# Überblick

4. Objektorientiertes Design
   - Interfaces, Container, Iterator, Visitor
   - Composition, Decorator
   - Schichtenmodell und Filter-Pipelines

# “Composition over Inheritance”

Eine Möglichkeit, daß eine Klasse *B* die Funktionalität von *A* übernimmt, ist *Vererbung*:

```
class B extends A {
  ...
}
```

Eine andere Möglichkeit ist *Komposition*.
Klasse *B* *enthält* ein Object *A*:

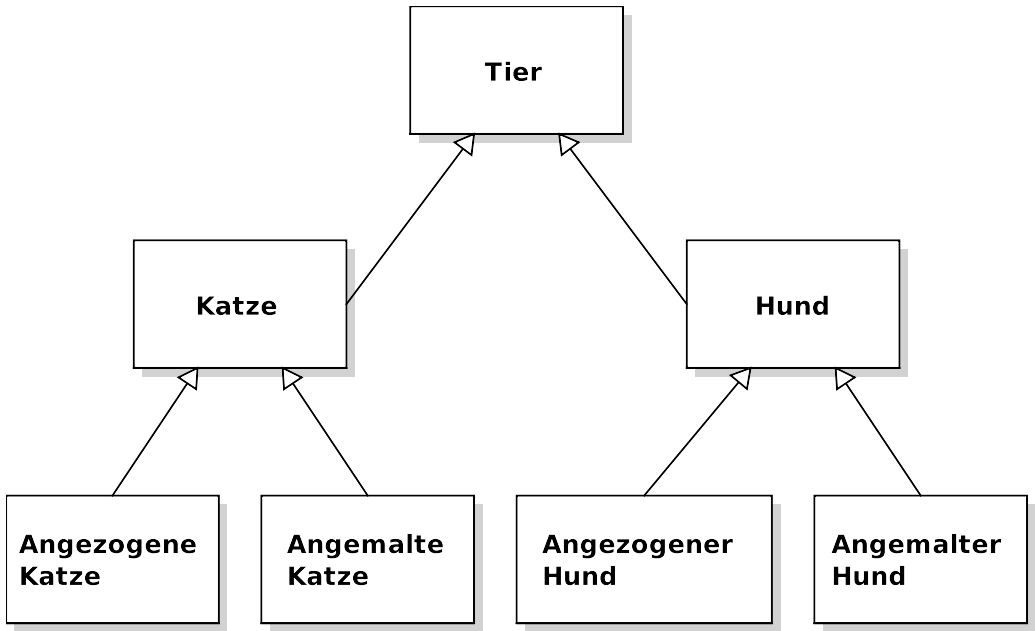
```
class B {
  A a;
  ...
}
```

Dadurch kann *B* im Prinzip auch alles, was *A* kann.

Komposition kann flexibler sein und
ist oft besser als Vererbung.

# Vererbung

Betrachten wir folgendes Klassendiagramm:




Wenn wir mehr Tiere hinzufügen, gibt es eine Unzahl von spezialisierten Klassen.

Wir können keinen „angezogenen, angemalten Hund“ durch Vererbung erzeugen (keine Mehrfachvererbung).

```
interface Tier {
  public double getGewicht();
  public String zeichnung();
}
```

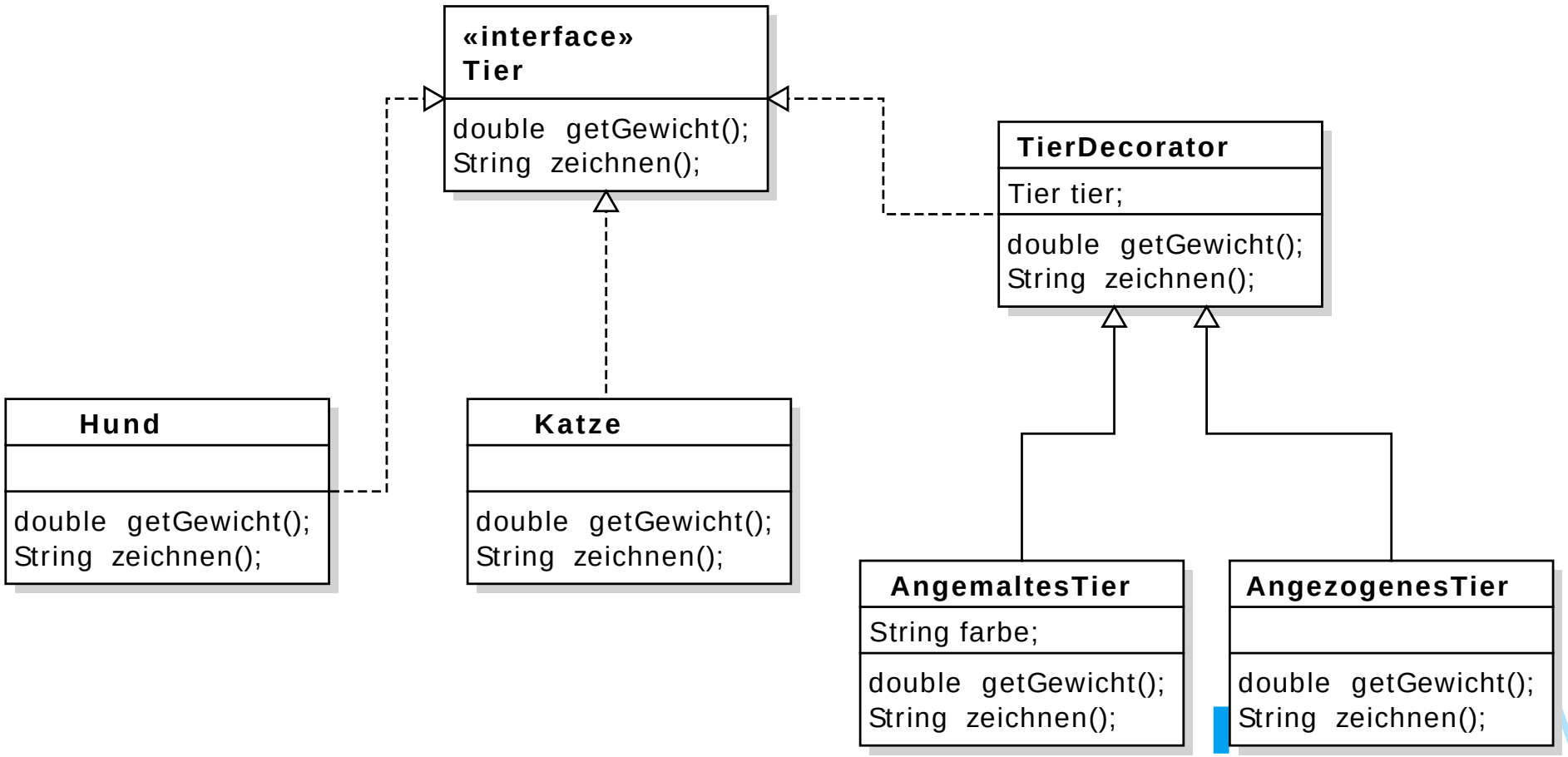
Einige konkrete Tiere:

```
class Katze implements Tier {
 public double getGewicht() {return 2.5;}
 public String zeichnung() {return "Katze";}
}
```

```
class Hund implements Tier {
 public double getGewicht() {return 3.5;}
 public String zeichnung() {return "Hund";}
}
```

# Composition: Decorator

Eine mögliche Lösung: Wir verwenden einen *Decorator*, der dynamisch Funktionalität zu einer Klasse hinzufügen kann.



```
interface Tier {
  public double getGewicht();
  public String zeichnung();
}
```

Die abstrakte Klasse *TierDecorator* ist auch ein *Tier*:

```
abstract class TierDecorator implements Tier {
 private Tier tier;
 public TierDecorator(Tier tier) {
   this.tier = tier;
 }
 public double getGewicht() {
  return tier.getGewicht();
 }
 public String zeichnung() {
  return tier.zeichnung();
 }
}
```

Wir können jetzt einige *TierDecorator*s implementieren:

```
class AngezogenesTier extends TierDecorator {
 public AngezogenesTier(Tier tier) {
  super(tier);
 }
 public double getGewicht() {
  return super.getGewicht() + 0.5;
 }
 public String zeichnung() {
  return super.zeichnung() + " mit Mantel";
 }
}
```

Ein angezogenes Tier ist ein Tier mit einem Mantel.
Der Mantel wiegt ein bißchen.

Ein „angemaltes Tier“ hat eine Farbe, die wir dem Konstruktor mitteilen müssen:

```
class AngemaltesTier extends TierDecorator {
  private String farbe;
  public AngemaltesTier(Tier tier, String farbe) {
    super(tier);
    this.farbe = farbe;
  }
  public String zeichnung() {
    return super.zeichnung() + "  " + farbe + " angemalt";
  }
}
```

Wir können jetzt beliebige Dekorationen an einem Tier „anbringen“:

```
public static void main(String args[ ]) {
   Tier hachiko = new Hund();
   hachiko = new AngezogenesTier(hachiko);
   hachiko = new AngemaltesTier(hachiko, "weiss");
   Tier garfield = new Katze();
   garfield = new AngemaltesTier(garfield, "gelb-schwarz");
   System.out.println(hachiko.zeichnung());
   System.out.println(garfield.zeichnung());
}
```

Ausgabe:

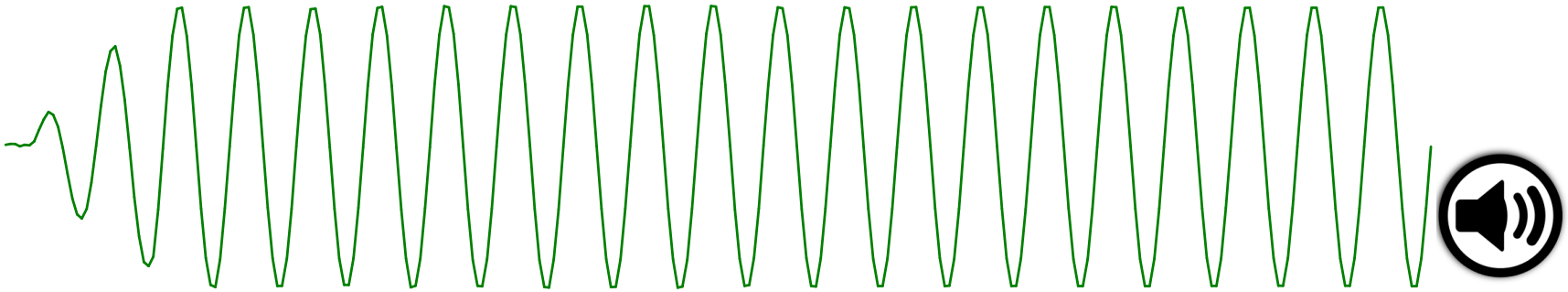
```
Hund mit Mantel blau angemalt
Katze gelb-schwarz angemalt
```

# Überblick

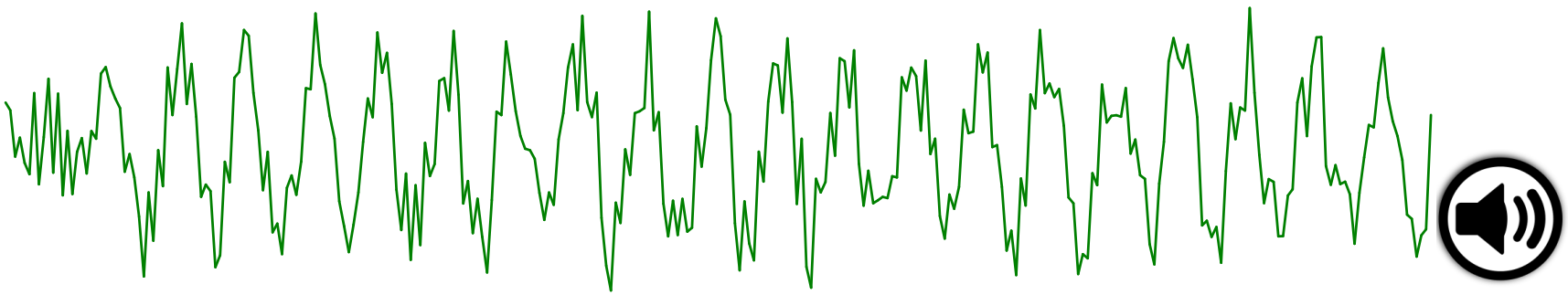
4. Objektorientiertes Design
   - Interfaces, Container, Iterator, Visitor
   - Composition, Decorator
   - Schichtenmodell und Filter-Pipelines

# Beispiel: Morsecode dekodieren

Morsecode, sauberer Ton:



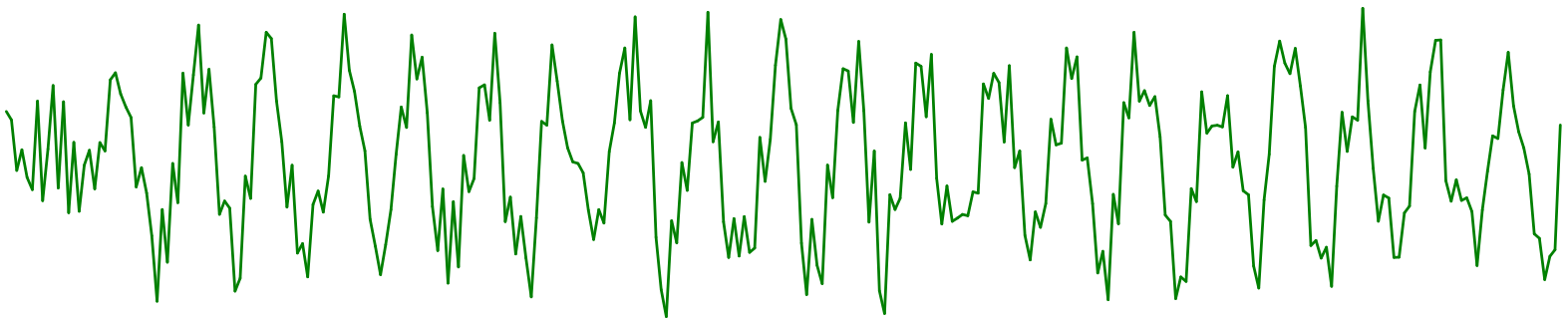
Etwas verrauscht:



Wie können wir den Klartext erhalten?

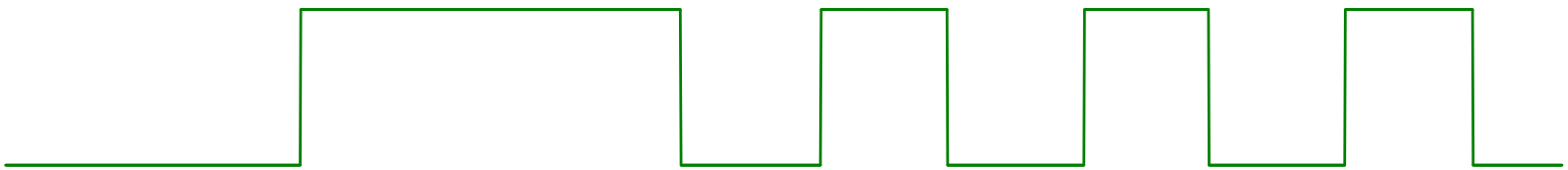
Die Audiodaten werden durch mehrere Filter geschickt.

- *buffer* = *filter*.*readFile*(`"withnoise.raw"`);

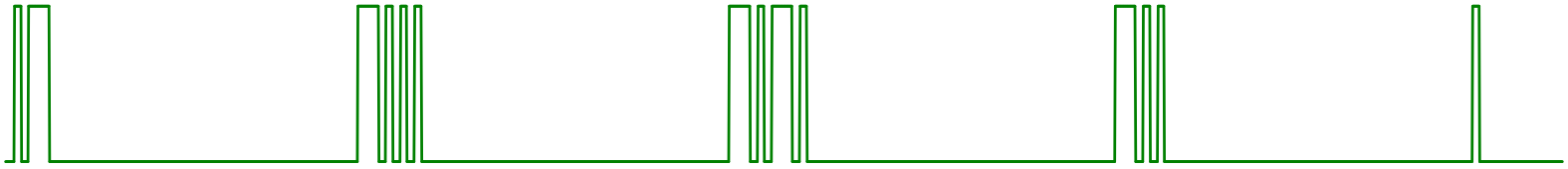


- *buffer* = *filter*.*average*(200, *buffer*)

- *buffer* = *filter*.*threshold*(*buffer*)



- *buffer* = *filter*.*threshold*(*buffer*);



- *buffer* = *filter*.*cummulate*(*buffer*);
  [-1234, 968, -1085, 2951, -44005, 2954, -1076, 977, -1051, 976, -1052, 977, -43977, 2954, -1076, 977, -1077, 2955, -1075, 976, -43978, 2954, -1076, 976, -1053, 976, -43951, 976, -43952, 977, -1051,...]
- *String dotdashs* = *filter*.*symbolize*(*buffer*);
  .- -... -.-. -.. . ..-. --. .... .. .--- -.- .-.. --
  -. --- .--. --.- .-. ... - ..- ...- .-- -..- -.-- --..
- *List*⟨*String*⟩ *tokens* = *filter*.*tokenize*(*dotdashs*);
  [.-, -..., -.-., -.., ., ..-., --., ...., .., .---, -.-, .-.., --, -., ---, .--., --.-, .-., ..., -, ..-, ...-, .--, -..-, -.--, --..]
- *String cleartext* = *filter*.*decode*(*tokens*);
  ABCDEFGHIJKLMNOPQRSTUVWXYZ