Fachtagung Zentralabitur 2012
Fachworkshop Informatik

Klaus Dingemann
Bezirksregierung Münster
Jan Vahrenhold
Technische Universität Dortmund

Übersicht

Ablaufplan:

- Vorstellung der Beteiligten.
- Klassen für lineare Strukturen und Baum-/Graphstrukturen.
  - Klassen `List`, `BinaryTree`, `BinarySearchTree`, `Graph`.
- Automaten/Grammatiken (kurz), Datenbanken (kurz).
- Klassendiagramme.
  - Assoziationen, Terminologie.
  - Entwurfs- und Implementierungsebene.
- Entwurf von Aufgaben.
- Besprechung der Ergebnisse.

Kommentare:

- Die hier (und in den Vorgaben) besprochenen Anforderungen sind Minimalanforderungen für den Unterricht, stellen aber gleichzeitig auch die Maximalanforderungen für die Inhalte des Zentralabiturs dar.
- Es ist selbstverständlich jeder Lehrperson frei gestellt, in eigenem Ermessen über die Vorgaben hinaus gehende Fachinhalte zu unterrichten.

- Möglichst implementationsunabhängig.
- Vor- und Nachbedingungen werden nicht ausgewiesen.

Hinweis:

- Für die Konstruktion der Vorgaben wurde darauf geachtet, jeweils mindestens eine Datenstruktur rekursiv (Klasse `BinaryTree`) als auch nicht-rekursiv (Klasse `List`) zu definieren.

Klassenbeschreibung

Objekte der Klasse `List` verwalten beliebig viele, linear angeordnete Objekte.

Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt.

Das erste oder das letzte Objekt einer Liste können durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden.

Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt werden.

(Beschreibung der Klasse ist implementationsunabhängig.)

Dokumentation

Auftrag

`void insert(Object pObject)`

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird ein neues Objekt vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (`hasAccess() == false`), wird `pObject` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt (`hasAccess() == false`) und die Liste nicht leer ist oder `pObject` gleich `null` ist, bleibt die Liste unverändert.

- Syntax wird mit angegeben.
- Verzicht auf Exceptions.
- Sonderfälle werden angegeben.

Änderungen:

- Einfache Verkettung.
- Dokumentation nicht mehr an der Implementation orientiert.
- Weniger Methoden.
- Neue Methodenbezeichner.
- Funktionalität der Methoden.

Methoden der Klasse List

Anfragen

```
boolean isEmpty();
boolean hasAccess();

Object getObject();
```

Aufträge (Navigation)

```
void next();
void toFirst();
void toLast();
```

Aufträge (Ändern der Liste)

```
void setObject(Object pObject);

void append(Object pObject);

void insert(Object pObject);

void concat(List pList);

void remove();
```

- Prüft, ob ein Zugriff an der aktuellen Listenposition möglich ist.
- Ermöglicht einen vollständigen Listendurchlauf ohne eine Sonderbehandlung des letzten Listenelements.

```
bspListe.toFirst();
while (bspListe.hasAccess())
{
    /* Operation(en) auf dem aktuellen Element:
       ...
     */

    bspListe.next();
}
```

- Die Namensänderung betont die binäre Eigenschaft des Baumes.
- Der Baum ist rekursiv definiert.

- Mithilfe der Klasse `BinaryTree` können beliebig viele Inhaltsobjekte in einem Binärbaum verwaltet werden. Ein Objekt der Klasse stellt entweder einen leeren Baum dar oder verwaltet ein Inhaltsobjekt sowie einen linken und einen rechten Teilbaum, die ebenfalls Objekte der Klasse `BinaryTree` sind.

- Die Methoden sind weitgehend geblieben:
  - `setObject` statt `setRootItem`.
  - `getObject` statt `getRootItem`.

Kommentar:

- Bei der Beschreibung der Klasse `BinaryTree` wurde bewusst auf den Begriff „Wurzel“ verzichtet, da dieser *formal korrekt* nur unter Verwendung von Konzepten aus der Graphentheorie definiert werden kann. Eine informelle Verwendung dieses Begriffs ist im Unterricht zulässig, wird aber für das Zentralabitur nicht vorausgesetzt.

Die Klasse BinarySearchTree

- Betonung auf Binärbaum und Suchfunktion durch den neuen Namen.
- Ordnungseigenschaft wird (wie bisher) durch die Klasse `Item` und deren abstrakte Methoden erzwungen.

- Zusätzliche Methoden `getItem`, `getLeftTree` und `getRightTree` ermöglichen die Traversierung auch des Suchbaums.

- Die Methode `getSortedList` entfällt.
- Neue Methodenbezeichnungen:
  - `insert` statt `insertItem`.
  - `search` statt `searchItem`.
  - `remove` statt `removeItem`.

Änderungen

- Dokumentation unabhängig von der Implementation (Adjazenzmatrix, Adjazenzlisten).
- Verzicht auf eigenständige Klasse `Edge`.
- Klassen `Graph` und Klasse `GraphNode`.
- Neue Methoden der Klasse `Graph`.
- Parameter.

Anfragen

```
boolean isEmpty();
boolean hasNode(String pName);
boolean hasEdge(GraphNode pNode1,GraphNode pNode2);
boolean allNodesMarked();

GraphNode getNode(String pNode);
double    getEdgeWeight(GraphNode pNode1, GraphNode pNode2);

List getNodes();
List getNeighbours(GraphNode pNode);
```

Aufträge

- Aufbau

```
void addNode(GraphNode pNode);
void addEgde(GraphNode pNode1, GraphNode pNode2, double pWeight);
```

- Abbau

```
void removeNode(GraphNode pNode);
void removeEdge(GraphNode pNode1, GraphNode pNode2);
```

- Markierung

```
void resetMarks();
```

Die Klasse GraphNode

- Ein Knoten hat einen Namen und kann markiert werden.

- Konstruktor

```
GraphNode(String pName);
```

- Anfragen

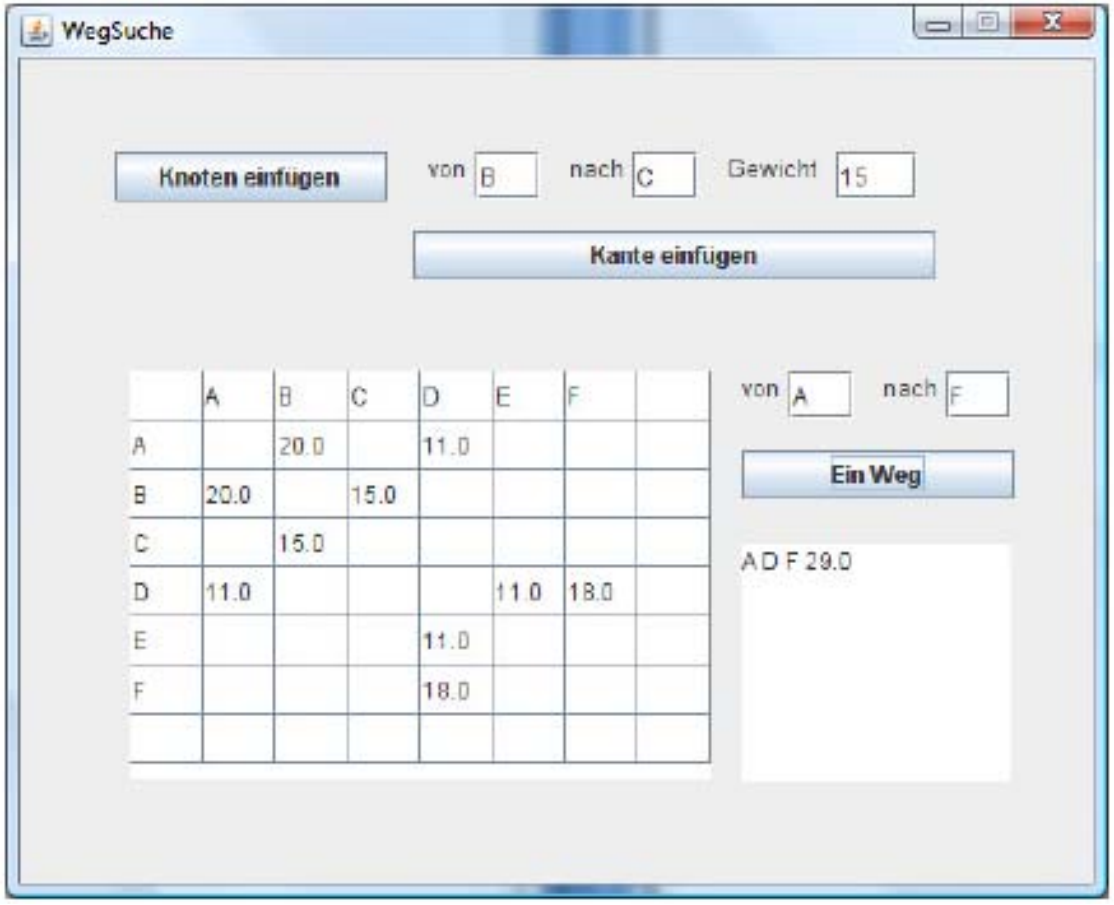
```
boolean isMarked();
String  getName();
```

- Aufträge

```
void mark();
void unmark();
```

Aufgabenbeispiel (Graph)

Mit Hilfe eines Backtrackingalgorithmus (Tiefensuche) soll ein Weg zwischen zwei Knoten ermittelt und die Weglänge angezeigt werden.



	A	B	C	D	E	F	
A		20.0		11.0			
B	20.0		15.0				
C		15.0					
D	11.0				11.0	18.0	
E				11.0			
F				18.0			

- Inhaltliche und sprachliche Präzisierungen.
- Beispiele.

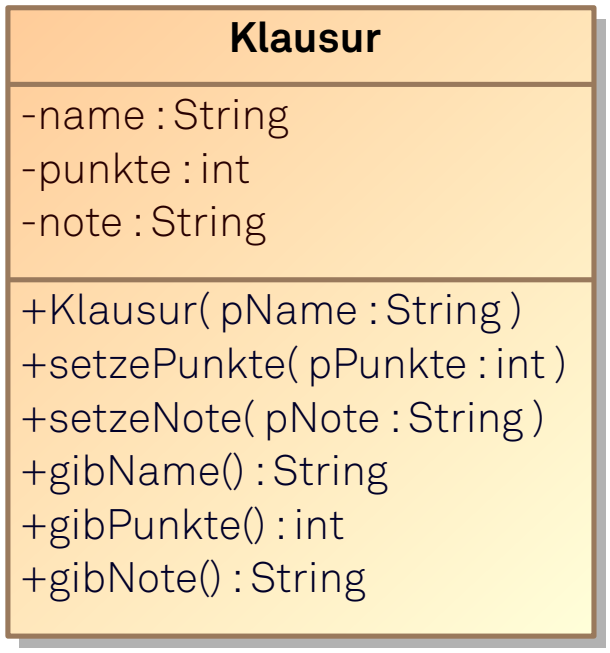
- Beschränkung auf deterministische, endliche Automaten (erkennend).
  - Definition durch 5-Tupel (A, Z, d, q_0, E).
- Beschränkung auf reguläre Grammatik.
  - Definition durch 4-Tupel (N, T, S, P).
- Im Leistungskurs: Parser.
  - Definition und Zusammenhang zu Automaten und Grammatiken.

- Relationenalgebra als Grundlage der Datenbanktheorie.
- Datenbankoperatoren: Selektion, Projektion, kartesisches Produkt, neu: **Vereinigung, Differenz, Umbenennung**.

- Datenbankbeispiele definieren und normieren Inhalte, Bezeichnungen und Darstellungsformen.
- Beispiele erklären die Datenbankoperatoren.

- Entity-Relationship-Modelle und Datenbankschemata werden wechselweise benutzt.

(Abstrakte) Klassen



Klasse

- Attribute [optional]
- Methoden [optional]

UML-Notation: "<Name> : <Typ>"

Sichtbarkeit

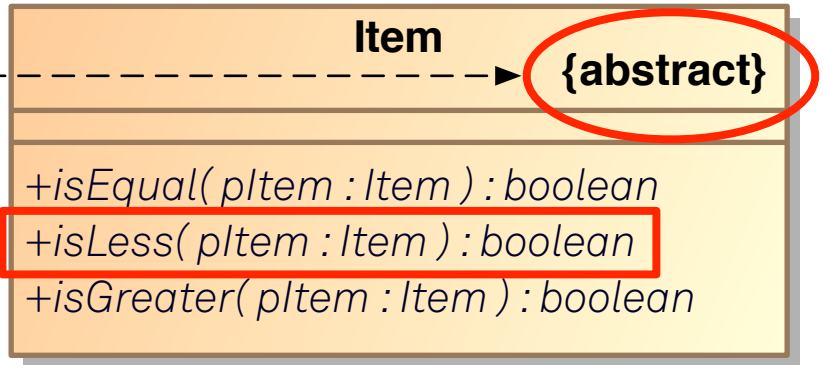
- + (public)
- - (private)

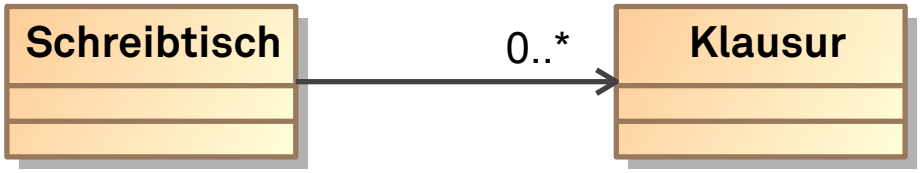
Abstrakte Klasse

Abstrakte Methode

- Name kursiv geschrieben **oder**
- Name mit Wellenlinie unterstrichen

Achtung: "Normale" Unterstreichung bedeutet im UML-Standard "static".





Assoziation:

- Semantische Beziehung zwischen Objekten der beteiligten Klassen.
  - Ein Objekt der Klasse **`Schreibtisch`** kann Objekte der Klasse **`Klausur`** verwalten.
- Darstellung einer gerichteten Beziehung mit Pfeil, optional mit Namen.
  - Kein Pfeil: Bidirektionale Beziehung.
- Multiplizitäten hier: `1` / `0..1` / `0..*` / `1..*`.
- Spezialfall: „Teil-Ganzes-Beziehung“ (Komposition) [Optional]
  - Existentielle Abhängigkeit der Teile vom Ganzen.
  - Darstellung mit ausgefüllter Raute auf der Seite des „Ganzen“.

Kommentare:

- Eine fehlende Multiplizität ist als ‘*’=‘`0..*`’ zu lesen.

> The multiplicity of an association end is suppressed if it is ‘*’ (default in UML).
> [Object Management Group, 2006, S. 23]

- Multiplizitäten können auch in der allgemeinen Form

  „`untereSchranke..obereSchranke`“ (also z.B. „`3..7`“)

  angegeben werden. Dies ist jedoch **nicht** Bestandteil der Vorgaben.
- Der UML-Standard [Object Management Group, 2006, S. 115] erlaubt die Angabe einer Multiplizität auch an der Seite einer gerichteten Assoziation, von der diese ausgeht. Da die technische Realisierung einer solchen Einschränkung den für das Zentralabitur vorgesehenen Stoff deutlich überschreitet, wird davon abgeraten, diese Modellierung, die **nicht** Bestandteil der Vorgaben ist, zu verwenden.

Kommentare (Fortsetzung):

- Die Komposition ist eine *Spezialisierung* der Assoziation (genauer: eine Spezialisierung der Aggregation, die eine Spezialisierung der Assoziation ist). Aus diesem Grund ist die Komposition nicht Bestandteil der Vorgaben, kann aber optional im Unterricht verwendet werden.

> An association may represent a composite aggregation (i.e., a whole/part relationship). Only binary associations can be aggregations. Composite aggregation is a strong form of aggregation that requires a part instance be included in at most one composite at a time.
>
> [Object Management Group, 2006, S. 112]

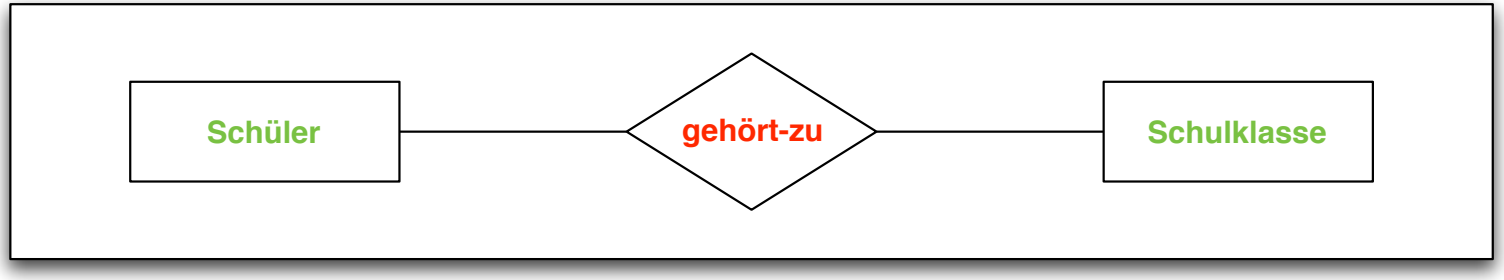
> A composite aggregation is shown using the same notation as a binary association, but with a solid, filled diamond at the aggregate end.
>
> [Object Management Group, 2006, S. 113]

Interpretation

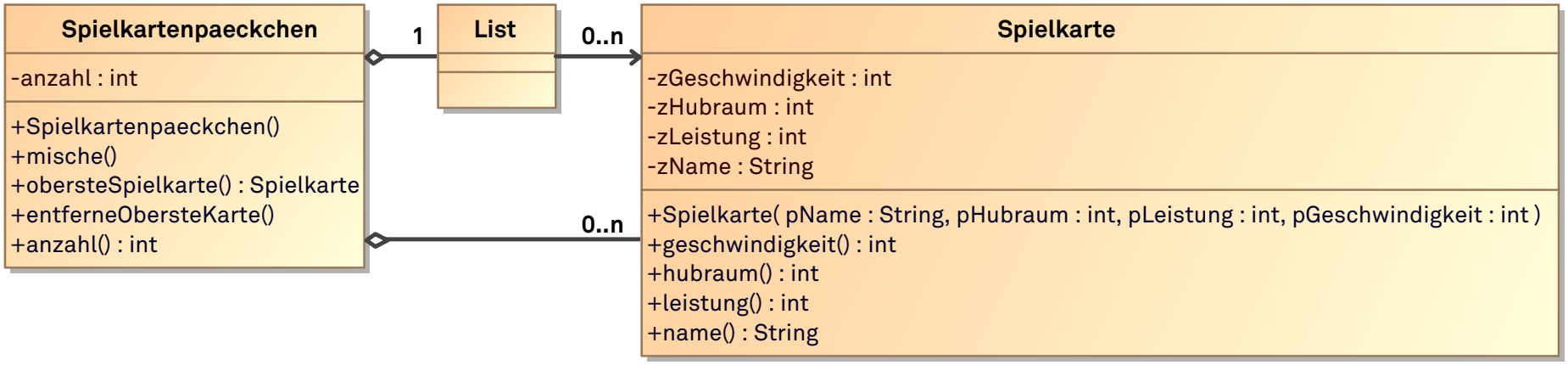
> An association declares that there can be links between instances of the associated types. A link is a tuple with one value for each end of the association, where each value is an instance of the type of the end.
>
> [Object Management Group, 2006, S. 110]

Vergleich mit dem ER-Modell (Datenbanken):

> Zwischen Entitäten kann es Beziehungen geben, wobei man gleichartige Beziehungen zwischen Entitäten zweier Entitätsmengen zu Beziehungsmengen zusammen fasst.
>
> [Vorgaben 2012, S. 24]



Eine gute/korrekte Modellierung?



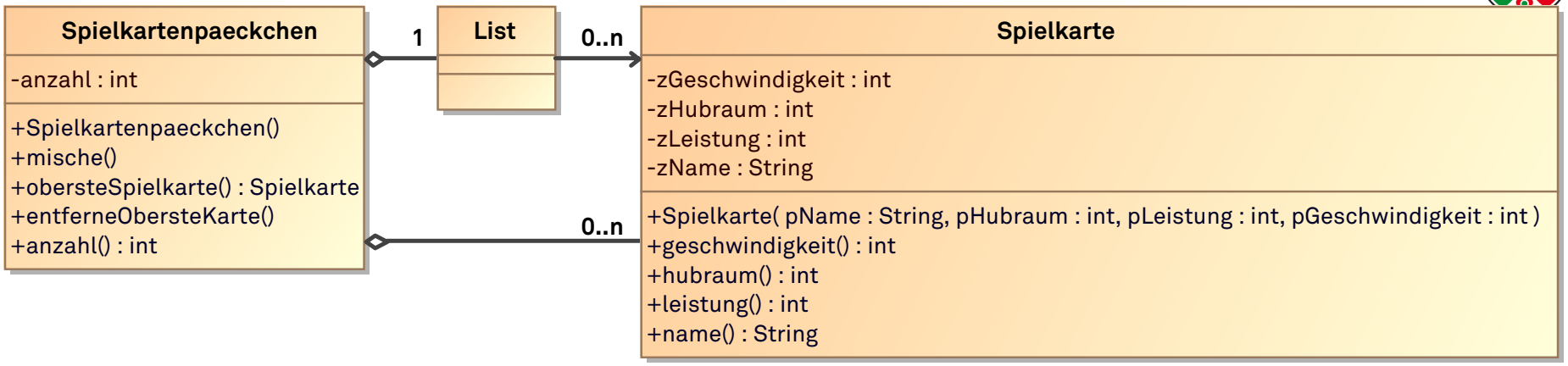
Weil das Spielkartenpäckchen keine Methode zum Aufnehmen von Spielkarten besitzt, muss es die Spielkarten selbst erzeugen. Daher besteht eine hat-Beziehung zwischen dem Spielkartenpäckchen und den Spielkarten. Die Kardinalität beträgt $0..n$, da am Anfang n Spielkarten erzeugt werden. Nach Austeilen der Karten besitzt das Spielkartenpäckchen weniger Karten. [...]
Das Spielkartenpäckchen hat eine Liste, die ihm bei der Verwaltung der Spielkarten hilft. Diese Spielkartenliste muss dafür alle Karten des Päckchens kennen.

Was bedeutet hier „haben“ und „kennen“?

Weil das Spielkartenpäckchen keine Methode zum Aufnehmen von Spielkarten besitzt, muss es die Spielkarten selbst erzeugen. Daher besteht eine hat-Beziehung zwischen dem Spielkartenpäckchen und den Spielkarten. Die Kardinalität beträgt $0..n$, da am Anfang n Spielkarten erzeugt werden. Nach Austeilen der Karten besitzt das Spielkartenpäckchen weniger Karten. [...]
Das Spielkartenpäckchen hat eine Liste, die ihm bei der Verwaltung der Spielkarten hilft. Diese Spielkartenliste muss dafür alle Karten des Päckchens kennen.

- „Haben“ bedeutet nicht:
  - Eine Methode kann den Konstruktor der anderen Klasse aufrufen.
  - Eine Methode hat eine lokale Variable vom Typ der anderen Klasse.
- „Kennen“ bedeutet nicht:
  - Eine Liste speichert Objekte einer bestimmten Klasse.
- Assoziation bedeutet:
  - Jedes Objekt einer Klasse `A` hat **konstruktionsbedingt** die Möglichkeit, Referenzen auf x Objekte der Klasse `B` **dauerhaft** zu verwalten.

Entwurfs- und Implementationsebene

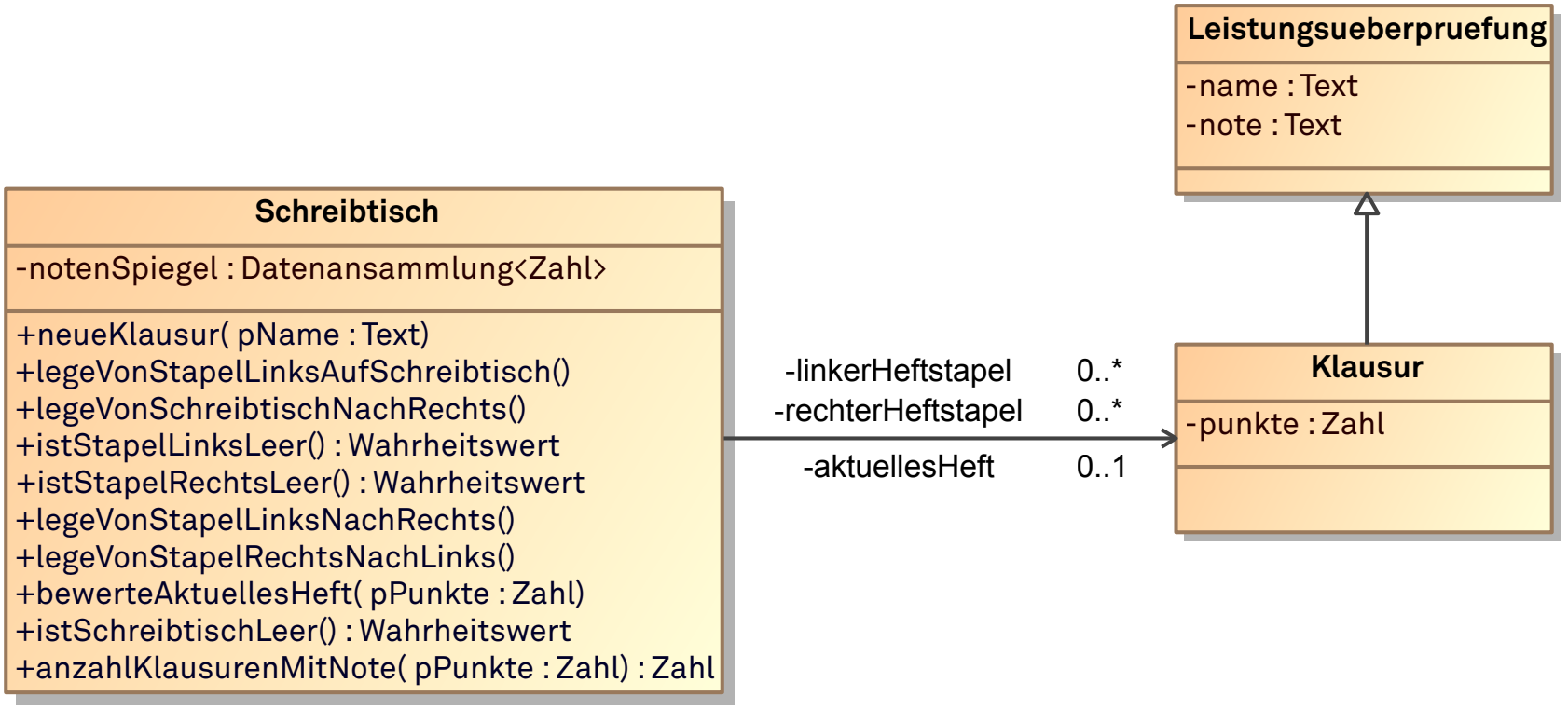


Ausweg aus diesem Dilemma:

- Klare Trennung von Entwurfs- und Implementationsebene.
- Entwurfsebene: Konzeptionelle Darstellung.
- Implementationsebene: Konkreter Bezug zur Implementation.

> Auf der Entwurfsebene stellen Klassen also nicht mehr nur Konzepte dar, aber auch noch nicht Konstrukte einer speziellen Implementierungssprache. Auf dieser Ebene lässt sich abstrakt über einen technischen Sachverhalt reden, während der fachliche Bezug vorhanden bleibt. [Störrle, 2005, S. 57]

Entwurfsdiagramm – I



- Basisdatentypen `Zahl`, `Text`, `Wahrheitswert`.
  - UML 2.0: `Integer`, `String`, `Boolean`.
- Datentyp `Datenansammlung`$\langle \cdot \rangle$.
  - Didaktische Reduktion der offiziellen UML-Notation.

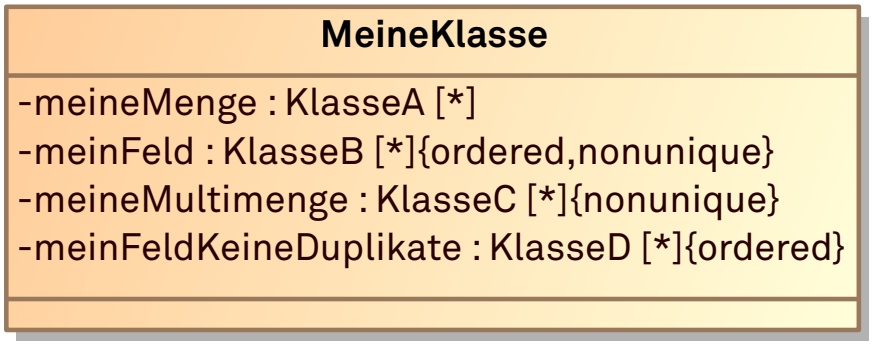
Kommentare:

- Im Vergleich zur UML 2.0 [Object Management Group, 2006, S. 169] fehlt bei den Basisdatentypen der Typ `UnlimitedNatural`, der die natürlichen Zahlen sowie den Wert '$*$' ($= \infty$) enthält.
- Bei dem im Zentralabitur verwendeten Basisdatentypen `Zahl` ist explizit nicht ausgeschlossen, dass zur Implementation Datentypen verwendet werden können, die Zahlen mit Nachkommastellen repräsentieren.

Entwurfsdiagramm – II

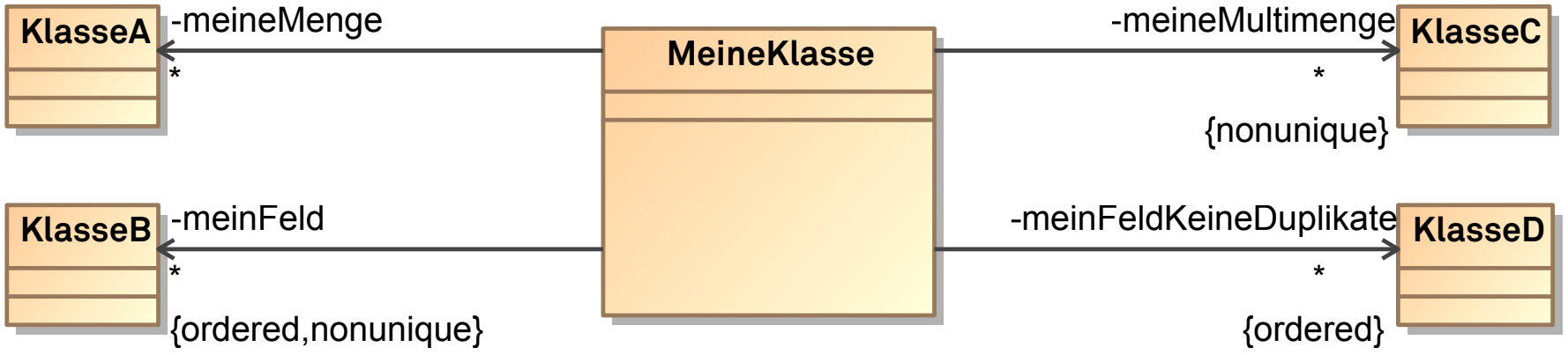
Offizielle UML-Notation:

- Vorhandensein einer (linearen) Ordnung und/oder Eindeutigkeit der gespeicherten Elemente.



[Object Management Group, 2006, S. 121]

Äquivalente Darstellung:



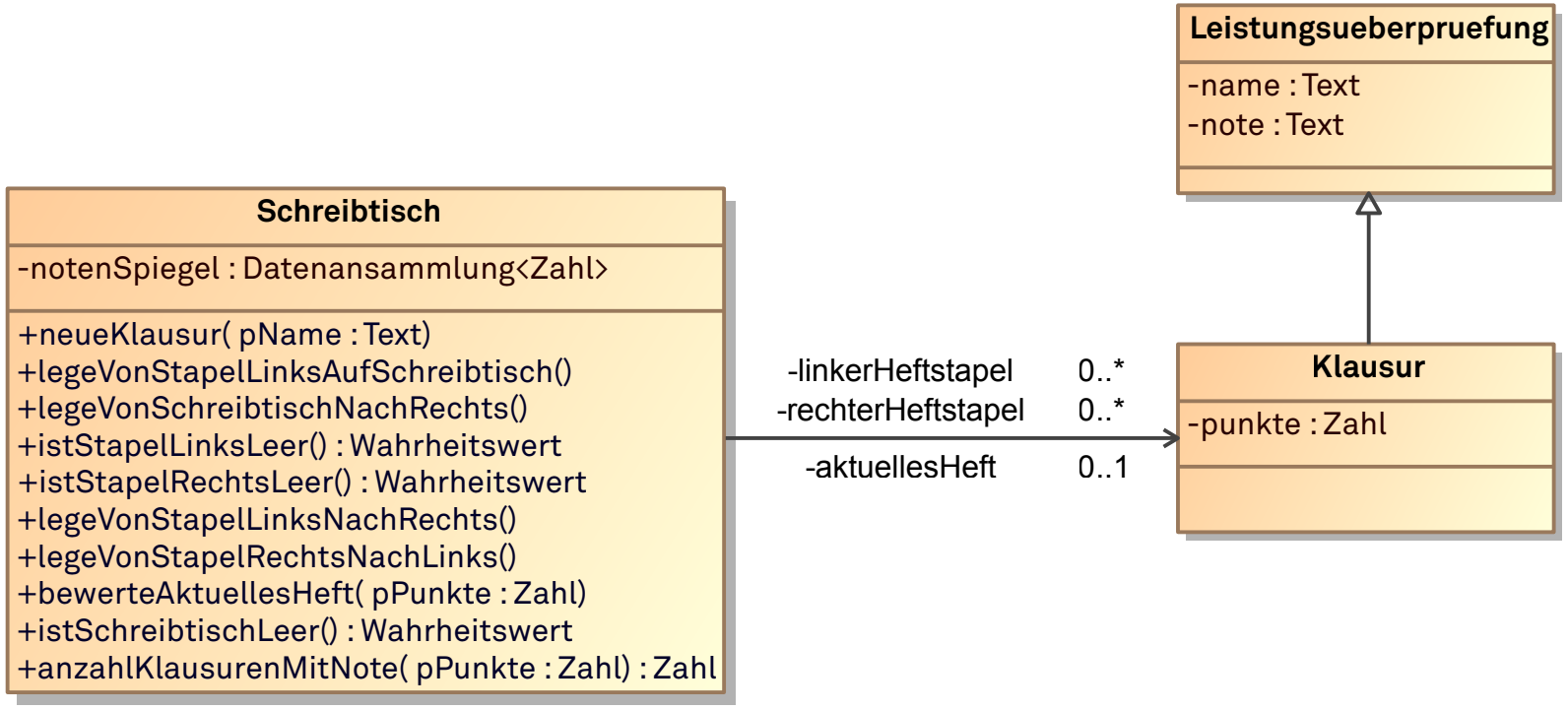
Statt dessen im Zentralabitur: `Datenansammlung`$\langle \cdot \rangle$.

Kommentare:

- Der UML-Standard [Object Management Group, 2006, S. 115] erlaubt die Angabe einer Multiplizität auch an der Seite einer gerichteten Assoziation, von der diese ausgeht. Da die technische Realisierung einer solchen Einschränkung den für das Zentralabitur vorgesehenen Stoff deutlich überschreitet, wird davon abgeraten, diese Modellierung, die **nicht** Bestandteil der Vorgaben ist, zu verwenden.
- Wenn es erwünscht ist, können Rollennamen bei Assoziationen angegeben werden, die an Stelle von Attributen verwendet werden; dies ist aber generell nicht vorgesehen.

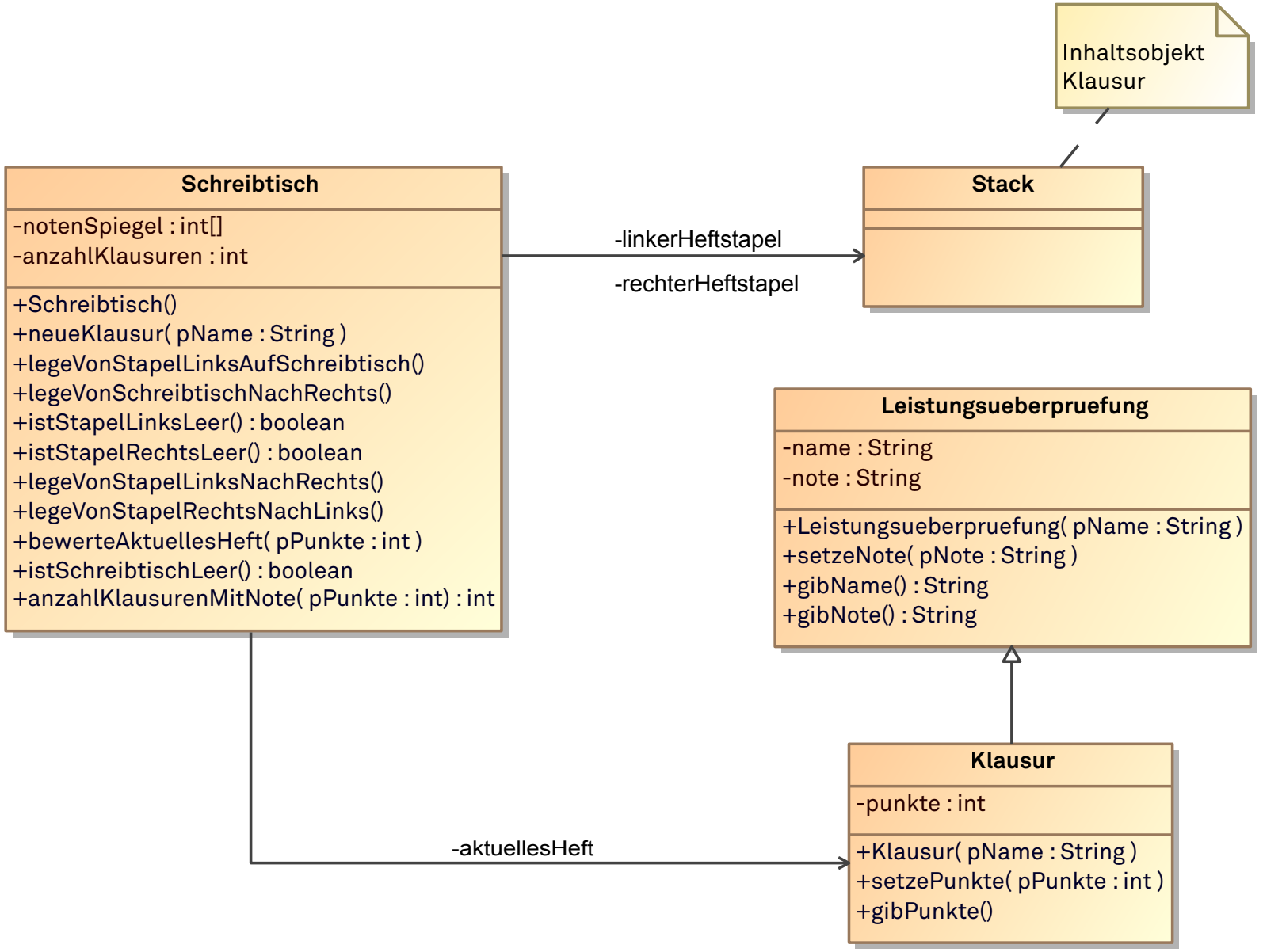
> When directed associations are specified in lieu of attributes, the multiplicity on the undirected end is assumed to be '*' (default in UML) and the role name should not be used.
> [Object Management Group, 2006, S. 23]

Entwurfsdiagramm – III

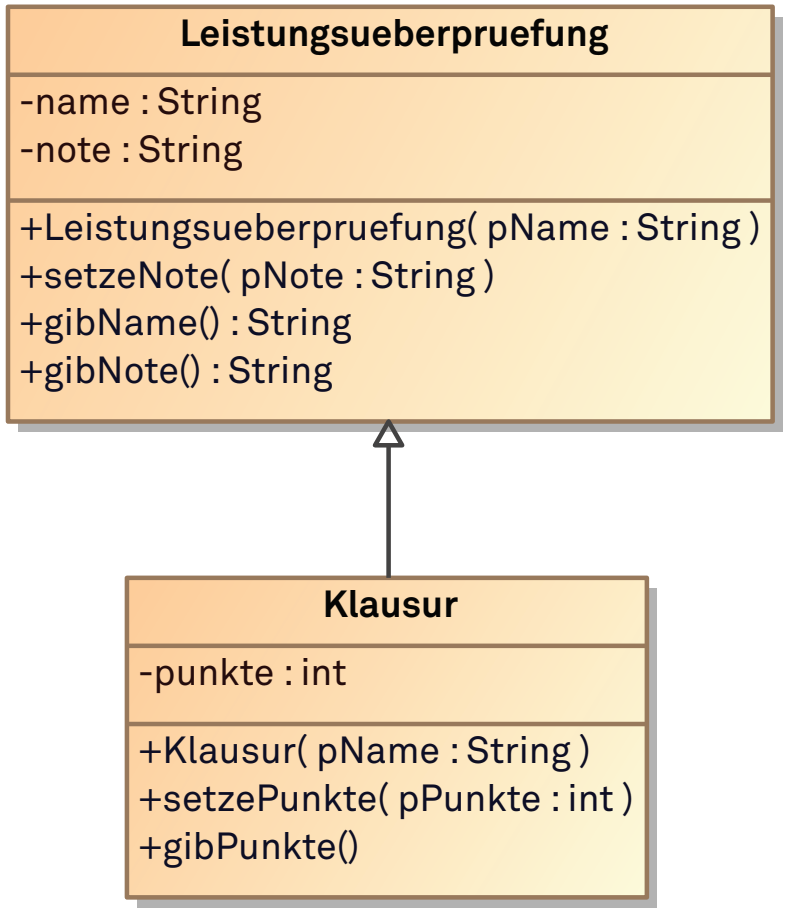


N.B. Die Modellierung umfasst hier weniger Aspekte als im nachfolgenden Implementationsdiagramm. Es fehlen Konstruktoren und Selektoren.

Implementationsdiagramm



Diskussion

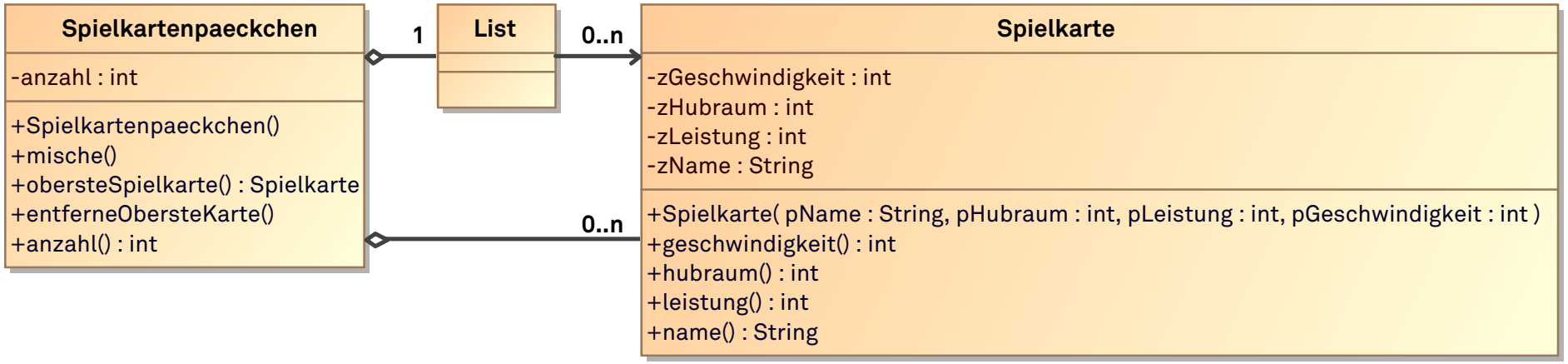


Ist diese Modellierung verbesserungsfähig?
Falls ja, wie sollte sie verbessert werden?

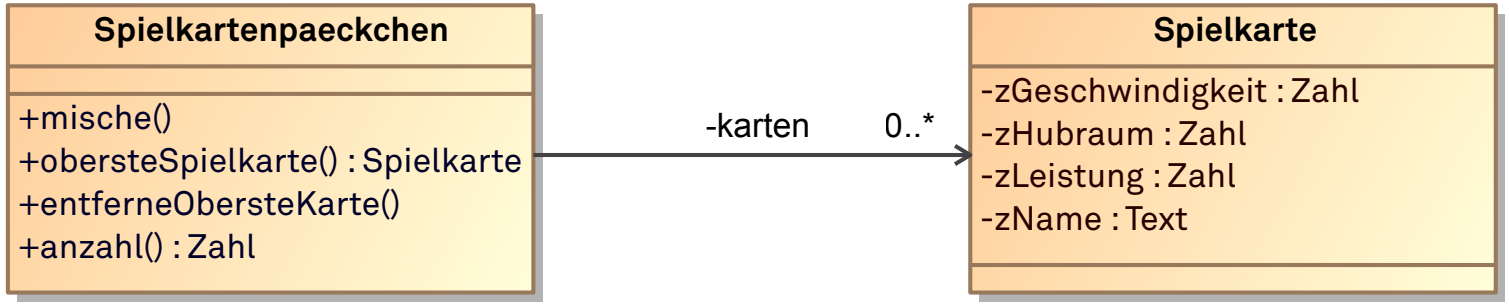
Kommentare:

- Die Speicherung der Bewertung ist redundant.
  - Ein Objekt der Klasse `Leistungsueberpruefung` speichert die Note als Text.
  - Ein Objekt der Klasse `Klausur` speichert die Note **zusätzlich** als Zahl.
- Wird die Redundanz entfernt (aber die Vererbungsbeziehung beibehalten), benötigt man einen Umrechnungsmechanismus, der von Objekten der Klasse `Klausur` genutzt werden kann.
  - Zu diskutieren wäre hier die Verwendung einer Klassenmethode oder die Verwendung einer „Umrechnungsklasse“.
  - Zu einer „Umrechnungsklasse“ sollte nur ein Objekt existieren; dies führt zum Entwurfsmuster „Singleton“.
- Alternativ wäre zu überlegen, ob nicht eine abstrakte Oberklasse modelliert werden sollte, die eine direkte Generalisierung sowohl von `Leistungsueberpruefung` als auch von `Klausur` ist.
  - Ist bei einer Klausur wirklich vorgesehen, die Note als Text nutzen zu können?

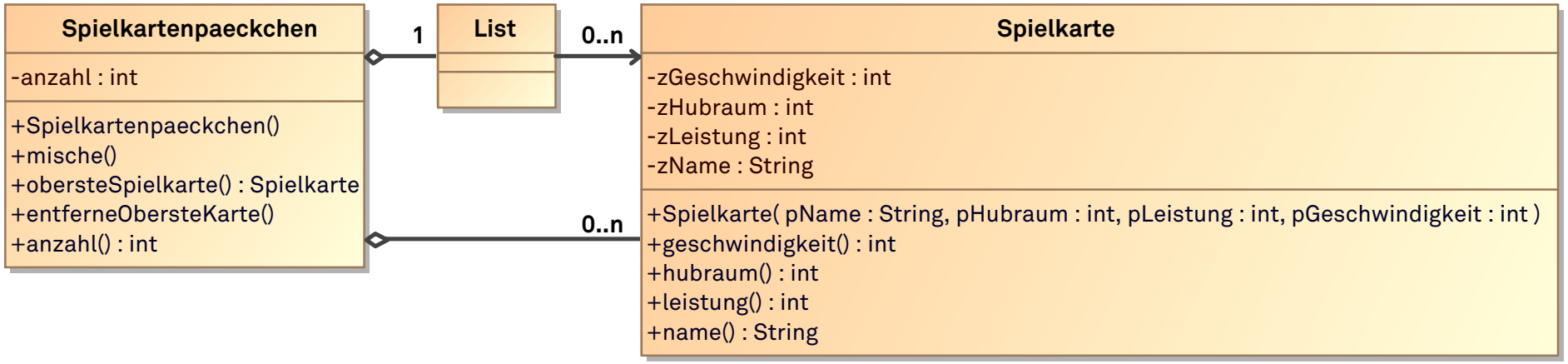
Rückschau – I



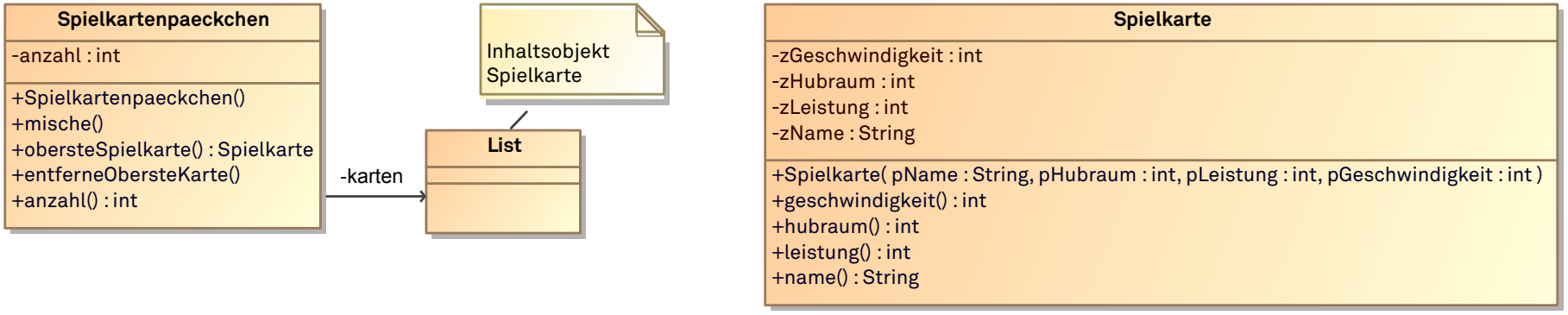
Entwurfsdiagramm:



Rückschau – II



Implementationsdiagramm:



Entwurfsdiagramm:

- Darstellung des konzeptionellen Entwurfs.
- Verwendung programmiersprachenunabhängiger (Basis-)Datentypen.
  - Insbesondere keine Festlegung auf konkrete (z.B. lineare) Strukturen.
  - Gerichtete Assoziation an Stelle von `Datenansammlung`$\langle \cdot \rangle$ hebt (Existenz der) Beziehung zwischen den Klassen hervor.

Implementationsdiagramm:

- Darstellung der konkreten Umsetzungsvorgaben.
- Verwendung programmiersprachenspezifischer Basisdatentypen.
- Insbesondere Entscheidung für konkrete (z.B. lineare) Strukturen.

Neue Aufgabentypen

- Umwandlung eines Entwurfsdiagramms in Implementationsdiagramm.
  - Entwurfsdiagramm programmiersprachenunabhängig.
  - Festlegung der Modellierung (→ vereinfachte Korrektur).
  - Möglichkeit, AFB III abzudecken (Begründung der Entscheidung für konkrete Struktur).

- Rekonstruktion eines Entwurfsdiagramms aus einem Implementationsdiagramm.
  - Abstraktion von einer programmiersprachenbezogenen Realisierung.

- Vergleich alternativer Entwurfs- bzw. Implementationsdiagramme.
  - Möglichkeit, AFB III abzudecken (Begründung der Wahl einer bestimmten Modellierungs auf der Entwurfsebene).

Literaturverzeichnis

[Object Management Group, 2006] Object Management Group: *Unified Modeling Language: Infrastructure. Version 2.0, formal/05-07-05*, März 2006. Online im Internet: `http://www.omg.org/spec/UML/2.0/Infrastructure/PDF/` [Stand: 2011-03-03].

[Störrle, 2005] Störrle, Harald: *UML 2 für Studenten*. Pearson Studium, München, 2005.