Von der Analyse zum Entwurf

Bei UML-Klassendiagrammen

Was bisher geschah … (Analyse)

- Analyse des Lastenhefts
- Modellierung des Analyseklassendiagramms
  - Ausschließlich Klassen, die das Analyseergebnis repräsentieren
  - Nur Attribute und Methoden, die aus Lastenheft ersichtlich sind
  - Keine exakten Attributtypen (int, long, float, …)
  - Keine Interfaces und abstrakte Klassen
  - Keine *Enums* und *Enumerations*
  - Keine detaillierte GUI-Modellierung
  - Keine Util- bzw. Helperklassen
  - Keine konkrete Datenbankschicht (JPA, JDBC, …)

All das kommt jetzt dran … (Entwurf)

Entwurfsklassendiagramm durch Erweiterung und Optimierung des AKD:

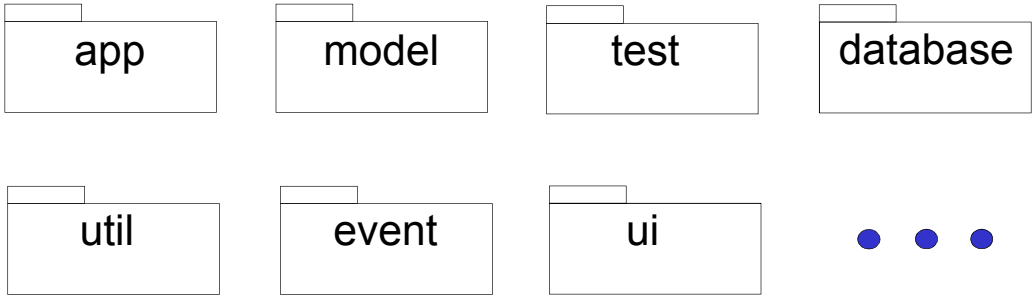
- Aufteilung in Packages
- Zusammenfassung von Klassen (Vererbung,)
- Erweiterung um weitere Attribute und Methoden (add-, set-, remove-, …)
- Exakte Attributtypen (int, long, float, …)
- Modellierung von Interfaces und abstrakten Klassen zur besseren Verwendbarkeit der Klassen im Applikationsumfeld
- Modellierung von *Enums* (*Enumerations)* für zahlenmäßig begrenzte Elemente und Konstanten
- Detaillierte GUI-Modellierung (Observer usw.)
- Modellierung von Util- bzw. Helperklassen, Entity-Provider usw.
- konkrete Datenbankschicht (JPA, JDBC, …)

Einzelne Schritte

Schritt 1: Modularisieren mit Packages

- Modellklassen
- Applikationsbezogene Klassen (und Packages)
- Utility-Klassen
- UI-Klassen
- Event- und Listener-Klassen
- Exceptions
- Test-Klassen
- usw.

z.B.:



Schritt 2: Optimierung und Erweiterung des AKD

- Modellklassen
  - Ziele:
    - Modellklassen sollen keine Referenzen „nach außen“ (Package) haben
    - Modellklassen sollen ausschließlich per Konstruktor sowie *get-*/*set*-Methoden manipulierbar sein.
    - Modellklassen sollen ausschließlich über angeschlossene Listener mit der Außenwelt kommunizieren (z.B. *PropertyChangeListener* (Java-Beans))
  - Erweiterung um weitere Attribute und Methoden
  - Exakte Attributtypen festlegen (int, long, float, …)
  - Typen der Argumente und Rückgabewerte festlegen (int, long, float, REFs…)

Schritt 2 (Fortsetzung)

Erweiterungen zum Entwurfsklassendiagramm

- Klassen in den Packages
  - Zusammenfassung von Klassen
  - Modellierung von Interfaces und abstrakten Klassen zur besseren Verwendbarkeit der Klassen im Applikationsumfeld
  - Evtl. Modellierung von *Enums* für zahlenmäßig begrenzte Elemente (auch zur Unterklassenunterscheidung) und Konstanten

- Für Modellklassen
  - konkrete Datenbankschicht (JPA, JDBC, …)

Schritt 4

Erweiterungen zum Entwurfsklassendiagramm

- Detaillierte GUI-Modellierung (Observer usw.)
- Einsatz von Entwurfsmustern
- Modellierung von
  - Util- bzw. Helperklassen,
  - „Entity-Providern“ (Factories, Singletons, Multitons, …)
  - Events & Listenern
  - Exceptions

Schritt 5

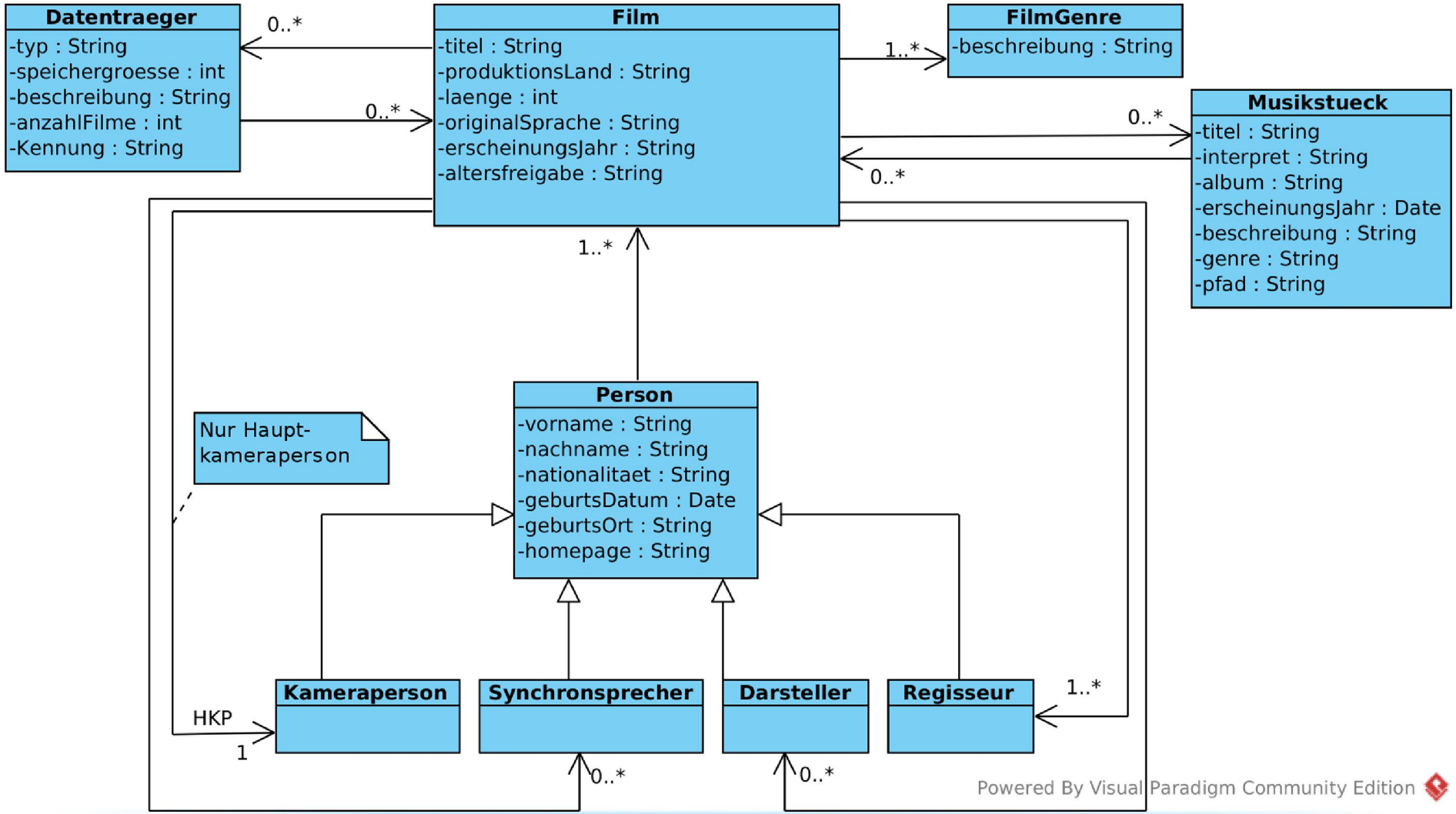
- Dokumentation der Änderungen und Erweiterungen
  - Was hat sich im Klassendiagramm geändert?
  - Welche Klassen wurden weshalb zusammengefasst, verschoben, ersetzt, entfernt, vererbt, …?
  - Weshalb wurden welche Interfaces und abstrakte Klassen eingeführt?
  - Weshalb und welche *Enum*s verwendet?

Entwurfserweiterungen der Semesteraufgabe

Filmverwaltung

Semesterbeispiel „Filmverwaltung“

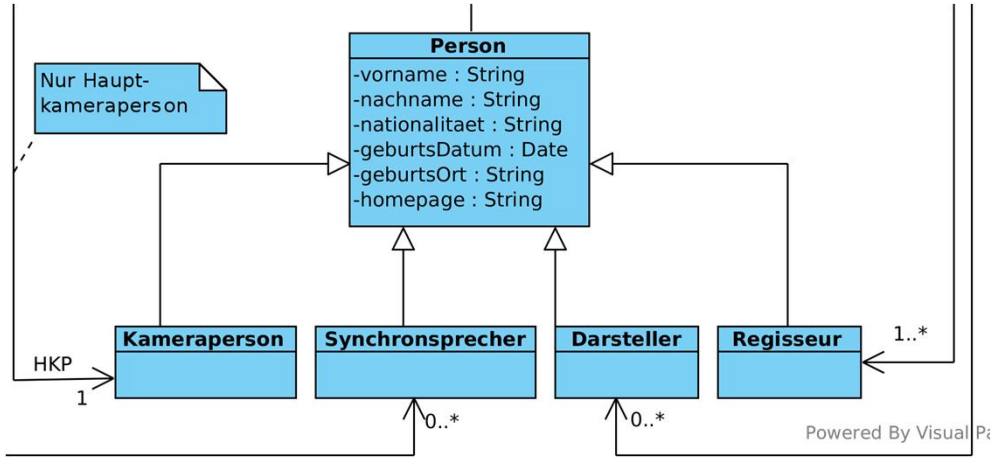
Analyseklassendiagramm:



Semesterbeispiel „Filmverwaltung“

Einzelne Schritte:

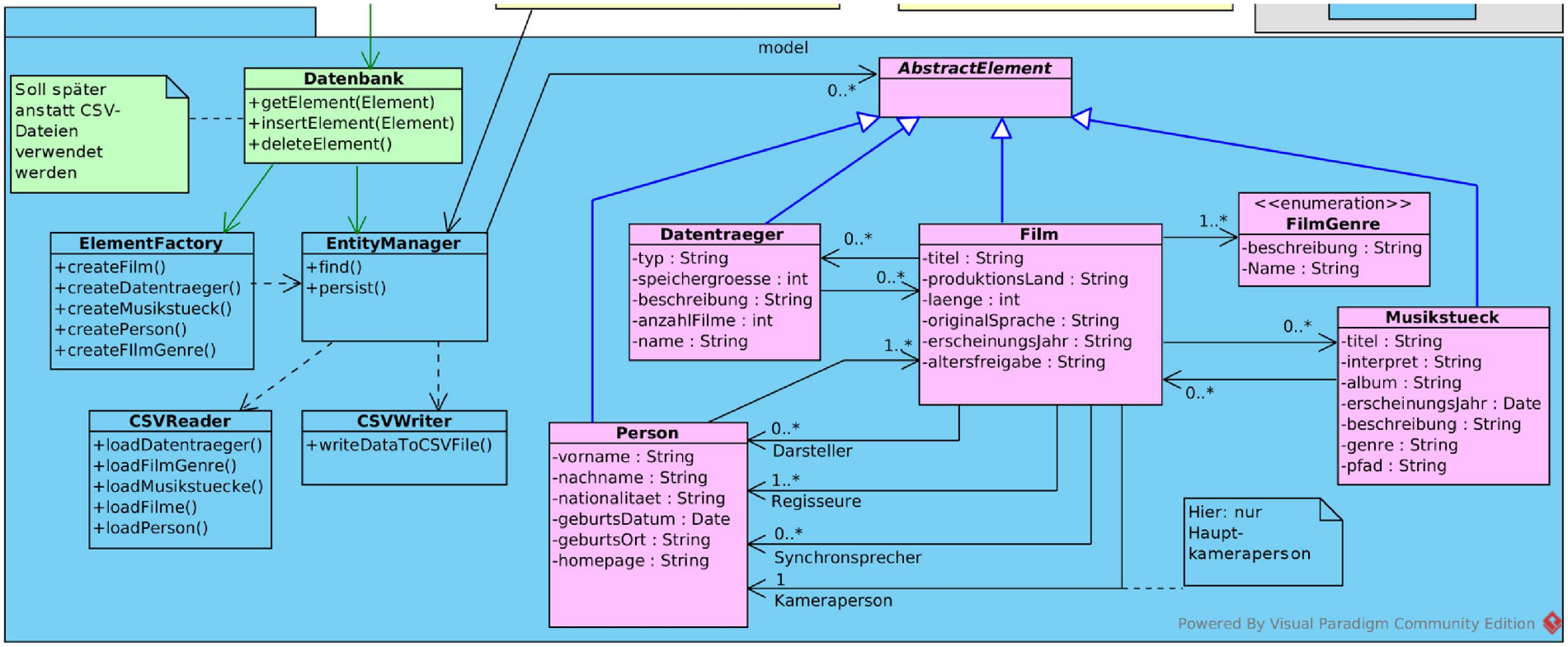
- Die Klassen *Kameraperson*, *Synchronsprecher*, *Darsteller* und *Regisseur* unterscheiden sich prinzipiell nicht => Rollen anstatt Unterklassen
- *Genre:* müssen sie nicht erweitert werden => *Enum*
  sonst: als Klasse belassen
- Hinzufügen von Klassen, welche die einzelnen Elemente verwalten (*CSVReader*, *CSVWriter*, *EntityManager*, „*Datenbank*“ (als Klasse) usw.



Semesterbeispiel „Filmverwaltung“

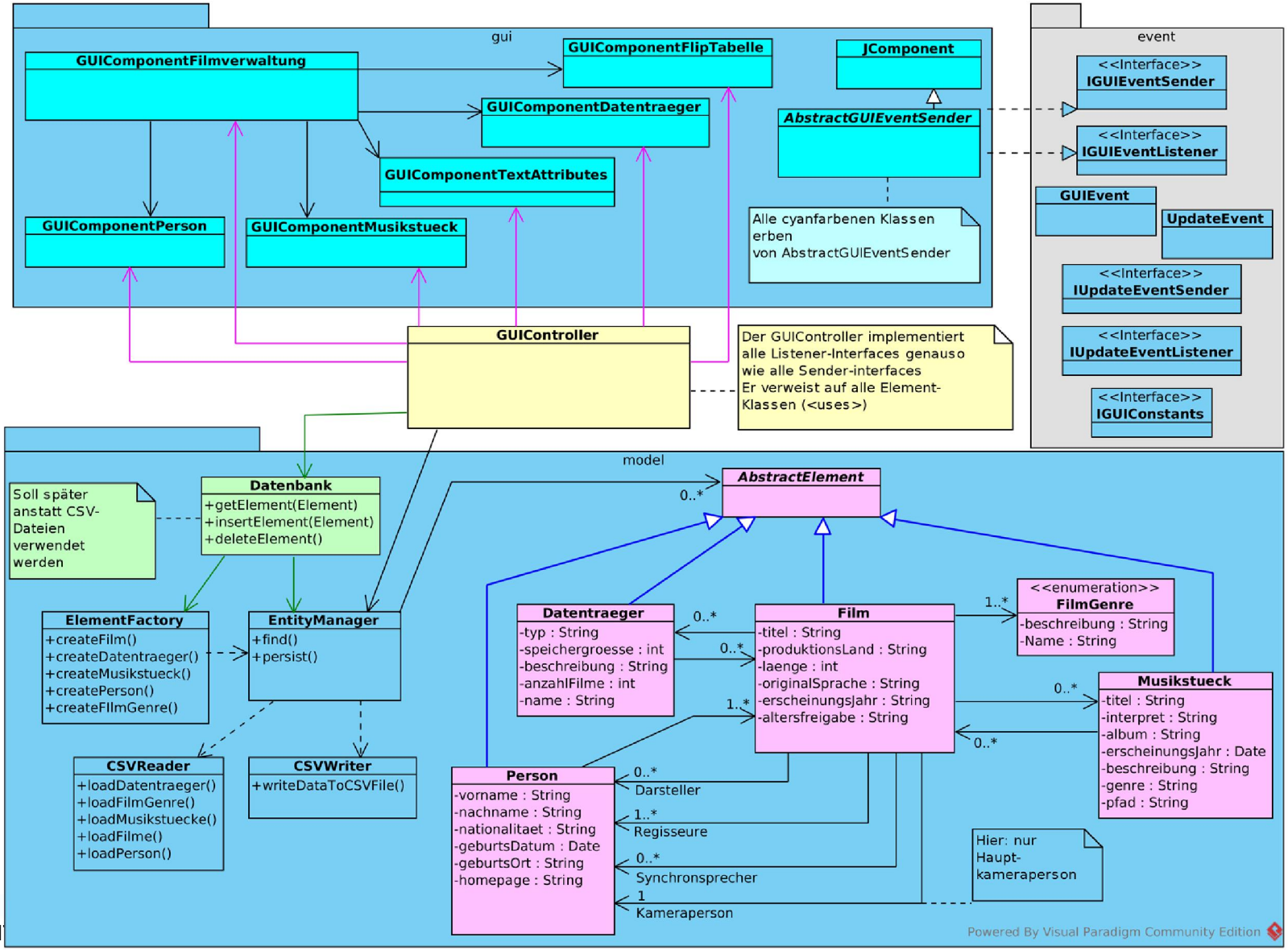
Entwurfsklassendiagramm, Modell-Package:

(mit ElementFactory, EntityManager und Importer/Exporter)



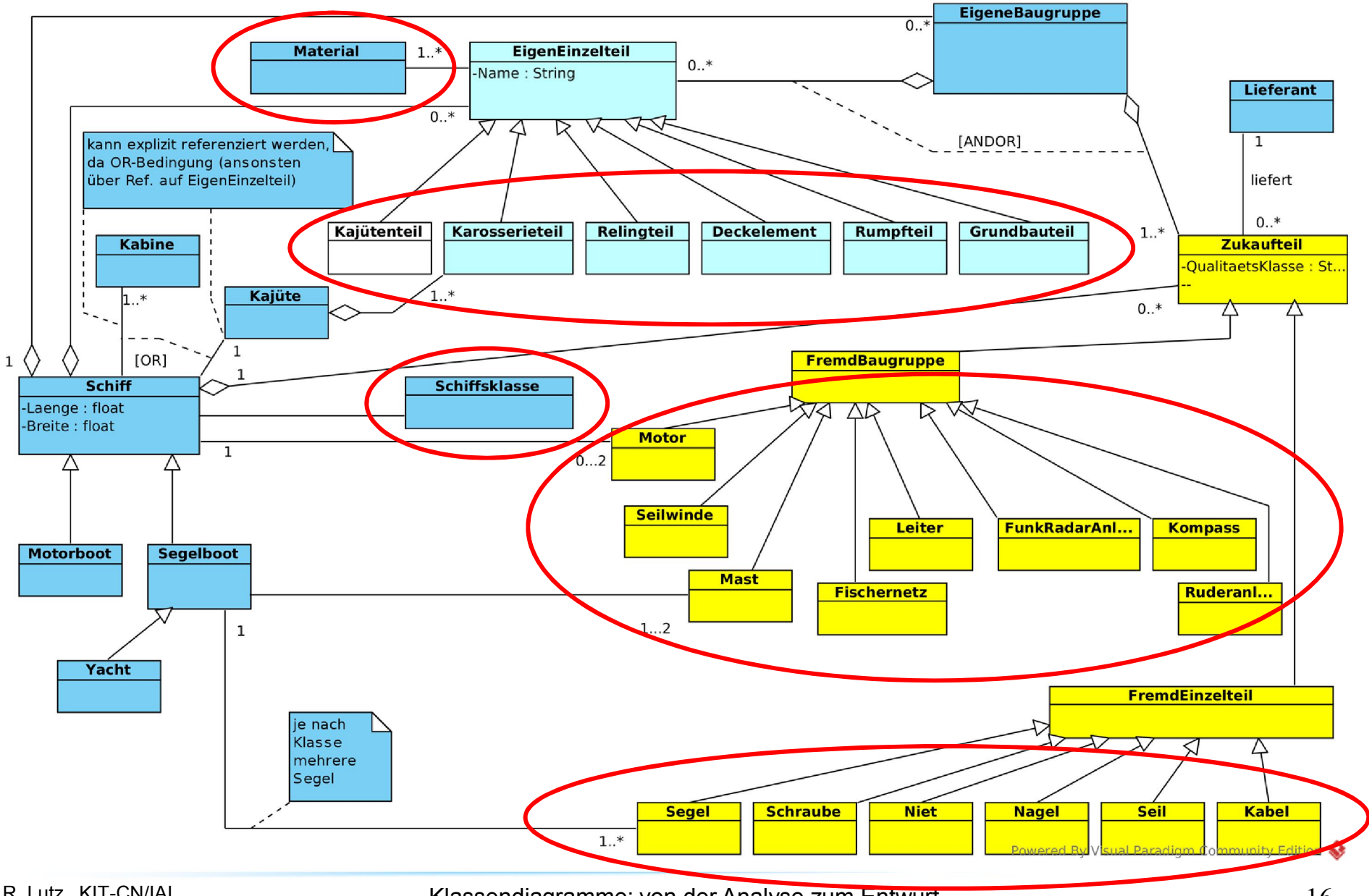
Semesterbeispiel „Filmverwaltung“

Entwurfsklassendiagramm, Modell-, Controller-, GUI- und Event-Packages:



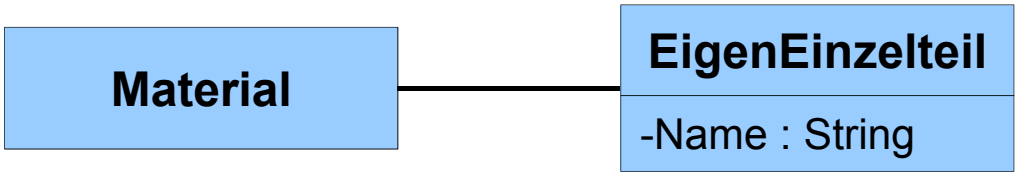
Entwurfserweiterungen der Übungsaufgaben

Aufgabe „Schiffswerft“

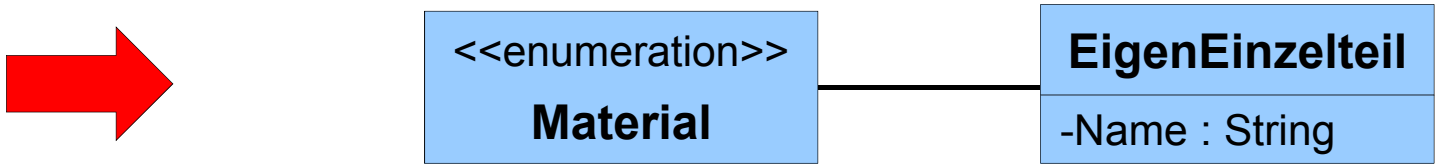


Aufgabe „Schiffswerft“

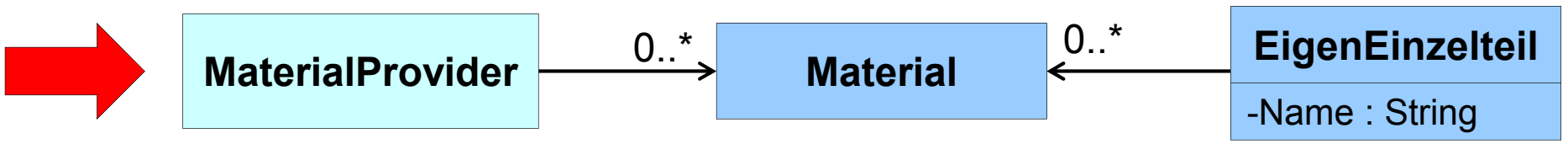
Relation Material – EigenEinzelteil:



- Wenn Anzahl von Materialien fest oder begrenzt und Attributanzahl gering => **Enum**



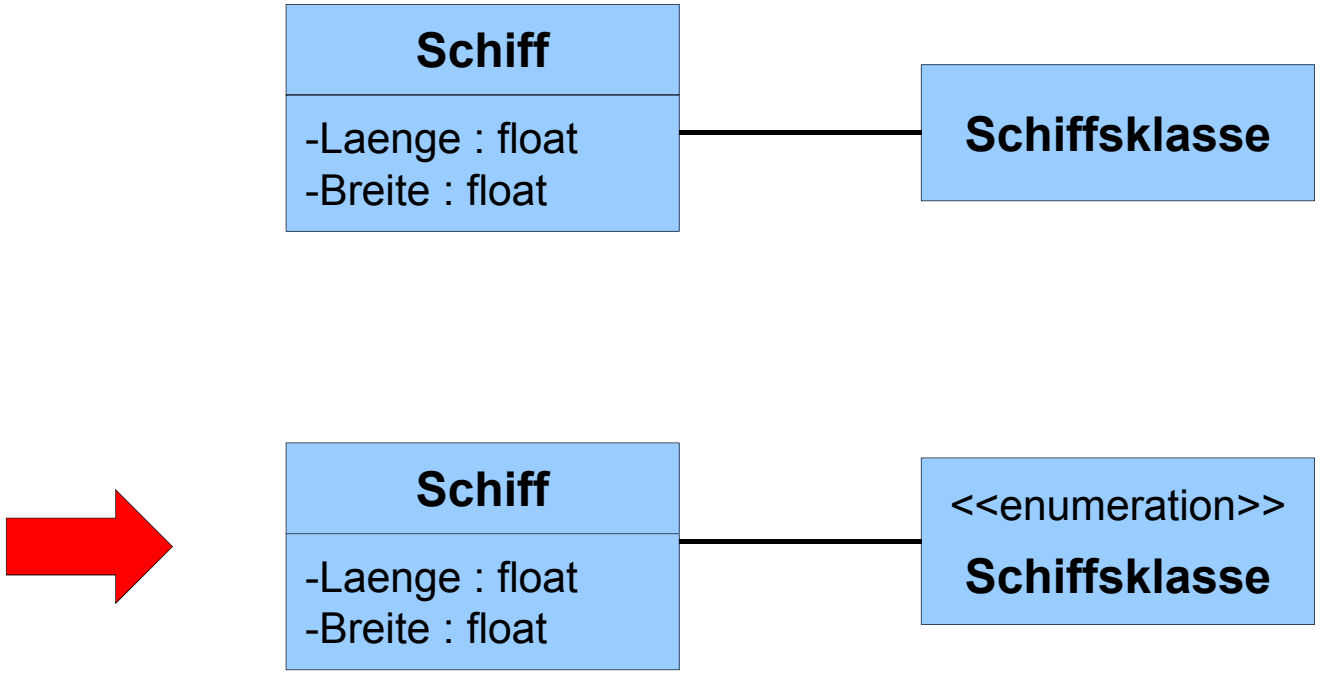
- Sonst Klasse belassen und dafür sorgen, dass von Datei (Properties) oder DB (Tabelle mit Konstanten) die möglichen Werte geladen werden.



Aufgabe „Schiffswerft“

Relation Schiff – Schiffsklasse:

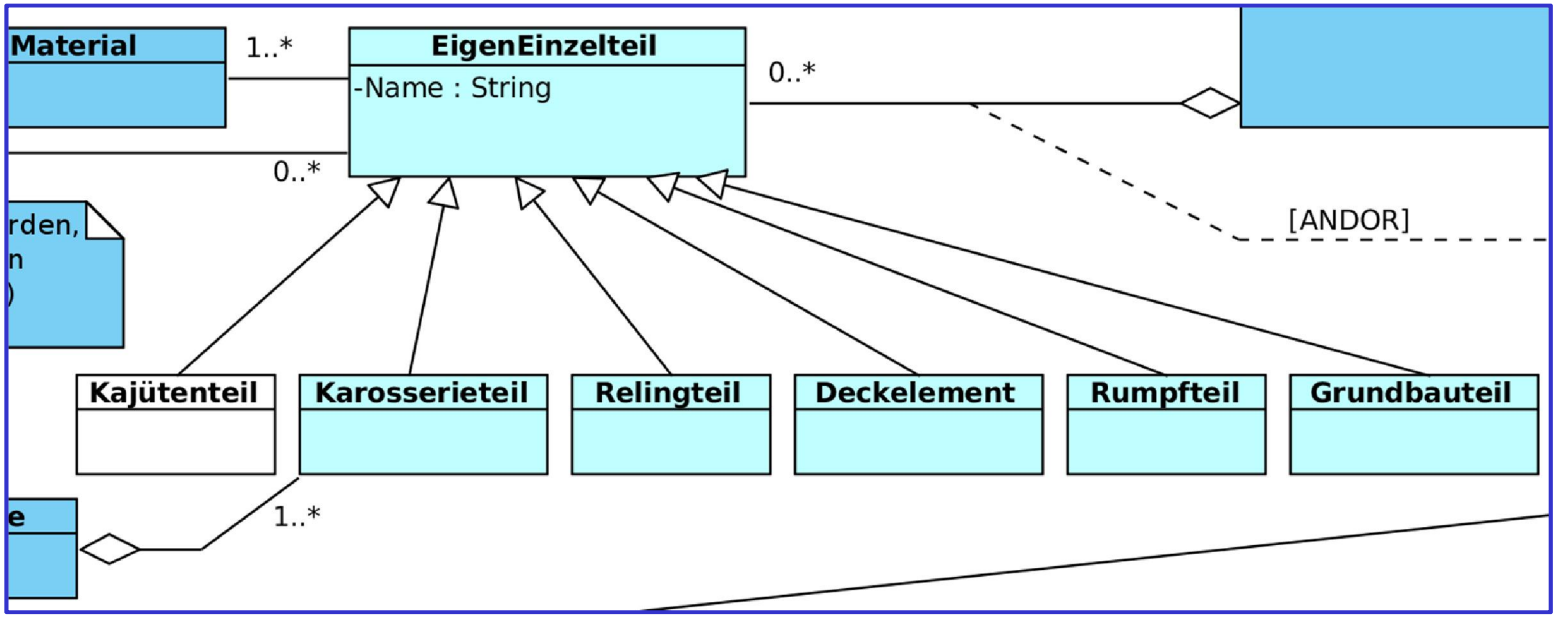
Anzahl der Schiffsklassen begrenzt und fest => **Enum**

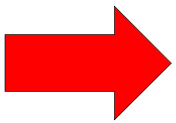


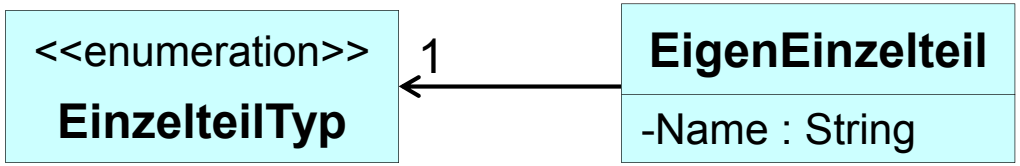
Aufgabe „Schiffswerft“

Mögliche Reduktion oder Wegfall vieler erbenden Unterklassen (1):

Alle Unterklassen haben **keine** individuellen Referenzen „nach außen“ und unterscheiden sich nicht wesentlich => **Enum, sonst abstrakte Oberklasse**



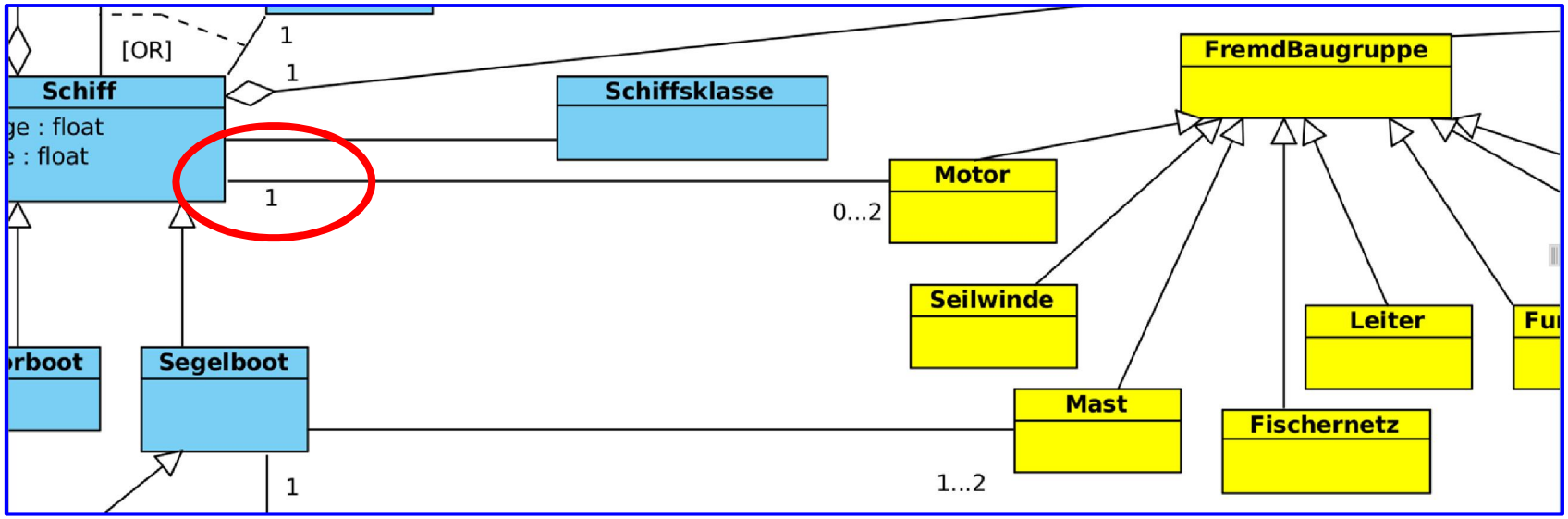


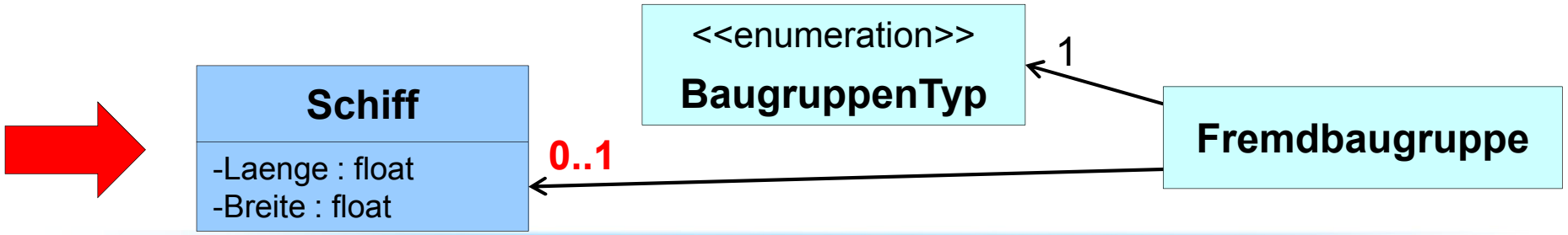


Aufgabe „Schiffswerft“

Mögliche Reduktion oder Wegfall vieler erbenden Unterklassen (2):

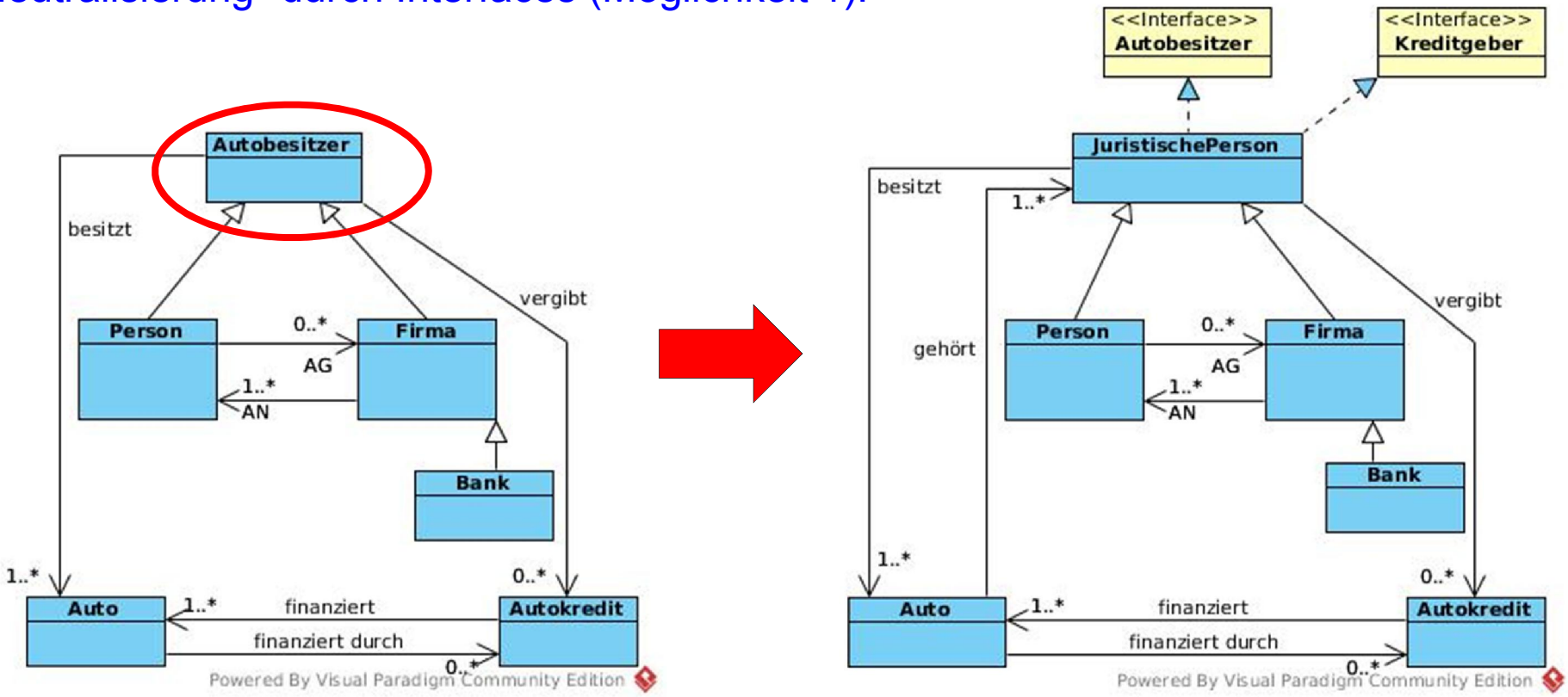
Einzelne Unterklassen mit **sehr wenigen** individuelle Referenzen „nach außen“
=> **Enum + optionale Referenzen, sonst abstrakte Oberklasse**





Aufgabe „Autokredit“

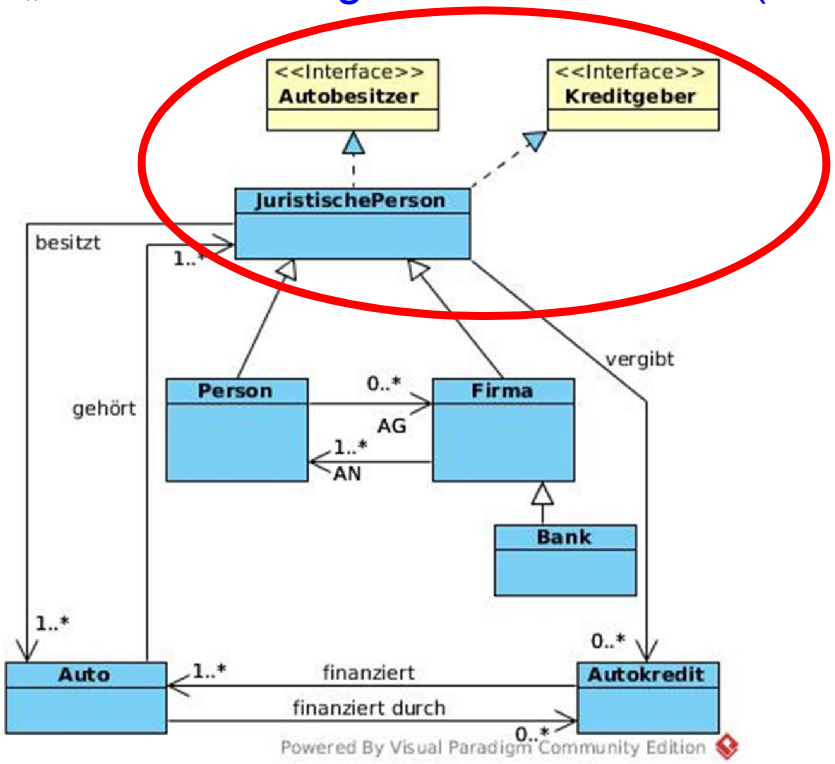
„Neutralisierung“ durch Interfaces (Möglichkeit 1):

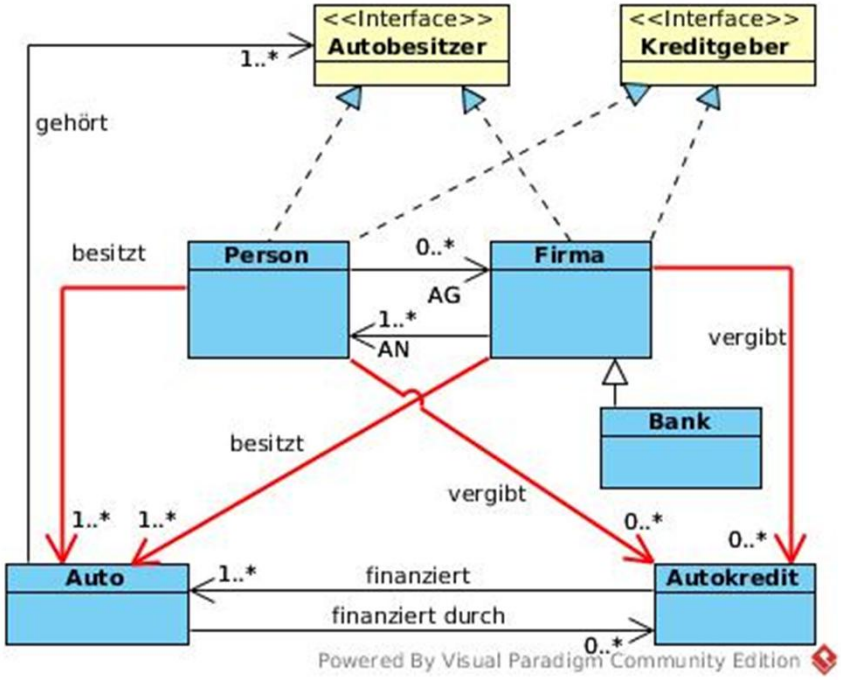


- *JuristischePerson*: universeller verwendbar als Klasse *Autobesitzer*
- *Autobesitzer, Kreditgeber* als Interface: flexibler (nur das Verhalten wird realisiert)

Aufgabe „Autokredit“

„Neutralisierung“ durch Interfaces (Möglichkeit 2):





Vorteil: Reduktion der Klassenhierarchie, klare „Aufgabenverteilung“

Nachteil: zusätzliche Referenzen erforderlich
(können durch Interfaces nicht vermieden werden)

Weitere Aufgabenbeispiele

- Siehe eigene Aufgabensammlung bzw. bereits erstellte AKDs aus den Aufgaben