

# PROCESSING

BILDER UND SOUND

Created by Michael Kirsch & Beat Rossmly

# INHALT

## 1. Rückblick

1. Processing Basics
2. Vererbung
3. Interfaces
4. Vererbung & Interfaces

## 2. Theorie

1. Bilder
2. Was ist ein Pixel?
3. PImage
4. Modulo
5. Bibliotheken

## 3. Anwendung

1. Image

## 4. Verknüpfung

1. Bibliotheken
2. SoundFile
3. Data

## 5. Ausblick

1. Nächste Sitzung
2. Übung

RÜCKBLICK

# PROCESSING BASICS

## Transformations

Verschieben

```
int x = 300;  
int y = 200;  
translate(x, y);  
rect(200,200,200,200);
```

---

Rotieren

```
rotate(PI/2);  
rect(200,200,200,200);
```

---

Scale

```
scale(2.0);  
rect(200,200,200,200);
```

# PROCESSING BASICS

## Transformations

verhindere Auswirkung der  
Transformation auf  
nachfolgende Objekte

```
pushMatrix();  
translate(200,200);  
rect(0,0,200,200);  
popMatrix();
```

```
pushMatrix();  
translate(100,300);  
rect(0,0,200,200);  
popMatrix();
```

# PROCESSING BASICS

## Tabs

Tabs helfen deinen Sketch zu organisieren. Es bietet sich an für Klassen neue Tabs anzulegen.

Klicke auf den ▼ neben dem Sketch Namen und wähle "Neuer Tab". Benenne diesen so, dass nachvollziehbar ist, was der Tab enthält.

```
void setup () {...}

void draw () {...}

// -----
// Klasse -> neuer Tab

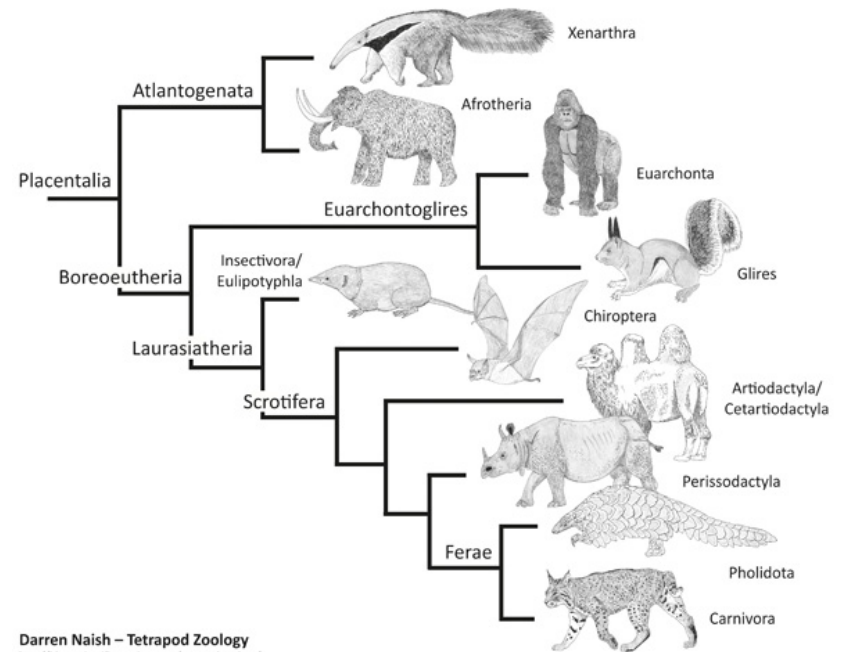
class Ball {...}

// -----
// Klasse -> neuer Tab

class Wall {...}
```

# VERERBUNG

- Vererbung beschreibt das Konzept, dass sich viele Klassen von einer Elternklasse ableiten lassen, bzw. diese erweitern.
- Dabei kann eine Klasse viele Kinder haben, aber nur eine Elternklasse!



Darren Naish – Tetrapod Zoology  
<http://blogs.scientificamerican.com/tetrapod-zoology/>

[http://blogs.scientificamerican.com/blogs/assets/tetrapod-zoology/File/placentals-molecular-phylogeny-600-px-tiny-july-2015-Darren-Naish-Tetrapod-Zoology\(1\).jpg](http://blogs.scientificamerican.com/blogs/assets/tetrapod-zoology/File/placentals-molecular-phylogeny-600-px-tiny-july-2015-Darren-Naish-Tetrapod-Zoology(1).jpg)

# VERERBUNG

- Vererbung bedeutet im Detail, dass alle Felder und Methoden der Elternklasse Teil der Kindklasse sind.
- Weitere Felder und Methoden können ergänzt werden.
- Der Inhalt von Methoden kann überschrieben (→ verändert) werden.

```
class Parent {
    int a,b,c;
    public Parent () {
        a = 0;
        b = 1;
        c = 2;
    }
    void out () {}
}
```

```
class Child extends Parent {
    // erbt: Felder a,b,c
    int d,e,f;
    public Child () {
        d = 3;
        e = 4;
        f = 5;
    }
    // überschreibt: out
    void out () {
        println(a+" "+b+" "+c);
    }
}
```



# INTERFACES

- Mit Interfaces lassen sich Schnittstellen zwischen Klassen definieren.
- Man bestimmt, dass es eine bestimmte Funktionalität (in Form einer Methode) in einer Klasse gibt, die genaue Umsetzung wird aber nicht vorgegeben.



<http://nos.twinsnd.co/image/127563852947>

# INTERFACES

- In der Beschreibung des Interfaces werden alle zu implementierenden Methoden aufgelistet, mit Rückgabebetyp, Name und Übergabeparameter.
- Die Funktionalität wird erst in der implementierenden Klasse umgesetzt.

```
interface Output {  
    void out ();  
}
```

```
class TextOut implements Output {  
    String text;  
  
    public Child () {  
        text = "Hello?";  
    }  
  
    void out () {  
        println(text);  
    }  
}
```

```
class ShapeOut implements Output {  
    public ShapeOut () {}  
  
    void out () {  
        rect(100,100,50,50);  
    }  
}
```

# VERERBUNG & INTERFACES

- Interfaces und Elternklassen können als Typen bei Arrays verwendet werden.
- So können z.B. Klassen der unterschiedlichsten Art, aber mit sich deckender Funktionalität in einem Array gespeichert werden.

```
interface Output {...}
```

```
class Haus implements Output {...}
```

```
class Auto implements Output {...}
```

```
class Baum implements Output {...}
```

```
Output[] o;
```

```
void setup () {  
    size(600,400);
```

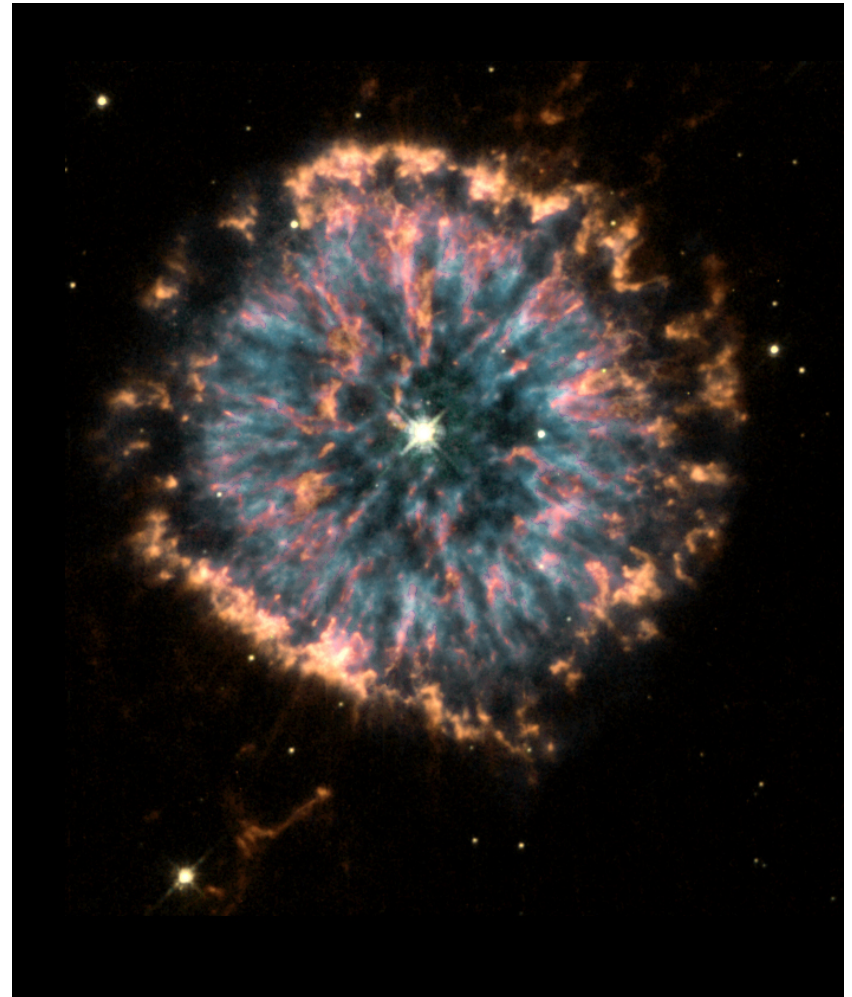
```
    o = new Output[]{  
        new Haus(...),  
        new Auto(...),  
        new Baum(...)  
    };
```

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

THEORIE

# BILDER

- Ein wichtiger Bestandteil von grafischen Programmen ist die Darstellung und Manipulation von Bildern.
- Kommerzielle Lösungen bieten eine breite Palette an Bearbeitungstools und vorgefertigten Filtern/Effekten.



# BILDER

- Was sind Bilder?
- Ein digitales Bild ist eine Ansammlung von Pixeln. Also ein Raster von winzigen Farbpunkten.
- Mathematisch können wir die Positionen in einem Raster mit Koordinaten angeben, oder wir nummerieren alle Punkte einzeln durch.

How the pixels look:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

How the pixels are stored:

0	1	2	3	4	5	6	7	8	9	.	.	.	
---	---	---	---	---	---	---	---	---	---	---	---	---	--

# BILDER

- In Processing hat jedes Pixel eines Bildes eine Nummer.
- Wir beginnen im linken oberen Eck mit der Ziffer 0 und enden im rechten unteren Eck mit der größten Zahl.
- Wir rechnen:  
 $\text{Breite} * \text{Höhe} = \text{Anzahl aller Pixel}$

How the pixels look:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

How the pixels are stored:

0	1	2	3	4	5	6	7	8	9	.	.	.		
---	---	---	---	---	---	---	---	---	---	---	---	---	--	--

# WAS IST EIN PIXEL?

- Ein Bild ist eine Menge an Pixeln.
- Ein Pixel entspricht einer Farbe an einer Position.
- In Processing ist dies als **color[ ]** gelöst.
- Die Länge des Arrays entspricht der Anzahl aller Pixel.

How the pixels look:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

How the pixels are stored:

0	1	2	3	4	5	6	7	8	9	.	.	.	
---	---	---	---	---	---	---	---	---	---	---	---	---	--



# PIMAGE

- Die Klasse **PImage** ermöglicht es uns Bilder in Processing zu laden und zu manipulieren.
- Um ein Bild zu initialisieren benutzen wir **loadImage**. Als Parameter wird der Dateipfad des Bildes übergeben.
- Tipp: speichere Bilder im Sketch-Ordner → kurzer Pfad

```
// Bild deklarieren
PImage img;

void setup () {
  size(600,400);
  // initialisiere Bild
  img = loadImage("test.jpg");
}
```

# PIMAGE

- Die Klasse **PImage** ermöglicht es uns Bilder in Processing zu laden und zu manipulieren.
- Um ein Bild zu initialisieren benutzen wir **loadImage**. Als Parameter wird der Dateipfad des Bildes übergeben.
- Tipp: speichere Bilder im Sketch-Ordner → kurzer Pfad

```
// Bild deklarieren
PImage img;

void setup () {
  size(600,400);
  // initialisiere Bild
  img = loadImage("test.jpg");
}

void draw () {
  background();
  image(img);
}
```

# PIMAGE

- Das Bild kann gezeichnet werden, indem der Befehl **image()** mit einem **PImage** Objekt als Parameter aufgerufen wird.
- Zugriff auf die Pixel erhält man, indem man auf das Array **pixels** auf dem Objekt zugreift.
- So können Werte gelesen und verändert werden.

```
// Bild deklarieren
PImage img;

void setup () {
  size(600,400);
  // initialisiere Bild
  img = loadImage("test.jpg");
}

void draw () {
  background();

  // speichere einen roten Pixel an
  Stelle 100
  img.pixels[100] = color(255,0,0);

  image(img);
}
```

# PIMAGE

- Änderungen am Pixel Array müssen mit einem Aufruf der Methode **updatePixels()** sichtbar gemacht werden.

```
// Bild deklarieren
PImage img;

void setup () {
  size(600,400);
  // initialisiere Bild
  img = loadImage("test.jpg");
}

void draw () {
  background();

  // speichere einen roten Pixel an
  Stelle 100
  img.pixels[100] = color(255,0,0);

  // mache Änderungen sichtbar
  img.updatePixels();

  image(img);
}
```

# MODULO

- Wie kann ich auf bestimmte Pixel zugreifen? Z.B. Alle Pixel am linken Bildrand?
- Oder jeden 2. Pixel, jeden 10. Pixel, ...
- Wie kann ich Pixelzeilen oder -spalten im Bild behandeln?

# MODULO

- Wir erinnern uns: Alle Pixel sind in einem Array abgelegt.
- Um eine Zeile zu behandeln, muss dieses Array quasi in regelmäßigen Abständen zerschnitten werden.
- Vorgehen: Teilt man den Index des Pixels durch die Breite (in Pixel) des Bildes, so erhält man die Zeile, in der sich der Pixel befindet.

```
PImage img;
int w,h;

void setup () {
  size(600,400);
  img = loadImage("test.jpg");

  w = img.width;
  h = img.height;
  // w*h == img.pixels.length
}

// In welcher Zeile liegt 100?
// Wenn das Bild 47px breit ist?

// int zeile = 100/47 -> Zeile 2!
// ohne Kommastellen wegen int
// erste Zeile hat Index 0!
```

# MODULO

- Wie können wir die Spalte errechnen?
- Mathematisch können wir dies wie folgend errechnen:  
Index - Zeile\*Breite
- Oder mit einem neuen Operator!

```
PImage img;
int w,h;

void setup () {
  size(600,400);
  img = loadImage("test.jpg");

  w = img.width;
  h = img.height;
}

// In welcher Zeile liegt 100?
// Wenn das Bild 47px breit ist?

// int zeile = 100/47 -> Zeile 2!
// ohne Kommastellen wegen int
// erste Zeile hat Index 0!
```

# MODULO

- Der Modulo-Operator liefert den ganzzahligen Rest einer Division!
- In Worten: Wenn ich eine Zahl durch eine andere teile, wie groß ist der Abstand zu der nächst kleineren Zahl, die (glatt) teilbar ist?

$$4\%2 = 0$$

```
// 5 ist nicht rund teilbar  
// die nächste teilbare Zahl ist 4  
// die Differenz beträgt 1
```

$$5\%2 = 1$$

$$6\%2 = 0$$

...

$$3\%3 = 0$$

$$4\%3 = 1$$

$$5\%3 = 2$$

$$6\%3 = 0$$

...



# BIBLIOTHEKEN

- Muss man das Rad immer neu erfinden?
- Die meisten Probleme hat jemand anderes doch sicher schon besser gelöst?

```
// Wir möchten Sound  
  
// auf die Kinect reagieren  
  
// Quellen aus dem Internet  
  
void setup () {}  
  
void draw () {}
```

# BIBLIOTHEKEN

- Bibliotheken erlauben es uns Funktionalitäten zu benutzen, die Processing standardmäßig nicht beherrscht.
- Bibliotheken sind nichts anderes als eine Sammlung von Klassen, die eine Bestimmte Funktionalität ermöglichen.

```
// Wir möchten Sound  
  
// auf die Kinect reagieren  
  
// Quellen aus dem Internet  
  
void setup () {}  
  
void draw () {}
```

# BIBLIOTHEKEN

- Im Menüpunkt **Sketch** unter dem Punkt **Library importieren ...** findet man den bereits integrierte Bibliotheken.
- Im selben Menü können externe Bibliotheken einfach eingebunden werden. Man wählt **Library hinzufügen...** und kann externe Bibliotheken herunterladen.

```
import processing.sound.*;

// auf die Kinect reagieren

// Quellen aus dem Internet

void setup () {}

void draw () {}
```

ANWENDUNG

# IMAGE

Vertausche Pixel!

```
// deklariere Bild

void setup () {
  // lade Bild
}

void draw () {
  // Wähle zufälligen Pixel
  // Tausche Farbe mit anderem zufälligem Pixel
}
```

VERKNÜPFUNG

# BIBLIOTHEKEN

- Um nun Sound einbinden zu können, müssen wir eine Bibliothek mit dem Namen Sound importieren.
- Dazu laden wir diese herunter, und importieren sie anschließend über das Menü Sketch.
- Nun können wir die Bibliothek verwenden.

```
import processing.sound.*;

Animation[] animations;

void setup () {
    ...
}

void draw () {
    ...
}
```

# SOUNDFILE

- Nun wird der Klasse Animation im Konstruktor ein **SoundFile** übergeben, das in einem Feld abgelegt und somit abrufbar ist.
- Dieser Sound wird nun in **triggerOn** gestartet.

```
class Animation implements Plotable
, Triggerable {
    int animationCounter;
    char triggerKey;
    SoundFile sound;

    public Animation (char c,
    SoundFile s) {
        triggerKey = c;
        animationCounter = 0;
        sound = s;
    }

    void triggerOn (char c) {...}
    void triggerOff (char c) {...}
    void plot () {}
}
```



# SOUNDFILE

- Der Methodenaufruf **play()** auf dem **SoundFile** startet die Wiedergabe des Sounds.
- Wo erzeugen wir das Objekt, das im Konstruktor übergeben werden soll?

```
class Animation implements Plotable
, Triggerable {
    int animationCounter;
    char triggerKey;
    SoundFile sound;

    public Animation (char c,
        SoundFile s) {...}

    void triggerOn (char c) {
        if (triggerKey == c) {
            animationCounter = 1;
            sound.play();
        }
    }

    void triggerOff (char c) {...}

    void plot () {}
}
```

# SOUNDFILE

- Wir erzeugen das **SoundFile** Objekt direkt im Konstruktoraufruf.
- Als zweiten Parameter übergeben wir den Dateipfad.
- Als ersten Parameter **this**. Dies bezieht sich in diesem Fall auf unseren Sketch.

```
Animation[] animations;

void setup () {
    animations = new Animation[]{
        new Animation('1',new SoundFile
            (this,"DateiName.mp3"));
    }
}

void draw () {...}

void keyPressed () {...}

void keyReleased () {...}
```

# DATA

- Im Sketchordner können wir einen Ordner mit dem Namen **data** erzeugen.
- Dateien in diesem Ordner können direkt mit dem Namen referenziert werden.

```
Animation[] animations;

void setup () {
    animations = new Animation[]{
        new Animation('1',new SoundFile
            (this,"DateiName.mp3"));
    }
}

void draw () {...}

void keyPressed () {...}

void keyReleased () {...}
```

AUSBlick

NÄCHSTE SITZUNG

ÜBUNG

# QUELLEN

- [http://blogs.scientificamerican.com/blogs/assets/tetrapod-zoology/File/placentals-molecular-phylogeny-600-px-tiny-July-2015-Darren-Naish-Tetrapod-Zoology\(1\).jpg](http://blogs.scientificamerican.com/blogs/assets/tetrapod-zoology/File/placentals-molecular-phylogeny-600-px-tiny-July-2015-Darren-Naish-Tetrapod-Zoology(1).jpg)
- <http://nos.twinsnd.co/image/127563852947>
- <http://nos.twinsnd.co/image/123557322230>
- 
- 
-