



LUDWIG-  
MAXIMILIANS-  
UNIVERSITÄT  
MÜNCHEN



# Javakurs für Anfänger

Einheit 11: Vererbung

Lorenz Schauer  
Lehrstuhl für Mobile und Verteilte Systeme



## 1. Teil Einführung in die Vererbung

- Motivation
- Das Schlüsselwort `extends`
- Einführendes Beispiel: Student und Person

## 2. Teil: Vererbung

- Ein paar Regeln...
- Verwendung
- Subtyping und Type casting
- Die Klasse `Object`

### Praxis:

- Übung mit Vererbung

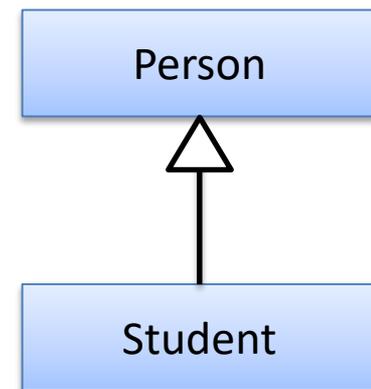
### Lernziele

- Konzept der Vererbung kennenlernen und verstehen
- Vererbung in Java implementieren können
- Konzepte der Vererbung einüben

Die **Vererbung** ist ein Kernprinzip der objektorientierten Programmierung

### **Motivation:** Wiederverwendbarkeit von Klassen

- Neue Klassen müssen nicht immer komplett neu geschrieben werden!
- Oft reicht es aus, neue Klassen aus bestehenden abzuleiten => Vererbung
  - Neue Klasse übernimmt die Eigenschaften und das Verhalten der alten Klasse
    - Die neue Klasse wird auch als Sub-, Unter-, oder Kindklasse bezeichnet
    - Die alte Klasse wird auch als Super-, Ober-, Basis-, Wurzel-, oder Vaterklasse bezeichnet
  - Die Unterklasse kann weitere Attribute und Methoden definieren
  - Die Unterklasse kann Methoden der Oberklasse überschreiben oder erweitern
- Vererbung stellt eine „is-a“ Beziehung dar.
  - Ein Objekt der Unterklasse ist damit auch ein Objekt der Oberklasse
  - Bsp.: Ein Student ist auch eine Person



Eine Vererbung („is-a“-Beziehung) wird mit dem Schlüsselwort `extends` in der Kopfzeile der Unterklasse nach dem Klassennamen eingeleitet

Beispiel: Person und Student:

- Person hat einen Namen, und einen Vornamen
- Person implementiert Getter- und Setter Methoden für name und vorname
- Student hat darüber hinaus noch eine Matrikelnummer
- Student implementiert nur noch die Getter- und Setter Methoden für matrikelnummer

```
public class Student extends Person{  
  
    //Weitere Attribute und Methoden  
}
```

```
// Klasse Person (Oberklasse):

public class Person {

    private String name;
    private String vorname;

//Konstruktor
    public Person(String name, String vorname) {
        this.name = name;
        this.vorname = vorname;
    }

// Getter und Setter
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getVorname() {
        return vorname;
    }
    public void setVorname(String vorname) {
        this.vorname = vorname;
    }
}
```

```
// Klasse Student (Unterklasse)

public class Student extends Person {

    private int matrikelnr;

// Konstruktor
    public Student(String name, String vorname,
        int matrikelnr){
        super(name,vorname);
        this.matrikelnr = matrikelnr;
    }

// Getter und Setter
    public int getMatrikelnr() {
        return matrikelnr;
    }
    public void setMatrikelnr(int matrikelnr) {
        this.matrikelnr = matrikelnr;
    }
}
```

Der Zugriff auf Instanzvariablen der Oberklasse:

- Direkter Zugriff über das Schlüsselwort `super`
  - Bsp.: `super.InstanzVariableOberklasse`
  - Nur möglich, wenn `InstanzVariableOberklasse` als `public` oder `protected` deklariert wurde
  - Private Instanzvariablen sind zwar vorhanden aber nicht sichtbar  
=> Zugriff nur über Methoden möglich

Private Methoden der Oberklasse werden nicht vererbt!

Konstruktoren werden nicht vererbt!

- Der Konstruktor der Oberklasse kann jedoch im Konstruktor der Unterklasse mit `super` aufgerufen werden!
  - `super();` // Konstruktor der Oberklasse ohne Parameter
  - `super(typ param1, typ param2,...)` // Konstruktor der Oberklasse mit Parameter
- Dieser Aufruf muss als **erstes** im Unterklassenkonstruktor definiert sein!

Wir haben nun die beiden Klassen Person und Student definiert, wobei:

- Student von Person erbt
- Student ist also auch immer eine Person, aber nicht jede Person ist ein Student
  - D.h.: Wir können den Studenten auch als eine Person im Code betrachten und dementsprechend benutzen (**Subtyping**)
    - Die Unterklasse ist ein Subtyp der Oberklasse
    - Andersrum ist das nicht möglich!

```
// Beispiel: Benutzung der Klassen
```

```
Student stud1 = new Student("Matthias","Maier",9951);  
Person person1 = new Student("Hansi","Hinterseer",1234);
```

```
//stud1 bietet alle Methoden eines Studenten:  
String name1 = stud1.getName();  
int matr1 = stud1.getMatrikelnr();
```

```
//person1 bietet bisher nur die Methoden einer Person!  
int matr2 = person1.getMatrikelnr();  
FEHLER: The method getMatrikelnr() is undefined for the type Person
```

Wir wissen jedoch, dass die Person `person1` eigtl. auch ein Student ist und alle Informationen eines Studenten besitzt.

- In so einem Fall, können wir die Person `person1` **typkonvertieren**, so dass daraus ein Student wird:
  - Bsp.: `Student neuer_student = (Student) person1;`
- Bei der Typkonvertierung (type cast) wird überprüft, ob die tatsächliche Klasse von `person1` eine Unterklasse von `Person` darstellt.
  - Wenn ja: dann kein Problem
  - Wenn nein: dann Laufzeitfehler: `ClassCastException`
- Mit dem Schlüsselwort `instanceof` lässt sich überprüfen, ob das Objekt tatsächlich eine Instanz der angefragten Klasse ist
  - Beispiel:

```
if(person1 instanceof Student){  
    // liefert true, falls person1 tatsächlich vom Typ Student ist  
  
    System.out.println("Passt");  
  
}
```

## Implizit erben **alle** Klassen in Java von der Klasse `Object`

- Dadurch werden implizit bereits einige Methoden dieser Oberklasse angeboten
- Beispiele:
  - `String toString();` // in einen String konvertieren
  - `boolean equals(Object other);` // auf Gleichheit testen
  - `Object clone();` // Kopie des Objektes anlegen
- Dies kann man auch sehen, wenn man in einer eigenen Klasse bspw. im Konstruktor `this.` eingibt und sich von Eclipse die Attribute und Methoden anzeigen lässt (Strg+Leertaste)

```

1 public class Test {
2
3
4
5     public Test(){
6         super();
7
8         this.
9
10    }
11 }
12

```

- `getClass(): Class<?> - Object - 0,83%`
- `toString(): String - Object - 0,08%`
- `equals(Object obj): boolean - Object - 0,06%`
- `hashCode(): int - Object - 0,05%`
- `clone(): Object - Object`
- `finalize(): void - Object`
- `notify(): void - Object`
- `notifyAll(): void - Object`
- `wait(): void - Object`
- `wait(long timeout): void - Object`
- `wait(long timeout, int nanos): void - Object`
- `new`

Press 'Ctrl+Space' to show Template Proposals

**getClass**

```
public final Class<?> getClass ()
```

Returns the runtime class of this Object. The returned Class object is the object that is locked by static synchronized methods of the represented class.

**The actual result type is `Class<? extends |X|>` where `|X|` is the erasure of the static type of the expression on which `getClass` is called.** For example, no cast is required in this code fragment:

```
Number n = 0;
Class<? extends Number> c = n.getClass();
```

**Returns:** The `Class` object that represents the runtime class of this

Press 'Tab' from proposal table or click for focus

Legen Sie ein neues Eclipse-Projekt „Uebung09“ an und bearbeiten Sie darin folgende Aufgabe zur Vererbung in Java:

Ein Dozent hält die Vorlesung nur, wenn mindestens 4 Studenten anwesend sind. Zudem muss ein Techniker da sein, damit er/sie anfangen kann.

- Schreiben Sie zunächst eine Klasse „Person“, die als Oberklasse für Dozent, Student und Techniker dient.
  - Die Klasse Person hat lediglich einen Namen (String), der im Konstruktor gesetzt werden muss und einen Getter für den Namen
- Schreiben Sie dann eine Klasse „Student“, die von Person erbt.
  - Die Klasse Student hat eine Matrikelnummer (int), die im Konstruktor gesetzt werden muss und einen Getter für die Matrikelnummer.
- Schreiben Sie dann eine Klasse „Techniker“, die von Person erbt.
  - Der Techniker hat eine Methode void ueberpruefen(), die auf der Konsole ausgibt „Techniker überprüft die Technik.“

- Schreiben Sie nun eine Klasse „Dozent“, die von Person erbt.
  - Der Dozent hat ein Unterrichtsfach (String), das im Konstruktor gesetzt werden muss und einen Getter für das Unterrichtsfach.
  - Zudem hat der Dozent eine Liste an Personen, die im Unterricht anwesend sind.
  - Die Methoden des Dozenten sind:
    - void addPerson(Person p), um eine Person in die Liste einzutragen
    - void removePerson(Person p), um eine Person aus der Liste auszutragen
    - void anfangen(), um mit dem Unterricht anzufangen.  
Diese Methode soll auf der Konsole ausgeben „Dozent fängt an...“ wenn:
      - mindestens 4 Studenten und ein Techniker anwesend sind.
      - der Techniker vorab mindestens einmal die Technik überprüft.Falls das nicht zutrifft, soll die Methode anfangen() „Dozent kann nicht anfangen“ ausgeben.

Schreiben Sie nun eine Anwendung (main-Methode), in der Sie einen Dozenten, ein paar Studenten und einen Techniker erzeugen. Dann überprüfen Sie, ob der Dozent anfangen kann, wenn genug Studenten und Techniker anwesend sind.