Garbage Collection and Dispose Pattern in .NET - Questions and Answers

1. ## What is a Garbage Collector?

   Answer:
   GC in CLR is an automatic memory manager.

2. ## When Garbage Collector starts working in .Net?

   Answer:
   When there is not enough memory space for running application

3. ## What happens when you call a Collect() method with “Forced” parameter?

   Answer:
   It forces a collection to occur **immediately** for all generations up to and including the specified generation.

4. ## What are strong references and weak references in GC?

   Answer:
   Strong reference - an object in use by an application while the application’s code can reach that object.
   A weak reference permits the garbage collector to collect the object while still allowing the application to access the object.
   Thus, a weak reference can work with an object like a strong reference, but if necessary, the object will be deleted, even if there is a weak reference to it. When you use a weak reference, the application can still obtain a strong reference to the object, which prevents it from being collected.

5. ## What data types does the garbage collector clean?

   Answer:
   GC cleans reference types in the heap. It makes no sense for the garbage collector to consider collecting stack memory because the stack is not managed that way: Everything on the stack is considered to be “in use.” And memory used by the stack is automatically reclaimed when you return from method calls.

6. ## Is there any way by which we can pin object for later reference i.e can we tell GC to not to delete or move that object and will re-reference it at same location later may be by non-managed code?

   Answer:
   This is related with the concept of “Pinned Objects” and can be done using “fixed” statement.

   The CLR allows us to "pin" an object so that it is not moved during garbage collection.

7. ## Explain how garbage collection deals with circular references.

   Answer:
   Usually in a parent-child relationship, situations occur where a child interacts with the parent object and has a reference held to the parent object.

The .NET, the objects that are reachable from the root can be cleaned up easily. Thus, this can even be applied to circular reference and have the objects holding the resources cleaned up.

8. What are Destructors?

Answer:

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.

9. What are the different generaions of Garbage Collection and how do they work ?

Answer:

Generations in the Garbage Collector is a way of enhancing the garbage collection performance**.**
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.

IDisposable interface

10. What is the use of using statement in C#?

Answer:

Using statement used for calling dispose method of Idisposable interface properly by using “try-finally” pattern.

Example:

```
using (FileStream fs = new FileStream("Temp.dat", FileMode.Create)) {
fs.Write(bytesToWrite, 0, bytesToWrite.Length);
 }
```

Is equal to:

```
FileStream fs = new FileStream("Temp.dat", FileMode.Create);
 try {
    fs.Write(bytesToWrite, 0, bytesToWrite.Length);    }
 finally {
    if (fs != null)
      fs.Dispose();
 }
```

11. Code working result? Why? How to fix?

```
Byte[]bytesToWrite = new Byte[] { 1, 2, 3, 4, 5 };

FileStream fs = new FileStream("Temp.dat", FileMode.Create);

File.Delete("Temp.dat");
```

Answer:
System.IO.IOException**:** 'The process cannot access the file 'c:\users\..\Debug\Temp.dat' because it is being used by another process.'

Fix:

```
Byte[] bytesToWrite = new Byte[] { 1, 2, 3, 4, 5 };
FileStream fs = new FileStream("Temp.dat", FileMode.Create);
try {
   fs.Write(bytesToWrite, 0, bytesToWrite.Length);    }
finally {
   if (fs != null)
     fs.Dispose();
}
File.Delete("Temp.dat");
```

12. After calling Dispose() method for some resource, is it immediately removed from memory?

Answer:
No, for such resources there is separate queue. They go through one more GC run and then leave.

13. Why it is not recommended to use Finalize methods?

Answer:
CLR doesn't make any guarantees as to the order in which Finalize methods are called. So, you should avoid writing a Finalize method that accesses other objects whose type defines a Finalize method; those other objects could have been finalized already.

14. What is the difference between Dispose and Finalize methods?

Answer:

Dispose	Finalize
It is used to free unmanaged resources at any time.	It can be used to free unmanaged resources held by an object before that object is destroyed.

It is called by user code and the class which is implementing dispose method, must has to implement IDisposable interface.	It is called by Garbage Collector and cannot be called by user code.
It is implemented by implementing IDisposable interface Dispose() method.	It is implemented with the help of Destructors
There is no performance costs associated with Dispose method.	There is performance costs associated with Finalize method since it doesn't clean the memory immediately and called by GC automatically.