OOP – Abstraktion

Abstraktion: Übersicht

- ✓ Abstrakte Klassen
- ✓ Schnittstellen(Interface)
- ✓ Vergleich

Abstrakte Klassen Beispiel Ia

Eine Klasse, die abstrakte Methoden enthält, muss selbst als abstrakt deklariert werden

Methoden ohne Implementierung (Methodenrumpf) werden als abstrakt bezeichnet

Abstrakte Methoden werden nur deklariert aber nicht implementiert

```
abstract class Animal
{
  abstract void soundOfAnimal();
}
```

Aus einer Klasse Tier kann man kein „Objekt“ erzeugen,
wohl aber andere konkrete Tiere ableiten, von denen Objekte erzeugt werden können!
Abstrakte Klassen nennt man auch unvollständige Klassen, da diese nicht vollständig implementiert sind!

Abstrakte Klassen Beispiel Ib

Cat wird von Animal implementiert

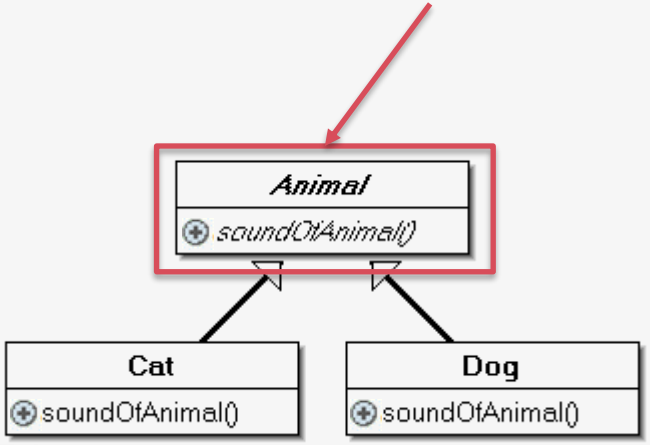
Implementierung

```
class Cat extends Animal {
  void soundOfAnimal(){
    System.out.println("Miau");
  }
}

class Dog extends Animal{
  void soundOfAnimal(){
    System.out.println("Wau Wau");
  }
}
```

Abstrakte Klassen Beispiel Ib

Abstrakte Klassen werden im Klassendiagramm
kursiv geschrieben



Spezialisierung

Generalisierung

Abstrakte Klassen Beispiel Ic

```
public class AnimalAbstractClassExample01{
   public static void main(String[] args){
      Cat myCat = new Cat();
      myCat.soundOfAnimal();
      Animal myAnimal = new Animal();
   }
}
```

Erzeugung eines Objektes vom Typ Cat

error: Animal is abstract; cannot be instantiated

Abstrakte Klassen Beispiel - Anzahl der abstr./konkreten Methoden

```
abstract class Animal
{
  abstract void soundOfAnimal();
  int noOfHeads(){
    return 1;
    }
}
```

Abstrakte Klassen können auch bereits implementierte Methoden enthalten!
Diese sind natürlich public ohne dass eine Angabe eines Zugriffsmodifizierers notwendig ist!
Eine Klasse kann 0..m abstrakte Methoden und 0..n implementierte Methoden enthalten!

Abstrakte Klassen Beispiel - Implementierung

```
abstract class Animal
{
  abstract void soundOfAnimal();
  abstract int noOfLegs();
}
```

```
class Cat extends Animal {
  void soundOfAnimal(){
    System.out.println("Miau");
  }
  int noOfLegs(){
    return 4;
  }
}
```

Alle Methoden müssen implementiert werden, sonst muss die abgeleitete Klasse wieder als abstrakt deklariert werden!

Abstrakte Klassen Beispiel - Fehlerquellen

```
abstract class Animal
{
  abstract int noOfLegs;
  abstract Animal(){};
  private void soundOfAnimal();
  abstract static void soundOfAnimal();
}
```

Attribute und Konstruktoren können nicht abstrakt sein!

Noch zu implementierende Methoden (abstrakte) dürfen nicht privat sein!

Abstrakte Methoden dürfen nicht statisch sein!

Abstrakte Klassen Regeln und Erläuterungen I

- ✓ Abstrakte Klassen sind NICHT instanziierbar, der Compiler verbietet dann die Erzeugung von Objekten dieser Klasse, d.h. new abstractClass (...) gibt einen Fehler aus
- ✓ Die abgeleiteten Klassen müssen entweder alle abstrakte Methoden implementieren oder selbst abstrakt sein!
- ✓ Abstrakte Klassen sind nützlich als gemeinsame Schnittstelle für ihre Unterklassen(abgeleitete Klassen)
- ✓ Wenn man z.B. eine abstrakte Klasse für geometrische Objekte in der Ebene definiert (Kreise, Rechtecke, etc.), so wurde es Sinn machen, eine Methode zur Berechnung des Flächeninhaltes vorzusehen.
- ✓ Man kann die Flächenberechnung aber für allgemeine geometrische Objekte nicht sinnvoll implementieren(das geht nur für die Subklassen, d.h. konkrete Formen).

Abstrakte Klassen Beispiel II

```
abstract class Animal
{
 private void soundOfAnimal();
}
```

Fehler: Nicht implementierte Klassen benötigen das Schlüsselwort abstract!

Fehler: Abstrakte Klassen brauchen keine Zugriffsmodifizier: Sie sind ohne Angabe bereits public, da Sie implementiert werden müssen!

Abstrakte Klassen Regeln und Erläuterungen II

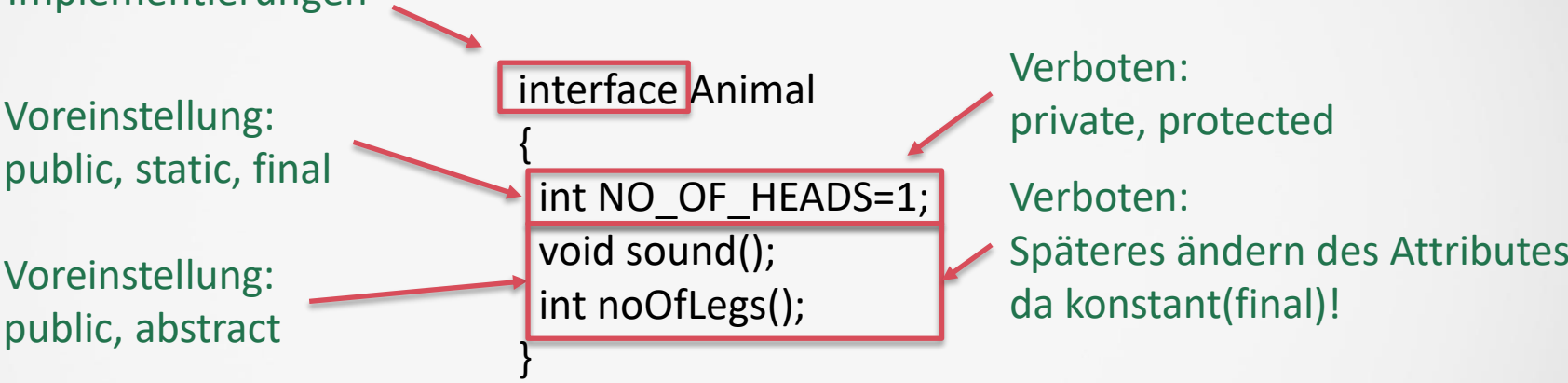
- ✓ Eine abstrakte Klasse muss nicht unbedingt nur abstrakte Methoden haben: Manche Methoden lassen sich vielleicht schon auf der allgemeinen Ebene implementieren.
- ✓ Falls eine abstrakte Klasse nur abstrakte Methoden und keine Attribute hat, sollte man eventuell ein “Interface” zu verwenden
- ✓ Damit man mit der abstrakten Klasse etwas anfangen kann, muss es eine Unterklasse geben, in der die abstrakten Methoden überschrieben (und damit tatsächlich implementiert) sind.
- ✓ Wenn eine Subklasse nicht alle ererbten abstrakten Methoden überschreibt, muss sie selbst „abstract” sein.

Interfaces I

- ✓ Java verwendet “Single Inheritance”, d.h. jede Klasse kann maximal eine Oberklasse haben(C++ läßt “Multiple Inheritance” zu).
- ✓ Ein Interface ist eine abstrakte Klasse mit nur abstrakten Methoden.
- ✓ Es ist angegeben, welche Methoden mit welchen Argument- und Resultat-Typen es geben muss, aber es ist keine Implementierung für diese Methoden angegeben, d.h. kein Methoden-Rumpf
- ✓ Eine Klasse kann
  - ✓ nur eine Oberklasse haben, aber
  - ✓ beliebig viele Interfaces implementieren

Schnittstellen Beispiel I

Ein Interface ist eine Gruppe von zusammenhängenden Methoden ohne Implementierungen



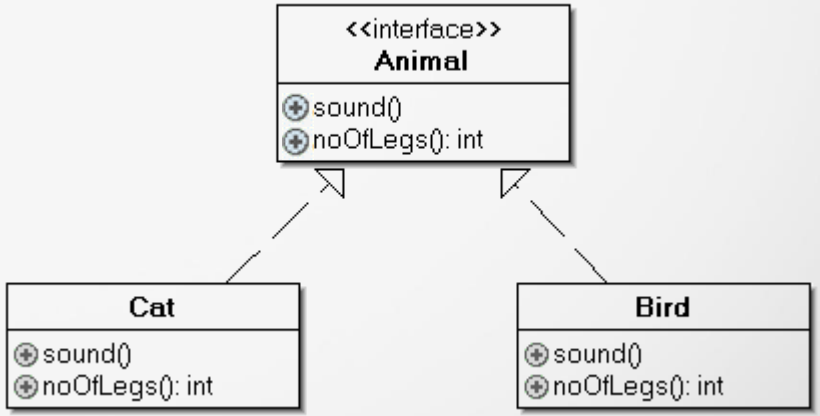
Schnittstellen Beispiel I

Mit dem Schlüsselwort implements
wird die Klasse implementiert

```
class Cat implements Animal {
  public void sound(){
    System.out.println("Miau");
  }
  public int noOfLegs(){
    return 4;
    }
}
```

ALLE Methoden müssen
implementiert werden!

nötig, da sonst
Voreinstellung:private



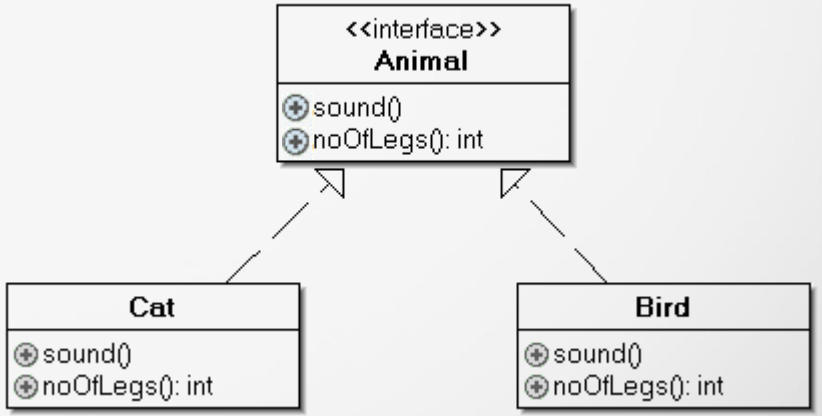
Schnittstellen Beispiel I

Mit dem Schlüsselwort implements
wird die Klasse implementiert

```
class Cat implements Animal {
  public void sound(){
    System.out.println("Miau");
  }
  public int noOfLegs(){
    return 4;
    };
}
```

ALLE Methoden müssen
implementiert werden!

nötig, da sonst
Voreinstellung:private



Interfaces III

- ✓ Ein Interface kann folgendes enthalten:
  - Abstrakte Methoden
  - Konstanten
  - Geschachtelte Klassen und Interfaces
- ✓ Ein Interface kann nicht enthalten:
  - Statische Methoden(können nicht abstrakt sein)
  - Attribute(Wäre schon Implementierung!)
  - Konstruktoren(Man kann keine Objekte erzeugen!)
- ✓ Alle Bestandteile des Interfaces sind implizit public
- ✓ Man läßt „abstract“ bei Methoden weg(Versteht sich von selbst!)
- ✓ Man läßt „final“ bei Konstanten weg(Sieht wie initialisierte Attribute aus!)
- ✓ Die implementierten Methoden müssen „public“ sein!

Abstrakte Klassen versus Interfaces

- ✓ Beide deklarieren Methoden, die die implementierenden bzw. erbenden Klassen realisieren müssen.
- ✓ Von beiden kann kein Objekt erzeugt werden.
- ✓ Eine Klasse kann beliebig viele Schnittstellen besitzen, aber nur eine Oberklasse.
- ✓ Abstrakte Klassen können eigenen Programmcode besitzen, was bei Interfaces (Schnittstellen) nicht möglich ist.

Mit `public class CClass implements interface1, interface2;` kann man Klassen aus zwei Interfaces implementieren