Polymorphie

openHPI-Java-Team

Hasso-Plattner-Institut

Eine Papageienliste

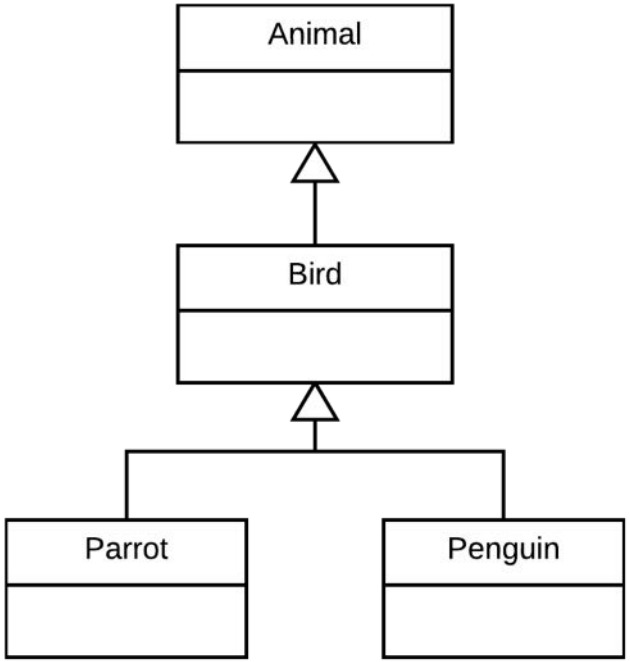
```
[...]
1   Parrot paco  = new Parrot();
2   Parrot polly = new Parrot();
3   Parrot penny = new Parrot();
4
5   Parrot[] array = new Parrot[3];
6   array[0] = paco;
7   array[1] = polly;
8   array[2] = penny;
9
10  for (int i = 0; i < array.length; i++) {
11      array[i].move();
12  }
[...]
```

Ausgabe:

```
Ich fliege
Ich fliege
Ich fliege
```

Auch Papageien sind Vögel

```
  [...]
1   Bird penguin = new Penguin();
2   Bird parrot = new Parrot();
  [...]
```



- Einer Variable vom Typ `Bird` können Objekte aller Unterklassen von `Bird` zugewiesen werden
- Ein Objekt der Klasse Bird kann auch nur Methoden der Klasse Bird aufrufen

Eine Vogelliste

Hasso Plattner Institut

```
[...]
    Bird penguin = new Penguin();
    Bird parrot = new Parrot();

    Bird[] array = new Bird[2];
    array[0] = parrot;
    array[1] = penguin;

    for (int i = 0; i < array.length; i++) {
        array[i].move();
    }
[...]
```

Wir können zwar keine Objekte vom Typ `Bird` instanziieren (da dies eine abstrakte Klasse ist), aber wir können ein Array erzeugen, dass Objekte vom Typ `Bird` enthält!

Eine Vogelliste

```
[...]
1     Bird penguin = new Penguin();
2     Bird parrot = new Parrot();
3
4     Bird[] array = new Bird[2];
5     array[0] = new Parrot();
6     array[1] = new Penguin();
7
8     for (int i = 0; i < array.length; i++) {
9         array[i].move();
10    }
[...]
```

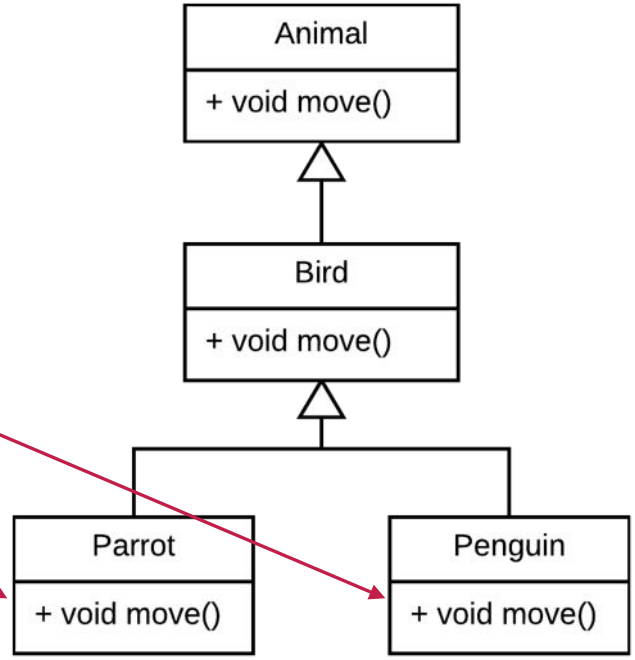
Eine Vogelliste

```
[...]
1    Bird penguin = new Penguin();
2    Bird parrot = new Parrot();
3
4    Bird[] array = new Bird[2];
5    array[0] = new Parrot();
6    array[1] = new Penguin();
7
8    for (int i = 0; i < array.length; i++) {
9        array[i].move();
10   }
[...]
```

Ausgabe:

```
Ich fliege
Ich schwimme
```

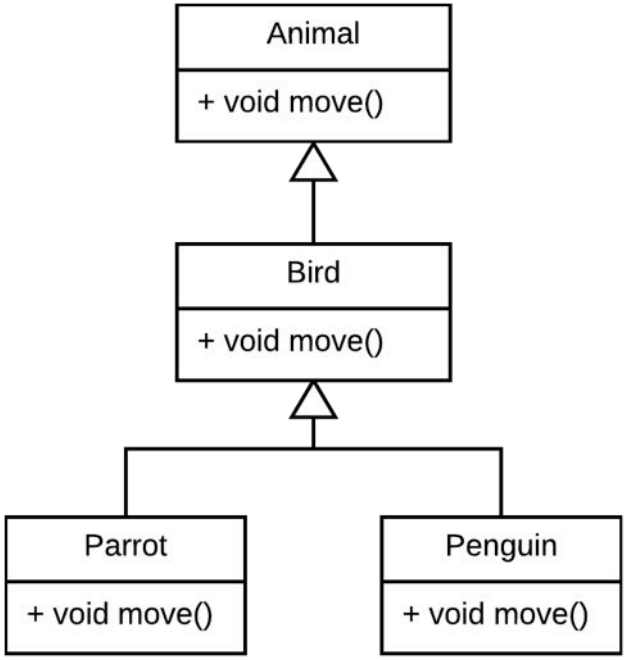

```
[...]
1    Bird parrot = new Parrot();
2    Bird penguin = new Penguin();
3
4    penguin.move();
5    parrot.move();
[...]
```



Vererbung

- Definiert ein “Protokoll” für eine Klassenfamilie:
  - Alle Subklassen besitzen alle geerbten Methoden und Attribute der Superklasse


```
[...]
1    Bird parrot = new Parrot();
2    Bird penguin = new Penguin();
3
4    penguin.move();
5    parrot.move();
[...]
```

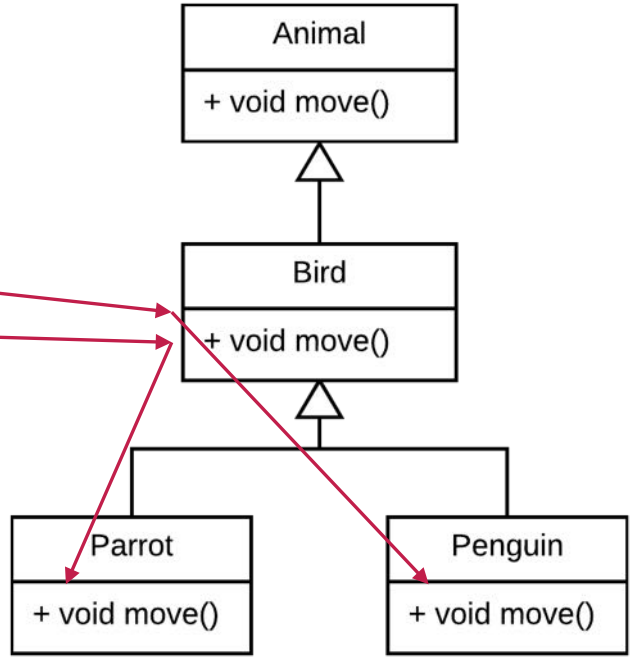


Polymorphie

- Einer Variablen können wir sowohl Objekte ihres Typs als auch aller ihrer Subtypen zuweisen
- Diese erfüllen per Definition unser Protokoll, daher können wir **immer** eine beliebige Subklasse anstelle der Superklasse einsetzen

Polymorphie

```
[...]
1    Bird parrot = new Parrot();
2    Bird penguin = new Penguin();
3
4    penguin.move();
5    parrot.move();
[...]
```

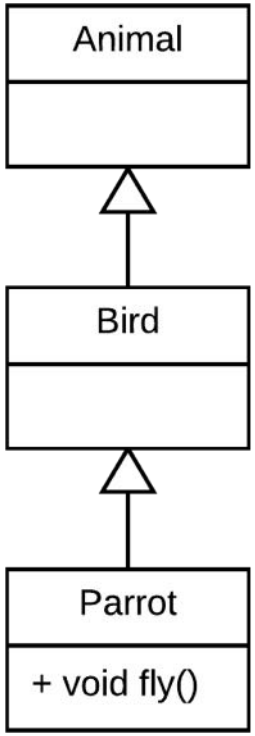


Ausgabe:

```
Ich schwimme
Ich fliege
```

Ein Vogel, der nicht mehr fliegen kann?

```
[...]
1   Animal parrot = new Parrot();
2   parrot.fly();
[...]
```

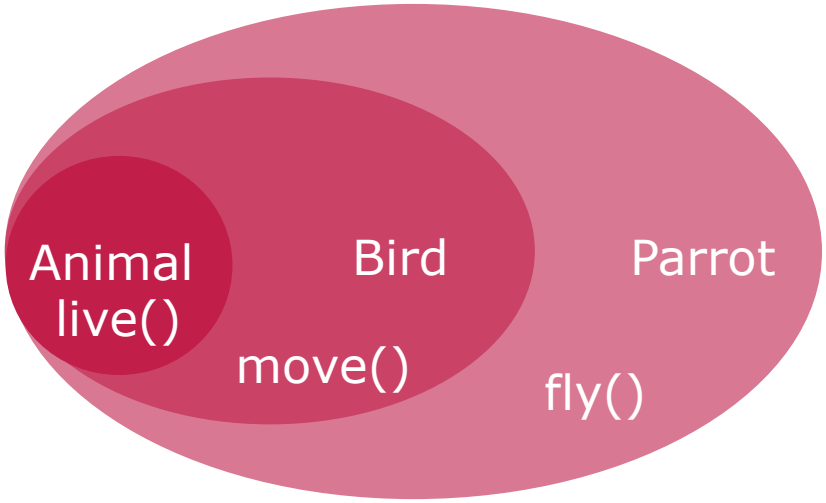


- Leider ein Fehler
- Java (der Compiler) entscheidet auf Grund des Datentyps ob man eine Methode aufrufen kann
  - Die Klasse `Animal` hat keine Methode `fly()`
  - Nur die Klasse Parrot implementiert die Methode


```
1 public class Robot {
2
3     // ...
4     public void investigateAnimal(Animal animal) {
5         // ...
6     }
7 }
```

- Die Methode `investigateAnimal()` erwartet ein Objekt der Klasse `Animal` als Parameter
- `investigateAnimal()` akzeptiert daher alle Subklassen von `Animal` als Argument
  - Neue Subklassen erfordern keine Anpassung des Codes innerhalb der Methode ➔ bessere Wartbarkeit

Polymorphie



- `Animal paco = new Parrot(); paco.live(); ~~paco.move();~~ ~~paco.fly();`
- `Bird paco = new Parrot(); paco.live(); paco.move(); ~~paco.fly();~~`
- `Parrot paco = new Parrot(); paco.live(); paco.move(); paco.fly();`

- Polymorphie = “Vielgestaltigkeit”
- Man kann ein Objekt als Instanz mehrerer Klassen betrachten