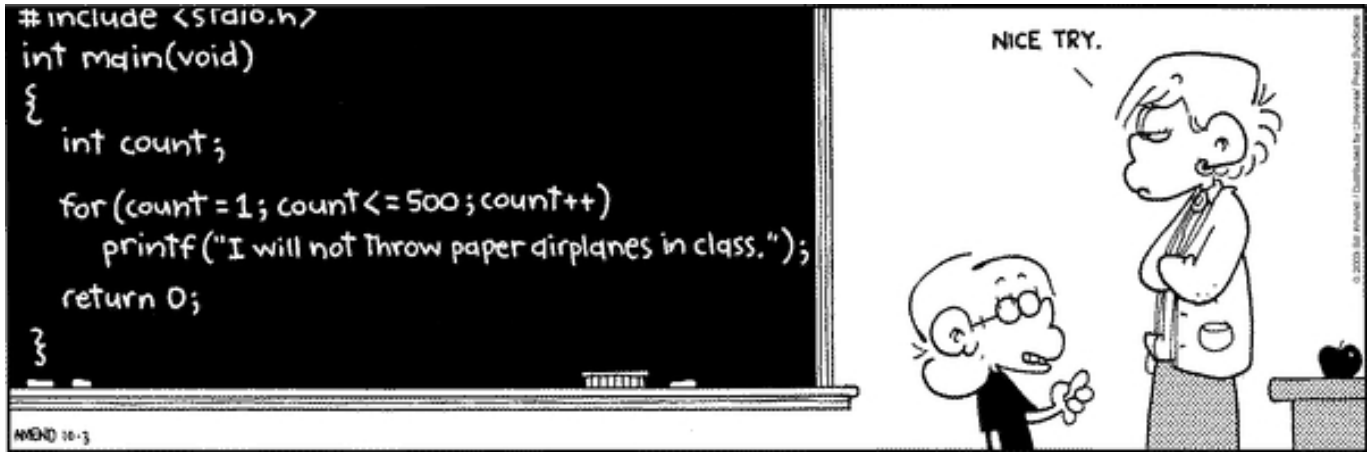
Grundlagen der Programmierung

VL12: Klassen – Teil 2

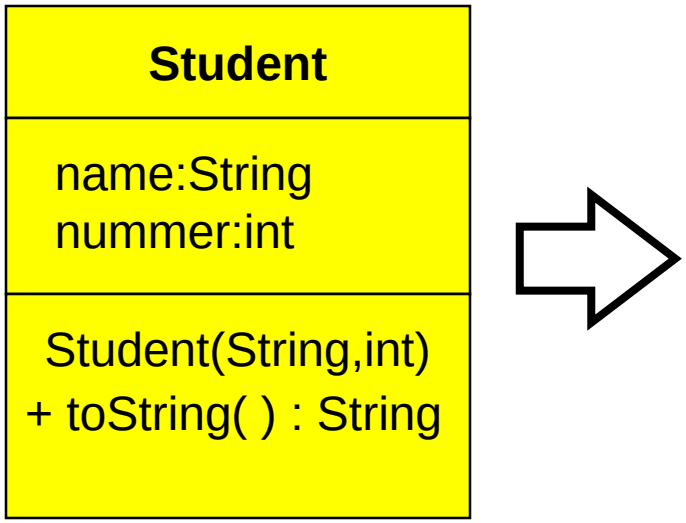
Prof. Dr. Samuel Kounev
M.Sc. Norbert Schmitt



Motivation

Java

- Letze Vorlesung: Klasse **`Student`**
- Code aus abstrakter Klassenbeschreibung (UML-Modell)

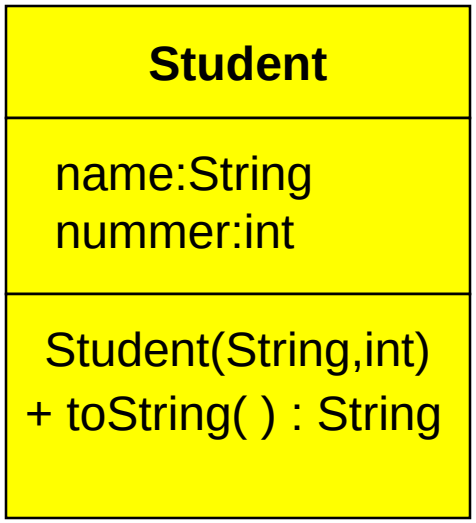



```
public class Student {

  String name;
  int nummer;

  Student(String n, int nr) {
    this.name = n;
    this.nummer = nr;
  }

  public String toString() {
    return "Student "
           +this.name
           +" hat die Matrikelnummer "
           +this.nummer
           +".";
  }

}
```

Motivation

Java

- Heute: Mehr Aspekte aus Java für bessere Student – Klasse
- Schlüsselwort **`static`**

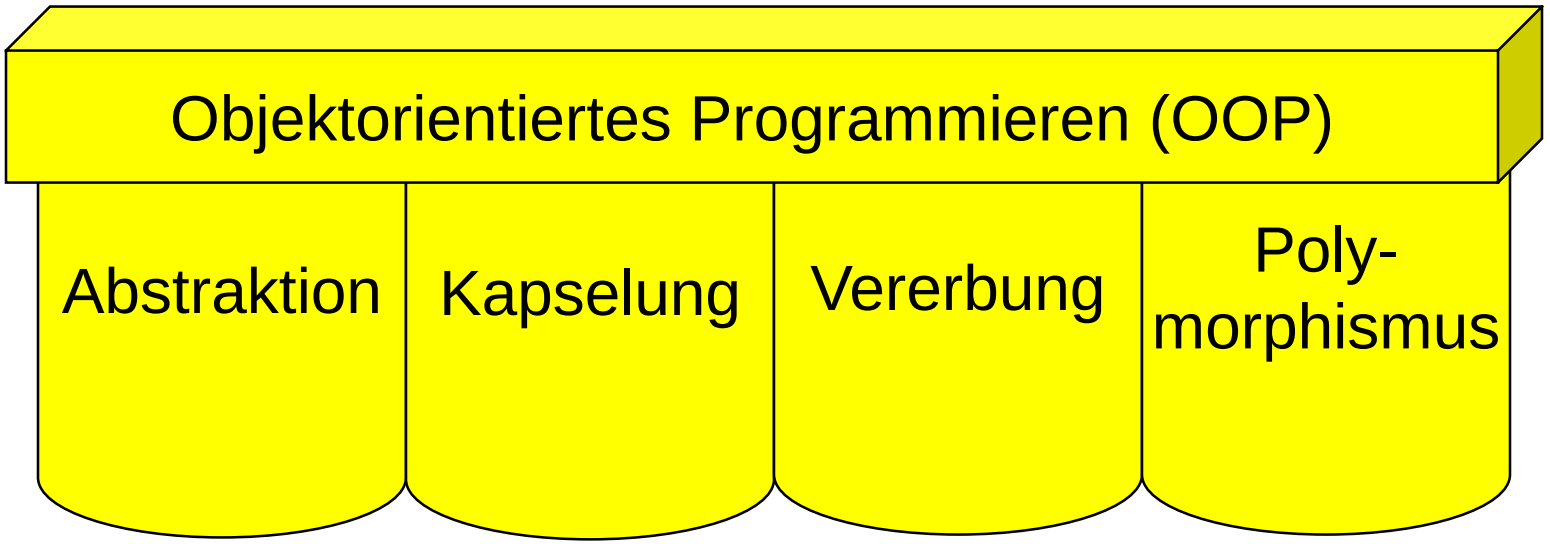


Inhalt

- **Zugriffsrechte unter Klassen**
- Klassenvariablen und –methoden

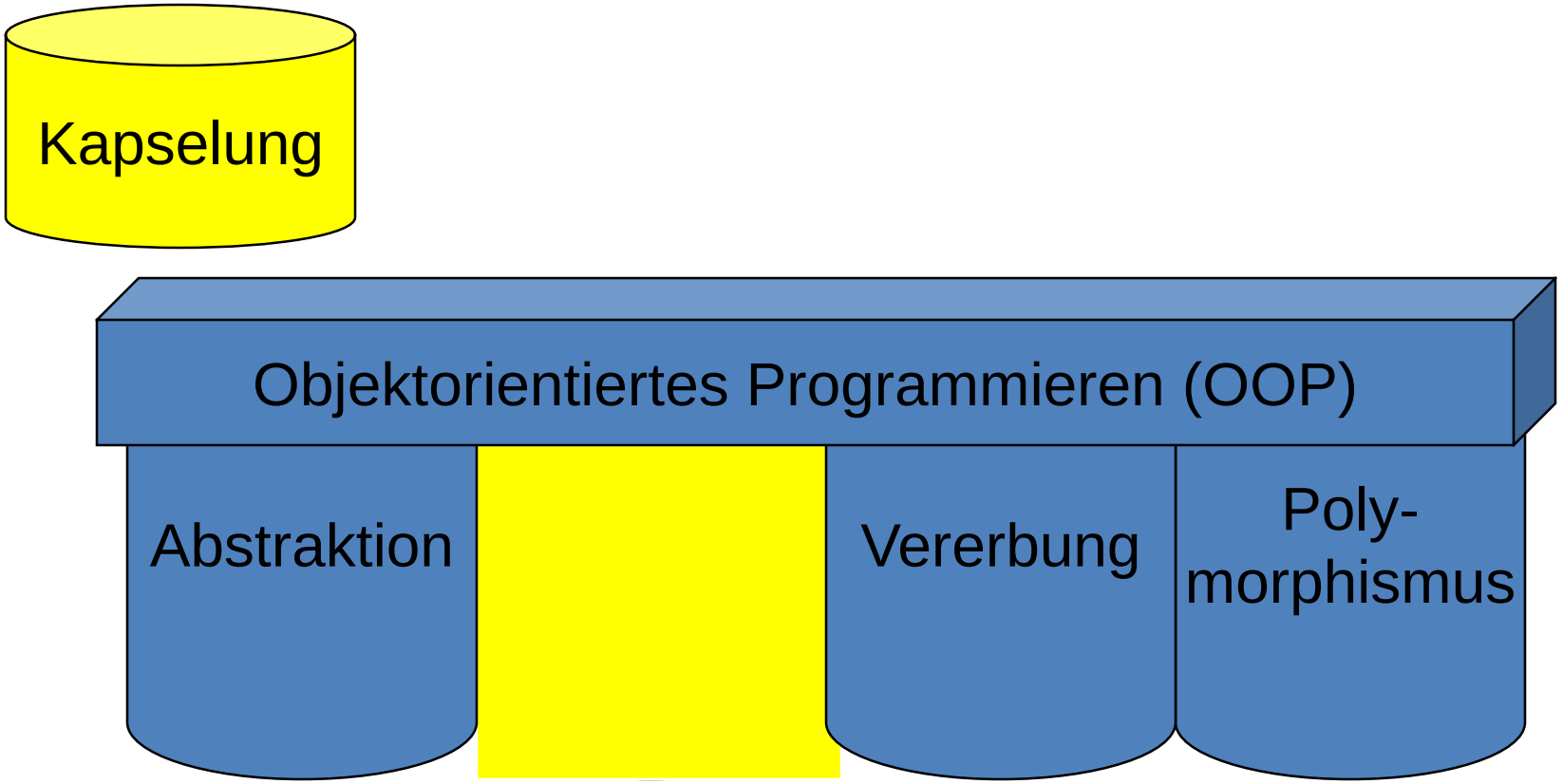
Zugriffsrechte unter Klassen

Vier grundlegende Prinzipien des objektorientierten Programmierens:



Als nächstes: **Kapselung**

Zugriffsrechte unter Klassen



Zugriffsrechte unter Klassen

ist die Philosophie, Daten und Methoden fest an ihr zugehöriges Objekt zu binden

Daten werden

- dem Benutzer nicht direkt zugänglich gemacht
- vor Zugriffen von außen geschützt (**information hiding / data hiding**)

→ Das Verhalten eines Objekts wird nur durch seine Methoden bestimmt

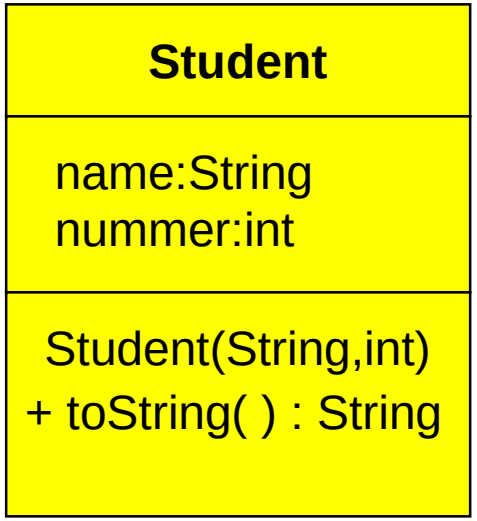
Die Trennung der **Schnittstelle** (der nach außen sichtbaren Methoden) und der **internen Realisierung** bewahrt Programmierer vor Problemen, die sich bei einer internen Umstrukturierung der Klasse ergeben.

Zugriffsrechte unter Klassen

Java

Bis jetzt, Zugriff auf Instanzvariablen mit:

```
<Objektname>.<Variablenname>
```



Beispiel **`Student`**:

Daten (**`name`** und **`nummer`**) sowie Methoden (**`toString`**) sind an ihre zugehörigen Objekte gebunden - eine Grundforderung des Prinzips der Kapselung

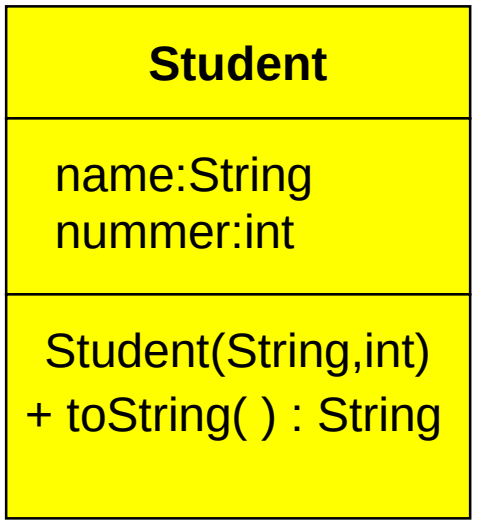
Direkter Zugriff auf **`name`** und **`nummer`** verstößt gegen die zweite Grundforderung: das **data hiding**

Zugriffsrechte unter Klassen

Java

Bis jetzt, Zugriff auf Instanzvariablen mit:

`<Objektname>.<Variablenname>`



Beispi

Dat oden

(**`toS`**

Obje derung

des Prinz g

Direkter Z iff auf **`name`** und **`nummer`** verstößt gegen die zweite Grundforderung: das **data hiding**

Wie können wir unsere Daten besser „verbergen“?

Zugriffsrechte unter Klassen

Java

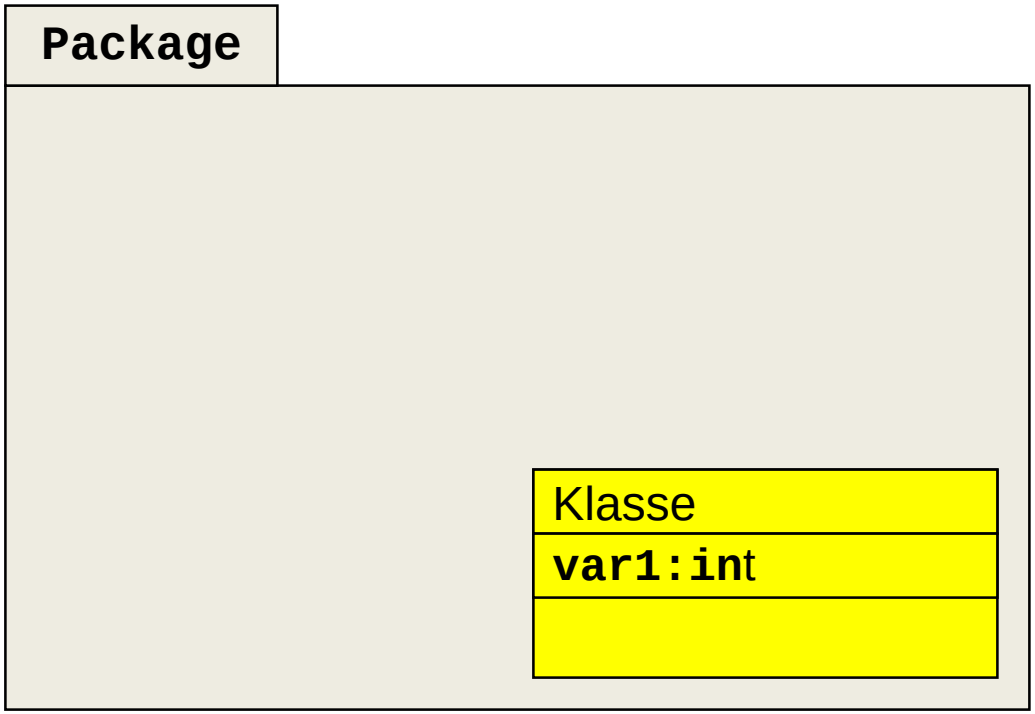
- Java erlaubt **Zugriffs-Modifizierer** (**access modifier**), davon gibt es vier verschiedene:

① **public**

② **private**

③ **protected**

④ **package**



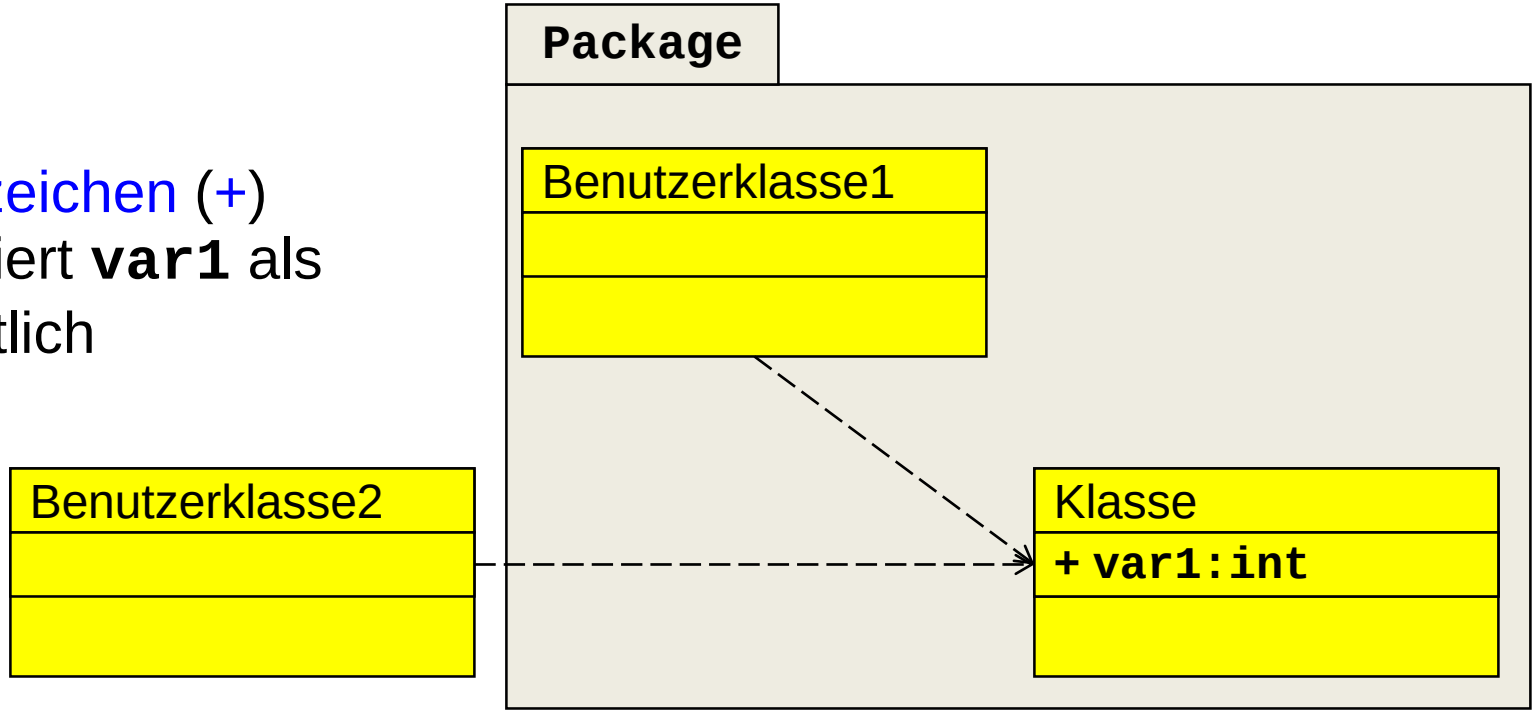
Zugriffsrechte unter Klassen

Java

`public` (öffentlich): Jede Klasse darf auf eine öffentliche Klasse, Methode oder Variable zugreifen

UML:

Pluszeichen (+) markiert **`var1`** als öffentlich



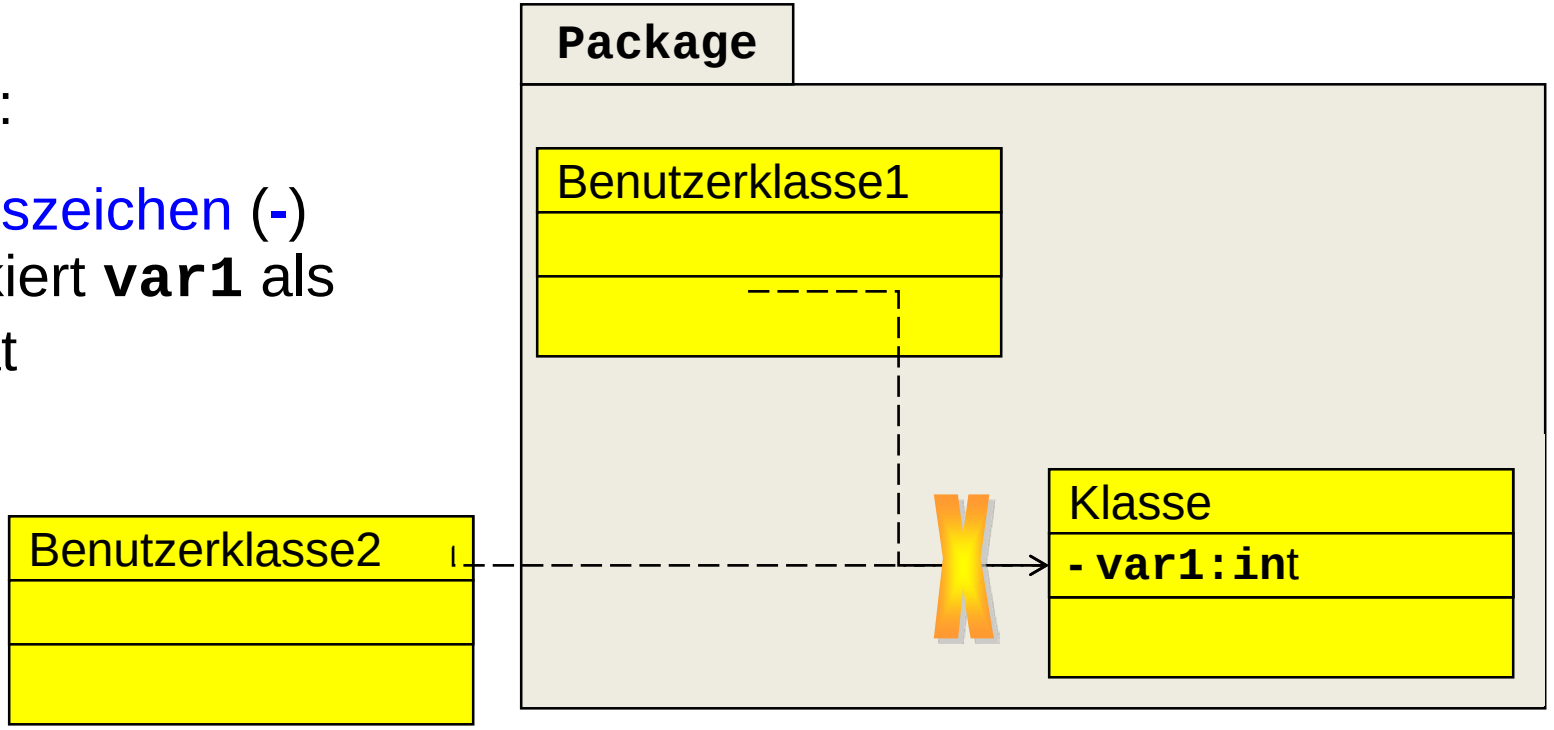
Zugriffsrechte unter Klassen

Java

`private` (privat): Private Methoden und Variablen können nur innerhalb der Klasse verwendet werden, in der sie definiert sind (sie sind nur dort sichtbar)

UML:

Minuszeichen (**-**) markiert **`var1`** als privat



Zugriffsrechte unter Klassen

Java

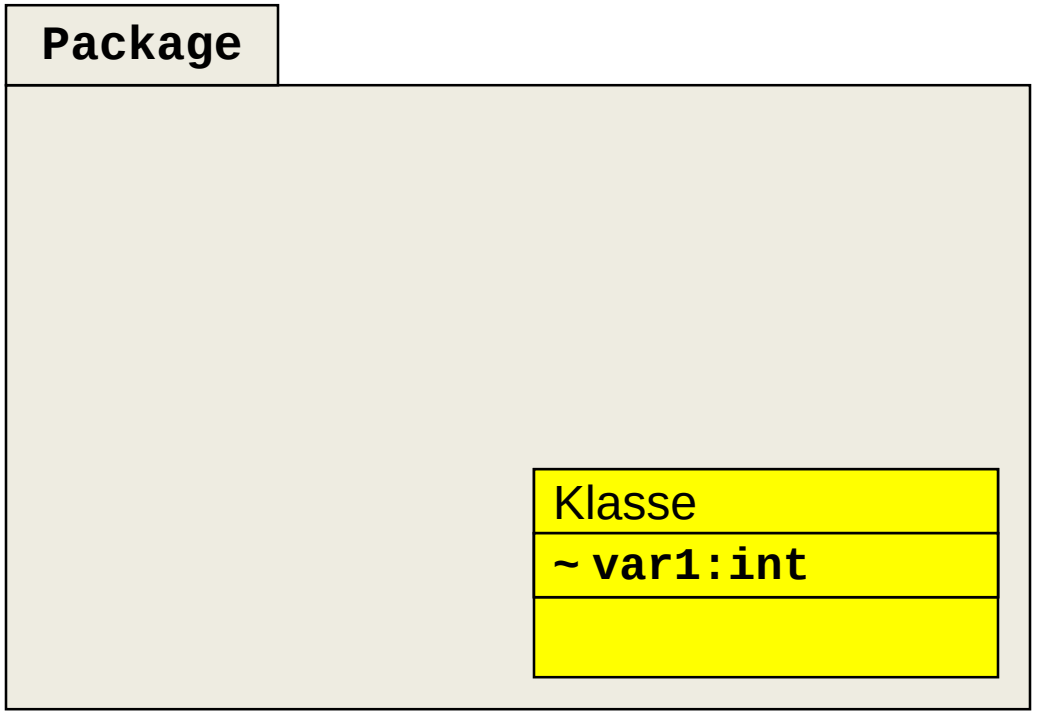
`package` (Paket): Kann nur innerhalb des Pakets, in dem sich die Klasse befindet, verwendet werden

Anmerkung: Dies ist das Standard-Verhalten (kein Modifizierer)

UML:

keine Unterstützung!

Wir verwenden Tilde (**~**)



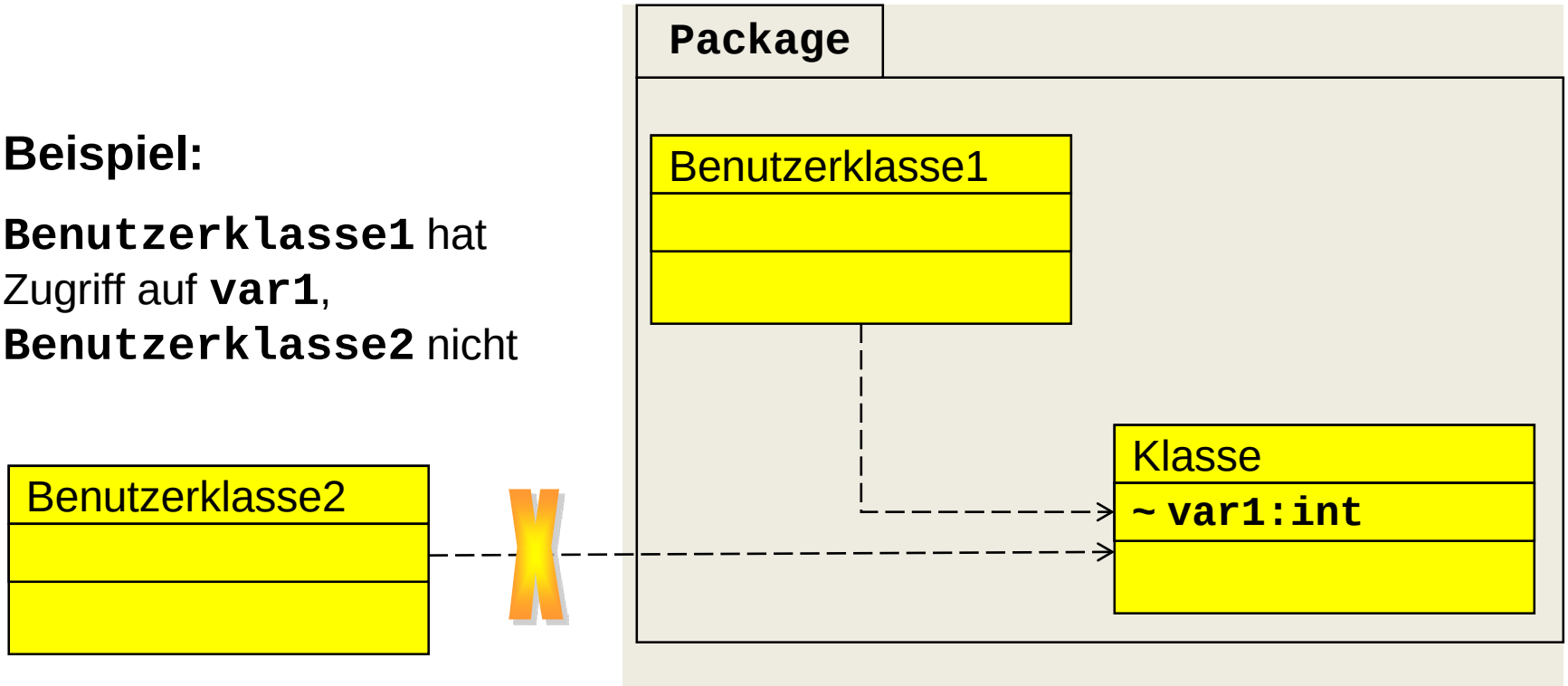
Zugriffsrechte unter Klassen

Java

Pakete sind Zusammenfassungen von Klassen
z.B.: **`java.lang`** oder das **`java.util`**

Beispiel:

`Benutzerklasse1` hat Zugriff auf **`var1`**,
`Benutzerklasse2` nicht



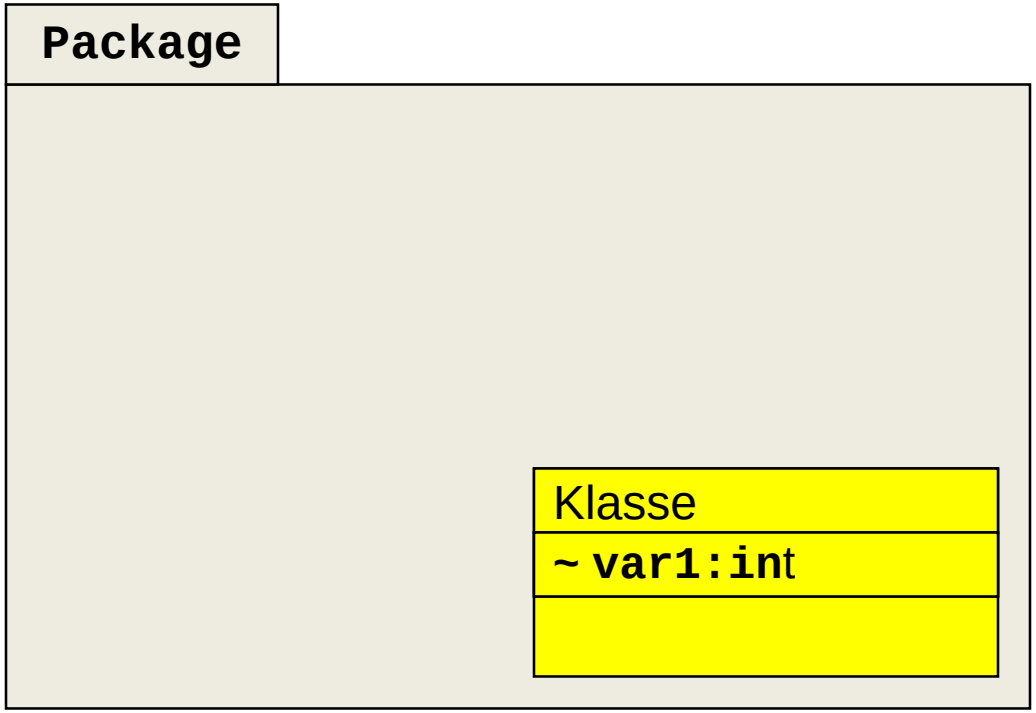
Zugriffsrechte unter Klassen

Standard-Modifizierer von UML und Java sind unterschiedlich:

- Java: **`package`**
- UML: **`public`**

Wir verwenden Java-Konventionen:

Wir werden in allen nachfolgenden UML-Modellen öffentliche (**`public`**) Variablen und Methoden durch ein (+) kennzeichnen



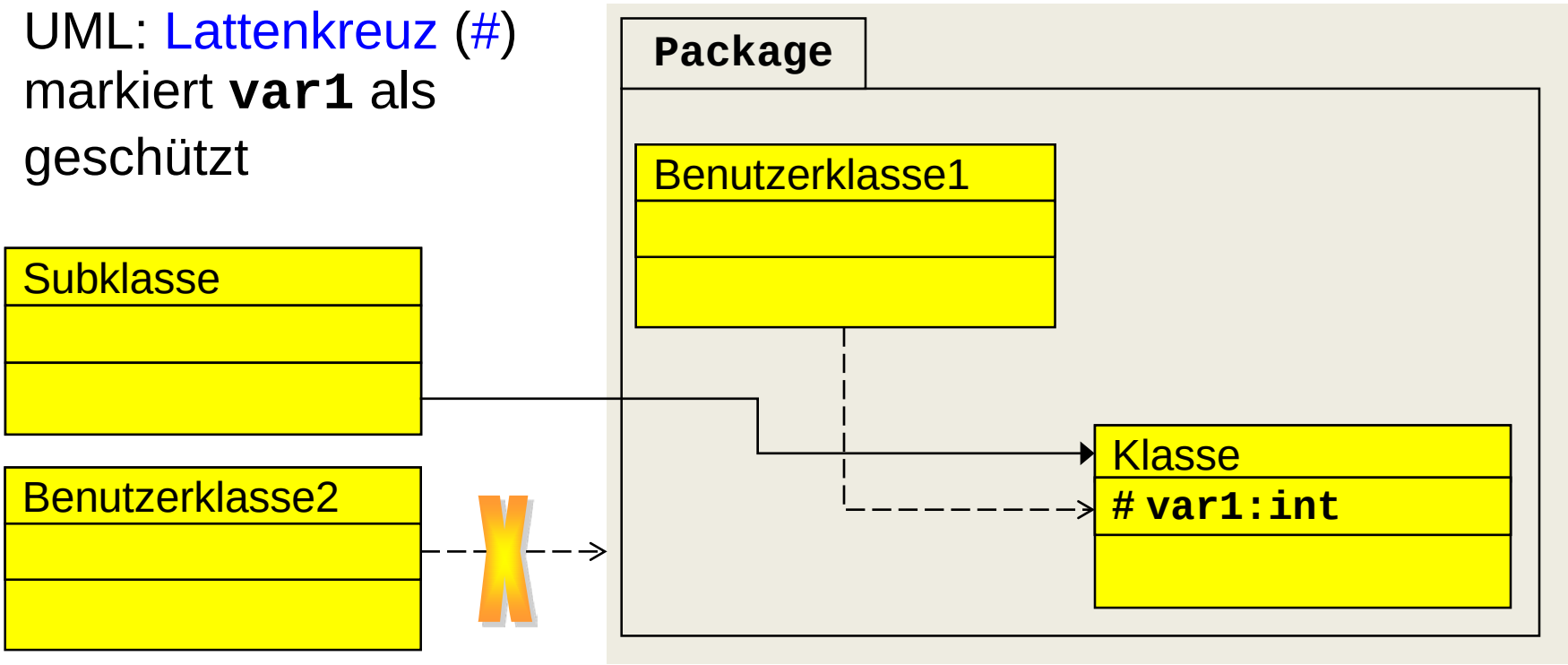
Zugriffsrechte unter Klassen

Java

`protected` (geschützt): Kann nur innerhalb des Pakets, in dem sich die Klasse befindet verwendet werden oder von Subklassen der Klasse

Subklassen sind Klassen mit einer Vererbungsbeziehung (→ VL 13)

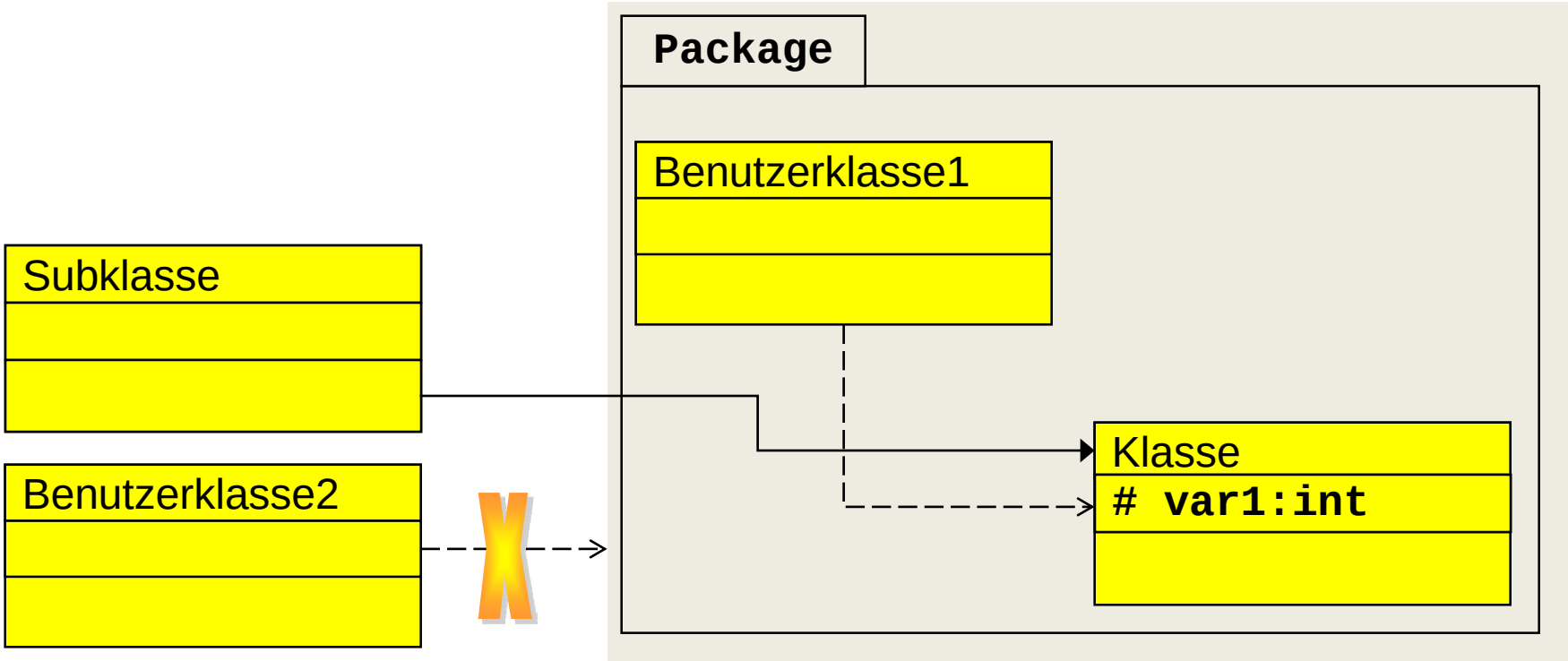
UML: Lattenkreuz (#) markiert **`var1`** als geschützt



Zugriffsrechte unter Klassen

Java

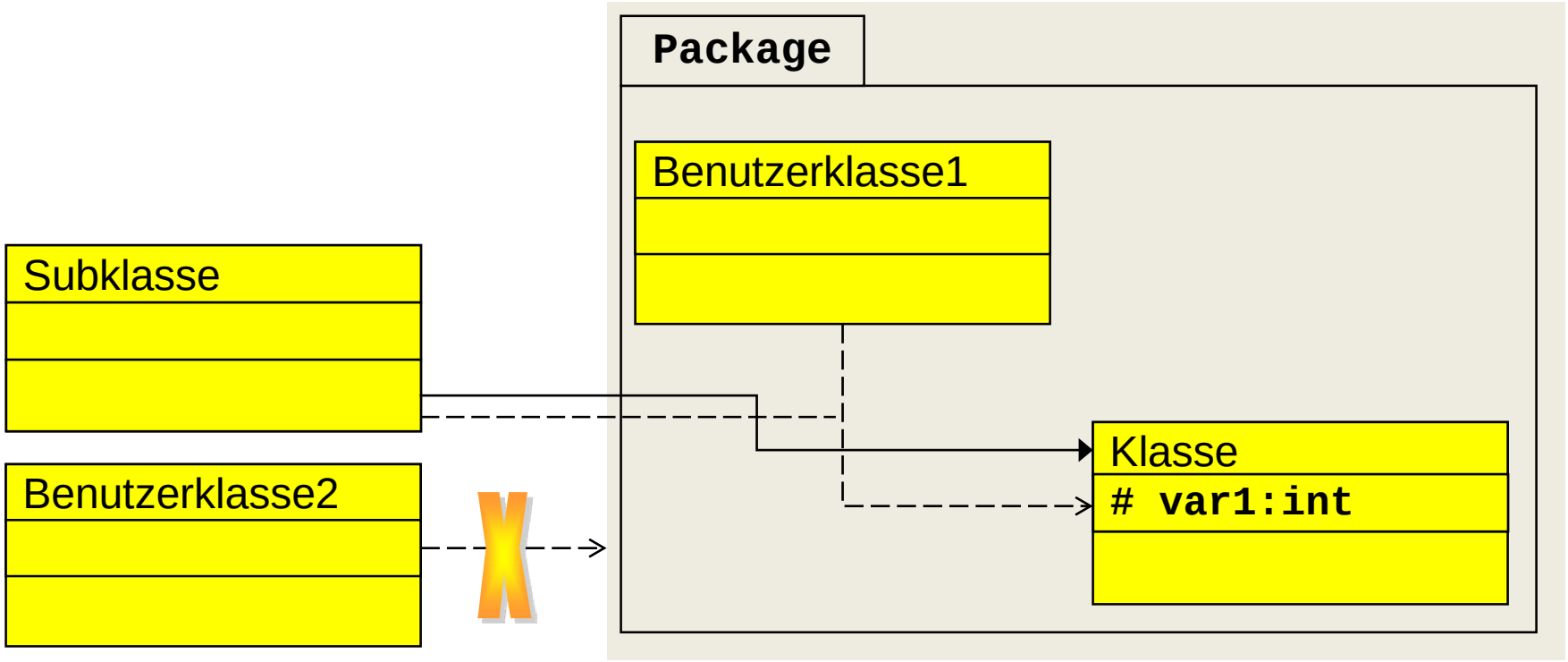
Der Modus **`protected`** lässt sich als eine Erweiterung des Standardmodus **`package`** betrachten. Neben Klassen, die sich in demselben Paket befinden, haben hier auch alle Subklassen Zugriff auf ihre Superklasse.



Zugriffsrechte unter Klassen

Java

In unserem Klassenmodell steht **`Subklasse`** in einer „ist-ein“-Beziehung zur **`Klasse`**. Sie erhält somit ebenfalls Zugriff auf die Variable **`var1`**



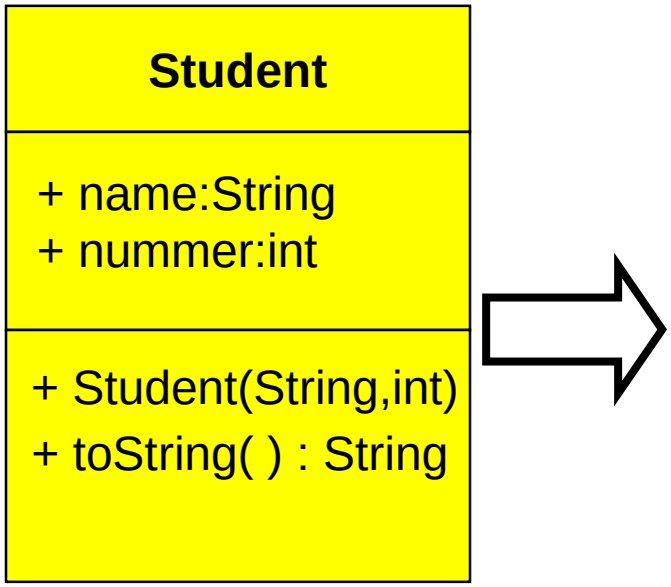
Zugriffsrechte unter Klassen

Java

Wir wollen nun alle Bestandteile von **`Student`** als **`public`** deklarieren

Dazu fügen wir ein:

- **`public`** in der Variablendeklaration / Methodensignatur
- + im UML-Diagramm

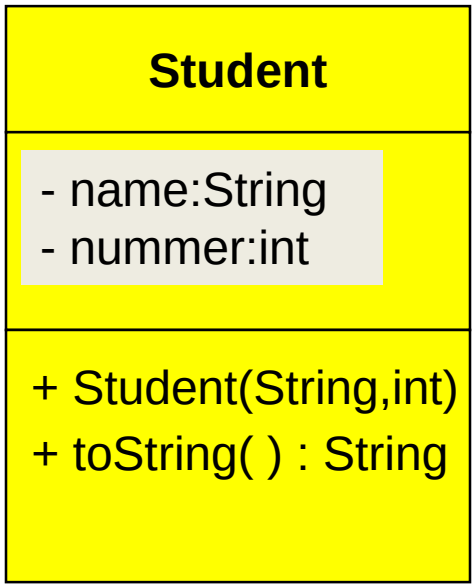



```
public class Student {

  public  String name;
  public  int nummer;

  public  Student(String n, int nr)
   {...}

  public  String toString()
   {...}
}
```

Zugriffsrechte unter Klassen

Java

Gemäß dem Prinzip des **data hiding** wollen wir unsere Instanzvariablen vor einem Zugriff von außen schützen. Wir ändern die Rechte für **`name`** und **`nummer`** deshalb in „privat“ um.



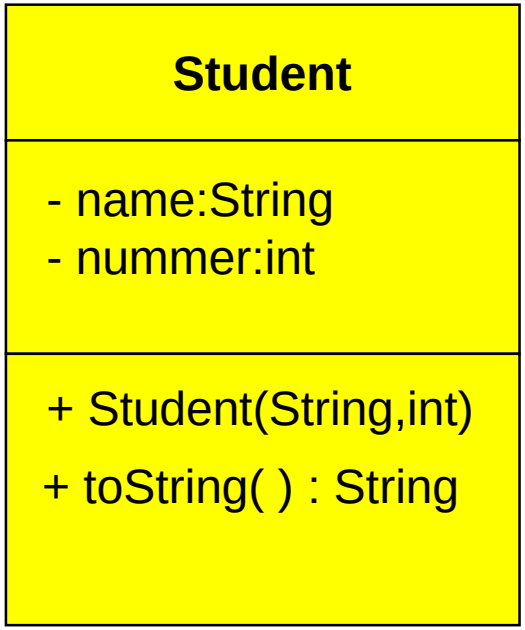

```
public class Student {

  private  String name;
  private  int nummer;

   public Student(String n, int nr)
   {...}

   public String toString()
   {...}
}
```

Zugriffsrechte unter Klassen

Java

Unsere Daten sind nun vor unbefugten Zugriffen „geschützt“.

Problem: Wie kann eine andere Klasse Name und Matrikelnummer erfragen?



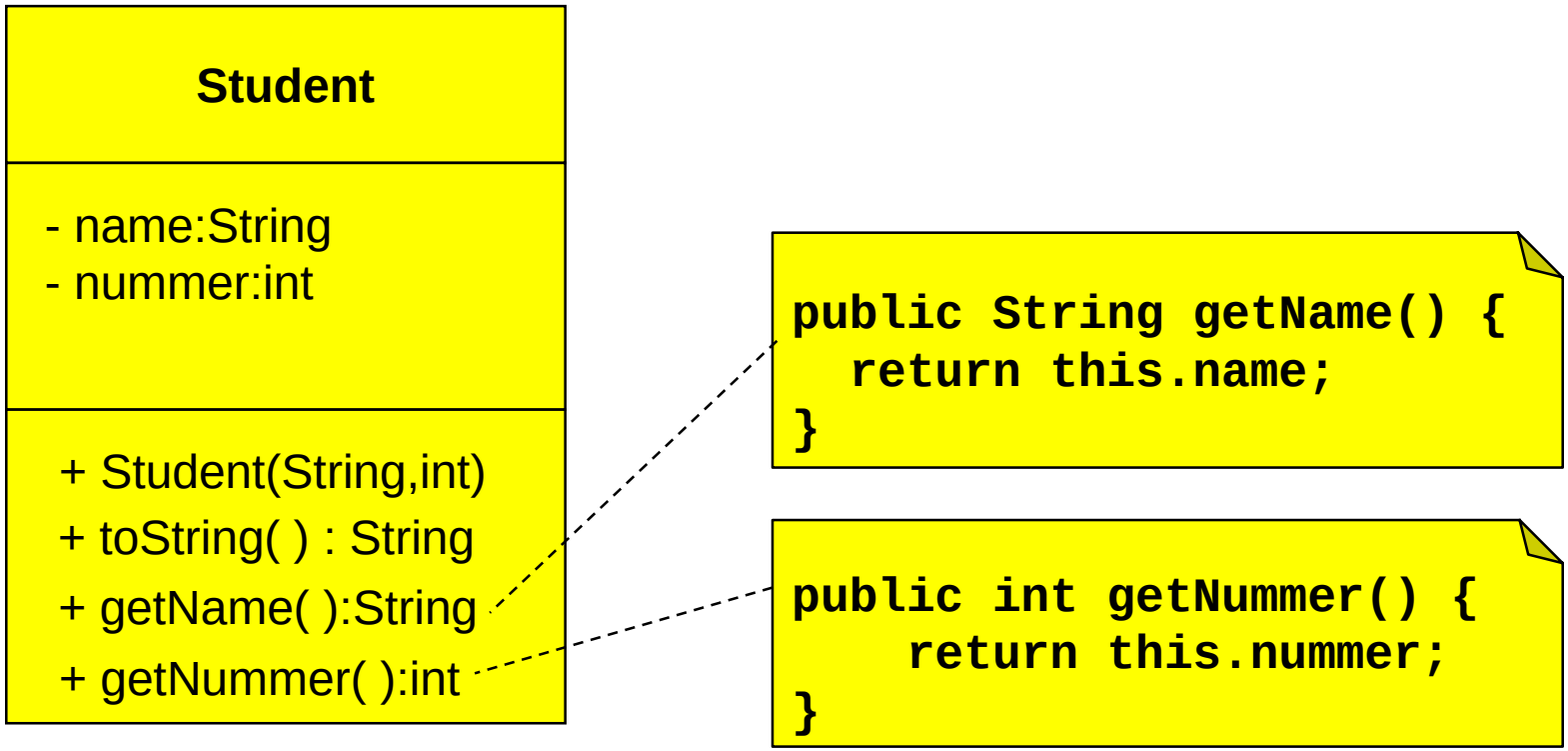

```
public class Student {

  private String name;
  private int nummer;

  public Student(String n, int nr)
  {...}

  public String toString()
  {...}
}
```

Zugriffsrechte unter Klassen

Java

Zugriff erreichen wir mit Instanzmethoden

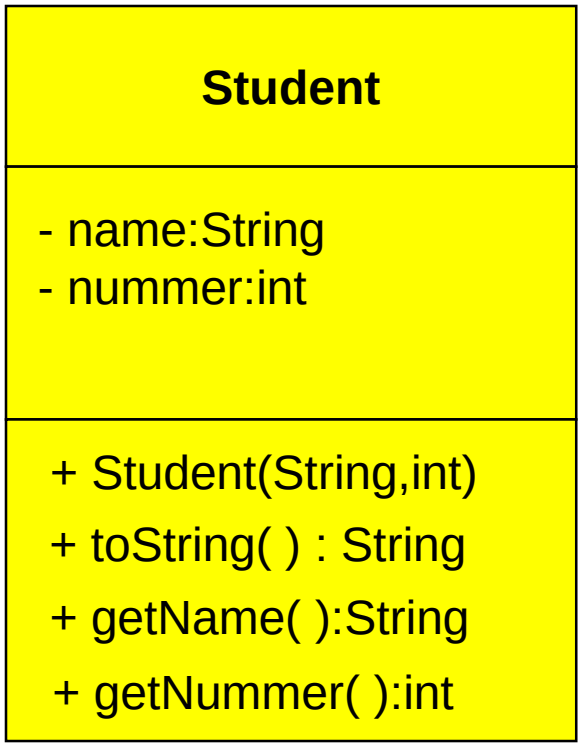
- **`getName`**
- **`getNummer`**



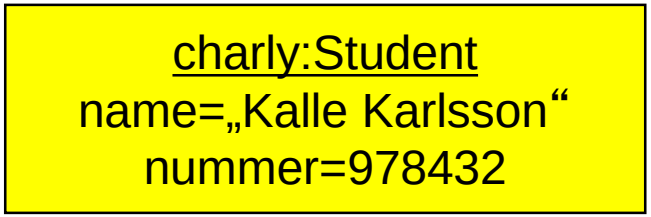
Zugriffsrechte unter Klassen

Java

Aufruf der Instanzmethode **`charly.getName()`**, um den Namen zu erhalten




```
String charlysName = charly.getName();
System.out.println(charlysName);
```



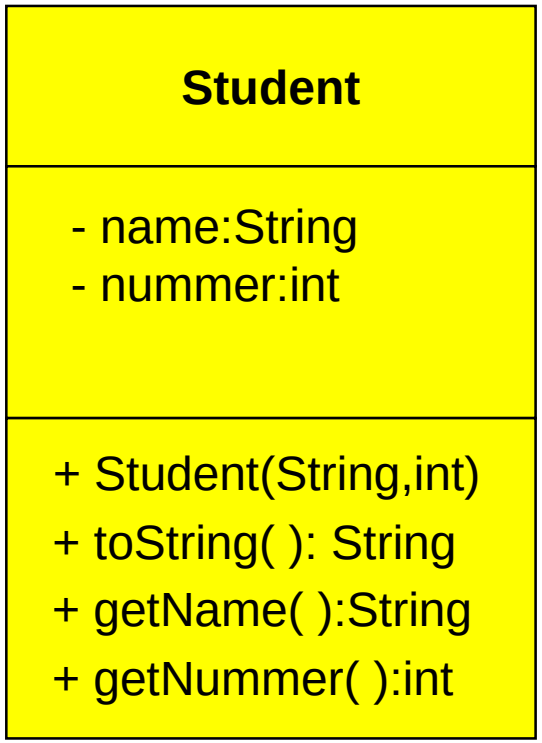
Inhalt

- Zugriffsrechte unter Klassen
- **Klassenvariablen und –methoden**

Klassenvariablen und -methoden

Java

Wir erweitern die Studentenkartei für den Gebrauch an mehreren Universitäten



Problem:

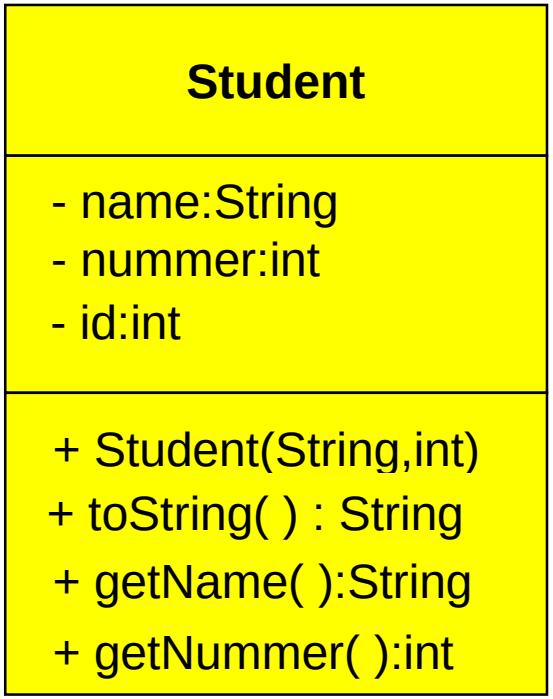
Name und Matrikelnummer sind nur in einer Universität eindeutig. Wir haben keine Garantie dafür, dass es nicht beispielsweise an den Universitäten Würzburg und München zwei verschiedene Studenten namens „Kalle Karlsson“ mit Matrikelnummer 978432 gibt.

Wie sollen wir diese Studenten voneinander unterscheiden?

Klassenvariablen und -methoden

Java

Zwei Objekte sind gleich, wenn sie **das selbe Objekt** sind (**`charly1==charly2`**). Dies ist nur genau dann der Fall, wenn ihre Referenzen auf die gleiche Stelle zeigen.



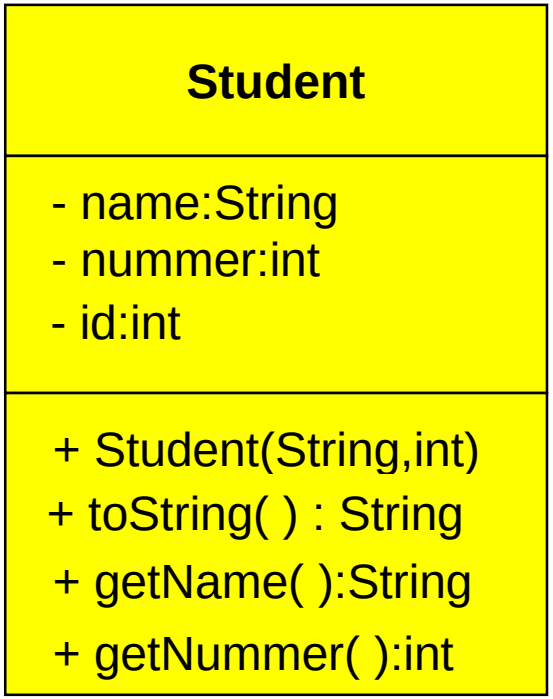
Jedes **`Student`**-Objekt soll mit einer eindeutigen Identifikationsnummer (**`id`**) versehen werden. Die Nummer wird im Konstruktor gesetzt und kann zu Lebzeiten des Objektes nie wieder verändert werden.

Ziel: Zwei Objekte sollen als gleich gelten, wenn sie dieselbe Nummer besitzen.

Klassenvariablen und -methoden

Java

Problem: Wie können wir die Instanzvariable **`id`** mit einem Wert belegen, der für jede Instanz der Klasse **`Student`** unterschiedlich ist? Woher soll der Konstruktor wissen, welche Werte schon vergeben wurden?




```
...
private int id;

public Student(String n,int nr) {
  this.name=n;
  this.nummer=nr;
  this.id = ???
}
...
```

Klassenvariablen und -methoden

Java

Der Konstruktor unserer Klasse **`Student`** benötigt Informationen darüber, welche Nummern bereits für andere Objekte vergeben wurden

Allgemeiner: Wie speichern wir eine Information, die nicht zu einem speziellen Objekt, sondern zu der gesamten Klasse gehört?

Klassenvariablen und -methoden

Java

Klassenvariablen und Klassenmethoden:

- Existieren mit der Klasse
- Werden nicht mit jedem Objekt neu erzeugt
- Werden auch statische Variablen/Methoden genannt
- Werden mit dem Schlüsselwort **`static`** deklariert

Der Zugriff auf statische Teile einer Klasse erfolgt in der Form

<Klassenname>.<Methoden-/Variablenname>

oder in der Form

<Objektname>.<Methoden-/Variablenname>

wobei <Objektname> der Name einer Instanz der Klasse ist

Klassenvariablen und -methoden

Java

Konstruktor unserer Klasse **`Student`**:

Student
- name:String - nummer:int - id:int
+ Student(String,int) + toString():String + getName():String + getNummer():int

```
...
private int id;


public Student(String n, int nr) {
  this.name = n;
  this.nummer = nr;
  this.id = ???

}
...
```

Klassenvariablen und -methoden

Java

Wir speichern die nächste freie ID in einer Klassenvariable namens **`zaehler`**

UML: statische Elemente werden unterstrichen

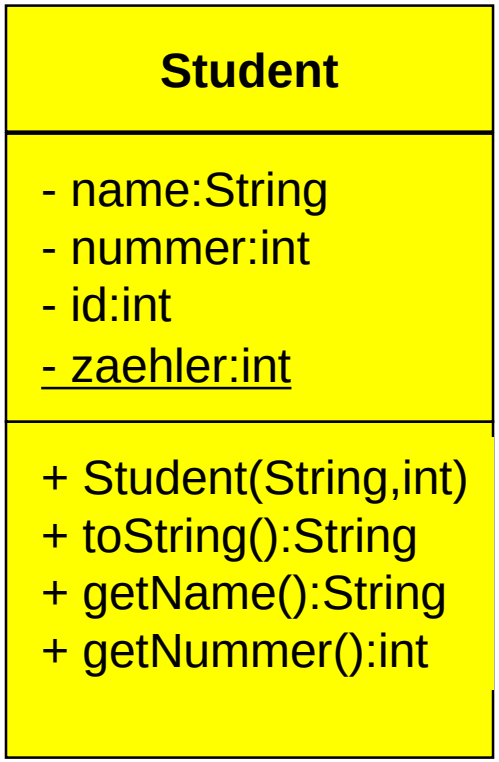
Student
- name:String - nummer:int - id:int **- zaehler:int**
+ Student(String,int) + toString():String + getName():String + getNummer():int

```
...
private int id;


public Student(String n, int nr) {
  this.name = n;
  this.nummer = nr;
  this.id = ???

}
...
```

Klassenvariablen und -methoden

Java

- Statische Klassenvariable **`zaehler`**
- **`id`** wird mit dem Wert von **`zaehler`** initialisiert
- Danach wird der **`zaehler`** inkrementiert

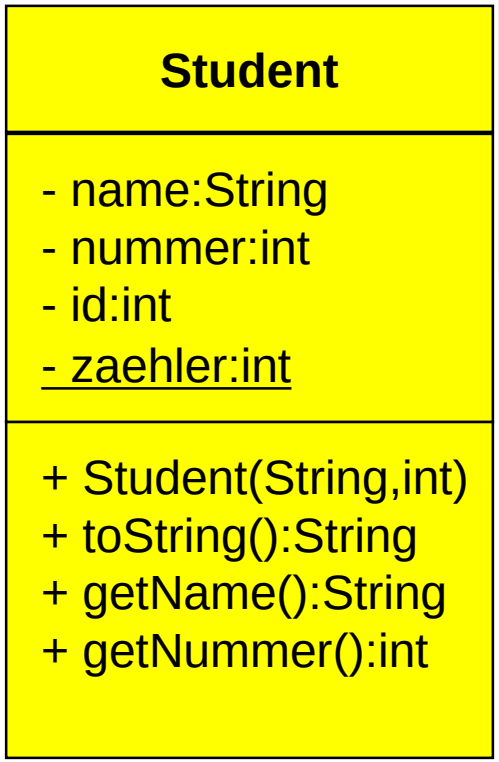



```
...
private int id;
private static int zaehler = 1;

public Student(String n,int nr) {
  this.name=n;
  this.nummer=nr;
  this.id = Student.zaehler;
  Student.zaehler++;
}
...
```

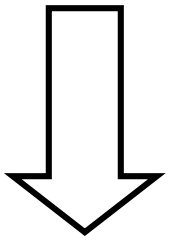
Klassenvariablen und -methoden

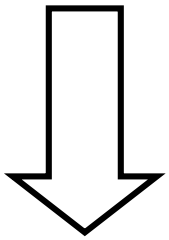
Java

Die **`ids`** sind unterschiedlich, obwohl wir an dem Aufruf des Konstruktors keine Veränderungen vorgenommen haben!




```
Student charly = new Student
                   ("Kalle Karlsson", 978432);
Student lena   = new Student
                   ("Lena Lustig", 888808);
```





charly:Student
name=„Kalle Karlsson“
nummer=978432
id=7

lena:Student
name=„Lena Lustig“
nummer=888808
id=8

Klassenvariablen und -methoden

Java

Die **`ids`** sind unterschiedlich, obwohl wir an dem Aufruf des Konstruktors keine Veränderungen vorgenommen haben!

Data hiding!

- Interne Darstellung unserer Klasse wurde verändert
- alle bisher verfassten Programme lassen sich weiterhin verwenden

Weil wir die interne Struktur versteckt und dem Benutzer nur über eine separate Schnittstelle zugänglich gemacht haben!

Klassenvariablen und -methoden

Java

Wichtige Regel zur Verwendung statischer Methoden:

```
public static void main(String[] args) {
  ...
}

public static int fakultaet(int n) {
  if (n==0) return 1;
  else return n*fakultaet(n-1);
}

public static int tuwas(double d) {
  ...
}
```

Klassenvariablen und -methoden

Java

Wichtige Regel zur Verwendung statischer Methoden:

Statische Methoden haben keinen Zugriff auf nicht-statische Elemente!

Klassenvariablen und -methoden

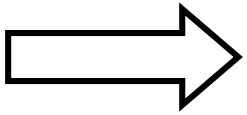
Java

Statische Methode: **`letzteID`** soll die letzte vergebene Identifikationsnummer zurückliefern

Student
- name:String - nummer:int - id:int - zaehler:int
+ Student(String,int) + toString():String + getName():String + getNummer():int **+ letzteID():int**

`zaehler` ist statisch → **`letzteID`** kann darauf zugreifen

Der Klassenname (**`Student`**) könnte auch entfallen, weil wir uns in der Klasse **`Student`** befinden



```
public static int letzeID() {
  return Student.zaehler - 1;
}
```

Klassenvariablen und -methoden

Java

Die statische Methode **schreibeName** liefert folgenden Compilerfehler:

```
Can´t make a static reference
        to a nonstatic variable
```

Student
- name:String - nummer:int - id:int - zaehler:int
+ Student(String,int) + toString():String + getName():String + getNummer():int + letzteID():int

Grund: Zugriff auf Instanzvariable **name**!

Wir befinden uns aber in keiner Instanz (also einem Objekt) der Klasse **Student**, d.h. die Variable existiert nicht!

```
public static void schreibeName() {
  System.out.println(name);
}
```

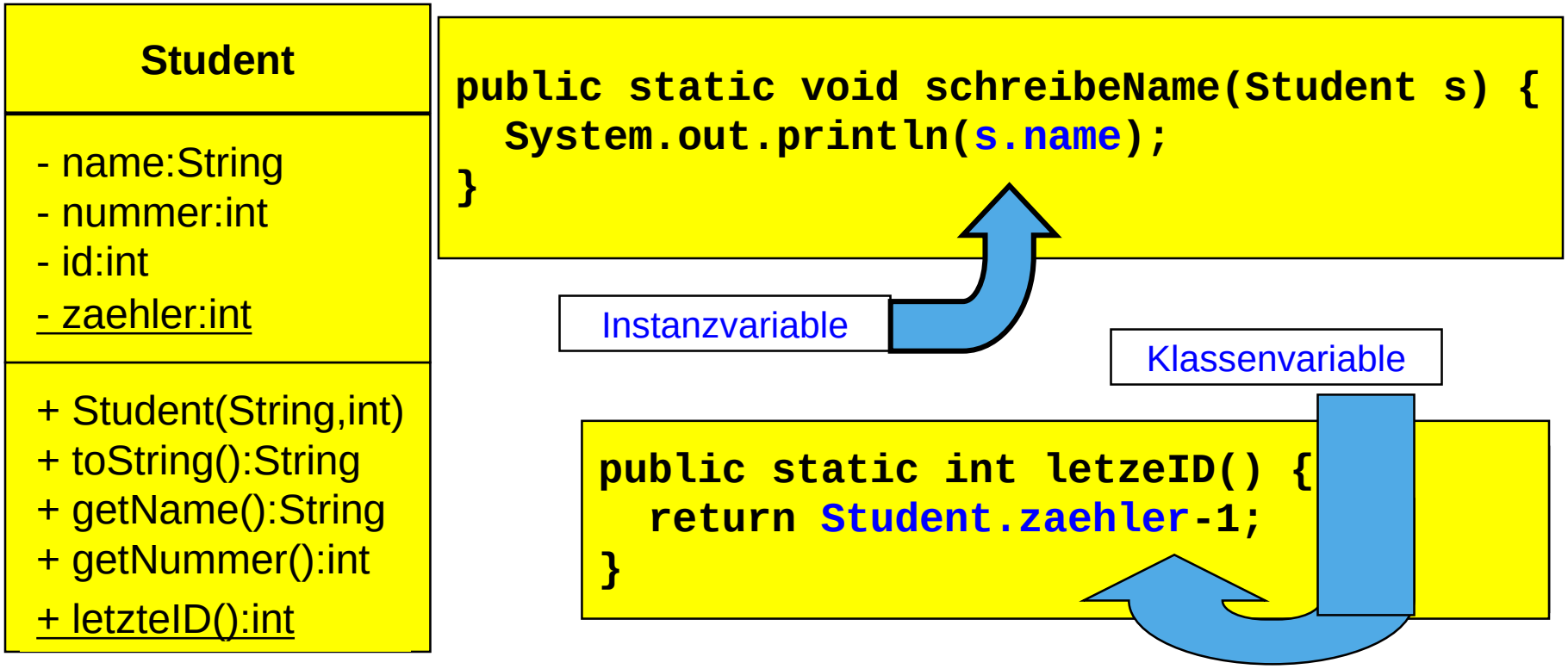
Klassenvariablen und -methoden

Java

Beispiel

Zugriffe auf **Instanzvariablen** benötigen ein Objekt!

Zugriffe auf **Klassenvariablen** benötigen **kein** Objekt!



Fragen?

SOFTWARE TERMINOLOGY

Then	Now
application	app
program	app
operating system	app
script	app
shell	app
batch file	app
compiler	app
daemon	app
service	app
game	app
patch	app
software	app

Danksagung

- Vorlesungsmaterialien von Prof. Dr. Detlef Seese wurden als Basis verwendet
- Unterstützung bei der technischen und inhaltlichen Gestaltung des Vorlesungsmaterials leisteten:

  Jóakim v. Kistowski

  Dietmar Ratz, Joachim Melcher, Roland Küstermann, Jana Weiner, Hagen Buchwald, Matthes Elstermann, Oliver Schöll, Niklas Kühl, Tobias Diederich