Naming & Snippets (Tab+Tab)

Typ	Beispiel (Pascal/camel)	Snippet	Nutzen / Syntax
Klasse	`public class Geraet`	`class`	`class Name { }`
Methode	`public void Berechne()`	`ctor`	`public Name() { }` (Konstr.)
Property	`public int Ram { get; }`	`prop`	`public T X { get; set; }`
Variable	`int count = 0;`	`cw`	`Console.WriteLine($"+{v}");`
Schleife	`foreach(var i in list)`	`foreach`	Listen/Arrays durchlaufen
Fehler	`try { ... }`	`try`	`try { } catch { }` (JSON/IO)

Vererbung & Konstruktoren (base)

```
public abstract class Geraet {
    public string SN { get; set; }
    public string Hersteller { get; set; }
    public Geraet(string sn, string h) { this.SN = sn; this.Hersteller = h; }
}

public class Computer : Geraet {
    public int RAM { get; set; }
    // base() reicht die Parameter an den Konstruktor der Elternklasse weiter
    public Computer(string sn, string h, int r) : base(sn, h) {
        this.RAM = r;
    }
}
```

Polymorphismus (virtual / override)

```
public class Kurs {
    public decimal Basis { get; set; }
    public virtual decimal Berechne(int t) => Basis + t;
}

public class OnlineKurs : Kurs {
    public override decimal Berechne(int t) {
        // Ruft die Logik der Elternklasse auf
        decimal gebuehr = base.Berechne(t);
        if (t < 5) { return gebuehr + 10m; }
        return gebuehr;
    }
}
```

Statische Member & Methoden (Klassen-Ebene)

Statische Dinge gehören der Klasse selbst. Man braucht kein Objekt (new), um sie zu nutzen.

```
public class Ticket {
    // Privates statisches Feld (Zähler für alle Instanzen)
    private static int _ticketCount = 0;

    public Ticket() { _ticketCount++; }
    // Statische Methode: Aufruf via 'Ticket.GetTicketCount()'
    public static int GetTicketCount() { return _ticketCount; }
}
```

Input, Menü & Datum (Prüfungsklassiker) | TryParse / DateTime

```
// 1. Robuster Input via TryParse (Verhindert Absturz)
public static int ReadInt(string prompt) {
    int res; Console.Write(prompt);
    while (!int.TryParse(Console.ReadLine(), out res)) {
        Console.Write("Fehler! " + prompt);
    }
    return res;
}

// 2. Datum & Zeit (Alter berechnen)
int alter = DateTime.Now.Year - baujahr;

// Common Test-Use-Cases
string input = "Max;Muster;25"; string[] parts = input.Split(';'); // Aufteilen
string text = "Hello World"; bool contains = text.Contains("World"); // Prüfen
int len = text.Length; // Länge
string userInput = "  Max "; string clean = userInput.Trim(); // Leerzeichen entfernen

DateTime geb = new DateTime(2000,5,15); int alter = DateTime.Now.Year - geb.Year; if(DateTime.Now.DayOfYear<geb.DayOfYear) alter--; // Alter
DateTime datum; DateTime.TryParse("15.08.2025", out datum); // Datum einlesen
TimeSpan diff = DateTime.Now - new DateTime(2025,1,1); int tage = diff.Days; // Tage Unterschied

List<int> zahlen = new List<int>{1,2,3,4,5,6}; var gerade = zahlen.Where(z=>z%2==0).ToList(); // Gerade Zahlen
List<string> words = new List<string>{"Haus","Auto","Baum","Straße"}; var kurz = words.Where(w=>w.Length<=4); // Wörter <=4 Buchstaben
```

Kapselung & Berechnete Properties

```
public class Produkt {
    public decimal Netto { get; set; }

    // Berechnete Property (Read-Only) in klassischer Schreibweise
    public decimal Brutto {
        get { return Math.Round(Netto * 1.081m, 2); }
    }

    // Full Property mit privatem Backing-Field für Validierung
    private int _lager;
    public int Lager {
        get { return _lager; }
        set {
            if (value < 0) { _lager = 0; }
            else { _lager = value; }
        }
    }
}
```

Use Case: Suchen & Sortieren in Listen

```
public void Suche(List<Geraet> liste) {
    Console.Write("Hersteller suchen: ");
    string suche = Console.ReadLine().ToLower();

    foreach (var g in liste) {
        if (g.Hersteller.ToLower().Contains(suche)) {
            Console.WriteLine("Gefunden: " + g.SN);
        }
    }
}
```

Dateien & JSON (System.Text.Json)

```
using System.IO;
using System.Text.Json;

// Speichern
string jsonText = JsonSerializer.Serialize(meineListe);
File.WriteAllText("daten.json", jsonText);

// Laden
if (File.Exists("daten.json")) {
    string inhalt = File.ReadAllText("daten.json");
    var liste = JsonSerializer.Deserialize<List<Geraet>>(inhalt);
}
```

Checkliste Keywords & Sichtbarkeit

- **public**: Überall sichtbar (Schnittstelle).
- **private**: Nur in dieser Klasse sichtbar (Standard für Felder).
- **protected**: In dieser Klasse und allen Unterklassen (Kindern) sichtbar.
- **abstract**: Nur Elternklasse, kein `new()`.
- **virtual** / **override**: Basis erlaubt Anpassung / Subklasse führt sie aus.
- **base**: Ruft Logik oder Konstruktor der Elternklasse auf.
- **static**: Gehört der Klasse, existiert global nur 1x.
- **readonly**: Wert kann nur bei Erstellung oder im Konstruktor gesetzt werden.
- **this**: Bezieht sich auf das aktuelle Objekt (vermeidet Namenskonflikte).
- **decimal**: Pflicht für Geld (Suffix `m`).

Interfaces (Der "Vertrag")

Wird genutzt, wenn Klassen gleiches Verhalten haben (z.B. `Druckbar`), aber nicht verwandt sind.

```
public interface IDruckbar {
    void Drucken(); // Nur Signatur, kein Body!
}

public class Bericht : IDruckbar {
    public void Drucken() => Console.WriteLine("Drucke Bericht...");
}
```

Listen-Logik & LINQ

```
var defekt = lab.Where(g => g.Zustand == Zustand.Defekt).ToList();
var search = lab.FirstOrDefault(g => g.SN == "SN123");
decimal total = kurse.Sum(k => k.Berechne(10));
```