Hello All, We are going to start new batch from next week. message/call or mail us for m

## C Sharp Interview Questions/Answers Part-1

**What are the new features introduced in c# 4.0?**
This question is basically asked to check, if you are passionate about catching up with latest technological advancements. The list below shows a few of the new features introduced in c# 4.0. If you are aware of any other new features, please submit those using the from at the end of this post.
**1.** Optional and Named Parameters
**2.** COM Interoperability Enhancements
**3.** Covariance and Contra-variance
**4.** Dynamic Type Introduction

**What's the difference between IEnumerable<T> and List<T> ?**
**1. IEnumerable** is an interface, where as **List** is one specific implementation of IEnumerable. List is a class.
**2.** FOR-EACH loop is the only possible way to iterate through a collection of IEnumerable, where as List can be iterated using several ways. List can also be indexed by an int index, element can be added to and removed from and have items inserted at a particular index.
**3.** IEnumerable doesn't allow random access, where as List does allow random access using integral index.
**4.** In general from a performance standpoint, iterating thru IEnumerable is much faster than iterating thru a List.

**Difference between EXE and DLL?**
**1.** .EXE is an executable file and can run by itself as an application, where as .DLL is usullay consumed by a .EXE or by another .DLL and we cannot run or execute .DLL directly.
**2.** For example, In .NET, compiling a Console Application or a Windows Application generates .EXE, where as compiling a Class Library Project or an ASP.NET web application generates .DLL. In .NET framework, both .EXE and .DLL are called as assemblies.

**What are the difference between interfaces and abstract classes?**
**1.** Abstract classes can have implementations for some of its members, but the interface can't have implementation for any of its members.
**2.** Interfaces cannot have fields where as an abstract class can have fields.
**3.** An interface can inherit from another interface only and cannot inherit from an abstract class, where as an abstract class can inherit from another abstract class or another interface.
**4.** A class can inherit from multiple interfaces at the same time, where as a class cannot inherit from multiple classes at the same time.
**5.** Abstract class members can have access modifiers where as interface members cannot have access modifiers.

**What is a delegate?**
A delegate is a type safe function pointer. Using delegates you can pass methods as parameters. To pass a method as a parameter, to a delegate, the signature of the method must match the signature of the delegate. This is why, delegates are called type safe function pointers.

**What is the main use of delegates in C#?**
Delegates are mainly used to define call back methods.

**What do you mean by chaining delegates?**
Or
**What is a multicast delegate?**
The capability of calling multiple methods on a single event is called as chaining delegates. Let me give you an example to understand this further.
**1.** Create a new asp.net web application
**2.** Drag and drop a button control and leave the ID as Button1.
**3.** On the code behind file, add the code shown below.

```
protected void Page_Load(object sender, EventArgs e)
{
    Button1.Click += new EventHandler(Method1);
    Button1.Click += new EventHandler(Method2);
}

protected void Method1(object sender, EventArgs e)
{
    Response.Write("Method1 Called on Button1 Click");
}

protected void Method2(object sender, EventArgs e)
{
    Response.Write("Method2 Called on Button1 Click");
}
```

When you click the Button now, both Method1 and Method2 will be executed. So, this capability of calling multiple methods on a single event is called as chaining delegates. In the example, we are using EventHandler delegate, to hook up Method1 and Method2 to the click event of the button control. Since, the EventHandler delegate is now pointing to multiple methods, it is also called as multicast delegate.

**What are the advantages of using interfaces?**
Interfaces are very powerful. If properly used, interfaces provide all the advantages as listed below.
**1.** Interfaces allow us to implement polymorphic behaviour. Ofcourse, abstract classes can also be used to implement polymorphic behaviour.
**2.** Interfaces allow us to develop very loosely coupled systems.
**3.** Interfaces enable mocking for better unit testing.
**4.** Interfaces enables us to implement multiple class inheritance in C#.
**5.** Interfaces are great for implementing Inverson of Control or Dependancy Injection.
**6.** Interfaces enable parallel application development.

**What are the advantages and disadvantages of using arrays?**
Advantages of using arrays:
**1.** Arrays are strongly typed, meaning you can only have one type of elements in the array. The strongly typed nature of arrays gives us
**2** advantages. One, the performance will be much better because boxing and unboxing will not happen. Second, run time errors can be prevented because of type mis matches. Type mis matches and runtime errors are most commonly seen with collection classes like ArrayList, Queue, Stack etc, that are present in System.Collections namespace.
**Disadvantages of using arrays:**
**1.** Arrays are fixed in size and cannot grow over time, where ArrayList in System.Collections namespace can grow dynamically.
**2.** Arrays are zero index based, and hence a little difficult to work with. The only way to store or retrieve elements from arrays, is to use integral index. Arrays donot provide convinient methods like Add(), Remove() etc provided by collection classes found in System.Collections or System.Collections.Generics namespaces, which are very easy to work with.

**Explain what is an Interface in C#?**
An Interface in C# is created using the interface keyword. An example is shown below.

**Code: [Select all]**

```
using System;
namespace Interfaces
{
interface IBankCustomer
{
void DepositMoney();
void WithdrawMoney();
}
public class Demo : IBankCustomer
{
public void DepositMoney()
{
Console.WriteLine("Deposit Money");
}

public void WithdrawMoney()
{
Console.WriteLine("Withdraw Money");
}
public static void Main()
{
Demo DemoObject = new Demo();
DemoObject.DepositMoney();
DemoObject.WithdrawMoney();
}
}
}
```

**Can an Interface contain fields?**
No, an Interface cannot contain fields.

**Can an interface inherit from another interface?**
Yes, an interface can inherit from another interface. It is possible for a class to inherit an interface multiple times, through base classes or interfaces it inherits. In this case, the class can only implement the interface one time, if it is declared as part of the new class. If the inherited interface is not declared as part of the new class, its implementation is provided by the base class that declared it. It is possible for a base class to implement interface members using virtual members; in that case, the class inheriting the interface can change the interface behavior by overriding the virtual members.

**Can you create an instance of an interface?**
No, you cannot create an instance of an interface.

**What is a partial class. Give an example?**
A partial class is a class whose definition is present in 2 or more files. Each source file contains a section of the class, and all parts are combined when the application is compiled. To split a class definition, use the partial keyword as shown in the example below. Student class is split into 2 parts. The first part defines the study() method and the second part defines the Play() method. When we compile this program both the parts will be combined and compiled. Note that both the parts uses partial keyword and public access modifier.

**Code: [Select all]**

```
using System;
namespace PartialClass
{
  public partial class Student
  {
    public void Study()
    {
      Console.WriteLine("I am studying");
    }
  }
  public partial class Student
  {
    public void Play()
    {
      Console.WriteLine("I am Playing");
    }
  }
  public class Demo
  {
    public static void Main()
    {
      Student StudentObject = new Student();
      StudentObject.Study();
      StudentObject.Play();
    }
  }
}
```

**It is very important to keep the following points in mind when creating partial classes.**
1. All the parts must use the partial keyword.
2. All the parts must be available at compile time to form the final class.
3. All the parts must have the same access modifiers - public, private, protected etc.
4. Any class members declared in a partial definition are available to all the other parts.

5. The final class is the combination of all the parts at compile time.

**What are the advantages of using partial classes?**
1. When working on large projects, spreading a class over separate files enables multiple programmers to work on it at the same time.

2. When working with automatically generated source, code can be added to the class without having to recreate the source file. Visual Studio uses this approach when it creates Windows Forms, Web service wrapper code, and so on. You can create code that uses these classes without having to modify the file created by Visual Studio.

**Is it possible to create partial structs, interfaces and methods?**
Yes, it is possible to create partial structs, interfaces and methods. We can create partial structs, interfaces and methods the same way as we create partial classes.

**Can you create partial delegates and enumerations?**
No, you cannot create partial delegates and enumerations.

**Can different parts of a partial class inherit from different interfaces?**
Yes, different parts of a partial class can inherit from different interfaces.

**Can you specify nested classes as partial classes?**
Yes, nested classes can be specified as partial classes even if the containing class is not partial. An example is shown below.

**Code: [Select all]**

```
class ContainerClass
{
  public partial class Nested
  {
    void Test1() { }
  }
  public partial class Nested
  {
    void Test2() { }
  }
}
```

**How do you create partial methods?**
To create a partial method we create the declaration of the method in one part of the partial class and implementation in the other part of the partial class. The implementation is optional. If the implementation is not provided, then the method and all the calls to the method are removed at compile time. Therefore, any code in the partial class can freely use a partial method, even if the implementation is not supplied. No compile-time or run-time errors will result if the method is called but not implemented. In summary a partial method declaration consists of two parts. The definition, and the implementation. These may be in separate parts of a partial class, or in the same part. If there is no implementation declaration, then the compiler optimizes away both the defining declaration and all calls to the method.

**The following are the points to keep in mind when creating partial methods.**
1. Partial method declarations must begin partial keyword.
2. The return type of a partial method must be void.
3. Partial methods can have ref but not out parameters.
4. Partial methods are implicitly private, and therefore they cannot be virtual.
5. Partial methods cannot be extern, because the presence of the body determines whether they are defining or implementing.

**What is the use of partial methods?**
Partial methods can be used to customize generated code. They allow for a method name and signature to be reserved, so that generated code can call the method but the developer can decide whether to implement the method. Much like partial classes, partial methods enable code created by a code generator and code created by a human developer to work together without run-time costs.

**What is a nested type. Give an example?**
A type(class or a struct) defined inside another class or struct is called a nested type. An example is shown below. InnerClass is inside ContainerClass, Hence InnerClass is called as nested class.

**Code: [Select all]**

```
using System;
namespace Nested
{
  class ContainerClass
  {
    class InnerClass
    {
      public string str = "A string variable in nested class";
    }

    public static void Main()
    {
      InnerClass nestedClassObj = new InnerClass();
      Console.WriteLine(nestedClassObj.str);
    }
  }
}
```

**What is a Destructor?**
A Destructor has the same name as the class with a tilde character and is used to destroy an instance of a class.

**Can a class have more than 1 destructor?**
No, a class can have only 1 destructor.

**Can structs in C# have destructors?**
No, structs can have constructors but not destructors, only classes can have destructors.

**Can you pass parameters to destructors?**
No, you cannot pass parameters to destructors. Hence, you cannot overload destructors.

**Can you explicitly call a destructor?**
No, you cannot explicitly call a destructor. Destructors are invoked automatically by the garbage collector.

**Why is it not a good idea to use Empty destructors?**
When a class contains a destructor, an entry is created in the Finalize queue. When the destructor is called, the garbage collector is invoked to process the queue. If the destructor is empty, this just causes a needless loss of performance.

**Is it possible to force garbage collector to run?**
Yes, it possible to force garbage collector to run by calling the Collect() method, but this is not considered a good practice because this might create a performance over head. Usually the programmer has no control over when the garbage collector runs. The garbage collector checks for objects that are no longer being used by the application. If it considers an object eligible for destruction, it calls the destructor(if there is one) and reclaims the memory used to store the object.

**Usually in .NET, the CLR takes care of memory management. Is there any need for a programmer to explicitly release memory and resources? If yes, why and how?**
If the application is using expensive external resource, it is recommend to explicitly release the resource before the garbage collector runs and frees the object. We can do this by implementing the Dispose method from the IDisposable interface that performs the necessary cleanup for the object. This can considerably improve the performance of the application.

**When do we generally use destructors to release resources?**
If the application uses unmanaged resources such as windows, files, and network connections, we use destructors to release resources.

**What is a constructor in C#?**
Constructor is a class method that is executed when an object of a class is created. Constructor has the same name as the class, and usually used to initialize the data members of the new object.

**In C#, What will happen if you do not explicitly provide a constructor for a class?**
If you do not provide a constructor explicitly for your class, C# will create one by default that instantiates the object and sets all the member variables to their default values.

**Structs are not reference types. Can structs have constructors?**

Yes, even though Structs are not reference types, structs can have constructors.

**We cannot create instances of static classes. Can we have constructors for static classes?**
Yes, static classes can also have constructors.

**Can you prevent a class from being instantiated?**
Yes, a class can be prevented from being instantiated by using a private constructor as shown in the example below.

**Code: [Select all]**

```
using System;
namespace TestConsole
{
  class Program
  {
    public static void Main()
    {
      //Error cannot create instance of a class with private constructor
      SampleClass SC = new SampleClass();
    }
  }
  class SampleClass
  {
    double PI = 3.141;
    private SampleClass()
    {
    }
  }
}
```

**Can a class or a struct have multiple constructors?**
Yes, a class or a struct can have multiple constructors. Constructors in csharp can be overloaded.

**Can a child class call the constructor of a base class?**
Yes, a child class can call the constructor of a base class by using the base keyword as shown in the example below.

**Code: [Select all]**

```
using System;
namespace TestConsole
{
  class BaseClass
  {
    public BaseClass(string str)
    {
      Console.WriteLine(str);
    }
  }

  class ChildClass : BaseClass
  {
    public ChildClass(string str): base(str)
    {
    }

    public static void Main()
    {
      ChildClass CC = new ChildClass("Calling base class constructor from child class");
    }
  }
}
```

**If a child class instance is created, which class constructor is called first - base class or child class?**

When an instance of a child class is created, the base class constructor is called before the child class constructor. An example is shown below.

**Code: [Select all]**

```
using System;
namespace TestConsole
{
  class BaseClass
  {
    public BaseClass()
    {
      Console.WriteLine("I am a base class constructor");
    }
  }
  class ChildClass : BaseClass
  {
    public ChildClass()
    {
      Console.WriteLine("I am a child class constructor");
    }
    public static void Main()
    {
      ChildClass CC = new ChildClass();
    }
  }
}
```

**Can a class have static constructor?**
Yes, a class can have static constructor. Static constructors are called automatically, immediately before any static fields are accessed, and are generally used to initialize static class members. It is called automatically before the first instance is created or any static members are referenced. Static constructors are called before instance constructors. An example is shown below.

**Code: [Select all]**

```
using System;
namespace TestConsole
{
  class Program
  {
    static int I;
    static Program()
    {
      I = 100;
      Console.WriteLine("Static Constructor called");
    }
    public Program()
    {
      Console.WriteLine("Instance Constructor called");
    }
    public static void Main()
    {
      Program P = new Program();
    }
  }
}
```

**Can you mark static constructor with access modifiers?**
No, we cannot use access modifiers on static constructor.

**Can you have parameters for static constructors?**
No, static constructors cannot have parameters.

**What happens if a static constructor throws an exception?**
If a static constructor throws an exception, the runtime will not invoke it a second time, and the type will

remain uninitialized for the lifetime of the application domain in which your program is running.

**Give 2 scenarios where static constructors can be used?**
1. A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.
2. Static constructors are also useful when creating wrapper classes for unmanaged code, when the constructor can call the LoadLibrary method.

**Does C# provide copy constructor?**
No, C# does not provide copy constructor.

Email ThisBlogThis!Share to TwitterShare to FacebookShare to Pinterest
G+1 Recommend this on Google

## 29 comments:

1.
   *Jesica Paul*27 August 2015 at 13:53

   Thanks for sharing use interview questions on .Net technology. While preparing for my job interview, your article helped me a lot to sharpen my skills and do well in my interview. One of trainer from leading dot net training institutes in Chennai suggests me about your site.

   ReplyDelete
   Replies

   1.
      *dev prog*7 June 2016 at 15:50

      it's ok to show some appreciation and say 'great post'
      Asp .NET developer

      Delete

   Reply

2. *Anonymous*8 September 2015 at 10:04

   Tks very much for your post.

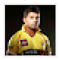
   Avoid surprises — interviews need preparation. Some questions come up time and time again — usually about you, your experience and the job itself. We've gathered together the most common questions so you can get your preparation off to a flying start.

   You also find all interview questions at link at the end of this post.

   Source: Ebook Ultimate Guide To Job Interview Answers:

   Best rgs

   ReplyDelete

3. 
   *murali karthik*8 October 2015 at 18:06

   Being flagship too for Microsoft, Asp.Net is loaded with various essential tools that allow developers to build robust web applications. PHP Training in Chennai

   ReplyDelete

4.
   *SLA INSTITUTE Chennai*14 October 2015 at 15:25

   Awesome post. Thanks you very much for you post. Best Dot Net Training Institute in Chennai

   ReplyDelete

5. *Anonymous*18 November 2015 at 15:40

   can you give me the code for role based login using three tier artitecture

   ReplyDelete

6. *sreenivas*19 November 2015 at 09:09

   please give some more easy examples related to Dotnet programs

   ReplyDelete