**Einführung in Records**

**Warum Records?**

* **Problemstellung:** Klassen und Strukturen eignen sich nicht ideal für **unveränderliche** Datenobjekte.
* **Lösung:** Das Schlüsselwort record vereinfacht das Arbeiten mit **immutable Daten**.

**Grundlagen: Syntax und Verhalten**

**Beispiel:** Vergleich zwischen class, struct und record

public **class** PersonClass

{

public string Name { get; set; }

public int Alter { get; set; }

}

public **struct** PersonStruct

{

public string Name { get; set; }

public int Alter { get; set; }

}

public **record** PersonRecord

{

public string Name { get; init; }

public int Alter { get; init; }

}

**Wertsemantik vs. Referenzsemantik**

* **Klassen (class)**: Referenztyp → Equals vergleicht **Referenzen**
* **Strukturen (struct)**: Werttyp → Equals vergleicht **Inhalte**
* **Records (record)**: Referenztyp, aber Equals vergleicht standardmäßig die **Werte** der Properties!

**Beispiel:**

var p1 = new PersonRecord("Max", 30);

var p2 = new PersonRecord("Max", 30);

Console.WriteLine(p1 == p2); // True, weil Werte verglichen werden!

**Der with-Ausdruck**

Records sind **unveränderlich** (immutable), aber können mit **with** kopiert und verändert werden:

var p1 = new PersonRecord("Max", 30);

var p2 = p1 with { Alter = 31 };

Console.WriteLine(p1); // PersonRecord { Name = Max, Alter = 30 }

Console.WriteLine(p2); // PersonRecord { Name = Max, Alter = 31 }

**Positionale Records**

Eine kompaktere Schreibweise für Datenklassen:

public record Auto(string Marke, string Modell, int Baujahr);

var auto = new Auto("VW", "Golf", 2022);

Console.WriteLine(auto.Marke); // VW

**Vererbung mit Records**

public record Fahrzeug(string Marke);

public record Auto(string Marke, string Modell) : Fahrzeug(Marke);

* Records unterstützen **Vererbung** wie Klassen.
* Equals und ToString() werden entsprechend übernommen.

**Typische Einsatzgebiete von Records**

**1. Datenmodelle (DTOs, Immutable Models)**

* Wenn Sie Daten **transportieren** oder **abbilden** wollen, ohne Logik im Vordergrund:
  + API-Requests/Responses
  + Ergebnisse von Datenbankabfragen
  + Konfigurationsobjekte

Beispiel:

public record PersonDto(string Name, int Alter);

* Kompakt, immutable, Wertgleichheit – perfekt für Datenübertragung.

**2. Immutable Objekte**

* Records sind standardmäßig **unveränderlich** (Properties mit init).
* Änderungen laufen über **with-Ausdrücke** → dabei wird ein neues Objekt erzeugt.

**Beispiel**:

var p1 = new PersonDto("Anna", 30);

var p2 = p1 with { Alter = 31 }; // neues Objekt

→ Ideal für funktionale Programmiermuster, wo Zustände nicht verändert, sondern neu erzeugt werden.

**3. Wertbasierte Gleichheit**

* Wenn es nicht darauf ankommt, **ob es die gleiche Instanz ist**, sondern ob die **Inhalte gleich** sind.

**Beispiel**:

var a = new Auto("VW", "Golf", 2022);

var b = new Auto("VW", "Golf", 2022);

Console.WriteLine(a == b); // True (bei class wäre das False)

→ Sehr nützlich für Vergleiche in Business-Logik.

**4. Pattern Matching**

* Records arbeiten super mit **Switch Expressions** und **Deconstruction**.

**Beispiel**:

public record Fahrzeug(string Marke, string Modell);

public record Auto(string Marke, string Modell, int Baujahr) : Fahrzeug(Marke, Modell);

Fahrzeug fzg = new Auto("VW", "Golf", 2022);

string info;

switch (fzg)

{

case Auto(var marke, var modell, var baujahr):

info = $"{marke} {modell} ({baujahr})";

break;

case Fahrzeug(var marke, var modell):

info = $"{marke} {modell}";

break;

default:

info = "Unbekannt";

break;

}

Console.WriteLine(info); // VW Golf (2022)

**5. Domänenmodellierung**

* In **DDD (Domain-Driven Design)** gibt es sogenannte **Value Objects** – also kleine Objekte, die allein durch ihre Werte definiert sind (z. B. Geldbetrag, Adresse).
* Records sind die perfekte Wahl dafür.

**Beispiel**:

public record Adresse(string Straße, string Stadt, string PLZ);

**Fazit**

Records sind am nützlichsten, wenn Sie:

* **Datenobjekte** definieren, die nicht verändert werden sollen.
* **Wertgleichheit** statt Referenzgleichheit brauchen.
* **Wenig Boilerplate** schreiben wollen (Equals, ToString, GetHashCode).
* Mit **Pattern Matching** oder **funktionalen Ansätzen** arbeiten

**Übungen**

**Aufgabe 1: Einfache Datenstruktur**

Definiere ein Record für ein **Produkt** mit Name, Preis und Kategorie.

**Aufgabe 2: Kopieren mit with**

Erstelle eine Instanz eines Records und erstelle eine leicht geänderte Kopie.

**Aufgabe 3: Vererbung mit Records**

Erstelle ein Basisklassen-Record Mitarbeiter und leite Manager mit zusätzlicher Abteilung davon ab.

**Zusammenfassung**

record für **immutable Datenstrukturen**  
with für einfache Modifikationen  
record hat **Wertsemantik**, obwohl es ein Referenztyp ist  
**Positionale Records** und **Vererbung** machen den Code kompakt