ASP.NETITI

▼ C#

▼ The history of the form of writing the code in the beginning

- ▼ **linear**

  written on top of each other,

- ▼ Structured

  then it became divided into Main, Function

- ▼ OOP

  is about Classes, and parameters. This is the final form so that the code is easy to access, and if there is an error in it, it is easy to fix it and know the location of each part separately.

Why do we use C Sharp?

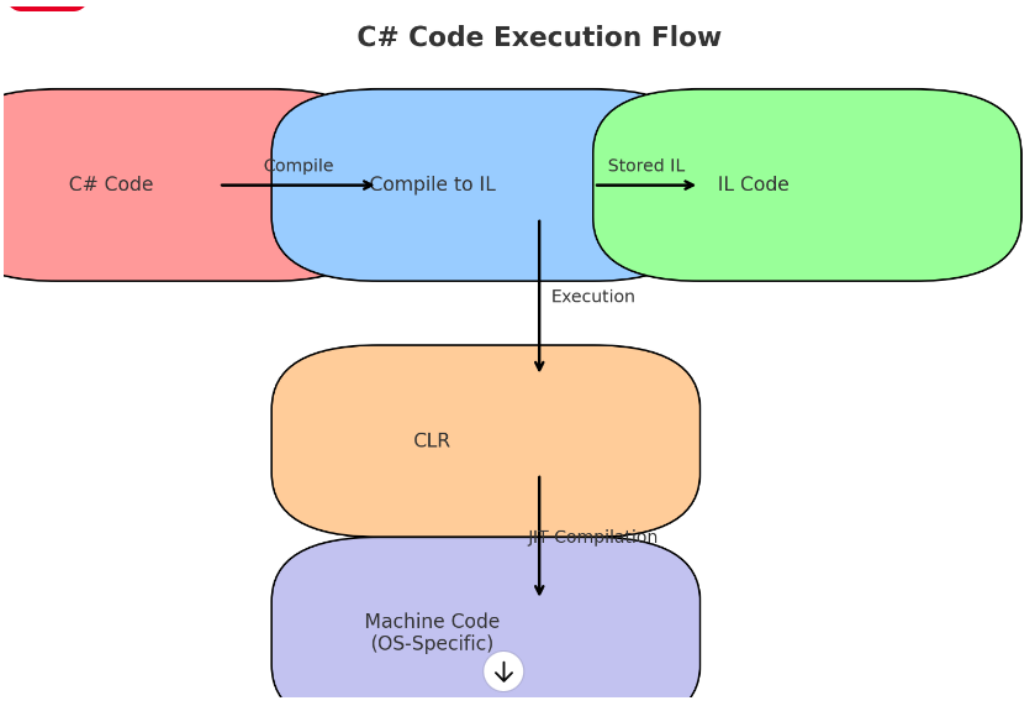
CLI (Common Language Infrastructure)

Unifying the way to run the program on any operating system, so that when I come to compile the program for the first time, my program turns into something called intermediate language IL. This code will be independent of the operating system, and then it will come to CLR.

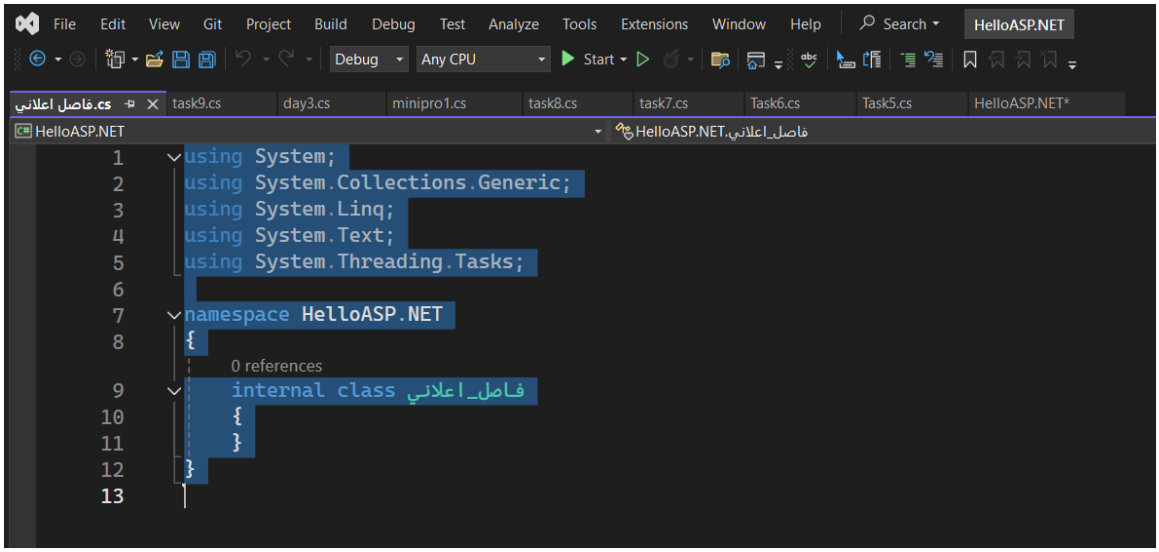
CLR (Common Language Runtime)

It makes my code not restricted to a specific operating system. The code works on all of them without me having to bring a new compiler. It makes the code more secure and flexible.

CLR, by the way, is part of CLI and is responsible for running IL by converting it to jit compilation in the language appropriate to the operating system and processor of the device on which the program runs.



C# Syntax



> In short, Syntax is like any language, but for different needs, the word Using allows you to use a specific package. This namespace is used to separate projects from each other, and there is a class inside it to divide the code, and there is a Main function inside it.

▼ OPERATORS

Arithmetic Operators

Operator	Description	Example (A=10, B=20)
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by denominator	B / A will give 2
%	The remainder after an integer division	B % A will give 0
++	Increment operator increases integer value by one	A++ will give 11
--	Decrement operator decreases integer value by one	A-- will give 9

Relational Operators

Operator	Description	Example (A=10, B=20)
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

Operator	Description	Example(A is TRUE and B is FALSE)
&&	**Called Logical AND operator. If both the operands are non zero then condition becomes true.**	**(A && B) is false.**
||	**Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.**	**(A || B) is true.**
!	**Called Logical NOT Operator. Used to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.**	**!(A && B) is true.**

▼ If Else new style

```
using System;

namespace HelloASP.NET
{
    internal class Task2
    {
        static void Main()
        {
            // Prompt the user to enter an integer
            Console.WriteLine("Please enter an integer:");
            string input = Console.ReadLine();
            int number = Convert.ToInt32(input);

            // Check if the number is odd or even
            string result = (number % 2 == 0) ? $"{number} is
            Console.WriteLine(result);
        }
    }
}
```

▼ Break & Continue

```
class ContinueTest
{
    static void Main()
    {
        for (int i = 1; i <= 10; i++)
        {
            if (i < 9)
            {
                continue;
            }
            Console.WriteLine(i);
        }

        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/*
Output:
9
10
*/
```

```
class BreakTest
{
    static void Main()
    {
        for (int i = 1; i <= 100; i++)
        {
            if (i == 5)
            {
                break;
            }
            Console.WriteLine(i);
        }

        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/*
 Output:
    1
    2
    3
    4
*/
```

▼ Method Type

> The method is a code that performs a specific function.

> It has a name and takes a parameter. must be part of class.

- public I can call on anywhere.
- The protected class will be available to the class it is in and the classes it inherits from.

- The privat will only be available for the class it is in
- . What does this Static mean that makes me able to call for the Method without any object.

```
using System;


namespace HelloASP.NET
{
    internal class task4
    {

        static void Main()
        {




            Console.WriteLine("Square Len");
            double Len = Convert.ToDouble(Console.ReadLine())

            double AreaAmount = calcarea(Len);

            Console.WriteLine($"Area: {AreaAmount:F2}");
        }

        static double calcarea(double Len)
        {
            return Len*4;
        }
    }
}
```

▼ Method Call

in main{

method m= new method();

objectname.methodname(parameter);

}

▼ Passing Value

▼ Call by Value

📌 The function takes a copy of the variable's value but does not change its original value.

```
void UpdateValue(int x)
{
    x = x + 10;
}

int num = 5;
UpdateValue(num);
Console.WriteLine(num);  // 5 هتكون النتيجة
```

▼ Call by REF

📌 The function takes a reference to the same address in memory, meaning any change changes the original value.

```
void UpdateValue(ref int x)
{
    x = x + 10;
}

int num = 5;
```

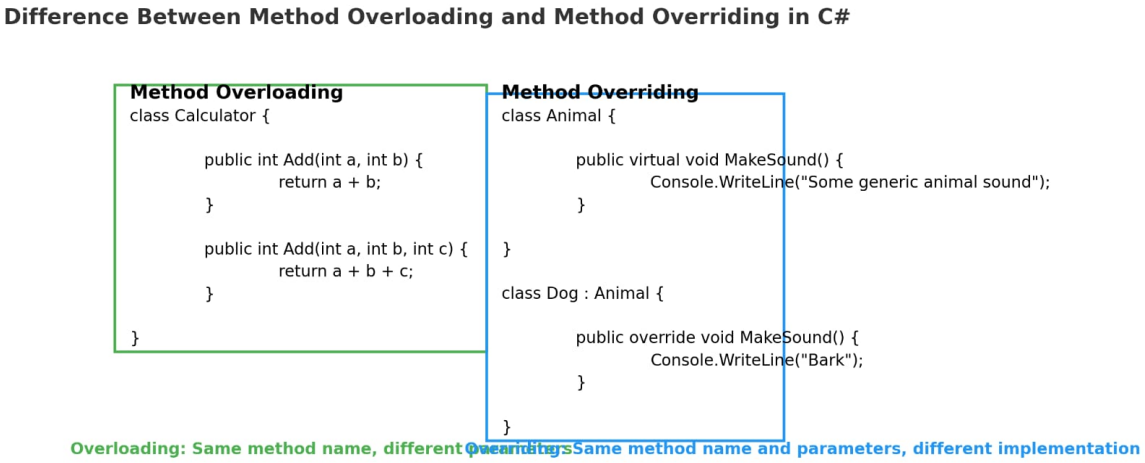
```
UpdateValue(ref num);
Console.WriteLine(num);  // 15 هتكون النتيجة
```

▼ Overload concept

📌 Same name Function different parameters in the same class.

▼ Override Concept

📌 Modify the behavior of a function (what the function will execute) in an inherited class. keyword virtual، abstract، override.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloASP.NET
{
    internal class task9
    {
```

```
// Base class
class Item
{
    public void Company()
    {
        Console.WriteLine("Item Code = Base");
    }
}

// Derived class
class Fan : Item
{
    public string ModelName { get; set; }

    // Constructor to initialize the model name
    public Fan(string modelName)
    {
        this.ModelName = modelName;
    }

    // Override the ToString method
    public override string ToString()
    {
        return $"Fan Model: {ModelName}";
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Create an array of Fan objects
        Fan[] fans = new Fan[3];
```

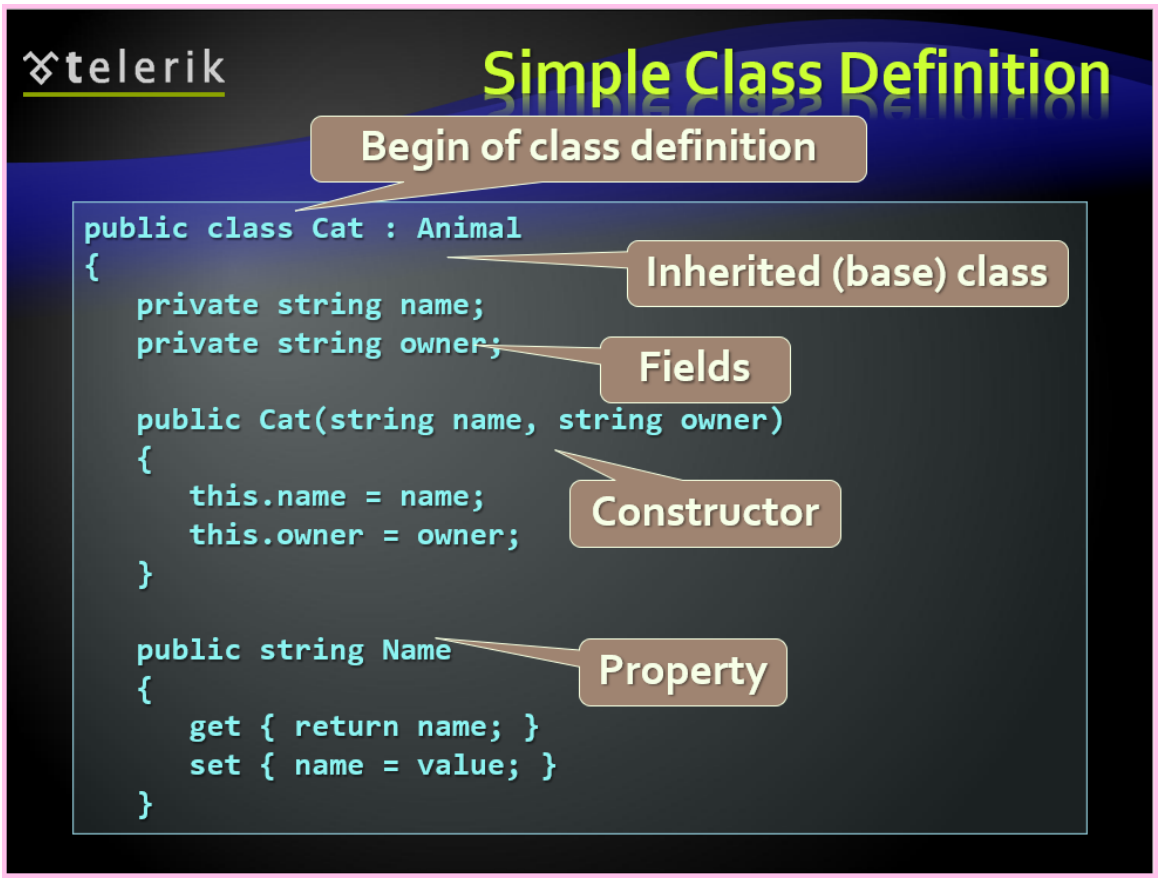
```
                // Initialize the array with Fan objects
                fans[0] = new Fan("Model A");
                fans[1] = new Fan("Model B");
                fans[2] = new Fan("Model C");

                // Display the details of each fan
                foreach (var fan in fans)
                {
                    Console.WriteLine(fan.ToString());
                }

                Console.ReadLine();
            }
        }
    }

}
```

▼ Operator Overloading

📌 let the objects interact with each other, such as +,-, ==, ! = ,like int ,float...etc.

```
Point p1 = new Point(1, 2);
Point p2 = new Point(3, 4);
Point result = p1 + p2;

Console.WriteLine($"Result: ({result.X}, {result.Y})"); // Ou
```

▼ Class




```
using System;


namespace HelloASP.NET
{
    internal class Task5
    {



        public class Book
        {

            private int book_id;
            private string book_name;
```

```
            private string author;
            public Book() { }
            public void set(int id, string name, string autho
            {
                book_id = id;
                book_name = name;
                this.author = author;
            }
            public void get()
            {
                Console.WriteLine(book_id);
                Console.WriteLine(book_name);
                Console.WriteLine(author);
                Console.WriteLine("*");
            }
            static void Main()
            {
                Book book1 = new Book();
                book1.set(1, "A", "isd1");
                book1.get();
                Book book2 = new Book();
                book2.set(2, "B", "isd2");
                book2.get();
                Book book3 = new Book();
                book1.set(3, "C", "isd3");
                book1.get();

            }
        }
    }

}
```

📌 NOTE

1. ***Using Constructor*:** You can include messages in the constructor that will appear when an object of the class is created, which happens when the program runs. the constructor initializes the attribute and does not accept the inheritance.
2. ***Updating Values*:** Initial values set in the constructor are not automatically updated but can be modified through methods later.
3. ***Getters*:** You can use getter methods to retrieve stored values and display them on the console.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloASP.NET
{
    internal class Task6
    {



        class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("Please enter an integer:")
                string input = Console.ReadLine();
                int number = Convert.ToInt32(input);
                Number numberObj = new Number(number);
```

```
                Console.WriteLine(numberObj.CheckEvenOdd());
            }
        }

        public class Number
        {
            private int _value;

            public Number(int value)
            {
                _value = value;
            }

            public string CheckEvenOdd()
            {
                if (_value % 2 == 0)
                {
                    return $"{_value} is an even integer";
                }
                else
                {
                    return $"{_value} is an odd integer";
                }
            }
        }

}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace HelloASP.NET
{
    internal class minipro1
    {


        class Program
        {
            static void Main(string[] args)
            {




                Bank per1 = new Bank(61, "Rawan", 32000);
                per1.Deposit(500.0f);
                per1.Withdraw(3500.0f);
                per1.Display();

            }
        }

        public class Bank
        {
            private int _id;
            private string _name;
            private float _balance;

            public Bank(int id, string name,float balance)
            {
                _id = id;
                _name = name;
                _balance= balance;

            }

            public void Deposit(float amount)
```

```
            {
                _balance += amount;
                Console.WriteLine($"Deposited: {amount}, New
            }

            public void Withdraw(float amount)
            {
                if (amount <= _balance)
                {
                    _balance -= amount;
                    Console.WriteLine($"Withdrawn: {amount},
                }
                else
                {
                    Console.WriteLine("Insufficient balance."
                }
            }

            public void Display()
            {
                Console.WriteLine($"Account ID: {_id}, Name:
            }
        }
    }

}
```

📌 NOTE:

1. ***Class***:
    - *Private*: Accessible only within the same assembly.

- *Public*: Accessible from anywhere, including other assemblies.
- *Protected*: Accessible within the class itself and by derived classes.
- *Inheritance*: A class can be a base class from which other classes can inherit.

2. ***Attributes (Fields)***:
   - *Private*: Accessible only within the class itself.
   - *Public*: Accessible from anywhere.
   - *Protected*: Accessible within the class itself and by derived classes.
   - *Protected Internal*: Accessible within the same assembly and by derived classes.
3. ***Methods***:
   - *Public*: Accessible from anywhere, and can be inherited and overridden by derived classes.
   - *Protected*: Accessible within the class and by derived classes, and can be overridden.
   - *Private*: Accessible only within the class itself, not accessible by derived classes.
   - *Overridable*: Methods defined as virtual or abstract in the base class can be overridden using the override keyword in derived classes.

▼ INHERITANCE

Inheritance allows code to be reused flexibly while respecting access and privacy rules.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloASP.NET
{
```

```
internal class day3
{
    // Base class
    public class Person
    {
        public int Id { get; set; }
        public string Name { get; set; }

        public Person(int id, string name)
        {
            Id = id;
            Name = name;
        }

        public void DisplayInfo()
        {
            Console.WriteLine($"ID: {Id}, Name: {Name}"
        }
    }

    // Derived class
    public class Employee : Person
    {
        public double Salary { get; set; }

        public Employee(int id, string name, double sal
            : base(id, name)
        {
            Salary = salary;
        }

        public void DisplayEmployeeInfo()
        {
            DisplayInfo();
            Console.WriteLine($"Salary: {Salary}");
        }
    }
```

```
        // Further derived class
        public class Manager : Employee
        {
            public double Bonus { get; set; }

            public Manager(int id, string name, double sala
                : base(id, name, salary)
            {
                Bonus = bonus;
            }

            public void DisplayManagerInfo()
            {
                DisplayEmployeeInfo();
                Console.WriteLine($"Bonus: {Bonus}");
            }
        }

        class Program
        {
            static void Main(string[] args)
            {
                // Create an instance of Manager
                Manager manager = new Manager(1, "Rawan", 5
                manager.DisplayManagerInfo();
            }
        }
    }

}
```

Using public methods to access private data: A parent class may contain public methods that allow a child class or any other class to access private properties or methods in a safe way.

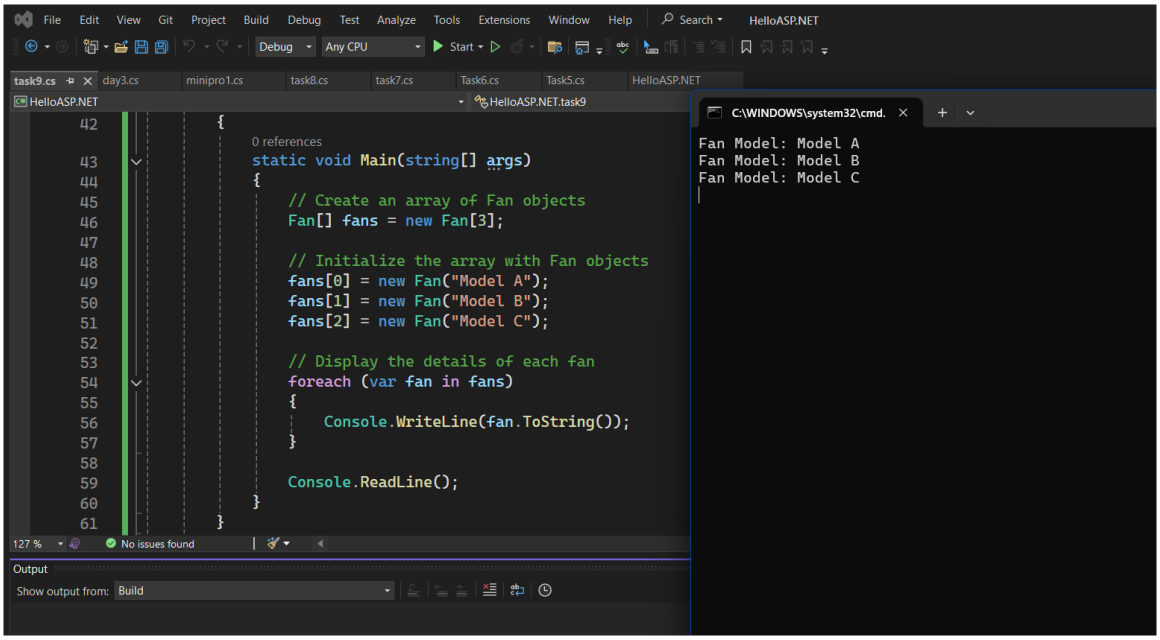
```
public class Parent
{
    private int privateField;

    public void SetPrivateField(int value)
    {
        privateField = value;
    }

    public int GetPrivateField()
    {
        return privateField;
    }
}

public class Child : Parent
{
    public void UseParentMethods()
    {
        SetPrivateField(5);
        int value = GetPrivateField();
    }
}
```

▼ Array

📌 Array help us organize data better. Managing a large group of data, and I can implement specific data on them, for example, if I am going to modify something

```
Person[] people = new Person[3];
people[0] = new Person { Name = "Ahmed", Age = 30 };
people[1] = new Person { Name = "Sara", Age = 25 };
people[2] = new Person { Name = "Mona", Age = 27 };
for (int i = 0; i < people.Length; i++)
```

```
{
    Console.WriteLine($"Name: {people[i].Name}, Age: {people[
}
```



▼ Struct

📌 Used to store simple and direct data.

```
public struct Point
{
    public int X { get; set; }
    public int Y { get; set; }

    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }
}

// Usage
Point point = new Point(5, 10);
Console.WriteLine($"Point is at ({point.X}, {point.Y})"
```

▼ When do we use class and when Struct?

Using a class or a struct depends on your programming needs and the characteristics of the objects you're working with. Here are some reasons to use a class instead of a struct:

1. Inheritance

- **Class**: Supports inheritance, allowing you to derive new classes from existing ones. This is useful when you have a hierarchy of data and behaviors.
- **Struct**: Does not support inheritance. If you need inheritance, you must use a class.

2. Flexibility in Managing Data

- **Class**: Acts as a reference type, meaning when you copy a class object, both references point to the same data. This is helpful when you want changes to be reflected across different parts of the code.

- **Struct**: Acts as a value type, meaning each copy is independent. This can be undesirable if you need shared data modifications.

3. Memory and Performance

- **Class**: Uses heap memory, which can have a performance cost if you're creating many small objects that need to be accessed quickly.
- **Struct**: Uses stack memory, making it faster in some cases, especially for small, simple data structures.

4. Memory Management

- **Class**: Managed by garbage collection, which automatically handles memory allocation and deallocation. This is convenient if you don't want to manage memory manually.
- **Struct**: Does not require garbage collection, as it is typically deallocated when it goes out of scope, providing faster cleanup.

5. Behaviors and Interactions

- **Class**: Suitable for complex objects that require methods and advanced functionalities. You can add behavior (methods) and complex logic to a class.
- **Struct**: Generally simpler, used for storing data without complex behavior.

When to Use Each

- **Use Class**: When you have complex data, need inheritance, or require flexibility in interacting with the data.
- **Use Struct**: When you have simple data, need high performance, and want to avoid the overhead of memory management that comes with classes.

▼ Nullable

allow you to create a variable that accepts the value Null.

```
int? age = null; // Here the variable age can take a number o
```

▼ Interfaces

Interfaces define a set of functions that a class must implement. They enable us to implement multiple inheritance in a safe way.

Summary of Using Interfaces

- **The** `interface` defines what needs to be done (the methods that must be implemented) but does not specify how to do it.
- **Each class** that implements the `interface` provides its own way of performing the methods defined by the interface.
- **Using the** `interface` allows you to work with different objects consistently, even if their implementations of the methods differ.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloASP.NET
{
    internal class TaskDAY5
    {
        public interface IShape
        {
            int CalculateSurface();
        }

        public interface IMovable
        {
            void Move(int deltaX, int deltaY);
            void SetPosition(int x, int y);
```

```
}

public abstract class MovableShape : IShape, IMoval
{
    private int x, y;

    public void Move(int deltaX, int deltaY)
    {
        this.x += deltaX;
        this.y += deltaY;
    }

    public void SetPosition(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public abstract int CalculateSurface();
}

public class SunShape : MovableShape
{
    private int radius;

    public SunShape(int x, int y, int radius)
    {
        SetPosition(x, y);
        this.radius = radius;
    }

    public override int CalculateSurface()
    {
        // Assuming the surface area is calculated
        return (int)(Math.PI * radius * radius);
    }
}
```

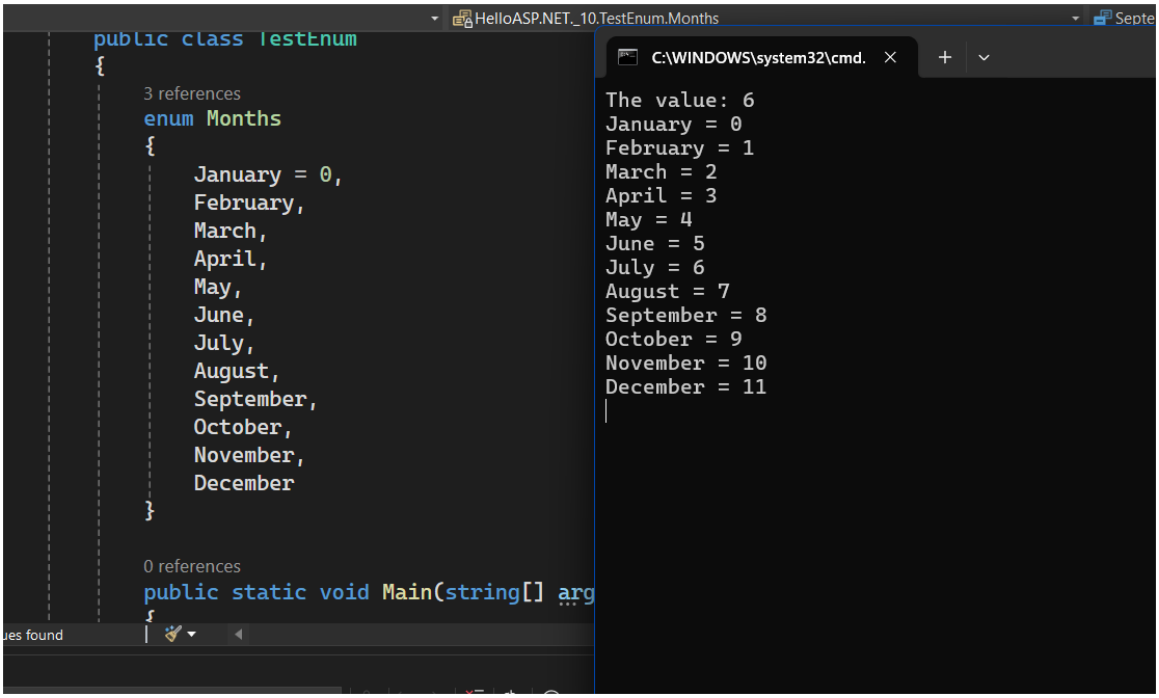
```
        class Program
        {
            static void Main(string[] args)
            {
                // Create a new SunShape object
                SunShape sun = new SunShape(0, 0, 5);

                // Set a new position
                sun.SetPosition(10, 10);

                // Move the shape
                sun.Move(2, 3);

                // Calculate and display the surface area
                int surface = sun.CalculateSurface();
                Console.WriteLine($"The surface area of the
            }
        }
    }

}
```

```
interface IFly
{
    void Fly();
}

interface ISwim
{
```

```
    void Swim();
}

class Bird : IFly, ISwim
{
    public void Fly()
    {
        Console.WriteLine("The bird can fly.");
    }

    public void Swim()
    {
        Console.WriteLine("The bird can swim.");
    }
}
```

📌 Multiple inheritance means that one class can inherit from another class, but this is not supported directly in C Sharp, so I use the interface.

▼ Enums



▼ Tool

Online C# Compiler (Editor) - Programiz

▼ Database

▼ write-ups

▼ Database & SQL for beginners — Part(1)

https://bit.ly/3LRW9IZ

▼ Database & SQL — Part(2) ERD

https://bit.ly/46wYAEa

▼ Database & SQL — Part(3) SQL

https://bit.ly/3SAKZFT

▼ How to convert the ERD to a Relational Model Database while maintaining the principles of Set Theory “Mapping”

https://bit.ly/4deCpoE

▼ Normalization

https://bit.ly/3SK228S

- Mapping

  https://bit.ly/3SK228S

Tools

TO design ERD:

https://erdplus.com/

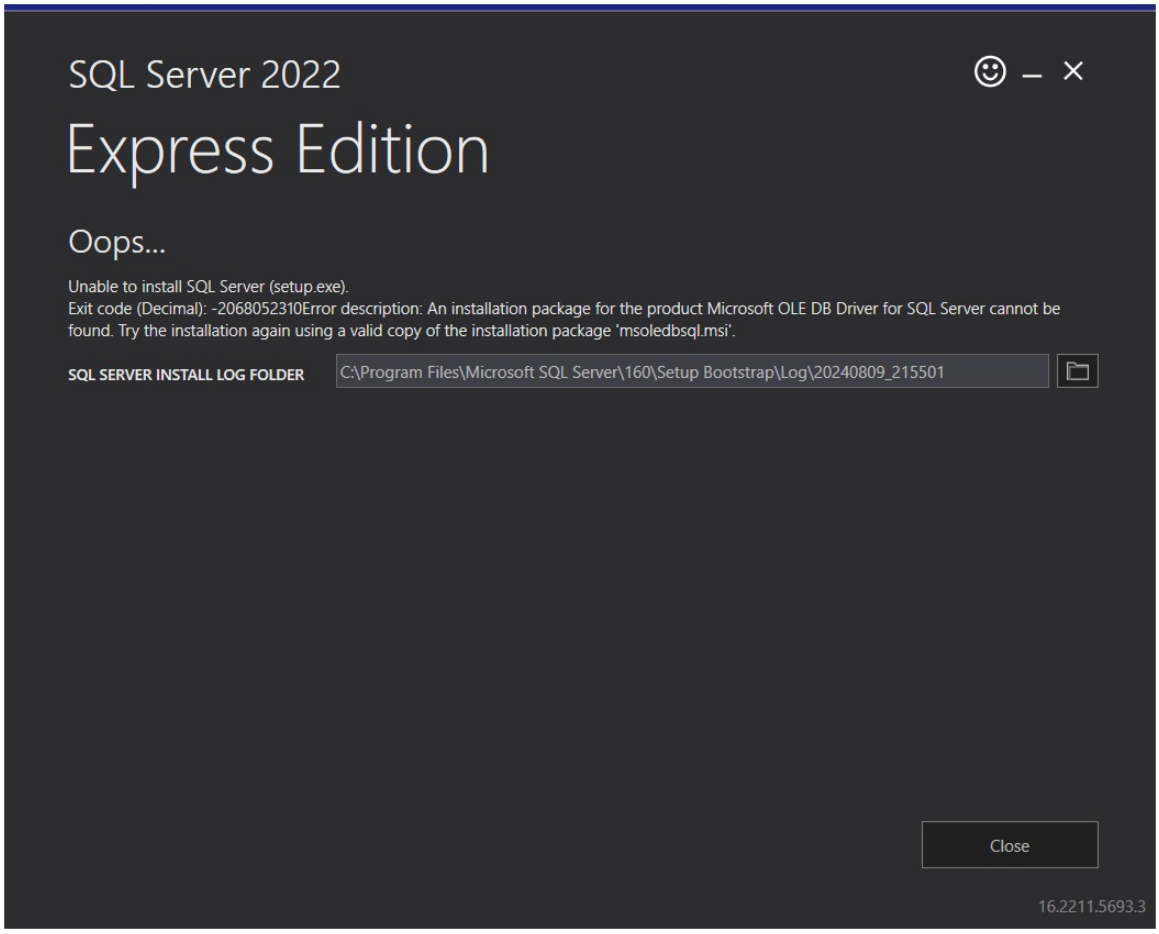
Download SQLServer:

https://www.microsoft.com/en-us/sql-server/sql-server-downloads

FoR Help:

https://www.youtube.com/watch?v=dPs7BQ4Zx_Q

Problems: Unable to install SQL Server



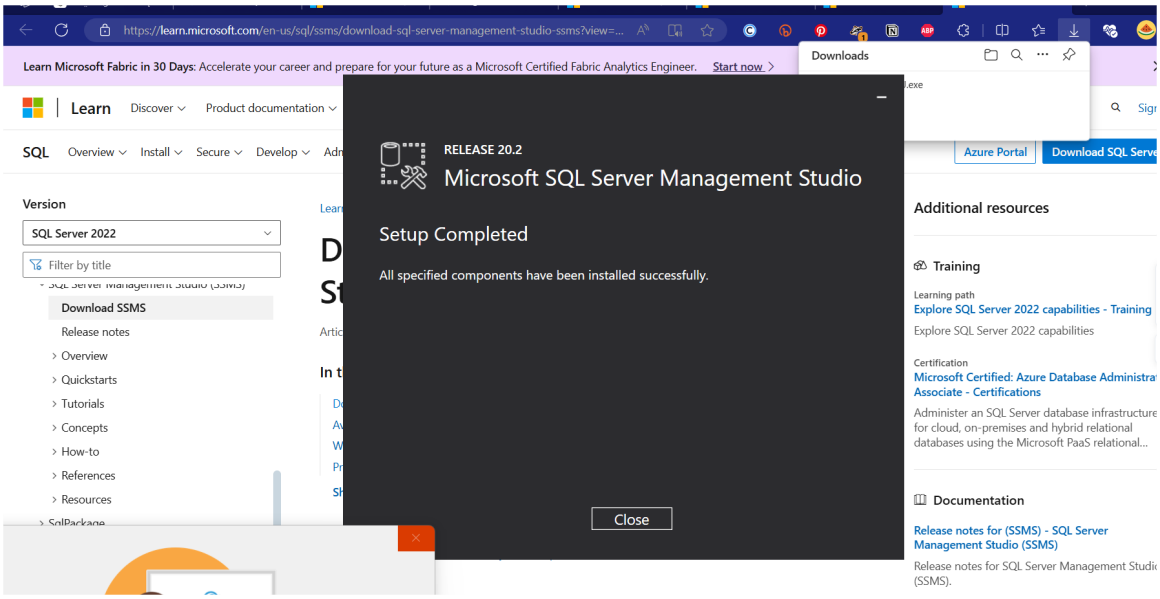
SOLU:

Sometimes there are files left over from previous installations that cause problems. Try removing any SQL Server related files by going to⟶ Control Panel > Programs and Features and deleting anything related to SQL Server or OLE DB Driver.

After uninstalling, delete the files from the installation path:
C:\Program Files\Microsoft SQL Server
C:\Program Files (x86)\Microsoft SQL Server
C:\ProgramData\Microsoft\SQL Server

and redownload SQL server

AND FINALLY:



▼ Relational Database Model VS Object-Relational Mapping (ORM)

📌 The relational database model is the form in which data is stored in the database, and the ORM is the way to connect your code to that data conveniently.

📌

- *Relational Database Model*: It is the system that organizes data in databases using tables and relationships between them.
- *Object-Relational Mapping (ORM)*: It is a technique you use in your code to link objects in programming and tables in these databases, making it easier for you to deal with data without having to write SQL queries directly.

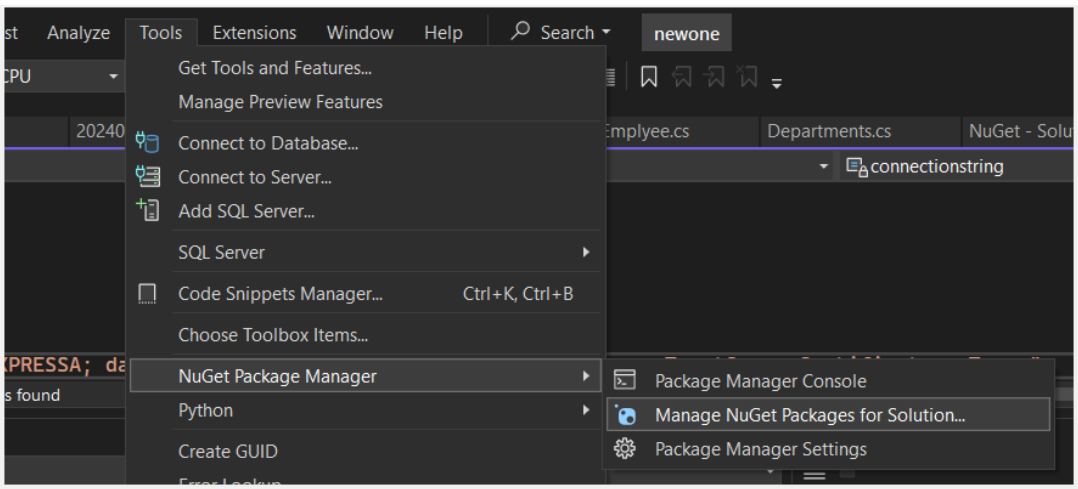
Simply put, *ORM* helps you deal with *Relational Databases* more easily through code.

▼ First Project [Dot Net]

> When creating a project in Visual Studio:

> Choose .NET as the development environment.

> Download the packages required for the project such as:
>
> ▼ How to download packages?



When you want to download packages in Visual Studio, follow these steps:

1. *Open the NuGet Package Manager*:

- In the Tools menu, choose "NuGet Package Manager" and then click on "Manage NuGet Packages for Solution."

1. *Search for the required packages*:

- In the search box, type the name of the package you need such as "Entity Framework" or "Entity Framework SQL Server" or others.

1. *Choose the appropriate version*:

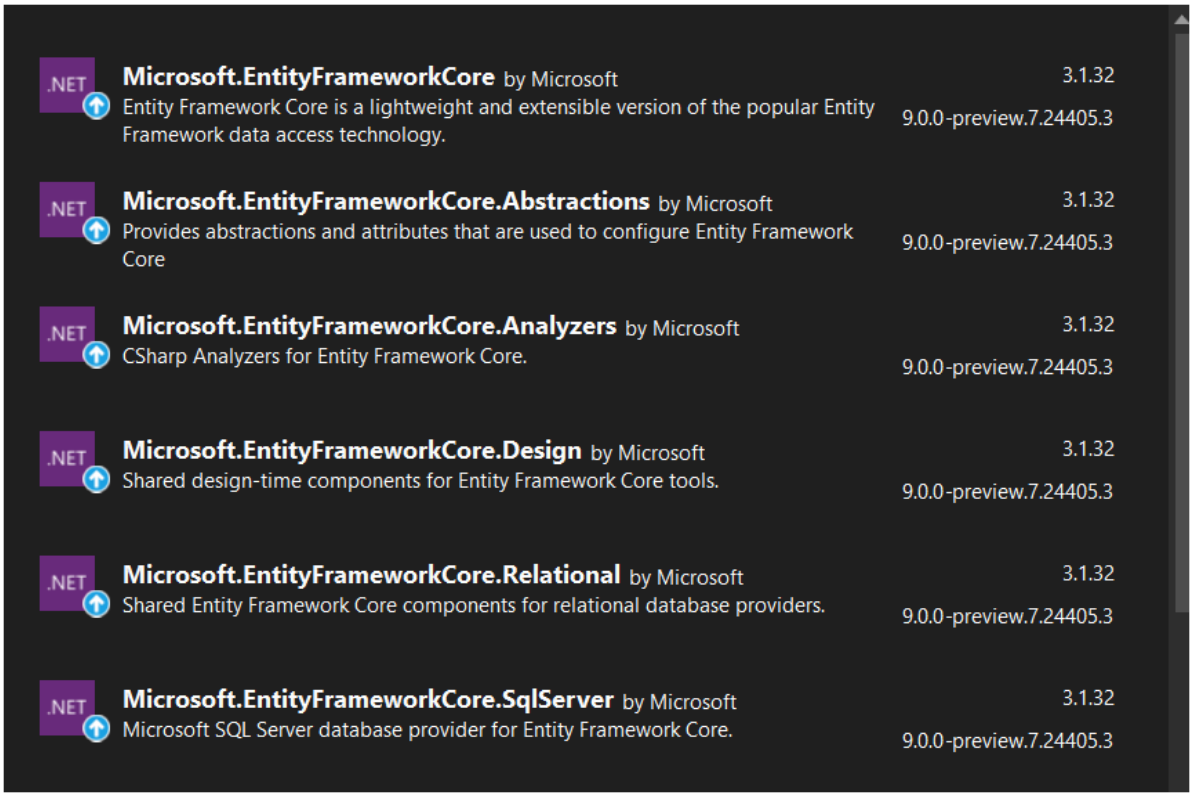
- Once you find the package, select the appropriate version for your project from the drop-down list.

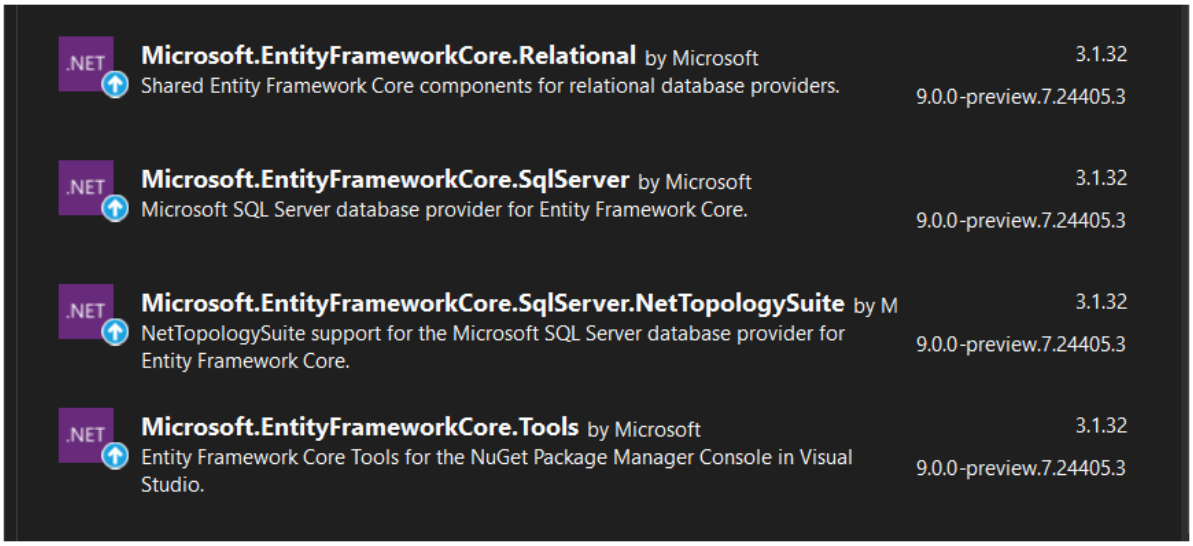
1. *Install the packages*:

- Click "Install" to start the download and installation process for the package.

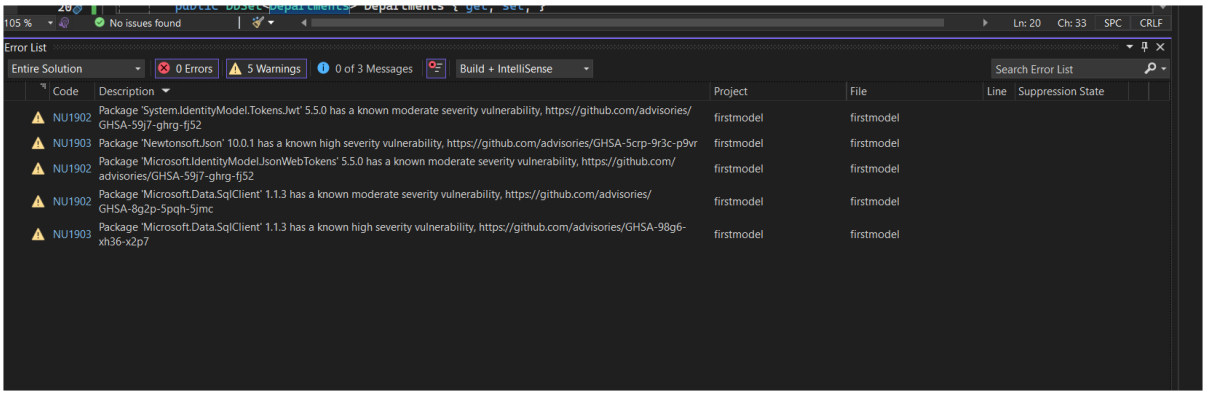
> 1. *Review and approve the changes*:
>
>   - After installation, you may be asked to confirm some changes, click "OK" and "Accept" to complete the process.
>
>   This way, you have downloaded the packages required for the project correctly.

1. *Entity Framework*
2. *Entity Framework Tools*
3. *Entity Framework SQL Server*
4. *Entity Framework Design*
5.





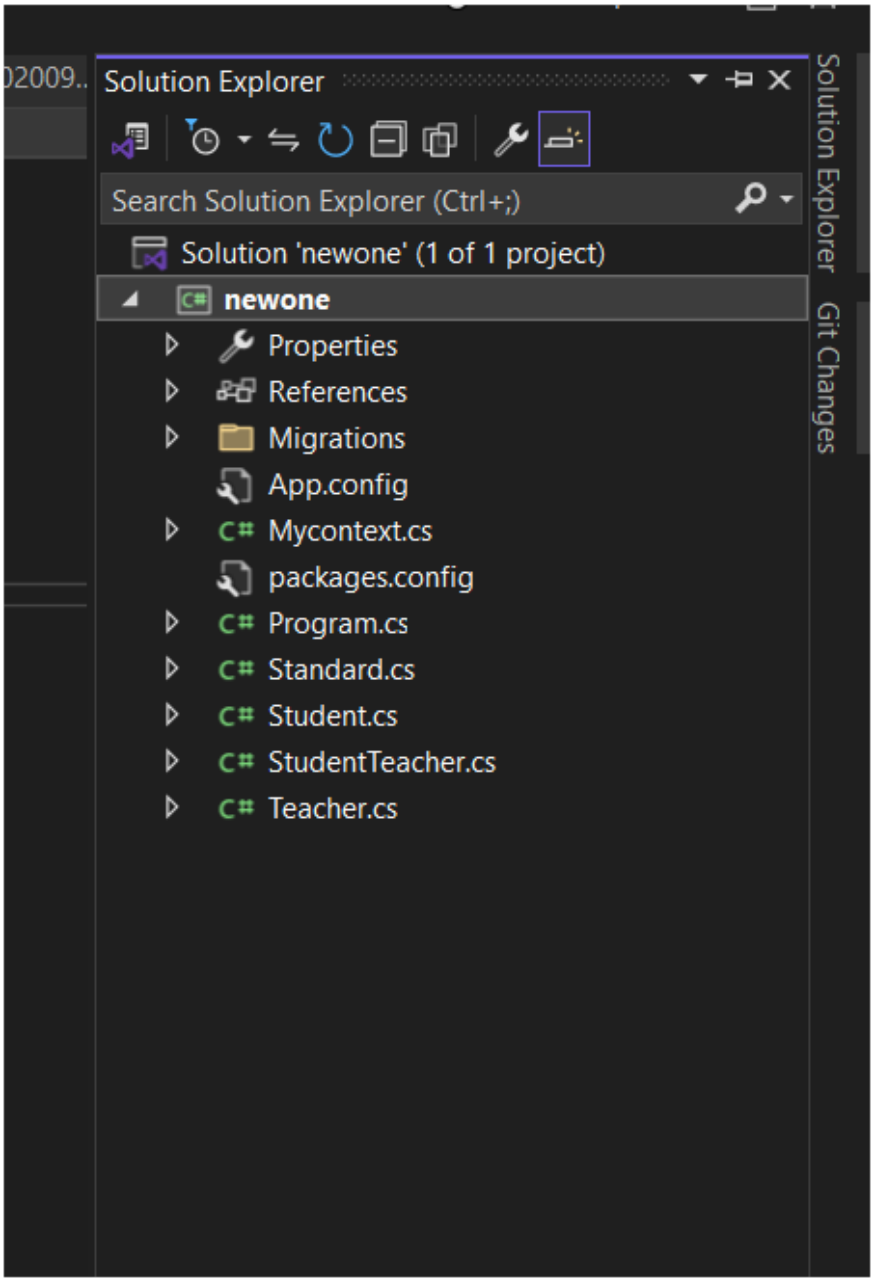
> Make sure to choose the appropriate version for the packages based on the version compatible with your device.



> 📌 You may find the normal Warning, the visual will work

> After preparing the project and packages:

> Start writing the code and creating classes.

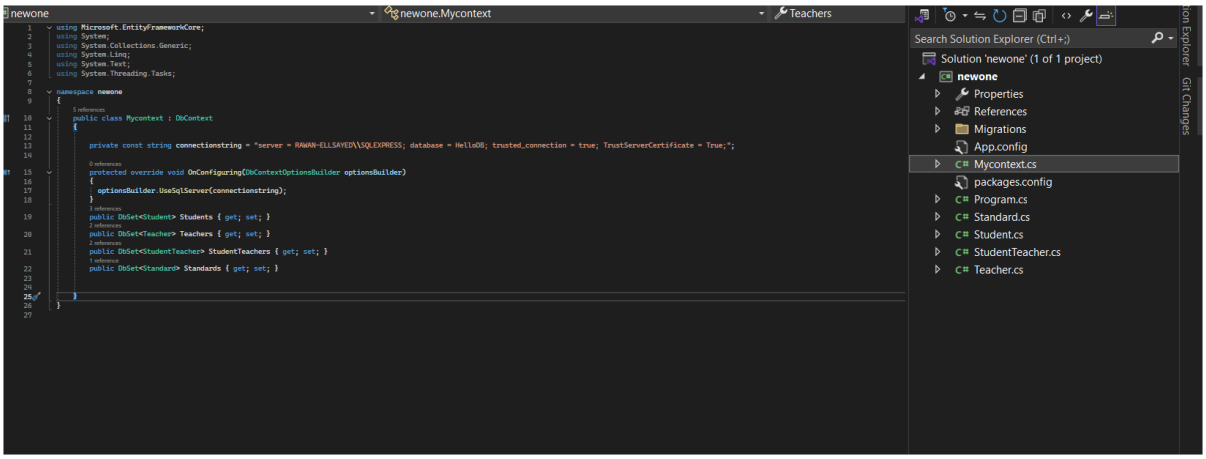


> Specify the attributes and keys inside the classes.

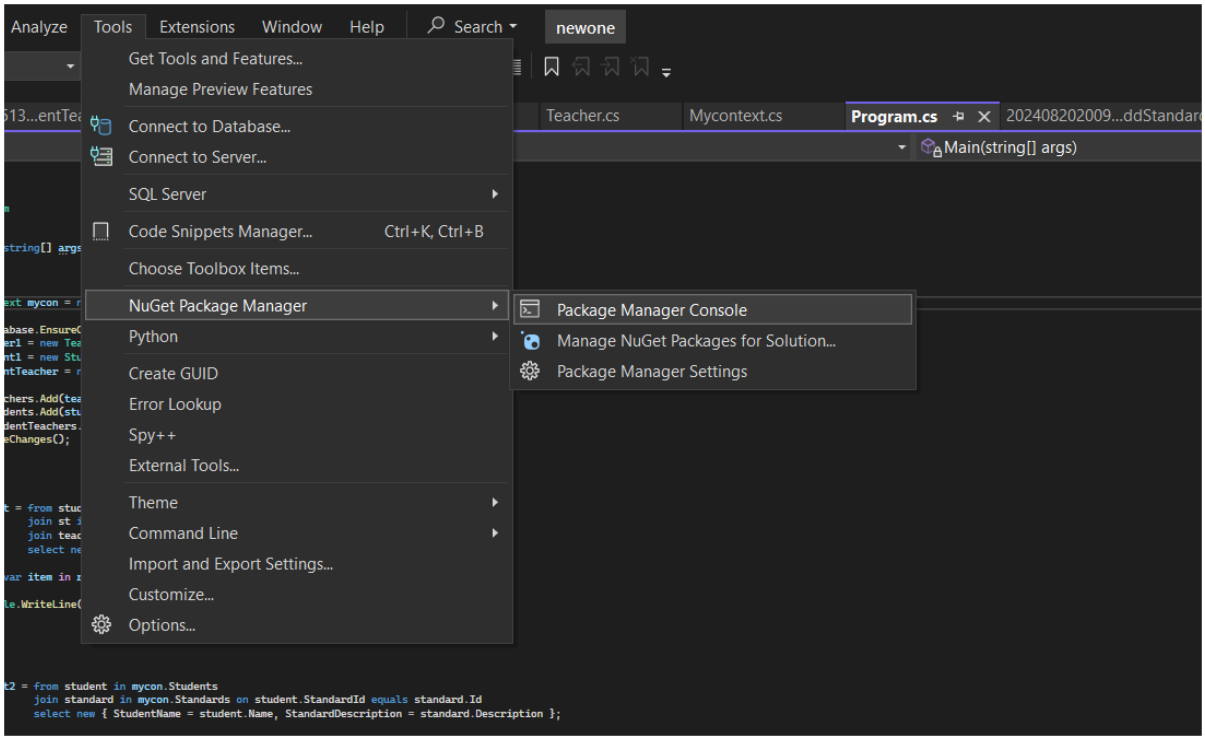
> If you use the id property as the Primary Key, it will be set automatically.

> If you write the name of the table followed by id, if this id is not the Primary Key, you must specify it by typing [Key] above it.

```
namespace newone
{
    4 references
    public class Emplyee
    {
        [Key]
        1 reference
        public int SSN { get; set; }
        1 reference
        public string Name { get; set; }
        1 reference
        public DateTime DOB { get; set; }
        1 reference
        public string Gender { get; set; }
        1 reference
        public decimal Salary { get; set; }
    }
}
```



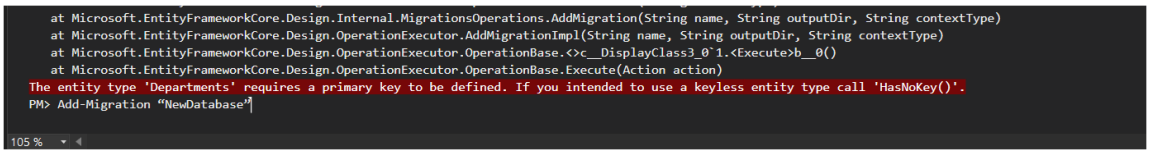
```
0 references
internal class Program
{
    0 references
    static void Main(string[] args)

    {

        using (Mycontext mycon = new Mycontext())
        {
            mycon.Database.EnsureCreated();
            var teacher1 = new Teacher { Name = "Mr. John", Address = "123 Main St" };
            var student1 = new Student { Name = "Alice" };
            var studentTeacher = new StudentTeacher { Student = student1, Teacher = teacher1 };

            mycon.Teachers.Add(teacher1);
            mycon.Students.Add(student1);
            mycon.StudentTeachers.Add(studentTeacher);
            mycon.SaveChanges();
```



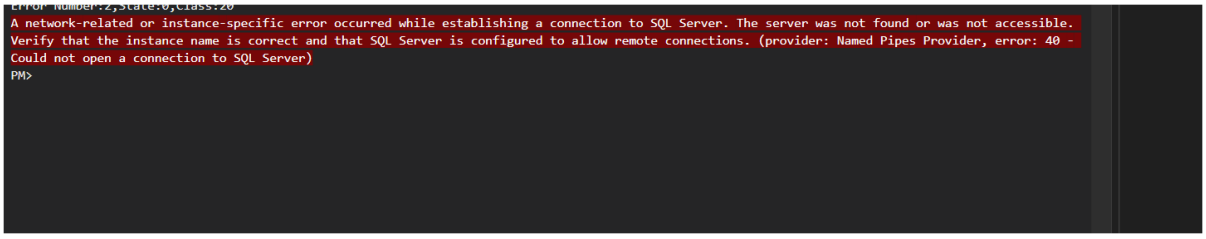

```
The name 'NewDatabase' is used by an existing migration.
PM> Add-Migration HelloDB
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> update-database
Build started...
Build succeeded.
Applying migration '20240818123442_NewDatabase'.
Applying migration '20240818125331_HelloDB'.
Done.
PM>
```

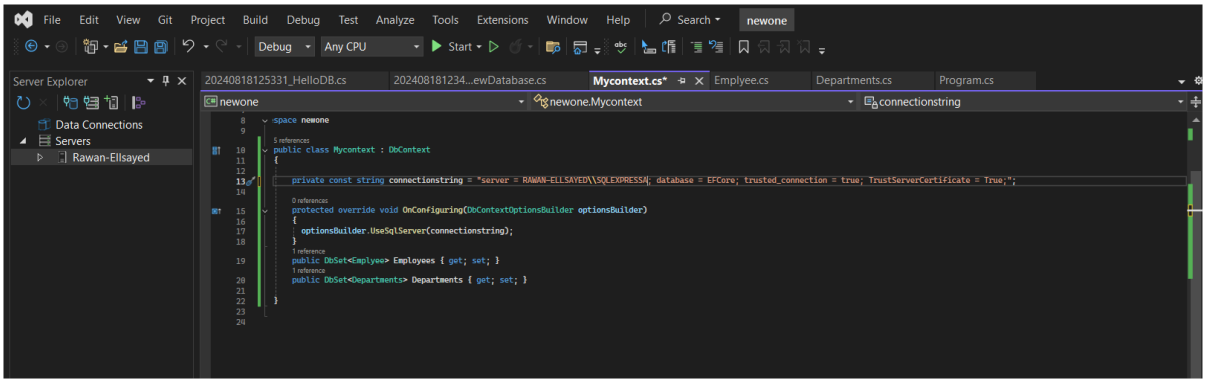
▼ My problems in linking the database between Visual and MS SQL server First Code Approach

- Make sure you have set the PK for each Attribute



- Ensure you have correctly written the connection and that Visual connects to your server.
-





```
namespace newone
{
    4 references
    public class Mycontext : DbContext
    {

        private const string connectionstring = "RAWAN-ELLSAYED\\SQLEXPRESS;" + "Database=EFCore;Trusted_Connection=True;";

        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(connectionstring);
        }
        1 reference
        public DbSet<Emplyee> Employees { get; set; }
        1 reference
        public DbSet<Departments> Departments { get; set; }

    }
}
```

- Add Migration

Adding Migration

- Now, our model is ready. The model has TWO entities. We have created Mycontext class to manage the entities. Now it is time to create the database.
- Click on **Tools – > NuGet Package Manager > Package Manager Console** to open the Console
- Run the Add-Migration to create the migration.

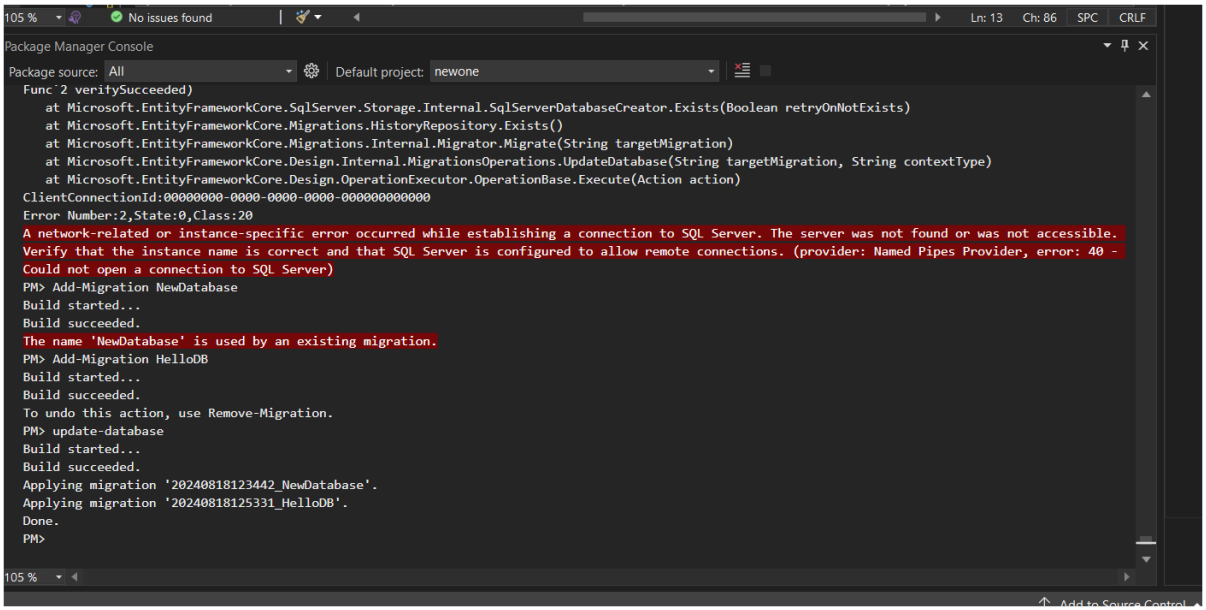
Add-Migration "NewDatabase"

- The Open the **Package Manager Console**

and run the command **update-database**

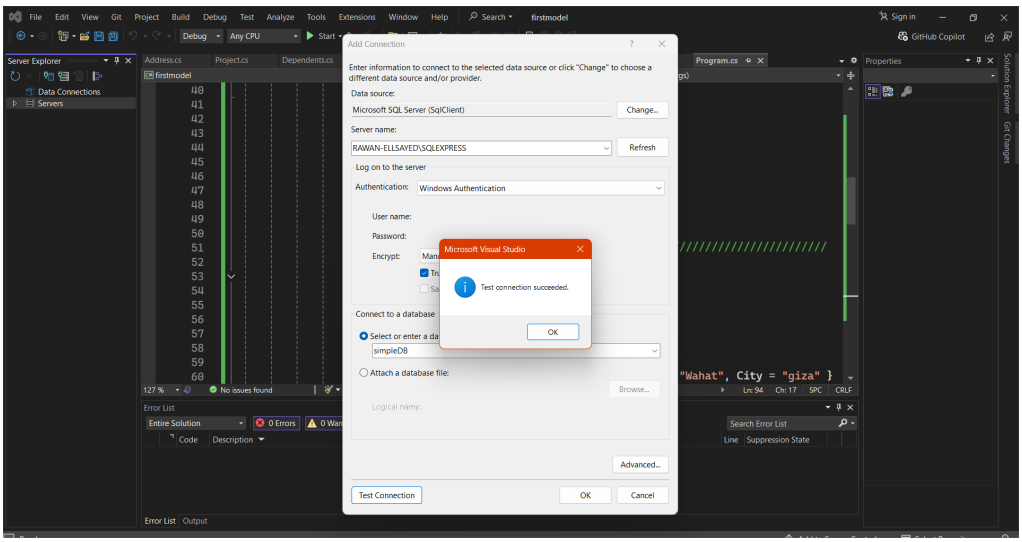
```
Package Manager Console
Package source: All        Default
PM> update-database
Applying migration '20180814170914_NewDatabase'.
Done.
```

```
3
4   namespace EFCore.Migrations
5   {
6       public partial class NewDatabase :
7       {
8           protected override void Up(Migr
9           {
10              migrationBuilder.CreateTabl
11                  name: "Products",
12                  columns: table => new
13                  {
14                      Id = table.Column<i
110 %
Package Manager Console
Package source: All        Default proj
PM> Add-Migration "NewDatabase"
To undo this action, use Remove-Migration.
PM>
```



- Main part

```
0 references
static void Main(string[] args)

{

    using (Mycontext mycon = new Mycontext())
    {
        mycon.Database.EnsureCreated();
        var employee1 = new Emplyee
        {
            SSN = 123456,
            Name = "John Doe",
            DOB = new DateTime(1985, 6, 1),
            Gender = "Female",
            Salary = 50000,

        };

        var department1 = new Departments { Number = 1, Name = "IT" };
        mycon.Employees.Add(employee1);
        mycon.Departments.Add(department1);
        mycon.SaveChanges();
    }


}


}
```


Add Connection
Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.
Data source:
Microsoft SQL Server (SqlClient)
Change...
Server name:
RAWAN-ELLSAYED\SQLEXPRESS
Refresh
Log on to the server
Authentication:
Windows Authentication
User name:
Password:
Encrypt:
Microsoft Visual Studio
Test connection succeeded.
OK
Connect to a database
Select or enter a da
simpleDB
Attach a database file:
Browse...
Logical name:
Advanced...
Test Connection
OK
Cancel

Finally, Make a test connection to the server.

And