IT I: Heute

- Nachbetrachtung Wissensüberprüfungen
- Einführung Vererbung

Nachbetrachtung Wissensüberprüfung

Gegeben: Klasse *Anlage* mit Methode

boolean bearbeite(Produkt p),

die Produkt p bearbeitet und genau dann true zurückgibt, wenn Bearbeitung möglich.

```
Anlage anlage = ...;
Produkt prod = ...;
if(anlage.bearbeite(prod)){
        anlage.bearbeite(prod);
}
```

Nachbetrachtung Wissensüberprüfung

Gegeben: Klasse *Anlage* mit Methode

boolean bearbeite(Produkt p),

die Produkt p bearbeitet und genau dann *true* zurückgibt, wenn Bearbeitung möglich.

```
Anlage anlage = ...;
Produkt prod = ...;
if(anlage.bearbeite(prod)){
        anlage.bearbeite(prod);  // bearbeitet 2x!
}
```

Vererbung (Inheritance) (1)

- Das Konstrukt der Vererbung erlaubt es, Gemeinsamkeiten in eine *Oberklasse* zusammenzufassen, sodass alle *Unterklassen* über diese gemeinsamen Eigenschaften verfügen.

- Das automatische Übernehmen von Eigenschaften der Oberklasse heißt Vererbung.
  - Die Unterklasse erbt die Eigenschaften der Oberklasse.

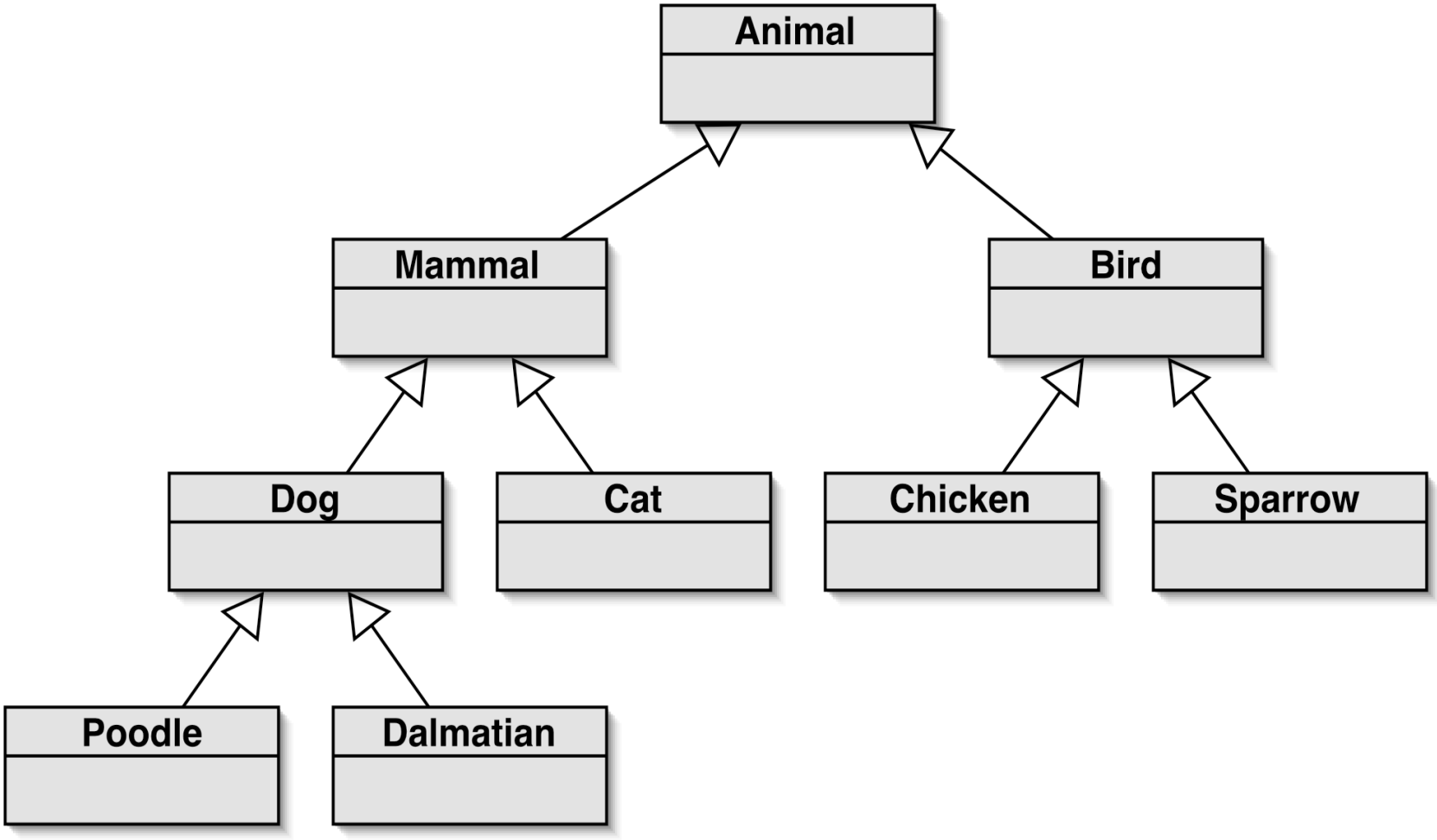
- Gemeinsame Eigenschaften sind vor allem
  - Attribute,
  - Methoden.

Vererbung (Inheritance) (2)

- Zusätzlich zu den Eigenschaften der Oberklasse kann eine Unterklasse weitere Eigenschaften haben.
- Die Unterklasse *erweitert* die Eigenschaften der Oberklasse.
- Erweitern bedeutet hier das Definieren von weiteren Eigenschaften, d.h. die Unterklasse ist eine Spezialisierung der Oberklasse.
- Eine Unterklasse kann Oberklasse einer anderen Klasse sein:

  Es ergibt sich eine Vererbungshierarchie.

Inheritance hierarchies



Vererbung in Java

- Die Oberklasse steht für sich und kennt ihre Unterklassen auch nicht.
- Bei der Definition der Unterklasse wird die Oberklasse angegeben:

```
class Unterklasse extends Oberklasse
{
        …
}
```

- Beim Erzeugen eines Objektes der Unterklasse wird zuerst ein Objekt der Oberklasse erzeugt, das dann um die Attribute der Unterklasse erweitert wird.

Konstruktoren für Unterklassen

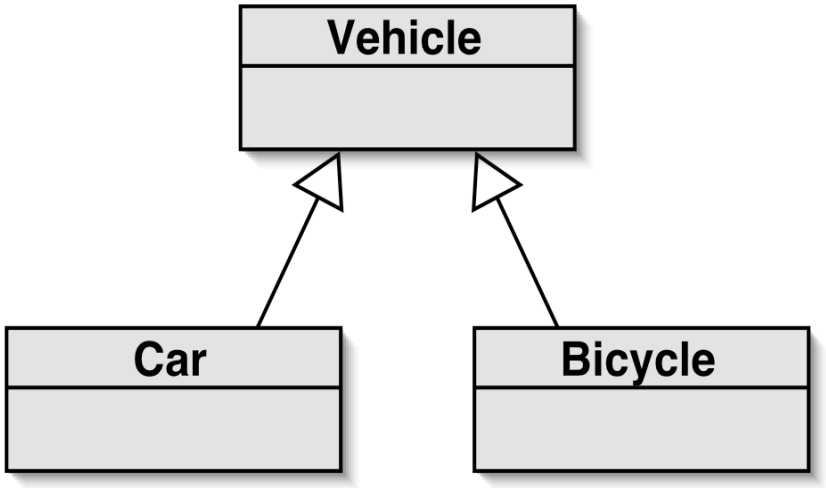
- Ein Konstruktor der Unterklasse muss *als erste Aktion* einen Konstruktor der Oberklasse aufrufen. Dann erst kann der Konstruktor der Unterklasse weitere Aktionen durchführen.
  - Wenn kein Konstruktor der Oberklasse explizit aufgerufen wird, dann wird automatisch der parameterlose Default-Konstruktor der Oberklasse aufgerufen.
  - Das ist weniger empfehlenswert. Besser ist es, immer explizit einen Konstruktor der Oberklasse aufzurufen.
- Der Aufruf des Konstruktors der Oberklasse erfolgt durch:

  super(*Argumentliste*)

Subtyping

- Ein Objekt der Unterklasse kann überall dort verwendet werden, wo ein Objekt der Oberklasse verwendet werden kann.
  - Denn die Unterklasse ist ja eine Spezialisierung der Oberklasse, behält aber alle Eigenschaften der Oberklasse.
- Umgekehrt ist das nicht möglich, denn ein Objekt der Oberklasse hat nicht notwendigerweise alle Eigenschaften der Unterklasse.

Beispiel: Subtyping



subclass objects may be assigned to superclass variables

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

Beispiel: Subtyping

```
Vehicle v;
Car c = new Car();
v = c;   /* correct */
c = v;   /* Error! */
```

- Hier sind wir sicher, dass *v* tatsächlich ein *Car* ist.
- Diesen Umstand können wir dem Compiler durch *Casting* mitteilen:

```
c = (Car)v;   /* correct */
```

Casting

Wenn ein Objekt der Oberklasse vorliegt, der Programmierer aber sicher ist, dass es sich auch um ein Objekt der Unterklasse handelt, kann dies dem Compiler durch *Casting* mitgeteilt werden:

- Der Ausdruck

(*Unterklasse*)*einObjekt*

wird vom Compiler wie ein Objekt der Unterklasse behandelt.

Casting

- Sollte beim Programmablauf *einObjekt* doch kein Objekt der Unterklasse sein, so führt

  (*Unterklasse*)*einObjekt*

  zu einem Runtime error.

- Mit

  einObjekt instanceof Klasse

  kann abgefragt werden, ob *einObjekt* ein Objekt von *Klasse* ist.

- Casting sollte möglichst sparsam eingesetzt werden.

Overloading/Überladen

- In einer Klasse können mehrere Methoden gleichen Namens definiert werden, sofern sie sich in den Parameterlisten unterscheiden.
- Die Methoden können ganz unterschiedliche Aktionen ausführen.
  - Das empfiehlt sich aber aus Gründen der Verständlichkeit nicht.
  - Vielmehr werden u.U. Methoden ähnlicher Bedeutung aber mit verschiedenen Parametern mit demselben Namen versehen.
- Das Verwenden gleicher Methodennamen für unterschiedliche Methoden nennt man Overloading.
- Es kann für eine Klasse auch mehrere Konstruktoren geben, sofern sie sich in den Parameterlisten unterscheiden.

Überschreiben (Overriding)

- Das Neudefinieren einer Methode der Oberklasse in einer Unterklasse heißt Überschreiben (*Overriding*).

- Die Methode der Oberklasse kann in der Unterklasse mittels

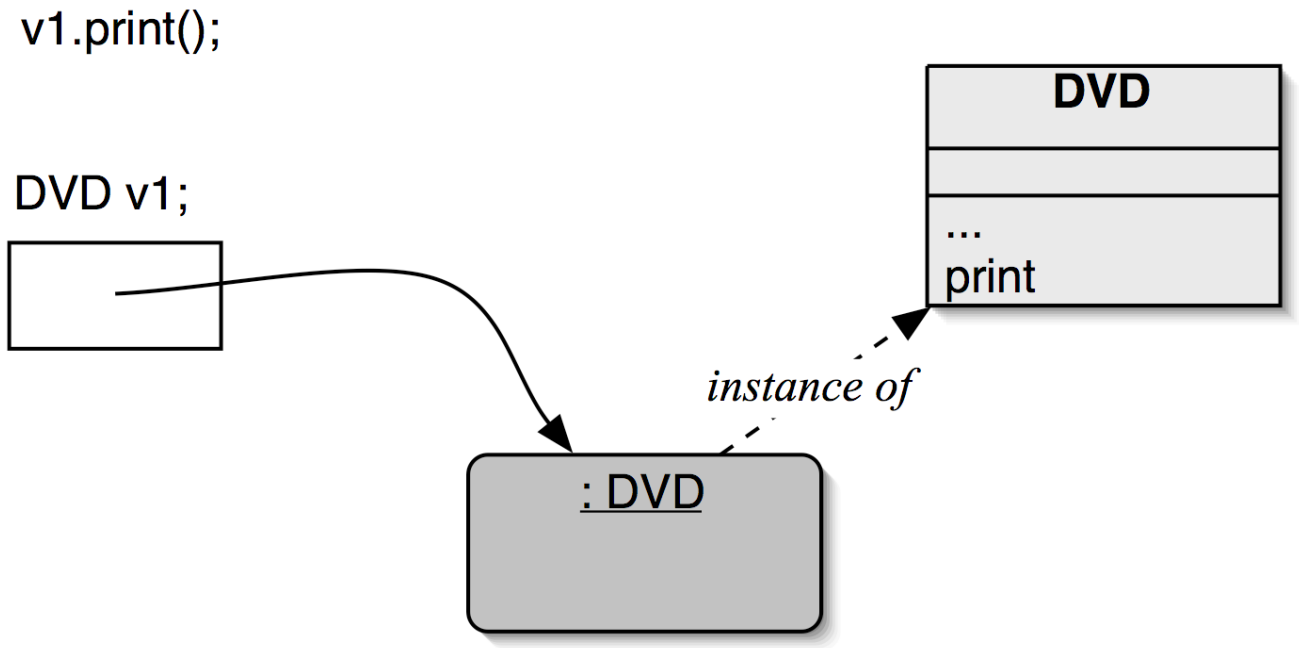
  super.*methodenname*()

  aufgerufen werden.
  - Das ist oft nützlich, wenn die Unterklasse den Aktionen der Oberklasse nur etwas hinzufügen möchte.

Welche Methode wird nun ausgeführt?

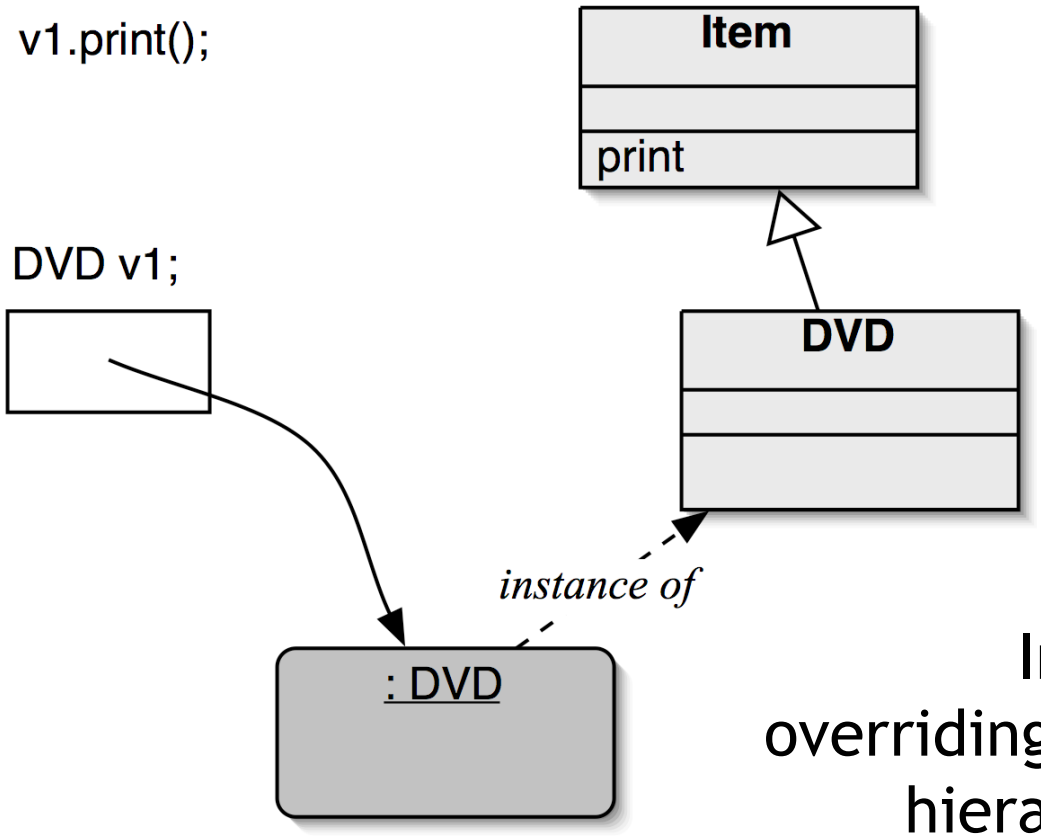
- Wird für ein Objekt eine Methode aufgerufen, wird die Methode ausgeführt, die in der Klassenhierarchie des Objekts am weitesten unten definiert ist.

Lookup of own method



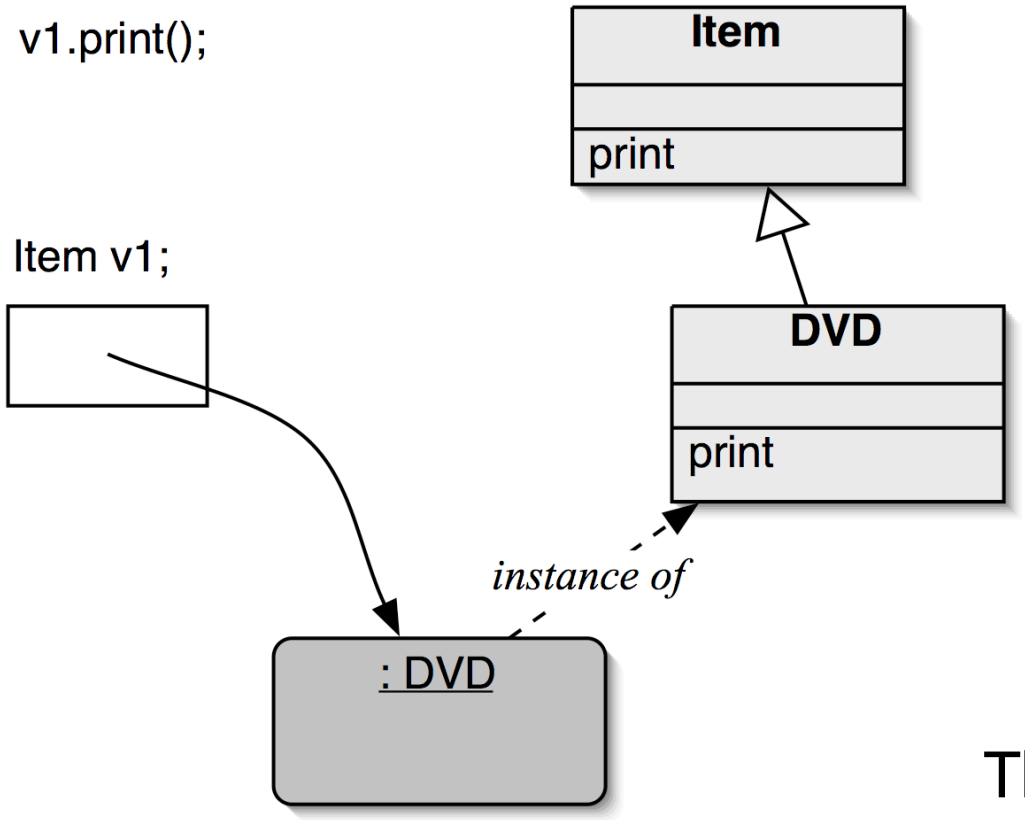
No inheritance.
The obvious method is selected.

Lookup of inherited method



Inheritance but no overriding. The inheritance hierarchy is ascended, searching for a match.

Lookup of overridden method



Overriding:
The ‘first’ version found is used.