**Assemblies:**

An assembly refers to a compiled unit of code that contains executable code, metadata, and resources needed to run a program and they play a crucial role in the Common Language Runtime (CLR), which is the runtime environment provided by the .NET framework.

Executable code: It is a intermediate language which have a low level language that clr can understand.

Metadata: It contains information about the types, members, and references used in the code. It includes details such as class names, method signatures,

Resources: lt also include resources such as images, strings, and other data.

The compilation process generates an assembly file (usually with a .dll or .exe extension). At runtime, the CLR loads the assembly and executes the code. The CLR takes care of tasks like memory management, security checks, and other runtime services.

**Dynamic link library:**

DLL files contain compiled code and data that multiple programs can use simultaneously. They allow code to be modular and shared among different applications, reducing redundancy and improving efficiency.

Dll loaded up only when a specific program asks for them

Instead of every time it starts they prevent ram for filling up unnecessary codes.

dll cannot invoke by himself so he needs a hoster.

Dll runs inside other memory address or address space of exe .

Functionality: DLL files often contain functions and procedures that programs can call during execution. These functions are shared among different programs, allowing them to access the same piece of code without having to duplicate it in each program's executable file.

**Exe(Executable):**

When you run an EXE file, it initiates the execution of a program. The code and data within the EXE file are loaded into memory, and the program begins to execute. It contains only necessary resources and code to run independently. Exe runs on its own memory address and execute.

**Difference between exe and Dll**

|  |  |  |
| --- | --- | --- |
| **NO** | **DLL** | **EXE** |
| 1 | Can not run Individually | Runs Individually |
| 2 | Used as supportive file for other Application | Itself an Application |
| 3 | Does not contain an entry point (no main function) so can not run individually | Contains an entry point (Main function) so can run individually |
| 4 | A Program /Application with out main creates a DLL after compilation. | A Program /Application With main creates an EXE after compilation. |
| 5 | OS does not create a separate process for any DLL rather DLL will run in the same process created for an EXE | OS Creates a separate process for each EXE it executes. |

**Single-file Assemblies**:

Description: This type of assembly includes all the necessary code and dependencies in a single executable file.This can simplify deployment and distribution of the application.

**Multi-file Assemblies:**

In multi-file assemblies, the code is split across multiple files, and each file is a separate module.

Example: When you compile a large project into multiple DLLs (Dynamic Link Libraries), each DLL is a separate module that contributes to the overall multi-file assembly.

**Private Assemblies:**

Private assemblies are intended for use by a single application. They are typically stored in the application's directory or a subdirectory.

Example: When you create a Windows Forms application, the associated DLLs that contain the application's code and dependencies are often private assemblies.

**Shared Assemblies:**

Shared assemblies are intended to be used by multiple applications. They are stored in the Global Assembly Cache (GAC) and have a **strong name** to ensure uniqueness.

Example: The .NET framework itself is a collection of shared assemblies stored in the GAC. Libraries like Entity Framework or log4net are also examples of shared assemblies.

**Satellite Assemblies:**

They are used to store resources, such as strings, images, or other localized content, for different languages or regions. This allows developers to create applications that can be easily adapted to different languages and cultures without modifying the main executable.

Example: Main Application Assembly: MyApp.exe

Satellite Assemblies:

MyApp.resources.dll (neutral resources)

MyApp.resources.fr.dll (French resources)

MyApp.resources.es.dll (Spanish resources)

In this scenario, if a user with the French culture preference runs the application, the runtime will load the MyApp.resources.fr.dll satellite assembly, providing a French-language experience.

**Dynamic Assemblies:**

Dynamic assemblies are generated at runtime, often using reflection or the System.Reflection.Emit namespace.

Example: When you use reflection to create and execute code dynamically based on user input or other runtime conditions, you are working with dynamic assemblies.

**Note:**

Managing dependencies can be complicated when using shared assemblies because different applications may require different versions of a library. This is where versioning and strong-naming came

Versioning is the process of assigning a unique version number to each assembly. This allows different versions of the same library to coexist on the same machine. When an application requests a specific version of a library, the CLR will load the correct version based on the version number specified in the application’s configuration file.

Strong-naming is another technique used to manage dependencies. When an assembly is strong-named, it is given a unique cryptographic signature that ensures its secured. This allows different versions of the same library to be used by different applications without conflict. Strong-naming also provides a way to verify that an assembly has not been tampered with or modified since it was signed.

Working with Strongly Named Assemblies

Strong-naming is essential for working with shared assemblies because it helps to prevent DLL Hell by ensuring that each application is using the correct version of the assembly.

**Adding an assembly to GAC:**

1. Create an assembly
2. Create a key pair

Strong named assembly can be saved into GAC.

In visual studio developer cmd :

             sn.exe -k C:\MyStrongKeys.sk

Specify the below line in the assembly properties/assemblyinfo.cs ( at the end)

             [assembly: AssemblyKeyFile("C:\\MyStrongKeys.sk")]

1. Go to the assembly created .dll file
2. C:\Users\himaja\_geetha\Documents\github\dotnet\_training\dotnet\_training\Codes\arthimeticcalci\arthimeticcalci\bin\Debug>gacutil.exe -i arthmeticcalci.dll
3. Adding to GAC

gacutil.exe -i arthimeticcalci.dll

1. Check in C:\Windows\Microsoft.NET\assembly\GAC\_MSIL if arthimaticcalci is added or not.
2. To uninstall the GAC added assembly

gacutil.exe -u arthimeticcalci.dll

1. Using The Gac assembly in program

arthimeticcalci.Class1 cl=new arthimeticcalci.Class1();