UNIVERSITY OF APPLIED SCIENCES
FHDW
FACHHOCHSCHULE DER WIRTSCHAFT
STAATLICH ANERKANNT

UNIFIED MODELING LANGUAGE™

UML Klassendiagramme - Kurzzusammenfassung

Michael Whittaker / 10. August 2008
Software Engineering - Fachhochschule der Wirtschaft Paderborn

Inhalt

- UML - was ist das? ①
- OOP - wozu Objekte? ②
- Klassen, Methoden, Attribute ③
- Beziehungen ④
- Real-World-Beispiel ⑤
- Fragen? ⑥

UML - Was ist das?

> ***The OMG's Unified Modeling Language™*** (**UML®**) helps you **specify, visualize, and document models of software systems**, including their structure and design, in a way that meets all of these requirements.
>
> (You can use UML for business modeling and modeling of other non-software systems too.)
>
> *[1]*

1 2 3 4 5 6

OBJECT MANAGEMENT GROUP

UNIFIED MODELING LANGUAGE

UML - Was ist das?

- **U**nified **M**odelling **L**anguage = „vereinheitlichte“ Modellierungssprache
- legt eine Sprache fest, mit der **Modelle beschrieben und modelliert** werden (**Spezifikation** ist frei verfügbar [2])
- besteht aus Sprache, Beziehungen, **grafischer Notation** und Austauschformat
- wahrscheinlich die **meist genutzte** Modellierungssprache
- umfassend und **erweiterbar**, wird weiterentwicklet (aktuell: 2.1.2 [2])

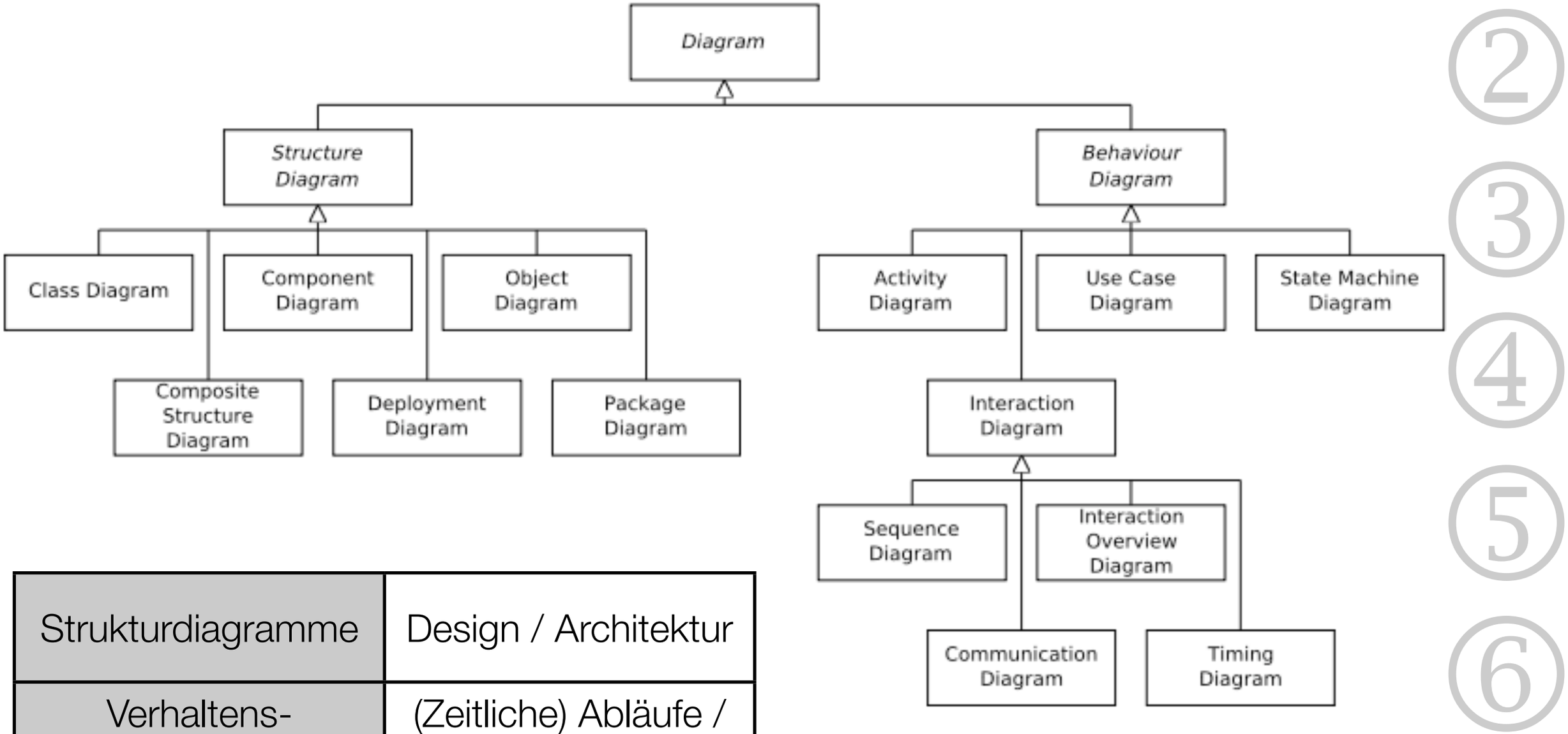
1 2 3 4 5 6

OMG OBJECT MANAGEMENT GROUP

UNIFIED MODELING LANGUAGE UML

UML - Was ist das?

- es existieren verschiedene Typen an Standard-UML-Diagrammen

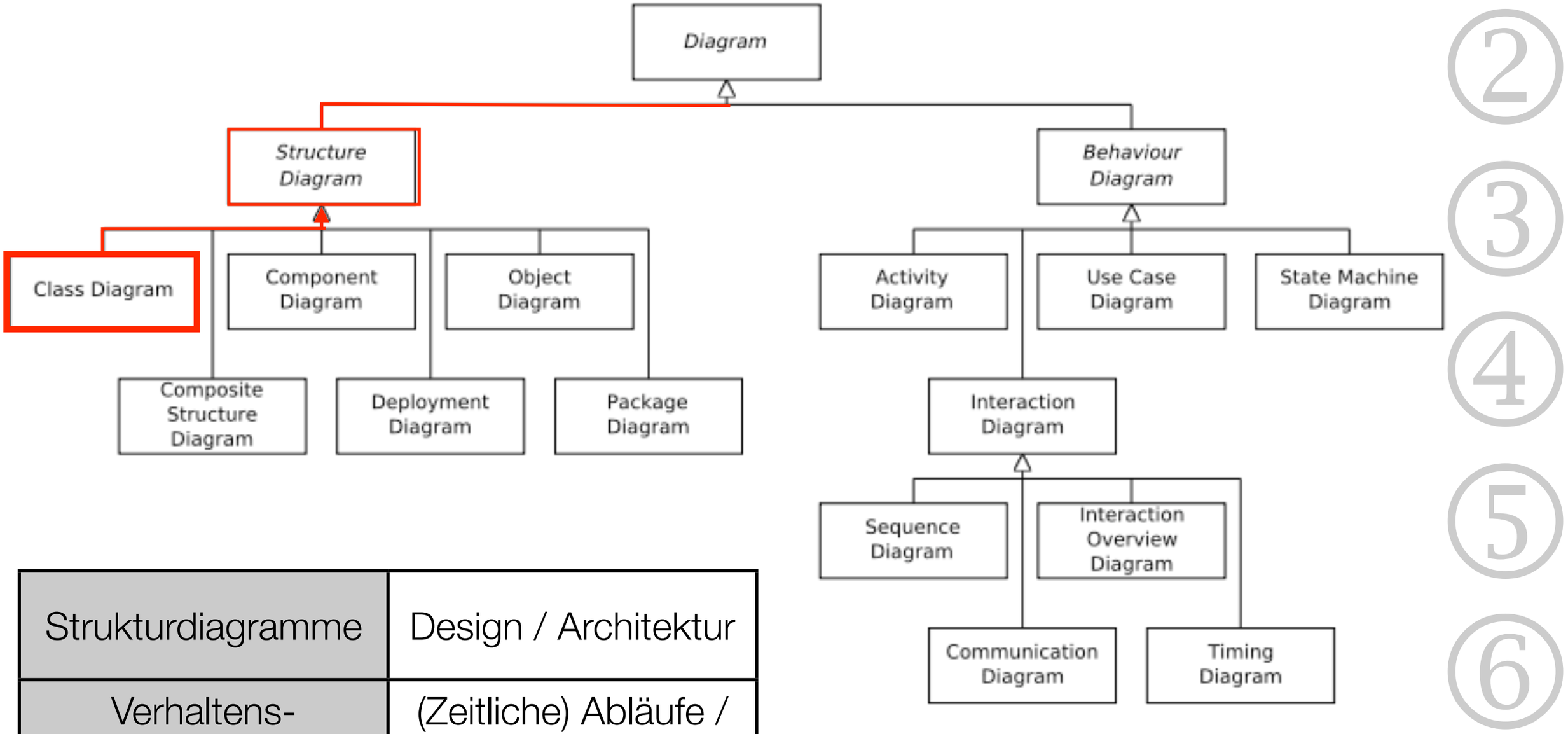


Strukturdiagramme	Design / Architektur
Verhaltens-diagramme	(Zeitliche) Abläufe / Prozesse

1 2 3 4 5 6

UML - Was ist das?

- es existieren verschiedene Typen an Standard-UML-Diagrammen



Strukturdiagramme	Design / Architektur
Verhaltens-diagramme	(Zeitliche) Abläufe / Prozesse

1 2 3 4 5 6

OOP - Wozu Objekte?

- **Programmierparadigma** / programming paradigm
  (Prinzip, wie man programmiert bzw. eine Programmiersprache aufgebaut ist)

- Zusammenfassung von Daten und Funktionen in **gekapselten Objekten**

  --> im **Gegensatz** zum Trennen von Daten und Funktionen beim traditionelleren Programmierparadigmen

- Jedes Objekt kann als **selbstständige „Maschine“** gesehen werden, welches häufig selbstständig existieren kann

  --> dadurch entsteht **Wiederverwendbarkeit**

① ② ③ ④ ⑤ ⑥

OOP - Wozu Objekte?

- **anstatt** eines Programms mit vielen **Unterprogrammen**:
  --> viele **Objekte** mit **Attributen, Methoden und Fähigkeiten,** die **interagieren**

- weiteres:
  - **Instanz**: „konkretes Exemplar im Speicher“
  - **Klasse** vs. **Objekt**: **Definition**/Vorlage eines Objekts vs. seine **Instanz**
  - **Methode**: Funktion/**Verhalten** eines Objekts
  - **Nachricht**: Kommando zum **Aufruf einer Methode**
  - **Vererbung**: verschiedene Modellierungsdetails, Hierarchien („Schäferhund“ erbt von „Hund“ erbt von „Säugetier“ erbt von „Wirbeltier“ erbt von „Tier“ erbt von „Lebewesen“)
  - **Abstraktion**: verschiedene **Abstraktionsgrade** werden für Interaktion bereitgestellt oder als **Rahmenwerk** vorgegeben („Rex“ kann „Schäferhund“, „Hund“, „Tier“, „Lebewesen“ sein)
  - **Polymorphie**: gleicher Umgang mit versch. Kindsklassen über selbe Methoden (Nachricht „sprich“ an Hund, Schwein, Mensch; „starte“ an „PKW“, „LKW“, „Bus“, „Motorrad“)

OOP - Wozu Objekte?

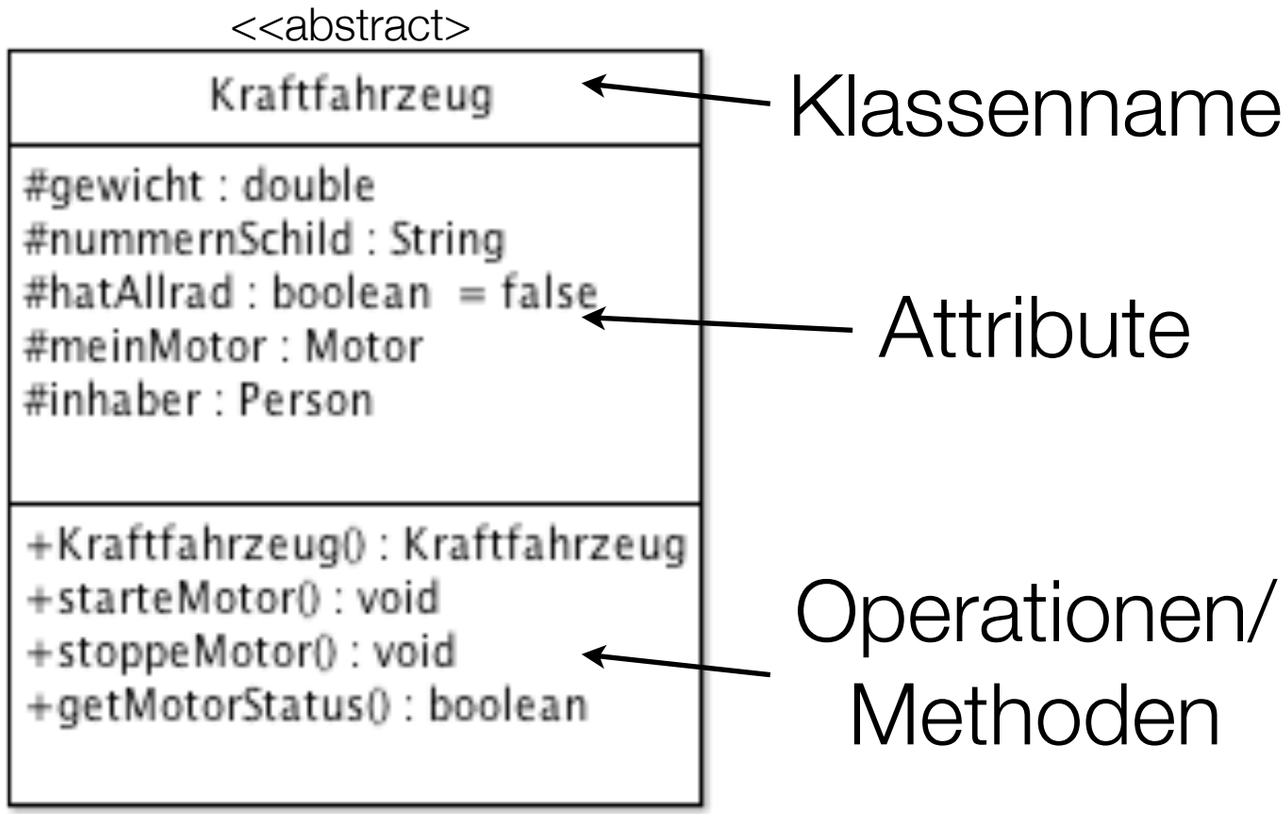
① ② ③ ④ ⑤ ⑥

> Durch Anwendung der ***Objektorientierung*** hofft man auf eine **Verkürzung der Entwicklungszeit**, eine **Senkung der Fehlerrate** und auf eine deutlich **verbesserte Erweiter- und Anpassungsfähigkeit** der Software.

[3]

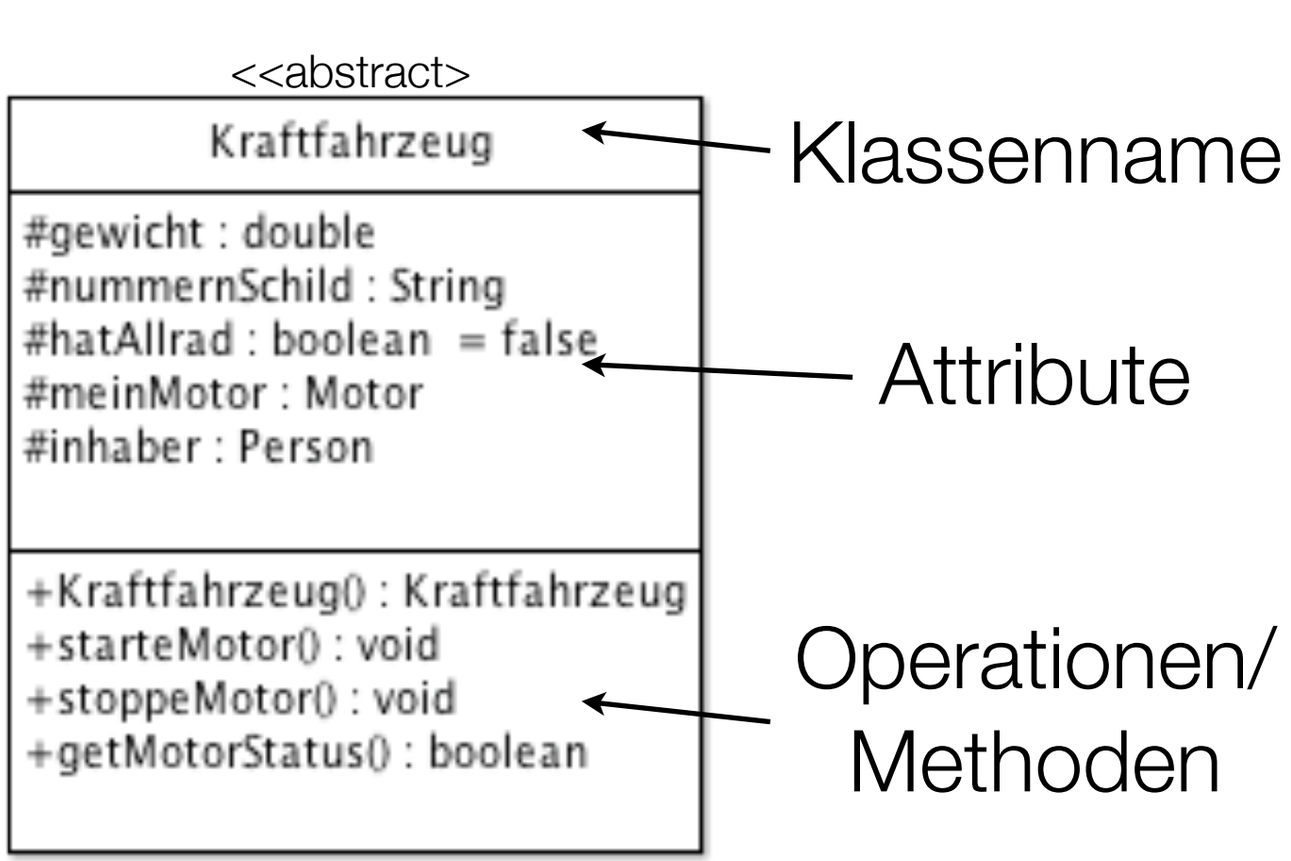
Klassen, Methoden, Attribute

- **Rund ums Auto** als **Beispiel**
- Unser **KFZ** mit versch. **Attributen** (Variablen des Objekts), einem Motor, einem Inhaber (einfach modelliert)
- **Notation: + public, # protected, – private**



Klassen, Methoden, Attribute

- **Rund ums Auto** als **Beispiel**
- Unser **KFZ** mit versch. **Attributen** (Variablen des Objekts), einem Motor, einem Inhaber (einfach modelliert)
- **Notation: + public, # protected, – private**



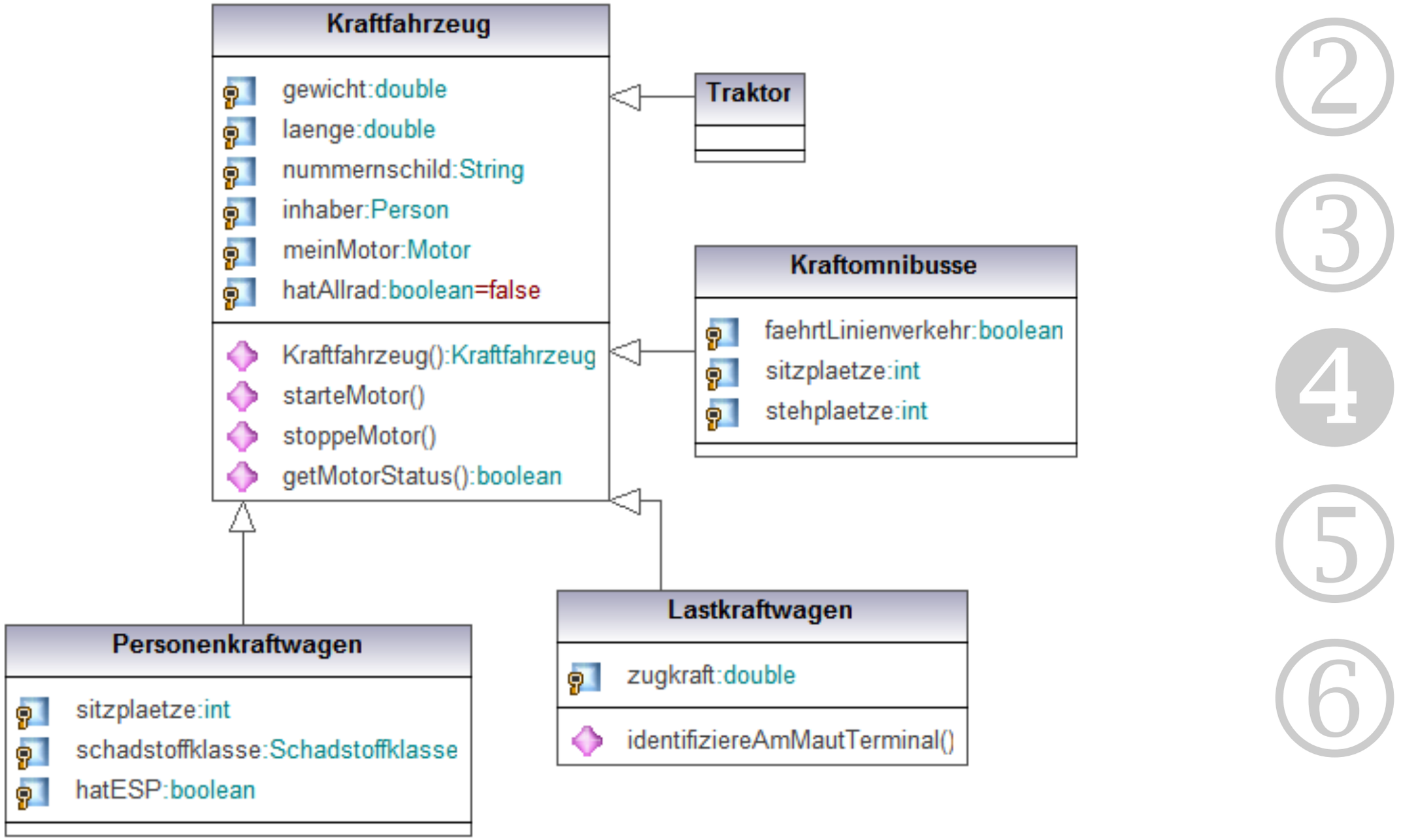
① ② ③ ④ ⑤ ⑥

Beziehungen: Vererbung (Ist-Beziehung)

- Definieren von verschiedenen Kfz-Typen, die alle die **Attribute und Methoden der Vaterklasse(n) übernehmen**
- **Notation:** Pfeil mit unausgefüllter Pfeilspitze auf Vaterklasse

Beziehungen: Vererbung (Ist-Beziehung)

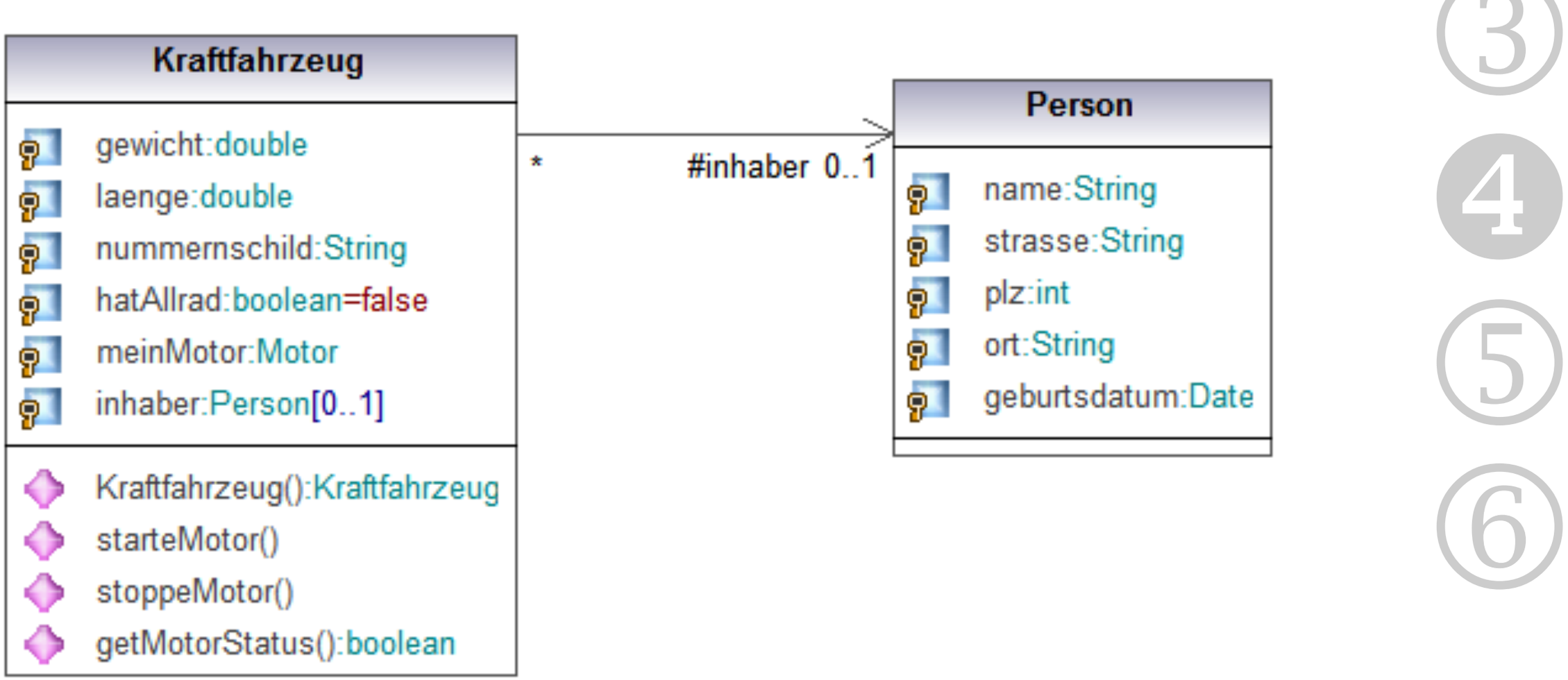
- Definieren von verschiedenen Kfz-Typen, die alle die **Attribute und Methoden der Vaterklasse(n) übernehmen**



① ② ③ **④** ⑤ ⑥

Beziehungen: Assoziation (Hat-Beziehung)

- Ein Objekt **hat** ein (oder mehrere) andere Objekte
- Beispiel: Eine Person hat keins, ein oder mehrere Kraftfahrzeuge,
  ein Kraftfahrzeug hat einen oder keinen Inhaber.
- **Notation**: Strich oder Strich mit Pfeilen (**Navigation**), **Attributangaben** und **Multiplizität**. // Lesen: Angaben gegenüber!



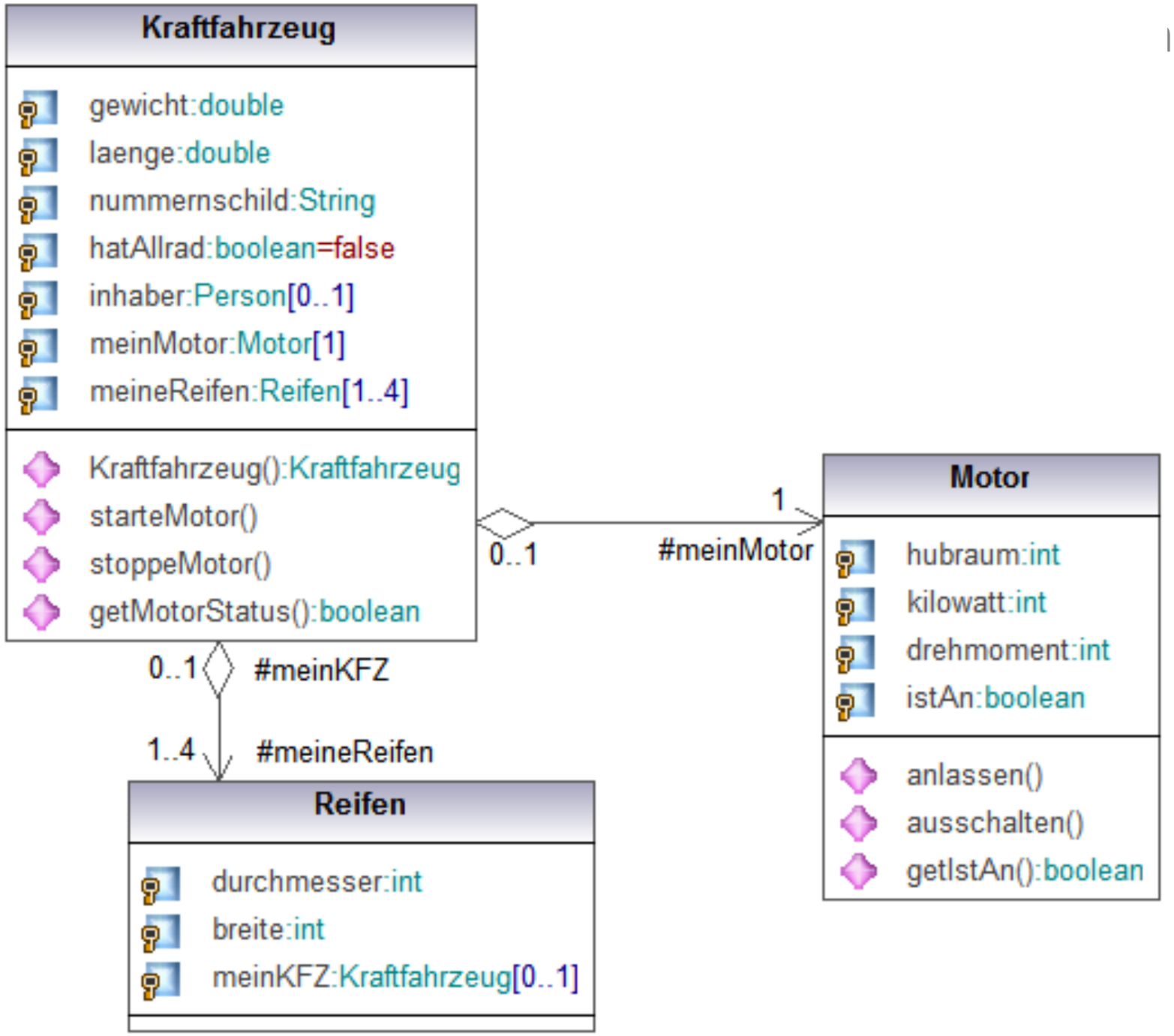
① ② ③ ④ ⑤ ⑥

Beziehungen: Aggregation (Besteht-aus-Beziehung)

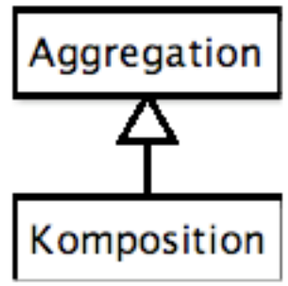
- Ein Objekt **besteht aus** einem (oder mehreren) anderen Objekten
- die einzelnen Komponenten des Aggregators sind **auch ohne ihn** existierend und **zu benutzen**
- Beispiel: Ein KFZ **besteht** aus einem Motor und vier Reifen

Beziehungen: Aggregation (Besteht-aus-Beziehung)

- Ein Objekt **besteht aus** einem (oder mehreren) anderen Objekten



Beziehungen: Komposition (starke Besteht-aus-Beziehung)



- Ein Objekt **besteht aus** einem (oder mehreren) anderen Objekten
- die einzelnen Komponenten des Aggregators sind i. d. R. **ohne ihn nicht existent** und nicht zu **benutzen**
- Beispiel („Exkurs“): ein Buch besteht aus mehreren Seiten (die Seiten ergeben ohne das Buch aber keinen Sinn und werden nicht verwendet)

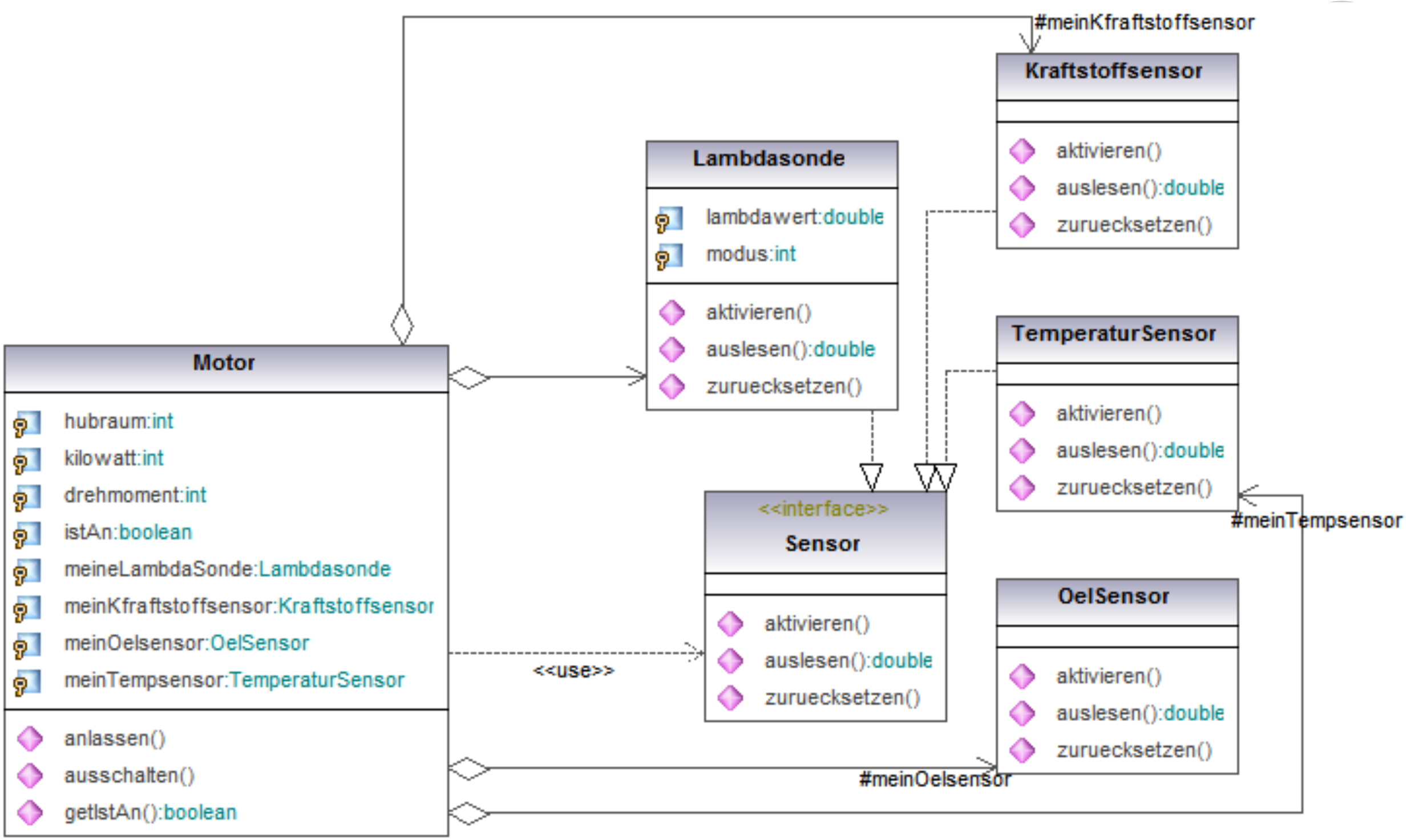
Beziehungen: Abhängigkeit

- eine Assoziation ohne damit direkt verbundene Instanzen
- verschiedene Arten der Abhängigkeit:
  - <<use>>-Abhängigkeit (Schreiner <<use>> Hammer; Schnittstellen)
  - Realisierungsabhängigkeit (s. nächste Folie)
  - <<include>>-Abhängigkeit (bei Operationen sind auch andere Objekte betroffen)
  - <<abstract>>-Abhängigkeit bei abstrakten Klassen
  - verschiedene andere geschäftsspezifische Arten, die nicht konkret mir Software zusammen hängen (müssen)

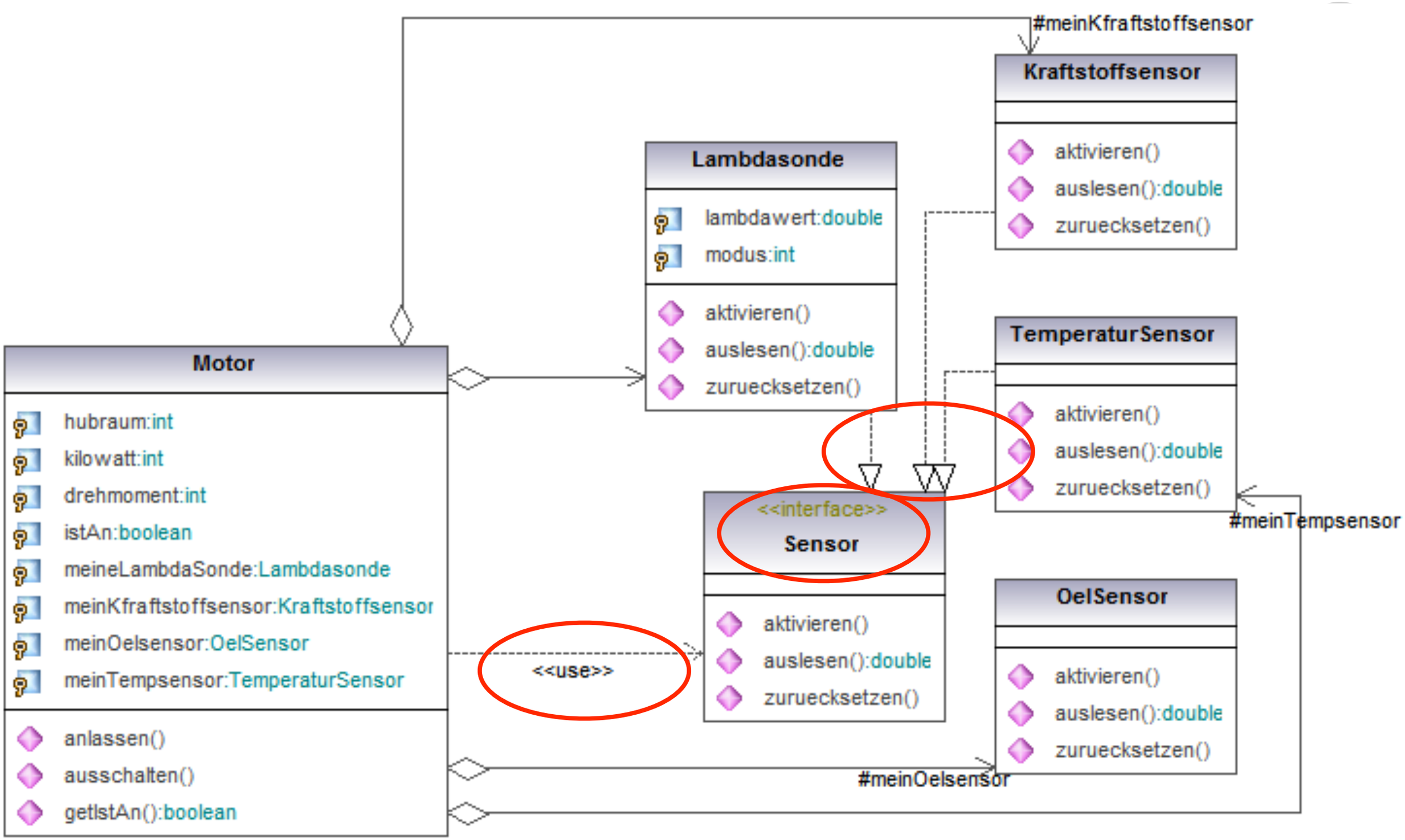
Beziehungen: Schnittstellen

- **Schnittstellen** definieren/**standardisieren** bestimmte Operationen und Verhaltensweisen, die verschiedene **schnittstellenkonforme Klassen implementieren**
- **Attribute** dabei **nicht** zwingend **gleich**, **Operationen** jedoch **schon**!
- Objekte **realisieren** oder **nutzen** Schnittstellen!
- Notation: Benutzen über Abhängigkeit mit „<<use>>“,
  Realisieren über „gestrichelten Vererbungspfeil“

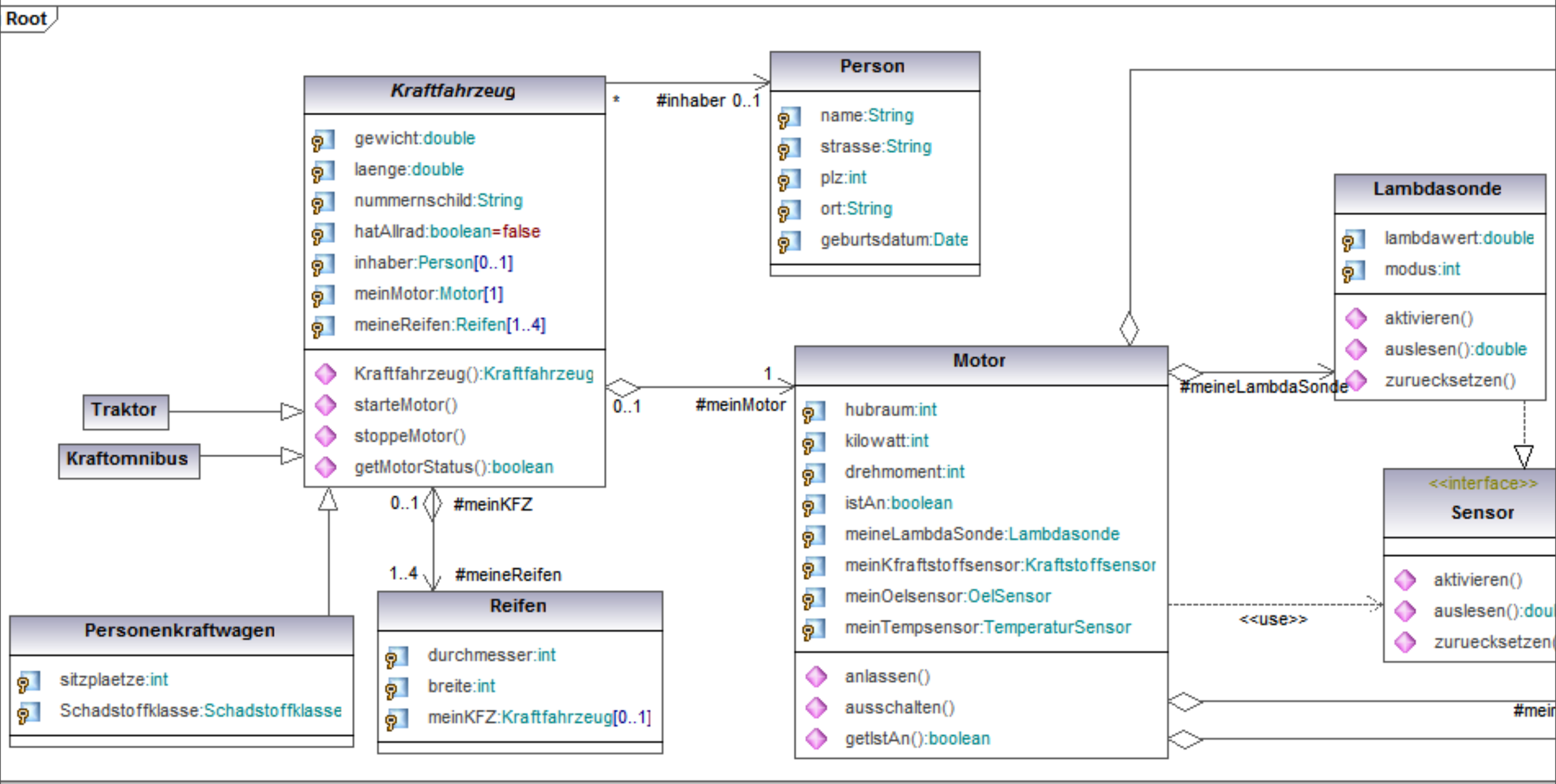
Beziehungen: Schnittstellen



Beziehungen: Schnittstellen

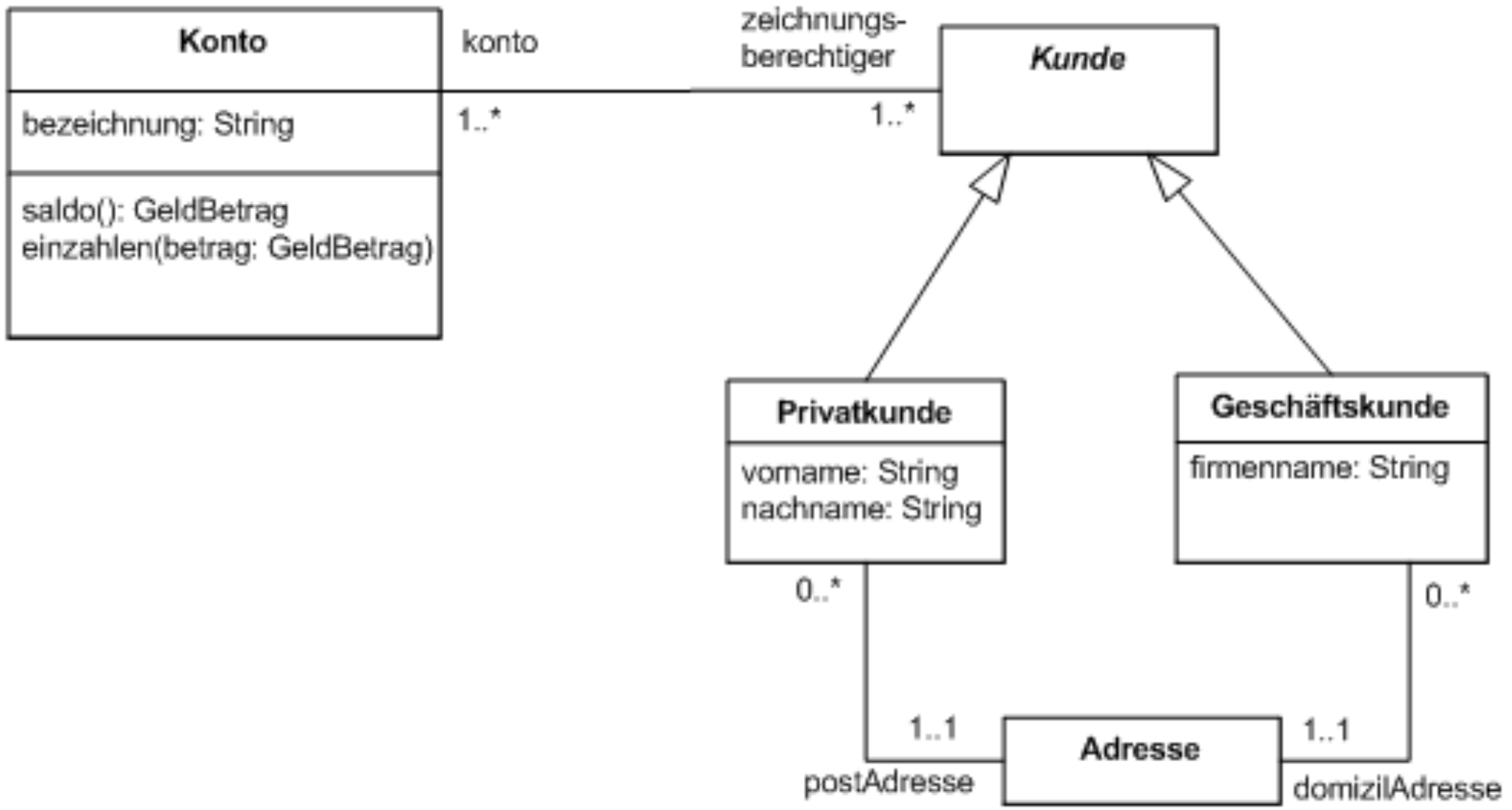


Real-World-Beispiele

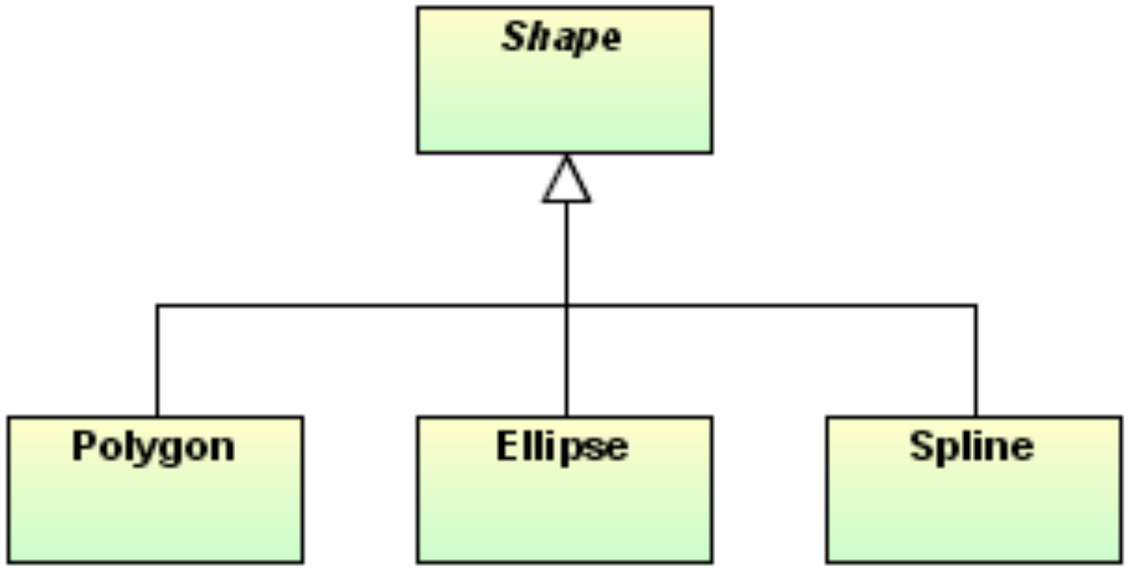


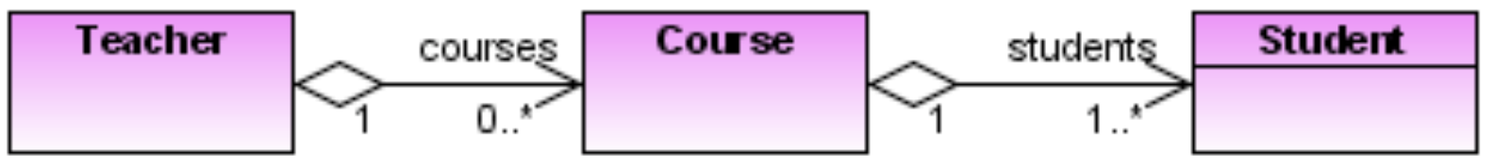
Generated

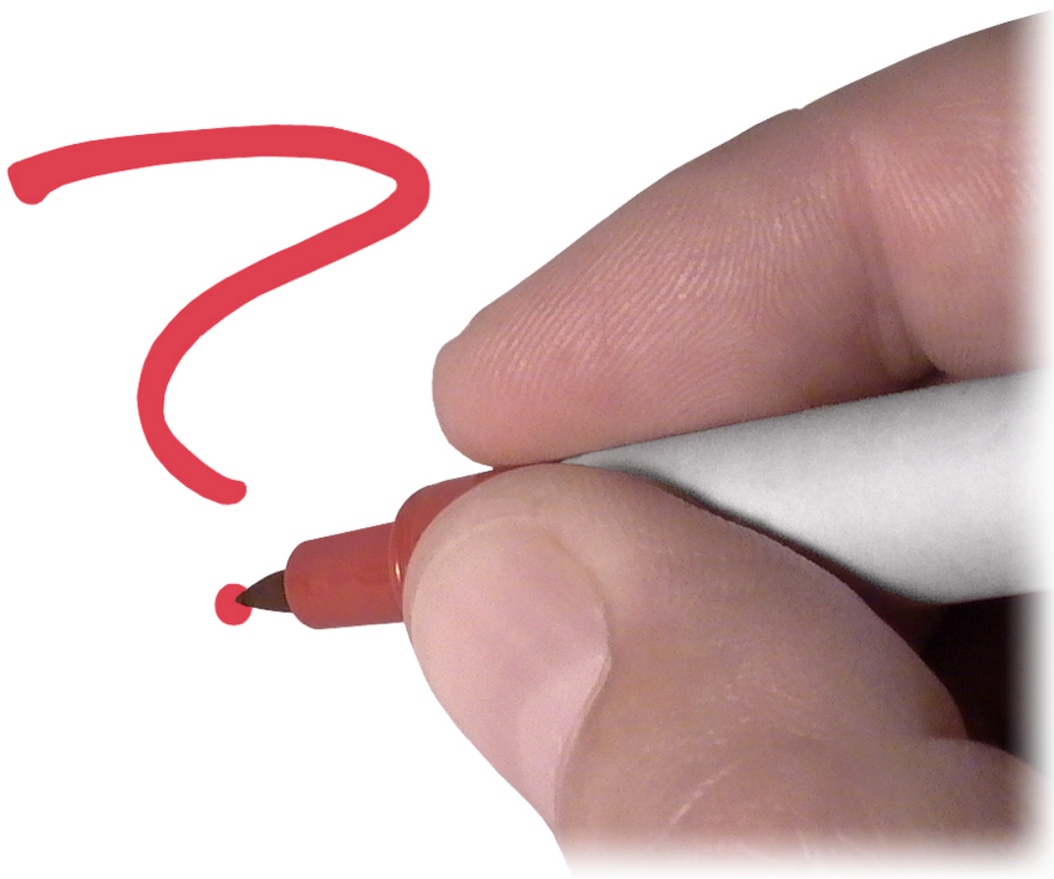
Real-World-Beispiele



Real-World-Beispiele







Fragen?

Quellen

[1] Website: The Current Official UML Specification; URL: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

[2] Website: Introduction to OMG UML; URL: http://www.omg.org/gettingstarted/what_is_uml.htm

[3] Buch: U. Schneider; D. Werner: Taschenbuch der Informatik, S. 204, Hanser (2007)

weiteres:

Website: List of UML tools - Wikipedia; URL: http://en.wikipedia.org/wiki/List_of_UML_tools

www.michael-whittaker.de

michael whittaker
webdesign. edv-dienstleistungen.