

Klassendiagramm und OOP

Klassendiagramm

Was bedeutet überladen?

Beim überladen einer Methode werden mehrere Methoden gleichen Namens innerhalb einer Klasse definiert, die sich durch ihre Parameterlisten unterscheiden.

-> Methode hat den selben Namen, aber verschiedene Parameter!

Was bedeutet überschreiben?

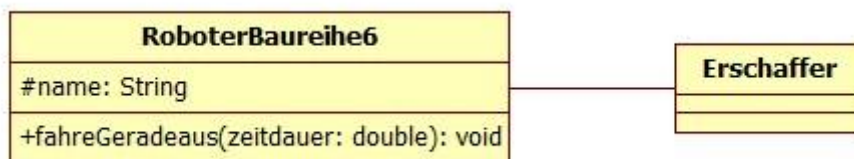
Beim überschreiben einer Methode wird eine Methode der Basisklasse in der abgeleiteten Klasse neu definiert. Die Parameterliste ändert sich dabei nicht. Hier wird die Methode "toString" überschrieben, indem in jeder abgeleiteten Klasse der jeweilige Typ mit angegeben wird.

-> Methode wird in Unterklasse überschrieben!

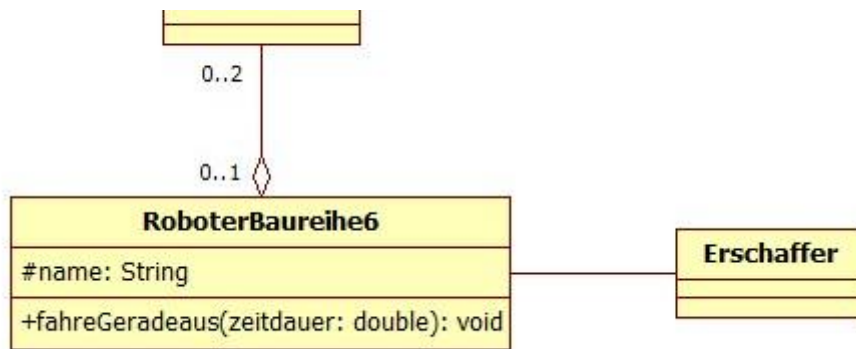
Was bedeuten die Zugriffsparameter

- - : private -> Also nur innerhalb der eigenen Klasse verfügbar
- # : protected -> Sind ausschließlich innerhalb ihrer Klasse und der von ihnen abgeleiteten Klassen verwendbar
- + : public -> Unterliegen keinen Zugriffsbeschränkungen

Was ist eine Assoziation



Was ist eine Aggregation

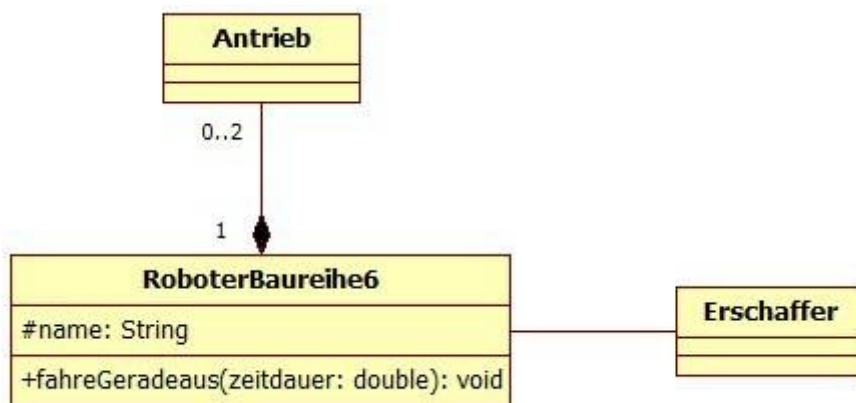


- Wird statt einer

Assoziation verwendet, wenn es sich um eine Teil-Ganzes Beziehung handelt

- Zusammensetzung wird Multiplizität genannt

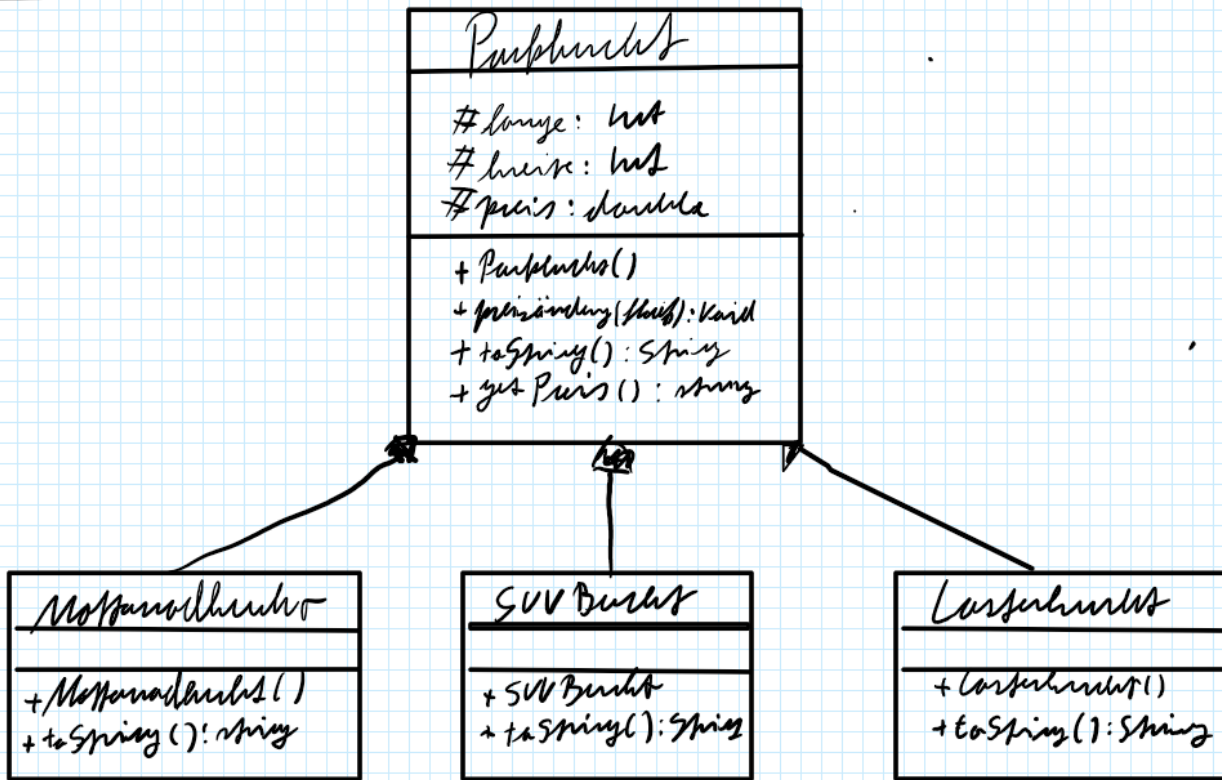
Komposition



- Bei einer Komposition

stehen zwei Klassen nicht nur in einer Teil-Ganzes Beziehung sondern das "Teil" existiert ohne das "Ganze" auch nicht. (Existenzabhängigkeit)

- Die Multiplizität muss auf der Seite des Ganzen zwangsläufig 1 sein



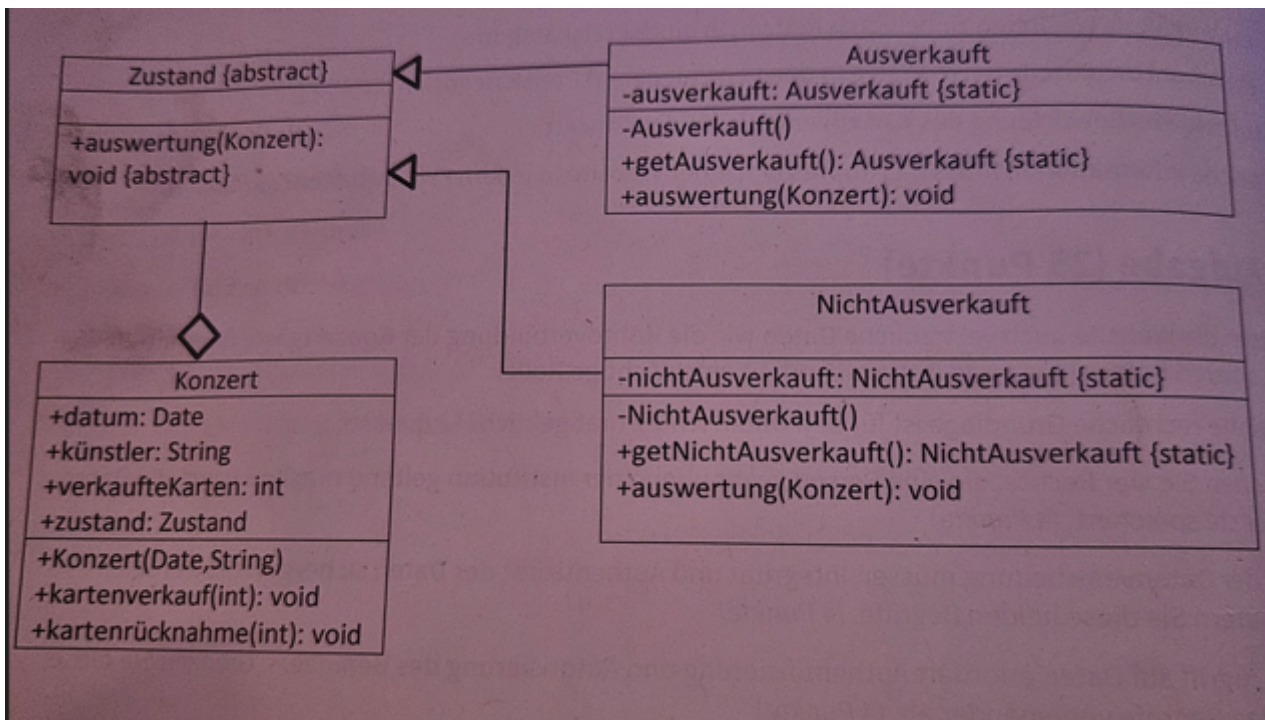
Was bedeutet {abstract} im Klassendiagramm

- Abstrakte Klassen sind nicht vollständig - man kann davon keine Objekte direkt erzeugen
- Sie dienen als Grundlage für andere Klassen (Basisklassen)
- Können abstrakte Methoden enthalten, nur deklariert aber nicht implementiert

Was bedeutet {static} im Klassendiagramm

- Statische Mitglieder (Attribute oder Methoden) gehören zur Klasse, nicht zu einzelnen Objekten
- Man kann sie verwenden um ein Objekt zu erzeugen.

Aufgaben



Die Methode "getAusverkauft" ist folgendermaßen implementiert:

```

public static Ausverkauft getAusverkauft()
{
    falls ausverkauft = null
        ausverkauft = new Ausverkauft()
    rueckgabe ausverkauft
}
  
```

```

public static Ausverkauft getAusverkauft()
{
    falls ausverkauft = null
        ausverkauft = new Ausverkauft()
    rueckgabe ausverkauft
}
  
```

Erläutern Sie hieran die Besonderheiten der Singleton-Klasse "Ausverkauft"

- Die Methode sorgt dafür das nur ein einziges Objekt der Klasse Ausverkauft existiert.
- Wenn noch keins da ist, wird es erstellt.
- Danach wird immer dieses eine Objekt zurückgegeben
- -> Das nennt man Singleton-Pattern

Warum ist diese Methode "static"?

- Die Methode kann man ohne Objekt aufrufen, also es muss kein Objekt instanziiert werden
- Sie gehört zur Klasse selbst, nicht zu einem Objekt

```
Ausverkauft a = Ausverkauft.getAusverkauft();
```

- Das ist wichtig, weil man ja noch gar kein Objekt hat, bevor man diese Methode aufruft.

Implementieren Sie einen Konstruktor der Klasse "Konzert", der u.a. die Zahl der verkauften Karten auf 0 und den Zustand auf "nichtAusverkauft" setzt.

```
public Konzert(datum Date, kuenstler String)
{
    this.datum = datum,
    this.kuenstler = kuenstler,
    this.verkaufteKarten = 0,
    this.Zustand = NichtAusverkauft.getNichtAusverkauft()
}
```

Implementieren Sie die Methode "auswertung" der Klasse "nichtAusverkauft". Diese prüft, ob für das betreffende Konzert mehr als 1000 Karten verkauft werden wurden. In diesem Fall setzt sie den Zustand des Konzerts auf "Ausverkauft".

```
Public void auswertung(Konzert konzert)
{
    wenn konzert.verkaufteKarten > 1000
        konzert.Zustand = Ausverkauft.GetAusverkauft()
}
```

Implementieren Sie die Methode "kartenverkauf", der die Zahl der verkauften Karten erhöht und anschließend eine Auswertung des Zustands auslöst.

```
public void kartenverkauf(int anzahl)
{
    verkaufteKarten = verkaufteKarten + anzahl
    zustand.auswertung(this) // this, weil wir uns in der Klasse "Konzert" befinden.
}
```

Implementieren Sie die folgende Funktion, die die Anzahl ausverkaufter Konzerte bestimmt.

```
public int anzahlAsuverkaufterKonzerte (Konzert[] liste)
{
```

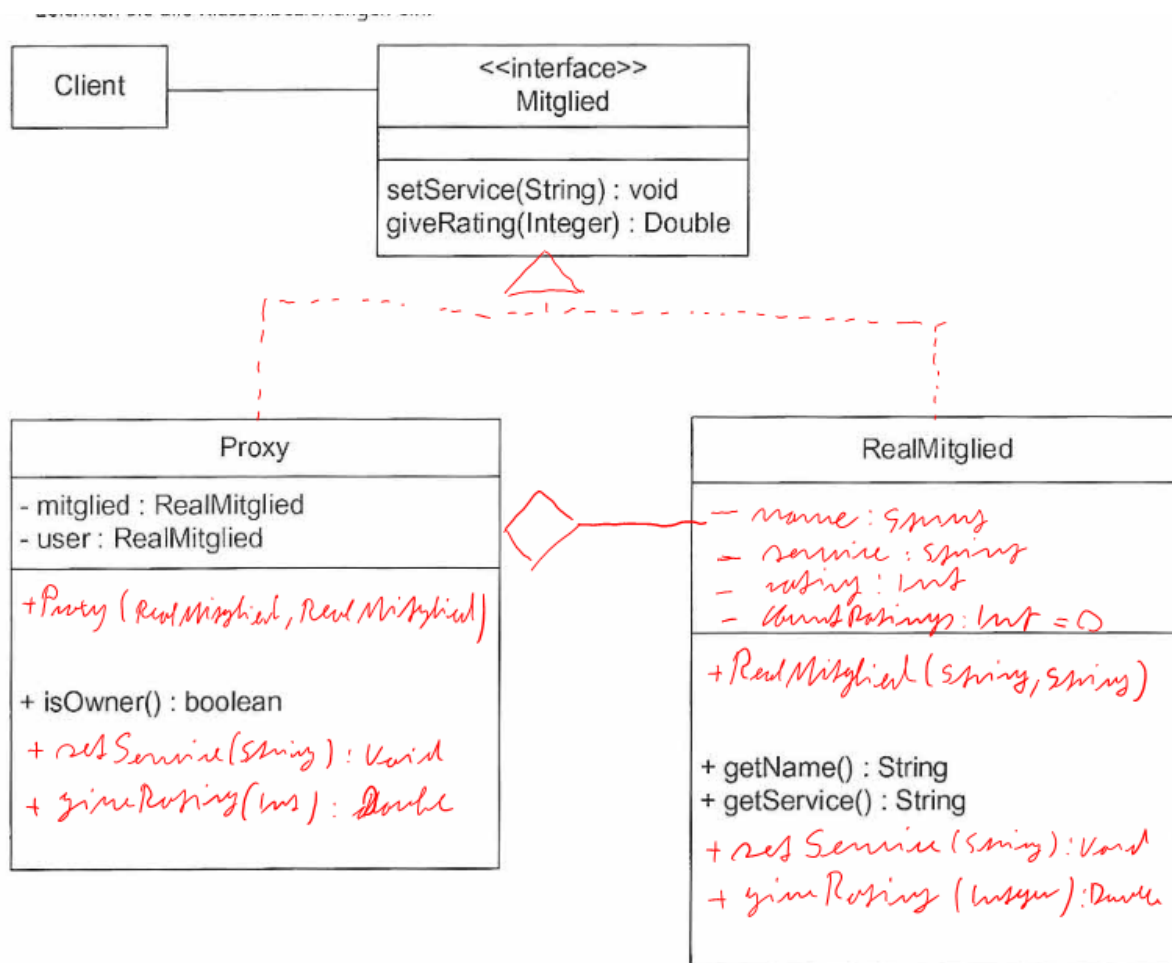
```

zaehler = 0
fuer alle Konzert in liste
falls konzert.zustand hatTyp Ausverkauft
    zaehler = zaehler + 1
}

```

Ergänzen Sie das folgende unvollständige UML-Klassendiagramm nach den folgenden Vorgaben.

- Die Klasse "RealMitglied" soll die nur klasseninternen sichtbaren Instanzvariablen "name", "service", "rating", und "countRatings" beinhalten.
- Der öffentliche Konstruktor der Klasse "RealMitglied" soll zwei Übergabeparameter haben, mit denen "name" und "service" initialisiert werden.
- Der öffentliche Konstruktor der Klasse "Proxy" soll zwei "RealMitglied"-Objekte übergeben bekommen und diese mit diesen mit den Referenzen "mitglied" und "user" verknüpfen.
- Die Klassen "Proxy" und "RealMitglied" sollen beide das Interface Mitglied implementieren.
- Zeichnen Sie alle Klassenbeziehungen ein:



Es ist folgendes Pflichtenheft gegeben:

- ein Artikel hat einen Namen und einen Preis
- Beim Anlegen eines neuen Artikels werden Name und Preis festgelegt
- Der Name eines Artikels kann abgefragt, aber nicht verändert werden.
- Der Preis eines Artikels kann abgefragt und verändert werden.
- Einem "Warenkorb" können verschiedene Artikel in beliebigen Stückzahlen hinzugefügt werden.
- Für den Inhalt des Warenkorbs kann der Gesamtwert berechnet werden.
- Instanzvariablen sind nach außen nicht sichtbar (information hiding/geheimnisprinzip)
- Methoden sollen von überall her aufrufbar sein.

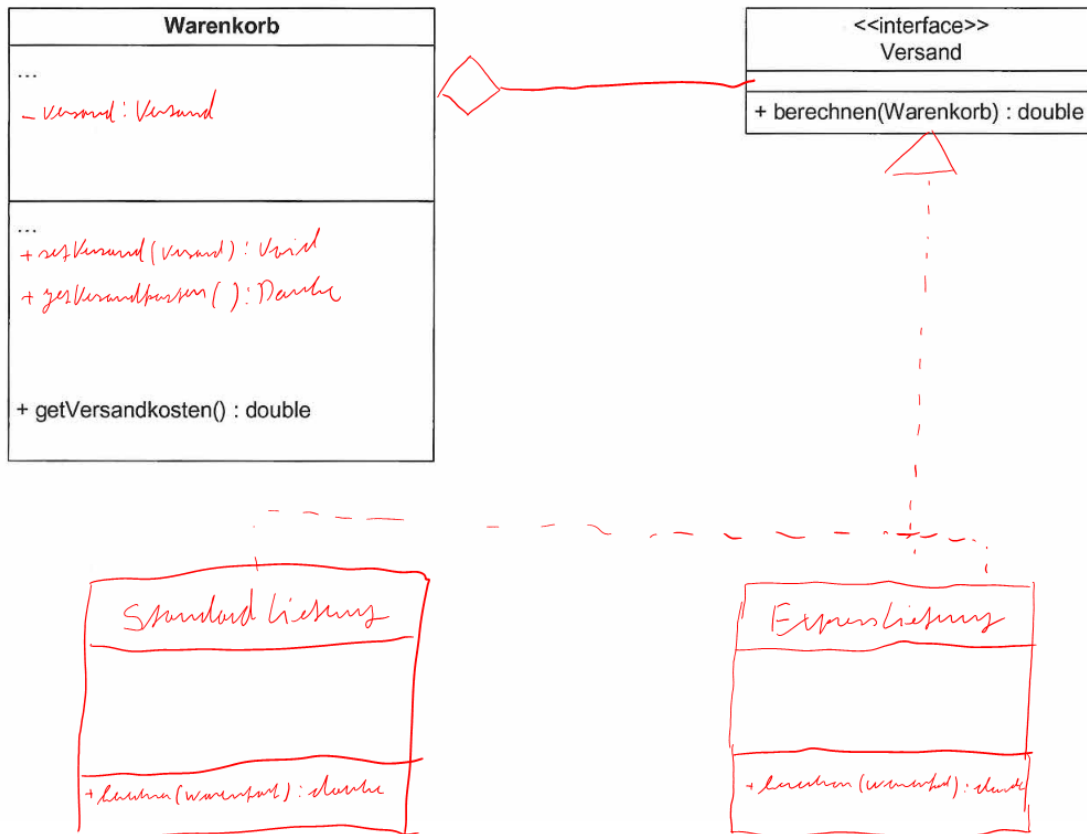
Hinweis: Geben Sie bei Attributen und Methoden die vollständige UML-Syntax (siehe Belegatz) an.

Artikel	Warenkorb
<ul style="list-style-type: none"> - <i>Name</i> : String - <i>Preis</i> : Double 	<ul style="list-style-type: none"> - <i>Artikel</i> : Artikel [] - <i>anzahl</i> : int []
<ul style="list-style-type: none"> + <i>Artikel</i> (<i>name</i>, <i>double</i>) + <i>getName</i> () : String + <i>getPreis</i> () : Double + <i>changePreis</i> (<i>double</i>) : Void 	<ul style="list-style-type: none"> + <i>Warenkorb</i> () + <i>add Artikel</i> (<i>Artikel</i>, <i>int</i>) : void + <i>berechne Gesamt</i> (<i>Double</i>) : Double

Das Pflichtenheft wird um folgende Punkte erweitert:

- Die Versandkosten sollen ebenfalls ermittelt werden.
- Bei Standard-Lieferungen und Express-Lieferungen werden unterschiedliche Berechnungsalgorytmen verwendet.
- Die Berechnungsalgorytmen der Versandkosten wechseln häufig, deshalb sollen sie vom **Warenkorb entkoppelt sein**
- Dazu soll die Klasse "Warenkorb" um eine **Referenz "versand" vom Typ des Interdace "Versand"**, eine Methode "setVersand" und Setzen der Versandart und eine Methode "getVersandkosten" zur Abfrage der Versandkosten erweitert werden.
- -> Ergänzen Sie das Klassendiagramm!

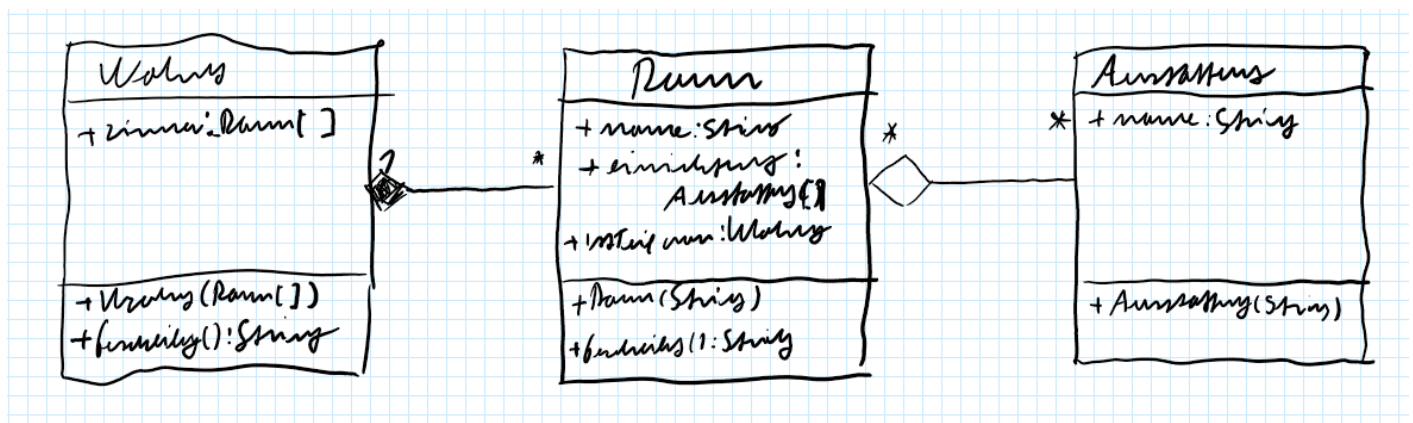
Hinweis: Die in Aufgabe a) angegebenen Attribute und Methoden der Klasse Warenkorb müssen nicht wiederholt werden.



Implementieren Sie in Pseudocode die Methode "getVersandkosten" der Klasse Warenkorb

```

public double getVersandkosten()
{
    double versandkosten = versand.berechnen(this)
    rueckgabe versandkosten
}
  
```



Erläutern Sie den Unterschied zwischen den Relationen "Aggregation" und "Komposition"

Bei einer Aggregation handelt es sich um eine Teil-Ganzes Beziehung, und bei einer Komposition um eine Abhängigkeitsbeziehung, was bedeutet das eine Klasse nicht ohne die Andere existieren kann. Beide Beziehungen beschreiben Abhängigkeiten voneinander und enthalten Multiplizitäten.

Die Relation "Raum" enthält Ausstattung entspricht einer m:n-Relation in einem Entity-Relationship-Diagramm. Erläutern Sie die unterschiedliche Umsetzung einer m:n Relation in einer Datenbank gegenüber der entsprechenden Relation in einem objektorientierten Modell. (Hinweis: Beziehen Sie sich in ihrer Erläuterung auf die 1. Normalform)

Das ein Raum mehrere Ausstattungsmerkmale haben kann und ein Ausstattungsmerkmal bei mehreren Räumen auftreten kann, wird in einem Klassendiagramm durch listenförmige Attribute dargestellt.

Die erste Normalform verbietet listenförmige Attribute in einer Datenbank. Stattdessen werden m:n-Relationen durch eine Zwischentabelle dargestellt, die die Schlüsselattribute beider Relationspartner miteinander kombiniert.

Die Umsetzung der Relation "Wohnung enthält Raum" ist redundant gestaltet

- **Erläutern Sie dies anhand des Klassendiagramms**
 - Raum enthält das Attribut istTeilVon Wohnung. Das ist eine Dopplung.
- **Nennen Sie je ein Vorteil und einen Nachteil dieser redundanten Speicherung.**
 - Vorteil: Sie ermöglicht eine schnelle Zuordnung zu Wohnung
 - Nachteil: Sie verbraucht Speicherplatz und macht das Modell komplexer

Geben Sie den Quellcode des Konstruktors der Klasse "Wohnung" an, der den Wert des Attributs "zimmer" setzt und für jeden Raum angibt, dass er Teil dieser Wohnung ist.

```
public Wohnung (Raum[] raeume)
{
    zimmer = raeume
    foreach(Raum r in Zimmer)
    {
        r.istTeilvon(this);
    }
}
```

Geben Sie den Quellcode der Methode "beschreibung" der Klasse "Wohnung" an. Die Methode soll für alle Räume der Wohnung deren Beschreibung liefern.

```
public string beschreibung()
{
    string ergebnis = "";
```

```
foreach(Raum r in Zimmer)
{
    ergebnis += r.beschreibung();
}
return ergebnis;
}
```

Die in Aufgabe 2 genannte Klasse "Raum" verfügt über folgende Methode

```
public string beschreibung()
{
    string ausgabe = name + " verfügt über folgende Ausstattungsmerkmale: ";
    for (int i = 0; i <= einrichtung.size; i++)
    {
        ausgabe += einrichtung[i].name + ", ";
    }
    return ausgabe;
}
```

Beim Aufruf dieser Methode erscheint zunächst die Meldung "NullPointerException". Nach entsprechender Korrektur des Programms erscheint die Meldung "ArrayOutOfBoundsException".

aa) Erklären Sie den Begriff Exception im Allgemeinen

Eine Exception ist ein Objekt welches beim Debuggen Aufschluss über die Art des Fehlers liefert

ab) Um welche speziellen Exceptions handelt es sich hier?

NullPointerException gibt an, wenn eine undeklarierte Variable vor der Deklaration verwendet wird

ArrayOutOfBoundsException wird ausgegeben wenn der falsche Index übergeben wird, der die Gesamtgröße des Arrays übersteigt.

ac) Wie kann man ein Programm erweitern, um eine Exception abzufangen?

Mit Try Catch Blöcken kann eine Exeption zur Laufzeit erkannt und ausgegeben werden.

ad) Korrigieren Sie das Programm so, dass beide Exceptions nicht mehr auftreten

```

public string beschreibung()
{
    string ausgabe = this.name + " verfügt über folgende Ausstattungsmerkmale: ";

    for(int i = 0; i < einrichtung.size; i++)
    {
        ausgabe += einrichtung[i].name + ", ";
    }

    return ausgabe;
}

```

b) Sollte das Attribut "name" der Klasse "Raum" keinen Wert haben, führt das Programm zu einem nicht zufriedenstellenden Ergebnis. Formulieren Sie den Code des Konstruktors der Klasse "Raum" so, dass dieses Problem nicht entsteht

```

Public Raum(name string)
{
    this.name = "Kein Name";
}

```

c) Nennen Sie einen weiteren Grenzfall, der beim Testen des Programms zu beachten ist und der zu einem nicht zufriedenstellenden Ergebnis führt

Es sollte in Betracht gezogen werden, dass Raum keinen Namen haben könnte.

Die Kfz-Kennzeichen werden in einem Array gespeichert. Zur Suche nach einem bestimmten Kennzeichen soll die folgende Funktion verwendet werden:

```

int suchen(String suchwert, String[] parkliste, int startposition, int endposition)
{
    int mitte=(startposition+endposition)/2;
    falls suchwert < parkliste[mitte]
        rueckgabe suchen(suchwer, parkliste, startposition, mitte+1)
    falls suchwert > parliste[mitte]
        rueckgabe suchen(suchwert, parkliste, mitte+1, endposition)
}

```

Schreiben Sie in Pseudocode einen Algorithmus, der ein Arrayelement an der Stelle i löscht. Die länge des Arrays beträgt parkliste.laenge

Weitere OOP Begrifflichkeiten

- **Garbage Collector** -> Räumt nicht mehr verwendete Objekte aus dem Speicher.
- **Getter und Setter** -> Dienen zum Zugriff auf private Attribute einer Klasse und ermöglichen Kontrolle beim Lesen und Ändern der Daten
- **Konstruktor** -> Wird aufgerufen wenn ein Objekt neu erzeugt wird. Kann Parameter annehmen, um das Objekt gleich richtig zu initialisieren
- **this** -> Verweist auf das aktuelle Objekt, innerhalb der Klasse
- **Polymorphie** -> Objekte können unterschiedliches Verhalten Zeigen, obwohl sie denselben Typ haben.
- **interface** -> Eine Schnittstelle, die vorgibt was eine Klasse tun soll, Implementierung erfolgt in Klassen
- **Abstrakte Klasse** -> Kann nicht direkt instanziiert werden und dient als gemeinsame Oberklasse mit fertiger Funktionalität
- **Kapselung** -> Daten werden versteckt (private) und nur über Getter/Setter zugänglich gemacht.

Revision #5

Created 22 April 2025 16:43:46 by Admin

Updated 5 May 2025 15:05:32 by Admin