**Snippet 06 — What is a Class in C#?**

* **Code Recap**

public class MyClass

{

    protected void MyMethod()

    {

        Console.WriteLine("Hello");

    }

}

1. **Working Theory (keywords & concepts)**

**Definition**

* A **class** is a **blueprint** for creating objects.
* It groups **data (fields/properties)** and **behavior (methods)** into one unit.
* Most common way to model real-world concepts in C#.

**Example:**

public class Car

{

    public string Model;

    public void Drive() => Console.WriteLine($"{Model} is driving");

}

class Program

{

    static void Main()

    {

        var car = new Car { Model = "Honda" };

        car.Drive();  // Honda is driving

    }

}

**Key Properties of Classes**

1. **Reference Type**
   * Stored on the **heap**.
   * Variables hold references (pointers).
2. **Default Access**
   * If you don’t specify, a class is internal.
3. **Supports Inheritance**
   * One class can inherit from another.

public class Animal { public void Eat() => Console.WriteLine("Eating"); }

public class Dog : Animal { public void Bark() => Console.WriteLine("Bark"); }

class Program

{

    static void Main()

    {

        var dog = new Dog();

        dog.Bark();

        dog.Eat();

    }

}

1. **Can Implement Interfaces**
   * A class can promise to implement specific methods.

interface IMovable { void Move(); }

public class Car : IMovable { public void Move() => Console.WriteLine("Car moves"); }

class Program

{

    static void Main()

    {

        // 1. Call via class reference => Class reference (Car)

        Car car = new Car();

        car.Move();

        // 2. Call via interface reference => Interface reference (IMovable)

        IMovable moveable = new Car();

        moveable.Move();

    }

}

1. **Access Modifiers**
   * Same list you saw: public, private, protected, internal, file.
   * Control who can see/use the class and its members.
2. **Constructors & Destructors**
   * constructor: runs when you new an object.
   * destructor: rarely used, runs when object is garbage collected.

class Person

{

    // Variable

    public string Name;

    // Constructor

    public Person(string name)

    {

        Name = name;

    }

}

class Program

{

    static void Main()

    {

        var p = new Person("Anas");

        Console.WriteLine(p.Name);

    }

}

1. **Polymorphism**
   * Child classes can **override** base class methods.

class Animal { public virtual void Speak() => Console.WriteLine("This is original me."); }

class Dog : Animal { public override void Speak() => Console.WriteLine("I override the context of this method."); }

class Program

{

    static void Main()

    {

        Animal animal = new Animal();

        animal.Speak();

        Dog dog = new Dog();

        dog.Speak();

    }

}

**✅ Classes vs Structs vs Records (Quick Recap)**

* **Class** → reference type, best for complex models, supports inheritance.
* **Struct** → value type, lightweight, no inheritance.
* **Record** → reference type but with **value equality** and immutability focus.

**Practical Mini Exercise with new things.**

1. **Define a new file name as “Student.cs”:**

class Student

{

    public string name;

    public int age;

    public void Introduce()

    {

        Console.WriteLine($"I am {name} and I am {age} years old.");

    }

}

1. **Use it in Program.cs**

using System;

class Program

{

    static void Main()

    {

        var introduce = new Student { name = "Anas", age = 21 };

        introduce.Introduce();

    }

}

1. **Output**

I am Anas and I am 21 years old.

✅ so yes — snippet 05/06 started talking about **access modifiers on classes**, but the transcript/video also expands on **full class behavior**: inheritance, abstract, interfaces, constructors, etc.