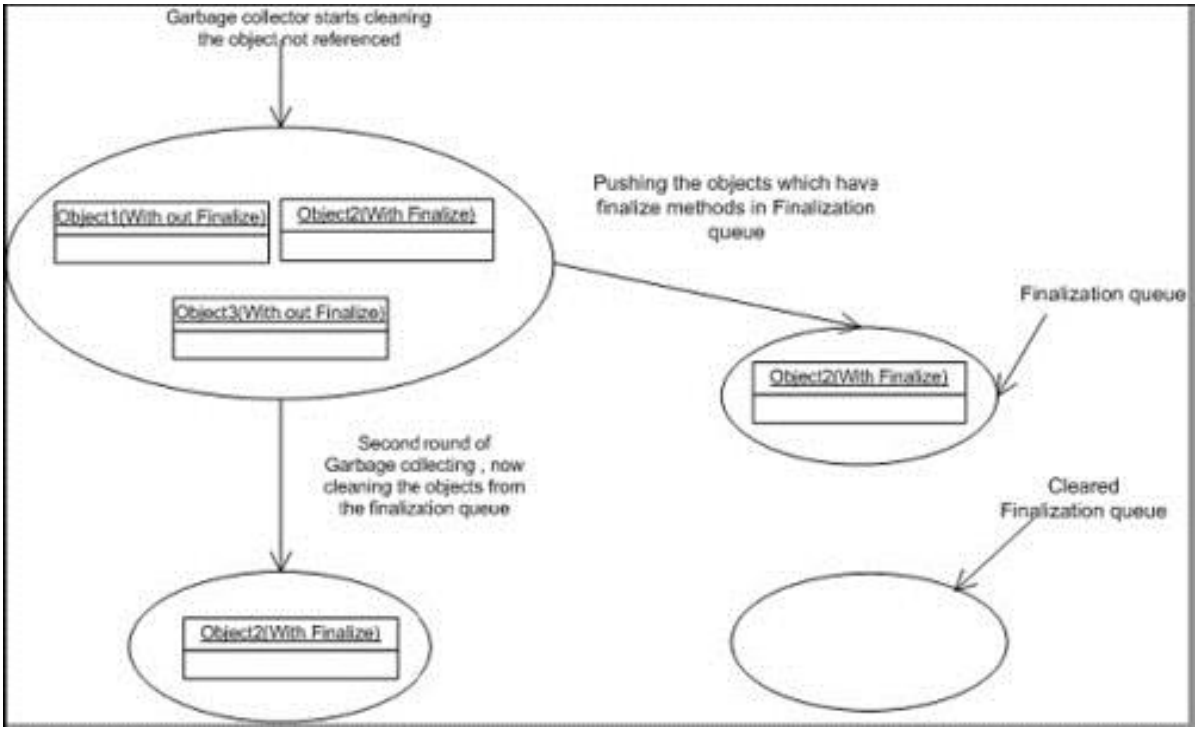
C# and ASP.Net Interview Question and Answers

What is the significance of Finalize method in .NET?

.NET Garbage collector does almost all clean up activity for your objects. But unmanaged
resources (ex: - Windows API created objects, File, Database connection objects, COM objects
etc) is outside the scope of .NET framework we have to explicitly clean our resources. For these
types of objects, .NET framework provides Object. Finalize method, which can be overridden and
clean up code for unmanaged resources can be put in this section?

Why is it preferred to not use finalize for clean up?

Problem with finalize is that garbage collection has to make two rounds in order to remove
objects which have finalize methods.
Below figure will make things clear regarding the two rounds of garbage collection rounds
performed for the objects having finalized methods.



Note:Few of the content is taken from various blogs/articles.

In this scenario there are three objects Object1, Object2, and Object3. Object2 has the finalize
method overridden and remaining objects do not have the finalize method overridden.
Now when garbage collector runs for the first time it searches for objects whose memory has to
free. He can see three objects but only cleans the memory for Object1 and Object3. Object2 it
pushes to the finalization queue.
Now garbage collector runs for the second time. He see's there are no objects to be released and
then checks for the finalization queue and at this moment, it clears object2 from the memory.
So if you notice that object2 was released from memory in the second round and not first. That is
why the best practice is not to write clean up Non.NET resources in Finalize method rather use
the DISPOSE.

What is the use of DISPOSE method?
Dispose method belongs to 'IDisposable' interface. We had seen in the previous section how bad
it can be to override the finalize method for writing the cleaning of unmanaged resources. So if
any object wants to release its unmanaged code best is to implement I Disposable and override
the Dispose method of I Disposable interface. Now once your class has exposed the Dispose
method it is the responsibility of the client to call the Dispose method to do the cleanup.

How do I force the Dispose method to be called automatically, as clients can forget to call Dispose method?
Call the Dispose method in Finalize method and in Dispose method suppress the finalize method
using GC.SuppressFinalize. Below is the sample code of the pattern. This is the best way we do

clean our unallocated resources and yes not to forget we do not get the hit of running the Garbage
collector twice.

```
public class CleanClass : IDisposable
  {
     public void Dispose()
     {
        GC.SuppressFinalize(this);
     }
     protected override void Finalize()
     {
        Dispose();
     }
  }
```

What is an interface and what is an abstract class? Please, expand by examples of using both. Explain why.

Answers1:

In a interface class, all methods are abstract without implementation where as in an abstract class some methods we can define concrete. In interface, no accessibility modifiers are allowed. An abstract class may have accessibility modifiers. Interface and abstract class are basically a set of rules which u have to follow in case u r using them(inheriting them).

Answers2:

Abstract classes are closely related to interfaces. They are classes that cannot be instantiated, and are frequently either partially implemented, or not at all implemented. One key difference between abstract classes and interfaces is that a class may implement an unlimited number of interfaces, but may inherit from only one abstract (or any other kind of) class. A class that is derived from an abstract class may still implement interfaces. Abstract classes are useful when creating components because they allow you specify an invariant level of functionality in some methods, but leave the implementation of other methods until a specific implementation of that class is needed. They also version well, because if additional functionality is needed in derived classes, it can be added to the base class without breaking code.

Answers3:

Abstract Classes

An abstract class is the one that is not used to create objects. An abstract class is designed to act as a base class (to be inherited by other classes). Abstract class is a design concept in program development and provides a base upon which other classes are built. Abstract classes are similar to interfaces. After declaring an abstract class, it cannot be instantiated on it's own, it must be inherited. Like interfaces, abstract classes can specify members that must be implemented in inheriting classes. Unlike interfaces, a class can inherit only one abstract class. Abstract classes can only specify members that should be implemented by all inheriting classes.

Answers4:

An interface looks like a class, but has no implementation. They're great for putting together plug-n-play like architectures where components can be interchanged at will. Think Firefox Plug-in extension implementation. If you need to change your design, make it an interface. However, you may have abstract classes that provide some default behavior. Abstract classes are excellent candidates inside of application frameworks.

Answers5:

One additional key difference between interfaces and abstract classes (possibly the most important one) is that multiple interfaces can be implemented by a class, but only one abstract class can be inherited by any single class.

Some background on this: C++ supports multiple inheritance, but C# does not. Multiple inheritance in C++ has always be controversial, because the resolution of multiple inherited implementations of the same method from different base classes is hard to control and anticipate. C# decided to avoid this problem by allowing a class to implement multiple interfaces, which do not contain method implementations, but restricting a class to have at most a single parent class. Although this can result in redundant implementations of the same method when different classes implement the same interface, it is still an excellent compromise.

Another difference between interfaces and abstract classes is that an interface can be implemented by an abstract class, but no class, abstract or otherwise, can be inherited by an interface.

Answers6:

What is an Abstract class?

An abstract class is a special kind of class that cannot be instantiated. So the question is why we need a class that cannot be instantiated? An abstract class is only to be sub-classed (inherited from). In other words, it only allows other classes to inherit from it but cannot be instantiated. The advantage is that it enforces certain hierarchies for all the subclasses. In simple words, it is a kind of contract that forces all the subclasses to carry on the same hierarchies or standards.

What is an Interface?

An interface is not a class. It is an entity that is defined by the word Interface. An interface has no implementation; it only has the signature or in other words, just the definition of the methods without the body. As one of the similarities to Abstract class, it is a contract that is used to define hierarchies for all subclasses or it defines specific set of methods and their arguments. The main difference between them is that a class can implement more than one interface but can only inherit from one abstract class. Since C# doesn't support multiple inheritance, interfaces are used to implement multiple inheritance.

How does output caching work in ASP.NET?

Output caching is a powerful technique that increases request/response throughput by caching the content generated from dynamic pages. Output caching is enabled by default, but output from any given response is not cached unless explicit action is taken to make the response cacheable.

To make a response eligible for output caching, it must have a valid expiration/validation policy and public cache visibility. This can be done using either the low-level OutputCache API or the high-level @ OutputCache directive. When output caching is enabled, an output cache entry is created on the first GET request to the page. Subsequent GET or HEAD requests are served from the output cache entry until the cached request expires.

The output cache also supports variations of cached GET or POST name/value pairs. The output cache respects the expiration and validation policies for pages. If a page is in the output cache and has been marked with an expiration policy that indicates that the page expires 60 minutes from the time it is cached, the page is removed from the output cache after 60 minutes. If another request is received after that time, the page code is executed and the page can be cached again. This type of expiration policy is called absolute expiration - a page is valid until a certain time.

What is connection pooling and how do you make your application use it?
Opening database connection is a time consuming operation.
Connection pooling increases the performance of the applications by reusing the active database connections instead of create new connection for every request.
Connection pooling Behaviour is controlled by the connection string parameters.
Follwing the the 4 parameters that control most of the connection pooling behaviour.
1. Connect Timeout
2. Max Pool Size
3. Min Pool Size
4. Pooling

What are different methods of session maintenance in ASP.NET?
3 types:
In-process storage.
Session State Service.
Microsoft SQL Server.

In-Process Storage
The default location for session state storage is in the ASP.NET process itself.

Session State Service
As an alternative to using in-process storage for session state, ASP.NET provides the ASP.NET State Service. The State Service gives you an out-of-process alternative for storing session state that is not tied quite so closely to ASP. Net's own process.

To use the State Service, you need to edit the sessionState element in your ASP.NET application's web.config file:
You'll also need to start the ASP.NET State Service on the computer that you specified in the stateConnectionString attribute. The .NET Framework installs this service, but by default it's set to manual startup. If you're going to depend on it for storing session state, you'll want to change that to automatic startup by using the Services MMC plug-in in the Administrative Tools group.

If you make these changes, and then repeat the previous set of steps, you'll see slightly different behavior: session state persists even if you recycle the ASP.NET process.

There are two main advantages to using the State Service. First, it is not running in the same process as ASP.NET, so a crash of ASP.NET will not destroy session information. Second, the stateConnectionString that's used to locate the State Service

includes the TCP/IP address of the service, which need not be running on the same computer as ASP.NET. This allows you to share state information across a web garden (multiple processors on the same computer) or even across a web farm (multiple servers running the application). With the default in-process storage, you can't share state information between multiple instances of your application.

The major disadvantage of using the State Service is that it's an external process, rather than part of ASP.NET. That means that reading and writing session state is slower than it would be if you kept the state in-process. And, of course, it's one more process that you need to manage. As an example of the extra effort that this can entail, there is a bug in the initial release of the State Service that allows a determined attacker to crash the ASP.NET process remotely. If you're using the State Service to store session state, you should install the patch from Microsoft Security Bulletin MS02-66, or install SP2 for the .NET Framework.

Microsoft SQL Server

The final choice for storing state information is to save it in a Microsoft SQL Server database. To use SQL Server for storing session state, you need to perform several setup steps:

Run the InstallSqlState.sql script on the Microsoft SQL Server where you intend to store session state. This script will create the necessary database and database objects. The .NET Framework installs this script in the same folder as its compilers and other tools– for example, C:\WINNT\Microsoft.NET\Framework\v1.0.3705 on a Windows 2000 computer with the 1.0 version of the Framework. Edit the sessionState element in the web.config file for your ASP.NET application as follows:

Supply the server name, user name, and password for a SQL Server account that has access to the session state database in the sqlConnectionString attribute.

Like the State Service, SQL Server lets you share session state among the processors in a web garden or the servers in a web farm. But you also get the additional benefit of persistent storage. Even if the computer hosting SQL Server crashes and is restarted, the session state information will still be present in the database, and will be available as soon as the database is running again. That's because SQL Server, being an industrial-strength database, is designed to log its operations and protect your data at (almost) all costs. If you're willing to invest in SQL Server clustering, you can keep the session state data available transparently to ASP.NET even if the primary SQL Server computer crashes.

Like the State Service, SQL Server is slower than keeping session state in process. You also need to pay additional licensing fees to use SQL Server for session state in a production application. And, of course, you need to worry about SQL Server-specific threats such as the “Slammer” worm.

What does the "EnableViewState" property do? Why would I want it on or off?
Enable ViewState turns on the automatic state management feature that enables server controls to re-populate their values on a round trip without requiring you to write any code. This feature is not free however, since the state of a control is passed to and from the server in a hidden form field. You should be aware of when ViewState is helping you and when it is not. For example, if you are binding a control to data on every round trip (as in the datagrid example in tip #4), then you do not need the control to maintain it’s view state, since you will wipe out any re-populated data in any case. ViewState is enabled for all server controls by default. To disable it, set the EnableViewState property of the control to false.

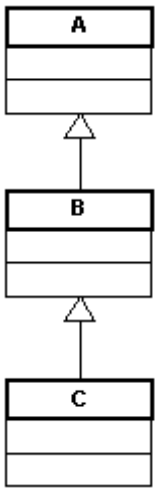
What is the difference between Server.Transfer and Response.Redirect?
Why would I choose one over the other? Server.Transfer() : client is shown as it is on the requesting page only, but the all the content is of the requested page. Data can be persist across the pages using Context.Item collection, which is one of the best way to transfer data from one page to another keeping the page state alive.
Response.Dedirect() :client know the physical location (page name and query string as well). Context.Items loses the persistence when navigate to destination page. In earlier versions of IIS, if we wanted to send a user to a new Web page, the only option we had was Response.Redirect. While this method does accomplish our goal, it has several important drawbacks. The biggest problem is that this method causes each page to be treated as a separate transaction. Besides making it difficult to maintain your transactional integrity, Response.Redirect introduces some additional headaches. First, it prevents good encapsulation of code. Second, you lose access to all of the properties in the Request object. Sure, there are workarounds, but they’re difficult. Finally, Response.Redirect necessitates a round trip to the client, which, on high-volume sites, causes scalability problems. As you might suspect, Server.Transfer fixes all of these problems. It does this by performing the transfer on the server without requiring a roundtrip to the client.

Polymorphism, Method hiding and overriding :
One of the fundamental concepts of object oriented software development is **polymorphism**. The term polymorphism (from the Greek meaning "having multiple forms") in OO is the characteristic of being able to assign a different meaning or usage

to something in different contexts - specifically, to allow a variable to refer to more than one type of object.

Example Class Hierarchy

Let's assume the following simple class hierarchy with classes A, B and C for the discussions in this text. A is the super- or base class, B is derived from A and C is derived from class B. In some of the easier examples, we will only refer to a part of this class hierarchy.



Inherited Methods

A method Foo() which is declared in the base class A and not redeclared in classes B or C is inherited in the two subclasses

```
using System;
namespace Polymorphism
{
    class A
    {
        public void Foo() { Console.WriteLine("A::Foo()"); }
    }

    class B : A {}

    class Test
    {
        static void Main(string[] args)
```

```
        {
            A a = new A();
            a.Foo();  // output --> "A::Foo()"

            B b = new B();
            b.Foo();  // output --> "A::Foo()"
        }
    }
}
```

The method Foo() can be overridden in classes B and C:

```
using System;
namespace Polymorphism
{
    class A
    {
        public void Foo() { Console.WriteLine("A::Foo()"); }
    }

    class B : A
    {
        public void Foo() { Console.WriteLine("B::Foo()"); }
    }

    class Test
    {
        static void Main(string[] args)
        {
            A a;
            B b;

            a = new A();
            b = new B();
            a.Foo();  // output --> "A::Foo()"
            b.Foo();  // output --> "B::Foo()"

            a = new B();
            a.Foo();  // output --> "A::Foo()"
        }
```

```
    }
}
```

There are two problems with this code.
The output is not really what we, say from Java, expected. The method Foo() is a non-virtual method. C# requires the use of the keyword virtual in order for a method to actually be virtual. An example using virtual methods and polymorphism will be given in the next section.
Although the code compiles and runs, the compiler produces a warning:
...\polymorphism.cs(11,15): warning CS0108: The keyword new is required on 'Polymorphism.B.Foo()' because it hides inherited member 'Polymorphism.A.Foo()'
This issue will be discussed in section Hiding and Overriding Methods.

Virtual and Overridden Methods

Only if a method is declared virtual, derived classes can override this method if they are explicitly declared to override the virtual base class method with the **override** keyword.

```
using System;
namespace Polymorphism
{
    class A
    {
        public virtual void Foo() { Console.WriteLine("A::Foo()"); }
    }

    class B : A
    {
        public override void Foo() { Console.WriteLine("B::Foo()"); }
    }

    class Test
    {
        static void Main(string[] args)
        {
            A a;
            B b;

            a = new A();
            b = new B();
            a.Foo();  // output --> "A::Foo()"
```

```
            b.Foo();  // output --> "B::Foo()"

            a = new B();
            a.Foo();  // output --> "B::Foo()"
        }
    }
}
```

Method Hiding

Why did the compiler in the second listing generate a warning? Because C# not only supports method overriding, **but also method hiding**. Simply put, if a method is not overriding the derived method, it is hiding it. A hiding method has to be declared using the **new** keyword. The correct class definition in the second listing is thus:

```
using System;
namespace Polymorphism
{
    class A
    {
        public void Foo() { Console.WriteLine("A::Foo()"); }
    }

    class B : A
    {
        public new void Foo() { Console.WriteLine("B::Foo()"); }
    }

    class Test
    {
        static void Main(string[] args)
        {
            A a;
            B b;

            a = new A();
            b = new B();
            a.Foo();  // output --> "A::Foo()"
            b.Foo();  // output --> "B::Foo()"

            a = new B();
```

```
            a.Foo();  // output --> "A::Foo()"
        }
    }
}
```

Combining Method Overriding and Hiding

Methods of a derived class can both be virtual and at the same time hide the derived method. In order to declare such a method, both keywords virtual and new have to be used in the method declaration:

```
class A
{
    public void Foo() {}
}

class B : A
{
    public virtual new void Foo() {}
}
```

A class C can now declare a method Foo() that either overrides or hides Foo() from class B:

```
class C : B
{
    public override void Foo() {}
    // or
    public new void Foo() {}
}
```

Conclusion

- C# is not Java.
- Only methods in base classes need not override or hide derived methods. All methods in derived classes require to be either defined as new or as override.
- Know what your doing and look out for compiler warnings.

Few of the references taken from ,
http://www.dotnetinterviewquestions.in/article_net-interview-questions-6th-edition-sixth-edition-by-shivprasad-koirala_144.html