2020-11-10

```
int i = 101;
object obj = (object)i; // Boxing
```

Unboxing - Converting a reference type to a value typpe is called unboxing. An example is shown below.

```
obj = 101;
i = (int)obj; // Unboxing
```

Is boxing an implicit conversion?
Yes, boxing happens implicitly.

Is unboxing an implicit conversion?
No, unboxing is an explicit conversion.

What are the advantages of using interfaces

1. Interfaces allow us to implement polymorphic behaviour. Ofcourse, abstract classes can also be used to implement polymorphic behaviour.

2. Interfaces allow us to develop very loosely coupled systems.
3. Interfaces enable mocking for better unit testing.

4. Interfaces enables us to implement multiple class inheritance in C#.

5. Interfaces are great for implementing Inverson of Control or Dependancy Injection.

6. Interfaces enable parallel application development.

What do you mean by casting a data type?
Converting a variable of one data type to another data type is called casting. This is also called as data type conversion.

What are the 2 kinds of data type conversions in C#?
Implicit conversions: No special syntax is required because the conversion is type safe and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.

Explicit conversions: Explicit conversions require a cast operator. The source and destination variables are compatible, but there is a risk of data loss because the type of the destination variable is a smaller size than (or is a base class of) the source variable.

What is the difference between an implicit conversion and an explicit conversion?

1. Explicit conversions require a cast operator where as an implicit converstion is done automatically.
2. Explicit conversion can lead to data loss where as with implicit conversions there is no data loss.

What type of data type conversion happens when the compiler encounters the following code?

```
ChildClass CC = new ChildClass();
ParentClass PC = new ParentClass();
```

Implicit Conversion. For reference types, an implicit conversion always exists from a class to any one of its direct or indirect base classes or interfaces. No special syntax is necessary because a derived class always contains all the members of a base class.

Will the following code compile?

```
double d = 9999.11;
int i = d;
```

No, the above code will not compile. Double is a larger data type than integer. An implicit conversion is not done automatically bcos there is a data loss. Hence we have to use explicit conversion as shown below.

```
double d = 9999.11;
int i = (int)d; //Cast double to int.
```

If you want to convert a base type to a derived type, what type of conversion do you use?
Explicit conversion as shown below.

```
//Create a new derived type.
Car C1 = new Car();
// Implicit conversion to base type is safe.
Vehicle V = C1;

// Explicit conversion is required to cast back to derived type. The code below will compile but throw an exception at run time if the right-side object is not a Car object.
Car C2 = (Car) V;
```

What operators can be used to cast from one reference type to another without the risk of throwing an exception?
The is and as operators can be used to cast from one reference type to another without the risk of throwing an exception.

If casting fails what type of exception is thrown?
InvalidCastException

Is C# a strongly-typed language?
Yes

What are the 2 broad classifications of data types available in C#?
1. Built in data types.
2. User defined data types.

Give some examples for built in datatypes in C#?
1. int
2. float
3. bool

How do you create user defined data types in C#?
You use the struct, class, interface, and enum constructs to create your own custom types. The .NET Framework class library itself is a collection of custom types provided by Microsoft that you can use in your own applications.

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

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

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

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

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

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

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

Usually in .NET, the CLR takes care of memory management. Is there any need for a programmer to explicitly release memory and resources? If yes, why and how?

If the application is using expensive external resource, it is recommend to explicitly release the resource before the garbage collector runs and frees the object. We can do this by implementing the Dispose method from the IDisposable interface that performs the necessary cleanup for the object. This can considerably improve the performance of the application.

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

In Microsoft.NET version 1.0 there were collections, such as the ArrayList for working with groups of objects.
An ArrayList is much like an array, except it could automatically grow and offered many convenience methods that arrays don't have.
The problem with ArrayList and all the other .NET v1.0 collections is that they operate on type object. Since all objects derive from the object type, you can assign anything to an ArrayList. The problem with this is that you incur performance overhead converting value type objects to and from the object type and a single ArrayList could accidentally hold different types, which would cause a hard to find errors at runtime because you wrote code to work with one type. Generic collections fix these problems.

A generic collection is strongly typed (type safe), meaning that you can only put one type of object into it.
This eliminates type mismatches at runtime. Another benefit of type safety is that performance is better with value type objects because they don't incur overhead of being converted to and from type object. With generic collections, you have the best of all worlds because they are strongly typed, like arrays, and you have the additional functionality, like ArrayList and other non-generic collections, without the problems.

It is always good to use generics rather than using ArrayList,Hashtable etc, found in System.Collections namespace. The only reason why you may want to use System.Collections is for backward compatibility.

Unit Test private method in C# .NET

This is a very common c# interview question. As a developer all of us know, how to unit test public members of a class. All you do is create an instance of the respectiveclass and invoke the methods using the created instance. So, unit testing public methods is very straight forward, but if the method that we want to unit test is a private method, then we cannot access it outside the class and hence cannot easily unit test it.

Consider the example class shown below. CalculatePower() method with in the Maths class is private and we want to unit test this method. Also, note that CalculatePower() is an instance private method. In another articel we will discuss

the concept of unit testing a private static method. Microsoft's unit testing assembly contains a class called PrivateObject, which can be used to unit test private methods very easily.
Microsoft.VisualStudio.TestTools.UnitTesting.PrivateObject is the fully qualified name. Click here, to read an articel on, unit testing a private static method with an example.

```
public class Maths
{
   private int CalculatePower(int Base, int Exponent)
   {
      int Product = 1;

      for (int i = 1; i <= Exponent; i++)
      {
         Product = Product * Base;
      }
      return Product;
   }
}
```

To unit test this method, We create an instance of the class and pass the created instance to the constructor of PrivateObject class. Then we use the instance of the PrivateObject class, to invoke the private method. The fully completed unit test is shown below.

```
[TestMethod()]
public void CalculatePowerTest()
{
   Maths mathsclassObject = new Maths();
   PrivateObject privateObject = new PrivateObject(mathsclassObject);
   object obj = privateObject.Invoke("CalculatePower", 2, 3);
   Assert.AreEqual(8, (int)obj);
}
```

Explain polymorphism in C# with a simple example?
Polymorphism allows you to invoke derived class methods through a base class reference during run-time. An example is shown below.

```
using System;
public class DrawingObject
{
public virtual void Draw()
{
Console.WriteLine("I am a drawing object.");
}
}
public class Triangle : DrawingObject
{
public override void Draw()
```

```
{
Console.WriteLine("I am a Triangle.");
}
}
public class Circle : DrawingObject
{
public override void Draw()
{
Console.WriteLine("I am a Circle.");
}
}
public class Rectangle : DrawingObject
{
public override void Draw()
{
Console.WriteLine("I am a Rectangle.");
}
}
public class DrawDemo
{
public static void Main()
{
DrawingObject[] DrawObj = new DrawingObject[4];

DrawObj[0] = new Triangle();
DrawObj[1] = new Circle();
DrawObj[2] = new Rectangle();
DrawObj[3] = new DrawingObject();

foreach (DrawingObject drawObj in DrawObj)
{
drawObj.Draw();
}
}
}
```

When can a derived class override a base class member?
A derived class can override a base class member only if the base class member is declared as virtual or abstract.

What is the difference between a virtual method and an abstract method?
A virtual method must have a body where as an abstract method should not have a body.

Can fields inside a class be virtual?
No, Fields inside a class cannot be virtua. Only methods, properties, events and indexers can be virtual.

Give an example to show for hiding base class methods?
Use the new keyword to hide a base class method in the derived class as shown in the

example below.

```
using System;
public class BaseClass
{
public virtual void Method()
{
Console.WriteLine("I am a base class method.");
}
}
public class DerivedClass : BaseClass
{
public new void Method()
{
Console.WriteLine("I am a child class method.");
}

public static void Main()
{
DerivedClass DC = new DerivedClass();
DC.Method();
}
}
```

Can you access a hidden base class method in the derived class?
Yes, Hidden base class methods can be accessed from the derived class by casting the instance of the derived class to an instance of the base class as shown in the example below.

```
using System;
public class BaseClass
{
public virtual void Method()
{
Console.WriteLine("I am a base class method.");
}
}
public class DerivedClass : BaseClass
{
public new void Method()
{
Console.WriteLine("I am a child class method.");
}

public static void Main()
{
DerivedClass DC = new DerivedClass();
((BaseClass)DC).Method();
}
}
```

ASP.NET Page is very slow. What will you do to make it fast

This is a very common asp.net interview question asked in many interviews. There are several reasons for the page being slow. We need to identify the cause.

1. Find out which is slow, is it the application or the database : If the page is executing SQL queries or stored procedures, run those on the database and check how long do they take to run. If the queries are taking most of the time, then you know you have to tune the queries for better performance. To tune the queries, there are several ways and I have listed some of them below.
   a) Check if there are indexes to help the query
   b) Select only the required columns, avoid Select *.
   c) Check if there is a possiblity to reduce the number of joins
   d) If possible use NO LOCK on your select statements
   e) Check if there are cursors and if you can replace them with joins

2. If the queries are running fast, then we know it is the application code that is causing the slowness. Isolate the page event that is causing the issue by turning tracing on. To turn tracing on, set Trace="true" in the page directive. Once you have tracing turned on you should see trace information at the bottom of the page as shown in the image below. In this case Page Load event is taking the maximum time. So we know, the code in Page_Load event is causing the issue. Once you look at the code, you should be able to nail down the issue.

What are the advantages and disadvantages of a layered architecture
The following are the advantages of a layered architecture:

Layered architecture increases flexibility, maintainability, and scalability. In a Layered architecture we separate the user interface from the business logic, and the business logic from the data access logic. Separation of concerns among these logical layers and components is easily achieved with the help of layered architecture.

Multiple applications can reuse the components. For example if we want a windows user interface rather than a web browser interface, this can be done in an easy and fast way by just replacing the UI component. All the other components like business logic, data access and the database remains the same. Layered architecture allows to swap and reuse components at will.

Layered architecture enables teams to work on different parts of the application parallely with minimal dependencies on other teams.

Layered architecture enables develop loosely coupled systems.

Different components of the application can be independently deployed, maintained, and updated, on different time schedules.

Layered architecture also makes it possible to configure different levels of security to different components deployed on different boxes. sO Layered architecture, enables you to secure portions of the application behind the firewall

and make other components accessible from the Internet.

Layered architecture also helps you to test the components independently of each other.

The following are the disadvantages of a layered architecture:

There might be a negative impact on the performance as we have the extra overhead of passing through layers instead of calling a component directly.

Development of user-intensive applications can sometime take longer if the layering prevents the use of user interface components that directly interact with the database.

The use of layers helps to control and encapsulate the complexity of large applications, but adds complexity to simple applications.

Changes to lower level interfaces tend to percolate to higher levels, especially if the relaxed layered approach is used.
