✅ .NET Core Notes

🔹 1. What is .NET?

- .NET is a **developer platform** by Microsoft for building:
  - Web apps, desktop apps, mobile apps, cloud services, and more.
- Supports **multiple languages** (C#, F#, VB.NET)
- Offers a **common runtime (CLR)** and a **base class library (BCL)**.

🔹 2. What is .NET Framework?

- .NET Framework is the **original Windows-only** version of .NET.
- Includes:
  - **CLR (Common Language Runtime)** for code execution
  - **BCL (Base Class Libraries)** like System.IO, System.Net
- Windows apps, ASP.NET Web Forms, ADO.NET, Windows Forms use this.

🔹 3. IL / MSIL / CIL

Term	Full Form	Meaning
IL	Intermediate Language	Code generated from .NET languages like C#, VB.NET
MSIL	Microsoft Intermediate Language	Microsoft's name for IL
CIL	Common Intermediate Language	Official ECMA-standard term

IL is a **CPU-independent** instruction set. CLR later converts it to machine code.

🔹 4. JIT (Just-In-Time Compiler)

- **Part of CLR** that converts IL → native machine code **at runtime**.
- Types:

- **Pre-JIT**: Compiles entire code at install time
- **Normal JIT**: Compiles methods as they are called
- **Econo-JIT**: For resource-limited environments

🔹 5. CLR (Common Language Runtime)

- Core runtime of .NET: **executes IL**, manages:
  - JIT compilation
  - Garbage Collection (GC)
  - Memory management
  - Security
  - Exception handling
  - Threading
  - AppDomains

🔹 6. Garbage Collection (GC)

- Automatically **frees memory** of objects no longer used.
- Divides memory into **Generations**:
  - Gen 0: Short-lived
  - Gen 1: Medium-lived
  - Gen 2: Long-lived
- GC.Collect() can force GC (not recommended normally)

🔹 7. Memory Management

Type	Stored In	GC Handled?
Value Types	Stack	No
Ref Types	Heap	Yes

- Memory is **automatically allocated and freed** in managed heap.

- Uses **Finalizers** and **IDisposable** to clean resources (e.g. files, DB)
- **Finalizer vs IDisposable**
- Finalizer (~ClassName()) runs **when GC collects the object** (but **timing is uncertain**)
- IDisposable is for **manual release of resources** (like file streams, DB connections)

🔹 8. AppDomain (Application Domain)

- Lightweight process within CLR
- Provides **isolation between applications**
- Assemblies can be loaded/unloaded in AppDomains
- Safer than threads: a crash in one AppDomain won't affect others

---

🔹 9. Security in .NET

.NET provides:

- **Code Access Security (CAS)** – restricts what code can access (e.g. file, DB)
- **Role-Based Security** – restricts based on user roles
- **Security policies** at 3 levels:
  - Enterprise
  - Machine
  - User

🔹 10. CLS (Common Language Specification)

- Set of rules all .NET languages must follow
- Ensures **interoperability between languages** (e.g. C#, VB.NET)
- Example: uint is not CLS-compliant, because VB.NET doesn't support it

---

🔹 11. CTS (Common Type System)

- Defines **all data types** in CLR so that all .NET languages are compatible.
- Example:

C# VB.NET CLR Type

int Integer Int32

- Value Types (int, struct) go in stack
- Reference Types (class, object) go in heap

🔹 12. Assembly

- Compiled **output** of a .NET application (.dll or .exe)
- Contains:
  - IL code
  - Metadata
  - Manifest (assembly info)
  - Optional resources
- Types of assemblies:
  - **Private** – for one app (local)
  - **Shared** – installed in GAC (global)
  - **Satellite** – for localization (language-specific)

Managed Code – Notes

- **Managed Code** is code that runs **under the supervision of the .NET CLR (Common Language Runtime)**.
- It is written in **.NET-supported languages** like C#, VB.NET, F#.
- CLR provides services like:
  - Garbage Collection (GC)
  - Memory management

    - Security
    - Exception handling
    - Type safety
- **Produces IL (Intermediate Language)** code during compilation.
- Developers **do not need to manually manage memory or handle low-level details**.

🔸 **Examples:**

csharp

```
int x = 5;        // C# managed code
Console.WriteLine(x);
```

✅ **Unmanaged Code – Notes**

- **Unmanaged Code** is executed **directly by the OS** (no CLR involvement).
- Written in low-level languages like **C, C++, or Assembly**.
- Developer is responsible for:
    - Memory allocation and deallocation
    - Handling pointers, security, and type safety manually
- **Not safe or portable** like managed code.
- Still usable in .NET via **Interop/Platform Invocation (P/Invoke)**.

🔸 **Examples:**

```
printf("Hello, unmanaged world"); // C/C++ unmanaged code
```

🔄 **Difference Between Managed and Unmanaged Code**

Feature	Managed Code	Unmanaged Code
Executed By	CLR (Common Language Runtime)	Operating System
Memory Management	Automatic (by GC)	Manual (by programmer)

Feature	Managed Code	Unmanaged Code
Languages	C#, VB.NET, F#	C, C++, Assembly
Output	IL code (Intermediate Language)	Native Machine Code
Portability	Cross-platform with .NET Core	OS-specific
Type Safety	Enforced by CLR	Programmer responsibility
Security	CLR Enforced	Limited, manually handled
Tools	Runs in Visual Studio / .NET Runtime	Needs compiler like GCC or MSVC

✅ **ILDASM (Intermediate Language Disassembler) – Notes**

- **ILDASM** is a **.NET SDK tool** used to inspect compiled .NET assemblies (.exe or .dll).
- It shows:
  - IL (Intermediate Language) code
  - Assembly metadata
  - Manifest information
  - Class structures and methods
  - References, attributes, security details
- Useful for:
  - Debugging
  - Understanding IL structure
  - Learning how C# code is converted to IL
  - Verifying compilation output

🔹 **How to Use ILDASM:**

1. Open **Developer Command Prompt for Visual Studio**.
2. Type: ildasm → Press Enter.

3. In the GUI window:
    - Go to File > Open
    - Choose a .dll or .exe file
    - Expand namespaces to view IL of classes and methods