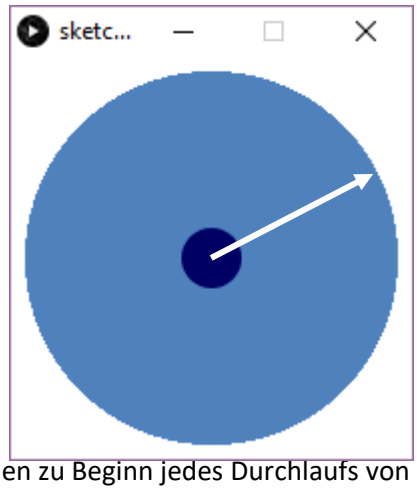
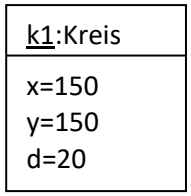
Animationen mit Processing



1. Der Radius eines blauen Kreises soll langsam anwachsen, damit es so aussieht als ob man einen Ballon aufpusten würde (vgl. Abb. rechts).

Für die Animation muss der Durchmesser des vorhandenen Objekts schrittweise verändert werden.

- Wenn in Processing die Funktion **`draw ()`** vorhanden ist, wird sie in bestimmten Zeitabständen (**Intervall**) immer wieder ausgeführt.
  Das Intervall kann mit der Anweisung `frameRate(n)` verändert werden. Dann wird `draw()` n mal pro Sekunde ausgeführt (default ist 60 Frames pro Sekunde).
  Anmerkung: `translate()` und `rotate()` - Anweisungen werden zu Beginn jedes Durchlaufs von `draw()` zurückgesetzt.
  - Verwende als Vorlage das Programm `kreis1` (Vorlagedatei `v07_kreis1`). Das Attribut `linie` wird nicht mehr benötigt, da die Kreislinie nicht gezeichnet werden soll. Nimm die erforderlichen Änderungen vor uns ergänze die Funktion `draw()` wie unten beschrieben.




```
...
void draw(){
  k1.zeichne();
}
class Kreis {
...
  void zeichne() {
    fill(0,0,255);
    noStroke();
    ellipse(x,y,d,d);
    d++;
  }
}
```

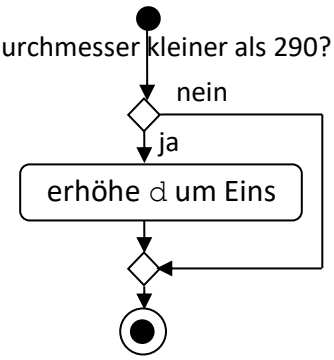
Die Methode `zeichne()` wird jetzt nicht mehr in der Funktion `setup()` aufgerufen, sondern in der Funktion `draw()`.

Der Wert der Variablen `d` wird bei jeder Ausführung der Methode `zeichne ()` um 1 erhöht.

  - Speichere das HTML-Dokument als `kreisani1`. (*vgl. .\262-materialien\kreis\02-kreisani1*)

Einseitige Auswahl

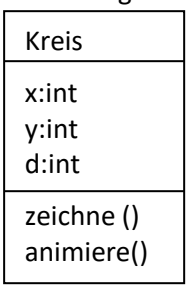
2. Der Durchmesser des Kreises soll nur vergrößert werden, *wenn* er *kleiner als* 290 ist. Dadurch wird der „Ballon“ nicht endlos „aufgepustet“.



Aktivitätsdiagramm für die einseitige Auswahl

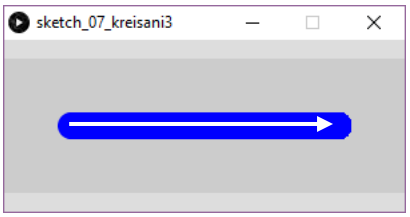
- *Wenn* der Programmablauf auf Grund einer *Bedingung* verzweigt, liegt eine **Auswahlstruktur** vor. Die Schreibweise in Processing für eine solche *wenn-dann*-Beziehung lautet:

```
if (d<290) {
  d++;
}
```



  - Wenn die Zeichnungen und Animationen komplexer werden,ist es übersichtlicher, die Veränderungen an den Attributwerten in einer zusätzlichen Methode vorzunehmen.
  - Ergänze das Programm `kreisani1` gemäß des Klassendiagramms oben: Die Auswahlstruktur soll in die neue Methode `animiere()` eingetragen werden. Speichere als `kreisani2`. (*vgl. .\262-materialien\kreis\03-kreisani2*)

3. Modifiziere das Programm `kreisani2` so, dass der Kreis nicht“aufgepustet”, sondern in Schritten von einem Pixel um 200 Pixel nach rechts verschoben wird.
   Die x-Koordinate des Mittelpunkts soll zu Beginn 50 betragen.
   Speichere die Datei als `kreisani3`.
   (*vgl. .\262-materialien\kreis\04-kreisani3*)



   - Der Inhalt der Zeichenfläche wird zu Beginn eines Durchlaufs der Funktion `draw()` nicht gelöscht. Dadurch entsteht eine Linie mit abgerundeten Enden (siehe Abbildung rechts).

   ➢ Die Anweisung `background()` füllt den gesamten Zeichenbereich mit der angegeben Farbe. Damit kann also die Zeichenfläche gelöscht werden.

   Einen hellgrauen Hintergrund erhältst du mit der Anweisung `background(200);`

   - Ergänze diese Anweisung zu Beginn der Funktion `draw()` (`kreisani4`).
     (*vgl. .\262-materialien\kreis\05-kreisani4*)

4. Zwei Kreise (vgl. Objektdiagramme rechts) sollen in einer Animation schrittweise vergrößert werden.
   Ergänze `kreisani2` und speichere das Programm unter `kreisani5` (Vorlagedatei: `v08_kreisani2`).
   (*vgl. .\262-materialien\kreis\06-kreisani5*)

k1:Kreis
x=150
y=150
d=20
r=0
g=0
b=255

k2:Kreis
x=450
y=150
d=20
r=255
g=0
b=0

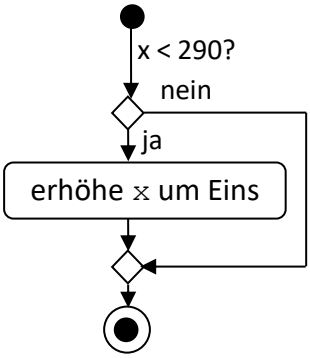
   Hinweise:
   - Das Zeichenfenster muss jetzt 600 Pixel breit sein.
   - Bei objektorientierter Programmierung werden Klassen mit Attributen und Methoden erstellt, aus denen beliebig viele Objekte konstruiert werden können.
     Damit können einfach zwei Objekte einer Klasse erstellt werden.

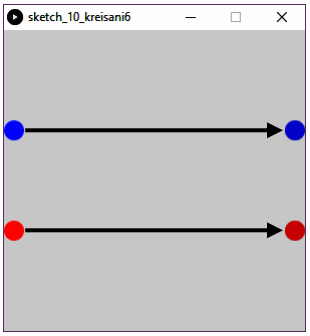
5. Ein blauer und ein roter Kreis, die übereinander am linken Rand des Zeichenfensters „starten“, sollen schrittweise um jeweils ein Pixel bis zum rechten Rand des Zeichenfensters verschoben werden.

   - Erstelle Objektdiagramme und ein Aktivitätsdiagramm für die Methode `animiere()`.

k1:Kreis
x=10
y=100
d=20
r=0
g=0
b=255

K2:Kreis
x=10
y=200
d=20
r=255
g=0
b=0





   - Ergänze `kreisani4` und speichere das Programm unter `kreisani6`.
     (Vorlagedatei: `v09_kreisani4`) (*vgl. .\262-materialien\kreis\07-kreisani6*)

6. Warum kann die Anweisung `background(200);` nicht in der Methode `zeichne()` ergänzt werden, sondern in der Funktion `draw()`? Probiere es im Programm aus.

   Dann würde der Inhalt der Zeichenfläche abwechselnd für beide Objekte gelöscht.

   Dadurch würde immer das jeweils andere Objekt vor dem Zeichnen des nächsten gelöscht.

   Am Schluss wäre dann nur das Objekt `k2` sichtbar.

Zählschleifen

7. Eventuell hast du schon mit einem anderen Programmierwerkzeug wie z B. EOS Animationen erstellt. Wird in EOS ein blau gefüllter Kreis schrittweise vergrößert, damit es so aussieht, als ob man einen Ballon „aufpustet“, kann eine *Wiederholungsstruktur* verwendet werden.
   Hier wird mit der Anweisung `wiederhole` 300 mal schrittweise der Radius des Kreises erhöht (*vgl. Arbeitsblatt 2.6.1 – 04, S. 2: 01-ballon1.eos*):

```
...
ballon1:KREIS
...
wiederhole 300 mal
   ballon1.radiusSetzen(ballon1.radius+1)
*wiederhole
```

Ballon1:KREIS
Mittex=0
Mittey=0
Radius=14
Füllfarbe=blau
Randfarbe=blau

- Eine **Zählschleife** ist eine Kontrollstruktur, mit der eine Sequenz wiederholt ausgeführt und die Anzahl der Wiederholungen angegeben wird.
  - Das funktioniert in Processing nicht: Hier wird die Ausführung des Programms so optimiert, dass das Ergebnis möglichst schnell berechnet und am Bildschirm angezeigt wird.
    Verändert man eine Zeichnung schrittweise mit Hilfe einer Zählschleife, wird mit einer kurzen Verzögerung das Ergebnis der Programmausführung angezeigt
    (Programm siehe `v10_kreis_for`).
    (*vgl. .\262-materialien\kreis\08_kreis_for*)
  - Schreibweise der Zählschleife in Processing mit der Anweisung `for`:

```
...
for (i=d; i<200; i++) {
  ellipse(x,y,i,i);

    for (j=0; j<100; j++)
      println(j);
    }
}
```

- Zunächst wird einer Zählvariablen ein Wert zugewiesen.
- Dann wird die Endebedingung festgelegt.
- Zum Schluss wird angegeben, wie mit der Zählvariablen bei jedem Schleifendurchlauf zu verfahren ist. `i++` bedeutet, dass sie um Eins erhöht wird. Das ist gleichwertig zur Schreibweise `i=i+1`
- Auch eine Verzögerung durch eine sinnlose Wiederholungsstruktur nützt nichts: Es dauert lediglich etwas länger, bis das Ergebnis der Programmausführung angezeigt wird.

- Eine *Zählschleife* wird auch als **For-Schleife** bezeichnet. Die Schreibweise lautet:

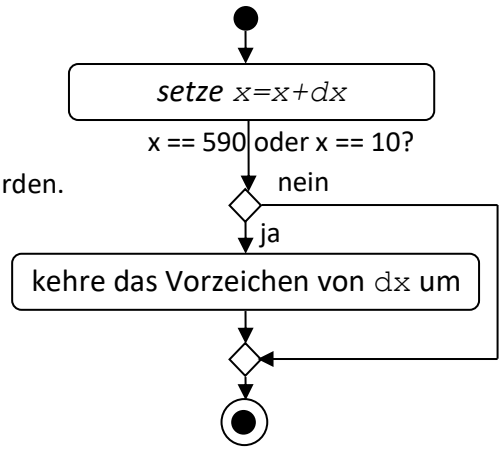
```
for (i=(Wert); (Bedingung, z. B. i<200); i++) {
  (Anweisungen)
}
```

Logische Operationen

8. Der blaue Kreis aus dem Programm `kreisani4` (Vorlagedatei: `v09_kreisani4`) soll immer vom linken Rand des Zeichenfensters zum rechten Rand und zurück verschoben werden.

   Hinweise:
   - Das Zeichenfenster soll 600 Pixel breit und 300 Pixel hoch sein.
   - Verwende die Einstellung `frameRate(500);` um die Bewegung zu beschleunigen.
   - Indem ein ganzzahliges Attribut `dx` (für Delta x) eingeführt wird, kann die Bewegungsrichtung umgekehrt werden: Ist dx gleich +1, wird der Ball schrittweise nach rechts verschoben, ist dx gleich -1, wird er nach links verschoben.
   - Durch welche Anweisung muss jetzt `x++;` ersetzt werden? Ergänze das Aktivitätsdiagramm rechts.
   - Die Bedingungen `(x == 590)` sowie `(x == 10)` können mit dem logischen Operator `oder` verknüpft werden.
     - Die Schreibweise für die `oder`-Verknüpfung ist `||`.
   - Nenne die Anweisung, mit der das Vorzeichen für `dx` umgekehrt werden kann.

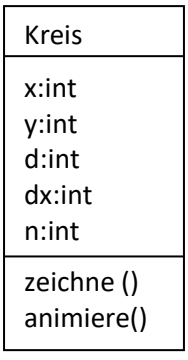
     `dx=-dx;`
   - Ergänze die erforderlichen Anweisungen und speichere das Programm unter `kreisani7`. (*vgl. .\262-materialien\kreis\09-kreisani7*)

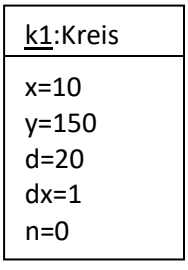


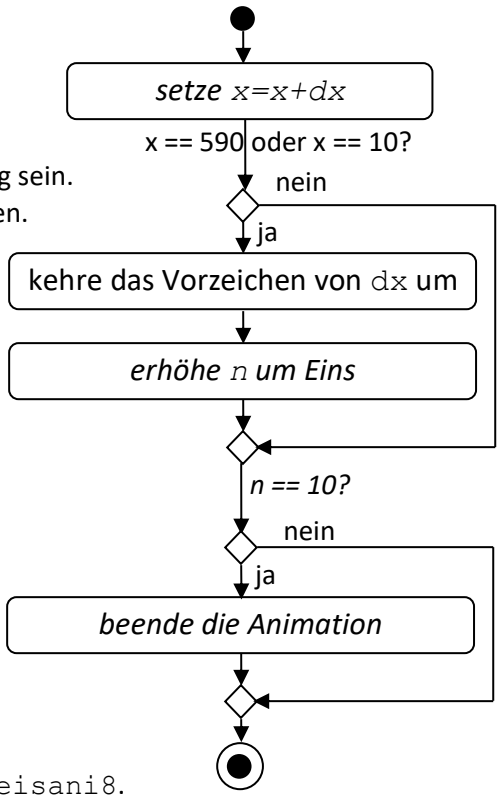
Aktivitätsdiagramm für die Methode animiere()

9. Der „Ball“ soll genau zehnmal hin und her „fliegen“.

   Hinweise:
   - Eine Zählschleife ist für Animationen nicht geeignet. Mit Hilfe eines weiteren Attributs kann das Programm aber mitzählen, wie oft der Ball hin- und hergeflogen ist.
   - Mit der Anweisung `noLoop()` kann die Animation gestoppt werden. Wenn `noLoop()` in der Funktion `setup()` aufgerufen wird, muss es die letzte Anweisung sein. Mit `loop()` kann die Animation wieder gestartet werden.
   - Erstelle ein Klassendiagramm und ein Objektdiagramm für das Objekt `k1`. Ergänze das Aktivitätsdiagramm rechts für die Methode `animiere()`.







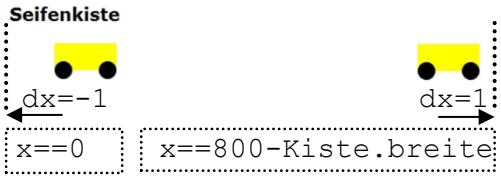
   - Codiere das Modell und speichere das Programm als `kreisani8`. (Vorlagedatei: `v11_kreisani7`) (*vgl. .\262-materialien\kreis\10-kreisani8*)

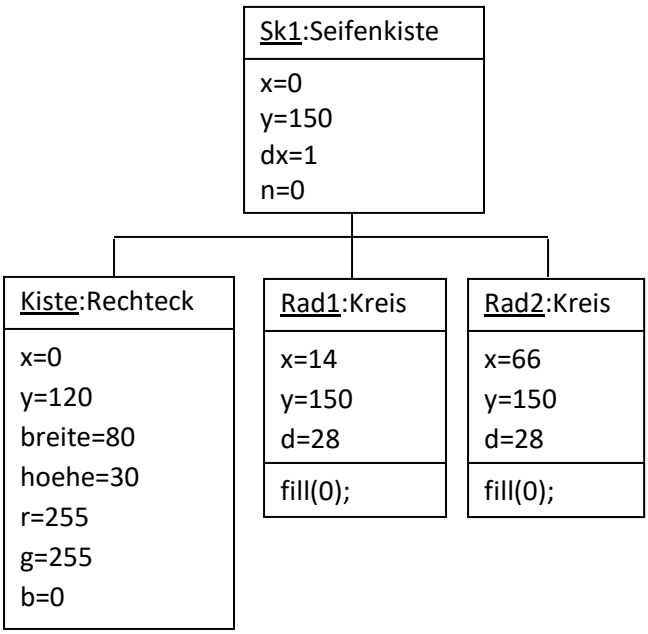
10. In dem Objektdiagramm unten sind die Attributwerte für eine Seifenkiste gegeben. Diese Seifenkiste soll in einem Processing-Programm gezeichnet und animiert werden, indem sie viermal vom linken zum rechten Rand des Zeichenfensters und zurück bewegt wird.

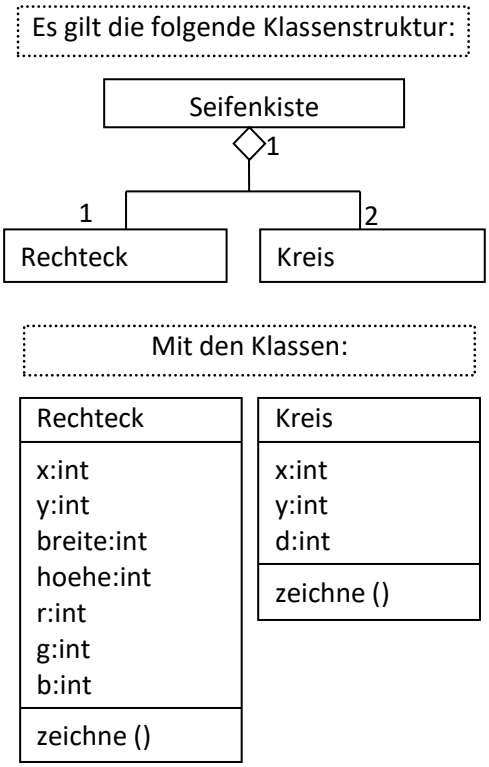
Seifenkiste
x:int y:int dx:int n:int Kiste:Rechteck Rad1:Kreis Rad2:Kreis
animiere ()

Hinweise:

- Zur Programmstruktur kannst du dich in den Programmversionen zu dem Vorfahrtszeichen orientieren (vgl. Arbeitsblatt 03). Skizziere die Seifenkiste wie du es von dem Vorfahrtszeichen her kennst. Als Referenzpunkt (x|y) ist die linke obere Ecke der Kiste gut geeignet.
- Zu der Animation ist die Vorlagendatei `v12_kreisani8` eine gute Orientierungshilfe. Evtl. kannst du auch Programmcode direkt per *copy-paste* verwenden.
- Gliedere dein Vorhaben in Teilschritte auf:
  - Erstelle zuerst die Klasse Seifenkiste, die nur aus zwei schwarzen Kreisen für die Räder besteht und noch nicht animiert wird. (*.\262-materialien\seifenkiste\01_seifenkiste1v1*)
  - Ergänze ein gelbes Rechteck für die Kiste. Anmerkung: Die Farbe Gelb wird aus den RGB-Werten (255,255,0) erzeugt. (*.\262-materialien\seifenkiste\02_seifenkiste1v2*)
  - Dann kann die Seifenkiste animiert werden. (*vgl. .\262-materialien\seifenkiste\03_seifenkiste1*)





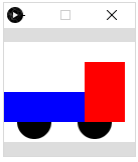


Für die Objekte in Abhängigkeit von x und y gilt:

- `Kiste`: $x_{LinksOben} = x$ *und* $y_{LinksOben} = y - 30$
- `Rad1`: $x_{Mittelpunkt} = x+14$ *und* $y_{Mittelpunkt} = y$
- `Rad2`: $x_{Mittelpunkt} = x+66$ *und* $y_{Mittelpunkt} = y$

Probiere, die Aufgabe selbstständig zu bearbeiten. Wenn du dir das noch nicht zutraust – oder um Details nachzusehen, findest du auf der folgenden Seite einen Lösungsvorschlag.

11. Zusatzaufgabe: Gestalte die Seifenkiste ansprechender, indem du weitere Objekte ergänzt. Zum Beispiel könnte man einen LKW zeichnen. (*Vorschlag vgl. .\262-materialien\seifenkiste\LKW1*)



```
Seifenkiste Sk1;
void setup() {
  size(800,300);
  frameRate(500);
  Sk1=new Seifenkiste(0,150,1); }
void draw(){
  background(255);
  Sk1.animiere(); }
class Seifenkiste {
  int x;
  int y;
  int dx;
  int n;
  Rechteck Kiste;
  Kreis Rad1;
  Kreis Rad2;
  Seifenkiste(int px,int py,int pdx) {
    x=px;
    y=py;
    dx=pdx;
    n=0;
    Kiste=new Rechteck(x,y-30,80,30,255,255,0);
    Rad1=new Kreis(x+14,y,28);
    Rad2=new Kreis(x+66,y,28);
  }
  void animiere() {
    Kiste.zeichne();
    Rad1.zeichne();
    Rad2.zeichne();
    Kiste.x=Kiste.x+dx;
    Rad1.x=Rad1.x+dx;
    Rad2.x=Rad2.x+dx;
    if (Kiste.x==800-Kiste.breite || Kiste.x==0) {
      dx=-dx;
      n++; }
    if (n==8) {
      noLoop();
    }
  }
}
class Rechteck {
  int x;
  int y;
  int breite;
  int hoehe;
  int r;
  int g;
  int b;
  Rechteck(int px,int py,int pbreite,
  int phoehe, int pr,int pg,int pb) {
    x=px;
    y=py;
    breite=pbreite;
    hoehe=phoehe;
    r=pr;
    g=pg;
    b=pb;
  }
  void zeichne() {
    fill(r,g,b);
    noStroke();
    rect(x,y,breite,hoehe);
  }
}
```

Seifenkiste
x:int y:int dx:int n:int Kiste:Rechteck Rad1:Kreis Rad2:Kreis
animiere ()

Rad2:Kreis	Rad1:Kreis
x=66 y=150 d=28	x=14 y=150 d=28

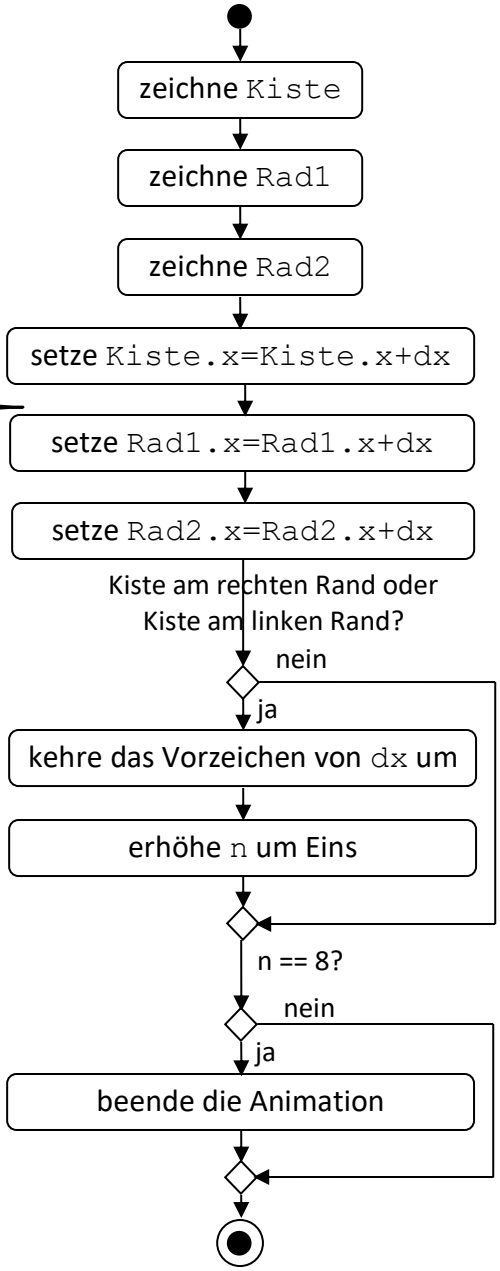
Kiste:Rechteck
x=0 y=120 breite=80 hoehe=30 r=255 g=255 b=0

Eine Zeile

Rechteck
x:int y:int breite:int hoehe:int r:int g:int b:int
zeichne ()

```
class Kreis {
  int x;
  int y;
  int d;
  Kreis(int px,int py,int pd) {
    x=px;
    y=py;
    d=pd;
  }
  void zeichne() {
    fill(0);
    noStroke();
    ellipse(x,y,d,d);
  }
}
```

Kreis
x:int y:int d:int
zeichne ()



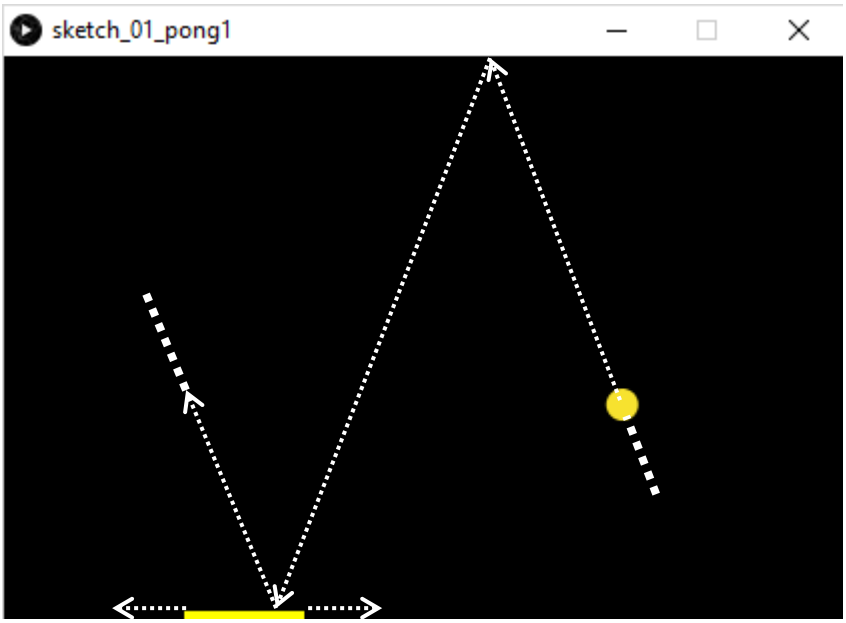
Aktivitätsdiagramm für die Methode animiere()

Interrupt Request und Polling

Bei modernen Computern besteht die Möglichkeit, dass Peripheriegeräte eine *Unterbrechungsanforderung* senden (engl. *Interrupt Request*). Dadurch wird das laufende Programm unterbrochen, um einen anderen Vorgang abzuarbeiten. Danach wird das ursprüngliche Programm wieder fortgesetzt. Zum Beispiel sendet die Tastatur einen Interrupt Request, wenn eine Taste gedrückt wurde.
Programmiersprachen stellen dafür einen *Interrupt Handler* zur Verfügung. In Processing wird die Funktion **`keyPressed()`** aufgerufen, wenn eine Taste gedrückt wurde – sofern die Funktion vorhanden ist.
Wird die Taste wieder losgelassen, wird die Funktion **`keyReleased()`** aufgerufen.
Das Zeichen der betreffenden Taste wird in der Variablen **`keyCode`** gespeichert.
Wird beispielsweise die Taste <A> gedrückt, erhält die Variable `keyCode` den Wert `'a'`.
Zur Steuerung eines Pong-Schlägers bieten sich die Pfeiltasten auf der Tastatur an. Diese entsprechen den Werten `LEFT` und `RIGHT`.

Die beiden folgenden Funktionen stellen dem Objekt `p1` der Klasse `Pong` in den Attributen `links` und `rechts` vom Datentyp `boolean` die Information zur Verfügung, ob eine Pfeiltaste gedrückt wurde.

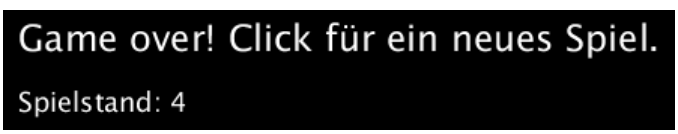
```
void keyPressed() {
 if (keyCode == LEFT) {
   p1.links = true;
 }
 if (keyCode == RIGHT) {
   p1.rechts = true;
 }
}
void keyReleased() {
 if (keyCode == LEFT) {
   p1.links = false;
 }
 if (keyCode==RIGHT) {
   p1.rechts = false;
 }
}
```



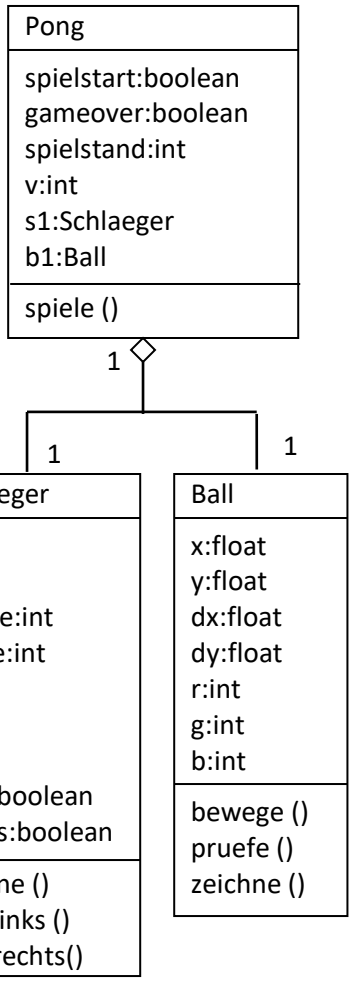
Hinweise:

- Damit der Programmcode etwas weniger Zeilen benötigt, kann die Deklaration von Attributen eines Datentyps auch aufzählend erfolgen, z. B.: `boolean gameover, links, rechts;`
- Um das Spiel später mittels eines zweiten Schlägers und eines weiteren Balls ausbauen zu können, werden der Klasse `Schlaeger` alle Attributwerte als Parameter übergeben, der Klasse `Ball` die Füllfarbe.
- Das Spiel sollte mit der Framerate 200 gestartet werden, die bei jedem Treffer um 10 erhöht wird. Dafür ist das Attribut `v` vorgesehen.
- Wenn die Reaktionsgeschwindigkeit auf ein Ereignis nicht so wichtig ist, kann man Wertänderungen auch innerhalb des Programms durch zyklisches Abfragen von Werten ermitteln.
  Diese Vorgehensweise nennt man **Polling**.
  Eine denkbare Anwendung ist die Abfrage eines Mausklicks um ein neues Spiel zu beginnen, z. B.:

```
if (mousePressed) {
  // Beginne neues Spiel
}
```



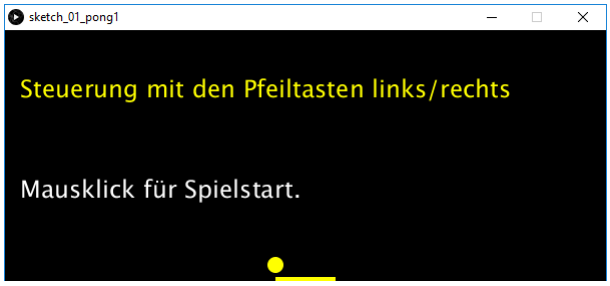
12. Gegeben ist das Klassendiagramm für die Klasse `Pong` und ein Vorschlag für die Objekte.
Entwickle den Programmcode zu dem Spiel Pong (`pong1`).
(*vgl. .\262-materialien\pong\01_pong1*)



Struktur des Pong-Spiels
In einem Klassendiagramm

Hinweise:

- Beginne mit einer Programmversion, in der nur der Schläger hin- und herbewegt werden kann.
- Entwickle dein eigenes Spiel. Die Diagramme rechts und unten dienen als Anregung.
- Nach dem Starten des Programms könnte zuerst ein Startbildschirm eingeblendet werden:



- Für Textausgaben im Zeichenfenster steht die Anweisung text("Text",x,y); zur Verfügung (vgl. Lerninhalte S. 6).
- Der Ball prallt an den Spielfeldrändern links, rechts und oben ab. Hier muss also das Vorzeichen für die Bewegungsrichtung in den Attributen `dx` bzw. `dy` umgekehrt werden (`Ball.pruefe()`).
Am Spielfeldrand unten prallt der Ball nur ab, wenn er den Schläger trifft (`Pong.spiele()`).
- Bei Attributen des Datentyps `boolean` benötigt man keinen Vergleich mit ==, denn sie liefern einen Wahrheitswert (`true` oder `false`).
Z. B. `if (!gameover)` (zu lesen als „if not gameover“) ist gleichbedeutend mit `if (gameover==false)`.
- Die Anzahl der Treffer wird in dem Attribut `spielstand` mitgezählt und nach Spielende mit einem geeigneten Kommentar angezeigt (z. B. „Game over!“).