
FACHGRUPPE INFORMATIK
RWTH AACHEN UNIVERSITY
TDI 2.0 06
Die Fachgruppe Informatik der RWTH Aachen lädt ein zum
TAG DER INFORMATIK 2006
Freitag, 1. Dezember 2006
Informatikzentrum, Aula 2
Ahornstraße 55, 52074 Aachen
http://www.tdi2006.de
Vorträge — Ausstellung — Studienabschlussfeier
09:00 Doktorandenvorträge — Aktuelles aus unserer Forschung
12:30 Beginn des Hauptprogramms, Vorstellung neuer Professoren
14:45 Hauptvorträge
Rewind the Future. On the deflationary time currency and generational shifts.
Herman Konings
Pocket Marketing/nXt, Antwerpen
Fahrerassistenzsysteme: Herausforderungen für System- und Softwareentwurf
Dr. Thomas Kropf
Robert Bosch GmbH
16:30 Preis- und Stipendienverleihung
18:00 Zeugnisüberreichung
Festrede von Prof. em. Dr. Klaus Indermark
19:15 Abschlussfeier mit Buffet und Musik
Mit freundlicher Unterstützung von
AMB GENERALI
IVU Traffic Technologies AG
Sun
dSPACE
VEMAC
ERICSSON
ascom
Microsoft
SMART
accenture
INFORM
LANCOM
QSC
LAMBERT-Z

Diese Vorlesung (V2/Ü1): 4 ECTS

1 Credit = 30 Std.

4 Credits = 120 Std. (pro Person!)

13 Vorlesungen x 90 min = 19,5 h

13 Übungen x 45 min = 9,75 h

⇒ 90,75 h ≈ 7 h / Woche bleiben übrig

Lesen: 3 min / Seite x 30 Seiten / Kapitel = 1,5 h

Übungsblatt: mindestens 4 h / Woche

⇒ 1,5 h / Woche bleiben übrig

Schein-/Bachelorklausur Teil 1 am 15.12.2006

Zulassung: 50% der Übungspunkte aus den Übungen 1-6

Alle Studiengänge melden sich für die Klausur über die Lehrstuhlwebseite zwischen dem 04-08.12.2006 an. Anmeldeschluß ist Fr. 08.12.2006 um 17:00 Uhr.

http://media.informatik.rwth-aachen.de/programmierung_ws0607.html

Bitte unbedingt einen gültigen Studentenausweis und Personalausweis zu der Klausur mitbringen!

Bachelorklausur Teil 2 am 13.02.2007:
Alle Bachelorstudenten müssen sich bereits im Dezember anmelden.

Vordiplom-/ Zwischenprüfungsklausur am 27.03.2007:
Die Anmeldung findet ebenfalls im Dezember statt.

Beachten Sie unbedingt die Anmeldetermine beim ZPA bzw. auf der Lehrstuhlwebseite.

http://www-zhv.rwth-aachen.de/zentral/abt13_index.htm

http://media.informatik.rwth-aachen.de/programmierung_klausuren.html

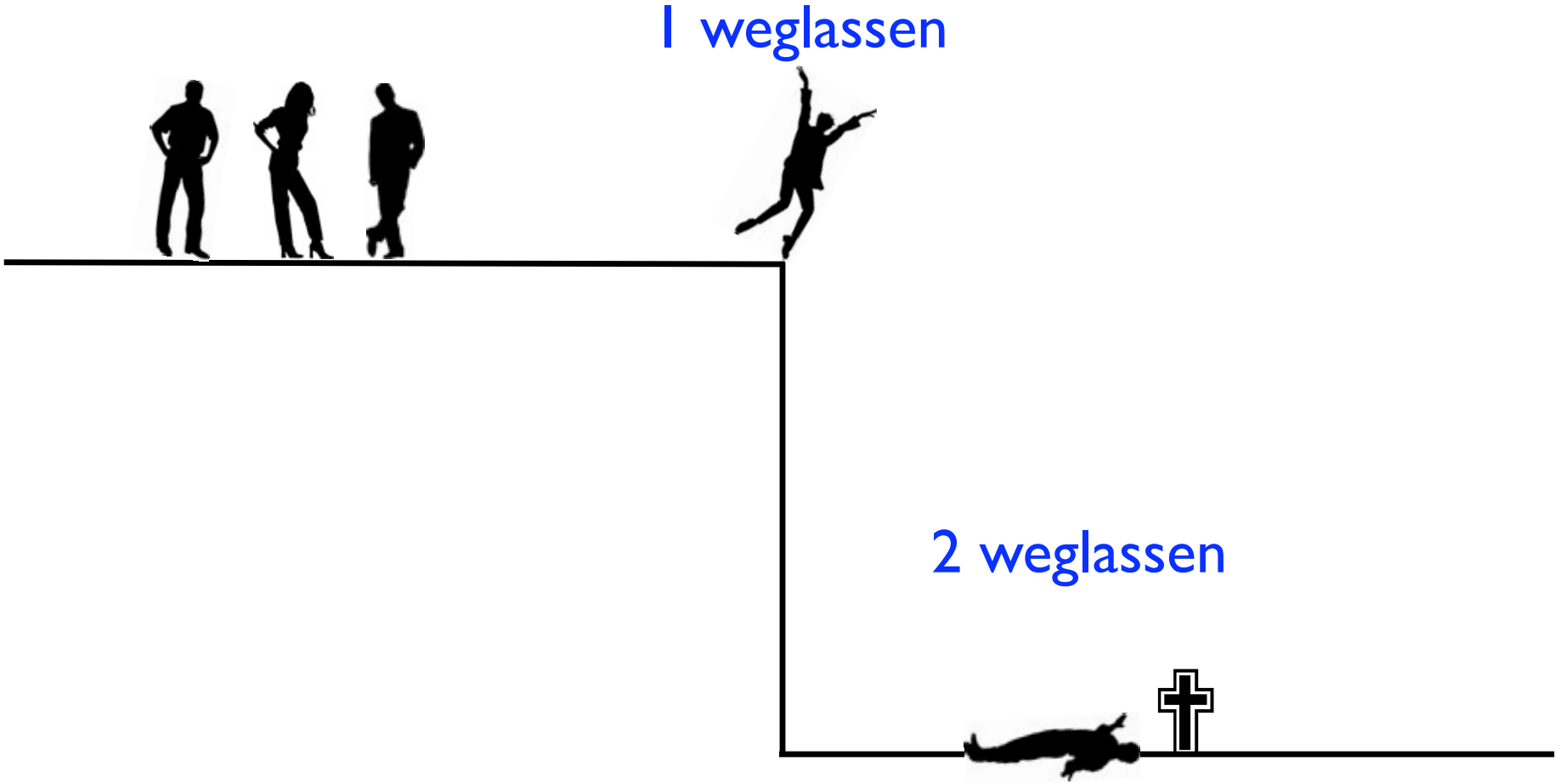
Sie müssen **beide** Schein-/Bachelor Teilklausuren bestehen.

In jeder Teilklausur müssen Sie mindestens
25% der Punkte erreichen.

In beiden Teilklausuren zusammen müssen Sie mindestens
50% der Gesamtpunkte **aus beiden Klausuren**
erreichen.

25% in jeder Teilklausur
!=
50% in beiden Teilklausuren

V + Ü + ÜB + Buch



Übungen immer abgeben, auch wenn Sie 50% der Übungspunkte erreicht haben!

Übungsgruppen
(Sie!)

zu wenige Übungsabgaben

RWTH

keine Tutoren!

Rückblick

Unterschied zw. Klasse und Objekt ?

Unterschied zw. Superklasse und Unterklasse ?

Member einer Klasse?

Zugriffsmodifier für Member?

Zugriffsmodifier für Klassen?

Methoden überschreiben?

Methoden der Superklasse aufrufen?

Was ist Polymorphie?

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() !!!

Einschub für Geeks

Arrays und Klammern

```
Flugzeug[] flieger = new Flugzeug[5];
Flugzeug flieger[] = new Flugzeug[5];
Flugzeug [] [] flieger = new Flugzeug[2][5];
Flugzeug [] flieger [] = new Flugzeug[2][5];
```

```
int ergebnis () [] { return new int[200]; }
```

```
int[] ergebnis () {
   return new int[200];
}
```

// [] - Klammern neben den Typ
// zu schreiben ist übersichtlicher!

Einschub für Geeks

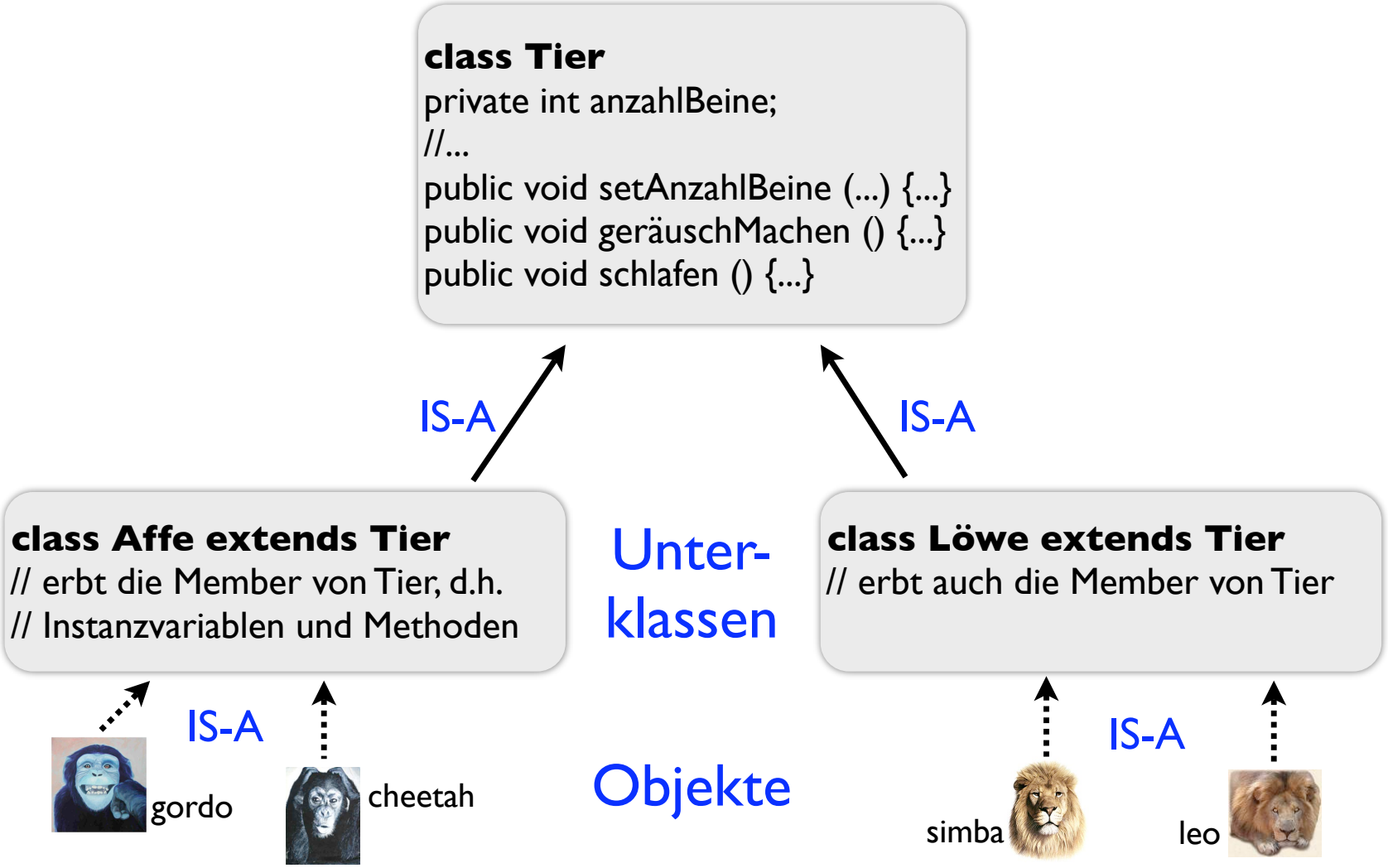
geschweifte Klammern

```
if (a == b)                    // { } - Klammern
  a++;                         // können bei einzelnen
else                           // Anweisungen fehlen
  a--;

while (a < 10) a++;
do a++; while (a < 10);
for (a = 0; a < 10; a++) b--;
```

verkürzte Schreibweise für Operatoren

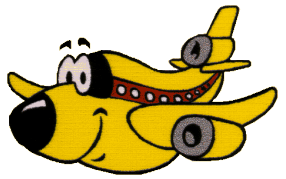
```
a += b;  a -= b;                    // a = a + b;  a = a - b;
a /= b;  a *= b;  a %= b;
```

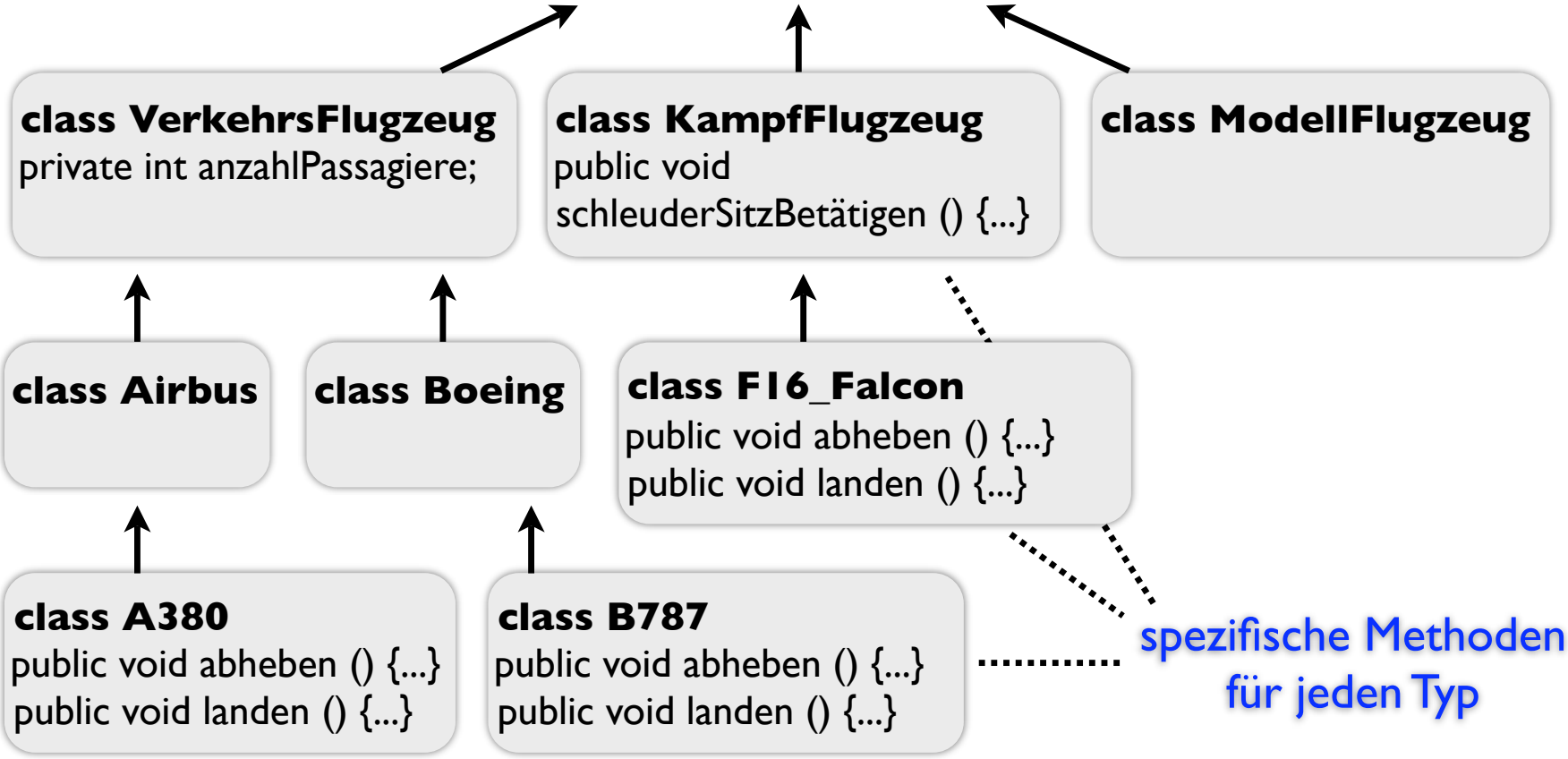
Wdh.


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
class Löwe extends Tier
// erbt auch die Member von Tier
Unter-klassen
IS-A
IS-A
gordo
cheetah
Objekte
simba
leo

Wdh.

Stimmt der IS-A-Test?





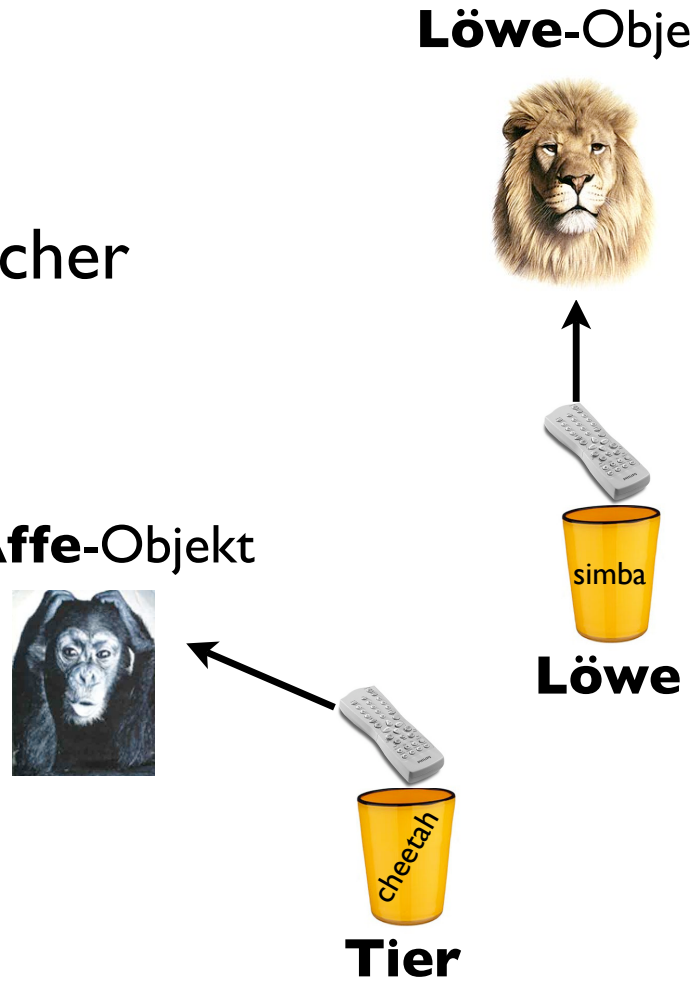
// gleicher Referenz- und Objekt-Typ

Löwe simba = new Löwe ();

// Polymorphie: unterschiedlicher
// Referenz- und Objekt-Typ

Tier cheetah = new Affe ();

Affe "IS-A" Tier !!!



Aus Spaß wurde

ernsthafte

Polymorphie

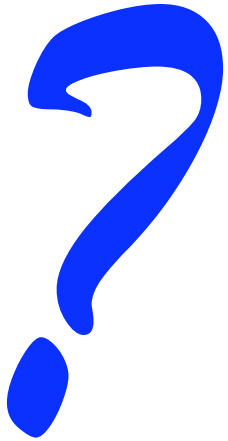
+ Interfaces

abstrakte Objekte ?

// gleicher Referenz- und Objekt-Typ

Tier tier = **new Tier ()**;

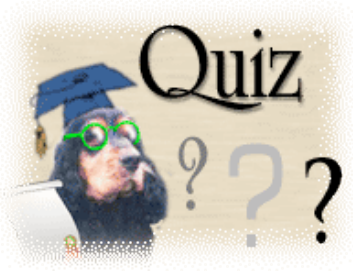
Flugzeug flieger = **new Flugzeug ()**;



// Wie bitte sieht ein Tier / Flugzeug aus?

// Welche Werte haben die Instanzvariablen?

abstrakte Klassen




```
abstract public class Tier { //... }
abstract public class Flugzeug { //... }

// Tier kann nicht mehr instantiiert werden
Tier tier = new Tier ();          // Compiler-Fehler !!!

// geht das?
Tier[] tiere = new Tier[5];
```

Ja, denn wir erzeugen ein Array-Objekt vom Typ Tier und keine individuellen Tier-Objekte!

abstrakt
oder
konkret?

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

class VerkehrsFlugzeug
private int anzahlPassagiere;

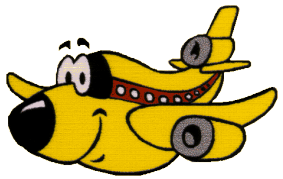
class KampfFlugzeug
public void
schleuderSitzBetätigen () {...}

class ModellFlugzeug

class Airbus

class Boeing

class F16_Falcon
public void abheben () {...}
public void landen () {...}



class A380
public void abheben () {...}
public void landen () {...}

class B787
public void abheben () {...}
public void landen () {...}

abstrakt
oder
konkret?

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

abstract class VerkehrsFlugzeug
private int anzahlPassagiere;

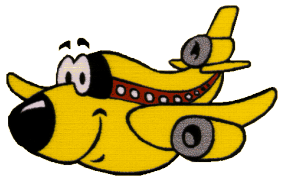
abstract class KampfFlugzeug
...

class ModellFlugzeug

abstract class Airbus

abstract class Boeing

class F16_Falcon
public void abheben () {...}
public void landen () {...}



class A380
public void abheben () {...}
public void landen () {...}

class B787
public void abheben () {...}
public void landen () {...}

abstrakt / konkret
hängt vom Entwurf ab!

Abstrakte Methoden

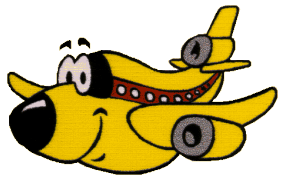
```
abstract public class Flugzeug {
   private String typ;
   private int abrißGeschwindigkeit;
   ...
   public void setTyp (String marke) { typ = marke; }
   public String getTyp () { return typ; }
   ...
   public abstract void abheben ();  // ohne { }
   public abstract void landen ();    // d.h., ohne Rumpf
}
```

Abstrakte Klassen müssen erweitert werden!

Abstrakte Methoden müssen überschrieben werden!

abstrakt
oder
konkret?

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



```
abstract class VerkehrsFlugzeug
private int anzahlPassagiere;
public setAnzahlPassagiere (int anzahl) {...}
public int getAnzahlPassagiere () {...}
```

```
abstract
class Airbus
```

```
class A380
public void abheben () {...}
public void landen () {...}
```

Die erste konkrete Unterklasse muß alle abstrakten Methoden implementieren!

Die Magie der **ArrayList**

```
// speichert Referenzen auf A380-/Boeing/F16/Modellfl.-Objekte
ArrayList<Flugzeug> alleFlugzeuge = new ArrayList<Flugzeug> ();

// speichert Referenzen auf Affen, Löwen, Bären, Hasen-Objekte
ArrayList<Tier> alleTiere = new ArrayList<Tier> ();

A380 eineA380 = new A380 ();
B787 eineB787 = new B787 ();
Löwe simba = new Löwe ();

alleFlugzeuge.add (eineA380);
alleFlugzeuge.add (eineB787);
alleTiere.add (simba);
```

Wieso kann ArrayList mit allen beliebigen Typen umgehen, die wir erzeugen?

Object: Die ultimative Superklasse

Einige der Methoden von ArrayList:

boolean add (**Object** elem)

int indexOf (**Object** elem)

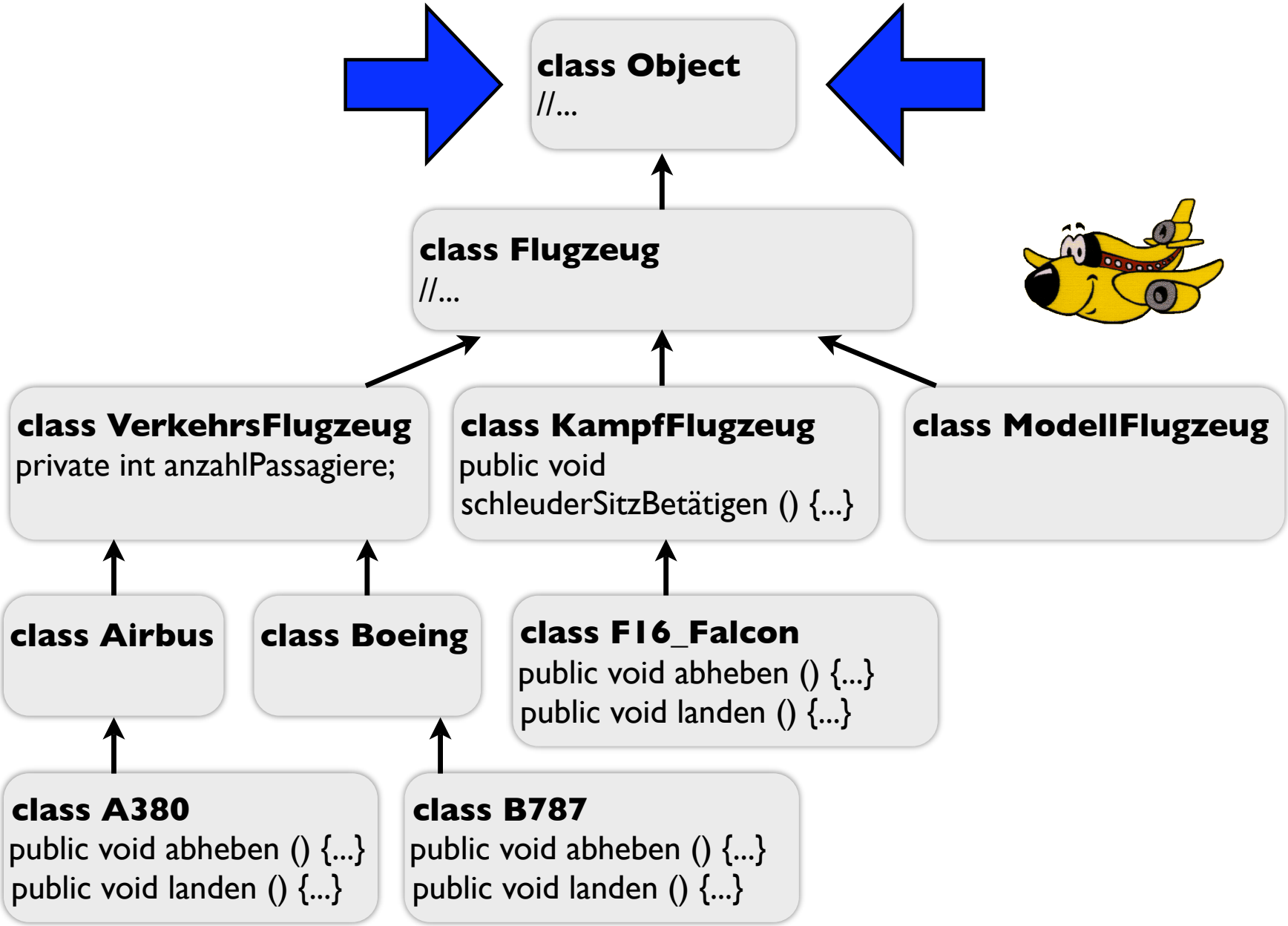
boolean remove (**Object** elem)

ergänzt Java implizit für Sie

public class Tier **extends Object** { }

public class Flugzeug **extends Object** { }

Jede Klasse erweitert implizit Object, wenn sie nicht explizit eine andere Klasse erweitert.


class Object
//...
class Flugzeug
//...
class VerkehrsFlugzeug
private int anzahlPassagiere;
class KampfFlugzeug
public void
schleuderSitzBetätigen () {...}
class ModellFlugzeug
class Airbus
class Boeing
class F16_Falcon
public void abheben () {...}
public void landen () {...}
class A380
public void abheben () {...}
public void landen () {...}
class B787
public void abheben () {...}
public void landen () {...}

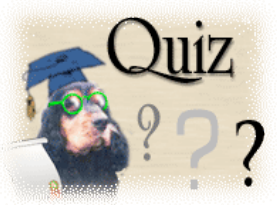
Object: Die ultimative Superklasse

Einige Methoden der Klasse Object:

```
String toString () {...}
Class getClass (Object anderesObjekt) {...}
...
```

```
A380 eineA380 = new A380 ();     // unsere Objekte erben die
B787 eineB787 = new B787 ();     // Methoden der Klasse Object

System.out.println (eineA380.toString ());   // A380@7d277f
System.out.println (eineA380.getClass ());   // A380 (kein String!)
```

Referenzen vom Typ Object




```
ArrayList<Object> alleFlugzeuge = new ArrayList<Object> ();
A380 eineA380 = new A380 ();
alleFlugzeuge.add (eineA380);                    // eineA380 an Pos. 0

// geht das?
A380 dieselbeA380 = alleFlugzeuge.get (0);       // nein !!!
dieselbeA380.abheben ();                         // nein !!!
dieselbeA380.landen ();                          // nein !!!

Object auchDieselbeA380 = alleFlugzeuge.get (0); // das geht
auchDieselbeA380.abheben ();   // nein !!! (siehe Kap. 7, Folie 26)
```

ArrayList<Object> liefert Object-Referenzen zurück !!!

Referenzen vom Typ A380

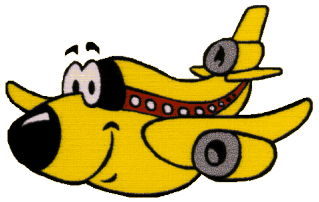
```
ArrayList<A380> alleA380 = new ArrayList<A380> ();
A380 eineA380 = new A380 ();
alleA380.add (eineA380);                    // eineA380 an Pos. 0

A380 dieselbeA380 = alleA380.get (0);       // ok, Typ A380
dieselbeA380.abheben ();                    // ok
dieselbeA380.landen ();

Object auchDieselbeA380 = alleA380.get (0); // Polymorphie
auchDieselbeA380.abheben ();                // nein !!!
```

Die Methoden, die wir auf ein Objekt aufrufen können, hängen vom Typ der Referenz-Variablen ab, mit dem das Objekt referenziert wird!

kennt Löwen- und Tier- und Object-Methoden

kennt Gummibär- und Object-Methoden

kennt Flugzeug- und Object-Methoden



ArrayList<Object>

Object

Object

Object

kennen nur Object-Methoden

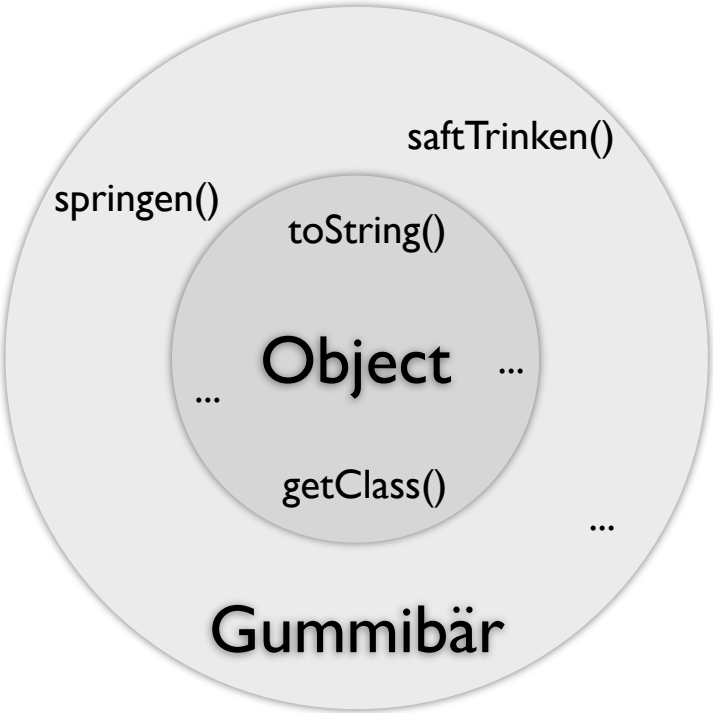
Das innere Object

class Object
toString () {...}
getClass () {...}
...

Die ultimative Superklasse !!!

IS-A

class Gummibär
springen () {...}
saftTrinken () {...}
...



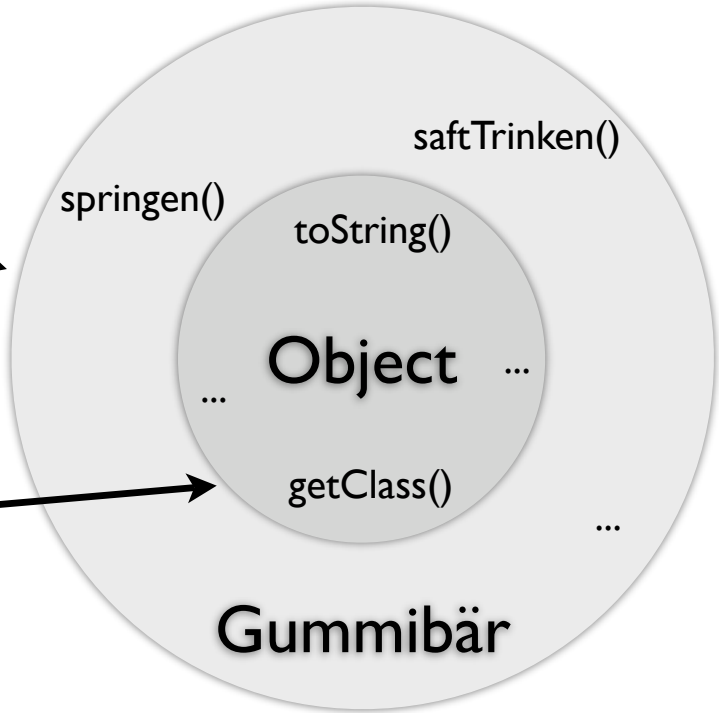
Gummibär cubbi = new Gummibär ();

ein einzelnes Objekt auf dem Heap

Das innere Object

// sieht **alle** Methoden:
Gummibär cubbi = new Gummibär ();

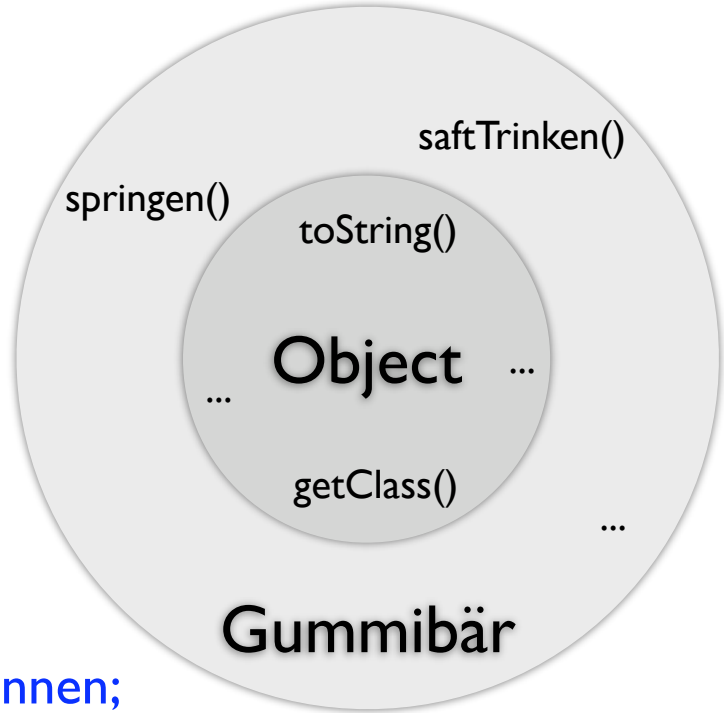
// sieht **nur Object-**Methoden:
Object innen = cubbi;



ein einzelnes Objekt
auf dem Heap

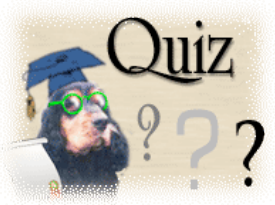
Object-Referenzen casten

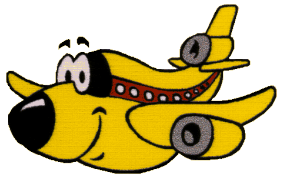
```
// sieht alle Methoden
Gummibär cubbi = new Gummibär ();
cubbi.springen ();
cubbi.toString ();

// sieht nur Object-Methoden
Object innen = cubbi;          // Polymorphie
String s = innen.toString ();
innen.springen ();             // nein !!!

// ist das Objekt ein Gummibär ?
if (innen instanceof Gummibär) {
   Gummibär auchCubbi = (Gummibär) innen;
   auchCubbi.springen ();
}
```



ein einzelnes Objekt
auf dem Heap

Einen Klassenbaum modifizieren





class Flugzeug
private String typ;
private float abrißGeschwindigkeit;
//...

class VerkehrsFlugzeug
private int anzahlPassagiere;

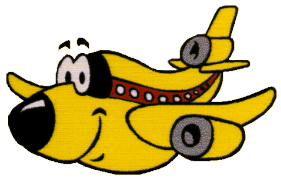
class KampfFlugzeug
public void
schleuderSitzBetätigen () {...}

class ModellFlugzeug

Flugzeuge müssen auch betankt werden!

Welche Klasse erhält die Methode tanken() ?

Einen Klassenbaum modifizieren



class Flugzeug
public void tanken () {...} // ???

class VerkehrsFlugzeug
public void tanken () {...} // ???

class KampfFlugzeug
public void tanken () {...} // ???

class ModellFlugzeug

Lösung 1:
Wir schreiben tanken() in die Klasse Flugzeug.
Aber nicht jedes (Modell-)Flugzeug kann tanken!

Lösung 2:
Wir schreiben tanken() in die Klassen VerkehrsFlugzeug und KampfFlugzeug.
Aber Polymorphie funktioniert dann nicht mehr!

Lösung 1:

Nicht alle Flugzeuge können tanken!

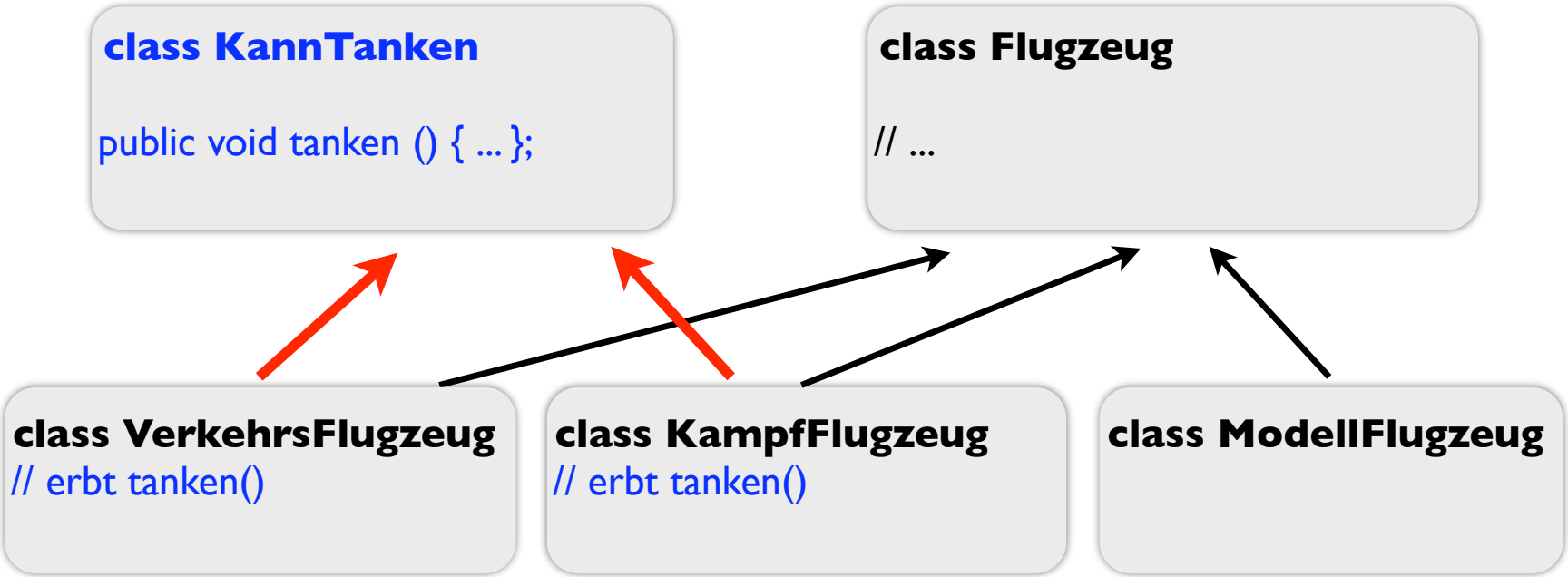
Lösung 2:

Methoden überladen.

Keine Polymorphie!

```
class Werkstatt {
   void flugzeugTanken (Flugzeug einFlieger) {
      einFlieger.tanken ();    // ModellFlugzeuge ???
   }

   void flugzeugTanken (VerkehrsFlugzeug einFlieger) {
      einFlieger.tanken ();    // nur VerkehrsFlugzeuge !!!
   }

   void flugzeugTanken (KampfFlugzeug einFlieger) {
      einFlieger.tanken ();    // nur KampfFlugzeuge !!!
   }
}
```

Lösung 3: Zwei Superklassen ?



Mehrfachvererbung ist in Java nicht erlaubt!
Klassen können nur eine Superklasse haben!

Interfaces: abstrakte Klassen

```
public interface KannTanken {

   // alle Interface-Methoden sind öffentlich und abstrakt
   public abstract void tanken ();

}
```

```
public class VerkehrsFlugzeug extends Flugzeug implements KannTanken {
   // ...
}
```

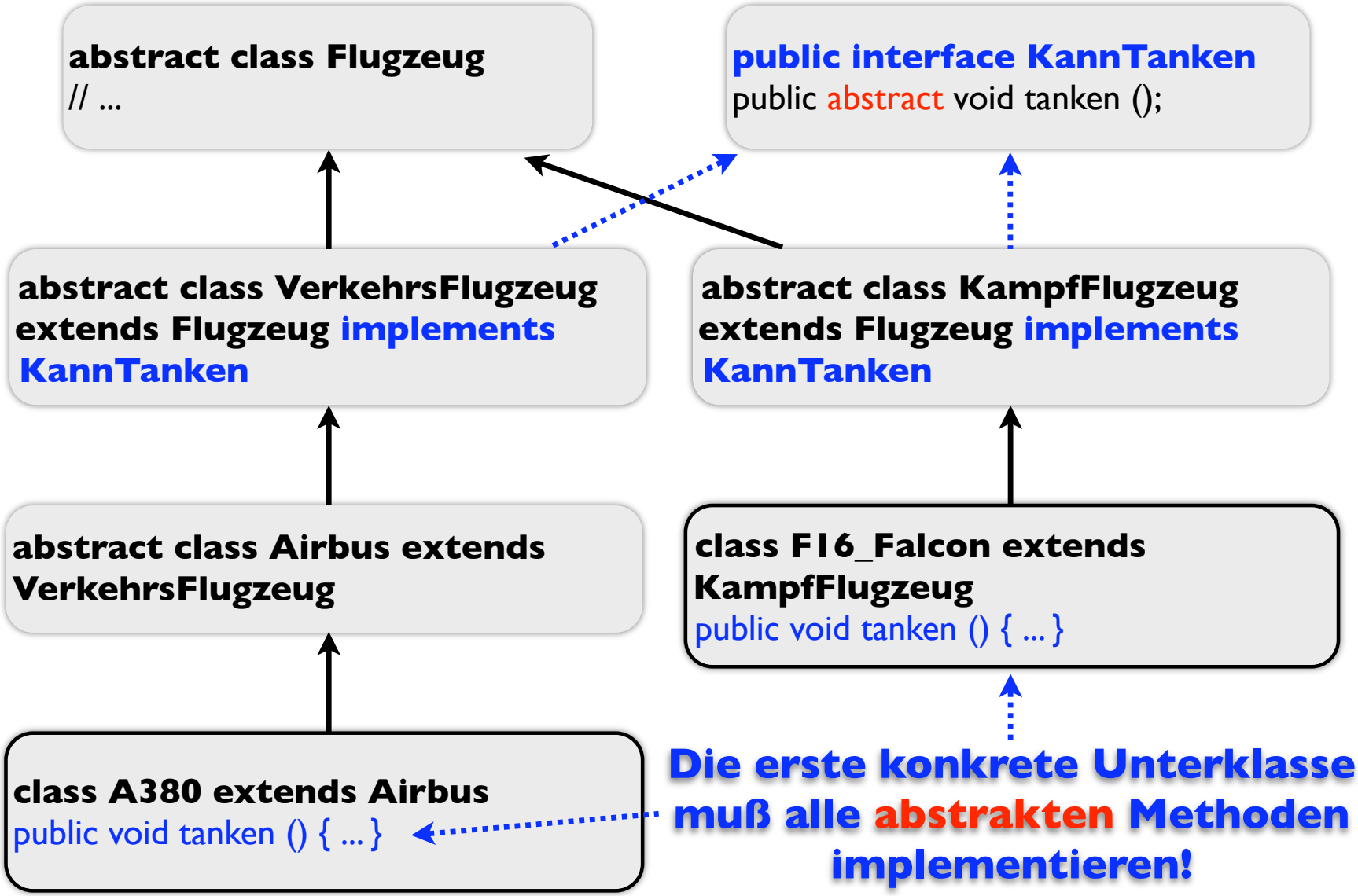
```
public class KampfFlugzeug extends Flugzeug implements KannTanken {
   // ...
}
```

Interfaces: abstrakte Klassen

```
public interface KannÜberschallFliegen {

   public abstract void mitÜberschallFliegen ();    // ohne Rumpf

}
```

```
public class KampfFlugzeug extends Flugzeug implements KannTanken, KannÜberschallFliegen {
   //...
}
```

Eine Klasse kann mehrere Interfaces implementieren. Sie hat aber nur eine Superklasse.


abstract class Flugzeug
// ...
public interface KannTanken
public abstract void tanken ();
abstract class VerkehrsFlugzeug extends Flugzeug implements KannTanken
abstract class KampfFlugzeug extends Flugzeug implements KannTanken
abstract class Airbus extends VerkehrsFlugzeug
class F16_Falcon extends KampfFlugzeug
public void tanken () { ... }
class A380 extends Airbus
public void tanken () { ... }
Die erste konkrete Unterklasse muß alle abstrakten Methoden implementieren!

Interfaces und Polymorphie

```
class Werkstatt {
    // Motor warten und Flugzeug tanken
    public void flugzeugTanken (KannTanken einFlieger) {
        einFlieger.tanken ();
    }
}
```

```
ArrayList<KannTanken> tankbareFlugzeuge = new ArrayList<KannTanken> ();
tankbareFlugzeuge.add (new A380 ());
tankbareFlugzeuge.add (new F16_Falcon ());

WerkStatt pitStop = new Werkstatt ();
for (KannTanken einFlugObjekt :  tankbareFlugzeuge ) {
    pitStop. flugzeugTanken (einFlugObjekt);
}
```

Jetzt sind Sie wieder an der Reihe!

Lesen Sie zu Interfaces und Polymorphie
Kapitel 8.