

Lektion 13

Packages

Implizite Vererbung (toString, equals)

Relationen

varargs

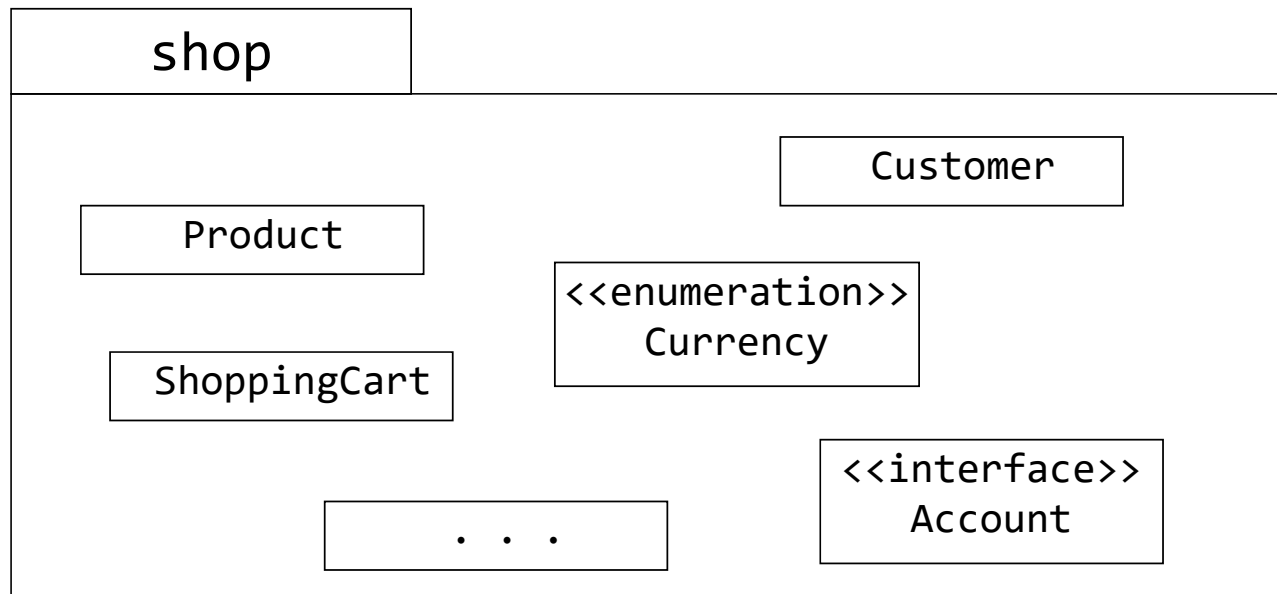
Packages

Größer werdende Programme werden schnell
unübersichtlich.

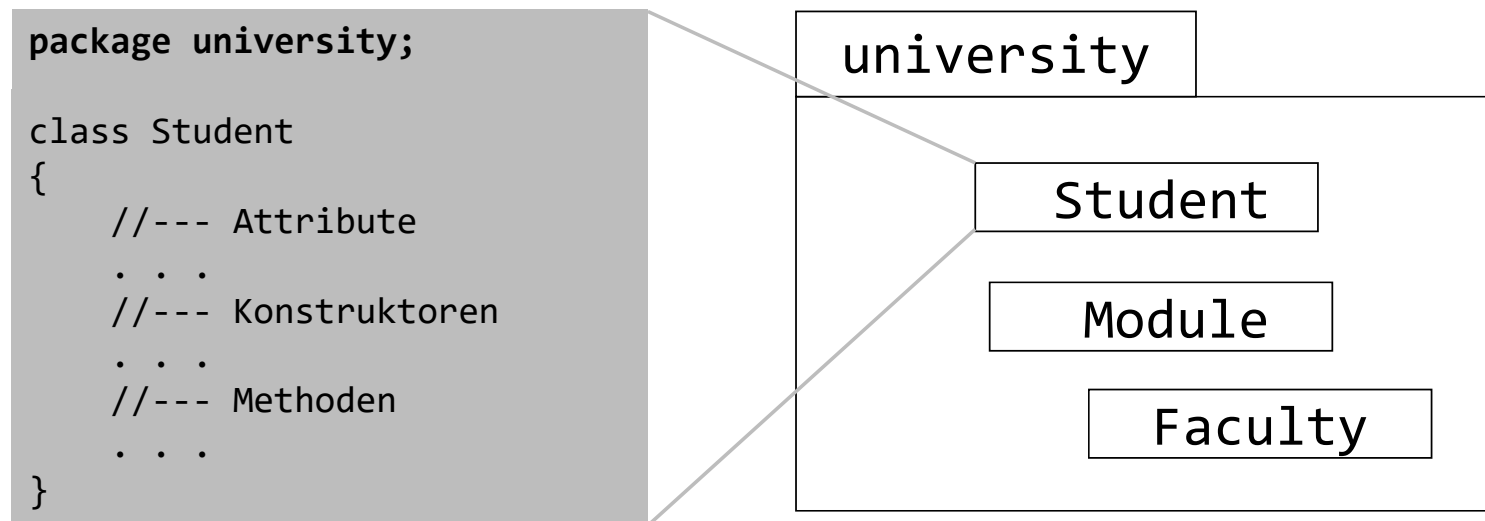
Es droht die doppelte Verwendung von Klassennamen – vor
allem wenn Klassen in anderen Programmen verwendet
werden sollen.

➤ Lösung: Einführung von unterschiedlichen Namensräumen,
sog. Packages

Logisch zusammengehörige Klassen werden in einem **Package** zusammengefasst.



Mit der **package-Deklaration** wird eine Klasse genau einem Package zugeordnet.

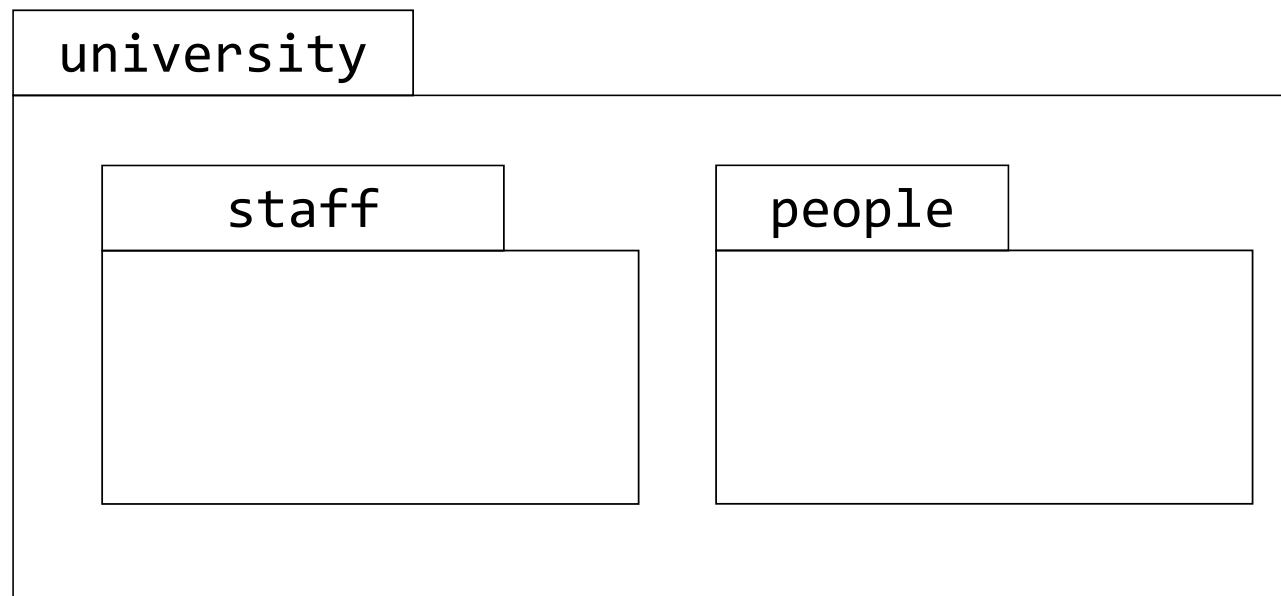


package Namen werden komplett in Kleinbuchstaben geschrieben.

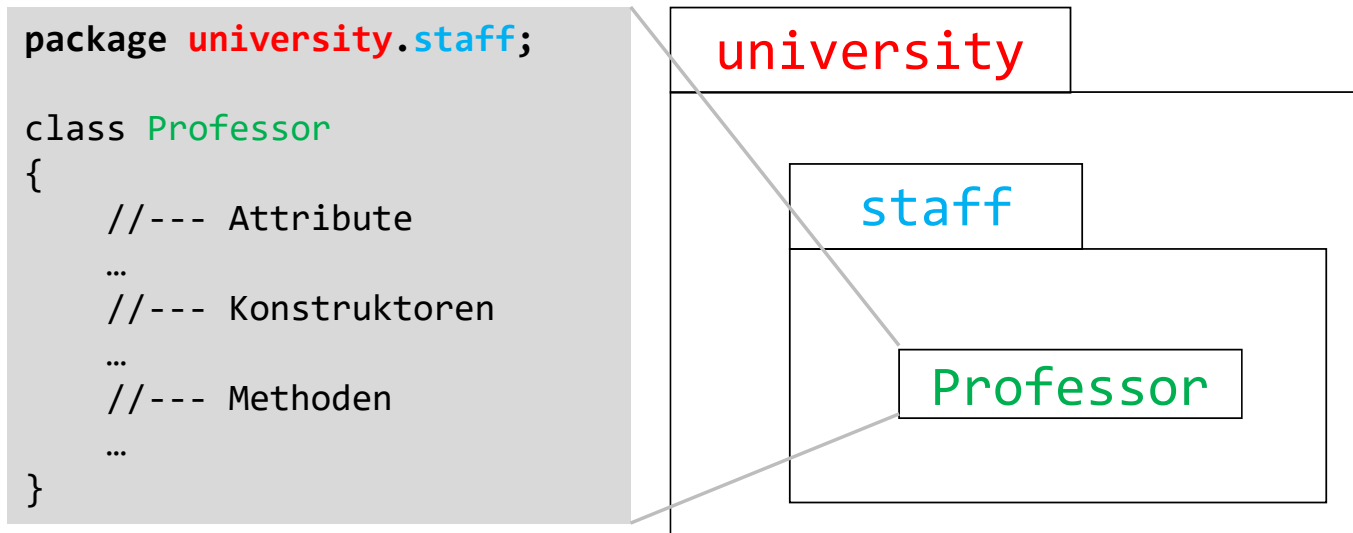
Ohne package-Deklaration gehört die Klasse dem Default Package an.

Packages lassen sich auch hierarchisch anordnen.

- D.h. Packages können wiederum Packages enthalten

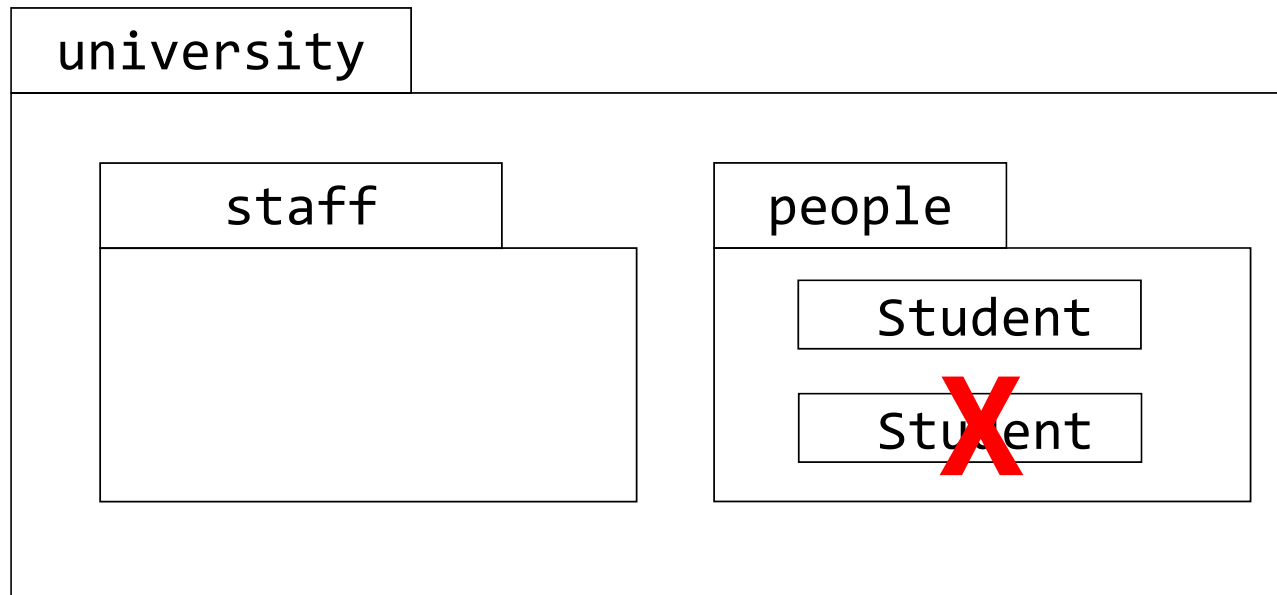


In der package-Deklaration wird sich anhand des Punktoperators durch die Hierarchie „gehängt“.



Der vollqualifizierte Klassenname ist
university.staff.Professor

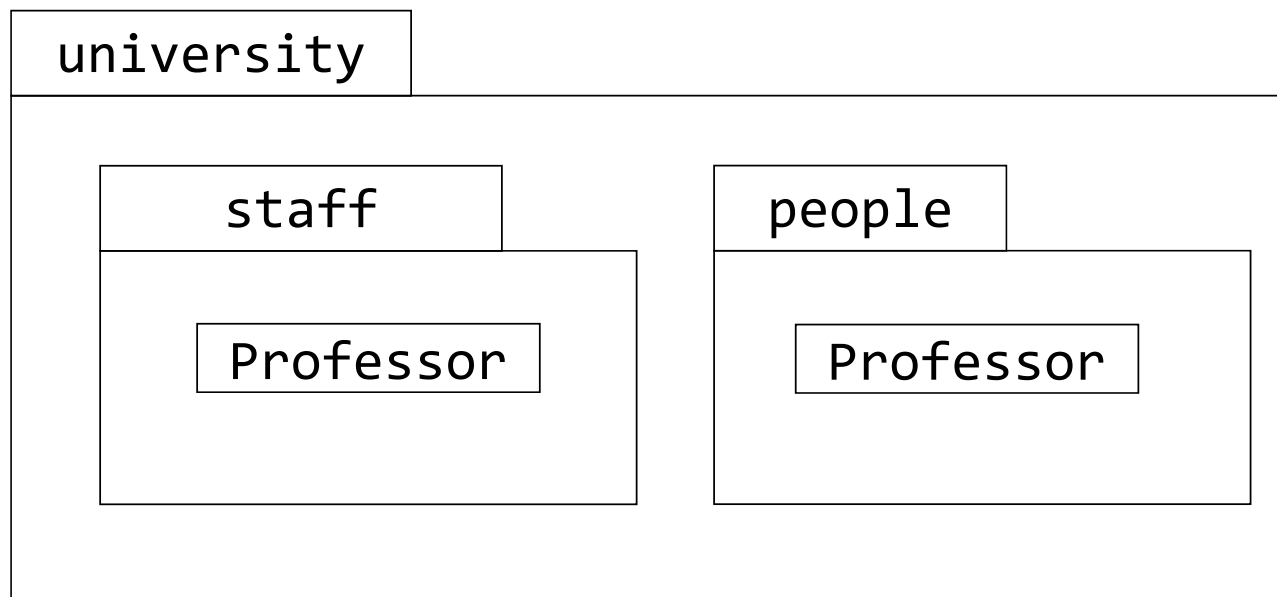
Alle Klassennamen innerhalb eines Packages müssen eindeutig sein.



Anders formuliert: Alle vollqualifizierten Klassennamen innerhalb einer Applikation müssen eindeutig sein.

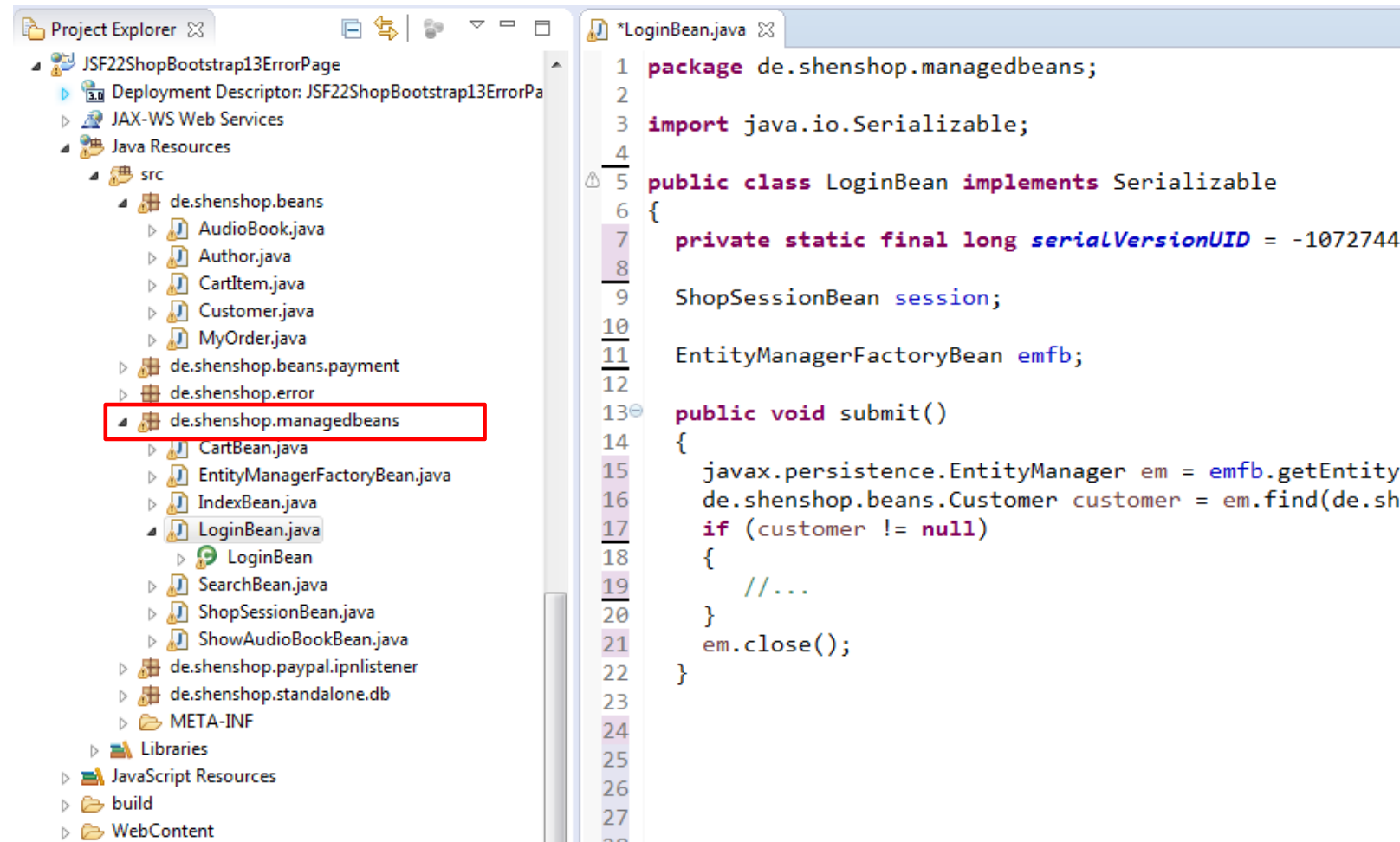
Oben wäre **university.people.Student** doppelt.

In unterschiedlichen Packages kann der gleiche Name verwendet werden.



Der vollqualifizierte Klassenname unterscheidet sich:
university.staff.Professor
university.people.Professor

In Eclipse werden
Packages als
Ordnerknoten im
Project Explorer
angezeigt.



Innerhalb einer Klasse sind zunächst nur Klassen des eigenen Packages sichtbar.

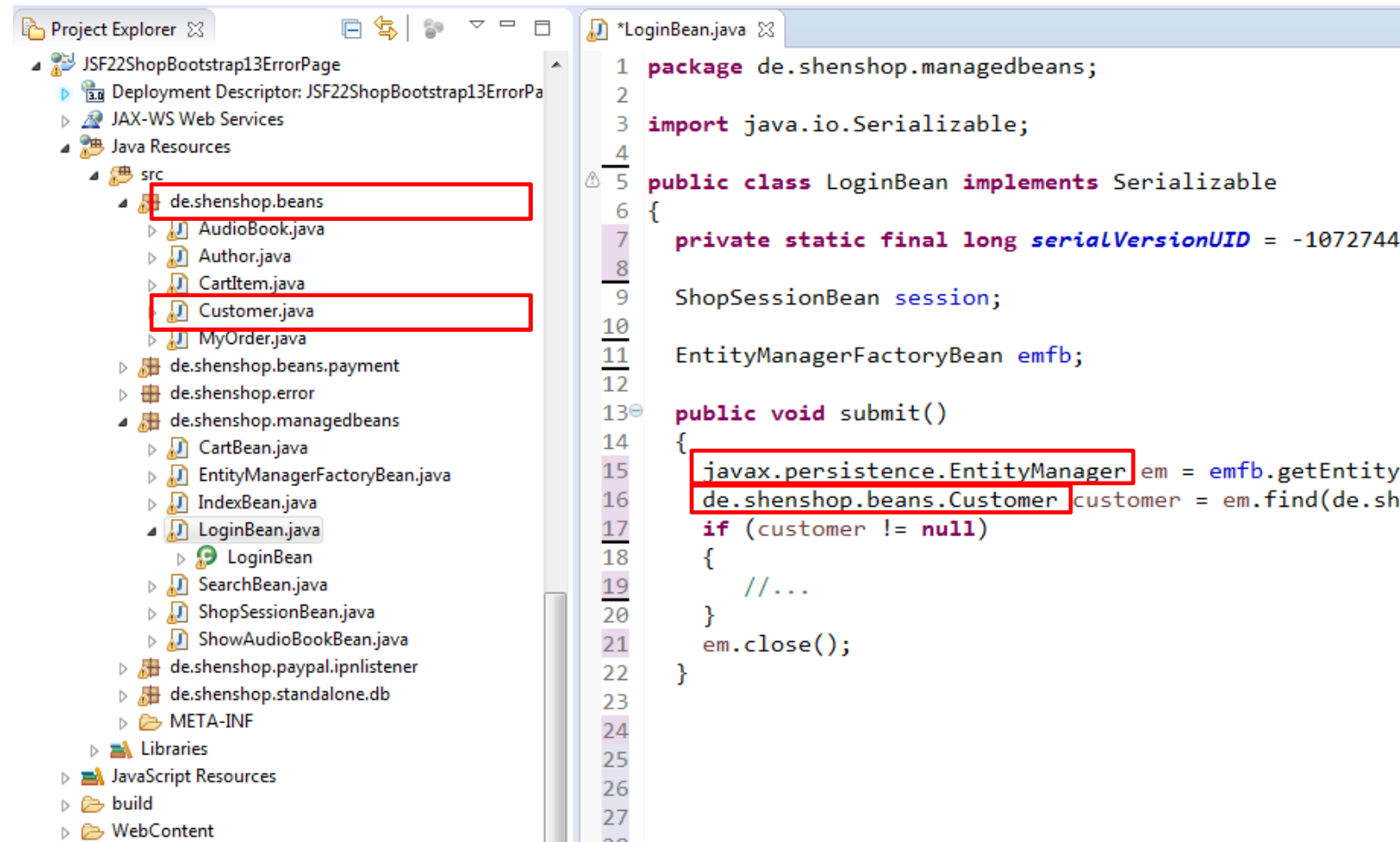
The screenshot displays an IDE interface. On the left, the 'Project Explorer' shows a project structure with the following hierarchy:

- JSF22ShopBootstrap13ErrorPage
 - Deployment Descriptor: JSF22ShopBootstrap13ErrorPa
 - JAX-WS Web Services
 - Java Resources
 - src
 - de.shenshop.beans
 - AudioBook.java
 - Author.java
 - CartItem.java
 - Customer.java
 - MyOrder.java
 - de.shenshop.beans.payment
 - de.shenshop.error
 - de.shenshop.managedbeans
 - CartBean.java
 - EntityManagerFactoryBean.java
 - IndexBean.java
 - LoginBean.java
 - LoginBean
 - SearchBean.java
 - ShopSessionBean.java
 - ShowAudioBookBean.java
 - de.shenshop.paypal.ipnlistener
 - de.shenshop.standalone.db
 - META-INF
 - Libraries
 - JavaScript Resources
 - build
 - WebContent

On the right, the code editor shows the 'LoginBean.java' file with the following code:

```
1 package de.shenshop.managedbeans;
2
3 import java.io.Serializable;
4
5 public class LoginBean implements Serializable
6 {
7     private static final long serialVersionUID = -1072744
8
9     ShopSessionBean session;
10
11     EntityManagerFactoryBean emfb;
12
13     public void submit()
14     {
15         javax.persistence.EntityManager em = emfb.getEntity
16         de.shenshop.beans.Customer customer = em.find(de.sh
17         if (customer != null)
18         {
19             //...
20         }
21         em.close();
22     }
23
24
25
26
27
28
```

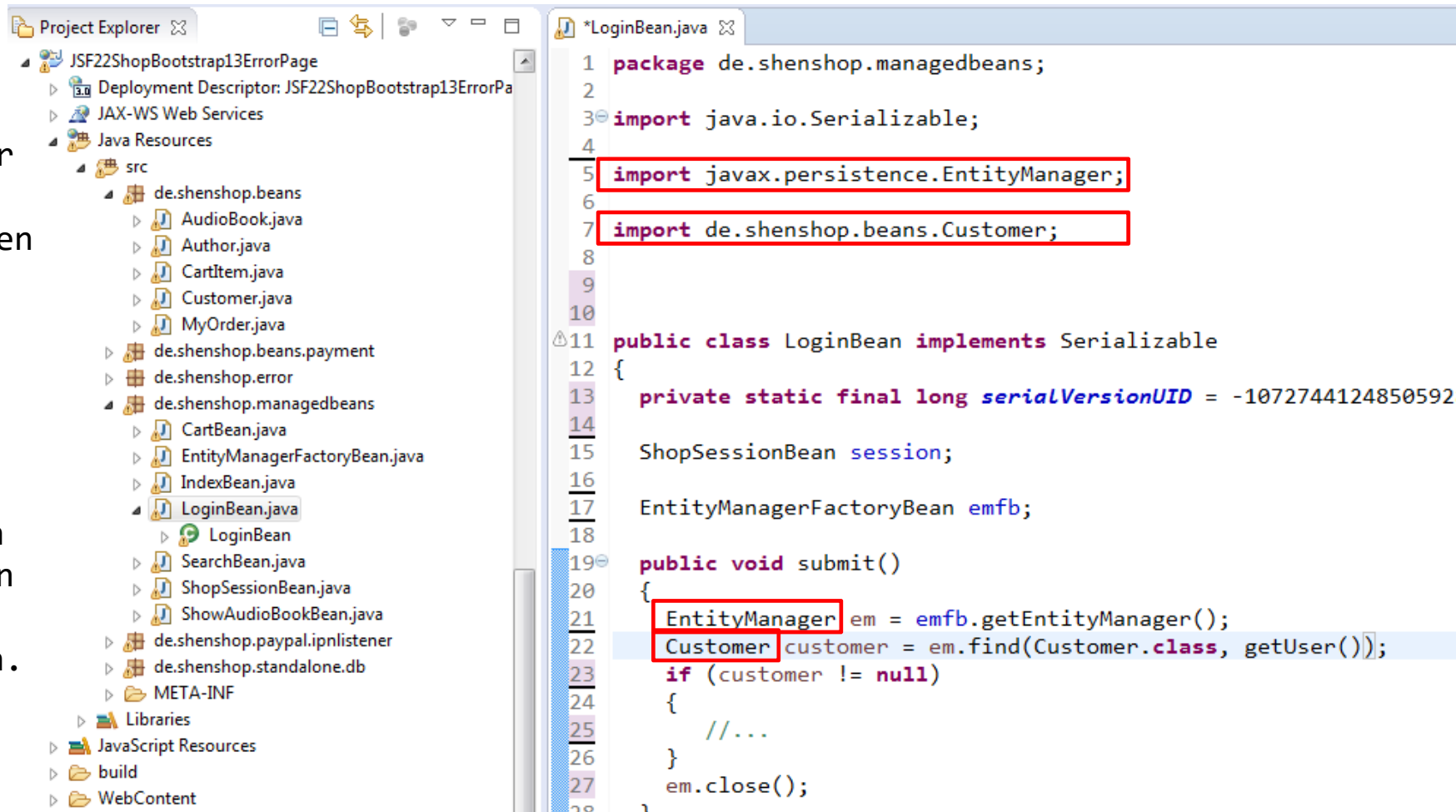
Um Klassen anderer Packages zu nutzen, ist die Angabe des vollqualifizierten Namens erforderlich.



Zu umständlich immer den ganzen Pfad zu tippen?

Um Klassen anderer Packages auf die gleiche Weise nutzen zu können, wie Klassen aus dem gleichen Package,

müssen diese zunächst durch **import** über ihren vollqualifizierten Klassennamen eingebunden werden.



The screenshot shows an IDE with two main panes. The left pane is the 'Project Explorer' showing a project structure. The right pane is the 'Editor' showing the code of `*LoginBean.java`.

Project Explorer Structure:

- JSF22ShopBootstrap13ErrorPage
 - Deployment Descriptor: JSF22ShopBootstrap13ErrorPa
 - JAX-WS Web Services
 - Java Resources
 - src
 - de.shenshop.beans
 - AudioBook.java
 - Author.java
 - CartItem.java
 - Customer.java
 - MyOrder.java
 - de.shenshop.beans.payment
 - de.shenshop.error
 - de.shenshop.managedbeans
 - CartBean.java
 - EntityManagerFactoryBean.java
 - IndexBean.java
 - LoginBean.java (selected)
 - LoginBean
 - SearchBean.java
 - ShopSessionBean.java
 - ShowAudioBookBean.java
 - de.shenshop.paypal.ipnlistener
 - de.shenshop.standalone.db
 - META-INF
 - Libraries
 - JavaScript Resources
 - build
 - WebContent

Code in *LoginBean.java:

```
1 package de.shenshop.managedbeans;
2
3 import java.io.Serializable;
4
5 import javax.persistence.EntityManager;
6
7 import de.shenshop.beans.Customer;
8
9
10
11 public class LoginBean implements Serializable
12 {
13     private static final long serialVersionUID = -1072744124850592
14
15     ShopSessionBean session;
16
17     EntityManagerFactoryBean emfb;
18
19     public void submit()
20     {
21         EntityManager em = emfb.getEntityManager();
22         Customer customer = em.find(Customer.class, getUser());
23         if (customer != null)
24         {
25             //...
26         }
27         em.close();
28     }
29 }
```

*Es ist möglich, gezielt eine einzelne
Paketkomponente mit einer import-Deklaration zu
importieren:*



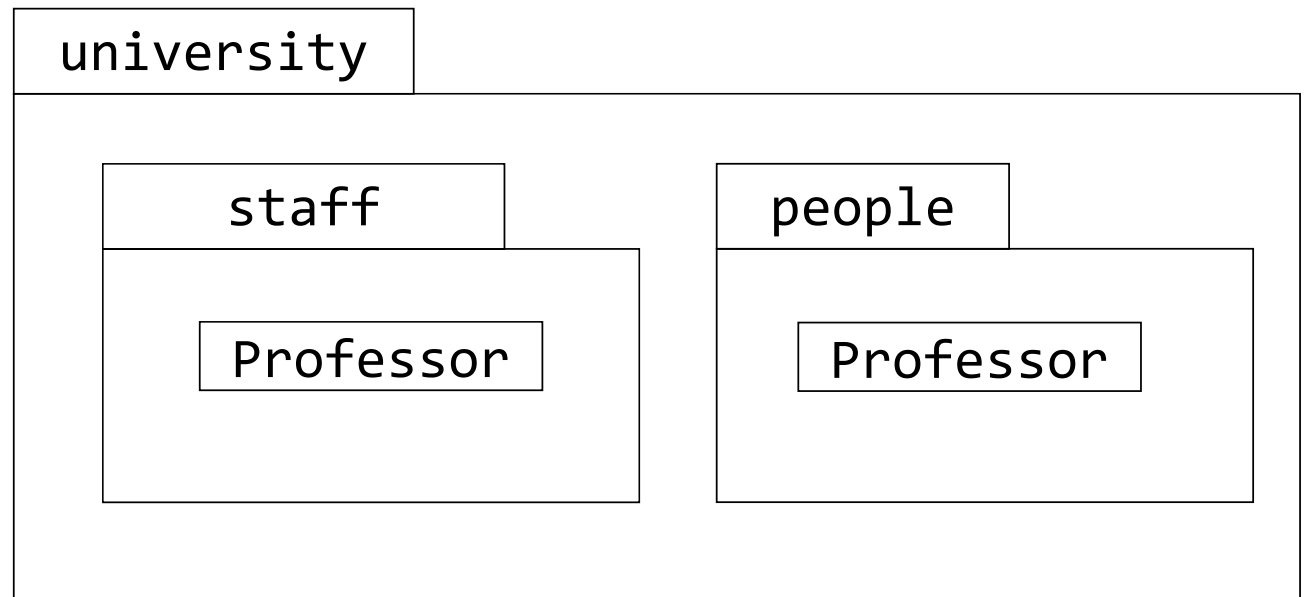
```
i mport  <paketfol ge>. <komponente>;
```

*Oder alle Komponenten aus
einem Paket:*



```
i mport  <paketfol ge>. *;
```

```
package university;  
  
import university.people.*;  
import university.staff.*;  
  
public class Meeting  
{  
    Professor p = new Professor();  
}
```



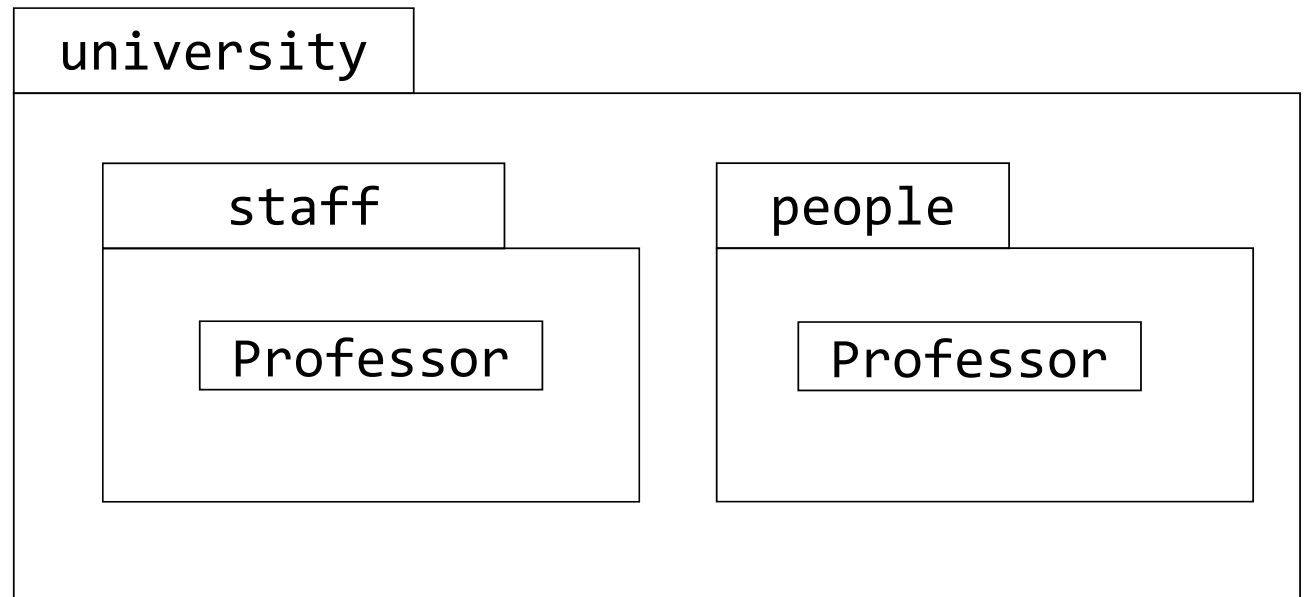
Was passiert hier?

```
package university;

import university.people.*;
import university.staff.*;

public class Meeting
{
    university.staff.Professor p = new university.staff.Professor();
}
```

Herrscht keine Eindeutigkeit,
so muss nach wie vor der
vollqualifizierte Klassenname
angegeben werden.



Wenn eine Klasse im Default Package liegt, kann diese nicht importiert werden.

Das Default Package sollte generell nicht verwendet werden, da bei der Verwendung von Frameworks häufiger Probleme auftreten.

```
import <paketfolge>*;
```

importiert keine Unterpakete. Diese müssen mit eigenen import-Deklarationen importiert werden.

```
package something.different;
```

```
class A
```

```
{
```

```
  ✗ Meeting m = new Meeting();
```

```
  ✗ Professor p = new Professor();
```

```
    ...
```

```
}
```

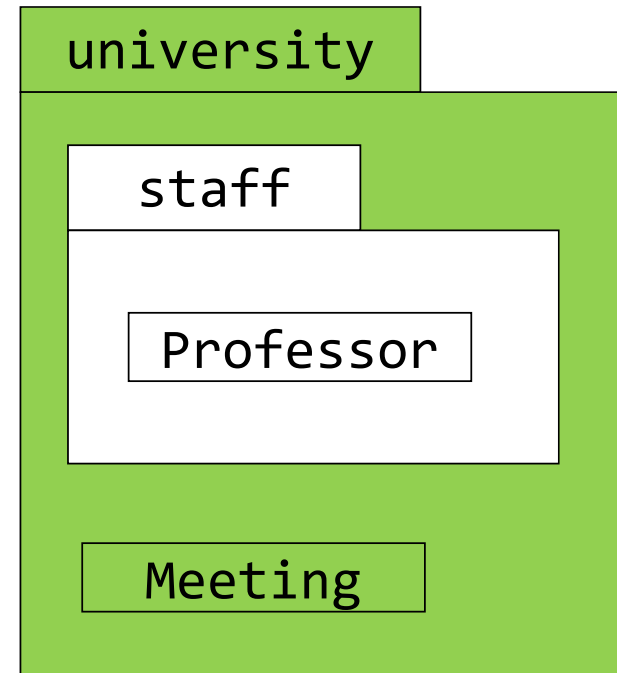
university

staff

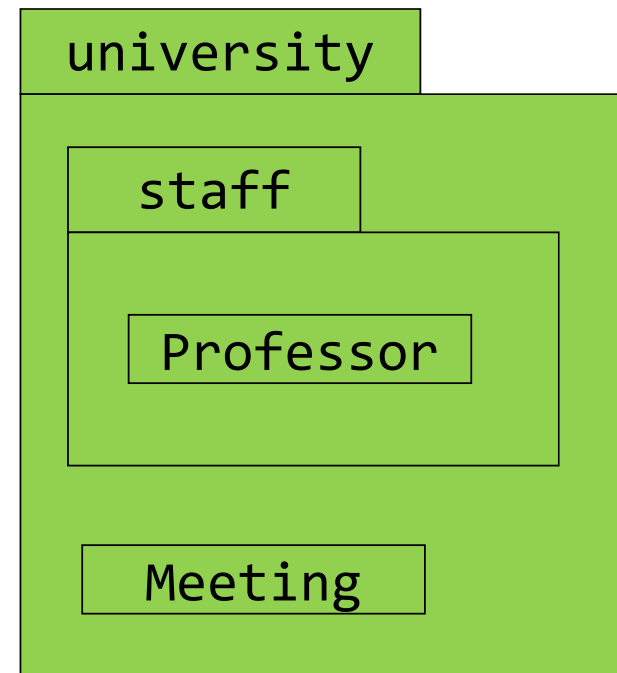
Professor

Meeting

```
package something.different;  
  
import university.*;  
  
class A  
{  
    ✓ Meeting m = new Meeting();  
    ✗ Professor p = new Professor();  
    ...  
}
```



```
package something.different;  
  
import university.*;  
import university.staff.*;  
  
class A  
{  
    ✓ Meeting m = new Meeting();  
    ✓ Professor p = new Professor();  
    ...  
}
```



Warum kann man in Java Klassen wie Math, String oder System einfach so verwenden?

Der Compiler fügt vor eine Klassendeklaration automatisch folgende import-Deklaration ein:

```
import java.lang.*;
```

Daher können alle Klassen des Pakets java.lang, wie beispielsweise Math, String, Object, Class, Double ohne explizite import-Deklaration verwendet werden.

Häufig werden Package-Namen in umgekehrter Reihenfolge des Webseitennamen der Firma/Einrichtung vergeben.

Webseite der FHWS: fhws.de

```
package de.fhws;
```

Für eine Anwendung werden Unterpackages angelegt:

```
package de.fhws.staff;
```

Welche Klassen gruppiert man zu einem Package?

Eine gute Orientierung liefern die beiden folgenden Ansätze:

“Classes that change together are packaged together.”

“Classes that are used together are packaged together.”

Robert Martin: The Principles of OOD

<http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod>

Das könnte bspw. eine Unterteilung in Frontend- und Backend-Klassen sein.

Soll eine einem Package zugeordnete Java-Klasse mit einer main-Methode gestartet werden, muss der vollqualifizierte Klassenname angegeben werden.

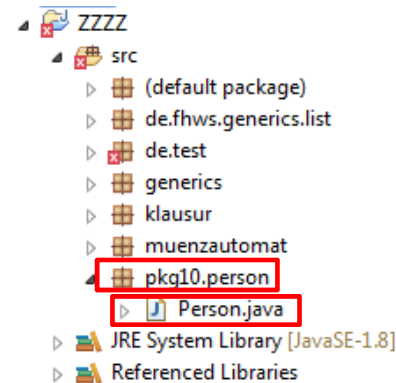
Da der Dateiname nach wie vor nur der einfache Klassenname ist, müssen die Java-Dateien in unterschiedlichen Ordnern liegen.

Daher wird eine Ordnerhierarchie angenommen, die der Package-Hierarchie entspricht.

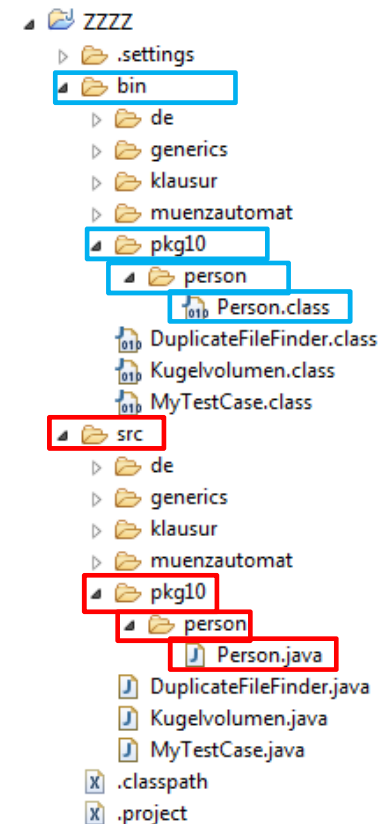
Die Java-Source Dateien
liegen in Eclipse in dem
Ordner **src**.

Eclipse legt
standardmäßig die
kompilierten Dateien im
Ordner **bin** ab.

Package Explorer Ansicht in Eclipse

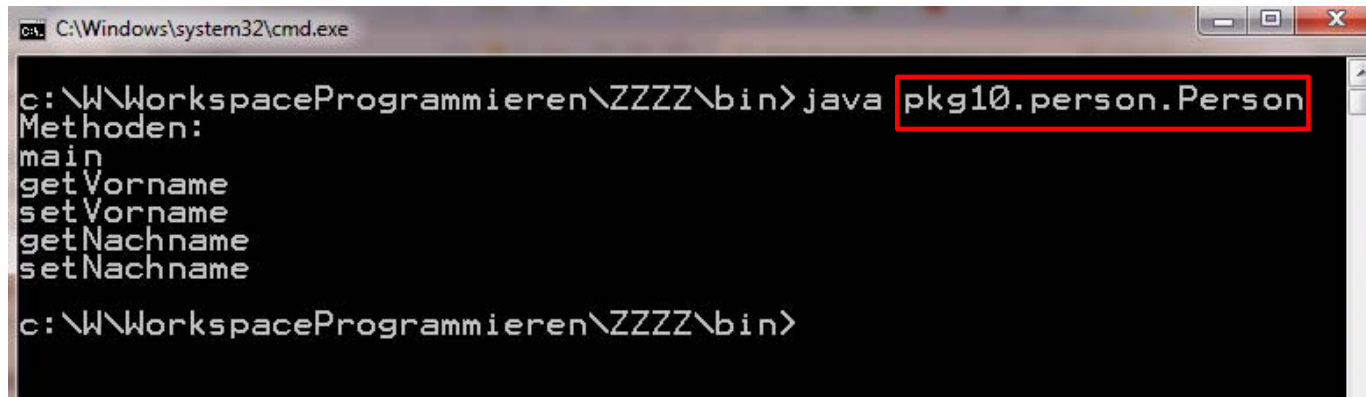


Navigator-Ansicht in Eclipse



Der Navigator zeigt die zugrundeliegende
Ordnerstruktur auf der Festplatte an

Will man jetzt eine Java-Klasse in einem Package starten:



```
C:\Windows\system32\cmd.exe

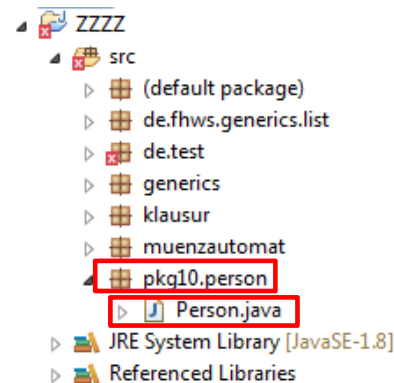
c:\W\WorkspaceProgrammieren\ZZZZ\bin>java pkg10.person.Person
Methoden:
main
getVorname
setVorname
getNachname
setNachname

c:\W\WorkspaceProgrammieren\ZZZZ\bin>
```

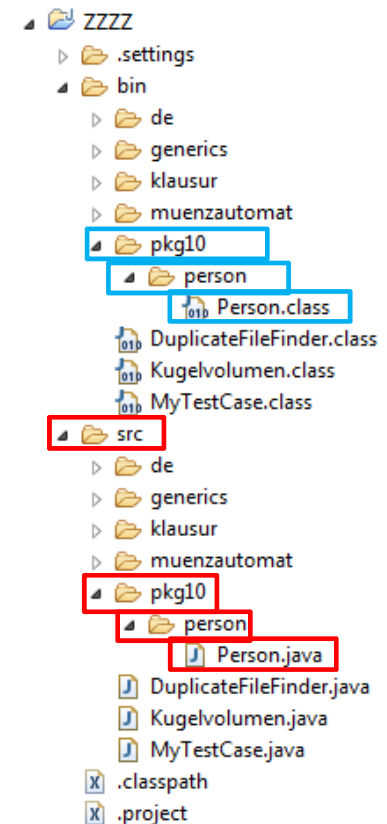
vollqualifizierter Klassenname

Package Explorer
Ansicht in Eclipse

Es wird bei obigem Befehl davon
ausgegangen, dass es von diesem
Verzeichnis aus die
Ordnerhierarchie:
`pkg10/person/`
gibt und sich die Datei
`Person.class`
in dem Ordner befindet.



Navigator-Ansicht
in Eclipse



```

package pkg10.person;

import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class Person {

    String vorname;
    String nachname;

    public Person(String vorname, String nachname) {
        this.vorname = vorname;
        this.nachname = nachname;
    }

    public String getVorname() {
        return vorname;
    }

    public void setVorname(String vorname) {
        this.vorname = vorname;
    }

    public String getNachname() {
        return nachname;
    }

    public void setNachname(String nachname) {
        this.nachname = nachname;
    }
}

```

Übrigens können zur Laufzeit durch Reflection Informationen über Klassen abgerufen werden:

```

public static void main(String[] args) {
    Person p = new Person("Joe", "Cool");
    System.out.println(p.getClass().getName());
    System.out.println();
    System.out.println("Felder: ");
    Field[] fields = p.getClass().getDeclaredFields();
    for (int i = 0; i < fields.length; i++)
        System.out.println(fields[i].getName());
    System.out.println();
    System.out.println("Methoden: ");
    Method[] methods = p.getClass().getDeclaredMethods();
    for (int i = 0; i < methods.length; i++)
        System.out.println(methods[i].getName());
}

```

pkg10.person.Person

Felder:
vorname
nachname

Methoden:
main
getVorname
setVorname

Eclipse Shortcuts

- Eclipse:
 - Organize Imports (Strg+Shift+o)
 - Getter und Setter
 - Constructor

Implizite Vererbung

Jedes Objekt verfügt in Java über grundlegende Eigenschaften.

Um das umzusetzen, **erbt** jede Klasse (auch selbstgeschriebene) in Java automatisch von der Klasse **java.lang.Object**

Folgende Eigenschaften werden vererbt (d.h. diese Eigenschaften sind in der selbstgeschriebenen Klasse verfügbar):

Methode	Beschreibung
<code>equals(Object)</code>	Vergleicht die Referenz mit der übergebenen (standardmäßig ist das Ergebnis das gleiche wie bei Verwendung von <code>==</code>).
<code>getClass()</code>	gibt die Klasse des Objekts zurück
<code>hashCode()</code>	gibt den Hashcode des Objekts (eindeutige ID) zurück, auf das die Referenz verweist
<code>toString()</code>	gibt die Stringrepräsentation des Objekts zurück, standardmäßig: <code>getClass().getName() + "@" + Integer.toHexString(hashCode())</code>
<code>wait(...)</code>	Der aktuelle Ausführungsfaden (Thread) wartet darauf, dass ein anderer Thread <code>notify()</code> oder <code>notifyAll()</code> aufruft.
<code>notify()</code>	Weckt einen Thread auf, der am Monitor des Objekts wartet.
<code>notifyAll()</code>	Weckt alle Threads auf, die am Monitor des Objekts warten.

Implizite Vererbung

toString()

Beispiel: toString()

```
public class Person {  
    String name;  
    String vorname;  
  
    public static void main(String[] args) {  
        Person p = new Person();  
        System.out.println(p);  
    }  
}  
  
    public void println(Object x) {  
        String s = String.valueOf(x);  
        ...  
    }  
  
    public static String valueOf(Object obj) {  
        return (obj == null) ? "null" : obj.toString();  
    }  
  
    public String toString() {  
        return getClass().getName() + "@" + Integer.toHexString(hashCode());  
    }  
}
```

Ausgabe:
Person@4dd761d0

Beispiel: toString()

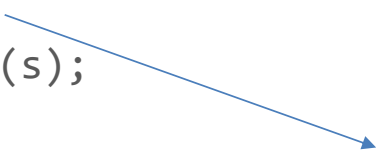
```
package pkg10.person;
```

```
public class Person {  
    String name;  
    String vorname;
```

```
    public static void main(String[] args) {  
        Person p = new Person();  
        String s = "" + p;  
        System.out.println(s);  
    }  
}
```

Ausgabe:

pkg10.person.Person@251c4123



Durch die Verknüpfung eines Strings und eines Objekts mit dem + Operator wird automatisch die toString()-Methode des Objekts aufgerufen.

Überschreiben von toString()

```
public class Rechteck {  
    double laenge;  
    double breite;  
  
    public Rechteck(double laenge, double breite) {  
        this.laenge = laenge;  
        this.breite = breite;  
    }  
  
    public double berechneFlaeche() {  
        return laenge * breite;  
    }  
  
    @Override  
    public String toString() {  
        return "Rechteck: \n  Breite: " +  
            breite + " Länge: " + laenge;  
    }  
}
```

```
public static void main(String[] args)  
{  
    Rechteck r;  
    r = new Rechteck(50.0, 20.0);  
    System.out.println(r);  
}
```

Ausgabe
Rechteck:
Breite: 20.0 Länge: 50.0

Implizite Vererbung

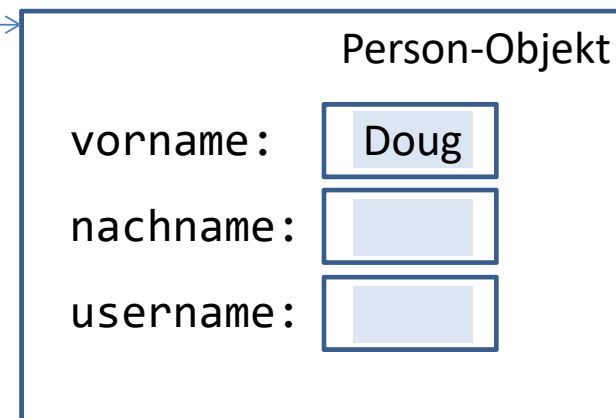
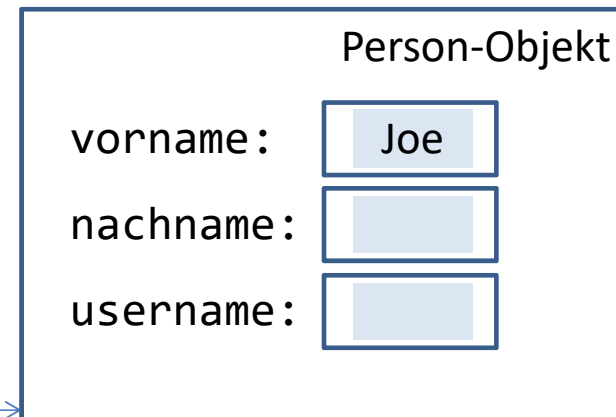
`equals()`

Beispiel: equals (geerbt von Object)

```
Person p1 = new Person();  
Person p2 = new Person();  
p1.setVorname("Joe");  
p2.setVorname("Doug");  
System.out.println(p1==p2);  
System.out.println(p1.equals(p2));
```

p1

p2



Ausgabe:

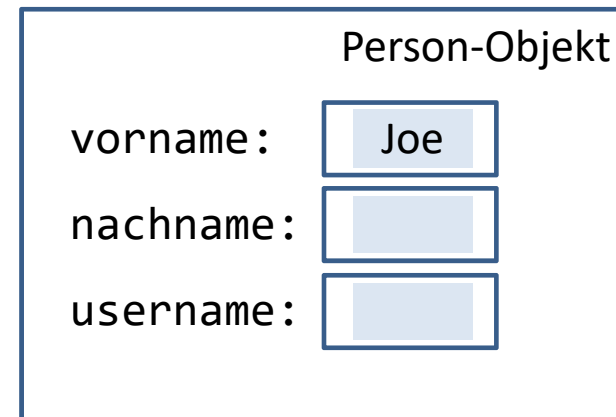
false

false

Die Referenzen sind unterschiedlich.

Beispiel: equals (geerbt von Object)

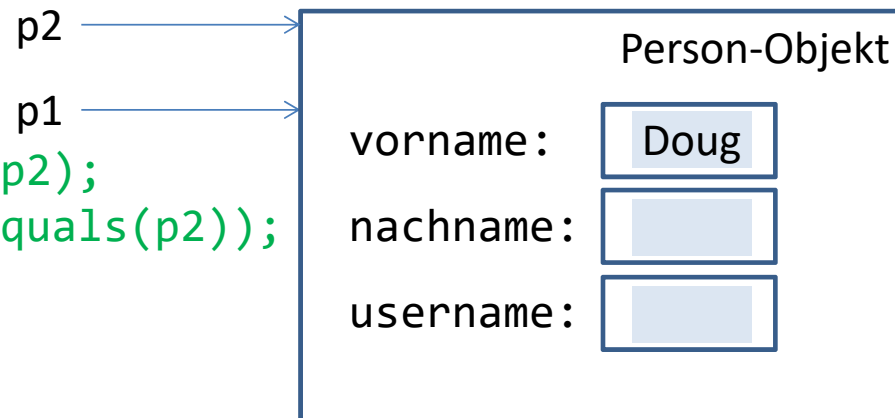
```
Person p1 = new Person();  
Person p2 = new Person();  
p1.setVorname("Joe");  
p2.setVorname("Doug");  
System.out.println(p1==p2);  
System.out.println(p1.equals(p2));
```



```
p1 = p2;
```

```
System.out.println(p1==p2);  
System.out.println(p1.equals(p2));
```

Ausgabe:
true
true



Die Referenzen sind gleich.

Beispiel: equals bei Strings

- Die Klasse String **überschreibt** die geerbte equals()-Methode.
- equals() vergleicht den Inhalt des String-Objekts mit dem Inhalt des übergebenen String-Objekts (und nicht die Referenzen)

```
String s = "Hallo Welt";  
String s2 = "Hallo";  
System.out.println(s == s2);           false  
System.out.println(s.equals(s2));      false  
s2 = s;  
System.out.println(s == s2);           true  
System.out.println(s.equals(s2));      true  
s2 = new Scanner(System.in).nextLine(); ← Eingabe von Hallo Welt  
System.out.println(s == s2);           false  
System.out.println(s.equals(s2));      true
```


equals ist eine Äquivalenzrelation

`equals()` wird als Äquivalenzrelation implementiert, d.h.:

Wenn `equals()` überschrieben wird, muss `equals()` für Nicht-null-Referenzen die folgenden drei Eigenschaften erfüllen:

- Reflexivität
- Symmetrie
- Transitivität

Grundsätzliche Definitionen und Begriffe

- „Unter einer „Menge“ verstehen wir jede Zusammenfassung M von bestimmten wohlunterschiedenen Objekten m unserer Anschauung oder unseres Denkens (welche die „Elemente“ von M genannt werden) zu einem Ganzen.“

– Georg Cantor

Bsp.: Menge $M = \{1,2,3\}$

- Quantoren:
 - Allquantor: \forall
 - Existenzquantor: \exists
 - $\forall a \in A$ Für alle Elemente a aus der Menge A
 - $\exists x \in \mathbb{N}$ Es existiert (mind.) ein x aus den natürlichen Zahlen
- Operatoren:
 - $:$ \Leftrightarrow ist definiert als äquivalent mit
 - $:$ so dass
 - $|$ für die/das gilt
- Teilmengendefinitionen:
 - $A \subseteq B : \Leftrightarrow \forall a \in A \Rightarrow a \in B$
 - $A \subset B : \Leftrightarrow \forall a \in A \Rightarrow a \in B \text{ und } A \neq B$
- A teilt b
 - $a|b : \Leftrightarrow \exists n \in \mathbb{N} : n \cdot a = b$

a teilt b ist definiert als äquivalent mit: es existiert ein n aus den natürlichen Zahlen, so dass n mal a b ergibt.

Kartesisches Produkt

- Das Kartesische Produkt: $M \times N := \{(a, b) \mid a \in M, b \in N\}$
sind alle geordneten Paare, die sich aus den
Mengen M und N bilden lassen.
- Bsp: $M = \{1, 2, 3\}, N = \{4, 5\},$
 $M \times N = \{(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)\}$

Relationen

- Eine (binäre) Relation R zwischen M und N ist definiert als eine beliebige Teilmenge des Kartesischen Produktes, d. h.

$$R \subseteq M \times N.$$

- Eine Relation nimmt eine Zuordnung von Elementen einer Menge zu den Elementen einer anderen Menge vor.

Schreibweisen:

- $(a, b) \in R$
- aRb (a ist b durch R zugeordnet)

Beispiel für die Relation =

- $(a, b) \in =$
- $a = b$ (a ist gleich b)

Beispiel für die Relation <

- $(a, b) \in <$
- $a < b$ (a ist kleiner als b)

Äquivalenzrelation

- Sei M eine Menge.
- Eine Relation $R \subseteq M \times M$ heißt **Äquivalenzrelation**, wenn sie folgende Eigenschaften erfüllt:
 - Reflexivität: $(a, a) \in R$ für alle $a \in M$
 - Symmetrie: $(a, b) \in R \Rightarrow (b, a) \in R$ für alle $a, b \in M$
 - Transitivität: $(a, b) \in R$ und $(b, c) \in R \Rightarrow (a, c) \in R$ für alle $a, b, c \in M$
- Beispiel: Sei M die Menge aller Geraden in der Ebene.
- $(a, b) \in R$ bedeute a und b sind parallel.
 - Reflexivität: a ist parallel zu a .
 - Symmetrie: Wenn a parallel zu b ist, dann ist b auch parallel zu a .
 - Transitivität: Wenn a parallel zu b und b parallel zu c ist, dann ist a auch parallel zu c .

equals ist eine Äquivalenzrelation

`equals()` wird als Äquivalenzrelation implementiert, d.h.:

wenn `equals()` überschrieben wird, muss `equals()` für Nicht-null Referenzen die folgenden drei Eigenschaften erfüllen:

- reflexiv: `a.equals(a)` muss `true` sein
- symmetrisch: Wenn `a.equals(b)` `true` ist, muss `b.equals(a)` auch `true` sein.
- transitiv: Wenn `a.equals(b)` und `b.equals(c)` `true` sind, muss auch `a.equals(c)` `true` sein.

Ordnungsrelation (Halbordnung)

- Sei M eine Menge.
- Eine Relation $R \subseteq M \times M$ heißt **Ordnungsrelation (Halbordnung)**, wenn sie folgende Eigenschaften erfüllt:
 - Reflexivität: $(a, a) \in R$ für alle $a \in M$
 - Anti-Symmetrie: $(a, b) \in R$ und $(b, a) \in R \Rightarrow a = b$ für alle $a, b \in M$
 - Transitivität: $(a, b) \in R$ und $(b, c) \in R \Rightarrow (a, c) \in R$ für alle $a, b, c \in M$
- Beispiel: Sei $M = \mathbb{N}$. $(a, b) \in R$ bedeute $a|b$ (d.h. a teilt b).
 - Reflexivität: $a|a$, z.B. $3|3$. Denn $\exists n \in \mathbb{N}: n \cdot a = a$, nämlich $n = 1$.
 - Anti-Symmetrie: Wenn $a|b$ und $b|a$, dann $a = b$, denn
$$\begin{aligned} &\exists n_1 \in \mathbb{N}: n_1 \cdot a = b \text{ und } \exists n_2 \in \mathbb{N}: n_2 \cdot b = a \\ &\Rightarrow n_1 \cdot n_2 \cdot b = b \\ &\Rightarrow n_1 = n_2 = 1, \text{ da } n_1, n_2 \in \mathbb{N} \\ &\Rightarrow a = b \end{aligned}$$

Ordnungsrelation (Halbordnung)

- Sei M eine Menge.
- Eine Relation $R \subseteq M \times M$ heißt **Ordnungsrelation (Halbordnung)**, wenn sie folgende Eigenschaften erfüllt:
 - Reflexivität: $(a, a) \in R$ für alle $a \in M$
 - Anti-Symmetrie: $(a, b) \in R$ und $(b, a) \in R \Rightarrow a = b$ für alle $a, b \in M$
 - Transitivität: $(a, b) \in R$ und $(b, c) \in R \Rightarrow (a, c) \in R$ für alle $a, b, c \in M$
- Beispiel: Sei $M = \mathbb{N}$. $(a, b) \in R$ bedeute $a|b$ (d.h. a teilt b).
 - Transitivität: Wenn $a|b$ und $b|c$, dann $a|c$, z. B. $3|6$ und $6|18 \Rightarrow 3|18$
 $\exists n_1 \in \mathbb{N}: n_1 \cdot a = b$ und $\exists n_2 \in \mathbb{N}: n_2 \cdot b = c$
 $\Rightarrow n_2 \cdot n_1 \cdot a = c$
 $\Rightarrow n_2 \cdot n_1 = n_3 \in \mathbb{N}$, da auch $n_1, n_2 \in \mathbb{N}$
 $\Rightarrow \exists n_3 \in \mathbb{N}: n_3 \cdot a = c$
 $\Rightarrow a|c$

Ordnungsrelation (Striktordnung)

- Sei M eine Menge.
- Eine Relation $R \subseteq M \times M$ heißt **Ordnungsrelation (Striktordnung)**, wenn sie folgende Eigenschaften erfüllt:
 - Asymmetrie: $(a, b) \in R \Rightarrow (b, a) \notin R$ für alle $a, b \in M$
 - Transitivität: $(a, b) \in R$ und $(b, c) \in R \Rightarrow (a, c) \in R$ für alle $a, b, c \in M$
- Beispiel: Sei M die Menge aller Teilmengen von \mathbb{N} .
 $(A, B) \in R$ bedeute $A \subset B$ (d.h. A ist echte Teilmenge von B).
 - Asymmetrie:
$$(A, B) \in R \Leftrightarrow A \subset B \underset{\text{Def.}}{\Leftrightarrow} \forall a \in A \Rightarrow a \in B \text{ und } A \neq B$$
Annahme: $(B, A) \in R \Leftrightarrow B \subset A \underset{\text{Def.}}{\Leftrightarrow} \forall b \in B \Rightarrow b \in A \text{ und } B \neq A$ Aus $A \subset B$ und $B \subset A$ folgt $A = B$. Widerspruch zu $A \neq B$. Daher $(B, A) \notin R$.

Ordnungsrelation (Striktordnung)

- Sei M eine Menge.
- Eine Relation $R \subseteq M \times M$ heißt **Ordnungsrelation (Striktordnung)**, wenn sie folgende Eigenschaften erfüllt:
 - Asymmetrie: $(a, b) \in R \Rightarrow (b, a) \notin R$ für alle $a, b \in M$
 - Transitivität: $(a, b) \in R$ und $(b, c) \in R \Rightarrow (a, c) \in R$ für alle $a, b, c \in M$
- Beispiel: Sei M die Menge aller Teilmengen von \mathbb{N} .
 $(A, B) \in R$ bedeute $A \subset B$ (d.h. A ist echte Teilmenge von B).

- Transitivität:

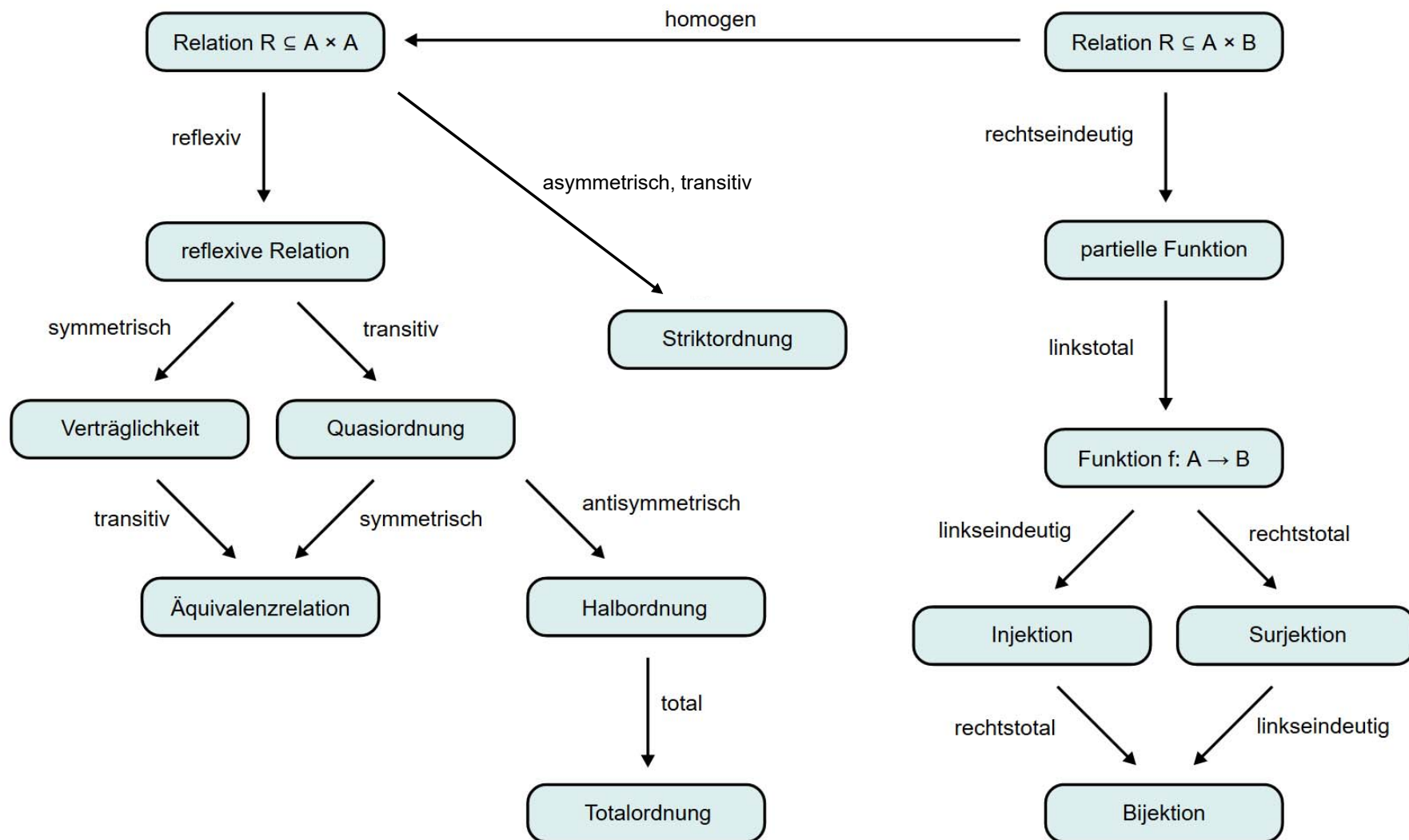
$$(A, B) \in R \Leftrightarrow A \subset B \stackrel{\text{Def.}}{\Leftrightarrow} \forall a \in A \Rightarrow a \in B \text{ und } A \neq B$$

$$(B, C) \in R \Leftrightarrow B \subset C \stackrel{\text{Def.}}{\Leftrightarrow} \forall b \in B \Rightarrow b \in C \text{ und } B \neq C$$

Da $\forall a \in A \Rightarrow a \in B$ und $\forall b \in B \Rightarrow b \in C$ folgt $\forall a \in A \Rightarrow a \in C$

Da $A \neq B$ und $B \neq C$ folgt $A \neq C$

$$\forall a \in A \Rightarrow a \in C \text{ und } A \neq C \stackrel{\text{Def.}}{\Leftrightarrow} A \subset C \Leftrightarrow (A, C) \in R$$



source: https://upload.wikimedia.org/wikipedia/commons/0/08/Types_of_relation_ti.svg

Aufgaben

Aufgabe 1:

Gegeben Sei Menge $M = \{\Delta, \nabla, \cdot\}$.

- a) Bilden Sie das kartesische Produkt $M \times M$.
- b) Gegeben Sei die Relation $\sim = \{(\Delta, \Delta), (\Delta, \nabla), (\nabla, \cdot)\}$. Ergänzen Sie \sim , so dass \sim zu einer Ordnungsrelation auf M wird.

Aufgabe 2:

Zeigen Sie, dass \leq eine Ordnungsrelation auf \mathbb{R} ist.

Nehmen Sie dabei folgende Definition zur Hilfe:

$$\text{Für } a, b \in \mathbb{R} \text{ gilt: } a \leq b \Leftrightarrow b - a \in \mathbb{R}_0^+$$

Beispiel aus der Operationellen Semantik

$$\rightarrow \subseteq (Cmd \times \Sigma) \times \Sigma$$

\rightarrow ist eine Relation, die eine Verknüpfung aus Befehl und Anfangszustand in einen Endzustand überführt

$$\langle c, \sigma \rangle \rightarrow \sigma_{neu} \quad , c \in Cmd, \sigma, \sigma_{neu} \in \Sigma$$

Variadic functions (varargs)

Wir erinnern uns an die `System.printf` Methode:

```
System.out.printf("%d\t\t%.2f\n", fahrenheit , celsius);
```

```
System.out.printf("%" + anzahlLeerzeichen + "%.2f", pizzas[i].getPreis());
```

Wie funktioniert es, dass die `printf`-Methode 2, 3 oder mehr Parameter besitzt?

Die `printf`-Methode ist eine sogenannte **variadic function**.

variadic function/varargs-Methode

- Eine variadic function enthält einen sogenannten varargs-Parameter:

```
public PrintStream printf(String format, Object... args)
```

- Der varargs-Parameter muss der letzte in der Parameterliste sein!
- Es darf nur einen varargs-Parameter in einer Methodensignatur geben!

variadic function (II)

- Ein varargs-Parameter wird innerhalb einer Methode wie ein Array verwendet.

```
public static String concat(String... strings)
{
    String result = "";
    for (int i = 0; i < strings.length; i++)
    {
        result += strings[i];
        if (i != strings.length - 1) result += ", ";
    }
    return result;
}
```

- Die Aufrufe können wie folgt erfolgen:

```
System.out.println(concat("Hallo"));
System.out.println(concat("Hallo", "Welt", "!"));
System.out.println(concat(new String[] {"Hallo", "Welt", "!"}));
```

Der Compiler behandelt für Varargs-Methoden den Aufruf:

```
concat("Hallo", "Welt", "!")
```

wie

```
concat(new String[] {"Hallo", "Welt", "!"})
```

Nehmen wir ein Polynom als Beispiel:

```
public class PolynomDrittenGrades
{
    double a;
    double b;
    double c;
    double d;

    public PolynomDrittenGrades(double a, double b, double c, double d)
    {
        this.a = a;
        this.b = b;
        this.c = c;
        this.d = d;
    }
    ...
    public static void main(String[] args)
    {
        PolynomDrittenGrades p = new PolynomDrittenGrades(1.5, 4, 1.3, -8);
    }
}
```

Wir wollen ein Polynom n-ten Grades ermöglichen
bei gleichbleibender Benutzung:

```
public class Polynom
{
    double[] coefficients;

    public Polynom(double... coefficients)
    {
        this.coefficients = coefficients;
    }
    ...
    public static void main(String[] args)
    {
        Polynom p = new Polynom(1.5, 4, 1.3, -8);
    }
}
```

