# Top 50 C# Interview Questions With Answers

**Q #1) What is an Object and a Class?**

**Ans:**A Class is an encapsulation of properties and methods that are used to represent a real-time entity. It is a data structure that brings all the instances together in a single unit.

An Object in an instance of a Class. Technically, it is just a block of memory allocated that can be stored in the form of Variables, Array or a Collection.

**Q #2) What are the fundamental OOP concepts?**

**Ans: The four fundamental concepts of Object Oriented Programming are:**

* **Encapsulation** – The Internal representation of an object is hidden from the view outside object’s definition. Only the required information can be accessed whereas the rest of the data implementation is hidden.
* **Abstraction** – It is a process of identifying the critical behavior and data of an object and eliminating the irrelevant details.
* **Inheritance** – It is the ability to create new classes from another class. It is done by accessing, modifying and extending the behavior of objects in the parent class.
* **Polymorphism** – The name means, one name, many forms. It is achieved by having multiple methods with the same name but different implementations.

**Q #3) What is Managed and Unmanaged code?**

**Ans:** **Managed code** is a code which is executed by CLR (Common Language Runtime) i.e all application code based on .Net Platform. It is considered as managed because of the .Net framework which internally uses the garbage collector to clear up the unused memory.

**Unmanaged code** is any code that is executed by application runtime of any other framework apart from .Net. The application runtime will take care of memory, security and other performance operations.

**Q #4) What is an Interface?**

**Ans:** **An Interface** is a class with no implementation. The only thing that it contains is the declaration of methods, properties, and events.

**Q #5) What are the different types of classes in C#?**

**Ans: The different types of class in C# are:**

* **Partial class** – Allows its members to be divided or shared with multiple .cs files. It is denoted by the keyword Partial.
* **Sealed class** – It is a class which cannot be inherited. To access the members of a sealed class, we need to create the object of the class.  It is denoted by the keyword Sealed.
* **Abstract class** – It is a class whose object cannot be instantiated. The class can only be inherited. It should contain at least one method.  It is denoted by the keyword abstract.
* **Static class** – It is a class which does not allow inheritance. The members of the class are also static.  It is denoted by the keyword static. This keyword tells the compiler to check for any accidental instances of the static class.

**Q #6) Explain Code compilation in C#.**

**Ans: There are four steps in code compilation which include:**

* Compiling the source code into Managed code by C# compiler.
* Combining the newly created code into assemblies.
* Loading the Common Language Runtime(CLR).
* Executing the assembly by CLR.

**Q #7) What are the differences between a Class and a Struct?**

**Ans: Given below are the differences between a Class and a Struct:**

| **Class** | **Struct** |
| --- | --- |
| Supports Inheritance | Does not support Inheritance |
| Class is Pass by reference (reference type) | Struct is Pass by Copy (Value type) |
| Members are private by default | Members are public by default |
| Good for larger complex objects | Good for Small isolated models |
| Can use waste collector for memory management | Cannot use Garbage collector and hence no Memory management |

**Q #8) What is the difference between Virtual method and Abstract method?**

**Ans:**A **Virtual method** must always have a default implementation. However, it can be overridden in the derived class, though not mandatory. It can be overridden using override keyword.

An **Abstract method** does not have an implementation. It resides in the abstract class. It is mandatory that the derived class implements the abstract method. An override keyword is not necessary here though it can be used.

**Q #9) Explain Namespaces in C#.**

**Ans:** They are used to organize large code projects. “System” is the most widely used namespace in C#. We can create our own namespace and use one namespace in another, which are called Nested Namespaces.

They are denoted by the keyword “namespace”.

**Q #10) What is “using” statement in C#?**

**Ans:** “Using” Keyword denotes that the particular namespace is being used by the program.

**For Example,** using System. Here System is a namespace. The class Console is defined under System.  So we can use the console.writeline (“….”) or readline in our program.

**Q #11) Explain Abstraction.**

**Ans: Abstraction** is one of the OOP concepts. It is used to display only the essential features of the class and hides the unnecessary information.

**Let us take an Example of a Car:**

A driver of the car should know the details about the Car such as color, name, mirror, steering, gear, brake, etc. What he doesn't have to know is an Internal engine, Exhaust system.

So, Abstraction helps in knowing what is necessary and hiding the internal details from the outside world. Hiding of the internal information can be achieved by declaring such parameters as Private using the private keyword.

**Q #12) Explain Polymorphism?**

**Ans:** Programmatically, **Polymorphism** means same method but different implementations.

It is of 2 types, Compile-time and Runtime.

**Compile time polymorphism** is achieved by operator overloading.

**Runtime polymorphism** is achieved by overriding. Inheritance and Virtual functions are used during Runtime Polymorphism.

**For Example**, If a class has a method Void Add(), polymorphism is achieved by Overloading the method, that is, void Add(int a, int b), void Add(int add) are all overloaded methods.

**Q #13) How is Exception Handling implemented in C#?**

**Ans: Exception handling is done using four keywords in C#:**

* **try** – Contains a block of code for which an exception will be checked.
* **catch** – It is a program that catches an exception with the help of exception handler.
* **finally** – It is a block of code written to execute regardless whether an exception is caught or not.
* **Throw** – Throws an exception when a problem occurs.

**Q #14) What are C# I/O Classes? What are the commonly used I/O Classes?**

**Ans:**C# has System.IO namespace, consisting of classes that are used to perform various operations on files like creating, deleting, opening, closing etc.

**Some commonly used I/O classes are:**

* **File** – Helps in manipulating a file.
* **StreamWriter** – Used for writing characters to a stream.
* **StreamReader** – Used for reading characters to a stream.
* **StringWriter** – Used for reading a string buffer.
* **StringReader** – Used for writing a string buffer.
* **Path** – Used for performing operations related to path information.

**Q #15) What is StreamReader/StreamWriter class?**

**Ans: StreamReader and StreamWriter** are classes of namespace System.IO. They are used when we want to read or write charact90, Reader-based data, respectively.

**Some of the members of StreamReader are:** Close(), Read(), Readline().

**Members of StreamWriter are:** Close(), Write(), Writeline().

|  |  |  |
| --- | --- | --- |
| 1 | Class Program1 | |
| 2 | { |

|  |  |  |
| --- | --- | --- |
| 3 | using(StreamReader sr = new StreamReader(“C:\ReadMe.txt”) | |
| 4 | { |

|  |  |  |
| --- | --- | --- |
| 5 | //----------------code to read-------------------// | |
| 6 | } |

|  |  |  |
| --- | --- | --- |
| 7 | using(StreamWriter sw = new StreamWriter(“C:\ReadMe.txt”)) | |
| 8 | { |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | //-------------code to write-------------------// | | |
| 10 | | } |

|  |  |
| --- | --- |
| 11 | } |

**Q #16) What is a Destructor in C#?**

**Ans:**A **Destructor** is used to clean up the memory and free the resources. But in C# this is done by the garbage collector on its own. System.GC.Collect() is called internally for cleaning up. But sometimes it may be necessary to implement destructors manually.

**For Example:**

~Car()

{

Console.writeline(“….”);

}

**Q #17) What is an Abstract Class?**

**Ans:** An **Abstract class** is a class which is denoted by abstract keyword and can be used only as a Base class. An Abstract class should always be inherited. An instance of the class itself cannot be created. If we do not want any program to create an object of a class, then such classes can be made abstract.

Any method in the abstract class does not have implementations in the same class. But they must be implemented in the child class.

**For Example:**

|  |  |  |
| --- | --- | --- |
| 1 | abstract class AB1 | |
| 2 | { |

|  |  |  |
| --- | --- | --- |
| 3 | Public void Add(); | |
| 4 | } |

|  |  |  |
| --- | --- | --- |
| 5 | Class childClass : AB1 | |
| 6 | { |

|  |  |  |
| --- | --- | --- |
| 7 | childClass cs = new childClass (); | |
| 8 | int Sum = cs.Add(); |

|  |  |
| --- | --- |
| 9 | } |

All the methods in an abstract class are implicitly virtual methods. Hence virtual keyword should not be used with any methods in  abstract class.

**Q #18) What are Boxing and Unboxing?**

**Ans:** Converting a value type to reference type is called **Boxing**.

**For Example:**

int Value1 -= 10;

//————Boxing——————//

object boxedValue = Value1;

Explicit conversion of same reference type (created by boxing) back to value type is called **Unboxing**.

**For Example:**

//————UnBoxing——————//

int UnBoxing = int (boxedValue);

**Q #19) What is the difference between Continue and Break Statement?**

**Ans:** Break statement breaks the loop. It makes the control of the program to exit the loop. **Continue statement** makes the control of the program to exit only the current iteration. It does not break the loop.

**Q #20) What is the difference between finally and finalize block?**

**Ans: finally block** is called after the execution of try and catch block. It is used for exception handling. Regardless of whether an exception is caught or not, this block of code will be executed. Usually, this block will have clean-up code.

finalize method is called just before garbage collection. It is used to perform clean up operations of Unmanaged code. It is automatically called when a given instance is not subsequently called.

#### **Questions on Arrays and Strings**

**Q #21) What is an Array? Give the syntax for a single and multi-dimensional array?**

**Ans: An Array** is used to store multiple variables of the same type. It is a collection of variables stored in a contiguous memory location.

**For Example:**

double numbers = new double[10];

int[] score = new int[4] {25,24,23,25};

A Single dimensional array is a linear array where the variables are stored in a single row. Above **example** is a Single dimensional array.

Arrays can have more than one dimension. Multidimensional arrays are also called rectangular arrays.

**For Example**, int[,] numbers = new int[3,2] { {1,2} ,{2,3},{3,4} };

**Q #22) What is a Jagged Array?**

**Ans:** A Jagged array is an array whose elements are arrays. It is also called as the array of arrays. It can be either single or multiple dimensions.

int[] jaggedArray = new int[4][];

**Q #23) Name some properties of Array.**

**Ans: Properties of an Array include:**

* **Length** – Gets the total number of elements in an array.
* **IsFixedSize** – Tells whether the array is fixed in size or not.
* **IsReadOnly** – Tells whether the array is read-only or not.

**Q #24) What is an Array Class?**

**Ans:** An **Array class** is the base class for all arrays. It provides many properties and methods. It is present in the namespace System.

**Q #25) What is a String? What are the properties of a String Class?**

**Ans:**A **String** is a collection of char objects. We can also declare string variables in c#.

string name = “C# Questions”;

A string class in C# represents a string.

The properties of String class are **Chars and Length.**  
**Chars** get the Char object in the current String.  
**Length** gets the number of objects in the current String.

**Q #26) What is an Escape Sequence? Name some String escape sequences in C#.**

**Ans:** An **Escape sequence** is denoted by a backslash (\). The backslash indicates that the character that follows it should be interpreted literally or it is a special character. An escape sequence is considered as a single character.

**String escape sequences are as follows:**

\n – Newline character  
\b – Backspace  
\\ – Backslash  
\’ – Single quote  
\’’ – Double Quote

**Q #27) What are Regular expressions? Search a string using regular expressions?**

**Ans: Regular expression** is a template to match a set of input. The pattern can consist of operators, constructs or character literals. Regex is used for string parsing and replacing the character string.

**For Example:**

\* matches the preceding character zero or more times. So, a\*b regex is equivalent to b, ab, aab, aaab and so on.

Searching a string using Regex

|  |  |  |
| --- | --- | --- |
| 1 | static void Main(string[] args) | |
| 2 | { |

|  |  |  |
| --- | --- | --- |
| 3 | string[] languages = { "C#", "Python", "Java" }; | |
| 4 | foreach(string s in languages) |

|  |  |
| --- | --- |
| 5 | { |
| 6 | if(System.Text.RegularExpressions.Regex.IsMatch(s,"Python")) | |

|  |  |
| --- | --- |
| 7 | { |
| 8 | Console.WriteLine("Match found"); | |

|  |  |  |
| --- | --- | --- |
| 9 | } | |
| 10 | | } | |

|  |  |
| --- | --- |
| 11 | } |

The above example searches for “Python” against the set of inputs from the languages array. It uses Regex.IsMatch which returns true in case if the pattern is found in the input. The pattern can be any regular expression representing the input that we want to match.

**Q #28) What are the basic String Operations? Explain.**

**Ans: Some of the basic string operations are:**

* **Concatenate** –Two strings can be concatenated either by using System.String.Concat or by using + operator.
* **Modify** – Replace(a,b) is used to replace a string with another string. Trim() is used to trim the string at the end or at the beginning.
* **Compare** – System.StringComparison() is used to compare two strings, either case-sensitive comparison or not case sensitive. Mainly takes two parameters, original string, and string to be compared with.
* **Search** – StartWith, EndsWith methods are used to search a particular string.

**Q #29) What is Parsing? How to Parse a Date Time String?**

**Ans: Parsing** is converting a string into another data type.

**For Example:**

string text = “500”;

int num = int.Parse(text);

500 is an integer. So, Parse method converts the string 500 into its own base type, i.e int.

**Follow the same method to convert a DateTime string.**  
string dateTime = “Jan 1, 2018”;  
DateTime parsedValue = DateTime.Parse(dateTime);

#### **Advanced Concepts**

**Q #30) What is a Delegate? Explain.**

**Ans:**A **Delegate** is a variable that holds the reference to a method. Hence it is a function pointer of reference type. All Delegates are derived from System.Delegate namespace. Both Delegate and the method that it refers to can have the same signature.

Declaring a delegate: public delegate void AddNumbers(int n);

After the declaration of a delegate, the object must be created of the delegate using the new keyword.

AddNumbers an1 = new AddNumbers(number);

The delegate provides a kind of encapsulation to the reference method, which will internally get called when a delegate is called.

|  |  |  |
| --- | --- | --- |
| 1 | public delegate int myDel(int number); | |
| 2 | public class Program |

|  |  |
| --- | --- |
| 3 | { |
| 4 | public int AddNumbers(int a) | |

|  |  |
| --- | --- |
| 5 | { |
| 6 | int Sum = a + 10; | |

|  |  |  |
| --- | --- | --- |
| 7 | return Sum; | |
| 8 | } |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | public void Start() | | |
| 10 | | { |

|  |  |  |
| --- | --- | --- |
| 11 | myDel DelgateExample = AddNumbers; | |
| 12 | } |

|  |  |
| --- | --- |
| 13 | } |

In the above example, we have a delegate myDel which takes an integer value as a parameter. Class Program has a method of the same signature as the delegate, called AddNumbers().

If there is another method called Start() which creates an object of the delegate, then the object can be assigned to AddNumbers as it has the same signature as that of the delegate.

**Q #31) What are Events?**

**Ans: Events** are user actions that generate notifications to the application to which it must respond. The user actions can be mouse movements, keypress and so on.

Programmatically, a class that raises an event is called a publisher and a class which responds/receives the event is called a subscriber. An Event should have at least one subscriber else that event is never raised.

Delegates are used to declare Events.

Public delegate void PrintNumbers();

Event PrintNumbers myEvent;

**Q #32) How to use Delegates with Events?**

**Ans:** Delegates are used to raise events and handle them. Always a delegate needs to be declared first and then the Events are declared.

**Let us see an Example:**

Consider a class called Patient. Consider two other classes, Insurance, and Bank which requires Death information of the Patient from patient class. Here, Insurance and Bank are the subscribers and the Patient class becomes the Publisher. It triggers the death event and the other two classes should receive the event.

|  |  |  |
| --- | --- | --- |
| 1 | namespace ConsoleApp2 | |
| 2 | { |

|  |  |  |
| --- | --- | --- |
| 3 | public class Patient | |
| 4 | { |

|  |  |  |
| --- | --- | --- |
| 5 | public delegate void deathInfo();//Declaring a Delegate// | |
| 6 | public event deathInfo deathDate;//Declaring the event// |

|  |  |  |
| --- | --- | --- |
| 7 | public void Death() | |
| 8 | { |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | deathDate(); | | |
| 10 | | } |

|  |  |
| --- | --- |
| 11 | } |
| 12 | public class Insurance | |

|  |  |
| --- | --- |
| 13 | { |
| 14 | Patient myPat = new Patient(); | |

|  |  |  |
| --- | --- | --- |
| 15 | void GetDeathDetails() | |
| 16 | { |

|  |  |  |
| --- | --- | --- |
| 17 | //-------Do Something with the deathDate event------------// | |
| 18 | } |

|  |  |  |
| --- | --- | --- |
| 19 | void Main() | |
| 20 | { |

|  |  |  |
| --- | --- | --- |
| 21 | //--------Subscribe the function GetDeathDetails----------// | |
| 22 | myPat.deathDate += GetDeathDetails; |

|  |  |
| --- | --- |
| 23 | } |
| 24 | } |

|  |  |  |
| --- | --- | --- |
| 25 | public class Bank | |
| 26 | { |

|  |  |  |
| --- | --- | --- |
| 27 | Patient myPat = new Patient(); | |
| 28 | void GetPatInfo () |

|  |  |
| --- | --- |
| 29 | { |
| 30 | //-------Do Something with the deathDate event------------// | |

|  |  |
| --- | --- |
| 31 | } |
| 32 | void Main() | |

|  |  |
| --- | --- |
| 33 | { |
| 34 | //--------Subscribe the function GetPatInfo ----------// | |

|  |  |  |
| --- | --- | --- |
| 35 | myPat.deathDate += GetPatInfo; | |
| 36 | } |

|  |  |
| --- | --- |
| 37 | } |
| 38 | } |

**Q #33) What are the different types of Delegates?**

**Ans: The Different types of Delegates are:**

**Single Delegate** – A delegate which can call a single method.

**Multicast Delegate** – A delegate which can call multiple methods. + and – operators are used to subscribe and unsubscribe respectively.

**Generic Delegate** – It does not require an instance of delegate to be defined. It is of three types, Action, Funcs and Predicate.

* **Action**– In the above example of delegates and events, we can replace the definition of delegate and event using Action keyword. The Action delegate defines a method that can be called on arguments but does not return a result

Public delegate void deathInfo();

Public event deathInfo deathDate;

//Replacing with Action//

Public event Action deathDate;

Action implicitly refers to a delegate.

* ***Func*** – A Func delegate defines a method that can be called on arguments and returns a result.

Func <int, string, bool> myDel is same as delegate bool myDel(int a, string b);

* ***Predicate*** – Defines a method that can be called on arguments and always returns the bool.

Predicate<string> myDel is same as delegate bool myDel(string s);

**Q #34) What do Multicast Delegates mean?**

**Ans:**A Delegate that points to more than one method is called a Multicast Delegate. Multicasting is achieved by using + and += operator.

**Consider the Example from question 32.**

There are two subscribers for deathEvent, GetPatInfo, and GetDeathDetails. And hence we have used += operator. It means whenever the myDel is called, both the subscribers get called. The delegates will be called in the order in which they are added.

**Q #35) Explain Publishers and Subscribers in Events.**

**Ans:** **A Publisher** is a class responsible for publishing a message of different types of other classes. The message is nothing but Event as discussed in the above questions.

From the **Example** in Question 32, Class Patient is the Publisher class. It is generating an Event deathEvent, which the other classes receive.

**Subscribers** capture the message of the type that it is interested in. Again, from the **Example** of Question 32, Class Insurance and Bank are Subscribers. They are interested in event deathEvent of type void.

**Q #36) What are Synchronous and Asynchronous operations?**

**Ans:** Synchronization is a way to create a thread-safe code where only one thread can access the resource at any given time.

Asynchronous call waits for the method to complete before continuing with the program flow. Synchronous programming badly affects the UI operations, when the user tries to perform time-consuming operations since only one thread will be used.

In Asynchronous operation, the method call will immediately return so that the program can perform other operations while the called method completes its work in certain situations.

In C#, Async and Await keywords are used to achieve asynchronous programming. Look at **Question 43** for more details on synchronous programming.

**Q #37) What is Reflection in C#?**

**Ans:** Reflection is the ability of a code to access the metadata of the assembly during runtime. A program reflects upon itself and uses the metadata to inform the user or modify its behavior. Metadata refers to information about objects, methods.

The namespace System.Reflection contains methods and classes that manage the information of all the loaded types and methods. It is mainly used for windows applications, for **Example**, to view the properties of a button in a windows form.

The MemberInfo object of the class reflection is used to discover the attributes associated with a class.

Reflection is implemented in two steps, first, we get the type of the object, and then we use the type to identify members such as methods and properties.

To get type of a class, we can simply use

Type mytype = myClass.GetType();

Once we have a type of class, the other information of the class can be easily accessed.

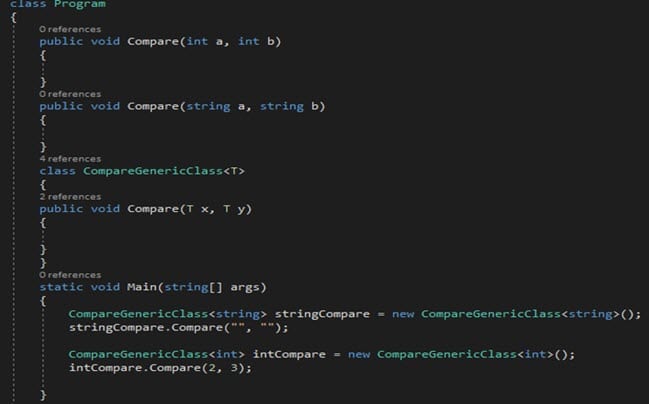
System.Reflection.MemberInfo Info = mytype.GetMethod(“AddNumbers”);

Above statement tries to find a method with name AddNumbers in the class myClass.

**Q #38) What is a Generic Class?**

**Ans:** Generics or Generic class is used to create classes or objects which do not have any specific data type. The data type can be assigned during runtime, i.e when it is used in the program.

**For Example:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/02/38a.jpg)

So, from the above code, we see 2 compare methods initially, to compare string and int.

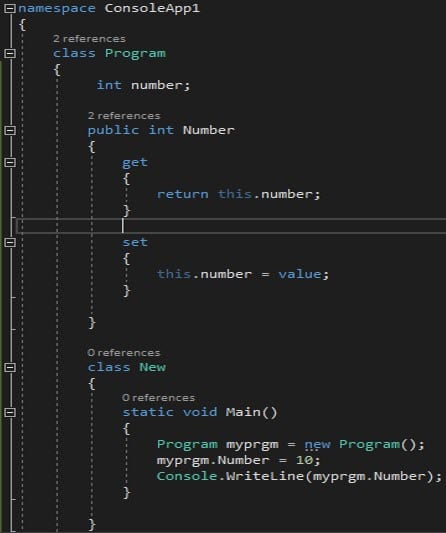
In case of other data type parameter comparisons, instead of creating many overloaded methods, we can create a generic class and pass a substitute data type, i.e T. So, T acts as a datatype until it is used specifically in the Main() method.

**Q #39) Explain Get and Set Accessor properties?**

**Ans: Get and Set** are called Accessors. These are made use by Properties. A property provides a mechanism to read, write the value of a private field. For accessing that private field, these accessors are used.

Get Property is used to return the value of a property  
Set Property accessor is used to set the value.

**The usage of get and set is as below:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/02/39a.jpg)

**Q #40) What is a Thread? What is Multithreading?**

**Ans:**A **Thread** is a set of instructions that can be executed, which will enable our program to perform concurrent processing. Concurrent processing helps us do more than one operation at a time. By default, C# has only one thread. But the other threads can be created to execute the code in parallel with the original thread.

Thread has a life cycle. It starts whenever a thread class is created and is terminated after the execution. System.Threading is the namespace which needs to be included to create threads and use its members.

Threads are created by extending the Thread Class. Start() method is used to begin thread execution.

//CallThread is the target method//

ThreadStart methodThread = new ThreadStart(CallThread);

Thread childThread = new Thread(methodThread);

childThread.Start();

C# can execute more than one task at a time. This is done by handling different processes by different threads. This is called MultiThreading.

**There are several thread methods that are used to handle the multi-threaded operations:**

Start, Sleep, Abort, Suspend, Resume and Join.

Most of these methods are self-explanatory.

**Q #41) Name some properties of Thread Class.**

**Ans: Few Properties of thread class are:**

* **IsAlive** – contains value True when a thread is Active.
* **Name** – Can return the name of the thread. Also, can set a name for the thread.
* **Priority** – returns the prioritized value of the task set by the operating system.
* **IsBackground** – gets or sets a value which indicates whether a thread should be a background process or foreground.
* **ThreadState**– describes the thread state.

**Q #42) What are the different states of a Thread?**

**Different states of a thread are:**

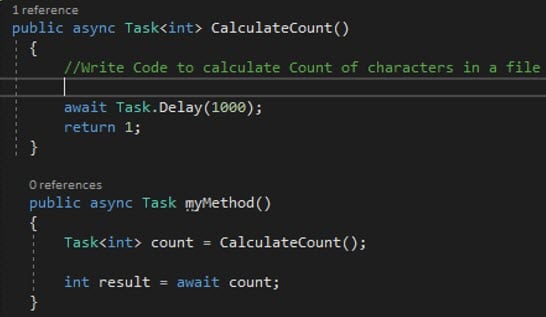
* **Unstarted** – Thread is created.
* **Running** – Thread starts execution.
* **WaitSleepJoin** – Thread calls sleep, calls wait on another object and calls join on another thread.
* **Suspended** – Thread has been suspended.
* **Aborted** – Thread is dead but not changed to state stopped.
* **Stopped** – Thread has stopped.

**Q #43) What are Async and Await?**

**Ans:**Async and Await keywords are used to create asynchronous methods in C.

Asynchronous programming means that the process runs independently of main or other processes.

**Usage of Async and Await is as shown below:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/02/43a.jpg)

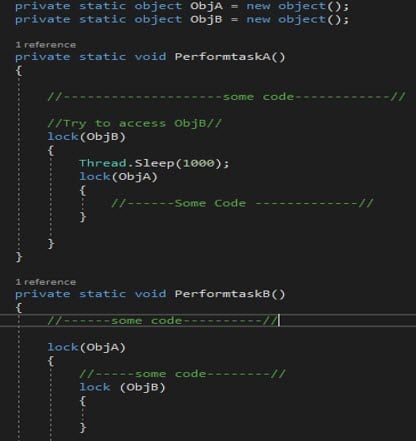
* Async keyword is used for the method declaration.
* The count is of a task of type int which calls the method CalculateCount().
* Calculatecount() starts execution and calculates something.
* Independent work is done on my thread and then await count statement is reached.
* If the Calculatecount is not finished, myMethod will return to its calling method, thus the main thread doesn’t get blocked.
* If the Calculatecount is already finished, then we have the result available when the control reaches await count. So the next step will continue in the same thread. However, it is not the situation in the above case where Delay of 1 second is involved.

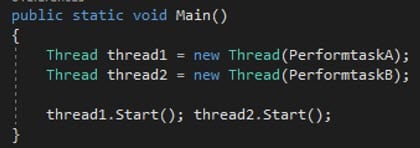
**Q #44) What is a Deadlock?**

**Ans:**A **Deadlock** is a situation where a process is not able to complete its execution because two or more processes are waiting for each other to finish. This usually occurs in multi-threading.

Here a Shared resource is being held by a process and another process is waiting for the first process to release it and the thread holding the locked item is waiting for another process to complete.

**Consider the below Example:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/02/44a.jpg)

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/02/44b.jpg)

* Perform tasks accesses objB and waits for 1 second.
* Meanwhile, PerformtaskB tries to access ObjA.
* After 1 second, PeformtaskA tries to access ObjA which is locked by PerformtaskB.
* PerformtaskB tries to access ObjB which is locked by PerformtaskA.

This creates Deadlock.

**Q #45) Explain Lock, Monitors, and Mutex Object in Threading.**

**Ans: Lock** keyword ensures that only one thread can enter a particular section of the code at any given time. In the above **Example**, lock(ObjA) means the lock is placed on ObjA until this process releases it, no other thread can access ObjA.

A **Mutex** is also like a lock but it can work across multiple processes at a time. WaitOne() is used to lock and ReleaseMutex() is used to release the lock. But Mutex is slower than lock as it takes time to acquire and release it.

Monitor.Enter and Monitor.Exit implements lock internally. a lock is a shortcut for M**onitors**. lock(objA) internally calls.

Monitor.Enter(ObjA);

try

{

}

Finally {Monitor.Exit(ObjA));}

**Q #46) What is a Race Condition?**

**Ans:**A **Race condition** occurs when two threads access the same resource and are trying to change it at the same time. The thread which will be able to access the resource first cannot be predicted.

If we have two threads, T1 and T2, and they are trying to access a shared resource called X. And if both the threads try to write a value to X, the last value written to X will be saved.

**Q #47) What is Thread Pooling?**

**Ans:**A **Thread pool** is a collection of threads. These threads can be used to perform tasks without disturbing the primary thread. Once the thread completes the task, the thread returns to the pool.

System.Threading.ThreadPool namespace has classes which manage the threads in the pool and its operations.

System.Threading.ThreadPool.QueueUserWorkItem(new System.Threading.WaitCallback(SomeTask));

The above line queues a task. SomeTask methods should have a parameter of type Object.

**Q #48) What is Serialization?**

**Ans: Serialization** is a process of converting a code to its binary format. Once it is converted to bytes, it can be easily stored and written to a disk or any such storage devices. Serializations are mainly useful when we do not want to lose the original form of the code and it can be retrieved anytime in the future.

Any class which is marked with the attribute [Serializable] will be converted to its binary form.

The reverse process of getting the c# code back from the binary form is called **Deserialization**.

To Serialize an object we need the object to be serialized, a stream which can contain the serialized object and namespace System.Runtime.Serialization can contain classes for serialization.

**Q #49) What are the types of Serialization?**

**Ans:** **The different types of Serialization are: XML serialization, SOAP, and Binary.**

* **XML serialization** – It serializes all the public properties to the XML document. Since the data is in XML format, it can be easily read and manipulated in various formats. The classes reside in System.sml.Serialization.
* **SOAP** – Classes reside in System.Runtime.Serialization. Similar to XML but produces a complete SOAP compliant envelope which can be used by any system that understands SOAP.
* **Binary Serialization** – Allows any code to be converted to its binary form. Can serialize and restore public and non-public properties. It is faster and occupies less space.

**Q #50) What is an XSD file?**

**Ans:**An XSD file stands for XML Schema Definition. It gives a structure for the XML file. It means it decides the elements that the XML should have and in what order and what properties should be present. Without an XSD file associated with XML, the XML can have any tags, any attributes, and any elements.

Xsd.exe tool converts the files to XSD format. During Serialization of C# code, the classes are converted to XSD compliant format by xsd.exe.