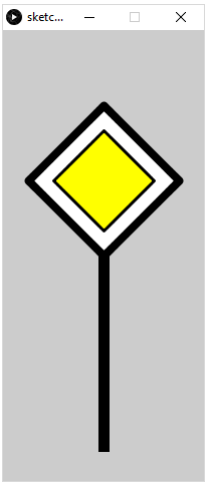
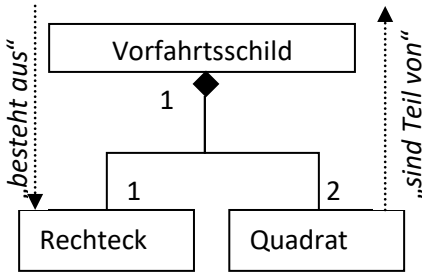
Weitergehende Konzepte der Objektorientierten Programmierung

Vererbung

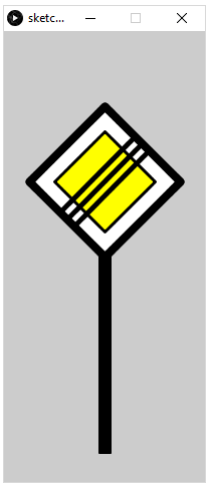
In dem Klassendiagramm rechts ist die Struktur eines Vorfahrtsschilds dargestellt (vgl. Lerninhalte 01 S. 8).



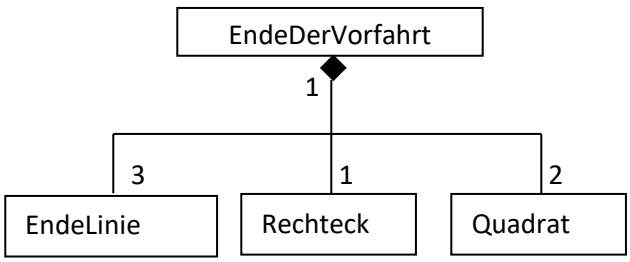


Zusammensetzung (Struktur) des Vorfahrtszeichens in einem Klassendiagramm

Das Vorfahrtszeichen kann so ergänzt werden, dass mit Hilfe dreier Linien das Verkehrsschild „Ende der Vorfahrtsstraße“ gezeichnet wird.

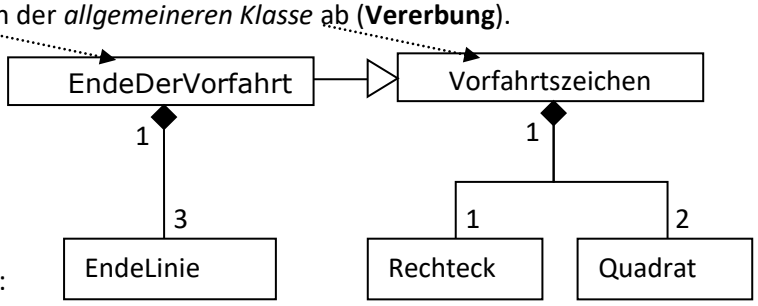


Bisher wurden Ergänzungen in einer vorhandenen Klassenstruktur durch *Aggregation* oder *Komposition* implementiert.
Ein mögliches Klassendiagramm dazu könnte so aussehen:



Damit ist aber das ursprüngliche Verkehrszeichen alleine nicht mehr verfügbar.
Man kann auch vorhandene Klassen nutzen und ergänzen. So bleibt das Vorfahrtszeichen erhalten und kann zusätzlich für das Verkehrszeichen „Ende der Vorfahrtsstraße“ genutzt werden:

- Neue Klassen können aus vorhandenen Klassen abgeleitet werden.
  Damit stammt eine *speziellere Klasse* von der *allgemeineren Klasse* ab (**Vererbung**).
  Die spezielle Klasse erbt die Attribute und Methoden von der „Elternklasse“.
  In der UML ist der Pfeil  das Symbol für die Vererbung, wobei die Pfeilspitze auf die allgemeinere Klasse zeigt.



  Erläuterung des Klassendiagramms dazu:
  Auf die Klasse, die die Rolle des „Elternteils“ hat, zeigt als Pfeilspitze ein Dreieck.
  Von der Klasse mit der Rolle „Kind“ – die speziellere Klasse, die von der allgemeineren Klasse Attribute und Methoden erbt – geht der Pfeil aus.

  A Bearbeite das Arbeitsblatt 07, S. 1-2: Vererbung

Gültigkeitsbereich von Variablen und Datenkapselung

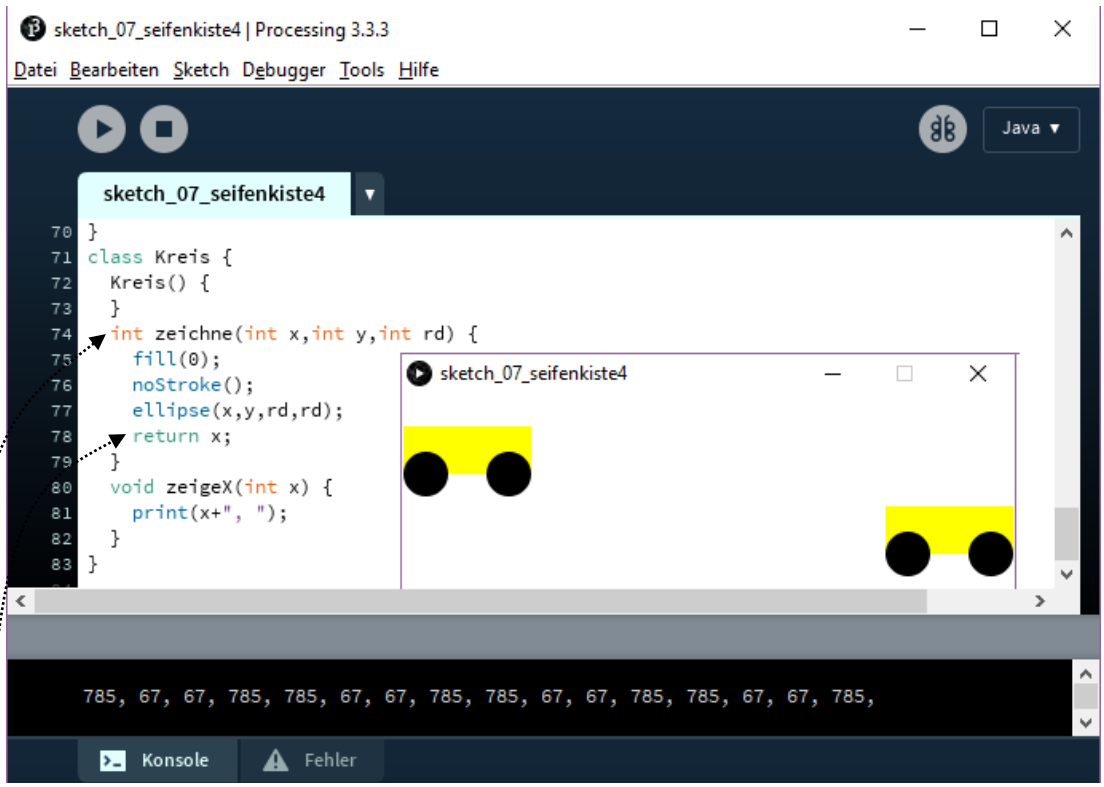
- Bearbeite das Arbeitsblatt 07, S. 3

➢ 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



Im Beispiel werden in der Methode `zeichne()` mit dem Wert der Variablen `x` die Mittelpunkte der Kreise für die Räder der Seifenkiste festgelegt. Um auf diesen Wert auch in der Methode `zeigeX()` zugreifen zu können, muss er mit `return` als Rückgabewert an die aufrufende Methode zurückgegeben und dann als Parameter beim Aufruf der Methode `zeigeX()` übergeben werden.

Weil die Methode `zeichne()` jetzt einen Wert zurückgibt, muss bei der Deklaration der Datentyp des Rückgabewerts (`int`) anstatt der Angabe `void` verwendet werden.

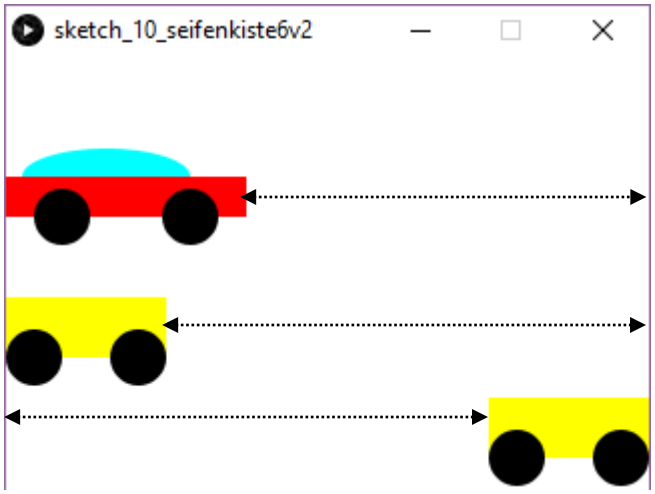
In der aufrufenden Methode Seifenkiste.animiere() wird der Rückgabewert auf ein Attribut zugewiesen:

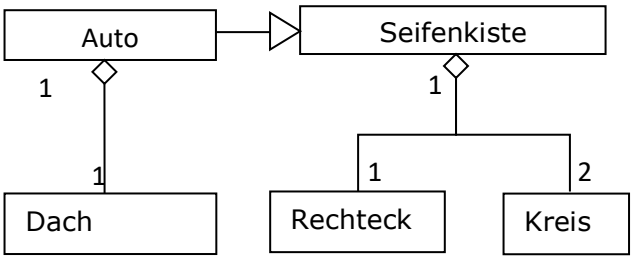
```
void animiere() {
  Kiste.zeichne(x,y-khoehe,kbreite,khoehe,kr,kg,kb);
  xRad=Rad1.zeichne(x+rd/2,y,rd);
  xRad=Rad2.zeichne(x+kbreite-rd/2,y,rd);
...
    Rad1.zeigeX(xRad);
```

- Bearbeite das Arbeitsblatt 07, S. 4

Polymorphie

Das Programm `Seifenkiste` kann durch Herstellen einer Vererbungsbeziehung so ergänzt werden, dass auf Basis der Seifenkiste durch Aufsetzen eines Dachs als „Führerhaus“ ein „Auto“ gezeichnet wird.
Die Klasse `Seifenkiste` ist die *allgemeinere Klasse*, die Klasse `Auto` die *speziellere Klasse*.
Die Methode `animiere()` der `Seifenkiste` kann zwar nicht verwendet werden, weil dort kein Dach gezeichnet wird.
Wenn aber in der Klasse `Auto` ebenfalls eine Methode `animiere()` erstellt wird, kann dort das `Dach` ergänzt werden. In diesem Fall wird auch nur die Methode `animiere()` der Klasse `Auto` ausgeführt.





- Werden in einer Vererbungsbeziehung Eigenschaften der allgemeineren Klasse durch Deklarierung derselben Eigenschaft in der spezielleren Klasse verändert, nennt man das **Überschreiben**.

Wenn man eine Methode *überschreibt*, wird die Methode der spezielleren Klasse ausgeführt, im Beispiel die Klasse `Auto` und **nicht** die Methode der allgemeineren Klasse („Elternklasse“).

- In mit einer Vererbungsbeziehung definierten speziellen Klassen können Eigenschaften hinzugefügt oder vorhandene verändert werden. Objekte, die auf derselben Klasse basieren, können also unterschiedliche Verhaltensweisen haben. Das wird als **Polymorphie** bezeichnet.

Beispiel:

Wenn in der spezielleren Klasse dieselbe Methode vorhanden ist, wird dadurch die Methode der allgemeineren Klasse überschrieben.
Es gilt die Methode der spezielleren Klasse.
Im Beispiel wird `Seifenkiste.animiere()` durch `Auto.animiere()` überschrieben.

```
class Seifenkiste {
...
  void animiere() {
    Kiste.zeichne(x,y-khoehe,
          kbreite,khoehe,kr,kg,kb);
    Rad1.zeichne(x+rd/2,y,rd);
    Rad2.zeichne(x+kbreite-rd/2,
          y,rd);
    wende();
    beende();
  }
...
}
```

In der Klasse `Auto` wird eine Vererbungsbeziehung zu der Klasse `Seifenkiste` hergestellt.

```
class Auto extends Seifenkiste {
  Auto() {
    super(0,80,2,120,20,255,0,0,28);
    d1=new Dach();
    d1.zeichne(x+65,y,rd);
  }
  void animiere() {
    d1.zeichne(x+50,y-khoehe,rd);
    Kiste.zeichne(x,y-khoehe,
          kbreite,khoehe,kr,kg,kb);
    Rad1.zeichne(x+rd,y,rd);
    Rad2.zeichne(x+kbreite-rd,y,rd);
    wende();
    beende();
  }
}
```

Bearbeite das Arbeitsblatt 07, S. 5-6

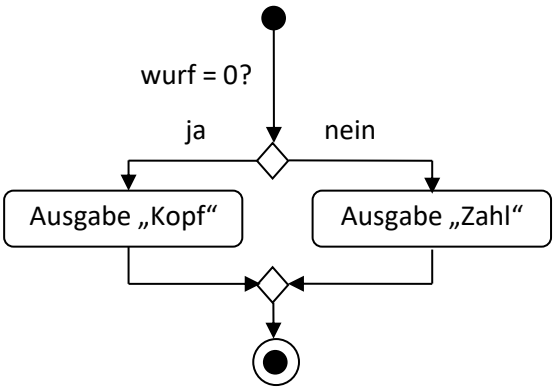
Algorithmische Grundstrukturen II

Auswahlstrukturen

- Bearbeite das Arbeitsblatt 08, S. 1: Zweiseitige Auswahl

- Bei der **zweiseitigen Auswahl** wird *entweder* die eine *oder* die andere von zwei Sequenzen des Programms ausgeführt.
  Schreibweise für die zweiseitige Auswahl:

```
if (m1.zeigeWurf () == 0) {
  text("Kopf",15,80);
}
else {
  text("Zahl",15,80);
}
```



Zweiseitige Auswahl

- **Mehrseitige Auswahl**:
  Bei der *mehrseitigen Auswahl* wird genau eine aus mehreren Sequenzen des Programms ausgeführt.
  Beispiel in Processing:

```
switch (Wert) {
   case 1:
     ... (Anweisungen);
     break;
   case 2:
     ... (Anweisungen);
     break;
   default:
     ... (Anweisungen);
}
```

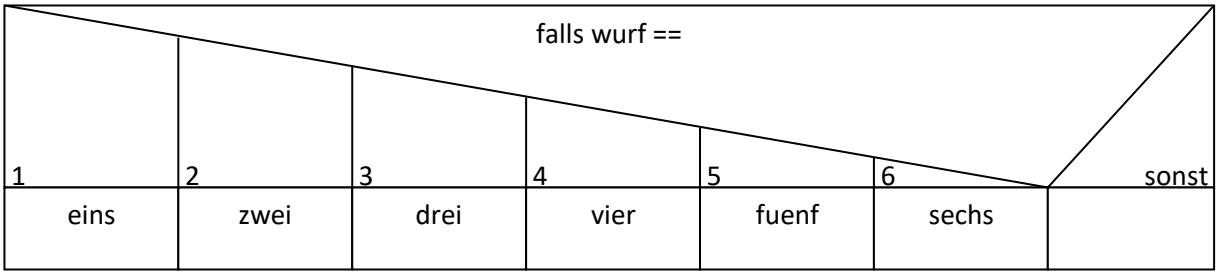
Bei der mehrseitigen Auswahl wird der Wert einer Variablen geprüft. Mit der Anweisung `case` werden die zu unterscheidenden Fälle angegeben.

Die Sequenzen für die jeweiligen Fälle werden **nicht** mit geschweiften Klammern festgelegt.
Vielmehr muss das Ende der Sequenz mit der Anweisung `break` gekennzeichnet werden.

Der „sonst-Fall“ wird mit der Anweisung `default` angegeben.

- Bei der *mehrseitigen Auswahl* wird genau eine aus mehreren Sequenzen des Programms ausgeführt. Das lässt sich in einem Struktogramm übersichtlicher darstellen als in einem Aktivitätsdiagramm.

  Im Beispiel ein Struktogramm für die Auswertung eines Würfels:



Mehrseitige Auswahl

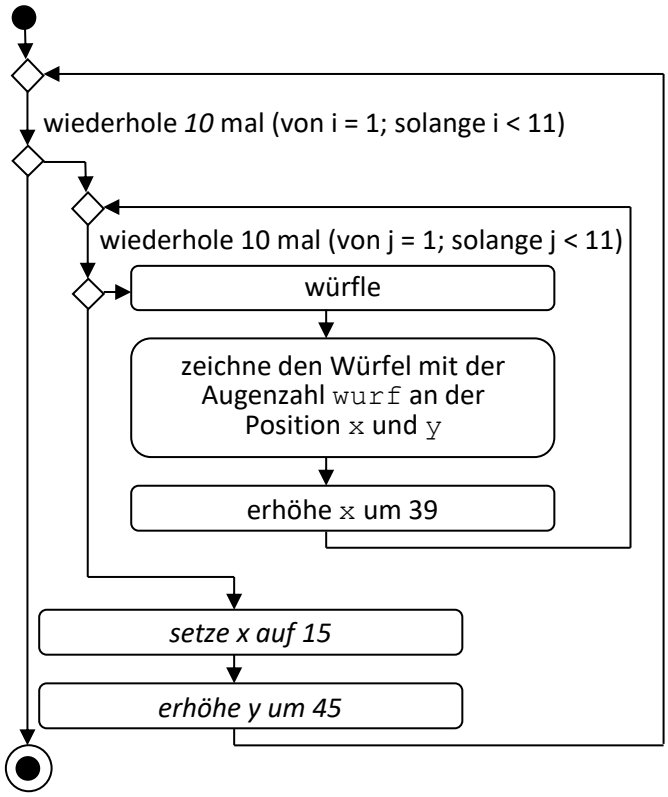
- Bearbeite das Arbeitsblatt 08, Seite 2-3.

Wiederholungsstrukturen

Bei einer Wiederholungsstruktur wird ein Teil des Programmcodes mehrfach abgearbeitet.

Zählschleifen

Im Arbeitsblatt 04, S. 3 wurde besprochen, dass in Processing Zählschleifen nicht für Animationen verwendet werden können (`v10_kreis_for`). (*vgl. .\262-materialien\kreis\08_kreis_for*)
Ein Beispiel für die Verwendung der Zählschleife Hast du im Arbeitsblatt 05, S. 2 im Zusammenhang mit Arrays kennengelernt (`gluecksrad2`). (*vgl. .\262-materialien\zufall\02_gluecksrad2*)
Bei einer Zählschleife ist die Anzahl der Wiederholungen festgelegt.



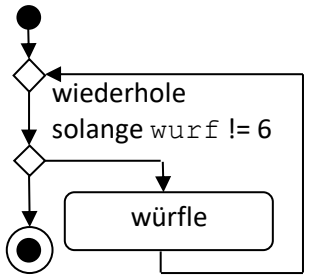
- Mit einer Zählschleife („for-Schleife“) kann man eine Folge von Anweisungen mit einer bestimmten Anzahl von Wiederholungen ausführen.
  In Processing lautet die Syntax für eine Zählschleife:

```
for (var i=(Startwert); (Endebedingung); i++) {
  Anweisungen; ...
}
```

- A Bearbeite das Arbeitsblatt 08, Seite 4-5.

Wiederholung mit Ausführungsprüfung

Wenn der Programmablauf auf Grund einer Bedingung wiederholt wird, spricht man von einer bedingten Wiederholung.
Wenn der Fall eintreten kann, dass eine Schleife überhaupt nicht ausgeführt werden darf, verwendet man eine *Wiederholung mit Ausführungsprüfung*.



- Prüft man die Bedingung zur Ausführung einer Sequenz zu Beginn der Ausführung ab, liegt eine **Wiederholung mit Anfangsbedingung** vor.
- A Bearbeite das Arbeitsblatt 08, Seite 6: Wiederholung mit Ausführungsprüfung

Rekursion

- A Bearbeite das Arbeitsblatt 09
- Als Rekursion bezeichnet man eine Programmiertechnik, in der eine Funktion sich selbst aufruft. Jeder Aufruf der rekursiven Funktion muss sich in endlich viele Schritte auflösen lassen, sie darf also nicht in eine Endlosschleife geraten.

  Ein Beispiel ist die Methode `berechneFakultaet ()` mit rekursiver Struktur im Aktivitätsdiagramm rechts.