Was bisher geschah...

```
byte zahl = 7;
```

// Die Bits für 7 werden
// in die Variable gesteckt


```
Gummibär cubbi = new Gummibär ();
```

Gummibär

// Die Klasse Gummibär ist der **Bauplan**
// für ein Gummibär-Objekt

// **new** erzeugt ein neues Objekt

// Die Bits für den **Referenzwert**
// werden in die Referenzvariable gesteckt

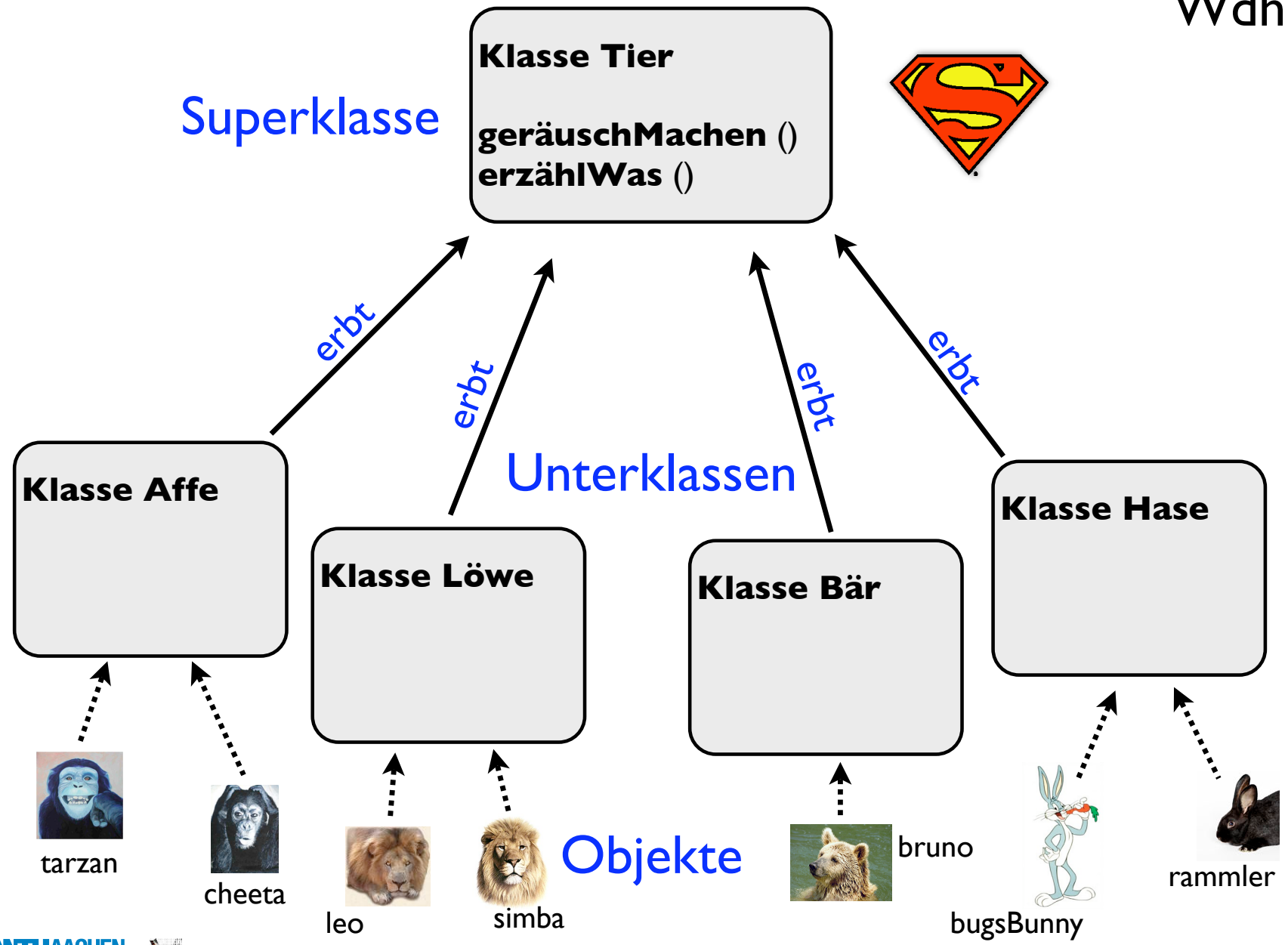
// Objekte leben auf dem Heap Speicher
// und nicht in einer Variablen !

```
class Flugzeug {
  private int anzahlMotoren;
  private int reichweite;
  ...
  public void setReichweite () {...}
  public void beschleunigen () {...}
}
```

private Instanzvariablen

öffentliche Methoden

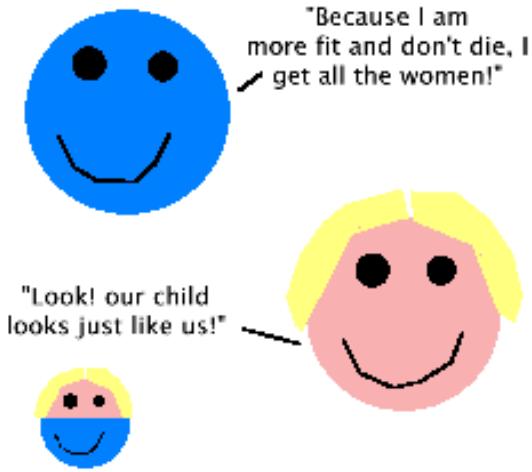
```
Flugzeug eineA380 = new Flugzeug ();
Flugzeug eineB787 = new Flugzeug ();

eineA380.setReichweite (15000);   // Jedes Objekt hat seine eigenen
eineB787.setReichweite (15700);   // Instanzvariablen

eineA380.beschleunigen ();        // Alle Objekte teilen sich
eineB787.beschleunigen ();        // die Methoden der Klasse
```


Superklasse
Klasse Tier
geräuschMachen ()
erzählWas ()
erbt
erbt
erbt
erbt
Unterklassen
Klasse Affe
Klasse Löwe
Klasse Bär
Klasse Hase
tarzan
cheeta
leo
simba
Objekte
bruno
bugsBunny
rammler

Vererbung

&

Polymorphie



Die Superklasse

...abstrahiert die Gemeinsamkeiten mehrerer verwandter Unterklassen.

Die **Member** der Superklasse sind wie immer die Instanzvariablen und die Methoden.

```
class Tier {
  private int anzahlBeine;
//...
  public void setAnzahlBeine (...) {...}
  public void geräuschMachen () {...}
  public void schlafen () {...}
}
```

Die Unterklasse

...ist konkret. Sie erbt die "Member" der Superklasse.

```
class Affe extends Tier {
  // Alles was ein Tier kann, das kann ich auch!
}

class Loewe extends Tier {
  // Dito.
  // Ich muß dafür keine Zeile Code schreiben!
}
```

Wer ist Superklasse, wer ist Unterklasse?

Die Unterklasse "IS-A" Superklasse!

```
Tier IS-A Affe ?        // leider nein
Affe IS-A Tier ?        // ok
tarzan IS-A Affe ?      // ja, aber tarzan ist ein Objekt !!!

Zoo HAS-A Affe ?        // Affe als Instanzvariable benutzt,
                        // keine Vererbung!
```

Zoo "HAS-A-Verhältnis" zu Affe

Wer ist Superklasse, wer ist Unterklasse?

Superklasse

```
class Tier
private int anzahlBeine;
//...
public void setAnzahlBeine (...) {...}
public void geräuschMachen () {...}
public void schlafen () {...}
```

IS-A

IS-A

```
class Affe extends Tier
// erbt die Member von Tier, d.h.
// Instanzvariablen und Methoden
```

Unter-klassen

```
class Löwe extends Tier
// erbt auch die Member von Tier
```

IS-A

tarzan

IS-A

cheeta

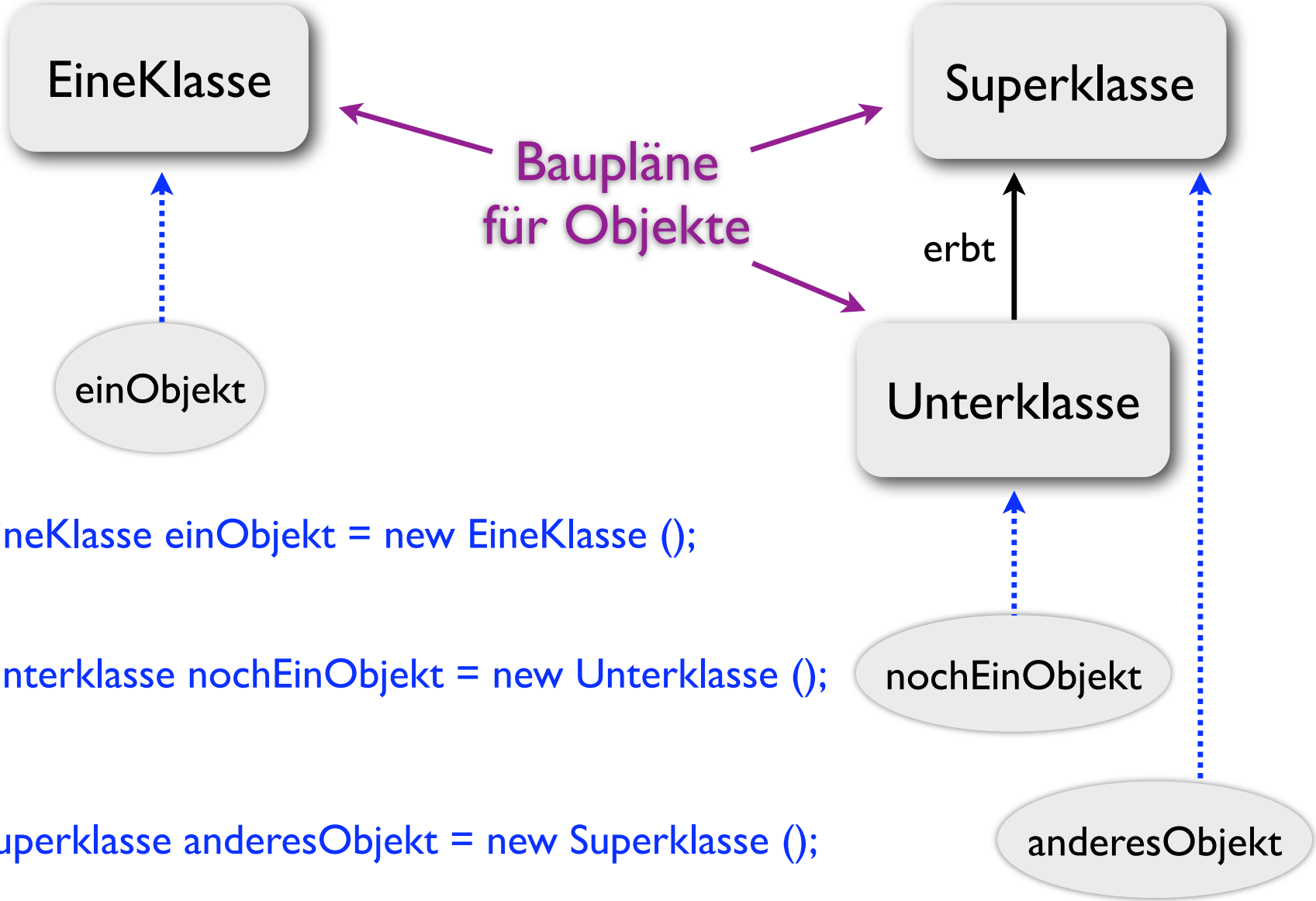
Objekte

IS-A

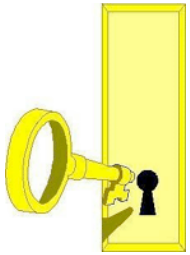
simba

IS-A

leo


EineKlasse
Superklasse
Baupläne
für Objekte
erbt
einObjekt
Unterklasse
EineKlasse einObjekt = new EineKlasse ();
Unterklasse nochEinObjekt = new Unterklasse ();
nochEinObjekt
Superklasse anderesObjekt = new Superklasse ();
anderesObjekt

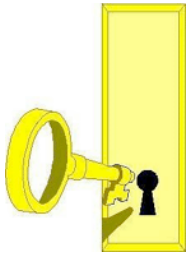
Zugriffsmodifier für Klassen



public	// sichtbar für alle Klassen überall
default	// sichtbar für alle Klassen
(ohne Modifier)	// im gleichen Paket

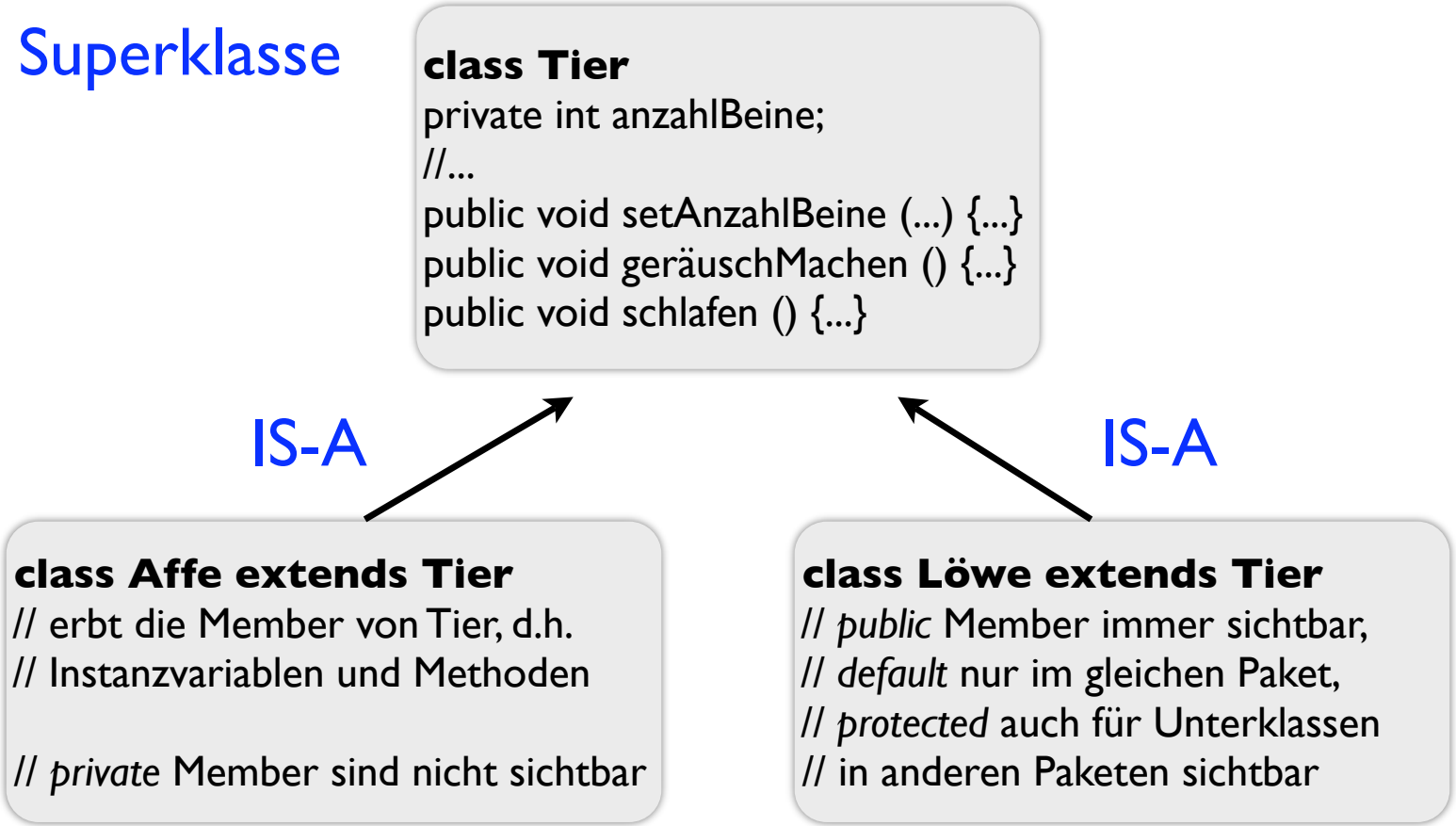
```
public class KlasseÜberallSichtbar {
}

class KlasseImPaketSichtbar {
}
```

Zugriffsmodifier für Instanzvariablen und Methoden



public	// sichtbar für alle Klassen überall
protected	// für alle Klassen im gleichen Paket // und für Unterklassen ausserhalb
default (ohne Modifier)	// für alle Klassen im gleichen Paket
private	// nur in der definierenden Klasse

Zugriff auf Member aus der Unterklasse



Geht das?

```
class Tier
private int anzahlBeine;
//...
public void setAnzahlBeine (...) {...}
public void geräuschMachen () {...}
public void schlafen () {...}
```


```
affe.setAnzahlBeine (2);        // Instanzvariablen + Methoden
affe.schlafen ();               // sind in der Superklasse definiert

affe.anzahlBeine = 2;           // nein, private Member sind
                                // unsichtbar für die Unterklasse

löwe.setAnzahlBeine (4)         // ja, public Member sind
löwe.geräuschMachen ();         // sichtbar für die Unterklasse
```

```
Modifier1 class Tier {
  Modifier2 int anzahlBeine;

  // Setter
  Modifier3 setAnzahlBeine (int anzahl) {
    anzahlBeine = anzahl;
  }

  // Getter
  Modifier4 getAnzahlBeine () { return anzahlBeine; }
  }
}
```

SMS mit Ergebnis an:

Die SMS soll genau 4 Modifier in der richtigen Reihenfolge enthalten: private? protected? default? public?

Aufgabe: Alle Klassen überall dürfen den Wert von anzahlBeine nur abfragen, aber alle Unterklassen überall sollen den Wert setzen können.

```
public class Tier {
  private int anzahlBeine;

  // Setter
  protected setAnzahlBeine (int anzahl) {
    anzahlBeine = anzahl;
  }

  // Getter
  public getAnzahlBeine () { return anzahlBeine; }
  }
}
```

SMS mit Ergebnis an:

Die SMS soll genau 4 Modifier in der richtigen Reihenfolge enthalten: private? protected? default? public?

Aufgabe: Alle Klassen überall dürfen den Wert von anzahlBeine nur abfragen, aber alle Unterklassen überall sollen den Wert setzen können.

Die Unterklasse

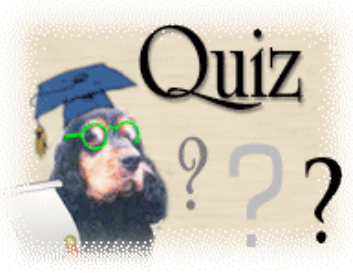
...ist spezifisch. Sie überschreibt geerbte Methoden.

```
class Hase extends Tier {
  public void geräuschMachen () {
    // Die geerbte geräuschMachen()-Methode
    // von Tier ist für einen Hasen wie mich unpassend!
    // Ich schreibe sie einfach neu.
  }
```

Welche Methode wird aufgerufen?

Die spezifischste Methode gewinnt!

```
class Tier
private int anzahlBeine;
private String nahrung;
//...
public void geräuschMachen () {...};
public void schlafen () {...};
```



```
class Affe extends Tier
```

```
class Hase extends Tier
public void geräuschMachen () {...}
```

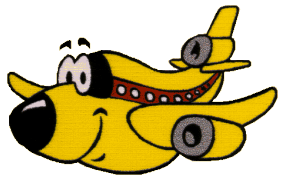
```
Affe tarzan = new Affe ();
tarzan.geräuschMachen ();        // ruft Methode aus Tier auf

Hase bunny = new Hase ();
bunny.geräuschMachen ();         // ruft Methode aus Hase auf
```

Superklassen-Methoden aufrufen

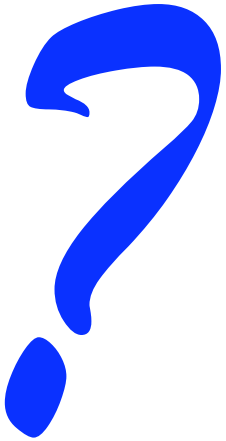
Oft erledigen Superklassen Arbeit, die man selber nicht erledigen kann oder darf.

```
class Hase extends Tier {
  public void geräuschMachen () {
        // lästige Sound-Hardware-Initialisierung
        // von der Superklasse erledigen lassen
        super.geräuschMachen ();
        // Und nun das hasenspezifische Geräusch...
  }
```

Entwerfen Sie einen Vererbungsbaum!



Superklasse (abstrakt)
Unterklasse (konkret)



Superklasse natürlich!

Was wären dann mögliche Unterklassen?

Verkehrsflugzeug

Überschallflugzeug

Segelflugzeug

Militärflugzeug

Boeing

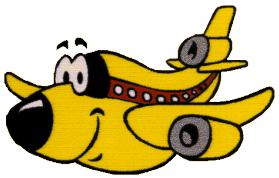
Modellflugzeug

Airbus

F16 Falcon

...

```
class Flugzeug
private String typ;
private float abrißGeschwindigkeit;
//...
public void abheben () {...};
public void landen () {...};
```



Gibt es weitere Abstraktionsmöglichkeiten

Verkehrsflugzeug

Überschallflugzeug

Segelflugzeug

Militärflugzeug

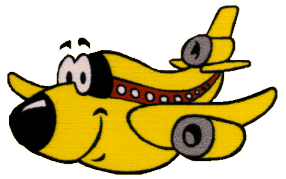
Boeing

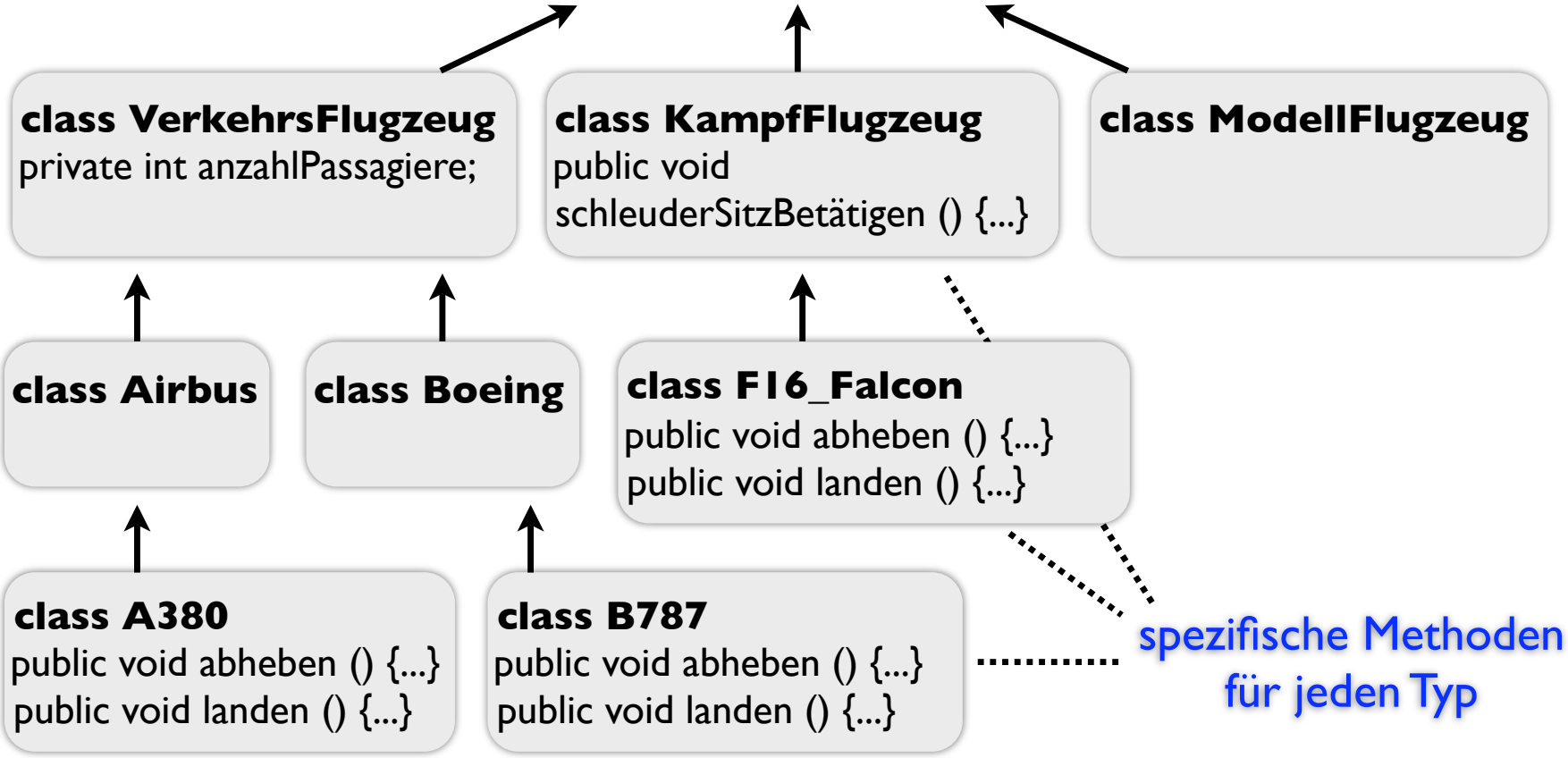
Modellflugzeug

Airbus

F16 Falcon

Stimmt der IS-A-Test?





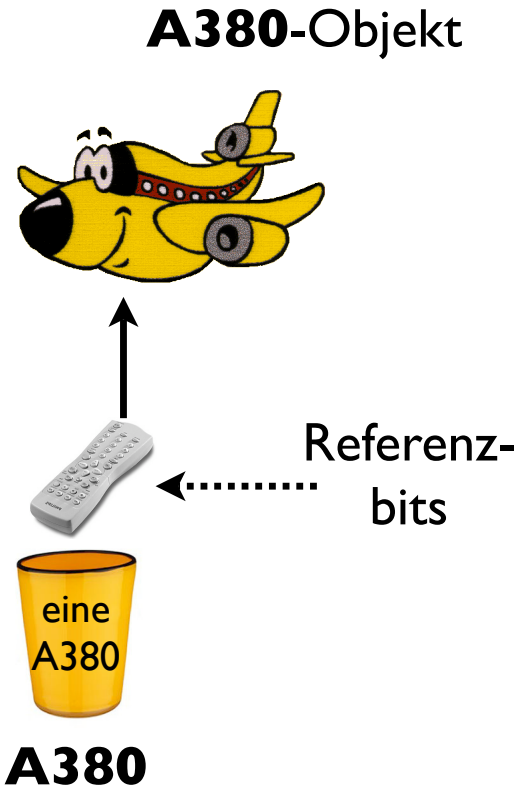
Objektdeklaration und Zuweisung

A380 eineA380 = new **A380** ();

Referenzvariable deklarieren

Objekt erzeugen

Referenztyp und Objekttyp sind gleich!

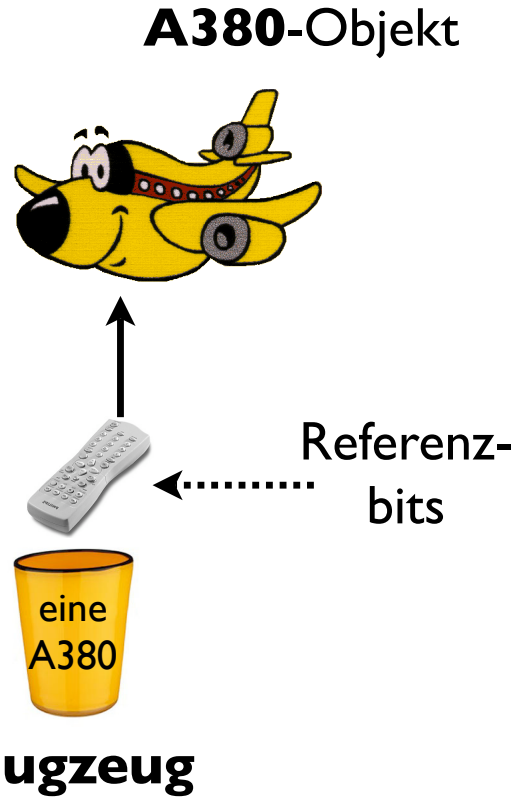


Polymorphie

Flugzeug eineA380 = new **A380** ();

Referenzvariable deklarieren

Objekt erzeugen



Referenztyp und Objekttyp sind unterschiedlich!

A380 "IS-A" Flugzeug.

A380 ist eine Unterklasse von Flugzeug.

Polymorphe Arrays

```
// Array des Typs Flugzeug
Flugzeug[] flieger = new Flugzeug[5];

// beliebige Unterklasse von Flugzeug
// in das Flugzeug-Array stecken
flieger[0] = new A380 ();                  // A380-Objekt
flieger[1] = new B787 ();                  // B787-Objekt
flieger[2] = new F16_Falcon ();            // F16_Falcon-Objekt
flieger[3] = new ModellFlugzeug ();        // Modellflugzeug-
flieger[4] = new Flugzeug ();              // Objekt
```

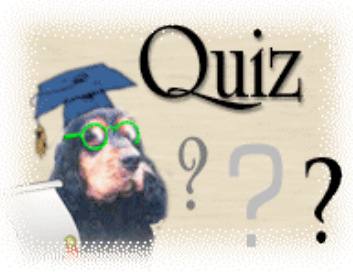
Polymorphe Arrays


```
// Flugzeug-Array durchlaufen
for (int i = 0; i < flieger.length; i++) {
  flieger[i].abheben ();
  flieger[i].landen ();
}
```

```
// Welche Methode wird aufgerufen?
flieger[0].abheben ();          // abheben() von A380
flieger[2].landen ();           // landen() von F16_Falcon
```

Die überschreibende Methode des Objekts wird aufgerufen!

Polymorphe Arrays




```
Flugzeug[] flieger = new Flugzeug[5];
flieger[0] = new A380 ();          // A380 "IS-A" Flugzeug
flieger[1] = new B787 ();          // B787 "IS-A" Flugzeug
flieger[2] = new F16_Falcon ();    // F16... "IS-A" Flugzeug

// Geht das ?
flieger[0].schleuderSitzBetätigen ();      // nein, gibt's nicht
flieger[2].schleuderSitzBetätigen ();      // nein
```

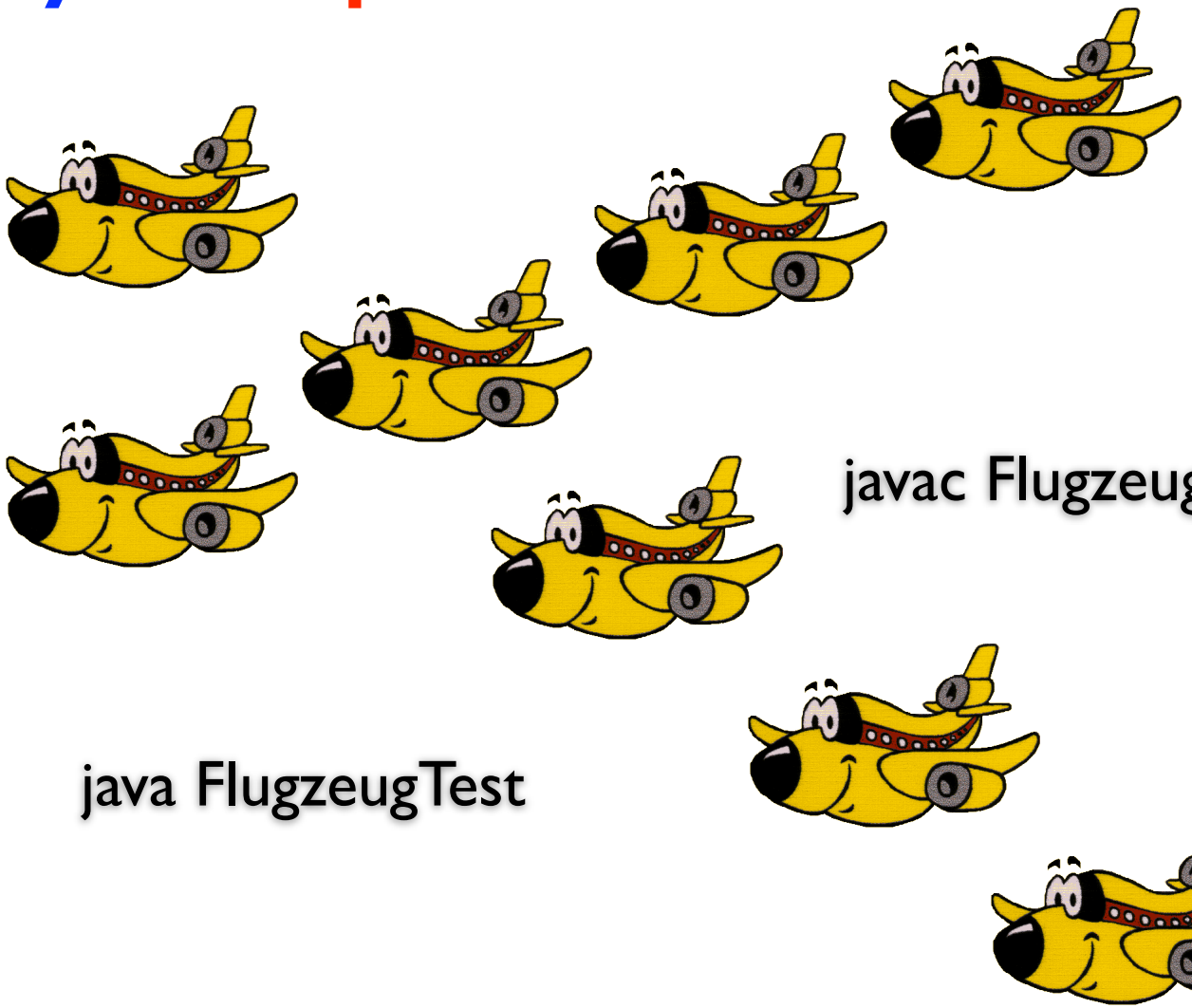
Die Klasse Flugzeug weiß nichts von schleudersitzBetätigen() !!!

Polymorphe Argumente und Rückgabetypen

```
class Werkstatt {
   public void motorWarten (Flugzeug einFlieger) {
      // Motor warten und Testflug unternehmen
      einFlieger.abheben();
      einFlieger.landen();
   }
}
```

```
Werkstatt pitStop = new Werkstatt ();
for (Flugzeug einFlugObjekt :  flieger) {
   pitStop.motorWarten (einFlugObjekt);
}
```

Polymorphe Demo



javac FlugzeugTest.java

java FlugzeugTest

Sind alle Klassen erweiterbar?

```
public class Flugzeug {...}      // von allen Klassen überall
                                 // erweiterbar

class Flugzeug {...}             // nur von Klassen im
                                 // gleichen Paket erweiterbar

final class A380 {...}           // Klassse nicht erweiterbar

class Flugzeug {                 // Klasse ist erweiterbar,
  final void abheben () {...}    // aber abheben () ist nicht
}                                // überschreibbar
```

Methoden überschreiben vs. Methoden überladen

```
class Flugzeug {
   void autopilotFliegen () {...}
}
```

// soll vererbt werden

```
class Airbus extends Flugzeug {
   void autopilotFliegen () {...}
}
```

// wird überschreiben

```
class Airbus extends Flugzeug {
   void autopilotFliegen (float kurs) {...}
}
```

// wird überladen

Methoden überschreiben vs. Methoden überladen

```
class A380 extends Airbus {
   boolean autopilotFliegen () {...}
}
```

// nur Rückgabetyp
// ändern geht nicht !

```
class A380 extends Airbus {
   boolean autopilotFliegen (float kurs) {...}
}
```

// ok, überladen

```
class A380 extends Airbus {
   private void autopilotFliegen () {...}
}
```

// Zugriff einschränken
// geht nicht !

Methoden

zu **überladen** hat nichts mit Vererbung

oder mit Polymorphie zu tun!

```
class Werkstatt {

   public void motorWarten (Flugzeug einFlieger) {...}
   public void motorWarten (Flugzeug flieger1, Flugzeug flieger2) {...}
   public boolean motorWarten (Flugzeug[] flieger) {...}

}
```

Überladene Methoden haben nur den gleichen Namen. Sie dienen der Bequemlichkeit und sparen Arbeit für den Aufrufer.

Jetzt sind Sie wieder an der Reihe!

Mehr zu Vererbung und Polymorphie finden Sie in **Kapitel 7**.