C# Programming Language Fundamentals

Prerequisites

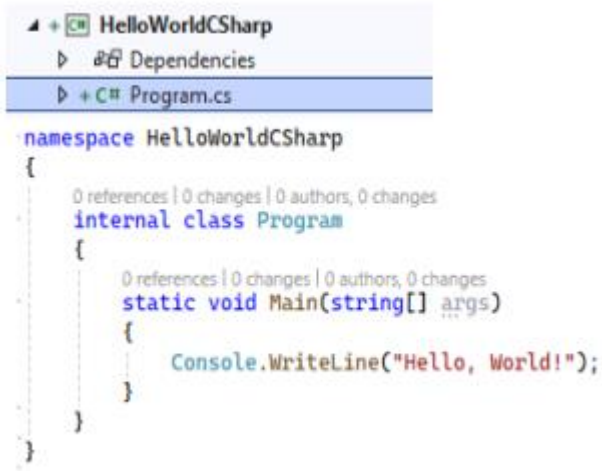
- Computer and its basic knowledge.
- Integrated Development Environment or Code Editor
  - Visual Studio (Recommended)
  - Visual Studio Code

Overview

- Introduction
- .NET Overview?
- Our First C# program
- IDE – Visual Studio quick tour
- C# syntax
- Variables and Data Types
- Type Conversions
- Conditionals
- Loops
- Classes
- Object Oriented Programming
- Value Types vs Reference Types
- Compilation and CLR
- Assemblies and Referencing
- Exceptional Handling

C# Source Code Execution

C# Source Code

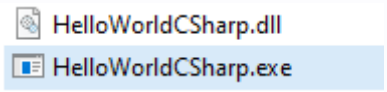


Console Application

C# Compiler (Roslyn)

Compiler is a C# program which converts C# source code into Microsoft Intermediate Language (MSIL) code as a .dll or .exe

MSIL Code (.dll or .exe)

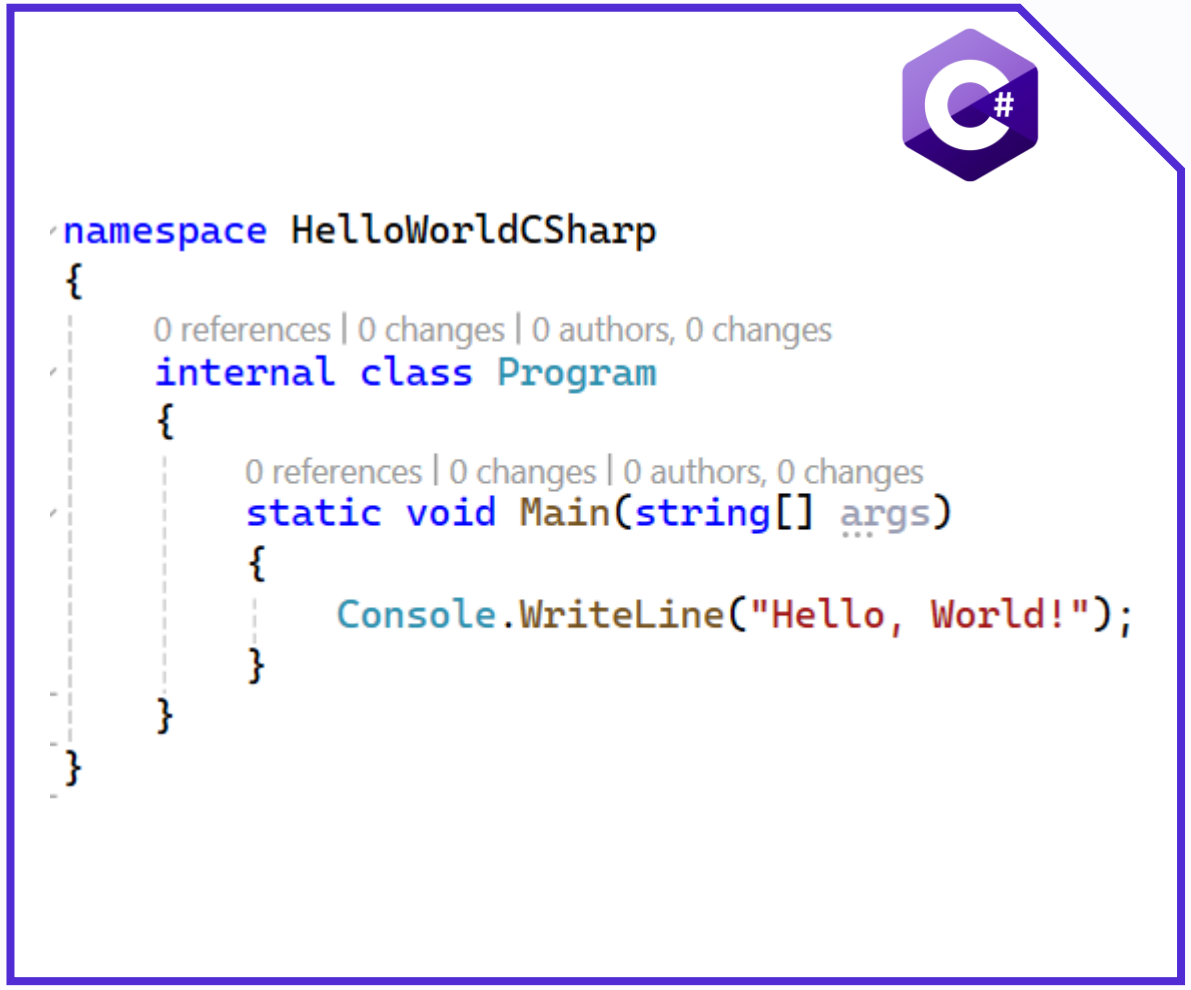


Common Language Runtime is a program which converts MSIL code into machine understandable native code. (010101010)

CLR Just In Time Compiler

|---------- Compilation Process ----------| |---------- Runtime Process ----------|

Our First C# Program



```
namespace HelloWorldCSharp
{
    0 references | 0 changes | 0 authors, 0 changes
    internal class Program
    {
        0 references | 0 changes | 0 authors, 0 changes
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

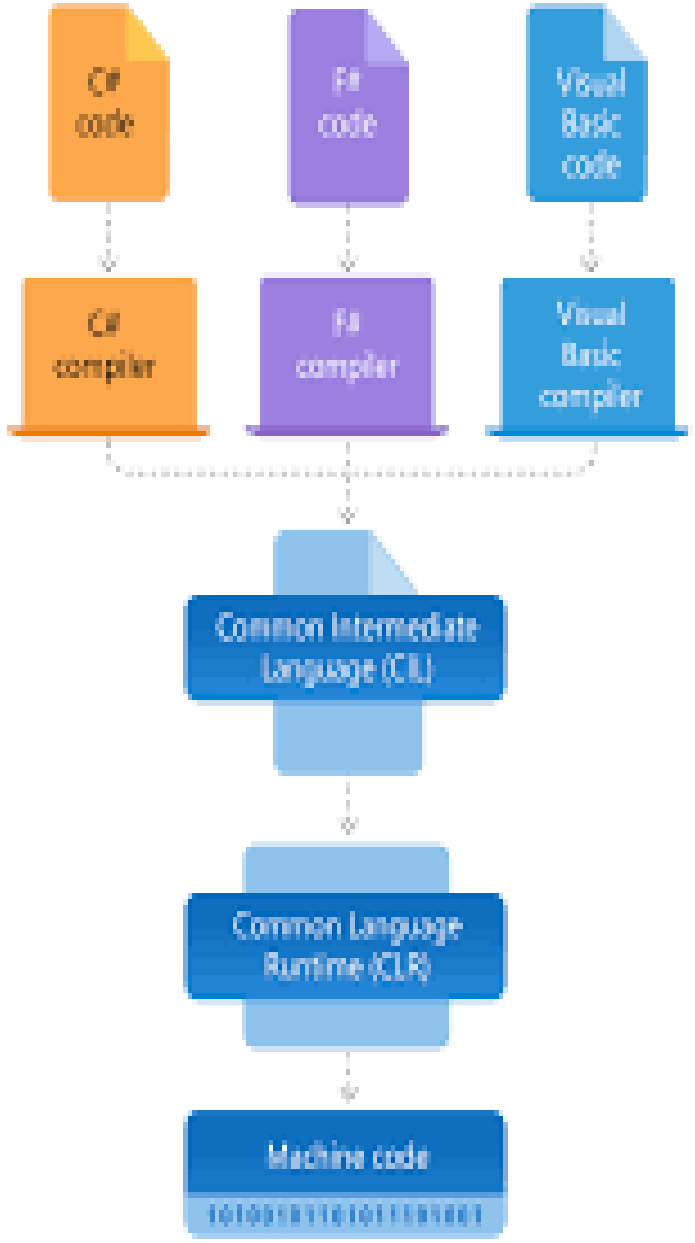
Architecture

Two major components of .NET ecosystem are

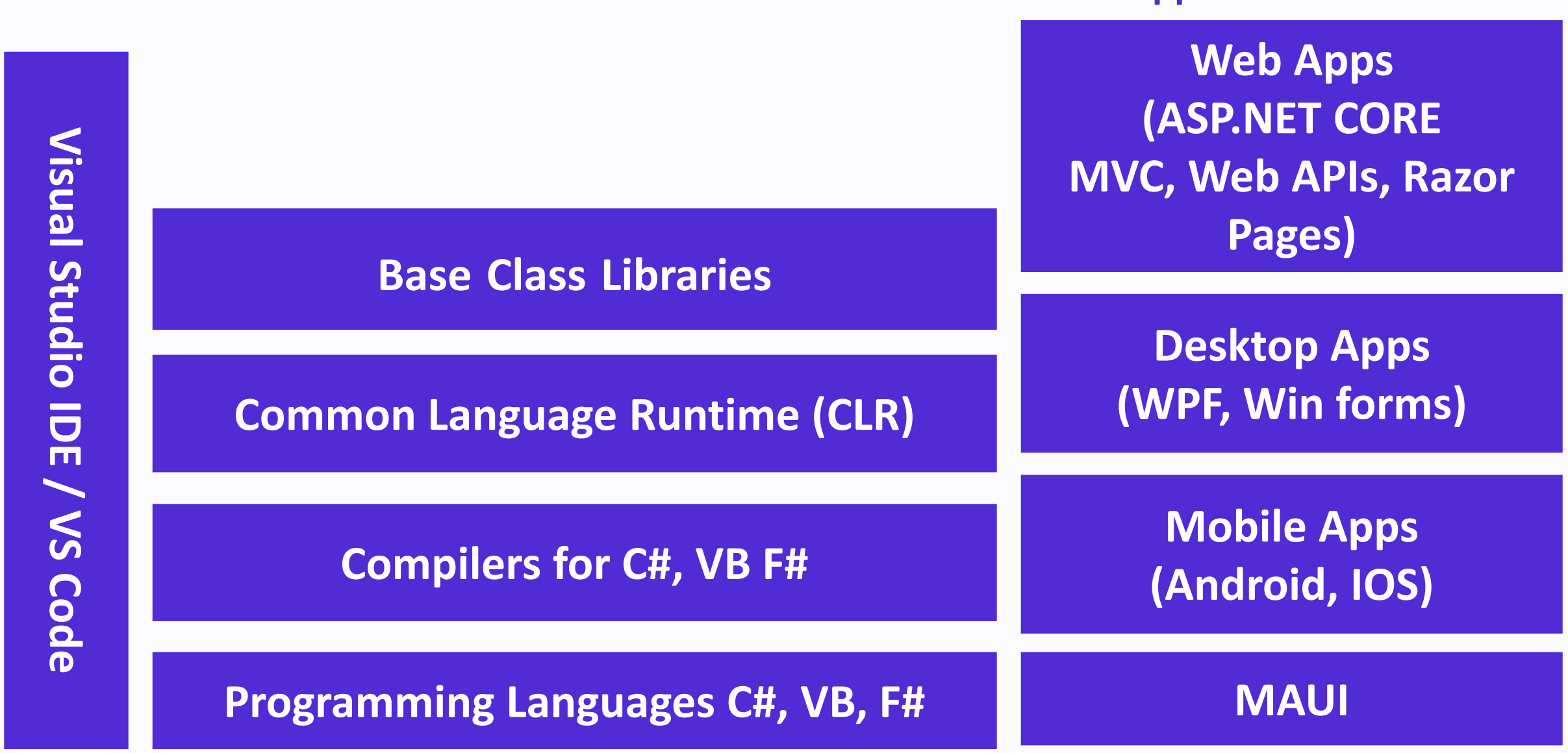
- The **Common Language Runtime (CLR)** is the execution engine that handles running applications. It provides services like thread management, garbage collection, type-safety, exception handling, and more.
- The **Class Library** provides a set of APIs and types for common functionality. It provides types for strings, dates, numbers, etc. The Class Library includes APIs for reading and writing files, connecting to databases, drawing, and more.

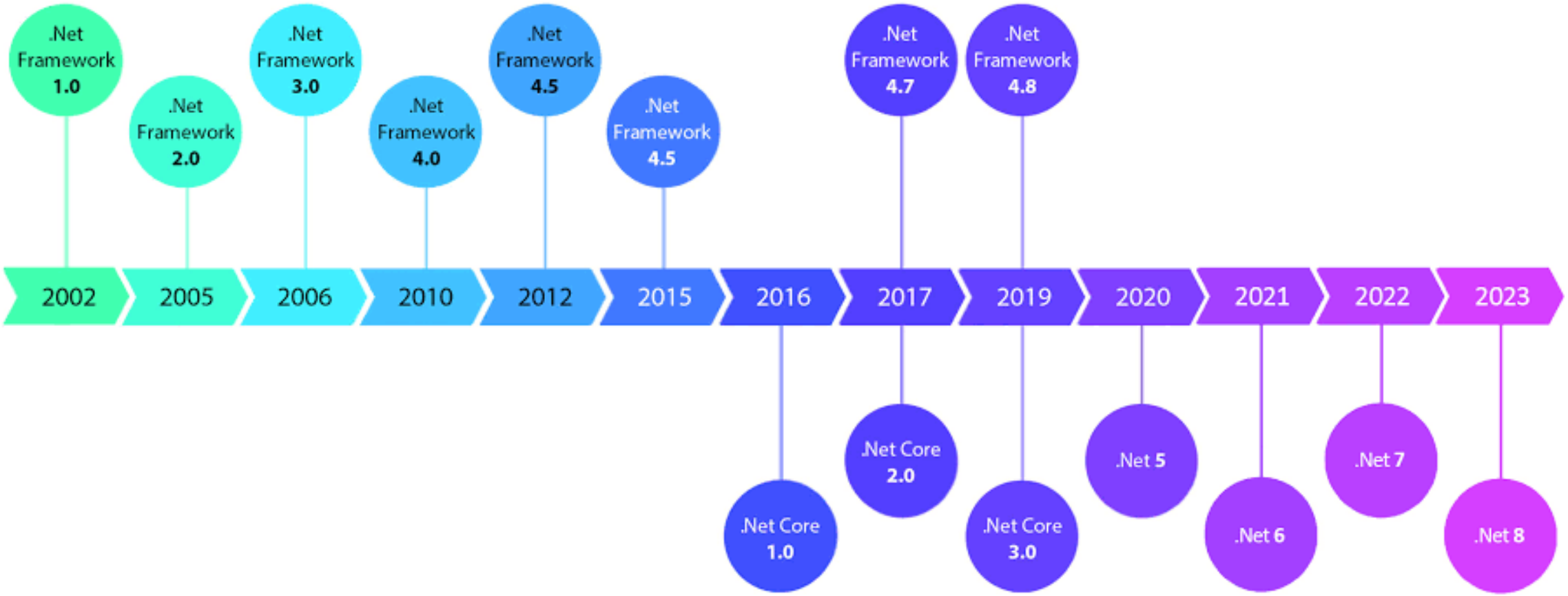
.NET applications are written in the C#, F#, or Visual Basic programming language. Code is compiled into a language-agnostic Common Intermediate Language (CIL). Compiled code is stored in assemblies—files with a .dll or .exe file extension.

When an app runs, the CLR takes the assembly and uses a just-in-time compiler (JIT) to turn it into machine code that can execute on the specific architecture of the computer it is running on.



.NET Ecosystem and its tools




.Net Framework 1.0
.Net Framework 2.0
.Net Framework 3.0
.Net Framework 4.0
.Net Framework 4.5
.Net Framework 4.5
.Net Framework 4.7
.Net Framework 4.8
2002
2005
2006
2010
2012
2015
2016
2017
2019
2020
2021
2022
2023
.Net Core 1.0
.Net Core 2.0
.Net Core 3.0
.Net 5
.Net 6
.Net 7
.Net 8

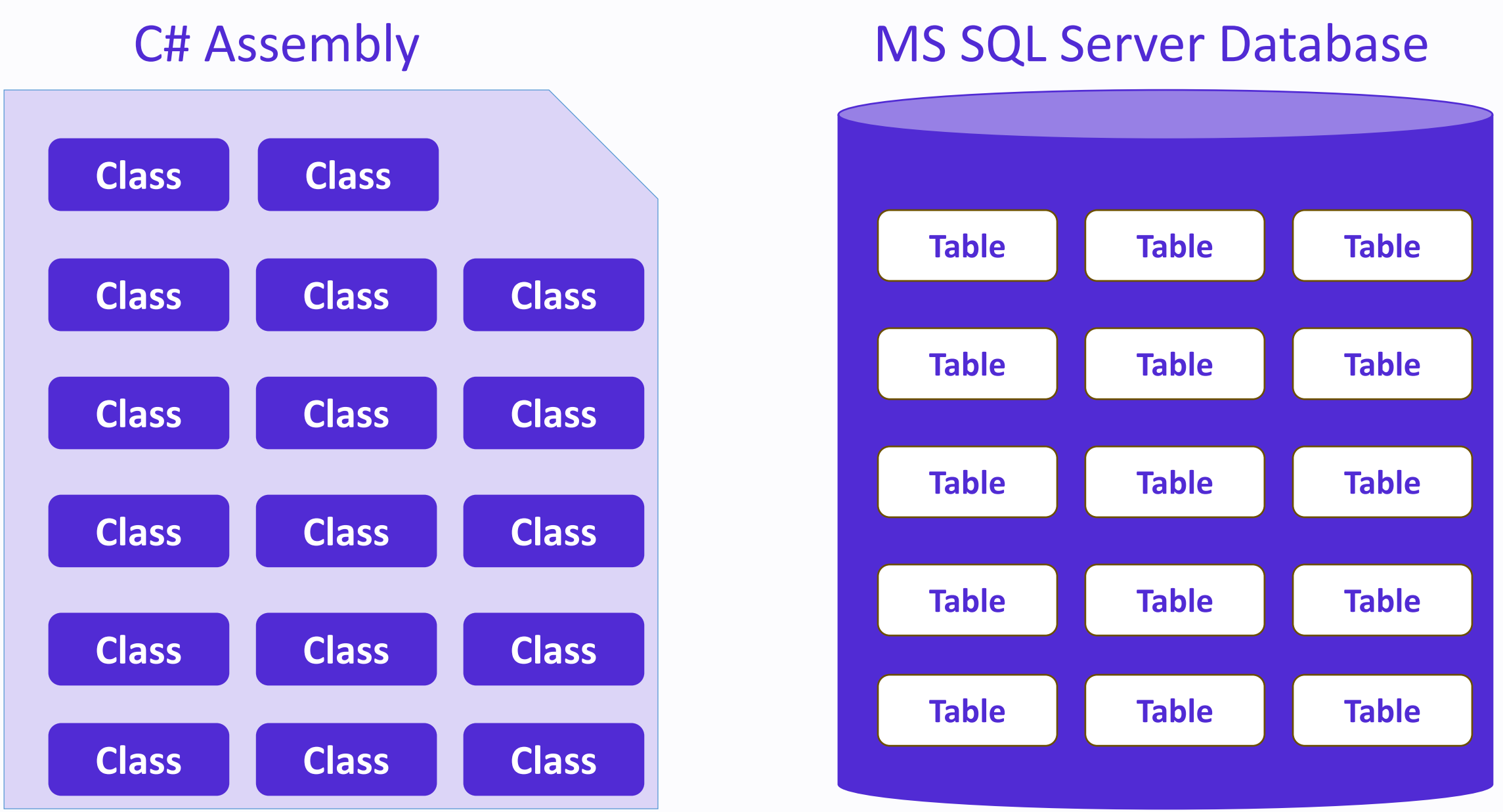
C#

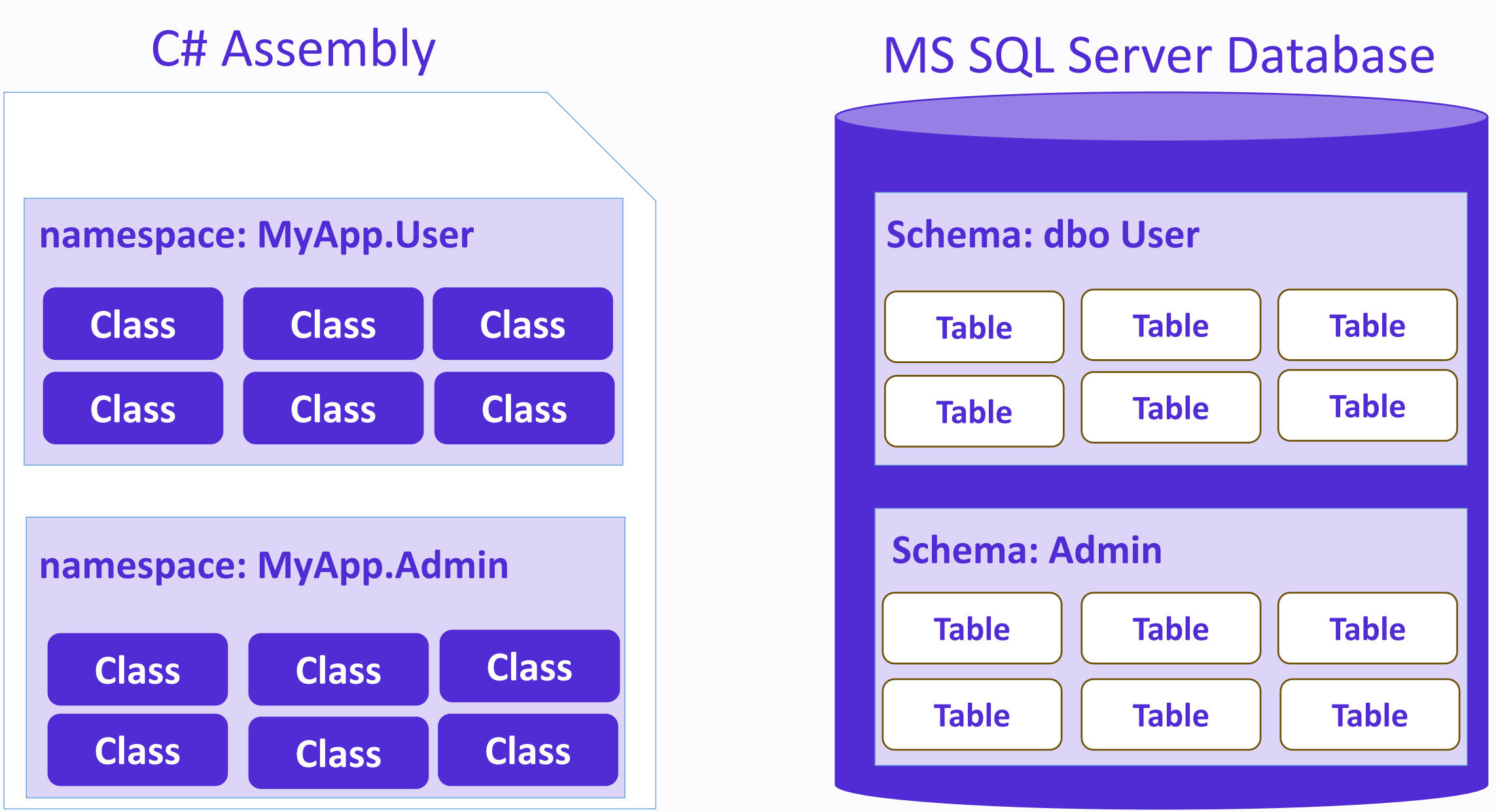
- Open Source
- Cross Platform
- Strongly Typed
- Case Sensitive
- Memory Management
- Object Oriented

- Usage
  - Desktop Applications
  - Web Applications
  - Mobile Applications
  - Cloud Applications
  - Web APIs
  - SSIS Packages
  - Many More…

Version History

Target	Version	C# language version default
.NET	8.x	C# 12
.NET	7.x	C# 11
.NET	6.x	C# 10
.NET	5.x	C# 9.0
.NET Core	3.x	C# 8.0
.NET Core	2.x	C# 7.3
.NET Standard	2.1	C# 8.0
.NET Standard	2	C# 7.3
.NET Standard	1.x	C# 7.3
.NET Framework	all	C# 7.3


C# Assembly
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
Class
MS SQL Server Database
Table
Table
Table
Table
Table
Table
Table
Table
Table
Table
Table
Table
Table
Table
Table

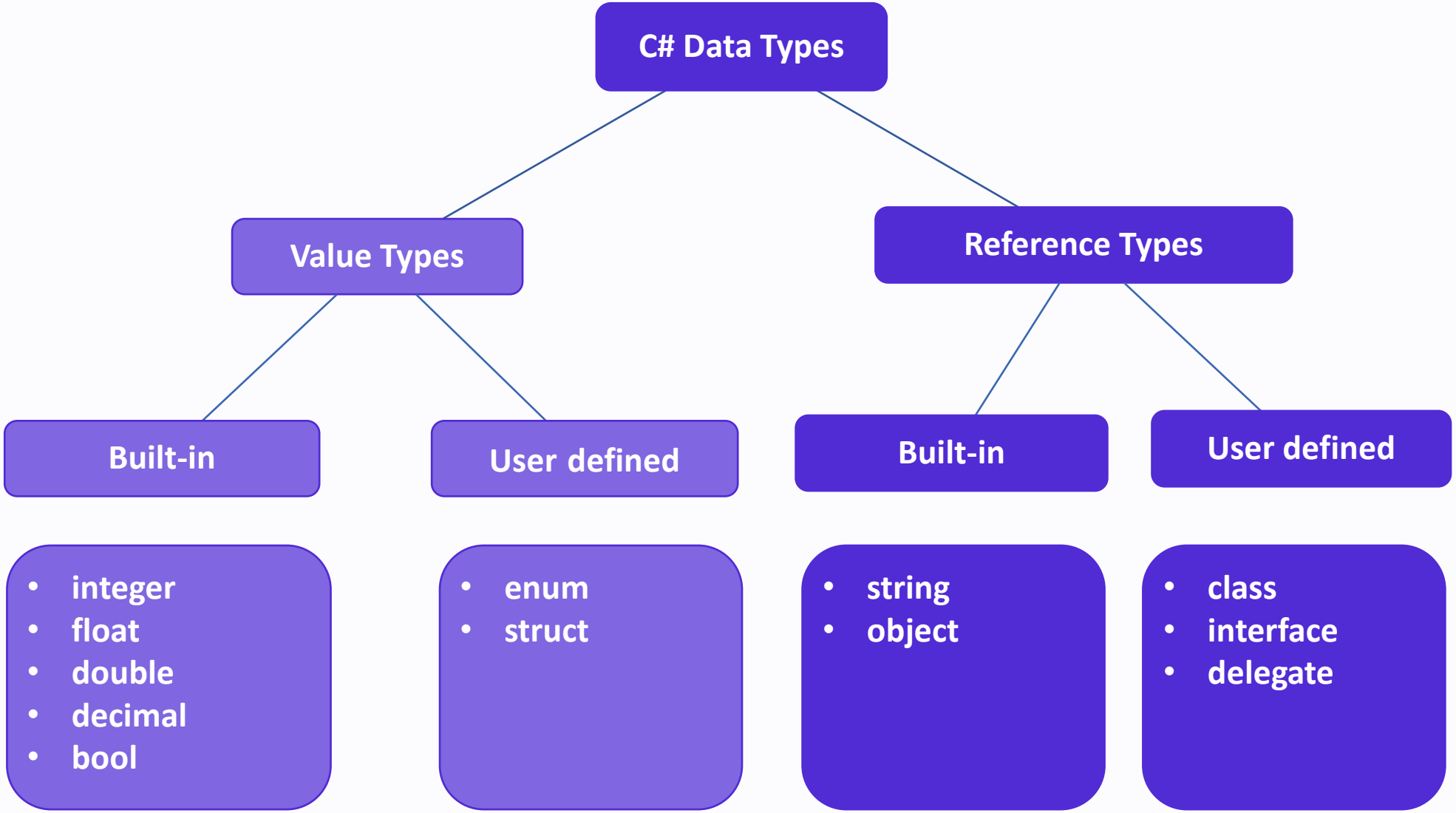

C# Assembly
namespace: MyApp.User
Class
Class
Class
Class
Class
Class
namespace: MyApp.Admin
Class
Class
Class
Class
Class
Class
MS SQL Server Database
Schema: dbo User
Table
Table
Table
Table
Table
Table
Schema: Admin
Table
Table
Table
Table
Table
Table

Keywords

abstract	delegate	if	override	struct	volatile
as	do	implicit	params	switch	while
base	double	in	private	this	
bool	else	int	protected	throw	
break	enum	interface	public	true	
byte	event	internal	readonly	try	
case	explicit	is	ref	typeof	
catch	extern	lock	return	uint	
char	false	long	sbyte	ulong	
checked	finally	namespace	sealed	unchecked	
class	fixed	new	short	unsafe	
const	float	null	sizeof	ushort	
continue	for	object	stackalloc	using	
decimal	foreach	operator	static	virtual	
default	goto	out	string	void	

Contextual Keywords

add	group	record
and	init	remove
alias	into	required
ascending	join	scoped
args	let	select
async	managed	set
await	nameof	unmanaged
by	nint	value
descending	not	var
dynamic	notnull	when
equals	nuint	where
File	on	with
from	or	yield
get	orderby	
global	partial	


C# Data Types
Value Types
Reference Types
Built-in
User defined
Built-in
User defined
integer
float
double
decimal
bool
enum
struct
string
object
class
interface
delegate

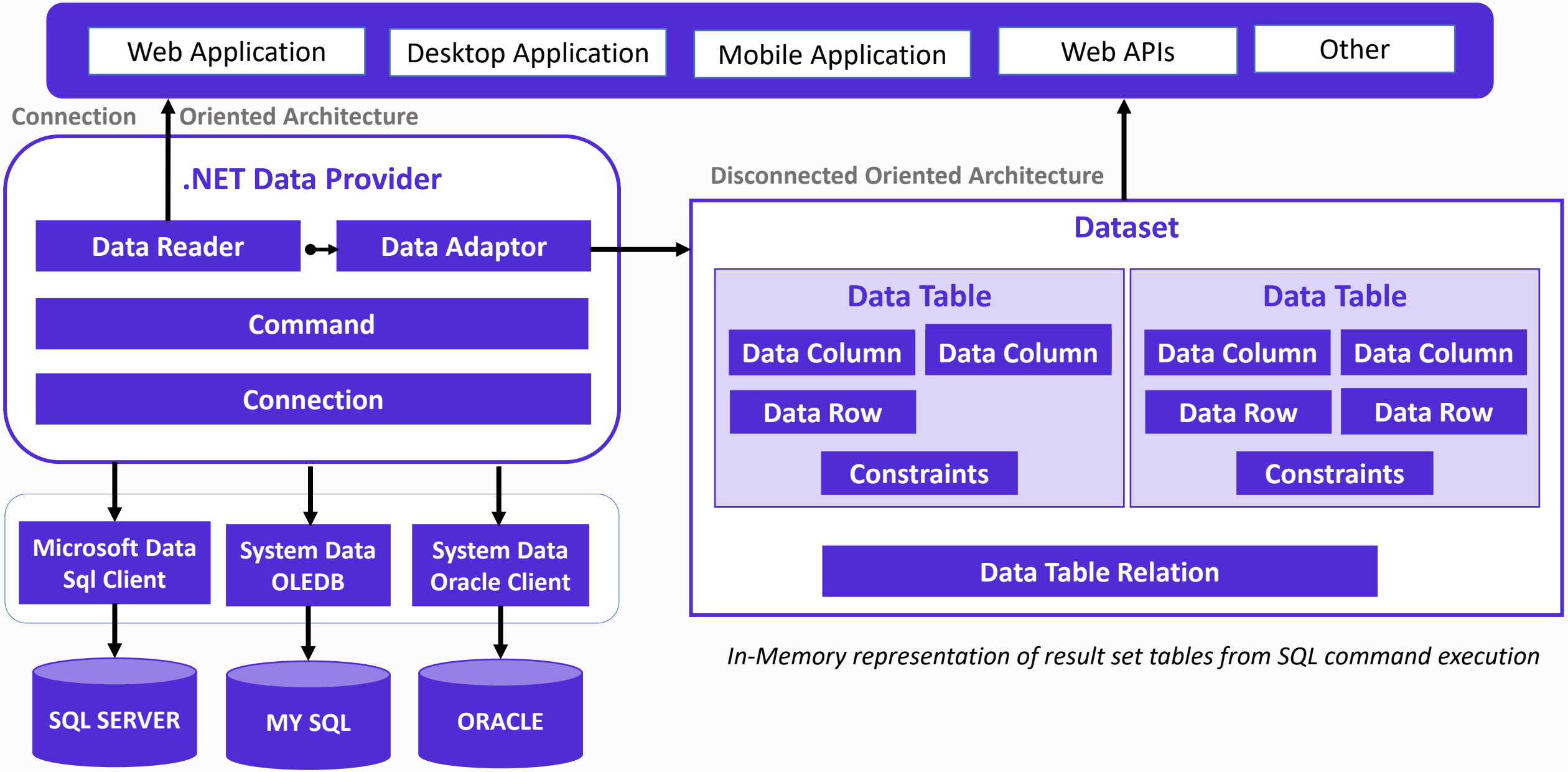
Data Types

Data Type	Size	Description
Integer		
sbyte	8 bits	-2^7 to 2^7-1
int	32 bits	-2^{31} to $2^{31}-1$ (-2,147,483,648 to 2,147,483,647)
short	16 bits	-2^{15} to $2^{15}-1$
long	64 bits	-2^{63} to $2^{63}-1$ (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
Real Or Floating Point		
float	32 bits	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ (7 decimal digits). Suffix: f
double	64 bits	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ (15 decimal digits). Suffix: D
decimal	128 bits	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9228 \times 10^{28}$. Suffix: M
bool	1 bit	true or false
Free Text		
char	2 bytes	'@'
string	2 bytes per character	"Hello". Size = (2* no of characters in the sequence) bytes.

ADO.NET – Data Access API

“ADO.NET is a data access api in .NET Platform to interact with different data sources such as databases (sql server, oracle, etc.), xml, Microsoft access, and other in a standard, and structured approach.”

ADO.NET Architecture



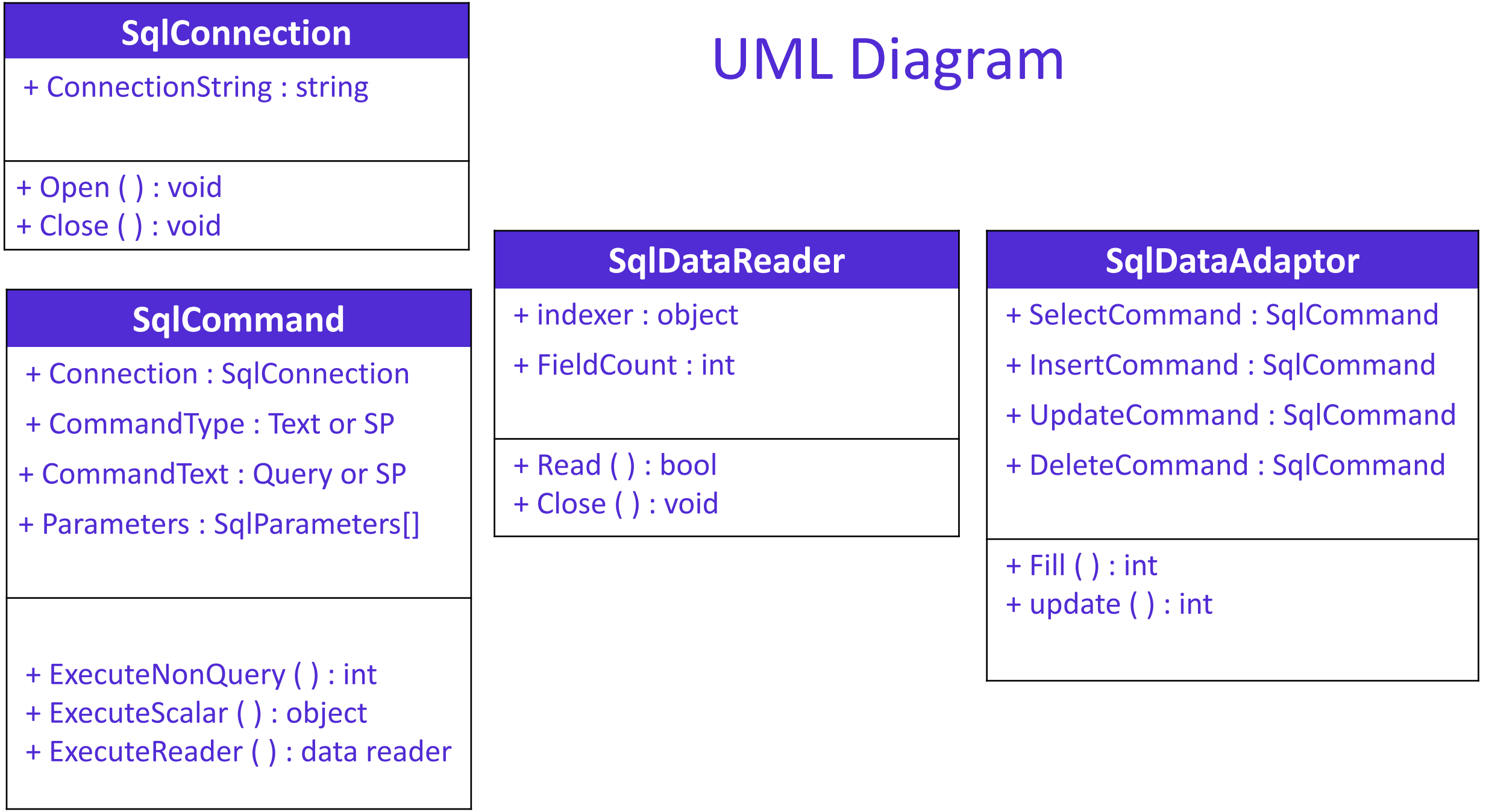
In-Memory representation of result set tables from SQL command execution

MS SQL SERVER Data Access

- Add below DLL as project reference through manage NuGet package manager.
  - **Microsoft.Data.SqlClient**
  - **System.Data.SqlClient (Legacy library)**
- SQL Server data provider provides the following classes to interact with database.

Class	Description
SqlConnection	Establishes a connection to a datsbase.
SqlCommand	Represents an individual SQL statement or stored procedure that can be executed against the database connected.
SqlDataReader	Provides read-only, forward-only access to the data in a database.
SqlDataAdaptor	Acts as a bridge between the command and connection objects and a dataset

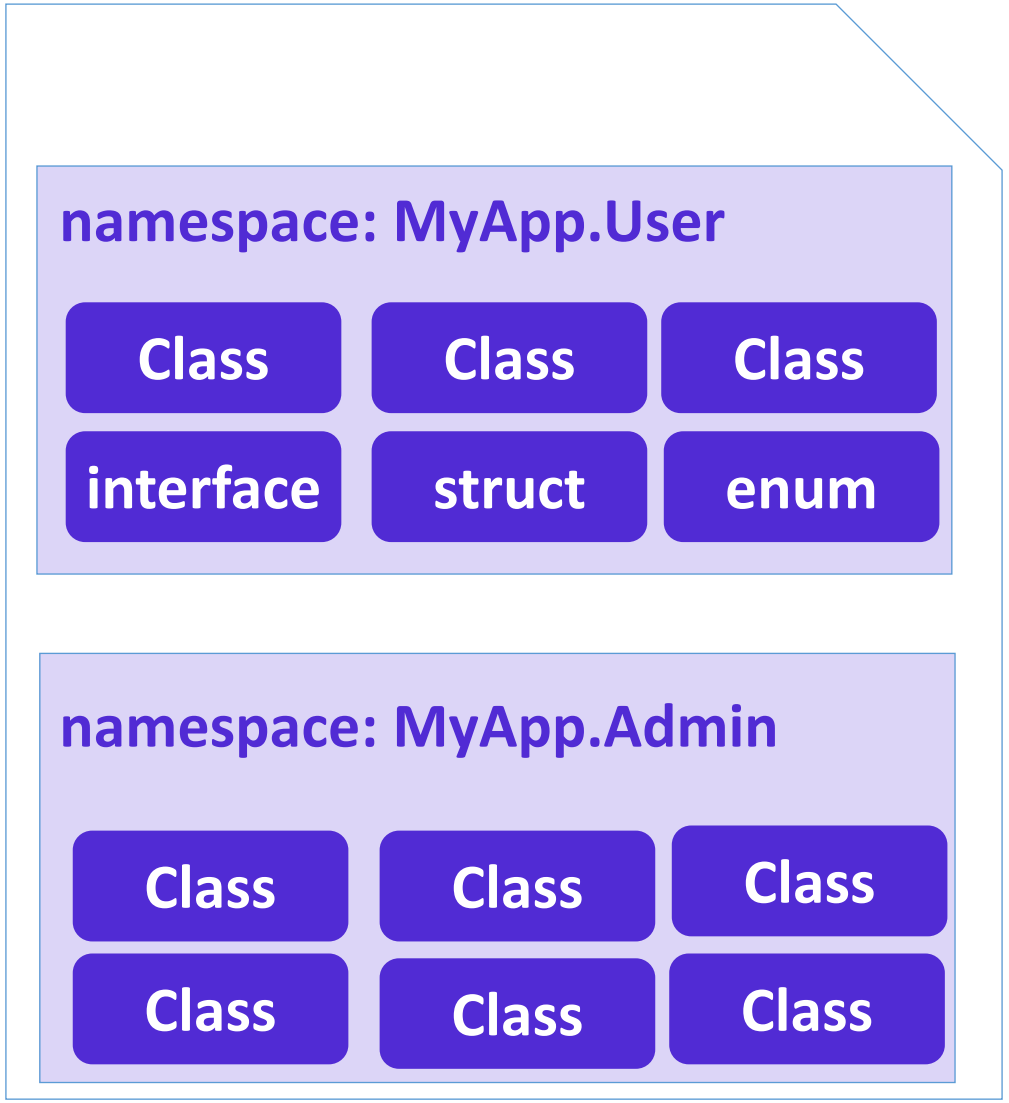
UML Diagram


SqlConnection
+ ConnectionString : string
+ Open ( ) : void
+ Close ( ) : void
SqlCommand
+ Connection : SqlConnection
+ CommandType : Text or SP
+ CommandText : Query or SP
+ Parameters : SqlParameters[]
+ ExecuteNonQuery ( ) : int
+ ExecuteScalar ( ) : object
+ ExecuteReader ( ) : data reader
SqlDataReader
+ indexer : object
+ FieldCount : int
+ Read ( ) : bool
+ Close ( ) : void
SqlDataAdaptor
+ SelectCommand : SqlCommand
+ InsertCommand : SqlCommand
+ UpdateCommand : SqlCommand
+ DeleteCommand : SqlCommand
+ Fill ( ) : int
+ update ( ) : int

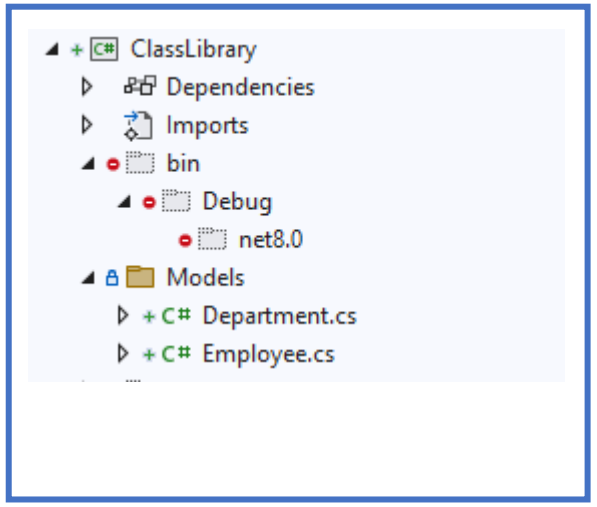
Thank You!

C# Assembly

- Assembly is a collection of types such as classes, interfaces, enums and resources that are built to work together and form a logical unit of functionality.
- Assembly can be a DLL or EXE based on the project type template that we choose.
  - **Class Library** project is a collection of classes and namespaces in C# without any entry point method like Main. **Output Type is .dll**
  - **Console App** project is an application that takes input and displays output at a command line console and behaves as an app host to run .dll (a collection of classes and namespaces in C# with any entry point method like Main). **Output Type is .exe with .dll file**



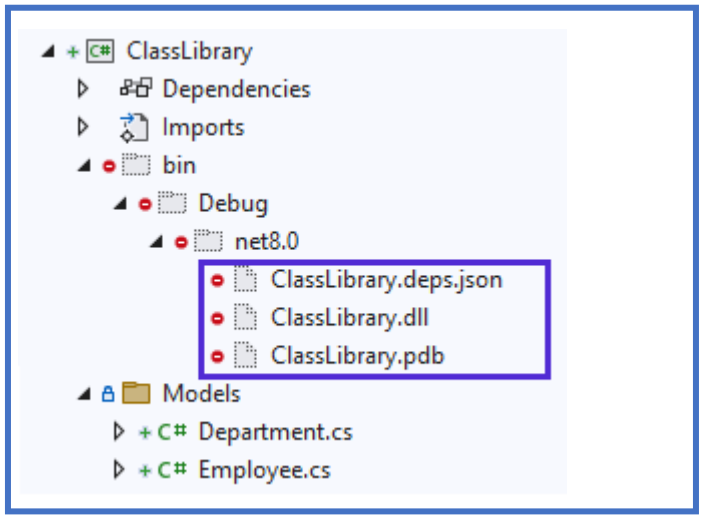
Class Library



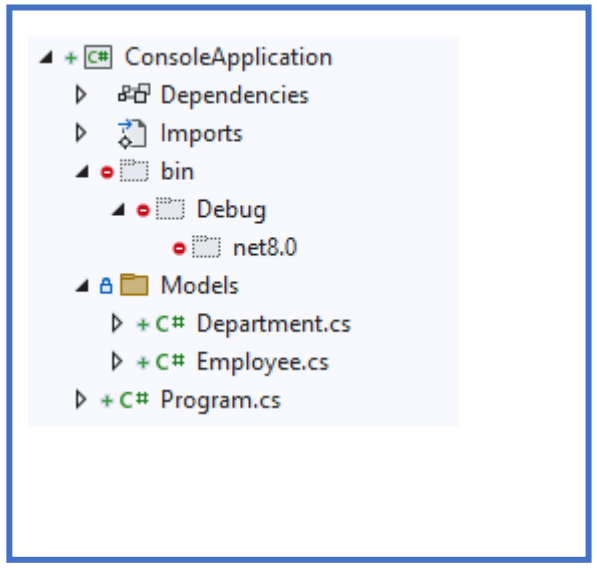
C# Compiler

Build the Class Library project, then C# compiler converts C# code into MSIL code and packages all the types into .dll file

.dll



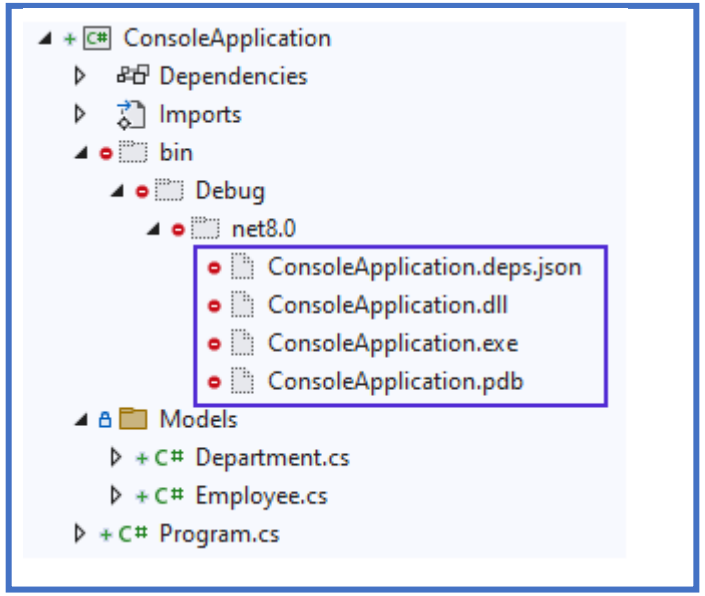
Console Application



C# Compiler

Build or Run the Console App project, then C# compiler converts C# code into MSIL code and packages all the types into .exe and .dll files

.exe



using statement

- **using** block can be used on the objects or instances of the classes which inherits from IDisposable class and implements Dispose method.
- **using** block does ensure that the object Dispose method will always be invoked, no matter if an exception is thrown or not.
- Dispose is a method used to clean up resources. In the case of a DB connection, the connection is released or closed, which is important.

Note: The equivalent of using is a try finally, which includes a call to Dispose within the finally block.

```
4 references | 0 changes | 0 authors, 0 changes
public class MyDisposableClass : IDisposable
{
    1 reference | 0 changes | 0 authors, 0 changes
    public void Dispose()
    {
        //Clean up any unmanagable resources
    }
}
```

```
0 references | 0 changes | 0 authors, 0 changes
public class DisposeDemo
{
    0 references | 0 changes | 0 authors, 0 changes
    public void UsingStatementDemoRun()
    {
        using (MyDisposableClass myDisposableClassObjectInstance = new())
        {
            // Implement logic
        }// using statement ensures to invoke Dispose method of MyDisposableClass,
         // even if an exception is thrown.
    }
}
```

using statement

```
4 references | 0 changes | 0 authors, 0 changes
public class MyDisposableClass : IDisposable
{
    1 reference | 0 changes | 0 authors, 0 changes
    public void Dispose()
    {
        //Clean up any unmanagable resources
    }
}
```

```
0 references | 0 changes | 0 authors, 0 changes
public class DisposeDemo
{
    0 references | 0 changes | 0 authors, 0 changes
    public void UsingStatementDemoRun()
    {
        using (MyDisposableClass myDisposableClassObjectInstance = new())
        {
            // Implement logic
        }// using statement ensures to invoke Dispose method of  MyDisposableClass,
         // even if an exception is thrown.
    }

    0 references | 0 changes | 0 authors, 0 changes
    public void TryCatchFinallyDemoRun()
    {
        MyDisposableClass myDisposableClassObjectInstance = new();
        try
        {
            // logic to be Imlpemented
        }
        catch (Exception)
        {
            throw;
        }
        finally
        {
            myDisposableClassObjectInstance.Dispose();
        }
    }
}
```

using statement example

```
0 references | 0 changes | 0 authors, 0 changes
public class SqlConnectionDisposeDemo
{
    0 references | 0 changes | 0 authors, 0 changes
    public void UsingStatementDemoRun()
    {
        using (Microsoft.Data.SqlClient.SqlConnection connection = new())
        {
            connection.ConnectionString = "Data Source=.\\sqlexpress;Initial Catalog=DisposeDemo;Integrated Security=True;E
            connection.Open();
        }// using statement ensures to invoke Dispose method of SqlConnection,
         // which internally invokes Close() method of SqlConnection object,
         // even if an exception is thrown.
    }

    0 references | 0 changes | 0 authors, 0 changes
    public void TryCatchFinallyDemoRun()
    {
        Microsoft.Data.SqlClient.SqlConnection connection = new();
        try
        {
            // Implement logic
        }
        catch (Exception)
        {
            throw;
        }
        finally
        {
            if (connection.State == ConnectionState.Open)
            {
                connection.Close();
                //connection.Dispose();
            }
        }
    }
}
```