile time these files get compiled into a single class. For instance you can see in the following figure we have the customer class divided into two different files "customer1.cs" and "customer2.cs".

During compilation these files gets compiled into a single class internally. So when you create an object of the customer class you will be able to see methods lying in both the physical files. For instance you can see the "Add" method belongs to "customer1.cs" and the "Delete" method belongs to "customer2.cs" , but when the customer object is created we can see both "Add" and "Delete" methods.

**Fundamentals of partial methods**

There is one more important concept in partial classes called partial methods. Partial methods helps us to define a method definition in one of the physical files and we can implement that method in the other physical files as shown in the following figure.

In the following figure you can see we have defined the "Validate" method in "Customer1.cs" and this validate method is implemented in "Customer2.cs". Please note the partial keywords attached to both of these methods.

* What are the new features added in C# 5.0(Ans: Async and Await, caller information attributes)
* Can Catch block invoke multiple exception types? (Ans: No)
* How to implement the same methods present in the two interfaces? (Ans : Use Explicit Interface Implementation)
* What is Lambda expression and what are its uses?
* Lambda expressions are how anonymous functions are created.  
    
  Lambda expressions are anonymous functions that contain expressions or sequence of operators. All lambda expressions use the lambda operator =>, that can be read as “goes to” or “becomes”.   
    
  The left side of the lambda operator specifies the input parameters and the right side holds an expression or a code block that works with the entry parameters. Usually lambda expressions are used as predicates or instead of delegates (a type that references a method).  
    
  Expression Lambdas

Parameter => expressionParameter-list => expressionCount => count + 2;Sum => sum + 2;n => n % 2 == 0

* What is Linq?
* A class and a method has an integer variable declared? Where exactly the value will be stored (Stack or Heap)? Heap. A Reference Type always goes on the Heap
* What are extension methods and its uses?
* Extension methods enable you to add methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.   
    
  An extension method is a special kind of static method, but they are called as if they were instance methods on the extended type.  
    
  **How to use extension methods?**  
    
  An extension method is a static method of a static class, where the "this" modifier is applied to the first parameter. The type of the first parameter will be the type that is extended.  
    
  Extension methods are only in scope when you explicitly import the namespace into your source code with a using directive.
* Caching and its types?
* What is fragment caching?
* Can an application contains multiple web.config files?
* What the difference is between debug and release modes?
* 6.       How to improve Performance Application side and SQL side.

1. .       What is extension Method
2. How to optimize the performance of a web page if it is slow?
3. What are the page life cycle events? When view state is invoked?
4. Stage 1 – Initialization

Stage 2 - Load View State

Stage 3 - Load Postback Data

Stage 4 – Load

Stage 5 - Raise Postback Event

Stage 6 - Save View State

Stage 7 – Render