Übung
„Einführung in die Softwaretechnik“

Lösungshinweise zum Übungsblatt 12

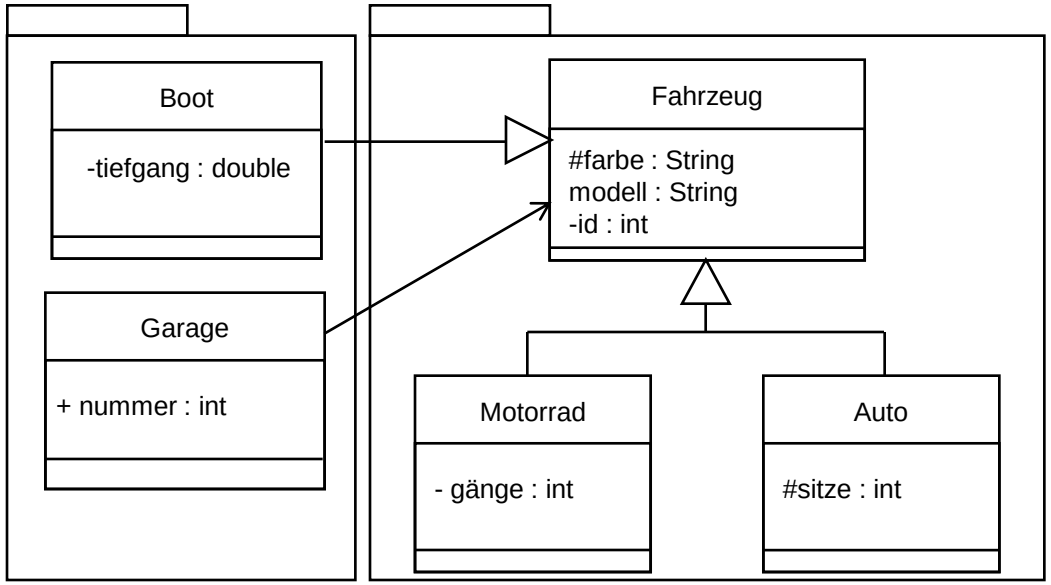
Aufgabe 28

	Sichtbarkeits-Symbol			
UML	+	#	–	
Java	**public**	**protected**	**private**	*(default)*
Sichtbar für:				
Gleiche Klasse	**ja**	**ja**	**ja**	**ja**
Andere Klasse, gleiches Paket	**ja**	**ja / nein***	**nein**	**ja**
Andere Klasse, anderes Paket	**ja**	**nein**	**nein**	**nein**
Unterklasse, gleiches Paket	**ja**	**ja**	**nein**	**ja**
Unterklasse, anderes Paket	**ja**	**ja**	**nein**	**nein**

* In UML und C++ “nein”, in Java “ja”.

Aufgabe 28

In der UML und in Java gibt es verschiedene Sichtbarkeiten. Geben Sie für folgendes Klassendiagramm an, welche unten stehenden Aussagen gelten.



Aufgabe 28

a) Autos können ihre Id auslesen.
- id: private
- Unterklasse, gleiches Paket

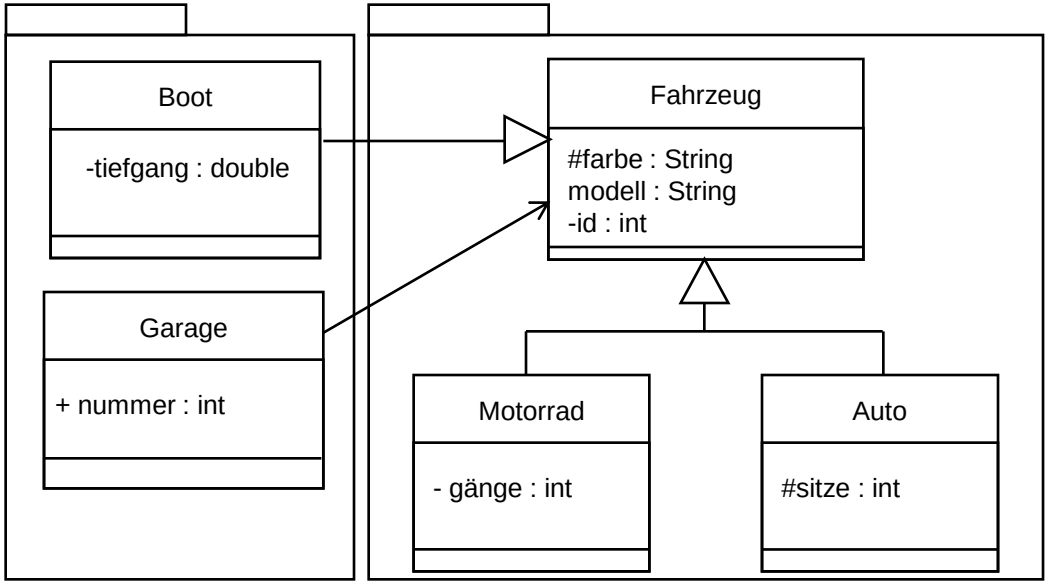
-> Nein

b) Motorräder können ihr Modell auslesen.
- modell: default
- Unterklasse, gleiches Paket

-> Ja

c) Motorräder können ihre Farbe auslesen.
- farbe: protected
- Unterklasse, gleiches Paket

-> Ja



	Sichtbarkeits-Symbol			
UML / Java	+ **public**	# **protected**	– **private**	*(default)*
Sichtbar für:				
Gleiche Klasse	**ja**	**ja**	**ja**	**ja**
Andere Klasse, gleiches Paket	**ja**	**ja / nein**	**nein**	**ja**
Andere Klasse, anderes Paket	**ja**	**nein**	**nein**	**nein**
Unterklasse, gleiches Paket	**ja**	**ja**	**nein**	**ja**
Unterklasse, anderes Paket	**ja**	**ja**	**nein**	**nein**

Aufgabe 28

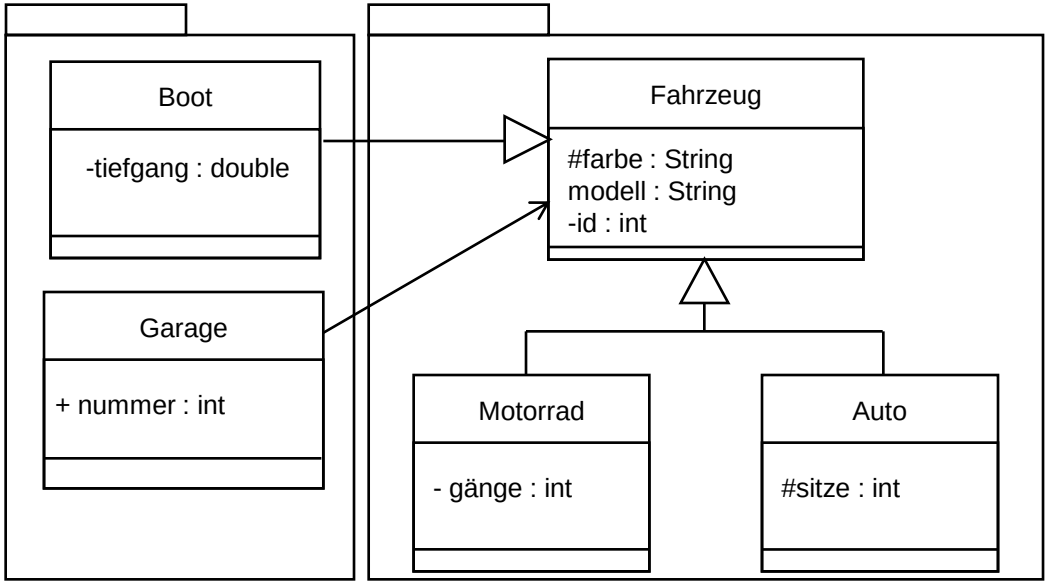
d) Motorräder können ihre Gänge auslesen.
- id: private
- Gleiche Klasse

-> Ja

e) Boote können ihre Farbe auslesen.
- farbe: protected
- Unterklasse, anderes Paket

-> Ja

f) Boote können ihr Modell auslesen.
- d) Modell: default
- e) Unterklasse, anderes Paket
- f) -> Nein



Sichtbarkeits-Symbol				
UML Java	+ **public**	# **protected**	– **private**	*(default)*
Sichtbar für:				
Gleiche Klasse	ja	ja	ja	ja
Andere Klasse, gleiches Paket	ja	ja / nein	nein	ja
Andere Klasse, anderes Paket	ja	nein	nein	nein
Unterklasse, gleiches Paket	ja	ja	nein	ja
Unterklasse, anderes Paket	ja	ja	nein	nein

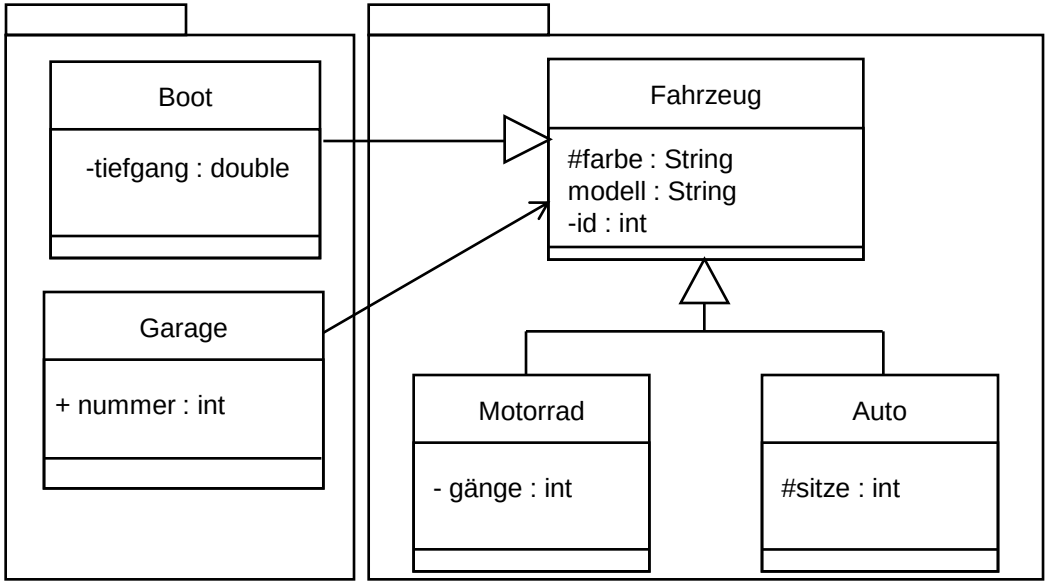
Aufgabe 28

g) Garagen können die Farbe von Autos auslesen.
- farbe: protected
- Andere Klasse, anderes Paket

-> Nein

h) Fahrzeuge können die Nummer ihrer Garage auslesen.
- nummer: public
- Andere Klasse, anderes Paket
- Aber: Keine Navigationsrichtung von Fahrzeug zu Garage.

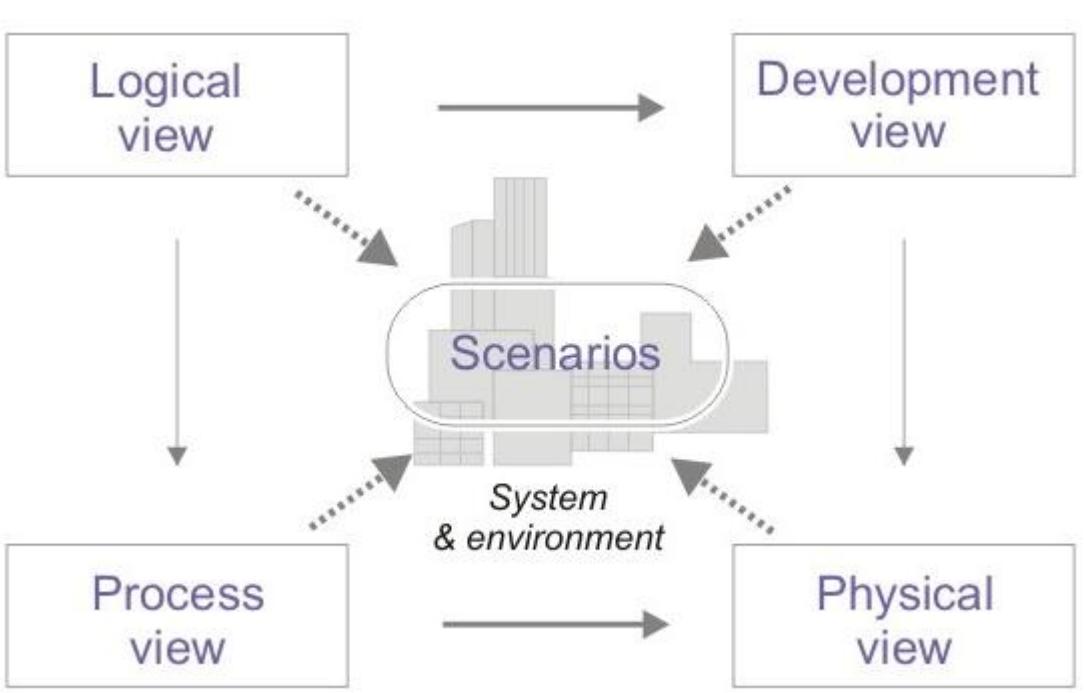
-> Nein



Sichtbarkeits-Symbol				
UML / Java	+ **public**	# **protected**	– **private**	*(default)*
Sichtbar für:				
Gleiche Klasse	ja	ja	ja	ja
Andere Klasse, gleiches Paket	ja	ja / nein	nein	ja
Andere Klasse, anderes Paket	ja	nein	nein	nein
Unterklasse, gleiches Paket	ja	ja	nein	ja
Unterklasse, anderes Paket	ja	ja	nein	nein

Aufgabe 29a

- 4+1 Sichten nach Kruchten



Aufgabe 29a

Logical View:

Elemente der Anwendungsdomäne und deren Verbindungen untereinander.

Process View:

Zuordnung der Elemente der logischen Sicht zu Kontrollflüssen. Beinhaltet Parallelität/Nebenläufigkeit, logische Verteilung, Performance, Systemintegrität und Fehlertoleranz.

Development View:

Entwicklung des Systems in Form von Klassen, Paketen, Modulen, Dateien etc.

Physical View:

Verteilung auf physische Ressourcen

Scenarios:

Verbindet die 4 Sichten unter Nutzung typischer Anwendungsszenarien.

Aufgabe 29b - Logical View

- Booch class diagrams
- UML Klassendiagramme
- UML Kompositionsstrukturdiagramm
- UML Statecharts
- Weitere State-Transition-Notationen
- E-R-Diagramme

Aufgabe 29b – Development View

- Booch class diagrams
- UML Klassendiagramme
- UML Statecharts
- Weitere State-Transition-Notationen
- E-R-Diagramme

- UML Komponentendiagramm
- UML Paketdiagramme

Aufgabe 29b – Process View

- Booch process diagrams
- UML Timingdiagramme
- UML Sequenzdiagramme
- UML Aktivitätsdiagramme
- UML Kompositionsstrukturdiagramme

Aufgabe 29b – Physical View

- Variante von Booch-Notation aus [Kru95]
- UML Verteilungsdiagramm

Aufgabe 29b – Scenarios

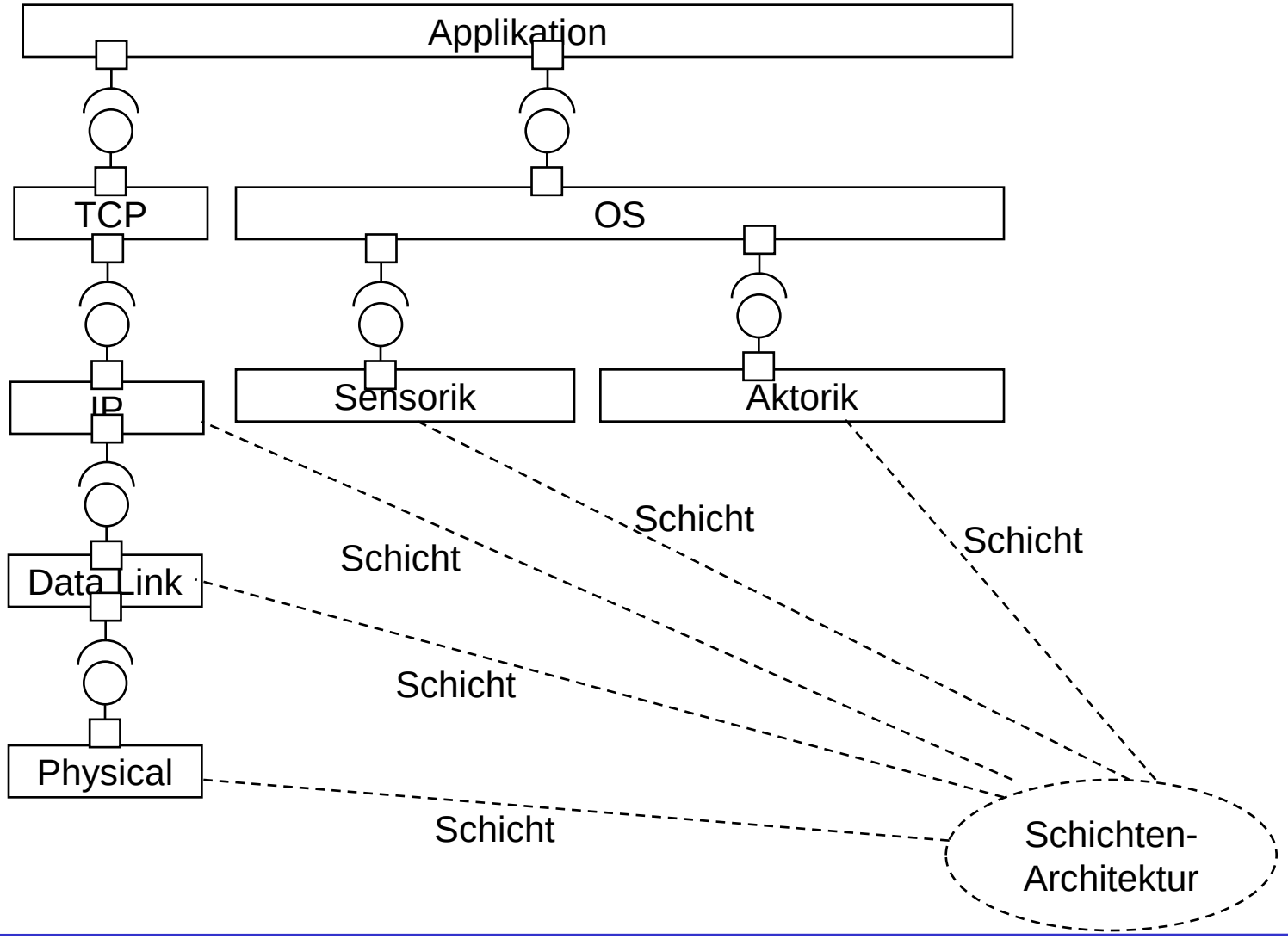
- Notation aus [Kru95]
- UML Sequenzdiagramme
- UML Use Cases
- UML Kommunikationsdiagramm

Aufgabe 30a

- Am meisten geeignet: Schichten

- Vorteile:
  - Wiederverwendbarkeit einzelner Schichten
  - Standardisierung einzelner Schichten möglich
  - Geringe Abhängigkeit zwischen den Komponenten:
    Gute Modifizierbarkeit
  - Austauschbarkeit einzelner Schichten
- Probleme:
  - Geringe Effizienz bei vielen Schichten
    - Daten durchlaufen viele Schichten und werden nur an wenigen Stellen bearbeitet
    - Jede Schicht erhält alle Daten, benötigt aber eventuell nur wenige davon
  - Anzahl der Schichten
    - Zu wenig: Muster entfaltet nicht das volle Potential
    - Zu viele: Muster führt zu unnötigem Overhead.

Aufgabe 30a - Beispiel



Aufgabe 30b

- Am meisten geeignet: Pipes & Filters

- Vorteile:
  - Flexibilität durch Austausch und Rekombination von Filtern
  - Hohe Kohäsion innerhalb der Filter
  - Geklärte Zuständigkeit
  - Einheitliche Definition der Schnittstellen
- Nachteile:
  - Jeder Filter muss Daten eventuell erneut parsen
  - Fehlerbehandlung schwierig, da verteilt

Aufgabe 31 - Knotenklassen

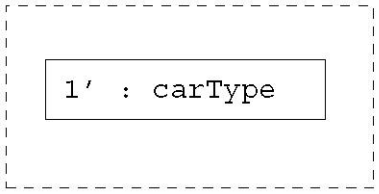
```
node class CAR
  intrinsic
    price : integer := 0;
  derived
    Price : integer = self.price;
end;

node class EXTRA is a CAR
  redef derived
    Price = self.price + self.<-enhanced_by-.Price;
end;
```

Aufgabe 31 - Transformationen

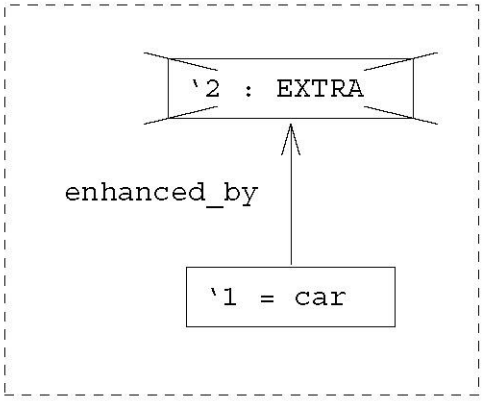
```
transformation CreateNewCar ( carType : type in CAR) =
  requires
    not (carType in EXTRA)
  end;
```

::=

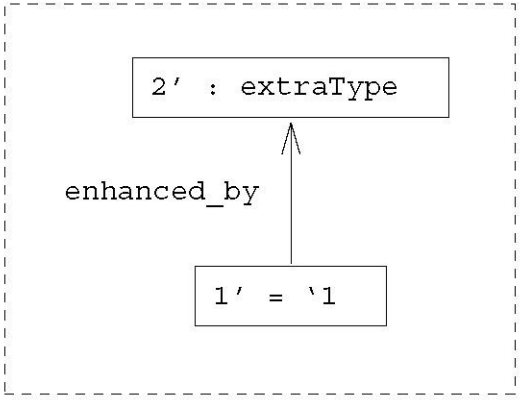



```
end;
```

```
transformation AddExtra
  ( car : CAR ; extraType : type in EXTRA) =
```



::=




```
end;
```