**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);

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;

}

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;

}

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

var s2 = s1; // reference copy

s2.Name = "Sara";

Console.WriteLine(s1.Name); // Sara ❌ (both point to same 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**
2. dotnet new console -n Snippet04Demo
3. cd Snippet04Demo
4. **Program.cs**
5. using System;
6. record MyRecord(string FieldA, string FieldB);
7. struct MyStruct
8. {
9. public string FieldA;
10. public string FieldB;
11. }
12. class MyClass
13. {
14. public string FieldA;
15. public string FieldB;
16. }
17. class Program
18. {
19. static void Main()
20. {
21. // Record: value equality
22. var r1 = new MyRecord("A", "B");
23. var r2 = new MyRecord("A", "B");
24. Console.WriteLine(r1 == r2); // True
25. // Struct: value copy
26. var s1 = new MyStruct { FieldA = "X", FieldB = "Y" };
27. var s2 = s1;
28. s2.FieldA = "Changed";
29. Console.WriteLine(s1.FieldA); // X
30. Console.WriteLine(s2.FieldA); // Changed
31. // Class: reference copy
32. var c1 = new MyClass { FieldA = "One", FieldB = "Two" };
33. var c2 = c1;
34. c2.FieldA = "Changed";
35. Console.WriteLine(c1.FieldA); // Changed
36. }
37. }
38. **Expected Output**
39. True
40. X
41. Changed
42. Changed

**🔧 Extras**

* **Record deconstruction**:
* 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#.