zurück

### Aufgabe: Klassendiagramm entwerfen

Du sollst ein System für eine Bibliothek modellieren. Folgende Anforderungen sind gegeben:

- Eine Bibliothek verwaltet mehrere Bücher.
- Jedes Buch hat einen Titel, Autor, ISBN und Status (verfügbar/verliehen).
- Ein Benutzer kann Bücher ausleihen.
- Die Klasse Ausleihe dokumentiert, welches Buch wann von welchem Benutzer ausgeliehen wurde.

### Aufgabenstellung:

1. Entwirf ein UML-Klassendiagramm mit allen relevanten Klassen, Attributen und Beziehungen.
2. Kennzeichne die Sichtbarkeit der Attribute.
3. Welche Klasse sollte Methoden enthalten – z. B. ausleihen() oder zurückgeben()?

### Lösung:

classDiagram class Bibliothek { name buchHinzufuegen() benutzerRegistrieren() ausleihen() zurueckgeben() findeBuch() } class Buch { titel autor isbn status istVerfuegbar() } class Benutzer { benutzerId name anzahlAktiverAusleihen() } class Ausleihe { ausleihDatum faelligAm rueckgabeDatum istUeberfaellig() dauerInTagen() } Bibliothek "1" o-- "0.." Buch Bibliothek "1" o-- "0.." Benutzer Bibliothek "1" o-- "0..*" Ausleihe Ausleihe "1" --> "1" Buch Ausleihe "1" --> "1" Benutzer Benutzer "1" --> "0..*" Ausleihe

---

### Aufgabe : Beziehungstypen erkennen

Gegeben sind folgende Klassen:

- Rechnung
- Position
- Produkt

Beschreibung:

- Eine Rechnung besteht aus mehreren Positionen.
- Jede Position bezieht sich auf genau ein Produkt.
- Ein Produkt kann auf mehreren Rechnungen erscheinen.

### Aufgabenstellung:

1. Zeichne das Klassendiagramm mit den passenden Beziehungstypen.
2. Welche Beziehung besteht zwischen Rechnung und Position? Aggregation oder Komposition?
3. Wie würde sich die Modellierung ändern, wenn Position ohne Rechnung nicht existieren kann?

### Lösung

### 1) Klassendiagramm (mit Beziehungstypen & Multiplizitäten)

**Empfohlen (fachlich korrekt: Komposition zwischen Rechnung und Position):**

classDiagram class Rechnung class Position class Produkt Rechnung "1" *-- "1..*" Position Position "0..*" --> "1" Produkt

Hinweis: Dadurch ergibt sich indirekt eine **n:m-Beziehung** zwischen *Rechnung* und *Produkt* (über *Position*).

**2) Aggregation oder Komposition?**

**Komposition.** Positionen sind Teil der Rechnung (Lebenszyklus gebunden); löscht man die Rechnung, verschwinden die Positionen.

**3) Wenn Position ohne Rechnung nicht existieren kann …**

… dann ist genau das die **Komposition** (gefüllter Diamant). Falls du zuvor Aggregation modelliert hattest, tausche einfach o- - gegen *- - aus:

**Alternative (Aggregation, nur falls Positionen eigenständig existieren dürften):**

classDiagram class Rechnung class Position class Produkt Rechnung "1" o-- "1..*" Position Position "0..*" --> "1" Produkt

---

## Aufgabe 06.10.2025

classDiagram class Abteilung { +abteilungsId : int +name : string } class Mitarbeiter { +mitarbeiterId : int +vorname : string +nachname : string +email : string } class Techniker { +technikerId : int +skillset : string +telefonDurchwahl : string } class Supportanfrage { +ticketNr : string +status : string +erstelltAm : Date +kurzbeschreibung : string } Abteilung "1" o-- "0..*" Mitarbeiter : umfasst Mitarbeiter "1" --> "0..*" Supportanfrage : erstellt Supportanfrage "1" --> "0.." Techniker : zugewiesenAn Mitarbeiter <|-- Techniker

# b) Assoziation vs. Aggregation (am Ticketsystem erklärt)

- **Assoziation**: Eine lose Beziehung zwischen zwei Klassen ohne „Ganze-Teil“-Semantik.
  - Beispiel*: **Mitarbeiter — erstellt — Supportanfrage**. Ein Mitarbeiter kann viele Anfragen erstellen; die Objekte existieren unabhängig voneinander.* **Aggregation** (leere Raute „o–“): „Ganze-Teil“ mit *geteilter* Lebensdauer (Teil kann auch ohne Ganzes existieren).
  - Beispiel*: **Abteilung o– Mitarbeiter**. Eine Abteilung *umfasst* Mitarbeiter, aber Mitarbeiter können unabhängig existieren bzw. in eine andere Abteilung wechseln. (Keine *Komposition*, weil das Leben des Mitarbeiters nicht von der Abteilung abhängt.)

# c) 1:n oder m:n zwischen „Mitarbeiter“ und „Supportanfrage“?

- **Begründet 1:n**: In den Anforderungen steht, dass *jeder Mitarbeiter mehrere Supportanfragen stellen kann* und eine Supportanfrage von **einem** Mitarbeiter stammt (Ersteller). Damit: **Mitarbeiter 1 — n Supportanfrage**.
- **Wann m:n?** Nur wenn das Domänenmodell erlauben würde, dass **mehrere** Mitarbeiter gemeinsam als Ersteller derselben Anfrage gelten (z. B. Co-Ersteller oder Ticket-Übernahme als „Erstellerrolle“), was hier **nicht** gefordert ist. Deshalb ist **1:n** korrekt und einfacher.