[GarbageCollector](http://csharpinterviewfaq.blogspot.com/2010/01/what-is-garbage-collectiongc.html)

[**What is garbage collection(GC)?**](http://www.dotnetinterviewfaqs.com/what-is-gc-garbage-collection-in-microsoft-dot-net.aspx)  
The .NET Framework's garbage collector manages the allocation and release of memory for your application. Each time you use the new operator to create an object, the runtime allocates memory for the object from the managed heap. As long as address space is available in the managed heap, the runtime continues to allocate space for new objects. However, memory is not infinite. Eventually the garbage collector must perform a collection in order to free some memory. The garbage collector's optimizing engine determines the best time to perform a collection, based upon the allocations being made. When the garbage collector performs a collection, it checks for objects in the managed heap that are no longer being used by the application and performs the necessary operations to reclaim their memory.  
**What Object.Finalize method does?**  
allows an object to clean up its unmanaged resources properly when the garbage collector reclaims the memory used by the object.

**What are the limitations of Finalizers?**  
· The exact time of execution during garbage collection is undefined.

* · The finalizers of two objects are not guaranteed to run in any specific order, even if one object refers to the other.   
  · The thread on which the finalizer is run is unspecified.  
  Finalize cannot be called on resurrected objects during garbage collection.

**What is strong reference and what is weak reference?**

There are two types of memory references, strong & weak. When a root references an object, it is said to be a strong reference as the object is being pointed to by application code. The other type of object, that is not being referenced by the application code is called the weak reference, and this may be collected by the GC. However, this may still be accessed by the application code if required. But for the application to access the weakly referenced object, this has to be converted to a strong reference (and note that this has to be done before the GC collects the weakly referenced object).  
**What is Dispose Method?**  
A type's Dispose method should release all the resources that it owns. It should also release all resources owned by its base types by calling its parent type's Dispose method. The parent type's Dispose method should release all resources that it owns and in turn call its parent type's Dispose method, propagating this pattern through the hierarchy of base types. To help ensure that resources are always cleaned up appropriately, a Dispose method should be callable multiple times without throwing an exception.  
A Dispose method should call the GC.SuppressFinalize method for the object it is disposing. If the object is currently on the finalization queue, GC.SuppressFinalize prevents its Finalize method from being called. Remember that executing a Finalize method is costly to performance. If your Dispose method has already done the work to clean up the object, then it is not necessary for the garbage collector to call the object's Finalize method.  
**What will you do to prevent the garbage collector from calling [Object.Finalize](ms-help://MS.MSDNQTR.v90.en/fxref_mscorlib/html/99598e55-a3d9-940d-93de-e09c36440968.htm) on an object that does not require it?**call GC.SuppressFinalize() method .

**What are Destructors?**

Destructors are the C# mechanism for performing cleanup operations. Destructors are used to destruct instances of classes. A class can only have one destructor.Destructors cannot be inherited or overloaded. · Destructors cannot be called. They are invoked automatically.

} The destructor implicitly calls Finalize on the base class of the object. Finalize method is called recursively for all instances in the inheritance chain, from the most-derived to the least-derived. Empty destructors should not be used. 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. The programmer has no control over when the destructor is called because this is determined by the garbage collector. 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 any) and reclaims the memory used to store the object. Destructors are also called when the program exits. It is possible to force garbage collection by calling Collect, but most of the time, this should be avoided because it may create performance issues

**Explain About latency modes?**

To reclaim objects, the garbage collector must stop all of an application's executing threads. In some situations, such as when an application retrieves data or displays content, a full garbage collection can occur at a critical time and impede performance. You can adjust the intrusiveness of the garbage collector by setting the LatencyMode property to one of the GCLatencyMode values.  
Latency refers to the time that the garbage collector intrudes in your application. During low latency periods the garbage collector is more conservative, and less intrusive, in reclaiming objects. Generation 2 collections occur less frequently, which causes the application working set to grow over time. As a result, it is recommended that you use the LowLatency mode only for the short period of time when it is needed. Otherwise, if the system is under memory pressure, the garbage collector will trigger a collection, which can briefly pause the application and disrupt a time-critical operation.  
You should use the latency mode with applications that include a block of code that runs over a brief period of time and must run with minimal interruptions from the runtime. Although the LowLatency mode is designed to be used in scenarios where there are some time constraints, it is not intended to be a solution for scenarios where there are strict real-time constraints.

**How many Options of Latencymode for garbage collector? what are they?**

There are three latency modes

1)Batch: Disables garbage collection concurrency and reclaims objects in a batch call. This is the most intrusive mode.This mode is designed for maximum throughput at the expense of responsiveness.

2)Interactive:The default latency mode. Enables garbage collection concurrency and reclaims objects while the application is running.

3)LowLatency:Enables garbage collection that is more conservative in reclaiming objects. Collections occur less frequently. This is the least intrusive mode.  
This mode is not available on the server garbage collector.

Other Questions:

**1)If i have 100 objects in my application are out of scope.when first time garbage collected how many objects memory reference are free?**

All the objects of genereation 0 are freed.

**2)destructor, Dispose, Finalize - How they relate to each other?**

These are used to release unmanaged resources

**3)How we free the memory in C#.NET.**

The .NET Framework's garbage collector manages the allocation and release of memory for your application.

**4)How u call destructor and dispose methode in c#.NET**

for destructor :

class Car  
{  
~Car() // destructor  
{  
// cleanup statements...  
}  
}

For Dispose:

DisposableResource TestObj = new DisposableResource(fs);

TestObj.Dispose();

**5)Where Destructors Can not be defined?**

in structs

**6)What are the different generaions of Garbage Collection and how do they work ?**

Generations in the Garbage Collector is a way of enhancing the garbage collection performance**.**There are 3 Generations...0,1,2.

Generation 0 - When an object is initialized, its in generation 0. These are new objects that have never been played around with by the GC. As and when more objects get created, the process of Garbage Collection is invoked by the CLR. Generation 1 - The objects that survive the garbage collection process are considered to be in generation 1. These are the old objects.Generation 2 - As more new objects get created and added to the memory, the new objects are added to generation 0, the generation 1 old objects become older, and so are considered to be in generation 2. Generation 2 is the highest level generation in the garbage collection process. Any further garbage collection process occuring causes the level 1 objects promoted to level 2, and the level 2 objects stay in level 2 itself, as this generation level is the highest level.

**7)Does c# supports destructors?**

yes .The destructor implicitly calls Finalize on the base class of the object.