

# Vererbung

# Inhalt

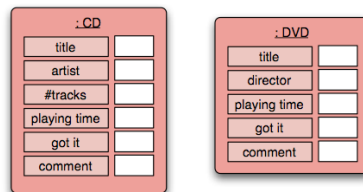
- Vererbung
- Subtyping
- Substitution
- Polymorphe Variablen

# The DoME example

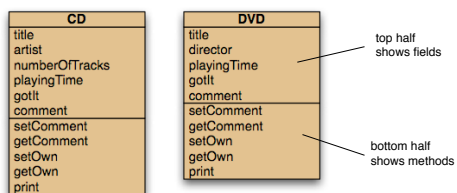
"Database of Multimedia Entertainment"

- stores details about CDs and DVDs
  - CD: title, artist, # tracks, playing time, got-it, comment
  - DVD: title, director, playing time, got-it, comment
- allows (later) to search for information or print lists

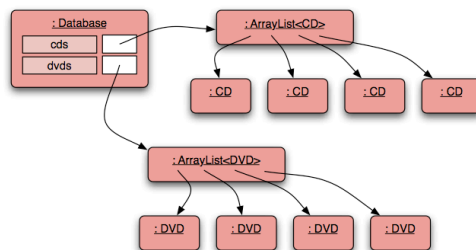
# DoME objects



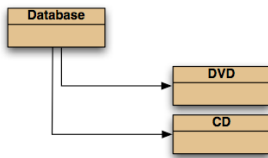
# DoME classes



# DoME object model



## Class diagram



## CD source code

```

public class CD
{
    private String title;
    private String artist;
    private String comment;

    public CD(String theTitle, String theArtist)
    {
        title = theTitle;
        artist = theArtist;
        comment = " ";
    }

    public void setComment(String newComment)
    { ... }

    public String getComment()
    { ... }

    public void print()
    { ... }
    ...
}
    
```

incomplete (comments!)

## DVD source code

```

public class DVD
{
    private String title;
    private String director;
    private String comment;

    public DVD(String theTitle, String theDirector)
    {
        title = theTitle;
        director = theDirector;
        comment = " ";
    }

    public void setComment(String newComment)
    { ... }

    public String getComment()
    { ... }

    public void print()
    { ... }
    ...
}
    
```

incomplete (comments!)

## Database source code

```

class Database {
    private ArrayList<CD> cds;
    private ArrayList<DVD> dvds;
    ...

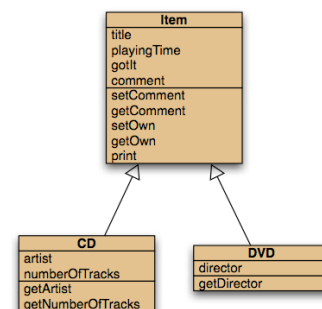
    public void list()
    {
        for(CD cd : cds) {
            cd.print();
            System.out.println(); // empty line between items
        }

        for(DVD dvd : dvds) {
            dvd.print();
            System.out.println(); // empty line between items
        }
    }
}
    
```

## Critique of DoME

- code duplication
  - CD and DVD classes very similar (large part are identical)
  - makes maintenance difficult/more work
  - introduces danger of bugs through incorrect maintenance
- code duplication also in Database class

## Using inheritance

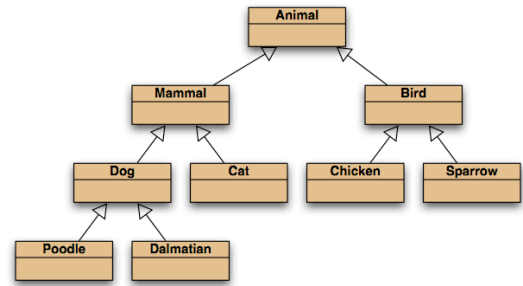


## Using inheritance

- define one **superclass**: Item
- define **subclasses** for Video and CD
- the superclass defines common attributes
- the subclasses **inherit** the superclass attributes
- the subclasses add own attributes

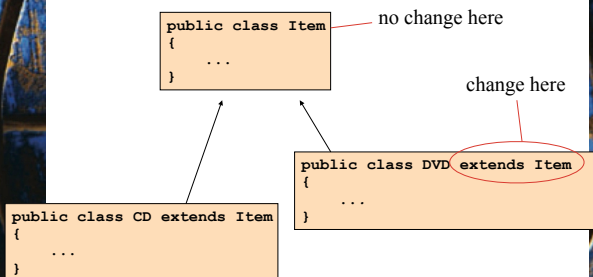
ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Inheritance hierarchies



ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Inheritance in Java



ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Superclass

```
public class Item
{
    private String title;
    private int playingTime;
    private boolean gotIt;
    private String comment;

    // constructors and methods omitted.
}
```

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Subclasses

```
public class CD extends Item
{
    private String artist;
    private int numberOfTracks;

    // constructors and methods omitted.
}

public class DVD extends Item
{
    private String director;

    // constructors and methods omitted.
}
```

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Inheritance and

```
public class Item
{
    private String title;
    private int playingTime;
    private boolean gotIt;
    private String comment;

    /**
     * Initialise the fields of the item.
     */
    public Item(String theTitle, int time)
    {
        title = theTitle;
        playingTime = time;
        gotIt = false;
        comment = "";
    }

    // methods omitted
}
```

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Inheritance and constructors

```

public class CD extends Item
{
    private String artist;
    private int numberOfTracks;

    /**
     * Constructor for objects of class CD
     */
    public CD(String theTitle, String theArtist,
              int tracks, int time)
    {
        super(theTitle, time);
        artist = theArtist;
        numberOfTracks = tracks;
    }

    // methods omitted
}

```

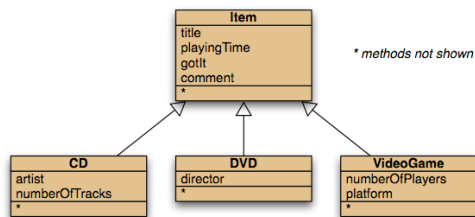
ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Superclass constructor call

- Subclass constructors must always contain a 'super' call.
- If none is written, the compiler inserts one (without parameters)
  - works only, if the superclass has a constructor without parameters
- Must be the first statement in the subclass constructor.

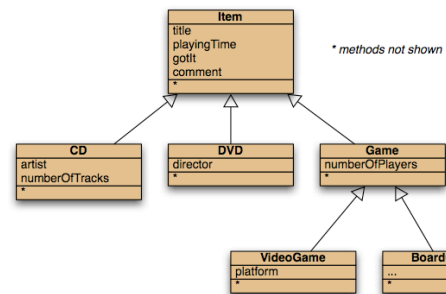
ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Adding more item types



ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Deeper hierarchies



ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Review (so far)

Inheritance (so far) helps with:

- Avoiding code duplication
- Code reuse
- Easier maintenance
- Extendibility

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

```

public class Database
{
    private ArrayList<Item> items;

    /**
     * Construct an empty Database.
     */
    public Database()
    {
        items = new ArrayList<Item>();
    }

    /**
     * Add an item to the database.
     */
    public void addItem(Item theItem)
    {
        items.add(theItem);
    }
    ...
}

```

Neuer Database-Code

*vermeidet Code-Duplizierung im Klienten!*

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Neuer Database-Code

```
/**
 * Print a list of all currently stored CDs and
 * DVDs to the text terminal.
 */
public void list()
{
    for(Item item : items) {
        item.print();
        // Print an empty line between items
        System.out.println();
    }
}
```

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Subtyping

Zuerst hatten wir:

```
public void addCD(CD theCD)
public void addVideo(DVD theDVD)
```

Jetzt haben wir:

```
public void addItem(Item theItem)
```

Wir rufen die Methode wie folgt auf:

```
DVD myDVD = new DVD(...);
database.addItem(myDVD);
```

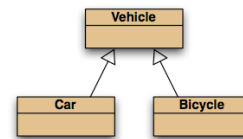
ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Unterklassen und Subtyping

- Klassen definieren Typen.
- Unterklassen definieren Untertypen (Subtypes).
- Objekte der Unterklasse können benutzt werden wo Objekte der Oberklasse erwartet werden. (Dies heißt **Substitution**.)

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Subtyping und Zuweisung



Unterklassenobjekte können Variablen des Oberklassentyps zugewiesen werden.

```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Subtyping und Parameterübergabe

```
public class Database
{
    public void addItem(Item theItem)
    {
        ...
    }
}

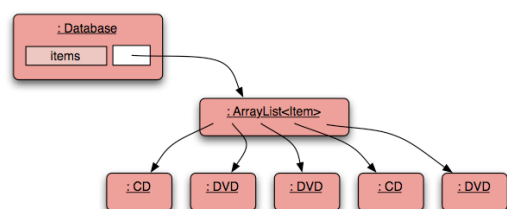
DVD dvd = new DVD(...);
CD cd = new CD(...);

database.addItem(dvd);
database.addItem(cd);
```

Unterklassenobjekte können an Parameter vom Oberklassentyp übergeben werden

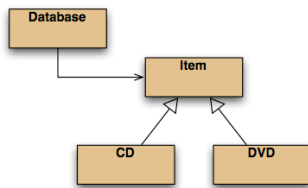
ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Objektdiagramm



ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Klassendiagramm



ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Polymorphe Variablen

- Objektvariablen in Java sind **polymorph.**  
(Sie können Objekte mehrerer Typen halten.)
- Sie können Objekte des deklarierten Typs speichern, oder eines Untertyps.

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Casting

- Untertyp kann an Obertyp zugewiesen werden.
- Andersrum nicht!  

```
Vehicle v;  
Car c = new Car();  
v = c; // correct;  
c = v; compile-time error!
```
- Casting löst das Problem:  

```
c = (Car) v;
```

(nur wenn in v wirklich ein Car-Objekt ist!)

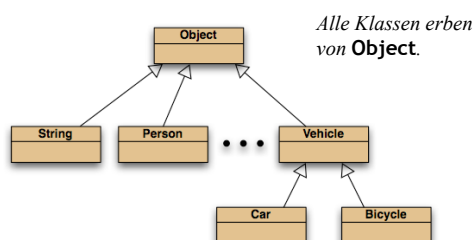
ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Casting

- Ein Objekttyp in Klammern.
- Um 'Typenverlust' zu lösen.
- Das Objekt selbst wird hierbei nicht verändert.
- Ein Runtime-Check wird eingefügt, um zu prüfen, ob das Objekt tatsächlich von angegebenen Typ ist:
  - `ClassCastException` falls nicht!
- Nur in Ausnahmefällen benutzen.

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Die Object-Klasse



ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Polymorphe Sammlungen

- Alte Versionen der Sammlungen sind polymorph.
- Methoden hatten Parameter vom Typ 'Object'.

```
public void add(Object element)
```

```
public Object get(int index)
```

ProInformatik III: Objektorientierte Programmierung, © David J. Barnes, Michael Kölling

## Sammlungen und primitive Typen

- Objekte können in Sammlungen eingefügt werden.
- Gut! Aber was ist mit primitiven Typen?

## Wrapper-Klassen

- Primitive Typen (`int`, `char`, *etc*) sind keine Objekte. Sie müssen in Objekte eingepackt ("wrapped") werden!
- Wrapper-Klassen existieren für alle primitiven Typen:

<i>simple type</i>	<i>wrapper class</i>
<code>int</code>	<code>Integer</code>
<code>float</code>	<code>Float</code>
<code>char</code>	<code>Character</code>
...	...

## Wrapper-Klassen

```
int i = 18;
Integer iwrap = new Integer(i); —— wrap the value
...
int value = iwrap.intValue(); —— unwrap it
```

Tatsächlich müssen wir das nicht häufig tun, da es *autoboxing* und *unboxing* gibt.

## Autoboxing und unboxing

```
private ArrayList<Integer> markList;
...
public void storeMark(int mark)
{
    markList.add(mark); —— autoboxing
}
```

```
int firstMark = markList.remove(0); —— unboxing
```

## Zusammenfassung

- Vererbung erlaubt die Definition einer Klasse als Erweiterung einer anderen Klasse.
- Vererbung
  - vermeidet Code-Duplizierung
  - erlaubt Wiederverwendung von Code
  - vereinfacht den Code
  - vereinfacht Wartung und Erweiterung
- Variablen können Subtype-Objekte speichern.
- Subtype-Objekte können benutzt werden, wo immer Supertype-Objekte erwartet werden (Substitution).