**When to Use a struct Instead of a class or record**

| Criteria | Use struct | Use class or record |
|---|---|---|
| **Type Semantics** | You want **value semantics** (copy by value) | You need **reference semantics** (shared object instances) |
| **Size and Performance** | Type is **small** (typically < 16 bytes) and **performance-critical** | Type is **large**, or allocation/copy cost is acceptable |
| **Memory Allocation** | Prefer **stack allocation** (or inline in other structs) | Requires **heap allocation** (e.g., shared across methods or long-lived) |
| **Mutability** | Should be **immutable** (fields set via constructor only) | Needs **mutability** or more flexible object lifecycle |
| **Equality Behavior** | Want **value-based equality** by default | Need **reference equality** (class) or auto-generated **value equality** (record) |
| **Inheritance / Polymorphism** | Not required | Required (inheritance, virtual methods, polymorphism) |
| **Interop / Layout Control** | Interop with native code (e.g., P/Invoke) or precise memory layout needed | Not interop-focused |
| **Lifespan / Scope** | Object is **short-lived** or confined in scope (e.g., local calculations) | Object is **long-lived**, shared, or part of a complex object graph |
| **Use Case Examples** | Point, Vector2D, Color, DateTime | User, Customer, Order, ApiResponse, Person |

---

**Summary:**

- **Use struct**: small, immutable, performance-critical, no inheritance needed.
- **Use class**: large, mutable, object identity or OOP features needed.
- **Use record**: immutable reference type with built-in value equality (great for data models).