**Snippet 04 — Records, Structs, and Classes**

* **Code Recap**

record MyRecord(string fieldA, string fieldB);

struct MyStruct

{

    public string FieldA;

    public string FieldB;

}

class MyClass

{

}

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

**1. record**

* Introduced in **C# 9**.
* Designed for **data-centric** types → holds values rather than behavior.
* Properties are **immutable by default** (you can’t change them once created, unless you use init or explicitly make them mutable).
* Provides **value-based equality**: two records with the same values are considered equal, unlike classes which check by reference.

**Example:**

record Person(string Name, int Age);

class Program

{

    static void Main()

    {

        var p1 = new Person("Anas", 22);

        var p2 = new Person("Anas", 22);

        Console.WriteLine(p1 == p2); // True ✅ (compares values)

    }

}

**2. struct**

* A **value type** (stored on the **stack** instead of the heap).
* Best for **small, lightweight data** that doesn’t need inheritance.
* Copied **by value** when passed around.

Example:

struct Point

{

    public int X;

    public int Y;

}

class Program

{

    static void Main()

    {

        var p1 = new Point { X = 5, Y = 10 };

        var p2 = p1;   // copy created

        p2.X = 20;

        Console.WriteLine(p1.X); // 5 ✅ unchanged

        Console.WriteLine(p2.X); // 20

    }

}

**3. class**

* The most common type in C#.
* A **reference type** (stored on the **heap**, variables hold references).
* Supports **inheritance** and **polymorphism**.
* Copied **by reference** → two variables can point to the same object.

Example:

class Student

{

    public string Name;

}

class Program

{

    static void Main()

    {

        // Example 01:

        var s1 = new Student { Name = "Ali" };

        var s2 = s1;   // reference copy

        s2.Name = "Sara";

        Console.WriteLine(s1.Name); // Sara ❌ (both point to same object)

        // Example 02: // Not coping any reference.

        var s3 = new Student { Name = "Ali" };

        var s4 = new Student { Name = "Ali" };

        s4.Name = "Sara";

        Console.WriteLine(s3.Name);

        Console.WriteLine(s4.Name);

        // Now, Both have different object.

    }

}

**✅ Summary Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Record | Struct | Class |
| Type | Reference type (with value semantics) | Value type (stack) | Reference type (heap) |
| Equality | By value (default) | By value | By reference (default) |
| Immutability | Immutable by default | Mutable unless coded otherwise | Mutable by default |
| Inheritance | No (but can implement interfaces) | No inheritance | Supports inheritance |
| Best Use | Data transfer, DTOs, immutables | Lightweight data, coordinates | Business logic, complex models |

**Practical (from scratch)**

1. **Create project**

* dotnet new console -n Snippet04Demo
* cd Snippet04Demo

1. **Program.cs**

using System;

record MyRecord(string FieldA, string FieldB);

struct MyStruct

{

    public string FieldA;

    public string FieldB;

}

class MyClass

{

    public string FieldA;

    public string FieldB;

}

class Program

{

    static void Main()

    {

        // Record: Values Equality => Case 01

        var r1 = new MyRecord("Car1", "Car2");

        var r2 = new MyRecord("Car1", "Car2");

        Console.WriteLine(r1 == r2);

        // Record: Values Equality => Case 02

        // var r3 = new MyRecord("Car1", "Car2");

        // var r4 = r3;

        // r2.FieldA = "Car3";

        // Console.WriteLine(r1 == r2); // Terminal Error because can not assign value.

        // Struct: Copy Value

        var s1 = new MyStruct { FieldA = "X", FieldB = "Y" };

        var s2 = s1;

        s2.FieldA = "Changed";

        Console.WriteLine(s1.FieldA); // X

        Console.WriteLine(s2.FieldA); // Changed

        // Class: Reference copy

        var c1 = new MyClass { FieldA = "one", FieldB = "two" };

        var c2 = c1;

        c2.FieldA = "changed";

        Console.WriteLine(c1.FieldA);

        Console.WriteLine(c2.FieldA);

    }

}

1. **Expected Output**

True

X

Changed

changed

changed

**🔧 Extras**

* **Record deconstruction**:

record MyRecord(string FieldA, string FieldB);

class Program

{

    static void Main()

    {

        var (a, b) = new MyRecord("Hello", "World");

        Console.WriteLine($"{a}, {b}");

    }

}

* **When to use what**:
  + Use **record** for DTOs, immutable models, serialization.
  + Use **struct** for performance-critical, small data types (points, colors, coordinates).
  + Use **class** for business/domain models, where inheritance and polymorphism matter.

✅ snippet 04 explained: now you know the difference between **records, structs, and classes** — one of the most important concepts in C#.