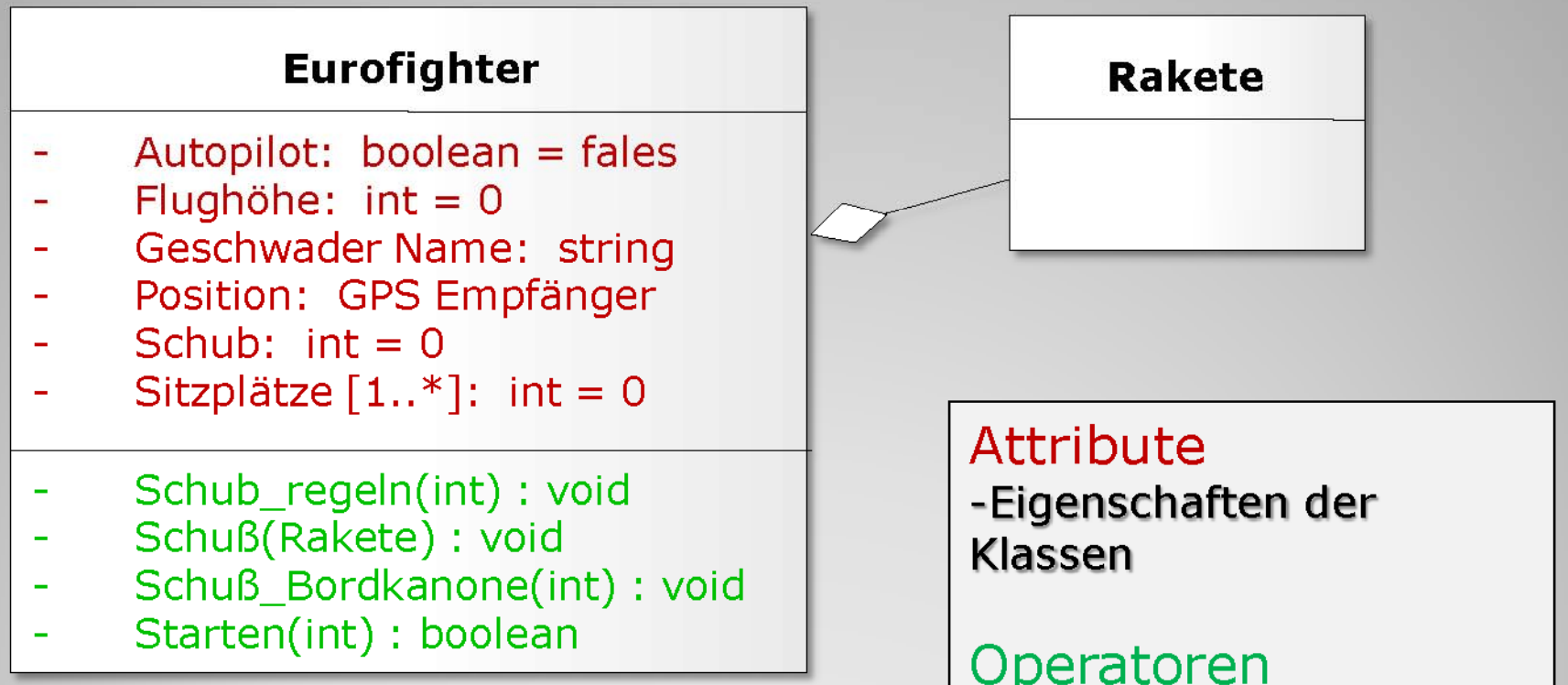
- Aufbau eines Klassendiagramm
- Was ist Kohäsion?
- Was ist Kopplung?

Was erwartet euch?



Attribute
-Eigenschaften der Klassen

Operatoren
- Mögliche Funktionen (Methoden)

Klassendiagramme

Schematischer Aufbau einer Attributdeklaration (UML-Standard):

<Sichtbarkeit> <Name> <Kardinalität> : <Typ> {<Eigenschaft>} = <Ausdruck>

Sichtbarkeit: definiert Sichtbarkeit eines Attributs außerhalb der Klasse

- ✓ public + für alle sichtbar und benutzbar
- ✓ protected # Klasse selbst und ihre Unterklassen haben Zugriff
- ✓ private - nur Klasse selbst hat Zugriff
- ✓ package ~ nur Klassen im gleichen Paket haben Zugriff auf das Attribut

Name: Attributbezeichnung, die innerhalb der Klasse eindeutig ist

Kardinalität mit Aufbau [<untere Grenze> .. <obere Grenze>]

- ✓ [0..1] optionales Attribut mit erlaubtem "null"-Wert
- ✓ [0..*] mengenwertiges Attribut (auch leere Menge)
- ✓ [m..n] Attributmenge mit m bis n Elementen

Klassendiagramme

Schematischer Aufbau einer Attributdeklaration (UML-Standard):

<Sichtbarkeit> <Name> <Kardinalität> : <Typ> {<Eigenschaft>} = <Ausdruck>

Typ eines Attributs:

- ✓ in Programmiersprache definierter Typ wie int, float, …
- ✓ Name einer „Hilfs"-Klasse für Attributwerte, die (einfache) Objekte sind

Menge von Attribut-Eigenschaften wie etwa:

- ✓ **frozen:** nach erster Wertzuweisung nicht mehr änderbar
- ✓ **addonly:** nur neue Werte zu Attributmenge hinzufügen

Ausdruck = Rechenvorschrift für Attribut:

- ✓ bei normalen Attributen gibt der Ausdruck einen Initialwert an
- ✓ bei abgeleiteten Attributen die Berechnungsvorschrift

Klassendiagramme

Schematischer Aufbau einer Operatordeklaration (UML-Standard):

<Sichtbarkeit> <Name> <Parameterliste> : <Rückgabetyp>

Sichtbarkeit: wie bei Attributen

Name: innerhalb der Klasse eindeutig Name

Parameterliste: vorher definierte Parameter

Rückgabetyp: Typ des Ergebniswerts der Operation (Funktion); nicht jede Operation muss einen Rückgabewert besitzen

Klassendiagramme

Beziehungen der objektorientierten Programmierung

Assoziation

Aggregation

Komposition

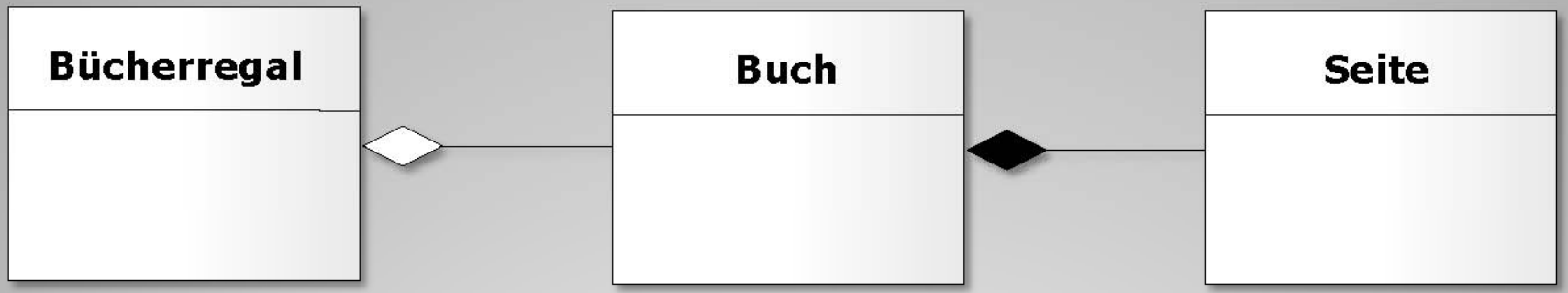
Assoziation

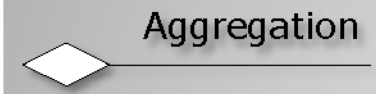
- ✓ einfachste Beziehung zwischen Klassen
- ✓ mit einem einfachen Strich symbolisiert
- ✓ Beziehung voneinander unabhängig
- ✓ sind gleichberechtigt
- ✓ ein Objekt einer Klasse zerstört, existiert die andere Klasse unabhängig weiter

Klassendiagramme

Beziehungen der objektorientierten Programmierung

Aggregation und Komposition



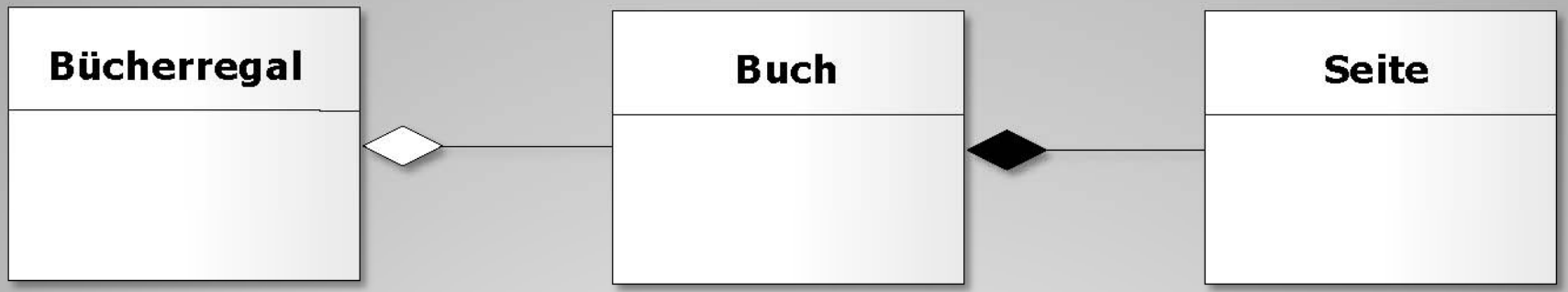


- ✓ häufig verwendete Beziehungsform
- ✓ eine Klasse n Klassen übergeordnet ist
- ✓ und einbindet
- ✓ die Bindung ist fest
- ✓ Objekte der untergeordneten Klassenkönnen ohne die der übergeordneten Klasse existieren

Klassendiagramme

Beziehungen der objektorientierten Programmierung

Aggregation und Komposition



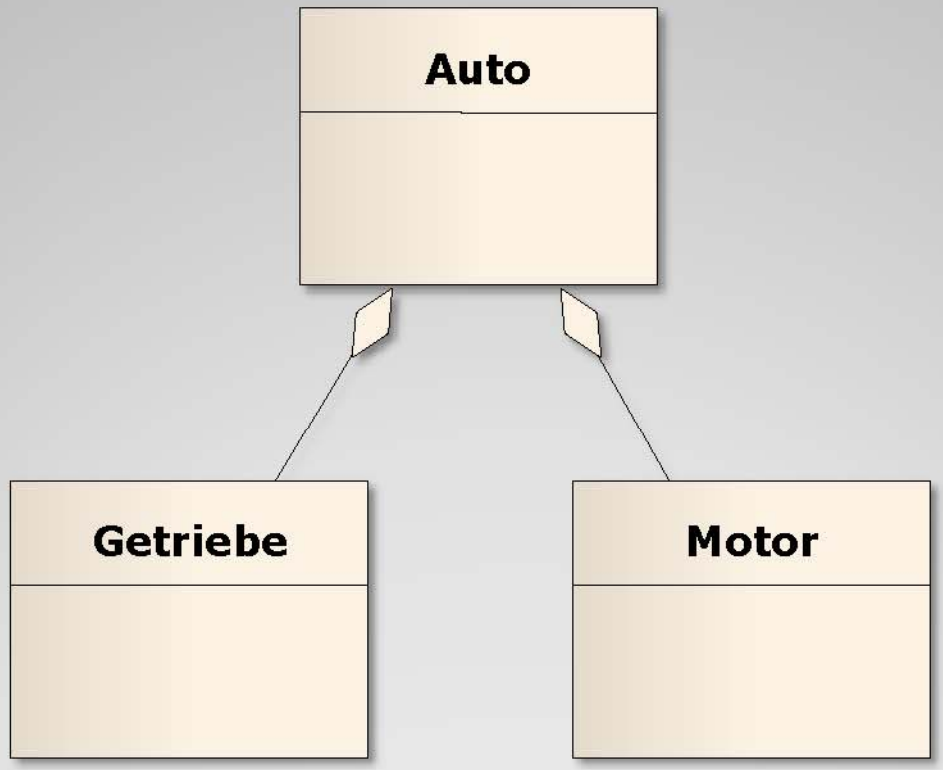
- ✓ ebenfalls eine feste Bindung
- ✓ übergeordneten Klasse n untergeordneten Klassen
- ✓ untergeordneten Klassen kann nicht ohne die der übergeordneten Klassen existieren
- ✓ übergeordneten Klasse zerstört, sind auch die untergeordneten Objekte zerstört

die Raute liegt immer auf der Seite der übergeordneten Klasse

Klassendiagramme

Beziehungen der objektorientierten Programmierung

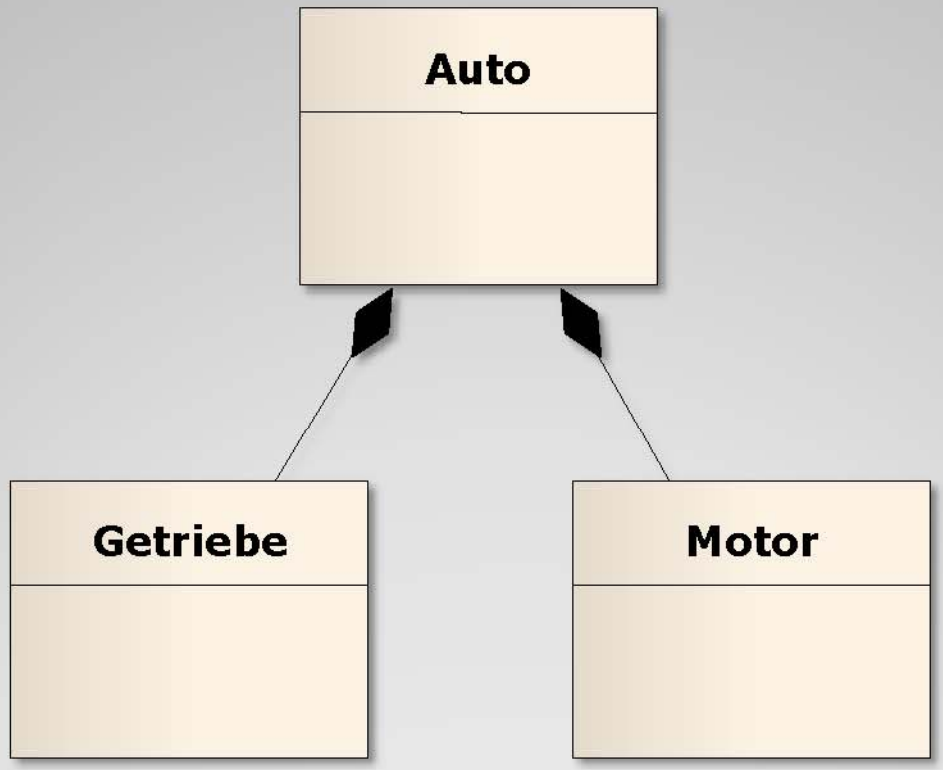
Aggregation vs Komposition – Interpretationssache



Klassendiagramme

Beziehungen der objektorientierten Programmierung

Aggregation vs Komposition – Interpretationssache



Klassendiagramme

Beziehungen der objektorientierten Programmierung

Vererbung

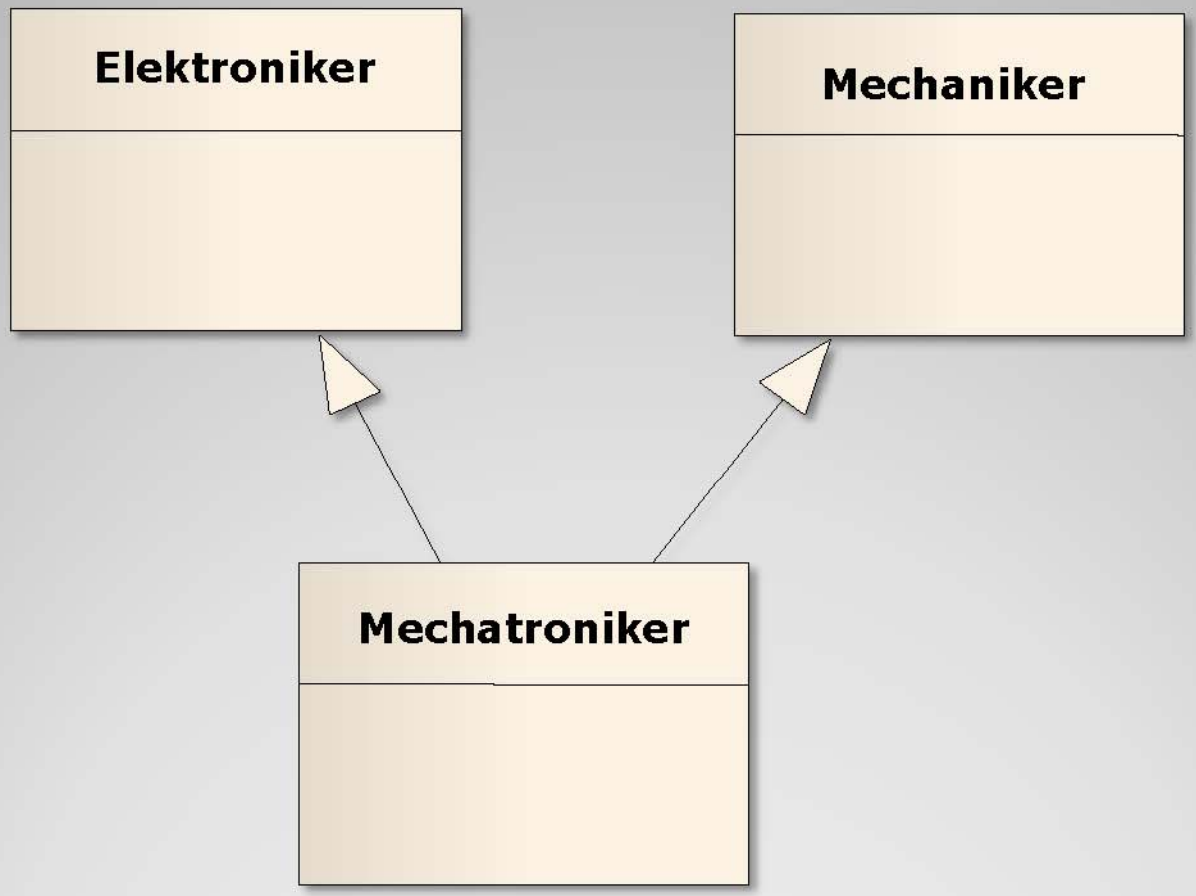
- ✓ Beziehung der Klassen bzw. der Klassenobjekte untereinander
- ✓ Datentypen Klassen können so “Eltern” bzw. “Kinder” haben
- ✓ Eigenschaften (Attribute) und Methoden an ihre Kinder **vererben**
- ✓ gesamten Operationen werden vererbt
- ✓ **Implementierungsvererbung**

- ✓ eine Vererbung wird mit Pfeilen dargestellt
- ✓ der Pfeil zeigt vom Erben zum Vererber
- ✓ **Mehrfachvererbung**, wenn eine Klasse von mehr als einer Klasse erbt

Klassendiagramme

Beziehungen der objektorientierten Programmierung

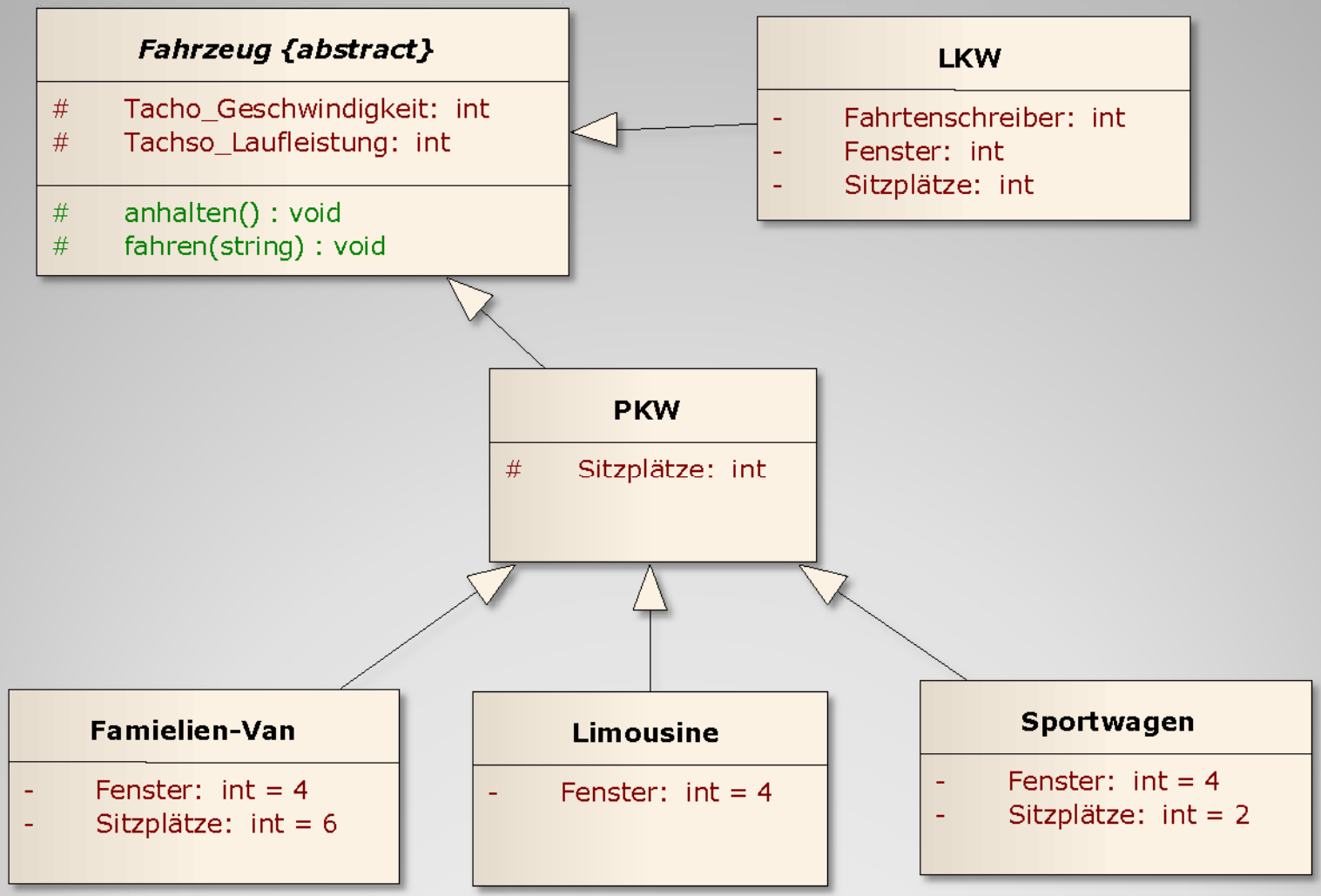
Beispiel an Hand der Mehrfachvererbung:



Klassendiagramme

Beziehungen der objektorientierten Programmierung

Beispiel an Hand der Mehrfachvererbung:



Klassendiagramme

Beziehungen der objektorientierten Programmierung

Kopplung und Kohäsion

Ausgangssituation

Klassen sollten, besonders für größere Projekte, gut strukturiert und auch für andere Entwickler leicht zu verstehen sein. Fehler bei der Programmierung werden oft durch eine Code-Duplizierung verursacht. Man kopiert einfach den Code aus einer Klasse in eine andere und vergisst, dass man andere Klassen ggf. auch anpassen muss.

Kopplung

- ✓ bezieht sich auf die Art, wie Klassen verknüpft sind
- ✓ Grad der Abhängigkeit zwischen Klassen
  - Bei hoher Kopplung verursachen Änderungen in einer Klasse auch Änderungen in einer anderen Klasse

X symptomatisch für schlechten Klassenentwurf ist:

- Klassen haben enge/hohe Kopplung!

- ✓ Besser: Möglichst lose Kopplung
  - Erreicht man durch weitgehend unabhängige Klassen

Kopplung – saubere Kapselung

- ✓ Kopplung reduzieren
- ✓ Verringerung der Kopplung:
  - ✓ Keine öffentlichen Datenfelder benutzen!
  - ✓ Sondierende Methode einführen, die Art und Zahl der Ausgänge kapselt

```
class Raum {
    private String beschreibung;
    private Raum nordausgang;
    private Raum suedausgang;
    private Raum ostausgang;
    private Raum westausgang;

// ...
 public Raum gibAusgang(String richtung)
  {
   if (richtung.equals("north")) {
   return nordausgang;
   }
   if (richtung.equals("east")) {
   return ostausgang;
   }
   // ...
   return null;
 }
}
```

Kohäsion

- ✓ beschreibt, wie gut eine Programmeinheit (Klasse, Methode) eine logische Aufgabe oder Einheit abbildet

- ✓ Hohe Kohäsion bedeutet:
  - Eine Klasse/Methode ist verantwortlich für genau eine wohl definierte Aufgabe oder Einheit

Guter Klassenentwurf hat hohen Grad an Kohäsion

- Es gibt immer nur eine Stelle, an der eine Aufgabe erledigt wird
- Bei Änderungen muss auch nur dort geändert werden

Kohäsion - Implizite Kopplung

- ✓ weitere Kopplungsart
- ✓ eine Klasse stützt sich auf interne Informationen einer anderen Klasse
- ✓ Abhängigkeit ist aber nicht erkennbar

Problem:

- ✓ eventuelle Fehler werden durch Compiler nicht gemeldet
- ✓ und bleiben somit unentdeckt

Kohäsion - Implizite Kopplung

- ✓ In der Klasse Befehlswörter tragen wir den Befehl „look“ ein

```
// ein konstantes Array mit den gültigen Befehlswörtern

Private static final String gueltige[Befehle] = {
„go“, „quit“, „help“, „look“
};
```

- ✓ Nach dieser Eingabe passiert jedoch nichts

  „Ich weiß nicht, was Sie meinen…“
- ✓ Eine Aktion muss implementiert werden

Kohäsion - Implizite Kopplung

```
private void umsehen()
{
        System.out.print(aktuellerRaum.gibLageBeschreibung());
}
 ------------------------------------------------------------------------
if (befehlwort.quals(„help“))  {
        hilstextAusgeben();
}
else if (befehlswort.quals(„go“))  {
}
else if (befehlswort.equals(„look“))  {
        Umsehen();
}
else if (befehlswort.quals(“quit”))  {
        moechteBeenden = beenden(befehl);
}
```

Kohäsion - Implizite Kopplung

- ✓ Änderung scheinbar erledigt, Befehl wird angezeigt
- ✓ Bei der Eingabe des Befehls „help“ wird das Problem aber deutlich

- ✓ Ihnen stehen folgende Befehle zur Verfügung:
  go quit help

- ✓ Hilfstext bearbeiten und Befehl hinzufügen

Kohäsion - Implizite Kopplung

```
/*
  * Gib alle gültigen Befehlswörter auf der Konsole aus.
  */

public void alleAusgeben()
{
        for (String befehl : queltigeBefehle) {
                System.out.print(befehl + " ");
    {
    System.out.printn();
}
```

Kohäsion

Lesbarkeit &
Wiederverwendung

Ein guter Klassenentwurf weißt
eine lose Kopplung
und eine höhe Kohäsion auf

…immer im Hinterkopf behalten

- ✓ Wann ist eine Methode zu lang?
- ✓ Wenn sie mehr als eine Aufgabe erfüllt.
- ✓ Wann ist eine Klasse zu komplex?
- ✓ Wenn sie mehr als eine logische Einheit modelliert.

ENDE