Vererbung

Sven Eric Panitz

Vererbung

- Klassen beschreiben Mengen von Objekten, die Instanz dieser Klasse sind.
- Mit der Definition einer **Unterklasse** können Teilmengen von Objekten beschrieben werden.
- In einer Unterklasse kann es zusätzliche Felder für zusätzliche Eigenschaften geben.
- Funktionalität in Form von Methoden kann in einer Unterklasse neu definiert werden.
- Mit final markierte Klssen dürfen keine Unterklassen haben.
- Record-Klassen sind implizit mit final markiert.

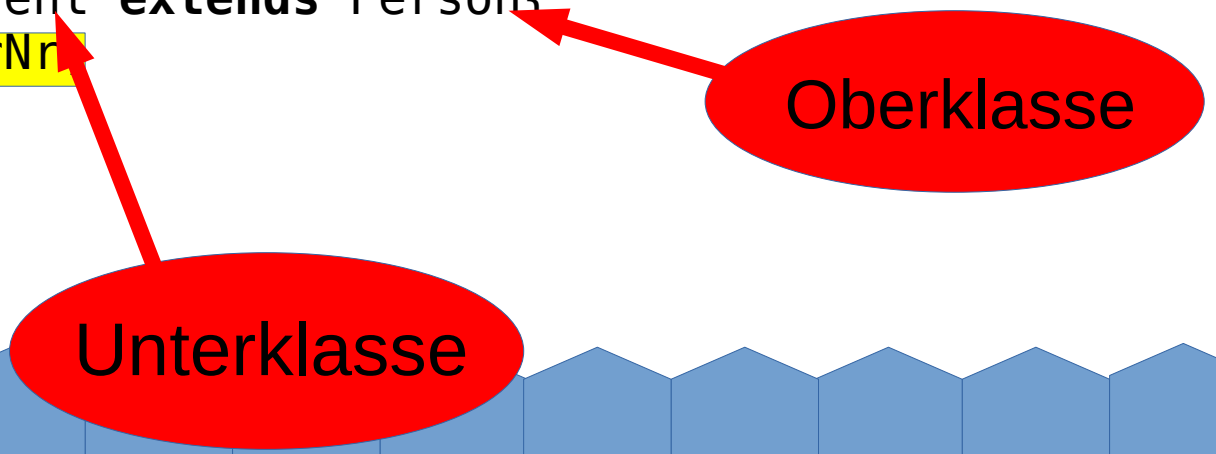
```
class Person{
  String name;


  void grüßen(){
    System.out.println( "Guten Tag. Ich bin " + name);
  }
  String getName(){return name;}
}


  class Student extends Person{
    int matrNr;



  }
```



Unterklassen

- Eine Unterklasse (hier `Student`) wird mit dem Schlüsselwort **extends** gefolgt von Namen der Oberklasse (hier `Person`) deklariert.
- Jedes Objekt der Unterklasse ist auch Objekt der Oberklasse.
  (Jeder Student ist eine Person).
  Aber nicht umgekehrt. (Nicht jede Person ist ein Student).
- Objekte der Unterklasse erben alle Eigenschaften der Oberklasse.
  (auch `Student` Objekte haben ein Feld `name` und die Methoden `getName` und `grüßen`).
- Unterklassen können weitere Felder und Methoden haben.

```
class Person{
  String name;
  Person(String n){name = n;}

  void grüßen(){
    System.out.println( "Guten Tag. Ich bin " + name);
  }
  String getName(){return name;}
}
class Student extends Person{
  int matrNr;
  Student(String n, int mN){
    super(n);
    matrNr = mN;
  }


}
```

Konstruktoren der Unterklasse

- Der erste Befehl in einem Konstruktor muss immer ein Aufruf eines Konstruktors der Oberklasse sein.
- Der Aufruf eines Konstruktors der Oberklasse geschieht mit dem Schlüsselwort **super** gefolgt von Parametern.

```
class Test{
  public static void main(String[] args){
    Person p = new Person("Murat");
    Student s = new Student("Hans", 153753);

    Person ps = new Student("Tatjana", 345543);

    System.out.println(p.name);
    System.out.println(p.getName());
    System.out.println(p.grüßen());

    System.out.println(s.name);
    System.out.println(s.getName());
    System.out.println(s.grüßen());
    System.out.println(s.matrNr);

    System.out.println(ps.name);
    System.out.println(ps.getName());
    System.out.println(ps.grüßen());
    //System.out.println(ps.matrNr);
  } }
```

Überschreiben von Methoden

- Geerbte Methoden können in Unterklassen neu definiert werden.
- Man spricht von »Überschreiben« (eng. »Override«)
- Überschreibende Methoden sollten mit der Annotation `@Override` markiert sein

```
class Person{
  String name;
  Person(String n){name = n;}

  void grüßen(){
    System.out.println( "Guten Tag. Ich bin " + name);
  }
  String getName(){return name;}
}
class Student extends Person{
  int matrNr;
  Student(String n, int mN){
    super(n);
    MatrNr = mN;
  }
  @Override void grüßen(){
    System.out.println( "Alder, was geht? Ich bin Student "
                        + name+ "mit Matrikelnummer "+matrNr);
  }
}
```

Geerbte Methode wird überschrieben.

```
class Test{
  public static void main(String[] args){
    Person p = new Person("Murat");
    Student s = new Student("Hans", 153753);

    Person ps = new Student("Tatjana", 345543);

    System.out.println(p.grüßen());
    System.out.println(s.grüßen());
    System.out.println(ps.grüßen());

  }
}
```

Späte Bindung

- Beim Aufruf einer Methode, wird das Objekt gefragt, welche spezifische überschriebene Version der Methode die Klasse hat, von der das Objekt erzeugt wurde.
  (Ein `Student`-Objekt vergisst nicht, dass es von der Klasse `Student` ist.)
- Die Auflösung, welche Methode ausgeführt wird, findet zur Laufzeit statt, nicht durch den Compiler.
  Daher **späte Bindung** (late binding).

Die Klasse Object

- wenn für eine Klasse keine Oberklasse mit extends deklariert ist, so hat diese Klasse die Oberklasse Object.
- Damit sind alle Objekte auch vom Typ Object
- In der Standardklasse Object sind ein paar Methoden definiert, die jedes Objekt kann.
- Diese Methoden sind in der Regel zu überschreiben.
- Diese Methoden sind:
  toString(), equals(Object) und hashCode()

Überschreiben von toString()

```
class Person{
  String name;
  @Override
  public String toString(){
    return „Person mit Namen: „+name;
  }
  void grüßen(){
    System.out.println( "Guten Tag. Ich bin " + name);
  }
  String getName(){return name;}
}
```

Komponentenbildung durch Vererbung

- ```
  class KnopfLogik{
    String eval(String input){
      return input;
    }
  }
  ```
- ```
  class Dialogue{
    Dialogue(KnopfLogik l){
      …
    }
    …
  }
  ```

Beispiel: Komponentenbildung

- Die Klasse `Dialogue` ist eine komplexe GUI-Klasse, die ein Fenster öffnet. In diesem ist ein Eingabefeld, ein Knopf und ein Ausgabefeld.
- Bei Drücken des Knopfes wird der Text der Eingabe der Methode *`eval`* des KnopfLogik-Objektes übergeben und anschließend die Rückgabe im Ausgabefeld angezeigt.
- Durch spezielle Unterklasse von `KnopfLogik` können nun dem GUI unterschiedliche Anwendungslogiken übergeben werden.

```
class Factorial extends KnopfLogik{
  @Override String eval(String input){
     return fac(new Integer(input))+““;
  }
  int fac(int n){
    return n<=1 ? 1 : n*fac(n-1);
  }

  public static void main(String[] args){
    new Dialogue(new Factorial());
  }
}
```

Zusammenfassung

- Unterklassen bilden Untermengen.
- Unterklassen erben die Eigenschaften der Oberklassen.
- Unterklassen können Klassen erweitern um Felder und Methoden.
- Unterklassen können Methoden überschreiben.