LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Javakurs für Anfänger

Einheit 11: Vererbung

Kyrill Schmid
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

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

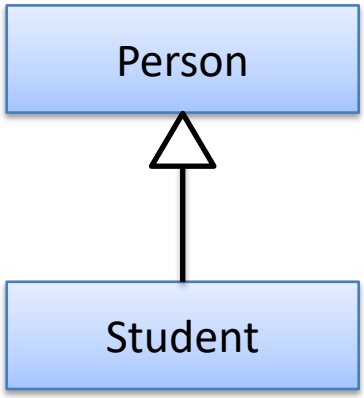
Lernziele

- 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



Das Schlüsselwort `extends`

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
}
```

Beispiel: Student extends Person


```
// 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;
   }

}
```

Vererbung: Ein paar Regeln!

Der Zugriff auf Instanzvariablen der Oberklasse:

- Direkter Zugriff über das Schlüsselwort `super`
  - Bsp.: `super.InstanzVaraibleOberklasse`
  - Nur möglich, wenn `InstanzVaraibleOberklasse` 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!

Verwendung der Klassen

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 studi1 = new Student("Matthias","Maier",9951);
Person person1 = new Student("Hansi","Hinterseer",1234);

//studi1 bietet alle Methoden eines Studenten:
     String name1 = studi1.getName();
     int matnr1 = studi1.getMatrikelnr();

//person1 bietet bisher nur die Methoden einer Person!
     int matnr2 = person1.getMatrikelnr();
     FEHLER: The method getMatrikelnr() is undefined for the type Person
```

Typkonvertierung

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

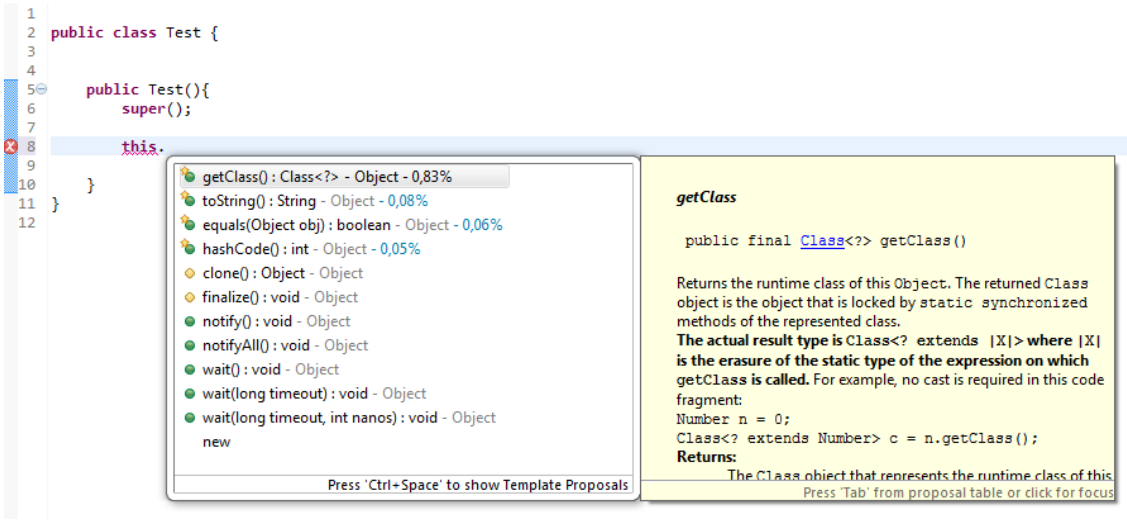
- In so einem Fall, können wir die Person person1 **typkonvertieren**, so dass daraus ein Student wird:
  - Bsp.: `Student neuer_student = (Student) person1;`
- Bei der Typkonvertierung (type cast) wird überprüft, ob die tatsächliche Klasse von person1 eine Unterklasse von Person darstellt.
  - Wenn 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");

}
```

Die Klasse `Object`

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)



Programmieraufgabe zur Vererbung

Legen Sie ein neues Eclipse-Projekt „Uebung11“ 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.“

Programmieraufgabe zur Vererbung

- 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.