Frequently Ask Interview Questions

20 November 2020 03:57 PM

Q1.What is Difference between SQL and MYSQL ?

->

SQL is a query language, whereas **MySQL is** a relational database that uses **SQL** to query a database. You can use **SQL** to access, update, and manipulate the data stored **in a** database. However, **MySQL is** a database that stores the existing data **in a** database **in an** organized manner.

Q2.What is JAVA SE and JAVA EE ?(Types of Java Applications)

->

Java SE (formerly **J2SE**) stands for Java standard edition **and** is normally used for developing desktop applications, forms the core/base API. (e.g. calculator) **Java EE** (formerly **J2EE**) stands for Java enterprise edition for applications which run on servers, for example web sites.(e.g. red bus).

Q2.Wrapper Classes In java?

->

The wrapper classes in Java are used to convert primitive types (int, char, float, etc) into corresponding objects.

Each of the 8 primitive types has corresponding wrapper classes.

Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Convert Primitive Type to Wrapper Objects

We can also use the valueOf() method to convert primitive types into corresponding objects.

E.g :-

```
class Main {
  public static void main(String[] args) {
// create primitive types
    int a = 5;
    double b = 5.65;

//converts into wrapper objects
```

```
    Integer aObj = Integer.valueOf(a);
    Double bObj = Double.valueOf(b);

if(aObj instanceof Integer) {
      System.out.println("An object of Integer is created.");
    }

if(bObj instanceof Double) {
      System.out.println("An object of Double is created.");
    }
  }
}
```

In the above example, we have used the `valueOf()` method to convert the primitive types into objects.

Here, we have used the `instanceof` operator to check whether the generated objects are of `Integer` or `Double` type or not.

However, the Java compiler can directly convert the primitive types into corresponding objects. For example,

```
int a = 5;
// converts into object
Integer aObj = a;
double b = 5.6;
// converts into object
Double bObj = b;
```

This process is known as **auto-boxing**.

Wrapper Objects into Primitive Types

To convert objects into the primitive types, we can use the corresponding value methods (`intValue()`, `doubleValue()`, etc) present in each wrapper class.

Example 2: Wrapper Objects into Primitive Types

```
class Main {
  public static void main(String[] args) {
// creates objects of wrapper class
    Integer aObj = Integer.valueOf(23);
    Double bObj = Double.valueOf(5.55);
```

```
// converts into primitive types
   int a = aObj.intValue();
   double b = bObj.doubleValue();

System.out.println("The value of a: " + a);
   System.out.println("The value of b: " + b);
  }
}
```

Q3.What is LINQ?

->

LINQ in C# is used to work with data access from sources such as objects, data sets, SQL Server, and XML. LINQ stands for Language Integrated Query. LINQ is a data querying API with SQL like query syntaxes. LINQ provides functions to query cached data from all kinds of data sources. The data source could be a collection of objects, database or XML files. We can easily retrieve data from any object that implements the IEnumerable<T> interface.

The official goal of the LINQ family of technologies is to add "general purpose query facilities to the .NET Framework that apply to all sources of information, not just relational or XML data".

Q3.What do you Mean Differed Execution?

->

Q4.What do you Mean Immediate Execution?

->

The basic **difference between** a **Deferred execution** vs **Immediate execution** is that **Deferred execution of** queries produce a sequence **of** values, whereas **Immediate execution of** queries return a singleton value and is **executed** immediately.

Q5.What is Difference Between Class and Structure?

->

Class can create a subclass that will inherit parent's properties and methods, whereas **Structure** does not support # the inheritance. A **class** has all members private by default. A **struct** is a **class** where members are public by default.

Class ->

Class may have Methods, fields, Properties, Operators, Indexers, Events and other user defined types.
Access Specifiers are Private, Protected, Public and in C# Internal , Protected Internal.
class definiations can be splitted in files.
Static Classes are Sealed classes with static methods.
Only Single Inheritance is allowed.
Multiple Interface Inheritance is allowed.
Dyanamic Memory allocation in class.

struct ->

In C#, a structure is a value type data type. It helps you to make a single variable hold related data of various data types. The struct keyword is used for creating a structure.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- Subject
- Book ID

When passing a struct to a method, it is passed by value instead of as a reference.

Struct can be instantiated without using a new operator.

Structs Can be declare Constructor but they must take parameters.

Inheritance not allowed only a struct can implement interfaces.

static memory allocation in struct.

Q.6 C# Sealed Keyword ?

->

In c#, **sealed** is a keyword that is used to stop inheriting the particular class from other classes and we can also prevent overriding the particular properties or methods based on our requirements.

Generally, when we create a particular class we can inherit all the properties and methods in any class. In case, if you want to restrict access of defined class and its members, then by using a **sealed** keyword we can prevent other classes from inheriting the defined class.

Q.7 Const and ReadOnly Keyword ?

-> **Const -:** The variable declared as const, is a constant variable and cannot be modified anywhere. The const field's value should be given at compile time, where it is declared in class.

ReadOnly -: The variable declared as readonly, is assigned in constructor and cannot be modified in any other method afterwards. The readonly field's value may not be known at the compile time.

Q8.abstract keyword?

->

if function is declared as abstract within a class then we cannot create object of that class. Such class must be declared as abstract.

Functions declared abstract in base class must be overriden in the derived class. Otherwise we cannot even create ob ject of the derived classs.

Q.9 What is Interface in C# ?

->

Interface in C# is a blueprint of a class. It is like abstract class because all the methods which are declared inside the **interface** are abstract methods. It cannot have method body and cannot be instantiated. It is used to achieve multiple inheritance which can't be achieved by class.

Inshort Way to Achieve Abstraction that's Interface.

The class containing at least one pure virtual function are called as Abstract class.

Q.10 What is Fragile Base Class Problem ?

->

Take a simple scenario (below is the image) where we have a simple parent child class relationship. Assume that this child class is used in lot of projects and installed in lot of places. Now assume after the child class has been deployed across locations, after some months, there are some changes in

the parent class.
These changes can have cascading and unexpected behavior in child class as well. This unexpected behavior on child classes is termed as “Fragile class problem”.

Q.11 IComparable Interface,IComparer interface and IEquatable Interface ?

->

IComparable Interface:

Interface has a CompareTo method that takes a reference type as a parameter and returns an integer based on if current instance precedes, follows or occurs in the same position in the sort order as the other object (MSDN).

The implementation of the CompareTo(Object) method must return an Int32 that has one of the three values, as in the following table.

IComparer interface

The CompareTo method from IComparable interface can sort on only one field at a time, so sorting on different properties with it is not possible. IComparer interface provides Compare method that Compares two objects and returns a value indicating whether one is less than, equal to, or greater than the other.

A class that implements the IComparer interface must provide a Compare method that compares two objects.

For example, you could create a CarComparer class that implements IComparer and that has a Compare method that com¬pares Car objects by Name. You could then pass a CarComparer object to the *Array.Sort method*, and it can use that object to sort an array of Car objects.

IEquatable Interface

If a class implements the IComparable interface, it provides a CompareTo method that enables you to determine how two objects should be ordered. Sometimes, you may not need to know how two objects should be ordered, but you need to know instead whether the objects are equal. The IEquatable interface provides that capability by requiring a class to provide an Equals method.

Icloneable Interface

Supports cloning, which creates a new instance of a class with the same value as an existing instance.
#
The ICloneable interface contains one member, Clone, which is intended to support cloning beyond that supplied by MemberwiseClone. It is a procedure that can create a true, distinct copy of an object and all its dependent object, is to rely on the serialization features of the .NET framework.
There are two ways to clone an instance:
An instance is an actual object created to the specification defined by a class.

1. **Shallow copy** - may be linked to data shared by both the original and the copy
2. **Deep copy** - contains the complete encapsulated data of the original object

Q.11 What is Properties in C# ?

->

Property in C# is a member of a class that provides a flexible mechanism for classes to expose private fields. Internally, C# properties are special methods called accessors. A

C# property have two accessors, get property accessor and set property accessor. A get accessor returns a property value, and a set accessor assigns a new value. The value keyword represents the value of a property.
Properties in C# and .NET have various access levels that is defined by an access modifier. Properties can be read-write, read-only, or write-only. The read-write property implements both, a get and a set accessor. A write-only property implements a set accessor, but no get accessor. A read-only property implements a get accessor, but no set accessor.

Q.12 What is Auto-Implemented Properties in C# ?

->

auto-implemented properties make property-declaration more concise when no additional logic is required in the property accessors. They also enable client code to create objects. When you declare a property as shown in the following example, the compiler creates a private, anonymous backing field that can only be accessed through the property's get and set accessors.

```
// Auto-implemented properties for trivial get and set
public double TotalPurchases { get; set; } public string Name { get; set; } public int CustomerId { get; set; }
```

Q.13 What is Pre-Preprocessor Directives in c# ?

->

The preprocessor directives give instruction to the compiler to preprocess the information before actual compilation starts.
All preprocessor directives begin with #, and only white-space characters may appear before a preprocessor directive on a line. Preprocessor directives are not statements, so they do not end with a semicolon (;).
C# compiler does not have a separate preprocessor; however, the directives are processed as if there was one. In C# the preprocessor directives are used to help in conditional compilation. Unlike C and C++ directives, they are not used to create macros. A preprocessor directive must be the only instruction on a line.

This section contains information about the following C# preprocessor directives:

- #if
- #else
- #elif
- #endif
- #define
- #undef
- #warning
- #error
- #line
- #nullable
- #region
- #endregion
- #pragma
- #pragma warning

Q.14 what is Indexer in c# ?

-> An indexer allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a virtual array. You can then access the instance of this class using the array access operator ([ ]).

Q.15 what is delegate in c# ?
->
A delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance.
Delegates are used to pass methods as arguments to other methods. Event handlers are nothing more than methods that are invoked through delegates. You create a custom method, and a class such as a windows control can call your method when a certain event occurs. The following example shows a delegate declaration:

E.g :- public delegate int PerformCalculation(int x, int y);

Any method from any accessible class or struct that matches the delegate type can be assigned to the delegate. The method can be either static or an instance method. This makes it possible to programmatically change method calls, and also plug new code into existing classes.
Note

Delegates Overview

Delegates have the following properties:

- Delegates are similar to C++ function pointers, but delegates are fully object-oriented, and unlike C++ pointers to member functions, delegates encapsulate both an object instance and a method.
- Delegates allow methods to be passed as parameters.
- Delegates can be used to define callback methods.
- Delegates can be chained together; for example, multiple methods can be called on a single event.
- Methods do not have to match the delegate type exactly. For more information, see Using Variance in Delegates.
- C# version 2.0 introduced the concept of anonymous methods, which allow code blocks to be passed as parameters in place of a separately defined method. C# 3.0 introduced lambda expressions as a more concise way of writing inline code blocks. Both anonymous methods and lambda expressions (in certain contexts) are compiled to delegate types. Together, these features are now known as anonymous functions. For more information about lambda expressions, see Lambda expressions.

Q.16 what is Event in c# ?

->

Events in C#

The Event is something special that is going to happen. Here we will take an example of an event, where Microsoft launches the events for the developer. In this Event, Microsoft wants to aware the developer about the feature of the existing or new products. For this, Microsoft will use Email or other advertisement options to aware the developer about the Event. So, in this case, Microsoft will work as a publisher who raises the Event and notifies the developers about it. Developers will work as the subscriber of the Event who handles the Event.

Similarly, in C#, Events follow the same concept. In C#, Event can be subscriber, publisher, subscriber, notification, and a handler. Generally, the User Interface uses the events. Here we will take an example of Button control in Windows. Button performs multiple events such as click, mouseover, etc. The custom class contains the Event through which we will notify the other subscriber class about the other things which is going to happen. So, in this case, we will define the Event and inform the other classes about the Event, which contains the event handler.

The event is an encapsulated delegate. C# and .NET both support the events with the delegates. When the state of the application changes, events and delegates give the notification to the client application. Delegates and Events both are tightly coupled for dispatching the events, and event handling require the implementation of the delegates. The sending event class is known as the publisher, and the receiver class or handling the Event is known as a subscriber.

Key Points about the Events are:

The key points about the events are as:

1. In C#, event handler will take the two parameters as input and return the void.
2. The first parameter of the Event is also known as the source, which will publish the object.
3. The publisher will decide when we have to raise the Event, and the subscriber will determine what response we have to give.
4. Event can contain many subscribers.
5. Generally, we used the Event for the single user action like clicking on the button.
6. If the Event includes the multiple subscribers, then synchronously event handler invoked.

Declaration of the Event

Syntax

```
public event EventHandler CellEvent;
```

Q.17 What is Reflection in c# ?

->

Reflection provides objects (of type Type) that describe assemblies, modules, and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them.

```
// Using GetType to obtain type information:
int i = 42;
Type type = i.GetType();
Console.WriteLine(type);
```

Reflection overview

Reflection is useful in the following situations:

- When you have to access attributes in your program's metadata. For more information, see Retrieving Information Stored in Attributes.
- For examining and instantiating types in an assembly.
- For building new types at runtime. Use classes in System.Reflection.Emit.
- For performing late binding, accessing methods on types created at run time. See the topic Dynamically Loading and Using Types.

CLR loads the Assembly into appdomain.
Assemblies contain modules, modules contain types, and types contain members.
Reflection provides objects that encapsulate assemblies, modules, and types.
Reflection can be used to dynamically create an instance of a type, bind the type to an existing object, or get the type to an existing object, or get the type from an existing object.
The related classes are declared in system.Reflection namespace.
the classes of the System.Reflection.Emit namespace provides type structure that keeps information of type at run time.
the GetType method or typeof operator provides type structure that keeps information of type.

C# Reflection

In C#, reflection is a *process to get metadata of a type at runtime*. The System.Reflection namespace contains required classes for reflection such as:

- Type
- MemberInfo
- ConstructorInfo
- MethodInfo
- FieldInfo
- PropertyInfo
- TypeInfo
- EventInfo
- Module
- Assembly
- AssemblyName
- Pointer etc.

The System.Reflection.Emit namespace contains classes to emit metadata.

Q.18 What is Attributes in c# ?

->

Attributes provide a powerful method of associating metadata, or declarative information, with code (assemblies, types, methods, properties, and so forth). After an **attribute** is associated with a program entity, the **attribute** can be queried at run time by using a technique called reflection.

Q.19 What is Regular Expression in c# ?

->

A **regular expression** is a pattern that could be matched against an input text. The . Net framework provides a **regular expression** engine that allows such matching. A pattern consists of one or more

character literals, operators, or constructs.

Q.20 What is Threads in c# ?
->
C# threading allows developers to create multiple threads in C# and .NET.
When a new application starts on Windows, it creates a process for the application with a process id and some resources are allocated to this new process. Every process contains at least one primary thread which takes care of the entry point of the application execution. A single thread can have only one path of execution but as mentioned earlier, sometimes you may need multiple paths of execution and that is where threads play a role.
In .NET Core, the common language runtime (CLR) plays a major role in creating and managing threads lifecycle. In a new .NET Core application, the CLR creates a single foreground thread to execute application code via the Main method. This thread is called primary or main thread. Along with this main thread, a process can create one or more threads to execute a portion of the code. Additionally, a program can use the ThreadPool class to execute code on worker threads that are managed by the CLR.
A C# program is single threaded by design. That means, only one path of the code is executed at a time by the main or primary thread. The entry point of a C# program starts in the Main method and that is the path of the primary thread.

Q.21 How do we achieve synchronization in dotnet?
->By lock object

C# Thread Synchronization

Synchronization is a technique that allows only one thread to access the resource for the particular time. No other thread can interrupt until the assigned thread finishes its task.
In multithreading program, threads are allowed to access any resource for the required execution time. Threads share resources and executes asynchronously. Accessing shared resources (data) is critical task that sometimes may halt the system. We deal with it by making threads synchronized.
It is mainly used in case of transactions like deposit, withdraw etc.

Advantage of Thread Synchronization

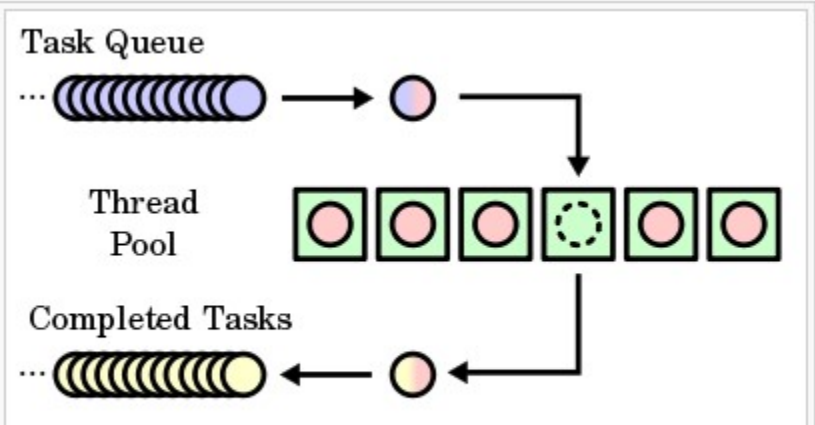
- Consistency Maintain
- No Thread Interference

C# Lock

We can use C# **lock keyword** to execute program synchronously. It is used to get lock for the current thread, execute the task and then release the lock. It ensures that other thread does not interrupt the execution until the execution finish.

What is Task in C#?

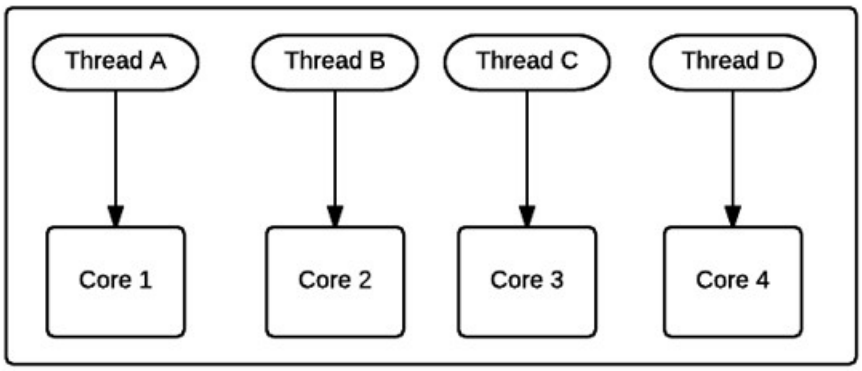
.NET framework provides Threading.Tasks class to let you create tasks and run them asynchronously. A task is an object that represents some work that should be done. The task can tell you if the work is completed and if the operation returns a result, the task gives you the result.



A sample thread pool (green boxes) with waiting tasks (blue) and completed tasks (yellow)

What is Thread?

.NET Framework has thread-associated classes in System.Threading namespace. A Thread is a small set of executable instructions.



Why we need Tasks

It can be used whenever you want to execute something in parallel. Asynchronous implementation is easy in a task, using' async' and 'await' keywords.

Why we need a Thread

When the time comes when the application is required to perform few tasks at the same time.

Here is a beginner tutorial on [Introduction to Threading in C#]

How to create a Task

```
1. static void Main(string[] args) {
2.     Task < string > obTask = Task.Run(() => (
3.         return" Hello"));
4.     Console.WriteLine(obTask.result);
5. }
```

How to create a Thread

```
1. static void Main(string[] args) {
2.     Thread thread = new Thread(new ThreadStart(getMyName));
3.     thread.Start();
4. }
5. public void getMyName() {}
```

Differences Between Task And Thread

Here are some differences between a task and a thread.

1. The Thread class is used for creating and manipulating a thread in Windows. A Task represents some asynchronous operation and is part of the Task Parallel Library, a set of APIs for running tasks asynchronously and in parallel.
2. The task can return a result. There is no direct mechanism to return the result from a thread.
3. Task supports cancellation through the use of cancellation tokens. But Thread doesn't.
4. A task can have multiple processes happening at the same time. Threads can only have one task running at a time.
5. We can easily implement Asynchronous using ’async’ and ‘await’ keywords.
6. A new Thread()is not dealing with Thread pool thread, whereas Task does use thread pool thread.
7. A Task is a higher level concept than Thread.