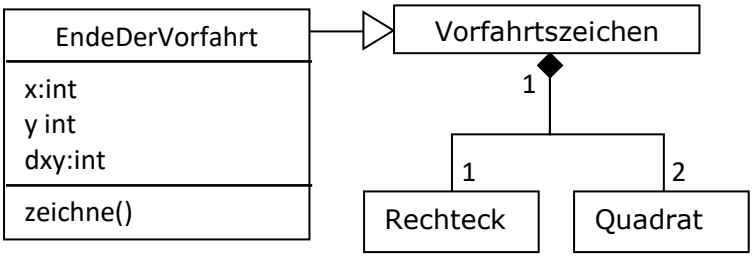
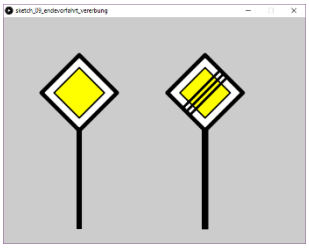
Weitergehende Konzepte der Objektorientierten Programmierung

Vererbung

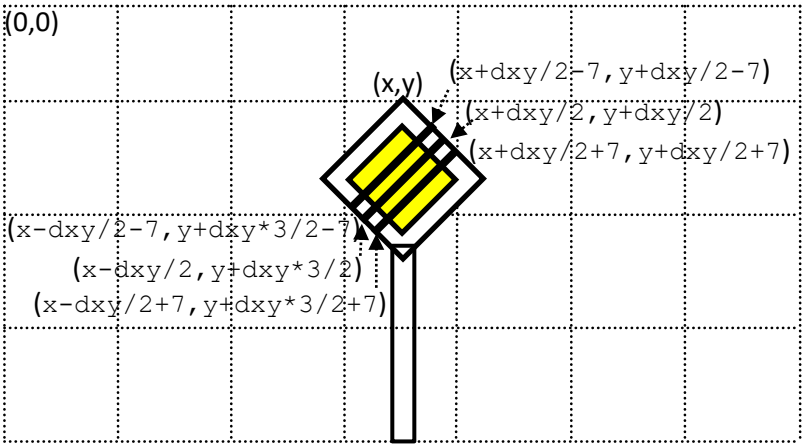
Vorfahrtsschild
x:int p1:Pfosten q1:Quadrat q2:Quadrat
zeichneVorfahrt ()

1. Ergänze in dem Programm „Vorfahrtsschild“ ein zusätzliches Attribut `x:int` (Vorlagedatei: `v18_vorfahrtsschild_komposition`). Dadurch kann in dem Konstruktor für das Objekt `v1:Vorfahrtsschild` der Wert 150 übergeben werden. Außerdem muss die x-Koordinate des Pfostens mit `x-5` angegeben werden und für die beiden Quadrate mit `x`. Speichere das Programm unter `vorfahrtsschild_xschild`. (*vgl. .\262-materialien\vorfahrtszeichen\08_vorfahrtsschild_xschild*)
2. Jetzt kann das Programm so ergänzt werden, dass mit Hilfe der Vererbungsbeziehung unten das Verkehrszeichen „Ende der Vorfahrtsstraße“ rechts neben dem Vorfahrtszeichen gezeichnet wird. Hinweis: Das `Zeichenfenster` sollte 600 Pixel breit sein.





Für das neue Verkehrszeichen müssen zusätzlich drei Linien gezeichnet werden.
Die Koordinaten in Abhängigkeit des oberen Eckpunkts sind in der Skizze schon angegeben.



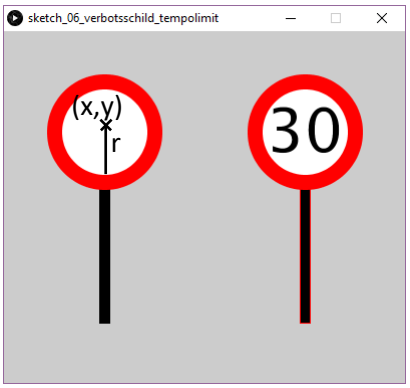
- Ergänze in dem Programm `vorfahrtsschild_xschild` die Klasse `EndeVorfahrt` mit der Vererbungsbeziehung wie unten beschrieben.
- Speichere das Programm unter dem Dateinamen „endevorfahrt_vererbung“. (*vgl. .\262-materialien\vorfahrtszeichen\09_endevorfahrt_vererbung*)

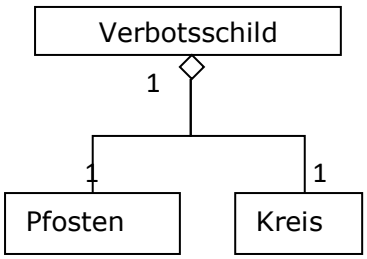
```
class EndeVorfahrt extends Vorfahrtsschild {
  int x;
  int y;
  int dxy;
  EndeVorfahrt(int px,int py,int pdxy) {
    super(400);
    x=px;
    y=py;
    dxy=pdxy;
  }
  void zeichne() {
    strokeWeight(5);
    fill(0,0,0);
    line( ...
    // Vervollständige die Zeichenbefehle für die drei Linien.
} }
```

Die Vererbungsbeziehung zu der allgemeineren Klasse wird mit der Anweisung `extends` hergestellt.

Auf die „Elternklasse“ wird mit `super()` zugegriffen. Damit wird der Konstruktor der allgemeineren Klasse aufgerufen. Diese Anweisung muss deshalb die erste im Konstruktor der spezielleren Klasse sein.
Denke auch daran, in der Funktion `setup()` ein Objekt der Klasse `EndeVorfahrt` zu erstellen!

3. Ergänze das Programm `kreis1` (Vorlagedatei `v07_kreis1`), um ein Verkehrsschild „Verbot für Fahrzeuge aller Art“ in Abhängigkeit des Mittelpunkts (x|y) des Kreises mit einem Pfosten zu zeichnen (das linke Verkehrszeichen in der Abbildung rechts). Speichere das Programm als `verbotsschild`.

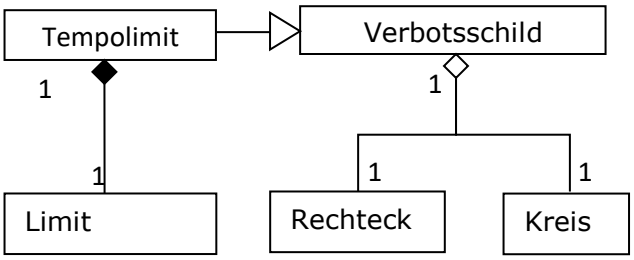




Klassenstruktur zu dem Verkehrszeichen „Verbot für Fahrzeuge aller Art“

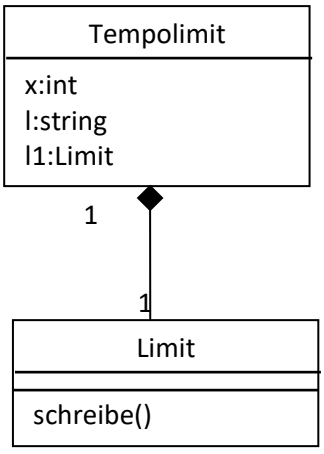
Hinweis: Für diese und die nächste Aufgabe kannst du Programmcode aus dem Programm `endevorfahrt_vererbung` verwenden.
(*vgl. .\262-materialien\verkehrszeichen\05-verbotsschild*)

4. Das Programm `verbotsschild` (Vorlagedatei `v19_verbotsschild`) kann mit einer Vererbungsbeziehung wie in `endevorfahrt` so ergänzt werden, dass rechts daneben das Verkehrszeichen „Tempolimit“ gezeichnet wird.
(*vgl. .\262-materialien\verkehrszeichen\06-verbotsschild_tempolimit*)



Lösungsvorschlag:

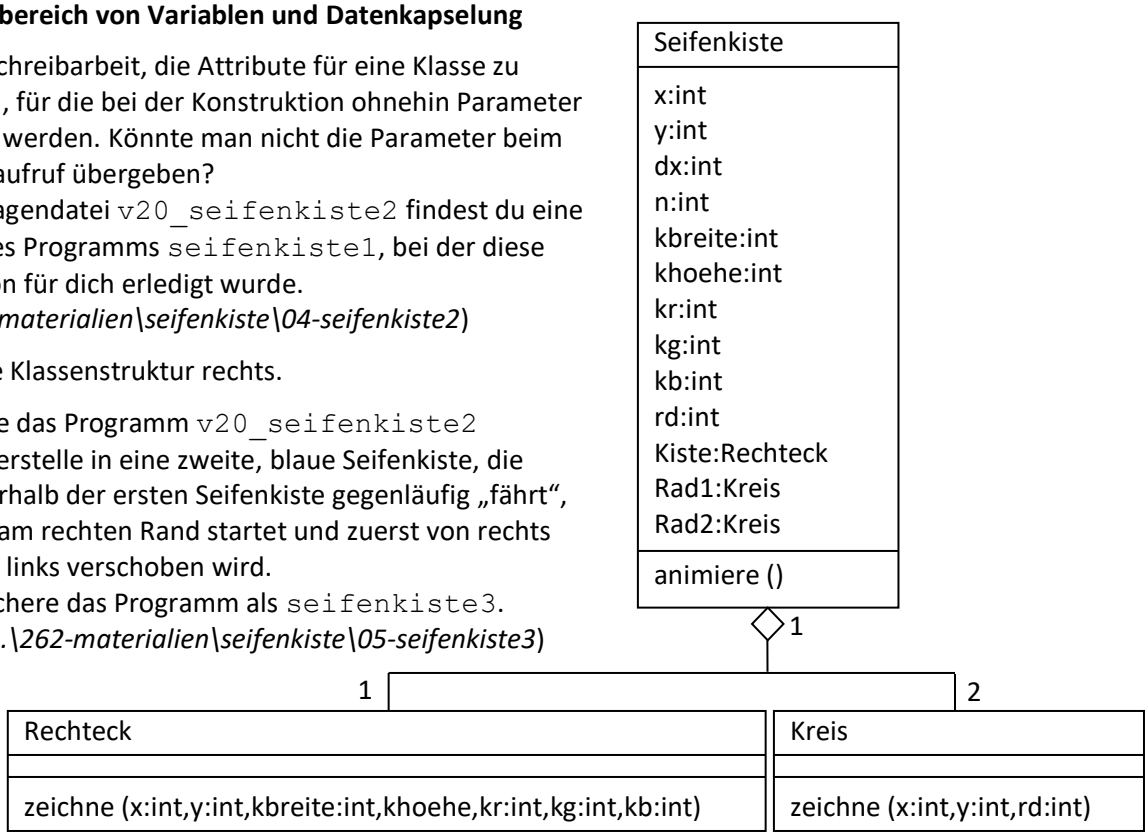
```
Verbotsschild v1;
Tempolimit t1;
void setup() {
  size(400,350);
  v1=new Verbotsschild(100);
  t1=new Tempolimit(300,"30");
}...
class Tempolimit extends Verbotsschild {
  int x;
  String l;
  Tempolimit(int px,String pl) {
    super(300);
    x=px;
    l=pl;
    Limit l1;
    l1=new Limit();
    l1.schreibe();
  }
  class Limit { // Limit ist Komposition
    Limit() {
    }
    void schreibe() {
      fill(0,0,0);
      textAlign(CENTER);
      textSize(60);
      text(l,x,120);
    }
  }
}
```



Gültigkeitsbereich von Variablen und Datenkapselung

Es ist viel Schreibarbeit, die Attribute für eine Klasse zu deklarieren, für die bei der Konstruktion ohnehin Parameter übergeben werden. Könnte man nicht die Parameter beim Methodenaufruf übergeben?
In der Vorlagendatei `v20_seifenkiste2` findest du eine Variante des Programms `seifenkiste1`, bei der diese Arbeit schon für dich erledigt wurde.
(*vgl. .\262-materialien\seifenkiste\04-seifenkiste2*)

Hier gilt die Klassenstruktur rechts.



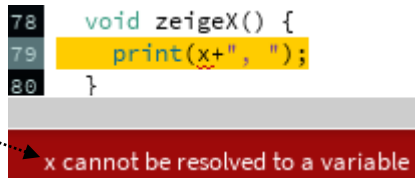
5. Öffne das Programm `v20_seifenkiste2` und erstelle in eine zweite, blaue Seifenkiste, die unterhalb der ersten Seifenkiste gegenläufig „fährt“, also am rechten Rand startet und zuerst von rechts nach links verschoben wird.
   Speichere das Programm als `seifenkiste3`.
   (*vgl. .\262-materialien\seifenkiste\05-seifenkiste3*)

   Hinweis: Die Attribute `kr`, `kg` und `kb` stehen für die Farbwerte R, G, B der Kiste. Das Attribut `rd` ist für den Durchmesser der Räder vorgesehen.

➢ Hinweis zur Softwareentwicklung: Sollte sich ein Programm nicht wie vorgesehen verhalten oder möchte man wichtige Stellen im Programmablauf genauer betrachten, kann man sich mit einer `print()`- bzw. `println()`-Anweisung Attributwerte in der Konsole anzeigen lassen.

6. In der Version *seifenkiste4v1* werden in der Methode `zeigeX()` die Werte für die Variable `x` der Räder in der Konsole angezeigt, wenn die Bewegungsrichtung der Seifenkiste umgekehrt wird.
   (Vorlagendatei `v21_seifenkiste4v1`)
   (*vgl. .\262-materialien\seifenkiste\06-seifenkiste4v1*)

   ○ Die Programmausführung wird mit der Fehlermeldung „x cannot be resolved to a variable” abgebrochen.



   ➢ Wird eine Variable innerhalb einer Methode deklariert, gilt diese Variable nur in dieser Methode. Andere Klassen oder Methoden können nicht darauf zugreifen.
   Dieses Prinzip nennt man **Datenkapselung**.
   Eine solche Variable nennt man **Lokale Variable**.
   Dadurch wird verhindert, dass der Wert einer Variablen versehentlich an einer anderen Stelle im Programm verändert werden kann.

Anmerkung: Man kann Variablen auch ganz zu Beginn des Programms deklarieren.
Variablen, die außerhalb der Klassenstruktur deklariert werden, gelten in allen Klassen (**Globale Variable**).

➢ Man sollte aber die Verwendung globaler Variablen vermeiden, weil diese versehentlich mehrfach genutzt werden können und dadurch falsche Ergebnisse entstehen könnten.

Rückgabewert

- Methoden haben keinen Zugriff auf die Attributwerte anderer Methoden. Man kann mit der Anweisung `return` einen Wert an die aufrufende Methode zurückgeben.

7. Ändere das Programm in der Vorlagendatei `v21_seifenkiste4v1` wie in den Lerninhalten 02, S.2 beschrieben ab.
(*vgl. .\262-materialien\seifenkiste\07-seifenkiste4*)

   Beachte die zugehörigen Klassendiagramme rechts.
   Die Methode zeichne () erhält einen **Datentyp für den Rückgabewert**.

Kreis
zeichne (x:int,y:int,rd:int)**:int** **zeigeX (x:int)**

Seifenkiste
x:int y:int dx:int n:int kbreite:int khoehe:int kr:int kg:int kb:int rd:int **xRad:int** Kiste:Rechteck Rad1:Kreis Rad2:Kreis
animiere ()

8. Gib die Werte für die Mittelpunkte der Räder an, wenn die Seifenkiste umkehrt: *785 bzw. 67.*

9. Der Radius der Kreise beträgt 14. Die Seifenkiste müsste also umkehren, wenn die Mittelpunkte bei 786 bzw. 66 sind. Liegt ein Programmfehler vor?

   Nein, denn der Wert von `x` wird nach dem Zeichnen aktualisiert.

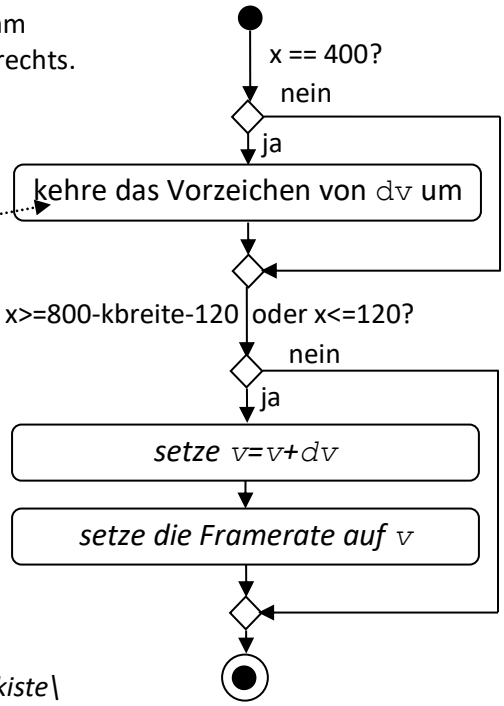
   Gezeichnet wird noch mit dem vorigen Wert, also mit 786 bzw. 66.

10. Zusatzaufgabe: Die Seifenkisten sollen langsam beschleunigen und vor dem Umkehren wieder abbremsen.

    Hinweis: Die Methode `animiere()` wird komplizierter und damit unübersichtlicher. Deshalb ist es sinnvoll, drei Methoden zu ergänzen:
    - Für das Umkehren der Bewegungsrichtung die Methode `wende()`.
    - Die Methode `beende()`.
    - Das Anfahren und Abbremsen wird in der Methode `bremse()` implementiert. Dafür werden auch die Attribute `v` und `dv` eingeführt.
    - Ergänze das Aktivitätsdiagramm für die Methode `bremse()` rechts.

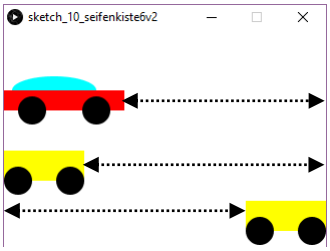
Seifenkiste
v:int **dv:int** x:int y:int dx:int n:int kbreite:int khoehe:int kr:int kg:int kb:int rd:int xRad:int Kiste:Rechteck Rad1:Kreis Rad2:Kreis
animiere () **wende()** **beende()** **bremse()**

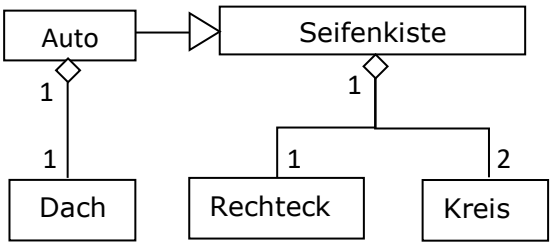
Nachdem die Seifenkiste beschleunigt hat, muss als nächstes gebremst werden. Dafür wird das Vorzeichen von `dv` etwa zur Mitte des Fensters umgekehrt (`dv` wird hier negativ). Zum Beschleunigen muss das Vorzeichen von `dv` positiv sein, also nochmals umgekehrt werden. Das kann in in der Methode `wende()` erledigt werden.



    - Speichere das Programm als `seifenkiste5`.
(*vgl. .\262-materialien\seifenkiste\08-seifenkiste5*)

11. Das Programm `Seifenkiste` kann als Basis verwendet werden, um durch Aufsetzen eines Dachs mit der Farbe Cyan als „Führerhaus“ ein Auto zu zeichnen. Das Auto soll oberhalb der Seifenkisten doppelt so schnell fahren. Außerdem könnte sich das Auto von den beiden Seifenkisten dadurch unterscheiden, dass die Karosserie 40 Pixel länger und 10 Pixel niedriger sowie rot ist.
In der Vorlagedatei `sketch_v22_seifenkiste6vorlage` ist der Aufruf `anmimiere()` für die jeweiligen Objekte innerhalb der Funktion `setup()` eingetragen, weshalb die Objekte nicht verschoben werden.
So kannst du in einem ersten Schritt sicherstellen, dass die Objekte korrekt gezeichnet werden.

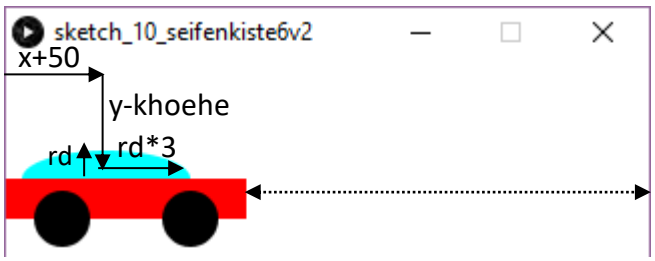




   - Ergänze das Auto nach der in den Lerninhalten beschriebenen Klassenstruktur rechts um eine Vererbungsbeziehung, zunächst ohne das Dach. (*vgl. .\262-materialien\seifenkiste\09-seifenkiste6v1*)

   Hinweise:
   - Die Syntax für eine Vererbungsbeziehung ist auf Seite 1 beschrieben. Du kannst auch z. B. das Programm `endevorfahrt_vererbung` zu Hilfe nehmen. (*vgl. .\262-materialien\vorfahrtszeichen\09_endevorfahrt_vererbung*)
   - Die Deklaration des Objekts `d1` der Klasse `Dach` sollte global zu Beginn des Programms erfolgen. (*vgl. .\262-materialien\seifenkiste\10-seifenkiste6v2*)
   - Wenn du diese Anweisungen in die Methode `draw()` verschiebst, bewegen sich die Objekte.

   Die Methode `Seifenkiste.animiere()` soll in der Klasse `Auto` überschrieben werden: Am einfachsten kopierst du die Methode und ergänzt den Methodenaufruf zum zeichnen des Dachs. Dieses kann mit Hilfe der vorhandenen Parameter gezeichnet werden (vgl. Skizze rechts).

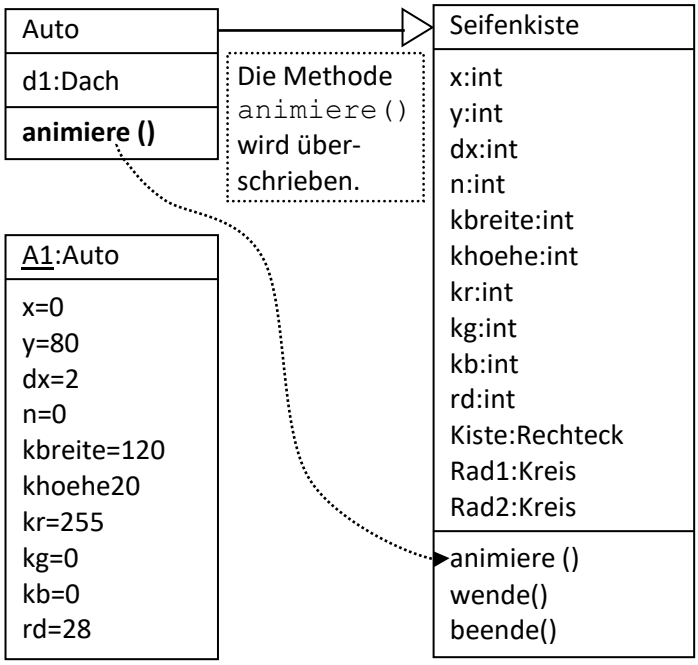


   - Speichere die Datei als `seifenkiste6`. (*vgl. .\262-materialien\seifenkiste\11-seifenkiste6*)

12. Die Seifenkisten bleiben mitten in der Bewegung stehen. Sie sollen bis zum Ende fahren (`seifenkiste7`). (*vgl. .\262-materialien\seifenkiste\12-seifenkiste7*)

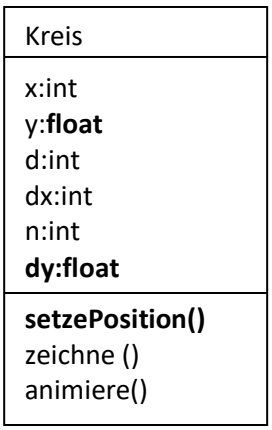
13. Zusatzaufgabe: Das Auto soll wirklich wenden (In Fahrtrichtung vorwärts sitzt das Führerhaus weiter hinten). Speichere die Datei als `seifenkiste8`. (*vgl. .\262-materialien\seifenkiste\13-seifenkiste8*)

14. Zusatzaufgabe: Ergänze Buttons `Start` und `Stop` (`seifenkiste9`). (*vgl. .\262-materialien\seifenkiste\14-seifenkiste9*)



15. In dem Programm `kreisani8` (vgl. Arbeitsblatt 1.8–04, S. 4) „fliegt“ein blauer Kreis zehnmal von links nach rechts und wieder zurück (*vgl. .\262-materialien\kreis\13-kreisani8*).
    - Der Kreis soll auch von oben nach unten fliegen und am Fensterrand „abbprallen“. Verwende die Vorlagedatei `v12_kreisani8` und speichere das Programm als `kreisani9`.
    - Ergänze das Objektdiagramm für das Zeichenfenster rechts unten.
    - Gib im Aktivitätsdiagramm für die Methode `animiere()` ganz unten die Bedingung an, wann der Kreis oben und unten „abprallen“ soll.
    - Codiere die Ergänzungen in Processing. (*vgl. .\262-materialien\kreis\14_kreisani9*)

    Hinweise:
    - Die Bewegung von unten nach oben findet in y-Richtung statt. Dafür wird das Attribut `dy` (für Delta y) eingeführt.
    - Die vertikale Bewegung soll langsamer sein als die horizontale, weshalb für `dy` der Wert 0,5 zugewiesen wird. Deshalb wird für `y` und `dy` der Datentyp `float` (*Fließkommazahl*) verwendet.
    - Die Werte für `y` und `dy` sollen nicht in dem Konstruktor der Klasse `Kreis` gesetzt werden, sondern in einer neuen Methode `setzePosition()`.



16. Ergänze einen roten und einen grünen Kreis.
    Die Objekte sollen in einem Array gespeichert werden. (siehe Arbeitsblatt 05, S. 2 bzw. `gluecksrad2`)
    Die Startposition `y` ist eine Zufallszahl zwischen 10 und 290, der Wert für `dy` soll zufällig im Intervall von -1 bis +1 festgelegt werden. Speichere das Programm als `kreisani10`.
    (*vgl. .\262-materialien\kreis\15_kreisani10*)

17. Ein weiterer Kreis mit dem Durchmesser 40 und der Farbe Magenta soll zu Beginn in der Mitte des Zeichenfensters mit doppelter Geschwindigkeit nach links starten (`kreisani11`).
    (Vorlagedatei: `v23_kreisani10`)
    (*vgl. .\262-materialien\kreis\17_kreisani11*)

    Hinweise:
    - Die Schreibweise für die Vererbungsbeziehung kannst du z. B. aus dem Programm `seifenkiste6` übernehmen. (*vgl. .\262-materialien\seifenkiste\11_seifenkiste6*)
    - Zuerst solltest du aber die Methode `animiere()` so verändern, dass die Kreise nicht mehr bei `590` bzw. `290` und `10` abprallen, sondern bei `600-d/2` bzw. `300-d/2` und `d/2`. (*vgl. .\262-materialien\kreis\16_kreisani11v1*)

18. Der große Kreis soll von den anderen Kreisen abprallen, indem bei einer Berührung die Werte für `dx` und `dy` umgekehrt werden (`kreisani12`).
    Reduziere die Framerate auf 100, um das besser beobachten zu können.
    (*vgl. .\262-materialien\kreis\18_kreisani12*)
    Hinweis: Damit die überschriebene Methode `animiere()` nicht zu komplex wird, ist es sinnvoll, eine weitere Methode einzuführen, z. B. `pruefeKollision()`.