Table of Contents

Debuting back in 2000, C# has succeeded in becoming one of the leading programming languages. As a multi-paradigm programming language, C# also has some features of functional programming (https://hackr.io/blog/functional-programming-concepts-advantages-disadvantages-applications) that takes its usefulness and versatility to a step further.

C# Interview Questions and Answers

In the following section, we have enlisted the most important C# interview questions. These questions prepare you for your next C# interview in addition to enhancing your C# knowledge and letting you evaluate your current C# understanding.

Question: What is C#? Write its features

Answer: C# is an object-oriented programming language that was developed by Microsoft in 2000. It is supported by different operating systems. C# is the primary language that is used to create .Net software applications. It allows us to create Windows UI apps, backend services, controls, libraries, android apps, and even blockchain applications. C# works on the concept of classes and objects just like Java.

Some of the C# features are as follows:

- Follows structured approach
- Parameters passing is easy
- Code can be compiled on a different platform
- Open-source
- Object-oriented
- Flexible and scalable

Question: Explain what are classes and objects in C#?

Answer: C# is an object-oriented language and classes are its foundation. A class generally depicts the structure of data, how data is stored and managed within a program. A class has its own properties, methods, and other objects that define the class.

Objects are the real-world entity having some characteristics and is created using the class instance. These classes define the type of the defined object.

For example, if we consider a program that covers the object related to the book. We call the class a Book which has two properties: name and the author. In real programming, Vedas is an object and an instance of the class Book.

Question: Describe the different C# classes in detail.

Answer: There are 4 types of classes that we can use in C#:

- **Static Class:** It is the type of class that cannot be instantiated, in other words, we cannot create an object of that class using the new keyword and the class members can be called directly using their class name.
- **Abstract Class:** Abstract classes are declared using the abstract keyword. Objects cannot be created for abstract classes. If you want to use it then it must be inherited in a subclass. You can easily define abstract or non-abstract methods within an Abstract class. The methods inside the abstract class can either have an implementation or no implementation.
- **Partial Class:** It is a type of class that allows dividing their properties, methods, and events into multiple source files, and at compile time these files are combined into a single class.
- **Sealed Class:** One cannot inherit a sealed class from another class and restricts the class properties. Any access modifiers cannot be applied to the sealed class.

Question: Explain different access modifiers in C#?

Answer: These are the keywords that help to define the accessibility of class, member, and data type in the program. These keywords are used to restrict the use of some data manipulation done by other classes. There are 4 types of access modifiers- public, private, protected, and internal. These modifiers define 6 other accessibility levels when working together- public, protected, internal, protected internal, private, and private protected.

Below is the access chart of the modifiers.

	PUBLIC	PROTECTED	INTERNAL	PROTECTED INTERNAL	PRIVATE	PRIVATE PROTECTED
Complete program	Yes	No	No	No	No	No
Derived types within the current assembly	Yes	Yes	No	Yes	No	Yes
Using class	Yes	Yes	Yes	Yes	Yes	Yes
Current assembly	Yes	No	Yes	Yes	No	No
Derived data types	Yes	Yes	No	Yes	No	No

Question: How can you describe object-oriented concepts in detail?

Answer: C# is an object-oriented programming language that supports 4 OOP concepts (https://hackr.io/blog/oops-concepts-in-java-with-examples).

- **Encapsulation**: defines the binding together code and the data and keeps it safe from any manipulation done by other programs and classes. It is a container that prevents code and data from being accessed by another program that is defined outside the container.

- **Abstraction**: this concept of object-oriented protects everything other than the relevant data about any created object in order to increase efficiency and security within the program.
- **Inheritance**: Inheritance is applied in such a way where one object uses the properties of another object.
- **Polymorphism**: is a feature that allows one interface to act as a base class for other classes. This concept is often expressed as a "single interface but multiple actions".

Question: Explain how code gets compiled in C#?

Answer: It takes 4 steps to get a code to get compiled in C#. Below are the steps:

- First, compile the source code in the managed code compatible with the C# compiler.
- Second, combine the above newly created code into assemblies.
- Third, load the CLR.
- Last, execute the assembly by CLR to generate the output.

Question: What is break and continue statements in C#, explain?

Answer: Below is the differences:

Break	Continue
You can use break statements in both switch and loop (for, while, and do-while) statements.	You can use continue statements only in the loop (for, while, do) statements.
The switch or loop statements terminate at the moment the break statement is executed and it ends abruptly from there.	You cannot make a continue statement terminate the loop, it carries on the loop to go to the next iteration level without executing the immediate next step.
The loop or switch exits immediately from the inner loop when the compiler encounters a break statement and comes out of the inner loop.	A continue that is placed inside a nested loop within a switch causes the next loop iteration.

Question: How you can explain the use of ‘using’ statements in C# in detail.

Answer: The using statement is used to control the usage of one or more resources that are being used within the program. The resources are continuously consumed and released. The main function of this statement is to manage unused resources and release them automatically. Once the object is created which is using the resource and when you are done you make sure that the object’s dispose method is called to release the resources used by that object, this is where using statements works well.

For example:

```
using (MyResource abc = new MyResource())
{
 abc.program();
}
Gets translated to,
MyResource abc= new MyResource();
try
{
 myRes.program();
}
finally
{
 // Check for a null resource.
 if (abc!= null)
     // Call the object's Dispose method.
     ((IDisposable)abc).Dispose();
}
```

Question: Describe the C# dispose of the method in detail.

Answer: Dispose of the method: The disposeof() method releases the unused resources by an object of the class. The unused resources like files, data connections, etc. This method is declared in the interface called **IDisposable** which is implemented by the class by defining the interface IDisposable body. Dispose method is not called automatically, the programmer has to implement it manually for the efficient usage of the resources.

Question: Explain in detail the finalize method in C#?

Answer: Finalize method- The finalize () method is defined in the **object** class which is used for cleanup activities. This method is generally called by the garbage collector whenever the reference of any object is not used for a long time. Garbage collector frees that managed resources automatically but if you want to free the unused resources like filehandle, data connection, etc., then you have to implement the finalize method manually.

Question: How you can define the exception handling in C#?

Answer: An exception is a raised problem that may occur during the execution of the program. Handling exceptions offers a simple way to pass the control within the program whenever an exception is raised. C# exceptions are handled by using 4 keywords and those are try, catch, finally, throw.

- **try**: a raised exception finds a particular block of code to get handled. There is no limit on the number of catch blocks that you will use in your program to handle different types of exception raised.
- **catch:** you can handle the raised exception within this catch block. You can mention the steps that you want to do to solve the error or you can ignore the error by suppressing it by the code.

- **Finally:** irrespective of the error, if you still want some set of instructions to get displayed then you can use those statements within the finally block and it will display it on the screen.
- **throw:** you can throw an exception using the throw statement. It will display the type of error you are getting.

Syntax:

```
try {
//exception handling starts with try block
} catch( ExceptionName ea1 ) {
   // errors are handled within the catch block
} catch( ExceptionName e2 ) {
   // more catch block
} catch( ExceptionName eN ) {
   // more catch block to handle multiple exception raised
} finally {
   // last block of the exception handling
}
```

Question: Explain the concept of Destructor in detail. Explain it with an example.

Answer: A destructor is a member that works just the opposite of the constructor. Unlike constructors, destructors mainly delete the object. The destructor name must match exactly with the class name just like a constructor. A destructor block always starts with the tilde (~) symbol.

Syntax:

```
~class_name()
{
//code
}
```

A destructor is **called automatically:**

1. when the program finishes its execution.
2. Whenever a scope of the program ends that defines a local variable.
3. Whenever you call the delete operator from your program.

Question: Define method overloading with example.

Answer: Method overloading allows programmers to use multiple methods but with the same name. Every defined method within a program can be differentiated on the basis of the number and the type of method arguments. It is a concept based on polymorphism.

Method overloading can be achieved by the following:

- By changing the number of parameters in the given method
- By changing the order of parameters passed to a method
- By using different data types as the passed parameters

For example:

```
public class Methodoveloading
  {
    public int sum(int a, int b)  //two int type Parameters method
    {
        return a + b;
            }
    public int sum(int a, int b,int c)  //three int type Parameters with same method s
ame as above
    {
        return a + b+c;
    }
    public float sum(float a, float b,float c,float d)  //four float type Parameters w
ith same method same as above two method
    {
        return a + b+c+d;

    }
  }
```

Question: What are the control statements that are used in C#?

Answer: You can control the flow of your set of instructions by using control statements and we majorly focus on if statements. There are a few types of if statements that we consider for making situations to control the flow of execution within a program.

These are the 4 types of if statements:

- If
- If-else
- Nested if
- If-else-if

These statements are commonly used within programs.

If statements checks for the user given condition to satisfy their programming condition. If it returns true then the set of instructions will be executed.

Syntax:

```
If(any condition)
{
//code to be executed if the condition returns true
}
```

If-else statement checks for the given condition, if the condition turns out to be false then the flow will transfer to the else statement and it will execute the else instructions. In case, the if condition turns out to be true then the if instructions will get executed.

Syntax:

```
If(condition)
{
//code to be run if the condition is true
}
Else
{
//code to be run if the if-condition is false
}
```

Nested if statement checks for the condition, if the condition is true then it will check for the inner if statement and keeps going on for the last if statement. If any of the conditions are true then it will execute the particular if instructions and stops the if loop there.

Syntax:

```
If (condition to be checked)
{
//code
If(condition 2)
{
//code for if-statement 2
}
}
```

If else-if checks for the given condition, if the condition is not true then the control will go to the next else condition, if that condition is not true it will keep on checking for next else conditions. If any of the conditions did not pass then the last else instructions will get executed.

Syntax:

```
If(condition 1 to be checked)
{
//code for condition 1
}
Else (condition 2 to be checked)
{
//code for condition 2
}
Else
{
//code will run if no other condition is true
}
```

Question: Explain the concept of boxing and unboxing of the value type and object type in C#.

Answer:

Boxing- is a process of converting a value type to an object type where value type is placed on the stack memory, and the object type is placed in the heap memory. This conversion is an implicit conversion and you can directly assign any value to an object, and C# will handle the rest of the conversion on its own.

Example:

```
public void function()
{
Int a=111;
Object b=a; //implicit conversion
Console.WriteLine(b);
}
```

Unboxing- it is the reverse process of the boxing process. It is a conversion of the object type to the value type and the value of the boxed object type placed on the heap memory which will be transferred to the value type which is placed on the stack. This conversion of the unboxing process has to be done explicitly.

Example:

```
public void function()
{
Object b=111;
Int a=(int)b; //implicit conversion
Console.WriteLine(a);
}
```

Question: How can you check if a number is an Armstrong number or not with C#?

Answer:

```
using System;
  public class ArmstrongDemo
   {
     public static void Main(string[] args)
      {
       int  n,b,sum=0,num;
       Console.Write("Enter the Number= ");
       n= int.Parse(Console.ReadLine());
       num=n;
       while(n>0)
       {
        b=n%10;
        sum=sum+(b*b*b);
        n=n/10;
       }
       if(num==sum)
        Console.Write("Armstrong Number.");
       else
        Console.Write("Not Armstrong Number.");
      }
  }
```

Output:

Enter the Number= 371
Armstrong Number.

Question: What is a different approach to the passing parameter in C#?

Answer: Parameters can be passed in three different ways to any defined methods and they are defined below:

Value Parameters: it will pass the actual value of the parameter to the formal parameter. In this case, any changes that are made into the formal parameter of the function will be having no effect on the actual value of the argument.

Reference Parameters: with this method, you can copy the argument that refers to the memory location into the formal parameter that means any changes made to the parameter affect the argument.

Output Parameters: This method returns more than one value to the method.

Question: What is a multicast delegate in C#?

Answer: A multicast delegate holds the references or addresses to more than one function at a single time. Whenever we invoke the multicast delegate, it will invoke all the other functions that are being referred by that multicast delegate. You should use the complete method signature the same as the delegate to call multiple methods. For example:

```
namespace MulticastDelegate
{
public class Rectangle
{
public void Area(double Width, double Height)
{
Console.WriteLine(@"Area is {0}", (Width * Height));
}
public void Perimeter(double Width, double Height)
{
Console.WriteLine(@"Perimeter is {0}", (2 * (Width + Height)));
}
static void Main(string[] args)
{
Rectangle rect = new Rectangle();
rect.Area(23.45, 67.89);
rect.Perimeter(23.45, 67.89);
Console.ReadKey();
}
}
}
```

Here, we created an instance of the **Rectangle** class and then called the two different methods. Now a single delegate will invoke these two methods **Area** and **Perimeter**. These defined methods are having the same signature as the defined delegates that hold the reference to these methods.

Creating multicast delegate:

```
namespace MulticastDelegateDemo
{
public delegate void RectangleDelete(double Width, double Height);
public class Rectangle
{
public void Area(double Width, double Height)
{
Console.WriteLine(@"Area is {0}", (Width * Height));
}
public void Perimeter(double Width, double Height)
{
Console.WriteLine(@"Perimeter is {0}", (2 * (Width + Height)));
}
static void Main(string[] args)
{
Rectangle rect = new Rectangle();
RectangleDelete rectDelegate = new RectangleDelete(rect.Area);
rectDelegate += rect.Perimeter;
rectDelegate(23.45, 67.89);
Console.WriteLine();
rectDelegate.Invoke(13.45, 76.89);
Console.WriteLine();
//Removing a method from delegate object
rectDelegate -= rect.Perimeter;
rectDelegate.Invoke(13.45, 76.89);
Console.ReadKey();
}
}
}
```

Question: How you can implement nullable<> types in C#? explain with the syntax of Nullable type.

Answer: In C#, you cannot put a null value directly into any variable and the compiler does not support it. So, the revised version **C# 2.0** provides you with a special feature that will assign a null value to a variable that is called as the Nullable type. You cannot make the nullable types to work with value types. Nullable value can only work with the reference types as it already has a null value. System.Nullable<T> structure creates the instance nullable type, where T defines the data type. This T contains a non-nullable value type that can be any data type you want.

Syntax

```
Nullable<data_type> variable_name=null;
```

OR

```
Datatype? variable_name=null;
```

There is no possibility that you can access the value of the nullable value type directly with assigning the value. For getting its original assigned value you have to use the method GetValueOrDefault(). If the value is null then it will provide zero as it is its default value.

Question: What do you mean by value types and reference types in C#?

Answer:

Value type:

The memory allocated for the value type content or assigned value is stored on the stack. When we create any variable, space is allocated to that variable and then a value can be assigned to that variable. Also if we want to copy the value of that variable to another variable, its value gets copied and that creates two different variables.

Reference Type:

It holds the reference to the address of the object but not the object directly. Reference types represent the address of the variable and assigning a reference variable to another does not copy the data but it creates a second copy of the reference which represents the same location on the heap as the original value. Reference values are stored on the heap and when the reference variable is no longer required it gets marked for garbage collection.

Question: What are various types of comments in C#, explain with example?

Answer: C# supports three types of comments-

1. Single line comment

Syntax: //single line

2. Multiple line comment

Syntax: /* multiple lines

*/

3. XML comment

Syntax: /// set error

Question: What are the constructors?

Answer: In C#, there is a special method that is invoked automatically at the time of object creation. It initializes the data members of a new object and has the same name as the class or the structure. There are two types of constructors:

- **Default constructor**: it has no parameter to pass.
- **Parameterized constructor:** it is invoked with parameters that are passed to the class during object creation.

Question: What are the different collection classes in C#?

Answer: Collection classes are classes that are mainly used for data storage and retrieval. These collection classes will serve many purposes like allocating dynamic memory during run time and you can even access the items of the collection using the index value that makes the search easier and faster. These collection classes belong to the object class.

There are many collection classes which are as follows:

Array list: it refers to the ordered collection of the objects that are indexed individually. You can use it as an alternative to the array. Using index you can easily add or remove the items off the list and it will resize itself automatically. It works well for dynamic memory allocation, adding or searching items in the list.

Hash table: if you want to access the item of the hash table then you can use the key-value to refer to the original assigned value to the variable. Each item in the hash table is stored as a key/value pair and the item is referenced with its key value.

Stack: it works on the concept of last-in and first-out collection of the objects. Whenever you add an item to the list it is called pushing and when you remove the item off the list it is called popping.

Sorted list: this collection class uses the combination of key and the index to access the item in a list.

Queue: this collection works on the concept of first-in and first-out collection of the object. Adding an item to the list is call enqueue and removing the item off the list is call deque.

BitArray: this collection class is used to represent the array in binary form (0 and 1). You can use this collection class when you do not know the number and the items can be accessed by using integer indexes that start from zero.

Question: Explain file handling in C#.

Answer: Whenever you open a file for reading or writing it becomes a stream which is a sequence of bytes travelling from source to destination. The two commonly used streams are input and output. The included namespace is system.IO that includes many classes for file handling. The stream is an abstract class that is the parent class for the file handling process. The file is a static class with many static methods to handle file operation.

Below are the used classes:

The following table describes some commonly used classes in the System.IO namespace.

Class Name	Description
FileStream	This stream read from and write to any location within a file
BinaryReader	read primitive data types from a binary stream
DirectoryInfo	perform operations on directories
FileInfo	perform operations on files

BinaryWriter	write primitive data types in binary format
StreamReader	to read characters from a byte Stream
StreamWriter	write characters to a stream.
StringReader	read from a string buffer
StringWriter	write into a string buffer

Question: Define interface class in C#? Explain with an example.

Answer: An interface class is completely an abstract class that contains abstract methods and properties. By default, the members of the interface class are abstract and public with no fields defined. If you want to access the interface methods then the interface must be implemented by another class using ':' symbol. If you want to define the body of the methods that can only be implemented in the implementing class.

For example:

```
// Interface
Interface IAnimal {
  void Sound(); // interface method (without body)
}
class Pig : IAnimal   // Pig class "implements" the IAnimal interface
{
  public void Sound()
  {
Console.WriteLine("The pig says: wee wee"); // The body of Sound() is provided her
  }
}
class Program
{
  static void Main(string[] args)
  {
 Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
  }}
```

Question: Explain the concept of thread in C#.

Answer: A thread can be defined as the execution flow of any program and defines a unique flow of control. You can manage these threads' execution time so that their execution does not overlap the execution of other threads and prevent deadlock or to maintain efficient usage of resources. Threads are lightweight programs that save the CPU consumption and increase the efficiency of the application. The thread cycle starts with the creation of the object of system.threading.thread class and ends when the thread terminates.

System.threading.thread class allows you to handle multiple threads and the first thread always runs in a process called the main thread. Whenever you run a program in C#, the main thread runs automatically.

Question: Define structure in C# with example.

Answer: A structure is a data type of a value type. A struct keyword is used when you are going to define a structure. A structure represents a record and this record can have many attributes that define the structure. You can define a constructor but not destructor for the structure. You can implement one or more interfaces within the structure. You can specify a structure but not as abstract, virtual, or protected. If you do not use the new operator the fields of the structure remain unassigned and you cannot use the object till you initialize the fields.

Question: What do you mean by user control and custom control in C#?

Answer: User controls are very easy to create and are very much the same as the ASP control files. You cannot place a user control on the toolbox and cannot even drag-drop it. They have unique design and individual code behind these controls. Ascx is the file extension for user control.

You can create custom code as the compiled code and can be added to the toolbox. You can include these controls to the web forms easily. Custom controls can be added to multiple applications efficiently. If you want to add a private custom control then you can copy it to dll and then to the bin directory of your web application and use its reference there.

Question: C# program to remove an element from the queue.

Answer:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Application
{
 class DemoProgram
 {
  static void Main(string[] args)
  {
   Queue qs = new Queue();
   qs.Enqueue(1);
   qs.Enqueue(2);
   qs.Enqueue(3);
   foreach (Object ob in qs)
   {
    Console.WriteLine(ob);
   }
    Console.WriteLine(); Console.WriteLine();
    Console.WriteLine("Total number of elements in the Queue " + qs.Count);
    Console.WriteLine("Does the Queue contain " + qs.Contains(3));
    Console.ReadKey();
   }
 }
}
```

Question: How to find if a number is a palindrome or not in C#.

Answer:

```
using System;
  public class PalindromeNum
   {
     public static void Main(string[] args)
      {
           int n,r,num=0,Dem;
           Console.Write("Enter the Number: ");
           n = int.Parse(Console.ReadLine());
           dem=n;
           while(n>0)
           {
            r=n%10;
            num=(num*10)+r;
            n=n/10;
           }
           if(dem==num)
            Console.Write("Number is Palindrome.");
           else
            Console.Write("Number is not Palindrome");
    }
  }
```

Question: **How will you differentiate between a Class and a Struct?**

Answer: Although both class and structure are user-defined data types, they are different in several fundamental ways. A class is a reference type and stores on the heap. Struct, on the other hand, is a value type and is, therefore, stored on the stack. While the structure doesn't support inheritance and polymorphism, the class provides support for both. A class can be of an abstract type, but a structure can't. All members of a class are private by default, while members of a struct are public by default. Another distinction between class and struct is based on memory management. The former supports garbage collection while the latter doesn't.

Question: Compare Virtual methods and Abstract methods.

Answer: Any Virtual method must have a default implementation, and it can be overridden in the derived class using the override keyword. On the contrary, an Abstract method doesn't have an implementation, and it resides in the abstract class. The derived class must implement the abstract method. Though not necessary, we can use an override keyword here.

Question: What are Namespaces in C#?

Answer: Use of namespaces is for organizing large code projects. The most widely used namespace in C# is System. Namespaces are created using the namespace keyword. It is possible to use one namespace in another, known as Nested Namespaces.

Question: What are I/O classes in C#? Define some of the most commonly used ones.

Answer: The System.IO namespace in C# consists of several classes used for performing various file operations, such as creation, deletion, closing, and opening. Some of the most frequently used I/O classes in C# are:

- File – Manipulates a file
- Path – Performs operations related to some path information
- StreamReader – Reads characters from a stream
- StreamWriter – Writes characters to a stream
- StringReader – Reads a string buffer
- StringWriter – Writes a string buffer

Question: What do you understand by regular expressions in C#? Write a program that searches a string using regular expressions.

Answer: Regular expression is a template for matching a set of input. It can consist of constructs, character literals, and operators. Regex is used for string parsing, as well as replacing the character string. Following code searches a string “C#” against the set of inputs from the languages array using Regex:

```
static void Main(strong[] args)
{
string[] languages = {“C#”, “Python”, “Java”};
foreach(string s in languages)
{
if(System.Text.RegularExpressions.Regex.IsMatch(s,“C#”))
{
Console.WriteLine(“Match found”);
}
}
}
```

Question: Give a detailed explanation of Delegates in C#.

Answer: Delegates are variables that hold references to methods. It is a function pointer or reference type. Both the Delegate and the method to which it refers can have the same signature. All Delegates derives from the

```
System.Delegate namespace.
```

Following example demonstrates declaring a delegate:

```
public delegate AddNumbers(int n);
```

After declaring a delegate, the object must be created of the delegate using the new keyword, such as:

```
AddNumbers an1 = new AddNumbers(number);
```

The Delegate offers a kind of encapsulation to the reference method, which gets internally called with the calling of the delegate. In the following example, we have a delegate myDel that takes an integer value as a parameter: public delegate int myDel(int number); public class Program { public int AddNumbers(int a) { Int Sum = a + 10; return Sum; } public void Start() { myDel DelgateExample = AddNumbers; } }

Question: Explain Reflection in C#.

Answer: The ability of code to access the metadata of the assembly during runtime is called Reflection. A program reflects upon itself and uses the metadata to:

- Inform the user, or
- Modify the behaviour

The system contains all classes and methods that manage the information of all the loaded types and methods. Reflection namespace. Implementation of reflection is in 2 steps:

- Get the type of the object, then
- Use the type to identify members, such as properties and methods

Question: Name some of the most common places to look for a Deadlock in C#.

Answer: For recognizing deadlocks, one should look for threads that get stuck on one of the following:

- .Result, .GetAwaiter().GetResult(), WaitAll(), and WaitAny() (When working with Tasks)
- Dispatcher.Invoke() (When working in WPF)
- Join() (When working with Threads)
- lock statements (In all cases)
- WaitOne() methods (When working with AutoResetEvent/EventWaitHandle/Mutex/Semaphore)

Question: Define Serialization and its various types in C#.

Answer: The process of converting some code into its binary format is known as Serialization in C#. Doing so allows the code to be stored easily and written to a disk or some other storage device. We use Serialization when there is a strict need for not losing the original form of the code. A class marked with the attribute [Serializable] gets converted to its binary form. A stream that contains the serialized object and the System.Runtime.Serialization namespace can have classes for serialization. Serialization in C# is of three types:

- **Binary Serialization –** Faster and demands less space; it converts any code into its binary form. Serialize and restore public and non-public properties.

- **SOAP –** It produces a complete SOAP compliant envelope that is usable by any system capable of understanding SOAP. The classes about this type of serialization reside in System.Runtime.Serialization.
- **XML Serialization –** Serializes all the public properties to the XML document. In addition to being easy to read, the XML document manipulated in several formats. The classes in this type of serialization reside in System.sml.Serialization.

Note: Retrieving the C# code back from the binary form is known as Deserialization.

Question: Give a brief explanation of Thread Pooling in C#.

Answer: A collection of threads, termed as a Thread Pool in C#. Such threads are for performing tasks without disturbing the execution of the primary thread. After a thread belonging to a thread pool completes execution, it returns to the thread pool. Classes that manage the thread in the thread pool, and its operations, are contained in the System.Threading.ThreadPool namespace.

Question: Is it possible to use this keyword within a static method in C#?

Answer: A special type of reference variable, this keyword is implicitly defined with each non-static method and constructor as the first parameter of the type class, which defines it. Static methods don't belong to a particular instance. Instead, they exist without creating an instance of the class and calls with the name of the class. Because this keyword returns a reference to the current instance of the class containing it, it can't be used in a static method. Although we can't use this keyword within a static method, we can use it in the function parameters of Extension Methods (https://en.wikipedia.org/wiki/Extension_method).

Question: What can you tell us about the XSD file in C#?

Answer: XSD denotes XML Schema Definition. The XML file can have any attributes, elements, and tags if there is no XSD file associated with it. The XSD file gives a structure for the XML file, meaning that it determines what, and also the order of, the elements and properties that should be there in the XML file. **Note**: - During serialization of C# code, the classes are converted to XSD compliant format by the Xsd.exe tool.

Question: What do you mean by Constructor Chaining in C#?

Answer: Constructor chaining in C# is a way of connecting two or more classes in a relationship as an inheritance. Every child class constructor is mapped to the parent class constructor implicitly by using the base keyword in constructor chaining.

Question: Explain different states of a Thread in C#?

Answer: A thread in C# can have any of the following states:

- Aborted – The thread is dead but not stopped
- Running – The thread is executing
- Stopped – The thread has stopped execution
- Suspended – The thread has been suspended

- Unstarted – The thread is created but has not started execution yet
- WaitSleepJoin – The thread calls sleep, calls wait on another object, and calls join on some other thread

Question: Why do we use Async and Await in C#?

Answer: Processes belonging to asynchronous programming run independently of the main or other processes. In C#, using Async and Await keywords for creating asynchronous methods.

Question: What is an Indexer in C#, and how do you create one?

Answer: Also known as an indexed property, an indexer is a class property allowing accessing a member variable of some class using features of an array. Used for treating an object as an array, indexer allows using classes more intuitively. Although not an essential part of the object-oriented programming, indexers are a smart way of using arrays. As such, they are also called smart arrays. Defining an indexer enables creating classes that act like virtual arrays. Instances of such classes can be accessed using the [] array access operator. The general syntax for creating an indexer in C# is:

```
< modifier > <
return type > this[argument list] {
get {
// the get block code
}
set {
// the set block code
}
}
```

Question: What is the Race condition in C#?

Answer: When two threads access the same resource and try to change it at the same time, we have a race condition. It is almost impossible to predict which thread succeeds in accessing the resource first. When two threads try to write a value to the same resource, the last value written is saved.

Question: What do you understand by Get and Set Accessor properties?

Answer: Made using properties, Get and Set are called accessors in C#. A property enables reading and writing to the value of a private field. Accessors are used for accessing such private fields. While we use the Get property for returning the value of a property, use the Set property for setting the value.

Question: Give a detailed explanation of the differences between ref and out keywords.

Answer: In any C# function, there can be three types of parameters, namely in, out and ref. Although both out and ref are treated differently at the run time, they receive the same treatment during the compile time. It is not possible to pass properties as an out or ref

parameter. Following are the differences between ref and out keywords:

- **Initializing the Argument or Parameter –** While it is not compulsory to initialize an argument or parameter before passing to an out parameter, the same needs to be initialized before passing it to the ref parameter.
- **Initializing the Value of the Parameter –** Using ref doesn't necessitate for assigning or initializing the value of a parameter before returning to the calling method. When using out, however, it is mandatory to use a called method for assigning or initializing a value of a parameter before returning to the calling method.
- **Usefulness –** When the called method requires modifying the passed parameter, passing a parameter value by Ref is useful. Declaring a parameter to an out method is appropriate when multiple values are required to be returned from a function or method.
- **Initializing a Parameter Value in Calling Method –** It is a compulsion to initialize a parameter value within the calling method while using out. However, the same is optional while using the ref parameter.
- **Data Passing –** Using out allows for passing data only in a unidirectional way. However, data can be passed in a bidirectional manner when using ref.

Question: **What is Singleton Design Patterns in C#? Explain their implementation using an example.**

Answer: A singleton in C# is a class that allows the creation of only a single instance of itself and provides simple access to that sole instance. Because the second request of an instance with a different parameter can cause problems, singletons typically disallow any parameters to be specified. Following example demonstrates the implementation of Singleton Design Patterns in C#:

```
namespace Singleton {
class Program {
static void Main(string[] args) {
Calculate.Instance.ValueOne = 10.5;
Calculate.Instance.ValueTwo = 5.5;
Console.WriteLine("Addition : " + Calculate.Instance.Addition());
Console.WriteLine("Subtraction : " + Calculate.Instance.Subtraction());
Console.WriteLine("Multiplication : " + Calculate.Instance.Multiplication());
Console.WriteLine("Division : " + Calculate.Instance.Division());
Console.WriteLine("\n----------------------\n");
Calculate.Instance.ValueTwo = 10.5;
Console.WriteLine("Addition : " + Calculate.Instance.Addition());
Console.WriteLine("Subtraction : " + Calculate.Instance.Subtraction());
Console.WriteLine("Multiplication : " + Calculate.Instance.Multiplication());
Console.WriteLine("Division : " + Calculate.Instance.Division());
Console.ReadLine();
}
}
public sealed class Calculate {
private Calculate() {}
private static Calculate instance = null;
public static Calculate Instance {
get {
if (instance == null) {
instance = new Calculate();
}
return instance;
}
}
public double ValueOne {
get;
set;
}
public double ValueTwo {
get;
set;
}
public double Addition() {
return ValueOne + ValueTwo;
}
public double Subtraction() {
return ValueOne - ValueTwo;
}
public double Multiplication() {
return ValueOne * ValueTwo;
}
public double Division() {
return ValueOne / ValueTwo;
}
}
}
```

A Singleton Design Pattern ensures that a class has one and only one instance and provides a global point of access to the same. There are numerous ways of implementing the Singleton Design Patterns in C#. Following are the typical characteristics of a Singleton Pattern:

- A public static means of getting the reference to the single instance created
- A single constructor, private and parameter-less
- A static variable holding a reference to the single instance created
- The class is sealed

Conclusion

That sums up the list of the top c# interview questions for experienced professionals and beginners as well. How many of the answers did you know already? Let us know via comments. Check out these best C# tutorials (https://hackr.io/tutorials/learn-c-sharp?ref=blog-post) to enhance your C# understanding further.

Looking for more C# coding problems interview questions? We suggest one of the best C# interview courses: C# Advanced Topics: Prepare for Technical Interviews (https://click.linksynergy.com/deeplink?id=Vq5kdUDL6n8&mid=39197&murl=https://www.udemy.com/course/csharp-advanced/).

Here, we recommend a great book for preparing for C# interviews. Rocking the C# Interview: A comprehensive question and answer reference guide for the C# programming language. 1st Edition (https://geni.us/PYHlkfn).

People are also reading:

Related Tutorials

C#

(https://hackr.io/tutorials/learn-c-sharp)

Visual Basic .NET

(https://hackr.io/tutorials/learn-visual-basic-net-vb)

ASP.NET

(https://hackr.io/tutorials/learn-asp-net)

Microsoft Azure

(https://hackr.io/tutorials/learn-microsoft-azure)

Windows Server Administration

(https://hackr.io/tutorials/learn-windows-server-administration)

Recommended Learning

C# Basics for Beginners (www.udemy.com)

(https://hackr.io/tutorial/csharp-basics-for-beginners)

The C# Yellow Book (www.robmiles.com)

(https://hackr.io/tutorial/the-c-sharp-yellow-book)

Learn C# (Learning Path from Beginner to Advanced) (pluralsight.pxf.io)

(https://hackr.io/tutorial/learn-c-sharp)

VIEW MORE (https://hackr.io/tutorials/learn-c-sharp)
