



# Java Einführung

## Vererbung und Polymorphie

Kapitel 13

# Inhalt

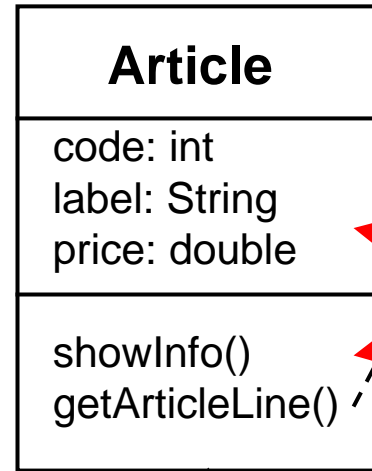
- Klassifikation (UML)
- Implementierung von Vererbungshierarchien
- Überschreiben von Methoden
  
- Polymorphismus: Up-Casting und Dynamisches Binden
  
- Schlüsselwort `final`

# Vererbung

- Vererbung ist ein Grundkonzept der Objektorientierung.
- **Sub-Klassen** erben dabei die Eigenschaften (Variablen, Methoden) von **Super-Klassen**
- In Java wird **nur Einfachvererbung** unterstützt, d.h. jede Sub-Klasse kann nur direkt von einer Klasse erben.

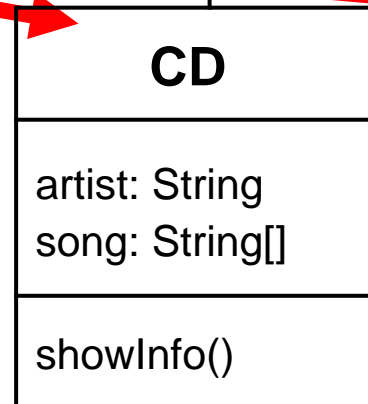
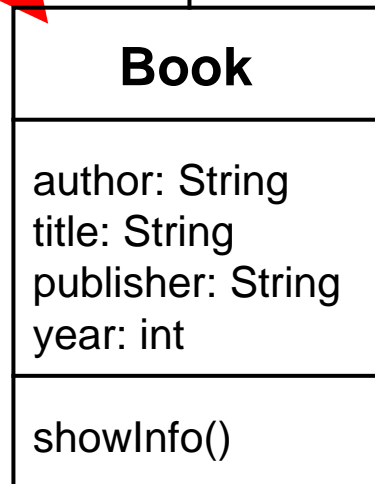
# Klassifikation Buchhandlung

"Erben" alle  
Eigenschaften  
von Article

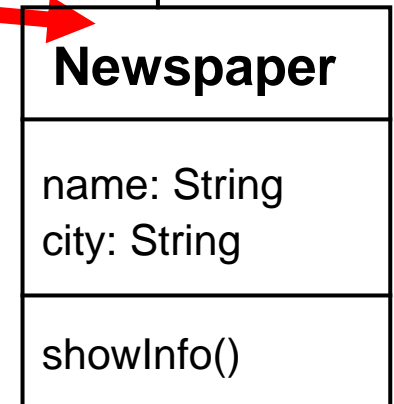


Anzeige für  
Warenkorb

Article beinhaltet  
alle gemeinsamen  
Eigenschaften



...



# Implementierung der Elternklasse

```
class Article {
```

```
    int code;
```

```
    String label;
```

```
    double price;
```

```
    void showInfo() {...}
```

```
    String getArticleLine() {...}
```

```
    Article (int code, String label, double  
            price) {...}
```

```
}
```

<b>Article</b>
code: int label: String price: double
showInfo() getArticleLine()

# Implementierung der Abgeleiteten Klasse Book

```
class Book extends Article {  
  
    String author, title, publisher;  
    int year;  
  
    void showInfo() {...}  
  
    Book (int code, String author, String title,  
        double price) {...}  
}
```

Book
author: String title: String publisher: String year: int
showInfo()

Bekommt alle Eigenschaften von Article, erweitert die Attribute und überschreibt die Methode showInfo()

# Implementierung der Abgeleiteten Klasse CD

```
class CD extends Article {  
  
    String artist;  
    String [] song;  
  
    void showInfo() {...}  
  
    CD(int code, String artist, double  
        price){...}  
}
```

CD
artist: String song: String[]
showInfo()

Bekommt alle Eigenschaften von Article, erweitert die Attribute und überschreibt die Methode `showInfo()`

# Überschreiben von Methoden

Die neu definierte Methode in der abgeleiteten Klasse ersetzt die gleichnamige Methode in der Elternklasse (Superklasse)

- Die neue Methode kann die Methode der Superklasse aufrufen:

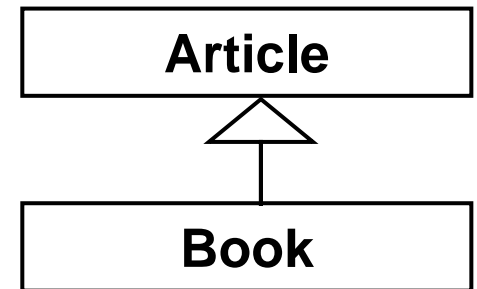
```
class Book extends Article {  
    ...  
    void showInfo() {  
        super.showInfo(); // Methode in Article  
        // show more stuff  
    } ...  
}
```

# Instanziierung einer abgeleiteten Klasse

```
Book myBook = new Book(10, "Jules Verne",  
    "20.000 Meilen unter dem Meer", 21.90)
```

Top-Down-Ablauf bei der Instanziierung:

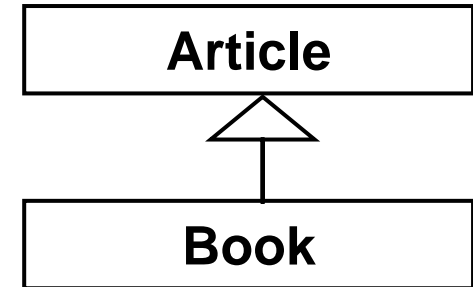
1. Konstruktor von Article
2. Konstruktor von Book



# Instanziierung einer abgeleiteten Klasse II

Bsp: Konstruktor der Super-Klasse mit Argumenten aufrufen:

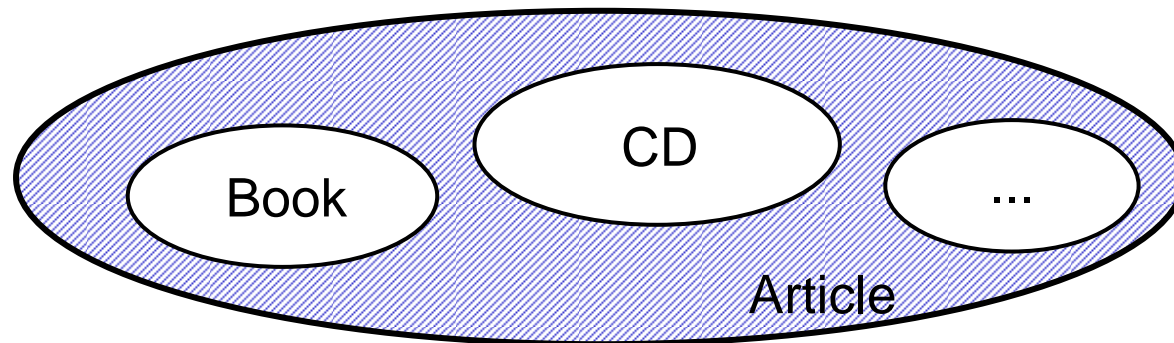
```
class Book extends Article {  
...  
Book (int code, String author, String  
title, double price){  
  super(code, autor+": "+title, price);  
  // muss die 1.Zeile sein  
...  
}  
...  
}
```



```
class Article {  
Article (int code, Sting label,  
double preis) {...}  
...  
}
```

# Polymorphie: Kompartibilität zwischen Ober- und Unterklasse

- Jedes Programm, das in der Lage ist, mit Instanzen der Oberklasse zu arbeiten, kann auch mit Instanzen der Unterklasse arbeiten. (*Polymorphismus*)
- D.h. jedes Programm, das Artikel (Article) verwalten kann, kann auch mit unseren Büchern (Book) und CDs (CD) oder allen anderen abgeleiteten Klassen arbeiten, da es das Interface von Article kennt.



# Zuweisung von Objekten

- Aufgrund der Kompartibilität kann ein Objekt der Unterklasse einer Oberklassenvariable zugewiesen werden (**up-casting**)

```
Article a = new Book();
```

```
// a ist ein book sieht aber wie ein Article aus
```

- Prüfen auf den Klassentyp

```
if (a instanceof Book) { ... }
```

- Umwandeln auf original Klassentyp

```
Book b = (Book) a; // Cast
```

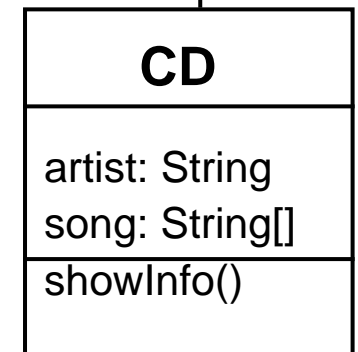
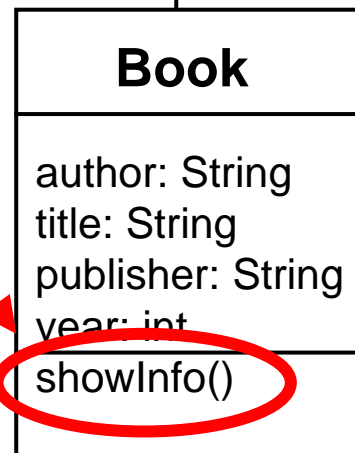
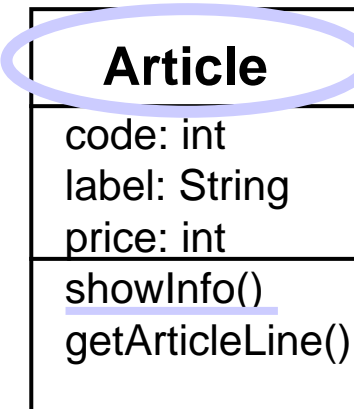
# Dynamische Bindung

- Problemstellung:
  - Methode wird überschrieben
  - Objekt der Unterklasse wird in einer Oberklassen-Variable gespeichert (up-casting)
  - Welche Methode wird verwendet?
- Lösung: **Dynamische Bindung** - Die Methode des "echten" Typs wird zur Laufzeit ermittelt und ausgeführt.

# Dynamische Bindung II

```
Article a = new Book();
```

```
a.showInfo();
```



Kommen später neue Artikel hinzu, braucht im Programm nichts mehr verändert werden!

# Dynamische Bindung III

Anwendung mit Arrays von Objekten verschiedener Unterklassen

```
Article [] someArticles = new [10] Articles;
someArticles[0] = new Book(12, "S.King",...);
someArticles[1] = new CD(31, "Sting",...);
... // Up-casts

for (int i=0; i<someArticles.lenght; i++) {
    someArticles[i].showInfo(); // dyn. Binden
}
```

# final + primitive Datentyp

Der Wert der „Variablen“ ist konstant.

```
final int I = 0;  
I++; /* Anweisung nicht möglich, der Wert von i kann nicht  
verändert werden.*/
```

## Fehlermeldung des Compilers:

```
Test.java:4: Can't assign a value to a final variable: I  
           I++;  
           ^
```

1 error

# final + Objekt-Datentyp

Die Referenz ist konstant.

```
Number.java:  
class Number {  
    int i;  
}
```

```
final Number N1 = new Number(19);
```

```
Number n2 = new Number(47);
```

```
N1 = n2; /*Anweisung nicht möglich. Der Variablen N1 kann  
keine neue Referenz zugewiesen werden.*/
```

```
N1.i = n2.i; /* Anweisung möglich! */
```

# final + Parameter

Parameter ist in der Methode nicht veränderbar.

```
public void deposit(final double VALUE){  
    VALUE = VALUE * 2; /* Anweisung nicht möglich. Der Variablen  
    VALUE kann kein neuer Wert zugewiesen werden. */  
  
    balance+= VALUE; // Anweisung möglich  
    ...  
}
```

# final + Methode

Das Überschreiben der Methode durch eine Subklasse wird verhindert.

```
class Lebewesen {  
    public final String getName() {...}  
}
```

- Gründe: Effizienzsteigerung (kein dynamische Binden; siehe Polymorphismus) oder Sicherheit.
- `private` + Methode entspricht impliziten `final` weil die Subklasse keinen Zugriff hat.

# final + Klasse

Eine Vererbung dieser Klasse ist nicht möglich.

```
final class FinalClass{  
    class SubClasse extends FinalClass{  
        //Anweisung nicht möglich
```

Fehlermeldung des Compilers:

```
Test.java:12: Can't subclass final classes: class Number  
public class Test extends Number{
```

- Damit soll die Sicherheit oder Effizienz gesteigert werden. Die Methoden einer final definierten Klasse sind implizit final definiert.

# Was sie nach dieser Einheit wissen sollten...

- Kompatibilität bei abgeleiteten Klassen.
- Was ist Polymorphismus.
- Was Up-Casting ist.
- Dynamisches Binden.
- Die unterschiedlichen Bedeutungen des Schlüsselworts `final`