

SS2010  
BAI2-DBP Gruppe 1  
Lösung zu Übungsblatt 2

R. C. Ladiges, D. Fast

21. April 2010

# Inhaltsverzeichnis

<b>3</b>	<b>Aufgabe 3 (UML-Klassendiagramme)</b>	<b>4</b>
3.0	Aufgabenstellung . . . . .	4
3.1	A01 . . . . .	4
3.1.1	Anforderung . . . . .	4
3.1.2	UML-Diagramm . . . . .	4
3.1.3	Beschreibung . . . . .	4
3.2	A02 . . . . .	5
3.2.1	Anforderung . . . . .	5
3.2.2	UML-Diagramm . . . . .	5
3.2.3	Beschreibung . . . . .	5
3.3	A03 . . . . .	6
3.3.1	Anforderung . . . . .	6
3.3.2	UML-Diagramm . . . . .	6
3.3.3	Beschreibung . . . . .	6
3.4	A04 . . . . .	7
3.4.1	Anforderung . . . . .	7
3.4.2	UML-Diagramm . . . . .	7
3.4.3	Beschreibung . . . . .	7
3.5	A05 . . . . .	8
3.5.1	Anforderung . . . . .	8
3.5.2	UML-Diagramm . . . . .	8
3.5.3	Beschreibung . . . . .	8
3.6	A06 . . . . .	9
3.6.1	Anforderung . . . . .	9
3.6.2	UML-Diagramm . . . . .	9
3.6.3	Beschreibung . . . . .	9
3.7	A07 . . . . .	10
3.7.1	Anforderung . . . . .	10
3.7.2	UML-Diagramm . . . . .	10
3.7.3	Beschreibung . . . . .	10
3.8	A08 . . . . .	11
3.8.1	Anforderung . . . . .	11
3.8.2	UML-Diagramm . . . . .	11
3.8.3	Beschreibung . . . . .	11
3.9	A09 . . . . .	12
3.9.1	Anforderung . . . . .	12
3.9.2	UML-Diagramm . . . . .	12
3.9.3	Beschreibung . . . . .	12
3.10	A10 . . . . .	13
3.10.1	Anforderung . . . . .	13
3.10.2	UML-Diagramm . . . . .	13
3.10.3	Beschreibung . . . . .	13
3.11	A11 . . . . .	14
3.11.1	Anforderung . . . . .	14
3.11.2	UML-Diagramm . . . . .	14
3.11.3	Beschreibung . . . . .	14
3.12	A12 . . . . .	15

3.12.1	Anforderung	15
3.12.2	UML-Diagramm	15
3.12.3	Beschreibung	15
3.13	A13	16
3.13.1	Anforderung	16
3.13.2	UML-Diagramm	16
3.13.3	Beschreibung	16
3.14	A14	17
3.14.1	Anforderung	17
3.14.2	UML-Diagramm	17
3.14.3	Beschreibung	17
3.15	A15	18
3.15.1	Anforderung	18
3.15.2	UML-Diagramm	18
3.15.3	Beschreibung	18
<b>4</b>	<b>Aufgabe 4 (ER-Modell)</b>	<b>19</b>
4.0	Aufgabenstellung	19
4.1	ER-Modell	19
4.2	Beschreibung	19
<b>5</b>	<b>Aufgabe 5 (SQL)</b>	<b>20</b>
5.0	Aufgabenstellung	20

### 3 Aufgabe 3 (UML-Klassendiagramme)

#### 3.0 Aufgabenstellung

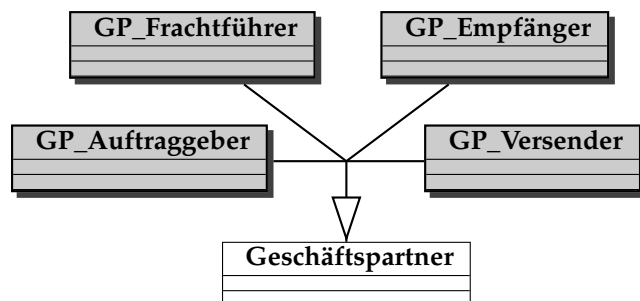
Modellieren Sie die Anforderungen A01 bis A15 des HAW Logistics-Lastenheftes in Form von UML-Klassendiagrammen. Fertigen Sie zusätzlich eine Dokumentation Ihres Vorgehens an, z.B. A01 führt zu den Klassen xyz mit den Attributen abc bzw. zu den Assoziationen klm.

#### 3.1 A01

##### 3.1.1 Anforderung

Im System sollen Geschäftspartner definierbar sein. Geschäftspartner können Auftraggeber, Frachtführer, Empfänger oder Versender darstellen.

##### 3.1.2 UML-Diagramm



##### 3.1.3 Beschreibung

Wir erstellen eine generelle<sup>1</sup> Klasse **Geschäftspartner** und vier spezielle<sup>2</sup> Klassen **GP\_Auftraggeber**, **GP\_Frachtführer**, **GP\_Empfänger** und **GP\_Versender**, die von **Geschäftspartner** erben. Das heißt jeder **Geschäftspartner** ist von einem der vier Klassen.

---

<sup>1</sup>oder auch abstrakte Klasse

<sup>2</sup>oder auch konkrete Klassen

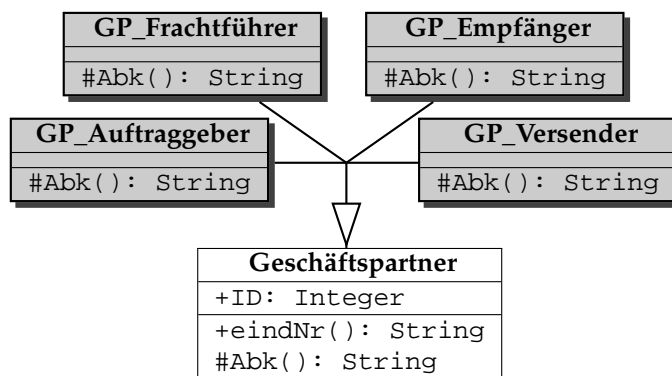
## 3.2 A02

### 3.2.1 Anforderung

Geschäftspartner sind durch eine Nummer der Form „GP-<Art des Geschäftspartners>-n“ (n ist eine natürliche Zahl) eindeutig identifizierbar. Die Art des Geschäftspartners ist wie folgt definiert:

- a. Auftraggeber: „AG“
- b. Frachtführer: „FRA“
- c. Lokation: „LOK“
- d. Empfänger: „EMP“
- e. Versender: „VER“

### 3.2.2 UML-Diagramm



### 3.2.3 Beschreibung

Dass laut dieser Definition die Art des **Geschäftspartners** eine **Lokation** sein kann wirkt fehlerhaft. Zum einen ist eine **Lokation** keine Person und kann schon deshalb kein **Geschäftspartner** sein. Zum anderen ist in [A01] **Lokation** nicht als möglicher **Geschäftspartner** aufgeführt. Deshalb ignorieren wir Definition c aus [A02]. Sollte es dennoch gewünscht sein **Geschäftspartner** mit einer Nummer die „LOK“ enthält zu erstellen, könnte man eine weitere spezielle Klasse von **Geschäftspartner** namens **GP\_Lokation** erstellen.

Zur Umsetzung der eindeutigen Nummer fügen wir der generellen Klasse **Geschäftspartner** die Template Methode `eindNr(): String` hinzu, die an alle speziellen Klassen vererbt wird. Die Methode konkateniert „GP-“ mit dem Rückgabewert der abstrakten Methode `Abk(): String` (die für jede spezielle Klasse anders implementiert ist) mit „-“ und einer eindeutigen Zahl die im Attribut `ID: Integer` gespeichert ist.

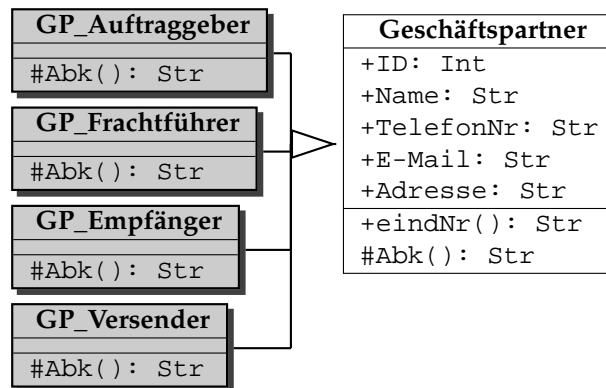
Integer ist eine Ganzzahl (natürliche Zahl). String eine Zeichenkette mit flexibler Länge. In folgenden Diagrammen werden sie mit `Int` und `Str` abgekürzt.

### 3.3 A03

#### 3.3.1 Anforderung

Jedem Geschäftspartner ist ein Name, eine Telefonnummer, eine E-Mail, sowie eine Adresse zugeordnet.

#### 3.3.2 UML-Diagramm



#### 3.3.3 Beschreibung

Jeder **Geschäftspartner** bekommt die zusätzlichen Attribute `Name: Str`, `TelefonNr: Str`, `E-Mail: Str` und `Adresse: Str` vererbt.

Die `TelefonNr` wird als String und nicht als Integer realisiert, um auch Zeichen die keine Ziffern sind in die Telefonnummer mit aufzunehmen.

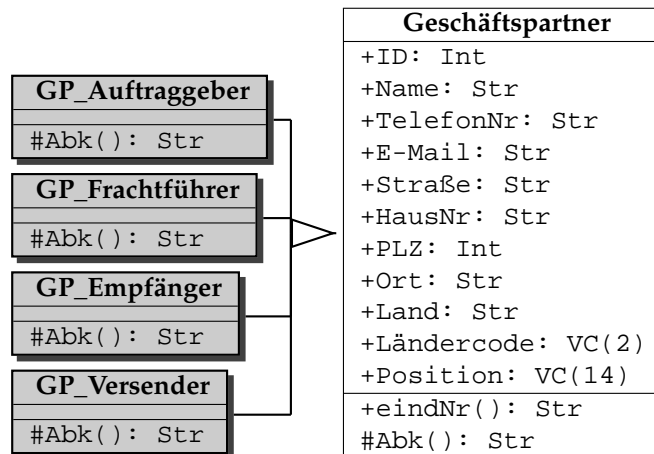
Beispiel: +49(0)40/12345-67

### 3.4 A04

#### 3.4.1 Anforderung

Jede Adresse besteht aus Strasse, Hausnummer, Postleitzahl, Ort, Land, Landercode nach ISO 3166-1 [LC], sowie geografischen Daten zu dessen Position im Format DD° MM' SS.SS" (z.B. 51° 03' 09.50").

#### 3.4.2 UML-Diagramm



#### 3.4.3 Beschreibung

Anstatt einem Attribut `Adresse: Str` haben wir nun mehrere Attribute.

`Straße: Str`, `HausNr: Str`, `PLZ: Int`, `Ort: Str`, `Land: Str`, `Ländercode: VC(2)` und `Position: VC(14)`.

`HausNr` ist als String und nicht als Integer implementiert, da zu einer Hausnummer oft noch zusätzlich Buchstaben kommen. Beispiele: 1a, 1b, 13d

`Ort` ist eigentlich eine redundante Information, da sie über die `PLZ` schon gegeben ist. Das gleiche haben wir auch bei `Land` durch den `Ländercode`. In großen Systemen wäre eine Implementation sinnvoll, die nur die `PLZ` und den `Ländercode` enthält. Welcher `Ländercode` welches `Land` ist und welche `PLZ` welcher `Ort` ist würde man dann über weitere Klassen zuweisen. Genaugenommen ließen sich auch alle diese Attribute über die `Position` zurückführen. In unserem Modell belassen wir es aber bei dieser einfachen redundanten Implementierung zwecks Übersichtlichkeit.

`VC` ist hier als Abkürzung für `VARCHAR` zu verstehen, einer Zeichenkette mit fester Länge die in Klammern angegeben ist.

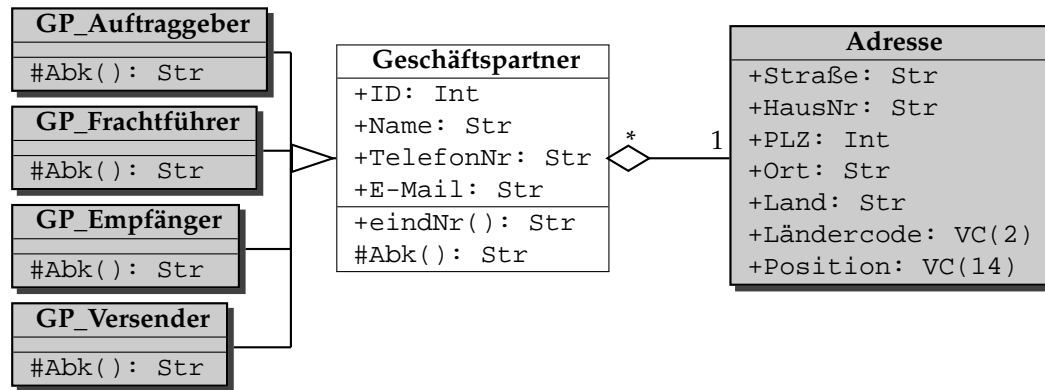
Das Attribut `Position: VC(14)` hat die vorgegebene Struktur mit Leer- und Sonderzeichen. Schöner wäre es `DD`, `MM` und `SS` atomar zu speichern und bei Bedarf zu konkatenieren.

### 3.5 A05

#### 3.5.1 Anforderung

Einer Adresse können mehrere Geschäftspartner zugeordnet sein.

#### 3.5.2 UML-Diagramm



#### 3.5.3 Beschreibung

Um die selbe Adresse für mehrere **Geschäftspartner** redundanzlos zu speichern nehmen wir die Attribute die in **Geschäftspartner** [A04] die Adresse darstellen in eine neue Klasse namens **Adresse**.

Über eine Aggregation enthalten beliebig viele **Geschäftspartner** ein und die selbe **Adresse**. Ein **Geschäftspartner** enthält genau eine **Adresse**.

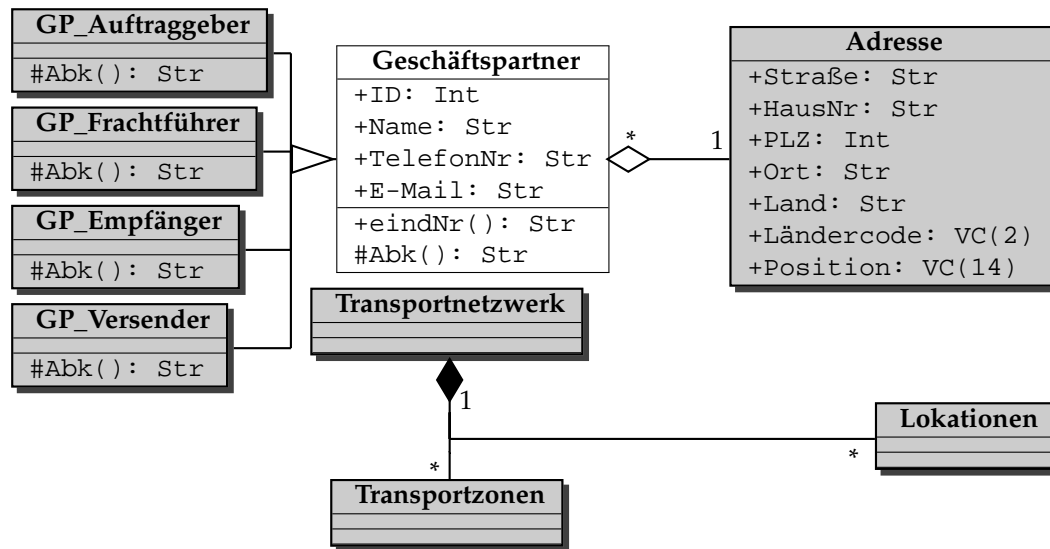


### 3.6 A06

#### 3.6.1 Anforderung

Mittels HLS soll ein Transportnetzwerk verwaltbar sein. Ein Transportnetzwerk besteht aus Lokationen [A07] und Transportzonen [A11].

#### 3.6.2 UML-Diagramm



#### 3.6.3 Beschreibung

Implementation durch drei Klassen. **Transportnetzwerk** besteht aus (Komposition) den Klassen **Lokationen** und **Transportzonen**. Ein **Transportnetzwerk** kann aus beliebig vielen **Lokationen** und **Transportzonen** bestehen. Es kann keine **Lokation** oder **Transportzone** existieren die nicht Teil genau eines **Transportnetzwerkes** ist.

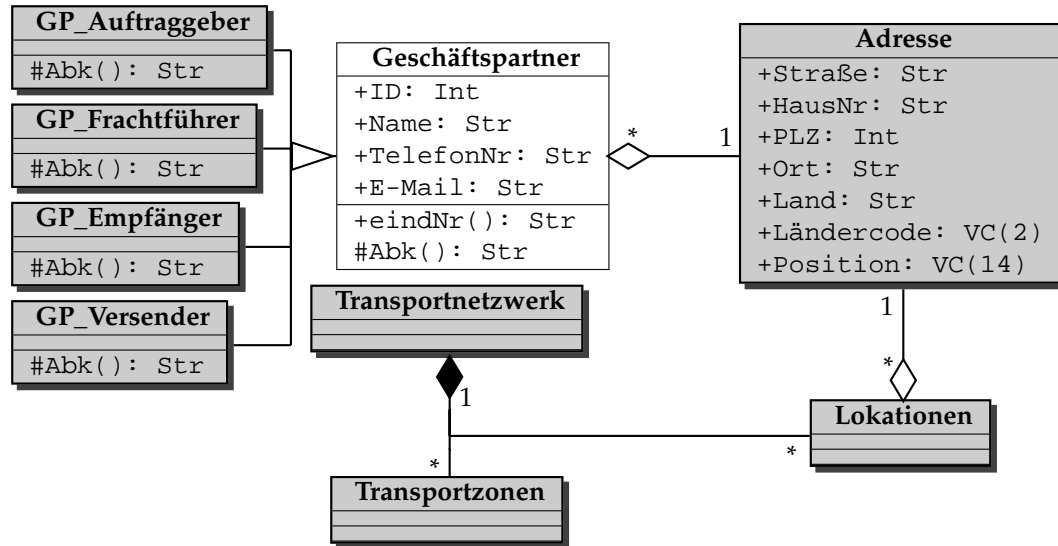
Die Darstellung hier steht für zwei verschiedene Kompositionen. Zur einfacheren Darstellung ist hier nur ein Knoten am **Transportnetzwerk** eingezeichnet. Es ist hier nicht zu verstehen, dass eine n:m Beziehung zwischen **Lokationen** und **Transportzonen** existiert.

### 3.7 A07

#### 3.7.1 Anforderung

Lokationen stellen Orte dar. Lokationen haben den Ort der Lokation als Adresse [A04] zugeordnet.

#### 3.7.2 UML-Diagramm



#### 3.7.3 Beschreibung

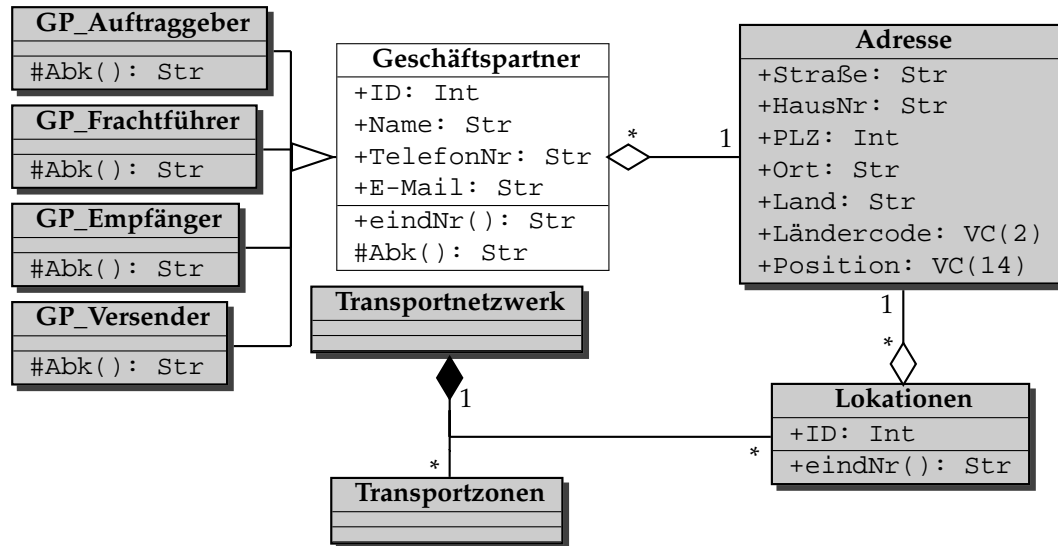
Die Klasse **Lokationen** aggregiert die Klasse **Adresse**, so dass jede **Lokation** eine **Adresse** enthält und jede **Adresse** in mehreren **Lokationen** (und **Geschäftspartnern**) enthalten sein kann.

### 3.8 A08

#### 3.8.1 Anforderung

Lokationen haben eine eindeutige Nummer. Diese besteht aus dem Landercode des Orts der Lokation, gefolgt von einem Schrägstrich („/“) und einer fortlaufenden ganzzahligen Nummer (beginnend mit dem Wert „1“). Beispiele: „DE/7“, „SE/1756“.

#### 3.8.2 UML-Diagramm



#### 3.8.3 Beschreibung

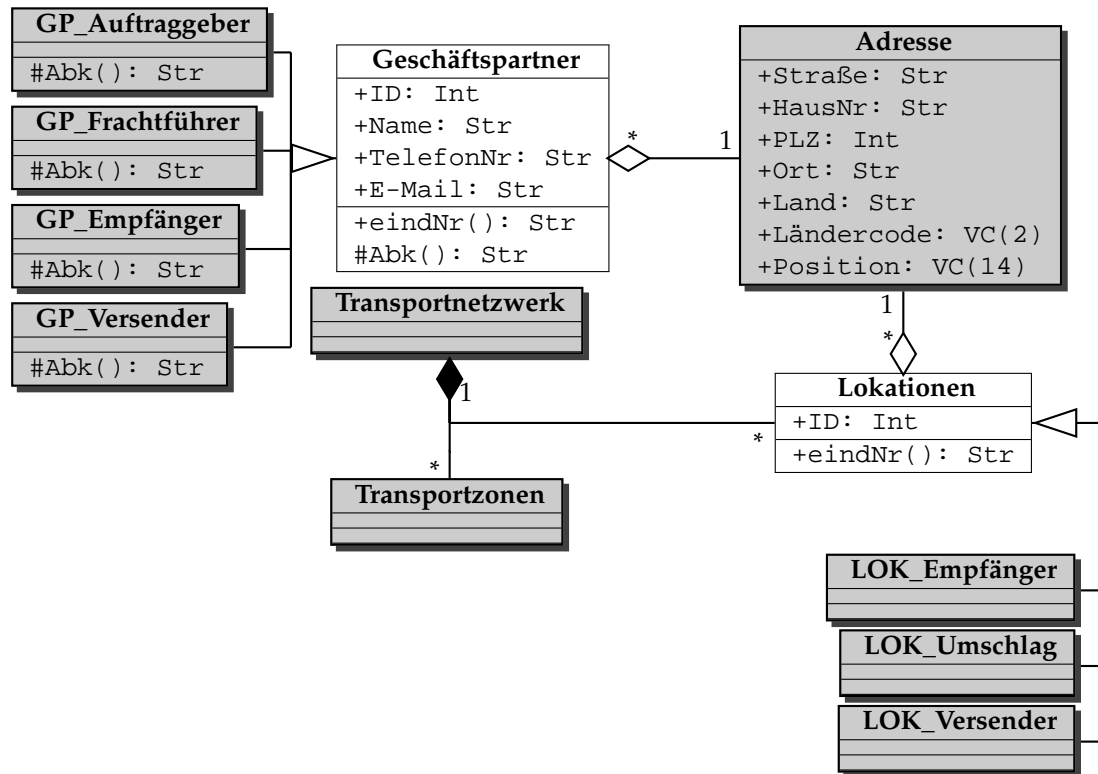
Ähnlich wie in [A02] bekommt die Klasse **Lokationen** eine Methode `eindNr(): Str`, die den Ländercode: `VC(2)` aus der aggregierten **Adresse** mit „/“ und dem eigenem Attribut `ID: Int` konkateniert.

### 3.9 A09

#### 3.9.1 Anforderung

Lokationen können von der Art „Versender“, „Empfänger“ oder „Umschlag“ sein.

#### 3.9.2 UML-Diagramm



#### 3.9.3 Beschreibung

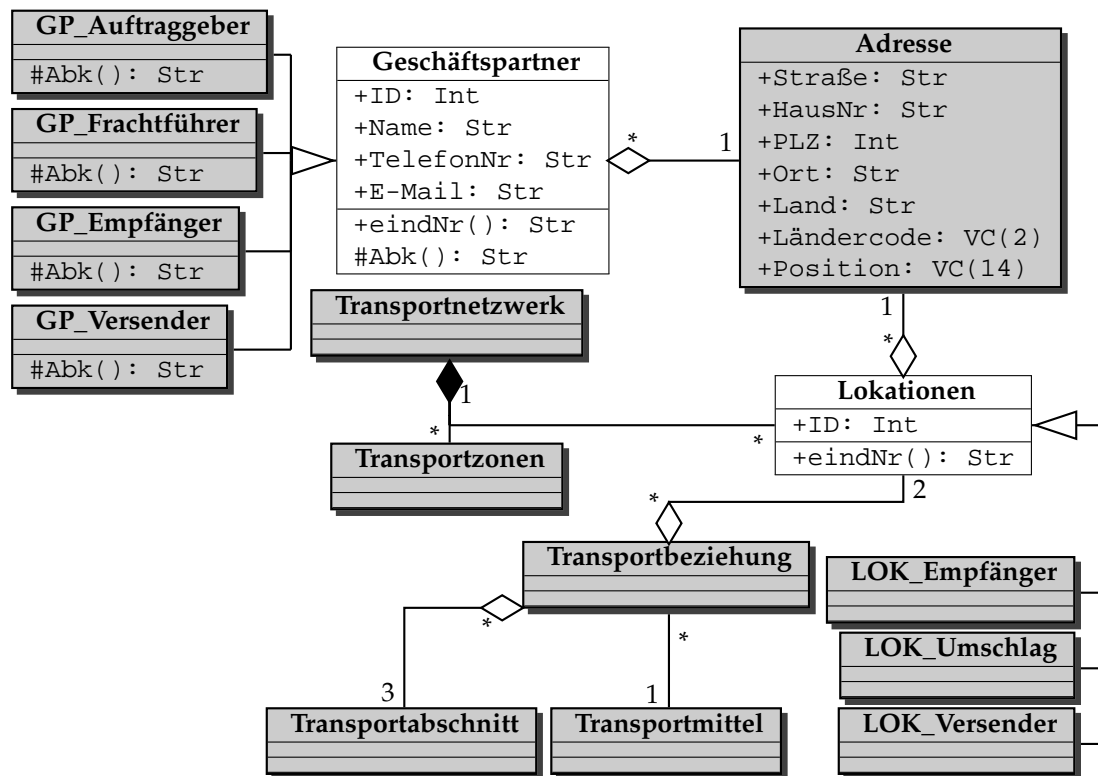
Wir machen aus der Klasse **Lokationen** eine generelle Klasse für die speziellen Klassen **LOK\_Versender**, **LOK\_Empfänger** und **LOK\_Umschlag**.

### 3.10 A10

#### 3.10.1 Anforderung

Zwischen Lokationen bestehen Transportbeziehungen, die eine Erreichbarkeit einer Ziellokation ausgehend von einer Quelllokation mit einem Transportmittel darstellen. Falls keine Transportbeziehung zwischen Lokationen besteht, kann sie nicht in dem Planungsprozess berücksichtigt werden [A30]. Der erste Transportabschnitt wird Vorlauf, der letzte Nachlauf genannt. Der Transportabschnitt dazwischen wird als Hauptlauf bezeichnet.

#### 3.10.2 UML-Diagramm



#### 3.10.3 Beschreibung

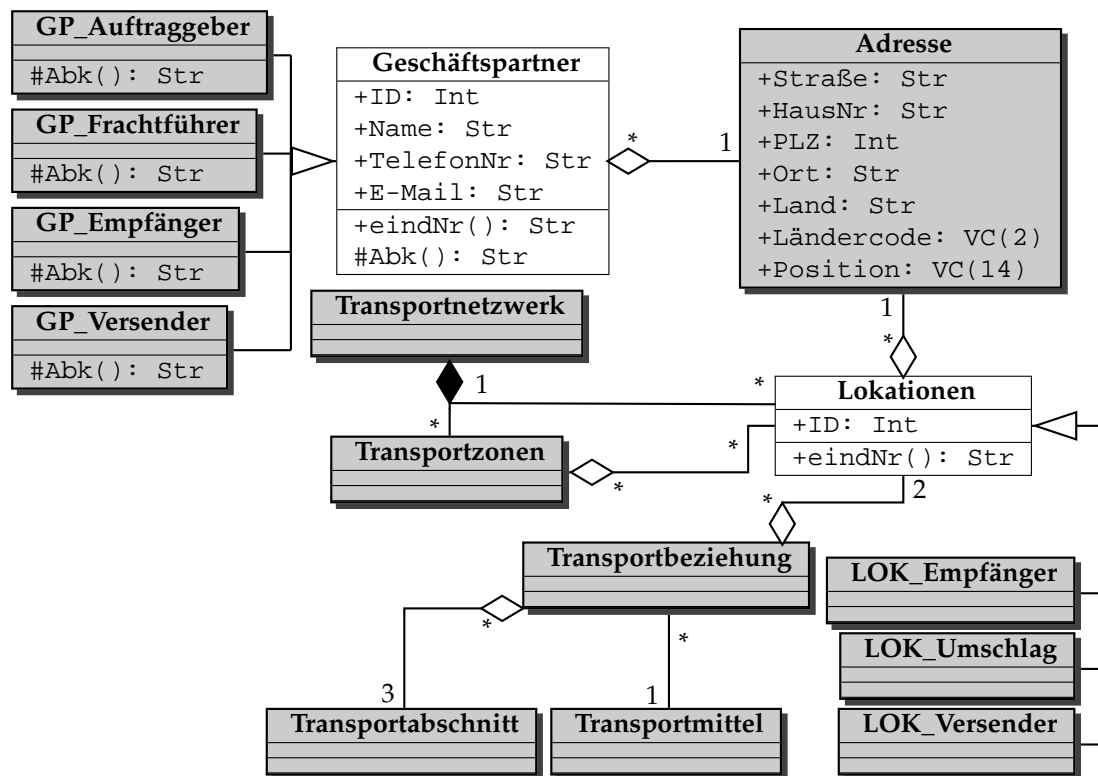
Zunächst erstellen wir die Klasse **Transportbeziehung** die genau zwei **Lokationen** aggregiert. **Lokationen** können in beliebig vielen **Transportbeziehungen** stehen. Dies könnte man auch präziser mit zwei Aggregationen darstellen, wovon eine als *Ziellokation* und eine als *Quelllokation* bezeichnet würde, wovon wir aus Mangel an Platz im Diagramm verzichten. Eine **Transportbeziehung** steht in Beziehung (könnte man auch als Aggregation auffassen) zu genau einem **Transportmittel**, welches in beliebig vielen **Transportbeziehungen** auftauchen kann. Diese Beziehung könnte man auch stattdessen zwischen **Transportmittel** und **Transportabschnitt** modellieren, dann könnte jeder **Transportabschnitt** mit einem anderem **Transportmittel** erfolgen. Ein **Transportbeziehung** aggregiert genau drei **Transportabschnitte**, welche in beliebig vielen **Transportbeziehungen** auftauchen können. Auch hier könnte man dies präziser mit drei Aggregationen und den Bezeichnungen *Vorlauf*, *Hauptlauf* und *Nachlauf* darstellen.

### 3.11 A11

#### 3.11.1 Anforderung

Lokationen können in Transportzonen gruppiert werden, die eine Definition von Transportbeziehungen erleichtern sollen.

#### 3.11.2 UML-Diagramm



#### 3.11.3 Beschreibung

Das „gruppieren“ modellieren wir über eine Aggregation von **Lokationen** in **Transportzonen** mit beliebiger Anzahl. Soll heißen: Eine **Transportzone** enthält beliebig viele **Lokationen** und eine **Lokation** kann in beliebig vielen **Transportzonen** enthalten sein.

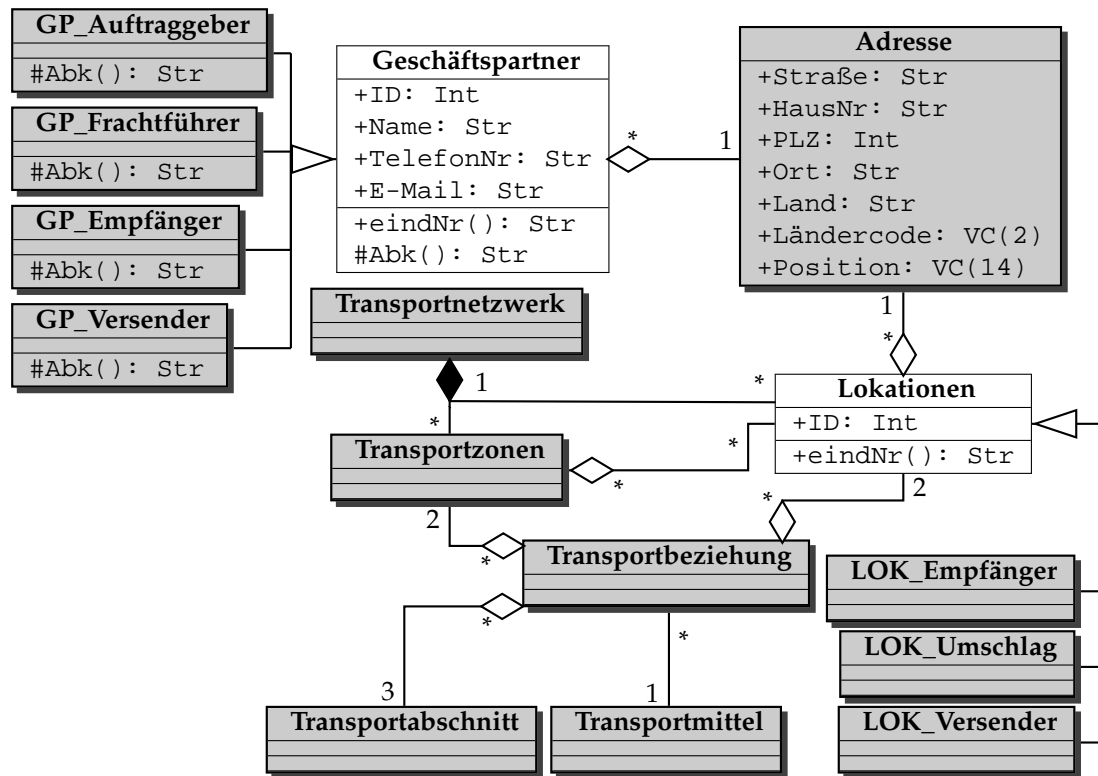
Die „Erleichterung“ wird in [A12] umgesetzt.

## 3.12 A12

### 3.12.1 Anforderung

Transportbeziehungen können auch zwischen Transportzonen [A11] definiert werden. Dies bedeutet, dass jede Lokation der Zielzone von jeder Lokation der Quellzone aus erreichbar ist.

### 3.12.2 UML-Diagramm



### 3.12.3 Beschreibung

Wie in [A10] zwischen **Lokationen** und **Transportbeziehungen** werden **Transportzonen** von **Transportbeziehungen** aggregiert.

Hierbei ist anzumerken, dass diese neue Aggregation mit der aus [A10] kontravalent<sup>3</sup> ist. Was heißt, dass eine **Transportbeziehung** entweder zwei **Transportzonen** oder zwei **Lokationen** enthält. Dies ist leider nicht präzise im Diagramm darzustellen.

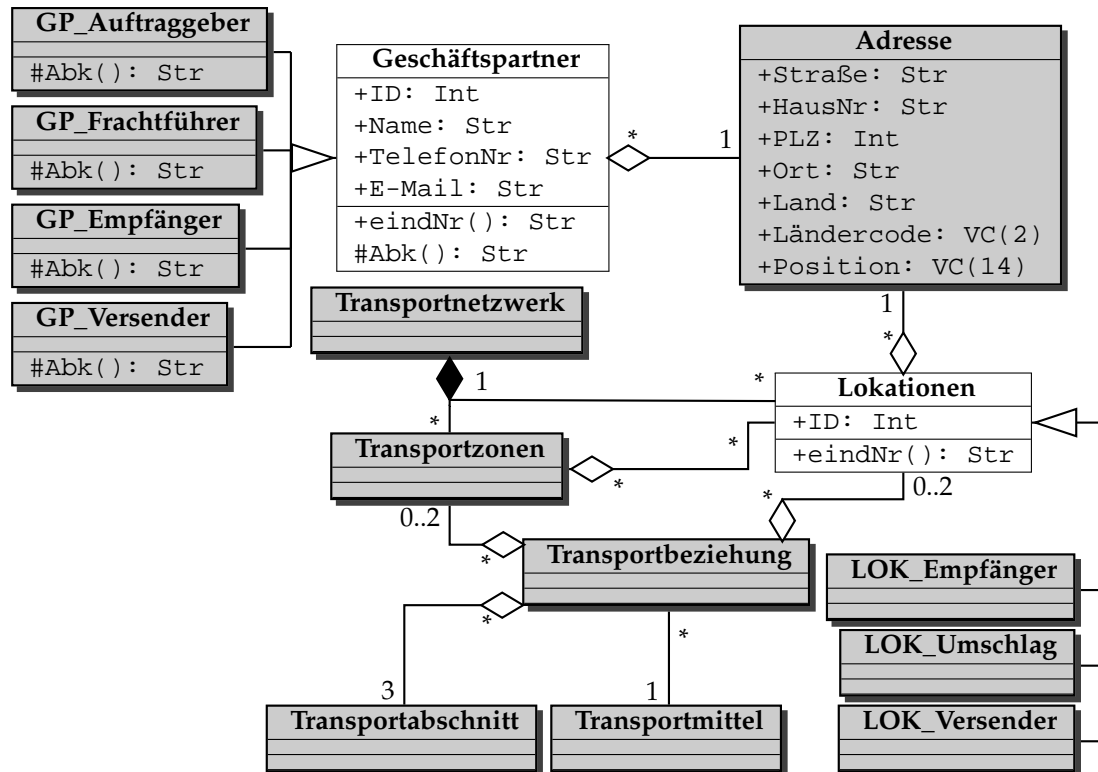
<sup>3</sup>oder auch: ausschließend disjunkt, also das exklusive Oder.  $A \dot{\vee} B := (A \vee B) \wedge \neg(A \wedge B)$

### 3.13 A13

#### 3.13.1 Anforderung

Transportbeziehungen zwischen einer Transportzone und einer Lokation bedeuten entsprechend, dass alle Lokationen der Transportzone von der Einzellokation aus erreichbar ist (im Falle einer Quelllokation) bzw. dass die Einzellokation von allen Lokationen der Transportzone aus erreichbar ist (im Falle einer Ziellokation).

#### 3.13.2 UML-Diagramm



#### 3.13.3 Beschreibung

Um dies darzustellen ändern wir die Anzahl bei den Aggregationen zwischen **Lokationen** und **Transportbeziehungen** sowie zwischen **Transportzonen** und **Transportbeziehungen** von 2 auf 0..2. Dadurch soll sich die Kontravalenz aus [A12] auf folgendes ändern:

Zwei **Transportzonen** sind in einer **Transportbeziehung** enthalten gdw.<sup>4</sup> keine **Lokation** in der gleichen **Transportbeziehung** enthalten ist.

Zwei **Lokationen** sind in einer **Transportbeziehung** enthalten gdw. keine **Transportzone** in der gleichen **Transportbeziehung** enthalten ist.

Eine **Lokation** ist in einer **Transportbeziehung** gdw. eine **Transportbeziehung** in der gleichen **Transportbeziehung** enthalten ist.

<sup>4</sup>steht für „genau dann wenn“, und meint die Logische Äquivalenz.  $A \Leftrightarrow B := (A \rightarrow B) \wedge (A \leftarrow B)$

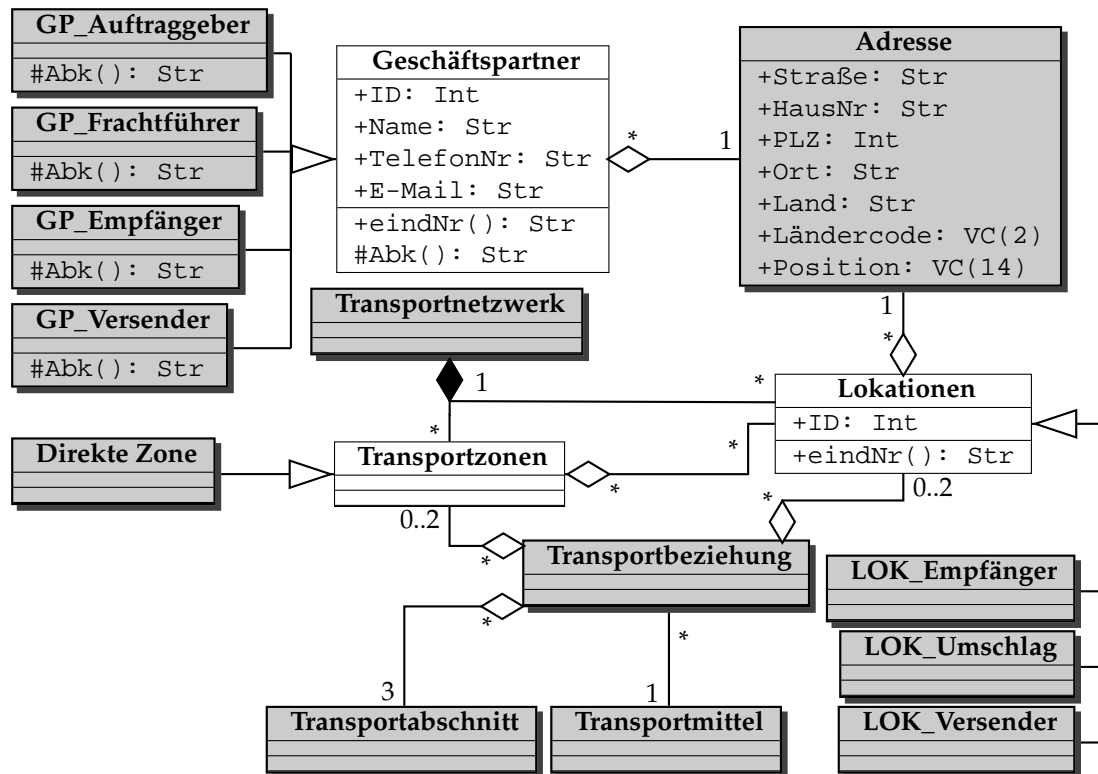


### 3.14 A14

#### 3.14.1 Anforderung

Eine Transportzone der Art „Direkte Zone“ enthält unmittelbar einzeln zugeordnete Lokationen.

#### 3.14.2 UML-Diagramm



#### 3.14.3 Beschreibung

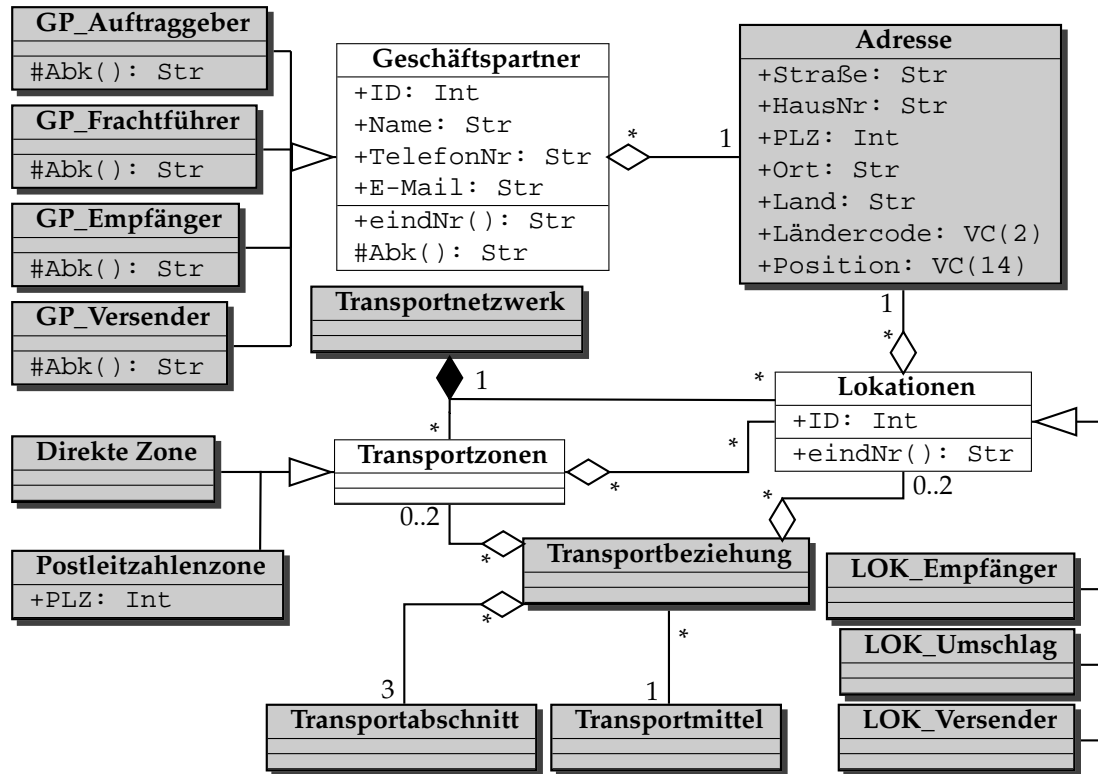
Die Klasse **Transportzone** wird eine Generalisierung und die neue Klasse **Direkte Zone** ist eine Spezialisierung dieser. Wobei **Direkte Zonen** genau so funktionieren wie wir es bisher von **Transportzonen** gewohnt waren.

### 3.15 A15

#### 3.15.1 Anforderung

Eine Transportzone der Art „Postleitzahlenzone“ enthält alle Lokationen, denen ein bestimmter Postleitzahlenbereich in ihrem Ort zugeordnet ist. Eine Teilspezifikation des Postleitzahlenbereichs soll im System möglich sein (z. B. „20099“, „20“).

#### 3.15.2 UML-Diagramm



#### 3.15.3 Beschreibung

Die neue Klasse **Postleitzahlenzone** ist eine Spezialisierung der **Transportzone** die das zusätzliche Attribut **PLZ: Int** enthält. Das Attribut muss keine vollständige Postleitzahl sein.

SQL-Beispiel:

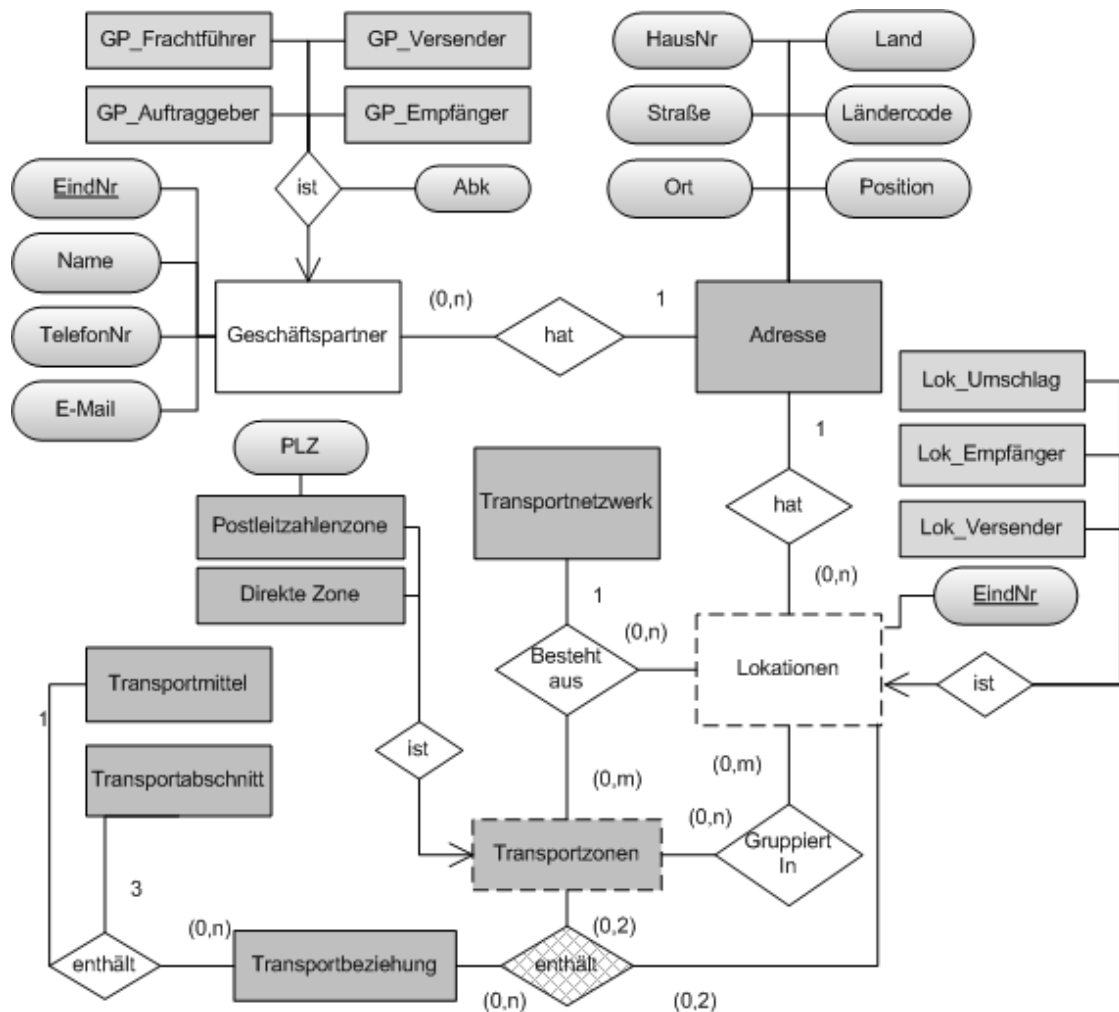
Hinzugefügt werden könnten bereits vorhandene **Lokationen** in dem die **PLZ** der **Adresse** der jeweiligen **Lokation** mit LIKE verglichen wird mit der **PLZ%** der **Postleitzahlenzone**.

## 4 Aufgabe 4 (ER-Modell)

### 4.0 Aufgabenstellung

Transformieren Sie nachfolgend die entstandenen UML-Klassendiagramme in ER-Modelle. Abgegeben ist ein PDF-Dokument mit den ERMen inkl. Dokumentation.

### 4.1 ER-Modell



### 4.2 Beschreibung

Bei der Transformation der UML-Klassendiagramme in ERM ist vor allem darauf zu achten, jede Assoziation zwischen Klassen als Beziehungstyp, in Form einer Raute, darzustellen. Was die Beziehung ist, sollte möglichst genau bezeichnet werden. So werden bei uns z.B. **Lokationen** in **Transportzonen** „gruppiert“.

Die speziellen Klassen die wir haben werden mit einer „<ist>“ Beziehung an ihre jeweilige generelle Klasse gebunden. So ist zum Beispiel die Klasse **GP\_Frachtführer** eine spezielle Klasse der generellen Klasse **Geschäftspartner**, die alle Attribute von ihr erbt.

Da die Klassen **Transportzonen** und **Lokationen** im UML-Diagramm mit einer Komposition an **Transportnetzwerk** gebunden ist, wurden diese im ERM mit gestrichelter Umrandung dargestellt, da die übliche Notation einer doppelten Umrandung im verwendeten Darstellungsprogramm<sup>5</sup> nicht zur Verfügung stand. Also bedeutet die gestrichelte Umrandung, dass die beiden Klassen vom jeweiligen **Transportnetzwerk** unseres HAW-Logistics-Systems abhängig sind. Es ist nicht möglich Instanzen dieser Klassen zu erstellen die nicht im **Transportnetzwerk** sind.

Zum Verdeutlichen der besonderen Beziehung [A13] von **Transportbeziehungen** mit **Lokationen** und **Transportzonen** ist die Beziehung schraffiert dargestellt.

Allgemein wurde die Kardinalität \* im ERM durch (0,n) bzw. (0,m) für  $n, m \in \mathbb{N}$  ersetzt was gleichbedeutend ist.

## 5 Aufgabe 5 (SQL)


### 5.0 Aufgabenstellung

Experimentieren Sie mit den bisher erlernten SQL-Befehlen auf Ihrer Instanz der DB herum. Hier ist keine Dokumentation erforderlich.

---

<sup>5</sup>Microsoft® Office Visio® 2007

## Informationen zur Signatur

	<b>Unterzeichner</b>	EMAILADDRESS=robin.ladiges@haw-hamburg.de, CN=Robin Christopher Ladiges
	<b>Datum/Zeit</b>	Sun Jun 27 00:09:23 CEST 2010
	<b>Austeller-Zertifikat</b>	CN=CAcert Class 3 Root, OU=http://www.CAcert.org, O=CAcert Inc.
	<b>Serien-Nr.</b>	44727
	<b>Methode</b>	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signatur)