**Interfaces (Schnittstellen)**

**Interfaces** kann man als **Baupläne** für die **Teil- oder Ganzstruktur** einer Klasse bezeichnen.

public class VipReservation: Reservation, IWithSurcharge, ISpecialServices

{

public bool HasWelcomeDrink { get; private set; } = true;

public decimal? Surcharge { get; set; }

public IEnumerable<string>? SpecialServices { get; set; }

}

**Was passiert hier?**

* **VipReservation** ist eine **spezialisierte Version** von **Reservation** (VipReservation **leitet** von der Basisklasse **Reservation** ab, sie **erbt** davon.).
* Sie **implementiert** aber auchzwei Interfaces:
  + IWithSurcharge → definiert, dass ein Zuschlag berechnet werden kann.
  + ISpecialServices → definiert, dass spezielle Zusatzleistungen möglich sind.

Dadurch erhalten alle Klassen, die diese Interfaces implementieren, die Möglichkeit, **Zuschläge** und/oder **Zusatzleistungen** anzubieten – unabhängig davon, wie sie intern aufgebaut sind.

**Vorteile von Interfaces**

* **Lose Kopplung:** Interfaces ermöglichen die Erweiterung von Klassen ohne Abhängigkeit vom konkreten Typ.
* **Modularität:** Du kannst Features wie **Zusatzleistungen**, **Zuschläge**, **Rabatte**, **Service-Level** usw. als Interfaces definieren und bei Bedarf kombinieren.

|  |  |
| --- | --- |
| Konzept | Beispiel |
| Mehrfach-Interface-Implementierung | IWithSurcharge, ISpecialServices |
| Property-basierte Interfaces | keine Methoden, nur get/set-Properties |
| Erweiterbarkeit ohne Vererbung | andere Klassen können die Interfaces auch nutzen |

**Zusammengefasst**

VipReservation wird durch diese Interfaces **flexibler** und **klarer strukturiert**: sie **"verpflichtet"** sich, **Zuschläge** und **Extraservices** bereitzustellen, ohne dass diese Logik in der Basisklasse Reservation verankert sein muss.