Programmieren I

Dr. Klaus Höppner

Hochschule Darmstadt – Wintersemester 2009/2010

Vererbung

Eine Klasse kann eine speziellere Version einer allgemeineren Klasse sein, z. B.

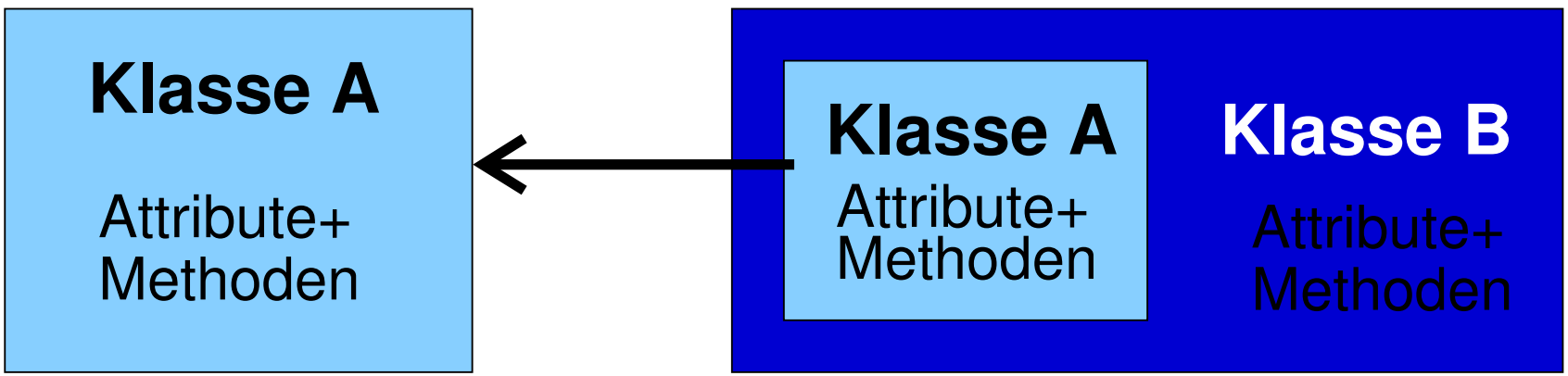
- Kfz (allgemein) – Pkw, Lkw (speziell)
- Konto (allgemein) – Girokonto, Sparkonto (speziell)

Diesen Mechanismus nennt man *Vererbung*, d. h. die spezielle Klasse *erbt* die Attribute und Methoden der allgemeinen Klasse.
In der abgeleiteten Klasse können neue Attribute/Methoden hinzu kommen, aber auch Attribute/Methoden der Basisklasse redefiniert werden.

Beispiel: In einer Leihbibliothek gibt es Waren in Form von Büchern, DVDs, CDs etc.:

Inventar – Buch, DVD, CD, . . .

Schaubild


Klasse A
Attribute+
Methoden
Klasse A
Attribute+
Methoden
Klasse B
Attribute+
Methoden

Einfaches Beispiel

Betrachten wir zwei einfache Klassen:

```
public class Tier {
    public void atme() {
        System.out.println("Ich atme");
    }
}

public class Ente extends Tier {
    public void quak() {
        System.out.println("quak");
    }
}
```

Einfaches Beispiel (Forts.)

In diesem Beispiel wird durch
`class Ente extends Tier`
`Ente` als Kindklasse von `Tier` definiert. Dabei erbt `Ente` dann alle Attribute und Methoden der Elternklasse `Tier`, in diesem Fall also die Methode `void atme()`.

Tatsächlich ist jede Instanz von `Ente` auch eine Instanz von `Tier`. Daher kann man Enten überall da verwenden, wo allgemein ein Tier erwartet wird.

Einfaches Beispiel (Forts.)

Beispielprogramm:

```
public class App {
    public static void main(String[] args) {
        Tier t1 = new Tier();
        Ente e = new Ente();
        Tier t2 = e; // zulässig, da jede Ente ein Tier ist

        t1.atme();
        e.atme(); // atme wird von Tier geerbt
        e.quak(); // geht
        t2.quak(); // FEHLER, t2 ist "nur" ein Tier
        System.out.println(e instanceof Tier); // true
    }
}
```

Type Casts – Typumwandlung

Da eine Ente ein Tier ist, kann jede Objektreferenz vom Typ `Tier` auf eine Instanz von `Ente` verweisen. Daher ist die Anweisung `Tier t2 = e;` zulässig. Dies nennt man *up cast*. Ein solcher ist immer zulässig.
`t2` „kann“ dann natürlich nur das, was alle Tier können!

Man kann aus einer Objektreferenz vom Typ `Tier` wieder eine Referenz vom Typ `Ente` machen, aber nur, wenn diese Referenz auf eine Instanz von `Ente` verweist! Dies nennt man *down cast*. Solche Casts müssen explizit durchgeführt werden, z. B.

```
Ente e2 = (Ente) t2;
```

Vererbung und Konstruktoren

Erweitern wir den Code von `Tier` so, dass jedes Tier einen Namen hat, der beim Instanzieren übergeben wird:

```
public class Tier {
    private String name;

    public Tier(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void atme() {
        System.out.println("Ich atme");
    }
}
```

Vererbung und Konstruktoren (Forts.)

Nun gibt es einen Fehler in `Ente`! Warum?
Beim Instanzieren eines neuen Objektes der Kindklasse wird implizit *zuerst* der Standardkonstruktor der Elternklasse ausgeführt. Dieser existiert im Beispiel nun aber nicht mehr, nachdem der Klasse `Tier` der Konstruktor mit einen Namen als Parameter definiert wurde!

Im Konstruktor einer Klasse darf als *erste* Anweisung *einer* der folgenden Aufrufe stehen:

- `this(...)` Aufruf eines anderen Konstruktors derselben Klasse, oder
- `super(...)` Angabe, welcher Konstruktor der Elternklasse aufgerufen werden soll.

Fehlt beides, so wird implizit `super()` angenommen.

Beispiel

Im vorigen Beispiel kann der Compiler-Fehler nun dadurch beseitigt werden, dass der Konstruktor von `Tier` mit Parameter aufgerufen wird:

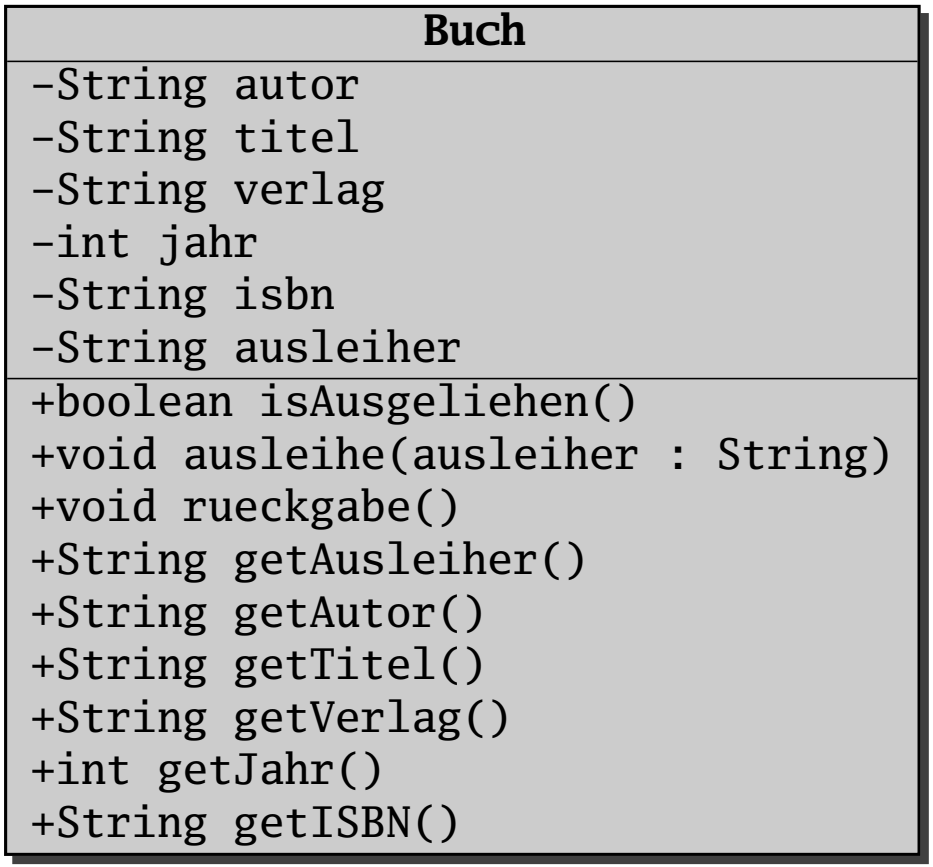
```
package tier;

public class Ente extends Tier {
    public Ente(String name) {
        super(name);
    }

    public void quak() {
        System.out.println("quak");
    }
}
```

Diagramm der Klasse Buch

In einer früheren Vorlesung wurde eine Klasse Buch definiert:

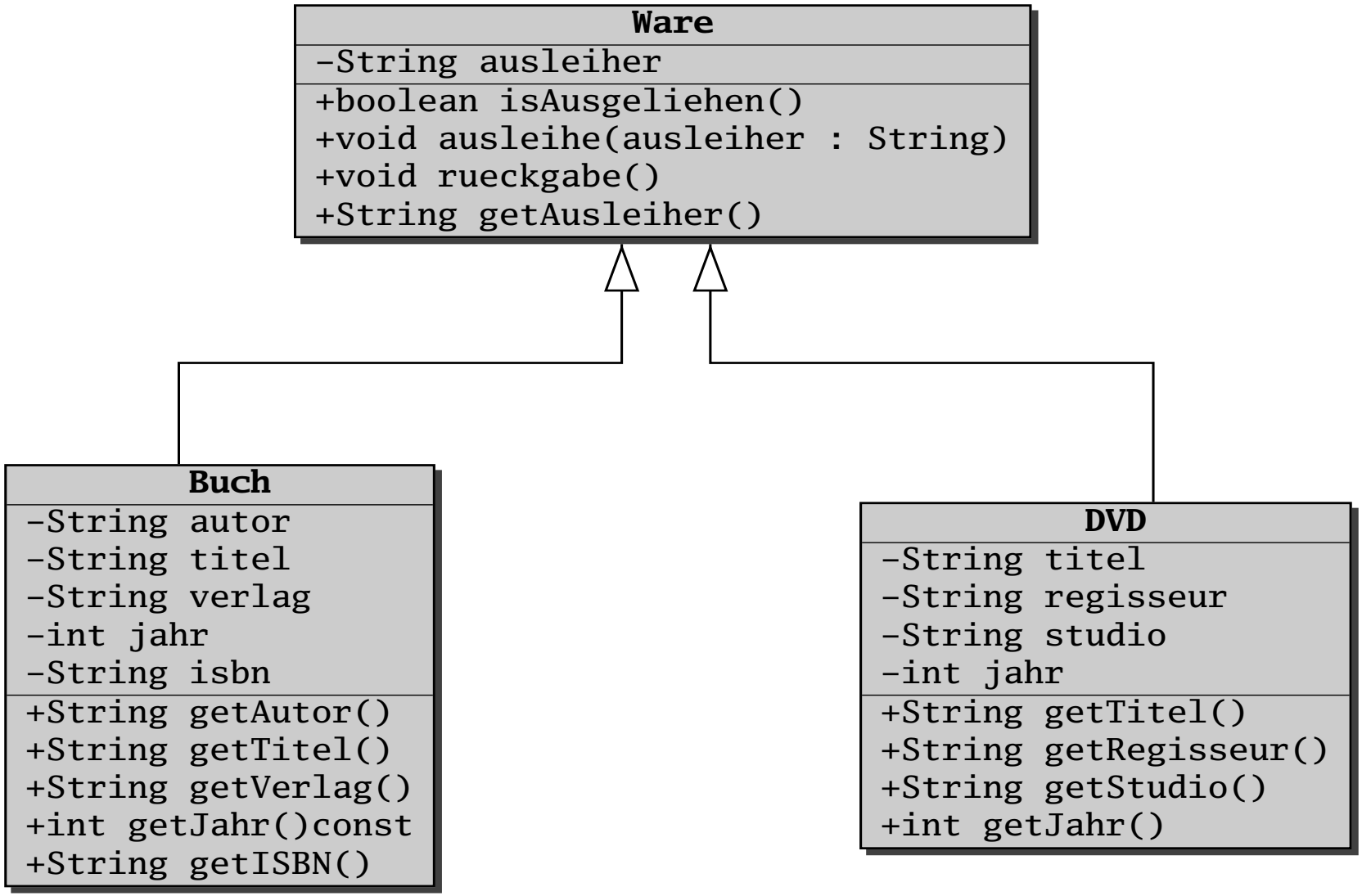


Erweiterung

Die Klasse Buch enthält Teile, die den Vorgang des Ausleihens betreffen, als auch buchspezifische Details. Möchte eine Bibliothek neben Büchern auch noch weitere Gegenstände anbieten (z. B. DVDs, Musik-CDs), so gibt es zwei Möglichkeiten:

- In neuen Klassen werden sowohl die Ausleihe als auch die Produktdetails definiert. Das ist unpraktisch, da für jede neue Warenart die Funktionalität der Ausleihe erneut definiert werden muss.
- Die Funktionalität, die für alle Waren identisch ist – also die Ausleihe –, wird in eine Basisklasse ausgelagert, von der für jede Warenart eine Kindklasse erbt.

Klassendiagramm: Bücher und DVDs als Warenarten



Implementierung von Ware.java

```
public class Ware {
    private String ausleiher = null;

    public String getAusleiher() {
        return ausleiher;
    }
    public void ausleiher(String ausleiher) {
        this.ausleiher = ausleiher;
    }
    public void rueckgabe() {
        ausleiher = null;
    }
    public boolean isAusgeliehen() {
        return ausleiher == null;
    }
}
```

Implementierung von Buch.java

```
public class Buch extends Ware {
    private String autor;
    private String titel;
    private String verlag;
    private int jahr;
    private String isbn;

    public Buch(String autor, String titel, String verlag,
                int jahr, String isbn) {
        super();
        this.autor = autor;
        this.titel = titel;
        this.verlag = verlag;
        this.jahr = jahr;
        this.isbn = isbn;
    }
```

Implementierung von Buch.java (Forts.)

```
    public String getAutor() {
        return autor;
    }
    public String getTitel() {
        return titel;
    }
    public String getIsbn() {
        return isbn;
    }
    public int getJahr() {
        return jahr;
    }
    public String getVerlag() {
        return verlag;
    }
}
```

Implementierung von DVD.java

```
public class DVD extends Ware {
    private String regisseur;
    private String titel;
    private String studio;
    private int jahr;

    public DVD(String regisseur, String titel,
               String studio, int jahr) {
        super();
        this.regisseur = regisseur;
        this.titel = titel;
        this.studio = studio;
        this.jahr = jahr;
    }
```

Implementierung von DVD.java

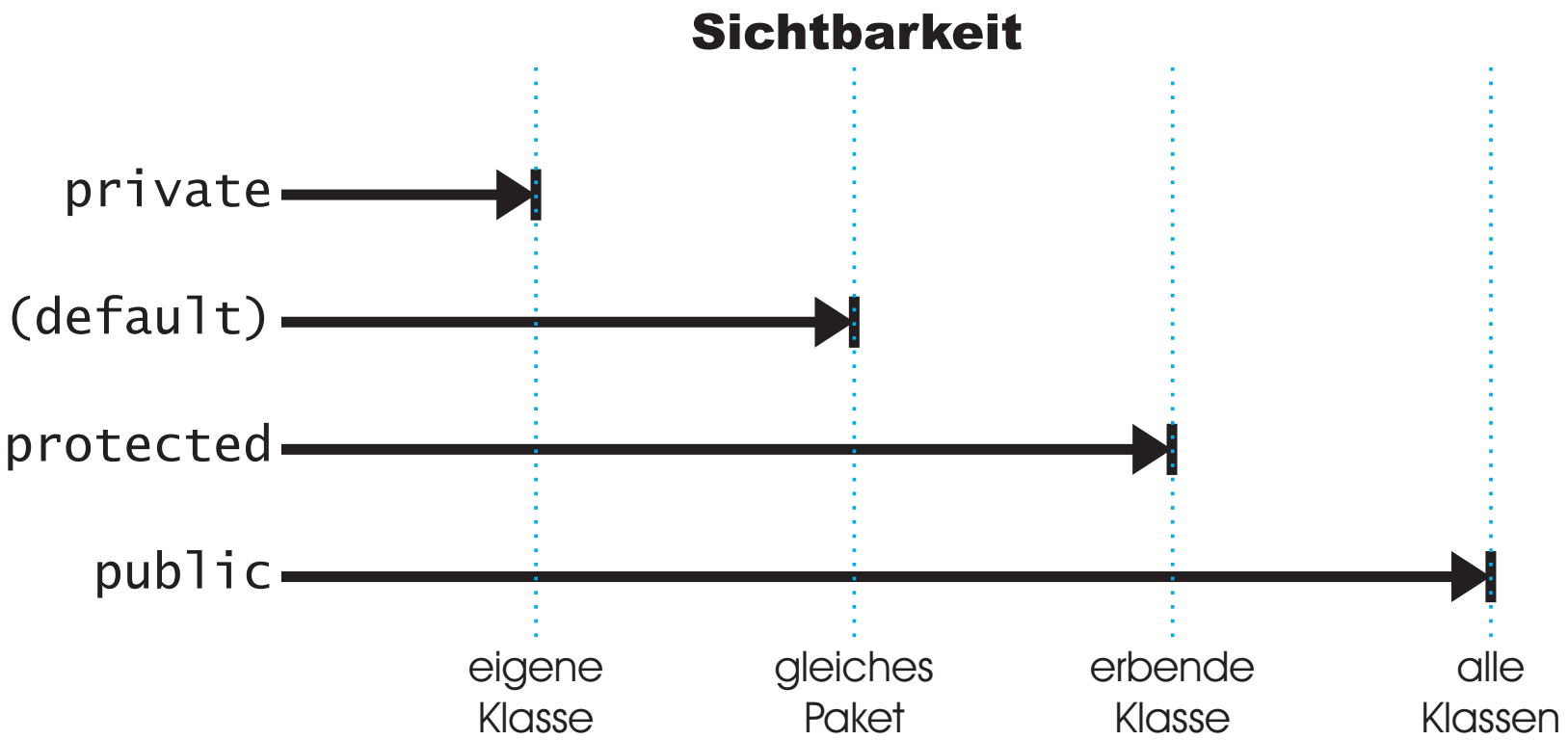
```
    public String getRegisseur() {
        return regisseur;
    }
    public String getTitel() {
        return titel;
    }
    public int getJahr() {
        return jahr;
    }
    public String getStudio() {
        return studio;
    }
}
```

Sichtbarkeit von Klassen, Methoden, Attributen

Man kann in Java für Klassen, Methoden und Attribute entscheiden, in welchem Kontext sie sichtbar sind. Die Sichtbarkeitsstufen lauten:

private	Sichtbarkeit nur innerhalb derselben Klasse,
(keine Angabe)	Sichtbarkeit innerhalb des selben Paketes (**package**),
protected	Sichtbarkeit in allen erbenden Klassen (zusätzlich zur Sichtbarkeit im selben Paket!)
public	Sichtbarkeit in allen Klassen

Schaubild


Sichtbarkeit
private
(default)
protected
public
eigene Klasse
gleiches Paket
erbende Klasse
alle Klassen