PROCESSING

EINE EINFÜHRUNG IN DIE INFORMATIK

Created by Michael Kirsch & Beat Rossmy

INHALT

TEAM

PROF. DR. HUSSMANN

- Professor der Medieninformatik an der LMU
- Dekan der Fakultät für Mathematik, Informatik und Statistik der LMU
- heinrich.hussmann@ifi.lmu.de

MICHAEL KIRSCH

- Master Informatik, Abschluss 2015
- Softwareentwickler bei Jambit
- michael.kirsch@ifi.lmu.de

BEAT ROSSMY

- Bachelor Kunst und Multimedia, Abschluss 2014
- Studium Master MCI
- beat.rossmy@campus.lmu.de

DARIO CASADEVALL

- Bachelor Mensch Computer Interaktion, Hamburg
- Studium Master MCI
- D.Casadevall@campus.lmu.de

SEMESTERÜBERSICHT

PROCESSING

- In der 1. Hälfte des Semesters werden die Grundlagen der objektorientierten Programmierung anhand von Processing beigebracht.
- In Vorlesung und Übung werden diese Kenntnisse praktisch angewandt.

JAVA

- In der 2. Hälfte des Semesters wird aufbauend darauf in die Programmierung mit Java eingestiegen.
- Ein Projekt wird Stück für Stück zuhause und in den Übungen erarbeitet.

PRÜFUNG

- Eine schriftliche Prüfung findet Ende des Semesters statt, in der der Stoff der Vorlesung geprüft wird.
- Der Termin wird noch bekanntgegeben.

CIP-KENNUNG & UNIWORX

CIP-KENNUNG

- Wird benötigt um während Vorlesung und Übungen mitarbeiten zu können.
- 19.-25. Oktober zwischen 19 und 20 Uhr in der Oettingenstr. 67 im Raum LU114
- http://www.rz.ifi.lmu.de/Merkblaetter/RechnerAnmeldung_WS.html/

UNIWORX

- Über Uniworx und die Vorlesungsseite werden alle Vorlesungs-Materialien verwaltet.
- Alle Studenten müssen sich eigenständig über Uniworx zum Kurs anmelden.

SEKUNDÄRLITERATUR & LINKS

PROCESSING

http://hello.processing.org/

JAVA

Head first Java

by Kathy Sierra & Bert Bates

LINKS

- https://uniworx.ifi.lmu.de/
- http://www.medien.ifi.lmu.de/lehre/ws1617/eipnf/

PROCESSING

EINE EINFÜHRUNG IN DIE INFORMATIK

Created by Michael Kirsch & Beat Rossmy

INHALT

EINLEITUNG

ZIELE

- Programmierung (imperativ und objektorientiert)
- Algorithmen und Datenstrukturen
- Anhand von Java

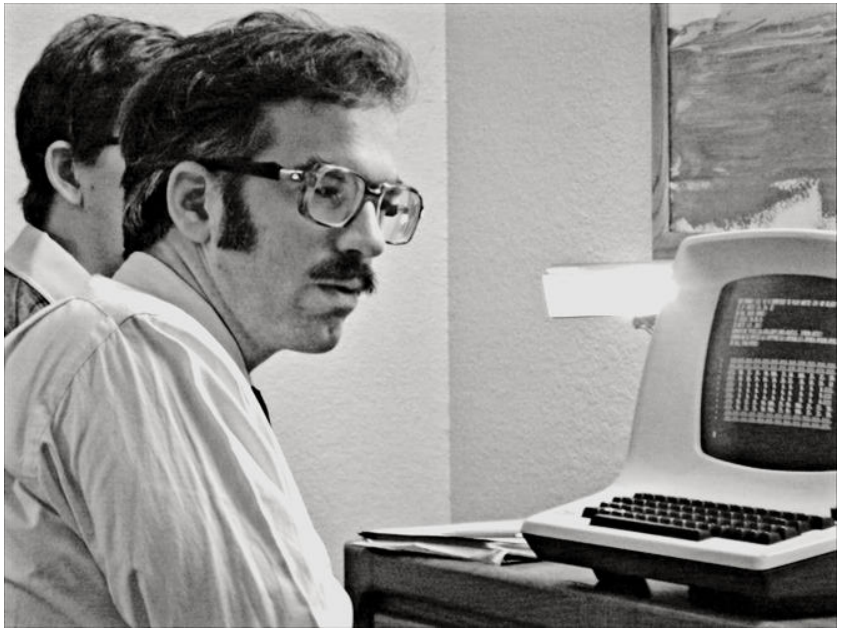
DER INFORMATIKER – EIN KLISCHEE

- Programmiert den ganzen Tag zuhause alleine in seinem dunklen Zimmer.
- Beschäftigt sich mit kryptischen Botschaften auf seinem Rechner (vor allem 0en und 1en).

http://www.cinema52.com/2013/wp-content/uploads/2013/09/Nedry.png

DER INFORMATIKER – EIN KLISCHEE

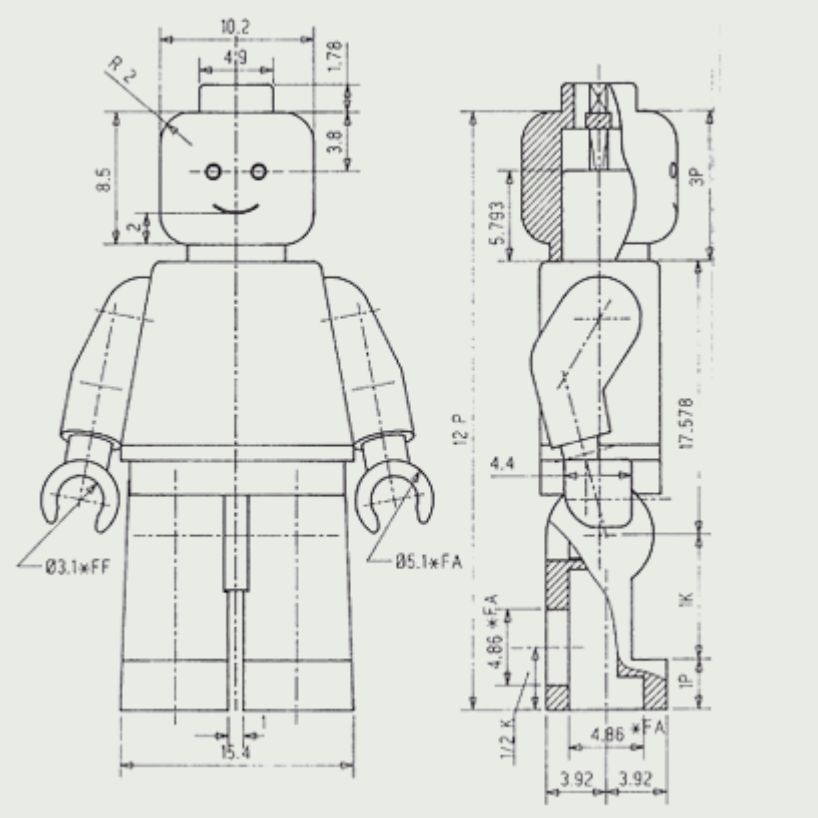
- Das Größte ist für ihn sein System neu aufzusetzen und bei Fehlern anzupassen.
- Programmieren um des Programmieren willens! Wozu auch sonst?



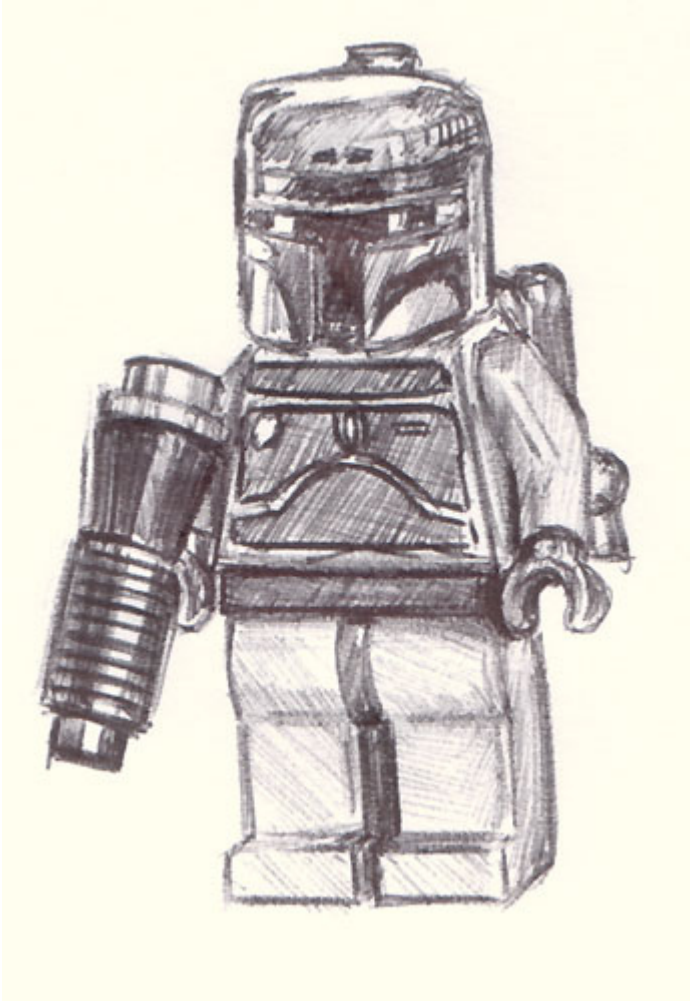
http://h.fastcompany.net/multisite_files/cocreate/imagecache/slideshow_large/slideshow/2013/07/1683410-slide-s-7-vintage-geek-computer-chess-looks-back-at-1980s-era-nerd-culture.jpg

Muss das so sein?

WERKZEUG NICHT SELBSTZWECK!



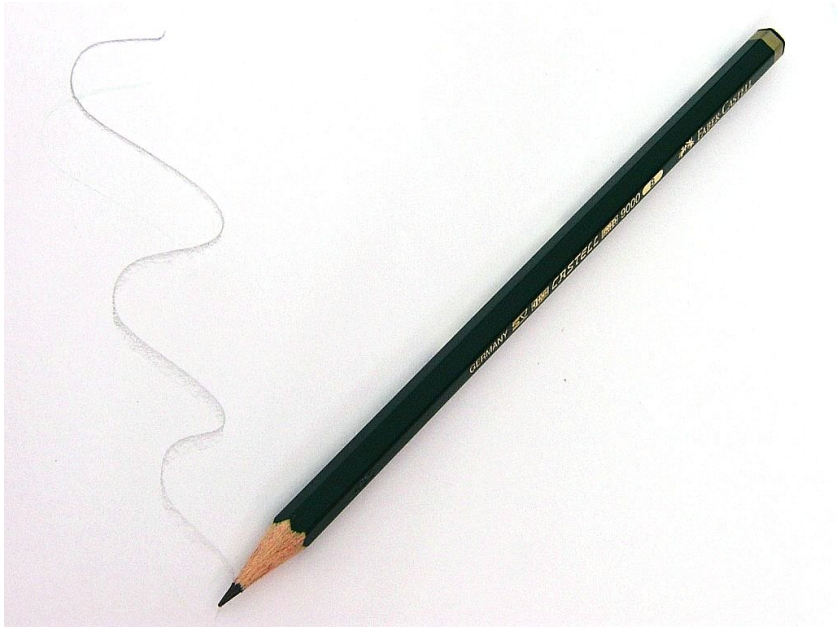
http://blog.dersven.de/user/files/gadgets/lego30-technischeZeichnung.png



https://c1.staticflickr.com/1/140/330870137_e8dec5b331.jpg

WERKZEUG NICHT SELBSTZWECK!

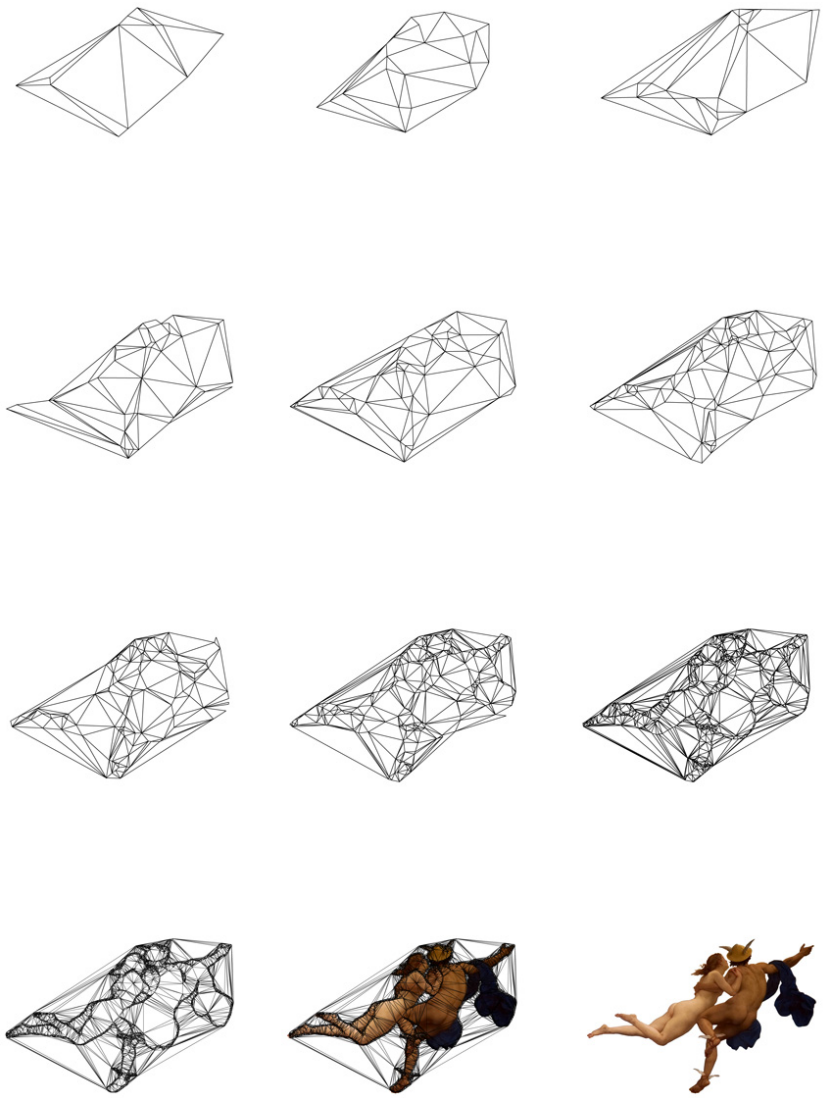
- Informatik kann ein Werkzeug sein wie Malerei, Zeichnung, …
- Wie und wozu wir diese einsetzen, ist uns überlassen!



https://upload.wikimedia.org/wikipedia/commons/2/23/Bleistift1.jpg

DIGITALE KUNST

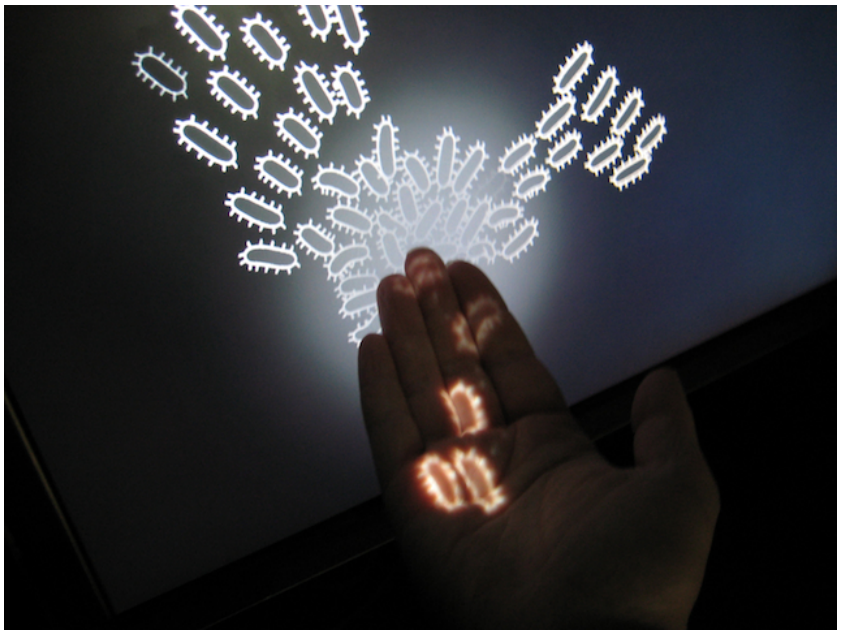
- Programmierung als Mittel um Ideen zu verwirklichen, die so nicht mit herkömmlicher Software umsetzbar sind.
- Videoeffekte, 3D-Animation, ...
- Strata #3



http://www.quayola.com/Qsite/wp-content/uploads/2011/07/process.jpg

GRENZBEREICH DIGITAL/ANALOG

- Aufeinandertreffen digitaler Technologie und realer Umgebung.
- Programmierung zur Koordinierung und Generierung.
- KEYFLEAS



http://csugrue.com/images/db/db_webpic_1.jpg

GRAUE EMINENZ

- Programmierung als unsichtbares Gehirn hinter Installationen, Maschinen, ...
- Worin steckt eigentlich überall Informatik?
- LIGHT KINETICS

http://payload372.cargocollective.com/1/4/156755/9744712/prt_594x203_1429610653.jpg

THEORIE

PROGRAMMIERSPRACHEN

- Programmiersprache: unser Weg zur Kommunikation mit dem Computer.
- Wie bei jeder Sprache gibt es Regeln und Konventionen, an welche man sich halten muss, damit man auch verstanden wird!

JAVA EINE SPRACHE VON VIELEN.

- Die unterschiedlichen Sprachen unterscheiden sich in ihrer Nähe entweder zur Maschine oder eben zum Menschen.
- D.h.: Eine Sprache ist entweder für uns leicht zu verstehen, muss dann aber für den Computer in mehreren Schritten verständlich gemacht werden (automatisch).
- Oder die Sprache ist schwer zu lesen/verstehen, kann aber schneller vom Computer verarbeitet werden.

WARUM JAVA?

- Java findet die richtige Balance zwischen Verständlichkeit und Computernähe.
- Java ist Plattform-unabhängig (läuft auf Windows, Mac und Linux).
- Java bietet einen Dialekt, der uns den Einstieg noch weiter vereinfacht: Processing!

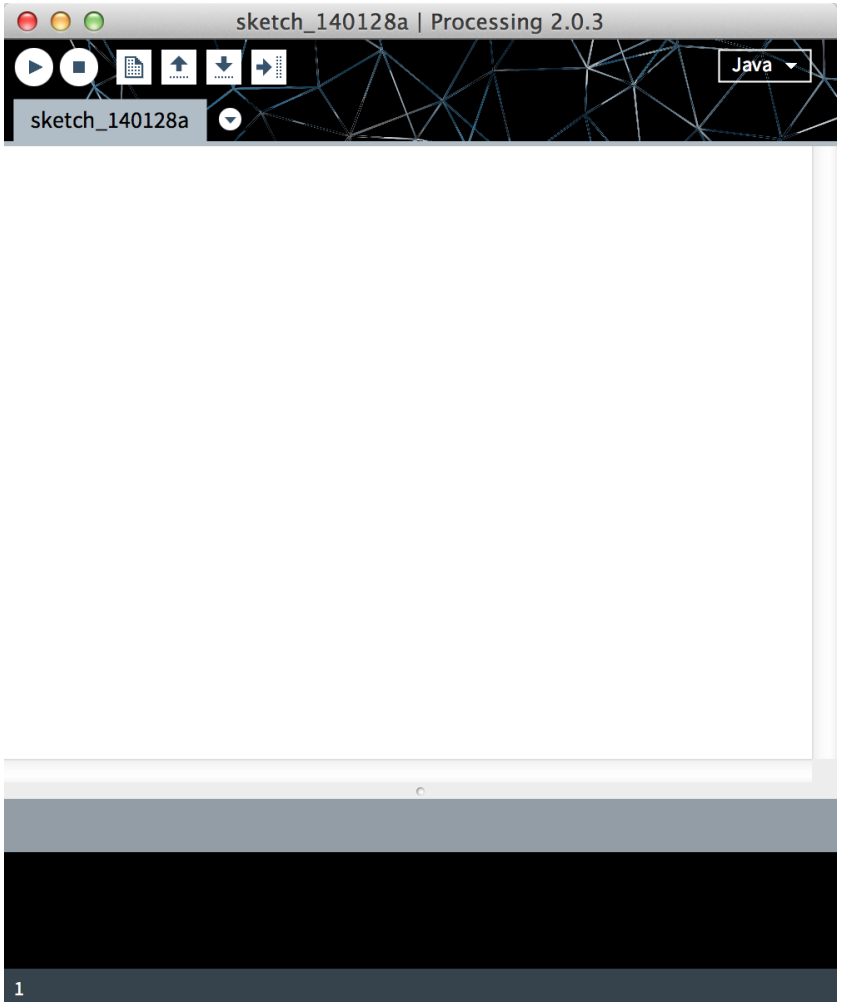
PROCESSING

- Java Dialekt erfunden um Künstlern und anderen Kreativen den Zugang zu Programmierung zu erleichtern.
- Selbe Syntax wie Java.
- Aber weniger Code um grafische Ausgabe zu erreichen!

http://upload.wikimedia.org/wikipedia/commons/thumb/5/59/Processing_Logo_Clipped.svg/1000px-Processing_Logo_Clipped.svg.png

PROCESSING IDE

- Play/Stop Button zum Starten des Programms.
- Textfeld zur Eingabe unseres Codes.
- Konsole: Textausgabe des Programms an uns, um Fehler mitzuteilen oder Werte zu ermitteln.



http://codecodigo.com/code/img/prosjs/points/proc_ide_02.png

ANWENDUNG

SPRUNG INS KALTE WASSER

Was ich sagen möchte:	**Wie ich es sage:**
Male ein Rechteck!	`rect(x,y,w,h);`
Schreibe "Hello World!".	`print("Hello World!");`
Erzeuge eine zufällige Zahl zwischen 0 und 1.	`random(1);`
Fülle den Hintergrund schwarz.	`background(0);`

WIR BASTELN EIN DAUMENKINO

- Vorbereitung (**`setup`**): wir schneiden Papier in der Größe 100mm * 100mm
- Zeichnen (**`draw`**): Male roten Hintergrund, zeichne einen Kreis darauf.

```
void setup () {
   size(100,100);
}

void draw () {
   background(255,0,0);
   ellipse(50,50,50,50);
}
```

SETUP? DRAW?

- **`setup`** wird genau einmal und zwar zu Beginn des Programms ausgeführt.
- **`draw`** wird danach immer und immer wieder Zeile für Zeile durchlaufen.
- Das betrifft alle Zeilen innerhalb der **`{}`**

```
void setup () {
   size(100,100);
}

void draw () {
   background(255,0,0);
   ellipse(50,50,50,50);
}
```

PROCESSING BASICS

STRUCTURE

Setup Funktion	`void setup() {...}`
Zeichen Funktion	`void draw() {...}`
Größe des Programms	`size(w,h);`
Schreibe einen Wert in die Konsole	`println("something");`

PROCESSING BASICS

2D PRIMITIVES

Rechteck	`rect(x,y,w,h);`
Ellipse	`ellipse(x,y,w,h);`
Linie	`line(x1,y1,x2,y2);`
Punkt	`point(x,y);`

PROCESSING BASICS

COLOR

Hintergrundfarbe	`background(r,g,b);`
Füllfarbe	`fill(r,g,b);`
Umrissfarbe	`stroke(r,g,b);`

VERKNÜPFUNG

KANN MAN WERTE SPEICHERN?

- Vor **`setup`** können Variablen definiert werden, auf welche im ganzen Programm zugegriffen werden kann.
- In **`setup`** werden diese initialisiert. D.h. ihr Wert wird bestimmt. Z.B. soll **`x`** zu Beginn 0 sein.

```
int x;

void setup () {
   x = 0;
   size(100,100);
}

void draw () {
   background(255,0,0);
   ellipse(x,50,50,50);
}
```

KANN MAN WERTE VERÄNDERN?

- **`x = x+1`**? mathematisch bedeutet das: 0=1?
- **`=`** fungiert in Java und Processing als Zuweisungsoperator. D.h. in die Variable **`x`** wird der Wert **`x+1`** gespeichert.

```
int x;

void setup () {
   x = 0;
   size(100,100);
}

void draw () {
   x = x+1;
   background(255,0,0);
   ellipse(x,50,50,50);
}
```

Was passiert in diesem Programm?

Wie können wir verhindern, dass der Kreis den sichtbaren Bereich verlässt?

WENN DAS SO IST, DANN...

Bedingungen helfen uns bestimmte Situationen zu behandeln.

```
if (x > 100) {
  x = 0;
}
```

`if` markiert die in den `()` folgende Aussage als Bedingung.

```
if (x == 100) {
  x = 0;
}
```

In den `{}` steht die daraus resultierende Handlungsabfolge (falls wahr).

```
if (true) {
  x = 0;
}
```

BOOLSCHE OPERATOREN

Größer	`if (a > b) {...}`
Größer gleich	`if (a >= b) {...}`
Gleich	`if (a == b) {...}`
Kleiner gleich	`if (a <= b) {...}`
Kleiner	`if (a < b) {...}`
Ungleich	`if (a != b) {...}`

AUSBLICK

NÄCHSTE SITZUNG

- Weitere mathematische Datentypen
- Strukturhilfen
- Processing Reference

ÜBUNG

Unser Kreis soll nun bei Verlassen einer Seite seine Richtung ändern!

QUELLEN

- http://www.cinema52.com/2013/wp-content/uploads/2013/09/Nedry.png
- http://h.fastcompany.net/multisite_files/cocreate/imagecache/slideshow_large/slideshow/2013/07/1683410-slide-s-7-vintage-geek-computer-chess-looks-back-at-1980s-era-nerd-culture.jpg
- http://blog.dersven.de/user/files/gadgets/lego30-technischeZeichnung.png
- https://c1.staticflickr.com/1/140/330870137_e8dec5b331.jpg
- https://upload.wikimedia.org/wikipedia/commons/2/23/Bleistift1.jpg
- http://www.quayola.com/Qsite/wp-content/uploads/2011/07/process.jpg
- http://csugrue.com/images/db/db_webpic_1.jpg
- http://payload372.cargocollective.com/1/4/156755/9744712/prt_594x203_1429610653.jpg
- http://upload.wikimedia.org/wikipedia/commons/thumb/5/59/Processing_Logo_Clipped.svg/1000px-Processing_Logo_Clipped.svg.png
- http://codecodigo.com/code/img/prosjs/points/proc_ide_02.png

PROCESSING

STRUKTUR UND INPUT

Created by Michael Kirsch & Beat Rossmy

INHALT

RÜCKBLICK

PROCESSING BASICS

STRUCTURE / COLOR

Setup Funktion	`void setup() {...}`
Draw Funktion	`void draw() {...}`
Größe des Sketchs	`size(w,h);`
Konsolenausgabe	`println("something");`
Hintergrundfarbe	`background(r,g,b);`
Füllfarbe	`fill(r,g,b);`
Umrissfarbe	`stroke(r,g,b);`
Zufallszahl	`random(1);`

PROCESSING BASICS

2D PRIMITIVES

Rechteck	`rect(x,y,w,h);`
Ellipse	`ellipse(x,y,w,h);`
Linie	`line(x1,y1,x2,y2);`
Punkt	`point(x,y);`

1,2,3,... – INTEGER!

- Mit **`int`** bezeichnen wir Variablen die ganzzahlige Werte enthalten.
- Addition und Subtraktion mit anderen Integern funktioniert problemlos.
- Was allerdings passiert bei Multiplikation und Division?

```
int x = 4;

println(x+2); // prints "6"
println(x-2); // prints "2"

println(x/2); // prints "2"
println(x*2); // prints "8"

println(x/3); // prints ???
```

BOOLSCHE OPERATOREN

Größer	`if (a > b) {...}`
Größer gleich	`if (a >= b) {...}`
Gleich	`if (a == b) {...}`
Kleiner gleich	`if (a <= b) {...}`
Kleiner	`if (a < b) {...}`
Ungleich	`if (a != b) {...}`

BEDINGUNGEN

- Mit **`if`** können wir auf bestimmte Situationen reagieren.
- Ein bestimmter Fall tritt nur unter der Bedingung ein, dass die Aussage in den **`( )`** wahr ist.
- Manche Aussagen können stets wahr sein.

```
int x;

void setup () {
  x = 0;
}

void draw() {
  x = x+1;
  if (x == 10) {
    x = 0;
  }
  if (true) {
    println(x);
  }
}
```

THEORIE

... SONST!

Mit mehreren **`if`** kann auch auf das Gegenteil der ursprünglichen Aussage reagiert werden.

```
if (x == 100) {
  println("x ist genau 100");
}
if (x != 100) {
  println("x ist nicht 100");
}
```

Mit **`else`** lässt sich einfacher das Gegenteil einer Aussage behandeln.

```
if (x == 100) {
  println("x ist genau 100");
}
else {
  println("x ist nicht 100");
}
```

Das ist hilfreich wenn unsere Bedingungen komplexer werden.

```
if (x == 100 && y > 50) {
  println("x ist genau 100");
}
else {
  println("x!=100 und/oder y kleinergleich 50");
}
```

WENN MEHR GELTEN SOLL!

Und (sowohl als auch)	`if (x > 50 && y < 50) {}`
Oder (das eine, das andere oder beides)	`if (x > 50 || y < 50) {}`
Nicht (das Gegenteil davon)	`if (x > 50 || !(y >= 50)) {}`
Klammern halten Aussagen zusammen!	`if ((x>5 || !(y>=5)) || (x==5 && y==5)) {}`

KLEINE SCHRITTE UND GROSSE SCHRITTE.

- Manchmal reichen ganze Zahlen nicht aus um jeden Sachverhalt entsprechend darstellen zu können.
- "Ein Apfel", "Ein Haus", ...
- "Eine halbe Stunde", "Ein halber Meter", ...

http://zak-site.com/Great-American-Novel/images/44-102/ff98-one-step.jpg

KLEINE SCHRITTE UND GROSSE SCHRITTE.

- Angenommen unser Kreis soll sich nun doppelt so schnell bewegen. Was müssen wir an unserem bisherigem Sketch ändern?
- Aber wenn er sich nur halb so schnell bewegen soll...

```
int x;

void setup () {
   size(100,100);
   x = 0;
}

void draw () {
   background(255,0,0);
   x = x+1;
   ellipse(x,50,50,50);
}
```

FLOAT

- Es gibt natürlich auch andere Datentypen als Integer.
- Mit **`float`** definieren wir Variablen, die Gleitkommazahlen enthalten.
- D.h. alle Zahlen, die bei uns üblicherweise ein Komma enthalten.
- Aber Vorsicht!

```
float x;

void setup () {
   size(100,100);
   x = 0;
}

void draw () {
   background(255,0,0);
   x = x+0.5;
   ellipse(x,50,50,50);
}
```

IMMER UND IMMER WIEDER...

- Von Zeit zu Zeit passiert es, dass man manche Sachen immer und immer wieder zu erledigen hat...
- Selbst wenn es einfache Aufgaben sind, wie "zeichne 5 zufällige Punkte" kann sich dadurch der Programmcode immens verlängern.

http://img.thesun.co.uk/multimedia/archive/01647/LOW-RES-GETTY-2694_1647748a.jpg

IMMER UND IMMER WIEDER...

- Aber wir erkennen ein Muster!
- Es entstehen Blöcke von drei Zeilen, die sich wiederholen.
- Könnte man diese Blöcke automatisch wiederholen...

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

  float x = random(width);
  float y = random(height);
  ellipse(x,y,5,5);

  x = random(width);
  y = random(height);
  ellipse(x,y,5,5);

  x = random(width);
  y = random(height);
  ellipse(x,y,5,5);

  x = random(width);
  y = random(height);
  ellipse(x,y,5,5);

  x = random(width);
  y = random(height);
  ellipse(x,y,5,5);
}
```

FOR

- **`for`** leitet unsere Schleife ein.
- In den **`()`** beschreiben wir die Wiederholungen: "Zähle von 0 bis 4, bei dem 5. Durchgang breche ab."
- **`{}`** enthalten die auszuführenden Befehle und wird Schleifenrumpf genannt.

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

  for (int n=0; n<5; n++) {
    float x = random(width);
    float y = random(height);
    ellipse(x,y,5,5);
  }
}
```

WIE KANN ICH EINFLUSS NEHMEN?

- Alles läuft so wie ich es geschrieben habe!
- Aber wie kann ich mein Programm während der Laufzeit beeinflussen?
- Wie funktioniert das mit der Maus und dem Keyboard?

http://dougengelbart.org/images/pix/img0030.jpg

WIE KANN ICH EINFLUSS NEHMEN?

- **`mouseX`** und **`mouseY`** sind Variablen, die immer die aktuelle Position der Maus enthalten!
- **`mousePressed`** ist **`true`** wenn die Linke Maustaste gedrückt ist, ansonsten **`false`**.
- Wie wir auf Tasten des Keyboards reagieren können, lernen wir in der nächsten Sitzung.

```
void setup () {
    size(800,800);
}

void draw () {
    background(0);
    ellipse(mouseX,mouseY,100,100);
}
```

PROCESSING BASICS

Einzeiliger Kommentar

```
int x = 0; // x ist 0
x = x+1; // x ist 1
```

Mehrzeiliger Kommentar

```
int x = 0;
/*
x ist 0
gleich ist x 1
*/
x = x+1;
```

Inkrementieren

```
int x = 0;
x = x+1;
x += 1;
x++; // x ist 3!
```

Dekrementieren

```
int x = 0;
x = x-1;
x -= 1;
x--; // x ist -3!
```

PROCESSING BASICS

Breite des Fensters

```
int w;

void setup () {
    size(700,350);

    w = width;
    // w ist 700
}
```

Höhe des Fensters

```
int h;

void setup () {
    size(700,350);

    h = height;
    // h ist 350
}
```

ANWENDUNG

RANDOM

```
void setup () {
    size(800,800);
}

void draw () {
    background(0);
    // zufälliger RGB-Hintergrund?
}
```

ELSE

```
int counter;

void setup () {
    size(800,800);
    // counter initialisieren
}

void draw () {
    background(0);
    // counter inkrementieren

    // counter zurücksetzen

    // Kreis oder Rechteck
    // abhängig von counter
}
```

FLOAT

```
float x1,y1,x2,y2;

void setup () {
    size(800,800);
    // Koordinaten initialisieren
}

void draw () {
    background(0);
    // x1, x2 minimal inkrementieren

    // x1 zu int konvertieren

    // wandernde Kreise zeichnen
}
```

FOR

```
void setup () {
    size(800,800);
}

void draw () {
    background(0);
    // zeichne mehrere Rechtecke mit zufälligen Parametern in Schleife
}
```

MOUSEX & MOUSEY

```
void setup () {
    size(800,800);
}

void draw () {
    // zeichne Hintergrund abhängig ob Maus gedrückt ist oder nicht

    // zeichne Kreis an der aktuellen Mausposition
}
```

VERKNÜPFUNG

DIE PROCESSING REFERENCE

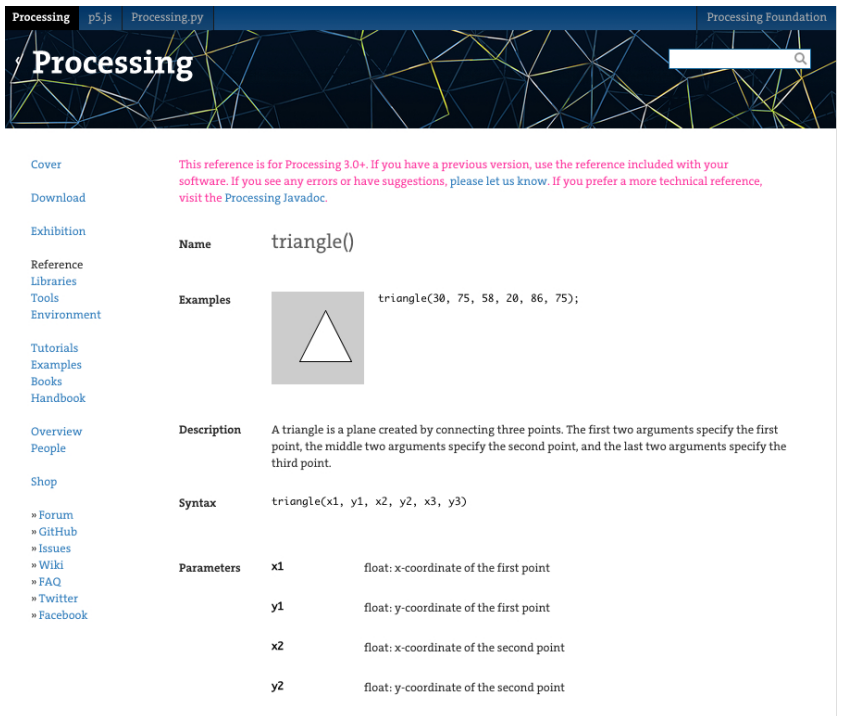
WIR KÖNNEN HIER LEIDER NICHT ALLES LERNEN...

- https://processing.org/reference/
- Inhalte sind übersichtlich nach Themen sortiert: "2D-Primitives", "Image", "Math", ...
- Wir könne auch finden was wir suchen, ohne genau zu wissen wie es heißt!

https://processing.org/reference/

DIE PROCESSING REFERENCE

- Die Dokumentation enthält immer Beispiel-code, beschreibende Texte und weiter-führenden Links zu verwandten Themen.
- Mit etwas Übung findet man sehr schnell was man sucht.
- Tipp: für Suchanfragen verwende zuerst die Processing interne Suche!



https://processing.org/reference/triangle_.html

EIN KREIS IST NICHT GENUG!

Wir lassen nun zwei Kreise sich in unterschiedliche Richtungen bewegen.

EIN KREIS IST NICHT GENUG!

- Wir deklarieren vier **`float`** Variablen für die Koordinaten der Kreise.
- Diese werden mit Werten innerhalb der Fensterdimensionen initialisiert.
- Durch inkrementieren / dekrementieren der X-Koordinaten bewegen sich die Kreise.

```
float x1,y1,x2,y2;

void setup () {
  size(500,500);
  x1 = 50;
  y1 = 50;
  x2 = 100;
  y2 = 100;
}

void draw () {
  background(0);
  x1++;
  x2--;

  ellipse(x1,y1,20,20);
  ellipse(x2,y2,20,20);

  if (x1>width) {x1 = 0;}
  if (x2<0) {x2 = width;}
}
```

BRINGEN WIR ETWAS ZUFALL INS SPIEL.

- Bei jedem neuen Start des Programms erscheinen nun die Kreise an anderen Orten innerhalb des Fensters.
- Auch die Bewegungs-geschwindigkeit der Kreise ist nun abhängig vom Zufall.

```
float x1,y1,x2,y2;

void setup () {
  size(500,500);
  x1 = random(width);
  y1 = random(height);
  x2 = random(width);
  y2 = random(height);
}

void draw () {
  background(0);
  x1 += random(10);
  x2 -= random(10);

  ellipse(x1,y1,20,20);
  ellipse(x2,y2,20,20);

  if (x1>width) {x1 = 0;}
  if (x2<0) {x2 = width;}
}
```

AUSBLICK

NÄCHSTE SITZUNG

- Ordnung in unseren Variablen
- Strukturierung unserer Befehle
- **`void`** und andere Zauberworte

ÜBUNG

...

QUELLEN

- http://zak-site.com/Great-American-Novel/images/44-102/ff98-one-step.jpg
- http://img.thesun.co.uk/multimedia/archive/01647/LOW-RES-GETTY-2694_1647748a.jpg
- http://dougengelbart.org/images/pix/img0030.jpg
- https://processing.org/reference/
- https://processing.org/reference/triangle_.html

PROCESSING

SCHUBLADEN UND ZEICHEN

Created by Michael Kirsch & Beat Rossmy

INHALT

RÜCKBLICK

PROCESSING BASICS

Einzeiliger Kommentar	`// das ist ein einzeiliger Kommentar`

Mehrzeiliger Kommentar	`/* das ist`
	`ein mehrzeiliger`
	`Kommentar */`

Inkrementieren	`x = x+1;`
	`x += 1;`
	`x++;`

Dekrementieren	`x = x-1;`
	`x -= 1;`
	`x--;`

Fensterbreite	`w = width;`
Fensterhöhe	`h = height;`

FLOAT

- **`float`** zeichnet Variablen aus, die Gleitkommazahlen enthalten.
- Achtung: diese markieren - anders als im deutschsprachigen Raum üblich - ihre Nachkommastellen mit einem Punkt!
- Vorsicht beim Mischen von Datentypen ist geboten!

```
int a = 5;
int b = 2;

float z = a/b;
println(z); // -> 2.0 !!!

float c = 5;
float d = 2;

z = c/d;
println(z); // -> 2.5

int e = 5;
float f = 2;

z = e/f;
println(z); // -> 2.5 !!!

float g = 5;
int h = 2;

z = g/h;
println(z); // -> 2.5 !!!
```

FOR

- **`for`** leitet eine Schleife ein.
- In den **`()`** wird die Wiederholung des Rumpfes beschrieben. Angefangen bei **`n`** bis (hier) **`5`** im Angegebenen Intervall (**`++`**).
- **`{}`** enthalten die auszuführenden Befehle.

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

  for (int n=0; n<5; n++) {
    float x = random(width);
    float y = random(height);
    ellipse(x,y,5,5);
  }
}
```

ELSE

`if` bedingt das Ausführen einzelner oder mehrerer Befehle abhängig von einer Aussage.

```
if (x == 100) {
  println("x ist genau 100");
}
```

Die Befehle im Rumpf von **`else`** treten nur ein, ist die Aussage von **`if`** falsch.

```
else {
  println("x ist nicht 100");
}
```

MOUSE INPUT

`mouseX` und **`mouseY`** enthalten die aktuelle Position des Cursors.

```
// Kreis an Cursor Position
ellipse(mouseX, mouseY, 100, 100);
```

`mousePressed` gibt den Status "gedrückt/ungedrückt" der linken Maustaste wieder.

```
if (mousePressed) {
  println("Linke Maustaste gedrückt!");
}
else {
  println("Linke Maustaste nicht  gedrückt!");
}
```

THEORIE

WIR BRAUCHEN MEHR ORDNUNG!

- Je komplexer unsere Sketche werden, desto mehr Variablen benötigen wir, desto unübersicht-licher wird das ganze.
- Wenn es so etwas wie Schubladen gäbe, in denen man alles zusammengehörige verstauen könnte...

http://nos.twnsnd.co/image/117523960441

WIR BRAUCHEN MEHR ORDNUNG!

- Stelen wir uns vor, wir würden nun 6 sich bewegende Kreise zeichnen wollen...
- Wir bräuchten 6 Variablen für X-Koordinaten, 6 für Y-Koordinaten, ...

```
int x1, x2, x3, x4, x5, x6;
int y1, y2, y3, y4, y5, y6;

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

  x1 = 0;
  x2 = 0;
  x3 = 0;
  x4 = 0;
  x5 = 0;
  x6 = 0;

  y1 = 0;
  y2 = 0;
  y3 = 0;
  y4 = 0;
  y5 = 0;
  y6 = 0;
}

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

ARRAY

- Ein Array wird mit **[ ]** markiert.
- Vor den **[ ]** steht der Datentyp den das Array *ausschließlich* enthält.
- Nach den **[ ]** steht der Name mit dem wir das Array deklarieren.

```
int[] x;
int[] y;

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

  x = new int[6];
  y = new int[6];

  x[0] = 0;
  x[1] = 0;
  x[2] = 0;
  x[3] = 0;
  x[4] = 0;
  x[5] = 0;

  y[0] = 0;
  y[1] = 0;
  y[2] = 0;
  y[3] = 0;
  y[4] = 0;
  y[5] = 0;
...
```

ARRAY

- Um das *neue* Array zu initialisieren, verwenden wir den Operator **`new`**, wiederum den Datentyp und anschließend in den **`[ ]`** die Länge unseres Speichers.
- In diesem Fall können wir also genau die 6 X-Koordinaten im Array **`x`** ablegen.

```
int[] x;
int[] y;

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

  x = new int[6];
  y = new int[6];

  x[0] = 0;
  x[1] = 0;
  x[2] = 0;
  x[3] = 0;
  x[4] = 0;
  x[5] = 0;

  y[0] = 0;
  y[1] = 0;
  y[2] = 0;
  y[3] = 0;
  y[4] = 0;
  y[5] = 0;
...
```

ARRAY

- Um nun auf die einzelnen im Array enthaltenen Werte zuzugreifen, verwenden wir wiederum **[ ]** und darin die Stelle von der wir lesen oder in die wir schreiben wollen.
- Wichtig: Die erste Stelle des Speichers hat stets die Nummer **0**!
- Aber das ist ja jetzt noch länger?

```
int[] x;
int[] y;

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

  x = new int[6];
  y = new int[6];

  x[0] = 0;
  x[1] = 0;
  x[2] = 0;
  x[3] = 0;
  x[4] = 0;
  x[5] = 0;

  y[0] = 0;
  y[1] = 0;
  y[2] = 0;
  y[3] = 0;
  y[4] = 0;
  y[5] = 0;
...
```

ARRAY

- Verknüpft mit unserem restlichen Wissen können wir nun einen Vorteil aus dieser neuen Struktur gewinnen.
- **`for`** hilft uns nun den wiederkehrenden Zugriff auf Teile des Arrays (hier die Initialisierung) zu verkürzen.

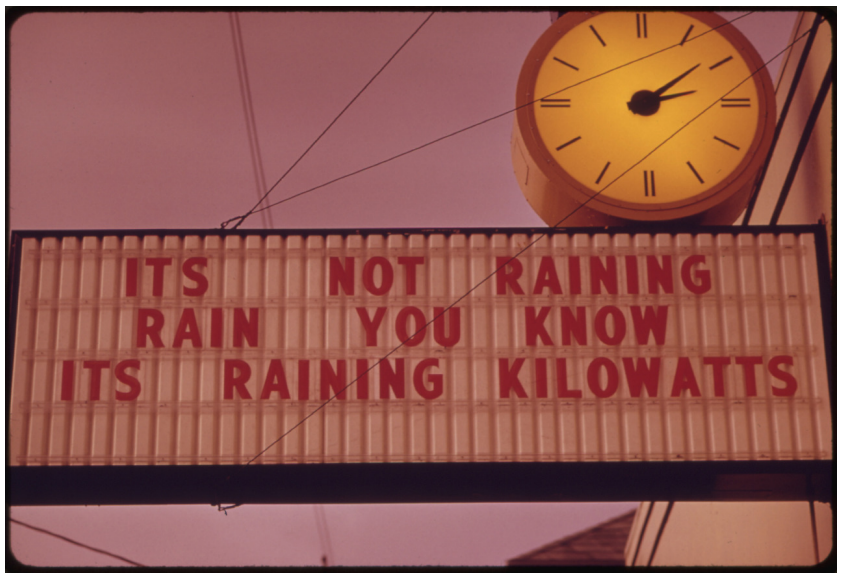
```
int[] x;
int[] y;

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

  x = new int[6];
  y = new int[6];

  for (int n=0; n<6; n++) {
    x[n] = 0;
    y[n] = 0;
  }
}
```

ARRAY

Initialisierung ohne konkrete Werte	`int[] a = new int[3];`
Initialisierung mit konkreten Werten	`int[] a = new int[] {1,2,3};`
Tipp: formatierte Ausgabe von Arrays	`int[] a = new int[] {1,2,3};` `printArray(a);`

NICHT MATHEMATISCHE DATENTYPEN?

- Natürlich gibt es neben **`int`** und **`float`** auch andere nicht mathematische Datentypen.
- Wir haben diese schon bei der Ausgabe von Text in der Konsole verwendet!



http://nos.twnsnd.co/image/124161898747

CHARACTERS & STRINGS

- Jedes Zeichen unseres Alphabets, unsere Ziffern und Sonderzeichen werden im Computer als Character gespeichert. Intern sind diese als Zahlen repräsentiert. Diese Codierung nennt sich Unicode.
- Variblen des Typen Character deklarieren wir mit **`char`** und initialisieren in `''`.

```
char c = 'a';
println(c);
// -> a

int i = 'a';
println(i);
// -> 97
```

CHARACTERS & STRINGS

- Wörter und Sätze sind Reihen von Zeichen. Strings enthalten alle Zeichen eines Worts oder eines Satzes.
- Wir deklarieren mit **`String`** und initialisieren in **`""`**.
- Java/Processing unterscheidet zwischen Groß- und Kleinschreibung. Also **`String`** immer mit großem **`S`**.

```
String h = "Hello";
println(h);
// -> Hello

String w = "World!";
println(w);
// -> World!

String hw = h+w;
println(hw);
// -> HelloWorld!

hw = h+" "+w;
println(hw);
// -> Hello World!

hw = "Hello World!";
println(hw);
// -> Hello World!
```

HANDLUNGEN ZUSAMMENFASSEN!

- Angenommen wir wollen nun nicht immer identische Höhe und Breite in **`ellipse`** eingeben um einen Kreis zu erhalten.
- Könnten wir nicht einen eigenen Befehl bestimmen der einfach einen Kreis zeichnet?

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

void draw () {
  stroke(255,0,0);
  fill(255,0,255);
  ellipse(33,44,55,55);

  stroke(0,255,0);
  fill(0,255,255);
  ellipse(22,77,33,33);

  stroke(0,0,255);
  fill(255,255,0);
  ellipse(55,11,22,22);
}
```

HANDLUNGEN ZUSAMMENFASSEN!

- Können wir einen Befehl **`kreis(x,y,d)`** bestimmen?
- Vielleicht könnte man diesem neben Koordinaten und Durchmesser auch Umriss und Füllfarbe übergeben?

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

void draw () {
  stroke(255,0,0);
  fill(255,0,255);
  kreis(33,33,55);

  stroke(0,255,0);
  fill(0,255,255);
  kreis(22,77,33);

  stroke(0,0,255);
  fill(255,255,0);
  kreis(55,11,22);
}
```

HANDLUNGEN ZUSAMMENFASSEN!

- Das würde die Möglichkeit eröffnen, sich wiederholende Strukturen verkürzt schreiben zu können.
- Aber wie können wir das erreichen?

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

void draw () {
 kreis(33,33,55,255,0,0,255,0,255);

 kreis(22,77,33,0,255,0,0,255,255);

 kreis(55,11,22,0,0,255,255,255,0);
}
```

FUNKTIONEN

- Außerhalb von **`setup`** und **`draw`** können wir sogenannte Funktionen definieren.
- Mit einen Namen **`kreis`** und in **`()`** festgelegten Variablen.
- Diese Variablen können nun im Rumpf verwendet werden.

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

void draw () {...}

void kreis (int x, int y, int d, int sR, int sG, int sB, int fR, int fG, int fB) {
  stroke(sR, sG, sB);
  fill(fR, fG, fB);
  ellipse(x,y,d,d);
}
```

ZAUBERWORT: VOID

- Was bedeutet jetzt eigentlich immer dieses **`void`**?
- Funktionen können an unterschiedlichen Orten aufgerufen werden.
- Auch z.B. nach Zuweisungsoperatoren.

```
maleEinenKreis(25,25,25);

int x = dasDoppelteVon(3);
```

ZAUBERWORT: VOID

- Also müssen Funktionen auch in der Lage sein Werte auszugeben, die dann weiterverarbeitet werden können!
- **`maleEinenKreis()`** soll keinen Wert ausgeben. Also schreiben wir **`void`** um den Computer mitzuteilen, dass diese Funktion nichts zurückgibt.

```
maleEinenKreis(25,25,25);

int x = dasDoppelteVon(3);

void maleEinenKreis(int x, int y, i
nt r) {
 ellipse(x,y,2*r);
}
```

ANDERE RÜCKGABETYPEN

- Wie sagen wir nun dem Computer, dass er bei **`dasDoppelteVon()`** einen Wert zurückgeben soll?
- Vor dem Namen der Funktion steht immer der returntype (**`int`**, **`char`**, ...)!
- Der letzte Befehl einer Funktion ist immer **`return ...;`** (außer bei **`void`**).

```
maleEinenKreis(25,25,25);

int x = dasDoppelteVon(3);

void maleEinenKreis(int x, int y, i
nt r) {
 ellipse(x,y,2*r);
}

int dasDoppelteVon (int n) {
    return n*2;
}
```

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);`

ANWENDUNG

ARRAY

```
// Arrays zum Speichern von Maus Koordinaten

void setup () {
  size(800,800);
  // initialisiere Arrays
}

void draw () {
  // vergangene Mauspositionen im Array verschieben

  // neue Position speichern

  // Linien zwischen Positionen zeichnen
}
```

STRING

```
// Arrays: Subjekt Prädikat Objekt

void setup () {
  size(800,800);
  // initialisiere Arrays
}

void draw () {
  // gebe zufällig erzeugte Sätze in der Konsole aus
}
```

FUNKTIONEN

```
void setup () {
  size(800,800);
}

void draw () {
  // rufe Zeichenfunktion auf
}

// deklariere Zeichenfunktion

// deklariere Funktion "erzeuge Y-Koordinate" abhängig von "X-Position"
// deklariere Funktion "erzeuge Größe" abhängig von "X-Position"
```

VERKNÜPFUNG

KEYBOARD VISUALS

- Wir entwickeln über die nächsten Sitzungen ein Programm, dass auf Tastendruck kleine Animationen abspielt.
- Zunächst entwerfen wir das Konzept für eine Taste und erweitern dann nach und nach das Programm.

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

void draw () {
  background(0);
  // reagieren auf Input
  // zeichnen der Animation
}
```

WAS IST EINE ANIMATION?

- Für uns besteht im wesentlichen eine Animation aus einer Abfolge von Bildern.
- Diese Abfolge ist abhängig von der Zeit, wie die Frames eines Videos.
- Der **`animationCounter`** repräsentiert das zu zeichnende Frame und zählt von 0 bis zu einer beliebig großen Zahl.

```
int animationCounter;

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

void draw () {
  background(0);
  // reagieren auf Input
  // zeichnen der Animation
}
```

REAGIEREN AUF DEN INPUT

- Zunächst dient uns die Maustaste als Input.
- Wenn diese gedrückt ist inkrementieren wir den counter.
- Ansonsten setzen wir auf 0 zurück.
- Wenn die Maustaste gedrückt ist, wird ebenfalls gezeichnet.

```
int animationCounter;

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

void draw () {
  background(0);
  // reagieren auf Input
  // zeichnen der Animation
  if (mousePressed) {
    animationCounter++;
    int r = animationCounter;
    ellipse(300,200,r,r);
  }
  else {
    animationCounter = 0;
  }
}
```

TRENNEN VON INPUT UND ANIMATION

- Wir trennen die Logik der Input-Überprüfung von den Zeichenbefehlen.
- Wir erhalten eine Struktur, die leicht vervielfacht und auf mehrere Tasten/Inputs angewandt werden kann.

```
int animationCounter;

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

void draw () {
  background(0);
  // reagieren auf Input
  if (mousePressed) {
    animationCounter++;
  }
  else {
    animationCounter = 0;
  }

  // zeichnen der Animation
  if (animationCounter > 0) {
    int r = animationCounter;
    ellipse(300,200,r,r);
  }
}
```

STRUKTURIEREN DURCH FUNKTIONEN

- Nun teilen wir die uns zuvor überlegte Struktur in die Funktionen **`handleInput`** und **`drawAnimation`** auf.
- Wir können auch auf Tastendrücke hören!
- Was ist der Nachteil von **`keyPressed`**?

```
int animationCounter;
void setup () {...}

void draw () {
  background(0);
  // reagieren auf Input
  handleInput();
  // zeichnen der Animation
  drawAnimation();
}

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

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

AUSBLICK

NÄCHSTE SITZUNG

- Klassen und Objekte

ÜBUNG

QUELLEN

- http://nos.twnsnd.co/image/117523960441
- http://nos.twnsnd.co/image/124161898747

PROCESSING

KLASSEN UND OBJEKTE

Created by Michael Kirsch & Beat Rossmy

INHALT

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 Rückgabetyp **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 Rückgabetyp! (Kein Rückgabewert → **`void`**)

```
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 = "Donaudampfschifffahrtsg
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 bewegenden 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 estellen 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`** makiert 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 Über-gabewerte 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, Funktions-name, 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 (intx ,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

// deklariere 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 genung 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`**, mun 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

PROCESSING

EINE ZUSAMMENFASSUNG

Created by Michael Kirsch & Beat Rossmy

INHALT

TYPEN UND OPERATOREN

DATENTYPEN

Der Computer kennt generell gewisse Arten von Daten. Diese primitiven Datentypen decken primär Zahlen und zeichen ab. Alles Darüber hinaus muss von uns festgelegt und dem Computer vermittelt werden.

DATENTYPEN

BOOLEAN

Wahrheitswert	`boolean b1 = false;` `boolean b2 = true;`
Wertebereich:	falsch oder wahr

DATENTYPEN

INTEGER

Ganzzahlige positive und negative Werte

```
int i1 = 2;
int i2 = -138;
```

Wertebereich: -2147483648 ... 2147483647

DATENTYPEN

FLOAT

Negative und positive Gleitkommazahlen

```
float f1 = 2.3902;
float f2 = -129.368;
```

Wertebereich: 10 Stellen

DATENTYPEN

CHARACTER

Einzelne Zeichen und Ziffern

```
char c1 = 'a';
char c2 = '7';
```

Merke: Intern werden Character von Zahlen repräsentiert!

```
char c2 = '7';
int i = c2; // -> i = 55
```

OPERATOREN

Operatoren erlauben es uns bestimmte Berechnungen, Vergleiche oder Veränderungen auf Daten durchzuführen.

OPERATOREN

ZUWEISUNGSOPERATOREN

Mit Hilfe von Zuweisungsoperatoren können wir Werte in Variablen Speichern oder diese verändern.

Einfache Zuweisung: **=**	`int x = 1; // -> x = 1` `String s = "Test";`
Addition und Zuweisung: **+=**	`x += 1; // -> x = 2`
Subtraktion und Zuweisung:**-=**	`x -= 1; // -> x = 1`

OPERATOREN

ARITHMETISCHE OPERATOREN

Einfache Mathematik ist möglich.

Addition: **+**	`int x = 1 + 1; // -> x = 2`
Subtraktion: **–**	`x = 10 - 1; // -> x = 9`
Multiplikation: *****	`x = 10 * 10; // -> x = 100`
Division: **/**	`x = 6 / 3; // -> x = 2`
Inkrementieren: **++** *Entspricht Addition mit 1*	`x++; // -> x = 3`
Dekrementieren: **––** *Entspricht Subtraktion mit 1*	`x--; // -> x = 2`

OPERATOREN

VERGLEICHSOPERATOREN

Das Ergebnis einer Vergleichsoperation ist immer ein boolscher Wert und kann somit als Aussagen verwendet werden.

Gleichheit: **==**	`a == b`
Ungleichheit: **!=**	`a != b`
Größer: **>**	`a > b`
Größer oder Gleich: **>=**	`a >= b`
Kleiner: **<**	`a < b`
Kleiner oder Gleich: **<=**	`a <= b`

OPERATOREN

BOOLSCHE OPERATOREN

Boolsche Operatoren bilden Aussagen, die wiederum boolsche Werte erzeugen.

Negation: **!**	`!true // -> false`

Und: **&&**	`true && true // -> true`
	`true && false // -> false`
	`false && true // -> false`
	`false && false // -> false`

Oder: **||**	`true || true // -> true`
	`true || false // -> true`
	`false || true // -> true`
	`false || false // -> false`

TYPKONVERSION

Typkonversion ist relevant um Daten z.B. in andere Werte-bereiche zu übertragen. Dabei kann es passieren, dass Informationen verloren gehen.

Downcast:

```
int i = (int)3.9; // -> i = 3
i = (int)4.1; // -> i = 4
```

Upcast:

```
float f = (float)3; // -> f = 3.0
f = (float)4; // -> i = 4.0
```

VARIABLEN

Variablen sind Speicher für Werte oder komplexere Datenstrukturen.

Deklarieren: `int xCoordinate;`
Typ Name;

Initialisieren: `xCoordinate = 1;`
Name Zuweisung;

STRUKTUREN

KONTROLLSTRUKTUREN

Durch Kontrollstrukturen wird es uns ermöglicht eine beliebige Anzahl von Befehlen unter bestimmten Bedingungen oder auf bestimmte Weise zu behandeln.

KONTROLLSTRUKTUREN

BEDINGUNGEN

Blöcke von Befehlen können abhängig von bestimmten Aussagen ausgeführt werden. Der zu **`else`** gehörige Rumpf wird nur ausgeführt wenn die Aussage der Bedingung **`false`** ist.

Kopf: *Schlagwort (Aussage)*	`if (a > b)`
Rumpf	`{ /* wenn a größer */}`
Schlagwort	`else`
Rumpf	`{ /* wenn a nicht größer */}`

KONTROLLSTRUKTUREN

SCHLEIFE

Schleifen ermöglichen es uns Blöcke von Befehlen abhängig von Bedingungen wiederholt auszuführen.

Kopf:
Schlagwort (Initialisierung; Limitierung; Zählen)

```
for (int i=0; i<10; i++)
```

Rumpf

```
{
  // hier stehen beliebig viele Befehle
}
```

FUNKTIONEN

Mehrere Befehle können als Block unter einem neuen Namen ausgeführt werden. Dabei ist es möglich eine Verallgemeinerung durch die Verwendung von Parametern zu erreichen.

Kopf:
Rückgabetyp Name (Parameter)

```
int dasDoppelteVon (int i)
```

Rumpf

```
{
  return 2*1;
}
```

FUNKTIONEN

Hat die Funktion einen Rückgabetyp, so ist der letzte Befehl im Rumpf immer **`return`** dies ermöglicht es einen Wert auszugeben. Dieser muss mit dem festgelegten Rückgabetyp übereinstimmen. Ansonsten ist der Rückgabetyp immer **`void`**.

Ohne Rückgabe	`void ... (...) {...}`
Mit Rückgabe	`int ... (...) {...; return ...;}`

KLASSEN UND OBJEKTE

KLASSEN UND OBJEKTE

Klassen ermöglichen es eigene Datenstrukturen und damit verbundene Funktionalität zu beschreiben. Instanzen dieser Struktur nennen sich Objekte.

Kopf:
Schlagwort Name

```
class Ball
```

Rumpf:
Felder, Konstruktor, Methoden

```
{
  // Felder
  int x,y,r;

  // Konstruktor
  public Ball (int x, int y) {
    this.x = x;
    this.y = y;
  }

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

KONSTRUKTOR

Der Konstruktor hat immer den selben Namen wie seine Klasse. Er kann von überall aus aufgerufen werden, dies beschreibt das Schlagwort **`public`**. **`this`** bezieht sich auf die Felder der durch den Konstruktor erzeugten Instanz.

Kopf:
Schlagwort KlassenName (Parameter)

```
public Ball (int x, int y)
```

Rumpf:

```
{
  this.x = x;
  this.y = y;
}
```

Konstruktoraufruf

```
Ball b = new Ball (100,100);
```

OBJEKTE

Wurde eine Instanz einer Klasse erzeugt so nennen wir diese eine Objekt. Dieses Objekt ist quasi ein Behälter für die in der Klasse beschriebenen Felder und kann sämtliche Funktionalität der Klasse ausführen. Dazu dient der **.** Operator.

Zugriff auf Felder

```
Ball b = new Ball (100,100);
println(b.x); // -> 100
b.x = 200;
println(b.x); // -> 200
```

Zugriff auf Methoden

```
b.plot();
```

STRING

Ein **`String`** ist eine Zeichenkette. Bei der Initialisierung werden die Zeichen der Kette zwischen **`" "`** aufgeführt.

Initialisieren	`String s = "Das ist ein String.";`
Zusammenfügen	`String s = "Das ist ";` `String t = s + "ein String.";`
Länge	`String s = "abc";` `println(s.length()); // -> 3`
Index des ersten Zeichens	`String s = "abc";` `println(s.indexOf('b')); // -> 1`

ARRAYS

Arrays ermöglichen es uns Daten eines Typs in einem Speicher abzulegen. Arrays sind Objekte und werden deshalb mit dem **`new`** Operator erzeugt. Die erste Stelle eines Arrays hat den Index 0.

Deklarieren: *Datentyp [] Name;*	`int[] intArray;`
Initialisieren: *ohne konkrete Werte*	`intArray = new int[7];`
Initialisieren: *mit konkreten Werten*	`intArray = new int[] {1,6,3,8,4,8,0};`
Zugriff	`println(intArray[3]); // -> 8` `intArray[3] = 100; // [3] -> 100`
Anzahl der Speicherplätze	`println(intArray.length); // -> 7`

GÜLTIGKEIT UND KONVENTIONEN

GÜLTIGKEITSBEREICHE

Wird eine Variable in einem Rumpf/Block deklariert, so ist sie in diesem und allen untergeordneten gültig. Variablen die im Kopf einer Funktion, Bedingung oder Schleife deklariert werden, sind im dazugehörigen Rumpf gültig.

Die Variable **`y`** ist nur gültig im Block ihrer Deklaration.

```
int x;

void setup () {
  // ...
  x = 0;
  int y = 0;
}

void draw () {
  ellipse(x,y,50,50); // -> Fehler!
  // y ist NUR in setup gültig!
}
```

BENENNUNG

Durch die einheitliche Benennung von Variablen und Klassen kann die Verständlichkeit des Codes gesteigert werden. Generell gilt, dass bei Namen aus zusammengesetzten Worten jedes weitere Wort mit einem Großbuchstaben an das letzte hinzugefügt wird.

Variablen *erster Buchstabe klein*	`int xCoordinate;`
Funktionen *erster Buchstabe klein*	`int doubleValue (int n) {...}`
Klassen *erster Buchstabe groß*	`class Animation {...}`

EINRÜCKUNG

Der Inhalt jedes Blocks wird auf eine neue Ebene mit dem Tabulator eingerückt. Die den Block schließende geschweifte Klammer steht also wieder auf der ausgehenden Ebene.

Eine Block

```
if () {
  // eine Ebene eingerückt
}
```

Block in Block

```
if (...) {
  // eine Ebene eingerückt
  for (...) {
    // eine weitere Ebene eingerückt
  }
}
```

QUELLEN

PROCESSING

ELTERN UND KINDER

Created by Michael Kirsch & Beat Rossmy

INHALT

RÜCKBLICK

PROCESSING BASICS

keyPressed: Variable

```
if (keyPressed) {...}
```

keyPressed: Funktion

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

void keyPressed () {
  // beliebiger code
  println(key);
}
```

keyReleased: Funktion

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

void keyReleased () {
  // beliebiger code
  println(key);
}
```

KLASSEN

Signalwort + Name + Rumpf-Anfang	`class Kreis {`
Felder	`int x,y,d;`

Konstruktor: Name + Übergabewerte	`public Kreis (intx ,int y, int d) {`
	`  ...`
	`}`

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

Klassenrumpf-Ende	`}`

OBJEKTE

Objekt deklarieren	`Kreis k1;`
Objekt initialisieren	`k1 = new Kreis(122,321,20);`
Objekt-Methoden Aufruf	`k1.plot();`
Objekt-Feld Zugriff	`println(k1.x);` `k1.x++;`

THEORIE

ES GIBT JA NICHT NUR EINE ART BALL.

- Anstatt einer Klasse **`Ball`** können wir auch mehrere bestimmen, wie z.B. Fußball, Tennisball, Golfball, ...
- Müssen wir nun für jede Art Ball eine ganz neue Klasse schreiben? Mit all den selben Feldern und Methoden von **`Ball`**?

```
Ball b;
Fussball f;
Tennisball t;
Golfball g;

class Ball {
  int x,y,d;

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

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

...
```

WO LIEGEN GEMEINSAMKEITEN, WORIN UNTERSCHIEDE?

- Unterschiede: Größe, Aussehen, ...
- Gemeinsamkeiten: Koordinaten, Methoden, ...
- Das sind doch alles Bälle! Warum Teilen sie sich nicht einfach all die Gemeinsamkeiten?

```
Ball b;
Fussball f;
Tennisball t;
Golfball g;
```

EXTENDS

- **`extends`** hilft uns dabei, dass unterschiedliche Klassen ihr Gemeinsamkeiten teilen können.
- Diese Beziehung unterschiedlicher Klassen zueinander nennt sich Vererbung.

```
class Ball {
  ...
}

class Fussball extends Ball {
  ...
}

class Tennisball extends Ball {
  ...
}

class Golfball extends Ball {
  ...
}
```

EXTENDS

- Kindklassen erben alle Felder und Methoden der Elternklasse.
- Eine Kindklasse muss also mindestens einen eigenen Konstruktor enthalten, kann aber auch eigene Felder und Methoden definieren.

```
class Ball {
  ...
}

class Fussball extends Ball {
  Color c;
  float bar;

  public Fussball (...) {
    ...
  }

  void aufpumpen () {}
}
```

EXTENDS

- Wie greife ich nun auf vererbte Felder zu?
- **`super`** erlaubt uns zwei Dinge:
  1. Aufruf des Eltern-klassen-Konstruktors
  2. Zugriff auf Felder und Methoden der Eltern-klasse

```
class Ball {
  ...
}

class Fussball extends Ball {
  Color c;
  float bar;

  public Fussball (int x, int y) {
    super(x,y,21);
    bar = 0.5;
    c = color(255);
  }

  void aufpumpen () {
    if (super.d<23) super.d++;
  }
}
```

EXTENDS

- In der Elternkasse bereits definierte Methoden können abgeändert/ überschrieben werden.
- Dazu wird einfach die selbe Methode mit neuem Rumpf definiert. Nun wird statt der in der Eltern-klasse enthaltenen Methode die der Kind-klasse verwendet.

```
class Ball {...}

class Fussball extends Ball {
  Color c;
  float bar;

  public Fussball (int x, int y) {
    super(x,y,21);
    bar = 0.5;
    c = color(255);
  }

  void aufpumpen () {
    if (super.d<23) super.d++;
  }

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

EXTENDS

- Hilfreich ist das ganze, da wir uns nun auf diese Gemeinsamkeiten verlassen können!
- Wir wissen jeder **`Ball`** besitzt eine Methode **`plot()`** egal ob geerbt oder überschrieben.

```
Ball[] b;

void setup () {
  size(600,600);
  b = new Ball[]{
    new Tennisball(100,100),
    new Fussball(400,400),
    new Ball(300,300,30)
  };
}

void draw () {
  background(0);
  for (int n=0; n<b.length; n++) {
    b[n].plot();
  }
}
```

AUCH UNTERSCHIEDLICHE DINGE KÖNNEN DIE SELBE FUNKTIONALITÄT BESITZEN.

- Wäre es nicht hilfreich, wenn alle grafischen Elemente im Sketch eine Methode **`plot()`** hätten?
- So könnten wir diese einfach mit nur einem Befehl zeichnen!

```
Ball b;
Pferd p;
Haus h;

void setup () {
  ...
}

void draw () {
  background(0);
  b.plot();
  p.plot();
  h.plot();
}
```

INTERFACES

- Ein **`interface`** erlaubt es uns eine Schnittstelle zu definieren, die sich unterschiedliche Klassen teilen können.
- **`interface`** steht vor dem Namen der Schnittstelle. Diese Namen enden oft auf "-ble", da sie gewisse Fähigkeiten beschreiben (Ability -> able).

```
interface Plotable {
  void plot();
}
```

IMPLEMENTS

- Alle im **`interface`** definierten Methoden, muss die implementierende Klasse enthalten.
- Hinter dem Klassennamen leitet **`implements`** eine Liste der zu implementierenden Interfaces ein.

```
interface Plotable {
  void plot();
}

class Ball implements Plotable {
  ...

  void plot () {
    ...
  }
}

class Haus implements Plotable, ...
 {
  ...

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

IMPLEMENTS

- So können auch die unterschiedlichsten Klassen in Arrays zusammengefasst weren, solange sie sich nur das Interface **`Plotable`** teilen.
- Dies kann hilfreich sein um gemeinsame Schritte vereinheitlicht auszuführen.

```
interface Plotable {...}

class Ball implements Plotable {...
}

class Haus implements Plotable {...
}

Plotable[] p;

void setup() {
  size(600,600);
  p = new Plotable[]{new Ball(...),
 new Haus(...)};
}

void draw() {
  for (int n=0; n<p.length; n++)
    p[n].plot();
}
```

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 {...}
```

ANWENDUNG

EXTENDS

```
void setup () {
  size(800,800);
}

void draw () {

}
```

IMPLEMENTS

```
// definiere Interface Randomizeable -> zufällige Werte für Objektvariablen

// definiere Klassen die Randomizeable implementieren

void setup () {
  size(800,800);
  // initialisiere Objekte
}

void draw () {
  // RANDOMIZE!
}
```

VERKNÜPFUNG

KEYBOARD VISUALIZER

- Rechts sehen wir den bisherigen Stand unseres Projektes.
- Nun wollen wir die Funk-tionalität des Programms durch Vererbung der Klasse an ihre Kindklassen erweitern.
- Wir versuchen das Konzept "Animation" in seine Bausteine zu zerbrechen.

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

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

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

  void plot () {
    if (animationCounter > 0) {...}
  }
}
```

INTERFACES

- Bestandteile, die nicht zwingend auf Animationen zu beschrenken sind, können in Interfaces ausgelagert werden.
- Auch andere Klassen könnten durch **`plot`** gezeichnet werden.
- Auch andere Klassen könnten auf Tasteninput reagieren.

```
interface Plotable {
  void plot();
}

interface Triggerable {
  void triggerOn(char c);
  void triggerOff(char c);
}

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

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

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

INTERFACES IMPLEMENTIEREN

- **`triggerOn()`** soll soäter aufgerufen werden wenn die Taste gedrückt wurde.
- **`triggerOff()`** wird aufgerufen wenn die Taste losgelassen wurde.
- Ist der **`animationCounter`** 0 so ist die Animation nicht aktiv.

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

  public Animation (char c) {...}

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

  void triggerOff (char c) {
    if (triggerKey == c) {
      animationCounter = 0;
    }
  }

  void plot () {}
}
```

KEYPRESSED & KEYRELEASED

- Wo werden **`triggerOn()`** und **`triggerOff()`** aufgerufen?
- Die Funktion **`keyPressed`** wird aufgerufen, wenn eine Taste gedrückt wurde. Wir behandeln den Tastendruck.
- Genauso gibt es die Funktion **`keyReleased()`**. Wir behandeln das Ende des Tastendrucks.

```
Animation a;

void setup () {...}

void draw () {
  a.plot();
}

void keyPressed () {
  a.triggerOn(key);
}

void keyReleased () {
  a.triggerOff(key)
}
```

KINDKLASSE

- Unsere Klasse erweitert die Klasse **`Animation`** und ist somit ein Kind davon.
- Ein Konstruktor muss implementiert werden, der den super-Konstruktor aufruft.
- Da **`triggerOn`** nur einmal überprüft wird, wird der **`animationCounter`** nun in **`plot`** inkrementiert.

```
class GrowingBall extends Animation
 {

  public GrowingBall (char c) {
    super(c);
  }

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

AUSBLICK

NÄCHSTE SITZUNG

ÜBUNG

QUELLEN

PROCESSING

BILDER UND SOUND

Created by Michael Kirsch & Beat Rossmy

INHALT

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

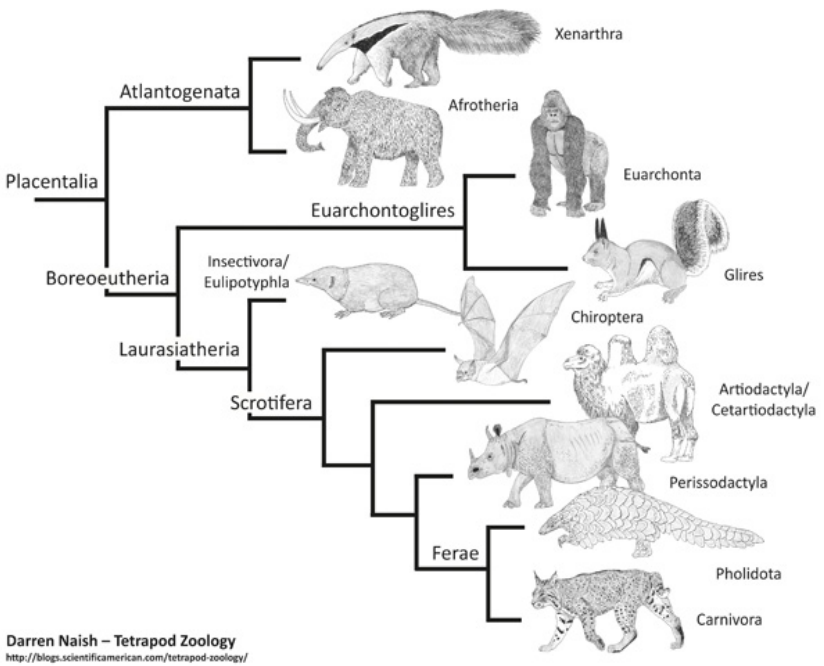
Tabs

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!



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.twnsnd.co/image/127563852947

INTERFACES

- In der Beschreibung des Interfaces werden alle zu implementierenden Methoden aufgelistet, mit Rückgabetyp, 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

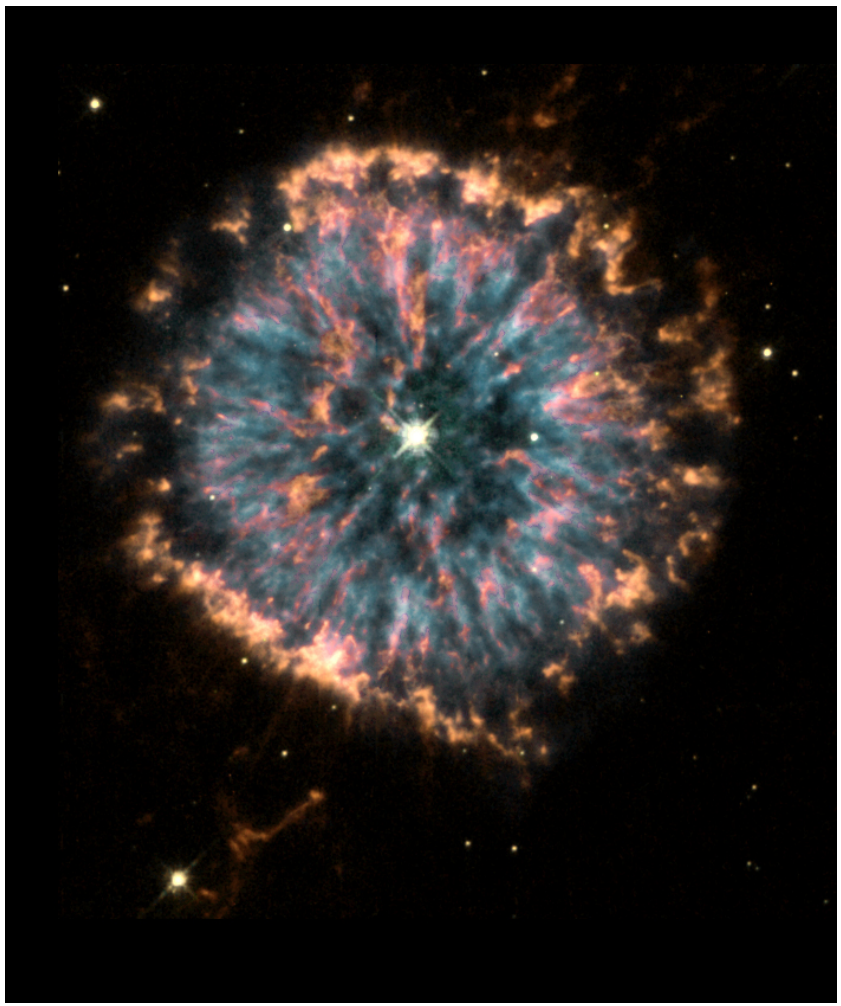
- Interfaces und Elternklassen können als Typen bei Arrays verwendet werden.
- So können z.B. Klassen der unterschiedlichsten Art, aber mit sich deckender Funktionalität in einem Array gespeichert werden.

```
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

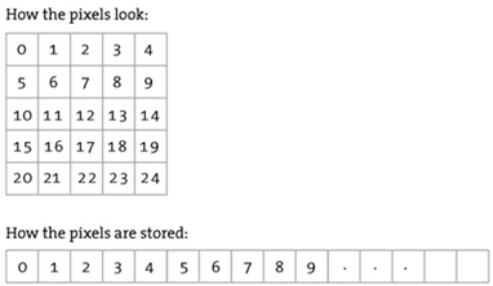
- Ein wichtiger Bestandteil von grafischen Programmen ist die Darstellung und Manipulation von Bildern.
- Kommerzielle Lösungen bieten eine breite Palette an Bearbeitungstools und vorgefertigten Filtern/Effekten.



http://nos.twnsnd.co/image/123557322230

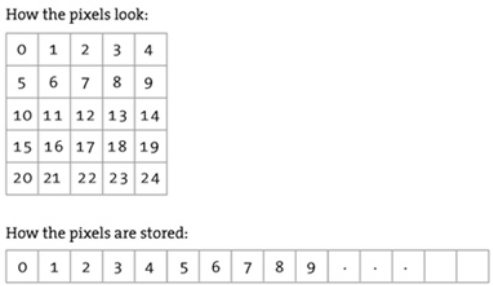
BILDER

- 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.



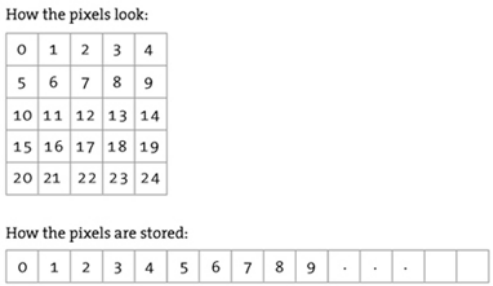
BILDER

- 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:
  Breite * Höhe = Anzahl aller Pixel



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.



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 Paramter 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`** erezugen.
- 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://nos.twnsnd.co/image/127563852947
- http://nos.twnsnd.co/image/123557322230
-
-
-

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Einführung in die Programmierung für Nebenfach Medieninformatik

Beat Rossmy, Michael Kirsch

Direct Feedback - Socrative

- Eure Mitarbeit ist uns wichtig!

- Installiert Euch dazu die kostenlose App „Socrative Student“ auf Eurem Smartphone oder nutzt das Webinterface unter http://b.socrative.com/login/student/

- Anonymer Login über den Raumnamen:

MSMJ0KRQ

Frage 1

Wie zufrieden seid Ihr mit dieser Vorlesung und den Übungen?

STUDENT

MSMJ0KRQ

A:

B:

C:

D:

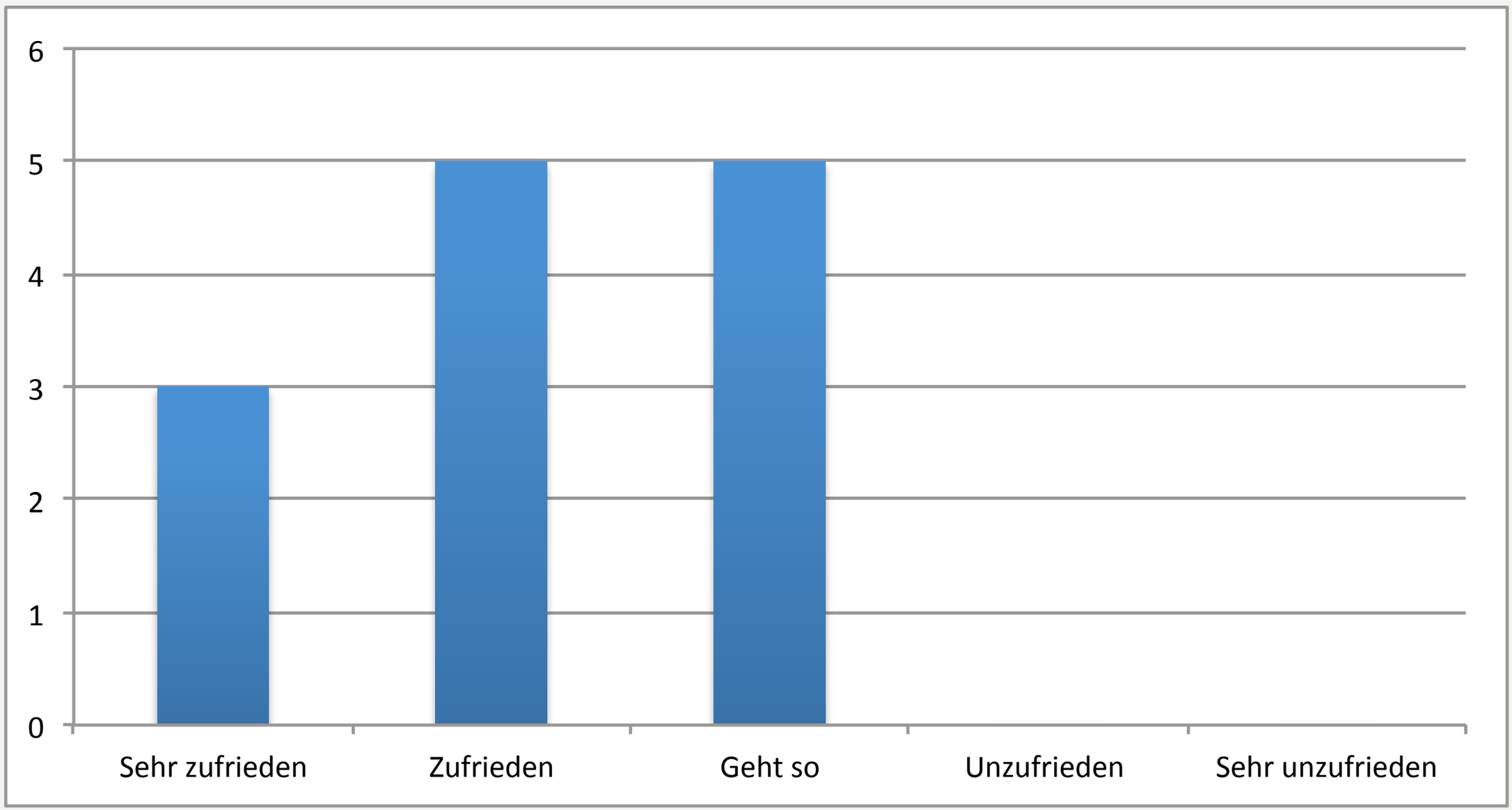
Direct Feedback - Socrative

Klausurtermin am 02.03.2017

Uhrzeit: tbd

Raum: tbd

Frage 1: Wie zufrieden seid Ihr mit dieser Vorlesung und den Übungen? (Ergebnisse aus dem letzten Semester)



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Einführung in Java

Beat Rossmy, Michael Kirsch

Inhalt

Einleitung

Übersicht -
Wie geht es weiter?

Einführung in die Programmierung

Processing

- Typen und Operatoren
- Kontroll-Strukturen
- Klassen und Objekte
- Gültigkeit und Konventionen
- Methoden
- Arrays
- Konstruktoren
- Eingaben-verarbeitung
- Animationen
- …

Java

- Grundlagen aus Processing
- …

Java & Processing

Processing

Java

- Java wurde 1995 von Sun Microsystems entwickelt und 2010 von Oracle gekauft und seitdem unter dem Namen „Oracle Java“ geführt
- Java ist eine Programmiersprache UND eine Technologie
- Java war bei der Entwicklung von Processing das Vorbild
  - Jedoch sollte Processing leichter zu verwenden sein
  - Zielgruppe von Processing sind „Visual Designers“
  - Zielgruppe von Java sind Softwareentwickler

Warum Java?

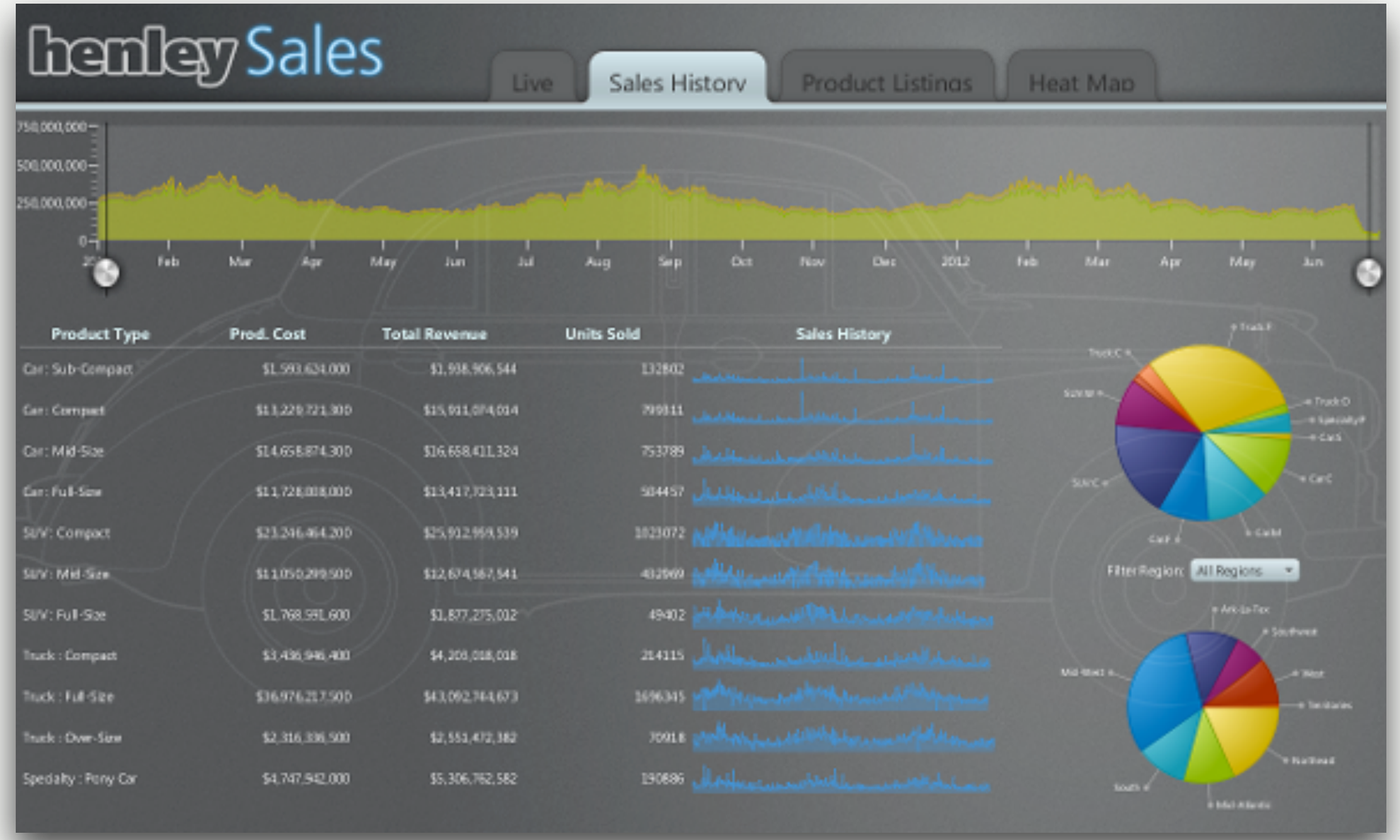
- Warum verwenden wir nicht weiterhin Processing?
  - Processing ist mächtig, aber dennoch begrenzt
  - Komplexere Anwendung bedürfen einer komplexeren Sprache und einer mächtigen Entwicklungsumgebung, die uns unterstützt
- Java bietet viele vorgefertigte Komponenten und erleichtert uns die Entwicklung von komplexeren Anwendung (Bspw. ein Spiel)

Java - Beispiele

Interactive Audio Player

http://cache.fxexperience.com/wp-content/uploads/2012/01/FXExperiencePlayer-BG.png

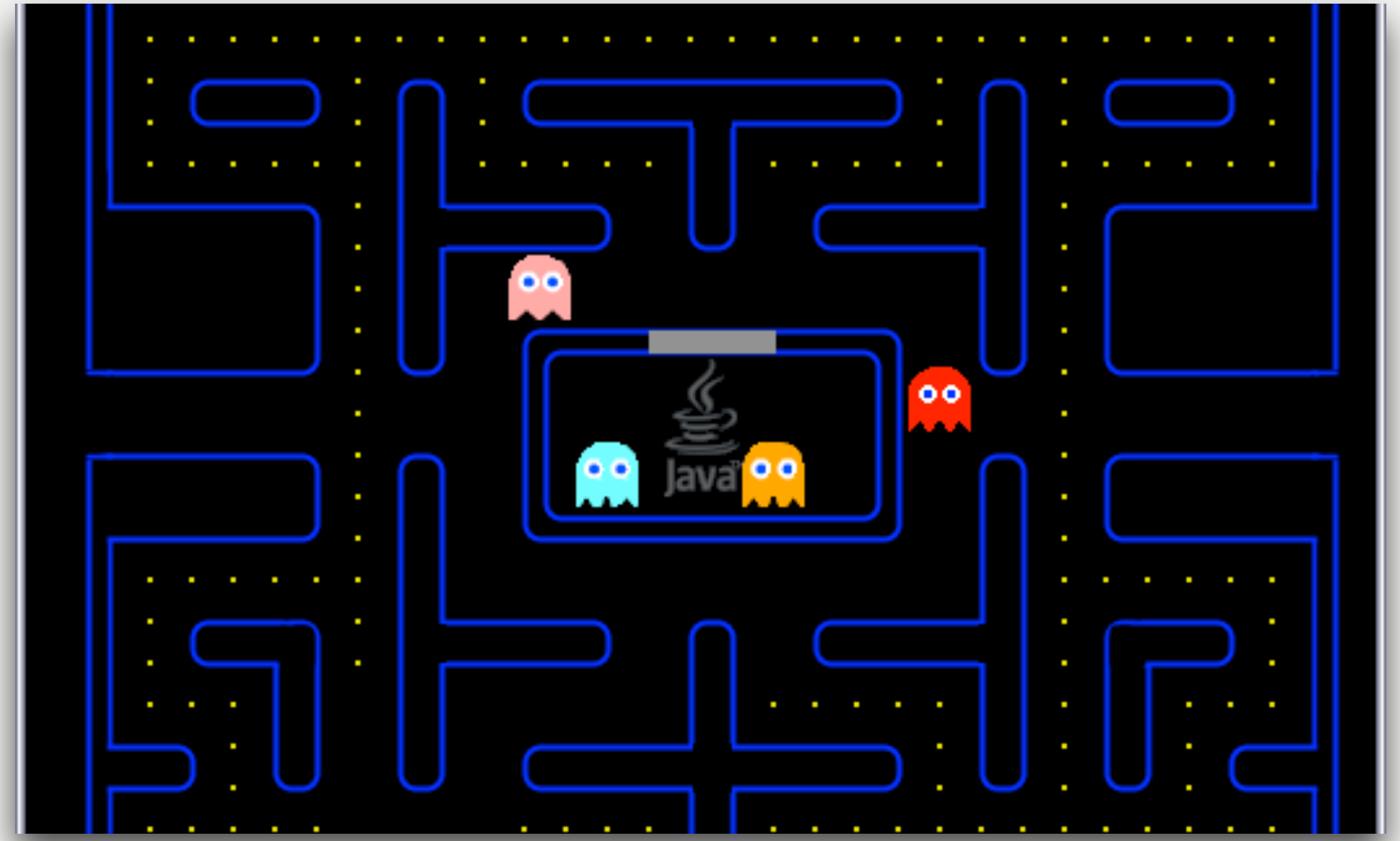
Java - Beispiele



Fancy Charts

https://docs.oracle.com/javafx/2/best_practices/jfxpub-best_practices.html

Fancy Games - Brick Breaker



Fancy Games - Pacman

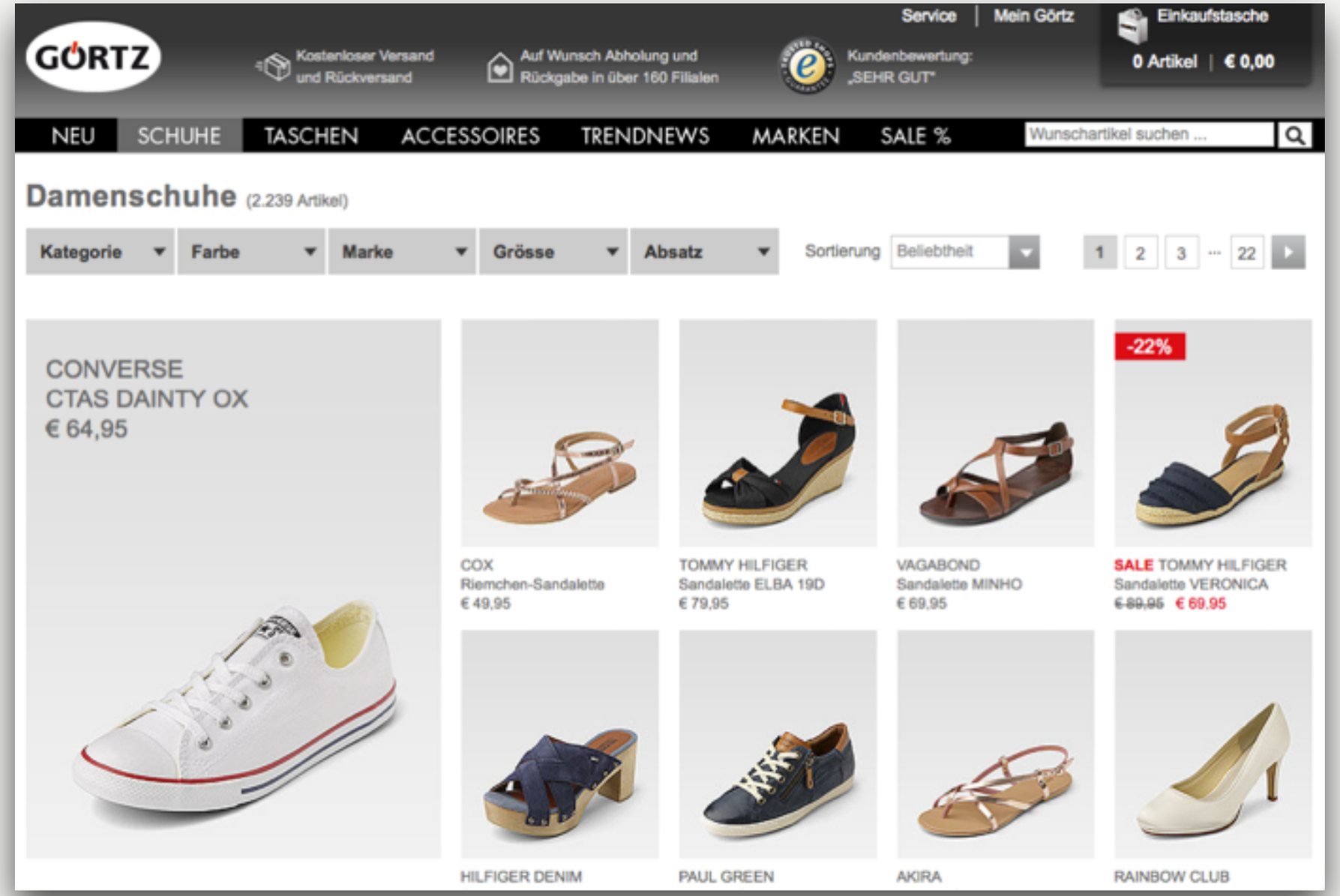
http://www.javafxgame.com/screenshoot.png

JavaFX - Beispiele

Benutzerdialoge

docs.oracle.com

Java - Beispiele



Industrial Webservices

www.goertz.de/

Schnittstellen & Kommunikation

„Ich bin aber kein Softwareentwickler…“

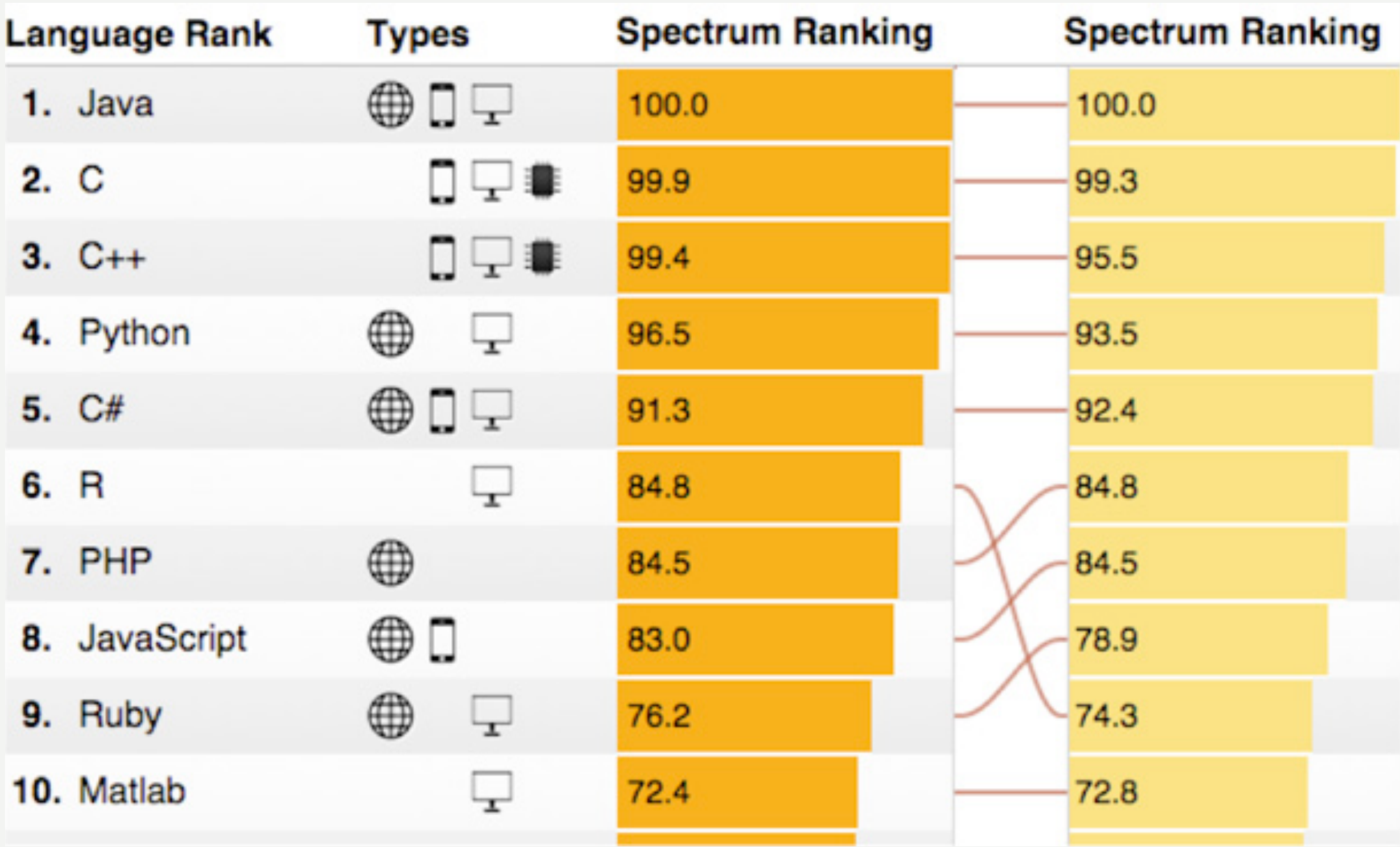
„Du wirst aber mit welchen zusammen arbeiten (müssen)!“

The 2015 Top Ten Programming Languages

Verbreitung von Programmiersprachen

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

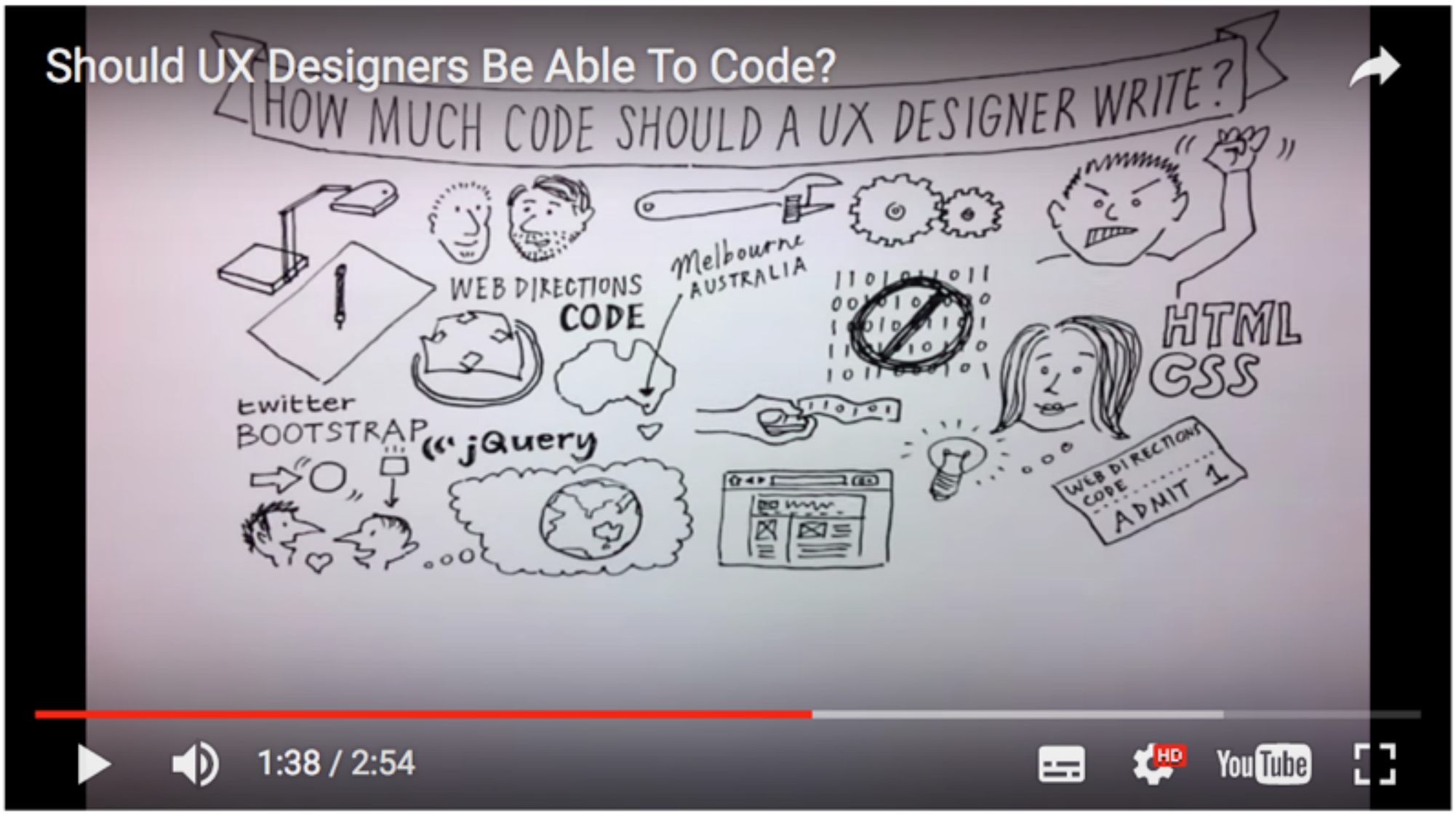
The 2015 Top Ten Programming Languages



Vergleich 2015 (links) mit 2014 (rechts)

http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages

News - 7 Gründe warum Coding-Skills den besseren Designer machen



https://entwickler.de/online/development/7-gruende-coding-skills-designer-297235.html

Theorie

Java - Einführung

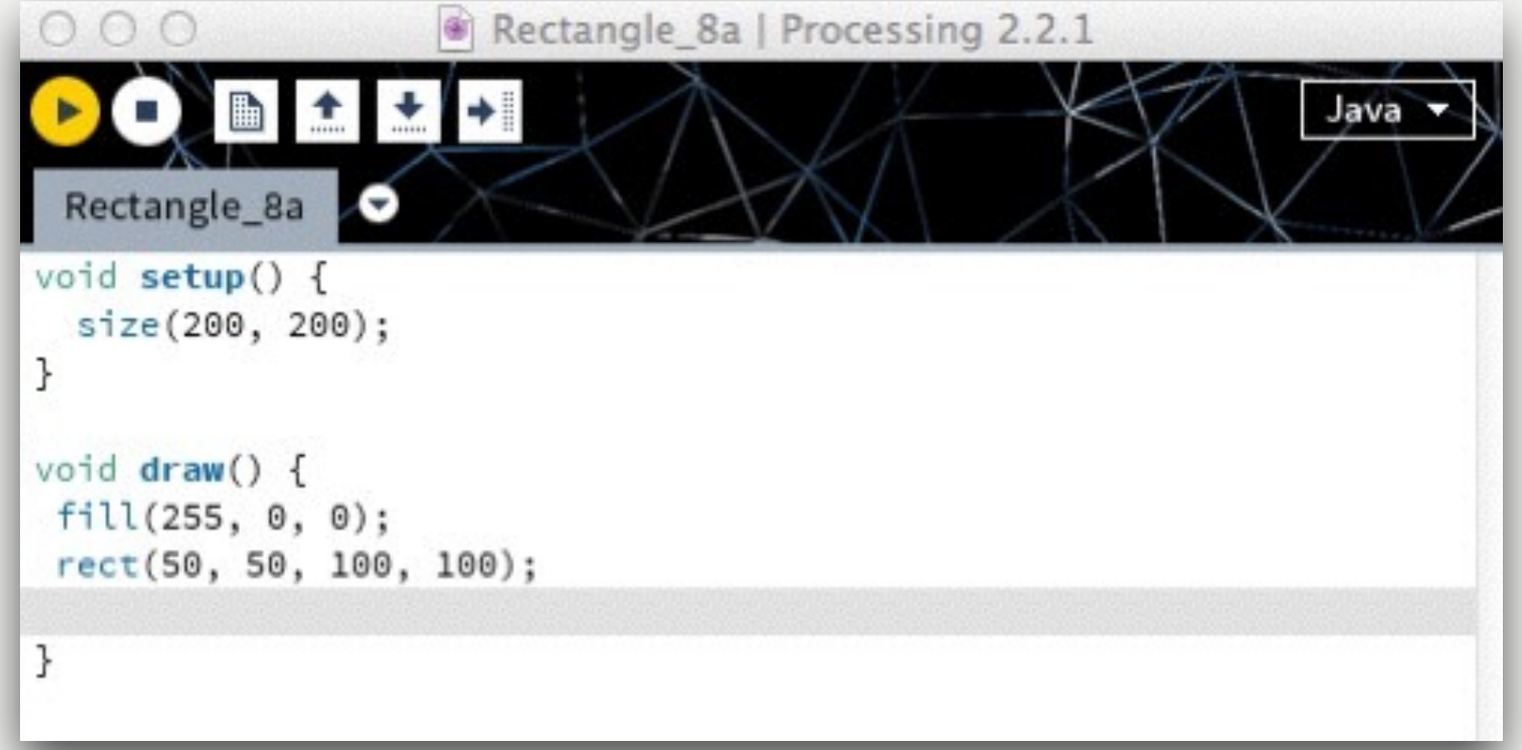
- Wie sieht Java aus?

```
public class HelloWorld {

    public static void main(String[] args) {

        for (int i = 0; i < 10; i++) {

            System.out.println(" Java is so cool! ");

        }
    }

}
```

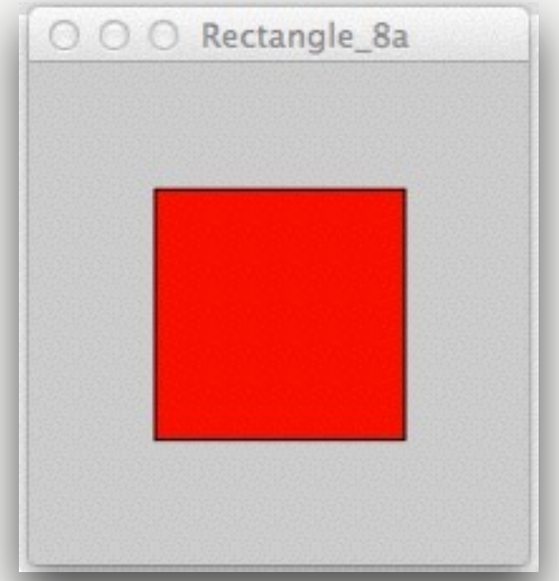
Hands on…

- Da Processing Java als Vorbild dient, ist es relativ leicht möglich Euer bisheriges Wissen von Processing auf Java zu übertragen…

- In Processing wird ein rot ausgefülltes Quadrat wie folgt erstellt:



Compile & Run



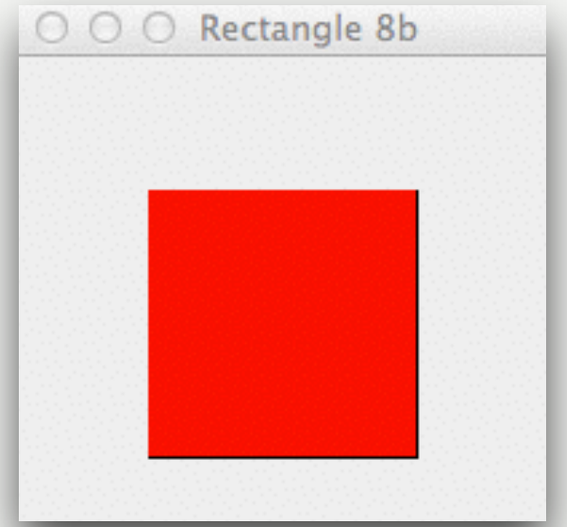
Hands on…

- In Java wird das gleiche Ergebnis mit etwas mehr Aufwand wie folgt erreicht:

Rectangle.java

```
1  import java.awt.Color;
2  import java.awt.Graphics;
3
4  import javax.swing.JComponent;
5  import javax.swing.JFrame;
6
7  class MyCanvas extends JComponent {
8
9      public void paint(Graphics g) {
10         g.drawRect(50, 50, 100, 100);
11         g.setColor(Color.RED);
12         g.fillRect(50, 50, 100, 100);
13
14     }
15 }
16
17 public class Rectangle {
18     public static void main(String[] a) {
19         JFrame window = new JFrame();
20         window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21         window.setSize(200, 200);
22         window.getContentPane().add(new MyCanvas());
23         window.setTitle("Rectangle 8b");
24         window.setVisible(true);
25
26     }
27 }
```

Compile & Run



Die Vorteile von Java

- Java ist aufwendiger als Processing, aber dafür gewinnen wir deutlich mehr Flexibilität und mehr Möglichkeiten

- Java ist plattformunabhängig, ebenso wie Processing

- Java bildet die Grundlage für sehr viele moderne Programmiersprachen (JavaScript, Processing usw.)

- Java ist *die* Programmiersprache im akademischen Umfeld und sehr verbreitet in der Industrie. Demnach: Viele (gute) Lehrbücher, Anleitungen und Tutorials im Internet

- Sehr guter Toolsupport (Hilfssoftware) für die Entwicklung von Java

- Große Anzahl an modernen Bibliotheken und Erweiterungen

Wie entwickeln wir in Java?

- Processing enthält bereits eine Entwicklungsumgebung (Integrated Development Environment, IDE) bei der Installation

- Für Java gibt es viele IDEs. Die verbreitetsten sind
  - Eclipse 
  - NetBeans
  - IntelliJ 

- Wir verwenden IntelliJ (IntelliJ 2016 (Community Edition))

Anwendung

IntelliJ Installation

- Anleitungen zur Installation von IntelliJ / Java findet Ihr viele im Internet. Eine (unverbindliche) Auswahl gibt es hier:

  - https://www.jetbrains.com/help/idea/2016.2/installing-and-launching.html
    IntelliJ Installation für alle Betriebssysteme (englisch)

  - https://www.youtube.com/watch?v=M0Y0T-s_mbQ
    IntelliJ Installation und „HelloWorld“ für Windows

Beispiele in IntelliJ

- Alle in der Vorlesung programmierten Beispiele werden Euch nach der Vorlesung zur Verfügung gestellt

- Versucht aufzupassen und/oder parallel selbst zu programmieren

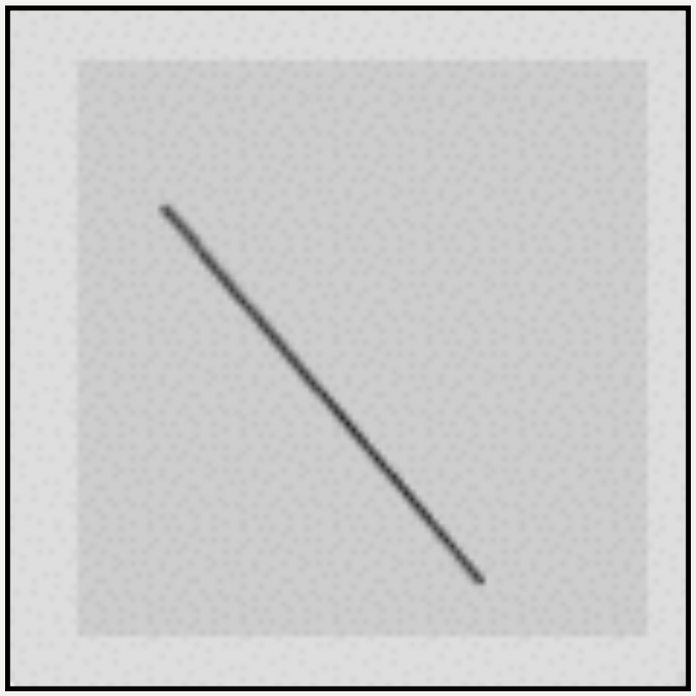
- Kommentare und Erklärungen sind im Quellcode enthalten

Variablen

- In Processing wurden Variablen wie folgt definiert:

- **int** begin_x = 15; //Integer-Variable
  int begin_y = 25;
  int end_x = 70;
  int end_y = 90;

- Ausgabe: *line(begin_x, begin_y, end_x, end_y);*

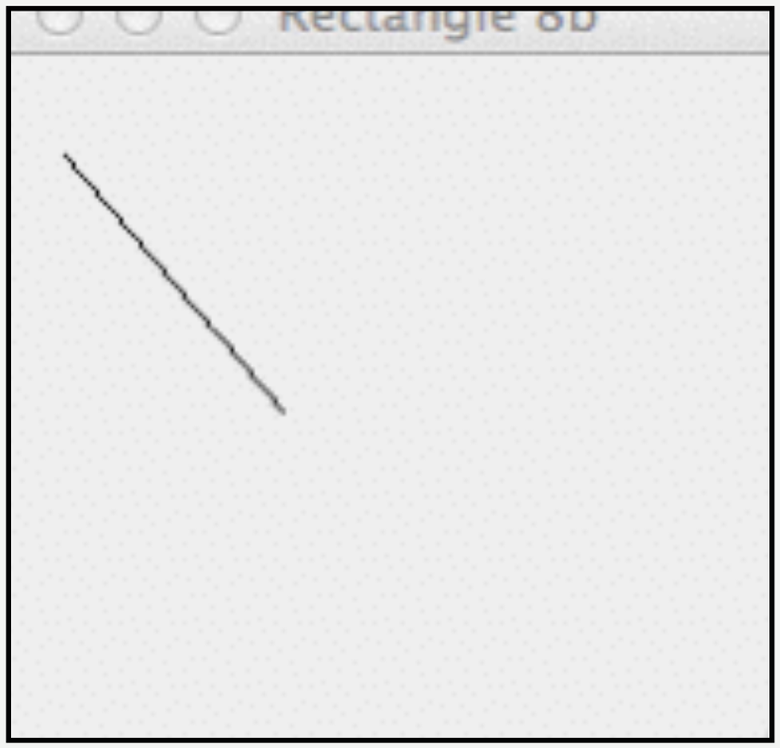


Variablen

- In Java sieht es ganz ähnlich aus:

- **int** begin_x = 15; //Integer-Variable
  int begin_y = 25;
  int end_x = 70;
  int end_y = 90;

- Ausgabe: *Graphics.drawLine(20, 100, 120, 100);*



Datentypen in Java

- Ganze Zahlen

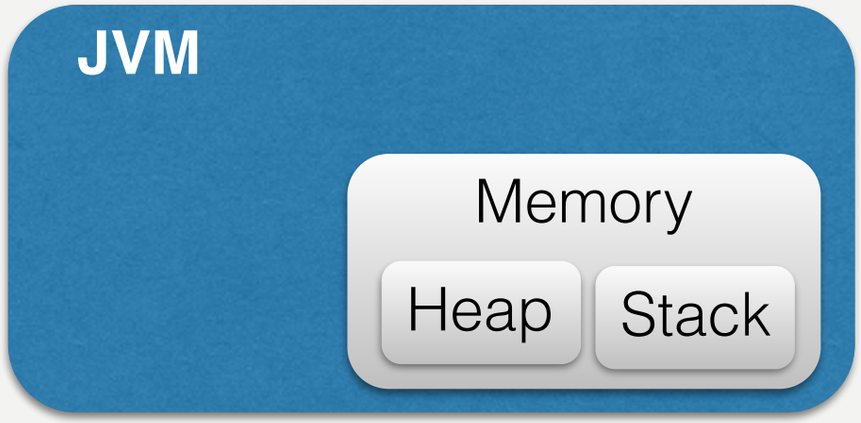
Name	Größe im Speicher	Zahlenbereich
byte	1 Byte	-128 bis 127 (2^8)
short	2 Byte	-32768 bis 32767 (2^16)
int	4 Byte	2.147.483.648 - 2.147.483.648 (2^32)
long	8 Byte	… 2^64

- Gleitkommazahlen

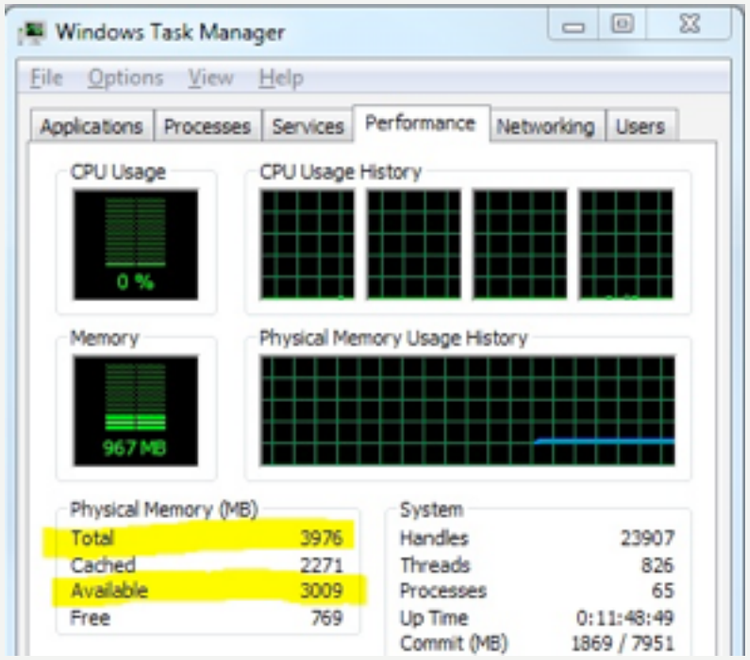
Name	Größe im Speicher	Zahlenbereich
float	4 Byte	Genauigkeit: 7-8 Dezimalstellen
double	8 Byte	Genauigkeit: 15-16 Dezimalstellen

Was bedeutet „Größe im Speicher“?

- Java verwaltet seine Variablen in zwei Speicherbereichen
  - **Stack** (Einfache Datentypen wie int, float)
  - **Heap** (Komplexe Datentypen wie Objekte)



bspw. „Java.exe“



Windows Speichermanagement

Datentypen in Java

- Der besondere Datentyp String (Zeichenketten)

- Strings sind Zeichenketten, die aus Buchstaben, Zahlen und Sonderzeichen bestehen können

- Darstellung durch Anführungszeichen

- Beispiel: String myString = “Java is so cool!“;
  Ausgabe: Java is so cool!

- Besonderheit: Zeilenumbruch mit „\n“
  String myString = “Java is \n so cool!“;
  Ausgabe: Java is
  so cool!

Variablenzuweisung / Deklarationen

- **Zuweisungen und Operationen** (ähnlich zu Processing)

```
1  public class General {
2
3      public static void main(String[] args) {
4          double d = 3.14; // Achtung! Komma ist der Punkt
5          int zahl_1 = 30;
6          int zahl_2 = 50;
7          int ergebnis = zahl_1 + zahl_2;
8          System.out.println("Das Ergebnis lautet: " + ergebnis);
9
10         String myString = "Hi there!";
11         System.out.println(myString);
12
13     }
14
15 }
16
```

Kontrollstrukturen

- **If / else**

```
1  public class Kontrollstrukturen_If {
2
3      public static void main(String[] args) {
4          int age = 16;
5          if (age < 18) { //Wenn Alter unter 18 Jahren
6              System.out.println("Ey! Du kommst hier net rein!");
7          } else { //Wenn Alter über 18 Jahren
8              System.out.println("Willkommen! :-)");
9          }
10
11     }
12
13 }
```

Kontrollstrukturen

- ***While*** **und** ***For*****-Schleifen**

```
1  public class Loop {
2      public static void main(String[] args) {
3          boolean isInteresting = true;
4          while (isInteresting) {// solange es interessant ist
5              System.out.println("That's so interesting!");
6          }
7          for (int i = 0; i < 10; i++) { // zehn mal durchlaufen
8              System.out.println("Aktueller Schleifendurchlauf:" + i);
9          }
10
11     }
12 }
```

Verknüpfung

Direct Feedback - Socrative

Quiztime :-)

MSMJ0KRQ

A:

B:

C:

D:

Quiztime - Raumnummer

MSMJ0KRQ

Quiz: Was ist IntelliJ und wofür verwenden wir es?

A: IntelliJ ist auch eine Programmiersprache und wir verwenden es nicht

B: IntelliJ ist eine Entwicklungsumgebung, mit der wir Java programmieren

C: So etwas gibt es nicht!

D: IntelliJ ist die Laufzeitumgebung, ohne die Java nicht funktionieren würde

Quiztime - Raumnummer

MSMJ0KRQ

Quiz: Was ist IntelliJ und wofür verwenden wir es?

A: IntelliJ ist auch eine Programmiersprache und wir verwenden es nicht

B: IntelliJ ist eine Entwicklungsumgebung, mit der wir Java programmieren

C: So etwas gibt es nicht!

D: IntelliJ ist die Laufzeitumgebung, ohne die Java nicht funktionieren würde

Quiztime - Raumnummer

MSMJ0KRQ

Quiz: Welche der folgenden Aussagen über Java und Processing sind richtig?

A: Java ist eine Programmiersprache und Processing nicht

B: Beides sind Programmiersprachen, jedoch ist Java komplexer

C: Java und Processing sind das gleiche

D: Processing ist eine Verbesserung/Weiterentwicklung von Java

Quiz: Welche der folgenden Aussagen über Java und Processing sind richtig?

A: Java ist eine Programmiersprache und Processing nicht

B: Beides sind Programmiersprachen, jedoch ist Java komplexer

C: Java und Processing sind das gleiche

D: Processing ist eine Verbesserung/Weiterentwicklung von Java

Quiz: Wie oft wird die folgende Schleife durchlaufen?

```
for (int i = 0; i <= 15; i++) { // ??? mal durchlaufen
    System.out.println("Aktueller Schleifendurchlauf:" + i);
}
```

A: Kein mal

B: 15 mal

C: 16 mal

D: 17 mal

Quiz: Wie oft wird die folgende Schleife durchlaufen?

```
for (int i = 0; i <= 15; i++) { // ??? mal durchlaufen
    System.out.println("Aktueller Schleifendurchlauf:" + i);
}
```

A: Kein mal

B: 15 mal

C: 16 mal

D: 17 mal

Ausblick

Ausblick

- Die kommenden Übungen dienen als “Beratungsstunde”
- Besucht die Übungen und bearbeitet die Übungsaufgaben dort
- Der Verzug zwischen Vorlesung und Übung wird damit reduziert

Nächstes Übungsblatt

- Das nächste Übungsblatt enthält Aufgaben zum Einstieg in Java
- Löst diese Aufgaben im Team

Euer Projekt

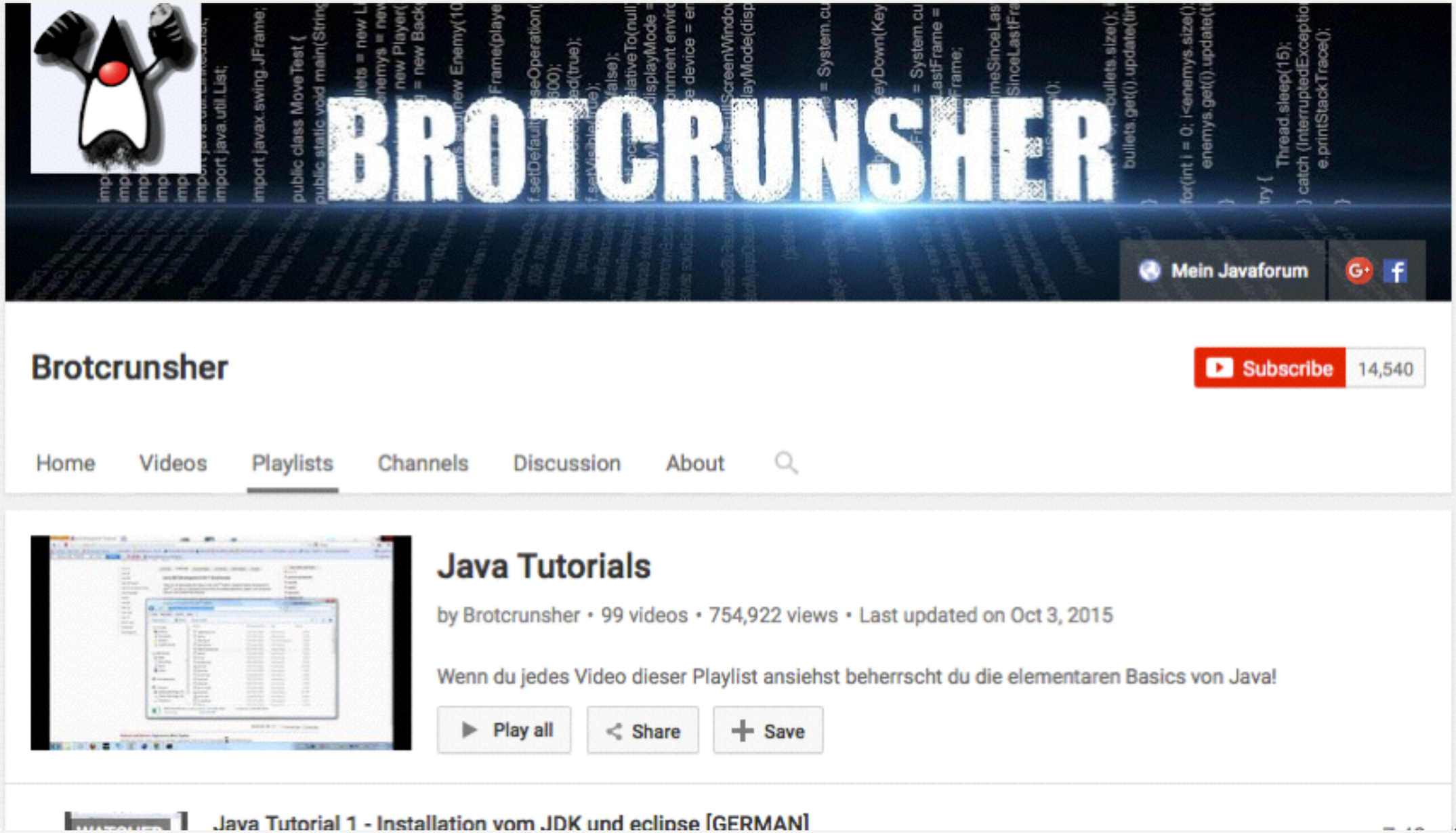
- Nach einer Eingewöhnungsphase in Java, werdet ihr in den Übungen Stück für Stück das Spiel „Pong“ entwickeln

- Komplexe Codefragmente werden von uns zur Verfügung gestellt

- Nach der letzten Übungsstunden wird die “Musterlösung” bekannt veröffentlicht

- Für alle, die „Pong“ nicht kennen: https://www.youtube.com/watch?v=ZmrJ3OBmfBM

- Eine Empfehlung eurer Vorgänger



https://www.youtube.com/playlist?list=PL71C6DFDDF73835C2

JVM/JDK installieren

- Bevor Ihr IntelliJ verwenden könnt, *müsst* Ihr Euch die Java Laufzeitumgebung für Entwickler installieren (Java JDK; Java Development Kit)

- http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

- Verwendet dabei die zu diesem Zeitpunkt aktuelle Version 8u111

Fragen?

Vielen Dank für Eure Zeit!

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Einführung in die Programmierung für Nebenfach Medieninformatik

Beat Rossmy, Michael Kirsch

Direct Feedback - Socrative

- Eure Mitarbeit ist uns wichtig!

- Installiert euch dazu die kostenlose App „Socrative Student“ auf Eurem Smartphone oder nutzt das Webinterface unter http://b.socrative.com/login/student/

- Anonymer Login über den Raumnamen:

MSMJ0KRQ

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Objekte und Methoden

Verwendung von IntelliJ, Klassen, Konzepte, Objekte, Verhalten von Objekten bei Java Swing, ActionListener

Beat Rossmy, Michael Kirsch

Verspätete Kurzvorstellung :-)

Wer bin ich?

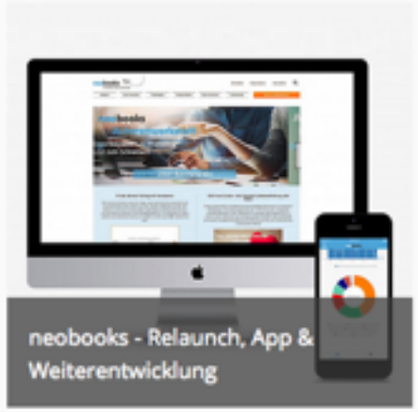
- Michael Kirsch (29 Jahre)
- Berufsausbildung zum Fachinformatiker (Anwendungsentwicklung)
- B.Sc. Hochschule Hof (Technische Informatik)
  - Auslandsaufenthalt am **I**nternational **I**nstitute of **I**nformation **T**echnology Bangalore / Indien (M.Eng.)
- M.Sc. LMU München (Informatik)
- Softwarearchitekt / Softwareentwickler bei jambit WHERE INNOVATION WORKS

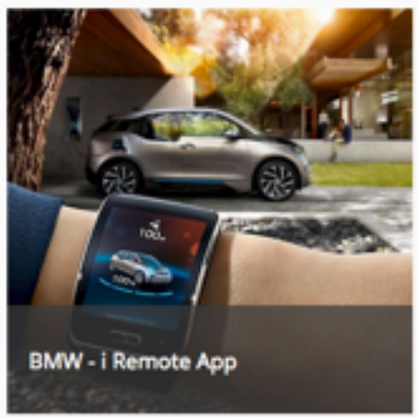
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Jambit - Where Innovation Works

jambit WHERE INNOVATION WORKS

Innovationstories





Innovationstories

Kaffeestory

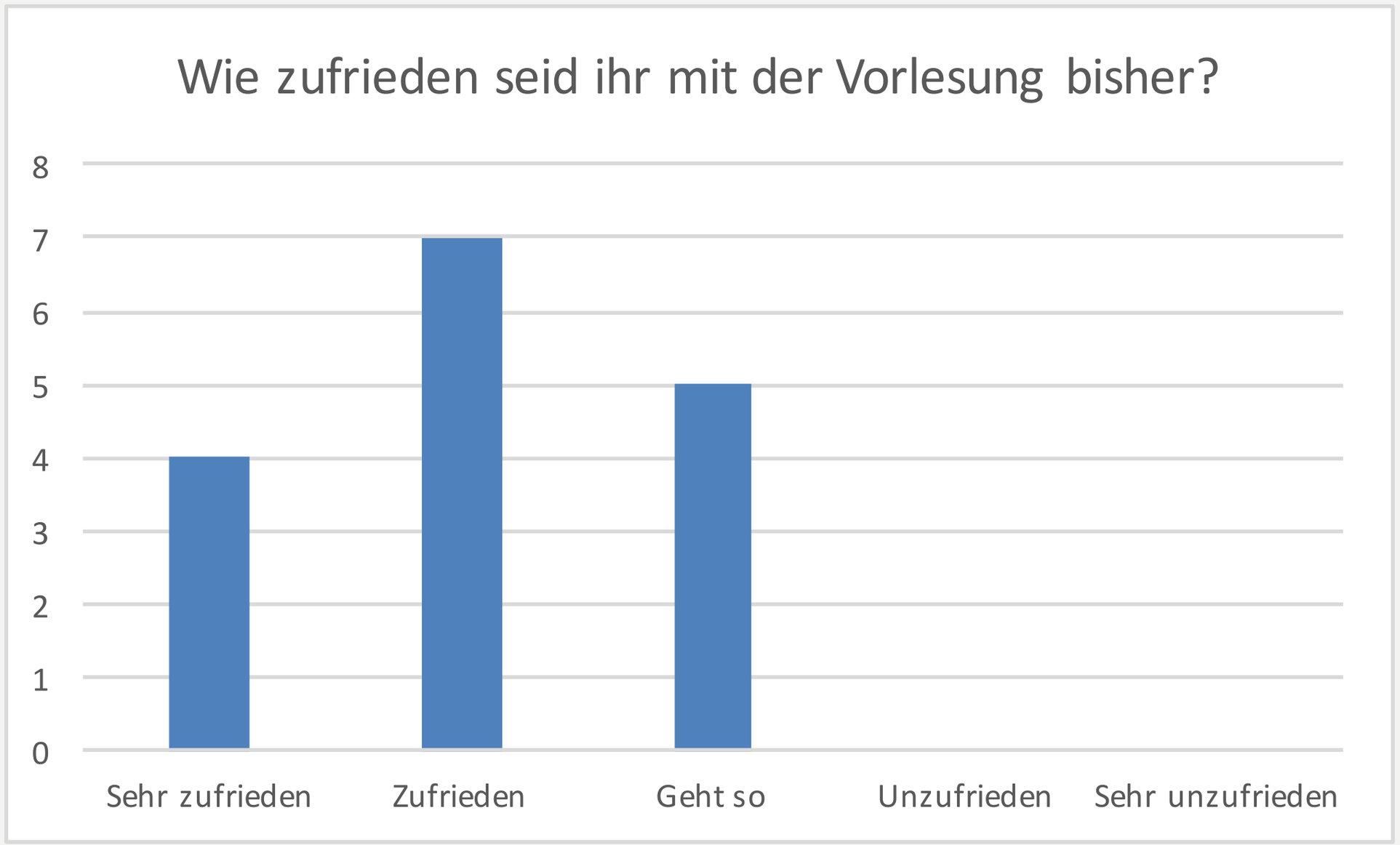
Ende 1999 saßen die beiden Geschäftsführer Peter Fellinger und Markus Hartinger bei einer Tasse Cappuccino in ihrem Lieblingscafé "kaffee & mehr" am Viktualienmarkt zusammen. Auf der Suche nach einem geeigneten Namen für ihre neue Firma blätterten sie in einem Kaffeehandbuch. Darin tauchte der Name "jambit" auf: Es sei die beste Kaffee-Plantage auf der indonesischen Insel Java, hieß es. Der Name enthielt "IT", in Suchmaschinen gab es dazu keine Einträge, Domains waren frei – beschlossene Sache!

Natürlich war die Versuchung groß, genau diesen legendären Kaffee auch tatsächlich probieren zu können. Die Suche nach der exakten Lage der Plantage war nicht einfach, und erst der Hinweis in einem Reiseführer über Indonesien, dass auch "Jampit" oder gar "Djampit" phonetisch korrekte Transkriptionen aus der indonesischen Sprache sind, führte zu der sagenumwobenen Hochebene auf dem Vulkangestein Javas.

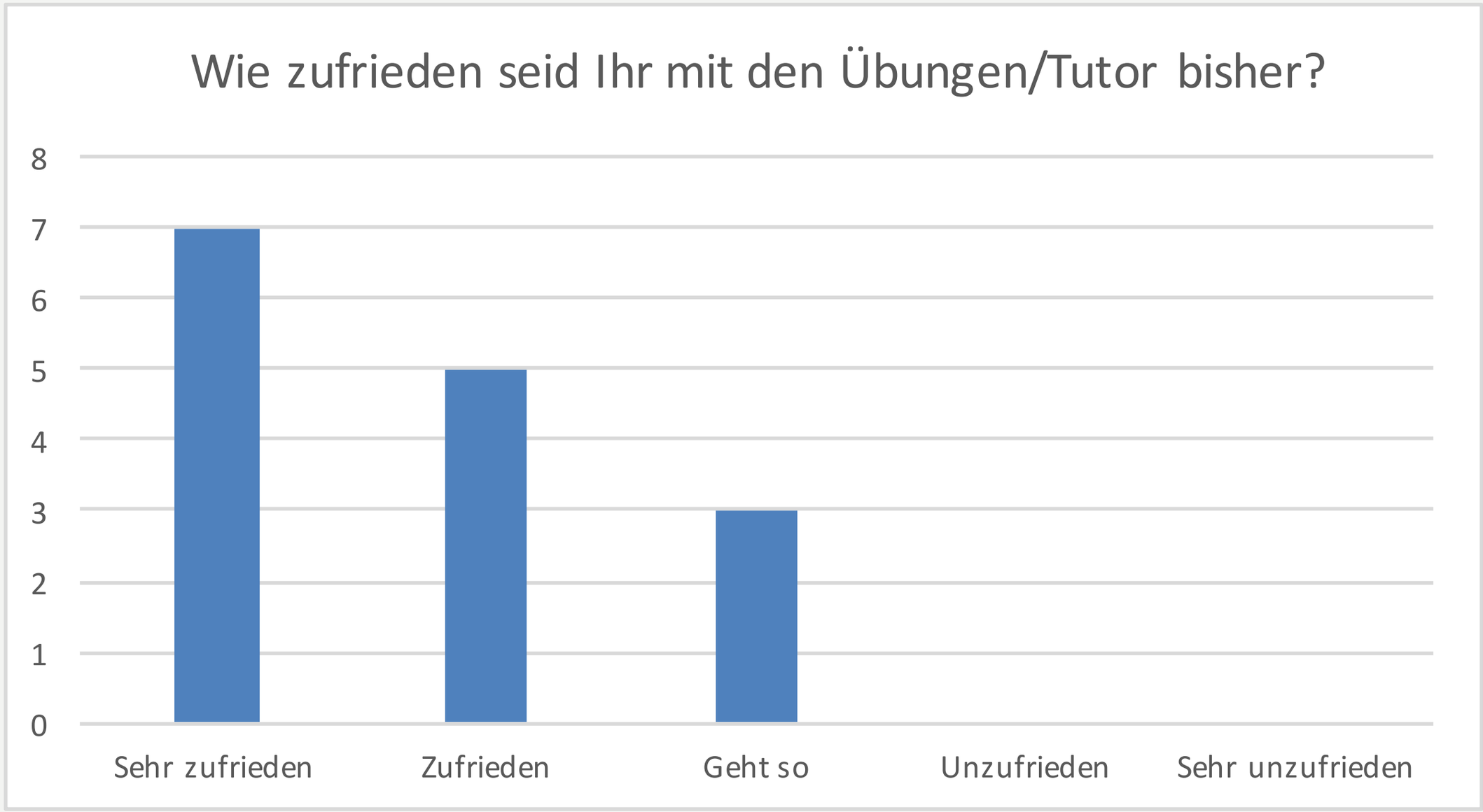
Seitdem lassen wir regelmäßig originalen, unvermischten Kaffee aus Jampit kommen und in München speziell für uns frisch von Hand rösten.

Ergebnisse der Umfrage

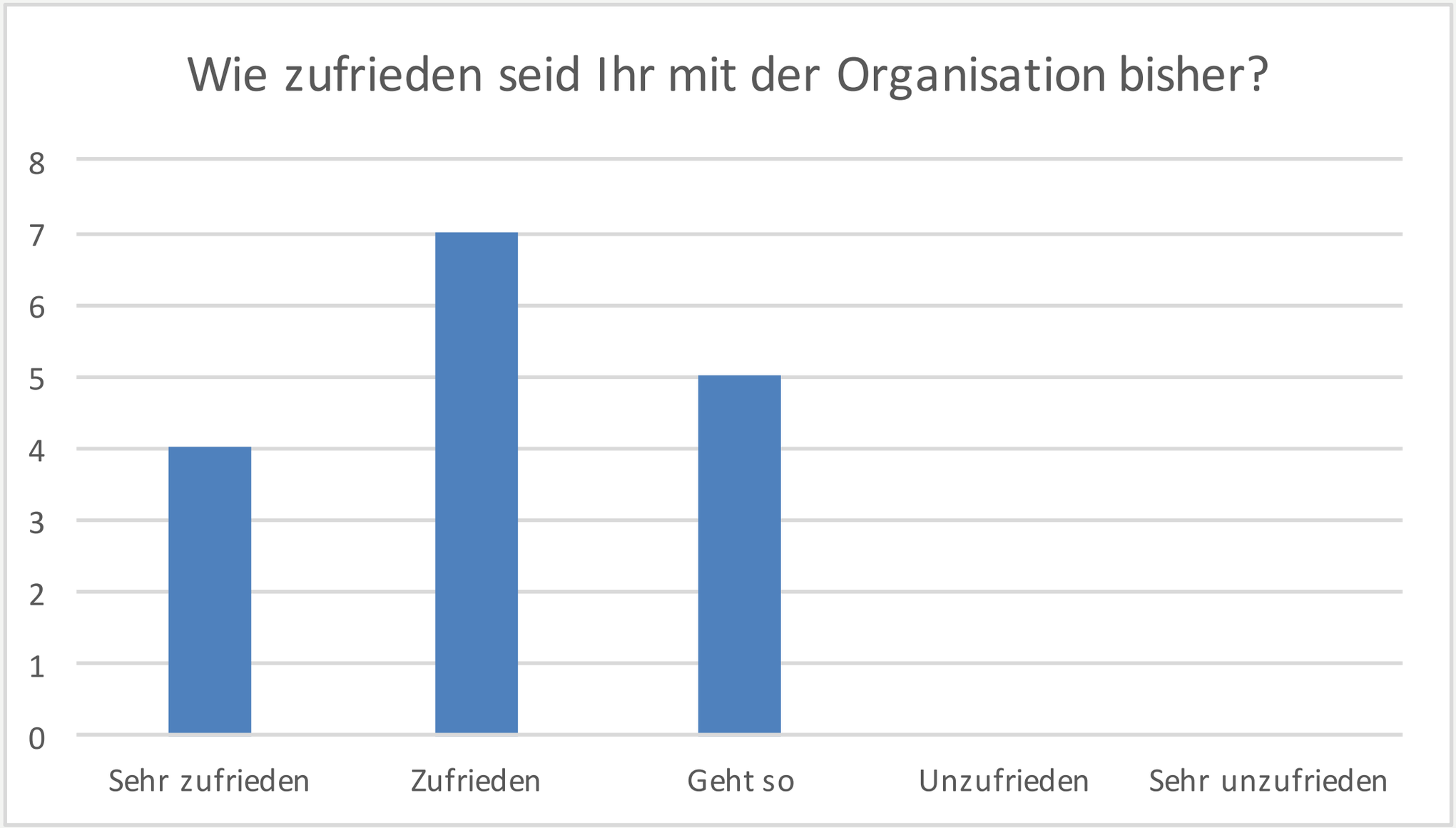
Ergebnis der Umfrage zur Vorlesungszufriedenheit



Ergebnis der Umfrage zur Vorlesungszufriedenheit



Ergebnis der Umfrage zur Vorlesungszufriedenheit



Quiztime :)

Quiz: Welche Ausgabe erzeugt das folgende Programm?

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Quiztime - Raumnummer

MSMJ0KRQ

```java
public class Quiz01 {

    public static void main(String[] args) {

        int result;
        result = add( zahl1: 3,  zahl2: 4);
        System.out.println("Das Ergebnis lautet: " + result);

        System.out.println("Das Ergebnis lautet: " + addiere( ersteZahl: 3,  zweiteZahl: 4));

    }

    public static int add(int zahl1, int zahl2) {
        int ergebnis;
        ergebnis = zahl1 + zahl2;

        return ergebnis;

    }

    public static int addiere(int ersteZahl, int zweiteZahl) {

        return ersteZahl + zweiteZahl;
    }

}
```

Quiz: Welche Ausgabe erzeugt das folgende Programm?

A: Das Ergebnis lautet: 7
Das Ergebnis lautet: 7

B: Das Ergebnis lautet: 7 Das Ergebnis lautet: 7

C: Das Ergebnis lautet: 7
Das Ergebnis lautet: 14

D: Das Ergebnis lautet: 14
Das Ergebnis lautet: 7

Quiz: Welche Ausgabe erzeugt das folgende Programm?

A: Das Ergebnis lautet: 7
Das Ergebnis lautet: 7

B: Das Ergebnis lautet: 7 Das Ergebnis lautet: 7

C: Das Ergebnis lautet: 7
Das Ergebnis lautet: 14

D: Das Ergebnis lautet: 14
Das Ergebnis lautet: 7

Quiz: Welche Ausgabe erzeugt das folgende Programm?

```java
public class Quiz02 {

    public static void main(String[] args) {

        int i, h;

        i = 4;
        h = i;

        if (i == h) {

            int k = 42;
            int a = 100000;
            h = k + i;

        }
        i = 0;
        h = i;

        System.out.println("i + a");
    }

}
```

Quiz: Welche Ausgabe erzeugt das folgende Programm?

A: 0

B: Das Programm funktioniert so nicht!

C: i + a

D: 100004

Quiz: Welche Ausgabe erzeugt das folgende Programm?

A: 0

B: Das Programm funktioniert so nicht!

C: i + a

D: 100004

Quiz: Was wird benötigt, um Java zu programmieren UND das Programm auszuführen?

A: Gute Drogen und viel Hoffnung

B: Die JRE (Java Runtime Environment) reicht aus

C: Das neue Macbook mit Touch Bar, 16 GB Arbeitsspeicher für 3199,00€

D: JDK (Java Development Kit) und JRE (Java Runtime Environment)

Quiztime - Raumnummer

MSMJ0KRQ

Quiz: Was wird benötigt, um Java zu programmieren UND das Programm auszuführen?

A: Gute Drogen und viel Hoffnung

B: Die JRE (Java Runtime Environment) reicht aus

C: Das neue Macbook mit Touch Bar, 16 GB Arbeitsspeicher für 3199,00€

D: JDK (Java Development Kit) und JRE (Java Runtime Environment)

Inhalt

Übersicht - Wie geht es weiter?

Einführung in die Programmierung

Processing		Java	
Typen und Operatoren	Kontroll-Strukturen	Grundlagen aus Processing	Swing
Klassen und Objekte	Gültigkeit und Konventionen	Objekte/Klassen	…
Methoden	Arrays		
Konstruktoren	Eingaben-verarbeitung		
Animationen	…		

Einleitung

Wie geht es weiter?

- Einführung in Java Swing (Oberflächenprogrammierung)
- Alle folgenden Vorlesungen bauen auf Eurem Projekt auf und es werden Stück für Stück neue Techniken vermittelt, damit Ihr Euer Spiel entwickeln könnt
- Dieses Mal: Einfache Oberflächen, Grafiken und ActionListener

Zwei Arten (Paradigmen) von Programmierung

Prozedural
(Anfänge der Programmierung)

- Fest definierte Reihenfolge des Ablaufs
- Sequentielle Abarbeitung
- Verwendung von einfachen Datentypen (bspw. int, double)

Objektorientiert
(seit den 90er Jahren)

- Reihenfolge abhängig von Verwendung der Objekte
- Objekten werden Eigenschaften und Verhalten zugewiesen
- Definition von eigenen Datentypen bzw. Klassen (bspw. Adressbucheintrag oder Klasse *Mensch*)

Java Swing

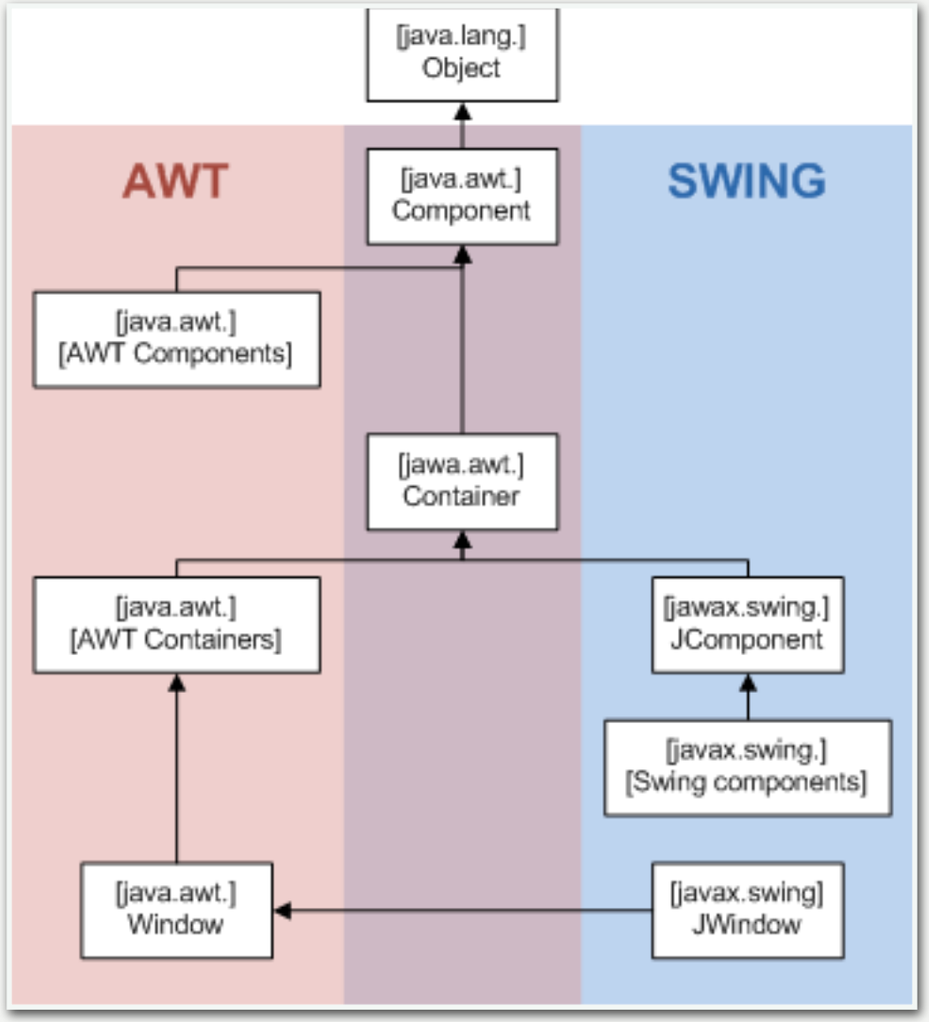
- Mit *Swing* werden Oberflächen oder auch GUI (Graphical User Interfaces) entwickelt

- Es besteht u.a. aus vielen verschiedenen Komponenten (JFrame, JPanel, JButton, JTextfield, …)

- **Diese Komponenten sind Objekte!**

- Viele Beispiele unter https://docs.oracle.com/javase/tutorial/uiswing/examples/components/

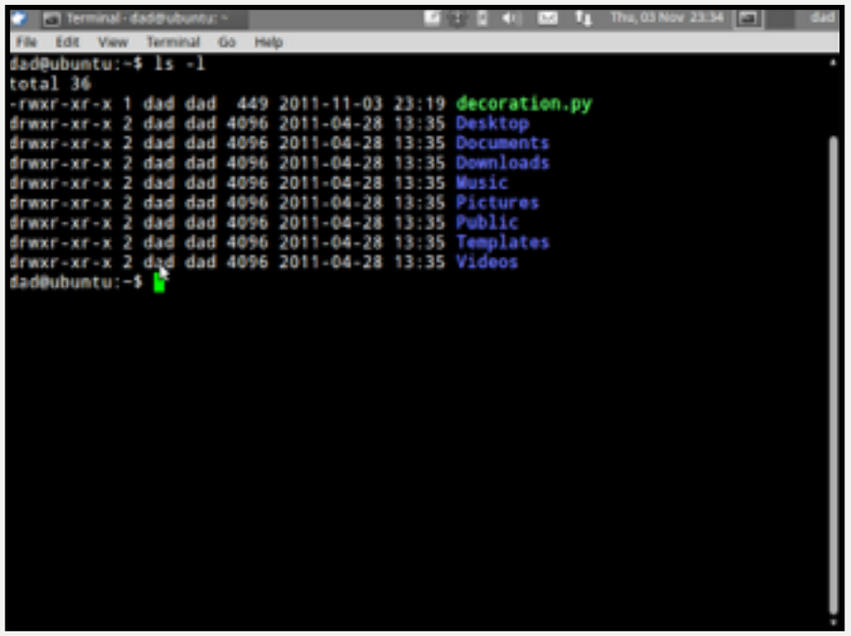
- Aufbau (siehe Abbildung)

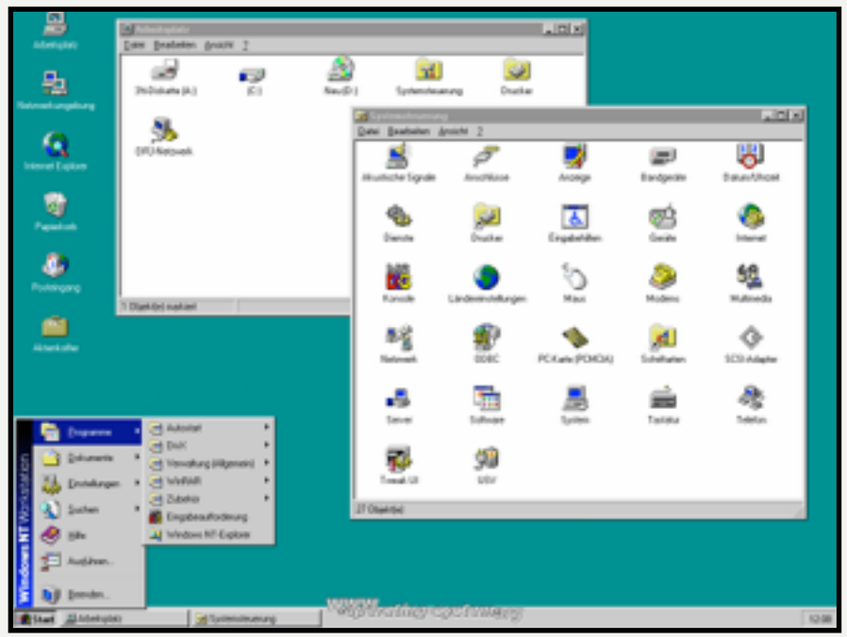


Theorie

GUI in Java

- Am Anfang gab es Terminals
- Anschließend kamen mehr und mehr grafische Anwendungen auf
- Anfangs noch einfach und funktional (Maus- & Tastatursteuerung)





GUI in Java

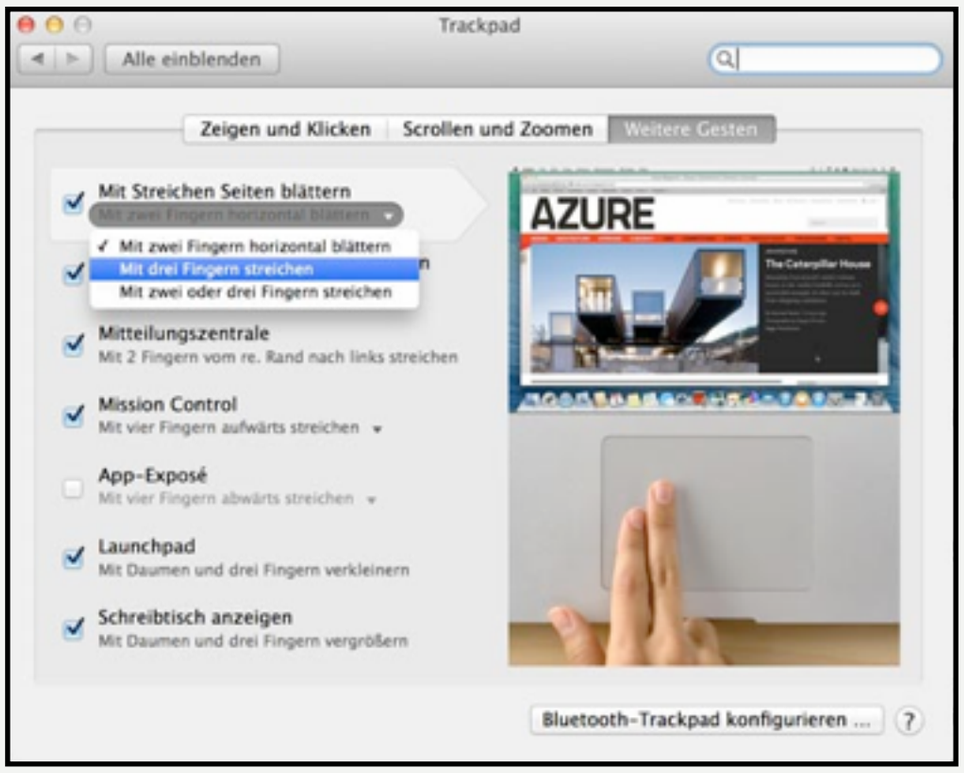
- Später komplexer und schicker (Einfache und Multitouch-Gesten)

- In Java ist das alles ebenfalls möglich

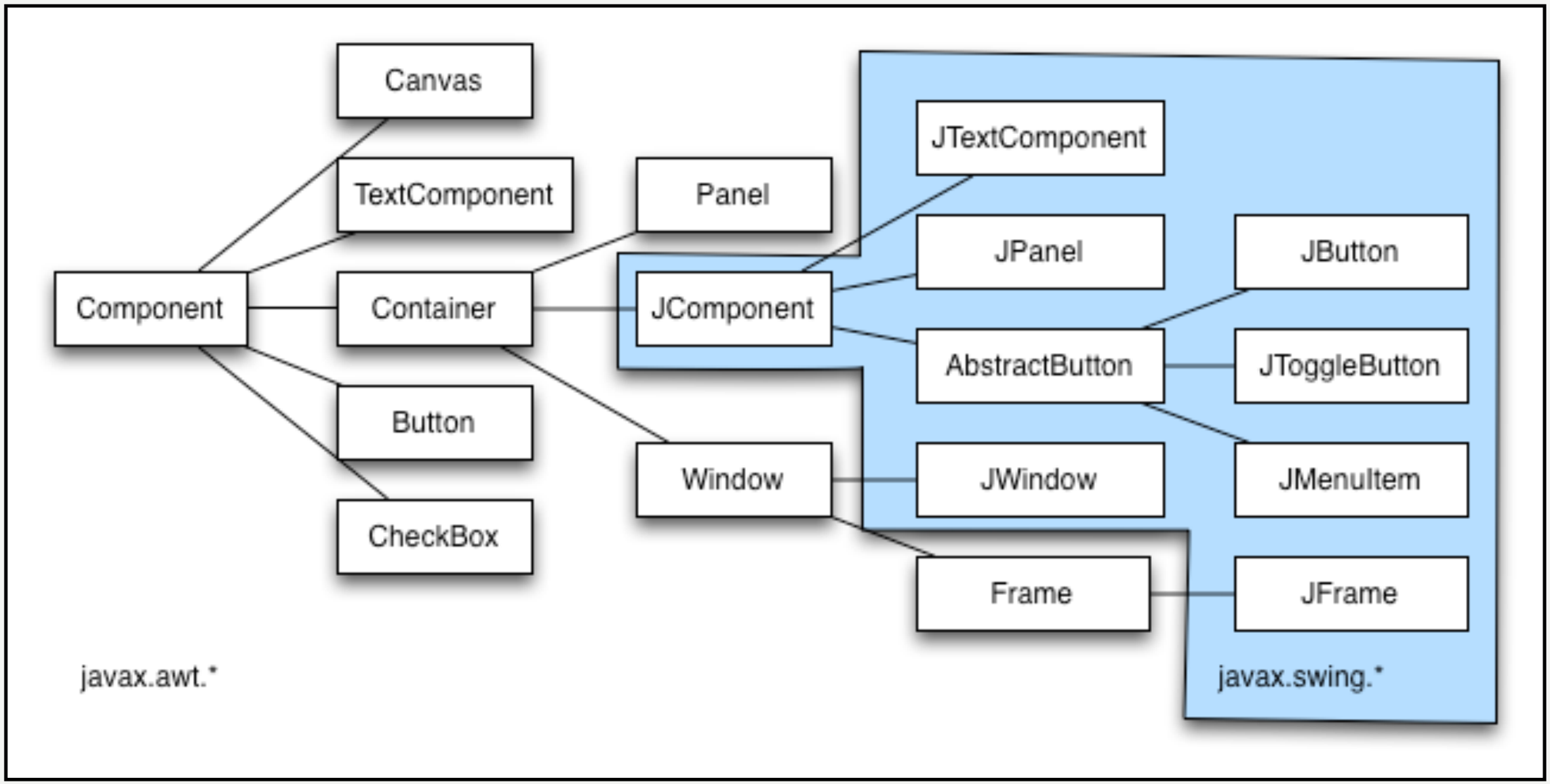
- Bibliothek in Java zur GUI-Programmierung: AWT bzw. SWING

- Swing ist wie Lego: Es werden Komponenten ineinander „gesteckt“

- AWT wurde später durch SWING abgelöst, aber manche Komponenten von AWT sind in SWING verfügbar

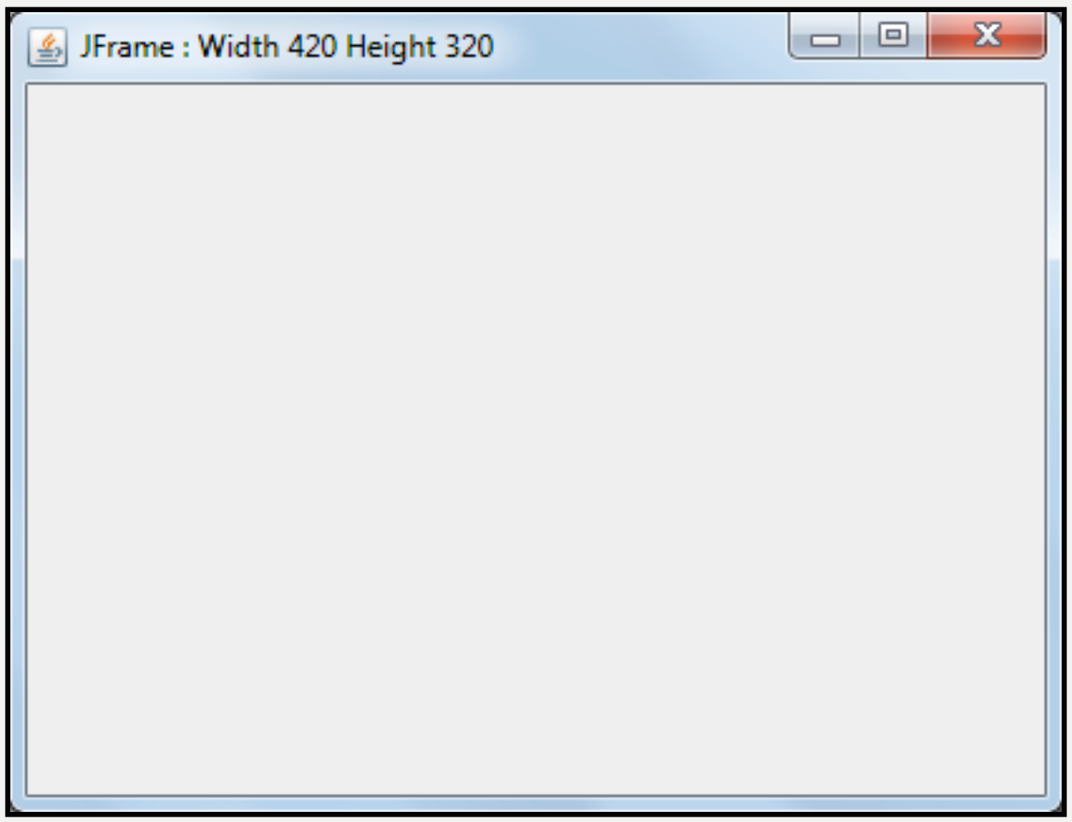


Aufbau von Swing



Vererbungshierarchie in Java (alles von Component abgeleitet)

JFrame

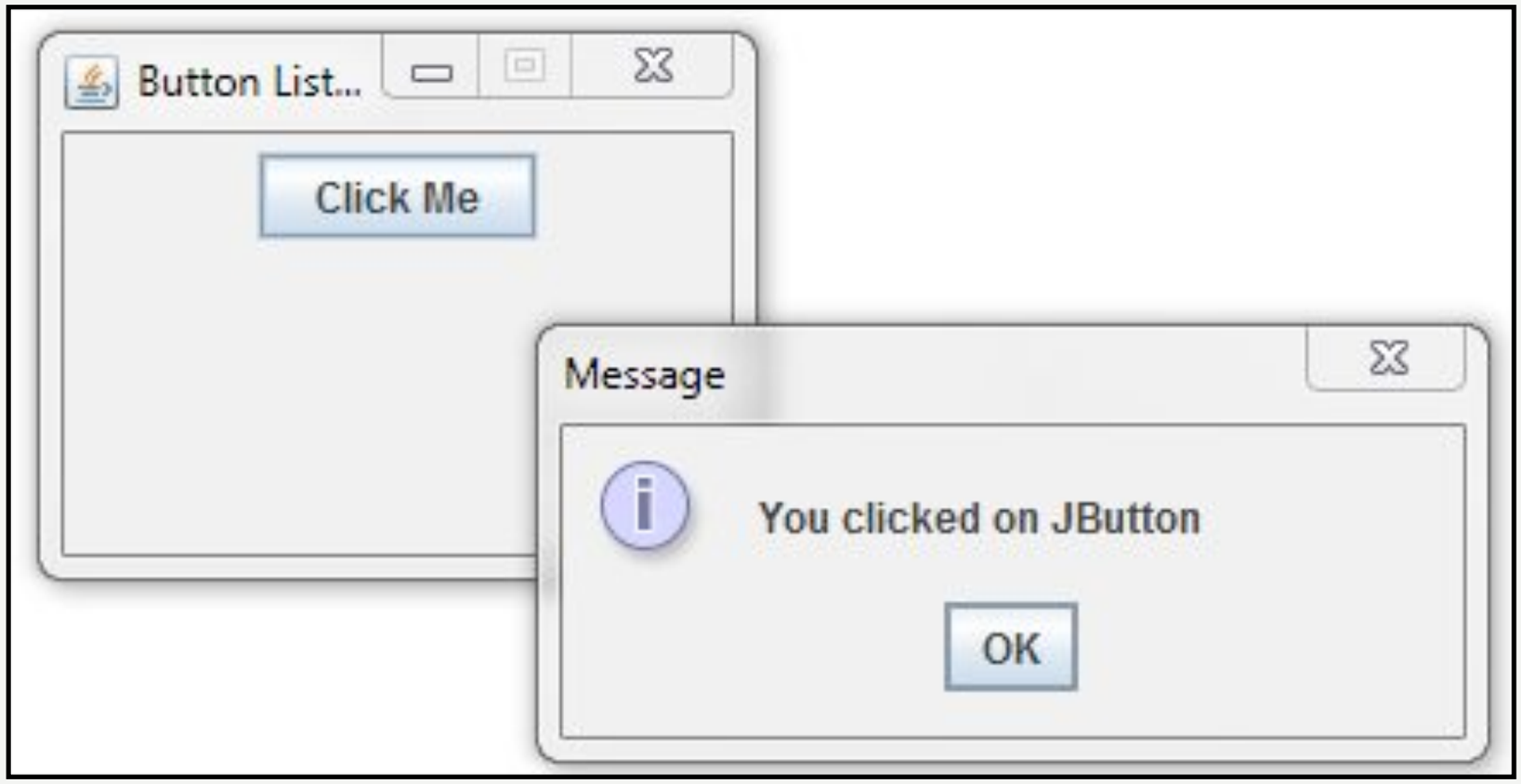


Einfacher JFrame

JFrame & JPanel

Ein JFrame enthält mehrere JPanels

JFrame & JPanel



ActionListener reagieren auf Actions (bspw. Tastendrücke)

Anwendung

JFrame

- Einfacher JFrame mit der Größe 600x800 Pixel

```
10
11 public class Main {
12
13     public static void main(String[] args) {
14
15         JFrame fenster = new JFrame(); //Erstelle einen neuen JFrame
16         fenster.setSize(new Dimension(600, 800)); //Fenstergröße 600 Pixel X 800 Pixel
17         fenster.setVisible(true); //Macht das Fenster sichtbar
18
19     }
20
21 }
22
```

JPanel in JFrame

- JPanel in einem JFrame

```
14 public class Main {
15
16     public static void main(String[] args) {
17
18         JFrame fenster = new JFrame(); //Erstelle einen neuen JFrame
19         JPanel panel = new JPanel(); //Erstelle ein neues JPanel
20
21         panel.setBorder(BorderFactory.createLineBorder(Color.BLACK)); //Füge Panel einen schwarzen Rand hinzu
22         fenster.add(panel); //Füge das Panel dem Fenster hinzu
23
24         fenster.setSize(new Dimension(600, 800)); //Fenstergröße 600 Pixel X 800 Pixel
25         fenster.setVisible(true); //Macht das Fenster sichtbar
26
27     }
28
29 }
30
```

KeyListener

- KeyListener ermöglichen die Abfrage von Tastatureingaben
- Verwendung von KeyListener in zwei Varianten
  - **Implementierung des Interfaces *KeyListener***
  - Implementierung in einer *anonymen inneren Klasse*

- Damit ein Fenster Key-Events abfragen kann, muss es im *Fokus* stehen

```
spielfeld.setFocusable(true); //Aktivierung der „Fokusierbarkeit“
```

- KeyListener können die folgenden drei Events abfragen

```
public void keyTyped(KeyEvent e)
public void keyPressed(KeyEvent e)
public void keyReleased(KeyEvent e)
```

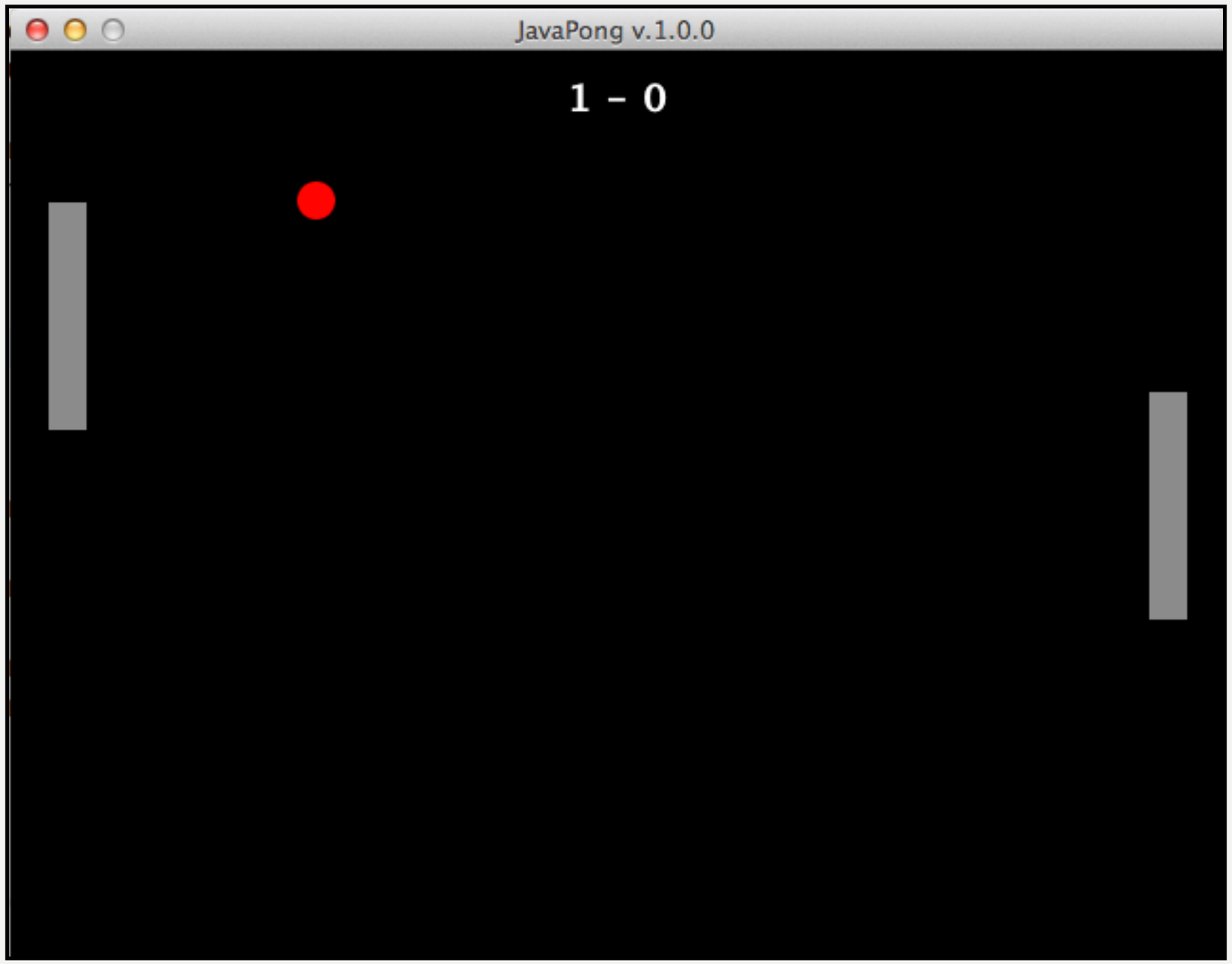
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

KeyListener

```java
public class Spielfeld extends JPanel implements KeyListener { //Spielfeld ist ein JPanel und implementiert das Interface Keylistener

    public void Spielfeld() { //Konstruktor; Bisher ohne weitere Funktion

    }

    @Override
    public void keyTyped(KeyEvent e) { //Wenn Taste gedrückt und wieder losgelassen wurde

    }

    @Override
    public void keyPressed(KeyEvent e) { //Wenn eine Taste gedrückt
    if (e.getKeyCode() == KeyEvent.VK_LEFT) { //Wenn die Taste == linke Pfeiltaste
        System.out.println("Linke Pfeiltaste");
    }

    else if (e.getKeyCode() == KeyEvent.VK_RIGHT) { //Wenn die Taste == rechte Pfeiltaste
        System.out.println("Rechte Pfeiltaste");
    }

    else if (e.getKeyCode() == KeyEvent.VK_DOWN) { //Wenn die Taste == untere Pfeiltaste
        System.out.println("Untere Pfeiltaste");

    }

    else if (e.getKeyCode() == KeyEvent.VK_UP) { //Wenn die Taste == obere Pfeiltaste
        System.out.println("Obere Pfeiltaste");

    }

    }

    @Override
    public void keyReleased(KeyEvent e) { //Wenn Taste losgelassen wurde
    }

}
```

Verknüpfung

Pong in Java (JPong)

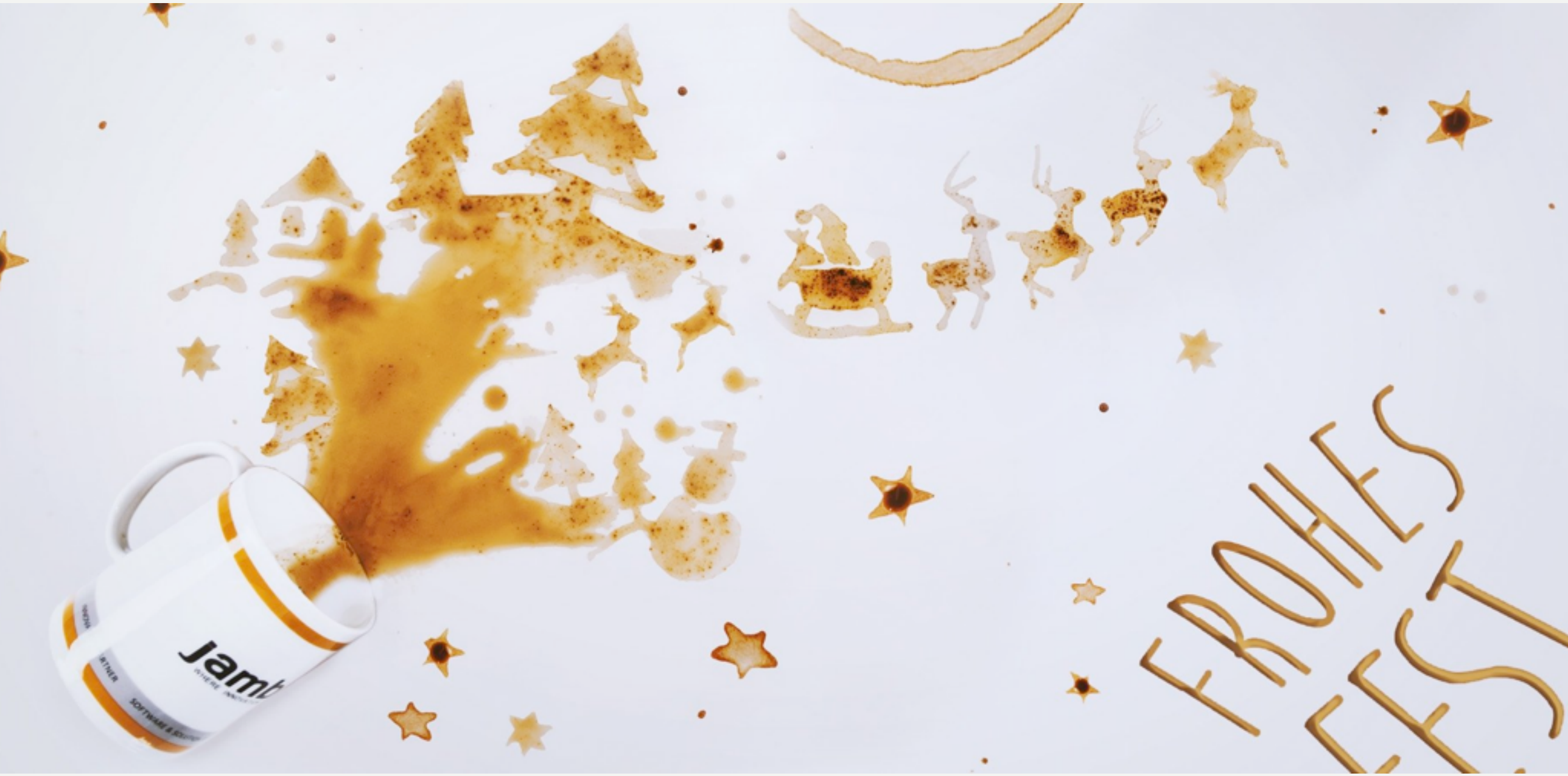


Ausblick

Ausblick

- In den Übungen werden die Inhalte der heutigen Vorlesung vertieft
  - Selbst Spielfeld erstellen
  - Ball erstellen und bewegen
  - Ball auf Spielfeld halten (Grenzen)
- Nach der letzten Übung, wird eine Musterlösung bereitgestellt
- Nach der Vorlesung wird euch ein “Code-Grundgerüst” bereitgestellt
- **Im neuen Jahr werden wir wir Animationen einführen**

Fragen?


FROHES
FEST

Vielen Dank für Eure Zeit

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Einführung in die Programmierung für Nebenfach Medieninformatik

Beat Rossmy, Michael Kirsch

Direct Feedback - Socrative

- Eure Mitarbeit ist uns wichtig!

- Installiert euch dazu die kostenlose App „Socrative Student“ auf Eurem Smartphone oder nutzt das Webinterface unter http://b.socrative.com/login/student/

- Anonymer Login über den Raumnamen:

MSMJ0KRQ

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Grafikobjekte

Verwendung von Java2D, Canvas und der Paint-Methode

Beat Rossmy, Michael Kirsch

Quiztime :)

Quiztime - Raumnummer

MSMJ0KRQ

Quiz: Was bedeutet die folgende Code-Zeile?

```
public class Spielfenster extends JFrame {…}
```

A: Die Klasse Spielfenster implementiert das Interface JFrame

B: Spielfenster ist ein primitiver Datentyp

C: JFrame erweitert die Klasse Spielfenster

D: Die Klasse Spielfenster erbt von JFrame und besitzt somit alle Eigenschaften und Methoden eines JFrames

Quiz: Welche der folgenden Aussagen bzgl. JPanel im Kontext von Java Swing sind/ist richtig?

A: JPanels werden beim Bau von Fertighäusern verwendet

B: Ein JPanel ist eine Komponente, die mit weiteren Komponenten „bestückt“ werden kann

C: JFrames und JPanels sind das gleiche

D: Ein JPanel kann einen JFrame enthalten

Quiz: Welche Aussage bzgl. KeyListener bei Java Swing sind/ist falsch?

A: KeyListener ermöglichen die Abfrage von allen mögliche Tastatureingaben

B: KeyListener können „nur“ Zustände bei Tastendrücken erkennen

C: KeyListener können ebenfalls bei JButtons verwendet werden

D: KeyListener besitzen als Parameter ein Key-Event

Inhalt

I. Einleitung

1. Wie geht es weiter?

II. Theorie

1. Der this-Operator
2. Keylistener (2)
3. Die paint-Methode
4. Threads & Animationen
5. Try/Catch

III. Anwendung

1. Praxis in Eclipse

IV. Verknüpfung

1. Tutorials

V. Ausblick

1. Nächste Vorlesung
2. Übung

Übersicht - Wie geht es weiter?

Einführung in die Programmierung

Processing		Java	
Typen und Operatoren	Kontroll-Strukturen	Grundlagen aus Processing	Swing
Klassen und Objekte	Gültigkeit und Konventionen	Objekte/Klassen	Interfaces
Methoden	Arrays	…	Vererbung
Konstruktoren	Eingaben-verarbeitung		
Animationen	…		

Einleitung

News - TIOBE Index for January 2017

Jan 2017	Jan 2016	Change	Programming Language	Ratings	Change
1	1		Java	17.278%	-4.19%
2	2		C	9.349%	-6.69%
3	3		C++	6.301%	-0.61%
4	4		C#	4.039%	-0.67%
5	5		Python	3.465%	-0.39%
6	7	^	Visual Basic .NET	2.960%	+0.38%
7	8	^	JavaScript	2.850%	+0.29%
8	11	^	Perl	2.750%	+0.91%
9	9		Assembly language	2.701%	+0.61%
10	6	⌄⌄	PHP	2.564%	-0.14%
11	12	^	Delphi/Object Pascal	2.561%	+0.78%
12	10	⌄	Ruby	2.546%	+0.50%
13	54	^^	Go	2.325%	+2.16%
14	14		Swift	1.932%	+0.57%
15	13	⌄	Visual Basic	1.912%	+0.23%
16	19	^	R	1.787%	+0.73%
17	26	^^	Dart	1.720%	+0.95%

http://www.tiobe.com/tiobe-index/

Wie geht es weiter?

- Bisher wurde das Spielfeld erstellt und Tasteneingaben verwertet
- Dieses Mal: Paint-Methode, Threads und Fehlerbehandlung

Theorie

this-Operator

- **this**-Operator

- Mit dem **this**-Operator kann auf die Klasse zugegriffen werden, die den Code enthält. Somit ist es möglich innerhalb dieser Klasse Attribute und Methoden der Klasse selbst zu verwenden.

```
16 public class Spielfenster extends JFrame { // Spielfenster leitet von JFrame ab u
17
18     Spielfeld spielfeld;
19     JLabel score;
20
21     public Spielfenster() { // In dem Konstruktor werden alle Konfigurationen vor
22
23         spielfeld = new Spielfeld(); // Ein neues Objekt vom Typ Spielfeld wird e
24         score = new JLabel("Aktueller Spielstand 0 : 0 "); // Ein neues Objekt vo
25
26         score.setFont(new Font("Calibri", Font.BOLD, 20)); // Setzt den Font der
27         score.setForeground(new Color(0x0EE00)); // Setzt die Farbe auf Grün bzw.
28
29         spielfeld.setFocusable(true); // Ermöglicht, dass die Keylistener auf die
30
31         this.setSize(new Dimension(600, 600)); // Setzt die Größe des Spielfenste
32         this.setLocation(500, 100); // Setzt die Position des Spielfensters
33         spielfeld.setBackground(new Color(0x212121)); // Setzt die die Hintergrun
34
```

KeyListener

- KeyListener ermöglichen die Abfrage von Tastatureingaben

- Verwendung von KeyListener in zwei Varianten
  - Implementierung des Interfaces *KeyListener*
  - Implementierung in einer *anonymen inneren Klasse*

- Damit ein Fenster Key-Events abfragen kann, muss es im *Fokus* stehen

```
spielfeld.setFocusable(true); //Aktivierung der „Fokusierbarkeit“
```

- KeyListener können die folgenden drei Events abfragen

```
public void keyTyped(KeyEvent e)
public void keyPressed(KeyEvent e)
public void keyReleased(KeyEvent e)
```

Die Paint-Methode

- Um Formen wie Kreise, Rechtecke usw. darstellen zu können, wird die „paint-Methode“ benötigt

- In ihr wird beschrieben, *was* “gemalt” werden soll

- Beispiel: Ein roter Kreis an der Position (x=300, y=300) mit der Größe 20x20 Pixel

```
@Override
 public void paint(Graphics g) {
   super.paint(g);
   Graphics2D g2d = (Graphics2D) g;
   g2d.setColor(Color.red);
   g2d.fillOval(300, 300, 20, 20);
 }
```

Die Paint-Methode

- Besonderheit: Um die Inhalte erneut zu zeichnen, muss sie aufgerufen werden

```
repaint(); //Erneut die paint-Methode aufrufen
```

- Um Änderungen bei den Inhalten zu ermöglichen, können Variablen verwendet werden

```
public void paint(Graphics g) {
 super.paint(g);
 Graphics2D g2d = (Graphics2D) g;
 g2d.setColor(Color.red);
 g2d.fillOval(ballXpos, ballYpos, 20, 20);
}
```

Threads & Animationen

- Um Bewegung in die gezeichneten Elemente zu bekommen, benötigen wir Threads

- Threads sind Unterprogramme in unserem Hauptprogramm

- D.h. das Unterprogramm kümmert sich um die Animation. Das Hauptprogramm um alles übrige (Tasteneingaben, Steuerung, Punktstand)

- Um mit Threads zu arbeiten benötigen wir zwei Dinge
  - Das Interface *Runnable*
  - Die *Run*-Methode

Threads & Animationen

- Das Interface *Runnable* ermöglicht uns die *Run-Methode* zu implementieren

```
public class Spielfeld extends JPanel implements KeyListener, Runnable {…}
```

- Die Run-Methode wird als Thread (Unterprogramm) ausgeführt

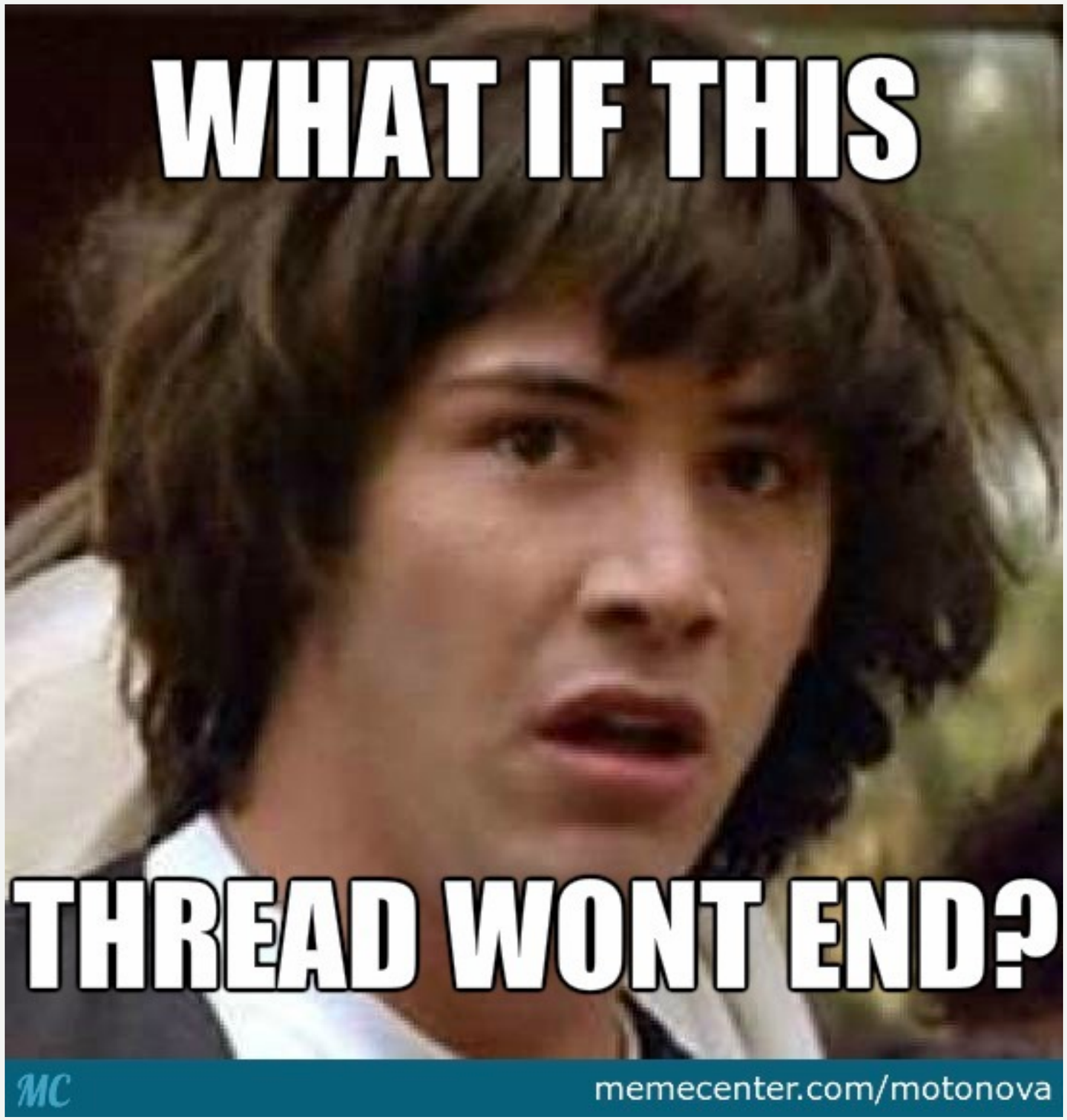
```
@Override
 public void run() {
   //Animation
   int offset = 1;
   while (true) {
      ballXpos = ballXpos + offset;
      ballYpos = ballYpos + offset;
      repaint();
   }
}
```

Threads & Animationen

- Threads können auch „schlafen gelegt werden“, wobei die „Schlafzeit“ in Millisekunden angegeben wird.

```
@Override
public void run() {
  //Animation
  int offset = 1;
  while (true) {
     ballXpos = ballXpos + offset;
     ballYpos = ballYpos + offset;
     repaint();
     Thread.currentThread().sleep(5);
  }
}
```

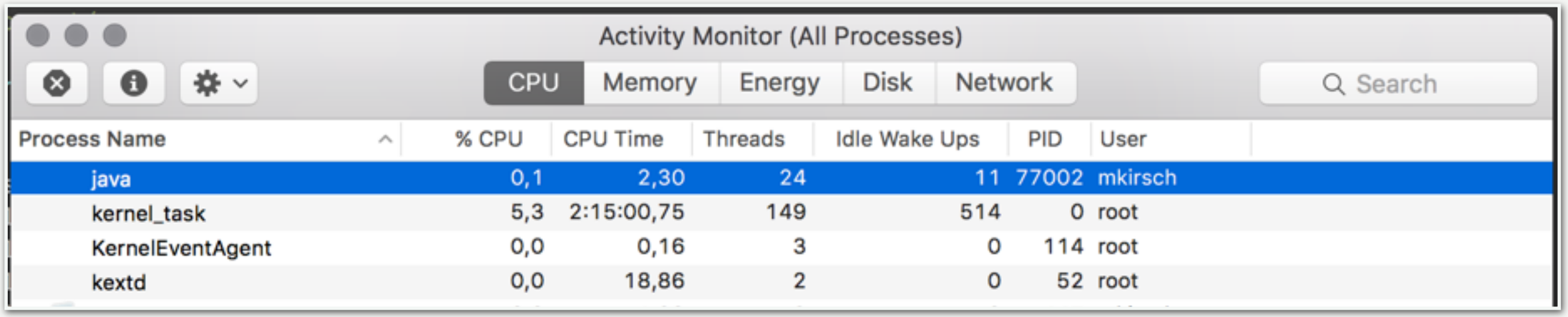
Threads & Animationen



Threads & Animationen

- Threads sind Unterprogramme des Hauptprogramms. In unserem Fall ist der Animations-Thread Teil von Java Pong.

- Wird das Spiel Java Pong beendet, so wird auch der Thread selbst beendet



- “Java” ist der Prozess von unserem Spiel, der wiederum 24 weitere Threads besitzt. Einer davon ist unser Animations-Thread.

Fehler & Behandlung

- Mittels Try/Catch-Blöcken, können problematische Codefragmente „abgesichert“ werden


```
@Override
public void run() {
int offset = 1;

while (true) {

    try { //Versuche das Folgende…
    ballXpos = ballXpos + offset;
    ballYpos = ballYpos + offset;
    repaint();
    Thread.currentThread().sleep(5);

  }catch (InterruptedException e) { //Wenn es schief geht, hier weiter…
      e.printStackTrace();
      }
  }
}
```

Anwendung

Anwendung in IntelliJ

- Hands on…

Verknüpfung

Tutorials / Literatur

- Bei vielen Internetseiten findet ihr die behandelten Themen weiter und ausführlicher erklärt

- Eine Auswahl:

  - Brotcrunsher (https://www.youtube.com/playlist?list=PL71C6DFDDF73835C2)

  - Java-Tutorials (http://www.java-tutorial.org/swing.html)

  - Head First http://shop.oreilly.com/product/9780596009205.do

Ausblick

Ausblick

- In den Übungen werden die Inhalte der heutigen Vorlesung vertieft
  - Spielelemente (Kreis, Rechtecke) zeichnen
  - Ball von Programm automatisch bewegen lassen
  - Ball auf Spielfeld halten (Grenzen)

Fragen?

Vielen Dank für Eure Aufmerksamkeit

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Einführung in die Programmierung für Nebenfach Medieninformatik

Beat Rossmy, Michael Kirsch

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Zusammenfassung

Beat Rossmy, Michael Kirsch

Klausurtermin

- Klausur am Donnerstag den 2.03.2016 im Raum B001der Oettingenstraße 67 in München

- Begin der Klausur um 10 Uhr s.t.

- Ende der Klausur um 12 Uhr s.t.

- Hilfsmittel: Keine

- Bringt einen Personalausweis/Reisepass **und** Studentenausweis mit

Übersicht - Wie geht es weiter?

Einführung in die Programmierung

Processing		Java	
Typen und Operatoren	Kontroll-Strukturen	Grundlagen aus Processing	Swing
Klassen und Objekte	Gültigkeit und Konventionen	Objekte/Klassen	Interfaces
Methoden	Arrays	Threads	Vererbung
Konstruktoren	Eingaben-verarbeitung		
Animationen	…		

Zusammenfassung

- Datentypen
- Variablen & Zuweisungen
- Methoden
- Klassen & Objekte
- Konstruktoren
- Interfaces (implements…)
- Vererbung (extends…)

Zusammenfassung

- Swing
  - JFrame
  - JPanel
- Threads
  - Run-Methode
  - Runnable-Interface
- Paint-Methode

Zusammenfassung

- Keylistener
  - Events
- Schleifen
- Arrays
- Gültigkeitsbereiche

Fragen?

Vielen Dank für Eure Aufmerksamkeit

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Einführung in die Programmierung für Nebenfach Medieninformatik

Beat Rossmy, Michael Kirsch

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Zusammenfassung

Beat Rossmy, Michael Kirsch

Klausurtermin

- Klausur am Donnerstag den 2.03.2017 im Raum B001 der Oettingenstraße 67 in München

- Begin der Klausur um 10 Uhr s.t.

- Ende der Klausur um 12 Uhr s.t.

- Hilfsmittel: Keine

- Bringt einen Personalausweis/Reisepass **und** Studentenausweis mit

Direct Feedback - Socrative

- Eure Mitarbeit ist uns wichtig!

- Installiert euch dazu die kostenlose App „Socrative Student“ auf Eurem Smartphone oder nutzt das Webinterface unter http://b.socrative.com/login/student/

- Anonymer Login über den Raumnamen:

MSMJ0KRQ

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Übersicht - Wie geht es weiter?

Einführung in die Programmierung

Processing

- Typen und Operatoren
- Kontroll-Strukturen
- Klassen und Objekte
- Gültigkeit und Konventionen
- Methoden
- Arrays
- Konstruktoren
- Eingaben-verarbeitung
- Animationen
- …

Java

- Grundlagen aus Processing
- Swing
- Objekte/Klassen
- Interfaces
- Threads
- Vererbung

Zusammenfassung

- Datentypen
- Variablen & Zuweisungen
- Methoden
- Klassen & Objekte
- Konstruktoren
- Interfaces (implements…)
- Vererbung (extends…)

Zusammenfassung

- Swing
  - JFrame
  - JPanel
- Threads
  - Run-Methode
  - Runnable-Interface
- Paint-Methode

Zusammenfassung

- Keylistener
  - Events
- Schleifen
- Arrays
- Gültigkeitsbereiche

Fragen?

Vielen Dank für Eure Aufmerksamkeit