ModSoft

Modellbasierte Software-Entwicklung mit UML 2 im WS 2014/15

Teil II: Strukturmodellierung

Prof. Dr. Joachim Fischer
Dr. Markus Scheidgen
Dipl.-Inf. Andreas Blunk

fischer@informatik.hu-berlin.de

Inhalt

1. Klassen und Assoziationen (zweiter Eindruck)
2. Interface
3. Datentyp, Signal, Signalempfangsspezifikation
4. Klassen und Verhaltensbeschreibungen (erster Eindruck)

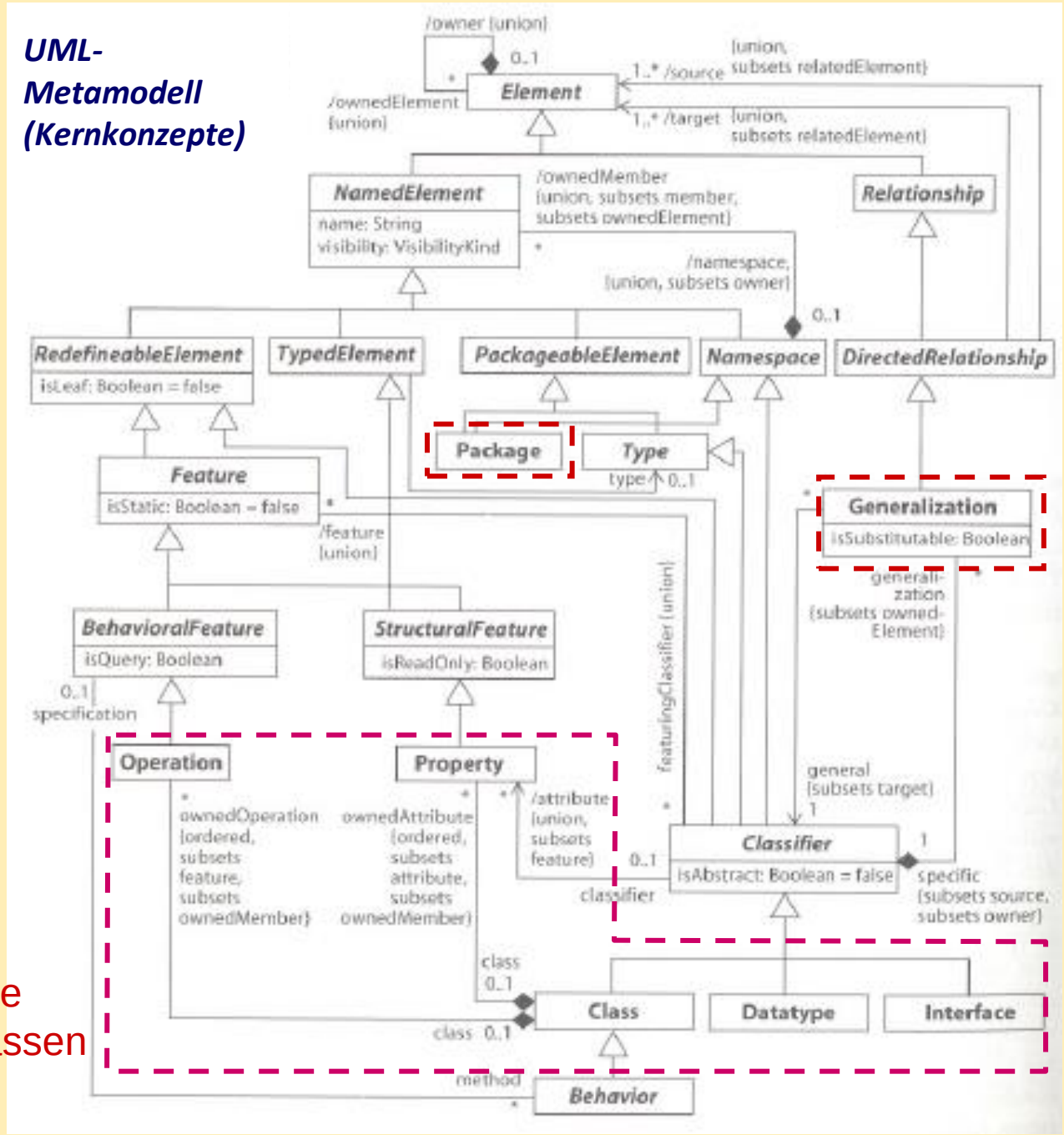
M3
MOF 2.0

M2
UML 2.0

M1
Benutzer-modell

M0
Objekte der Realität

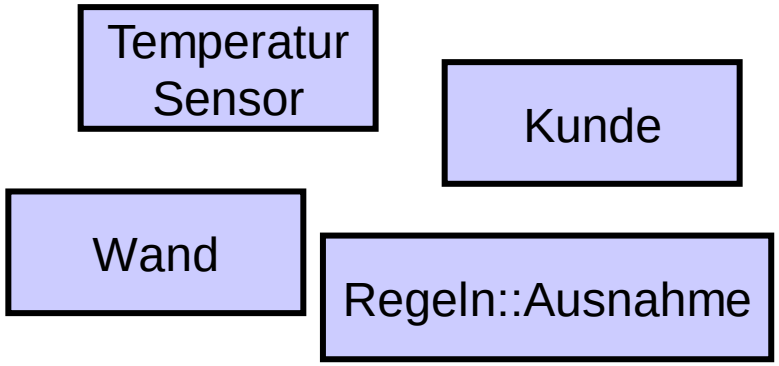
UML-Metamodell (Kernkonzepte)



Konkrete Metaklassen

Klassendiagramm (Wdh.)

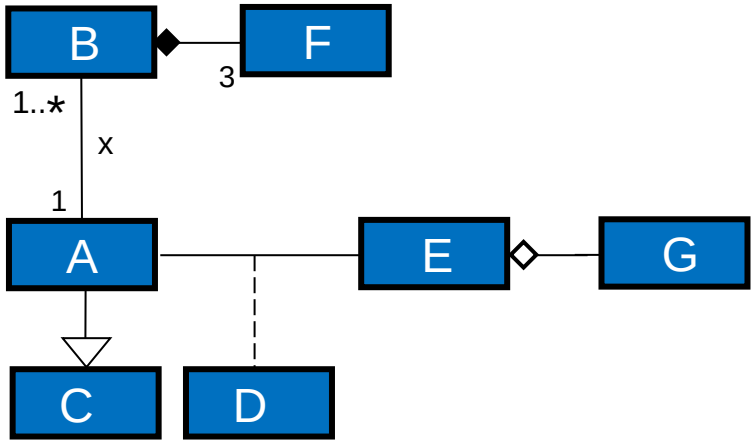
- … zeigt eine Menge von
  - Klassen
  - Interfaces
  - Kollaborationen (collaborations) und
  - deren Beziehungen untereinander
- Beziehungen:
  - Abhängigkeit,
  - Generalisierung und
  - verschiedene Assoziationen
- hat (wie jedes Diagramm) einen Namen
- kann (wie jedes Diagramm) Anmerkungen und Constraints enthalten



Paketname:: … ::Klassenname

<u>Namenskonvention</u>:
- beliebige Zeichenfolge (ohne Doppelpunkt)
- Empfehlung: Substantiv, Beginn mit Großbuchstabe
- 1-deutig in einem Kontext

Klassendiagramme (Wdh.)



- A steht mit B in einer Beziehung namens x
  jedem A-Objekt sind 1 bis beliebig viele B-Objekte zugeordnet,
  jedem B-Objekt ist genau ein A-Objekt zugeordnet
- A ist eine Spezialisierung von C
- D ist eine Assoziationsklasse, die die Beziehung zwischen A- und E-Objekten beschreibt
  jeder A-E-Link ist durch ein D-Objekt charakterisiert,
  dieses hat zwei Attribute: A-Referenz, E-Referenz
- Jedes E-Objekt besitzt ein G-Objekt (Aggregation)
  G-Objekt ist Teil von A (könnte aber gleichzeitig Teil von etwas anderem sein)
- Jedes B-Objekt besitzt 3 F-Objekte als (Komposition)
  F-Objekt kann (max.) nur Teil vom B-Objekt sein

Klassen und Objekte (Wdh.)

passive Klasse

aktive Klasse

Kunde

Kunde
name
titel
geburtstag
geschlecht

Kunde
name: **String**
titel: **String**
geburtstag: **Datum**
geschlecht: **Geschlecht**
alter(): **Integer**

Ticket
{abstract}

KommEinheit
partner: **KommEinheit**
«**signal**»
SigA(**integer**),
SigB(**float**)
op (**int**, **char**)

- beliebige **Detailierungsgrade** in der Darstellung einer Klasse möglich
  Tools sichern, dass in einem Namensraum keine Widersprüche zwischen Darstellungen einer Klasse auftreten

- Sichtbarkeit kann individuell eingeschränkt werden
  + public, # protected, - private, ~ package

Attribut-Belegung (Zustand zu einem Zeitpunkt) des Kunden-Objektes K1

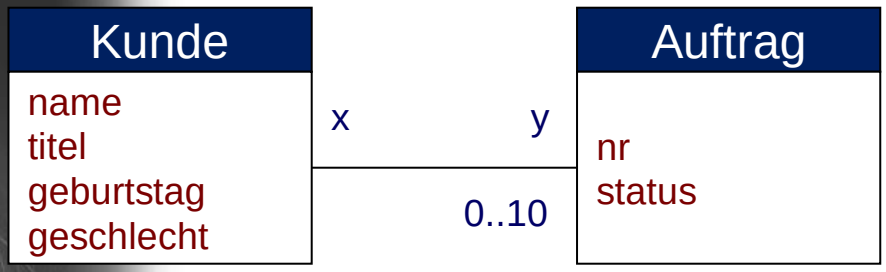
K1:Kunde
name = “Mustermann”
titel = “Dipl-Inf”
geburtstag= 10-09-1992
geschlecht= weiblich

:Kunde

anonymes Objekt

Dualität von Attributen und Assoziationsenden (Wdh.)

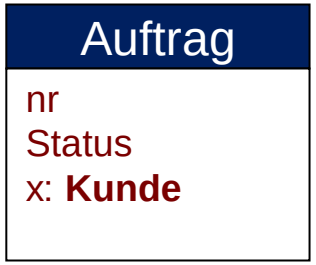
Sowohl **Attribut** als auch **Assoziationsende** sind im **Metamodell** Instanzen der Metaklasse **Property**



Lesart:
jedem Kunde-Objekt ist eine Kollektion y von Aufträgen zugeordnet

jedem Auftrag-Objekt ist genau ein Kunde-Objekt x zugeordnet

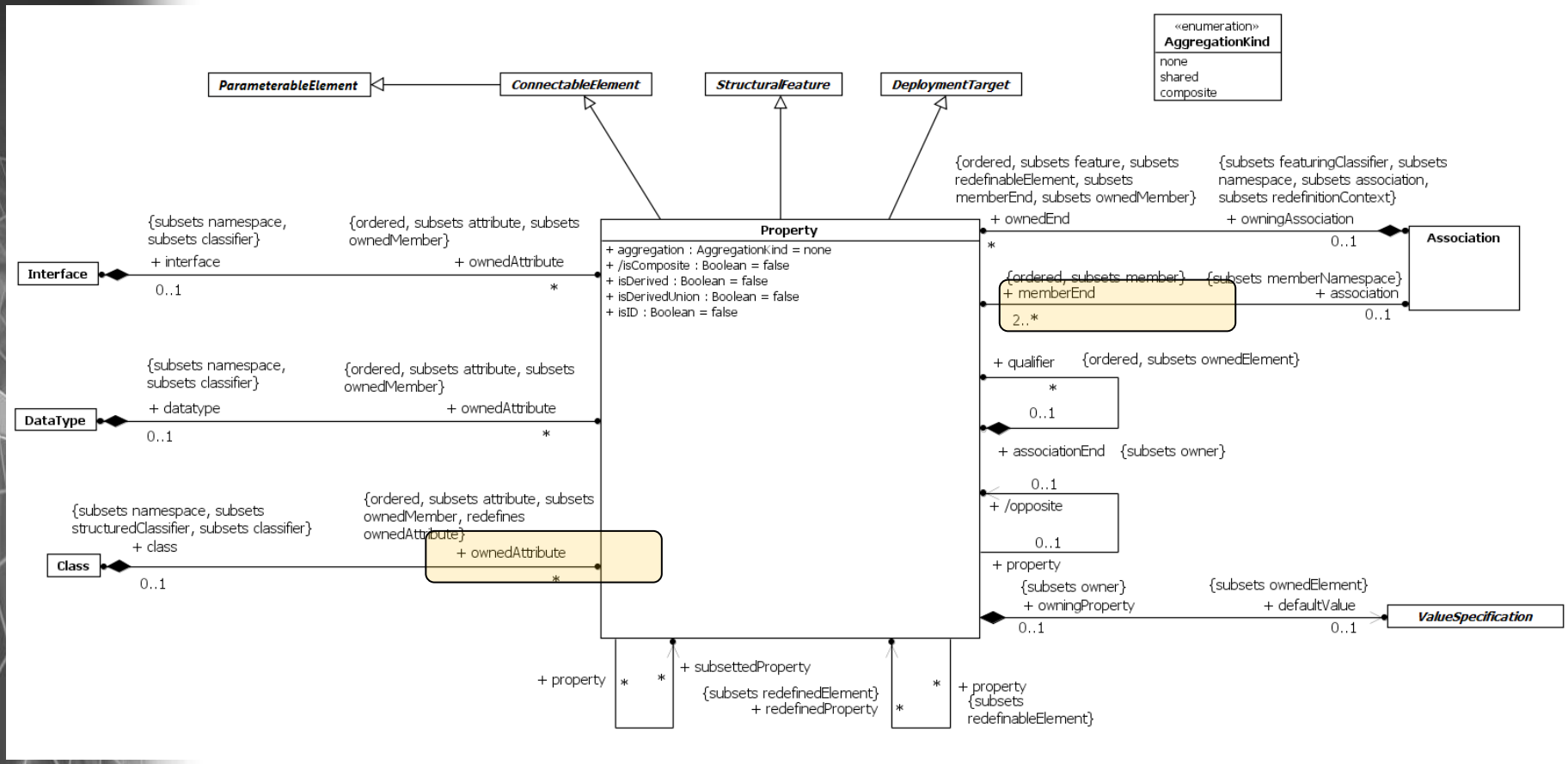
- *Das Assoziationsende x (vom Typ Kunde) dient der Navigation und entspricht implizit einem Attribut der Klasse Auftrag*
- *Das Assoziationsende y (vom Typ Menge über Auftrag) entspricht implizit einem Attribut von Kunde*



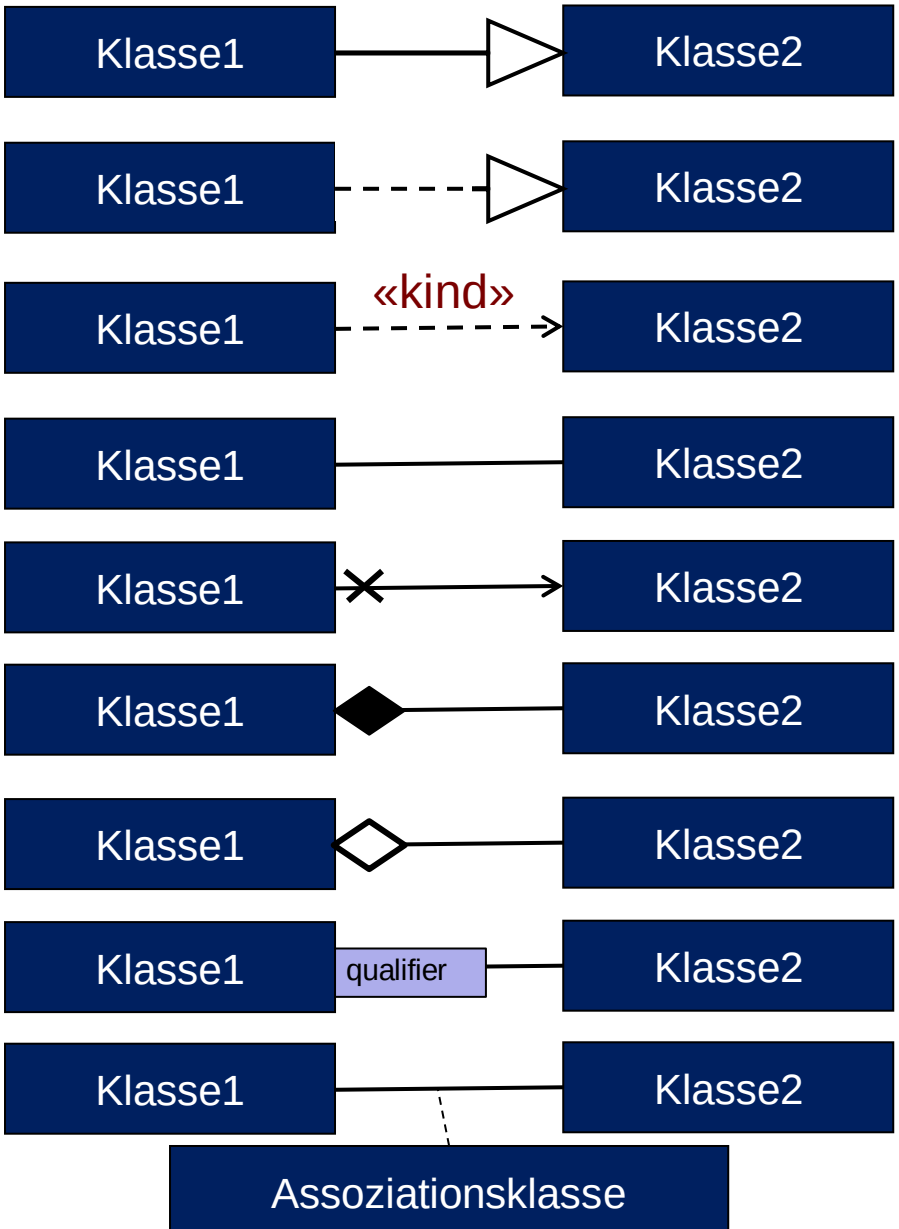
Kardinalität als Eigenschaften von Attributen/Assoziationsenden

falls Assoziationsenden nicht benannt worden sind (also wenn Angabe y fehlt)

Dualität von Property: Rolle als Attribut und Assoziationsende



Beziehungen *zwischen Klassen bzw. Objektmengen*



Spezialisierung

Realisierung

Abhängigkeit
(viele Spezialfälle)

Assoziation

gerichtete Assoziation

Komposition

Aggregation

qualifizierte Assoziation

attributierte Assoziation

Assoziation (Wdh.)

- beschreibt als Relation die gemeinsame Struktur und Semantik von Mengen von Objektverbindungen der beteiligten Klassen

- Objektverbindungen (Links) sind Instanzen einer Assoziation (eineAusprägung von Classifier)

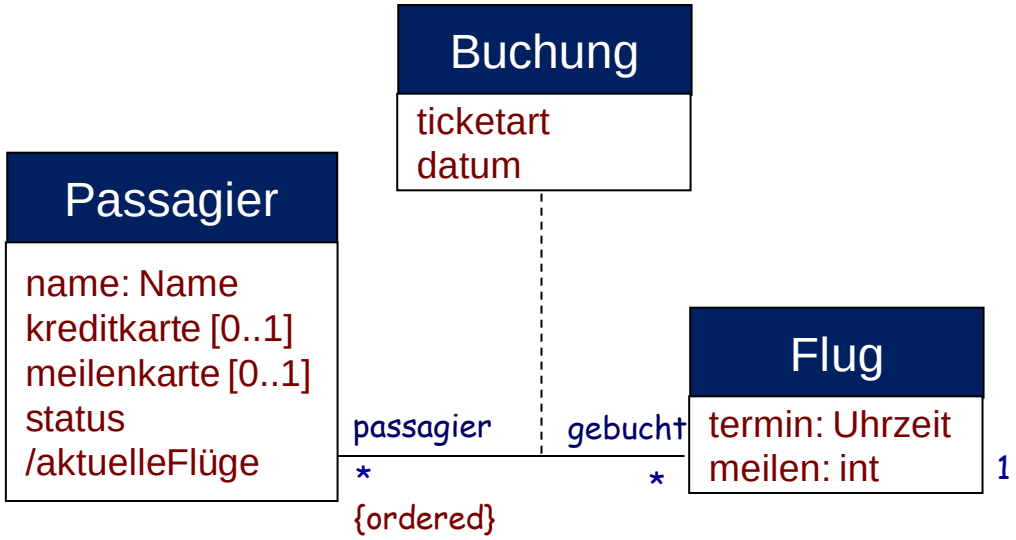
- Eine Assoziation kann einen Namen haben
  (beschreibt worin oder weshalb eine Beziehung besteht)

- Die Enden tragen immer einen Namen
  (beschreiben die Rollen, die die Objekte der Klassen am jeweiligen Ende spielen)
  Ist kein Name angegeben, wird der kleingeschriebene Klassenname angenommen

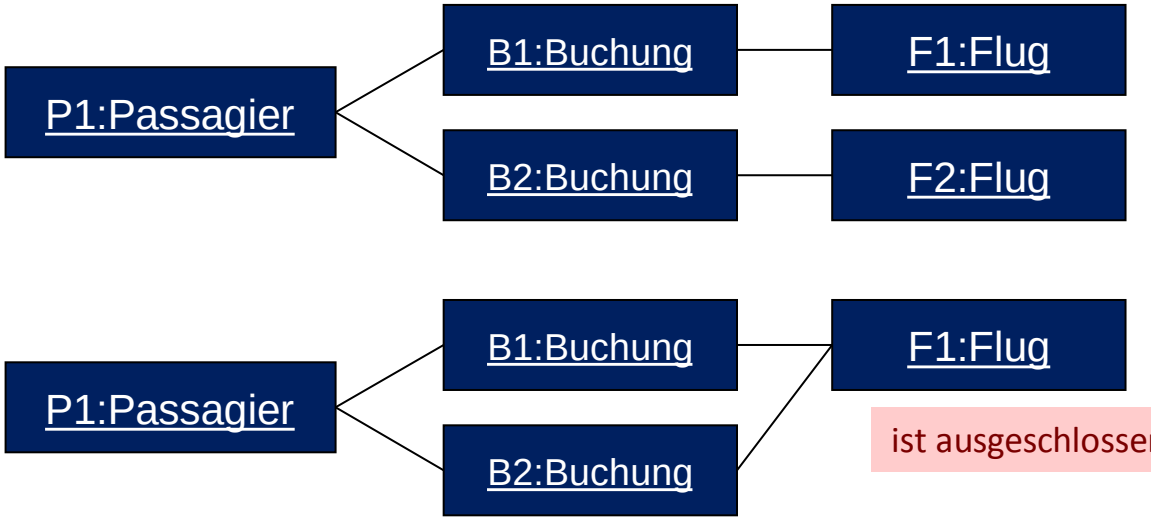
- Für jede Rolle kann eine Reihe zusätzlicher Angaben gemacht werden:
  - Multiplizität (Standardwert=1),
  - Sichtbarkeit,
  - Eigenschaften

- Dualität von Assoziationsende und Attribut (jedes Rollenende könnte ein Attribut der gegenüberliegende Klasse oder Attribut einer evtl. Assoziationsklasse darstellen)

Assoziationsklasse (Wdh.)

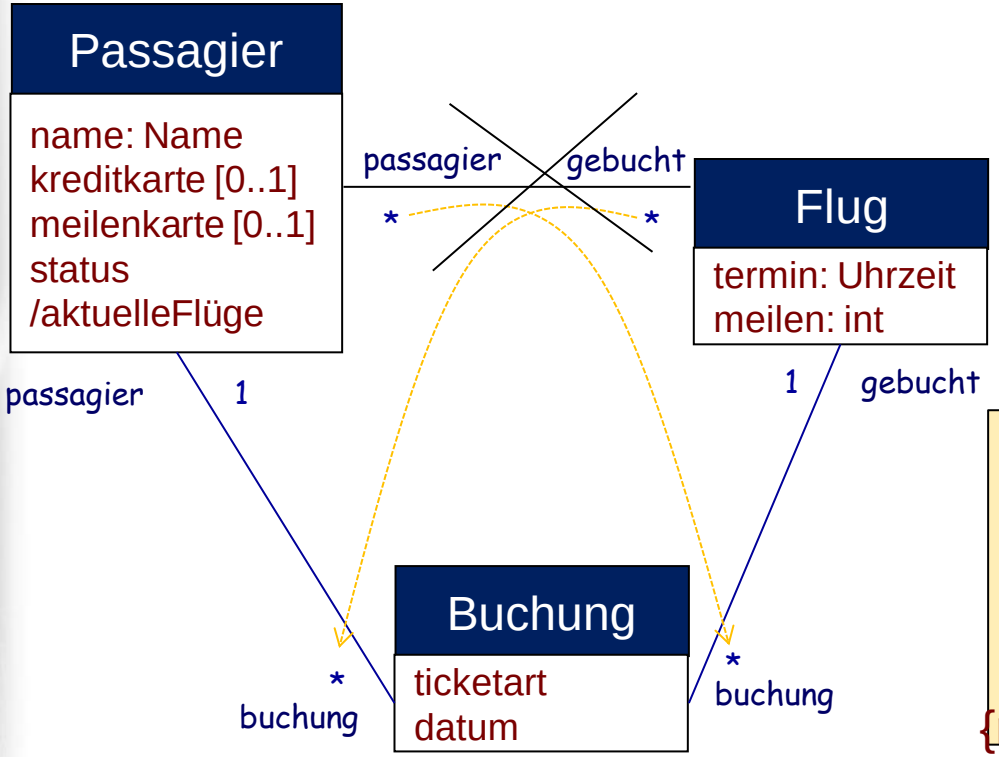


jedes Verbindungspaar ist einmalig



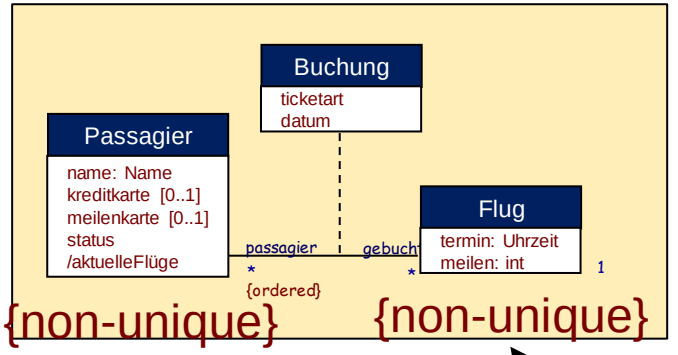
ist ausgeschlossen

Ersetzung: Assoziationsklasse → Normale Klasse



Frage: semantsich äquivalent?

jetzt: semantsich äquivalent,
aber die Semantik wollte man nicht



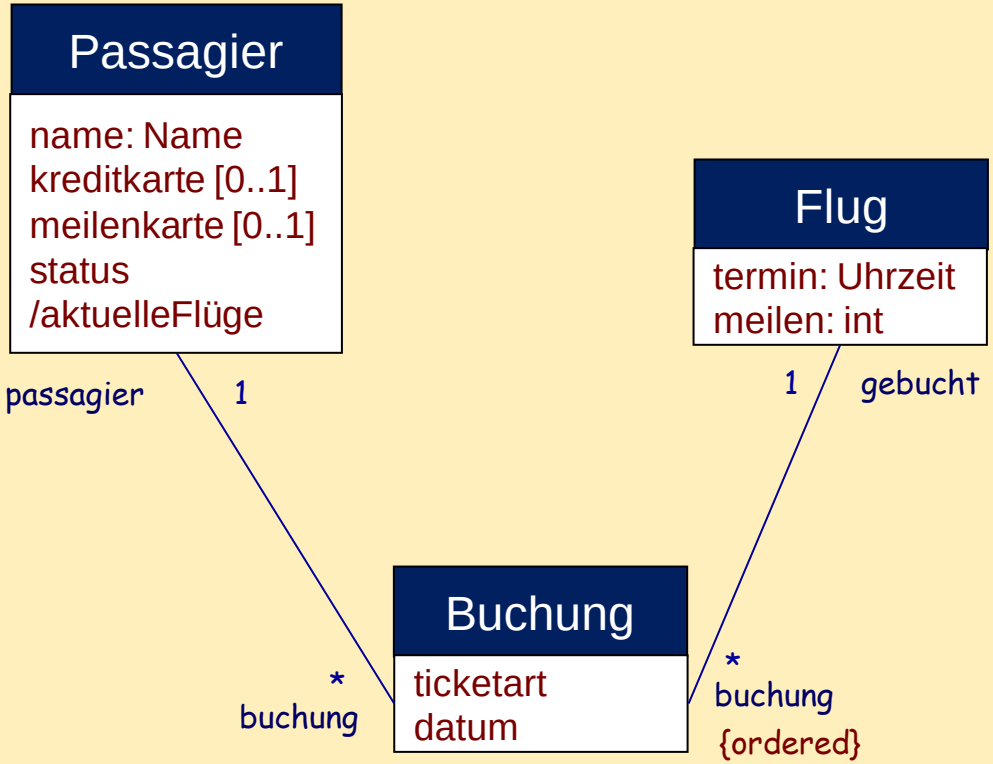
- ein Flug kann beliebig oft gebucht werden
- jede Buchung gehört zu genau einem Passagier
- damit kann für einen Flug mehrere Buchungen desselben Passagiers ausgelöst werden

➔ Die Assoziationsklasse verhinderte das

Man könnte die attributierte Assoziation aber anpassen

Ersetzung: Assoziationsklasse → Normale Klasse

context Flug: buchung ->collect(Passagier)-> **forAll** (p1,p2: Passagier (p1 <>P2)



- ein Flug kann von einem Passagier nicht mehr als einmal gebucht werden

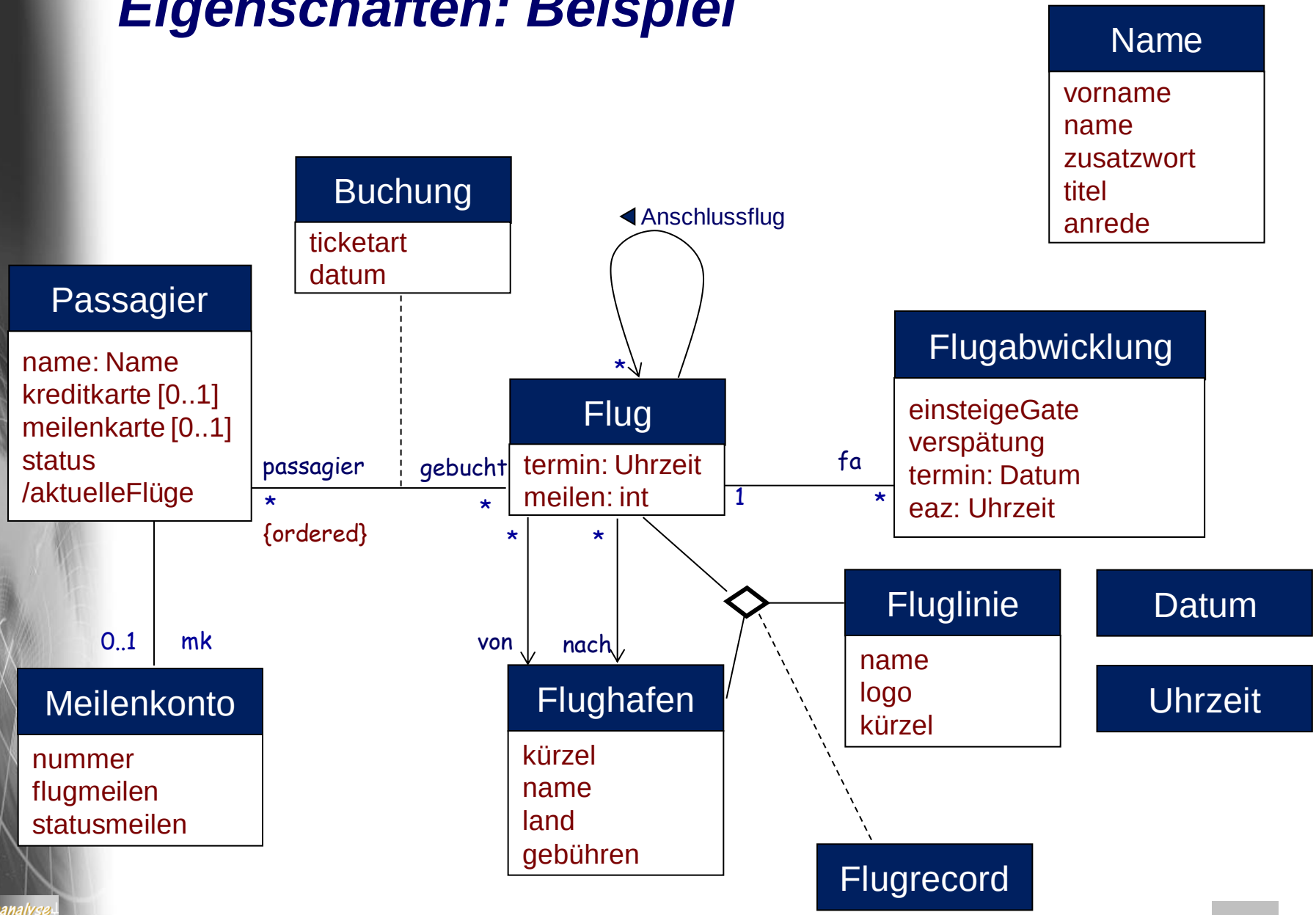
Liste möglicher Eigenschaften von Properties

- *… für Assoziationsenden (auch für Attribute) in {…} als Constraint*
  - / (abgeleitete Assoziation)
  - readonly ← → unrestricted
  - composite
  - redefines Attr
  - subsets Attr
  - union
  - unique ← → non-unique
  - ordered ← → unordered
  - seq (Vor.: Multiplizität > 1)
  - bag (Vor.: Multiplizität > 1)

default

bedingt kombinierbar

Eigenschaften: Beispiel



Zwischenfazit

UML

- Universelle Modellierungssprache mit
  - informal definierter dynamischer Semantik
  - semi-formal definierter statischer Semantik (UML und OCL)

  → UML bleibt damit nur gedanklich ausführbar

- Dynamische Semantik
  - enthält sog. semantische Variationspunkte
    (z.T. alternative semantische oder offene semantische Festlegungen)
- → Problem für Compiler (Zielcode, Simulatorcode)

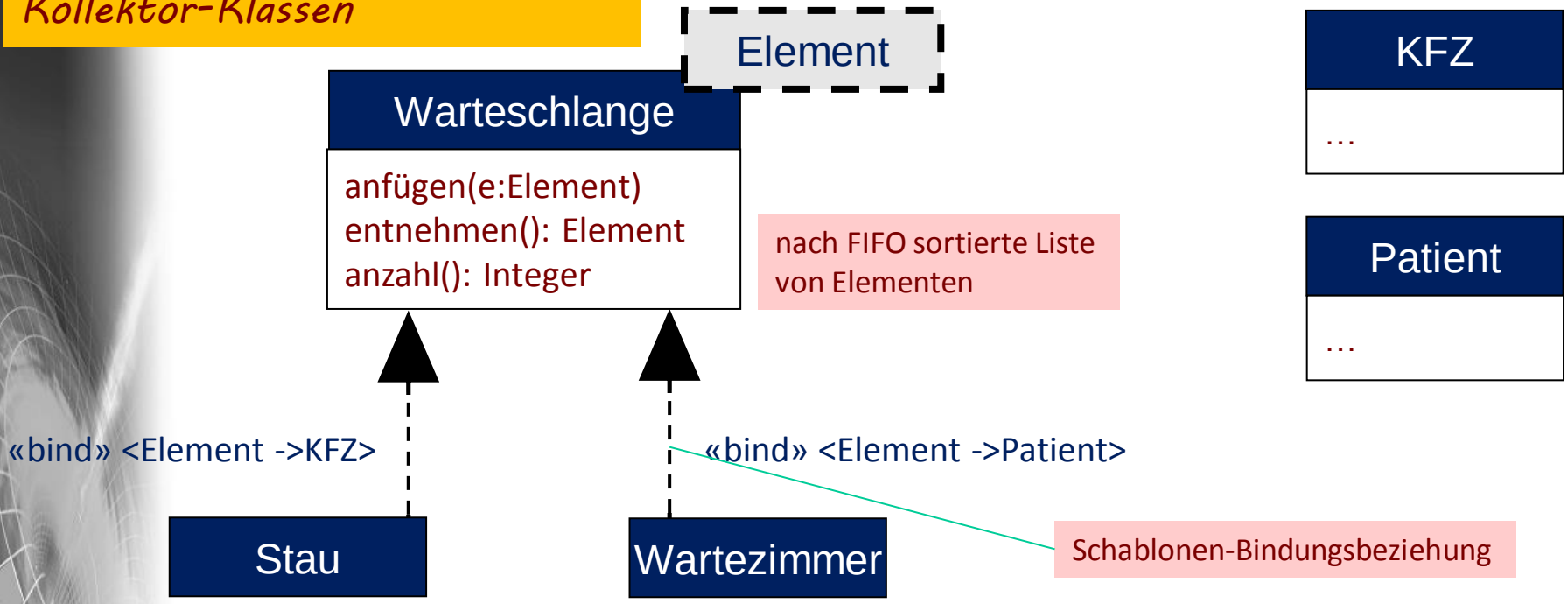
- Action-Sprache
- - befindet sich in Entwicklung
  - hat Einfluss auf Datentypen in UML (Definition von Literalen, Operatoren offen)
  - bisher DataTypes in UML mit Magie

- Pragmatischer Ansatz
  - SDL-ASN.1-UML-Compiler (HU Berlin) benutzt C als Action-Sprache
  - Workflow-Vorlesung benutzt SLX als Actionsprache
    (notiert in Form von Kommentaren)

Parametrisierte Klasse / Template

typischer Anwendungsfall sind Kollektor-Klassen

bereits existierende Klassen



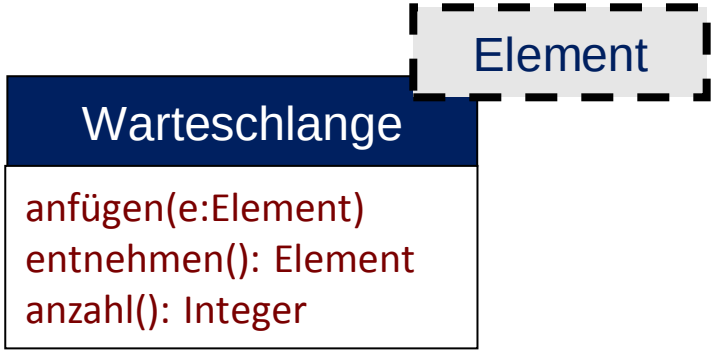
Resultat der Bindung: homogene Listen von KFZ-Objekten bzw. Patient-Objekten

st: Stau

wz: Wartezimer

FRAGE: Wie lässt sich eine heterogene Liste definieren ?

- mehrere Typangaben als Parameter möglich
- Verwendung als Typangabe für
  Attribute, Operationsparameter



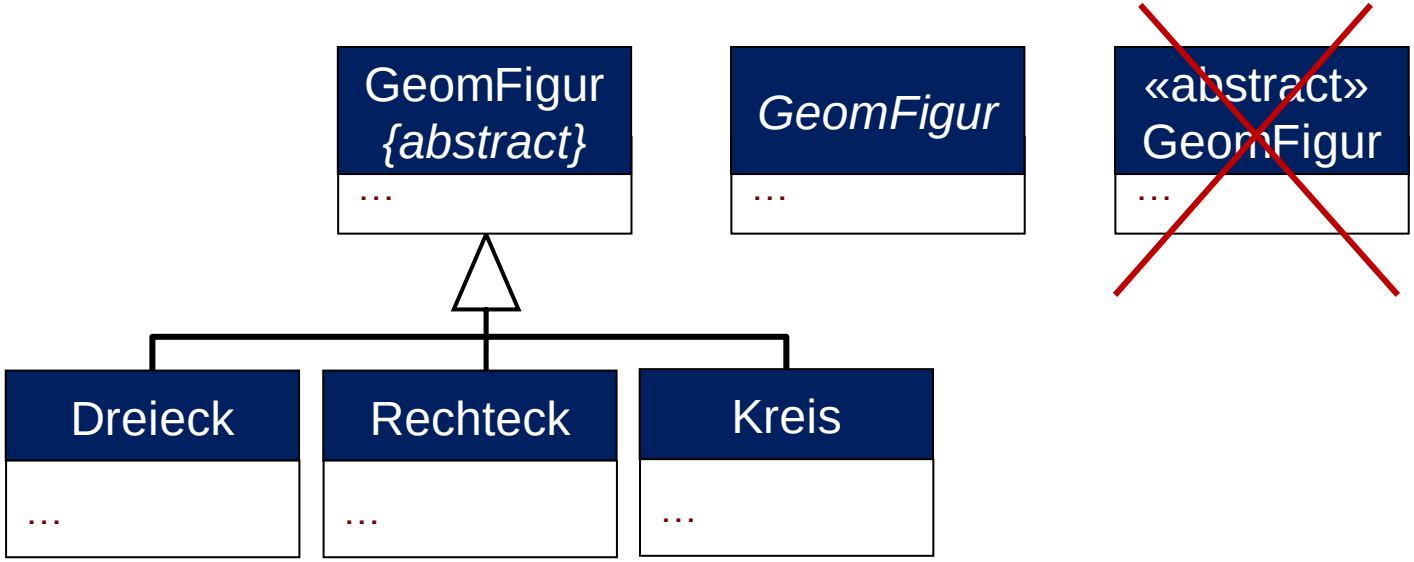

```
Template < class Element>
class Warteschlange {
        public:
                void anfügen(Element* e)
                Element * entnehmen()
                int anzahl()
};
```

Abstrakte Klassen

- nicht vollständig definiert, nicht instanziierbar
  meist fehlt die Implementiertung einer oder mehrerer Operationen
  (C++ pure virtual function)

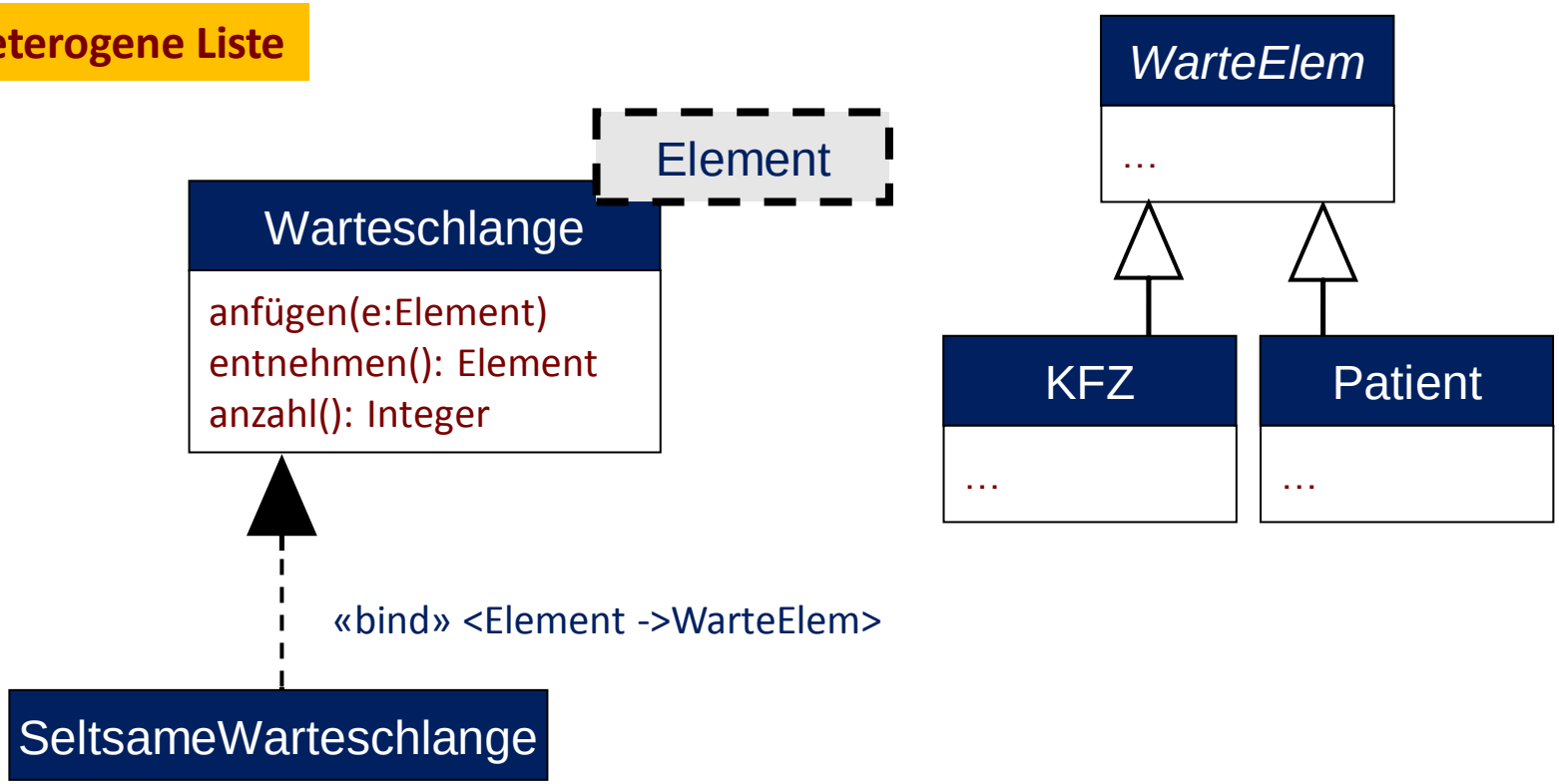
- eine abstrakte Klasse, die keine Spezialisierungen (Unterklassen/ Ableitungen) besitzt ist sinnlos

- häufiger Einsatz: Generalisierung vorhandener Konzepte



Templates mit abstrakter Klasse als Typ-Parameter

Heterogene Liste



Resultat der Bindung: heterogene Listen von KFZ-Objekten oder/und Patient-Objekten

swz: SeltsameWarteschlange

```
swz.anfügen (new KFZ);
swz.anfügen (new Patient);
```

Aktive Klasse

- beschreiben **Reaktive Komponenten**
  ~ eigenständiger Lebenslauf
  Reaktion auf ein Siagnal: ausführung einer Aktivität

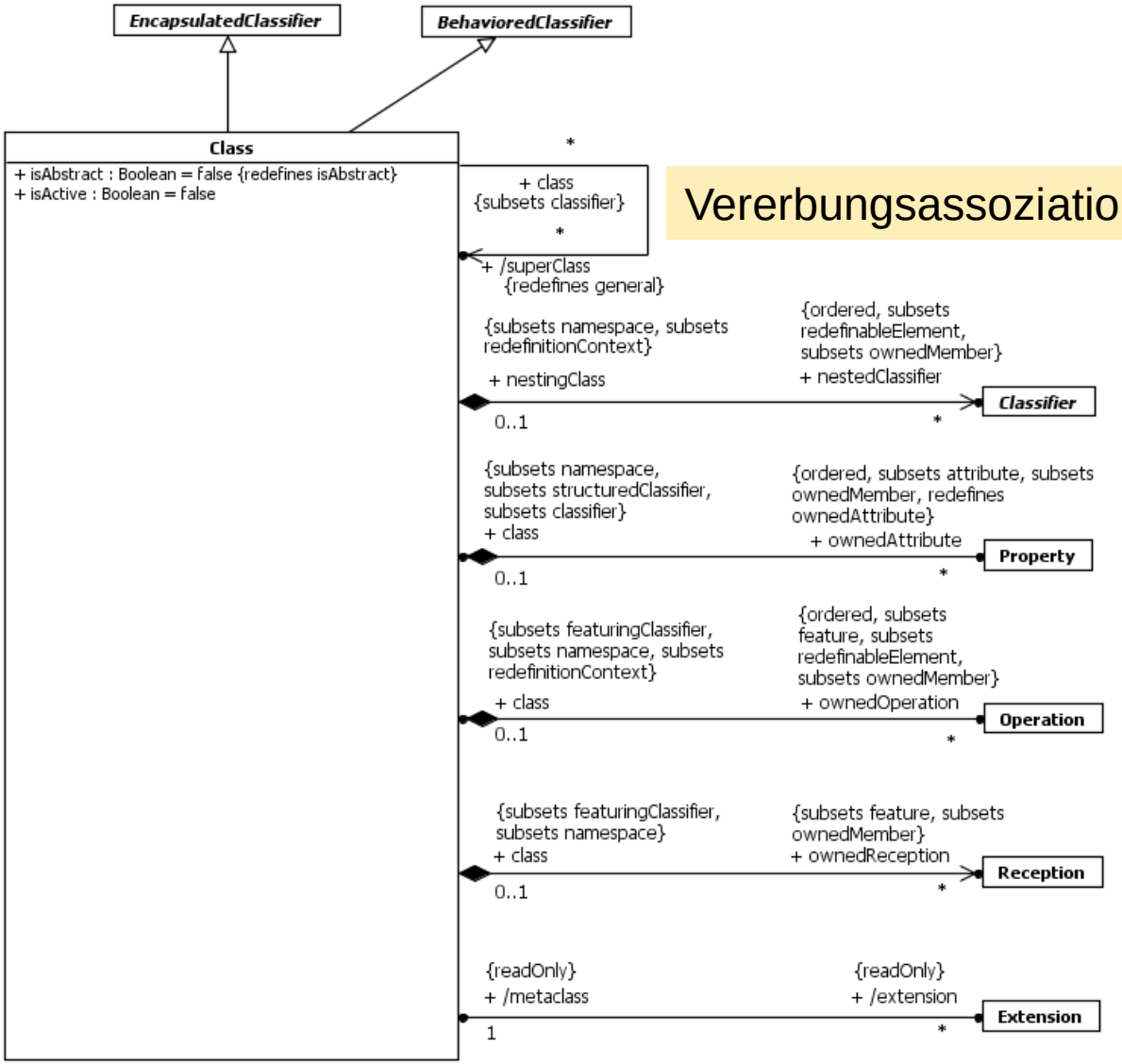
- **Anwendung**:
  - SW-Modellierung:
    Eingebettete Systeme, Echtzeitsysteme
  - Allgemeine Modellierung:
    eigenständig agierende Komponenten (Agenten, Prozesse, …)

Klasse, definiert im UML-Metamodell

- **Class** is the concrete realization of EncapsulatedClassifier and BehavioredClassifier.

- The **purpose** of a Class is to specify a classification of objects and to specify the Features that characterize
  - the structure and
  - behavior of those objects.

Class – abstrakte Syntax, *definiert im UML-Metamodell*



Vererbungsassoziation

lokale Classifier

Attribute

Operation

Signale

Stereotypes

Class-Semantik, definiert im UML-Metamodell (1)

- Class is a kind of **EncapsulatedClassifier**
  whose Features are Properties, Operations, Receptions, Ports and Connectors.

- **Attributes** of a Class are Properties that are owned by the Class. Some of these attributes may represent the ends of binary Associations.

- **Objects** of a Class must contain values for each attribute that is a member of that Class, in accordance with the characteristics of the attribute, for example its type and multiplicity.

- When an object is instantiated in a Class, for every attribute of the Class that has a specified default, if an initial value of the attribute is not specified explicitly for the instantiation, then the **default ValueSpecification** is evaluated to set the initial value of the attribute for the object.

- **Operations** of a Class can be **invoked** on an object, given a particular set of values for the parameters of the Operation, according to the semantics specified in 9.6.3.

- A Class cannot access **private Features** of another Class, or **protected Features** on another Class that is not its ancestor.

Class-Semantik, definiert im UML-Metamodell (2)

- A Class cannot access **private Features** of another Class, or **protected Features** on another Class that is not its ancestor.

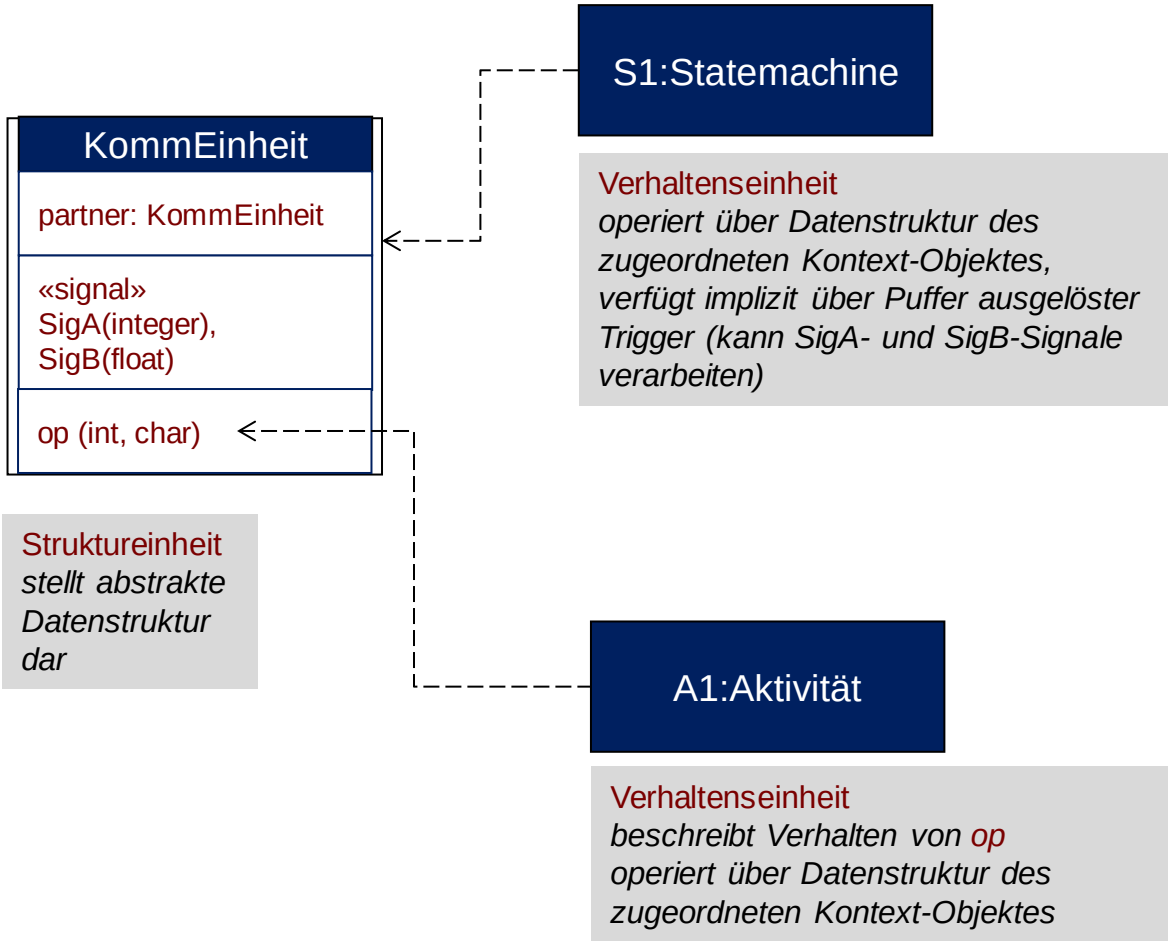
- A Class acts as the **namespace** for various kinds of Classifiers defined within its scope, including Classes. Nested Classifiers are members of the namespace of the containing Class. Classifier nesting is used for reasons of information hiding.

- A Class may be designated by setting **isActive** to true as active (i.e., each of its instances is an active object). When isActive is false the Class is passive (i.e., each of its instances executes within the context of some other object).

- An **active object** is an object that, as a direct consequence of its creation, commences to execute its classifierBehavior, and does not cease until either the complete Behavior is executed or the object is terminated by some external object. (This is sometimes referred to as “the object having its own thread of control.”) The points at which an active object responds to communications from other objects is determined solely by the Behavior of the active object and not by the invoking object. If the classifierBehavior of an active object completes, the **object is terminated**.
- Zustandsmaschine muss dazu in Terminated-Endzustand gehen, Stop reicht nicht.

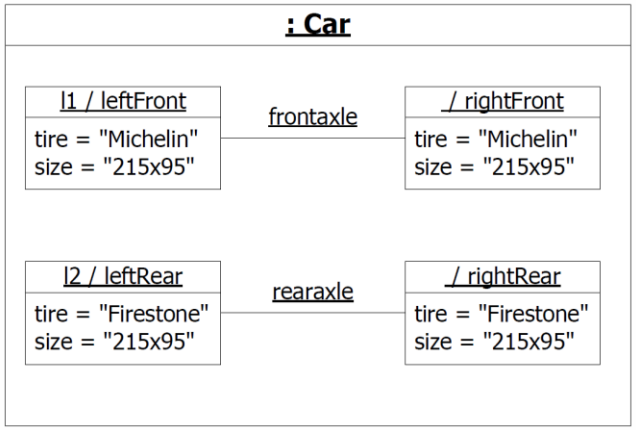
Class-Semantik, definiert im UML-Metamodell (3)

- A Class's **Receptions** specify which Signals the instances of this Class handle.

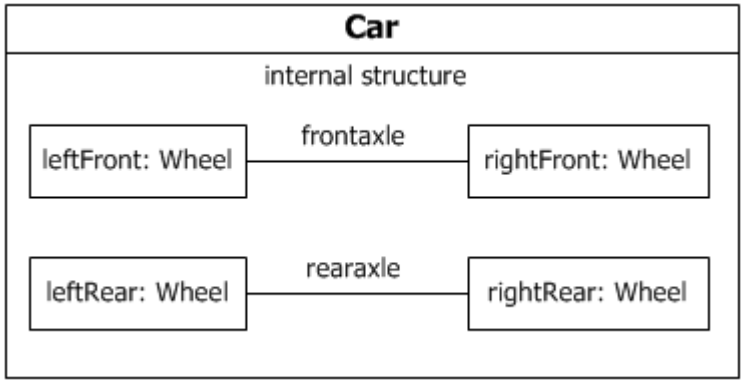


Class-Semantik, definiert im UML-Metamodell (4)

- An InstanceSpecification may be used to specify the **initial value** to be created for a Class.



- All instances corresponding to **parts** and **ports** of a Class are **destroyed recursively**, when an instance of that Class is deleted.



Class-Notation, definiert im UML-Metamodell (1)

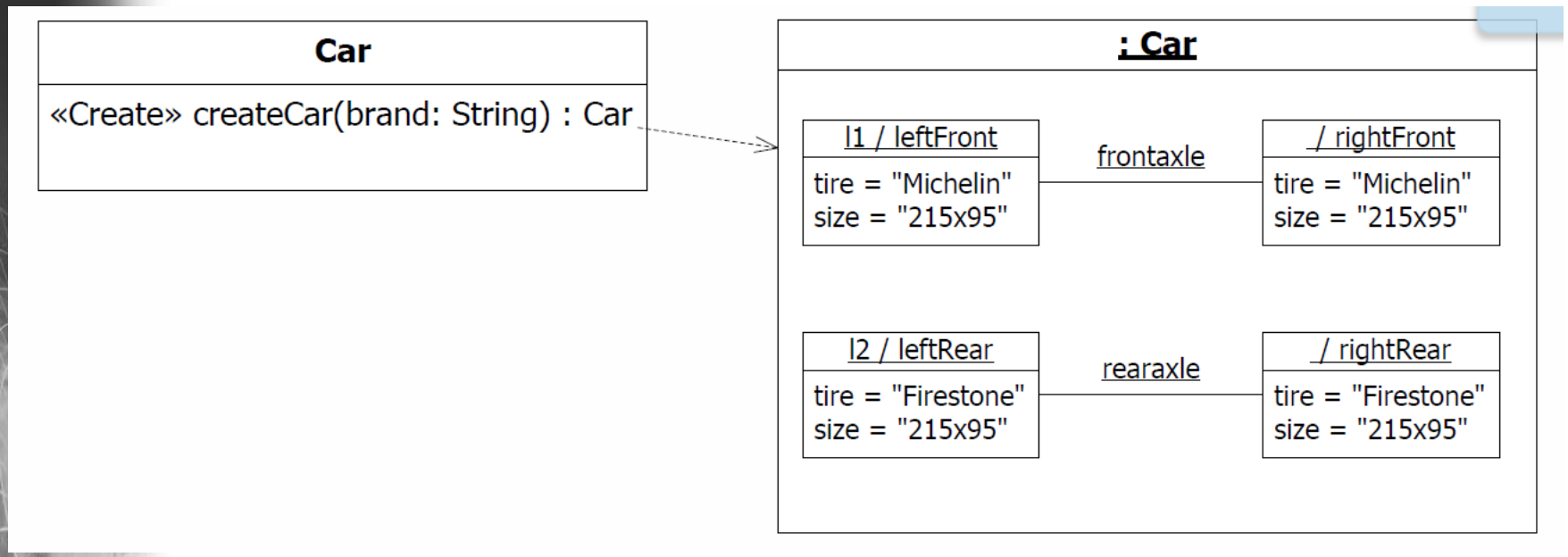
- A Class is shown using the **Classifier symbol**. As Class is the most widely used Classifier, **no keyword** is needed to indicate that the metaclass is Class.

- A Class has four mandatory **compartments**: attributes, operations, receptions (see 9.2.4) and internal structure (see 11.2.4). A Class may also have optional compartments as described for Classifiers in general (see 9.2.4).

- The **operations compartment** of a Class contains notation for its ownedOperations using the notation specified in 9.6.4. The receptions compartment contains ownedReceptions using the notation specified in 10.3.4.

- A **usage dependency** may relate an InstanceSpecification to a constructor for a Class, describing the single value returned by the constructor Operation. The Operation is the client, the created instance the supplier. The InstanceSpecification may reference parameters declared by the Operation. A **constructor** is an Operation having a single return result parameter of the type of the owning Class, and marked with the standard stereotype «Create». The InstanceSpecification that is the supplier of the usage dependency represents the default value of the single return result parameter of a constructor Operation.

Class-Notation, definiert im UML-Metamodell (2)

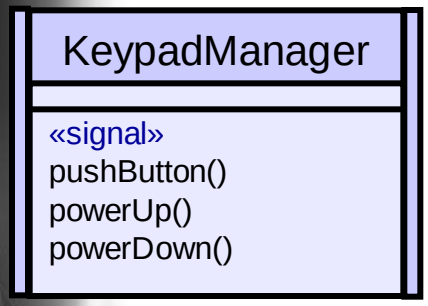


- A Class with the Property **isActive = true** can be shown by a Class box with an additional vertical bar on either side.

- A Class that represents a metaclass may be extended by the optional **stereotype «Metaclass»** (see StandardProfile in clause 22) shown above or before its name.

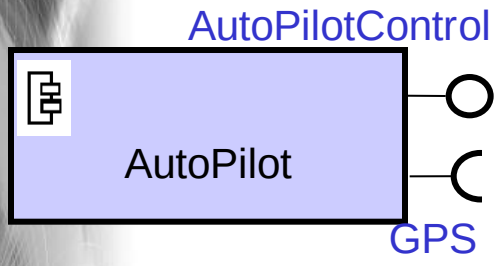
Beispiele: aktive Klasse, Komponente

~ ähnlich zur Entität einer Klasse



Aktive Klasse (active Class)

- beschreibt spezifische Klassen, deren Objekte selbst Auslöser von Aktionen sind
- Objekte können untereinander synchron oder asynchron Signale austauschen
- erlaubte Empfangssignale lassen sich spezifizieren



Komponente (Component)

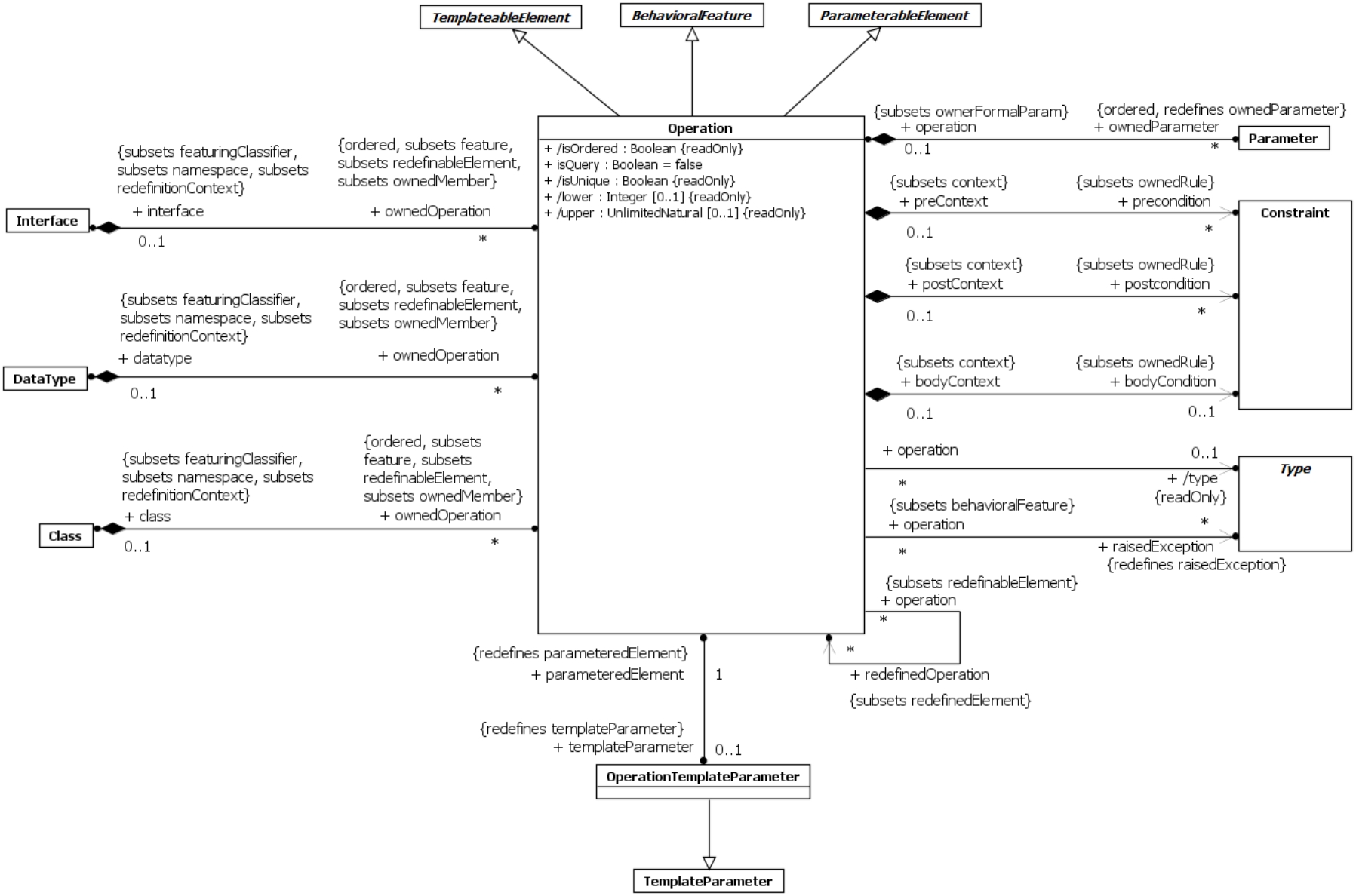
- modularer Teil eines Systems, dessen Implementierung sich hinter einer Menge von externen Interfaces verbirgt
- Systemkomponenten mit gleichen Interfaces sind austauschbar (Problem besteht dennoch mit Verhaltenskompatibilität !!!)
- Komponente ist die Zusammenfassung aller seiner beschreibenden und ausführbaren Artefakte

Eigenschaftswerte von Modellelementen

- Von Properties
  … kennen wir bereits

- Von Operationen
  - IsAbstract { isAbstract = true}
  - IsQuery …
  - IsStatic *Unterstrichener Op-Name*

  - Überschreiben geerbter Operationen
    { redfines foo}


TemplateableElement
BehavioralFeature
ParameterableElement
Operation
+ /isOrdered : Boolean {readOnly}
+ isQuery : Boolean = false
+ /isUnique : Boolean {readOnly}
+ /lower : Integer [0..1] {readOnly}
+ /upper : UnlimitedNatural [0..1] {readOnly}
Interface
{subsets featuringClassifier, subsets namespace, subsets redefinitionContext}
+ interface
0..1
{ordered, subsets feature, subsets redefinableElement, subsets ownedMember}
+ ownedOperation
*
DataType
{subsets featuringClassifier, subsets namespace, subsets redefinitionContext}
+ datatype
0..1
{ordered, subsets feature, subsets redefinableElement, subsets ownedMember}
+ ownedOperation
*
Class
{subsets featuringClassifier, subsets namespace, subsets redefinitionContext}
+ class
0..1
{ordered, subsets feature, subsets redefinableElement, subsets ownedMember}
+ ownedOperation
*
{subsets ownerFormalParam}
+ operation
0..1
{ordered, redefines ownedParameter}
+ ownedParameter
*
Parameter
{subsets context}
+ preContext
0..1
{subsets ownedRule}
+ precondition
*
Constraint
{subsets context}
+ postContext
0..1
{subsets ownedRule}
+ postcondition
*
{subsets context}
+ bodyContext
0..1
{subsets ownedRule}
+ bodyCondition
0..1
+ operation
*
0..1
+ /type
{readOnly}
Type
{subsets behavioralFeature}
+ operation
*
*
+ raisedException
{redefines raisedException}
{subsets redefinableElement}
+ operation
*
*
+ redefinedOperation
{subsets redefinedElement}
{redefines parameteredElement}
+ parameteredElement
1
{redefines templateParameter}
+ templateParameter
0..1
OperationTemplateParameter
TemplateParameter

Klasse (Zusammenfassung)

- … ist eine Instanz der Metaklasse class
  als konkrete Spezialisierung der Metaklasse Classifier
- … wird als Rechteck dargestellt, Name als Substantiv
- … verkörpert einen Typ
  (andere Beispiele Datentyp, Signal, Interface, Behaviour(Verhalten), …)
- … enthält die Beschreibung von Struktur u. Verhalten von Objekten,
  die sie erzeugt oder
  die mit ihr erzeugt werden können
- … setzt sich aus
  - Attributen, Operationen, Signalempfangsspezifikationen, lokale Elemente und Invarianten zusammen
- Verhalten der Objekte wird durch Signale beeinflusst,
  die den Objekten bekannt sind (verstanden werden)
  - möglich sind: Signalempfänge (Nachbildung von Operationsrufen),
  - zu jeder “verständlichen” Nachricht gehört eine passende Operation
- Nicht-verstandene Nachrichten
  werden ignoriert bzw. lösen Ausnahmen aus
- Zusicherungen (als Invarianten formuliert)
  bestimmen einzuhaltende Regeln bei der Ausführung von Operationen oder
  allgemein für die Einhaltung von Objekteigenschaften