



# Informatik I

Prof. Dr. Christian Pape

Kapitel 7

Kommentierung Objekt-  
orientierter Programme



# Inhalt

- Generell
- Beispiel
  - Aufgabe 3, Softwarelabor
  - Klassen
  - Konstruktor
  - Methoden
  - Attribute



# Generell (1/5)

## Aus einem Praktikumsbericht:

- Ein Pythonprogramm sollte erweitert werden
- „Der Programmierstil im Programm war sehr hackerisch – es gab so gut wie keine Kommentare, die Variablennamen waren alle unheimlich abgekürzt, oftmals nur aus 1-3 Buchstaben, Methodennamen waren selten länger als ein zwei abgekürzte Wörter. Das war ein richtig schlechter Programmierstil. Nur ein Mensch verstand das Programm – sein Autor, der seit Monate nicht mehr daran arbeitete.“
- „Nach dem die Demo vorbei war, haben die Teamkollegen beschlossen diese Anwendung in C++ zu reimplementieren. Ich war sehr froh, dass ich dieses Programm vergessen konnte.“



# Generell (1/5)

- Kommentare im Folgenden:
  - Javadoc Kommentare für public Klasse/Meth.  

```
/**  
 * Dies ist ein Javadoc Kommentar  
 */
```
  - Werden von Entwickler benötigt, die
    - keinen Zugriff auf Quelltext haben und/oder
    - sich Überblick über Klassen verschaffen wollen
  - Details Javadoc
    - siehe vergangene Rechnerübung(en)
- Keine Implementierungskommentare
  - Dienen Entwicklern für besseres Verständnis der Implementierungen im Quelltext  

```
/* ... */, // ...
```



# Generell (2/5)

- **Verwendete Sprache (Deutsch, Englisch, ...) nie mischen**
  - Kommentare und Namen für Klassen/Methoden in der selben Sprache
  - Ausnahmen sind übliche Präfixe wie get, set (add, remove)
- **Die Selbe Sprache für alle Teammitglieder wählen**
  - wie in Analyse/Entwurf verwendet
  - die der Kunde spricht (Anforderungen)
  - Muttersprache wählen
  - Sprache, die im Durchschnitt alle am besten beherrschen
- **Bei Vererbung / abstrakten Klassen oder Verwendung anderssprachiger Klassen (bei Beziehungen) Sprache wie verwendete (Ober)klasse wählen**
- **Übersetzungen kosten Zeit und sind meist irreführend**
  - Branchenspezifische Fachbegriffe (Leistungsrechnung, Leistungsreglement, Medizinalpartner) meist nicht in Wörterbüchern



# Generell (3/5)

- Umfang Kommentar
  - So *kurz* wie möglich (Entwickler wollen keine Romane lesen)
  - So lang und *spezifisch* wie nötig (Entwickler wollen alle für Benutzung der Klasse/Methode notwendigen Informationen)
  - Knapp und spezifisch muss ausgeglichen sein
- Bei öffentlichen Methoden/Klassen
  - *Nie* beschreiben, wie etwas implementiert wurde (Implementierung kann sich ändern)
  - *Nie* beschreiben, wo Objekte der Klasse verwendet werden (Darauf hat man als Entwickler der Klasse gar keinen Einfluss)
- Kurz:
  - **Was kann weglassen werden, ohne das spezifische Information verloren geht?**
- Spezifisch:
  - **Was kann hinzugefügt werde, so dass der Kommentar noch aussagekräftiger wird?**



# Generell (4/5)

- Kurz
  - Keine Wiederholung von Information
  - Vermeide inhaltsleere Verben oder Phrasen
- Spezifisch
  - Bezugnehmen auf Entwurfsentscheidungen der Klasse
  - Entwurfsentscheidungen haben mögliche Menge Objekte *eingeschränkt*



# Inhalt

- Generell
- Beispiel
  - Aufgabe 3, Softwarelabor
  - Klassen
  - Konstruktor
  - Methoden
  - Attribute

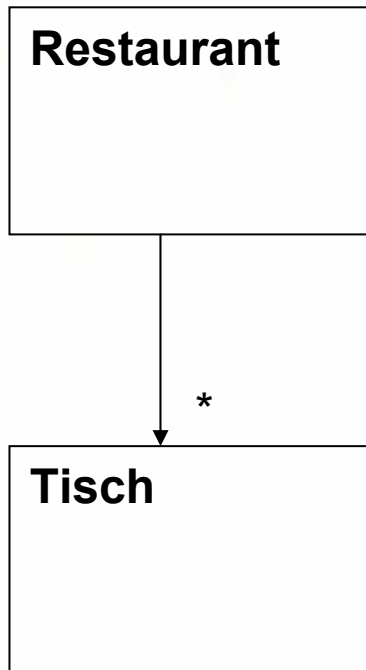


# Beispiel Restaurantsimulation (1/3)

- Aufgabe 3 aus Softwarelabor 2. Semester
- „Die Sitzplatzbelegung in einem Restaurant soll simuliert werden. Besucher betreten in Gruppen das Restaurant, suchen sich einen Platz an einem Tisch, essen und verlassen das Restaurant wieder.“
- „Verhalten der Besuchergruppen
  1. Restaurant betreten
  2. Platzsuche gemäß unten aufgeführter Strategie
  3. Aufenthalt im Restaurant
  4. Restaurant verlassen“



# Beispiel Restaurantsimulation (2/2)



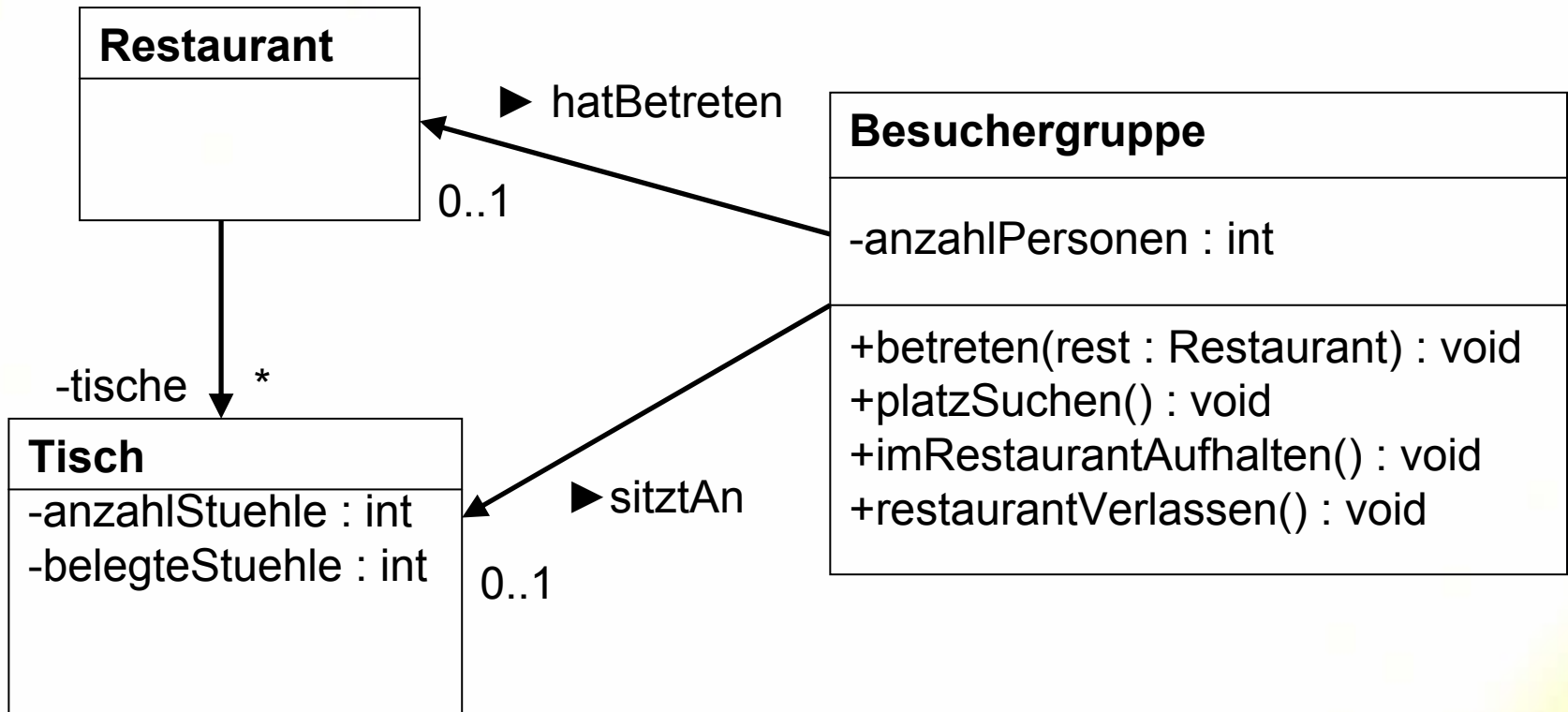
„bestehen aus einer bestimmten Anzahl Personen“



„Jeder Tisch hat eine bestimmte Anzahl von Stühlen. Ein Stuhl ist frei oder belegt.“

# Beispiel Restaurantsimulation (3/3)

- Verfeinerung des vorgegebenen Entwurfs





# Inhalt

- Generell
- Beispiel
  - Aufgabe 3, Softwarelabor
  - **Klassen**
  - Methoden
  - Konstruktor
  - Attribute

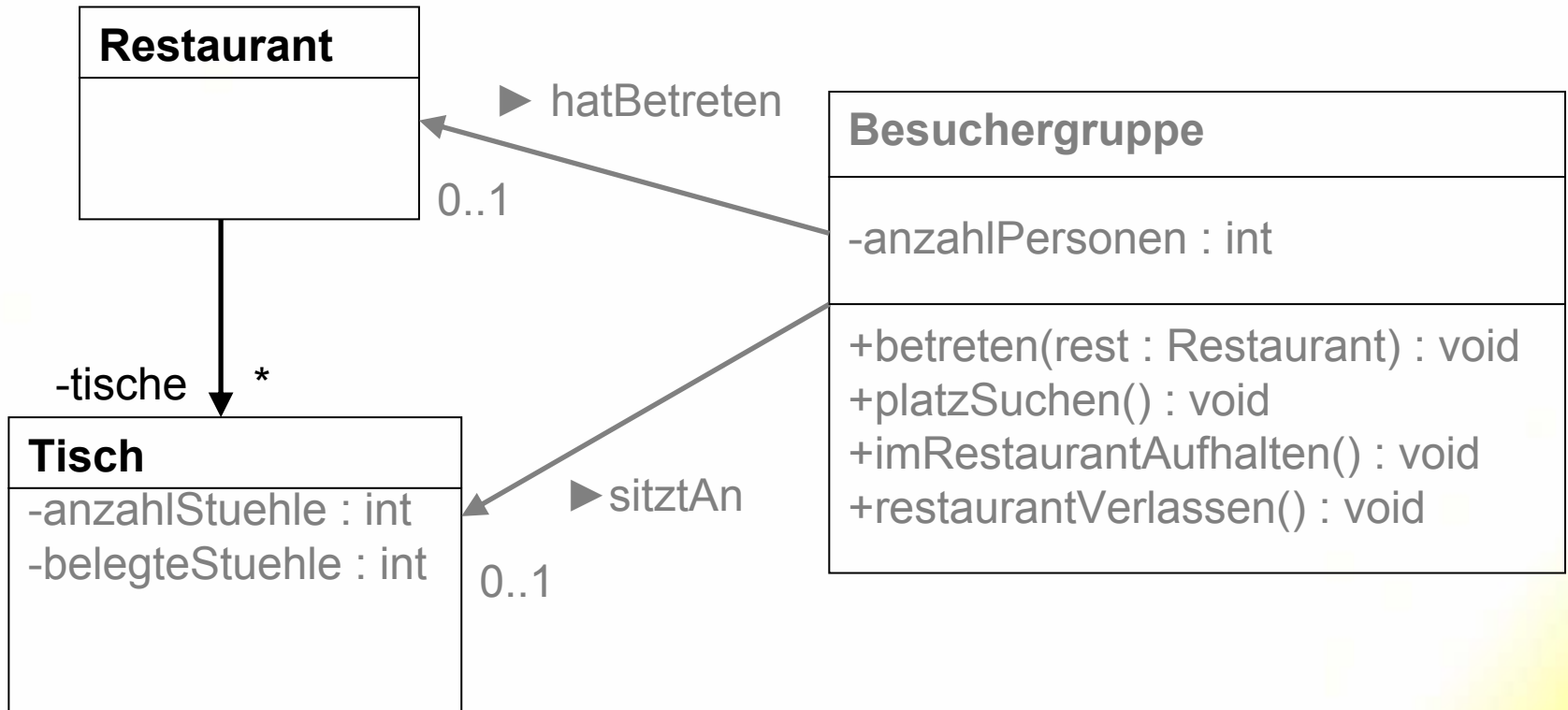


# Klassenkomentar (1/12)

- Objekt-Orientierung (Wdh.)
  - Klassen beschreiben Mengen von Objekten
  - Objekte haben Eigenschaften / Beziehungen und Verhalten
- Klassenkomentar muss folgende Fragen beantworten
  - „Was ist die *Verantwortung* dieser Klasse?“ (Entwurfsziel dokumentieren)
  - „Welches Verhalten dieser Klasse wird zu Objekte anderer Klasse *delegiert*?“
  - *Muss* auf wichtigsten Beziehungen und Attribute verweisen
  - *Kann* abstrakte Beschreibung des Verhaltens enthalten

# Klassenkommmentar (2/12)

- Restaurant
  - Nur eine Beziehung (Richtung entscheidend)
  - Kein nennenswertes Verhalten
  - Keine Attribute



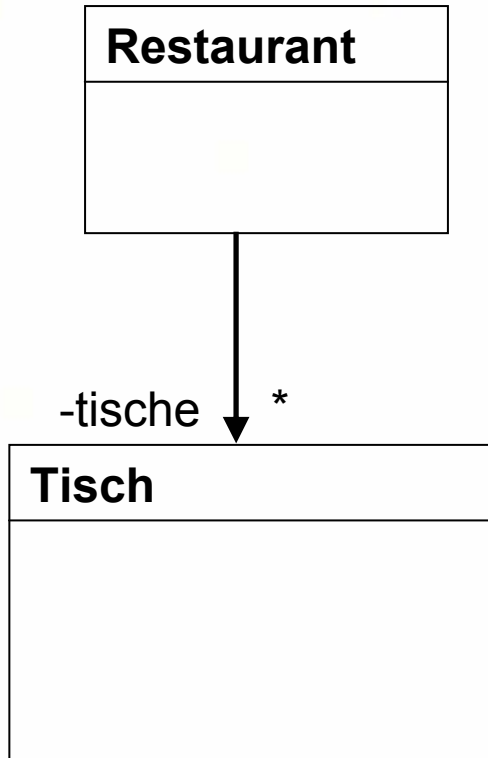
# Klassenkomentar (3/12)

- Nur Beziehungen relevant die von Restaurant aus navigiert werden können
- Attribute / Beziehungen navigierter Klassen nicht relevant

Unbestimmten Artikel verwenden

„Objekt“ benennen

Beziehung mit Multiplizität



```

/**
 * Ein Restaurant mit
 * mehreren Tischen.
 *
 * `@see Tisch
 */
public class Restaurant {
    ...
}
  
```

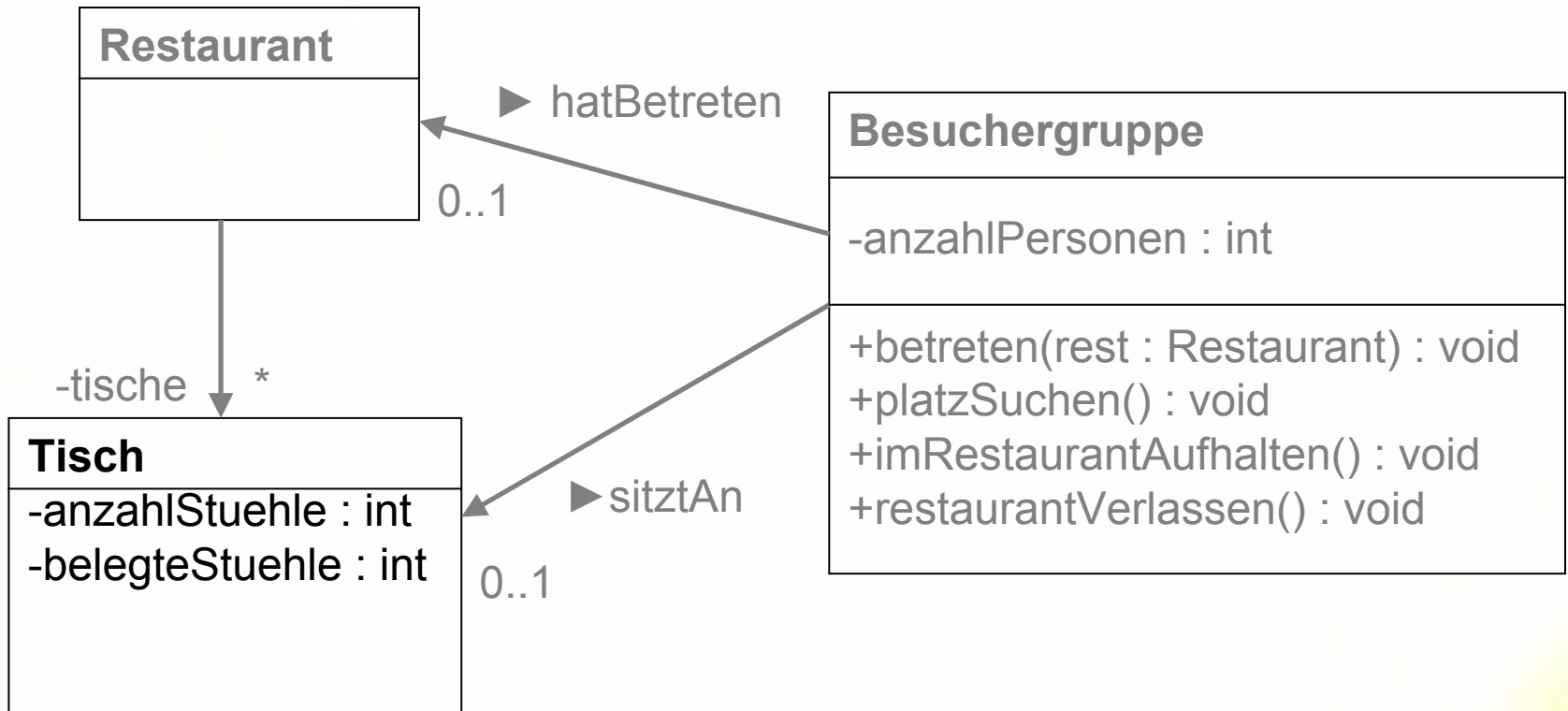


# Klassenkomentar (4/12)

- Schlechte Kommentare, da völlig *unspezifisch*
- „Modelliert ein Restaurant“  
(Modelliert kann gestrichen werden: Restaurant ist eine Klasse, Klassen modellieren Mengen von Objekten, in diesem Fall Restaurants“)
- „Eine Klasse, die ein Restaurant simuliert.“
- „Eine Klasse, mit der Objekte erstellt werden können, die ein bestimmtes Restaurant simulieren“  
(Das Restaurant eine Klasse ist, steht schon an entsprechender Stelle im Javadoc; „simulieren“, „bestimmtes“ sind unspezifisch, Restaurant simuliert gar nichts; dass von eine Klasse Objekte erstellt werden können ist normal)
- „Verwaltet Tische“  
(Verwaltet drückt nichts spezifisches für ein Restaurant aus, Bezug zu Objekt fehlt)

# Klassenkomentar (5/12)

- Entwurf im ersten Satz ausdrücken
  - Keine Beziehungen von Tisch
  - Attribute erwähnen



# Klassenkommentar (6/12)

## Tisch

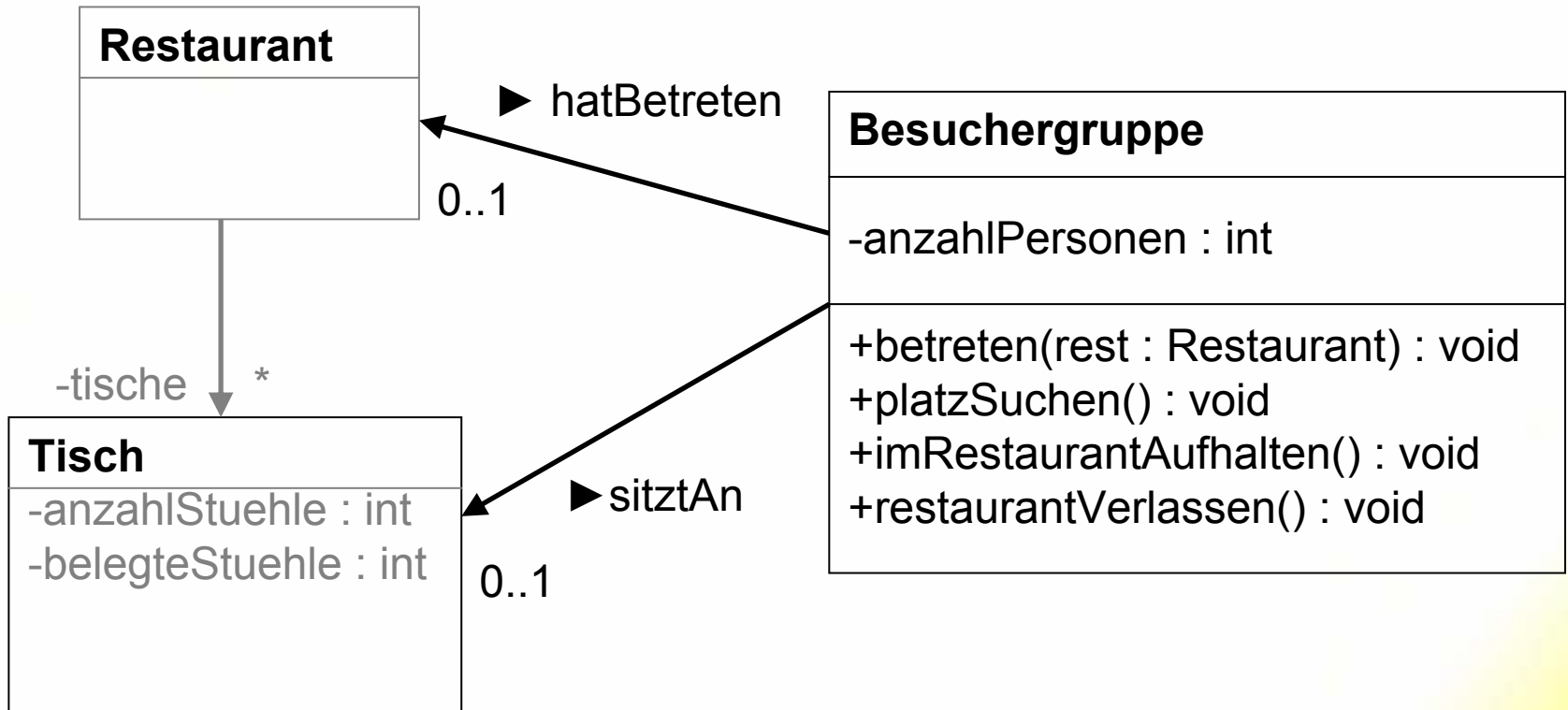
-anzahlStuehle : int  
-belegteStuehle : int

```
/**  
 * Ein Tisch mit einer Anzahl Stühlen,  
 * die teilweise belegt sein können.  
 */  
public class Tisch {  
    private int anzahlStuehle;  
    private int belegteStuehle;  
    ...  
}
```

- *Nicht* beschreiben
  - wo Tisch verwendet wird oder
  - was mit einem Tisch gemacht wird
- (ist für Verständnis der Klasse unnötig)

# Klassenkommmentar (7/12)

- Nur wichtigsten Beziehungen und wesentliches Verhalten beschreiben





# Klassenkommentar (8/12)

```
/**  
 * Eine Besuchergruppe bestehend aus einer  
 * Anzahl von Personen. Besuchergruppe ist  
 * verantwortlich für die Suche nach einem  
 * freien Tisch in einem Restaurant.  
 *  
 * @see Tisch, Restaurant  
 */  
public class Besuchergruppe {  
    private int anzahlPersonen;  
    private Tisch tisch;  
    private Restaurant restaurant;  
    ...  
}
```

# Klassenkomentar (9/12)

- Ggf. Codebeispiel in `<pre>`-Tag geben

```
/**
 * Eine Besuchergruppe...
 * <pre>
 * Restaurant restaurant = // Restaurant erzeugen
 * Besuchergruppe besuchergruppe = new Besuchergruppe(5);
 * besuchergruppe.betreten(restaurant);
 * besuchergruppe.platzSuchen();
 * besuchergruppe.imRestaurantAufhalten();
 * besuchergruppe.restaurantVerlassen();
 * </pre>
 *
 * @see Tisch Restaurant
 */
public class Besuchergruppe {
...
}
```

# Klassenkommentar (10/12)



de.fhka.fbi.softlab - Mozilla Firefox

File Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe

file:///D:/eclipse/restaurant/doc/index.ht Go

Vorlesung Informatik 1 JBoss JMX Managemen...

**All Classes**

- [Besucherguppe](#)
- [Restaurant](#)
- [Tisch](#)

**Package Class Use Tree Deprecated Index Help**

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

**Package de.fhka.fbi.softlab**

**Class Summary**

<a href="#">Besucherguppe</a>	Eine Besucherguppe bestehend aus einer Anzahl von Personen.
<a href="#">Restaurant</a>	Ein Restaurant mit mehreren Tischen.
<a href="#">Tisch</a>	Ein Tisch mit einer Anzahl Stühlen, die teilweise belegt sein können.

Fertig

# Klassenkommmentar (11/12)



The screenshot shows a Mozilla Firefox browser window titled "Besuchergruppe - Mozilla Firefox". The address bar contains the file path: `file:///d:/eclipse/restaurant/doc/index.ht`. The browser displays a Java class comment for `Besuchergruppe`. The comment includes the class signature, inheritance, a description, an example of usage, and author information.

**All Classes**

- [Besuchergruppe](#)
- [Restaurant](#)
- [Tisch](#)

```
public class Besuchergruppe
extends java.lang.Object
```

Eine Besuchergruppe bestehend aus einer Anzahl von Personen. Besuchergruppe ist verantwortlich für die Suche nach einem freien Tisch in einem Restaurant.

Beispiel für Verwendung:

```
Restaurant restaurant = // Restaurant erzeugen
Besuchergruppe besuchergruppe = new Besuchergruppe(5);

besuchergruppe.betreten(restaurant);
besuchergruppe.platzSuchen();
besuchergruppe.imRestaurantAufhalten();
besuchergruppe.restaurantVerlassen();
```

**Author:**  
pape

**See Also:**  
[Restaurant](#)

Fertig



# Klassenkommentar (12/12)

- Dokumentation der Beziehungen und Eigenschaften oft nicht ausreichend bei komplexeren Klassen oder manchmal „unnatürlich“
- In diesem Fall abstrakte „Verantwortung“ einer Klasse (eigentlich deren Objekte) beschreiben
- Class Responsibility Collaboration Cards (CRC Cards)
  - Class: Die Klasse, um die es geht
  - Responsibility: Was für Aufgaben können mit der Klasse erledigt werden? (nicht: wie wird sie verwendet)
  - Collaboration: An welche wichtigen anderen Klassen werden Teile der Aufgaben delegiert, um die Aufgaben der Klasse zu erledigen?

```
/**  
 * Besuchergruppe ist verantwortlich für das Betreten  
 * eines Restaurants und die Suche nach einem  
 * freien Platzes. <p>  
 * Besuchergruppe delegiert das Besetzen und Freigeben  
 * eines Platzes and Tisch.  
 */
```



# Inhalt

- Generell
- Beispiel
  - Aufgabe 3, Softwarelabor
  - Klassen
  - **Methoden**
  - Konstruktor
  - Attribute



# Methodenkommentar (1/5)

- Objekt-Orientierung (Wdh.)
  - Methoden implementieren Verhalten eines Objekt
- Methodenkommentar muss folgende Fragen beantworten
  - „Was macht die Methode?“
  - „Für was werden die Parameter benötigt?“
  - „Welcher Wert wird zurückgegeben?“
- Satz möglichst mit einem Verb (der Methode) beginnen (geht immer bei get/set-Methoden)
- Den ersten Satz mit Bezug zu Objekt beenden.



# Methodenkommentar (2/5)

<b>Besuchergruppe</b>
-anzahlPersonen : int -tisch : Tisch
+betreten(rest : Restaurant) : void +platzSuchen() : void +imRestaurantAufhalten() : void +restaurantVerlassen() : void +getTisch() : Tisch +setTisch(tisch : Tisch) : void

```

/**
 * Gibt den Tisch, an dem
 * diese Besuchergruppe Platz
 * genommen hat, zurück. Gibt
 * null zurück, falls diese
 * Besuchergruppe an keinen Tisch
 * sitzt.
 *
 * @return Tisch, an dem
 *         diese Besuchergruppe Platz
 *         genommen hat
 */
public Tisch getTisch() {
    return tisch;
}

```

Phrase wiederholen

Immer *alle* **Getter Methoden**  
*vollständig* kommentieren  
(Benutzer könnte sonst sich  
falsche Vorstellungen der Methoden  
machen oder ist unsicher)

Halbsatz möglichst mit Typ anfangen



# Methodenkommentar (3/5)

## Besuchergruppe

-anzahlPersonen : int  
-tisch : Tisch

+betreten(rest : Restaurant) : void  
+platzSuchen() : void  
+imRestaurantAufhalten() : void  
+restaurantVerlassen() : void  
+getTisch() : Tisch  
+setTisch(tisch : Tisch) : void

```
/**
 * Setzt den Tisch, an dem diese
 * Besuchergruppe sitzen soll, auf
 * den neuen tisch.
 *
 * @param tisch Tisch, an dem
 *           diese Besuchergruppe
 *           Platz nimmt
 */
public void setTisch(Tisch tisch) {
    this.tisch = tisch;
}
```

Immer *alle* **Setter Methoden**  
*vollständig* kommentieren

Jeden Parameter mit *mindestens*  
einem Halbsatz kommentieren

Direkten Bezug zu Parameter nehmen  
mit HTML-Code Tags.



# Methodenkommentar (4/5)

## Besuchergruppe

-anzahlPersonen : int  
-tisch : Tisch

+betreten(rest : Restaurant) : void  
+platzSuchen() : void  
+imRestaurantAufhalten() : void  
+restaurantVerlassen() : void  
+getTisch() : Tisch  
+setTisch(tisch : Tisch) : void

Kommentare zu anderen Methoden  
möglichst immer mit dem Verb des  
Methodennamens beginnen.

```
/**
 * Betritt das gegebene Restaurant,
 * bevor diese Besuchergruppe
 * dort eine Platz suchen kann
 * ({@link #platzSuchen()}).
 *
 * @param rest Restaurant, das
 *           von dieser Besucher-
 *           gruppe betreten wird
 */
public void betreten(Restaurant
                    rest) {
    this.rest = rest;
}
```



# Methodenkommentar (5/5)

<b>Besuchergruppe</b>
-anzahlPersonen : int -tisch : Tisch
+betreten(rest : Restaurant) : void +platzSuchen() : void +imRestaurantAufhalten() : void +restaurantVerlassen() : void +getTisch() : Tisch +setTisch(tisch : Tisch) : void

```
/**
 * Sucht einen freien Tisch im
 * zuvor betretenen Restaurant und
 * belegt ihn mit dieser Besucher-
 * gruppe. Falls ein Tisch mit ...
 */
public void platzSuchen() {
    ...
}
```

Gegebenenfalls auf Verhalten Bezug nehmen, das vorher ausgeführt werden sollte.

Platzsuchstrategie beschreiben, ohne auf dessen Implementierung einzugehen.



# Inhalt

- Generell
- Beispiel
  - Aufgabe 3, Softwarelabor
  - Klassen
  - Methoden
  - **Konstruktor**
  - Attribute



# Konstruktorkommentar (1/2)

- Konstruktor der Klasse X erzeugt ein neues Objekt
  - Kommentar mit „Erzeugt ein neues X mit ...“ beginnen
  - Beschreiben, was für ein Objekt erzeugt wird: welche Werte haben die wichtigsten Attribute und Beziehungen
  - Ansonsten wie bei Methoden

# Konstruktorkommentar (2/3)

<b>Besuchergruppe</b>
-anzahlPersonen : int -tisch : Tisch +MAX_ANZAHL_PERSONEN:int
+Besuchergruppe(anzahl : int) +betreten(rest : Restaurant) : void +platzSuchen() : void

```

/**
 * Erzeugt eine neue Besuchergruppe
 * mit der gegebenen Anzahl Personen.
 * Falls die gegebene Anzahl Personen
 * nicht positiv oder größer als
 * <code>MAX_ANZAHL_PERSONEN</code>
 * ist, so wird die Anzahl Personen
 * auf den maximalen Wert gesetzt.
 * Die neue Besuchergruppe befindet
 * sich an keinem Tisch und in keinem
 * Restaurant.
 *
 * @param anzahl ...
 */
public Besuchergruppe(int anzahl) {
    if (anzahl <= 0 || anzahl >=
        MAX_ANZAHL_PERSONEN) {
        anzahl = MAX_ANZAHL_PERSONEN;
    }
    this.anzahl = anzahl;
}

```



# Inhalt

- Generell
- Beispiel
  - Aufgabe 3, Softwarelabor
  - Klassen
  - Methoden
  - Konstruktor
  - **Attribute**



# Attributkommentar (1/2)

- Öffentliche Attribute (normalerweise nur Konstanten) mit Javadoc Kommentar beschreiben
  - Halbsatz reicht
  - Bezug zu Objekt:
    - „dieser Besuchergruppe“ bei Objekt-Attributen
    - „einer Besuchergruppe“ oder ähnliches bei Konstanten (static)
    - Bezug zur Klassen (Menge von Objekten) bei Klassenvariablen (static ohne final) „...für alle Besuchergruppen“
  - Kommentar möglichst mit Substantiven des Bezeichners beginnen



# Attribute (2/2)

Besuchergruppe
-anzahlPersonen : int
-tisch : Tisch
<u>+MAX_ANZAHL_PERSONEN : int</u>
<u>-fortlaufendeNummer : int</u>

Versuchen, spezifischer als  
Attributname zu sein, z.B.  
Einschränkungen der möglichen  
Werte beschreiben

```
/**
 * Positive Anzahl von Personen
 * dieser Besuchergruppe. Der
 * Wert darf nicht größer als
 * MAX_ANZAHL... sein.
 */
private int anzahlPersonen;
```

```
/**
 * maximale Anzahl Personen einer
 * Besuchergruppe
 */
public static final int MAX...
```

```
/**
 * Fortlaufende Nummer, für alle Besuchergruppen. Sie wird
 * nach jeder neu erzeugten Besuchergruppe um Eins erhöht.
 */
```

```
private static int fortlaufendeNummer = 0;
```