

PROCESSING

KLASSEN UND OBJEKTE

Created by Michael Kirsch & Beat Rossmly

INHALT

1. Rückblick

1. Processing Basics
2. Arrays
3. Characters
4. Strings
5. Funktionen
6. Funktionen - Aufbau

2. Theorie

1. Strings & besondere Funktionen
2. Und was ist das?
3. Weniger Durcheinander!
4. Klassen, new und Punkte?
5. Klassen
6. Objekte
7. Strings und Arrays
8. Processing Basics

3. Anwendung

1. Klassen
2. Objekte

4. Verknüpfung

1. Anwendung von Klassen
2. Extrahieren und Transferieren
3. Eine neue Klasse
4. Einbinden der Klasse
5. Verbesserung der Klasse

5. Ausblick

1. Nächste Sitzung
2. Übung

RÜCKBLICK

PROCESSING BASICS

Farben sind Datentypen

```
color a = color(255,0,0); // rgb  
color b = color(0); // schwarz  
color c = color(255); // weiß  
color d = color(100); // grauton
```

Füllfarbe

```
fill(a);
```

Umrissfarbe

```
stroke(b);
```

ohne Füllfarbe

```
noFill();
```

ohne Umrissfarbe

```
noStroke();
```

Stärke des Umrisses

```
strokeWeight(5);
```

ARRAYS

Initialisierung ohne konkrete Werte

```
int[] a = new int[3];
```

Initialisierung mit konkreten Werten

```
int[] a = new int[] {1,2,3};
```

Schreiben in Arrays

```
a[2] = 1234; // -> {1,2,1234}
```

Lesen aus Arrays

```
int b = a[2]; // -> b = 1234
```

formatierte Ausgabe von Arrays

```
printArray(a);
```

CHARACTERS

Erzeuge einen Character

```
char c = 'z';
```

Characters sind durch Zahlen
codierte Zeichen

```
int i = 'z';  
println(i); // -> 122
```

Groß- und Kleinschreibung ist
relevant!

```
char d = 'Z';  
println(d); // -> 90
```

STRINGS

Erzeuge einen String

```
String s = "Hello World!";
```

Strings zusammenfügen

```
String t = "TEST " + "1 2 3";
```

FUNKTIONEN

Mehrere Befehle...

```
stroke(255);  
fill(255);  
rect(200,100,50,50);
```

... können zu einem neuen
zusammengefasst werden.

```
void quadrat (int x, int y, int w, int c) {  
    stroke(c);  
    fill(c);  
    rect(x,y,w,w);  
}
```

Und jederzeit mit
unterschiedlichen Werten
aufgerufen werden.

```
quadrat(200,100,50,255);  
  
quadrat(234,637,40,0);  
  
quadrat(142,624,90,100);
```

Ist der Rückgabetyt **nicht
void** ist der letzte Befehl
immer **return**.

```
int dasDoppelteVon (int i) {  
    int v = 2*i;  
    return v;  
}
```

FUNKTIONEN - AUFBAU

Rückgabetyp: Jede Funktion hat einen Rückgabety

```
void ... (...) {...}  
int ... (...) {...}  
String ... (...) {...}
```

Name:

```
... helloWorld (...) {...}  
... dasDoppelteVon (...) {...}  
... maleKreis (...) {...}
```

Parameter: In den () werden die Werte übergeben, mit denen die Funktion arbeitet.

```
... ... () {...}  
... ... (int x) {...}  
... ... (int x, int y, color c) {...}
```

Rumpf: Alle im Rumpf stehenden Befehle werden bei Aufruf der Funktion ausgeführt.

```
... ... (...) {  
    fill(c);  
    ellipse(x,y,100,100);  
}
```

FUNKTIONEN - AUFBAU

Wichtig: mit `return` muss ein Wert oder eine Variable zurückgegeben werden, die dem selben Typ entspricht, der durch den Returntype festlegt wurde.

```
void helloWorld () {
    println("Hello world!");
    // void -> kein "return"
}

int dasDoppelteVon (int n) {
    int i = 2*n;
    return i;
    // i ist vom Typ int
}

String viel (String s) {
    return "viel " + s;
    // "viel " + s ist ein String
}
```

THEORIE

STRINGS & BESONDERE FUNKTIONEN

- Manche Strings möchte man gerne genauer untersuchen.
- Wie lang ist der String? Kommt ein gewisses Zeichen darin vor? Sind zwei Strings identisch?

```
String s = "Donaudampfschiffahrtsg  
esellschaftskapitän";
```

STRINGS & BESONDERE FUNKTIONEN

Länge eines Strings

```
s.length();
```

Index des ersten auftretenden
Characters **c**

```
s.indexOf('a');
```

Überprüft ob String **s** identisch
mit String **t** ist.

```
s.equals(t);
```

UND WAS IST DAS?

- Auch die Länge von Arrays lässt sich ermitteln.
- Durch das Anhängen von **.length** an den Variablen-Namen.
- Ohne **()**?
- Ist das noch eine Funktion?
- Was hat es nun mit diesem **.** auf sich?

```
int[] a = new int[] {1,2,3,4,5,6};  
  
int l = a.length;  
  
for (int n=0; n<l; n++) {  
    println(a[n]);  
}
```

WENIGER DURCHEINANDER!

- Klassen erlauben es uns eigene Datenstrukturen zu entwerfen.
- Wir können komplexe Strukturen bis in kleine uns bekannte Datentypen zerlegen und daraus zusammenfügen.
- Betrachten wir den sich bewegendem Kreis.

```
int x,y,d;

void setup () {
    size(600,600);
    x = 0;
    y = 0;
    d = 50;
}

void draw () {
    background(0);

    x++;

    fill(255,0,0);
    ellipse(x,y,d,d);
}
```

WENIGER DURCHEINANDER!

- Jeder weitere Kreis hätte wiederum die Variablen **x**, **y** und **d**.
- Und müsste bei jedem Durchgang von **draw** gezeichnet werden.

```
int x,y,d;

void setup () {
  size(600,600);
  x = 0;
  y = 0;
  d = 50;
}

void draw () {
  background(0);

  x++;

  fill(255,0,0);
  ellipse(x,y,d,d);
}
```

WENIGER DURCHEINANDER!

- Es wäre das einfachste könnte man einfach einen Typ **Kreis** bestimmen, der **x**, **y** und **d** enthält.
- Dann müsste man statt je drei Koordinaten-Paare und Durchmesser nur drei Kreise deklarieren.

```
// deklariere 3 Kreise
Kreis k1, k2, k3;

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

    // initialisiere die Kreise
}

void draw () {
    background(0);

    // zeichne die Kreise
}
```

WENIGER DURCHEINANDER!

- Einen **Kreis** ähnlich wie ein Array zu initialisieren wäre praktisch, da Variablen gleich festgelegt werden könnten.
- Das würde den Code kürzer, übersichtlicher und strukturierter machen.

```
// deklariere 3 Kreise
Kreis k1, k2, k3;

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

    // initialisiere mit: x,y,d
    k1 = new Kreis(35,13,45);
    k2 = new Kreis(96,45,74);
    k3 = new Kreis(13,85,36);
}

void draw () {
    background(0);

    // zeichne die Kreise
}
```

WENIGER DURCHEINANDER!

- Könnte man die Kreise mit Funktionen zeichnen?
- Daraus würde eine gute Lesbarkeit des Codes resultieren.

```
// deklariere 3 Kreise
Kreis k1, k2, k3;

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

    // initialisiere mit: x,y,d
    k1 = new Kreis(35,13,45);
    k2 = new Kreis(96,45,74);
    k3 = new Kreis(13,85,36);
}

void draw () {
    background(0);

    // zeichne die Kreise
    k1.plot();
    k2.plot();
    k3.plot();
}
```

KLASSEN, NEW UND PUNKTE?

- Was wir nun sehen können ist die Struktur, welche aus einer Klasse resultiert.
- Eine Klasse ist im wesentlichen ein Behälter für Variablen (**x**, **y** und **d**) und Funktionen (**plot()**).

```
// deklariere 3 Kreise
Kreis k1, k2, k3;

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

    // initialisiere mit: x,y,d
    k1 = new Kreis(35,13,45);
    k2 = new Kreis(96,45,74);
    k3 = new Kreis(13,85,36);
}

void draw () {
    background(0);

    // zeichne die Kreise
    k1.plot();
    k2.plot();
    k3.plot();
}
```

KLASSEN, NEW UND PUNKTE?

- Dies erleichtert uns Programme zu schreiben, die viele Male die selben Strukturen enthalten.
- Weil diese nun durch eine Klasse repräsentiert werden können!
- Wie vermitteln wir dem Computer was unserer Meinung nach ein Kreis ist und können muss?

```
// deklariere 3 Kreise
Kreis k1, k2, k3;

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

    // initialisiere mit: x,y,d
    k1 = new Kreis(35,13,45);
    k2 = new Kreis(96,45,74);
    k3 = new Kreis(13,85,36);
}

void draw () {
    background(0);

    // zeichne die Kreise
    k1.plot();
    k2.plot();
    k3.plot();
}
```

KLASSEN

- Außerhalb von **setup** und außerhalb von **draw** können wir unsere Klassen definieren.
- Dazu schreiben wir einfach das Schlagwort **class** und dahinter den Namen z.B. **Kreis**.
- Die anschließenden **{ }** enthalten nun die Definition der Klasse.

```
void setup () {...}  
  
void draw () {...}  
  
class Kreis {  
    ...  
}
```

KLASSEN

- Als erstes definieren wir alle Variablen die in der Klasse enthalten sein sollen.
- Das können Größen, Namen oder Identifikationshilfen sein. In unserem Fall aber **x**, **y** und **d**.
- Diese Variablen nennen sich Felder.

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

void draw () {...}

class Kreis {
  int x,y,d;

  ...
}
```

KLASSEN

- Als nächstes definieren wir, wie ein neuer **Kreis** erstellt wird.
- Was hier wie eine Funktion aussieht, nennt sich Konstruktor, hat den selben Namen wie die Klasse und keinen Returntype.
- Das Schlagwort **public** markiert, das der Konstruktor von überall aus aufgerufen werden kann.

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

void draw () {...}

class Kreis {
    int x,y,d;

    public Kreis (int x, int y, int d
) {
    this.x = x;
    this.y = y;
    this.d = d;
}

...
}
```

KLASSEN

- In den () des Konstruktors führen wir auf, was zum erstellen eines z.B. Kreises benötigt wird.
- Im Konstruktor weisen wir diese Werte den Feldern der Klasse zu. So kann sich der Computer die übergebenen Daten merken.

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

void draw () {...}

class Kreis {
    int x,y,d;

    public Kreis (int x, int y, int d
) {
    this.x = x;
    this.y = y;
    this.d = d;
}

...
}
```

KLASSEN

- **this** markiert nur, dass explizit das Feld der Klasse und nicht der übergebene Wert gemeint ist.
- Alternativ könnten auch unterschiedliche Namen für die Felder oder die entsprechenden Übergabewerte gewählt werden.

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

void draw () {...}

class Kreis {
    int x,y,d;
    // int xCoo, yCoo, ...

    public Kreis (int x, int y, int d
) {
    this.x = x;
    this.y = y;
    this.d = d;

    // xCoo = x;
    // yCoo = y;
    //...
}

...
}
```

KLASSEN

- Nach dem Konstruktor werden alle Funktionen der Klasse definiert.
- Dies geschieht wie das Definieren einer normalen Funktion.
- Returntype, Funktionsname, Übergabewerte, Rumpf
- Funktionen in Klassen nennen sich Methoden.

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

void draw () {...}

class Kreis {
    int x,y,d;

    public Kreis (int x, int y, int d
) {
    this.x = x;
    this.y = y;
    this.d = d;
}

void plot () {
    ellipse(x,y,d,d);
}
}
```

KLASSEN

Signalwort für Klassen + Name

```
class Kreis {
```

Felder

```
int x,y,d;
```

Konstruktor: Name +
Übergabewerte

```
public Kreis (int x ,int y, int d) {  
    ...  
}
```

Methoden

```
void plot () {  
    ...  
}
```

Ende des Klassenrumpfs

```
}
```

OBJEKTE

- **k1**, **k2** und **k3** sind Instanzen der Klasse **Kreis**. Diese Instanzen nennen wir Objekte.
- **new** ist der Operator zur Erzeugung neuer Objekte. Und steht vor dem Konstruktor-Aufruf.
- Der **.** ist ebenfalls ein Operator und lässt uns auf Felder und Methoden zugreifen.

```
Kreis k1, k2, k3;

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

    k1 = new Kreis(35,13,45);
    k2 = new Kreis(96,45,74);
    k3 = new Kreis(13,85,36);
}

void draw () {
    background(0);

    k1.plot();
    k2.plot();
    k3.plot();
}

class Kreis {
    ...
}
```

STRINGS UND ARRAYS

- Strings und Arrays sind also auch Objekte!
- Die Klasse **String** enthält also eine Methode **length()** die uns die Länge des Strings berechnet und zurückgibt!
- Arrays sind Objekte die mit **new** erzeugt werden und ein Feld **length** enthalten!

```
String s = "Hello World!";  
println(s.length());  
// -> 12
```

```
int[] i = new int[] {1,2,3,4};  
println(i.length);  
// -> 4
```

PROCESSING BASICS

key ist eine Variable, die immer den Character der zuletzt gedrückten Taste enthält.

```
// 'a' wird gedrückt gehalten
println(key); // -> 'a'
// 'a' wird losgelassen
println(key); // -> 'a'
// 'b' wird gedrückt und losgelassen
println(key); // -> 'b'
```

Die Variable **keyPressed** enthält den Zustand ob gerade irgendeine Taste gedrückt wird.

```
// 'a' wird gedrückt gehalten
println(keyPressed); // -> true
// 'a' wird losgelassen
println(keyPressed); // -> false
// 'b' wird gedrückt und losgelassen
println(keyPressed); // -> true
```

Die Funktion **keyPressed()** wird immer aufgerufen wenn eine Taste gedrückt wird.

```
void keyPressed ( ) {
    println(key);
}
```

ANWENDUNG

KLASSEN

```
// definiere Klasse "Car"  
// Felder: Koordinaten  
// Methoden: drawCar, setPosition  
  
void setup () {  
    size(800,800);  
    // initialisiere Car  
}  
  
void draw () {  
    // zeichne Car an Mausposition  
}
```

OBJEKTE

```
// definiere Klasse "Ball"  
// Felder: x,y,r,dx,dy  
// Methoden: handle, move, plot  
  
// deklarieren Array von Bällen  
  
void setup () {  
    size(800,800);  
    // initialisiere Array  
}  
  
void draw () {  
    // handle alle Bälle  
}
```

VERKNÜPFUNG

ANWENDUNG VON KLASSEN

- Betrachten wir unser Beispiel und versuchen alle wichtigen Aspekte zu erfassen.
- Über **setup** und **draw** hinaus kann alles weitere als zusammengehörig bestimmt werden.
- Wir müssen versuchen all das in einer Klasse zu vereinen.

```
int animationCounter;

void setup () {
  size(600, 400);
  animationCounter = 0;
}

void draw () {
  background(0);
  handleInput();
  drawAnimation();
}

void handleInput () {
  // ...
}

void drawAnimation () {
  // ...
}
```

EXTRAHIEREN UND TRANSFERIEREN

- Wir können beobachten, dass bereits alle wichtigen Komponenten einer Klasse vorhanden sind.
- Felder, (Teile vom) Konstruktor, Methoden
- Wichtig ist, diese Struktur zu übernehmen und in eine eigene Klasse zu übertragen.

```
// Feld einer Klasse
int animationCounter;

void setup () {
    size(600, 400);
    // Teil des Konstruktors
    animationCounter = 0;
}

void draw () {
    background(0);
    // Funktionsaufrufe
    handleInput();
    drawAnimation();
}

// Methoden der Klasse
void handleInput () {
    // ...
}
void drawAnimation () {
    // ...
}
```

EINE NEUE KLASSE

- Wir geben der Klasse den Namen **Animation**, dies beschreibt einerseits was wir in etwa von ihr erwarten können und ist allgemein genug um darauf aufbauen zu können.

```
class Animation {  
    // Felder  
  
    // Konstruktor  
  
    // Methoden  
}
```

EINE NEUE KLASSE

- Felder und Funktionen können nun einfach per copy & paste in unsere Klasse eingefügt werden.

```
class Animation {
    int animationCounter;

    // Konstruktor

    void handleInput () {
        if (keyPressed) {
            animationCounter++;
        } else {
            animationCounter = 0;
        }
    }

    void drawAnimation () {
        if (animationCounter>0) {
            int r = animationCounter*3;
            ellipse(300, 200, r, r);
        }
    }
}
```

EINE NEUE KLASSE

- Im Konstruktor handeln wir nun die Initialisierung aller Felder (falls notwendig) ab.
- Schon ist unsere Klasse fertig, kann verwendet werden und später bei Bedarf mit neuer Funktionalität erweitert werden.

```
class Animation {
    int animationCounter;

    public Animation () {
        animationCounter = 0;
    }

    void handleInput () {
        if (keyPressed) {
            animationCounter++;
        } else {
            animationCounter = 0;
        }
    }

    void drawAnimation () {
        if (animationCounter>0) {
            int r = animationCounter*3;
            ellipse(300, 200, r, r);
        }
    }
}
```

EINBINDEN DER KLASSE

- Nun können wir ein Objekt der Klasse **Animation** deklarieren und initialisieren, welches bereits alle nötigen Variablen und Funktionen wie vorher definiert enthält.
- Die Methoden der Klasse werden auf dem Objekt mit Hilfe des `.` Operators aufgerufen.

```
Animation a;

void setup () {
    size(600, 400);
    a = new Animation();
}

void draw () {
    background(0);
    a.handleInput();
    a.drawAnimation();
}

class Animation () {
    // ...
}
```

VERBESSERUNG DER KLASSE

- Wir bestimmen ein Feld, das den **Character** der damit verbundenen Taste enthält.
- Diesen übergeben wir als Parameter des Konstruktors.
- Bei **handleInput** übergeben wir später **key**, nun wissen wir ob die entsprechende Taste gedrückt ist!

```
class Animation {
    int animationCounter;
    char animationKey;

    public Animation (char c) {
        animationCounter = 0;
        animationKey = c;
    }

    void handleInput (char c) {
        if (keyPressed && c == animatio
nKey) {
            animationCounter++;
        } else {
            animationCounter = 0;
        }
    }

    void drawAnimation () {
        // ...
    }
}
```

VERBESSERUNG DER KLASSE

- Nun passen wir den Aufruf des Konstruktors in **setup** an.
- Und übergeben in **draw** **key** als Parameter von **handleInput**.

```
Animation a;

void setup () {
    size(600, 400);
    a = new Animation('1');
}

void draw () {
    background(0);
    a.handleInput(key);
    a.drawAnimation();
}

class Animation () {
    // ...
}
```

AUSBLICK

NÄCHSTE SITZUNG

- Eltern und Kinder!

ÜBUNG

QUELLEN