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; }
    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;
}

public class OnlineKurs : Kurs {
    public override decimal Berechne(int t) {
        decimal geb = base.Berechne(t); // Basis-Logik nutzen
        return (t < 5) ? geb + 10m : geb; // Kleingruppenzuschlag
    }
}
```

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...");
}
```

Input, Menü & Datum (Prüfungsklassiker)

```
// 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;

// 3. Menü-Struktur (Main)
bool exit = false;
while (!exit) {
    int wahl = ReadInt("1: Neu, 2: Liste, 3: Filter, 0: Exit: ");
    switch (wahl) {
        case 1: AddGeraet(); break;
        case 3: FilterNachHersteller(); break;
        case 0: exit = true; break;
        default: Console.WriteLine("Ungültig"); break;
    }
}
```

Use Case: Suchen & Filtern in Listen

```
public void FilterNachHersteller(List<Geraet> liste) {
    Console.Write("Welcher Hersteller? ");
    string suche = Console.ReadLine().ToLower();
    var treffer = liste.Where(g => g.Hersteller.ToLower().Contains(suche)).ToList();
    foreach (var g in treffer) Console.WriteLine($"{g.SN}: {g.Hersteller}");
}

// Bonus: Schnelle Testdaten-Liste erstellen
List<Geraet> lab = new List<Geraet> {
    new Computer("SN1", "Dell", 16),
    new Computer("SN2", "HP", 8)
};
```

Praktische String-Tricks & Sortierung

```
// 1. Tabellen-Optik (linksbündig -15 Zeichen)
string line = $"{g.GetType().Name,-15} | {g.SN,-10}";

// 2. Sortieren (LINQ)
var sortiert = liste.OrderBy(g => g.Baujahr).ToList(); // Aufsteigend
var teuerste = liste.OrderByDescending(g => g.Preis).First(); // Höchster Wert

// 3. Null-Check & Runden
string h = input ?? "Unbekannt";
decimal rund = Math.Round(preis, 2);
```

Dateien (JSON Save/Load)

```
using System.IO;
// Speichern: Liste -> JSON -> Datei
File.WriteAllText("daten.json", JsonSerializer.Serialize(liste));

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

Listen-Logik & LINQ

```
using System.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));
```

JSON, Singleton & Static

```
// Singleton: Globaler Manager (Zentraler Zugriff)
public class App {
    private static App _inst;
    private App() { }
    public static App Get() => _inst ??= new App();
}

// Static Counter: Auto-Increment ID
private static int _cnt = 1000;
public int ID { get; } = ++_cnt;
```

Checkliste Keywords

- **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.
- **decimal**: Pflicht für Geld (Suffix `m`).
- **this**: Bezieht sich auf das aktuelle Objekt (vermeidet Namenskonflikte).