Explain how memory management works in .NET with reference to garbage collection.

Garbage Collection is a process of collecting objects that are no longer used by the Application. **GC** serves as an automatic memory manager, where they manage the *allocation* and *release* the objects from the memory.

Memory Allocation

Runtime reserves a contiguous region of memory space for new Process.

This region is called the **managed heap.**

When the Application creates the first object (First object in the process), the runtime allocates space region for it and point to it by base address (point to the first space in managed heap), and when the second object is created the runtime allocate to it the next (followed) space region. And so on for each new object.

Memory Release

GC releases the memory from objects that are no longer used and can determine these objects by examining the Application's roots that contains all objects (static fields, local variables). The objects that cannot be reached from the root, called **Unreachable-Objects** and **GC** consider them as garbage and will release memory from them.

GC then looking for memory space allocated by Unreachable-Objects and copying other reachable objects to these spaces to compact the reachable objects in memory, freeing up the blocks of spaces allocated to unreachable objects.

The compaction does not happen, only if a collection discovers a significant number of unreachable objects.

Generations (GC algorithm)

GC divides the objects into three categories (generations) based on their lifetime

- **Generation 0** contains new objects; temporary variable are **short-lived** objects. Garbage collection occurs most frequently in this generation.
- **Generation 1** contains **short-lived** objects and serves as a buffer between short-lived objects and long-lived objects.
- **Generation 2** contains **long-lived** objects such as static variables.

Collection happened first in **Gen-0** and the objects serve from it are promoted to **Gen-1**, then in **Gen-1** and the objects serve from it are promoted to **Gen-2**, finally in **Gen-2** and the objects serve from it are still stay in **Gen-2.**

What is the purpose of async and await in C#? Provide an example scenario where asynchronous programming is beneficial.

async and **await** are used to write asynchronous code more easily and clearly, enabling operations to run without blocking the main thread**.**

async marks a method as asynchronous, allowing it to run in the background without blocking the calling thread.

await pause the execution of the method until the awaited asynchronous task completes, allowing other work to be done in the meantime, where the return of method is essential to use in the method.

Example Scenario

When retrieving data from a Database and then process it. A synchronous approach would block the server's thread while waiting for the database's response, making the application unresponsive under heavy loads. Using asynchronous programming can free up server resources, as the thread can handle other requests while waiting for data.

Describe dependency injections in .NET and how it enhances application design.

Dependency Injection (DI) is a design pattern used in .NET (and other frameworks) to achieve **Inversion of Control (IoC)**. In this approach, objects or classes don't create their dependencies directly; instead, they are provided with those dependencies from an external source, enhancing flexibility, testability, and maintainability. In .NET, Dependency Injection is typically implemented through a built-in **IoC container** that manages the creation, lifespan, and injection of services and objects. You define the dependencies in a central place (Program.cs) and specify how objects are created and which dependencies they require. This IoC container can then resolve and inject dependencies where needed.

Dependency Injection Benefits

- Decoupling
- Ease of Testing
- Flexibility and Configurability
- Single Responsibility Principle (SRP)
- Automatic Management of Object Lifetimes