PROCESSING

EINE EINFÜHRUNG IN DIE INFORMATIK

Created by Michael Kirsch & Beat Rossmy

INHALT

1. Stoff der Vorlesung
    1. PImage
    2. Modulo
2. Übung
    1. Aufgabe 1
    2. Aufgabe 2
    3. Animation-Array

STOFF DER VORLESUNG

PIMAGE

Wir laden und zeigen ein Bild.

```
PImage img;

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

void draw () {
  image(img,0,0);
}
```

PIMAGE

Wir greifen auf die einzelnen Pixel zu.

```
PImage img;

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

  img.loadPixels();
  img.pixels[100] = color(255,0,0);
  img.updatePixels();
}

void draw () {
  image(img,0,0);
}
```

PIMAGE

Wir greifen auf die die Höhe und Breite des Bildes zu.

```
PImage img;

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

void draw () {
  rect(0,0,img.width,img.height);
}
```

MODULO

Wir verwenden Modulo

```
int counter;

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

void draw () {
  background(0);
  int d = counter%400;
  rect(300,200,d,d);
}
```

ÜBUNG

AUFGABE 1

Zeichne Rechteck mit Mittelpunkt auf Ursprung.

```
class RotatingRect extends Animation {
  int x,y;

  public RotatingRect (int x, int y, char triggerKey) {
    super(triggerKey);
    this.x = x;
    this.y = y;
  }

  void animate () {
    if (animationCounter>0) {
      animationCounter++;
      // verschiebe Rechteck
      // rotiere Rechteck
      rect(-50,-50,100,100);
    }
  }
}
```

AUFGABE 1

Rotiere das Rechteck mit einem Faktor, der dem Bruchteil von **`2*PI`** entspricht -> 360°

```
class RotatingRect extends Animation {
  int x,y;

  public RotatingRect (int x, int y, char triggerKey) {
    super(triggerKey);
    this.x = x;
    this.y = y;
  }

  void animate () {
    if (animationCounter>0) {
      animationCounter++;
      // verschiebe Rechteck
      rotate(2*PI*animationCounter*0.3);
      rect(-50,-50,100,100);
    }
  }
}
```

AUFGABE 1

Verschiebe das Rechteck an die gewünschte Koordinatenposition.

```
class RotatingRect extends Animation {
  int x,y;

  public RotatingRect (int x, int y, char triggerKey) {
    super(triggerKey);
    this.x = x;
    this.y = y;
  }

  void animate () {
    if (animationCounter>0) {
      animationCounter++;
      translate(x,y);
      rotate(2*PI*animationCounter*0.3);
      rect(-50,-50,100,100);
    }
  }
}
```

AUFGABE 1

Verhindere eine Auswirkung der Transformationen auf andere Elemente.

```
class RotatingRect extends Animation {
  int x,y;

  public RotatingRect (int x, int y, char triggerKey) {
    super(triggerKey);
    this.x = x;
    this.y = y;
  }

  void animate () {
    if (animationCounter>0) {
      animationCounter++;
      pushMatrix();
      translate(x,y);
      rotate(2*PI*animationCounter*0.3);
      rect(-50,-50,100,100);
      popMatrix();
    }
  }
}
```

AUFGABE 2

Die Klasse **`ColorCircle`** ist ein Kind von **`Animation`**.

```
class ColorCircle extends Animation {
  // Felder

  // Konstruktor

  // animate()
}
```

AUFGABE 2

Als Felder benötigen wir Variablen für die Koordinaten. Diese werden im Konstruktor übergeben.

```
class ColorCircle extends Animation {
  int x,y;

  public ColorCircle (int x, int y, char c) {
    // Aufruf des Konstruktors der Elternklasse.
    super(c);
    this.x = x;
    this.y = y;
  }

  // animate()
}
```

AUFGABE 2

Wir überschreiben die Methode **`animate`** der Elternklasse.

```
class ColorCircle extends Animation {
  int x,y;

  public ColorCircle (int x, int y, char c) {
    // Aufruf des Konstruktors der Elternklasse.
    super(c);
    this.x = x;
    this.y = y;
  }

  void animate () {
    // erzeuge zufällige Fullfarbe
    // zeichne Kreis
    // setze Füllfarbe auf Standardfarbe zurück
  }
}
```

AUFGABE 2

Ein zufälliger RGB Wert wird in **`fill()`** verwendet.

```
class ColorCircle extends Animation {
  int x,y;

  public ColorCircle (int x, int y, char c) {
    // Aufruf des Konstruktors der Elternklasse.
    super(c);
    this.x = x;
    this.y = y;
  }

  void animate () {
    fill(random(255),random(255),random(255));
    // zeichne Kreis
    // setze Füllfarbe auf Standardfarbe zurück
  }
}
```

AUFGABE 2

Der Kreis wird gezeichnet.

```
class ColorCircle extends Animation {
  int x,y;

  public ColorCircle (int x, int y, char c) {
    // Aufruf des Konstruktors der Elternklasse.
    super(c);
    this.x = x;
    this.y = y;
  }

  void animate () {
    fill(random(255),random(255),random(255));
    ellipse(x,y,100,100);
    // setze Füllfarbe auf Standardfarbe zurück
  }
}
```

AUFGABE 2

Die Füllfarbe wird für kommende Elemente auf 255 zurückgesetzt.

```
class ColorCircle extends Animation {
  int x,y;

  public ColorCircle (int x, int y, char c) {
    // Aufruf des Konstruktors der Elternklasse.
    super(c);
    this.x = x;
    this.y = y;
  }

  void animate () {
    fill(random(255),random(255),random(255));
    ellipse(x,y,100,100);
    fill(255);
  }
}
```

ANIMATION-ARRAY

So müssen nur neue **`Animation`**'s im Array hinzugefügt werden, der Rest regelt sich automatisch!

```
Animation[] a;

void setup () {
  size(600,400);
  a = new Animation[]{new RotatingRect(150,200,'1'),new ColorCircle(450,200,'2')};
}

void draw () {
  background(255);
  for (int i=0; i<a.length; i++) {a[i].animate();}
}

void keyPressed () {
  for (int i=0; i<a.length; i++) {a[i].start(key);}
}

void keyReleased () {
  for (int i=0; i<a.length; i++) {a[i].stop(key);}
}
```