

Praktikum 3: Objektorientierte Analyse (OOA) I

Bearbeitung: SW 3 / KW 10

Velo

Gegeben ist folgender Projektauftrag:

- Zu entwickeln ist ein Modell eines Velos für eine Velo-Handels-Kette.

Aufgabe:

- Analysieren Sie die Problem-Domain.
- Zeichnen Sie ein entsprechendes Klassendiagramm mit den wichtigsten Klassen (max. ca. 8 Klassen).
- Erweitern Sie das Klassendiagramm um Verallgemeinerungen (Vererbung).

Eisenbahn

Gegeben seien folgende Begriffe aus der Problem-Domain:

- Personenzug, Gepäckwaggon, Personenwaggon, Lok, Motor*

Aufgabe:

- Identifizieren Sie die Klassen und entwerfen Sie ein Klassendiagramm.
- Erweitern Sie um eine oder mehrere Klassen, so dass die Vererbung eingesetzt werden kann.

Textverarbeitungssystem

Gegeben sei folgende Anforderung für ein Textverarbeitungssystem:

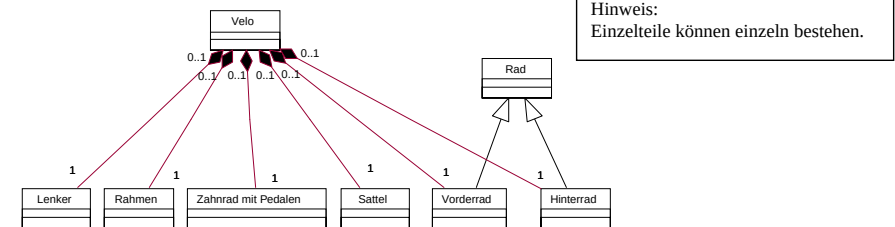
- Das Textverarbeitungssystem erlaubt es dem Benutzer Dokumente anzulegen und zu editieren.
- Ein Dokument kann Text und Bitmap-Graphik enthalten. Text besteht aus Abschnitten, jeder Abschnitt aus Zeichen.
- Ein Dokument enthält ausserdem verschiedene administrative Informationen wie seinen Titel, seinen Autor, den Filenamen in dem es abgelegt ist, sowie das Datum der letzten Änderung.

Aufgabe:

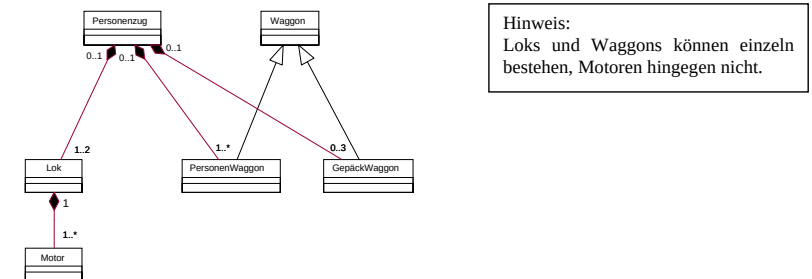
- Welche Klassen lassen sich in der Problem-Domain identifizieren?
- Welche Beziehungen bestehen zwischen den Klassen?
- Zeichnen Sie ein Klassendiagramm.

Musterlösung Praktikum 3: Objektorientierte Analyse (OOA) I

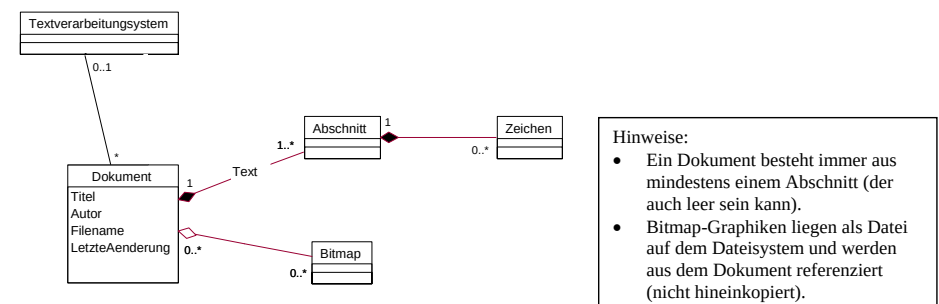
Velo:



Eisenbahn:



Textverarbeitungssystem:



Praktikum 4: Objektorientierte Analyse (OOA) II

Bearbeitung: SW 4 / KW 11

Verkehrsmittel

Gegeben seien folgende Begriffe aus der Problem-Domain:

- *PW, LKW, Autobus, Fähre, Frachtschiff, Floss, Segel-Yacht.*

Aufgabe:

- Identifizieren Sie die Klassen und entwerfen Sie eine Vererbungshierarchie.
- Finden Sie Verallgemeinerungen der Klassen und ergänzen Sie die Vererbungshierarchie.
- Welche Klassen sind abstrakte Klassen?
- Benötigten Sie eine Klasse *Segel-Yacht*?
- Wo ist eine Mehrfachvererbung möglich (denken Sie z.B. daran, dass manche Verkehrsmittel zur öffentlichen Personenbeförderung bestehen)?

Hinweis: Klassen müssen Unterschiede aufweisen.

OST-Verwaltung

Die OST-Verwaltung muss Studenten, Lehrbeauftragte, Hauptlehrer und Fest-Angestellte verwalten.
Es gilt folgende vereinfachte Beschreibung:

Von allen Personen werden die Namen verwaltet. Studenten haben eine Matrikelnummer, Fest-Angestellte eine interne, persönliche Telefonnummer. Bei Lehrbeauftragten und Hauptlehrern muss das Unterrichtpensum (Anzahl Stunden pro Woche) gespeichert werden. Hauptlehrer weisen gleichzeitig die Eigenschaften von Lehrbeauftragten und Fest-Angestellten auf. Für alle Lohnempfänger, d.h. Fest-Angestellte, Lehrbeauftragte und Hauptlehrer, kann das Salär berechnet werden. Der Algorithmus zur Salärberechnung ist davon abhängig, ob es sich um einen Fest-Angestellten, Lehrbeauftragten oder Hauptlehrer handelt.

Aufgabe:

- Analysieren Sie die Problem-Domain.
- Zeichnen Sie ein entsprechendes Klassendiagramm.

Versandhaus

In einem Versandhaus sollen die Kunden und ihre Bestellungen verwaltet werden. Ein Kunde bestellt mit einer Bestellung verschiedene Waren in jeweils unterschiedlicher Anzahl. Jede Ware hat einen bestimmten Preis.

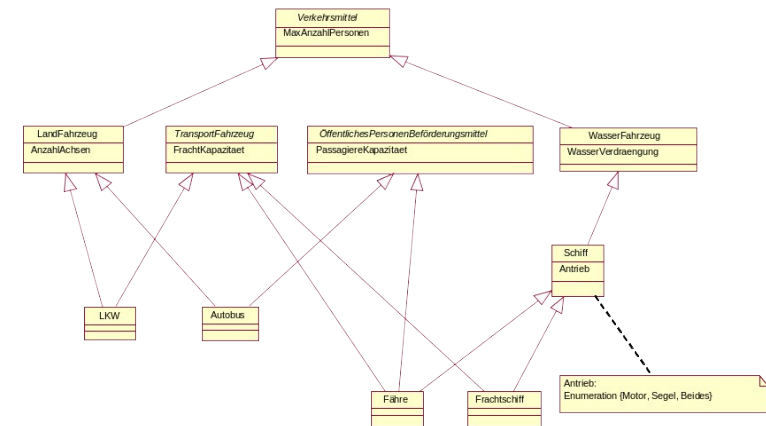
Aufgabe:

- Identifizieren Sie die Klassen und entwerfen Sie ein Klassendiagramm.
- Bestimmen Sie die wichtigsten Attribute und Methoden jeder Klasse.
- Es sollen keine redundanten Daten im System vorhanden sein.
- Erstellen Sie ein Objektdiagramm für den Fall „Berechne Bestellungen-Preis“, wenn mit dieser Bestellung zwei (gleiche) Hemden und drei (gleiche) Hosen bestellt wurden.
- Erweitern Sie das Objektdiagramm für den Fall, dass ein zweiter Kunde zwei Hosen kauft, und zwar dieselben wie der vorherige Kunde.

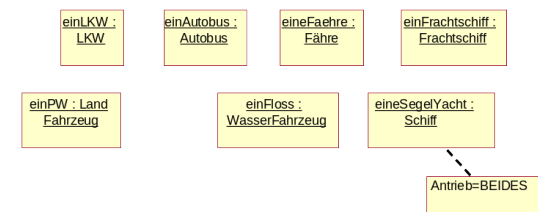
Musterlösung Praktikum 4: Objektorientierte Analyse (OOA) II

Verkehrsmittel

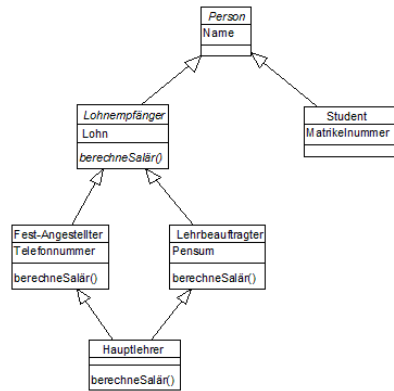
Klassendiagramm:



Objektdiagramm:

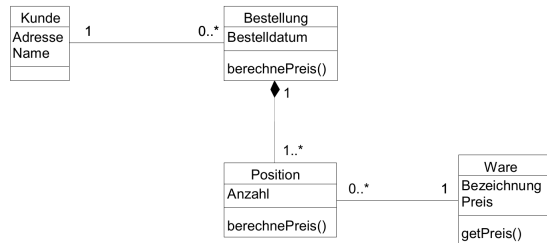


OST-Verwaltung

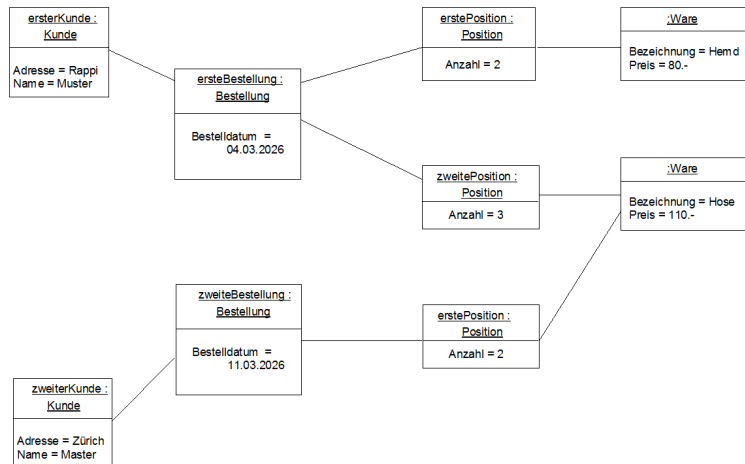


Versandhaus

Klassendiagramm:



Objektdiagramm:



Praktikum 5: Warenlager I

Bearbeitung: SW 5 / KW 12

Domainanalyse für Warenlager

Für ein Warenlager soll ein Software-System entwickelt werden, das die Lagerbewirtschaftung unterstützt.

Über das Warenlager liegen folgende Informationen vor:

Das Warenlager ist in Gänge unterteilt. Am Rande der Gänge stehen Behälter an festen Lagerpositionen. Im Warenlager lässt sich jeder Behälter mit einem Nummernpaar ansprechen, z.B. 12-10 bedeutet im Gang 12 der 10. Behälter. Die Waren gelangen von einem Eingangs-Dock ins Lager (Vor dem Eingangs-Dock parkieren die Lastwagen, welche die Ware anliefern.) Am Eingangsdock werden die Waren auf Paletten abgeladen. Dabei werden auf den entsprechenden Lieferantenbestellungen, die gelieferten Artikel markiert. Nun wird die Palette durch das Warenlager gezogen und die Gegenstände so in die Behälter verteilt, dass in jedem Behälter nur Stücke eines Artikels zu liegen kommen. Beim Verteilen werden zuerst Behälter berücksichtigt, welche alle auf der Palette befindlichen Stücke eines Artikels aufnehmen können. Gibt es keinen solchen Behälter, so wird ein leerer Behälter verwendet.

Kunden, die mit Name und Adresse bekannt sind, bestellen beim Warenlager Artikel. Um eine Kundenbestellung zu bearbeiten, werden die an Lager liegenden Artikel auf einer Palette zusammengestellt und einem Ausgangs-Dock zur Verpackung zugeordnet. Es gibt 5 Ausgang-Docks im Warenlager. Kann eine Bestellung nicht in einem Auftrag abgewickelt werden, weil nicht alle Artikel an Lager liegen, so wird ein Folgeauftrag zur späteren Bearbeitung erzeugt.

Das System soll folgende Funktionalität aufweisen:

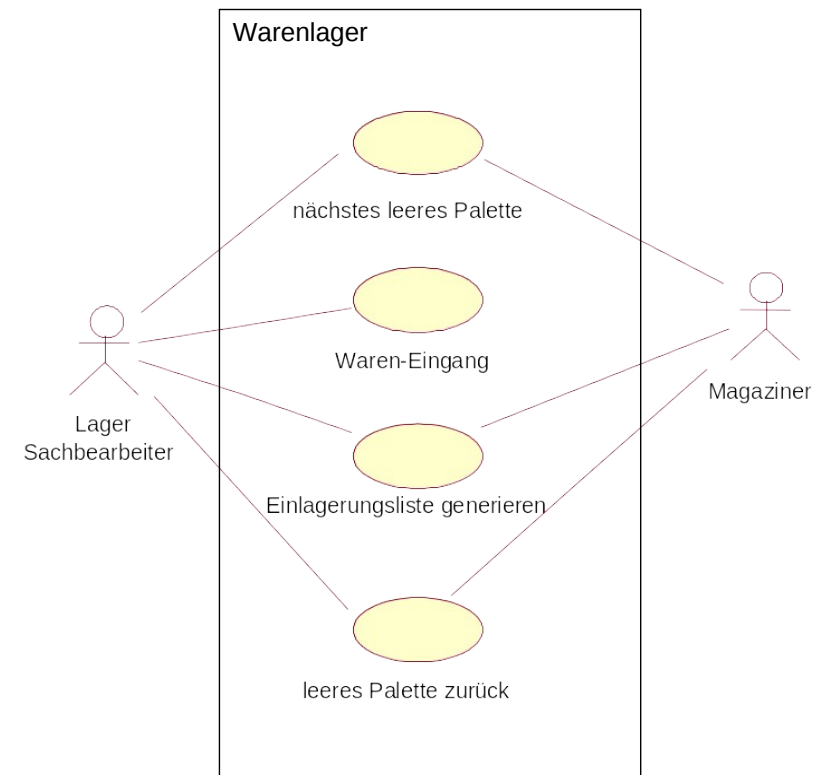
- Verwalten und Buchführen über den Inhalt aller Behälter. Dazu muss für jeden Behälter das Hinzufügen und Entfernen von Artikeln ins System eingegeben werden.
- Verwalten und Buchführen über die Ladung aller Paletten (Jede Palette wird durch eine Nummer identifiziert). Das System muss jederzeit wissen, welche Waren sich auf welchen Paletten befinden. Dazu muss für jede Palette das Hinzufügen und Entfernen von Artikeln ins System eingegeben werden. Wird eine leere Palette benötigt, so wird sie vom System zugewiesen. Dabei sollen immer die zuletzt freigewordenen Paletten wieder verwendet werden.
- Das System soll für am Eingangs-Dock mit einzulagernden Artikel beladene Paletten eine Einlagerungsliste generieren, die auflistet, in welche Behälter die Artikel abgelegt werden sollen.
- Das System soll für jede Kundenbestellung eine Auslagerungsliste generieren, die aufführt, welche Artikel in welchen Stückzahlen von welchen Behältern aus dem Lager geholt werden müssen. Kann mit einem Lagerauftrag eine Kundenbestellung nicht befriedigt werden, so wird automatisch ein Lagerauftrag zur späteren Bearbeitung erstellt.
- Die Zuordnung von Lieferantenbestellung zu Artikel und zu Behälter muss gewährleistet sein.
- Das System soll jederzeit Inventarlisten generieren können.
- Es gibt genau ein Eingabe-Terminal (PC) im Warenlager.

Folgende Dinge gehören nicht in den Verantwortungsbereich des Systems:

- Verwaltung des ausführenden Personals.
- Automatisierung des Lagerbetriebs. Das System ist nur eine logistische Unterstützung.

Musterlösung Praktikum 5: Warenlager I

Aufgabe 1: Use-Case-Diagramm



Aufgabe 1: Use-Case-Diagramm

Stellen Sie den Eingang von Warenlieferungen mit einem *Use-Case-Diagramm* dar.

Der Wareneingang funktioniert derart, dass die Waren erfasst und sofort eingelagert werden.
Oder auch, dass zuerst mehrere Lieferungen erfasst werden und erst daraufhin eingelagert werden.

Aufgabe 2: Use-Case-Beschreibung

Erstellen Sie für obige *Use-Cases* je eine Beschreibung mit der *Use-Case-Schablone* (gemäss Vorlesung LE 4: Folienskript S. 28+29; Buch S.68).

Wir führen dazu noch folgende Erweiterung ein:

Wenn eine *Iteration* durchgeführt werden soll (z.B. von der Palette soll ein Artikel nach dem anderen heruntergenommen werden), kann dies mit einem Stern '*' dargestellt werden.

Weitere Nummerierung innerhalb der Iteration mit zweiter, zusätzlicher Stelle.

Beispiel:

- 1 ...
- 2 * nächster Artikel von Palette nehmen
- 2.1 ...
- 2.2 ...
- 4 ...
- 5 ...

Aufgabe 2: Use-Case-Beschreibungen

Use-Case "nächstes leeres Palette":

Use-Case:

nächstes leeres Palette

Ziel:

Zuweisung des nächsten unbenutzten Palettes.

Kategorie:

Primär

Vorbedingung:

-

Nachbedingung Erfolg:

Palette ist als ‚benutzt‘ markiert.

Nachbedingung Fehlschlag:

-

Akteure:

Lager-Sachbearbeiter
Magaziner

Auslösendes Ereignis:

Leeres Palette wird benötigt.

Beschreibung:

- 1 Paletten-Nummer wird vom System bekanntgegeben.
Wobei zuletzt frei gewordene Paletten zuerst ausgewählt werden.

Erweiterungen:

-

Alternativen:

- 1a Es wird mitgeteilt, dass es keine leeren Paletten mehr hat.

Use-Case "Waren-Eingang":

Use-Case:

Waren-Eingang

Ziel:

Waren auf Palette sind erfasst und Lieferantenbestellung ist nachgeführt.

Kategorie:

Primär

Vorbedingung:

- Waren sind auf einer Palette, welche mit Use-Case "nächstes leeres Palette" zugewiesen wurde.
- Die Anzahl der Artikel ist nicht grösser als es auf einer Palette dafür Platz hat.

Nachbedingung Erfolg:

Waren auf Palette erfasst, Lieferantenbestellung(en) nachgeführt.

Nachbedingung Fehlschlag:

Fehlerhafte Waren sind nicht erfasst.

Meldung mit Beschreibung an Chef Abteilung "Beschaffung".

Akteure:

Lager-Sachbearbeiter

Auslösendes Ereignis:

Palette mit neuer Ware soll erfasst werden.

Beschreibung:

- 1 Paletten-Nummer wird eingeben.
- 2 * Nächster Artikel von Palette (Iteration):
 - 2.1 Entsprechende Bestellposition auf Lieferantenbestellung wird nachgeführt.
 - 2.2 Ware wird auf Palette eingetragen.

Erweiterungen:

-

Alternativen:

- 2.1.a Wenn Ware nicht bestellt oder nicht vollständig (Teillieferung):
Ware wird von Palette heruntergenommen (und separiert).
Ware wird nicht erfasst!
Weiter mit *nächster Artikel von Palette*.

Use-Case "Einlagerungsliste generieren":

Use-Case:

Einlagerungsliste generieren

Ziel:

Einlagerungsliste für ein bestimmtes Palette ist generiert.

Kategorie:

Primär

Vorbedingung:

Use-Case "Waren-Eingang" für das betreffende Palette erfolgreich abgeschlossen.

Nachbedingung Erfolg:

Einlagerungsliste generiert.

Einlagerungspositionen in den Behältern reserviert für die Artikel auf dem Palette.

Nachbedingung Fehlschlag:

Meldung mit Beschreibung an Chef Abteilung "Lager".

Akteure:

Lager-Sachbearbeiter

Magaziner

Auslösendes Ereignis:

Palette soll eingelagert werden.

Beschreibung:

- 1 Paletten-Nummer wird eingeben.
- 2 * Nächster Artikel von Palette (Iteration):
 - 2.1 System sucht für die Ware nach passendem Behälter (Behälter mit gleicher Ware und genügend Platz oder leerer Behälter).
 - 2.2 System reserviert Platz in gefundenem Behälter und druckt Ware mit Behälter auf Drucker (⇒ Einlagerungsliste).
- 3 Rückmeldung vom System an Akteur, nachdem Einlagerungsliste fertig gedruckt ist.

Erweiterungen:

- 2.1a Wenn kein leerer Behälter mehr vorhanden ⇒ "Fehler-Meldung".
Ware wird von Palette heruntergenommen (und separiert).
Weiter mit *nächstem Artikel von Palette*.

Alternativen:

-

Use-Case "leeres Palette zurück":

Use-Case:

leeres Palette zurück

Ziel:

Leeres Palette zurückmelden, sodass dieses wieder für die Einlagerung zur Verfügung steht.

Kategorie:

Primär

Vorbedingung:

Waren sind gemäss Einlagerungsliste eingelagert worden.

Nachbedingung Erfolg:

Artikel gemäss Einlagerungsliste als 'eingelagert' verbucht.

Palette ist als ‚nicht benutzt‘ markiert.

Nachbedingung Fehlschlag:

-

Akteure:

Lager-Sachbearbeiter

Magaziner

Auslösendes Ereignis:

Leeres Palette steht nach Wareneinlagerung wieder zur Verfügung.

Beschreibung:

- 1 Paletten-Nummer wird eingeben.
- 2 System bucht Waren definitiv in Behälter ein.
- 3 Ware wird auf Palette ausgetragen.
- 4 Palette wird als ‚nicht benutzt‘ markiert.

Erweiterungen:

-

Alternativen:

- 2a Ungültige Paletten-Nummer ⇒ "Fehler-Meldung".
Abbruch.

Praktikum 6: Warenlager II

Bearbeitung: SW 6 / KW 14

Domainanalyse für Warenlager

Aufgabe 1: Klassen-Diagramm

Erstellen Sie für das Warenlager aus dem Praktikum 5 ein entsprechendes Klassendiagramm.

Hinweis:

Konzentrieren Sie sich zuerst auf den *Wareneingang* (incl. Lieferantenbestellung).

Mit der Aufgabe 2 werden Sie dann den Wareneingang *testen*.

Aufgabe 2: Objekt-Diagramm

Es ist folgende Ausgangssituation gegeben:

Im Lager sind genau (und nur) zwei *Artikel* erfasst:

Name	Art.-Nr.	Max.Anzahl in Behälter
Locher	1000	100
Bostich	1001	300

Im Lager sind folgende Artikel bereits vorhanden:

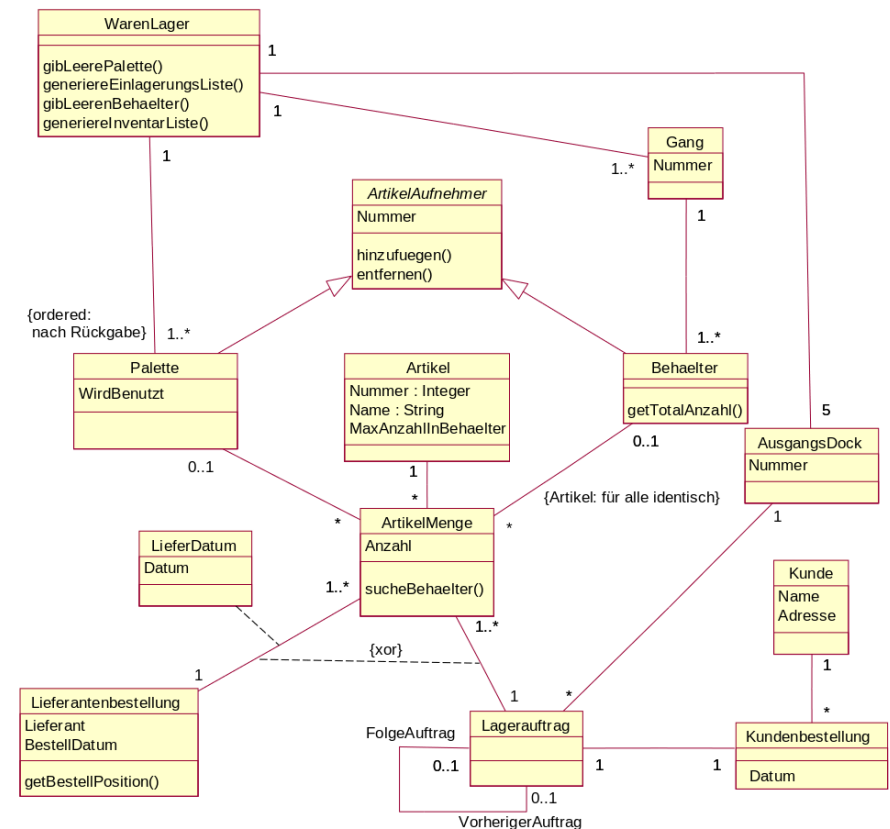
Name	Art.-Nr.	Anzahl	Behälter	Lieferant	Bestell-Datum	Liefer-Datum
Locher	1000	80	1-1	BüroImport AG	15.11.23	01.12.23
Bostich	1001	150	1-2	BüroImport AG	15.11.23	02.12.23

Bemerkung: alle Artikel wurden mit der gleichen Bestellung bestellt.

Erstellen Sie ein **Objekt-Diagramm** für obige Ausgangslage für den Zeitpunkt vom 04.12.23 .

Musterlösung Praktikum 6: Warenlager II

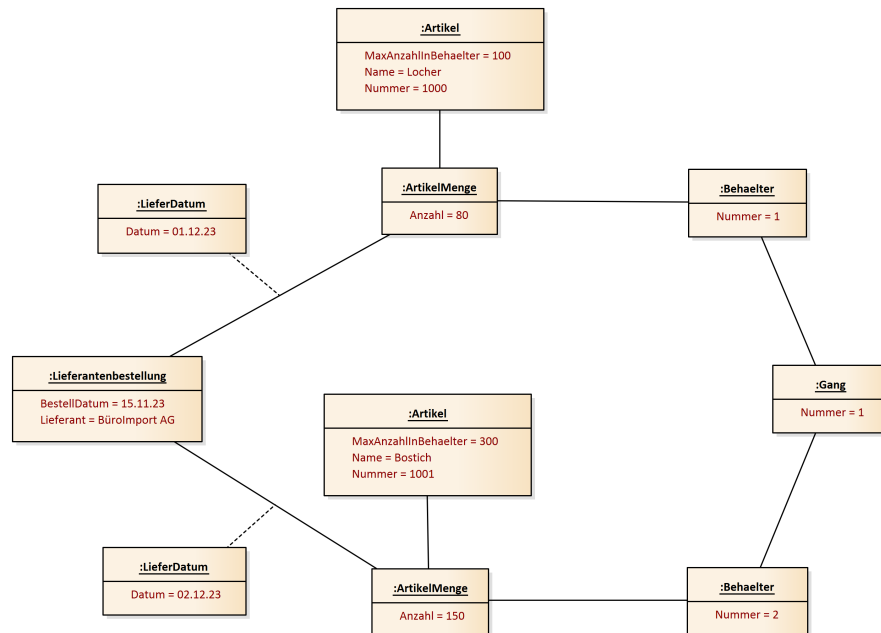
Aufgabe 1: Klassen-Diagramm



Bemerkungen:

- 'Gang' und 'Ausgangsdock' müssten nicht zwingend als Klasse modelliert werden (könnten auch nur Attribute in den entsprechenden Klassen sein).
In der Analyse aber noch zweckmässig.

Aufgabe 2: Objekt-Diagramm



Praktikum 7: Warenlager III

Bearbeitung: SW 7 / KW 16

Kommunikation- und Sequenz-Diagramme

Ausgehend vom Praktikum *Warenlager I* und dem Klassendiagramm *Warenlager II* (Musterlösung) ist folgende Situation gegeben:

Am 02.12.23 wurden weitere 200 Bostich's bei der Firma BüroImport AG bestellt.
Diese sind am 05.12.23 geliefert worden.

Aufgabe 1: Objektdiagramm

Erweitern Sie das Objektdiagramm aus dem Praktikum 6 um diese Lieferung (d.h. den Zustand nachdem die Lieferung komplett eingelagert wurde).

Aufgabe 2: Kommunikation- und Sequenz-Diagramm für Use Case „nächstes leeres Palette“

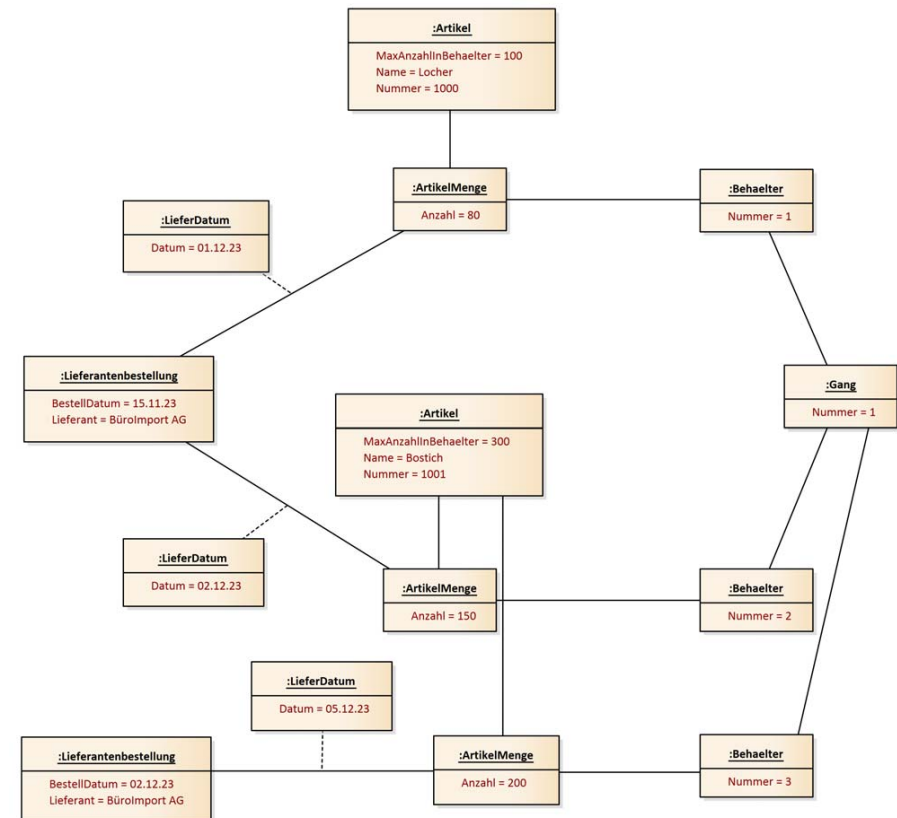
Erstellen Sie für genau diese Lieferung vom 05.12.23 je ein *Kommunikations-* und *Sequenz-Diagramm* für den Use-Case „nächstes leeres Palette“.

Hinweise:

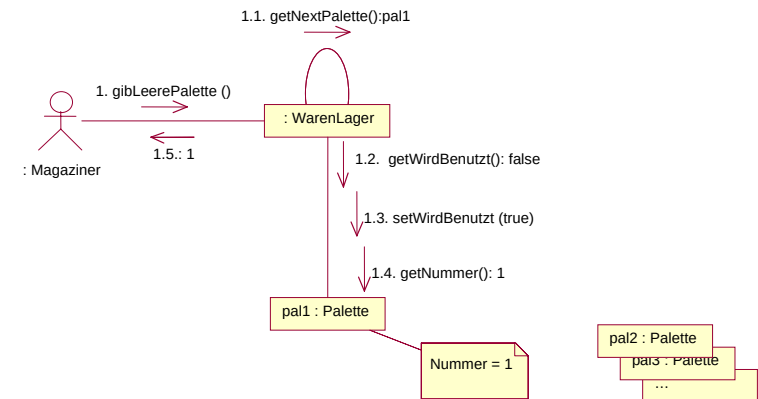
- Es sind momentan alle Paletten frei.
- Im Lager sind nur die Waren vorhanden gemäss Praktikum 6.

Musterlösung Praktikum 7: Warenlager III

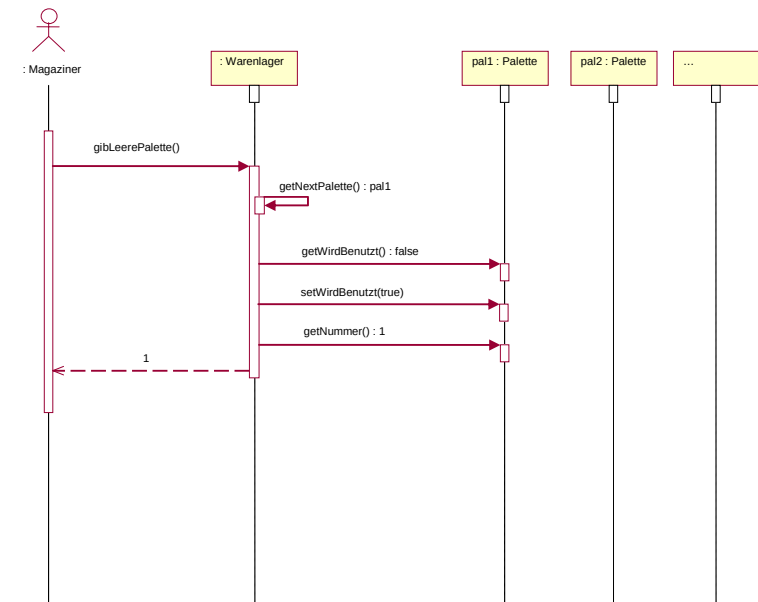
Aufgabe 1: Objektdiagramm



Aufgabe 2: Kommunikations-Diagramm "nächstes leeres Palette"



Sequenz-Diagramm "nächstes leeres Palette"



Aufgabe 3: Kommunikation- oder Sequenz-Diagramm für Use Case „Waren-Eingang“

Erstellen Sie für genau diese Lieferung ein *Kommunikations- oder Sequenz-Diagramm* für den Use-Case „Waren-Eingang“.

Hinweis:

Die Operation *LieferantenBestellung.getBestellPosition()* wird benutzt um eine bestimmte Bestell-Position zu suchen. Sie gibt dann die gefundene *ArtikelMenge* zurück.

Spezifikation:

`getBestellPosition(Anzahl:Integer, Bezeichnung:String): ArtikelMenge`

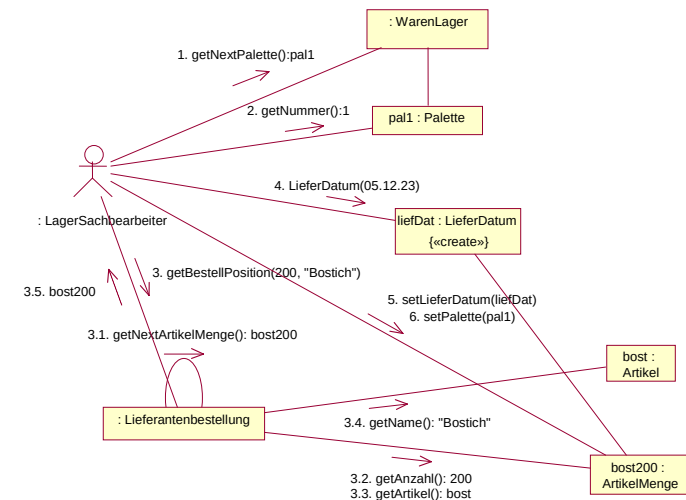
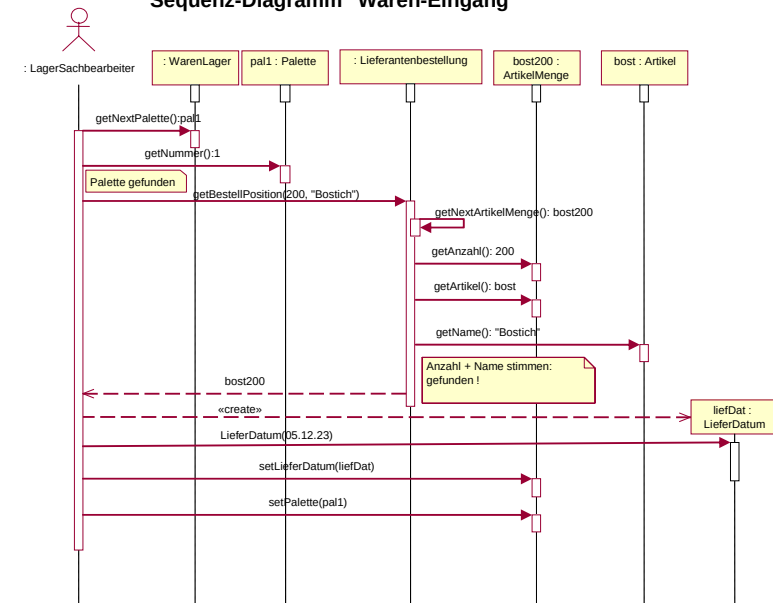
Beispiel:

`getBestellPosition(200, "Bostich")`

Allgemeine Bemerkung:

Um über Objekte einer Kollektion (Liste; *-Multiplizität) iterieren zu können, hat jede Klasse, wenn nötig, implizit entsprechende Methoden $\Rightarrow$ getNext*

Beispiel: **WarenLager::getNextGang() : Gang**

Aufgabe 3: Kommunikations-Diagramm "Waren-Eingang"**Sequenz-Diagramm "Waren-Eingang"**

Aufgabe 2 : Objektorientierte Analyse

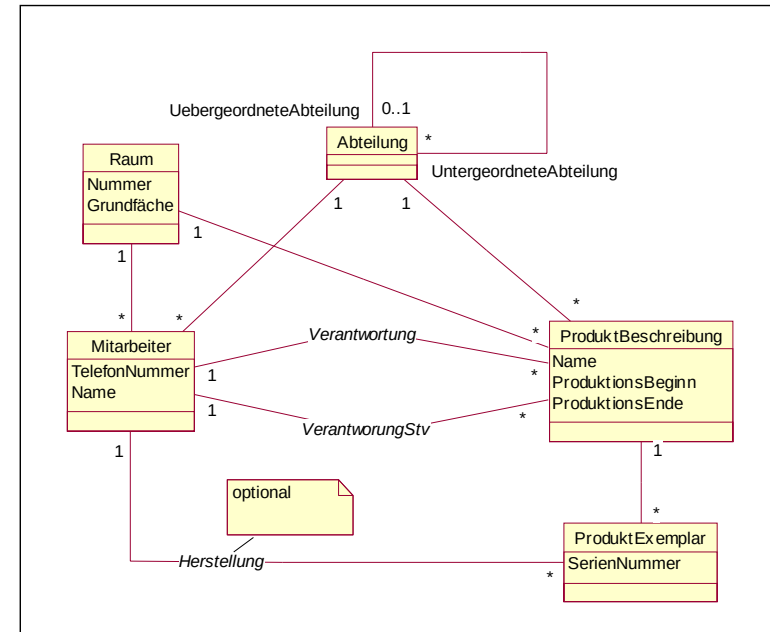
Sie haben den Auftrag eine objektorientierte Analyse von einer Firma durchzuführen.
Folgende Aussagen wurden zusammengetragen:

1. Die Firma besteht aus Abteilungen.
2. Abteilungen können selber wieder Abteilungen unterstellt sein.
3. Die Mitarbeiter sind jeweils einer Abteilung zugeteilt.
4. Von allen Räumen der Firma sind jeweils die Grundfläche erfasst.
5. Einige Abteilungen sind Produktions-Abteilungen, die verschiedene Produkte produzieren.
6. Pro Produkt ist dessen Name, der Produktions-Raum, sowie auch ab wann dieses Produkt produziert wurde, erfasst.
7. Pro Produkt gibt es einen verantwortlichen Mitarbieter. Es gibt zusätzlich auch einen stellvertretenden Produkt-Verantwortlichen.
8. Wenn ein einzelenes Produkt zur Reparatur in die Firma zurückkommt, muss der Mitarbeiter bestimmt werden können, der dieses Produkt gebaut hat.
9. Man kann jederzeit eine Liste generieren mit den Produkten die nicht mehr produziert werden.
10. Man kann jederzeit eine Telefonliste mit allen Mitarbeiter erzeugen. In dieser ist jeweils auch die Raum-Nummer ersichtlich, in welchem der Mitarbeiter seinen Arbeitsplatz hat.

Erstellen Sie ein entsprechendes Klassendiagramm.

Wenn bei Multiplizitäten ein Interpretationsspielraum vorliegt, nehmen Sie den allgemeinsten Fall an.

Verwenden Sie ein separates Lösungsblatt.

Aufgabe 2 : Objektorientierte Analyse

Aufgabe 5 : Design-Modellierung

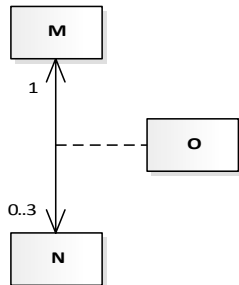
Gegeben sind die zwei Klassen **A** und **B**:

- ein **A** enthält ein **B**.
- **B** sieht **A** nicht.
- es können mehr **B**-Objekte als **A**-Objekte instanziiert sein.
- wenn **A** kopiert wird, wird das entsprechende **B** automatisch mitkopiert.

Erstellen Sie ein entsprechendes Klassendiagramm.

Aufgabe 6 : Assoziationsklasse

Programmieren Sie die Klassen gemäss nachfolgendem Klassendiagramm:



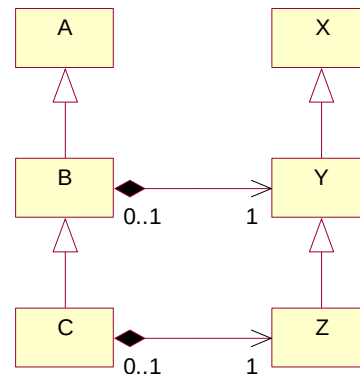
Bemerkungen:

Die Definitionen müssen sich alle in einem einzigen File befinden.
Sie müssen nur definieren und sicherstellen, was im Klassendiagramm dargestellt ist (somit z.B. keine zusätzlichen *Setter*- und *Getter*-Methoden, Konstruktoren etc.)
Das File muss nur kompilierbar sein (nicht linkbar).

(ca. ein Dutzend Zeilen können reichen)

Aufgabe 7 : Objekt-Diagramm

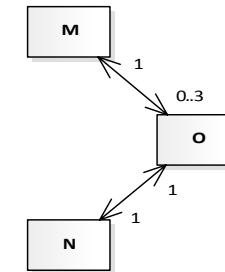
Gegeben ist folgendes Klassendiagramm:



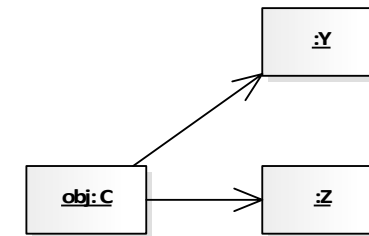
Zeichnen Sie ein *Objekt-Diagramm* für den Fall, dass ein Objekt mit dem Namen "**obj**" der Klasse **C** instanziiert ist.

Aufgabe 5 : Design-Modellierung**Aufgabe 6 : Assoziationsklasse**

```
class O;  
class M {  
    O* mO[3];  
};  
class N;  
class O {  
    M* mM;  
    N* mN;  
};  
class N {  
    O* mO;  
};
```

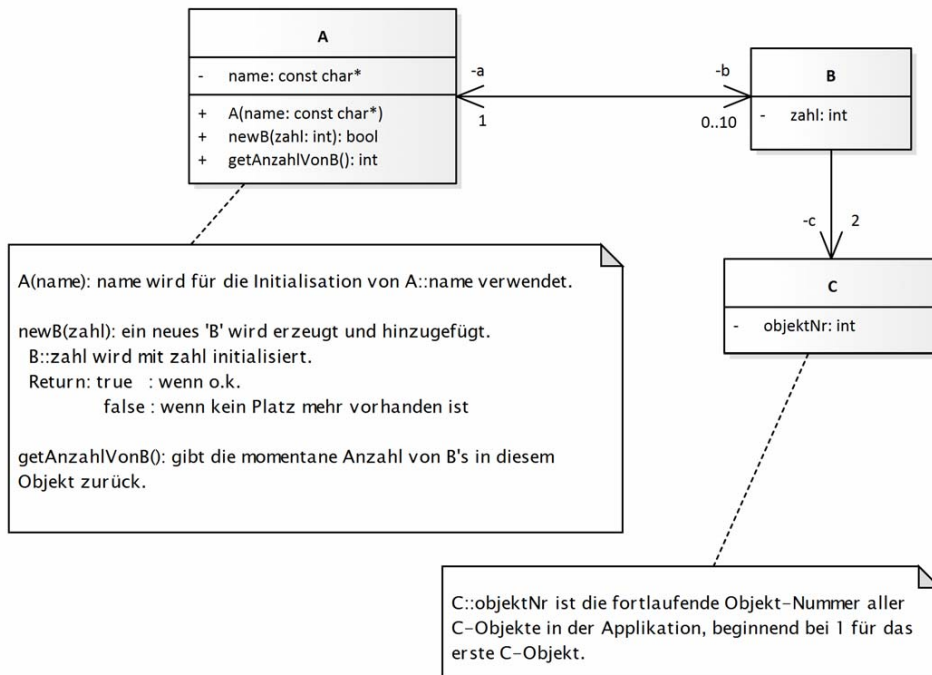


Hinweis: Reihenfolge der Klassen kann auch anders sein.

Aufgabe 7 : Objekt-Diagramm

Aufgabe 8 : Design und Implementation einer Applikation

Von einer Applikation ist die erste Version des Designs fertiggestellt:



Im Weiteren wurde folgende *Test-Applikation* definiert:

```
#include "A.h"

#include <iostream>

using namespace std;

int main() {
    A a;
    a.newB(1);
    cout << a.getAnzahlVonB();    // Ausgabe: 1
    return 0;
}
```

Programmieren Sie die Klassen A, B und C.

Dabei müssen die Vorgaben des obigen Designs eingehalten werden. Dazu können resp. müssen die Klassen um zusätzliche Elemente ergänzt werden.

Die Klassen müssen je in Header- und Implementationsfile aufgeteilt sein.

Die ganze Applikation (*Test-Applikation* mit Ihren implementierten Klassen) muss fehlerfrei kompilieren, linken und gemäss Vorgaben funktionieren.

Aufgabe 8 : Design und Implementation einer Applikation

```
=== A.h: =====
#ifndef A_H
#define A_H

class B;

class A
{
public:
    A();
    A(const char* name);
    bool newB(int zahl);
    int getAnzahlVonB();

private:
    const char* name;
    B* bArray[10];
    int anzahlB;
};

#endif
=== A.h. =====

=== B.h: =====
#ifndef B_H
#define B_H

class A;
class C;

class B
{
public:
    B(int zahl, A* a);

private:
    int zahl;
    A* a;
    C* cArray[2];
};

#endif
=== B.h. =====

=== C.h: =====
#ifndef C_H
#define C_H

class C
{
public:
    C();

private:
    int objektNr;
    static int objektZaehler;
};

#endif
=== C.h. =====
```

```
=== A.cpp: =====
#include "A.h"
#include "B.h"

A::A() : name(0), anzahlB(0) {
}

A::A(const char* name) : name(name), anzahlB(0) {
}

bool A::newB(int zahl) {
    if (getAnzahlVonB() == 10) {
        return false;
    }
    else {
        bArray[anzahlB] = new B(zahl, this);
        anzahlB++;
        return true;
    }
}

int A::getAnzahlVonB() {
    return anzahlB;
}

=== A.cpp. =====

=== B.cpp: =====
#include "B.h"
#include "C.h"

B::B(int zahl, A* a) : zahl(zahl), a(a)
{
    cArray[0] = new C;
    cArray[1] = new C;
}

=== B.cpp. =====

=== C.cpp: =====
#include "C.h"

int C::objektZaehler = 0;

C::C() {
    objektZaehler++;
    objektNr = objektZaehler;
}

=== C.cpp. =====
```

Aufgabe 3 : Use-Case

Beschreiben Sie den **Zusammenhang** zwischen Use-Case, Szenario, Interaktionsdiagramm, Kommunikationsdiagramm und Sequenzdiagramm.

Ein Use-Case besteht aus einem oder mehreren Szenarios.

Ein Szenario wird mit einem Interaktionsdiagramm beschrieben.

Ein Interaktionsdiagramm ist ein Sequenzdiagramm oder ein Kommunikationsdiagramm.

Aufgabe 4 : Verständnisfragen

Sind die folgenden Aussagen richtig oder falsch ?

(kreuzen Sie *richtig* oder *falsch* entsprechend an)

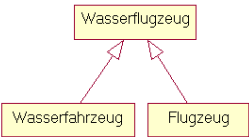
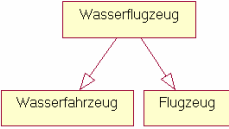
Aussage:	richtig	falsch
Objektdiagramme haben keine Multiplizitäten.	☒	☐
Eine Klasse kann mehrere Protokollzustandsautomaten haben.	☐	☒
Eine reflexive Assoziation verbindet zwei Klassen.	☐	☒
Kompositionen, wie auch Vererbungen, sind Beziehungen.	☒	☐
Die Multiplizität bei der Vererbung ist immer 0 oder 1.	☐	☒
Die Multiplizität bei einer Komposition ist auf Seite des "Ganzen" höchstens 1.	☒	☐
Wenn bei der Komposition das "Teil" gelöscht wird, wird auch das "Ganze" gelöscht.	☐	☒
Das Design ist abhängig von der Implementations-Technologie.	☒	☐
In UML gibt es Mehrfachvererbung.	☒	☐

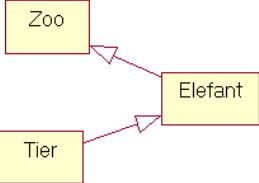
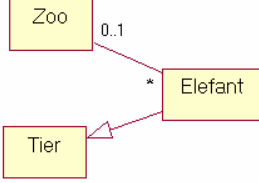
Aufgabe 1 : Klassendiagramme

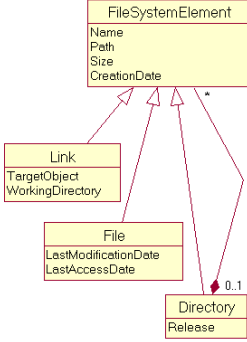
Gegeben sind nachfolgende Klassendiagramme.
Beurteilen Sie, ob diese richtig oder falsch sind.
Wenn ein Klassendiagramm falsch ist:

- erklären Sie, was falsch ist
- zeichnen Sie das richtige Klassendiagramm

Hinweis: die Klassen sind richtig (nicht ändern!). Eventuelle Fehler sind in den Beziehungen.

Klassendiagramm:	Richtig ☐ oder falsch ☒ ?	Korrigiertes Klassendiagramm:
	Was ist falsch ? ... <i>Nicht jedes Wasserfahrzeug ist auch ein Wasserflugzeug.</i> <i>Nicht jedes Flugzeug ist ein Wasserflugzeug.</i>	

Klassendiagramm:	Richtig ☐ oder falsch ☒ ?	Korrigiertes Klassendiagramm:
	Was ist falsch ? ... <i>Ein Elefant ist kein Zoo.</i> <i>Ein Zoo hat Elefanten.</i> <i>Ein Tier ist kein Elefant.</i> <i>Ein Elefant ist ein Tier.</i>	

Klassendiagramm:	Richtig ☒ oder falsch ☐ ?	Korrigiertes Klassendiagramm:
	Was ist falsch ? ... <i>Entspricht dem Entwurfsmuster 'Kompositum'.</i>	