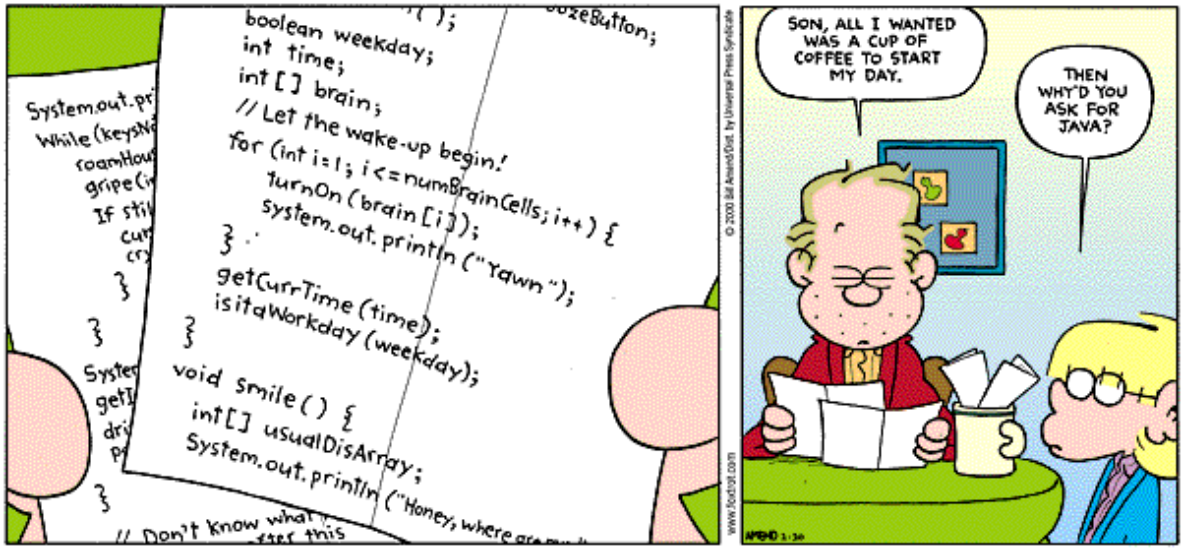
Grundlagen der Programmierung

VL11: Klassen – Teil 1

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



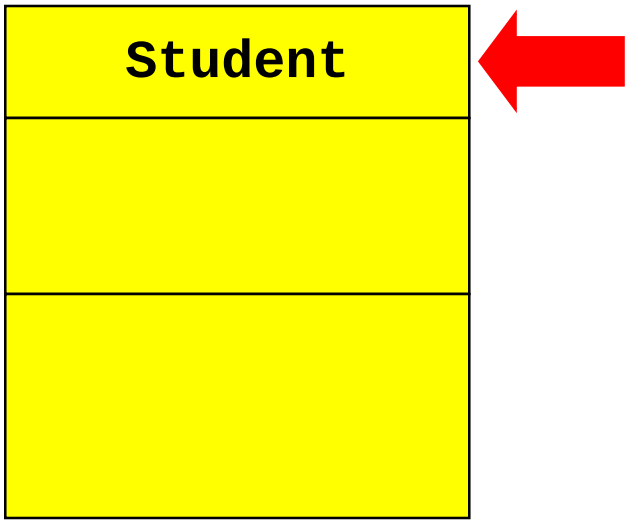
Inhalt

- **Instanzvariablen**
- Konstruktoren
- Instanzmethoden

Unsere erste Klasse

Java

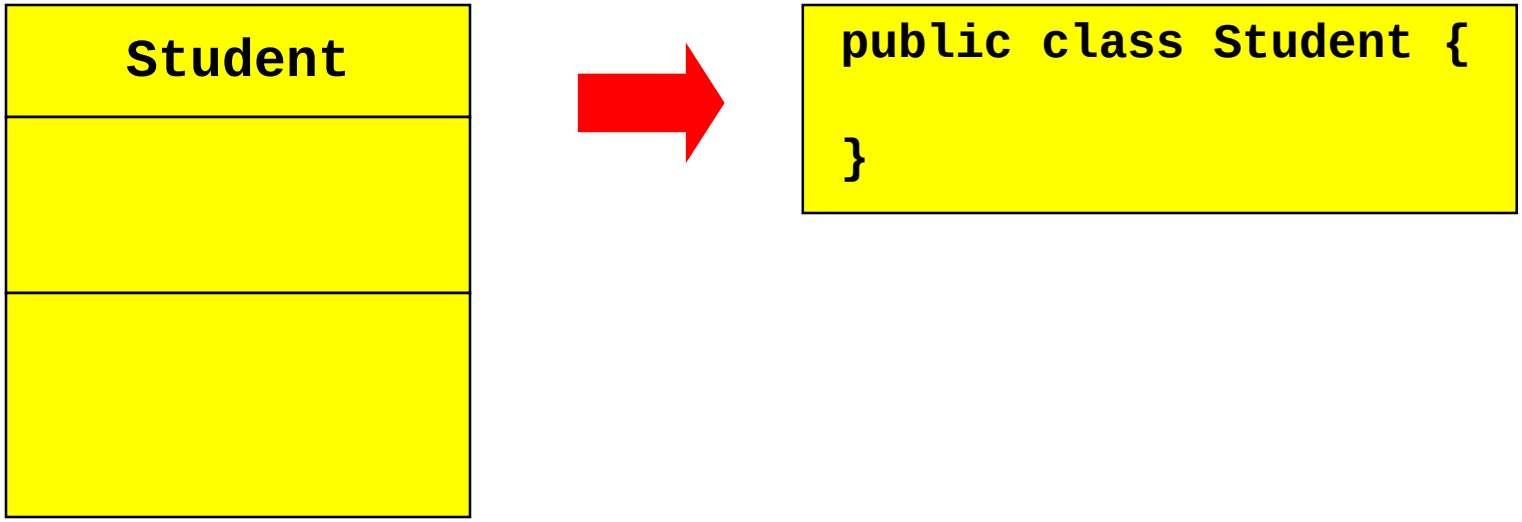
- Wir werden nun unsere erste Klasse von Objekten modellieren: die Klasse **`Student`**
- Zu diesem Zweck werden wir auf der einen Seite stets ein graphisches Modell der Klasse in UML betrachten...



Unsere erste Klasse

Java

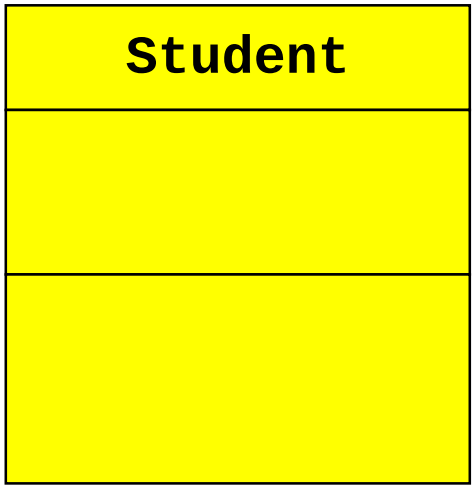
- ... und auf der anderen Seite einen Blick auf die entsprechende Realisierung in Java werfen



Unsere erste Klasse

Java

- Unser zweizeiliger Java-Code stellt bereits eine gültige Klassenbeschreibung dar
- Wir können die Klasse bereits übersetzen und **instantiieren** - also Objekte aus ihr erzeugen

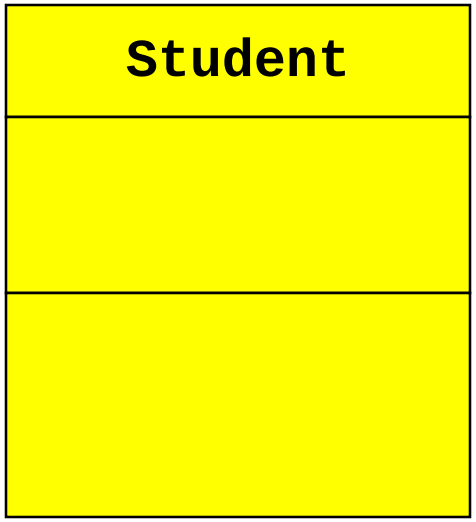



```
public class TestProgramm {
  // Meine main-Methode
  public static void main(String[] args) {
    // Erzeuge ein Objekt
    Student charly = new Student();
  }
}
```

Unsere erste Klasse

Java

- Die Anweisung innerhalb der **`main`**-Methode ist für uns nicht neu
- Sie dient der **Instantiierung**, erzeugt also ein **Objekt**



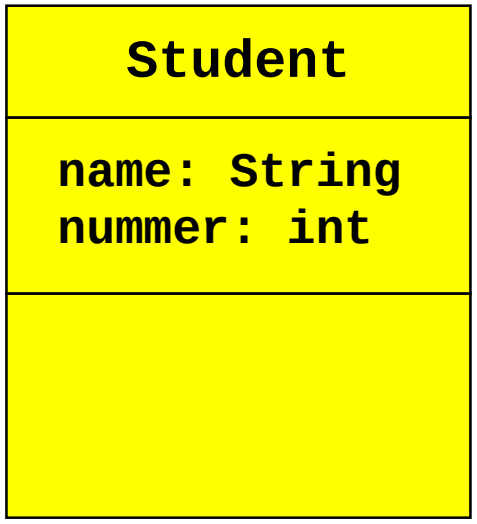

```
public class Student {

}
```

```
Student charly = new Student();
```

Instanzvariablen

Java

- Unsere Klasse soll in der Lage sein, Name und Matrikelnummer eines Studenten im zugehörigen Objekt zu hinterlegen
- Wir tragen unsere Erweiterungen in das UML-Klassendiagramm ein - hierbei stehen Daten immer im oberen der beiden Rechtecke

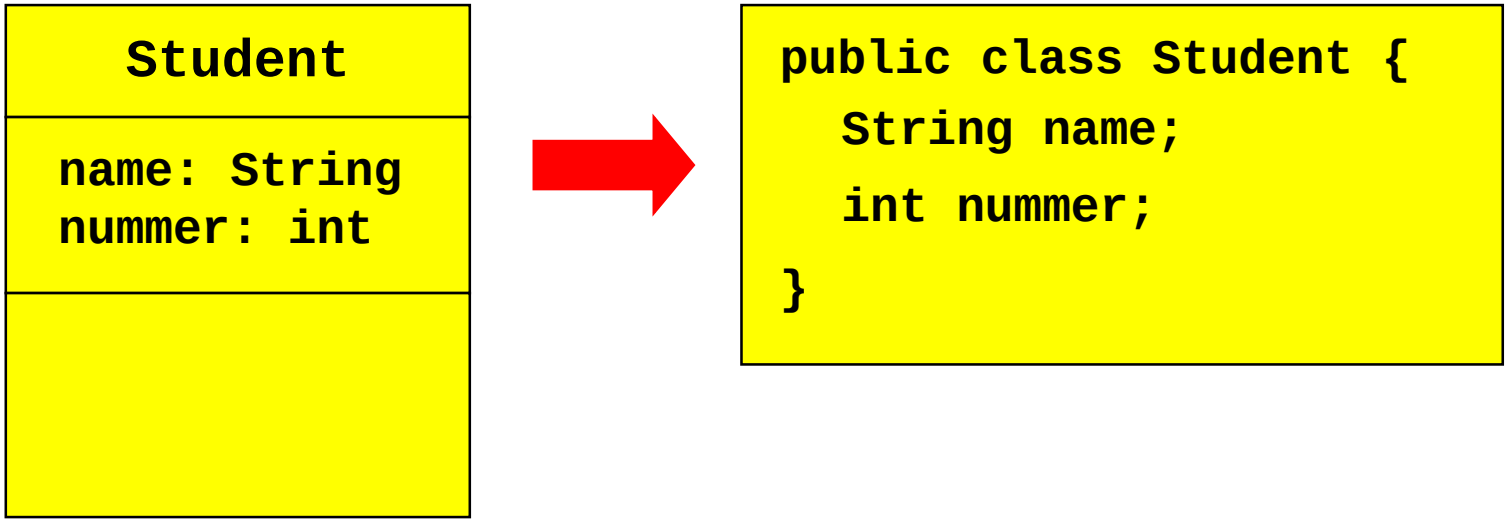



```
public class Student {



}
```

Instanzvariablen

Java

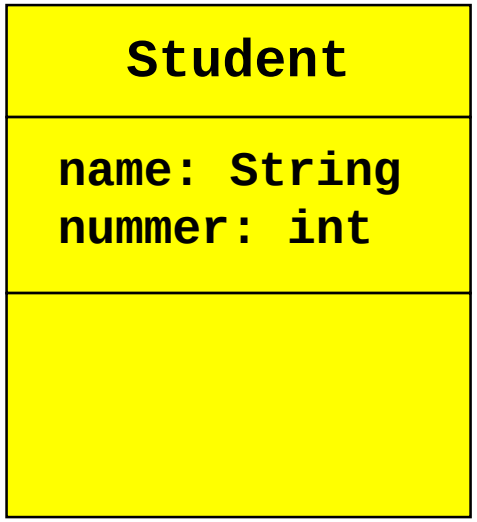
- Die so definierten Variablen werden beim Erzeugen eines neuen **`Student`**-Objekts für dieses Objekt automatisch miterzeugt und können vom Benutzer individuell belegt werden
- Sie gehören also der jeweiligen Instanz und werden deshalb als **Instanzvariablen** bezeichnet



Instanzvariablen

Java

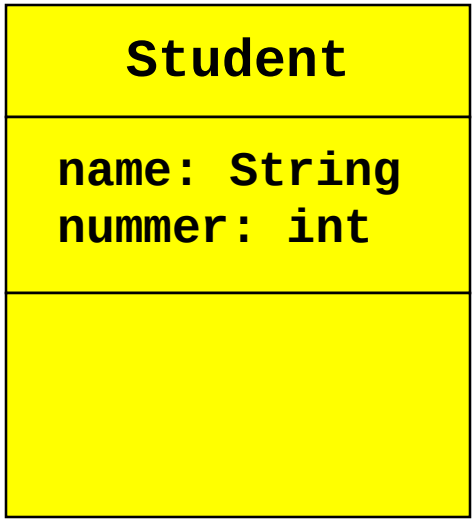
Wir wollen in unserem „Testprogramm“ einen Studenten namens „Kalle Karlsson“ mit der Matrikelnummer **978432** erstellen. Der Variablenname des Objekts soll hierbei **`charly`** lauten.



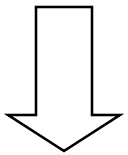
Instanzvariablen

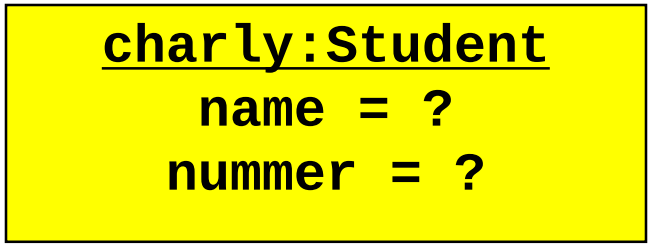
Java

Der Zugriff auf Instanzvariablen erfolgt in Java in der Form
<Objektname>.<Variablenname>




```
Student charly = new Student();
```

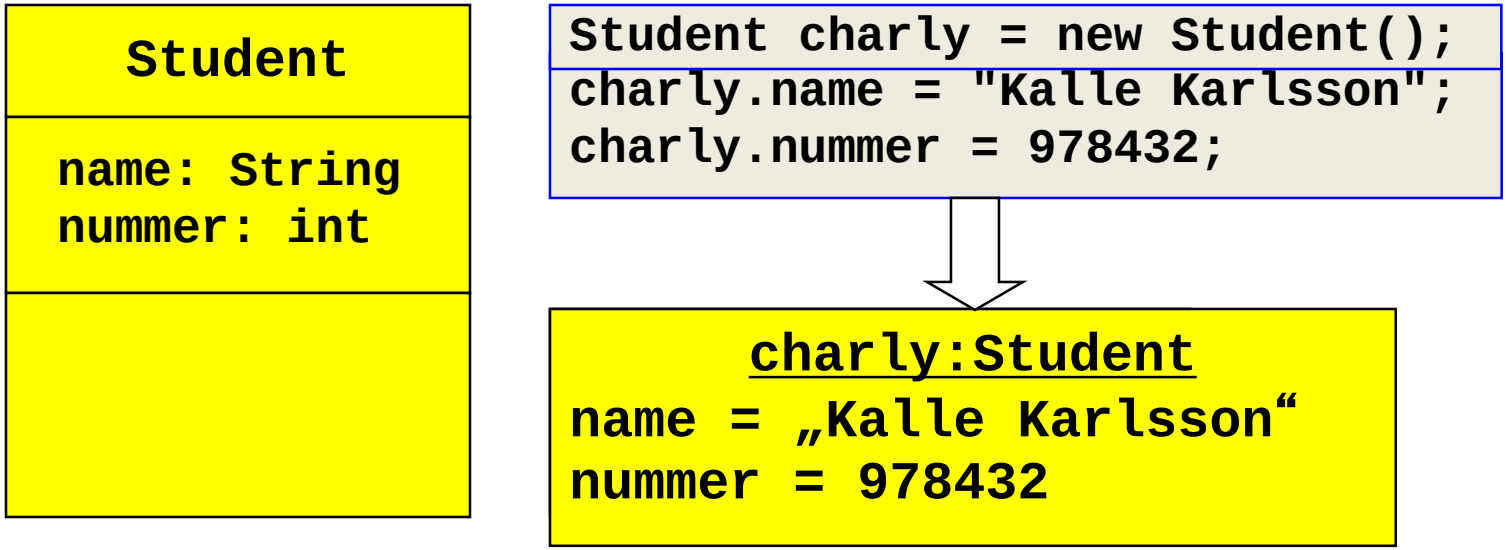




Instanzvariablen

Java

Wir können die Werte in **`name`** und **`nummer`** des Objekts **`charly`** durch einfache Wertzuweisungen manipulieren



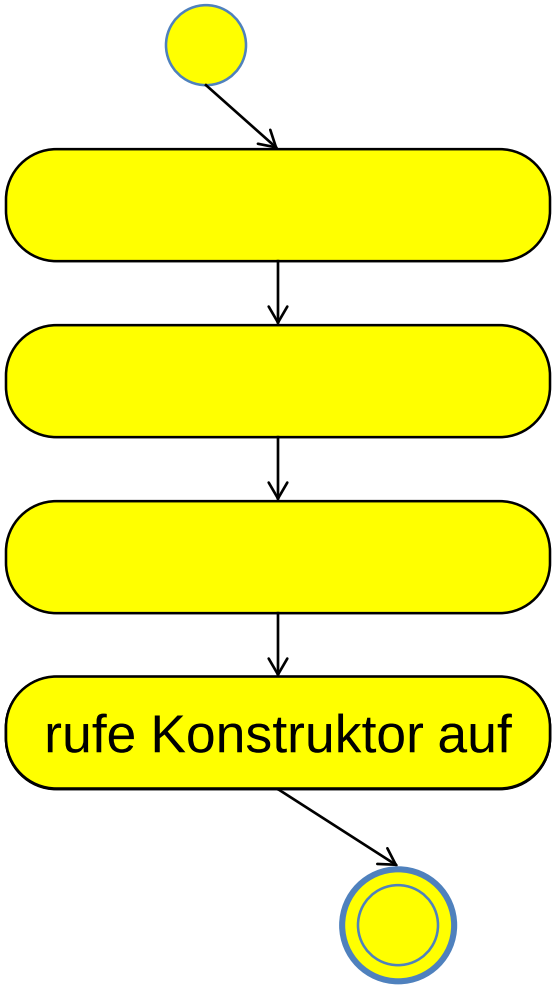
Inhalt

- Instanzvariablen
- **Konstruktoren**
- Instanzmethoden

Konstruktoren

Java

- Auch in Java fallen Objekte nicht einfach „vom Himmel“, sondern werden nach einem fest vorgegebenen Schema erstellt

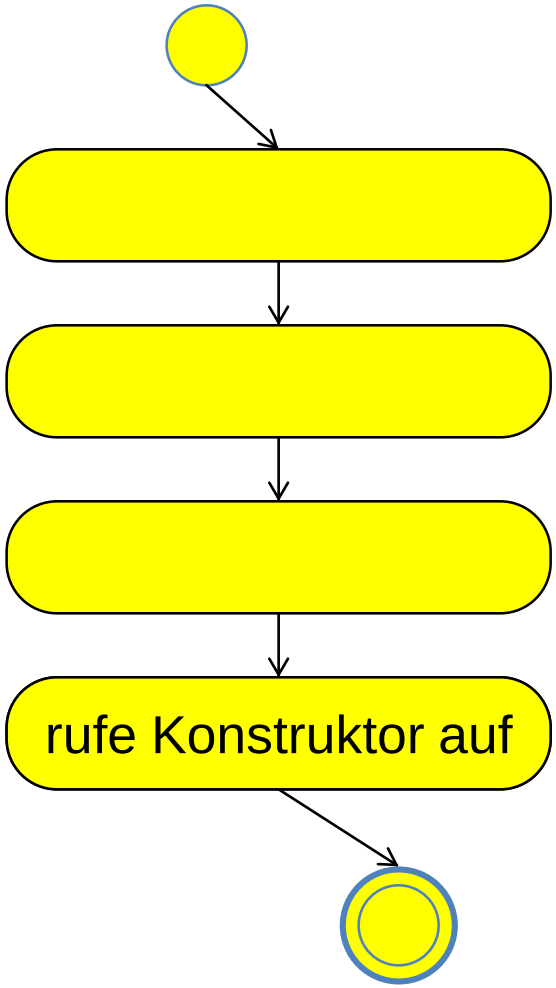


Wir wollen an dieser Stelle nicht auf jede einzelne dieser Aktivitäten eingehen; für uns ist lediglich der letzte Schritt von Interesse:

der **Aufruf des Konstruktors**

Konstruktoren

Java



- Konstruktoren kommen erst am Ende eines Instantiierungsprozesses ins Spiel. Das Objekt ist prinzipiell schon erzeugt: Speicher wurde belegt, Instanzvariablen vereinbart, gewisse Initialisierungen bereits vorgenommen...
- Es ist nun Aufgabe des Konstruktors, Initialisierungsarbeiten zu erledigen, die das System nicht „automatisch“ durchführen kann
- *Hierzu zählt unter anderem das Belegen von Instanzvariablen mit vom Benutzer vorgegebenen Startwerten*

Konstruktoren

Java

- Werfen wir erneut einen Blick auf unsere Klasse **`Student`**
- In unserem momentanen Entwicklungsstadium haben wir in der Klasse noch keinen Konstruktor definiert. *Das System benötigt jedoch unbedingt einen Konstruktor, um Objekte bilden zu können!*

```
public class Student {
  String name;
  int nummer;



}
```

- Damit also Java dennoch in der Lage ist, die Klasse zu instantiieren, wird bei der Übersetzung vom Compiler in der Klassendatei automatisch ein „leerer Konstruktor“ angefügt, dessen einzige Aufgabe es ist, „nichts“ zu tun. Wir bezeichnen diesen Konstruktor in Zukunft als den **Standard-Konstruktor** bzw. den **Default-Konstruktor**

Konstruktoren

Java

- Würden wir den „leeren“ Default-Konstruktor in unseren Quellcode aufnehmen, hätte dieser die folgende Gestalt:

```
public class Student {
  String name;
  int nummer;

  Student() {
  }


}
```

- Es fällt eine gewisse Ähnlichkeit mit der Vereinbarung von Methoden auf. Wir stellen jedoch fest, dass
  - der Name des Konstruktors kein beliebiger Bezeichner sondern der Name der Klasse ist
  - kein Rückgabetyp angegeben wurde (nicht einmal void)

Konstruktoren

Java

- Kommen wir zurück zu unserer Klasse **`Student`**
- Anstelle des Default-Konstruktors wollen wir uns einen eigenen Konstruktor definieren
- Diesem wollen wir Namen und Matrikelnr. eines Studenten als Argumente übergeben können

```
public class Student {
  String name;
  int nummer;

  Student(String n, int nr) {


  }
}
```

- In diesem Fall ist also der Name der Klasse wieder

  `<Klassenname>  = Student`

- und die Liste der Argumente nicht leer, sondern

  `<Argumentliste> = String n, int nr`

Konstruktoren

Java

- Erinnern wir uns nun daran, wie wir unsere Objekte bisher mit Werten initialisiert haben
- Um die einzelnen Werte setzen zu können, haben wir uns eines Zugriffs der Form ***<Objektname>.<Variablenname>*** bedient
- **Wie** sollen wir aber innerhalb des Konstruktors wissen, ob unser Objekt nun **`charly`**, **`lena`** oder **`lassy`** heißt?

```
public class Student {
  String name;
  int nummer;

  Student(String n, int nr) {


  }

}
```

```
Student charly = new Student();
charly.name = "Kalle Karlsson";
charly.nummer = 978432;
```

Konstruktoren

Java

- Innerhalb eines Objektes können wir das Objekt selbst mit dem Schlüsselwort **`this`** ansprechen

```
public class Student {
  String name;
  int nummer;

  Student(String n, int nr) {


  }

}
```

Der Zugriff auf Instanzvariablen innerhalb des Konstruktors erfolgt somit in der Form
this.<*Variablenname*>

Konstruktoren

- Wollten wir also innerhalb des Konstruktors die Instanz-variablen **name** und **nummer** auf „Kalle Karlsson“ und 978432 setzen, so könnten wir dies in der rechts angegebenen Form bewerkstelligen.

```
public class Student {
  String name;
  int nummer;

  Student(String n, int nr) {
    this.name = "Kalle Karlsson";
    this.nummer = 978432;
  }

}
```


```
Student charly = new Student();
charly.name = "Kalle Karlsson";
charly.nummer = 978432;
```

Genau genommen können wir das **this** auch weglassen. Da es im Konstruktor keine lokale Variable **name** oder **nummer** gibt, "weiß" die Methode trotzdem, dass sie die Instanzvariablen des eigenen Objekts ansprechen muss!

Konstruktoren

Java

- Natürlich wollen wir unseren Konstruktor allgemeiner halten und ihm die Werte aus der Argumentliste (die Werte von **n** und **nr**) zuweisen
- Das Vorgehen ist hierbei jedoch vollkommen identisch

```
public class Student {

  String name;
  int nummer;

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

}
```

Konstruktoren

Java

- Natürlich wollen wir unseren Konstruktor allgemeiner halten und ihm die Werte aus der Argumentliste (die Werte von **n** und **nr**) zuweisen
- Das Vorgehen ist hierbei jedoch vollkommen identisch

```
public class Student {

  String name;
  int nummer;

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

}
```

Konstruktoren

Java

- Natürlich wollen wir unseren Konstruktor allgemeiner halten und ihm die Werte aus der Argumentliste (die Werte von **n** und **nr**) zuweisen
- Das Vorgehen ist hierbei jedoch vollkommen identisch

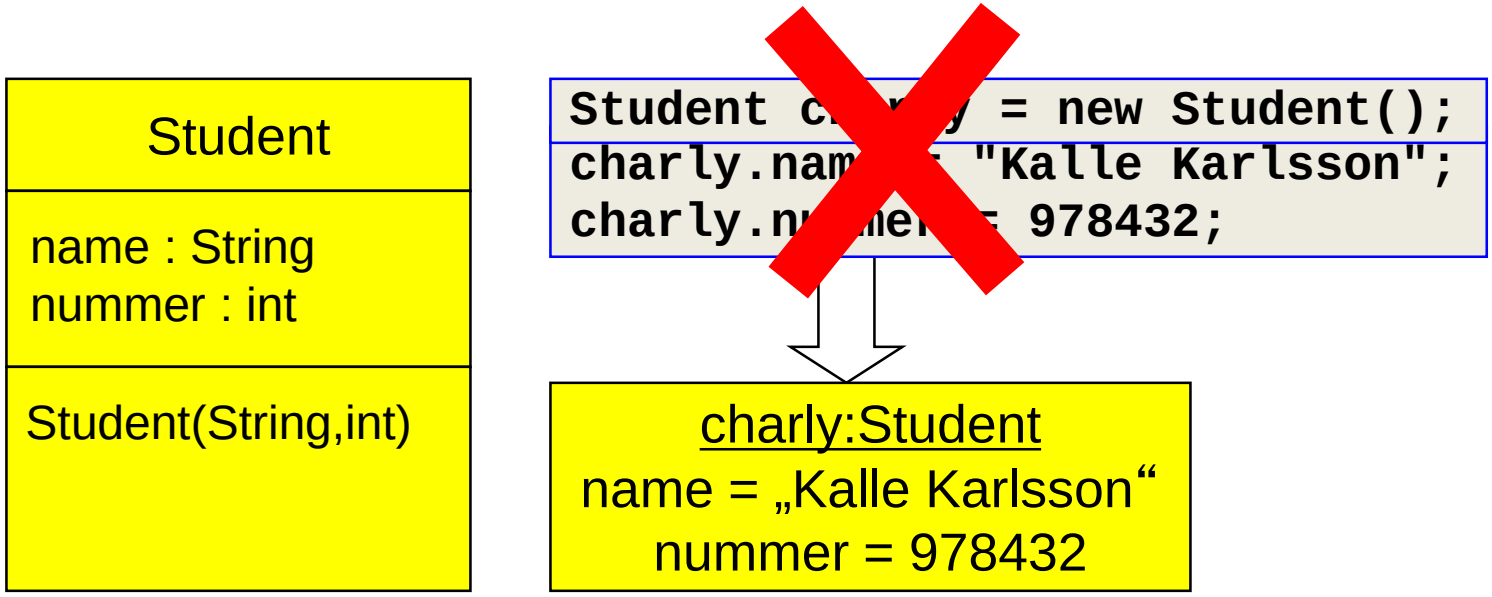
```
public class Student {

  String name;
  int nummer;

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

}
```

Anstelle der Parameter **n** und **nr** könnte der Konstruktor auch die Parameter **name** und **nummer** haben, und alles würde funktionieren. Dann könnten wir jedoch das **this** *nicht* weglassen, da es benötigt wird, um die lokalen Variablen bzw. die Parameter von den Instanzvariablen zu unterscheiden!

Konstruktoren

Java

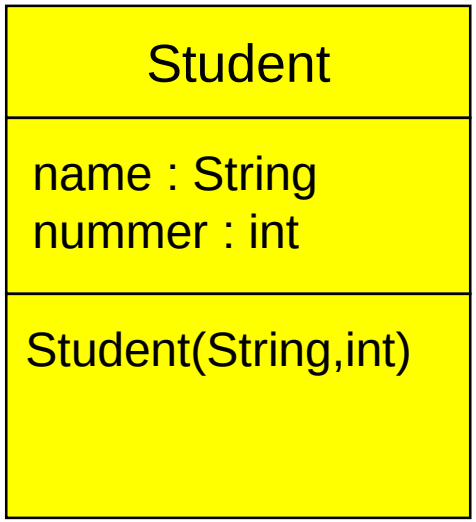
- Hierbei tragen wir den Konstruktor in das untere Feld unseres Klassendiagrammes ein
- Wir haben in unserer Klasse einen eigenen Konstruktor definiert. Beim Übersetzen unseres Quelltextes wird der Compiler also keinen Standardkonstruktor mehr einfügen. Der unten angegebene Aufruf des **new**-Operators ist somit also falsch!



Konstruktoren

Java

- Um unser Objekt **`charly`** also zu erzeugen, müssen wir dem Operator die von uns gewünschten Initialwerte als Argumente übergeben. Diese werden dann an den Konstruktor weitergeleitet.



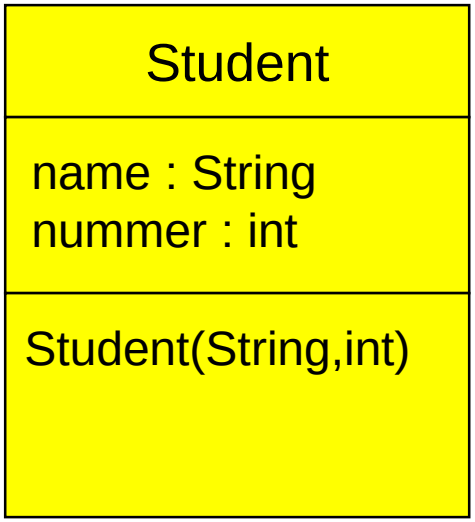

```
Student charly =
  new Student("Kalle Karlsson",978432);
```

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

Konstruktoren

Java

- Wir können also mit Hilfe des Konstruktors nicht nur Objekte erzeugen, sondern diesen auch gleich die gewünschten Startwerte zuweisen. Statt drei Befehlen benötigen wir nun nur noch einen!
- Diese Ersparnis kommt uns beim Programmieren insbesondere dann zugute, wenn wir viele Instanzen unserer Klasse erzeugen wollen.




```
Student charly =
  new Student("Kalle Karlsson",978432);
Student lena =
  new Student("Lena Lustig",888808);
Student rudi =
  new Student("Rudi Rentier",117704);
Student marion =
  new Student("Marion Maus",123456);
Student jessica =
  new Student("Jessica Java",9875632);
```

Konstruktoren

Java

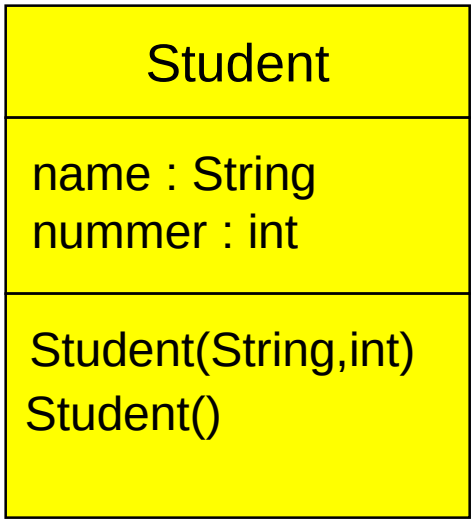
- Ähnlich wie bei Methoden ist es natürlich auch möglich, **Konstruktoren zu überladen**. Wir können mehrere Konstruktoren mit unterschiedlicher Argumentliste definieren, so dass der Benutzer je nach Bedarf zw. verschiedenen Alternativen wählen kann.

Student
name : String nummer : int
Student(String,int) Student()

Konstruktoren

Java

- Ähnlich wie bei Methoden ist es natürlich auch möglich, **Konstruktoren zu überladen**. Wir können mehrere Konstruktoren mit unterschiedlicher Argumentliste definieren, so dass der Benutzer je nach Bedarf zw. verschiedenen Alternativen wählen kann.



- Eine mögliche Anwendung hierfür wäre beispielsweise, den inzwischen verschwundenen Default-Konstruktor zu ersetzen. Wir könnten einen neuen Konstruktor mit leerer Argumentliste definieren, der entweder nichts tut oder die Instanzvariablen auf irgendeinen Standardwert setzt.

Inhalt

- Instanzvariablen
- Konstruktoren
- **Instanzmethoden**

Instanzmethoden

Java

- Eine Instanzmethode ist eine Methode, die zu einer speziellen Instanz einer Klasse (also einem Objekt) gehört
- Instanzmethoden werden (wie auch Instanzvariablen) innerhalb der Klassendefinition vereinbart
- Alle in den vorherigen Lektionen erlernten „Handgriffe“ im Umgang mit Methoden (Überladen, rekursive Definition,...) lassen sich auch auf Instanzmethoden problemlos anwenden

Instanzmethoden

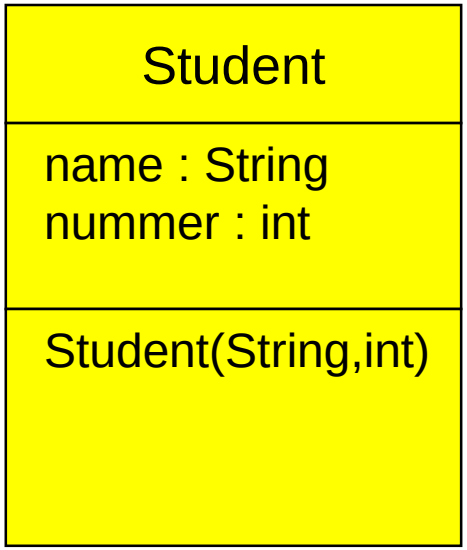
Java

Wir wollen unsere Klasse **`Student`** so erweitern, dass die in ihr gespeicherten Daten in aufbereiteter Form dargestellt werden können.

Hierbei soll diese „Aufbereitung“ einen String als Ergebnis haben, der in der Form

`Student <name> hat die Matrikelnummer <nummer>.`

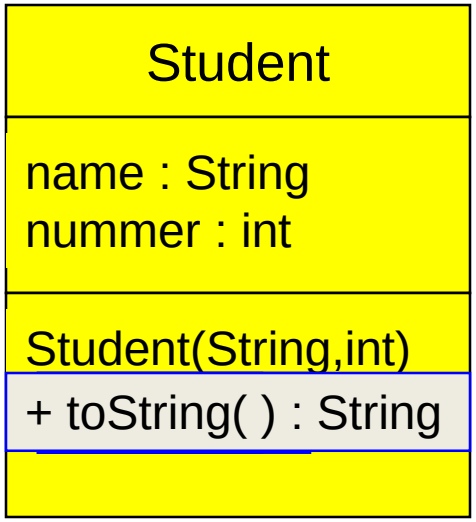
gestaltet sein soll.



Instanzmethoden

Java

- Wir wollen nun diesen Vorgang der „Aufbereitung“ in unser UML-Klassendiagramm eintragen:
- Wir wollen eine derartige Aufbereitung „auf Befehl“ ausführen können und entwerfen sie deshalb als Methode
- In unserem Modell werden Methoden in das untere Feld eingetragen
- Wir geben der Methode den Namen `toString`, da wir unser Studenten-Objekt in eine Darstellung als String umwandeln wollen
- Wir wollen einen String als Ergebnis zurückgeben



Instanzmethoden

Java

Wir wollen die Methode als Instanzmethode entwerfen, das heisst sie gehört zu dem zu bearbeitenden Objekt und hat somit bereits vollen Zugriff auf die Daten. Wir brauchen ihr also keine Argumente zu übergeben.

Student
name : String nummer : int
Student(String,int) + toString( ) : String

```
public class Student {
  ...

  public String toString() {
    return "Student "
           + this.name
           + " hat die Matrikelnummer "
           + this.nummer
           + ".";
  }
} // Ende der Klasse
```

Instanzmethoden

Java

Der Kopf der Methode entspricht im Prinzip den von uns bisher definierten Methoden - mit der Ausnahme, dass wir das **Schlüsselwort `static` weggelassen** haben. Wir werden jedoch erst später erfahren, welche Bedeutung dieses Wort genau hat.

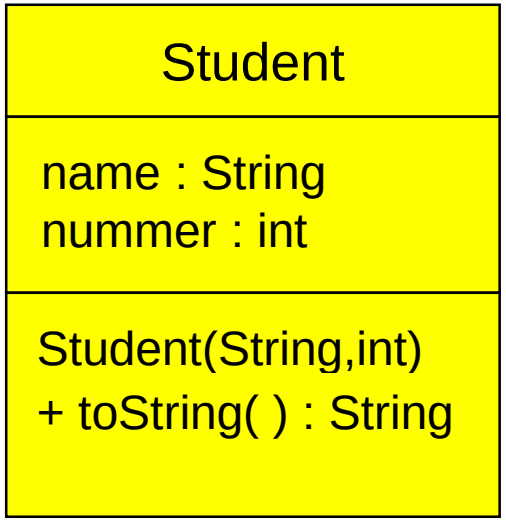
Student
name : String nummer : int
Student(String,int) + toString( ) : String

```
public class Student {
   ...

   public String toString() {
      return "Student "
             + this.name
             + " hat die Matrikelnummer "
             + this.nummer
             + ".";
   }
} // Ende der Klasse
```

Instanzmethoden

Java

Da wir im Methodenkopf den Rückgabetyp **`String`** vereinbart haben, muss unsere Methode natürlich auch einen Wert zurückgeben. Wir geben also durch die return-Anweisung das Ergebnis an die aufrufende Methode weiter.




```
public class Student {
  ...

  public String toString() {
    return "Student "
           + this.name
           + " hat die Matrikelnummer "
           + this.nummer
           + ".";
  }
} // Ende der Klasse
```

Instanzmethoden

Java

Der Zugriff auf die Instanzvariablen des Objektes erfolgt hierbei wieder durch das bereits bekannte Schlüsselwort **`this`**

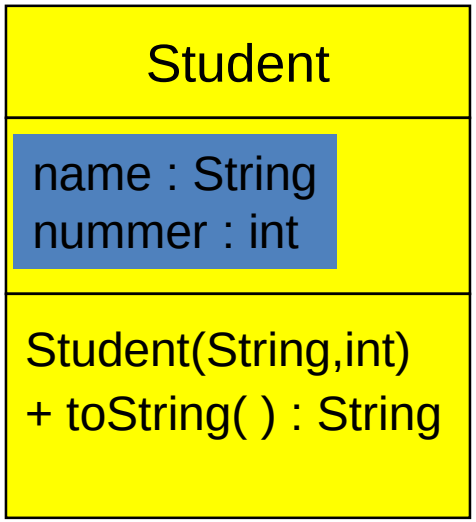
Student
name : String nummer : int
Student(String,int) + toString( ) : String

```
public class Student {
  ...

  public String toString() {
    return "Student "
           + this.name
           + " hat die Matrikelnummer "
           + this.nummer
           + ".";
  }
} // Ende der Klasse
```

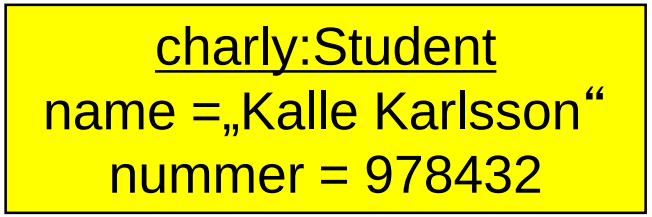
Instanzmethoden

Java

Bei dem entstandenen Objekt handelt es sich um einen normalen String, den wir wie gewohnt über die Methode **`println`** auf dem Bildschirm ausgeben können




```
Student charly =
  new Student("Kalle Karlsson",978432);
String beschreibung = charly.toString();
System.out.println(beschreibung);
```



Instanzmethoden

Java

Die soeben vorgestellten Befehlszeilen lassen sich natürlich noch verkürzt darstellen. Wenn man (wie in unserem Beispiel) die textuelle Beschreibung nur in der Ausgabe verwenden will, braucht man sie natürlich nicht explizit mit einem Variablennamen zu benennen.

Student
name : String nummer : int
Student(String,int) + toString( ) : String

```
Student charly =
  new Student("Kalle Karlsson",978432);
String beschreibung = charly.toString();
System.out.println(beschreibung);
```

```
System.out.println(charly.toString());
```

Instanzmethoden

Java

In diesem speziellen Fall können wir sogar den Aufruf der **`toString`**-Methode wegfallen lassen.

Grund hierfür ist die Definition der Methode **`println`** für Objekte. Diese ruft für das auszugebende Objekt automatisch die Methode **`toString`** auf und druckt das Ergebnis auf dem Bildschirm.

Student
name : String nummer : int
Student(String,int) + toString( ) : String

```
Student charly =
  new Student("Kalle Karlsson",978432);
String beschreibung = charly.toString();
System.out.println(beschreibung);
```

```
System.out.println(charly.toString());
```

```
System.out.println(charly);
```

Fragen

"It was much nicer before people started storing all their personal information in the cloud."

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